Les réseaux de neurones appliqués à la statistique publique : méthodes et cas d'usage

Documents de travail

N° M2023-01 – Février 2023

Damien BABET
Quentin DELTOUR
Thomas FARIA
Stéphanie HIMPENS

Insee
Mesurer pour comprendre

INSTITUT NATIONAL DE LA STATISTIQUE ET DES ÉTUDES ÉCONOMIQUES
Série des documents de travail « Méthodologie Statistique »
de la Direction de la Méthodologie et de la Coordination Statistique et Internationale

M 2023/01

Les réseaux de neurones appliqués à la statistique publique : méthodes et cas d'usages

Damien BABET
Insee-Dese, Département des études économiques
Quentin DELTOUR
Ensae au moment de l'étude ; Autorité de la concurrence
Thomas FARIA
Insee-Dmcsi, SSP Lab
Stéphanie HIMPENS
Insee-Dmcsi, SSP Lab ; Banque de France

Février 2023

Direction de la méthodologie et de la coordination statistique et internationale
Département des Méthodes Statistiques -Timbre L001 -
88 Avenue Verdier - CS 70058 - 92541 Montrouge Cedex - France -
Tél. : 33 (1) 87 69 55 00 - E-mail : -DG75-L001@insee.fr - Site Web Insee : http://www.insee.fr

Les réseaux de neurones appliqués à la statistique publique : méthodes et cas d'usages*

Damien Babet† Quentin Deltour‡

Thomas Faria § Stéphanie Himpens ¶

1er février 2023

Résumé

Les réseaux de neurones peuvent avoir des applications utiles pour la statistique publique. Ce document propose une introduction rapide aux réseaux de neurones, de leurs fondements théoriques jusqu'à leur mise en oeuvre pratique en R et python sur des problématiques spécifiques de statistique publique. Nous en illustrons les possibilités et les limites à travers trois cas d'usage détaillés : 1. l'imputation de valeurs manquantes dans une enquête, une activité importante en statistique et pour laquelle la performance prédictive est centrale. 2. L'exploitation de fichiers d'images, élargissant ainsi les possibilités d'utilisation des images comme des données statistiques. 3. La réduction de dimension qui permet de synthétiser des fichiers de données et ouvre la voie à de nombreuses applications. Ce document s'accompagne des codes permettant de mettre en oeuvre les méthodes présentées.

Mots clés : réseaux de neurones, machine learning, statistique publique, imputation, convolution, auto-encodeur

Codes JEL : C45, C50, C81, C82, C88

*Nous souhaitons remercier Marco Cuturi et les participants du séminaire du 30 mars 2021, Matthieu Doutreligne pour ses commentaires, Théo Roudil-Valentin qui était membre du projet Ensae de statistiques appliquées sur les auto-encodeurs. Merci à Élise Coudin, Matthieu Lequien, Sébastien Roux, Patrick Sillard et Sylvie Lagarde pour leur soutien et leurs relectures attentives.

† Insee-Dese, Département des études économiques

‡ Ensae au moment de l'étude ; Autorité de la concurrence

§ Insee-SSP Lab

¶ Insee-SSP Lab ; Banque de France

Neural networks: methods and use cases for official statistics

Abstract

Neural networks applied to official statistical data can have many useful applications. We propose a quick introduction to neural networks, from their theoretical foundations to their practical implementation in R and Python on specific official statistics issues. We illustrate their possibilities and limitations through three detailed use cases: 1. imputation of missing values in a survey, an important challenge for official statistics, for which predictive performance is central. 2. Image files usage, expanding the potential use of such files as statistical data. 3. Dimension reduction, which synthesises large data files and opens the way to many applications. This document is accompanied by codes to implement the methods presented.

Keywords: neural networks, machine learning, official statistics, imputation, convolution, autoencoder

JEL codes: C45, C50, C81, C82, C88

Table des matières

Introduction

Les réseaux de neurones peuvent se voir comme des modèles très flexibles capables d'analyser, traiter l'information contenue dans des données aux formats très divers et très peu structurés (images, vidéos, textes,...). Ils requièrent des données en grand nombre pour leur estimation/entraînement. Leur développement est notamment lié à l'essor des performances informatiques (parallélisation, Big Data, techniques cloud). Leurs domaines d'application sont larges : bioinformatique, analyse des images, du son, analyse du langage, applications de navigation en temps réel, bientôt la voiture autonome... Ces modèles sont déjà très présents dans notre quotidien. Sur nos téléphones, les applications de navigation situent les adresses à leur position géographique exacte, prennent en compte la signalisation routière, analysent la densité de circulation en temps réel, et calculent le chemin le plus rapide. Cela est évidemment rendu possible par l'exploitation de bases de données publiques et privées, mais également par le recours aux réseaux de neurones pour analyser automatiquement les images satellites et photographies aériennes (pour identifier les volumes et délimiter les bâtiments), les photos et vidéos à hauteur de circulation (pour reconnaître les panneaux routiers, les noms et numéros de rue ou les devantures de boutiques), ou pour prédire la circulation [1].

La statistique publique fait partie des domaines d'application possibles, à la fois parce que les réseaux de neurones ont le potentiel d'étendre le type de données desquelles on peut tirer une information statistique (images, textes), et aussi parce que leur flexibilité peut être un atout pour réaliser certaines tâches classiques dans la constitution et l'analyse des bases de données de statistique publique (imputation, prédiction, etc.). Pour autant, ce sont des modèles complexes, imbriqués, ad hoc, pouvant apparaître comme des boîtes noires, conduire à des biais, ou être difficiles à maintenir. Évaluer le gain potentiel en comparaison à des méthodes plus classiques ou plus simples est indispensable. Ce document de travail n'a pas vocation à fournir un cours de *deep learning* : c'est un domaine vaste en constante évolution. Ce document de travail poursuit plutôt deux premiers objectifs bien ciblés : identifier des cas d'usages des réseaux de neurones (RN) utiles pour la statistique publique, grâce à trois expériences présentées en détail, et fournir des bases théoriques et pratiques aux statisticiennes et statisticiens qui souhaiteraient s'initier à ces méthodes. Le but est de fournir des idées, des repères et des outils aux lectrices et lecteurs désireux d'entreprendre à leur tour de telles expérimentations. Dans ce but, ce document de travail est accompagné de la mise à disposition de tous les codes en python ou R [2] des cas d'usage et de la partie méthodologique, dont des extraits illustrent régulièrement le texte [3]. Ce document poursuit un troisième objectif, qui est de contribuer à mieux cerner les perspectives mais aussi les

1. Voir par exemple pour Google Derrow-Pinion, She, Wong, Lange, Hester, Perez, Nunkesser, Lee, Guo, Wiltshire, et al. (2021), Yu, Szegedy, Stumpe, Yatziv, Shet, Ibarz, et Arnoud (2015), Sirko, Kashubin, Ritter, Annkah, Bouchareb, Dauphin, Keysers, Neumann, Cisse, et Quinn (2021)

2. les questions logicielles, ainsi que les librairies mobilisées sont discutées dans le premier chapitre

3. `https://github.com/InseeFrLab/publication_DT_reseaux_neurones` Les codes sont ouverts et conçus pour être reproductibles, en particulier pour ceux qui s'appuient sur des données ouvertes dans les chapitres 1, 3 et 4.

limites des applications des RN pour la statistique publique. Les réseaux de neurones qui y sont décrits sont classiques et relativement simples pour des modèles de *deep learning*. C'est leur application à des données et des questionnements typiques de la statistique publique qui est nouvelle, et qui motive ce travail.

Un réseau de neurones artificiels, ou réseau neuronal artificiel, est un système dont la conception est à l'origine schématiquement inspirée du fonctionnement des neurones biologiques. On le représente communément sous cette forme de « neurones » reliés par des connexions, mais il peut être aussi bien décrit en termes mathématiques, comme un enchaînement de compositions de fonctions. Les RN évoqués ici font partie des méthodes d'apprentissage automatique (ou de *machine learning*) et plus précisément de l'apprentissage profond (*deep learning*). Ce sont des systèmes capables de s'adapter et d'apprendre à partir des données fournies en entrée.

L'intelligence artificielle et la branche de l'apprentissage profond (le *deep learning*) ont effectivement transformé l'analyse d'images et de vidéos, de sons et de données textuelles ou langagières. Ils sous-tendent les algorithmes de reconnaissance de visage, générant des images artificielles ou appliquant des filtres aux photos, la traduction par ordinateur ou les progrès des véhicules autonomes, et font l'objet d'une recherche scientifique intense. Ils restent pourtant très peu utilisés par les praticiens de la statistique publique principalement pour les raisons suivantes.

L'intérêt des réseaux de neurones se manifeste davantage lorsque la taille des données augmente, or la statistique publique travaille sur des données de taille moyenne : données d'enquête comptant quelques dizaines de milliers d'observations, données administratives en millions, ou un peu plus quand elles croisent des populations avec des événements fréquents. Ces bases comportent moins d'observations, mais aussi moins de variables, typiquement quelques centaines au plus, généralement bien moins que le nombre de pixels d'une petite image par exemple. Les données ne sont pas non plus exactement de même nature. Elles sont généralement structurées (questions fermées, usage de nomenclatures et de définitions normalisées, partage de concepts, etc.), et souvent hétérogènes (variables de types différents, valeurs manquantes) tandis que les données typiques des RN sont traitées pour être homogènes (des images de même format) à partir d'un matériel de base non structuré. Par ailleurs, les enjeux de représentativité et de confidentialité sont très importants dans le contexte de la statistique publique.

Du fait de la complexité des compositions de fonctions qu'ils recouvrent, les réseaux de neurones peuvent apparaître comme des « boîtes noires ». Une fois entraînés, ils peuvent prédire avec une très grande efficacité si une image représente un chien ou un chat, et quel serait le meilleur coup à jouer au go, mais il est difficile de comprendre comment sont faites ces prédictions, même approximativement. L'interprétabilité des algorithmes demande un travail supplémentaire, alors qu'il s'agit souvent d'une exigence des producteurs et

des usagers des statistiques publiques. On souhaite même souvent produire directement l'estimation $\hat{\beta}$ du paramètre d'un modèle, et les RN sont a priori peu adaptés à cet usage [4].

Dernier obstacle enfin, les réseaux de neurones ne sont pas encore fournis clés en main. Il faut d'abord en comprendre le fonctionnement, ils demandent de nombreux réglages, des options, des choix de modélisation, autant d'« hyperparamètres » difficiles à optimiser. Avant de « laisser parler les données », il faut une longue mise en place qu'on s'épargne parfois en choisissant un modèle classique ou un algorithme plus simple de *machine learning* comme une forêt aléatoire. Il faut aussi se familiariser avec le fonctionnement de nouveaux *packages* comme Keras (disponible pour R et python).

C'est ce dernier obstacle en particulier que ce document vise à lever, car il y a des usages prometteurs pour les statisticiens. Les RN peuvent être efficaces sur des données de petite taille et passent à l'échelle facilement lorsqu'on mobilise ensuite de plus gros fichiers. Ils sont très souples et peuvent ainsi s'adapter à certaines demandes de la statistique publique en particulier la protection de la confidentialité, les appariements, le codage automatique, la détection et la correction d'erreurs, etc, ou encore enrichir les types de données exploitables par la statistique publique (images, textes,...). En effet et comme d'autres méthodes relevant du *machine learning*, les réseaux de neurones peuvent être présentés comme des modèles statistiques très flexibles, mettant en jeu de nombreuses variables explicatives. Ils sont particulièrement adaptés dès lors qu'on s'intéresse à prédire la valeur d'une variable d'intérêt. On les qualifie alors de prédicteurs. Ils donnent des réponses à des problèmes de type $\hat{y}$, par opposition aux problèmes de type $\hat{\beta}$ qui consistent, eux, à mesurer le lien (ou la causalité) entre deux variables, pour reprendre la distinction utile de Mullainathan et Spiess (2017).

Le premier chapitre de ce document constitue une initiation à quelques principes du *machine learning* et décrit plus en détail le fonctionnement des réseaux de neurones. Il introduit également quelques ressources logicielles en comparant, sur un exemple simple, des réseaux de neurones programmés avec Pytorch et Keras. Pour approfondir, on pourra se référer à deux ressources librement accessibles en ligne, le livre *Deep Learning* (Goodfellow, Bengio, et Courville, 2016) [5] et le cours de Yann LeCun à New York University (Le Cun, Canziani, Misra, Lewis, et Bresson, 2020), disponible en français [6], ainsi qu'à la riche documentation des *packages* présentés [7].

À travers trois principaux cas d'usage, les chapitres 2 à 4 adoptent le point de vue du statisticien : comment mobiliser ces réseaux de neurones en pratique, quelles méthodes particulières s'appliquent ?

Le chapitre 2 aborde un cas d'usage courant en statistique publique, l'imputation des valeurs manquantes dans une enquête, à travers l'exemple de l'imputation du salaire dans

4. On aborde dans la partie sur d'autres types de RN en conclusion leur utilisation pour l'analyse causale

5. Version anglaise : http://www.deeplearningbook.org

6. https://atcold.github.io/pytorch-Deep-Learning/fr/

7. Keras : https://keras.io/getting_started/ et https://keras.rstudio.com/ pour son interface en R et Pytorch : https://pytorch.org/tutorials/

l'enquête Emploi. Le chapitre détaille les transformations des variables pour en faire des *inputs* adaptés au réseau de neurones. Jugé purement sur le plan prédictif, un RN est plus efficace que le modèle économétrique classique (l'équation de Mincer). Cependant l'imputation a aussi d'autres exigences que la performance prédictive, et les capacités des RN à y répondre sont discutées.

Le troisième chapitre décrit le fonctionnement et l'usage des réseaux convolutifs (*convnets*) qui permettent d'analyser des images. La possibilité ainsi offerte de mobiliser des données inédites pour la statistique publique est illustrée à travers l'exemple de la distinction entre des paysages de champs ouverts et de champs clos sur des images du cadastre. Cette nouvelle variable est ensuite mobilisée dans une analyse économétrique de la répartition du nouveau bâti. Le chapitre présente également une méthode pour « ouvrir la boîte noire » et essayer de mieux comprendre les déterminants des prédictions d'un *convnet*.

Le quatrième chapitre présente les auto-encodeurs, des réseaux de neurones entraînés à encoder dans un petit vecteur des données de grande dimension. Cette réduction de dimension a de nombreux usages, et peut notamment servir de pré-entraînement pour un prédicteur RN, ce qui est illustré à travers l'exemple de l'auto-encodage de l'enquête Emploi pour entraîner ensuite un prédicteur du risque individuel de chômage.

En conclusion, nous discutons certaines des perspectives et limites des RN pour la statistique publique au-delà des cas d'usage présentés : confidentialité et représentativité, amélioration des performances et capacités de calculs, application à des données textuelles ou en économétrie.

1 Apprentissage automatique et réseaux de neurones, concepts et prise en main

Ce premier chapitre constitue une initiation à la théorie sous-jacente des réseaux de neurones et aux principes de l'apprentissage statistique. Pour approfondir, le lecteur pourra se référer à deux ressources librement accessibles en ligne, le livre *Deep Learning* (Goodfellow, Bengio, et Courville, 2016) dont la version anglaise est en ligne [8] et le cours de Yann LeCun à New York University (Le Cun, Canziani, Misra, Lewis, et Bresson, 2020), disponible en français [9].

Les réseaux de neurones sont des algorithmes parmi d'autres qui relèvent de l'apprentissage statistique (*statistical learning*) ou apprentissage automatique (*machine learning*). Avant de les présenter en tant que tels, il est utile de rappeler les idées fondatrices de l'apprentissage automatique, le vocabulaire qui est généralement utilisé (rassemblé dans la partie 1.1.2), ainsi que ses différences et ses points de rapprochement avec la statistique mathématique, dont relève en particulier l'économétrie.

1.1 Les fondamentaux de l'apprentissage automatique

1.1.1 Apprentissage automatique et économétrie : deux « cultures » différentes ?

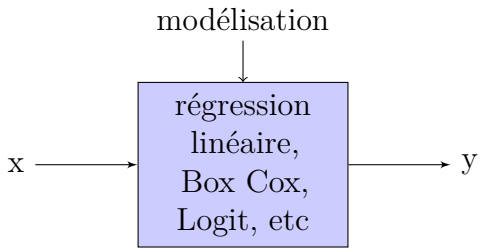
L'apprentissage automatique et la statistique mathématique, et notamment l'économétrie, peuvent tous les deux avoir comme finalité de construire un modèle prédictif d'une variable d'intérêt, à l'aide de variables explicatives (ou *features*). Pourtant, la statistique mathématique et l'économétrie relèvent d'une culture de modélisation probabiliste (*data modeling culture*) qui suppose que les données sont générées par un modèle probabiliste donné, alors que l'apprentissage automatique et les réseaux de neurones s'inscrivent dans une culture de modélisation algorithmique (*algorithmic modeling culture*) qui ne se prononce pas sur la génération des données et cherche des modèles très flexibles (Breiman, 2001). Avec un objectif commun, ces deux cultures donnent lieu à des principes et des hypothèses différents. Charpentier, Flachaire, et Ly (2018) décrivent précisément leurs origines, les principales différences ainsi que les points de synergie possibles entre les méthodes. Les éléments ci-dessous s'inspirent grandement de leur article.

L'économétrie et les techniques *supervisées* d'apprentissage automatique sont proches au sens où ces approches partent toutes les deux d'un ensemble d'observations $(y_i, x_{i,j=1\dots p})_{i=1\dots n}$, indicées par i et où les p variables $x_{i,j=1\dots p}$ sont censées expliquer ou *prédire* les y_i. Ces deux approches cherchent à trouver un *modèle*, c'est-à-dire une fonction $f(x_{i,j=1\dots p})$ telle que $f(x_{i,j=1\dots p}) = y_i,\ \forall i$. Cependant, même si la finalité semble identique, l'économétrie et l'apprentissage automatique présentent de réelles différences.

8. http://www.deeplearningbook.org
9. https://atcold.github.io/pytorch-Deep-Learning/fr/

L'économétrie s'inscrit dans la statistique mathématique, elle repose sur des modèles probabilistes (*data modeling culture*). Elle s'appuie le plus souvent sur une théorie économique et développe des modèles structurés par cette théorie. Les paramètres de ces modèles résument les liens entre les variables explicatives et la variable d'intérêt. L'économétrie s'intéresse le plus souvent à isoler et expliquer l'effet propre d'une variable explicative sur la variable d'intérêt. Ainsi, et comme le décrit le schéma 1, l'économètre fait souvent :

1. l'hypothèse d'un modèle (déduit de la théorie économique),
2. l'estimation des paramètres θ_k de son modèle,
3. en introduisant un terme d'erreur, avec une hypothèse distributionnelle.

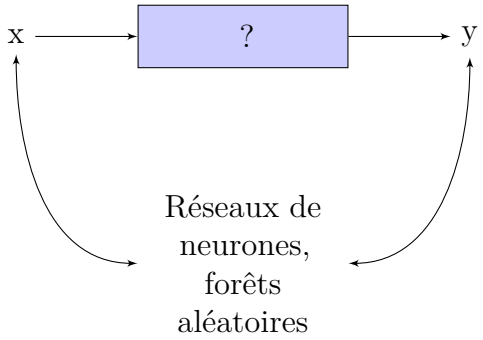


GRAPHIQUE **1 – Démarche en économétrie**

Comme l'économétrie relève de la statistique mathématique, la généralisation des résultats hors de l'échantillon d'estimation repose sur l'hypothèse de modélisation probabiliste. Les outils habituels d'inférence statistique s'appliquent (maximum de vraisemblance, etc.) et la théorie asymptotique peut y jouer un rôle important. La généralité des résultats est alors garantie si l'estimateur $\hat{\theta}_k$ a les bonnes propriétés statistiques. On peut par exemple inférer l'incertitude touchant l'estimation d'un paramètre, mener des tests et tirer des conclusions de significativité. Au prix d'hypothèses additionnelles sur le mécanisme générateur des données (données d'expérimentation, hypothèses d'exclusions, etc.), on peut avoir une interprétation causale des résultats obtenus, ce qui est souvent un objectif de l'analyse économétrique.

L'apprentissage automatique relève de l'*algorithmic modeling culture*, proche de la statistique computationnelle et de l'optimisation. L'objectif y est avant tout de prédire correctement la variable d'intérêt. Des classes de modèles très flexibles sont fréquemment mobilisées à cette fin (les réseaux de neurones font partie de ces modèles très souples). Le principe général de fonctionnement d'un tel modèle est le suivant : l'étape d'estimation consiste à minimiser une *fonction de coût*, somme des *fonctions de perte* qui mesurent la distance entre la prédiction du modèle et la valeur observée de la variable d'intérêt pour toutes les observations retenues pour cette étape (cet ensemble de données est appelé l'*échantillon d'apprentissage*). La forme définitive du modèle retenu n'est parfois ni explicite, ni interprétable (voir graphique 2). Par exemple, la prédiction d'une forêt aléatoire provient de l'agrégation d'une multitude de décisions d'arbres élémentaires. Il n'y a pas de forme explicite simple du modèle, uniquement ces dizaines ou centaines d'arbres de décision différents. Cela n'est pas un obstacle car l'accent est mis avant tout sur la qualité des prédictions. Il n'y a pas non plus d'hypothèse sur le processus probabiliste de génération

des données et donc pas de théorie asymptotique sur laquelle s'appuyer pour étendre les résultats. C'est pourquoi un tel modèle n'est *a priori* pas automatiquement performant si on l'applique à de nouvelles observations (prédiction sur un nouvel échantillon). Pour s'assurer de sa performance *out of sample*, de la généralité des résultats, on ne peut pas s'appuyer sur la théorie probabiliste pour estimer une incertitude. Il est primordial de vérifier à quel point les prédictions s'approchent des vraies valeurs de la variable d'intérêt observées dans un échantillon de données qui n'a pas été utilisé dans l'estimation du modèle. On parle alors d'*échantillon test*. L'erreur associée au modèle n'est pas inférée, elle est mesurée empiriquement sur cet échantillon.



GRAPHIQUE **2 – Démarche de l'apprentissage automatique**

La *sélection* du modèle est également une source de différences entre les deux cultures. En économétrie, les méthodes requièrent souvent que le nombre p de variables, la *dimension* du modèle, reste petit devant n, le nombre d'observations. La forme du modèle est soit fixée avant l'estimation des paramètres, soit lors d'une phase d'essais et tâtonnements où le choix final est réalisé *ex-post* entre plusieurs modèles ajustés sur l'ensemble des données avec une pénalisation pour la complexité dans les critères d'adéquation tels que le BIC ou l'AIC. Sur le fond, cette démarche est justifiée par le fait que les considérations théoriques ou structurelles jouent un rôle important dans l'élaboration du modèle.

En apprentissage automatique, la phase de sélection du modèle et la phase d'estimation sont confondues, réalisées « automatiquement » en minimisant la *fonction de coût* qui somme les pertes sur toutes les observations. Il s'agit de trouver la fonction $f(x_{i,j=1\ldots p})$ qui minimise la fonction de coût. La fonction f est directement sélectionnée lors de l'entraînement du modèle, sur l'échantillon d'entraînement. Le choix de modèle intervient donc en même temps que l'estimation. Pour s'assurer que les performances se maintiendront sur l'échantillon test, cette fonction de coût peut pénaliser directement la complexité du modèle, par exemple le nombre de paramètres à estimer. C'est la raison pour laquelle les techniques d'apprentissage automatique sont particulièrement adaptées en « *grande dimension* », c'est-à-dire lorsque les variables explicatives p sont très nombreuses, pouvant même dépasser le nombre d'observations n. Ces techniques procèdent à une forme de réduction de la dimension implicite ou explicite qui permet de traiter ces problèmes.

En réalité, la frontière entre les deux domaines est moins hermétique que cela surtout

depuis une dizaine d'années. L'économétrie semi-paramétrique ou en grande dimension (régressions pénalisées) trouve sa place entre les deux approches. Les régressions pénalisées peuvent servir à réduire la dimension d'un modèle en sélectionnant des variables. Il est possible d'avoir des estimateurs convergents avec $p > n$ si la dimension effective, c'est-à-dire le nombre de variables significatives (et donc de paramètres à estimer), est faible (concept de *sparsité*). L'opposition entre les problèmes de type $\hat{\beta}$ et les problèmes de type $\hat{y}$ est aussi à relativiser. D'un côté, la statistique et l'économétrie classiques génèrent un certain nombre de problèmes purement prédictifs (appariement, prédiction d'une variable d'intérêt par un instrument, etc.). De l'autre, l'interprétabilité et l'estimation causale sont des domaines de recherche importants en apprentissage automatique.

Les deux cultures s'enrichissent l'une l'autre, et peuvent être mobilisées de façon complémentaire. Les méthodes de descentes de gradient si performantes pour l'optimisation en apprentissage statistique ont permis des avancées en économétrie. L'identification causale peut intégrer des étapes d'apprentissage automatique (voir L'Hour (2020), Chernozhukov, Chetverikov, Demirer, Duflo, Hansen, Newey, et Robins (2018) et une présentation rapide en conclusion, partie 4.3.4). Parallèlement, les techniques algorithmiques mobilisent des méthodes statistiques. Ces dernières sont utiles afin de mieux comprendre les liens entre les variables. Souvent, il a été reproché aux méthodes algorithmiques leur côté boîte noire. Dans certains cas, il est impossible de mettre en évidence et/ou d'expliquer la fonction choisie par l'algorithme. Les méthodes classiques de traitement d'images ou de séries temporelles peuvent permettre d'améliorer les performances des méthodes algorithmiques. Par exemple, en pratique, les réseaux de neurones gèrent parfois très mal la saisonnalité. Il peut être utile de désaisonnaliser les séries. Un modèle paramétrique bien posé peut avoir des performances supérieures avec un coût computationnel moindre qu'un modèle algorithmique plus complexe. Le modèle économétrique sert alors de référence. Si les techniques d'apprentissage automatique ne permettent pas d'améliorer les résultats prédictifs, le modèle peut être considéré comme plus robuste. Un principe de parcimonie suggère qu'il n'est sans doute pas nécessaire de le complexifier.

1.1.2 Un peu de vocabulaire

Le terme de *machine learning* est attribué à Samuel (1959) qui définit une méthode de *machine learning* ou d'*apprentissage automatique* comme une méthode permettant de rendre un programme capable d'*apprendre* à partir d'exemples de données sans avoir été programmé. Il existe des algorithmes d'apprentissage *supervisés* et *non supervisés*.

L'*apprentissage non supervisé* couvre les techniques statistiques descriptives qui cherchent à synthétiser les données, faire émerger des structures dans les données sans *a priori*. Il se rapproche de l'analyse des données (analyse en composantes principales, classification ascendante hiérarchique, k-means). L'apprentissage non supervisé prend en entrée des données $x_{i,j=1...p}$ sans y, *labels* ou variables à prédire.

On se concentre ici sur l'*apprentissage supervisé*[10]. Un algorithme de *machine learning supervisé* est une méthode capable de déduire ou d'approcher la valeur y_l pour des caractéristiques $x_{l,j=1...p}$ non encore observées en ayant *appris* la relation, c'est-à-dire estimé un modèle, à partir de l'échantillon d'*entraînement* de données $(x_{i,j=1...p}, y_i)_{i=1...n}$. La variable y est parfois nommé *le label*, la variable à *prédire* ou en encore la sortie de l'algorithme. Par extension, on nomme l'estimation $\hat{y}$ la prédiction du modèle. Les variables explicatives ou encore les caractéristiques $x_{i,j=1...p}$ sont nommées les *features* ou encore les variables en entrée (*inputs*). Elles peuvent nécessiter des pré-traitements afin d'être exploitables par les algorithmes, par exemple des créations d'indicatrices. On parle de *feature engineering*.

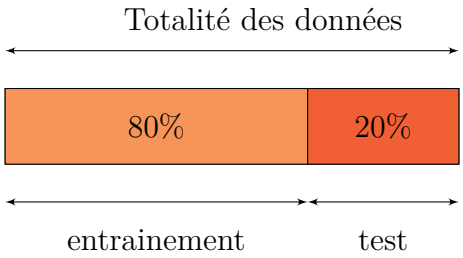
Lorsque la variable d'intérêt y est continue, on parle d'un problème de *régression*. Dans le cas d'une variable discrète, à deux modalités ou plus, on parlera d'une *classification*[11].

La phase d'*apprentissage* en apprentissage automatique désigne la sélection du modèle et son estimation, par minimisation de la *fonction de coût*. On parle aussi de la phase d'*entraînement*. Il s'agit de choisir la fonction, parmi un ensemble potentiellement très grand, qui reproduit le mieux les données de l'échantillon d'*entraînement*, la proximité étant captée par les *fonctions de perte* individuelles (*loss*) et leur somme, la fonction de coût (ou *risque empirique*). Pour prédire une variable continue, on choisira par exemple la perte quadratique $\frac{1}{n}\sum_{i=1}^{n}(f(x_{i,j=1...p}) - y_i)^2$. Pour une variable binaire, il pourra s'agir de l'entropie croisée $-\frac{1}{n}\sum_{i=1}^{n}(y_i \times log(f(x_{i,j=1...p})) + (1 - y_i) \times log(1 - f(x_{i,j=1...p})))$ [12].

En *apprentissage automatique*, les modèles doivent tout de même rester simples et parcimonieux dans le sens suivant : s'ils incluent trop de paramètres, ils sont trop adaptés à l'échantillon sur lequel ils ont été estimés (*échantillon d'apprentissage*) et prédisent mal sur de nouvelles données. On parle de cas de *sur-apprentissage*. Il peut alors être nécessaire de simplifier le modèle ou d'introduire des techniques de régularisation comme une pénalisation sur la valeur ou le nombre de paramètres à estimer. Si, en revanche, le modèle sélectionné est trop simple, il ne prend pas en compte toute la complexité des données. Son pouvoir prédictif sera faible. En économétrie, cela se produit lorsque l'on ignore certaines liaisons entre les variables. Par exemple, si on essaye d'ajuster une relation linéaire alors que la liaison entre les variables est quadratique. En apprentissage automatique, on parle alors de *sous-apprentissage*. Afin de diagnostiquer *le sous- ou le sur-apprentissage*, on évalue les performances des modèles sur un *échantillon test* qui n'a pas été utilisé dans la phase d'entraînement du modèle. On dit que l'algorithme ne devra pas avoir « vu » les données que cet échantillon contient. Il est ainsi usuel de séparer les données disponibles en échantillon *test* (20 % des données env.) et d'*apprentissage* (80 % des données env.)

10. Le chapitre 4 aborde un exemple de méthode non supervisée avec l'*auto-encodeur*
11. Régression et classification n'ont donc pas la même signification qu'en économétrie ou en analyse des données.
12. Ces fonctions de coût sont bien sûr les analogues des méthodes classiques des moindres carrés ordinaires et de la maximisation de la vraisemblance.

avant d'estimer le modèle. On parle de démarche de type « *hold-out* » puisqu'une partie des données est laissée de côté pour la seule évaluation des performances (voir graphique 3).



GRAPHIQUE **3 – Séparation des données entre échantillons d'entraînement et de test - Démarche de type "Hold-Out"**

Parfois, les méthodes utilisées comportent elles-mêmes des paramètres qui doivent être fixés en amont pour permettre l'estimation du modèle. On parle dans ce cas *d'hyperparamètres*. Par exemple, dans le cas d'une régression pénalisée de type Lasso, l'algorithme cherche à minimiser le programme d'une régression par moindres carrés avec un terme supplémentaire pénalisant les valeurs des coefficients et pondéré par l'*hyperparamètre* λ :

$$\underset{\beta_1,\dots\beta_p}{min} \frac{1}{2}\sum_{i=1}^{n}(y_i - \sum_{j=1}^{p}\beta_j x_{i,j})^2 + \lambda\sum_{j=1}^{p}|\beta_j|$$

Le paramètre λ indique le poids de la pénalisation sur la valeur absolue des paramètres. S'il est élevé, il conduira à ne retenir que peu de paramètres non nuls, s'il est faible, plus.

Pour choisir les valeurs des hyperparamètres, on va estimer le modèle et le tester pour différentes valeurs des hyperparamètres en procédant par *validation croisée*. L'échantillon test ne doit pas être utilisé lors de cette phase. Il s'agit de découper les données de l'échantillon d'entraînement afin de pouvoir entraîner et tester plusieurs fois le modèle sur des jeux partiellement différents. Cette méthode est utilisée afin de rendre plus robuste l'évaluation des performances du modèle. Ces points sont présentés avec plus de détails dans les paragraphes suivants.

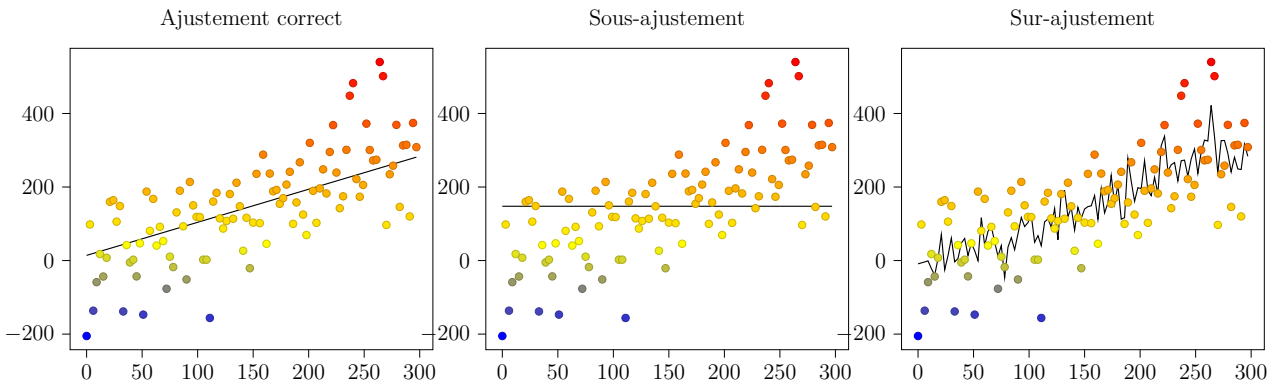
1.1.3 Le dilemme biais-variance

La question de *sous-* ou *sur- apprentissage* s'inscrit dans le cadre d'un dilemme biais-variance. Le graphique 4 page suivante illustre ce problème. On y cherche à ajuster les points de la relation $y = 3 \times x + \epsilon$ où $\epsilon \sim \mathcal{N}(0, 1)$. On utilise une régression pénalisée de type Lasso avec comme variables explicatives des puissances de x à différents degrés. La fonction à minimiser s'écrit :

$$\min_{\beta_1,\ldots,\beta_p} \frac{1}{2}\sum_{i=1}^{n}(\sum_{j=1}^{p}\beta_j x_i^j - y_i)^2 + \lambda\sum_{j=1}^{p}|\beta_j|$$

Lorsque l'hyperparamètre λ est élevé, la pénalisation des coefficients non nuls est très forte. Ajouter une variable ne sera possible que si elle réduit fortement la moyenne quadratique des erreurs. Pour λ très élevé, seule la constante figure dans le modèle : tous les autres coefficients sont nuls. Le modèle ignore la volatilité des données et donne une réponse constante. La variance des estimations est nulle mais le biais est maximal. Le modèle se trompe pour les petites et les grandes valeurs. C'est un cas extrême de *sous-apprentissage*. Il n'y a pas assez de paramètres pour permettre d'ajuster correctement la relation entre les données x et y.

A l'opposé, si le paramètre λ est très faible, beaucoup de coefficients sont non nuls. La courbe estimée est très volatile. La variance est forte. Le modèle modélise une partie du bruit de l'échantillon d'entraînement. Il risque de donner de mauvais résultats sur l'échantillon test. Il *sur-apprend*.



GRAPHIQUE 4 – **Ajustements des points $y_i = 3 \times x_i + \epsilon_i$ par une régression Lasso avec différentes pénalisations**

L'enjeu est alors de trouver un bon arbitrage entre biais et variance. La relation entre le biais, la variance et la complexité du modèle est schématisée dans le schéma 5 page 19. On peut formaliser cette idée dans un cadre plus général. Appelons $\mathbb{P}$ la vraie distribution (inconnue) de (X, Y) et $\mathcal{L}(f)$ le risque de f, c'est-à-dire l'espérance sur $\mathbb{P}$ de la perte L d'une fonction $f(x)$: $\mathcal{L}(f) = \mathbb{E}_{\mathbb{P}}(L(f(x), y))$. On cherche à s'approcher de la fonction f^* qui minimise $\mathcal{L}(f)$. On appelle f^* la fonction « oracle », qu'on peut rapprocher de « la vraie valeur du paramètre » dans une approche classique. Le minimum de $\mathcal{L}(f)$ représente un résidu, une part d'aléatoire indépassable dans le phénomène étudié. Avec une perte quadratique par exemple, on a $f^* = \mathbb{E}_{\mathbb{P}}(Y|X)$ et $\mathcal{L}(f^*) = Var_{\mathbb{P}}(Y|X)$. Un algorithme d'apprentissage produit un prédicteur $\hat{f}$ à partir d'un échantillon $S = (x_i, y_i)$ de données observées. Pour un algorithme donné, on note $\mathcal{H}$ l'ensemble des prédicteurs possibles, le

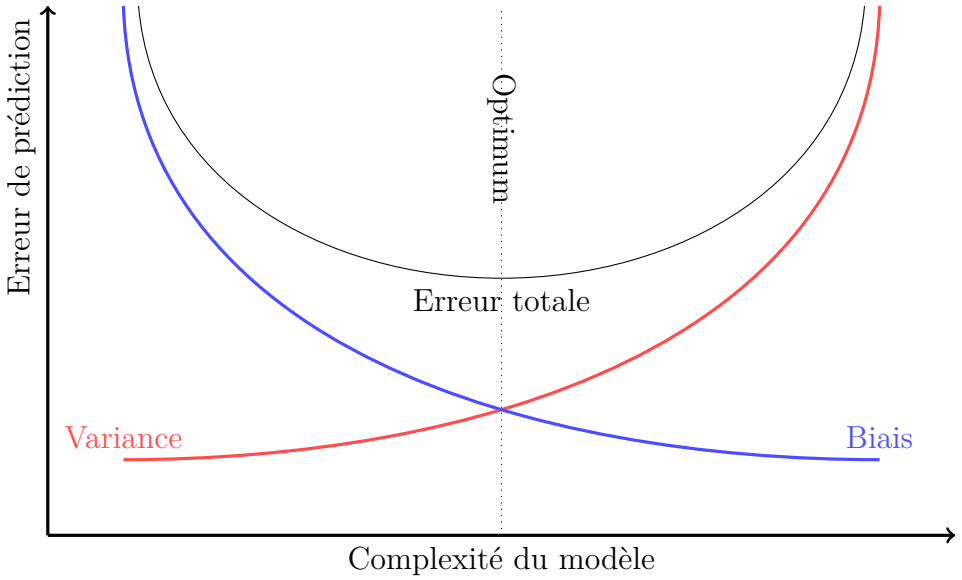
« dictionnaire » des fonctions parmi lesquelles l'algorithme va piocher. L'approche classique restreint fortement cet ensemble. Avec un modèle linéaire par exemple, il s'agit de l'ensemble des fonctions affines de x, et l'algorithme (par exemple, les moindres carrés ordinaires) consiste généralement à calculer la fonction affine qui minimise le risque empirique. Deux questions se posent alors : ce « dictionnaire » $\mathcal{H}$ contient-il ou permet-il de s'approcher de l'« oracle » f^* ? Et comment se comporte le prédicteur estimé $\hat{f}$ lorsque l'échantillon varie ? Il s'agit respectivement de la question du biais, et de la variance.

On compare avec l'« oracle » en posant l'espérance du risque additionnel $\mathbb{E}_S(\mathcal{L}(\hat{f}) - \mathcal{L}(f^*))$, l'espérance portant sur l'ensemble des échantillons $S = (x_i, y_i)_{i=1\dots n}$ possibles, la distribution suivant la loi $\mathbb{P}$. Cette espérance peut être décomposée en deux termes, en nommant $\bar{f} = \mathbb{E}_S(\hat{f})$ le prédicteur moyen sur l'ensemble de ces échantillons : une partie liée à $\mathcal{H}$ et à l'algorithme : $\mathcal{L}(\bar{f}) - \mathcal{L}(f^*)$. C'est le biais. Et une partie liée à la variabilité d'un échantillon à l'autre : $\mathbb{E}_S(\mathcal{L}(\hat{f})) - \mathcal{L}(\bar{f})$. C'est la variance. Lorsque la fonction de perte est l'erreur quadratique, on peut décomposer ainsi l'espérance du risque additionnel (en omettant l'observation x pour alléger la notation) :

$$\mathbb{E}_S(\mathbb{E}_{\mathbb{P}}((\hat{f} - y)^2) - \mathbb{E}_{\mathbb{P}}((f^* - y)^2)) = \underbrace{\mathbb{E}_{\mathbb{P}}((\bar{f} - f^*)^2)}_{Biais^2} + \underbrace{\mathbb{E}_{\mathbb{P},S}((\hat{f} - \bar{f})^2)}_{Variance}$$

Ces quantités restent purement théoriques, quand en pratique on ne connaît pas la fonction oracle et que l'on n'observe pas les espérances sur la totalité des échantillons possibles. Dans l'usage, on observe les erreurs de prédiction sur l'échantillon d'entraînement, sur le ou les échantillons de validation et sur l'échantillon test. L'erreur d'entraînement, observée sur l'échantillon d'entraînement, reflète à la fois le biais de l'algorithme, l'erreur irréductible, et le surapprentissage. Sur l'échantillon test, l'erreur reflète à la fois le biais, l'erreur irréductible, et la variance. La différence entre les deux, ou erreur de généralisation de l'algorithme, est donc liée à sa variance. On parle de dilemme biais-variance parce qu'en général, lorsqu'on modifie un algorithme pour diminuer son biais, par exemple en diminuant le paramètre λ ci-dessus, on augmente sa sensibilité à l'échantillon d'entraînement, et donc sa variance. Inversement, pour rendre un algorithme plus robuste au sur-apprentissage, on va chercher à le régulariser, le simplifier, en prenant ainsi le risque d'augmenter son biais.

Dans le schéma 5 page suivante, les modèles statistiques ou économétriques traditionnels se situent à gauche. Ils sont plutôt simples (peu de paramètres au total à estimer). Ils ont des biais potentiellement importants (par exemple lorsqu'un modèle linéaire vise à représenter un phénomène qui, en réalité, ne l'est pas), mais une variance faible et généralement calculable sous certaines hypothèses, ce qui explique qu'il est moins utile et plus rare de mesurer l'erreur de prédiction sur un échantillon test pour ces modèles.



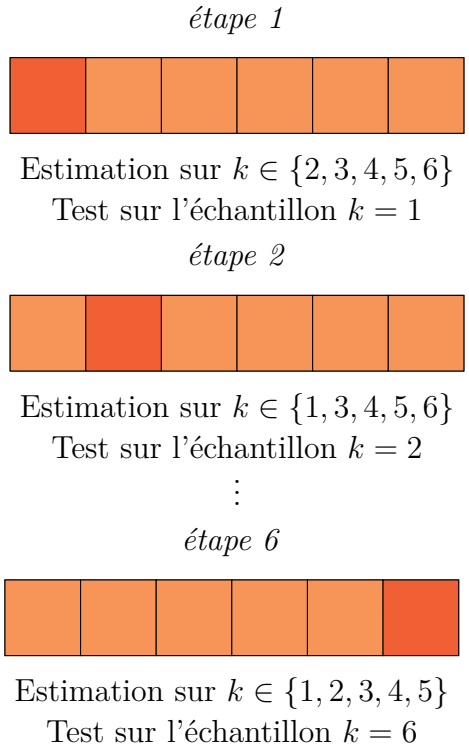
GRAPHIQUE 5 – **Variations du biais et de la variance en fonction de la complexité du modèle**

1.1.4 L'ajustement par validation croisée

En pratique, on peut souvent arbitrer entre biais et variance en modifiant des hyperparamètres du modèle, comme le paramètre λ du Lasso dans l'exemple illustré par le graphique 4. Le jeu de données est le plus souvent séparé en trois : le premier jeu de données, l'échantillon d'entraînement, sert à l'estimation, le deuxième, l'échantillon de validation, sert à la sélection des bons hyperparamètres et le troisième, l'échantillon test, sert à l'évaluation finale de la performance, une fois les hyperparamètres choisis. Utiliser le jeu de test afin de sélectionner les bons hyperparamètres biaiserait les résultats en faisant implicitement rentrer de l'information à propos de l'échantillon test dans le processus de sélection de modèle. L'évaluation des performances serait alors trop optimiste.

Lorsque l'échantillon est trop petit, les résultats obtenus peuvent être instables. La validation croisée permet alors de généraliser ces principes pour gagner en robustesse et fiabilité en multipliant les échantillons d'apprentissage et les échantillons de validation, tout en respectant toujours le fait que les performances d'un modèle, pour des hyperparamètres fixés, sont évaluées uniquement sur les données qui n'ont pas été utilisées pour son entraînement. L'idée est similaire à celle du bootstrap en statistique classique, par exemple pour estimer la variance d'un estimateur.

La démarche consiste à partager aléatoirement les données disponibles en K morceaux (on parle de K *folds* ou encore de plis) et, à tour de rôle, à estimer les paramètres sur certains plis et à évaluer la performance sur les plis restants (voir le graphique 6 page suivante pour un résumé). Les mesures de performance obtenues sur les K échantillons



GRAPHIQUE **6 – Démarche de type "K-fold" ou validation croisée**

sont ensuite moyennées pour être comparées.

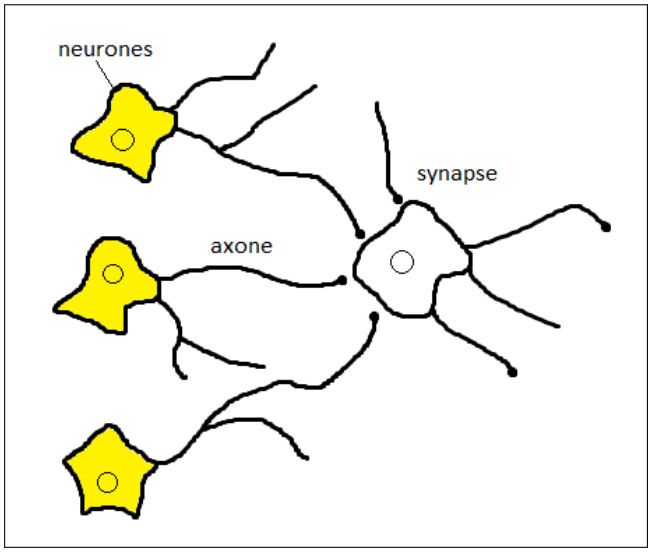
Cette démarche peut être conduite sur les données de l'échantillon d'entraînement pour tous les hyperparamètres d'une grille de valeurs. Chaque combinaison d'hyperparamètres fait l'objet de K évaluations du modèle. Il est ainsi possible de sélectionner le meilleur jeu d'hyperparamètres en comparant les moyennes des K mesures de performance. Le modèle final, mobilisant le meilleur jeu d'hyperparamètres, peut ensuite être testé sur l'échantillon test (n'ayant pas servi lors de la validation croisée).

1.2 Les principes des réseaux de neurones

Les *réseaux de neurones* relèvent du *machine learning*. Ce sont des systèmes complexes, impliquant souvent de nombreux paramètres ou coefficients à estimer et de nombreuses compositions de fonctions. Ils sont apparus, au moins de façon conceptuelle, dès les années 40. A l'origine, le fonctionnement d'un réseau de neurones a été inspiré par celui du cerveau humain. Les neurones humains reçoivent les signaux (impulsions électriques) par des extensions très ramifiées de leur corps cellulaire (les dendrites) et envoient l'information par de longs prolongements (les axones) comme le montre le schéma 7 page suivante.

1.2.1 Le perceptron

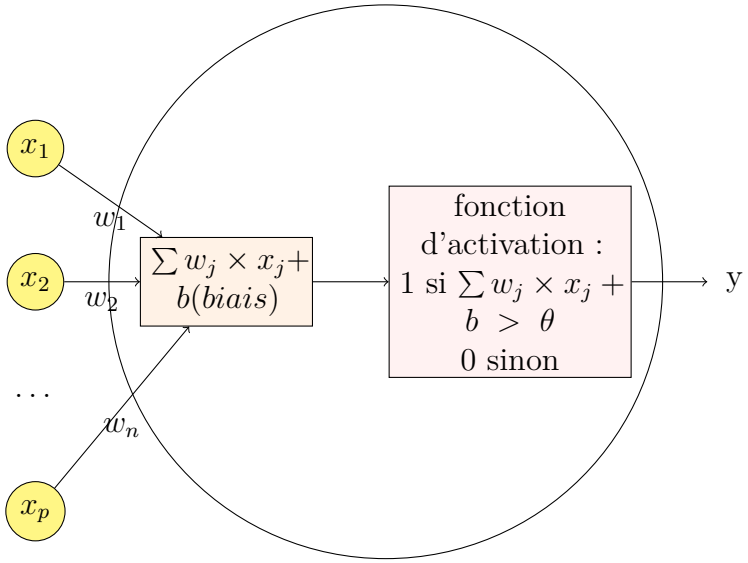
McCulloch et Pitts (1943) introduisent un modèle de neurone formel, le *perceptron*, censé reproduire le comportement d'un neurone biologique. Le neurone formel agit comme un automate doté d'une fonction d'activation qui transforme ses entrées en sortie selon des règles précises. Il effectue par exemple la somme pondérée des entrées et compare la somme résultante à une valeur seuil. Si la valeur obtenue est supérieure au seuil, le neurone s'active (voir schéma 8 page suivante). McCulloch et Pitts (1943) montrent que des neurones formels simples peuvent être combinés en *réseau* afin de réaliser des fonctions complexes. Un point important pour un usage pratique d'un tel système est la fixation des *pondérations* des entrées, les « poids *synaptiques* », dit autrement les *paramètres* du réseau. Ce problème trouve un début de réponse grâce à Hebbs (1949) qui propose une règle simple permettant de modifier la valeur des coefficients synaptiques (les *poids*) en fonction de l'*activation* des neurones. Rosenblatt (1958) crée le premier système artificiel, le *perceptron*, à l'origine une machine, capable d'apprendre par expérience. Le perceptron est un classifieur binaire à l'origine, mais son principe peut facilement être adapté au cas continu.



GRAPHIQUE **7 – schéma d'un neurone biologique**

Le *perceptron* (*neurone*) se définit donc comme l'application d'une *fonction d'activation* à une combinaison linéaire de données (x) en entrée, pondérée par des *poids* (*paramètres*), et d'un *biais* b (autre paramètre qui s'apparente à la constante dans un modèle probabiliste). Les poids sont calibrés/choisis/estimés afin que le réseau puisse s'activer dans des cas bien précis.

Le perceptron se comporte comme un algorithme (linéaire) d'apprentissage supervisé. Les poids neuronaux et le biais (les paramètres) sont estimés sur un échantillon de données pour lesquelles le label y est connu par un algorithme itératif qui minimise la fonction de coût mesurant l'écart entre la sortie estimée $\hat{y}$ et la vraie valeur y. Une fois entraîné, le



Ci-dessus : La sortie du *perceptron* est activée ($y = 1$) lorsque la combinaison linéaire des entrées $x_1, x_2, \ldots, x_p$ est supérieure à un seuil θ, $y = 0$ sinon. La fonction seuil peut être remplacée par d'autres fonctions d'activation : la tangente $tan(\sum w_j \times x_j + b)$, la fonction logistique $(\frac{1}{1+e^{\sum w_j \times x_j + b}})$, ...

GRAPHIQUE 8 – **Le perceptron**

système peut alors prédire la valeur de la variable y de sortie sur un ensemble de données non labellisées.

Dans le cas où y est continu, où la fonction de coût est l'erreur quadratique moyenne et où la fonction d'activation la fonction identité, le programme minimise la quantité :

$$\frac{1}{n}\sum_i (\sum_j w_j x_{i,j} + b - y_i)^2$$

Cela revient à effectuer une régression par moindres carrés ordinaires des y_i sur les $x_{i,j}$. Si y est une variable dichotomique, la fonction de coût est l'entropie croisée et la fonction d'activation la fonction sigmoïde, le programme de minimisation revient à estimer une régression logistique.

1.2.2 Un modèle plus complexe : le perceptron multicouche

Selon Minsky et Papert (1969), une des grandes limites du perceptron est son incapacité à traiter des problèmes non linéaires. Le perceptron multicouche (*multi-layer perceptron*, MLP) permet de dépasser cette limite. Ce système s'inspire du perceptron en le complexifiant. L'idée est de composer un *réseau* en répartissant les neurones en *couches* successives, ce qui permet d'organiser leurs connexions. Le réseau comporte une couche d'entrée où

chaque neurone prend la valeur d'une caractéristique x_i. Il possède également une couche de sortie avec les valeurs estimées $\hat{y}_i$. Les couches intermédiaires sont appelées *couches cachées*. Les neurones d'une couche ne sont reliés qu'à ceux des couches adjacentes. Ainsi, les sorties d'une couche sont les entrées de la couche suivante. L'information circule de proche en proche, couche par couche, de la couche d'entrée à la couche de sortie (on parle de réseau *feedforward*). L'empilement des couches de neurones, munis de fonctions d'activation non linéaires, mais généralement continues et dérivables, permet d'introduire des transformations non linéaires dans le modèle. Le nombre de couches définit la *profondeur* du réseau. On parle d'*apprentissage profond* (*deep learning*) lorsqu'on accumule ces couches cachées, mais une seule couche cachée suffit déjà à introduire des interactions non-linéaires permettant le traitement de problèmes non-linéaires. Enfin, chaque neurone d'une couche est généralement lié à l'ensemble des neurones de la couche précédente, on parle de *couche dense*. D'autres structures sont possibles, notamment dans le cas des RN convolutifs spécialisés dans le traitement des images, présenté au chapitre 3. L'architecture globale est résumée sur le schéma 9 page suivante. Les neurones de la première couche cachée $(C_1, \ldots, C_q)$ sont reliés à toutes les valeurs d'entrée c'est-à-dire les neurones $(E_1, E_2, \ldots, E_p)$ par l'intermédiaire de poids $w_{j,l}$. Les neurones C_l de la première couche cachée fonctionnent à la manière du perceptron :

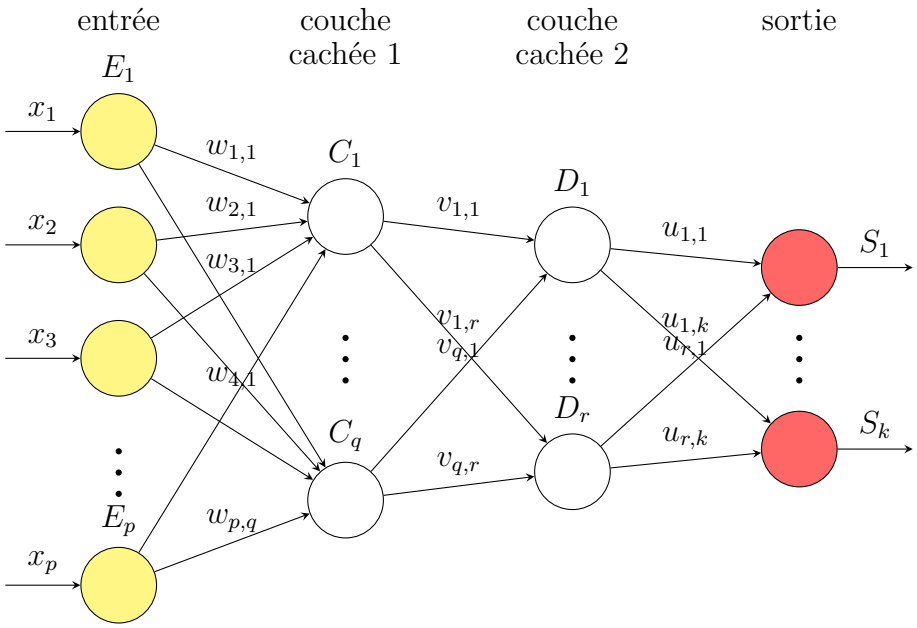
1. Ils sont caractérisés par les poids $w_{j,l}$ avec lesquels ils effectuent la combinaison linéaire des entrées x_j
2. Ils possèdent tous une fonction d'activation f qui effectue une transformation de la combinaison linéaire $\sum w_{j,l} \times x_j + b_l$, telle que la sortie du neurone $C_l = f(\sum w_{j,l} \times x_j + b_l)$

Les neurones de la deuxième couche cachée $(D_1, \ldots, D_r)$ prennent en entrée les sorties de la couche précédente. La sortie du réseau se déduit à partir des activations des neurones $S_1, \ldots, S_k$.

Les fonctions d'activation Les fonctions d'activation les plus courantes figurent dans le tableau 1 page 25. Les critères de choix de ces fonctions diffèrent selon que le neurone se situe en sortie du réseau ou à l'intérieur, dans une couche cachée. On choisira dans tous les cas des fonctions d'activation satisfaisant certaines propriétés (dérivabilité,...) de façon à faciliter l'optimisation (le calcul du gradient), et donc l'estimation des paramètres.

Pour une couche de sortie, on va choisir une fonction d'activation en cohérence avec le type de la variable à prédire et la forme retenue pour la fonction de coût. Par exemple, la fonction *identité* en sortie est adaptée aux problèmes de régression (prédiction d'une variable continue). La fonction *sigmoïde* s'utilise pour les problèmes de classification binaire. La fonction *softmax* qui s'applique simultanément à toutes les sorties et permet de les normaliser de façon à les rendre homogènes à une probabilité avec une somme égale à 1, pour les problèmes de classification comportant plus de deux classes.

D'autres fonctions permettent de borner la sortie à certaines valeurs telles que la fonction *seuil* ou la fonction *tanh*. La première n'est pas dérivable en 0 ce qui peut poser



GRAPHIQUE 9 – **Exemple de perceptron multicouche**

des problèmes d'estimation. La seconde est plus pentue autour de zéro que la fonction sigmoïde. Les gradients sont plus importants dans cette zone. Cela limite le risque de voir s'annuler (« disparaître ») le gradient au cours de l'estimation. Son usage en sortie nécessite cependant certaines précaution avec l'entropie croisée comme fonction de coût.

Dans les couches cachées, les fonctions d'activation servent à introduire des non-linéarités. Les fonctions *sigmoïde*, *tanh* ou encore *ReLu* peuvent être indiquées. La fonction *ReLu* a la particularité de parfois rendre inopérants certains neurones ce qui peut être souhaitable pour des raisons de sparsité. Quand c'est au contraire non souhaité, on a recours à la fonction *Leaky ReLu*.

La fonction de coût et son gradient Le choix de la fonction de coût, somme des pertes individuelles, est crucial car c'est elle qui définit l'objectif fixé au réseau lors de l'entraînement, et donc les caractéristiques de ses prédictions. Elle doit être dérivable (presque partout) : c'est son *gradient*, fonction des poids (paramètres), qui doit être calculé puis mis à jour pour la minimiser. Elle est choisie pour être compatible avec la fonction d'activation de la dernière couche, notamment en ayant un ensemble de définition qui inclut l'image de cette dernière activation (voir tableau 2 page 26).

Pour les problèmes de classification binaire, il n'y a en général qu'une seule sortie S_1 homogène à une probabilité p, à valeur dans $]0,1[$, comme la fonction sigmoïde. Dans ce cas, il est usuel d'utiliser l'*entropie croisée (binaire)* :

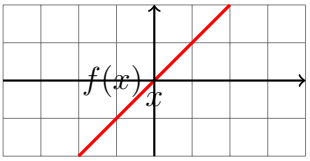
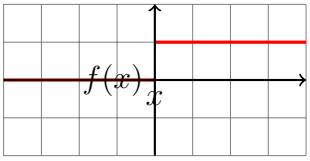
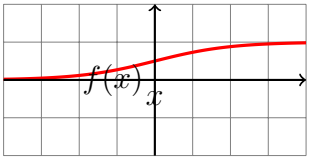
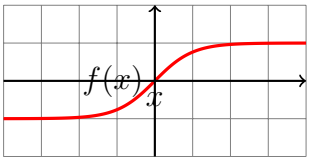
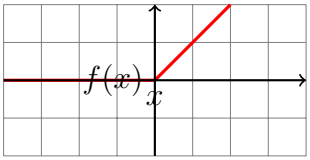
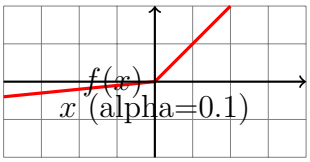
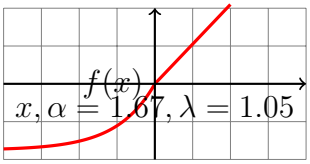
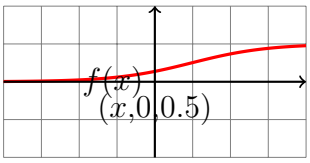
La fonction identité $f(x) = x$	
La fonction seuil $f(x) = \mathbb{1}_{[0,+\infty[}$	
La fonction sigmoïde $f(x) = \frac{1}{1+e^{-x}}$	
La fonction tanh $f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$	
La fonction *ReLu* $f(x) = max(0, x)$	
La fonction *Leaky ReLu* $f(x) = \begin{cases} \alpha x & si\ x \leq 0 \\ x & si\ x > 0 \end{cases}$	
La fonction *SELU* $f(\alpha, x) = \lambda \begin{cases} \alpha(e^x - 1) & si\ x \leq 0 \\ x & si\ x > 0 \end{cases}$	
La fonction *softmax* $f(x_1, \ldots, x_k)_p = \frac{\exp x_p}{\sum_{j=1}^{k} \exp x_j}$	

TABLEAU 1 – **Fonctions d'activation usuelles**

$$Q(\ldots) = \frac{1}{n}\sum_{i=1}^{n}(y_i \times log(S_{1,i}(\ldots)) + (1-y_i) \times log(1 - S_{1,i}(\ldots)))$$

Cette fonction peut être généralisée au cas d'une classification multiple avec des sorties $S_1, \ldots, S_k$. On obtient alors l'*entropie croisée (catégorielle)* :

$$Q(\ldots) = \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{k}(y_{i,j} \times log(S_{j,i}(\ldots)))$$

Pour des problèmes de régression où le label y est continu, la fonction d'activation de la sortie est souvent l'identité, on utilise généralement la perte quadratique, la fonction de coût est alors l'erreur quadratique moyenne :

$$Q(\ldots) = \frac{1}{2}\sum_{i=1}^{n}(y_i - S_i(\ldots))^2$$

On peut aussi utiliser l'erreur absolue moyenne [13]. La meilleure prédiction est alors la médiane (conditionnelle), et non plus la moyenne.

Type de problème	Activation en sortie	Fonction de coût
classification binaire	sigmoid	entropie croisée (binaire)
classification multiclasse (1 seul choix possible parmi plusieurs)	softmax	entropie croisée (catégorielle)
regression (dans $\Re$)	identité	erreur quadratique moyenne

TABLEAU **2 – Choix usuels de l'activation de la dernière couche et de la fonction de coût**

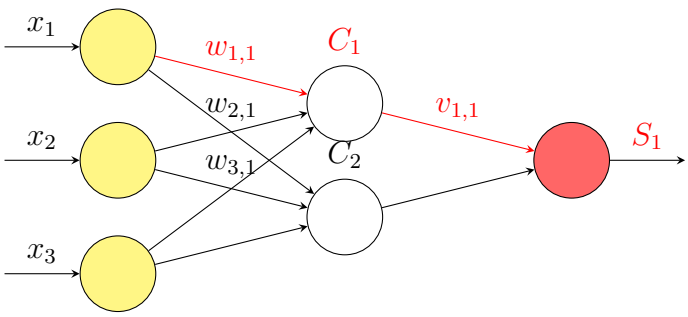
1.2.3 Comment estimer un réseau de neurones ?

Expression de la sortie à partir des entrées Prenons l'exemple d'un réseau de neurones avec 3 entrées x_1, x_2 et x_3, une seule couche cachée de seulement 2 neurones C_1 et C_2 et 1 seul neurone en sortie S_1 (voir Graphique 10 page suivante).

Les valeurs des noeuds des couches successives d'un perceptron multicouche s'écrivent en fonction des valeurs des couches précédentes. Ainsi, la valeur du noeud C_1 de la couche cachée s'écrit en fonction des valeurs des neurones de la couche en entrée (x_1, x_2 et x_3) :

$$C_1 = f_c(bc_1 + \sum_{l=1}^{3} w_{l,1}x_l)$$

13. La valeur absolue n'est pas dérivable en zéro, mais elle est dérivable presque partout, ce qui est suffisant en pratique. Les logiciels qui implémentent cette fonction de coût gèrent aussi le cas limite, par exemple en assignant une dérivée nulle en zéro



GRAPHIQUE **10 – Un réseau simple à trois couches**

où les $w_{l,1}$ sont les poids entre le neurone d'entrée l avec la valeur x_l et le neurone 1 de la couche cachée, f_c est la fonction d'activation du neurone de la couche cachée et bc_1 est la constante associée. Il est alors facile de déduire la valeur de sortie $\hat{S}_1$ de façon récursive en fonction des poids des couches successives et des valeurs d'entrée. Dans notre exemple, la valeur du noeud de sortie S_1 peut se déduire des valeurs d'entrée par la formule :

$$S_1(bc_1, bc_2, w_{l=1...3,j=\{1,2\}}, v_{1,1}, v_{2,1}, bs_1, x_{k=1...3}) = f_s(bs_1 + \sum_{m=1}^{2} v_{m,1} C_m)$$

avec

$$C_m(bc_m, w_{k=1...3,m}) = f_c(bc_m + \sum_{k=1}^{3} w_{k,m} x_k), \qquad \forall m = 1,\ 2$$

Comme on dispose d'un échantillon $x_{i=1...n,k=1...3}$ pour lequel les valeurs y_i en sortie du neurone S_1 sont connues. On cherche alors les poids $w_{l=1...3,k=\{1,2\}}$,bc_1,bc_2 et $v_{m=\{1,2\},1}$,bs_1 qui minimisent la fonction de coût Q. Ici, la variable de sortie est continue. La fonction de coût Q choisie est la moyenne des carrés des écarts entre les valeurs estimées $\hat{y_{1,i}}$ et les vraies valeurs y_i :

$$Q(bc_1, bc_2, w_{l=1...3,j=\{1,2\}}, v_{1,1}, v_{2,1}, bs_1) = \frac{1}{n} \sum_{i=1}^{n} (S_1(bc_1, bc_2, w_{l=1...3,j=\{1,2\}}, v_{1,1}, v_{2,1}, bs_1, x_{i,k=1...3}) - y_i)^2$$

ou encore

$$Q(bc_1, bc_2, w_{l=1\ldots3,j=\{1,2\}}, v_{1,1}, v_{2,1}, bs_1) =$$
$$\frac{1}{n}\sum_{i=1}^{n}(f_s(bs_1 + v_{1,1}f_c(bc_1 + \sum_{k=1}^{3} w_{k,1}x_{i,k}) + v_{2,1}f_c(bc_2 + \sum_{k=1}^{3} w_{k,2}x_{i,k})) - y_i)^2$$

La rétropropagation de l'erreur. L'estimation de systèmes multicouches a été rendue possible par la mise au point de l'algorithme de rétropropagation de l'erreur. Cet algorithme a d'abord été introduit dans Werbos (1974), Parker (1985) et Le Cun (1985). Il a ensuite été mis au point par Rumelhart, Hinton, et Williams (1986). Les méthodes de retropropagation permettent de calculer le gradient de la fonction de coût Q en fonction des poids du réseau et des valeurs observées en sortie. La rétropropagation consiste à calculer l'erreur du réseau en partant de la fin et en la propageant vers l'entrée du réseau de neurones. On choisit des poids initiaux et on calcule une première fois les valeurs de sortie du réseau (les $\hat{y}_i$). On va ensuite chercher à modifier les poids $w_{k,p}, bc_p$ et $v_{k,1}, bs_1$ de façon à minimiser l'écart entre les y_i et les valeurs estimées par le réseau.

La contribution de l'observation i à l'erreur quadratique finale s'écrit :

$$Q_i(bc_1, bc_2, w_{l=1\ldots3,j=\{1,2\}}, v_{1,1}, v_{2,1}, bs_1) =$$
$$(f_s(bs_1 + v_{1,1}f_c(bc_1 + \sum_{k=1}^{3} w_{k,1}x_{i,k}) + v_{2,1}f_c(bc_2 + \sum_{k=1}^{3} w_{k,2}x_{i,k})) - y_i)^2$$

Lors d'une petite modification du poids $v_{1,1}$, elle est modifiée :

$$\frac{\partial Q_i}{\partial v_{1,1}} =$$
$$2f_c(bc_1 + \sum_{k=1}^{3} w_{k,1}x_{i,k})f_s'(bs_1 + v_{1,1}f_c(bc_1 + \sum_{k=1}^{3} w_{k,1}x_{i,k}) + v_{2,1}f_c(bc_2 + \sum_{k=1}^{3} w_{k,2}x_{i,k}))$$
$$\times (f_s(bs_1 + v_{1,1}f_c(bc_1 + \sum_{k=1}^{3} w_{k,1}x_{i,k}) + v_{2,1}f_c(bc_2 + \sum_{k=1}^{3} w_{k,2}x_{i,k})) - y_i)$$

soit

$$\frac{\partial Q_i}{\partial v_{1,1}} = C_{1,i}\delta_{1,i}^{sortie}$$

en notant $\delta_{1,i}^{sortie} = 2(f_s(bs_1 + \sum_{m=1}^{2} v_{m,1}C_{m,i}) - y_i)f_s'(bs_1 + \sum_{m=1}^{2} v_{m,1}C_{m,i})$

On cherche maintenant à propager l'erreur obtenue au niveau de la couche de sortie vers la couche d'entrée. On considère une modification du poids $w_{1,1}$. Cette modification a

un impact sur les valeurs $C_{1,i}$ qui ont elles-mêmes un impact sur les valeurs en sortie $S_{1,i}$. Si on dérive la contribution de i à la fonction de coût par rapport à $w_{1,1}$, on a :

$$\frac{\partial Q_i}{\partial w_{1,1}} = 2(f_s(bs_1 + \sum_{m=1}^{2} v_{m,1}C_{m,i}) - y_i)f_s^{'}(bs_1 + \sum_{m=1}^{2} v_{m,1}C_{m,i})\frac{\partial C_{1,i}}{\partial w_{1,1}}v_{1,1}$$

soit

$$\frac{\partial Q_i}{\partial w_{1,1}} = \delta_{1,i}^{sortie}\frac{\partial C_{1,i}}{\partial w_{1,1}}v_{1,1},$$

avec

$$\frac{\partial C_{1,i}}{\partial w_{1,1}} = f_c^{'}(bc_1 + \sum_{k=1}^{3} w_{k,1}x_{k,i})x_{1,i}.$$

Le calcul de ces contributions marginales venant de l'observation i à l'erreur de sortie peut facilement être généralisé à des réseaux de n couches. Une fois l'erreur explicitée il est désormais possible d'optimiser la fonction de coût et d'estimer les paramètres en utilisant les algorithmes de descente de gradient.

Les algorithmes de descente de gradient. Les algorithmes de descente de gradient s'appliquent bien au-delà des RN. Ils s'attachent à mettre à jour les poids en fonction du gradient de façon itérative afin de minimiser la fonction de coût [14]. Les logiciels en proposent plusieurs tels que *adam*, *RMSprop* ou *SGD*. Ruder (2016) détaille les principaux. Tous cherchent à minimiser rapidement la fonction de perte en évitant certains problèmes comme l'annulation du gradient sur un large éventail de valeurs (on parle alors de *disparition*), sa non-convergence ou l'instabilité des résultats.

L'algorithme le plus simple est la *descente de gradient*. De façon usuelle, les poids sont mis à jour de façon itérative après chaque estimation de la fonction de coût sur l'ensemble de l'échantillon.

$$w_{j,k}^{t+1} = w_{j,k}^{t} - \eta \times \frac{\partial Q}{\partial w_{j,k}^{t}}$$

où η est le *taux* (ou encore le *pas*) *d'apprentissage*. Il permet de contrôler la vitesse à laquelle l'algorithme converge vers la solution. Un pas trop petit risque de rendre l'algorithme long à converger tandis qu'un pas trop grand peut créer des oscillations.

Pour gagner accélérer l'entraînement, on va chercher à mettre à jour les poids en ne mobilisant qu'une *partie des données*. On effectue alors une partition (des batchs) aléatoire

14. pour atteindre un minimum local

des données. On peut mettre à jour les poids après une mise à jour du gradient à chaque observation dans un ordre aléatoire (*Stochastic Gradient Descent*) ou par lot (ou *batch*) de données (*mini-batch Gradient Descent*). Un cycle d'apprentissage (ou *epoch*) est réalisé lorsque le gradient a été mis à jour sur tous les *batchs* qui composent les données, donc lorsque l'ensemble de l'échantillon a été utilisé. Il est possible de recommencer plusieurs fois (plusieurs *epochs*) l'opération sur des partitions aléatoires (batchs) différentes des données initiales. Le choix de la taille des batch fait l'objet d'un arbitrage : un pas d'apprentissage est d'autant plus rapide à calculer, mais d'autant plus aléatoire, que le batch est petit. Le calcul par batch étant optimisé, le temps de calcul est moins que proportionnel à la taille des batch : une *epoch* est d'autant plus rapide que les batch sont grands, mais le nombre de pas d'entraînement par *epoch* est d'autant plus faible. Par ailleurs l'aléas inhérent aux petits batchs pourrait jouer un rôle important de régularisation.

Lorsque le gradient est calculé sur des lots de données, la mise à jour des poids peut devenir très erratique. Pour stabiliser les poids, il est possible d'ajouter dans la mise à jour une prise en compte des gradients passés (*momentum*). Il est aussi possible d'adapter le taux d'apprentissage aux valeurs de ces gradients passés (*RMSprop*, introduit par Tieleman, Hinton, et al. (2012)). L'un des algorithmes les plus populaires actuellement est probablement *Adam* qui mixe ces deux logiques. *Adam* utilise les premier et second moments des gradients passés pour adapter le rythme de modification des poids. Comme dans la méthode *RMSprop*, une pondération est introduite permettant de pondérer les gradients passés par rapport au présent afin d'éviter que leur poids ne devienne trop important :

$$w_{j,k}^{t+1} = w_{j,k}^{t} - \frac{\eta}{\delta + \sqrt{r_i}} v_i$$

où η est le taux d'apprentissage, δ une petite constante servant à assurer la stabilité des estimations (10^{-8} par défaut), r_i est une moyenne mobile des carrés des gradients passés et v_i une moyenne mobile des gradients eux-mêmes :

$$r_i^t = \rho r_i^{t-1} + (1-\rho)(\frac{\partial Q}{\partial w_{j,k}^{t-1}})^2$$

et

$$v_i^t = \gamma v_i^{t-1} + (1-\gamma)\frac{\partial Q}{\partial w_{j,k}^{t-1}}$$

où ρ et γ sont deux hyperparamètres respectivement fixés à 0.999 et 0.9 dans Kingma et Ba (2014). Ces valeurs donnent généralement de bons résultats en pratique. Il n'est pas conseillé de les modifier. La méthode *Adam* est assez robuste au choix des hyperparamètres. Elle est largement utilisée en pratique.

Quelques mots sur l'initialisation des paramètres La bonne convergence peut-être liée au choix des paramètres initiaux. Par exemple, initialiser les paramètres à zéro peut conduire assez rapidement à la disparition du gradient (le gradient devient si faible que les poids n'évoluent plus alors que l'optimum n'est pas atteint). Les valeurs initiales des paramètres peuvent être tirées aléatoirement. Il est possible aussi d'utiliser les poids d'un réseau pré-entraîné sur un sujet proche de celui que l'on souhaite traiter. Dans certaines librairies telles que *Keras*, cette étape est entièrement transparente pour l'utilisateur.

Les techniques de régularisation Comme toute technique d'apprentissage automatique, les réseaux de neurones peuvent être sujets au sur- ou au sous-apprentissage (voir 1.1.3 page 16), d'autant plus qu'ils peuvent facilement comporter plusieurs centaines de milliers de paramètres à optimiser. Dans le cas d'un sur-apprentissage, il est possible de faire appel à des techniques de *régularisation*, qui incitent le modèle à rester "simple" et diminuent son erreur de généralisation (Kukačka, Golkov, et Cremers, 2017). Les méthodes les plus répandues sont :

— Le *dropout*. Une proportion donnée des neurones d'une couche est mise temporairement à zéro de façon aléatoire pendant l'entraînement. Le dropout contraint le réseau à un apprentissage redondant, et le modèle final est plus robuste (Srivastava, Hinton, Krizhevsky, Sutskever, et Salakhutdinov, 2014).

— L'*Early Stopping*. L'apprentissage s'arrête tôt pour favoriser les modèles simples. De cette façon, il ne s'ajuste pas trop au bruit des données d'apprentissage.

— Les *pénalisations*. Il s'agit d'introduire une pénalisation afin de privilégier des modèles simples, en ajoutant à la fonction de coût une pénalité de type L1 : $\lambda \sum_j |w_j|$ ou L2 : $\lambda \sum_j (w_j)^2$, avec (w_j) le vecteur des paramètres. Une pénalisation de type L2 favorisera des coefficients faibles. Une pénalisation de type L1 provoquera l'annulation de certains coefficients (comme dans une régression Lasso).

Les réseaux de neurones ont cependant tendance à se régulariser d'eux-mêmes et ne sur-apprennent pas autant qu'on peut le craindre. Cette tendance n'est pas entièrement expliquée, mais fait l'objet d'une recherche importante depuis la prépublication en 2017 d'un article influent sur le sujet (Zhang, Bengio, Hardt, Recht, et Vinyals, 2021). Le recours à la régularisation n'est donc pas toujours nécessaire.

1.3 Première mise en pratique : entraîner un perceptron multicouche à distinguer un lit d'une chaise

Les librairies *neuralnet* pour R ou *scikit-learn* en Python permettent d'entraîner rapidement un perceptron multicouche [15]. Cependant, pour quiconque souhaite se lancer dans un projet d'envergure, ces librairies sont très vite limitées. On passera alors à des librairies

15. Il existe aussi des outils interactifs tels que *weka* ou *neuroph* sur internet qui permettent d'entraîner des réseaux. Ces outils présentent l'avantage de la simplicité : il n'est pas nécessaire de connaître un langage spécifique mais les méthodes qu'ils proposent sont en revanche limitées. Seuls certains types de réseaux sont disponibles. Tous les paramètres ne sont pas modifiables.

plus avancées en Python [16], avec les librairies *Keras* (surcouche de *tensorflow* développé par Google) et *Pytorch* (initialement développée par Facebook) parmi les plus populaires.

Cette partie montre le fonctionnement des librairies *neuralnet*, *Keras* et *Pytorch* à l'aide d'un exemple : estimer un perceptron multicouche pour distinguer les images de chaises et celles de lits. L'exemple, purement illustratif, ne mobilise qu'un très petit nombre d'observations. Le code est en R [17], et on mobilise les librairies Python depuis R via la librairie *reticulate*. Pour ce faire, R et Python doivent être installés sur la même machine. Les programmes sont disponibles sous forme de fichiers .R dans le repo github [18]. Le problème de classification dans cet exemple (deux classes et une base de données relativement petite) est simple et peut aussi se traiter à partir d'une régression pénalisée. On s'attachera alors à comparer les performances des réseaux à celles de cette dernière.

Données. Les images utilisées ici sont extraites de la base de données d'images CIFAR-100. On importe tout d'abord les données de la base CIFAR-100 à partir de la librairie *Keras* sous *Rstudio*. Les données de la base CIFAR-100 sont déjà séparées en échantillons de test et d'entraînement (*train*). On sélectionne les images de chaises (classe 20) et de lits (classe 5). L'échantillon d'entraînement contient alors 1000 images et celui de test 200. Ces échantillons sont constitués à part égale d'images de chacune des catégories. Les images sont traduites en niveaux de gris puis converties au format numérique. Les données sont ensuite normalisées afin de prendre des valeurs dans $[0, 1]$ [19] et d'être plus facilement utilisables par des modèles d'apprentissage automatique. Les données au format 32×32 sont mises en ligne (1024 valeurs) dans un *data.frame* (voir la table 3). La dernière colonne donne le type (5 pour les lits et 20 pour les chaises).

A data.frame : 2×1025

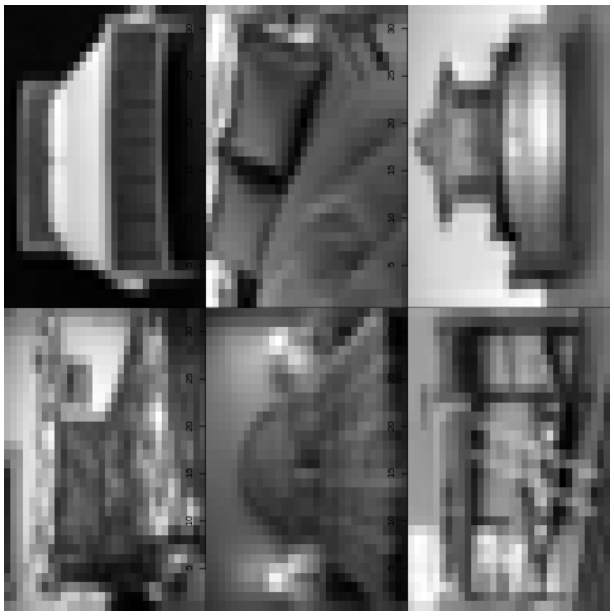
	X1	X2	X3	...	type
	<dbl>	<dbl>	<dbl>	...	<chr>
1	0.02352941	0.01960784	0.01568627	...	5
2	1.00000000	0.97647059	0.98039216	...	5

TABLEAU **3 – Données en entrée**

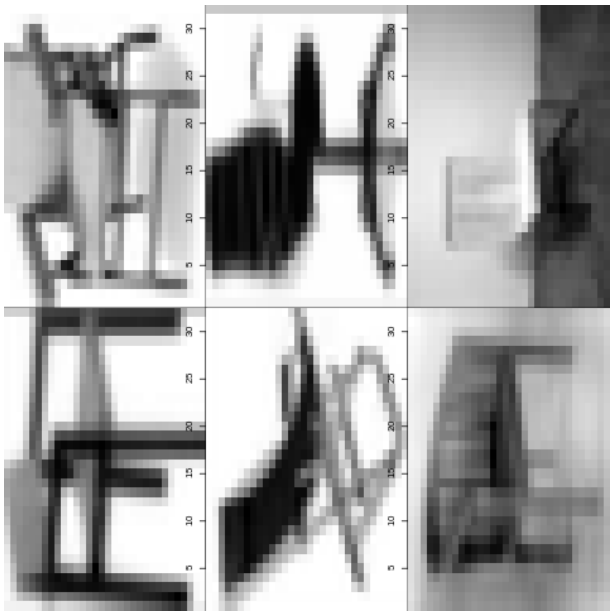
Les valeurs sont en ligne mais il s'agit bien d'images. Il est possible de les remettre au format 32×32 et de les afficher (cf. graphiques 11 page suivante et 12 page 34).

16. A l'Insee, Python est accessible via *pycharm* sous AUS v3. Il peut être installé sur le poste de travail avec *anaconda* (https ://www.anaconda.com) ou winpython (http ://winpython.sourceforge.net/). Pour permettre des calculs parallèles, une troisième solution consiste à utiliser un container jupyter sur la plateforme *datalab.sspcloud.fr* permettant de paralléliser simplement les calculs et utiliser de la GPU (Graphical Processing Unit).
17. Tous les autres exemples présentés dans ce document sont en Python
18. Les scripts du chapitre 1 sont conçus pour être aisément reproductibles, en particulier en utilisant le Datalab SSP Cloud, la plateforme Onyxia hébergée par l'Insee
19. 0 pour noir, 1 pour blanc.

```
par(mfrow=c(2,3))
par(mar=c(0, 0, 0, 0), xaxs='i', yaxs='i')

#selection des images de chaises
atracer = df_train[df_train[,"type"]==20,]

#six premières lignes mises au format image
for (i in 1:6) {
  img <- array_reshape(as.matrix(atracer[i,1:(ncol(atracer)-1)]),dim=c
    ↪ (32,32))
  image(1:32,1:32,img, col = gray((0:255)/255))
}
```



Graphique 11 – **Images de lits**



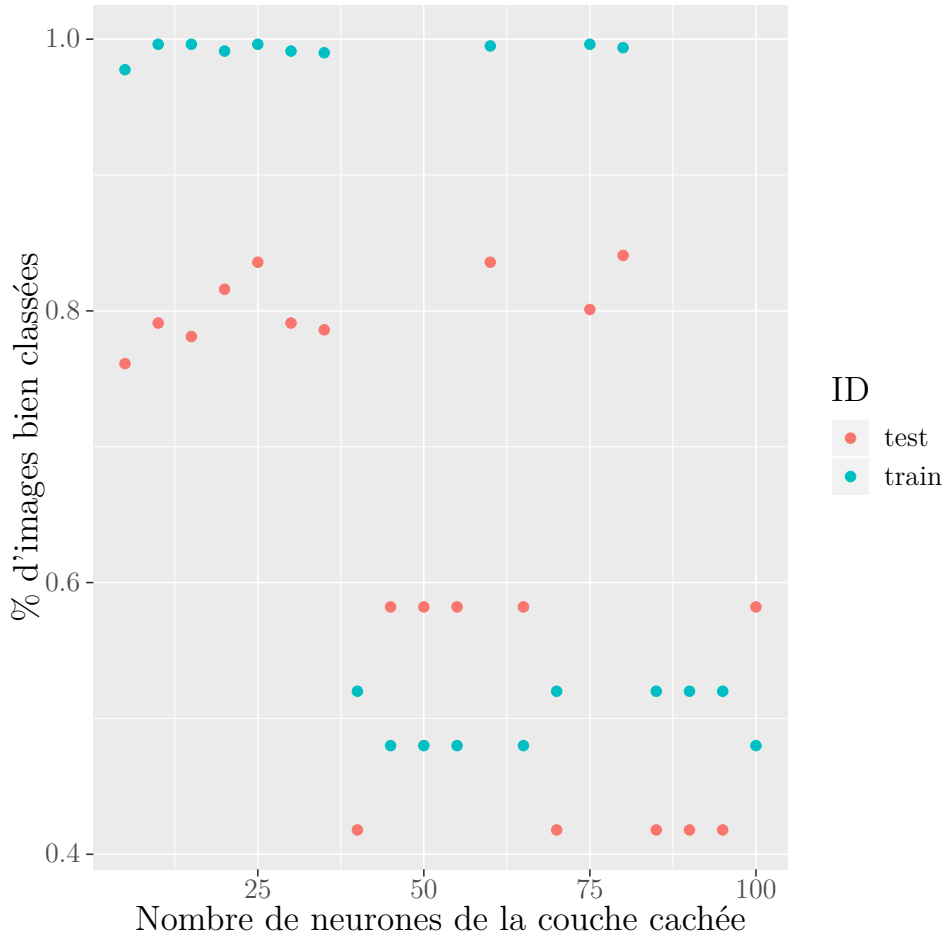
GRAPHIQUE **12 – Images de chaises**

Neuralnet : La syntaxe pour lancer l'estimation en utilisant la librairie *neuralnet* est donnée ci-dessous :

```
# hidden (nombre de neurones dans la couche cachée),
# act.fct (fonction d'activation),
# linear.output (classification),
# likelihood (calcul AIC et BIC),
# threshold (critère d'arrêt),
# lifesign (détail des itérations).
nn <- neuralnet::neuralnet(type=="20" ~ .,
    data=df_train, hidden=25,
    threshold = 0.00001, err.fct = "ce", act.fct = "logistic",
    linear.output = FALSE, likelihood = TRUE, lifesign="full"
    )
```

Cette syntaxe signifie que le réseau cherche à prédire une variable binaire vraie si l'image est un lit (type=="20") au moyen d'un perceptron multicouche avec 25 neurones dans la couche cachée (hidden=25). La fonction d'activation logistique (act.fct="logistic") s'applique derrière chaque neurone y compris le neurone de sortie (linear.output = FALSE). La fonction de coût utilisée est l'entropie croisée (err.fct="ce"). L'option likelihood=TRUE permet de calculer l'AIC et le BIC. Le paramètre *threshold* est un critère de convergence. Il spécifie la valeur maximale des dérivées partielles de la fonction de coût à atteindre pour arrêter les itérations. L'option lifesign permet d'obtenir le détail des itérations. Plus de détails figurent dans la documentation officielle ici.

Avec 25 neurones dans la couche cachée, le réseau atteint une **précision** (pourcentage d'images bien classées sur l'échantillon test) de 81%. Ce nombre de 25 a été ici choisi au hasard. Pour optimiser la valeur de cet hyperparamètre, on va entraîner l'algorithme sur une grille de valeurs possibles (nombre de neurones de la couche cachée allant de 5 à 100). L'échantillon d'entraînement est alors lui même divisé en deux. Un premier échantillon de 800 observations sert à estimer le modèle. Les données restantes (200 observations) servent à évaluer les performances des algorithmes estimés avec les différentes valeurs de l'hyperparamètre. Ici, dès que le nombre de noeuds de la couche cachée est élevé (>35), le réseau s'entraîne mal et devient impuissant à classer les images : elles peuvent être toutes dans la même catégorie (précision de 0.5 dans le graphique 13 page suivante). Le RN ne parvient pas à apprendre efficacement.



GRAPHIQUE **13 – Précision en fonction de la taille de la couche cachée**

La librairie *neuralnet* est finalement plutôt performante pour cet exemple. Pourtant, pour des modèles plus complexes, elle montre très vite ses limites. Seuls des perceptrons multicouches avec une ou plusieurs couches cachées sont mobilisables, avec seulement deux types de fonction d'activation possibles : *tanh* ou la fonction logistique. Ces fonctions d'activation sont spécifiées une fois pour toutes pour l'ensemble des neurones (sauf la couche de sortie si linear.output=TRUE). Des méthodes performantes de minimisation de la fonction de coût telles que *Adam* ne sont pas disponibles.

Keras et Pytorch : La librairie *Keras* permet de construire facilement et rapidement des réseaux de neurones et s'adresse aussi aux non-initiés. La librairie *Pytorch* est plus complexe d'utilisation. Elle requiert de fixer un certain nombre de paramètres (en particulier pour l'estimation) mais permet un accès à toutes les fonctions disponibles et donne donc plus de liberté dans l'architecture et l'estimation du réseau. On reprend le même exemple d'application en mobilisant chacune de ces deux librairies.

1. Paramétrage des données en entrée

Keras

Pour *Keras*, une conversion des données au format numérique suffit.

```
# les variables explicatives sont placées dans une matrice
x_train <- data.matrix(df_train[, 1:(ncol(df_train) - 1)])
# une indicatrice indique si l'image est celle d'un lit
y_train <- keras::to_categorical(ifelse(df_train[, "type"] == "20", 1, 0)
    ↪ )
```

Pytorch

Dans *Pytorch*, les données numériques doivent ensuite être converties en tenseur (objet spécifique en entrée d'un réseau). Le type des données doit être clairement défini.

```
# 2. Conversion des données en vecteurs et matrices R.
x_train <- as.matrix(apply(df_train[, 1:ncol(df_train)-1],2,"as.numeric")
    ↪ )
y_train <- ifelse(df_train$type=="20",2,1)

# 3. Conversion des données en tenseur.
x_train <- torch::torch_tensor(x_train, dtype = torch::torch_float())
y_train <- torch::torch_tensor(y_train, dtype = torch::torch_long())
```

2. Définition du réseau

Keras

Pour *Keras*, le modèle est défini de façon séquentielle, couche par couche. On définit une couche cachée avec 20 neurones (units=20) qui prend en entrée les valeurs des pixels des images (input=1024). Les fonctions d'activation des 20 neurones sont de type *ReLu*. La couche de sortie comprend 2 neurones (units=2). Une fonction *softmax* est utilisée afin d'obtenir deux probabilités.

```
model <- keras::keras_model_sequential()
model |>
  keras::layer_dense(units = 20, activation = 'relu',input=c(1024)) |>
  keras::layer_dense(units = 2) |>
  keras::layer_activation("softmax")
```

Pytorch

Dans *Pytorch*, l'écriture du modèle est un peu plus compliquée. Une première fonction (*initialize*) décrit les différentes couches. Les dimensions d'entrée et de sortie doivent être spécifiées. Ici, le réseau comporte deux couches. La première (*linear1*) prend en entrée

les valeurs des pixels des images (1024) et comporte 20 neurones (nombre de sorties). La seconde (*linear2*) prend en entrée 20 valeurs et comporte deux neurones. La deuxième fonction (*forward*) décrit la façon dont les couches s'enchaînent. Les neurones de la première couche (*linear1*) sont suivies d'une fonction d'activation *ReLu*. Les valeurs calculées sont transmises à la seconde couche (*linear2*). Les deux valeurs en sortie de cette couche sont fournies à une fonction *softmax*.

```
net <- torch::nn_module(
  "class_net",
  initialize = function() {
    self$linear1 <- torch::nn_linear(1024,20)
    self$linear2 <- torch::nn_linear(20,2)
  },
  forward = function(x) {
    x |>
      self$linear1() |>
      torch::nnf_relu() |>
      torch::self$linear2() |>
      torch::nnf_softmax(2)
  }
)

model = net()
```

3. Choix de l'*optimiser* et des paramètres pour l'estimation

Keras

Pour *Keras*, le modèle doit être compilé. Le choix de l'algorithme de minimisation est *Adam*. La fonction de coût choisie est l'entropie croisée (categorical_crossentropy). La métrique utilisée pour évaluer le modèle est la précision (*accuracy*), le pourcentage d'images bien classées.

```
model |> keras::compile(
  optimizer = "adam",
  loss = "categorical_crossentropy",
  metrics = c("accuracy")
)
```

Pytorch

Dans *Pytorch*, on définit la fonction de coût (l'entropie croisée catégorielle) et l'algorithme (*Adam*) pour la minimiser.

```
# Define cost and optimizer
criterion <- torch::nn_cross_entropy_loss()
optimizer <- torch::optim_adam(model$parameters)
```

4. Estimation du modèle

Keras

Pour *Keras*, il suffit d'une ligne de commande.

```
history <- model |> keras::fit(x_train, y_train, epochs = 100, verbose =
    ↪ 2)
```

Pytorch

Dans *Pytorch*, il faut décrire toutes les étapes. Si l'on désire un calcul par *batch* (sous-ensembles) de données, la table initiale doit être répartie dans les *batch* au préalable par la fonction *dataloader*[20].

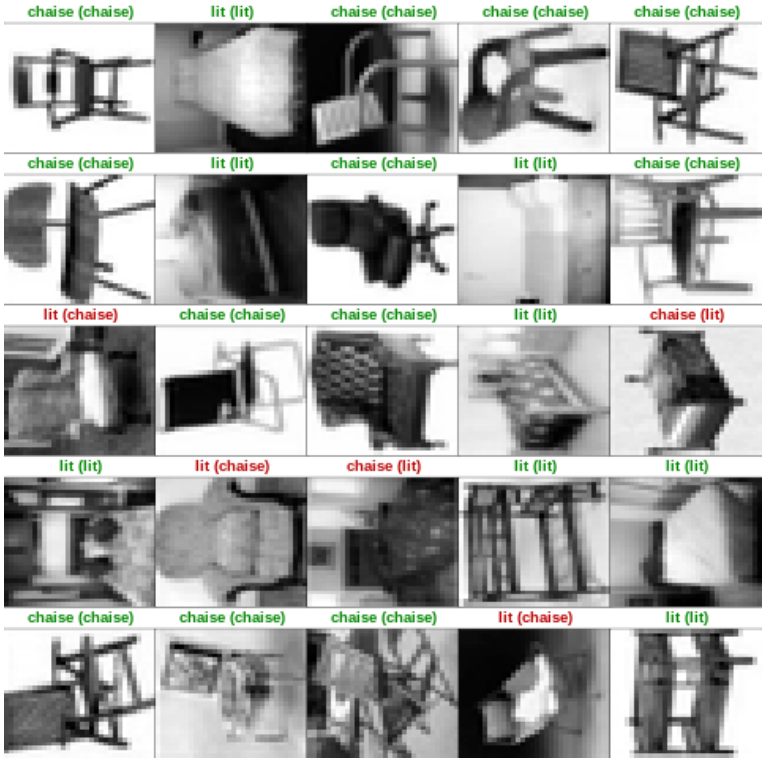
```
epochs <- 200

# Train the net
for (i in 1:epochs) {
  # mise à zéro du gradient
  optimizer$zero_grad()
```

20. Un exemple figure sur le repo *github*.

```
  y_pred <- model(x_train)
  loss <- criterion(y_pred, y_train)

  # retropropagation
  loss$backward()

  # mise à jour des poids
  optimizer$step()

  # Check Training
  if (i %% 10 == 0) {
    winners <- y_pred$argmax(dim = 2)
    corrects <- (winners == y_train)
    accuracy <- corrects$sum()$item() / y_train$size()

    cat("Epoch:", i, "Loss:", loss$item(), "Accuracy:", accuracy, "\n")
  }

}
```

En fixant le nombre de neurones de la couche cachée à 150, on obtient sur l'échantillon test une précision (pourcentage d'images biens classées) de 83 % avec la librairie *Keras* et de 81 % avec la librairie *pytorch*[21]. Il est aussi possible d'afficher les résultats obtenus sur une sélection d'images (cf. graphique 14 page suivante).

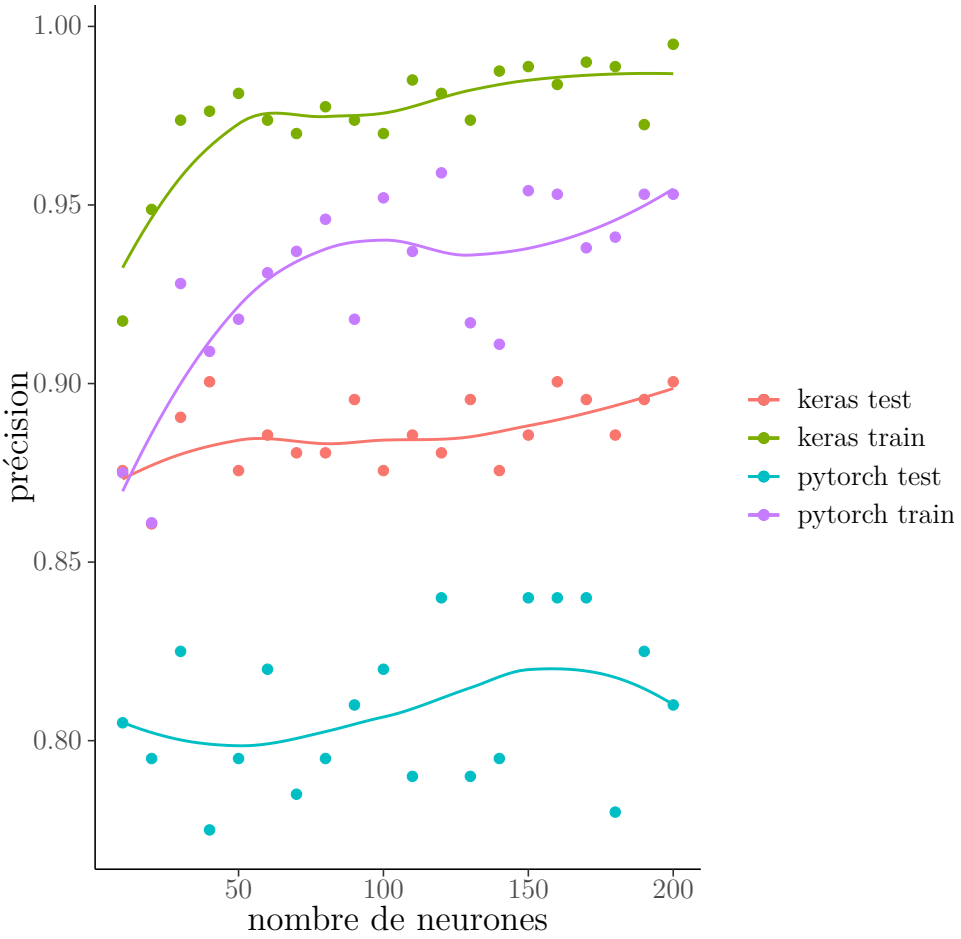
21. On ne fait pas figurer ici le code permettant d'évaluer la performance prédictive sur l'échantillon test.



GRAPHIQUE **14 – Résultats obtenus sur une sélection d'image (Keras)**

Notes : Valeurs prédites et vrais labels (entre parenthèse).

On fait ensuite varier le nombre de neurones de la couche cachée comme précédemment (voir le code dans le fichier .ipynb sous github). Le pourcentage d'images bien classées est donné sur le graphique 15 page suivante. Sur l'échantillon test, les performances sont relativement stables. Dans cet exemple, la librairie *Keras* permet d'atteindre une précision entre 88% et 90% avec les paramètres par défaut, supérieure à la précision atteinte par la librairie *Pytorch*, environ 82%. Pour cette dernière, un travail supplémentaire de *fine tuning* est nécessaire afin d'atteindre le maximum de précision et de rapidité d'exécution (un enjeu important pour des problèmes de plus grande taille que l'exemple illustratif développé ici).



GRAPHIQUE **15 – Sélection du nombre de neurones de la couche cachée**

5. Choix automatisé des hyperparamètres

Certaines librairies *Python* réalisent la recherche d'hyperparamètres de façon automatique telles que *Hyperopt*[22], *SignOpt*[23] ou encore *Ray-tune*[24] qui mixe les deux. Une utilisation de la librairie *Hyperopt* sous R avec *Keras* conduit ici à retenir 150 neurones pour la couche cachée. Le chapitre 3 détaille un autre exemple d'utilisation.

6. Analyse du fonctionnement du réseau

Une dernière étape consiste à étudier *a posteriori* ce qui va sous-tendre les décisions principales du réseau. On ne va pas rentrer dans le détail de ces méthodes dans le cadre de cet exemple introductif. Le lecteur trouvera un exemple d'application de ce type de

22. http ://hyperopt.github.io/hyperopt
23. https ://sigopt.com/. Il s'agit d'une plateforme avec un nombre limitée de connexions
24. https ://docs.ray.io/en/master/tune/index.html

méthodes dans le chapitre 3, partie 3.3.4. L'idée est de visualiser les entrées qui conduisent le plus à faire basculer la prédiction d'une catégorie à l'autre (via l'ampleur des gradients des dernières couches du réseau associés).

Plus simplement, dans l'exemple présent, les images de la Figure 14 page 41 ont tendance à montrer des lits clairs sur fond sombre. Les chaises ont plutôt tendance à être entourées de blanc. Si l'on inverse les couleurs des images (inversion du contraste), la performance du réseau à prédire la catégorie tombe à 41 %. Cet dégradation des performances est évidemment lié à la faible taille du jeu de donnése et à la simplicité des modèles utilisés. Cela illustre cependant l'enjeu de la généralisation des résultats lorsque la distribution des données évolue, qui reste une affaire délicate y compris pour des modèles beaucoup plus avancés.

Comparaison à la régression pénalisée Il est possible de comparer les résultats obtenus avec ceux d'une régression pénalisée (*elasticnet*). Il s'agit de trouver une solution au problème :

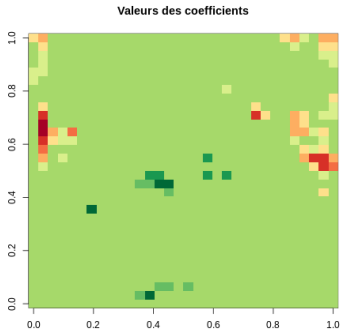
$$min_{\beta_0,\beta_1,\ldots,\beta_p}(1/n(\sum_{i=1}^{n}(y_i(\beta_0+\sum_{k=1}^{p}\beta_k x_{k,i})-\log(1+\exp(\beta_0+\sum_{k=1}^{p}\beta_k x_{k,i}))))+ \\ \lambda(\alpha\sum_{k=1}^{p}|\beta_k|+(1-\alpha)\sum_{k=1}^{p}\beta_k^2))$$

où $x_{k,i}$ est la valeur du pixel k de l'image i. y_i est le type de l'image (1 pour une chaise et 0 pour un lit). La première partie de l'expression correspond au programme de minimisation d'une régression logistique. L'hyperparamètre α est le poids relatif de la somme des valeurs absolues des coefficients par rapport à la somme des carrés des coefficients. L'hyperparamètre λ est l'intensité de la contrainte globale sur la valeur des coefficients. Cette méthode permet de classer les images de l'échantillon test avec une précision de 81 %, très proche donc de la précision obtenue avec *neuralnet*, moindre que celle obtenue avec *Keras*.

Si l'on représente la valeur des coefficients β_k sur une image (voir figure 16), on s'aperçoit que la régression semble détecter les chaises grâce aux pixels de côté (points rouges) et les lits grâce aux pixels du centre (points vert foncé).

Les zones détectées par la régression, comme celles détectées par les réseaux perceptrons multicouches précédents sont fixes sur les images. Pour prendre en compte les relations entre les pixels, on peut recourir à d'autres types de réseaux. Les réseaux de convolution sont spécifiquement adaptés aux images. Ils permettent de retrouver des motifs où qu'ils soient dans l'image [25]. Un cas pratique mobilisant un réseau de ce type figure dans le troisième chapitre.

25. Un tel réseau permet d'atteindre un taux d'images bien classées de 86 % (un exemple pratique figure sur le repo github)



GRAPHIQUE **16 – Valeurs des coefficients de la régression pondérée (ElasticNet)**

Notes : Un point rouge représente un coefficient positif : un pixel clair ($x_{k,i} proche de 1$) à cet endroit de l'image sera associé à un logit plus proche de 1 (chaise). Un point vert foncé représente un coefficient négatif : un pixel clair à cet endroit sera associé à une prédiction plus proche de 0 (lit).

2 Prédire pour imputer à l'aide d'un réseau de neurones

2.1 Pourquoi des réseaux de neurones pour l'imputation de valeurs manquantes ?

L'imputation de valeurs manquantes est une activité classique du statisticien. Il s'agit de remplacer une valeur manquante par une ou des valeurs de remplacement. L'imputation vise à réduire le biais de non-réponse. Elle permet aussi de fournir un fichier complet, généralement directement utilisable par les utilisateurs. Cependant lorsque les données imputées sont utilisées pour l'inférence, le modèle d'imputation peut échouer à réduire voire au contraire amplifier l'éventuel biais de non-réponse. Il peut conduire à sous-estimer l'incertitude des estimations obtenues voire introduire des corrélations fausses entre variables. La recherche d'un bon modèle d'imputation est donc importante. Cette tâche d'imputation se prête bien à la mise en œuvre d'algorithmes de *machine learning* puisqu'elle est purement prédictive dans son esprit. Le but de l'imputation n'est pas de fournir un modèle facilement explicable et interprétable, et le problème de la boite noire est relégué au second plan. En revanche, la précision de la prédiction est un atout important pour l'imputation. L'avantage *a priori* des réseaux de neurones pour cette tâche est de fournir des modèles très flexibles qui conduiraient à une moindre sous-estimation de la variance de l'échantillon. Nous expérimentons sur des données de salaire l'imputation par RN, en comparant ses performances de prédiction avec la méthode d'imputation actuellement en production et une équation de Mincer.

2.1.1 Prédiction et imputation de la non-réponse partielle

On se situe dans le cadre de l'imputation simple de la non-réponse partielle. Autrement dit, on s'intéresse à des valeurs manquantes pour une seule variable, et on cherche à remplacer chaque valeur manquante par une unique valeur imputée. Outre l'intérêt de proposer un fichier complet et facile d'utilisation, le but principal de l'imputation est de corriger le biais de non-réponse, découlant du fait que les valeurs manquantes ne sont pas distribuées au hasard. Pour espérer corriger le biais de non-réponse, on fait l'hypothèse que la non-réponse est ignorable (MAR ou *missing at random*, Rubin (1976)), c'est-à-dire que la probabilité de répondre d'un individu est indépendante de la valeur de la variable y qu'on impute, conditionnellement aux variables auxiliaires x utilisées pour imputer :

$$\mathbb{P}(r_i = 1|x_i, y_i) = \mathbb{P}(r_i = 1|x_i)$$

Cette hypothèse est agnostique sur le type de modèle utilisé pour prédire la non-réponse ou la valeur manquante. Elle ne suffit pas à assurer la validité de l'imputation si le modèle d'imputation est mal spécifié, mais un modèle bien spécifié peut en revanche corriger le biais de non-réponse sous cette hypothèse.

La question de l'imputation est cependant plus large que celle de la seule recherche d'un bon modèle prédictif, Chen et Haziza (2019) en présentent un panorama récent à partir du lien entre la méthode d'imputation et la quantité d'intérêt formalisée de manière très générale. Si les quantités d'intérêt sont des statistiques de distribution (par exemple, un rapport inter-quantile), il peut être préférable de recourir à une imputation aléatoire, au prix d'une variance additionnelle, plutôt qu'à une imputation déterministe, qui concentre les valeurs imputées autour de l'espérance (conditionnelle). De même, les méthodes avec donneurs (plus proche voisin, *hot deck*) ont l'avantage de fournir comme valeurs imputées des valeurs effectivement prises dans la réalité (l'échantillon). C'est la raison pour laquelle elles sont particulièrement recommandées pour des variables catégorielles. Elles peuvent aussi avoir du sens pour des variables a priori continues mais souvent arrondies par les répondants comme les salaires. Enfin, l'imputation est source de biais si les données sont ensuite analysées avec des modèles différents et surtout plus riches que les modèles d'imputation (voir Chen et Haziza (2019)).

Dans tous les cas cependant, un bon prédicteur reste utile. L'imputation aléatoire revient généralement à ajouter du bruit à un modèle prédictif déterministe, et bénéficie de la qualité de ce prédicteur. L'imputation par donneur s'appuie souvent sur la définition d'une distance entre observations, et un bon prédicteur peut aussi fournir une distance efficace.

Les réseaux de neurones sont de bons candidats pour leurs performances prédictives. Leur complexité est un atout : lorsque les valeurs imputées sont ensuite utilisées, par exemple pour inférer des grandeurs ou estimer des modèles économétriques, il est souhaitable que le modèle d'imputation soit plus riche que le modèle estimé, pour limiter le risque d'y transférer des relations artificielles entre variables. Certes, si un modèle simple comme une régression linéaire sur un petit nombre de variable accroît le risque d'introduire, via l'imputation, des corrélations artificielles et non maîtrisées, il permet aussi de les connaître en regardant directement les paramètres estimés du modèle d'imputation, ce qui est beaucoup plus difficile dans la « boîte noire » du RN. Mais en pratique, les usagers des fichiers avec valeurs imputées ont difficilement accès aux détails des modèles d'imputations. Par ailleurs, il est possible que les RN intègrent en partie certains des objectifs poursuivis par les différentes méthodes d'imputation. Ils ont naturellement un comportement proche des méthodes d'ensemble (en particulier avec l'utilisation du dropout), ce qui peut les rapprocher par exemple des méthodes « multiplement robustes » (Chen et Haziza, 2017) qui accumulent les modèles d'imputation. Dans la suite cependant, nous nous limitons à décrire et comparer les résultats des RN comme prédicteurs.

2.1.2 Prédiction de la non-réponse totale

Un autre usage possible des RN est la prédiction du comportement de non-réponse totale dans le but de redresser le biais de non-réponse totale, par exemple par repondération. Il est ainsi possible d'entraîner un modèle à prédire le comportement de non-réponse totale (le refus de participer à l'enquête par exemple) lorsqu'on dispose malgré tout de variables auxiliaires (généralement issues de la base de sondage). Dans cette situation en effet, à la

différence de la non-réponse partielle, le recours à l'imputation n'est pas possible, puisqu'il faudrait imputer toutes les variables de l'enquête, avec autant de modèles, et à partir des seules variables de la base de sondage. On procède donc par repondération.

On fait à nouveau l'hypothèse que la non-réponse (totale) est ignorable, c'est-à-dire que la probabilité de réponse est indépendante des variables non observées. Sous cette hypothèse, la prédiction d'une probabilité de réponse permet de corriger le biais de non-réponse, en l'utilisant dans un estimateur par double expansion (Haziza, Beaumont, et al. (2017)). En pratique, on pondère les observations (des répondants !) par l'inverse de leur probabilité de réponse estimée, comme s'il s'agissait d'un poids de sondage. On utilise souvent pour cela un modèle logit avec des groupes homogènes de réponse. Un RN peut lui être substitué.

L'entraînement ou l'estimation d'un modèle prédicteur du comportement de non-réponse n'est pas seulement utile dans le cadre de la non-réponse totale. Il peut être un complément de l'imputation simple dans une méthode doublement robuste (Bang et Robins, 2005). L'estimation doublement robuste d'une grandeur d'intérêt fonction d'une variable en partie imputée mobilise à la fois les valeurs imputées (donc un modèle d'imputation) et un modèle de non-réponse (par exemple via la pondération), et est doublement robuste car il reste convergent même lorsqu'un (et un seul) des deux modèles est mal spécifié.

2.2 Imputation des valeurs manquantes de salaire dans l'enquête Emploi

La question sur le salaire dans l'enquête Emploi est celle qui entraîne le plus grand taux de non-réponse. Elle est posée depuis 1982 (d'abord en tranche, puis en valeur à partir de 1989) aux salariés interrogés pour la première fois puis lors de leur sixième et dernière interrogation. Le salaire mensuel déclaré dans l'enquête Emploi correspond au salaire net tel que déclaré par les répondants, ou corrigé (en multipliant par 0,8) si les répondants disent donner le montant brut, additionné des bonus tels qu'ils sont déclarés par les répondants dans les questions suivantes de l'enquête. La variable résultante est nommée SALRED.

Déclarative, la variable de salaire dans l'enquête Emploi n'est pas la source des statistiques salariales publiées par l'Insee [26] mais elle est très souvent exploitée dans les études mobilisant l'EEC. Environ 19% des enquêtés interrogés refusent de donner leur salaire en valeur (variable vide, refus ou « ne sait pas »). On leur demande alors d'indiquer leur salaire par tranche : 15% des répondants le font et 4% refusent à nouveau. Les réponses par tranche permettent notamment de constater que la non-réponse est plus concentrée parmi les plus hauts salaires. ce qui suggère que la non-réponse sur le salaire n'est peut-être pas ignorable. Nous utilisons donc également les réponses en tranches pour évaluer le risque que ce problème de sélection peut faire peser sur les différents prédicteurs et l'imputation.

26. Elles sont principalement issues de données administratives.

Nous expérimentons l'usage d'un RN pour imputer les valeurs manquantes de SALRED. Le développement de l'algorithme est réalisé avec Keras en Python. Certaines parties du code sont présentées ci-dessous dans un but illustratif et pédagogique. Le code complet est disponible [27]. Nous comparons les performances d'un réseau à celles de modèles économétriques classiques : notamment un modèle s'appuyant sur une traditionnelle équation de Mincer (Mincer, 1974). Celle-ci relie le log du salaire horaire aux proxies des composantes du capital humain que sont l'expérience et le niveau d'éducation :

$$log(y) = log(y_0) + rs + \beta_1 x + \beta_2 x^2$$

Avec y le salaire horaire, s le nombre d'années d'études et x l'expérience potentielle (la différence entre l'âge au moment de l'observation et l'âge à la fin des études). L'équation prédit plutôt correctement le salaire dans de nombreux fichiers de données, même sous sa forme la plus simple (Lemieux, 2006). Elle sert naturellement de référence (Boelaert et Ollion, 2018).

Nous comparons aussi les performances de ce RN à la méthode d'imputation de valeurs manquantes actuellement en production et telle que décrite dans Babet (2020). Cette méthode d'imputation s'appuie sur des régressions linéaires mais s'éloigne du modèle de Mincer dans la mesure où c'est le (logarithme du) salaire mensuel et non horaire qui est mobilisé et les variables explicatives ne sont pas les mêmes. La méthode actuelle en 2020 a recours à cinq équations de salaire mensuel en log, réestimées chaque trimestre pour cinq catégories de salariés (selon le sexe croisé avec les groupes ouvriers/employés et professions intermédiaires/cadres, plus un groupe pour les contrats atypiques, précaires, et les mineurs). Ces équations mobilisent comme variables explicatives : le nombre d'heures habituellement travaillées par semaine (en niveau), la catégorie socio-professionnelle, le secteur d'activité, le nombre d'années d'expérience dans l'emploi occupé (ancienneté), le type d'employeur, le nombre de salariés dans l'entreprise, l'âge, le diplôme, ainsi que des indicatrices d'emploi multiple ou occasionnel, de bonus, de secteur public et de résidence en agglomération parisienne, et enfin le sexe pour le groupe des contrats atypiques. Les valeurs avec des résidus extrêmes sont mises à blanc avant une nouvelle estimation, et les valeurs imputées sont ensuite tirées dans une distribution déduite des équations estimées, de la variance des résidus, et contrainte par la réponse en tranche le cas échéant. Il s'agit donc d'une imputation aléatoire avec une modélisation paramétrique du salaire mais aussi de l'aléa.

27. Sur `https://github.com/InseeFrLab/DT-RN-chapitre2`. Le code est adapté à l'environnement Python préconstruit pour la datascience disponible sur les serveurs *AUS* accessibles aux personnels de l'Insee. Les données mobilisées, issues des fichiers de production de l'enquête, confidentiels, ne permettent pas de rendre les résultats du chapitre facilement reproductibles

2.3 Préparation des données

2.3.1 Sélection et retraitement des variables

Pour entraîner un RN sur ce même problème d'imputation, nous utilisons 1 220 000 observations de salaires, issues de l'EEC entre 1993 et le premier trimestre 2018. La variable de salaire à prédire est construite de la même manière que SALRED, hors imputation évidemment. Il s'agit donc d'un salaire mensuel dont on prendra le log.

Les variables explicatives, les *features*, prises en considération sont plus nombreuses que dans le modèle de SALRED. Pour mieux exploiter les possibilités des RN, nous mobilisons le maximum de variables possible, lorsqu'elles sont disponibles. L'absence de sélection de variable ne pose pas de problème d'estimation ni d'interprétation puisque l'objectif est purement prédictif. En revanche, certaines variables peuvent soulever des questions déontologiques, y compris dans un contexte d'imputation. Faut-il, par exemple, mobiliser la variable de pays de naissance, ou même de sexe, pour imputer le salaire ? On peut choisir de les exclure, notamment pour ne pas courir le risque que des corrélations fallacieuses les impliquant soient ancrées dans les valeurs imputées par un modèle imparfait. Même dans l'hypothèse où le modèle identifie correctement l'impact de ces variables, on peut vouloir les mettre de côté en fonction de l'usage qui sera fait ensuite des valeurs imputées, par exemple si les salaires imputés guident des décisions. D'un autre côté, quand les valeurs imputées alimentent des travaux purement descriptifs, on peut souhaiter qu'elles reflètent au mieux les régularités capturées dans les données d'enquête, et souhaiter intégrer ces variables. Ces questions ne sont pas spécifiques au *machine learning* et se posent aussi pour des modèles d'imputation plus classiques, avec quelques différences cependant. Les modèles classiques permettent au moins en théorie de connaître facilement le poids accordé par un prédicteur à telle ou telle variable. Cependant, reposant sur moins de variables, ils pourraient être davantage sujets à générer des corrélations fallacieuses. On ne traite pas davantage de ces questions ici, mais certains enjeux éthiques du *machine learning* sont évoqués en conclusion (4.3.4).

Ici l'objectif étant uniquement de montrer les performances prédictives, nous incluons les variables de sexe, année et trimestre d'enquête, date et lieu de naissance, lieu de naissance des parents, statut d'immigration, niveau d'éducation et année du diplôme le plus élevé, expérience dans l'emploi actuel, profession et professions des parents, secteur d'activité, heures travaillées par semaine (habituellement, et spécifiquement pendant la semaine de référence), temps partiel, type de contrat, type d'employeur, existence de bonus. Les variables catégorielles sont remplacées par des indicatrices des modalités (*one-hot encoding*) où nous considérons les valeurs manquantes comme une modalité supplémentaire de la variable. Pour les variables numériques (comme l'âge ou l'année de diplôme) nous imputons les valeurs manquantes par la moyenne et ajoutons une indicatrice de valeur manquante. Ces méthodes d'imputation brutales sont efficaces en input d'algorithmes de *machine learning*, car l'algorithme peut exploiter l'indicatrice de valeur manquante si elle est informative (par exemple si elle signale un comportement de réticence à répondre

corrélé au niveau de salaire), et n'est pas gêné par l'imputation à la moyenne des variables numériques sinon, voir Josse, Prost, Scornet, et Varoquaux (2019)).

Contrairement au modèle d'imputation de SALRED actuellement en production et réestimé chaque trimestre uniquement sur les données du trimestre, nous entraînons le RN sur les données d'une longue période d'enquête (1993-2018), au cours de laquelle le module « salaire » du questionnaire de l'enquête Emploi a évolué, et nous sommes passés du franc à l'euro. Nous avons mobilisé des variables relativement stables ou reconstruites dans le cadre d'une autre étude (Picart et Babet, 2020), travail coûteux mais fréquent lorsqu'on veut mobiliser les enquêtes dans le temps long. Une seule correction supplémentaire a été faite : jusqu'à ces dernières années, il était possible pour les enquêtrices et enquêteurs de l'enquête Emploi de coder les "Refus" et "Ne Sait Pas" à la question du salaire avec les raccourcis "9999999" et "9999998". De simples fautes de frappe ont parfois introduit par erreur des salaires très élevés. Nous enlevons tous les salaires du type 999999, 99999, 999998 et 99998 euros (ou francs). Le RN, tel que nous l'avons entraîné, n'était pas capable de repérer et corriger de lui-même ces erreurs qui ont cependant un impact très fort sur les estimations.

2.3.2 Sélection des échantillons

L'échantillon test, d'environ 10%, est composé de deux parties : l'ensemble du premier trimestre 2018, et l'ensemble des individus nés en mars. L'échantillon n'est donc pas tiré aléatoirement, mais en sacrifiant légèrement la représentativité, on s'assure d'une part un échantillonnage très facile à reproduire, et d'autre part un test robuste au problème de dérive de la distribution dans le temps. En testant sur le premier trimestre 2018, on vérifie en effet qu'un prédicteur entraîné sur longue période resterait pertinent sur le temps présent s'il devait être mis en production. On peut ainsi le mettre en situation dans des conditions proches de celles dans lesquelles les performances du prédicteur de SALRED sont mesurées [28]. Enfin, lors de chaque entraînement, les lignes restantes sont aléatoirement divisées en 995000 lignes d'entraînement (90%) et 110000 lignes de validation (10%). Cet échantillon de validation permet de guider l'optimisation des hyperparamètres du RN.

2.3.3 Réduire la dimension d'une nomenclature par plongement lexical : l'exemple des professions

Certaines variables catégorielles de l'enquête emploi ont de très nombreuses modalités, en particulier celles codées selon les nomenclatures françaises et européennes des

28. Cependant, les répondants à l'enquête emploi sont censés déclarer leur salaire deux fois en première et dernière interrogations. Au premier trimestre 2018, environ la moitié des salaires viennent de personnes qui ont déjà pu déclarer un salaire dans la base d'entraînement. Lorsqu'on contrôle cette éventuelle contamination en testant uniquement sur les répondants en première interrogation au premier trimestre 2018, les performances prédictives des RN se maintiennent (elles sont un peu plus élevées). Il semble donc que les RN ne surapprennent pas en identifiant les individus.

professions, des secteurs d'activité ou des diplômes. L'encodage *one-hot* (qui crée des variables indicatrices pour chaque modalité) de ces variables de grande cardinalité génère autant de colonnes que de modalités (plusieurs centaines) et rend le fichier d'entrées peu maniable pour des questions de mémoire et de temps de calcul. Plus fondamentalement, il est difficile pour le RN de s'entraîner sur ces nombreuses variables indicatrices dont beaucoup signalent des modalités rares. Afin de réduire la dimension de cet ensemble de *features*, on va chercher à représenter chaque modalité fine par un vecteur de nombres réels de faible dimension. On va donc projeter dans un espace vectoriel de plus faible dimension ces variables en tenant compte de la proximité sémantique entre les valeurs de ces variables. Cette approche, classique en analyse textuelle, s'appelle *plongement lexical*, *vectorisation des mots*, ou encore *word embeddings* en anglais. La proximité sémantique entre deux modalités, deux « mots », peut s'obtenir de différentes manières, selon les éléments de contexte dont on dispose. Pour la suite de l'exercice on fait notamment l'hypothèse que sont proches les modalités de profession exercées ou d'autre variables, pour les individus d'un même logement. D'autres critères de proximité sont possible : les réponses par un même individu aux différentes interrogations, les modalités des ascendants et descendants lorsqu'il existe des questions à leur sujet, d'une même grappe de logements, etc.. Cette hypothèse est forte et est retenue ici pour les besoins de l'exercice. Plus généralement, une telle étape d'*embeddings* dépend de l'objectif poursuivi et d'éventuelles considérations théoriques ou éthiques.

Parmi d'autres algorithmes similaires, *word2vec* a beaucoup fait progresser l'analyse du langage en permettant d'encoder les mots d'un vocabulaire à partir d'un corpus de phrases, voir Mikolov, Chen, Corrado, et Dean (2013). C'est cet algorithme que l'on va implémenter ici. On fait l'hypothèse que des mots qui partagent souvent un même contexte, c'est-à-dire qui apparaissent fréquemment dans des phrases similaires, ont un sens proche. Il s'agit d'apprentissage auto-supervisé analogue à l'auto-encodeur (voir chapitre 4.1) : chaque mot est d'abord traduit en un (très grand) vecteur des co-occurences avec tout le reste du vocabulaire, puis un réseau peu profond avec une unique couche cachée de faible dimension est entraîné à prédire, par exemple, un mot donné à partir des mots voisins dans une phrase, ou au contraire à prédire un contexte à partir d'un mot. A l'issue de l'entraînement, la couche cachée fournit un encodage (dont la faible dimension d est fixée à l'avance) de la totalité du vocabulaire. Contrairement aux vecteurs de co-occurences, qui sont de très grande dimension mais essentiellement constitués de zéros (*sparse*), le vecteur encodé est dense, il contient une grande partie de l'information, concentrée dans une petite dimension. L'algorithme est rapide, léger et efficace. Ce principe d'*embedding* est maintenant utilisé pour des problèmes spécialisés comme la vectorisation de concepts médicaux (Beam, Kompa, Schmaltz, Fried, Weber, Palmer, Shi, Cai, et Kohane (2019) ou, sur données françaises, Doutreligne, Leduc, Nguyen, et Vuagnat (2020)). On peut se procurer des modèles pré-entraînés sur de larges corpus (par exemple, les articles de Wikipedia), ou bien entraîner son propre modèle, ou ré-entraîner un modèle préexistant (*transfert learning*) lorsque le problème est très spécifique.

Nous appliquons ici *word2vec* à la vectorisation des nomenclatures statistiques, un

problème spécifique sur lequel il est difficile de mobiliser des modèles génériques préentraînés. Nous entraînons donc un nouveau modèle.

En pratique, chaque modalité (par exemple la modalité « 211d : Artisans plombiers, chauffagistes » de la variable P :profession de l'enquête Emploi) est considérée comme un « mot » dans un vocabulaire. Pour la profession, les phrases sont construites comme des suites de tels « mots » à partir des relations suivantes : cohabitation dans un ménage, transition professionnelle d'un individu au cours des 6 interrogations d'un ménage, professions actuelle et ancienne, professions des parents de l'individu interrogé. Si, par exemple, une personne enquêtée est barman lors de sa première interrogation, puis militaire professionnel lors de sa seconde interrogation, la phrase suivante (constituée de deux mots séparés par un point-virgule) :

`[561a : Serveurs, commis de restaurant, garçons (bar, brasserie, café ou restaurant) ; 532c: Hommes du rang (sauf pompiers militaires)]`

apparaîtra dans le corpus. De plus, s'il a déclaré dans l'enquête que son père était gendarme et que sa mère tenait un petit café, la phrase :

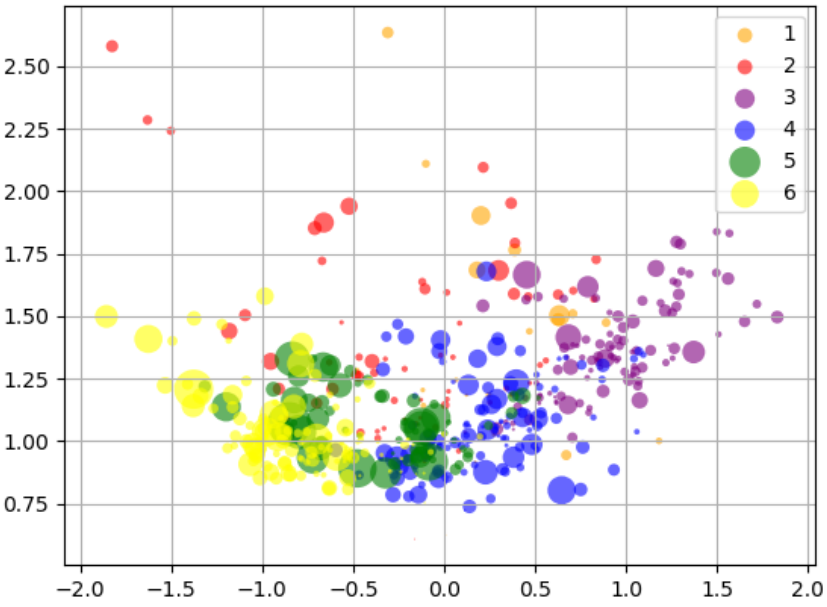
`[532a: Gendarmes (de grade inférieur à adjudant) ; 224a: Exploitants de petit restaurant, café-restaurant, de 0 à 2 salariés]`

sera également dans le corpus. Si on prend en compte les liens entre ascendants et descendants, on ajoutera dans le corpus les quatre phrases reprenant chacune des combinaisons entre les professions du père, de la mère, et les deux professions du fils. On fait donc une hypothèse d'ordre sociologique : les professions qui apparaissent dans un même contexte (même ménage, couples, différents emplois d'un même individu au cours du temps, professions des parents et des enfants) sont des professions proches. Cette proximité peut tenir, au sens large, du déterminisme social, mais aussi de facteurs plus concrets : proximité géographique et économique, capital humain, etc. Dans l'exemple des professions, un encodage de dimension $d = 4$ semble capturer une information substantielle (figure 17).

Il est difficile de juger directement de la qualité de l'encodage réalisé pour une telle vectorisation. Cette démarche relevant de l'apprentissage non supervisé, à la manière d'une classification, ou de la hiérarchie d'une nomenclature, il est difficile de prouver quantitativement son intérêt, faute de point de comparaison [29]. C'est à l'usage qu'on éprouve son efficacité. On notera par la suite que l'utilisation des nomenclatures vectorisées améliore les performances du RN par rapport à l'omission totale de ces variables de grande cardinalité et même par rapport à l'intégration d'un niveau plus agrégé des nomenclatures (par exemple, les professions regroupées en 32 postes dans la variable CSE), ce qui tend à valider *a posteriori* la démarche.

De manière plus descriptive, on peut cependant regarder si les résultats semblent conformes à l'intuition. Pour l'encodage des professions, on visualise assez facilement dans

29. Dans son article initial, l'algorithme *word2vec* a rencontré la même difficulté : il est délicat de trouver ou de construire des métriques de performance, des corpus et des *benchmarks* adaptés pour cette procédure non supervisée.



GRAPHIQUE **17 – Professions projetées sur deux axes d'un encodage**

Notes : Encodage en 4 dimensions projeté sur deux dimensions. Les couleurs correspondent aux groupes sociaux (1. Agriculteurs 2. Artisans, commerçants et chefs d'entreprise 3. Cadres et professions intellectuelles supérieures 4. Professions intermédiaires 5. Employés 6. Ouvriers) et la surface des cercles est proportionnelle aux effectifs.

Le graphique peut se lire de manière analogue à un graphique tiré d'une analyse en composantes principales (ACP). Chaque profession est encodée par l'algorithme en un vecteur de dimension quatre, les relations entre les professions dans cet espace condensant l'information contenue dans les contextes utilisés pour l'entraînement. Pour vérifier que ces relations spatiales ont du sens, on les visualise par des projections en deux dimensions. Dans la projection présentée ici, les professions d'un même groupe social sont effectivement proches les unes des autres. L'axe horizontal oppose les ouvriers aux cadres et semblent capturer une information sur le statut social. L'axe vertical oppose les professions intermédiaires aux agriculteurs et aux artisans, commerçants et chefs d'entreprises et distingue les salariés des indépendants.

différentes projections des dimensions intuitives de l'espace social (par exemple sur le graphique 17), telles que l'opposition salariat / indépendance (axe vertical sur le graphique), le statut social associé à une profession (axe horizontal), ou encore, sur d'autres axes de projection, l'opposition privé / public, les professions plus ou moins féminisées, ou même la proximité, par exemple, de professions sans doute typiques des littoraux (marins, militaires, restaurateurs...). Certaines proximités sembleront évidentes et banales, certaines de ces dimensions de l'espace social sont déjà constitutives de la construction des nomenclatures et sont donc intégrées dans les niveaux les plus agrégés de la hiérarchie. Cependant, la vectorisation permet une description à la fois automatique et plus fine de la position de

chaque profession dans cet espace. Elle contient davantage d'information. La méthode peut facilement être généralisée à d'autres enquêtes ou données administratives et d'autres types de variables (elle a par exemple été testée sur un fichier de prénoms).

Dans le code, cette étape prend la forme de deux fonctions. La première constitue un vocabulaire et un corpus de « phrases » pour une variable donnée `col` regroupée selon une autre variable (ici par défaut `IDENT`, identifiant d'un logement dans l'enquête Emploi). Ainsi l'appel de fonction `vectorize_col(df, col="P")` retournera une liste des professions (variables `P` dans l'enquête Emploi) avec le vecteur de dimension 8 associé à chacune, calculé à partir des données contenues dans le fichier `df` et du contexte induit par la cohabitation dans un même logement (variable `IDENT`). L'estimation elle-même est réalisée grâce à la fonction `Word2Vec` du package « gensim [30] » pour Python.

```
import gensim
import pandas as pd

def vectorize_col(df, col, dim=8, groupident="IDENT"):
    savecol = df[col]
    df[col] = df[col].astype(str)
    sentences = df.groupby([groupident])[col].unique()
    sentences = sentences.transform(lambda x: x.tolist()).tolist()
    model = gensim.models.Word2Vec(sentences, min_count=1, vector_size=
        ↪ dim)
    All = list(np.unique(df[col]))
    Pcoord = ([list(model.wv[i]) for i in All])
    colvec = pd.DataFrame(Pcoord)
    colvec = colvec.set_axis([col + '_vec' + str(x) for x in (range(dim))
        ↪ ], axis=1, inplace=False)
    colvec[col] = All
    df[col] = savecol
    return colvec
```

La seconde fonction applique la première à la liste `vectl` de toutes les variables à vectoriser, et remplace chacune de ces variables par le résultat de la vectorisation. Il n'est sans doute pas nécessaire d'effectuer cette étape pour chaque préparation des données. De la même manière qu'on peut utiliser pour des tâches spécialisées diverses des *embeddings* estimés sur un corpus généraliste comme Wikipedia (*transfert learning*), l'encodage d'une nomenclature à partir de l'enquête Emploi pourrait être sauvegardé et réutilisé à chaque fois qu'on souhaite utiliser un encodage de cette même nomenclature, quel que soit le fichier, à condition cependant que le contexte ou la proximité entre deux mots sur laquelle s'appuie l'encodage soit proche de la proximité recherchée dans le cadre de la nouvelle utilisation. Ici, les nomenclatures vectorisées sont déjà archivées sous forme de fichiers .csv dans un répertoire dédié (`"./data/vec data/"` dans notre cas). La fonction commence

30. https ://radimrehurek.com/gensim/

donc par essayer de charger ces fichiers existants, et ne calcule une nouvelle vectorisation que si aucune archive n'est disponible.

```
def col2vec(df, vectl, groupident="IDENT", dim=8, vec_repertory="./data/
    ↪ vec␣data/"):
    vect_col_list = [i for i in df.columns if i in vectl]
    colnum = len(vect_col_list)
    i = 0
    for col in vect_col_list:
        print(i, "/", colnum, "␣columns␣vectorized")
        i = i + 1
        filename = col + "vec.csv"
        if exists(vec_repertory + filename):
            print(filename, '␣already␣exists,␣matching␣col', col, "␣with␣
                ↪ it")
            colvec = pd.read_csv(vec_repertory + filename)
            df[col] = df[col].astype(str)
            colvec[col] = colvec[col].astype(str)
            df = df.merge(colvec, how='left', on=col)
        else:
            df = df.merge(vectorize_col(df, col, dim=dim, groupident=
                ↪ groupident), how='left', on=col)
        df = df.drop(columns=col)
    return df
```

Suite à cette vectorisation, chaque variable vectorisée est traduite par huit colonnes. A la suite de toutes les étapes de préparation des données (*one-hot encoding* des variables qualitatives, imputation et indicatrices pour les valeurs manquantes, et vectorisation des nomenclatures), l'input final du RN est un vecteur de dimension 215.

2.4 L'imputation par RN : une meilleure performance prédictive au prix d'une mise en oeuvre plus complexe

2.4.1 Un réseau de neurones simple

On peut maintenant chercher à prédire le salaire à l'aide d'un RN. La variable d'intérêt utilisée pour l'entraînement est le *logarithme* du salaire mensuel, l'utilisation du logarithme améliorant sensiblement les performances, comme souvent dans l'économétrie du salaire. Le type de réseau est un perceptron multicouche (*feed-forward neural network*), c'est-à-dire l'architecture standard détaillée au chapitre précédent, et toutes les couches sont denses (tous les neurones d'une couche sont connectés aux neurones de la couche suivante).

Dans l'échantillon d'entraînement, 10% des observations sont extraites à chaque entraînement comme échantillon de validation, et ont servi à sélectionner les hyperparamètres.

Le travail préparatoire a été mené sur un poste individuel, sans parallélisation, avec un temps de calcul de l'ordre de la minute pour chaque époque (ou *epoch*, c'est-à-dire un cycle d'apprentissage sur l'ensemble des données d'entraînement). L'exploration de l'espace des hyperparamètres a donc été faite "à la main" sur l'échantillon de validation tiré aléatoirement lors de chaque entraînement. Avec des ressources plus importantes, cette exploration peut être automatisée et optimisée par validation croisée.

Le RN finalement adopté a sept couches cachées [500, (dropout), 100, 50, 20, 8, 4, 2] et une couche de *dropout* avec un taux de 0.5 (une technique de régularisation consistant à supprimer temporairement et aléatoirement, une certaine proportion de neurones à chaque *batch* de données pendant l'apprentissage, voir section 1.2.3, Les techniques de régularisation). Nous utilisons la fonction d'activation SELU (voir section 1.2.2, Les fonctions d'activation), qui aide à stabiliser l'entraînement (Klambauer, Unterthiner, Mayr, et Hochreiter, 2017), sauf pour la dernière couche, qui est linéaire. La fonction de perte est l'erreur quadratique moyenne, l'optimiseur est Adam (voir section 1.2.3, Les algorithmes de descente de gradient). L'entraînement se fait sur 100 époques, avec une taille de *batch* de 200. Sous Keras, le modèle peut se décrire simplement couche par couche.

```
from keras.layers import Input, Dense, AlphaDropout
from keras.models import Model

inputs = Input(shape=(215,))
x = Dense(500, activation='selu', kernel_initializer='lecun_uniform',
    ↪ bias_initializer='lecun_uniform')(inputs)
x = AlphaDropout(0.5)(x)
x = Dense(100, activation='selu', kernel_initializer='lecun_uniform',
    ↪ bias_initializer='lecun_uniform')(inputs)
x = Dense(50, activation='selu', kernel_initializer='lecun_uniform', bias
    ↪ _initializer='lecun_uniform')(x)
x = Dense(20, activation='selu', kernel_initializer='lecun_uniform', bias
    ↪ _initializer='lecun_uniform')(x)
x = Dense(8, activation='selu', kernel_initializer='lecun_uniform', bias_
    ↪ initializer='lecun_uniform')(x)
x = Dense(4, activation='selu', kernel_initializer='lecun_uniform', bias_
    ↪ initializer='lecun_uniform')(x)
x = Dense(2, activation='selu', kernel_initializer='lecun_uniform', bias_
    ↪ initializer='lecun_uniform')(x)
predictions = Dense(1, activation='linear')(x)

model= Model(inputs=inputs, outputs=predictions)
model.compile(optimizer='adagrad', loss='mean_squared_error', metrics=['
    ↪ mse', 'mae'])

history = model.fit(x_train_scaled, ylog_train, epochs=100, batch_size
    ↪ =100, validation_split=0.1, shuffle=True) # starts training
```

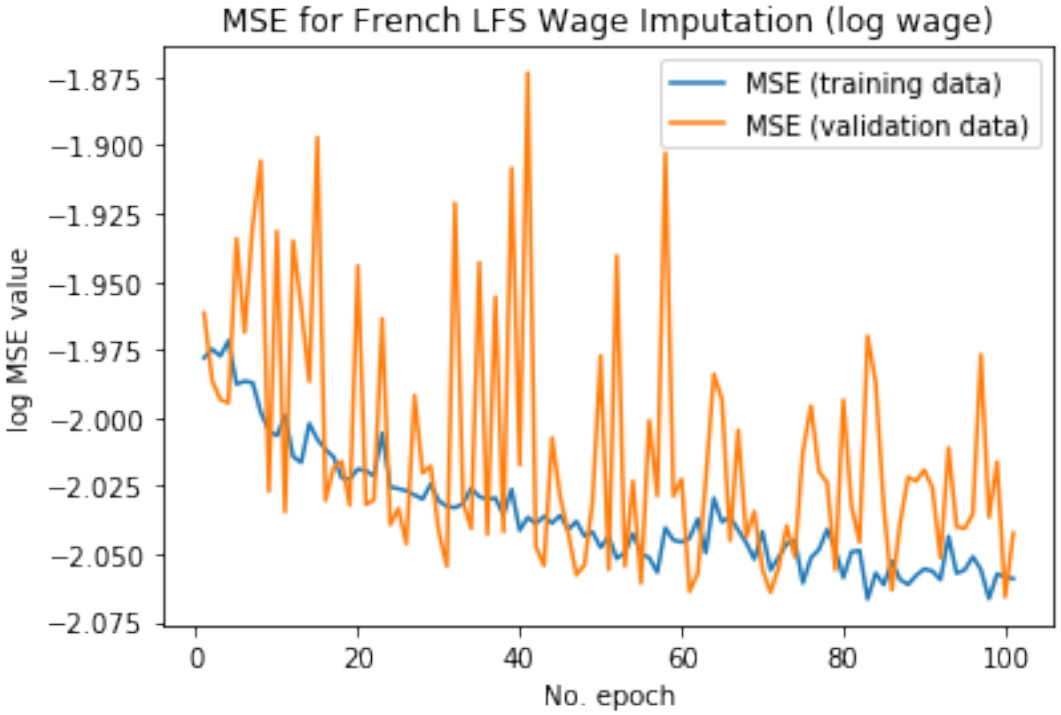
Dans une version plus élaborée, cette construction du RN peut être intégrée au sein d'une fonction python qui prend en entrée un dictionnaire des hyperparamètres. Cette méthode facilite ensuite l'optimisation de ces hyperparamètres :

```
def create_model(input_size=215, hparam_dict=init_param):
    optimizer = hparam_dict['optimizer']
    activation = hparam_dict['activation']
    dropout = hparam_dict['dropout']
    L1_regularization = hparam_dict['L1_regularization']
    L2_regularization = hparam_dict['L2_regularization']
    metrics = hparam_dict['metrics']

    from keras.layers import Input, Dense, AlphaDropout,
       ↪ BatchNormalization
    from keras.models import Model

    inputs = Input(shape=(input_size,))
    from keras import regularizers
    x = BatchNormalization()(inputs)
    for i in range(hparam_dict['number_of_layers']):
        if hparam_dict['layers_size'][i] == 0:
            x = AlphaDropout(dropout)(x)
        else:
            n = hparam_dict['layers_size'][i]
            x = Dense(n, activation=activation, kernel_initializer="lecun_
               ↪ uniform", bias_initializer='lecun_uniform',
                     kernel_regularizer=regularizers.l1_l2(l1=L1_
                        ↪ regularization, l2=L2_regularization))(x)

    predictions = Dense(1, activation='linear')(x)

    model1 = Model(inputs=inputs, outputs=predictions)
    model1.compile(optimizer=optimizer,
                   loss='mse',
                   metrics=metrics)
    return model1
```

2.4.2 Stabilité des performances

L'entraînement des RN est stochastique, et la fonction de perte connaît a priori des minima locaux sur l'espace des paramètres. Autrement dit, il n'est jamais garanti que

l'entraînement d'un RN aboutisse à la meilleure performance possible, ou même à une performance satisfaisante, et la stabilité, la fiabilité de l'entraînement est l'un des enjeux importants des RN. Dans notre configuration, les entraînements ont été particulièrement instables. Après quelques explorations, la correction la plus simple a consisté à sélectionner dès la fin de la première époque d'entraînement uniquement les modèles ayant atteint une performance raisonnable, par exemple supérieure à celle de la prédiction constante par la moyenne. Même ainsi la perte sur l'échantillon d'entraînement et surtout sur l'échantillon de validation reste instable au cours de l'entraînement ou entre deux modèles. La figure 18 illustre cette instabilité pour un modèle, entraîné pour 100 époques après avoir été sélectionné pour sa bonne performance après la première époque d'entraînement. La perte calculée sur l'échantillon d'entraînement est raisonnablement décroissante, mais la perte calculée sur l'échantillon de validation connaît des fluctuations importantes et sans tendance claire pendant les 60 premières époques. L'échantillon de validation restant le même au cours de l'entraînement d'un modèle donné, ces fluctuations sont entièrement dues aux évolutions du RN, par entraînement, d'une époque à l'autre.



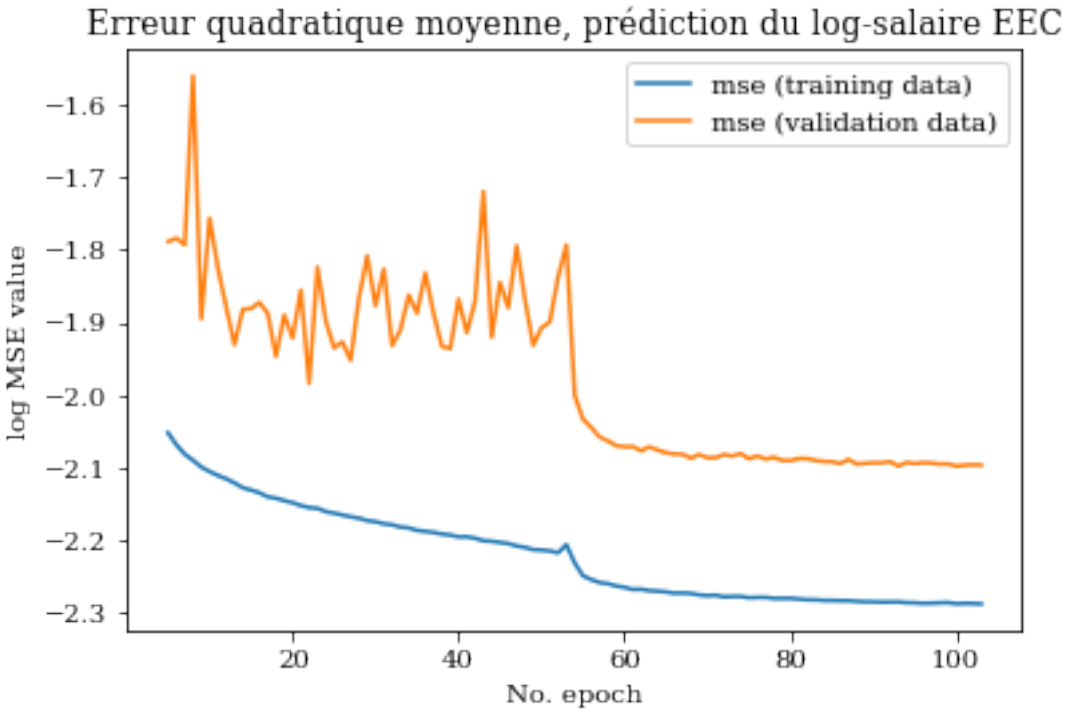
GRAPHIQUE **18 – entraînements de RN**

Notes : log perte sur les échantillons d'entraînement et de validation selon le nombre d'époques d'entraînement. Une différence de 0,1 en log MSE correspond à environ 3 points de R^2 sur le log-salaire. Le R^2 final de ce prédicteur sur l'échantillon test du premier trimestre 2018 est 0,727.

Cette instabilité tient à la nature du problème : la difficulté à prédire aisément le salaire à partir des inputs, et la distribution particulière de cette variable. La distribution des salaires

est non seulement contrainte et concentrée à gauche (salaires positifs et souvent supérieurs au smic) mais très étalée à droite. Les très hauts salaires sont rares mais peuvent être très élevés, et même la transformation logarithmique ne suffit pas à diminuer suffisamment leur grande influence dans la variance totale. Tous les modèles, RN ou non, peinent à prédire les très hauts salaires, et il y a une instabilité de la mesure des performances selon la présence ou non de ces hauts salaires dans les échantillons de validation ou de test, et la performance du modèle sur ce très petit nombre d'observations extrêmes.

Quelques éléments heuristiques sur la stabilité de l'apprentissage se dégagent cependant de l'expérience. Les grands *batchs* sont plus stables, et si les petits *batchs* permettent d'apprendre en moins d'époques, alterner les tailles de *batch* semble intéressant pour bénéficier des deux avantages. La figure 19 illustre ce point. Elle diffère également de la précédente par une réduction du paramètre du *dropout* : le surapprentissage est plus fort, mais l'apprentissage est plus stable.



GRAPHIQUE **19 – entraînements de RN**

Notes : Entraînement avec changement de taille de *batch* : 64 puis 10000 pour les 4 premières époques (non montrées), 64 pour les époques 5 à 54, 10000 pour les époques 55 à 104

Nous avons également cherché à construire des fonction de perte adaptées au problème, par exemple en pénalisant à la fois la distance au log-salaire et la distance au salaire, ou bien de manière à introduire la prédiction par tranche et une pénalisation de l'erreur de tranche. Ces modèles de prédiction mixtes sont encore plus sensible, et il est facile de générer une fonction de perte particulièrement instable.

Une autre piste pour limiter ces problèmes de stabilité consiste à fournir directement au RN l'output d'un modèle linéaire de salaire, ou, encore plus simple, à donner un double input à la dernière couche du réseau : d'une part, la sortie des couches précédentes, et d'autre part à nouveau les données en entrée. On permet ainsi au RN de retrouver rapidement un modèle linéaire efficace (l'avantage d'un RN sans couche cachée) sans sacrifier sa capacité à apprendre des non-linéarités compliquées (l'avantage d'un RN profond). Ce principe de *residual learning* a été fréquemment adopté depuis 2016 (He, Zhang, Ren, et Sun, 2016).

2.4.3 Comparaison des performances

Pour juger de la performance du RN d'imputation, on la compare à la procédure d'imputation en production pour SALRED, puis à la performance prédictive de différentes versions de l'équation de Mincer. La métrique privilégiée est le R^2 ou, plus généralement la réduction de l'erreur quadratique moyenne permise par un prédicteur $\hat{y}$, relativement à la variance [31] :

$$R^2 = 1 - (\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2)/Var(y)$$

On peut tester la précision de la méthode actuelle d'imputation de SALRED sur des observations dont on connaît le salaire, mais que l'on met à blanc (ni salaire, ni tranche) avant de faire tourner le code d'imputation. En pratique, on partitionne aléatoirement en cinq l'échantillon total (entraînement et test), on met à blanc un cinquième des salaires connus, et on fait tourner le code d'imputation de SALRED actuellement en production. On répète l'opération pour chacun des quatre autres cinquièmes [32]. Calculé sur le premier trimestre 2018, le R^2 du log-salaire observé sur le log-salaire prédit n'est que de $0,43$ (table 4). Calculé sur les salaires (et non plus les log-salaires) il descend à $0,19$. La performance prédictive est bien entendu diminuée par la nature aléatoire de la procédure mais elle reste largement améliorable.

Nous utilisons également l'équation de Mincer comme référence de prédicteur déterministe, sans alea donc, contrairement à SALRED. Le modèle est estimé sur le même échantillon d'entraînement que le RN [33]. Mesuré sur l'échantillon test du premier trimestre 2018, le R^2 du salaire observé atteint $0,64$ pour une équation de Mincer incluant les variables de SALRED ainsi que le logarithme des heures travaillées, le carré de l'âge et l'expérience potentielle. En revanche, l'ajout de toutes les variables utilisées par le RN (y

31. on réserve parfois la notion de R^2 au contexte de la régression. Nous l'utilisons ici plus généralement sur l'échantillon test, en l'absence de toute régression. Dans cet usage, le R^2 peut être négatif si le prédicteur est moins bon que la prédiction par la moyenne.

32. Le code de SALRED fonctionne par trimestre de données, il n'est donc pas possible de l'entraîner sur l'échantillon d'entraînement 1993-2017, puis d'imputer la totalité du premier trimestre 2018

33. L'estimation sur un échantillon restreint à une période plus récente n'améliore pas les performances.

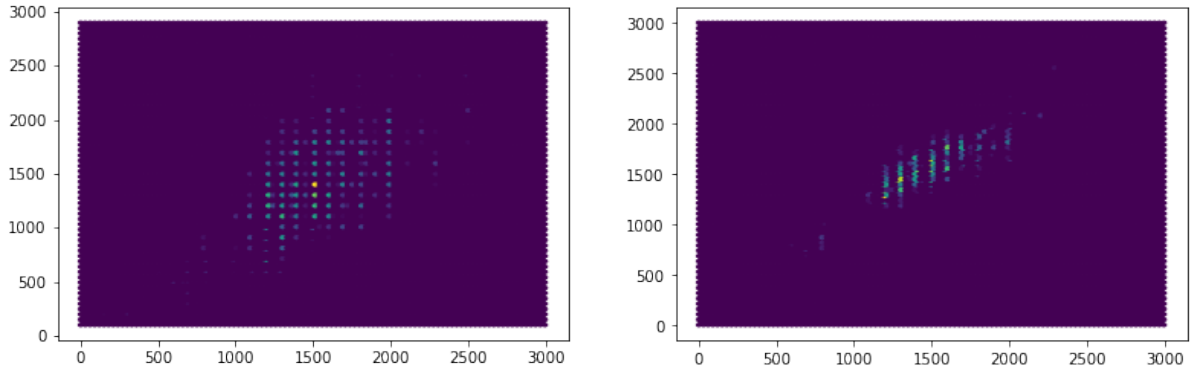
compris les nomenclatures vectorisées) n'améliore pas notablement les performances de l'équation de type Mincer : le R^2 atteint 0,65.

Le RN de référence atteint un R^2 de 0,68 sur cet échantillon, 0.72 sur l'échantillon des personnes nées en mars. L'écart entre Mincer et le réseau de neurone peut suggérer que le RN capte des non-linéarités ou interactions que le modèle supposé dans l'équation de Mincer n'est pas capable de prendre en compte. Calculé sur le salaire et non plus sur le salaire réel, ces R^2 sont respectivement de $0,19$ pour la méthode d'imputation actuelle de SALRED, $0,29$ pour l'équation de Mincer, et $0,33$ pour le RN (figure 20).

TABLEAU **4 – R^2 des modèles sur différents échantillons tests**

	Echantillon test			
	log-salaire		salaire	
Modèle :	2018	Nés en mars	2018	Nés en mars
RN	68,3	71,9	33,0	27,6
OLS sur l'input RN	64,8	62,8	29,1	23,4
Mincer enrichi	63,4	62,9	28,7	27,6
Mincer enrichi + secteurs détaillés	63,7	63,3	29,1	27,9
Mincer basique	51,9	45,0	21,5	20,5
modèle de SALRED	43,2		18,6	

Notes : « Mincer enrichi » intègre le log des heures et l'âge au carré, ainsi que les variables de l'input RN sauf les nomenclatures vectorisées. « Mincer basique » contient uniquement les variables d'expérience potentielle, d'âge et d'âge au carré, d'année d'enquête, des indicatrices de niveau de diplôme, et le logarithme du nombre d'heures travaillées.



GRAPHIQUE **20 – Salaire déclaré et prédit par imputation par SALRED et RN**

Notes : EEC 2018, premier trimestre. Graphique par « binning » avec censure des cases avec 10 observations ou moins (pour respecter le secret statistique). Les salaires prédits par imputation sont en ordonnée, les salaires déclarés par les enquêtés sont en abscisse. Pour l'imputation par SALRED, à gauche, le nuage de point est plus diffus que pour l'imputation par RN, à droite. On observe des lignes verticales correspondant au comportement d'arrondis des personnes enquêtées, et, pour SALRED, des lignes horizontales peut-être liées à la méthode de prise en compte des réponses par tranche.

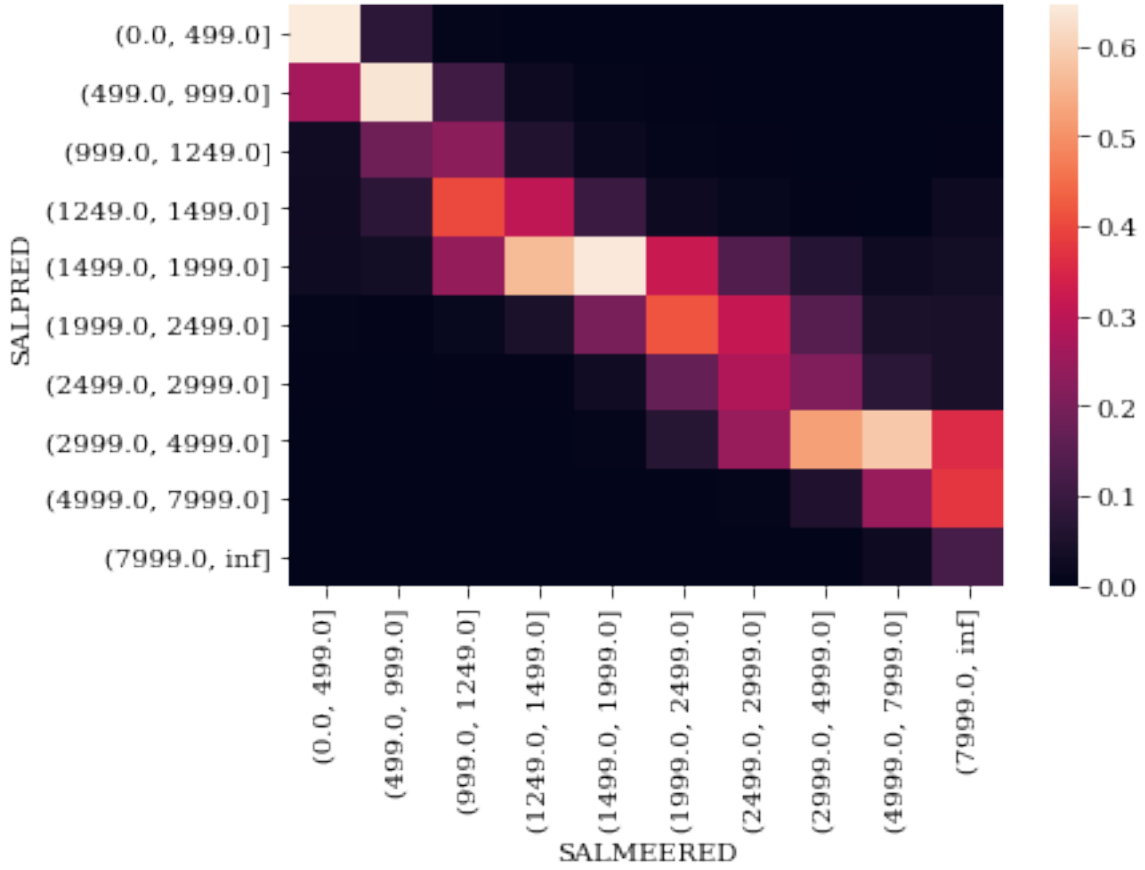
Finalement sur cet exercice, l'écart de performance prédictive entre le RN et un modèle quasi-linéaire inspiré par la théorie économique est limité mais notable. Le RN réduit de 10 à 25% l'erreur quadratique.

2.4.4 Une réduction du biais de non-réponse

Les questions sur le salaire dans l'enquête Emploi fournissent une opportunité rare d'explorer le biais de non-réponse. Celui-ci est par nature inconnu, et on ne peut pas en général être certain qu'un bon prédicteur d'une variable pour les répondants, entraîné sur des valeurs observées, est également un bon prédicteur pour les non-répondants. Mais dans le cas du salaire dans l'enquête Emploi, les non-répondants sont relancés, et on leur propose une réponse par tranche, ce qu'ils acceptent de donner dans trois cas sur quatre. On exploite cette information additionnelle pour étudier le comportement des prédicteurs entraînés sur la population des répondants, face à des cas de véritable non-réponse partielle.

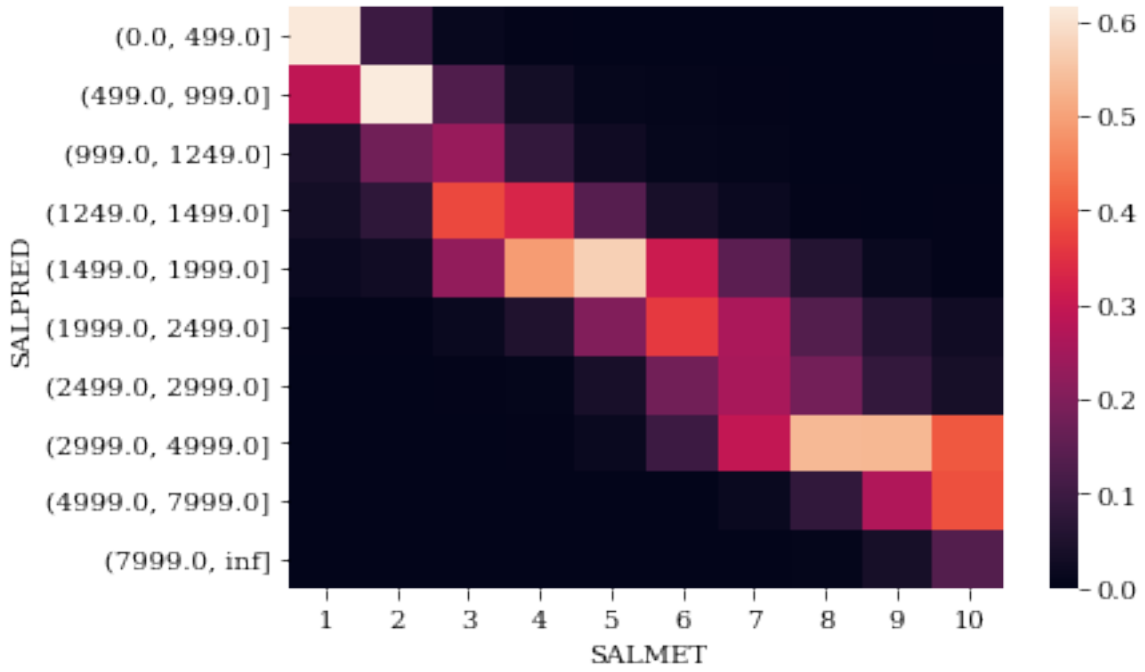
En se limitant à des observations absentes de l'échantillon d'entraînement, on peut classer les salaires observés (SALMEERED) et prédits (SALPRED) selon les tranches de l'enquête. On peut faire de même en cas de non-réponse sur le salaire, lorsque les enquêtés ont tout de même donné leur tranche de salaire (SALMET). Les figures 21 et 22 illustrent les distributions des tranches prédites par le RN, pour les tranches déclarées respectivement par des *répondants*, les personnes qui ont donné leur salaire en clair (sur l'échantillon test du premier trimestre 2018) et par les *non-répondants* pour le salaire en clair (entre 2013 et 2019, période de stabilité de la grille des tranches de salaire). Les deux ensembles de distributions apparaissent très similaires, en particulier pour les tranches élevées, et montrent que le phénomène de sous-estimation des salaires élevés par le prédicteur ne s'accentue pas sur la population des non-répondants.

En revanche, lorsqu'on fait la même analyse avec un prédicteur linéaire (un modèle de Mincer enrichi), la sous-estimation des hauts salaires s'accentue sur la population des non-répondants. Le phénomène apparaît nettement dans la figure 23 qui représente la différence entre les distributions des prédictions par RN et par modèle linéaire. Parmi les 315 non-répondants au salaire en clair qui ont déclaré gagner plus de 8000 euros par mois, les deux modèles se trompent en prédisant fréquemment des salaires trop faibles, par exemple dans la tranche 3000 à 5000, mais la fréquence de cette erreur est plus grande de 30 points pour le modèle de Mincer (74% contre 40%). Le RN prédit la bonne tranche dans 13% des cas, le modèle de Mincer ne prédit jamais la bonne tranche. De même, pour les non-répondants ayant des salaires entre 5000 et 8000, la bonne tranche est prédite dans 26% des cas par le RN, 1% des cas par le modèle de Mincer. Autrement dit, par rapport au modèle de Mincer, la meilleure performance prédictive du RN se traduit par une nette réduction du biais de non-réponse parmi les hauts et très hauts salaires.



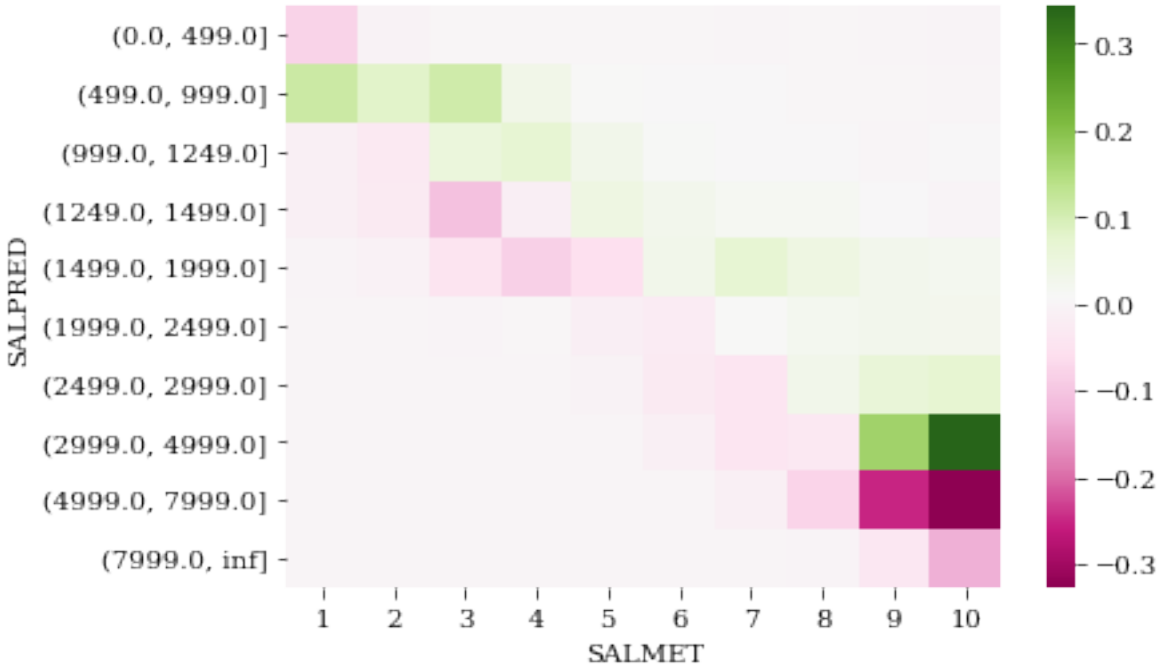
GRAPHIQUE **21 – Répondants en clair : distribution des tranches prédites par RN, selon la tranche déclarée**

Notes : EEC 2018, répondants en clair. La couleur indique la fréquence, pour une réponse en tranche observée (variable SALMEERED), des prédictions en tranches (variable SALPRED) : 67% des personnes interrogées ayant déclaré un salaire inférieur à 500 euros sont prédites dans la tranche 0-500 euros. Les fréquences somment à 1 en colonne. La sous-estimation des tranches imputées pour les hauts salaires apparaît clairement.



GRAPHIQUE **22 – Répondants en tranche : distribution des tranches prédites par RN, selon la tranche déclarée**

Notes : EEC 2013-2019, répondants en tranche (variable SALMET) n'ayant pas répondu en clair, même lecture que le graphique 21



Graphique 23 – Différence OLS - RN des distributions des prédictions par tranche, pour les non-répondants

Notes : EEC 2013-2019, répondants en tranche n'ayant pas répondu en clair. Différence entre les fréquences des prédictions issues du modèle OLS sur input RN et les fréquences des prédictions du RN (valeurs du graphique 22). Chaque colonne somme à 0. Les cases vertes indiquent les tranches plus fréquemment prédites par le modèle OLS que par le RN, les cases roses indiquent les tranches plus fréquemment prédites par le RN que par le modèle OLS. Les cases roses sont plus concentrées autour de la diagonale (tranches correctement prédites).
Lecture : pour les personnes ayant déclaré avoir un revenu dans la tranche 1 (entre 0 et 500 euros), le modèle RN prédit la bonne tranche avec une fréquence supérieure de 7 points à celle du modèle linéaire.

2.4.5 Des prolongements possibles

Ce travail peut connaître plusieurs prolongements pour améliorer la qualité de l'imputation selon les besoins correspondant aux usages qui seraient fait des données imputées. Un premier prolongement consiste à ajouter l'information fournie par les réponses par tranche. Il est facile d'intégrer cette variable parmi les inputs du RN dans l'échantillon d'entraînement, sous forme d'indicatrice de tranche ou en incluant les bornes en valeurs. Sans surprise, cette information accroît (légèrement) la capacité du modèle à prédire le log-salaire sur les échantillons tests. Le modèle n'apprend pourtant pas à prédire systématiquement des salaires dans la bonne tranche, bien qu'il dispose de l'information en input. En sortie, on peut donc également souhaiter que la valeur imputée appartienne à la tranche de salaire, lorsqu'elle est connue, et ce problème est plus difficile. Il est possible d'inciter ou de contraindre un RN à viser la bonne tranche, par exemple avec un modèle à *multiple outputs* entraîné à prédire à la fois le salaire et sa tranche, ou bien avec une fonction de perte construite sur mesure qui prend des valeurs élevées lorsque le salaire imputé sort de la tranche connue. Mais ces contraintes ont en retour un impact sur la qualité de la prédiction du salaire (elles peuvent par exemple encourager le RN à prédire des valeurs éloignées des limites des tranches). Ces prolongements seraient à comparer à des méthodes économétriques existantes qui permettent d'imputer un salaire, conditionnellement à l'appartenance déclarée à une tranche.

Un autre prolongement consiste à prédire non seulement une valeur, mais les paramètres d'une distribution, reflétant davantage à la fois la variance des données d'origine et la plus ou moins grande incertitude associée par le modèle à chacune de ses prédictions. On entre alors dans le domaine des RN probabilistes ou génératifs (depuis les machines de Boltzmann restreintes (Ackley, Hinton, et Sejnowski, 1985; Smolensky, 1986; Salakhutdinov, Mnih, et Hinton, 2007), jusqu'aux auto-encodeur variationnels (Kingma, Welling, et al., 2019), etc.), qui permettraient d'accomplir la même tâche que l'imputation aléatoire (telle qu'elle est mise en oeuvre actuellement dans le modèle d'imputation de SALRED).

D'autres pistes consistent à développer des idées déjà évoquées au début de ce chapitre. On peut utiliser les RN dans une procédure d'imputation doublement robuste, en exploitant également l'information communiquée par le comportement de non-réponse lui-même. En restant sous l'hypothèse de non-réponse ignorable, on combine un modèle de non-réponse et un modèle d'imputation de manière à ce que le résultat soit robuste à une mauvaise spécification d'un (et d'un seul) des deux modèles. On pourrait attendre d'une telle démarche une meilleure réduction du biais de non-réponse. Une autre piste relativement facile à mettre en oeuvre est d'utiliser le salaire prédit par un RN non plus pour imputer directement, mais pour déterminer les donneurs potentiels dans une méthode par plus proches voisins ou par *hot deck*. Ces méthodes par donneurs préservent davantage la variance des données d'origine, ainsi que des caractéristiques difficiles à simuler comme les comportements d'arrondis. Il est même possible de combiner double robustesse et imputation par hot-deck, comme dans Boistard, Chauvet, et Haziza (2016) : au sein de cellules d'imputation déterminées grâce à un modèle d'imputation, le donneur j est tiré au hasard avec une probabilité proportionnelle à $(1 - p_j)/p_j$, avec p_j sa probabilité d'être

répondant, déterminée par un modèle de non-réponse. Il est trivial d'exploiter des RN pour ces deux modèles, imputation et non-réponse.

Enfin, il est possible d'envisager l'imputation au sein d'un système plus général de correction ou de débruitage des données. Le chapitre 4 décrit les auto-encodeurs, des réseaux de neurones qui peuvent être notamment entraînés pour ces tâches. On peut confronter ces auto-encodeurs à un RN « adversaire », un *discriminateur* entraîné à distinguer les données originales des données imputées ou corrigées. Une telle démarche est susceptible en théorie et avec suffisamment de données de produire de manière autonome des imputations très réalistes (reproduisant dans notre exemple la cohérence entre valeur et tranche, les comportements d'arrondis, la variance, etc.) et elle est aujourd'hui très appliquée dans le traitement d'image, mais nous ne l'avons pas encore expérimentée sur des données de statistique publique.

3 Réseaux convolutifs et analyse d'image

3.1 Pourquoi des réseaux de neurones pour l'analyse d'image ?

Cette partie donne un exemple de réseau de neurones appliqué à des images. Il s'agit de construire une variable statistique, le type de parcelle (ouverte, clôturée ou urbaine) à partir d'images de 1km sur 1km des limites des parcelles cadastrales.

Les images sont un format de données très particulier, peu fréquemment utilisé en statistique publique. Elles conduisent souvent à des données très volumineuses. Chaque pixel de l'image constitue potentiellement une ou plusieurs variables d'entrée du modèle, ce qui génère beaucoup de *features*. Dans une image en noir et blanc, un pixel localisé en i, j prendra une valeur réelle $p_{i,j}$ selon sa nuance, par exemple de zéro si il est blanc et à 1 s'il est noir. Dans une image en couleur, le pixel sera représenté par plusieurs réels, correspondant aux différents canaux (rouge, vert et bleu par exemple). La localisation des pixels les uns par rapport aux autres est aussi importante. En effet, l'information de pixels proches est souvent corrélée. Les modèles classiques, comme les perceptrons multicouches peinent à tenir compte de la proximité des données en entrée. Il vaut mieux recourir à des réseaux de neurones adaptés aux images comme les réseaux de convolution ou *Convolutional Neural Networks* (CNN ou ConvNet). Ces réseaux permettent de reconnaître des formes spécifiques où que celles-ci se situent sur l'image. Ils tiennent compte des localisations relatives des pixels les uns par rapport aux autres. Ils sont moins difficiles à entraîner que les perceptrons multicouches dont le nombre de poids explose rapidement avec de telles données en entrée.

3.2 Les spécificités des réseaux d'analyse d'image

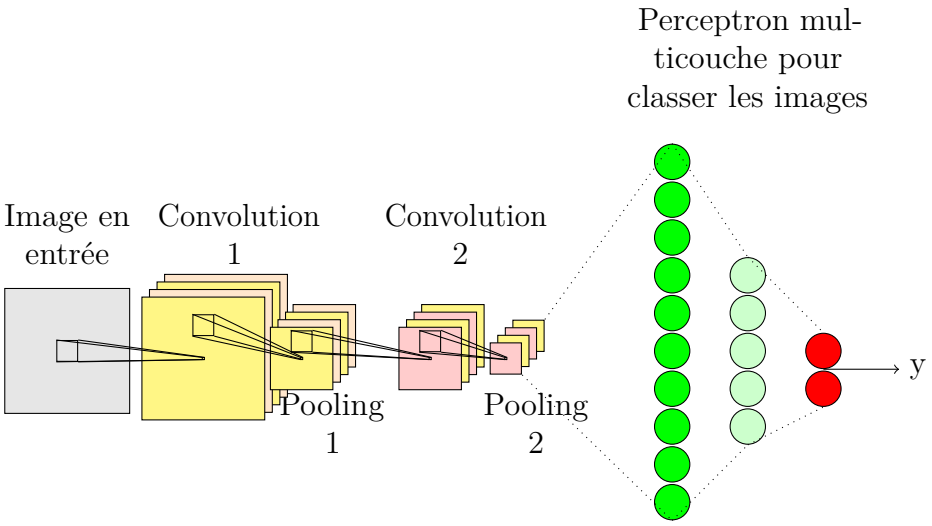
Les *réseaux convolutifs* sont adaptés aux images. Ils sont inspirés du fonctionnement du cortex visuel des animaux. Ils sont globalement organisés comme des perceptrons multicouches mais intègrent des couches spécifiques constituées

- de couches de **convolution**, dont l'objectif est de traiter/distinguer l'information contenue dans une portion limitée de l'image (fenêtre de convolution) ;
- de couches de **pooling** dont l'objectif est de résumer l'information obtenue par la convolution à un niveau plus agrégé, ce qui permet de réduire le nombre de poids à estimer ;
- d'une couche d'activation (**reLu** dont l'objectif est de filtrer certaines valeurs (positives))

A la différence des couches denses du perceptron qui prennent l'ensemble des données en considération, indépendamment de leur éventuelle proximité dans l'image, une **couche de convolution** analyse l'image **localement**, c'est-à-dire en intégrant la proximité des pixels dans l'analyse. Elle est donc capable de reconnaître des structures spécifiques où qu'elles soient dans l'image (par exemple les pattes ou les yeux d'un chat ou dans notre

exemple, les intersections des lignes délimitant les parcelles). La succession des couches de convolution fait prendre en compte au réseau des éléments de plus en plus grands et de plus en plus complexes (par exemple, le chat qui combine les éléments précédents ou les parcelles dans leur globalité avec les vides et les intersections). Les *fenêtres* de convolution (zones sur lesquelles s'opèrent les convolutions successives) se chevauchent partiellement. Des convolutions sont calculées sur des zones de l'image très proches. A la sortie de la couche, cela engendre une redondance de l'information. Les **couches de *pooling*** permettent de réduire cette redondance et d'alléger les traitements. Comme dans les réseaux denses (perceptron multicouche), les couches de convolution peuvent être suivies de couches d'activation (voir CNN et Couche de Convolution, qu'est-ce que c'est ?).

Plusieurs séries de couches de traitement peuvent ainsi se succéder avant de passer à l'étape finale de classification, laquelle peut se réaliser de façon classique par un perceptron multicouche ou une autre méthode d'apprentissage supervisé. Pour faire l'objet de la classification finale, les données à l'issue des couches de convolution (et de pooling) sont *mises à plat* c'est-à-dire que l'on passe d'une image en deux dimensions à un vecteur. La figure 24 montre un exemple d'architecture de réseau convolutif.



GRAPHIQUE **24 – Schéma général d'un réseau convolutif**

Notes : les couches de convolution sont généralement suivies d'un post-traitement (une fonction d'activation de type reLu).

3.2.1 La convolution

L'objectif de la **convolution** est d'extraire au niveau local les caractéristiques les plus pertinentes de chaque zone d'image. Un même filtre de convolution est appliqué pour toutes les zones de l'image (l'ensemble des fenêtres, c'est-à-dire les rectangles de taille $h \times l$ où (h, l) est la taille du filtre), un peu à la manière d'une moyenne mobile. Sur tous les rectangles contenus dans l'image, la convolution renvoie la moyenne des pixels

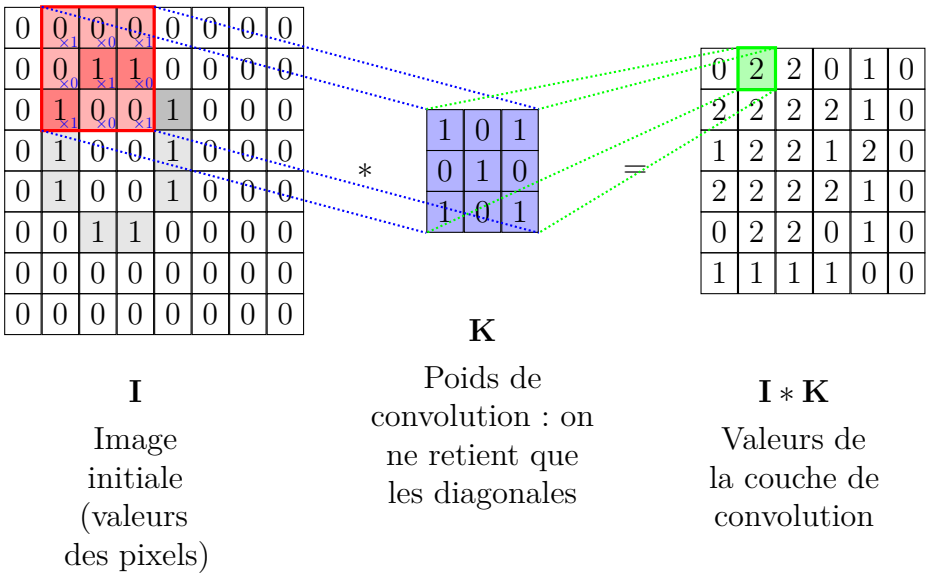
pondérée par les poids spécifiques du filtre de convolution utilisé (voir 25 pour un exemple de calcul). L'objectif est d'extraire les caractéristiques les plus pertinentes sur chaque rectangle. Les poids de ce filtre sont des paramètres qui font l'objet d'une estimation pendant l'entraînement du réseau.

Un exemple :
Pour tout pixel $p_{i,j}$ de l'image, la convolution par le filtre

$$\begin{pmatrix} f_{1,1} & f_{1,2} & f_{1,3} \\ f_{2,1} & f_{2,2} & f_{2,3} \\ f_{3,1} & f_{3,2} & f_{3,3} \end{pmatrix}$$

est calculée comme

$$\sum_{k=1,l=1}^{k=3,l=3} f_{k,l} \times p_{i+k-2,j+l-2}$$



I

Image initiale (valeurs des pixels)

K

Poids de convolution : on ne retient que les diagonales

I ∗ K

Valeurs de la couche de convolution

GRAPHIQUE **25 – Application d'un filtre de convolution à une image**

Notes : Pour une zone de l'image centrée sur un pixel donné, les pixels de la zone sont pondérés selon le filtre de convolution : il s'agit d'une multiplication élément par élément des matrices de la zone et du filtre (produit matriciel de Hadamard) suivi d'une somme des éléments de la matrice en résultant. Pour un filtre de convolution, chaque zone génère ainsi un pixel en sortie de la couche de convolution

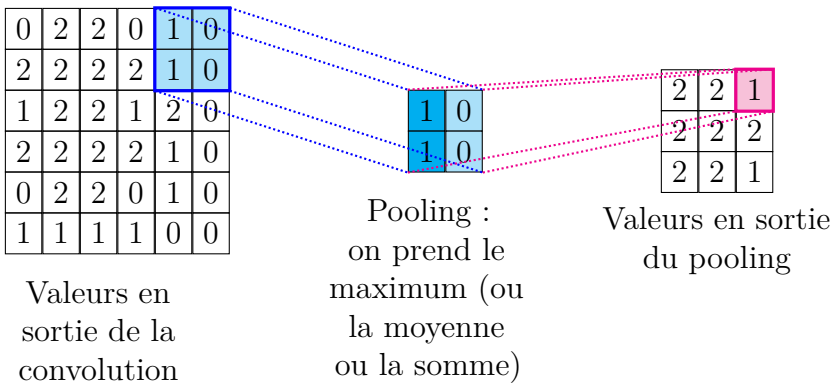
Les couches de convolution comportent en général trois types d'hyper-paramètres :
— Le nombre de filtres ou la profondeur de la couche (plusieurs convolutions avec des poids différents sont appliquées à l'image dans chaque couche).
— La taille du filtre ou la taille du rectangle sur lequel il s'applique ($h \times l$ en dimension 2). Plus le filtre est large et plus les fenêtres glissantes des convolutions se chevauchent et plus il y a de paramètres.

— La marge à zero (ou le *zero padding*) : il est parfois souhaitable de « mettre des zéros » (des pixels de nuance uniforme) en marge de l'image afin de conserver la même taille en sortie (sinon le filtre n'est pas calculable sur les côtés et la taille de l'image à l'issue de la convolution est réduite.

Les éléments des couches de convolution font généralement l'objet d'un traitement non linéaire en sortie (en général une couche de type **reLu** ce qui permet d'introduire une non-linéarité dans le réseau).

3.2.2 Le *pooling*

Le ***pooling***, étape de "mise en commun", est utilisé afin de synthétiser l'information extraite des couches de convolution. En dimension 2 (données spatiales), il consiste à ne retenir qu'une seule valeur (le maximun, la moyenne ou encore la somme) dans chaque fenêtre de dimension $(h \times l)$ de la couche (voir 26 pour un exemple de calcul). Contrairement à la convolution, les fenêtres de *pooling* ne se chevauchent pas. Le maximum donne de bons résultats en pratique.



GRAPHIQUE **26 – Exemple de *pooling* : maximum sur des fenêtres de taille 2 x 2**

Notes : Pour synthétiser une sortie de convolution, l'image est partitionnée en zones (ici de 2 x 2 pixels) et une fonction de pooling est appliquée aux valeurs des pixels de la zone (moyenne, somme, max). Il en résulte un pixel par zone.

3.3 Un exemple d'utilisation

On applique ici des réseaux de neurones convolutifs pour catégoriser des images de 1km sur 1km de parcelles du cadastre en fonction du type de paysage duquel elles proviennent. Est-ce que ces parcelles sont typiques des parcelles rencontrées dans les paysages de champ clôturés (bocage) ou bien des champs ouverts (*open-field*) ou bien encore d'un espace urbain ? Cet exemple est tiré de Himpens, Poulhes, et Sémécurbe (2021) qui cherche

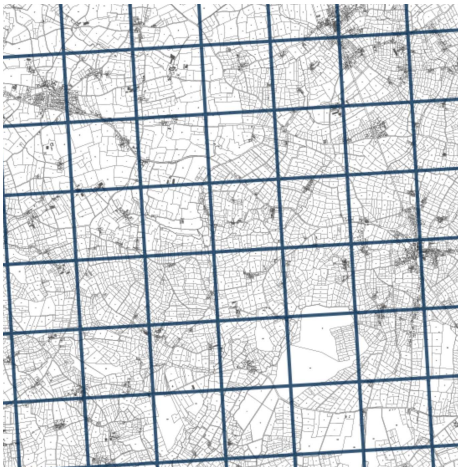
à mettre en regard la localisation effective des nouvelles constructions à ces différentes configurations paysagères.

3.3.1 Reconnaître les paysages à partir de la forme des parcelles cadastrales

En zone peu dense en France, il existe deux types principaux de paysages. D'une part, des paysages dits de champs ouverts ou « *open-field* » (sans clôture ni muret). Les parcelles cadastrales y sont alors allongées et alignées au sein des champs. Le réseau routier y est plutôt peu dense. Le bâti y est regroupé en villages et hameaux. D'autre part, des paysages dits de champs clôturés (par des haies ou des murets). Les parcelles cadastrales y sont plus petites et presque carrées. De nombreux chemins relient les parcelles cadastrales. Le bâti est souvent dispersé. La figure 27 page suivante des photographies aériennes des départements de la Marne (51) et de la Manche (50) illustre cette opposition entre les parcelles et les paysages typiques de ces deux départements, la Marne étant caractérisée par des champs fermés, la Manche par des champs ouverts.

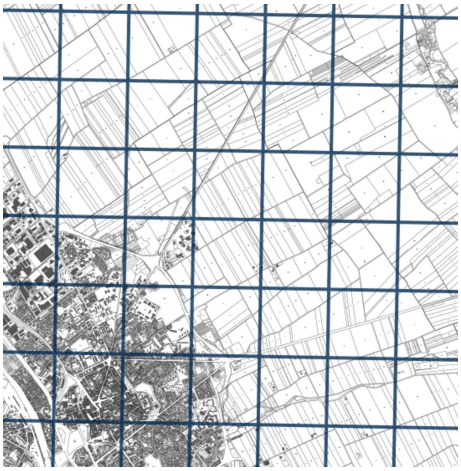
Il n'existe pas de répertoire exhaustif des types catégorisant l'ensemble du territoire métropolitain. Pour en établir un, on va entraîner un réseau de neurones convolutif à prédire si une image de parcelles cadastrales relève du type « champ fermé » appelé aussi « bocage », du type « champ ouvert » ou « open-field », ou encore d'un espace urbain, classé à part dans une troisième catégorie. Ce réseau de neurones exploite le contour des parcelles cadastrales sur l'image pour les classer dans l'une de ces trois classes. Il s'agit d'un problème de classification. Le réseau convolutif présenté dans la section précédente est une méthode *supervisée* : l'estimation des paramètres du réseau est réalisée à partir d'un échantillon préalablement labellisé.

Photographie aérienne de la Marne (51)



Parcelles cadastrales de la Marne (51)

Photographie aérienne de la Manche (50)



Parcelles cadastrales de la Manche (50)

GRAPHIQUE **27 – Photographies aériennes et parcelles cadastrales de deux départements types**

3.3.2 Les données et les échantillons d'entraînement et de test

Les images sont obtenues à partir du contour des parcelles cadastrales mis à disposition sur des données cadastrales ouvertes (ici). Ces données d'origine décrivent les contours de chaque parcelle sous forme de polygones définis par des suites de coordonnées géographiques. Pour chaque département, nous construisons une image en noir et blanc de l'ensemble du département montrant les limites des parcelles. Les images ne contiennent aucune autre information que ces contours.

Pour entraîner un prédicteur de type de parcelle, on doit disposer d'un échantillon d'images de parcelles bien classées en « bocage », « champ ouvert » ou « urbain », mais il n'existe pas de tel échantillon. Il s'agit évidemment d'un enjeu central en *machine learning*

en général : on a besoin de données « étiquetées », ou labellisées, en très grande quantité, mais il n'est pas toujours possible d'en trouver.

Dans notre cas, il serait fastidieux de construire des échantillons d'entraînement et de test à la main. Nous avons ici recours à une autre méthode : nous sélectionnons deux départements, la Marne et la Manche, reconnus par les géographes comme incarnant les deux types de paysage. Nous faisons l'hypothèse que toute image non urbaine provenant de la Marne sera représentative d'un « champ fermé », et toute image non urbaine provenant de la Manche d'un « champ ouvert » et constituons ainsi nos échantillons d'apprentissage et de test sur lesquels nous entraînons et testons notre prédicteur. En pratique, nous entraînons plutôt notre prédicteur à reconnaître des images (non urbaines) de type « Marne » et des images de type « Manche ». Cette approximation réduit sans doute l'efficacité du prédicteur, puisque les exemples d'entraînement ne sont pas classés de manière idéale. Elle pourrait aussi poser des problèmes de généralisation pour certains paysages totalement absents des deux départements d'entraînement. Ces enjeux sont discutés en conclusion du chapitre.

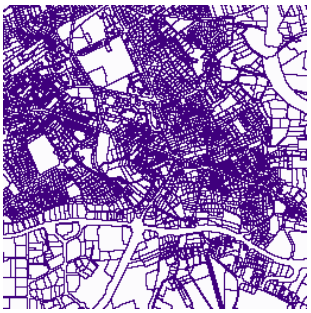
L'échantillonnage consiste à tirer aléatoirement des images de 1km de côté environ avec une résolution de 256 x 256 pixels (1 pixel représente 4 mètres) dans les images des départements. L'échantillon est constitué de 25 000 images relevant de la Manche (50) toutes classées en « bocage » (sauf celles des espaces urbains) et de 25 000 images de la Marne (51), toutes classées en « *open-field* » (sauf celles des espaces urbains). Il sera ensuite partagé entre échantillons d'entraînement et de test. Les images avec moins de 80 % de leur surface occupée par des parcelles sont retirées de l'analyse : il s'agit principalement d'images sur la côte bordant la mer ou en limite de département. Les images représentant des espaces urbains/villes ont préalablement été identifiées en s'appuyant sur la couche du cadastre représentant les bâtiments. On identifie ainsi des regroupements de bâtiments (continuité du bâti) à l'aide de l'algorithme DBSCAN[34]. Pour chaque regroupement, l'enveloppe convexe des bâtiments définit un « espace urbain ». Les images ayant plus de 50 % de leur surface dans un « espace urbain » sont considérées comme des villes. Le programme figure dans le script preprocessing.py.

La construction des échantillons d'images est détaillée dans le même script. Notons qu'il a fallu **augmenter artificiellement le nombre d'images classées en espaces urbains** dans l'échantillon d'entraînement car celui-ci était trop faible relativement aux autres catégories. Les premiers tests avaient en effet des performances médiocres. L'approche retenue de *data augmentation* est plutôt fréquente en analyse des images lorsque l'on cherche à apprendre à identifier une classe sous-représentée dans l'échantillon initial. On a donc transformé certaines images relatives aux villes de l'échantillon d'apprentissage et ajouté ces images transformées à l'échantillon d'apprentissage initial. Ces images transformées sont obtenues en effectuant des permutations (voir graphique 28 page suivante). D'autres

34. *Density-Based Spatial Clustering of Applications with Noise*, algorithme de regroupement d'entités en fonction de la densité locale proposé par Ester, Kriegel, Sander, Xu, et al. (1996).

transformations sont possibles comme des permutations des images en diagonale, des rotations, etc..

Image initiale



Permutation verticale

Permutation horizontale

Permutation horizontale et verticale

GRAPHIQUE **28 – Exemples d'augmentation artificielle de données**

3.3.3 L'architecture du réseau retenue et son implémentation

Un premier réseau avec 6 couches de convolution est estimé à l'aide de *Keras* sous Python. Un perceptron multicouche classe les données à l'issue des convolutions en trois classes. La classe obtenue est le maximum des trois nœuds (softmax). Le réseau possède ainsi la structure classique vue sur le figure 25. Le programme figure dans le répertoire sous le github. Il peut être exécuté sur la plateforme datalab.sspcloud.fr.

Sous *Keras*, le réseau est défini de façon séquentielle. On définit la couche de neurones d'entrée par :

```
inputs = Input(shape=(256, 256, 1))
```

Ici, le réseau prend en entrée un tenseur (vecteur multidimensionnel contenant des données) de taille 256 x 256 (nombre de pixels de l'image) x 1 (il y a un seul canal car l'image est en noir et blanc. Il pourrait y en avoir trois pour une image couleur (rouge, vert, bleu). Ensuite viennent les couches de convolution :

```
conv_1 = Conv2D(32, (s,s), strides=(1,1))(inputs)
```

```
act_1 = Activation('relu')(conv_1)
maxpool_1 = MaxPooling2D(pool_size=(2, 2), strides=(2, 2))(act_1)
```

La couche de convolution *conv_1* comporte 32 filtres (nombre de filtres différents estimés dans la couche, on parle aussi de "profondeur" des filtres de convolution), possède une taille s (taille du filtre fixée à 5 dans cet exemple). Le paramètre *strides* indique le nombre de pixels dont le filtre se déplace à chaque fois. Ici, il ne se déplace que d'un pixel à chaque fois. L'option *padding* indique au programme la façon dont le programme doit prolonger les images sur les bords de façon à ne pas provoquer de rétrécissement de l'image en sortie (effets de bords des moyennes mobiles). Les couches de convolution s'enchaînent jusqu'à la sixième :

```
conv_6 = Conv2D(16, (s,s), strides=(1,1), padding='same')(maxpool_5)
act_6 = Activation('relu')(conv_6)
maxpool_6 = MaxPooling2D(pool_size=(2, 2), strides=(2, 2))(act_6)
```

Ensuite, les valeurs en sortie de la 6ème couche de *pooling* sont placées dans un tenseur unidimensionnel (vecteur).

```
flat_1 = Flatten()(maxpool_6)
```

Ce vecteur est ensuite placé en entrée du perceptron multicouche qui effectue la classification des images en trois catégories (*open-field*, bocage et ville) :

```
fc_1 = Dense(128)(flat_1)
act_3 = Activation('relu')(fc_1)

fc_2 = Dense(64)(act_3)
act_4 = Activation('relu')(fc_2)

fc_3 = Dense(3)(act_4)
fc_3a = Activation("softmax")(fc_3)
```

Une activation de type *softmax* permet de normaliser la somme des trois valeurs de la couche de sortie (*fc_3*) à 1. La sortie *fc_3a* de la couche *softmax* peut donc s'interpréter comme le vecteur des probabilités d'appartenance à chaque type. C'est la sortie du RN. A partir de cette prédiction du réseau, il est possible de sélectionner le type correspondant à la probabilité prédite maximale.

Le modèle *convnet* est défini comme allant de la couche *input* à la couche d'activation finale *fc_3a* :

```
convnet = Model(inputs, fc_3a)
```

Le modèle doit ensuite être compilé :

```
convnet.compile(optimizer='adam', loss='binary_crossentropy')
```

Il peut ensuite être entraîné sur les images. L'algorithme choisi est *adam* qui donne de bons résultats en pratique pour de nombreux problèmes. La fonction de coût est l'entropie croisée catégorielle car il s'agit d'un problème de classification à trois catégories.

Remarque : les catégories (« bocage », « *open-field* » et ville) doivent être placées sous forme d'indicatrices afin de correspondre au format de sortie du réseau (3 neurones). Cela se fait simplement à l'aide de la librairie *Keras*.

3.3.4 Choix des hyperparamètres

L'ensemble des combinaisons possibles d'hyperparamètres est très grand. Il est difficile de le tester à l'aide d'une grille afin d'améliorer les performances. Certains tests ont toutefois été réalisés de façon indépendante sur le nombre de couches de convolution, le nombre de couches cachées ou encore la taille des filtres. Les hyperparamètres sont testés un à un en fixant tous les autres. Cela donne une idée de l'influence de ces paramètres sur les résultats.

Nombre de couches de convolution

Nous illustrons cette idée sur l'exemple du nombre de couches de convolution. L'échantillon d'entraînement est subdivisé en deux (un nouvel *x_train*) et (*x_validation*). Le réseau sera entraîné sur la première partie en parcourant une grille de valeurs possibles pour le nombre de couches de convolution. Les performances du réseau selon le nombre de couches de convolution seront évaluées sur la deuxième partie. On retiendra le nombre de couches de convolution conduisant aux meilleures performances.

En pratique, on place la définition séquentielle du modèle précédent dans une fonction *define_model_structure* avec comme paramètres tous les hyperparamètres pertinents définissant le convnet :.

```
def define_model_structure(
    input_dim,
    filters,
    kernel_sizes,
    strides_conv,
    padding,
    pool_sizes,
    strides_maxpool,
    dim_dense_layers,
    activations,
):
```

La fonction définit d'abord une première couche de convolution :

```
    nb_conv_layers = len(filters)
```

```
# Initialisation of the model
model = Sequential()
# First layer
model.add(
    Conv2D(
        filters[0],
        kernel_sizes[0],
        strides=strides_conv[0],
        activation=activations[0],
    )
)
model.add(MaxPooling2D(pool_size=pool_sizes[0], strides=strides_
    ↪ maxpool[0]))
```

puis on ajoute le nombre de couches supplémentaires nécessaire :

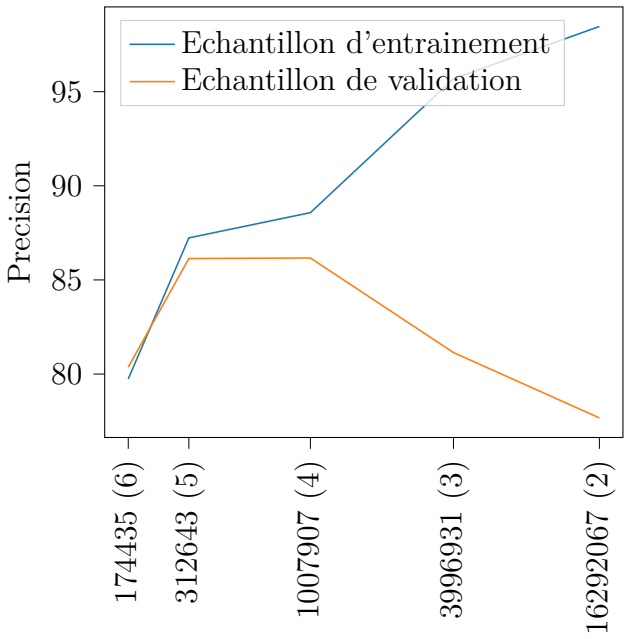
```
# Conv layers
for conv_lay in range(1, nb_conv_layers):
    model.add(
        Conv2D(
            filters[conv_lay],
            kernel_sizes[conv_lay],
            strides=strides_conv[conv_lay],
            padding=padding,
            activation=activations[conv_lay],
        )
    )
    model.add(
        MaxPooling2D(
            pool_size=pool_sizes[conv_lay], strides=strides_maxpool[
                ↪ conv_lay]
        )
    )

# Dense layers
for dense_lay in range(len(dim_dense_layers)):
    model.add(
        Dense(
            dim_dense_layers[dense_lay],
            activation=activations[nb_conv_layers + dense_lay],
        )
    )

nb_layers = len(filters) * 2 + len(dim_dense_layers) + 1
```

```
    input_used = Input(shape=input_dim)
    layers = model.layers[0](input_used)
    for i in range(1, nb_layers):
        layers = model.layers[i](layers)

    return Model(input_used, layers)
```

On peut ensuite appeler cette fonction successivement en faisant varier le nombre de couches de 2 à 6 et calculer les performances (mesurées ici par la précision, c'est-à-dire le pourcentage d'images biens classées) sur l'échantillon (*x_validation*).

```
for nb_couche in range(2, 6):

    model = tt.define_model_structure(
        (256, 256, 1),
        filters=[32] * nb_couche,
        kernel_sizes=[(5, 5)] * nb_couche,
        strides_conv=[(1, 1)] * nb_couche,
        padding="same",
        pool_sizes=[(2, 2)] * nb_couche,
        strides_maxpool=[(2, 2)] * nb_couche,
        dim_dense_layers=[128, 64, 3],
        activations=["relu"] * nb_couche + ["relu", "relu", "softmax"],
    )
    model.compile(optimizer=optimizer, loss=loss)

    model.fit(
        x_train,
        y_train,
        epochs=5,
        batch_size=218,
        shuffle=True,
        validation_data=(x_validation, y_validation),
    )
```

Ici, la succession de couches de convolution et de *pooling* rend l'image de plus en plus petite, ce qui conduit à moins de données en entrée du perceptron multicouche, et moins de paramètres à estimer. Le nombre total de poids du réseau diminue. Sur le graphique (29), le réseau sur-apprend lorsqu'il y a peu de couches de convolution (moins de 5). Ses performances sont en effet bien meilleures sur l'échantillon d'entraînement que sur celui de validation. A l'opposé, utiliser trop de couches de convolution dégrade les performances du réseau (pas assez de paramètres). On retient cinq couches de convolution.

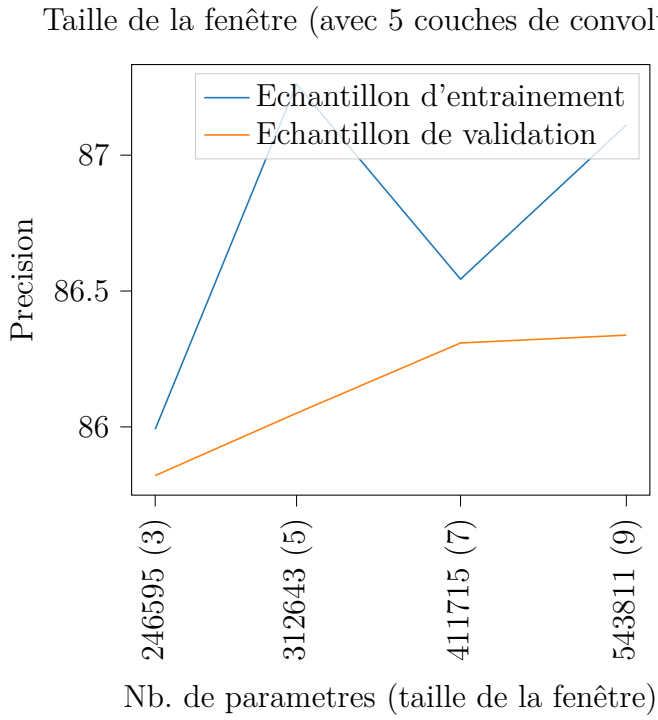


Nb. de parametres (nb. de couches), axe logarithmique

GRAPHIQUE **29 – Pourcentage (%) d'images biens classées selon le nombre de couches de convolution**

Taille des filtres des couches de convolution

On procède de même, sur une grille, pour choisir la taille des filtres des couches de convolution. La précision du réseau est relativement stable pour différents choix de taille de filtres (figure 30 page suivante). Le réseau devient cependant de plus en plus long à entraîner (de 3 minutes 41 secondes pour une taille de trois à 8 minutes 29 secondes pour une taille de sept). Il est possible que l'entraînement soit plus instable. Une taille de sept semble donner de bons résultats.



GRAPHIQUE **30 – Choix de la taille des filtres de convolution**

Profondeur des filtres de convolution : exemple d'utilisation de la librairie HyperOpt
Le nombre de filtres par couche de convolution ou profondeur des filtres de convolution est difficile à choisir car en faisant varier l'ensemble des paramètres possibles le nombre de modèles à estimer grandit rapidement. Avec cinq couches de convolution et seulement trois valeurs possibles pour ce paramètre $(16, 32, 48)$, le nombre de combinaisons à tester est de 243. Une façon automatisée consiste à mobiliser la librairie *HyperOpt*.

La librairie *HyperOpt* propose une approche bayésienne pour choisir des jeux d'hyperparamètres tout en minimisant une fonction de coût spécifiée par l'utilisateur. On va tirer aléatoirement dans des lois *a priori* spécifiées par l'utilisateur des valeurs possibles d'hyperparamètres, puis choisir à chaque itération les candidats (jeux d'hyperparamètres) les plus susceptibles de minimiser la fonction de coût. L'algorithme utilisé ici est le *tree parzen estimator* (Bergstra, Bardenet, Bengio, et Kégl, 2011). Cet algorithme choisit à chaque itération le jeu de paramètres qui maximise l'espérance de diminution de la fonction de coût. Plutôt que de modéliser la loi de la fonction de coût en fonction des paramètres, il s'appuie sur une modélisation de la loi du jeu de paramètres en fonction de la valeur de la fonction de coût pour tirer les candidats potentiels.

Pour utiliser la librairie, on doit définir une fonction *objectif* qui prend en entrée les valeurs des hyperparamètres que l'on souhaite optimiser et retourne en sortie une métrique de performance prédictive sur l'échantillon d'évaluation, après avoir construit et entraîné un modèle. On définit également un espace des hyperparamètres et une loi de

probabilité initiale sur cet espace. *HyperOpt* explore ensuite cet espace (avec un nombre d'itération maximum fixé). Plus de précisions sur la librairie *HyperOpt* peuvent être trouvées sur http ://hyperopt.github.io/hyperopt/. La librairie *Hyperas* des mêmes auteurs permet de mobiliser *HyperOpt* via une syntaxe simplifiée pour Keras. On la trouve sur https ://github.com/maxpumperla/hyperas.

Certains hyperparamètres peuvent être interdépendants : par exemple, le choix du nombre de couches de convolution affecte la structure globale du réseau et donc le nombre de profondeurs de filtres à sélectionner. Dans ce cas, le choix des hyperparamètres sur la base d'une grille de valeurs possibles peut rapidement s'avérer très complexe. La librairie *HyperOpt* permet de contourner cette difficulté. Elle offre en effet la possibilité de tirer les paramètres à partir d'un arbre. Un premier paramètre est tiré (le nombre de couches de convolution du réseau) puis on tire les autres hyperparamètres en fonction du résultat (la taille des filtres de convolution).

Les résultats

Les tests réalisés sur les hyperparamètres nous conduisent à sélectionner : 5 couches de convolution, avec une taille de filtre de 7 et une profondeur de 48 (voir tableau des paramètres 5). Cependant, pour des raisons de rapidité de calcul, les tests ont été réalisés avec un nombre d'itérations (*epochs*) égal à 5 ce qui est relativement faible. Si on augmente le nombre de couches et que l'on passe à un nombre d'itérations égal à 10, les résultats s'améliorent. Au final, après quelques tests, le réseau est le suivant :

Type de couche	dimension ($hauteur \times largeur \times profondeur$)
Image en entrée	$256 \times 256 \times 1$
Convolution 1 (7×7) + ReLu	$250 \times 250 \times 16$
Max Pooling 1 (2×2)	$125 \times 125 \times 16$
Convolution 2 (7×7) + ReLu	$125 \times 125 \times 16$
Max Pooling 2 (2×2)	$63 \times 62 \times 16$
Convolution 3 (7×7) + ReLu	$62 \times 62 \times 32$
Max Pooling 3 (2×2)	$31 \times 31 \times 32$
Convolution 4 (7×7) + ReLu	$31 \times 31 \times 32$
Max Pooling 4 (2×2)	$15 \times 15 \times 32$
Convolution 5 (7×7) + ReLu	$15 \times 15 \times 48$
Max Pooling 5 (2×2)	$7 \times 7 \times 48$
Convolution 6 (7×7) + ReLu	$7 \times 7 \times 48$
Max Pooling 6 (2×2)	$3 \times 3 \times 48$
Couche de perceptrons 1 + ReLu	128 noeuds
Couche de perceptrons 2 + ReLu	64 noeuds
Couche de perceptrons 3	3 noeuds

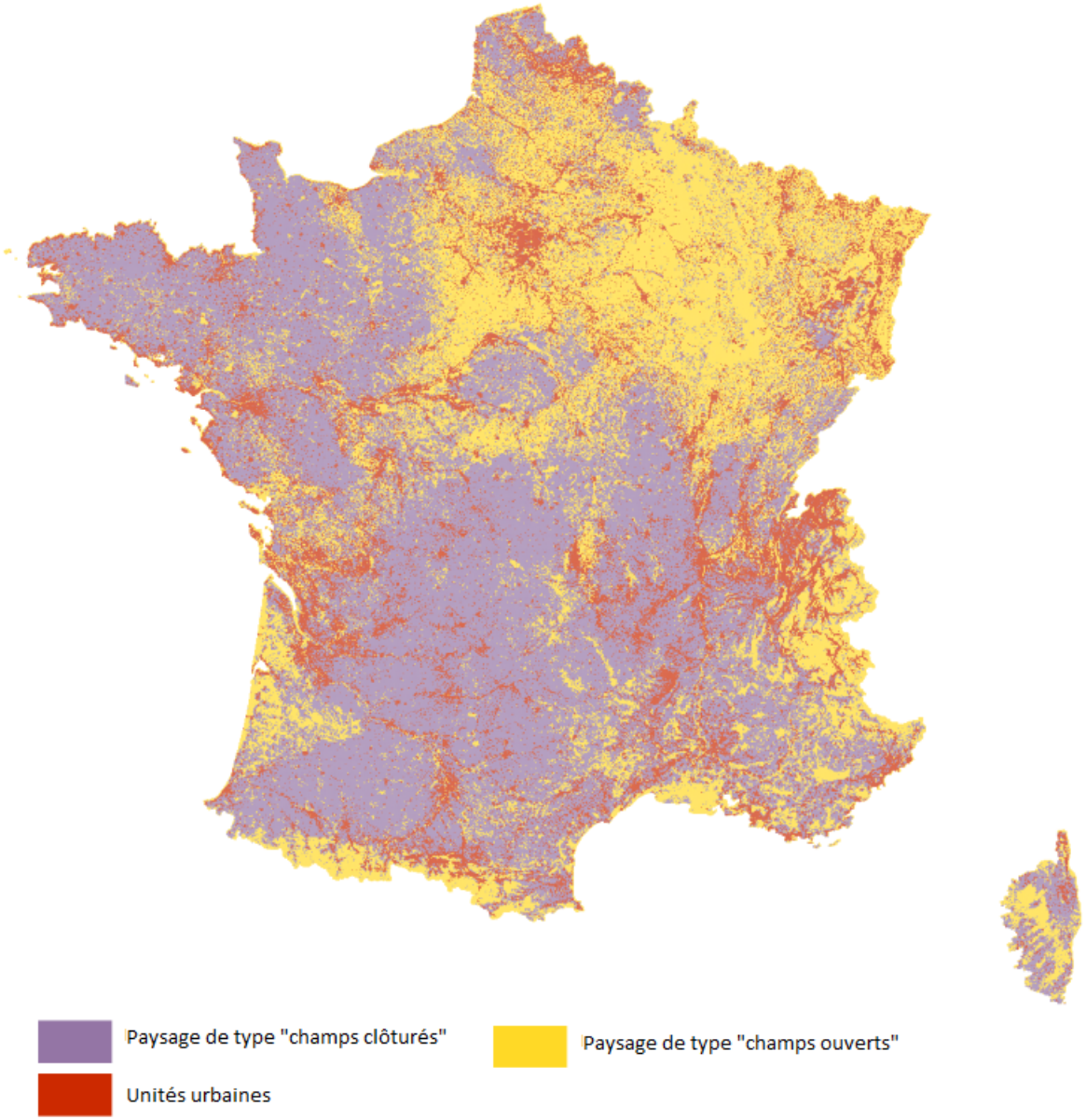
TABLEAU **5 – Choix du modèle final**

Sur l'échantillon de test, le pourcentage d'images bien classées atteint 88,2 %. Les performances détaillées sont présentées dans le tableau 6.

	Labels prédits par le réseau			
	« **Bocage** »	« ***Open-field*** »	**Villes**	
Vrais labels				
Manche (« bocage »)	92,7 %	2,3 %	5,0 %	100 %
Marne (« *open-field* »)	4,9 %	87,3 %	7,7 %	100 %
Villes	9,7 %	6,8 %	83,5 %	100 %

TABLEAU **6 – Performance du réseau sur l'échantillon de test**

Une fois le réseau entraîné, il est possible de classer dans ces trois catégories l'ensemble des images (carrés de 1024 x 1024 mètres) représentant les parcelles de la France métropolitaine (voir carte 31 page suivante).



GRAPHIQUE **31 – Type de parcelles déterminé par le réseau de neurones**

Au-delà de la validation par l'échantillon de test, qui teste avant tout la capacité du réseau à distinguer les parcelles de la Manche et de la Marne, des éléments complémentaires renforcent la crédibilité des résultats. Tout d'abord les zones identifiées par l'algorithme comme champs ouverts et champs clôturés sont cohérentes avec les résultats connus par ailleurs (opposition entre l'ouest et le sud-ouest à dominante de champs fermés, et le nord et le nord-est de la France à dominante de champs ouverts déjà décrits dans Dion (1934) à comparer à la carte 31). Ces zones semblent relativement homogènes, tout comme les

phénomènes géographiques qu'elles modélisent (il y a peu de carreaux violets au milieu des carreaux jaunes).

Il semble toutefois y avoir trop de villes sur la carte obtenue avec le réseau de convolution. Du rouge apparaît au sommet des Alpes et dans la région viticole de Bordeaux, témoignant de types de paysages spécifiques. En effet, il n'y a pas de catégorie de montagnes, ni d'observations relatives à des montagnes dans l'échantillon d'entraînement (les départements de la Marne et de la Manche). La Marne possède des paysages viticoles mais les images correspondantes n'ont pas de label spécifique. Elles sont peu présentes dans l'échantillon d'apprentissage (peu de parcelles sont concernées). Le réseau apprend majoritairement à classer les autres parcelles. Cela explique les piètres performances du réseau dans certaines zones, notamment les autres zones viticoles (comme la région de Bordeaux) qui se caractérisent par de petites parcelles.

Pour autant, le réseau de convolution montre des performances supérieures à d'autres méthodes de *machine learning*. Nous comparons au résultat de forêts aléatoires, un algorithme puissant sur des problèmes complexes, mais peu efficace pour traiter directement des images. On entraîne donc les forêts aléatoires sur des indicateurs géographiques de compacité et de taille des parcelles. Le nombre d'images biens classées sur l'échantillon test y atteint 82 %, ce qui est inférieur au résultat obtenu par le réseau de convolution (88 %) (voir run_random_forest.py).

Le type de paysages prédit par le réseau peut ensuite être utilisé afin d'étudier la localisation des nouvelles constructions. Grâce à cette méthodologie, il est possible d'estimer que 82 % des nouveaux bâtiments se font en continuité de bâti dans les paysages de champs ouverts, contre 65 % dans ceux de champs clôturés (cf. Himpens, Poulhes, et Sémécurbe (2021)).

Vérifier que le réseau identifie bien le type de parcelles

Il est possible d'identifier les zones modifiant le plus la classe prédite *via* l'observation de la contribution au gradient de certaines valeurs (algorithme Grad-CAM, voir Selvaraju, Cogswell, Das, Vedantam, Parikh, et Batra (2019)).

La première étape consiste à récupérer $\frac{\partial y^{max}}{\partial A_{i,j,k}}$, le gradient du score pour la classe prédite y^{max} avant l'activation softmax par rapport à l'activation $A_{i,j,k}$ de la dernière couche de convolution (en (i, j) pour le filtre k). Pour cela, dans *Keras*, on extrait les *names* des couches du modèle, à l'aide desquels on récupère le sous-modèle entre la couche d'entrée (*Inputs*) et la dernière couche de convolution (avant le perceptron multicouche) :

```
last_conv_layer = model.get_layer(last_conv_layer_name)
last_conv_layer_model = keras.Model(model.inputs, last_conv_layer.output)
```

ainsi que le perceptron multicouche :

```
classifier_input = keras.Input(shape=last_conv_layer.output.shape[1:])
x = classifier_input
for layer_name in classifier_layer_names:
    x = model.get_layer(layer_name)(x)
classifier_model = keras.Model(classifier_input, x)
```

classifier_layer_names contient le nom de toutes les couches après la dernière couche de convolution. On récupère la prédiction pour la deuxième image de l'échantillon test (en prenant soin d'enregistrer le gradient) :

```
img_array = x_test[2:3,]

with tf.GradientTape() as tape:
        # Compute activations of the last conv layer and make the tape
            ↪ watch it
        last_conv_layer_output = last_conv_layer_model(img_array)
        tape.watch(last_conv_layer_output)
        # Compute class predictions
        preds = classifier_model(last_conv_layer_output)
        top_pred_index = tf.argmax(preds[0])
        top_class_channel = preds[:, top_pred_index]
```

On récupère les gradients $\frac{\partial y^{max}}{\partial A_{i,j,k}}$ pour tous les (i, j) de la dernière couche de convolution pour tous les filtres k.

```
grads = tape.gradient(top_class_channel, last_conv_layer_output)
```

Les gradients sont ensuite moyennées sur les dimensions (i, j). Il n'y a plus qu'une seule valeur $\alpha_k^{max} = \sum_{i=1,j=1}^{I,J} \frac{\partial y^{max}}{\partial A_{i,j,k}}$ par filtre k.

```
pooled_grads = tf.reduce_mean(grads, axis=(0, 1, 2))
```

Les activations situées en (i, j) de la couche k sont multipliées par la moyenne du gradient α_k^{max} sur la couche k.

```
last_conv_layer_output = last_conv_layer_output.numpy()[0]
pooled_grads = pooled_grads.numpy()
for i in range(pooled_grads.shape[-1]):
    last_conv_layer_output[:, :, i] *= pooled_grads[i]
```

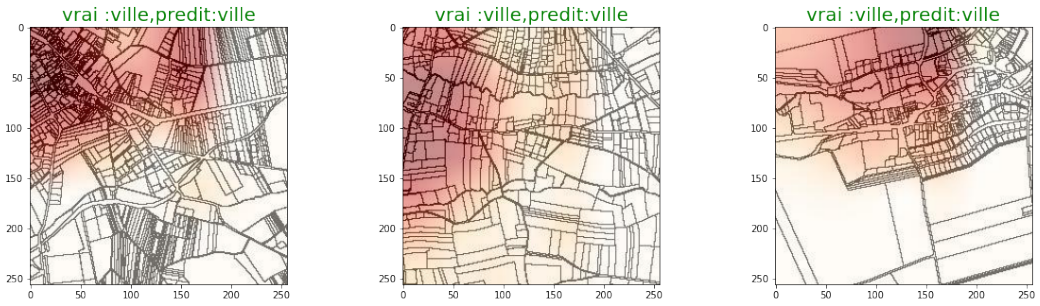
On effectue la moyenne des valeurs sur toutes les profondeurs de filtre. Il n'y a plus qu'une seule valeur par localisation (i, j).

```
heatmap = np.mean(last_conv_layer_output, axis=-1)
```

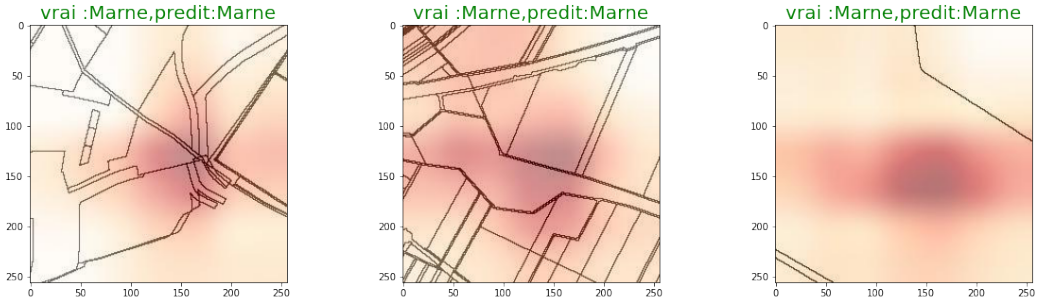
Seules les valeurs positives sont conservées. Elles sont normalisées.

```
heatmap = np.maximum(heatmap, 0) / np.max(heatmap)
```

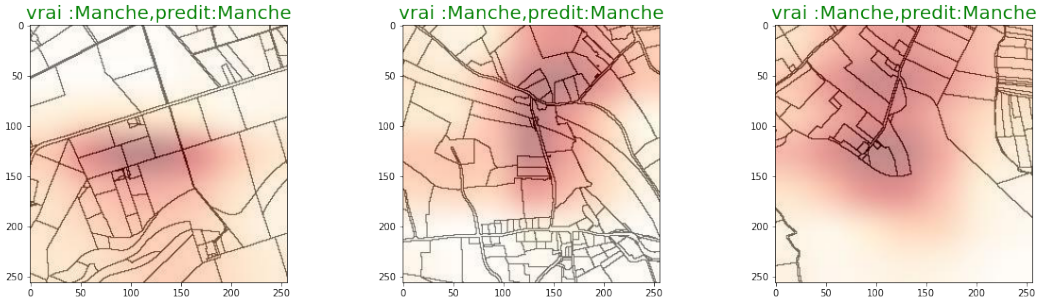
On superpose ensuite les valeurs de l'algorithme *Grad-cam* avec l'image initiale. Sur la Figure 32, les zones urbaines semblent clairement identifiées. Pour les images de la Marne, le réseau semble identifier la présence de parcelles allongées et d'angles droits. Pour les images de la Manche, ce sont les tracés courbes qui apparaissent et certaines routes. Ces constatations peuvent être vérifiées en affichant les activations des couches de convolution successives.



(a) Images urbaines



(b) Images de la Marne



(c) Images de la Manche

GRAPHIQUE **32 – Zones contribuant le plus au choix final**

Notes : L'intensité de la couleur correspond aux valeurs de *Grad_cam* : les zones les plus colorées sont celles qui ont le plus d'influence sur le classement de l'image dans une catégorie plutôt qu'une autre

Le comportement du réseau semble conforme à ce qui est attendu. Cela n'est pas toujours le cas. Dans l'exemple du chapitre 1 sur les chaises et les lits, le réseau convolutif attache beaucoup d'importance à l'arrière plan (sombre pour les lits et clair pour les chaises). Les résultats obtenus semblent de ce fait très dépendants de la base d'apprentissage. En analyse d'images, les jeux de données utilisés doivent être conséquents sous peine de tomber dans ce genre de travers.

3.3.5 Performances requises

L'entraînement des réseaux de cette partie ne prend que quelques minutes avec une GPU (carte graphique) mais plus d'une heure sans GPU. Des techniques traditionnelles (comme une classification des parcelles à partir d'indicateurs de forme) permettent d'atteindre des résultats honorables sans avoir besoin d'une architecture dédiée.

3.3.6 Intérêt des réseaux de convolution pour la statistique publique et limites

Cet exemple montre qu'il est possible de construire une variable à partir d'images. Pour autant, dans cet exemple, l'assimilation de toutes les images de la Manche à des paysages de type « champs clôturés » et de toutes les images de la Marne à des paysages de type « champs ouverts » est réductrice. Cela relève d'une construction. Dans la réalité, les deux types de paysages pourraient ne pas être si homogènes que cela. L'obtention de données correctement labellisées, par un long travail manuel, ou par un judicieux croisement de sources, augmenterait beaucoup la pertinence des résultats. Sur cet exemple spécifique, d'autres sources d'images que le cadastre sont d'ailleurs envisageables : données cartographiques, photos aériennes ou images satellites pourraient apporter des informations complémentaires (nature des voies, couvert végétal, etc.).

D'autres applications en statistique publique sont possibles : classification de produits à partir de leurs images, utilisation de photos satellites afin de détecter certains équipements... La principale limite est la disponibilité de jeux de données déjà labellisées pour la phase d'entraînement. Certaines tâches sont désormais classiques en *deep learning*. Le problème de la classification de meubles a été exploré de multiples fois. Les problèmes de la statistique publique peuvent cependant être très spécifiques (il n'existe pas de base pré-annotée du cadastre par exemple). Cela oblige à certaines approximations.

Il est important de rappeler que généralement lorsque l'échantillon d'apprentissage est trop petit, le réseau de neurones risque de s'attacher à des caractéristiques trop spécifiques. Par exemple, il apprend à classer des images en fonction des fonds clairs ou foncés des images comme dans l'exemple des meubles. L'algorithme n'est alors pas généralisable à d'autres types de données. Il est possible de palier ce manque de données en utilisant des réseaux préentrainés. Cette méthode de *transfer learning* consiste à récupérer un réseau entraîné à résoudre un problème proche puis de le ré-entraîner éventuellement en fixant certains poids sur nos données.

4 Réduire la dimension d'une enquête avant de prédire à l'aide d'un auto-encodeur

Les deux sections précédentes présentent des méthodes d'apprentissage *supervisé*, qui cherchent à prédire la valeur d'une variable, connue seulement pour un échantillon de données dites annotées, labellisées. Il existe également des méthodes d'*apprentissage non supervisé* qui cherchent à faire émerger des relations présentes dans les données mais non connues à l'avance sur un échantillon, à la manière des méthodes descriptives d'analyse des données. Ces méthodes peuvent ainsi s'appliquer à des données non labellisées. Parmi elles, les **auto-encodeurs** (AE) s'attachent à réduire la dimension.

Cette **réduction de dimension** est comparable à l'analyse en composantes principales (ACP). Pour rappel, le principe de l'ACP consiste à résumer en une petite dimension d l'information d'un nuage de points en grande dimension. La méthode sélectionne des axes du nuage orthogonaux entre eux et expliquant le maximum de variance, les axes étant des combinaisons linéaires des variables de départ. On choisit a priori soit la dimension d, soit le pourcentage de variance qu'on souhaite expliquer.

La réduction de dimension vise à simplifier l'analyse, en transformant une base de données avec de très nombreuses variables en une base plus réduite mais qui demeure une bonne approximation des données de départ. C'est une méthode importante en *machine learning*, où les données de grande dimension sont fréquentes. L'auto-encodeur peut être vu comme une généralisation non linéaire de l'ACP (voir l'encadré 1, page 93). L'idée en est apparue dans plusieurs articles au cours des années 1980 sous le nom de réseau encodeur (Ackley, Hinton, et Sejnowski, 1985) ou auto-associatif (Ballard, 1987).

L'AE a généré de nombreux types de RN adaptés à des applications diverses. Il peut servir à visualiser et explorer des données de grande dimension en les projetant (de manière non-linéaire) sur $d = 2$ ou $d = 3$ dimensions (Hinton et Salakhutdinov, 2006). Cette projection facilite le calcul de distances entre observations et peut être un préalable à des algorithmes de *clustering* ou de plus proches voisins.

Les AE ont également été une solution au problème d'initialisation des poids ou de pré-entraînement dans un RN. On entraîne un RN comme AE dans un premier temps, puis comme prédicteur sur une tâche supervisée qui est, généralement, plus pauvre en information et dont l'entraînement est plus ardu. On peut effectuer cette initialisation couche par couche, en approfondissant progressivement l'AE comme *stacking autoencoder* (Ranzato, Poultney, Chopra, et LeCun, 2006).

Une autre application concerne le débruitage de photos ou d'enregistrement sonores par exemple. À condition de disposer d'un échantillon de données non bruitées, et d'un modèle du bruit, on peut bruiter artificiellement les données d'entraînement et entraîner un AE à reconstituer les données non bruitées initiales à partir des données bruitées. La rencontre de ces deux idées, pré-entraînement et débruitage a été fertile : en bruitant les données en input, on contraint l'AE à extraire des représentations robustes des données,

renforçant ainsi sa performance comme méthode de pré-entraînement ou d'initialisation des poids (Vincent, Larochelle, Bengio, et Manzagol, 2008). Plus récemment, ces structures de débruitage sont devenues centrales dans des modèles performants et légers de génération d'images (Rombach, Blattmann, Lorenz, Esser, et Ommer, 2022).

Dans ce chapitre, nous nous concentrons sur l'utilisation de l'AE pour le pré-entraînement. La technique de réduction de dimension par auto-encodeur est illustrée par une application sur l'enquête Emploi. Les observations de l'enquête, qui comprend environ 700 variables, sont réduites à des vecteurs de dimension réduite, arbitrairement fixée à 100, grâce à un auto-encodeur. Dans un second temps, ces vecteurs sont utilisés comme inputs pour l'entraînement d'un réseau de neurones visant à une tâche de prédiction supervisée : la prédiction du risque de chômage au trimestre suivant pour les personnes interrogées. Tous les codes nécessaires sont partagés sur le dépôt github du Chapitre 4.

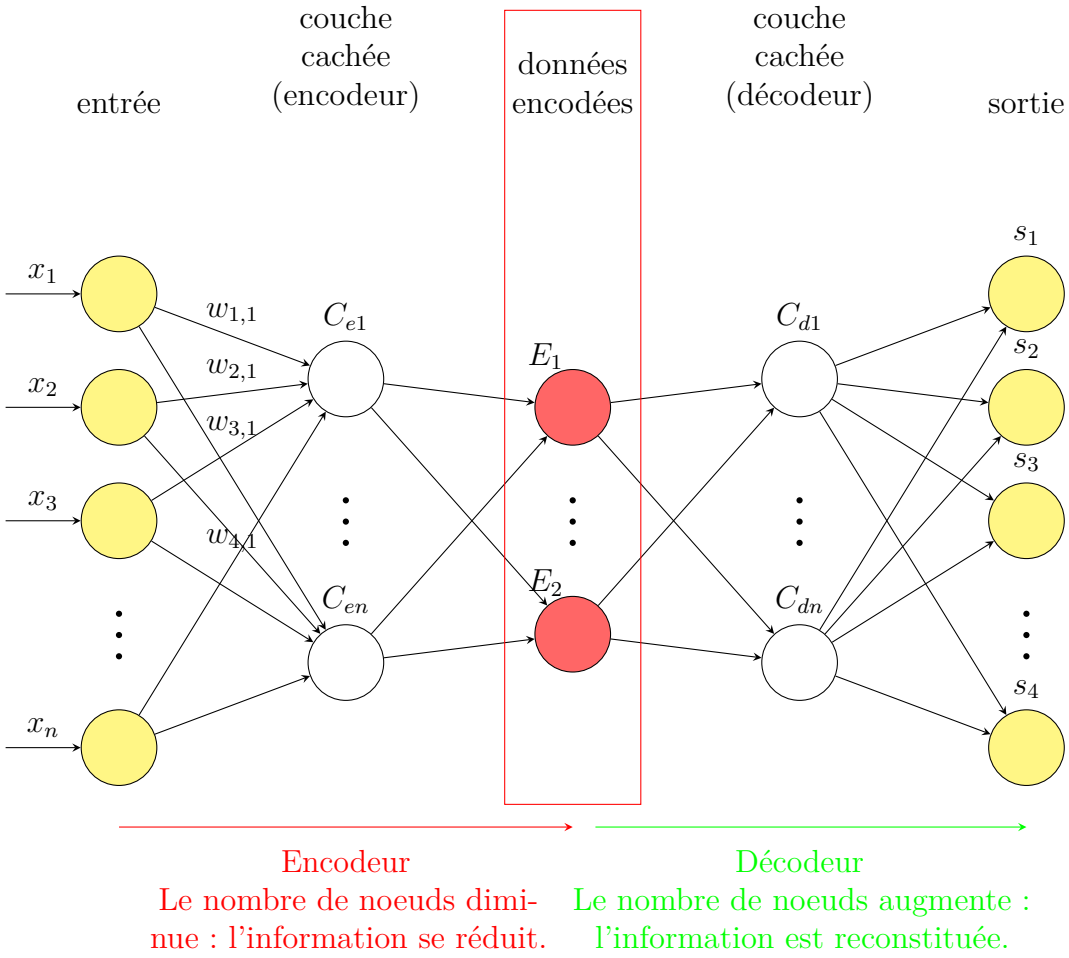
4.1 Le principe des auto-encodeurs

4.1.1 Des sorties identiques aux entrées, mais sous contrainte

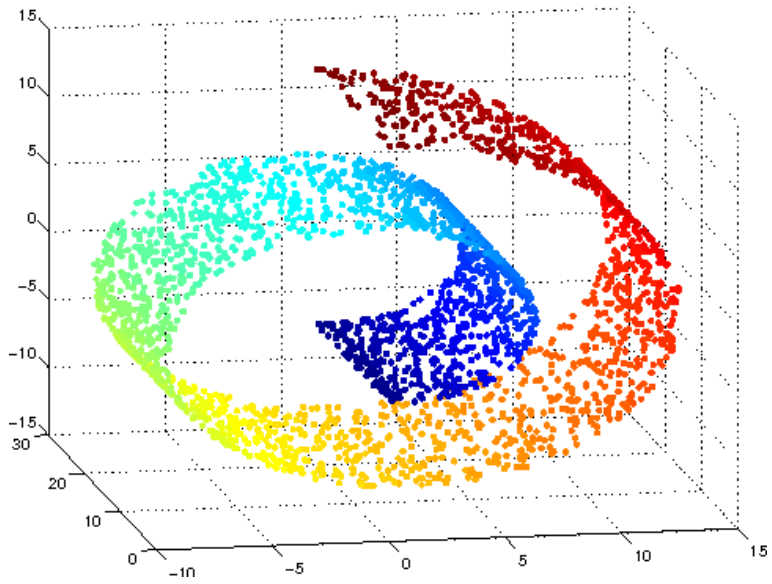
Un auto-encodeur (AE) est un RN constitué de deux parties, un encodeur et un décodeur, avec une couche centrale constituant l'encodage. L'AE est entraîné à prédire en sortie de décodeur les données qu'il reçoit en entrée d'encodeur. L'apprentissage est dit auto-supervisé. Pour éviter d'apprendre la fonction identité, sans intérêt, on ajoute une contrainte. La contrainte la plus simple consiste à limiter la dimension d de la couche d'encodage, et le réseau a alors la forme d'un sablier horizontal (figure 33, page 92). C'est alors dans cette couche centrale que s'opère la réduction de dimension. D'autres types d'auto-encodeurs sont construits à partir de différentes contraintes. Par exemple, pour les auto-encodeurs de débruitage, le bruit ajouté aux inputs tient lieu de contrainte.

Un AE entraîné avec une couche centrale de dimension d permet d'extraire directement le vecteur central de dimension d pour chaque observation : il s'agit de l'observation encodée. On peut donc représenter la totalité du fichier dans un espace vectoriel de dimension d qui peut être très inférieur au nombre de variables du fichier initial.

Une motivation fréquente du recours à l'AE est « l'hypothèse du manifold » (Cayton, 2005). Un *manifold*, ou une variété, est un espace topologique localement euclidien. Plus intuitivement, on peut se représenter un manifold de dimension 2 dans un espace de dimension 3 comme un tissu ou un papier froissé ou roulé (voir graphique 34 page suivante) : des données issues d'un motif en deux dimensions sur cette surface apparaîtront comme un nuage de points en trois dimensions. L'encodage consiste alors à « défroisser le manifold » par une projection non-linéaire. Les données peuvent être représentées par des coordonnées en moindre dimension dans un espace qui les décrit mieux.



GRAPHIQUE **33** – **Un auto-encodeur à partir d'un perceptron multicouche**



GRAPHIQUE **34** – **Une représentation de l'hypothèse du *manifold***

Notes : Figure tirée de Cayton (2005).

Encadré 1: Une généralisation non-linéaire de l'analyse en composantes principales

Un autoencodeur est une généralisation non-linéaire de l'ACP. On peut le montrer en considérant d'abord un AE linéaire. Il s'agit d'un AE dont l'intégralité des fonctions d'activation sont linéaires. Il ne contient qu'une seule couche cachée, la couche d'encodage, car dans un cadre linéaire les couches supplémentaires sont redondantes. Avec une fonction de perte quadratique, l'AE linéaire répond au même programme d'optimisation que l'ACP (Bourlard et Kamp, 1988). Le lien entre AE et ACP dépasse donc l'analogie.

Soit en effet un AE **linéaire** prenant en entrée des données $\mathbf{X}$, p vecteurs de taille n, avec une **couche cachée** de d unités, sans terme de biais, et une matrice des poids $\mathbf{W}$ de dimension $n \times d$. Pour simplifier la démonstration, on suppose en outre que la couche de décodage est la transposée de la couche d'encodage (les mêmes résultats s'obtiennent sans cette contrainte supplémentaire sur la structure de l'auto-encodeur). On a alors :

— Unités de la couche cachée : $h = \mathbf{XW}$

— Unités en sortie de l'AE : $\hat{\mathbf{X}}^{AE} = h\mathbf{W}^T$

D'où,

$$\hat{\mathbf{X}}^{AE} = \mathbf{XWW}^T \tag{1}$$

Et,

$$\underbrace{\underset{w_{i,j}}{\text{argmin}}||\hat{\mathbf{X}}^{AE} - \mathbf{X}||^2}_{\text{progamme de l'AE}} = \underbrace{\underset{w_{i,j}}{\text{argmin}}||\mathbf{X}(\mathbf{WW}^T - I)||^2}_{\text{progamme de l'ACP}} \tag{2}$$

L'optimum de ce programme d'optimisation correspond à la meilleure approximation de la matrice initiale des données par une matrice de rang d, (au sens de la norme de Frobenius), et il est atteint par l'ACP. Entre auto-encodeur linéaire et ACP il existe pourtant deux différences : l'AE atteint l'optimum par l'entraînement (avec délais et erreurs potentielles), et d'autre part l'ACP encode les données dans une base unique de l'espace vectoriel obtenu, la base orthonormée et ordonnée des d plus grands vecteurs propres de la matrice des données, tandis que l'AE linéaire encode dans une base quelconque du même espace vectoriel. Les données encodées et décodées, ou approximées par les deux méthodes seront les mêmes, mais pas l'encodage.

On peut considérer l'ACP comme une factorisation de matrice : on représente une matrice de rang élevé comme le produit de deux matrices de plus faible dimension [a]. En régime linéaire, l'approximation obtenue est d'un rang limité par la dimension de l'encodage. Au contraire, lorsque l'auto-encodeur est non-linéaire, la matrice en sortie peut être de rang plus élevé que la dimension de l'encodage. L'information supplémentaire est enregistrée dans le décodeur de l'AE entraîné, le décodage nonlinéaire n'est plus simplement qu'un produit de matrice. L'AE est donc une généralisation nonlinéaire de l'ACP qui peut reproduire les performances de celle-ci

en régime linéaire et les dépasser dans des situations nonlinéaires, notamment sur les images (Hinton et Salakhutdinov, 2006).

a. Il existe d'autres problèmes qui peuvent se décrire comme des factorisations de matrice sous contrainte (factorisation non-négative (NMF), *clustering* par *k-means* par exemple) dont les solutions ne sont pas directement calculables. Certains de ces problèmes peuvent également être traités par auto-encodeurs.

4.1.2 Une première implémentation

On peut facilement créer un AE sous Keras. On peut par exemple créer un modèle d'encodeur, un modèle de décodeur, un modèle d'auto-encodeur qui enchaîne les deux précédents, puis entraîner ce troisième modèle. Il est également possible de construire directement un modèle d'AE, puis d'en extraire ensuite l'encodeur. On définit préalablement une fonction qui permet d'extraire la sortie d'une couche donnée d'un modèle :

```
from keras.layers import Input, Dense, Dropout, LeakyReLU
from keras.models import Model, Sequential
import pandas as pd

# La variable "encoded_layer" indique le rang de la couche d'encodage
# (en général la couche centrale)

def create_encoder(input_dim ,model ,encoded_layer):
    input_used= Input(shape=(input_dim,))
    encoded = model.layers[0](input_used)
    for i in range(1,encoded_layer):
        encoded = model.layers[i](encoded)

    return Model(input_used, encoded)
```

La fonction `AE` ci-dessous permet de générer et d'entraîner facilement un AE et son encodeur associé, à partir d'hyperparamètres choisis. Elle est notamment utile pour explorer l'espace des hyperparamètres. On extrait l'encodeur grâce à la fonction `create_encoder` précédemment définie :

```
def AE(training_set, validation_set, epochs, batch_size, optimizer, loss,
    ↪  dimensions, activations, encoded_layer, alphas = None, dropouts =
    ↪ None):
    if alphas is None:
        alphas = [0.0] * (len(dimensions) - 2)
```

```
    if dropouts is None:
        dropouts = [0.0] * (len(dimensions) - 2)
    # (On omet ici le code de gestion des exceptions)
    # Initialisation of the model
    model=Sequential()

    # First layer
    model.add(Dense(dimensions[1], input_dim= dimensions[0], activation=
        ↪ activations[0]))
    model.add(LeakyReLU(alpha=alphas[0]))
    model.add(Dropout(rate = dropouts[0]))

    # Middle layers
    for dim in range(1,len(dimensions) - 2):
        model.add(Dense(dimensions[dim+1], activation=activations[dim]))
        model.add(LeakyReLU(alpha=alphas[dim]))
        model.add(Dropout(rate = dropouts[dim]))

    # Last layer
    model.add(Dense(dimensions[-1], activation= activations[-1]))

    model.compile(optimizer=optimizer, loss=loss)


    # Training
    autoencoder = model.fit(training_set, training_set,
                            epochs=epochs,
                            batch_size=batch_size,
                            shuffle=True,
                            validation_data=(validation_set, validation_set)
                                ↪ )

    encoder = create_encoder(dimensions[0] , model ,encoded_layer)

    loss = autoencoder.history['loss']
    val_loss = autoencoder.history['val_loss']
    losses = pd.DataFrame({'Train_loss' : loss, 'Validation_loss' : val_
        ↪ loss})


    return losses, encoder, autoencoder, model
```

4.2 Auto-encoder l'enquête Emploi pour entraîner un prédicteur du chômage

Ce cas d'usage porte sur l'enquête Emploi. Il consiste dans un premier temps à encoder l'ensemble des observations de l'enquête Emploi en continu (EEC) en un vecteur de dimension réduite (100) à l'aide d'un auto-encodeur, puis d'utiliser cet encodage comme input d'un prédicteur RN entraîné à prédire le risque individuel de chômage au trimestre suivant [35].

4.2.1 Les données

Les données étudiées se limitent à la période 2003-2018. Le questionnaire a connu de nombreuses modifications et refontes au fil des années, en particulier en 2013 : une très grande partie des variables a au moins changé de nom, généralement associé à un changement de questionnaire ou de méthode de calcul et certaines ont été supprimées ou rajoutées. L'échantillon de l'enquête Emploi a continuellement augmenté en taille depuis 2003. En 2018, plus de 100 000 individus ont été interrogés chaque trimestre. Tout comme la taille des échantillons le nombre de variables étudiées a également augmenté au fil des années. En 2003, il y avait plus de 530 variables différentes, contre plus de 700 en 2018.

L'Enquête Emploi en continu est une étude de panels rotatifs. Chaque logement est interrogé pendant 6 trimestres consécutifs puis disparaît de l'échantillon. Ainsi, il est possible de construire une certaine continuité de la situation professionnelle des individus pendant un an et demi. Chaque ménage dispose d'un identifiant unique, et les individus au sein d'un même ménage sont différenciés par un numéro d'identification.

Ces caractéristiques de l'enquête impliquent un travail de sélection et de construction de la base d'entraînement. Les changements de variables et de champ géographique d'une année à l'autre peuvent être négligés, dans l'espoir que l'AE apprenne et intègre ces changements dans son encodage. Il faut cependant construire une base « cylindrique » contenant d'emblée toutes les variables (et toutes les modalités possibles) de l'ensemble de la période. La rupture de 2013 est cependant trop importante. On choisit donc d'entraîner un AE pour la période 2003-2012 et un pour la période 2013-2018.

4.2.2 Le prétraitement des données

La première étape consiste à mettre les données au format de vecteurs réels adapté à l'AE. Contrairement aux modèles traditionnels ou de *machine learning* exploitant un nombre limité de variables, on souhaite ici mobiliser la totalité du fichier : le traitement des variables doit être automatisé dans la mesure du possible. Cette transformation n'est pas évidente et relève de la construction de *features*. Il faut exclure les variables

35. Une première partie de ce travail a été menée dans le cadre d'un projet étudiant qui a fait l'objet d'un mémoire (Deltour, Faria, et Roudil-Valentin, 2020)

d'identification, traiter les valeurs manquantes et les dates. Les variables catégorielles doivent généralement être encodées en autant d'indicatrices qu'elles ont de modalité, sous peine de dénaturer l'information qu'elles contiennent. Les variables catégorielles ordonnées peuvent être traduites directement par le numéro d'ordre des modalités. On conserve dans ce cas l'information ordinale sans pour autant soumettre le RN à l'information cardinale (grâce à la non-linéarité de l'algorithme). Le prétraitement de chaque variable dépend donc du type qui lui est attribué, et le principal travail consiste alors à déterminer le type de chaque variable.

Les méta-données issues des dictionnaires des variables ou des fichiers eux-mêmes ne suffisent pas toujours à automatiser cette transformation : certaines variables sont mixtes, les variables catégorielles ordonnées ne sont pas toujours formellement distinguées et certaines sont ouvertes à interprétation (le niveau de diplôme, par exemple), les identifiants ne sont pas séparés des autres variables catégorielles, l'encodage des valeurs manquantes peut différer d'une variable à l'autre, etc. Ici, le classement des types des 700 variables est fait en partie à la main en s'appuyant sur un premier classement heuristique tiré du dictionnaire de variables. Toutes les variables sont intégrées, à l'exception des identifiants et autres variables catégorielles différentes pour chaque individu ou presque (outre les identifiants individuels, les identifiants de logements ou de grappes, n° Siren, communes, etc.) et des variables de date (qui demandent un prétraitement spécifique pour être réellement exploitables). Les variables catégorielles ordonnées sont directement traduites en variables numériques. Les autres variables catégorielles sont transformées en indicatrices, la non-réponse étant considérée comme une simple indicatrice supplémentaire. Pour les variables numériques, la non-réponse est traitée en deux temps : l'ajout d'une variable indicatrice (« âge manquant », par exemple), et l'imputation de la valeur manquante par la moyenne dans le sous-échantillon d'entraînement.

Les variables catégorielles de grande cardinalité posent un problème déjà exposé (voir partie 2.3.3) : traduites en indicatrices, elles génèrent des fichiers trop larges (jusqu'à 11000 colonnes). Au-delà d'un certain nombre de modalités, d'un certain seuil de pré-encodage s qu'il faut déterminer, ces variables sont donc pré-encodées. La méthode présentée en 2.3.3 étant un peu coûteuse à systématiser, nous les encodons ici de manière très simple. Chaque modalité étant remplacée par le vecteur des moyennes, dans le groupe des personnes ayant cette modalité, de 8 variables. Ces 8 variables sont choisies car elles sont systématiquement renseignées dans l'enquête : sexe, âge, nombre d'enfants de moins de 18 ans, en couple ou non, nationalité d'un des 15 premiers pays de l'UE, zone d'habitation rurale ou urbaine, nombre de chômeurs dans le foyer, inclusion dans le halo du chômage ou non).

Concrètement, la modalité 21 de la variable CSP (qui représente les artisans) est encodée en un vecteur contenant : l'âge moyen des artisans, la part de femmes parmi les artisans, etc. Ce pré-traitement permet de réduire considérablement la taille de la base de données traitée, pour une dimension de l'ordre de 3000000×3000. Ce processus entraîne une perte d'information non négligeable, et constitue ainsi une piste d'amélioration, soit en utilisant de plus grandes capacités de calcul et de mémoire, soit avec un pré-encodage plus souple, par exemple via l'algorithme *word2vec* déjà présenté.

La base complète a ensuite été divisée en trois sous-échantillons : entraînement, validation et test. Chaque individu étant observé 6 fois dans l'enquête Emploi, on s'assure que les 6 interrogations d'un même individu sont présentes dans un seul des sous-échantillons, afin de ne pas biaiser l'apprentissage (pour un même individu, de nombreuses variables seront identiques lors des différentes interrogations). Dans la perspective de tester les performances de l'AE sur un prédicteur entraîné en aval, il faut conserver la même partition des échantillons pour les deux problèmes. Le prédicteur sera ensuite entraîné à prédire le risque de chômage individuel au trimestre $N + 1$, or ce problème demande un échantillon test suffisamment grand, car la variable à prédire est dichotomique et très déséquilibrée. En conséquence, l'échantillon test de l'AE, identique, est lui aussi relativement grand. Le modèle final est ainsi entraîné sur 50% de l'échantillon, 10% étant réservés à la validation et 40% au test.

Enfin, une fois les trois bases créées, l'ensemble des variables est normalisé afin de permettre une convergence plus rapide de l'algorithme. Concrètement il s'agit, pour chaque observation, de lui soustraire la valeur minimale de la variable observée dans le sous-échantillon et de la rapporter à la distance entre la valeur maximale et minimale. Ainsi, toutes les données sont comprises entre 0 et 1. La normalisation a le défaut d'être très sensible aux valeurs extrêmes et de diminuer la variance de certaines variables beaucoup plus que d'autres, et donc en pratique de modifier leur poids dans la fonction de perte. On peut parfois lui préférer pour cette raison la standardisation, c'est-à-dire de centrer et réduire chaque variable en la divisant par sa variance : toutes les variances sont alors ramenées à 1. La normalisation a cependant été choisie pour plusieurs raisons. D'une part, dans un contexte où les variables sont de natures très diverses, liées entre elles par des filtres de questionnaire, et subissant des prétraitements très variés, il est en partie illusoire de chercher à pondérer également chaque variable en égalisant les variances. D'autre part, la plupart des variables n'ont pas de valeurs extrêmes problématiques. Enfin, la normalisation est bornée. Dans le cas d'un AE, où ces variables servent de cible à la couche de sortie, cela a l'avantage d'être compatible avec l'utilisation d'une fonction sigmoïde en couche de sortie (la sigmoïde étant elle-même bornée), ce qui, en pratique, facilite l'apprentissage.

4.2.3 Optimisation de l'auto-encodeur

La fonction de perte est l'erreur quadratique moyenne, qui s'adapte bien aux différents types d'inputs, après normalisation. Sur les machines utilisées [36], le temps d'entraînement du modèle est d'environ 15 minutes par époque. L'entraînement complet nécessite au moins une cinquantaine d'époques pour obtenir une bonne convergence. Au vu du temps du calcul, il serait trop long d'explorer systématiquement l'espace des hyperparamètres ou d'utiliser systématiquement la validation croisée pour choisir les hyperparamètres. Le choix des hyperparamètres a donc été réalisé via des explorations *à la main*, sur des sous-échantillons, et à partir de recommandations issues de la littérature.

L'auto-encodeur est entraîné en utilisant l'optimisateur RMSprop (voir section 1.2.3, Les

36. Une machine virtuelle de l'ENSAE, sans parallélisation.

TABLEAU 7 – **Structure de l'auto-encodeur**

Nom de la couche	Fonction d'activation	Dimension
Couche d'entrée (EE)		2920* / 3362**
Couche cachée encodage	Leaky ReLu ($\alpha = 0.3$)	500
Couche centrale	Leaky ReLu ($\alpha = 0.3$)	100 = d
Couche cachée décodage	Leaky ReLu ($\alpha = 0.3$)	500
Couche de sortie (EE)	Sigmoïde	2920* / 3362**

* : sur la période 2003-2012
** : sur la période 2013-2018.

algorithmes de descente de gradient), qui ici converge plus vite que les autres optimisateurs de type *Adaptative Moment Estimation* (Adam, AdaMax, Nadam, etc.). Chaque couche intermédiaire est composée d'une fonction d'activation Leaky Relu avec un coefficient α fixé à 0.3, valeur par défaut de Keras. La couche de sortie du réseau est activée par une fonction sigmoïde. Aucune technique de régularisation n'est mise en place suite à l'absence de sur-apprentissage.

Le tableau 7 résume la structure choisie pour l'auto-encodeur.

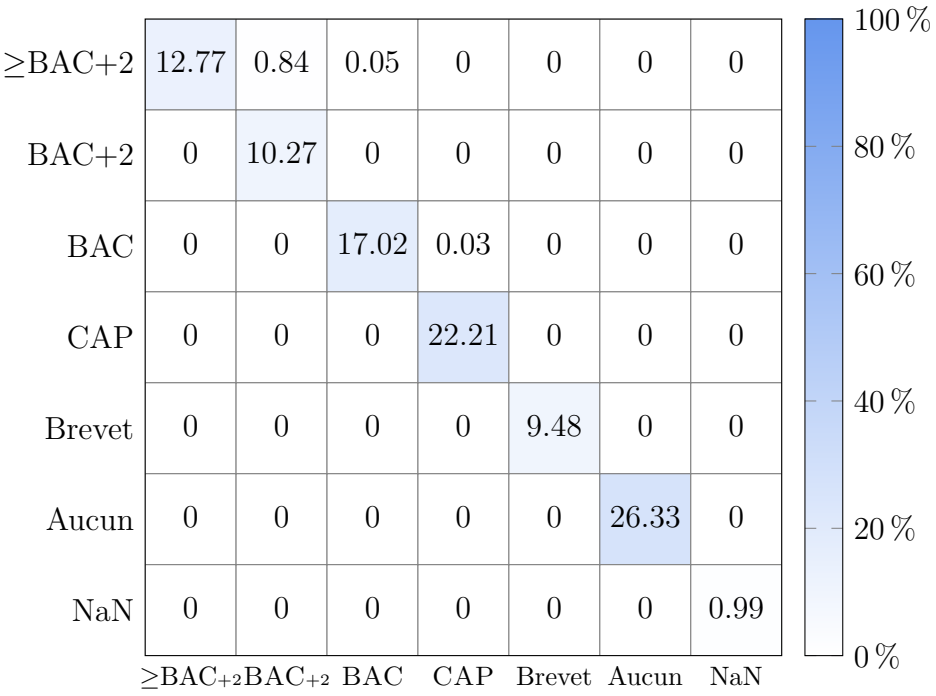
4.3 Résultats et perspectives

4.3.1 Précision de l'auto-encodeur

L'approche la plus simple pour évaluer la performance d'un auto-encodeur est de comparer, sur l'échantillon test, la base de données initiales à celle prédite par l'AE après réduction de la dimension. Pour cela des représentations classiques comme le score d'*accuracy* (la proportion des prédictions qui s'avèrent exactes), les matrices de confusion ou des nuages de points permettent tous d'étudier la similarité entre l'input et l'output de l'auto-encodeur.

La figure 35 représente la matrice de confusion de la variable DDIPL (niveau de diplôme). Cette figure montre que l'auto-encodeur réussit très bien à reconstruire l'input qui lui a été donné. Pour la variable DDIPL, le score d'*accuracy* est d'environ 99%. Lorsque l'AE se trompe, plus souvent lorsque le niveau d'étude observé est Bac+2 ou plus, la prédiction certes fausse reste proche de la valeur observée, ce qui témoigne d'une certaine cohérence dans les prédictions de l'auto-encodeur. L'*accuracy* est supérieure pour d'autres variables importantes dans l'enquête : $99,8\%$ pour le sexe ou le statut d'immigré par exemple, 100% pour les grandes catégories d'âge ou le statut d'emploi, cette dernière variable (ACTEU) étant utilisée pour calculer la principale grandeur d'intérêt de l'EEC, le taux de chômage.

Pour des variables quantitatives comme l'âge ou le salaire, on met en regard les variables de sortie de l'auto-encodeur et celles d'entrée, après inversion de la normalisation (voir la figure 36). L'âge est bien prédit par l'auto-encodeur, avec un R2 de 0,98. L'information



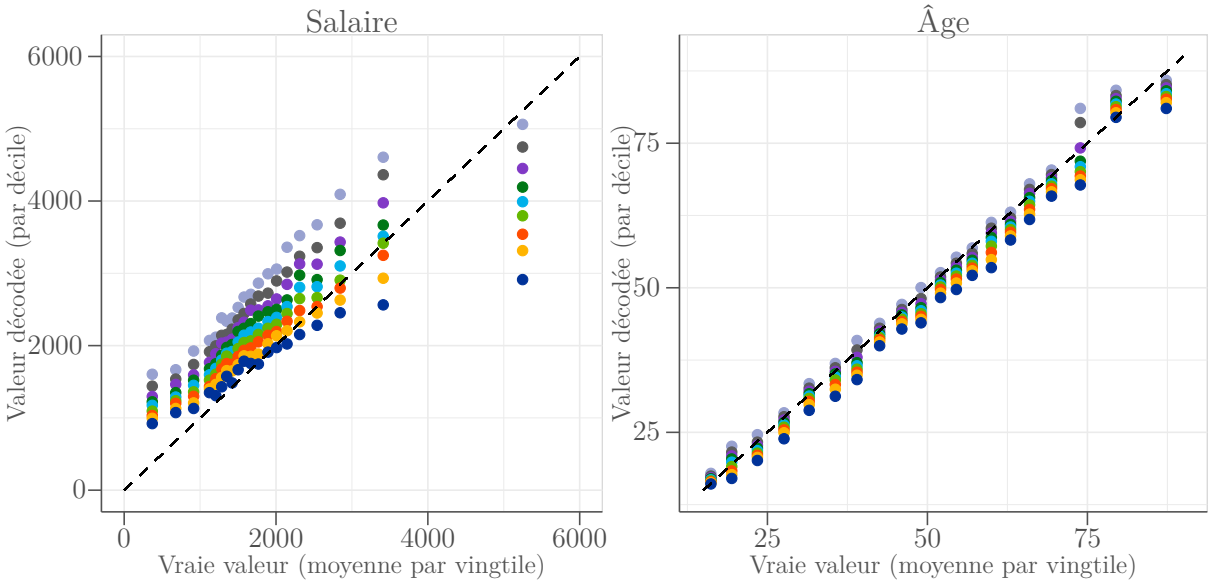
Graphique 35 – Matrice de confusion de DDIPL

Notes : Répartition selon le niveau de diplôme observé (variable DDIPL, en ordonnée) et le niveau de diplôme prédit (en abscisse), en pourcentage. 12.77% des individus de la base test ont un niveau de diplôme BAC+2 et sont effectivement catégorisés comme tels. 0.84% des individus sont catégorisés BAC+2 alors que leur niveau d'étude réel est ≥ BAC+2.

sur l'âge est présente dans de nombreuses variables de l'enquête Emploi, comme l'année de naissance ou les différentes catégories d'âge. Cette information répétée en fait probablement une variable facile à prédire. On observe cependant des effets de seuil, le plus important se situant en haut de la distribution : l'auto-encodeur ne prédit jamais de valeurs peu probables, dans ce cas les âges supérieurs à 90 ans.

Pour le salaire, les valeurs prédites, pour une vraie valeur donnée, sont bien plus étalées et tendent à se rapprocher de la moyenne. Il semble également y avoir des biais systématiques : les valeurs décodées les plus faibles sont presque systématiquement inférieures à leur vraie valeur.. La prédiction est sans doute plus difficile. Il s'agit d'une variable avec beaucoup de non-réponses, peu répétée dans l'enquête, mais les résultats restent corrects. Comme pour l'âge, un effet de seuil est observable.

La table 8 précise, pour ces deux mêmes variables, les limites de la reconstruction des variables continues par l'AE. Dans les deux cas, la variance des valeurs prédites sous-estime la vraie variance, le même phénomène de régression vers la moyenne que dans une régression linéaire. Les valeurs sont également biaisées, légèrement à la baisse pour l'âge, fortement à la hausse pour le salaire. Cela impacte évidemment les grandeurs d'intérêt calculées à partir des valeurs prédites (à titre d'exemple, la part des plus de 65 ans pour l'âge, et le rapport du neuvième décile au premier décile pour le salaire, un indicateur des inégalités souvent utilisé)



Graphique 36 – Performance de l'AE pour le salaire et l'âge

Notes : La relation entre les valeurs prédites et les vraies valeurs pour les variables de salaire (colonne de gauche) et l'âge (colonne de droite) sur l'échantillon test. Pour conserver la confidentialité des données, on représente (par des couleurs différentes) les déciles des valeurs prédites en fonctions des moyennes par vingtile des vraies valeurs.

TABLEAU 8 – **Distributions observées et prédites sur l'échantillon test pour l'âge et le salaire**

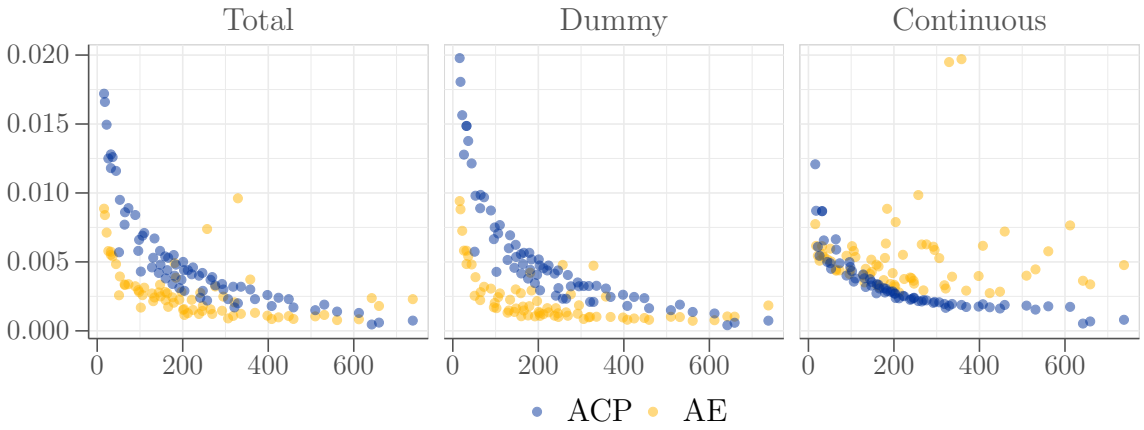
	Salaire observé	Salaire prédit	Âge observé	Âge prédit
Moyenne	1863	2300	49,7	48,5
Variance	1 208 062	758 741	395,4	385,4
Médiane	1616	2154,5	50	48
Part des + 65 ans			23.8%	22.9%
Rapport interdécile	3,75	2,58		

4.3.2 Comparaison avec l'ACP selon la dimension d'encodage et le seuil de pré-encodage

Une autre approche pour juger de l'AE est de comparer ses résultats à ceux de l'analyse en composantes principales (ACP). La puissance de calcul requise pour estimer et effectuer une ACP est bien plus faible que celle nécessaire pour un AE, à taille d'échantillon donné. L'estimation est bien plus rapide.

On compare les erreurs quadratiques moyennes évaluées sur l'échantillon de test des deux méthodes (figure 37 page suivante). On distingue ensuite l'erreur moyenne sur les variables qualitatives (panel 2) et les variables continues (panel 3). On compare les performances en faisant varier la dimension de l'encodage (d) pour l'AE et le nombre d'axes pour l'ACP. Pour une dimension d'encodage (ou un nombre d'axes) de 200, l'erreur quadratique moyenne de l'AE est d'environ $0,005$ contre environ $0,0025$ pour l'ACP. Plus la réduction de dimension (d en dessous de 150) est importante plus l'AE est performant par rapport à l'ACP (figure 37 page suivante). A ces niveaux, les relations non linéaires (captées par l'AE mais pas l'ACP) semblent importantes pour assurer la performance. Pour des niveaux d'encodage au dessus de 600, l'ACP est plus performante que l'AE. Par type de variables, l'AE performe mieux que l'ACP pour les variables catégorielles (pour des réductions de dimension importantes) mais fait plutôt jeu égal ou inférieur lorsqu'il s'agit de variables quantitatives. Les variables qualitatives sont sans doute difficiles à résumer de manière linéaire pour une ACP, et il est également possible que l'entraînement de l'AE le spécialise dans l'encodage des variables les plus « rentables » : les variables continues sont plus rares dans notre cas d'usage et la normalisation choisie (min-max) induit une variance plus faible pour les variables continues que pour les indicatrices, ce qui réduit d'autant leur impact sur la perte quadratique. Autre facteur : dans le cadre de l'enquête Emploi, les variables qualitatives peuvent aussi souvent être des variables filtres, qui orientent le déroulé du questionnaire et sont donc davantage corrélées aux autres variables.

Cependant dans notre cas un autre hyperparamètre impacte les résultats : le seuil de pré-encodage s, c'est-à-dire le nombre de modalités à partir duquel on pré-encode les variables ayant des nomenclatures très détaillées, qui impacte directement la dimension de l'input. Cet hyperparamètre affecte ainsi directement le temps d'entraînement et la mémoire (RAM) nécessaire à cet entraînement.



GRAPHIQUE **37 – Comparaison des pertes de l'AE et de l'ACP**

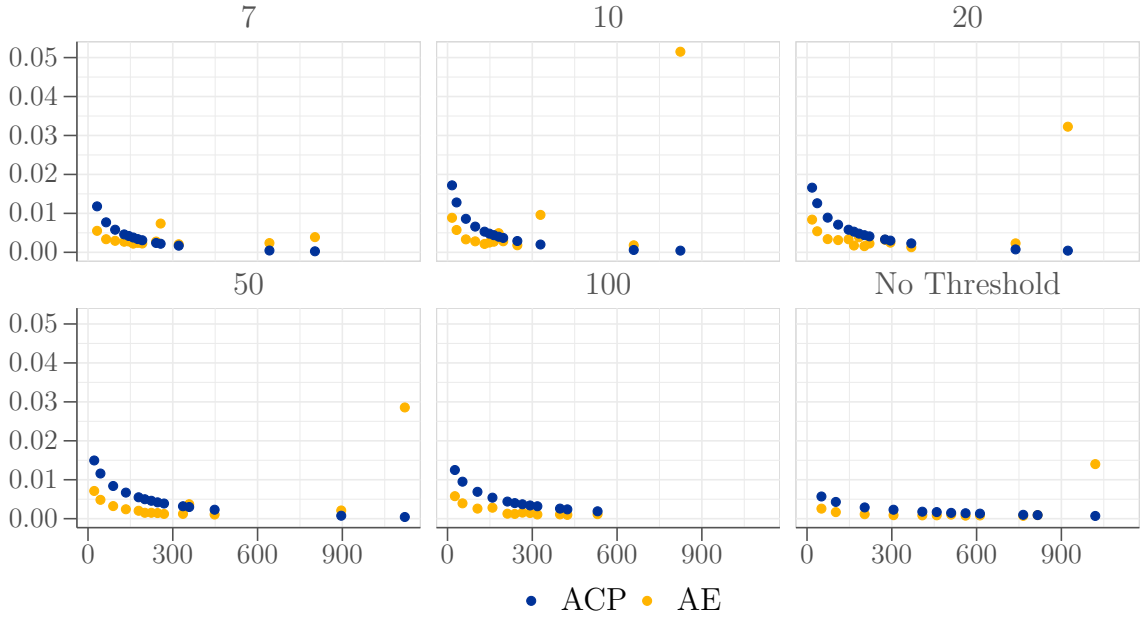
Notes : Perte (erreur quadratique moyenne) en fonction de la dimension d'encodage (en abscisse, correspondant au nombre d'axes pour l'ACP), pour l'ensemble des variables ("Total"), et en distinguant les variables qualitatives traduites en indicatrices ("Dummy") et les variables continues ("Continuous").

Pour analyser la dimension d'encodage on ne s'intéresse donc non pas seulement à la dimension absolue mais également à la dimension relative par rapport à la dimension de l'input.

La figure 38 montre que la dimension optimale qui minimise la perte dépend du seuil fixé. Plus le seuil est élevé (ie. moins on pré-encode et plus on a d'indicatrices), plus la perte de l'auto-encodeur est faible, et continue à diminuer à mesure qu'on augmente la dimension de la couche d'encodage. Un seuil trop faible fait perdre de l'information. On observe ici un arbitrage coût/efficacité, entre temps de calcul/RAM requise et performance de l'auto-encodeur entraîné. En raison des contraintes computationnelles, un seuil relativement bas de $s = 10$ a été retenu pour ce travail. Autrement dit, toutes les variables qualitatives de 11 modalités et plus ont été pré-encodées en dimension 8. La dimension d'encodage retenue pour la suite est $d = 100$.

4.3.3 Prédicteur de risque de chômage

L'un des usages possible de l'encodage par AE est d'utiliser directement le fichier encodé pour entraîner ensuite, plus rapidement et de manière plus stable, d'autres modèles. L'AE permet de mutualiser une étape de pré-entraînement gourmande en calcul. Nous testons cette démarche en entraînant un RN prédicteur du risque individuel de chômage qui reçoit en input la base encodée par l'AE (décrit dans la table 9). Il est également comparé à



GRAPHIQUE **38 – Comparaison de la perte en fonction du seuil de pré-encodage**

Notes : Erreur quadratique moyenne en fonction de la dimension d'encodage (en abscisse) et du seuil de pré-encodage s.

deux autres modèles : un modèle plus traditionnel, un logit estimé sur 12 variables de l'enquête Emploi [37], ainsi qu'un RN entraîné sur ces mêmes variables.

On cherche à prédire la situation d'emploi (en emploi, au chômage, inactif) au trimestre suivant pour chaque répondant. Pour cela, on entraîne le prédicteur du risque de chômage à partir de la base de données encodée de dimension 100 par l'AE, en incluant tous les trimestres. Pour prédire le statut d'emploi d'un individu de l'échantillon test au trimestre $t+1$, le RN reçoit en input la ligne correspondant à cet individu au trimestre t, mais il a été entraîné sur d'autres individus présents en $t+1$ et t. En ce sens, il connaît l'avenir (de l'échantillon d'entraînement). Il ne s'agit donc pas de prévision conjoncturelle, mais d'une démarche plus proche de l'imputation.

Un tel prédicteur pourrait avoir plusieurs applications. Il peut servir à imputer la situation d'emploi des non-répondants, par exemple dans un souci méthodologique pour évaluer les effets sur le taux de chômage de la non-réponse et de sa correction. On peut élargir virtuellement l'échantillon avec des prédictions sur une septième ou une huitième interrogation, ou à rebours pour les trimestres précédant l'enquête, ce qui permettrait notamment d'exploiter systématiquement l'information contenue dans le

37. Âge, sexe, niveau de diplôme, statut d'emploi au trimestre t et situation un an avant, statut d'immigration et nationalité, situation de couple, type de ménage, nombre d'enfants mineurs, type d'agglomération et adresse en zone urbaine sensible

TABLEAU **9 – Structure du prédicteur du risque de chômage**

Nom de la couche	Fonction d'activation ou fonction de perte	Dimension
Couche d'entrée		100
Couche cachée	ReLu	300
	Dropout $0,4$	
Couche cachée	ReLu	500
	Dropout $0,5$	
Couche cachée	Softmax	300
	Dropout $0,4$	
Couche de sortie	entropie croisée catégorielle	3

calendrier rétrospectif demandé aux enquêtés. Ce calendrier étant mensuel et rempli à chaque interrogation, on peut aussi envisager de prédire un risque de chômage mensuel pour chaque individu, dans la perspective, par exemple, de proposer une estimation du taux de chômage mensuel et non pas seulement trimestriel. Les probabilités prédites de situation d'emploi peuvent aussi être exploitées dans des analyses économétriques, par exemple pour contrôler l'effet de sélection dans une étude ne portant que sur les personnes en emploi. Nous n'explorons pas davantage ici ces applications potentielles.

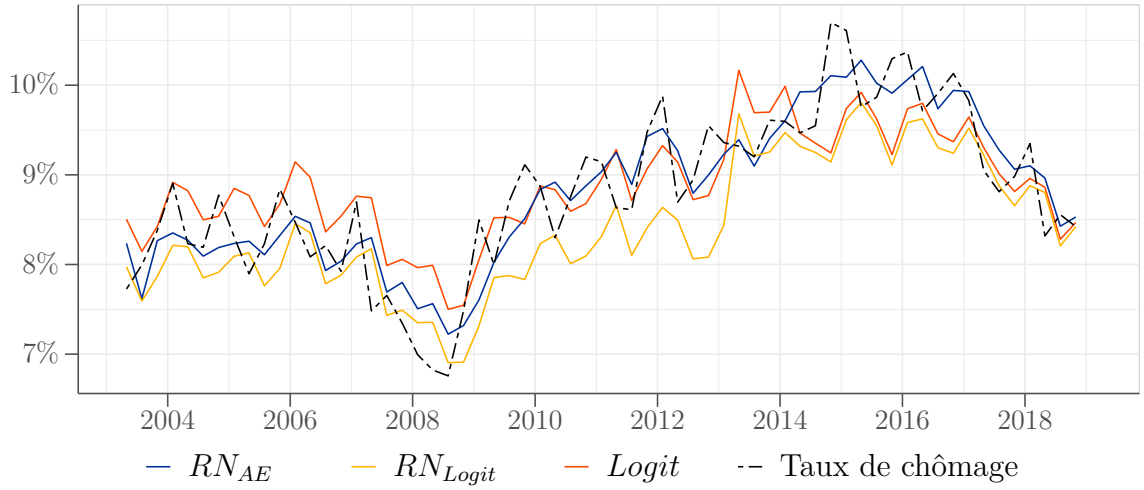
L'entraînement du RN utilisant la base encodée est stable et efficace. Pour estimer ses performances, et comparer les modèles, il faut cependant une métrique adaptée. Les modèles produisent une probabilité (ici, le risque de chômage prédit), que l'on confronte à une réalisation (chômage ou non) pour chaque individu. On peut passer de l'une à l'autre en choisissant, par exemple, un seuil de probabilité, mais ce choix reste en partie arbitraire. La méthode usuelle dans cette situation est d'utiliser une courbe ROC (pour *receiver operating characteristic*) qui représente le taux de vrais positifs (en ordonnée) en fonction du taux de faux positifs (en abscisse) lorsqu'on fait varier ce seuil pour prédire une alternative binaire. La courbe est croissante entre les points $(0,0)$ et $(1,1)$. Si le prédicteur est le hasard, la courbe rejoint la diagonale. Plus la courbe s'approche du point $(1,0)$ (100% de vrais positifs, et 0% de faux positifs), meilleur est le prédicteur. L'AUC (*area under the curve*, aire sous la courbe) permet de synthétiser cette performance et de comparer des modèles. Elle est comprise entre 0 (le hasard) et 1 (le prédicteur parfait). Une valeur de l'AUC inférieure à 0.5 signifie que le modèle est moins performant que le hasard pur.

La table 10 présente cette métrique pour l'alternative chômeur *vs* non-chômeur. Le score AUC obtenu sur la base pré-encodée par l'auto-encodeur est le plus élevé. C'est le cas également pour les deux autres comparaisons binaires possibles (actif *vs* inactif et en emploi *vs* hors emploi). Cette meilleure performance prédictive reste vérifiée sur des sous-populations diverses, par sexe, niveau de diplôme, région, etc. Cela illustre un intérêt de l'encodage par l'AE en première étape : le prédicteur a les moyens d'établir des relations réalistes entre le risque de chômage et l'ensemble des variables de l'enquête, à la différence des deux autres modèles reposant sur un ensemble pré-determiné de variables explicatives.

Tableau **10 – Performances des prédicteurs du chômage**

Modèle	Score AUC
RN avec input encodé par AE	0, 96
Logit	0, 91
RN avec input identique au logit	0, 92

Une autre métrique possible est de comparer au taux de chômage le risque moyen de chômage, c'est-à-dire la moyenne des probabilités de chômage prédites, pour des populations ou sur des périodes données. L'une des caractéristiques de l'entropie croisée (voir 1.2.2) comme fonction de perte est qu'elle est minimale lorsque la probabilité prédite est la vraie probabilité. Le risque moyen de chômage estimé à partir du prédicteur RN est très proche du taux de chômage, y compris sur de petits domaines, ce qui permet par exemple de construire une courbe du risque de chômage trimestriel (figure 39).



Graphique **39 – Comparaison du taux de chômage prédit par les modèles**

Notes : Risque moyen de chômage prédit par trimestre sur l'échantillon test par chacun des trois modèles, et taux de chômage observé (hors pondération et non corrigé des variations saisonnières)

Ainsi ce premier traitement effectué par l'AE conserve suffisamment d'information pour permettre au prédicteur RN du risque de chômage de capturer efficacement la chronique des fluctuations conjoncturelles du taux de chômage. Estimés sur un nombre plus réduit de variables, les deux autres modèles sont moins précis. Le logit semble dominé par une règle simple consistant à prédire au trimestre $t + 1$ le même statut d'emploi qu'au trimestre t, tandis que le RN entraîné sur les mêmes variables que le logit souffre visiblement d'un biais à la baisse. Ce défaut est sans doute à relier aux difficultés d'entraînement de ce dernier modèle, dont les performances sont beaucoup plus instables que le RN entraîné sur la base encodée.

Encadré 2: Un AE qui manque de lignes sur données macro-économiques

Peut-on exploiter la méthode présentée dans un autre contexte, où les fichiers sont de moindre taille ? Nous avons essayé sur les données de prix de la Banque Centrale Européenne. L'idée est de reproduire la méthodologie suivie dans le cas d'usage précédent en l'appliquant à des séries temporelles. L'intégralité du code source, écrit en python, est disponible dans le *repository* associé à ce document de travail et au Chapitre 4.
Les données correspondent à 482 séries mensuelles d'inflation de différents biens, allant de 1995 à 2021, pour 28 pays d'Europe (Union européenne et Royaume-Uni). Ces données publiques sont directement récupérées via l'API de l'ECB Statistical Data Warehouse.

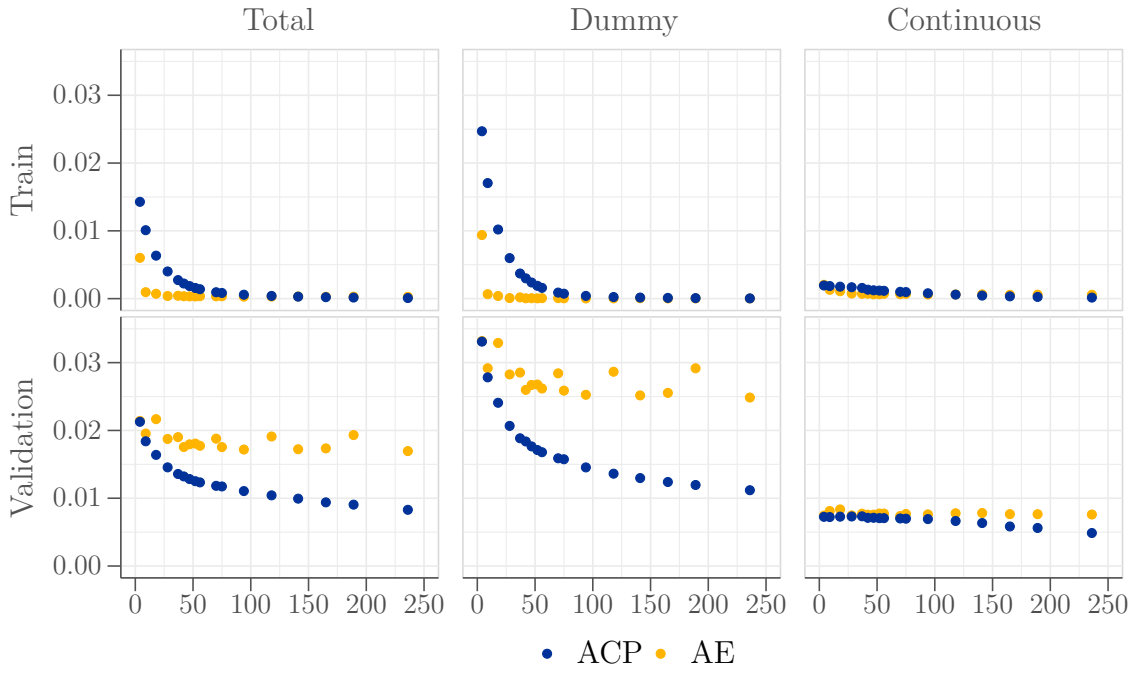
```
Results = ecb.getECBData(ListSeries)
```

De même que précédemment, les valeurs manquantes (toutes les séries ne sont pas disponibles sur la totalité de période étudiée) sont imputées par les taux d'inflation moyens, et des indicatrices de valeurs manquantes sont ajoutées. À la différence de l'enquête Emploi cette base de données contient exclusivement des données quantitatives (sauf le pays). Chaque observation est rapportée à un identifiant unique correspondant au couple pays / date. À la suite de ce simple pré-traitement la base de données est de dimension $(11000, 950)$, ce qui est considérablement plus faible qu'avec l'enquête Emploi.

```
# On définit les variables qualitatives et quantitatives de
    ↪ notre dataset
var_quali = ["COUNTRY"]
var_index =["COUNTRY_IDX","OBS_DATE"]
var_quanti = [x for x in Dataset.columns if x not in var_index
    ↪ if x not in var_quali]

# On définit nos index pour notre panel
Dataset.set_index(var_index, inplace=True)
```

La division en échantillons de validation, test et d'apprentissage est établie de sorte que toutes les observations d'un pays se trouvent dans un même échantillon, de manière analogue à ce qui a été fait pour les individus avec l'enquête Emploi [a]. Le modèle est entraîné sur 70% de l'échantillon, 10% étant réservés à la validation et 20% au test. Ces trois bases de données sont normalisées comme ci-dessus (on soustrait la valeur minimale, puis on divise par le maximum, pour obtenir des valeurs dans $[0, 1]$), afin d'être compatibles avec la fonction sigmoïde en couche de sortie de l'auto-encodeur.
Sur ce type de données, l'auto-encodeur semble moins performant avec une perte plus de 10 fois supérieure sur l'échantillon de validation que sur celui d'entraînement, et

plus élevée que celle d'une simple ACP (figure 40). La meilleure performance de l'AE sur l'échantillon d'entraînement témoigne d'un surapprentissage. Il est probable que cette sous-performance soit due au faible nombre d'observations pour un problème de *machine learning*. En revanche, la faiblesse relative de l'AE entraîné sur l'Enquête Emploi pour restituer les variables continues ne se vérifie pas ici, où les variables continues sont dominantes (les seules indicatrices étant celles de valeur manquante).



GRAPHIQUE 40 – **Erreur quadratique moyenne en fonction de la dimension d'encodage**

a. Ce choix est discutable au vu de la forte hétérogénéité des taux d'inflation des pays de l'Union européenne sur la période 1995-2021.

4.3.4 Perspectives

Sur des données typiques de la statistique publique, les cas d'usage présentés montrent que les auto-encodeurs peuvent être bien entraînés, à condition que le nombre d'observation soit suffisamment élevé. Ils encodent et restituent efficacement l'information. Ils peuvent donc servir d'étape de pré-entraînement à d'autres utilisations de RN, en particulier lorsque plusieurs prédicteurs différents doivent être entraînés sur les mêmes données. L'AE permet

alors d'envisager une industrialisation plus facile des RN, en mutualisant la préparation des données pour chaque source et en fournissant un input dense en information, stabilisant l'entraînement de deuxième étape. Ces usages répétés d'un même AE en première étape pourraient concerner en particulier des imputations multiples dans un fichier, ou de la détection et de la correction d'erreur.

D'autres usages sont envisageables en statistique publique, mais il y a peu d'exemples d'utilisation d'AE par les statisticiens. La perspective de générer des données de synthèse préservant la confidentialité, expérimentée notamment à l'*Office for National Statistics* britannique (Kaloskampis, 2019) est particulièrement intéressante et a constitué la motivation première de ce travail. Cet usage génératif est présenté plus en détail en conclusion.

A mi-chemin entre les usages d'imputation et la génération de données de synthèse, il est aussi envisageable d'exploiter des appariements ponctuels de fichiers pour générer des appariements synthétiques, par exemple à des fins de micro-simulation. Dans ce cas, si l'on dispose de deux sources A et B et d'un corpus réduit d'observations appariées entre deux sources, on entraîne deux AE sur chaque base, et on exploite le corpus apparié pour entraîner un traducteur entre les données encodées par chaque AE, à la manière de la traduction automatique. On peut alors construire des observations mixtes, dont une partie est observée dans un fichier tandis que l'autre est générée par le traducteur. Nous évoquons aussi en conclusion la perspective d'adapter à cette démarche des algorithmes de traduction capable de s'entraîner en l'absence de corpus déjà traduit.

Discussion et conclusion

Les cas d'usage présentés ici montrent que les RN peuvent trouver leur place dans la boîte à outils du statisticien. Il est possible de les mettre en oeuvre avec les moyens disponibles à l'Insee, et de les entraîner efficacement sur des données typiques de la statistique publique. Les RN sont des prédicteurs performants, ils permettent d'exploiter de nouveaux types de données, comme les images, et de mieux utiliser la richesse des données de haute dimension.

Le croisement des RN et des usages statistiques demande un travail d'adaptation, des concepts d'une part pour comprendre la logique du *machine learning* et les implications statistiques des choix algorithmiques, et des données d'autre part, qui ont besoin d'un travail préparatoire important et spécifique pour être transformées en inputs de RN.

Ce document reste cependant introductif et les réseaux présentés sont relativement simples. Nous évoquons donc rapidement en conclusion plusieurs développements possibles qui ne sont pas traités dans les cas d'usage précédents. Le premier est le sujet général de l'éthique de l'intelligence artificielle. Pour la statistique publique, cela se traduit principalement par deux préoccupations : celle de la confidentialité et celle de la représentativité. Le second développement est celui de l'amélioration des performances, qui devient critique dès lors que les RN sont exploités à plus grande échelle. Enfin, le troisième développement est celui des autres types et usages de RN que nous n'avons pas explorés ici : principalement les RN récurrents permettant d'exploiter des données présentées en série (en particulier du texte), les RN génératifs pouvant produire des données de synthèse, et l'utilisation des RN en économétrie dans un but d'inférence causale.

Confidentialité

La confidentialité est un enjeu majeur de l'utilisation des RN, et l'auto-complétion est un bon exemple. Les programmes d'auto-complétion qui proposent des mots ou des phrases lors de l'écriture de messages ou d'emails sont construits avec des RN entraînés sur des données privées. Sans précaution, ces programmes peuvent facilement dévoiler des secrets issus de l'échantillon d'entraînement, par exemple en complétant une phrase du type « mon numéro de carte bleue est... » avec un numéro réel. Ce risque n'est pas seulement théorique, mais démontrable en pratique (Carlini, Liu, Erlingsson, Kos, et Song, 2019). Ce risque existe même quand le RN ne présente pas de signes de surapprentissage et même lorsque le modèle est de taille modeste par rapport aux données.

En raison de leur grand nombre de paramètres, les RN sont des modèles qui peuvent mémoriser beaucoup plus d'informations issues de l'échantillon d'entraînement que la plupart des autres modèles (de *machine learning* ou non). Cette caractéristique les rend vulnérables à des attaques de confidentialité qui peuvent prendre plusieurs formes. Un adversaire mal intentionné ou curieux peut tenter de déterminer l'appartenance d'une observation individuelle x_i à l'échantillon d'entraînement. Il peut essayer de reconstituer les observations d'entraînement ou certaines d'entre elles (en particulier les plus atypiques).

Et, notamment dans un contexte industriel de concurrence sur les modèles, l'attaquant peut également s'intéresser davantage à voler le modèle que les données, en essayant de reconstituer les paramètres entraînés, les hyperparamètres de l'entraînement, ou encore certains des enseignements de valeur que le modèle a tirés de ses données d'entraînement.

Ces attaques peuvent se fonder sur différents accès : l'attaquant peut avoir accès au modèle entier (y compris ses paramètres et hyperparamètres) et / ou à une partie de l'échantillon d'entraînement, voire influencer directement le processus d'entraînement (par exemple en ajoutant des données à l'échantillon). Il peut aussi n'avoir accès au modèle que sous forme de boîte noire dont il contrôle les inputs et observe les outputs. Enfin, il peut n'avoir accès qu'aux outputs d'un échantillon d'inputs, qu'il connaît ou pas. Dans tous les cas, des attaques de confidentialité (plus ou moins puissantes) sont envisageables. On doit donc se poser la question de la confidentialité dès lors que les données d'entraînement sont confidentielles, la précaution la plus simple et la plus restrictive consistant à considérer les modèles et prédictions comme ayant le même niveau de confidentialité que les données d'entraînement. Dans un contexte d'étude, on peut choisir de ne pas partager les modèles entraînés et les prédictions. Dans d'autres situations, comme l'imputation, on souhaite au contraire partager au moins les prédictions. Les méthodes traditionnelles de la statistique publique pour préserver la confidentialité ont des limites connues, qui deviennent d'autant plus problématiques que les modèles sont plus complexes.

Les solutions retenues aujourd'hui en *machine learning* reposent souvent sur le principe de confidentialité différentielle (*differential privacy*), une formalisation développée en particulier par Cynthia Dwork depuis 2006 (Dwork, McSherry, Nissim, et Smith (2006), Dwork (2008)). Une bonne introduction générale est donnée dans les premiers chapitres de Dwork, Roth, et al. (2014). On suppose qu'une base de données confidentielle est utilisée pour publier des informations statistiques (ou répondre à des requêtes de façon dynamique). On souhaite minimiser le risque qu'un individu (ou une firme, une famille, etc.) présent dans la base puisse subir des conséquences du fait de cette présence, tout en préservant autant que possible la précision des informations ou des réponses aux requêtes. Il faut alors introduire de l'aléa, de telle sorte qu'un attaquant, même doté d'informations auxiliaires, ne puisse rien apprendre ou presque de nouveau sur aucun individu spécifique présent dans la base à l'aide des statistiques publiées ou des réponses aux requêtes, qu'il n'aurait pas pu apprendre également à partir d'une base où cet individu aurait été absent. Il existe nécessairement un arbitrage entre la protection de la confidentialité et la précision des informations publiées, qui dépend du niveau d'aléa introduit.

Plus formellement, l'idée est qu'un algorithme aléatoire entraîné sur un échantillon quelconque donne des résultats « proches » du même algorithme entraîné sur tout échantillon voisin, ne différant que par une seule observation. Cette proximité est définie mathématiquement comme un majorant de la différence entre les probabilités que chaque entraînement produise le même *output*. Avec A l'algorithme, pour tout D_1 et D_2 échantillons ne différant que d'un élément, on veut que pour tout ensemble S d'*outputs* de l'algorithme, inclut dans l'image de A $Im(A)$ (c'est-à-dire l'ensemble des sorties possibles) :

$$\mathbb{P}(A(D_1) \in S) \leq exp(\epsilon) * \mathbb{P}(A(D_2) \in S)$$

Le paramètre ϵ constitue le « budget » de confidentialité que l'on choisit. Plus il est proche de 0, plus la confidentialité est protégée [38]. Pour le respecter, la méthode consiste à ajouter un bruit aléatoire dans l'algorithme, calibré en fonction de la sensibilité des outputs de l'algorithme aux observations en entrée. La confidentialité différentielle est un formalisme : sa mise en application demande un travail spécifique pour chaque type d'algorithme. Cette méthode a été adaptée aux RN en 2016 (Abadi, Chu, Goodfellow, McMahan, Mironov, Talwar, et Zhang, 2016) : elle consiste alors à modifier l'optimisateur (par exemple l'algorithme de descente de gradient stochastique) en majorant la norme du gradient et en ajoutant un bruit aléatoire calibré sur cette majoration. Ces modifications entraînent une perte de performance, qui demande un arbitrage au regard de la rigueur du budget de confidentialité. Cette implémentation peut être mise en oeuvre via un *package* dédié pour Tensorflow [39].

La notion de « budget » est importante : chaque nouvelle requête sur les données confidentielle, et notamment chaque nouvelle estimation indépendante d'un algorithme aléatoire entame un peu plus la confidentialité (on additionne les ϵ), de la même manière qu'un ensemble de photos bruités peut permettre de reconstituer une image nette. Cela implique de gérer la confidentialité au niveau d'un fichier et de l'ensemble des usages qui en sont faits.

Les réseaux de neurones permettent aussi d'envisager la construction de fichiers de données de synthèse, des données réalistes mais fausses, préservant ainsi la confidentialité des données réelles tout en permettant de conduire des analyses. La confidentialité est protégée selon le même principe de *differential privacy* : l'algorithme génératif doit être tel que les données artificielles ont une forte probabilité de présenter des caractéristiques similaires même si l'échantillon d'apprentissage diffère d'une observation. L'intérêt est que les usages ultérieurs sur les données de synthèse n'entameront pas davantage le budget de confidentialité.

Cette construction se fonde sur les auto-encodeurs vus au chapitre 4. Une fois entraîné, un AE peut en effet être utilisé comme un *modèle génératif*, capable de produire de « fausses » données à partir d'un vecteur aléatoire. En un sens, cela pousse à l'extrême la logique du débruitage, l'AE générant *in fine* des données décodées à partir de pur bruit. D'autres modèles génératifs existent, mais ils souffrent souvent d'un obstacle computationnel venant de la nécessité d'échantillonner des distributions compliquées. Au contraire, un AE génératif peut être entraîné à encoder et décoder un vecteur aléatoire dont l'échantillonnage est très simple.

En revanche, il reste difficile de définir une fonction de perte adéquate, qui entraîne efficacement l'AE génératif à rapprocher les « fausses » données des « vraies » données, à

38. Dans une perspective bayesienne, un attaquant observant les outputs ne peut corriger ses croyances a priori que d'un facteur $exp(\epsilon)$ au pire.

39. https ://github.com/tensorflow/privacy/

produire des sorties « réalistes ». Il ne suffit pas en effet d'entraîner à AE à produire des sorties spécifiques qui ressemblent à des exemples précis d'un fichier d'entraînement, mais à générer une diversité qui ressemble à la diversité des données observées.

Ce problème trouve une solution dans un article très influent introduisant les modèles adversariaux ou GAN, pour *generative adversarial networks*, (Goodfellow, Pouget-Abadie, Mirza, Xu, Warde-Farley, Ozair, Courville, et Bengio, 2014) [40]. Les GAN entraînent deux RN dans des sens opposés : le générateur tente de fabriquer des faux exemples réalistes. On montre aléatoirement au discriminateur des données réelles ou fabriquées par le générateur, et il est entraîné à distinguer le faux du vrai. La « fonction de perte » qui entraîne le générateur n'est donc pas prédéfinie mais est elle-même apprise, puisqu'il s'agit de la réponse du discriminateur. Dans ce cadre, le souci de la confidentialité pèse sur le discriminateur, puisque c'est la seule partie du dispositif qui a accès aux données originales. Préserver la confidentialité consiste par exemple à entraîner un générateur capable de tromper des discriminateurs entraînés sur des échantillons distincts .

Représentativité

Pour le *machine learning*, les questions statistiques de représentativité se posent en termes de capacité de généralisation d'un modèle. Il s'agit d'un changement de perspective par rapport à la statistique classique : le but n'est plus de garantir l'inférence à une population totale à partir de principes de représentativité de l'échantillon, mais de poursuivre un objectif de généralisation à un domaine global (mais inconnu) à partir d'un échantillon dont on admet qu'il est probablement biaisé. Cet objectif peut par exemple amener à briser volontairement la représentativité, en « équilibrant » par exemple un échantillon pour sur-représenter une classe très rare, que le prédicteur aurait tendance à négliger sinon.

Bien qu'en théorie de l'apprentissage, on s'appuie souvent sur l'hypothèse que les données d'un échantillon test sont issues du même processus générateur que les données d'entraînement, dans la réalité, les algorithmes en production sont mis en oeuvre sur des données qui peuvent être très différentes (parce que l'échantillon d'entraînement n'était pas représentatif, ou parce que du temps a passé) : on parle de *domain shift* ou de validité externe. Les prédicteurs peuvent alors commettre des erreurs importantes. Dans ce document, un exemple en est fourni par la classification des parcelles : les zones de montagnes ou de vignes, absentes de l'échantillon d'entraînement, ne sont pas reconnues par le RN. Le problème est limité si l'algorithme reste performant sur un domaine d'intérêt (en l'occurrence, les régions effectivement en open field ou en bocage). Ce problème se présente également pour le *transfer learning* : les modèles réentraînés conservent des spécificités issues d'un premier échantillon d'entraînement dont on n'a généralement pas la maîtrise, et qui peut être très biaisé.

40. Cependant, Kaloskampis (2019), évoqué au chapitre 4 constate des performances supérieures de l'AE simple comparé à d'autres algorithmes plus avancés (GAN ou AE variationnel) sur un échantillon de 30000 observations et 15 variables (continues) du recensement américain.

Outre le problème de performance prédictive, ces défauts de généralisation peuvent poser des problèmes d'équité, en particulier lorsque les échantillons d'entraînement reflètent des injustices sociales. Ne sachant pas distinguer causalité et corrélation, le RN (ou un autre algorithme de *machine learning*) va alors apprendre à reproduire ces biais dans ses prédictions. S'il est utilisé pour prendre des décisions (de recrutement par exemple), le RN devient lui-même une cause d'injustice en transformant les corrélations apprises en causalité. Le sujet de l'équité algorithmique (*algorithmic fairness*) est déjà un champ de recherche mature avec une littérature abondante (pour une présentation synthétique récente, voir Mehrabi, Morstatter, Saxena, Lerman, et Galstyan (2021)).

Pour un usage de statistique publique, ces questions d'équité peuvent sembler moins sensibles puisque, d'une part, la représentativité des données d'entraînement est souvent contrôlée, et que, d'autre part, l'exploitation des données est descriptive et non pas prescriptive. C'est le cas notamment de la discussion sur ces questions éthiques dans le cadre de l'imputation du salaire en section 2.3.1. Cependant ces questions méritent une attention particulière, notamment si ces algorithmes alimentent une interface utilisateur, comme l'auto-complétion dans un questionnaire en ligne.

Mieux exploiter les RN

Nous avons tout au long de ce document négligé quatre questions qui occupent beaucoup l'industrie et la recherche. La première est celle du temps de calcul. Les RN ont connu un développement spectaculaire grâce à l'exploitation des cartes graphiques pour paralléliser massivement les calculs, au point que le fabricant de cartes graphiques Nvidia est devenu un acteur central de l'intelligence artificielle. Les *packages* que nous utilisons proposent des solutions simples pour paralléliser et accélérer les calculs, mais celles-ci demandent des ressources adaptées [41]. Du temps de calcul dépend en particulier l'optimisation plus systématique des hyperparamètres. Ce point est d'autant plus important que les hyperparamètres des réseaux de neurones peuvent être très nombreux. Quand l'entraînement est rapide, on peut tester de très nombreuses configurations et orienter le modèle vers les hyperparamètres les plus efficaces, y compris en entraînant des algorithmes de *machine learning* sur ce problème d'optimisation des RN. Là aussi, de nombreuses solutions logicielles sont disponibles, comme HyperOpt mobilisée dans la section 3.3.4 (l'intérêt de cette librairie est notamment de faciliter la sélection des hyperparamètres à tester). En ce sens, nous avons ici décrit des méthodes pour construire des prototypes, suffisants pour démontrer un usage ou pour obtenir une estimation dans le cadre d'une étude. Mis en production, il deviendrait intéressant d'investir dans leur optimisation.

La seconde question est justement celle de la mise en production. Elle implique de pouvoir reproduire les modèles dans différents contextes et sur différentes données, de les maintenir et de les réentraîner. La reproductibilité des RN est délicate en conséquence directe de la complexité des algorithmes. Le code doit être portable, facile à maintenir, et

41. Disponibles à l'Insee sur le SSP Cloud.

robuste aux évolutions des packages. Les modèles de RN eux-même doivent pouvoir être sauvegardés. Ils impliquent une lourde préparation des données, et cette étape doit pouvoir s'appliquer à de nouvelles données sans modifier ou réentraîner le modèle lui-même. L'aléa est présent partout dans le processus (séparation train et test, dropout, initialisation des poids, entraînement, etc.). Il doit être contrôlé, soit en le rendant déterministe quand c'est possible, soit en s'assurant de la stabilité de l'algorithme.

La troisième question est celle de la réduction de l'empreinte environnementale des algorithmes de *machine learning*. Les composants matériels utilisés requièrent des métaux qu'il peut être polluant d'extraire et de produire. Les algorithmes peuvent être gourmands en énergie et donc contribuer aux émissions de gaz à effet de serre quand l'énergie est carbonée. Des efforts sont donc faits dans l'industrie pour dimensionner les codes aux besoins réels, de sorte à éviter d'utiliser un algorithme très consommateur d'énergie quand un algorithme moins gourmand remplit l'objectif suffisamment bien. La question est moins préoccupante dans le cas des statistiques publiques, où les données et les usages restent limités. Mais elle pourrait se poser davantage à l'avenir.

Enfin, quatrième question importante, les méthodes permettant d'ouvrir la boite noire des RN et des algorithmes de *machine learning* en général et de les rendre interprétables. Il ne s'agit pas nécessairement de tirer des conclusions sur les relations réelles entre variables, mais de mieux comprendre comment le modèle fonctionne. La section 32 en fournit un exemple en colorant les pixels les plus déterminants. Là encore, de nombreuses solutions existent, et leur mobilisation dans un usage de statisticien mériterait d'être approfondie. Les différentes méthodes consistent généralement à mesurer « l'importance » d'une variable donnée, sa contribution à la qualité de la prédiction, par exemple en comparant la performance du modèle avec et sans l'information fournie par cette variable (en permutant aléatoirement ses valeurs). Par distinction avec ces méthodes « globales », il est souvent plus intéressant de rendre un modèle interprétable à un niveau local, pour une observation donnée. Plusieurs méthodes locales, qui estiment une approximation linéaire du comportement du modèle autour d'une observation donnée, peuvent être unifiées dans le cadre des « shapley values » (Lundberg et Lee, 2017) et sont implémentées dans le package SHAP [42] pour Python. Le *package* Skater [43] permet de mettre en oeuvre plusieurs méthodes d'interprétation globales et locales.

D'autres types de RN et d'autres usages possibles

D'autres types de RN peuvent être utiles aux statisticiens. Certains RN sont plus adaptés au traitement de données textuelles ou aux processus temporels comme les RNNs (*Recurrent Neural Networks*) ou les LSTMs (*Long Short Term Memory*). Ils représentent une large famille de RN (dont certains peuvent s'appuyer sur l'encodage des mots avec word2vec que nous avons évoqué section 2.3.3). Ils fondent par exemple les algorithmes de traduction ou d'auto-complétion, et peuvent servir à exploiter des données textuelles

42. https ://github.com/slundberg/shap
43. https ://oracle.github.io/Skater/overview.html

(par exemple des offres d'emplois scrappées en ligne, la classification de libellés de tickets de caisse dans une nomenclature, ou encore la classification des libellés des métiers dans la nomenclature des professions (Leroy, Malherbe, et Seimandi, 2022)) pour un usage statistique. Ils sont également testés sur la prédiction de séries temporelles de suffisamment grande taille (Paranhos, 2021). Depuis 2017 (Vaswani, Shazeer, Parmar, Uszkoreit, Jones, Gomez, Kaiser, et Polosukhin, 2017), les RN « transformeurs » fondés sur des mécanismes d'« attention » ont fait fortement progresser l'analyse du langage (entre autres) en permettant de mieux exploiter des bases de données massives pour l'entraînement. Ils ont permis la diffusion de modèles généralistes pré-entraînés de grande taille, qui peuvent ensuite être ré-entraînés sur des applications spécifiques.

Les réseaux variationnels visent à prédire la distribution des labels, et non pas les labels eux-mêmes. Ils peuvent être mobilisés notamment sous forme d'auto-encodeurs pour des réseaux génératifs, et servir à générer des données de synthèse.

Les RN utilisés pour la traduction automatique peuvent aussi inspirer des systèmes « d'appariements de synthèse » entre deux bases de données, deux enquêtes proches mais difficiles à apparier de manière classique. Dans la configuration la plus simple, les traducteurs s'entraînent sur un corpus déjà traduit (ou corpus d'observations déjà appariées entre deux sources statistiques différentes). Ce sont par exemple des RNN auto-encodeurs qui reçoivent un texte dans une langue A en entrée et visent le même texte traduit dans une langue B en sortie. Cependant, la taille du corpus déjà traduit (ou du fichier de données déjà appariées) peut limiter l'entraînement et la performance du modèle dans un tel schéma. Mais il existe des solutions pour entraîner des traducteurs à partir de corpus non traduits, par exemple en entraînant en même temps deux auto-encodeurs, dans chacune des deux langues, sous contrainte de partager la même couche d'encodage (Lample, Conneau, Denoyer, et Ranzato, 2017).

Enfin, les RN, comme d'autres méthodes de *machine learning*, connaissent un développement récent en économétrie. On peut alors surmonter le problème de la boîte noire et conjuguer la souplesse de ces modèles (notamment dans l'estimation des paramètres de nuisance) et une inférence rigoureuse dans la lignée des travaux sur l'économétrie en grande dimension (L'Hour, 2020). Ces recherches récentes découlent en grande partie de la description des estimateurs ML « doublement robustes et débiaisés » par Chernozhukov, Chetverikov, Demirer, Duflo, Hansen, Newey, et Robins (2018), qui corrigent deux biais, un biais de régularisation et un biais de surapprentissage. Ces principes sont présentés à partir d'un modèle semi-linéaire : on s'intéresse à l'estimation du paramètre θ_0, effet d'une variable causale D sur une variable d'intérêt Y, en présence de variables de contrôle X ayant des relations potentiellement non-linéaires avec D et Y.

$$\begin{aligned} Y &= D\theta_0 + g_0(X) + U, && \text{avec } \mathbb{E}[U|X, D] = 0 \\ D &= m_0(X) + V, && \text{avec } \mathbb{E}[V|X] = 0 \end{aligned}$$

Une utilisation naïve du *machine learning* consisterait à se préoccuper uniquement de la première équation, par exemple d'estimer $\hat{g}_0$ avec un algorithme ML puis d'estimer θ_0 à

partir de :

$$Y = D\theta_0 + \hat{g}_0(X) + U$$

Cette estimation serait biaisée car l'algorithme ML vise une bonne prédiction de Y mais néglige la modélisation du lien entre X et D. Il s'agit d'un biais de régularisation (avec un algorithme lasso par exemple, il est équivalent à un biais de variable omise). Pour l'éviter, il faut construire un estimateur « doublement robuste » qui fasse également intervenir la seconde équation en estimant $\hat{m}_0$ et donc $\hat{V}$ par ML :

$$\hat{\theta}_0 = (1/n \sum_i \hat{V}_i D_i)^{-1} 1/n \sum_i \hat{V}_i (Y_i - \hat{g}_0(X_i))$$

Un tel estimateur reste sans biais même si l'un des deux modèles est mal spécifié. Cette exigence de double robustesse n'est pas propre aux modèles de machine learning, mais dans leur cas, elle découle du fait que l'apprentissage peut-être considéré comme un processus de sélection de modèle. Les estimateurs doublement robustes doivent cependant être « débiaisés » d'un biais de surapprentissage, dont l'importance découle cette fois-ci directement de l'utilisation du machine learning : les erreurs dues au surapprentissage dans les estimations de $\hat{g}_0$ et $\hat{m}_0$ ne sont pas indépendantes. Pour débiaiser l'estimateur doublement robuste, on a recours au sample-splitting : on divise l'échantillon en deux, on entraîne, par exemple, un modèle de score de propension $\hat{m}_0$ sur une moitié et on estime l'effet causal sur l'autre moitié, puis on recommence en intervertissant les rôles.

Il existe sans doute, on le pressent, beaucoup d'usages encore à imaginer. Dès lors que les RN peuvent être entraînés avec succès sur les données existantes de la statistique publique, y compris des données d'enquêtes de taille relativement modeste, la porte s'ouvre sur leur utilisation. Dans les années qui viennent, la disponibilité de données plus nombreuses, plus variées et plus massives pourrait encourager davantage l'usage des RN. Ils sont incontournables pour tirer profit de données textuelles ou d'images. Sur d'autres types de données, ils sont un outil parmi l'ensemble des algorithmes d'apprentissage, complexes à mettre en oeuvre mais puissants et versatiles.

Les cas d'usages présentés ici ont aussi permis de mesurer à quel point les RN ne dispensent pas des précautions que prennent habituellement les statisticiens. Bien qu'ils visent à « laisser parler les données », ils demandent une réflexion importante pour être utilisés avec rigueur. En apprentissage supervisé, leur qualité reflète la qualité et la quantité des données d'entraînement labellisées dont on dispose. En économétrie, ils doivent se conjuguer avec le souci de l'identification et de l'inférence causale. Sur des données confidentielles, ils génèrent des modèles qui doivent également être traités comme confidentiels, sauf à mettre en oeuvre une stratégie explicite et maîtrisée d'ajout de bruit aléatoire. Enfin, les RN réactivent, sous une nouvelle forme, des questions classiques de représentativité et de déontologie dans l'utilisation qui est faite des modèles statistiques.

Références

Abadi, M., A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, et L. Zhang (2016) : "Deep learning with differential privacy," in *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pp. 308–318.

Ackley, D. H., G. E. Hinton, et T. J. Sejnowski (1985) : "A learning algorithm for Boltzmann machines," *Cognitive science*, 9(1), 147–169.

Athey, S., et G. W. Imbens (2019) : "Machine learning methods that economists should know about," *Annual Review of Economics*, 11, 685–725.

Babet, D. (2020) : "Wage Imputation with Deep Learning in the French Labor Force Survey," in *UNECE workshop on statistical data editing, conference of european statisticians*.

Ballard, D. H. (1987) : "Modular learning in neural networks," in *AAAI*, vol. 647, pp. 279–284.

Bang, H., et J. M. Robins (2005) : "Doubly robust estimation in missing data and causal inference models," *Biometrics*, 61(4), 962–973.

Beam, A. L., B. Kompa, A. Schmaltz, I. Fried, G. Weber, N. Palmer, X. Shi, T. Cai, et I. S. Kohane (2019) : "Clinical concept embeddings learned from massive sources of multimodal medical data," in *Pacific Symposium on Biocomputing 2020*, pp. 295–306. World Scientific.

Bellman, R. E. (1961) : *Adaptive Control Processes*. Princeton University Press, Princeton, New Jersey.

Bergstra, J., R. Bardenet, Y. Bengio, et B. Kégl (2011) : "Algorithms for Hyper-Parameter Optimization," in *NIPS'11 : Proceedings of the 24th International Conference on Neural Information Processing Systems*, p. 2546–2554.

Boelaert, J., et É. Ollion (2018) : "The Great Regression," *Revue française de sociologie*, 59(3), 475–506.

Boistard, H., G. Chauvet, et D. Haziza (2016) : "Doubly robust inference for the distribution function in the presence of missing survey data," *Scandinavian Journal of Statistics*, 43(3), 683–699.

Bourlard, H., et Y. Kamp (1988) : "Auto-association by multilayer perceptrons and singular value decomposition," *Biological cybernetics*, 59(4), 291–294.

Breiman, L. (2001) : "Statistical Modeling : The Two Cultures," *Statistical Science, Vol. 16, No. 3, 199–231.*

Carlini, N., C. Liu, Ú. Erlingsson, J. Kos, et D. Song (2019) : "The secret sharer : Evaluating and testing unintended memorization in neural networks," in *28th USENIX Security Symposium*, pp. 267–284.

Cayton, L. (2005) : "Algorithms for manifold learning," *Univ. of California at San Diego Tech. Rep*, 12(1-17), 1.

Charpentier, A., E. Flachaire, et A. Ly (2018) : "Econométrie et Machine Learning," *Economie et Statistique*, (505-506), 147–169.

Chen, S., et D. Haziza (2017) : "Multiply robust imputation procedures for the treatment of item nonresponse in surveys," *Biometrika*, 104(2), 439–453.

——— (2019) : "Recent developments in dealing with item non-response in surveys : a critical review," *International Statistical Review*, 87, S192–S218.

Chernozhukov, V., D. Chetverikov, M. Demirer, E. Duflo, C. Hansen, W. Newey, et J. Robins (2018) : "Double/debiased machine learning for treatment and structural parameters," *The Econometrics Journal*, 21(1).

Deltour, Q., T. Faria, et T. Roudil-Valentin (2020) : "Réseaux de neurones sur des données de statistiques publiques : vectorisation de l'enquête Emploi," ENSAE, mémoire de statistiques appliquées.

Derrow-Pinion, A., J. She, D. Wong, O. Lange, T. Hester, L. Perez, M. Nunkesser, S. Lee, X. Guo, B. Wiltshire, et al. (2021) : "Eta prediction with graph neural networks in google maps," in *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, pp. 3767–3776.

Dion, R. (1934) : *Essai sur la formation du paysage rural français*. Arrault, Tours.

Doutreligne, M., A. Leduc, D.-P. Nguyen, et A. Vuagnat (2020) : "Snds2vec, représentations continues pour les concepts médicaux du Système national des données de santé," *Revue d'Épidémiologie et de Santé Publique*, 68, S35.

Dwork, C. (2008) : "Differential privacy : A survey of results," in *International conference on theory and applications of models of computation*, pp. 1–19. Springer.

Dwork, C., F. McSherry, K. Nissim, et A. Smith (2006) : "Calibrating noise to sensitivity in private data analysis," in *Theory of cryptography conference*, pp. 265–284. Springer.

Dwork, C., A. Roth, et al. (2014) : "The algorithmic foundations of differential privacy," *Foundations and Trends® in Theoretical Computer Science*, 9(3–4), 211–407.

Ester, M., H.-P. Kriegel, J. Sander, X. Xu, et al. (1996) : "A density-based algorithm for discovering clusters in large spatial databases with noise," in *kdd*, vol. 96, pp. 226–231.

Goodfellow, I., Y. Bengio, et A. Courville (2016) : *Deep Learning*. MIT Press, http://www.deeplearningbook.org.

Goodfellow, I., J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, et Y. Bengio (2014) : "Generative adversarial nets," *Advances in neural information processing systems*, 27.

Haziza, D., J.-F. Beaumont, et al. (2017) : "Construction of weights in surveys : A review," *Statistical Science*, 32(2), 206–226.

He, K., X. Zhang, S. Ren, et J. Sun (2016) : "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778.

Hebbs, D. (1949) : *The Organization of Behavior*. John Wiley and Sons, New York.

Hermann, L. (2018) : "Le lotissement en France : histoire réglementaire de la construction d'un outil de production de la ville," *Géoconfluences*.

Himpens, S., M. Poulhes, et F. Sémécurbe (2021) : "Bâti dispersé, bâti concentré, des disparités territoriales persistantes," *Insee Analyses*, (63).

Hinton, G. E., et R. R. Salakhutdinov (2006) : "Reducing the dimensionality of data with neural networks," *Science*, 313(5786), 504–507.

Josse, J., N. Prost, E. Scornet, et G. Varoquaux (2019) : "On the consistency of supervised learning with missing values," *arXiv preprint arXiv :1902.06931*.

Kaloskampis, I. (2019) : "Synthetic data for public good," *Data science for public good*.

Kingma, D. P., et J. Ba (2014) : "Adam : A method for stochastic optimization," *arXiv preprint arXiv :1412.6980*.

Kingma, D. P., M. Welling, et al. (2019) : "An introduction to variational autoencoders," *Foundations and Trends® in Machine Learning*, 12(4), 307–392.

Klambauer, G., T. Unterthiner, A. Mayr, et S. Hochreiter (2017) : "Self-normalizing neural networks," *arXiv preprint arXiv :1706.02515*.

Kukačka, J., V. Golkov, et D. Cremers (2017) : "Regularization for deep learning : A taxonomy," *arXiv preprint arXiv :1710.10686*.

Lample, G., A. Conneau, L. Denoyer, et M. Ranzato (2017) : "Unsupervised machine translation using monolingual corpora only," *arXiv preprint arXiv :1711.00043*.

Le Cun, Y. (1985) : "Une procédure d'apprentissage pour réseau à seuil assymétrique," in *Proceedings of Cognitiva 85*, pp. 599–604, Paris.

Le Cun, Y., A. Canziani, I. Misra, M. Lewis, et X. Bresson (2020) : "Deep Learning," https://atcold.github.io/pytorch-Deep-Learning/.

Lemieux, T. (2006) : "The "Mincer equation" thirty years after schooling, experience, and earnings," in *Jacob Mincer a pioneer of modern labor economics*, pp. 127–145. Springer.

Leroy, T., L. Malherbe, et T. Seimandi (2022) : "Application de techniques de machine learning pour coder les professions en PCS 2020," in *Journées de méthodologie statistique de l'Insee 2022*.

Lettvin, J., H. Maturana, W. McCulloch, et W. Pitts (1959) : "What the Frog's Eye Tells the Frog's Brain," in *Proceedings of the IRE*, vol. 47, pp. 1940–51. John Wiley and Sons Inc (1 avril 1968).

L'Hour, J. (2020) : "L'économétrie en grande dimension," *Document de travail, Insee*, M2020/01.

Lu, L., Y. Shin, Y. Su, et G. Em Karniadakis (2020) : "Dying ReLU and Initialization : Theory and Numerical Examples," *Communications in Computational Physics*, 28(5), 1671–1706.

Lundberg, S., et S.-I. Lee (2017) : "A unified approach to interpreting model predictions," *arXiv preprint arXiv :1705.07874*.

McCulloch, W., et W. Pitts (1943) : "A logical calculus of the ideas immanent in nervous activity," *Bulletin of Mathematical Biophysics*, pp. 115–133.

Mehrabi, N., F. Morstatter, N. Saxena, K. Lerman, et A. Galstyan (2021) : "A survey on bias and fairness in machine learning," *ACM Computing Surveys (CSUR)*, 54(6), 1–35.

Mikolov, T., K. Chen, G. Corrado, et J. Dean (2013) : "Efficient estimation of word representations in vector space," *arXiv preprint arXiv :1301.3781*.

Mikolov, T., I. Sutskever, K. Chen, G. Corrado, et J. Dean (2013) : "Distributed representations of words and phrases and their compositionality," *arXiv preprint arXiv :1310.4546*.

Mincer, J. (1974) : *Schooling, Experience, and Earnings*, vol. 2 of *Human Behavior & Social Institutions*. NBER, New York.

Minsky, M. L., et S. Papert (1969) : *Perceptrons : An Introduction to Computational Geometry*. MIT Press, Cambridge, MA, USA.

Mullainathan, S., et J. Spiess (2017) : "Machine learning : an applied econometric approach," *Journal of Economic Perspectives*, 31(2), 87–106.

Paranhos, L. (2021) : "Predicting Inflation with Neural Networks," *arXiv preprint arXiv :2104.03757*.

Parker, D. B. (1985) : "Learning-logic : Casting the Cortex of the Human Brain in Silicon," Discussion paper, MIT, Center for Computational Research in Economics and Management Science.

Picart, C., et D. Babet (2020) : "Indépendants : une offre de travail plus contrainte depuis 2008," in *Insee Référence, Emploi et revenus des indépendants*. Insee.

Ranzato, M., C. S. Poultney, S. Chopra, et Y. LeCun (2006) : "Efficient Learning of Sparse Representations with an Energy-Based Model," in *NIPS*.

Rombach, R., A. Blattmann, D. Lorenz, P. Esser, et B. Ommer (2022) : "High-resolution image synthesis with latent diffusion models," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10684–10695.

Rosenblatt, F. (1958) : "The perceptron : a probabilistic model for information storage and organisation in the brain," *Psychological Review*, 65(6).

Rubin, D. B. (1976) : "Inference and missing data," *Biometrika*, 63(3), 581–592.

Ruder, S. (2016) : "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv :1609.04747*.

Rumelhart, D. E., G. E. Hinton, et R. J. Williams (1986) : "Learning representations by backpropagating errors," *Nature*, 323(6088), 533–536.

Salakhutdinov, R., A. Mnih, et G. Hinton (2007) : "Restricted Boltzmann machines for collaborative filtering," in *Proceedings of the 24th international conference on Machine learning*, pp. 791–798.

Samuel, A. L. (1959) : "Some Studies in Machine Learning Using the Game of Checkers," *IBM Journal of Research and Development*, 3(3), 210–229.

Selvaraju, R. R., M. Cogswell, A. Das, R. Vedantam, D. Parikh, et D. Batra (2019) : "Grad-CAM : Visual Explanations from Deep Networks via Gradient-based Localization," *International Journal of Computer Vision (IJCV)*.

Sirko, W., S. Kashubin, M. Ritter, A. Annkah, Y. S. E. Bouchareb, Y. Dauphin, D. Keysers, M. Neumann, M. Cisse, et J. Quinn (2021) : "Continental-Scale Building Detection from High Resolution Satellite Imagery," *arXiv preprint arXiv :2107.12283*.

Smolensky, P. (1986) : "Information processing in dynamical systems : Foundations of harmony theory," Discussion paper, Colorado Univ at Boulder Dept of Computer Science.

Srivastava, N., G. Hinton, A. Krizhevsky, I. Sutskever, et R. Salakhutdinov (2014) : "Dropout : a simple way to prevent neural networks from overfitting," *The journal of machine learning research*, 15(1), 1929–1958.

Tang, F., et H. Ishwaran (2017) : "Random forest missing data algorithms," *Statistical Analysis and Data Mining : The ASA Data Science Journal*, 10(6), 363–377.

Tieleman, T., G. Hinton, et al. (2012) : "Lecture 6.5-rmsprop : Divide the gradient by a running average of its recent magnitude," *COURSERA : Neural networks for machine learning*, 4(2), 26–31.

Vaswani, A., N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, et I. Polosukhin (2017) : "Attention is all you need," *Advances in neural information processing systems*, 30.

Vincent, P., H. Larochelle, Y. Bengio, et P.-A. Manzagol (2008) : "Extracting and composing robust features with denoising autoencoders," in *Proceedings of the 25th international conference on Machine learning*, pp. 1096–1103.

Watteaux, M. (2005) : "Sous le bocage, le parcellaire…," *Études rurales*, 175–176, 53–80.

Werbos, P. J. (1974) : "Beyond Regression : New tools for Prediction and Analysis in the Behavioral Sciences," Ph.D. thesis, Harvard University.

Yu, Q., C. Szegedy, M. C. Stumpe, L. Yatziv, V. Shet, J. Ibarz, et S. Arnoud (2015) : "Large scale business discovery from street level imagery," *arXiv preprint arXiv :1512.05430*.

Zhang, C., S. Bengio, M. Hardt, B. Recht, et O. Vinyals (2021) : "Understanding deep learning (still) requires rethinking generalization," *Communications of the ACM*, 64(3), 107–115.

Série des Documents de Travail « Méthodologie Statistique »

9601 : Une méthode synthétique, robuste et efficace pour réaliser des estimations locales de population.
G. DECAUDIN, J.-C. LABAT

9602 : Estimation de la précision d'un solde dans les enquêtes de conjoncture auprès des entreprises.
N. CARON, P. RAVALET, O. SAUTORY

9603 : La procédure FREQ de SAS - Tests d'indépendance et mesures d'association dans un tableau de contingence.
J. CONFAIS, Y. GRELET, M. LE GUEN

9604 : Les principales techniques de correction de la non-réponse et les modèles associés.
N. CARON

9605 : L'estimation du taux d'évolution des dépenses d'équipement dans l'enquête de conjoncture : analyse et voies d'amélioration.
P. RAVALET

9606 : L'économétrie et l'étude des comportements. Présentation et mise en œuvre de modèles de régression qualitatifs. Les modèles univariés à résidus logistiques ou normaux (LOGIT, PROBIT).
S. LOLLIVIER, M. MARPSAT, D. VERGER

9607 : Enquêtes régionales sur les déplacements des ménages : l'expérience de Rhône-Alpes.
N. CARON, D. LE BLANC

9701 : Une bonne petite enquête vaut-elle mieux qu'un mauvais recensement ?
J.-C. DEVILLE

9702 : Modèles univariés et modèles de durée sur données individuelles.
S. LOLLIVIER

9703 : Comparaison de deux estimateurs par le ratio stratifiés et application aux enquêtes auprès des entreprises.
N. CARON, J.-C. DEVILLE

9704 : La faisabilité d'une enquête auprès des ménages.
1. au mois d'août.
2. à un rythme hebdomadaire

C. LAGARENNE, C. THIESSET

9705 : Méthodologie de l'enquête sur les déplacements dans l'agglomération toulousaine.
P. GIRARD.

9801 : Les logiciels de désaisonnalisation TRAMO & SEATS : philosophie, principes et mise en œuvre sous SAS.
K. ATTAL-TOUBERT, D. LADIRAY

9802 : Estimation de variance pour des statistiques complexes : technique des résidus et de linéarisation.
J.-C. DEVILLE

9803 : Pour essayer d'en finir avec l'individu Kish.
J.-C. DEVILLE

9804 : Une nouvelle (encore une !) méthode de tirage à probabilités inégales.
J.-C. DEVILLE

9805 : Variance et estimation de variance en cas d'erreurs de mesure non corrélées ou de l'intrusion d'un individu Kish.
J.-C. DEVILLE

9806 : Estimation de précision de données issues d'enquêtes : document méthodologique sur le logiciel POULPE.
N. CARON, J.-C. DEVILLE, O. SAUTORY

9807 : Estimation de données régionales à l'aide de techniques d'analyse multidimentionnelle.
K. ATTAL-TOUBERT, O. SAUTORY

9808 : Matrices de mobilité et calcul de la précision associée.
N. CARON, C. CHAMBAZ

9809 : Échantillonnage et stratification : une étude empirique des gains de précision.
J. LE GUENNEC

9810 : Le Kish : les problèmes de réalisation du tirage et de son extrapolation.
C. BERTHIER, N. CARON, B. NEROS

9901 : Perte de précision liée au tirage d'un ou plusieurs individus Kish.
N. CARON

9902 : Estimation de variance en présence de données imputées : un exemple à partir de l'enquête Panel Européen.
N. CARON

0001 : L'économétrie et l'étude des comportements. Présentation et mise en œuvre de modèles de régression qualitatifs. Les modèles univariés à résidus logistiques ou normaux (LOGIT, PROBIT) (version actualisée).
S. LOLLIVIER, M. MARPSAT, D. VERGER

0002 : Modèles structurels et variables explicatives endogènes.
J.-M. ROBIN

0003 : L'enquête 1997-1998 sur le devenir des personnes sorties du RMI - Une présentation de son déroulement.
D. ENEAU, D. GUILLEMOT

0004 : Plus d'amis, plus proches ? Essai de comparaison de deux enquêtes peu comparables.
O. GODECHOT

0005 : Estimation dans les enquêtes répétées : application à l'Enquête Emploi en Continu.
N. CARON, P. RAVALET

0006 : Non-parametric approach to the cost-of-living index.
F. MAGNIEN, J. POUGNARD

0101 : Diverses macros SAS : Analyse exploratoire des données, Analyse des séries temporelles.
D. LADIRAY

0102 : Économétrie linéaire des panels : une introduction.
T. MAGNAC

0201 : Application des méthodes de calages à l'enquête EAE-Commerce.
N. CARON

C 0201 : Comportement face au risque et à l'avenir et accumulation patrimoniale - Bilan d'une expérimentation.
L. ARRONDEL, A. MASSON, D. VERGER

C 0202 : Enquête Méthodologique Information et Vie Quotidienne - Tome 1 : bilan du test 1, novembre 2002.
J.-A. VALLET, G. BONNET, J.-C. EMIN, J. LEVASSEUR, T. ROCHER, P. VRIGNAUD, X. D'HAULTFOEUILLE, F. MURAT, D. VERGER, P. ZAMORA

0203 : General principles for data editing in business surveys and how to optimise it.
P. RIVIERE

0301 : Les modèles logit polytomiques non ordonnés : théories et applications.
C. AFSA ESSAFI

0401 : Enquête sur le patrimoine des ménages - Synthèse des entretiens monographiques.
V. COHEN, C. DEMMER

0402 : La macro SAS CUBE d'échantillonnage équilibré
S. ROUSSEAU, F. TARDIEU

0501 : Correction de la non-réponse et calage de l'enquêtes Santé 2002
N. CARON, S. ROUSSEAU

0502 : Correction de la non-réponse par répondération et par imputation
N. CARON

0503 : Introduction à la pratique des indices statistiques - notes de cours
J-P BERTHIER

0601 : La difficile mesure des pratiques dans le domaine du sport et de la culture - bilan d'une opération méthodologique
C. LANDRE, D. VERGER

0801 : Rapport du groupe de réflexion sur la qualité des enquêtes auprès des ménages
D. VERGER

M2013/01 : La régression quantile en pratique
P. GIVORD,
X. D'HAULTFOEUILLE

M2014/01 : La microsimulation dynamique : principes généraux et exemples en langage R
D. BLANCHET

M2015/01 : la collecte multimode et le paradigme de l'erreur d'enquête totale
T. RAZAFINDROVONA

M2015/02 : Les méthodes de Pseudo-Panel
M. GUILLERM

M2015/03 : Les méthodes d'estimation de la précision pour les enquêtes ménages de l'Insee tirées dans Octopusse
E. GROS
K. MOUSSALAM

M2016/01 : Le modèle Logit Théorie et application.
C. AFSA

M2016/02 : Les méthodes d'estimation de la précision de l'Enquête Emploi en Continu
E. GROS
K.MOUSSALAM

M2016/03 : Exploitation de l'enquête expérimentale Vols, violence et sécurité.
T. RAZAFINDROVONA

M2016/04 : Savoir compter, savoir coder. Bonnes pratiques du statisticien en programmation.
E. L'HOUR
R. LE SAOUT
B. ROUPPERT

M2016/05 : Les modèles multiniveaux
P. GIVORD
M. GUILLERM

M2016/06 : Econométrie spatiale : une introduction pratique
P. GIVORD
R. LE SAOUT

M2016/07 : La gestion de la confidentialité pour les données individuelles
M. BERGEAT

M2016/08 : Exploitation de l'enquête expérimentale Logement internet-papier
T. RAZAFINDROVONA

M2017/01: Exploitation de l'enquête expérimentale Qualité de vie au travail
T. RAZAFINDROVONA

M2018/01: Estimation avec le score de propension sous 
S. QUANTIN

M2018/02: Modèles semi-paramétriques de survie en temps continu sous 
S. QUANTIN

M2019/01 : Les méthodes de décomposition appliquées à l'analyse des inégalités
B. BOUTCHENIK
E. COUDIN
S. MAILLARD

M2020/01 : L'économétrie en grande dimension
J. L'HOUR

M2021/01 : R Tools for JDemetra+ - Seasonal adjustment made easier
A. SMYK
A. TCHANG

M2021/02 : Le traitement du biais de sélection endogène dans les enquêtes auprès des ménages par modèle de Heckman
L. CASTELL
P. SILLARD

M2021/03 : Conception de questionnaires auto-administrés
H. KOUMARIANOS
A. SCHREIBER

M2022/01 : Introduction à la géomatique pour le statisticien : quelques concepts et outils innovants de gestion, traitement et diffusion de l'information spatiale
F. SEMECURBE
E. COUDIN

M2022/02 : Le zonage en unites urbaines 2020
V. COSTEMALLE
S. OUJIA
C. GUILLO
A. CHAUVET

M2023/01 : Les réseaux de neurones appliqués à la statistique publique : méthodes et cas d'usages
D. BABET
Q. DELTOUR
T. FARIA
S. HIMPENS