Titre: Title:	Apprentissage machine pour l'accélération de l'optimisation des blocs mensuels d'équipages aériens
Auteur: Author:	Alice Wu
Date:	2019
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Wu, A. (2019). Apprentissage machine pour l'accélération de l'optimisation des blocs mensuels d'équipages aériens [Master's thesis, Polytechnique Montréal]. PolyPublie. https://publications.polymtl.ca/4099/

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Apprentissage machine pour l'accélération de l'optimisation des blocs mensuels d'équipages aériens

ALICE WU
Département de mathématiques et de génie industriel

Mémoire présenté en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*
Mathématiques

Décembre 2019

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Ce mémoire intitulé :

Apprentissage machine pour l'accélération de l'optimisation des blocs mensuels d'équipages aériens

présenté par **Alice WU**
en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*
a été dûment accepté par le jury d'examen constitué de :

Nadia LAHRICHI, Ph. D., présidente
François SOUMIS, Ph. D., membre et directeur de recherche
Guy DESAULNIERS, Ph. D., membre et codirecteur de recherche
Richard LABIB, Ph. D., membre

DÉDICACE

À mon héros, à ma lumière,
et en mémoire de mon grand-père.

REMERCIEMENTS

Je tiens à remercier tout d'abord mon directeur de recherche François Soumis, qui est resté très disponible et qui m'a guidée tout au long de cette maîtrise, ainsi que mon co-directeur de recherche Guy Desaulniers pour son aide et sa motivation.

Je tiens également à remercier l'entreprise AD OPT qui a fourni les données d'une compagnie aérienne et a ainsi permis à ce projet d'avoir un ancrage réel. AD OPT et ses employés sont aussi restés disponibles pour répondre à mes questions et pour me guider sur mon projet ; je leur en suis reconnaissante.

Je remercie également IVADO et le GERAD pour les bourses financières accordées durant cette maîtrise.

Un grand merci à tous mes professeurs de mathématiques de l'École Polytechnique de Montréal. La transmission de vos connaissances s'est faite de manière pédagogique et agréable, et j'en retire un enseignement important.

Je remercie également mes camarades de projet et étudiants au doctorat Yassine Yaakoubi, qui est toujours resté disponible et qui m'a grandement aidée du côté de l'apprentissage machine, et Philippe Racette, avec qui j'ai collaboré tout au long de ce projet.

Merci également à l'équipe du GERAD pour leur soutien, ainsi qu'à tous mes collègues et amis qui m'ont soutenue au quotidien. Je n'oublie pas mes amis outre-Atlantique et en Asie pour leur aide et leur soutien, merci.

RÉSUMÉ

Les compagnies aériennes sont soumises à une forte pression pour diminuer leurs coûts, c'est pourquoi elles consacrent beaucoup d'efforts dans l'optimisation de la planification aérienne, et notamment dans l'optimisation des horaires d'équipages, car le personnel de bord est leur deuxième poste de dépense après le carburant.

Ce mémoire se situe dans le cadre de l'optimisation des blocs mensuels d'équipages aériens, qui consiste à construire les emplois du temps des membres d'équipage sur un mois, en affectant à tous les employés des rotations déjà construites, et ce de manière personnalisée. Une rotation est une séquence de vols commençant et terminant au même aéroport de base. Une méthode connue et efficace pour résoudre ces problèmes est la génération de colonnes, utilisée par l'entreprise AD OPT qui a collaboré sur ce projet. Dans cette méthode, un problème maître restreint optimise les coûts sur un sous-ensemble de blocs mensuels possibles en s'assurant de la couverture de toutes les rotations. D'autre part, des sous-problèmes se chargent de générer des nouveaux blocs mensuels qui vont améliorer la solution du problème maître restreint. La génération de colonnes est un algorithme itératif exact qui considère implicitement tous les blocs possibles sans les énumérer.

Dans notre cas, un sous-problème est un problème de plus court chemin dans un réseau contenant toutes les rotations possibles, un chemin représentant un bloc. Il y a un sous-problème par employé. Les données fournies par AD OPT concernent une compagnie aérienne réelle, ayant environ 2000 employés et 5000 rotations par mois. On a donc ici 2000 réseaux de 5000 rotations chacun. En outre, il faut résoudre les problèmes de plus courts chemins dans ces réseaux des centaines de fois. Or, un bloc ne contient jamais plus de 10 rotations.

L'objectif de cette maîtrise est alors d'accélérer les temps de calcul en réduisant la taille de ces réseaux d'un facteur 10 avec des techniques d'apprentissage machine. Grâce aux données historiques, la machine peut estimer la probabilité qu'une rotation soit dans le bloc d'un employé dans un mois futur. À partir de ces informations, on pourra extraire, pour chaque employé, les 500 rotations les plus probables et donner seulement celles-ci aux réseaux correspondants. Les réseaux réduits ainsi obtenus seront ensuite étendus progressivement, au fur et à mesure des itérations de la génération de colonnes, pour finalement considérer les réseaux entiers, garantissant ainsi l'optimalité. Le rôle de l'apprentissage machine est donc de fournir des informations pertinentes à un optimiseur exact de blocs mensuels.

Les caractéristiques d'apprentissage sont les suivantes : les préférences et qualifications des employés, ainsi que les vols et les qualifications requises des rotations. Quelques améliorations

du modèle classique d'apprentissage profond en classification sont présentées : une méthode de pondération de l'entropie croisée qui favorise la minimisation du rappel, ainsi que deux méthodes de sur-échantillonnage qui visent à rééquilibrer l'ensemble de données. L'une répète les échantillons de la classe minoritaire, et l'autre en génère des nouveaux par interpolation. À chaque fois, trois architectures de perceptron multicouche ont été testées, deux optimiseurs ont été envisagés, et plusieurs valeurs d'hyper-paramètres ont été explorées.

La mesure de performance utilisée est calculée par rapport à la solution dont nous disposons, alors qu'il en existe de nombreuses autres tout aussi bonnes quoique très différentes. Ainsi, la performance est sous-évaluée par rapport à celle qui considérerait plusieurs de ces solutions équivalentes. Idéalement, la performance serait en fait calculée par rapport à une nouvelle solution, donnée par la génération de colonnes qui intégrerait les résultats de l'apprentissage. Ce n'était pas envisageable pour des raisons pratiques, mais c'est assurément un prolongement possible de ces travaux.

Après avoir sélectionné les cas où l'ensemble des rotations apparaissaient au moins une fois dans les réseaux réduits des employés, les expériences effectuées donnent une performance de 37% dans le meilleur cas. Pour des raisons pratiques, l'intégration des résultats d'apprentissage dans la génération de colonnes n'a pas pu être réalisée dans le cadre de cette maîtrise. Ceci fera l'objet d'un projet futur, et on espère obtenir des meilleurs résultats en utilisant une mesure de performance plus adéquate à l'optimisation.

ABSTRACT

Airline companies are under a heavy pressure to lower their costs. Therefore, they put a lot of effort into their airline planning optimization, and especially into the crew scheduling optimization, since the onboard staff is their second biggest expense after fuel.

The framework of this project is the aircrew rostering problem, which consists of building the monthly schedules of all the crew members by assigning pairings in a personalized way. A pairing is a sequence of flights, starting and ending at the same airport. A well-known and efficient way to solve this problem is the column generation algorithm, used by the AD OPT company, which collaborated on this project. In this method, a restricted master problem minimizes the total cost on a subset of possible blocks, while ensuring the coverage of all the pairings. On the other hand, sub-problems generate new monthly blocks, which will improve the solution of the restricted master problem. Column generation is an iterative and exact algorithm that implicitly considers all the possible blocks without enumerating them.

In this case, a sub-problem is a shortest path problem in a network that contains all the possible pairings (a path represents a block). There is one sub-problem by employee. The data provided by AD OPT relates to a real-world airline company, that has around 2000 employees and 5000 pairings each month. Therefore, there are 2000 networks containing 5000 pairings each. However, a block never has more than 10 pairings.

Therefore, the purpose of this thesis is to accelerate the computational time by reducing the network sizes by a factor of 10 with machine learning techniques. Based on historical data, one can estimate the probability of a pairing being in the block of an employee in the future. From this information, one will be able to extract the 500 most likely pairings for each employee, and give only these to the corresponding networks. The resulting reduced networks will be progressively extended when the columns generated with the current reduced networks seem to be insufficient, and in the end, the whole networks will be considered to guarantee optimality. Thus, the role of machine learning is to provide relevant information to an exact optimizer of the crew rostering problem.

The learning features are: the employees' wishes and qualifications, together with the pairings' flights and required qualifications. We present a few improvements to the standard deep learning model used in classification: a method that weighs the cross-entropy loss function in a way that favors recall minimization, as well as two sampling methods, that aim to rebalance the dataset. The first one repeats minority samples, and the other one artificially generates new ones by interpolation. In each of these methods, three perceptron architec-

tures are tested, two optimizers are considered and several values of hyper-parameters are explored.

The selected performance measure is computed with respect to the solution at disposal, whereas many very different, but just as good solutions exist. Consequently, the performance will always be under-evaluated in comparison to a performance that would consider many of these solutions. In fact, ideally, the performance would be computed with respect to a new solution provided by the column generation algorithm, that integrates the information provided by machine learning. For practical reasons, that wasn't realistic in this case, but it would definitely be part of the future work.

Once we selected the cases in which all the pairings appear at least once in the reduced networks, the best performance among the experiences carried out in this project is 37%. For practical reasons, the integration of the learnt results within the column generation algorithm couldn't be completed in this masters' thesis. This will be implemented in the future, and we hope to deliver better results by using a performance measure that is more relevant to the optimization point of view.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES SIGLES ET ABRÉVIATIONS

ML	Machine Learning
ReLU	Rectified Linear Unit
MLP	Multi Layer Perceptron
SGD	Stochastique Gradient Descent
BCE	Binary Cross Entropy
i.i.d.	Indépendantes et Identiquement Distribuées
CG	Column Generation
RMP	Restricted Master Problem
PCC	Problème de plus Court Chemin
SMOTE	Synthetic Minority Over-sampling Technique
AdaM	Adaptive Moments

CHAPITRE 1 INTRODUCTION

Le problème de planification des horaires d'équipages aériens est un domaine de recherche opérationnelle très étudié. En effet, l'industrie aérienne est très compétitive et les compagnies sont soumises à une forte pression pour baisser leurs prix. Pour cela, les compagnies mettent beaucoup de temps et d'effort dans l'optimisation des emplois du temps de leur personnel de bord car, après le carburant, c'est leur deuxième poste de dépense. Par ailleurs, ces problèmes d'optimisation d'horaires sont très complexes et soumis à de nombreuses contraintes. Il s'agit souvent de problèmes de couverture par ensembles (*set covering problems*) ou de problèmes de partition. En pratique, surtout chez les grandes compagnies aériennes, ces problèmes ne sont pas résolus de manière exacte.

Bien que ces problèmes aient été fortement étudiés, leur taille ne cesse d'augmenter et les compagnies aériennes requièrent de plus en plus des temps de calcul relativement courts. Ainsi, il y a une demande constante d'amélioration des techniques de résolution pour ces problèmes. C'est dans cette optique que nous proposons d'étudier l'utilisation d'outils d'apprentissage machine afin d'accélérer les algorithmes d'optimisation pour les problèmes d'horaires du personnel de bord des compagnies aériennes.

1.1 Définitions et concepts de base

1.1.1 Terminologie

Les termes présentés dans cette section sont essentiellement repris de Kasirzadeh et al. [33].

- Un segment de vol (ou *leg* en anglais) est caractérisé par : son code de vol, la date et l'heure de départ, la date et l'heure d'arrivée, l'aéroport d'origine et l'aéroport de destination.
- Une escale (ou *layover*) est un arrêt entre deux journées de travail.
- Une rotation (ou *pairing*) est une séquence de vols et d'escales. Une rotation commence et finit à la même base. À l'intérieur d'une rotation, les vols sont effectués par des avions de même gabarit et les vols ont les mêmes contraintes de langues et de qualifications.
- Une base est un grand aéroport auquel un employé se rattache (souvent celui proche de son lieu de résidence). Un employé ne fait que des rotations qui commencent et terminent à sa base.
- Un bloc mensuel (*monthly block*) ou un horaire mensuel (*monthly schedule*) est un

emploi du temps d'un employé sur un mois. C'est une séquence de rotations et de jours de congé (*days off*). On ignorera toutes les autres sortes d'activité dans le cadre de ce projet.

1.1.2 Planification aérienne

De manière générale, la planification aérienne est souvent divisée en plusieurs étapes pour des raisons de taille et de complexité. Ces étapes sont généralement traitées successivement : quand une étape est résolue, on considère sa solution comme étant fixée pour résoudre l'étape suivante [33]. Il existe des techniques intégrant plusieurs de ces étapes [8, 9, 26, 29, 33, 46, 47, 50, 52, 53, 54], notamment dans les problèmes de petites et moyennes tailles, mais elles dépassent le cadre de ce mémoire. Cette section reprend des informations de Desaulniers et Soumis [13].

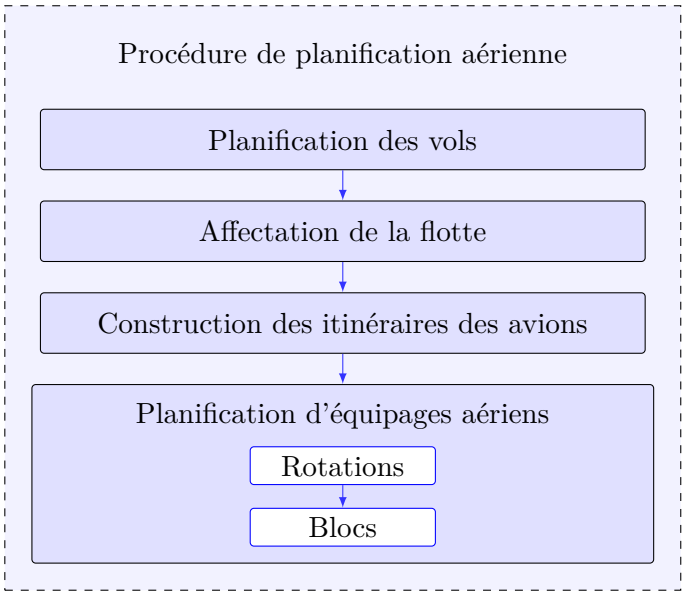


Figure 1.1 Procédure de planification aérienne

La Figure 1.1 présente les différentes étapes de la planification aérienne et reprend une figure de Kasirzadeh et al. [33]. La première étape est la planification des vols (*flight scheduling*), qui consiste à décider des vols à effectuer pour maximiser le profit prévisionnel : quel trajet (départ et arrivée), quelle date et quelle heure ? La deuxième étape est l'affectation de la flotte (*fleet assignment*). Il s'agit de décider des types d'avion qui vont effectuer les vols planifiés à l'étape d'avant, par exemple des Airbus A380 (capacité d'environ 500 passagers) ou des Airbus A320 (environ 150 passagers). Les flots par type d'avion doivent être conservés sur l'ensemble des vols tout en respectant le nombre d'avions disponibles par type. La troisième

étape est la construction des itinéraires des avions (*aircraft routing*) qui consiste à décider des trajets de chacun des avions physiques de la compagnie. En effet, après l'affectation de la flotte, à un moment et à un aéroport donnés, il peut y avoir plusieurs avions du même type en attente d'un autre vol. Il faut alors décider des trajets de chaque avion. La maintenance des avions est aussi intégrée dans cette étape.

La quatrième étape est celle de la planification d'équipages aériens (*crew scheduling*) où sont décidés les horaires du personnel navigant. C'est une étape sensible et cruciale car c'est celle qui peut diminuer le plus les coûts. Elle doit satisfaire beaucoup de contraintes, qui concernent majoritairement les préférences des employés, les compétences requises sur les vols, les normes de sécurité ainsi que les normes imposées par les conventions collectives.

On distingue dans cette étape deux problèmes qui sont aussi résolus successivement : d'abord le problème des rotations (*pairing*) qui se charge de créer les rotations en respectant les contraintes propres aux rotations. Puis celui des blocs mensuels, aussi appelé le *rostering problem* ou encore le problème d'affectation d'équipages (*crew assignment problem*) qui se charge de créer les blocs de tous les employés en considérant les rotations comme étant fixées. Le projet présenté dans ce mémoire se concentre plus particulièrement sur le problème des blocs mensuels.

L'équipage aérien est divisé en plusieurs catégories, avec d'une part les pilotes et co-pilotes (agents de *cockpit*), et d'autre part les agents de cabine. Chaque rotation requiert un pilote et un co-pilote, mais requiert un nombre variable d'agents de cabine suivant la taille de l'avion. Il y a plusieurs postes parmi les agents de cabine, sachant que certains employés ont les compétences pour plusieurs de ces postes. Les agents de *cockpit* sont planifiés indépendamment des agents de cabine, car les coûts et les contraintes sont différents. Les agents de cabine sont planifiés successivement par catégorie. Le sujet de ce mémoire concerne la catégorie des agents de bord (*flight attendants*), qui est la plus importante d'un équipage aérien. Elle est d'ailleurs optimisée presque en dernier. Chaque membre d'équipage a un aéroport de base, mais puisque les problèmes de blocs mensuels sont séparables par base, on considérera dans toute la suite qu'il n'y a qu'une seule base.

Deveci et Demirel [20] présentent une étude approfondie de la littérature dans la planification d'équipages aériens.

1.1.3 AD OPT et la génération de colonnes

AD OPT est une division de IBS, qui est une entreprise spécialisée dans le domaine de la planification d'équipages aériens. Ils utilisent pour cela la génération de colonnes qui est

une méthode pour résoudre ces problèmes. Grâce à la collaboration d'AD OPT, ce mémoire portera sur une étude de cas réel d'une grande compagnie du Moyen-Orient, avec environ 2000 agents de bord et 5000 rotations par mois. Une rotation dure environ 4 jours et un bloc a environ 6 rotations. L'objectif du problème des blocs mensuels est de couvrir ces 5000 rotations avec ces 2000 employés à coût minimal et satisfaction des employés maximale.

Le problème des blocs mensuels se formule comme un problème de couverture par ensembles. AD OPT utilise une méthode appelée la génération de colonnes pour le résoudre [24, 25, 37]. Intuitivement, pour formuler un problème de couverture, on utilise une variable de décision binaire pour chaque employé et chaque rotation, qui vaut 1 si l'employé effectue la rotation et 0 sinon. On a alors deux grands types de contraintes : des contraintes de flots qui s'assurent que les employés ont un horaire cohérent, et des contraintes de couverture. Une autre formulation possible est d'utiliser une variable binaire pour chaque bloc possible, qui vaut 1 si le bloc est effectué par un employé et 0 sinon. On se débarrasse alors des contraintes de flots qui sont considérées comme "faciles". Cependant, cela augmente considérablement le nombre de variables car il faut énumérer tous les blocs possibles pour minimiser le coût, sous contraintes de couverture. La relaxation linéaire de ce problème est appelée le problème maître (ou *master problem*). Or, on sait que dans une solution, très peu de ces variables seront utilisées. Seule une faible portion de l'ensemble des blocs possibles sera sélectionnée dans la solution finale. La génération de colonnes est un algorithme pour résoudre ce type de programme linéaire contenant un très grand nombre de variables. C'est un algorithme itératif qui permet de résoudre le problème maître sur un sous-ensemble de variables seulement. On l'appelle le problème maître restreint (*restricted master problem*).

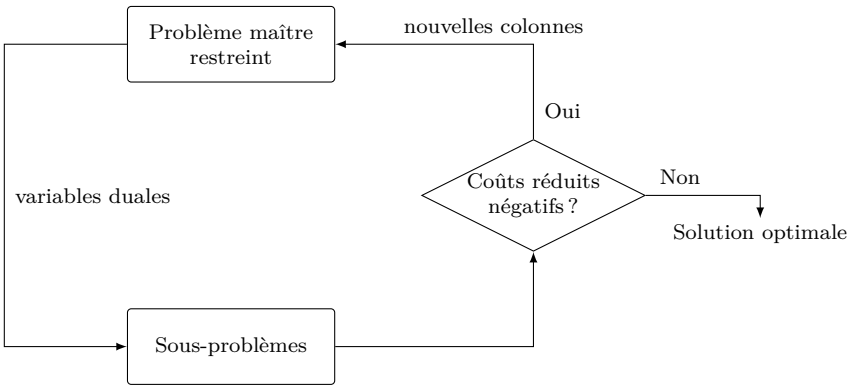


Figure 1.2 Génération de colonnes

La Figure 1.2 présente le principe de la génération de colonnes et reprend une figure de [13]. Initialement, un sous-ensemble de variables est choisi. Le coût est minimisé sur ce sous-

ensemble de variables (problème maître restreint). On extrait de cette solution les variables duales associées aux contraintes de couverture. Ces informations sont données à ce qu'on appelle le sous-problème, qui se charge de générer des nouvelles variables (i.e. des nouveaux blocs mensuels à considérer) qui vont améliorer la solution du problème maître restreint. Le sous-problème consiste en fait à trouver des variables non sélectionnées ayant des coûts réduits négatifs. Ces nouvelles variables sont ajoutées au sous-ensemble de variables initial, puis on résout à nouveau le problème maître restreint, et cela se répète. Lorsqu'il n'existe pas de variable ayant un coût réduit négatif, la solution optimale du problème maître restreint est aussi optimale pour le problème maître. On parle de génération de colonnes car les blocs mensuels sont représentés par des colonnes dans la matrice des contraintes du problème maître, et le rôle du sous-problème est de générer des *bonnes* colonnes pour le problème maître restreint.

Ainsi, la génération de colonnes fait des aller-retours entre le problème maître restreint et le sous-problème, et considère implicitement tous les blocs possibles.

Dans la génération de colonnes, le sous-problème doit être simple, connu et rapide à résoudre. Dans le cas des blocs mensuels, le sous-problème est un problème de plus court chemin (PCC) avec des contraintes de ressources dans un graphe acyclique contenant toutes les rotations possibles, où un chemin représente un bloc.

La génération de colonnes est une méthode souvent appliquée dans les problèmes de tournées de véhicules et de planification d'équipages [15, 16, 17].

1.1.4 L'apprentissage profond

L'apprentissage profond est un cas particulier de l'apprentissage machine. L'apprentissage machine consiste à utiliser des grandes bases de données pour en extraire de l'information généralisable à d'autres données inconnues du même type et réaliser une tâche. On distingue deux phases : l'apprentissage et le test. L'apprentissage profond consiste à chercher une fonction fortement non-linéaire qui reproduit au mieux les résultats des données connues. Cette fonction appartient à une famille de fonctions choisie par l'utilisateur, souvent représentée par un graphe ayant plusieurs couches. Le terme d'apprentissage profond vient de la profondeur de ces graphes. Le test consiste à mesurer la performance du modèle sur des données inconnues de la phase d'apprentissage. L'objectif est de minimiser l'erreur de généralisation. Un modèle d'apprentissage profond générique est présenté plus en détails dans la section 2.1.

1.2 Éléments de la problématique

1.2.1 Les sous-problèmes dans la génération de colonnes

Dans la génération de colonnes, comme les blocs sont générés de manière personnalisée (chaque employé a des préférences et des qualifications différentes), il y a en fait un sous-problème par employé. Autrement dit, pour chacun des 2000 employés, on résout un PCC avec contraintes de ressources dans un graphe contenant les 5000 rotations possibles. Or on sait qu'un bloc ne peut pas avoir plus de 10 rotations. Ainsi, le PCC va prendre un temps conséquent avant de se rapprocher d'une bonne solution, puisque le graphe est bien plus grand que la taille d'un chemin.

Par ailleurs, la génération de colonnes ne prend pas en compte les résultats des mois précédents, alors qu'ils sont assez semblables, tant au niveau des rotations qu'au niveau des employés. En effet, à chaque nouveau mois, l'optimiseur considère qu'il fait face à un problème complètement nouveau. Il commence par affecter des blocs un peu aléatoirement et met du temps avant de converger vers une solution mensuelle correcte.

L'idée directrice de ce projet est d'accélérer les calculs en réduisant beaucoup la taille des graphes des sous-problèmes. L'objectif fixé est de sélectionner pour chaque employé, un premier réseau réduit contenant 500 rotations, donc 10 fois plus petit qu'à l'origine. Celui-ci sera ensuite étendu progressivement : 1000 rotations, puis 3000 par exemple, pour finalement tous les sélectionner lorsqu'on est proche de l'optimum. Ainsi, les itérations de la génération de colonnes seront beaucoup plus courtes au début, vu que les graphes seront plus petits, et dès que les variables duales du problème maître restreint seront meilleures et que l'optimum sera suffisamment proche, les graphes entiers avec les 5000 rotations seront considérés. Il faut bien sûr mettre les rotations les plus pertinentes en premier, et c'est l'apprentissage machine, grâce aux données du passé, qui déterminera quelles rotations sont les plus probables pour chaque employé.

1.2.2 Contraintes globales

La sélection des 500 meilleures rotations pour chaque employé doit satisfaire des contraintes globales. D'une part, un employé doit avoir un réseau réduit contenant des rotations uniformément réparties dans le mois, et d'autre part, chaque rotation doit apparaître dans les réseaux réduits de suffisamment d'employés. En effet, il peut y avoir des rotations indésirables pour tous les employés, et si on accorde à chaque employé uniquement les rotations qu'il souhaite, certaines rotations peuvent ne jamais apparaître dans les réseaux réduits. Dans ce cas, les blocs générés par les sous-problèmes ne pourront pas former une solution réalisable pour

le problème maître. Ainsi, il faut que les rotations apparaissent toutes au minimum une fois parmi tous les réseaux réduits.

1.3 Objectifs de recherche

L'objectif pratique de cette recherche est d'utiliser les horaires d'équipages du passé pour fournir une fonction F donnant pour chaque rotation et chaque employé, une probabilité que la rotation soit dans le bloc de l'employé. Le but de cette fonction est d'être intégrée dans l'algorithme de génération de colonnes d'AD OPT pour accélérer les calculs à l'aide des réseaux réduits. Puisque vers la fin des itérations, on donne toutes les rotations possibles à tous les employés dans les sous-problèmes, les garanties de convergence de l'algorithme de génération de colonnes sont conservées. Le rôle de l'apprentissage s'en tient à apporter de l'information pertinente pour guider l'optimisation.

1.4 Plan du mémoire

La suite de ce mémoire comporte trois chapitres. Le premier présente un modèle d'apprentissage profond générique ainsi que les travaux d'intégration d'apprentissage en recherche opérationnelle existants dans la littérature. Le deuxième chapitre analyse les données fournies par l'entreprise AD OPT. Le dernier chapitre présente les différents modèles utilisés, leur implémentation et leurs résultats. Ce mémoire se conclut par une synthèse des travaux effectués et des perspectives de travaux futurs.

CHAPITRE 2 REVUE DE LITTÉRATURE

Ce chapitre comporte deux sections. La première présente un modèle d'apprentissage profond générique et explique les notions d'apprentissage supervisé, de classification, de perceptron multicouche, de fonction de perte, d'optimiseur, de rétropropagation du gradient, de généralisation, de régularisation et de mesure de performance. La deuxième présente l'état de l'art dans le domaine de l'intégration des techniques d'apprentissage en recherche opérationnelle, et notamment dans le domaine de la planification d'équipages aériens.

2.1 Modèle d'apprentissage profond générique

Ce qui est expliqué dans cette section est très largement repris du livre d'apprentissage profond Goodfellow et al. [27] ainsi que de Murphy [48] et Bishop [5].

2.1.1 Apprentissage supervisé

On peut distinguer trois grands domaines en apprentissage machine (ou *machine learning* (ML)) : l'apprentissage supervisé, l'apprentissage non-supervisé et l'apprentissage par renforcement. Dans l'apprentissage supervisé, en plus des éléments x en entrée, qui sont des réalisations d'une variable aléatoire X, on dispose d'un label (ou cible) y pour chaque x. Dans l'apprentissage non-supervisé, on ne dispose pas de labels. Le but est alors le plus souvent d'approcher la distribution de probabilité de la variable aléatoire X, ou encore de regrouper les données en *clusters*. Dans l'apprentissage par renforcement, la machine évolue dans un environnement qui peut prendre différents états. À chaque état, plusieurs actions sont possibles. Chaque action définit le passage vers un autre état et une récompense y est associée. La récompense est donnée par la fonction de récompense que l'utilisateur doit définir. La machine va apprendre à interagir avec l'environnement pour maximiser sa récompense totale. Plus de détails sur l'apprentissage par renforcement sont présentés par Sutton et al. [57].

L'apprentissage supervisé consiste à observer de nombreux exemples de valeurs d'entrée x associés à des labels y, pour pouvoir ensuite prédire la valeur du label suivant les valeurs en entrée. L'appellation "supervisé" vient de l'image d'un professeur qui indique la bonne réponse (i.e. le label) à l'élève (i.e. la machine) qui apprend. C'est l'apprentissage supervisé qui sera utilisé dans ce projet.

2.1.2 Problème de classification

L'apprentissage machine est souvent utilisé dans les problèmes de classification pour lesquels on dispose déjà d'un ensemble de données classifiées. Plus précisément, on dispose de plusieurs exemples de réalisations x d'une variable aléatoire X ainsi que des labels y associés à chaque x, où la valeur de y indique l'appartenance de x à une certaine classe. S'il existe K classes, y prendra, par exemple, la valeur d'un entier entre 0 et $K-1$. Ainsi, à un x donné, y est la réalisation d'une variable aléatoire $Y|X=x$ qui suit une loi de probabilité discrète avec K valeurs possibles pour y. En réalité, on devrait noter $y(x)$ au lieu de y, mais par souci de simplicité, on gardera y. L'objectif des problèmes de classification est de pouvoir prédire la valeur de y suivant la valeur de x, et plus exactement de donner les valeurs des probabilités conditionnelles $P(y=k|X=x)$ pour tous les $k \in [\![0, K-1]\!]$ et toutes les valeurs x de la variable aléatoire X.

L'exemple standard de classification en apprentissage machine est la reconnaissance de chiffres manuscrits. On dispose pour cela d'un grand nombre d'images de chiffres manuscrits. Un exemple x décrit alors les valeurs de chaque pixel de l'image associée. Chaque image est labellisée du chiffre qu'elle représente : il y a $K=10$ classes, et $y \in [\![0, K-1]\!]$. Le but est donc de donner pour chaque image de chiffre manuscrit, les probabilités que ce soit un 0, un 1, ..., un 9.

Dans toute cette partie, on se placera dans le cadre de la classification binaire, car c'est ce qui sera utilisé plus tard dans le chapitre 4. Ainsi $K=2$, et comme $\sum_{k=0}^{K-1} P(y=k|X=x)=1$, on cherche seulement une seule valeur de probabilité : $P(y=1|X=x)$, qui sera notée $P(y=1|x)$ ou encore $F(x)$ pour simplifier, l'autre étant $1-F(x)$.

Un modèle usuel dans ces problèmes de classification est le réseau de neurones profond à propagation avant et entièrement connecté (*feedforward fully connected deep neural network*). Cette section a pour but de présenter cette modélisation de base, qui sera reprise en grande partie ou en totalité par d'autres modèles présentés dans le chapitre 4.

2.1.3 Le perceptron multicouche

Le perceptron multicouche (ou *multilayer perceptron* (MLP)) est un modèle basique d'apprentissage profond, très souvent utilisé en pratique. Il est aussi appelé réseau de neurones profond à propagation avant (*deep feedforward neural network*). L'appellation de "propagation avant" vient du fait que l'information va circuler en avant, c'est-à-dire en commençant par la valeur d'une entrée x, puis en passant par les couches cachées, pour arriver à la dernière couche (la couche de sortie) qui doit prédire la valeur de $F(x)=P(y=1|x)$.

Le perceptron multicouche est souvent représenté par un graphe. Par exemple, pour un réseau à 3 couches, on a 3 fonctions $f^{(1)}$, $f^{(2)}$, $f^{(3)}$, qui s'appliquent successivement sur x pour former $f(x) = f^{(3)}(f^{(2)}(f^{(1)}(x)))$. Le but de f est de se rapprocher le mieux possible de F.

Les couches 1 et 2 sont appelées couches cachées, et la dernière couche est appelée couche de sortie. Il y a également une couche d'entrée, située avant la 1re couche cachée, qui est constituée de l'entrée x tout simplement. La couche de sortie donne la valeur de $f(x)$ pour un x donné. Dans le cas de la classification binaire, $f(x)$ est un scalaire entre 0 et 1 qui a pour but de refléter la probabilité d'être dans la classe 1. Le nombre de couches est appelé la profondeur du réseau, qu'on notera D.

Chaque couche cachée comporte des neurones, aussi appelés unités cachées (*hidden units*), qui représentent les états intermédiaires de la propagation de l'information depuis la couche d'entrée. Le nombre d'unités cachées dans une couche est appelé la largeur de la couche. Chaque couche comporte aussi des paramètres. Dans un cas simple, on a $f^{(i)}(z) = g^{(i)}(W^{(i)}z + b^{(i)})$ pour $i \in \{1, 2, 3\}$, les $g^{(i)}$ étant des fonctions non-linéaires, appelées fonctions d'activation, les $W^{(i)}$ étant des matrices de poids, et les $b^{(i)}$ étant des vecteurs de biais. Les fonctions d'activation sont généralement des fonctions qui s'appliquent terme à terme sur un vecteur. La recommandation usuelle du choix de la fonction d'activation des couches cachées est l'unité de rectification linéaire (*rectified linear unit*, ou ReLU), définie par $g(z) = \max\{0, z\}$ où z est un vecteur et $\max\{0, z\}$ est le vecteur z dont on a mis toutes les composantes négatives à 0. D'autres fonctions d'activation sont possibles, comme la sigmoïde $\sigma(z) = \dfrac{1}{1 + e^{-z}}$, ou la tangente hyperbolique $\tanh(z) = \dfrac{1 - e^{-2z}}{1 + e^{-2z}}$, et bien d'autres encore. La fonction ReLU semble être la plus simple, mais c'est surtout elle qui donne, en pratique, les meilleurs résultats. On notera que si toutes les fonctions d'activation étaient linéaires, comme par exemple la fonction identité, le modèle serait globalement linéaire, puisque ce serait une composition de plusieurs fonctions linéaires. L'intérêt des fonctions d'activation réside dans le fait qu'elles permettent de modéliser une fonction f potentiellement fortement non-linéaire, tout en étant différentiable presque partout.

Exemple

Prenons un exemple (voir Figure 2.1) : l'entrée est de taille 64, la 1re couche cachée comporte 6 neurones, la 2e 4, et la couche de sortie est scalaire (toujours dans le cas de la classification binaire). Toutes les fonctions d'activation sont des fonctions ReLU. On a alors :

— x une entrée de taille 64×1 ;
— $h^{(1)} = f^{(1)}(x) = \max\{0, W^{(1)}x + b^{(1)}\}$, où $W^{(1)}$ est de taille 6×64, et $b^{(1)}$ est de taille

6×1 ;

— $h^{(2)} = f^{(2)}(h^{(1)}) = \max\{0, W^{(2)}h^{(1)} + b^{(2)}\}$ où $W^{(2)}$ est de taille 4×6, $b^{(2)}$ est de taille 4×1 ;

— $f^{(3)}(h^{(2)}) = \sigma(W^{(3)}h^{(2)} + b^{(3)})$ où $W^{(3)}$ est de taille 1×4, et $b^{(3)}$ est de taille 1×1 ;

— $f(x) = f^{(3)}(f^{(2)}(f^{(1)}(x)))$ scalaire entre 0 et 1, noté aussi $\hat{y}$.

La fonction sigmoïde σ a été utilisée pour la couche de sortie car elle permet de remettre un scalaire entre 0 et 1 (pour que ce soit une valeur de probabilité) tout en étant une fonction très régulière. Sa courbe représentative est présentée dans la Figure 2.2.

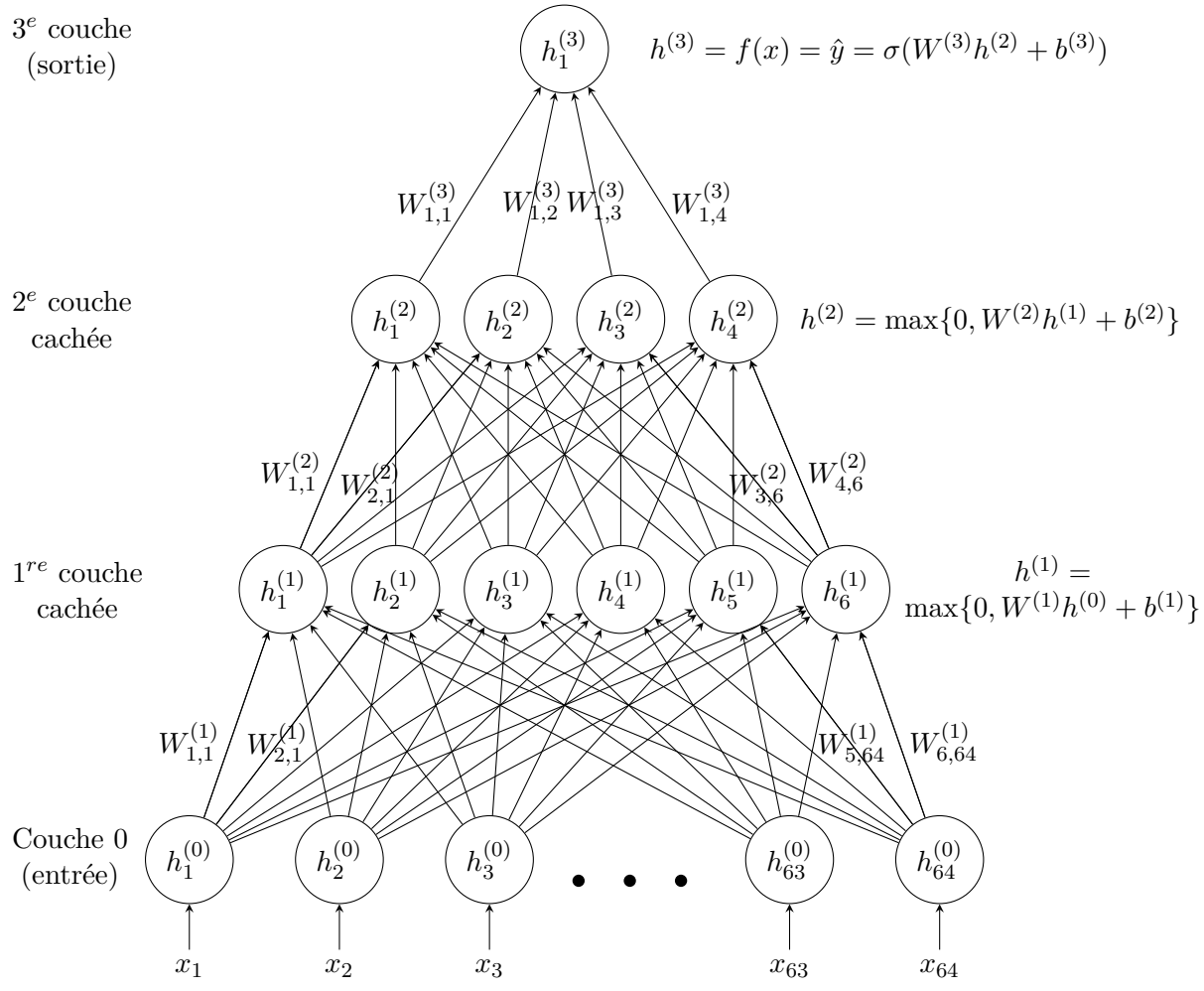


Figure 2.1 Exemple de perceptron multicouche

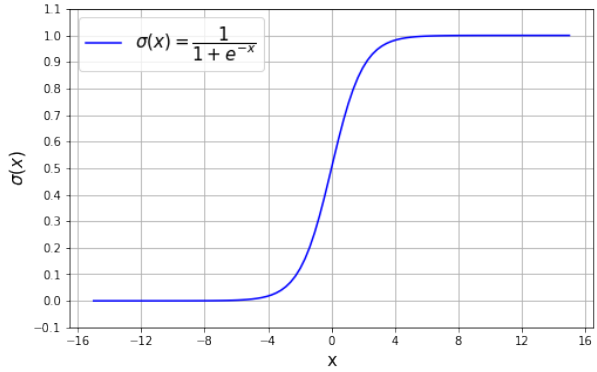


Figure 2.2 Courbe représentative de la fonction sigmoïde

Chaque neurone de chaque couche cachée correspond à une composante de l'état intermédiaire en sortie de la couche. Par exemple, les 6 neurones de la couche 1 correspondent aux 6 composants de $h^{(1)}$ et les 4 neurones de la couche 2 aux 4 composants de $h^{(2)}$. La notation $h^{(i)}$ pour les couches cachées vient du mot *hidden*. Par extension, on notera aussi $h^{(0)} = x$ et $h^{(3)} = f(x)$.

La notation $\hat{y}$ pour la valeur de sortie $f(x)$ est souvent utilisée car $\hat{y}$ doit globalement estimer le label y (même si y est entier alors que $\hat{y}$ est réel).

L'appellation "entièrement connecté" (ou *fully connected*) est utilisée quand tous les neurones d'une couche sont reliés à tous les neurones de la couche suivante. En effet, il y a un arc entre un neurone de la couche $i \in [\![0, D-1]\!]$ et celui de la couche $i+1$ si la valeur associée à l'un influe sur la valeur associée à l'autre. Traditionnellement, on note les poids $W^{(i)}_{k,j}$ sur l'arc $h^{(i)}_j$ – $h^{(i+1)}_k$ (voir Figure 2.1).

Le but de l'apprentissage sera alors d'apprendre les bonnes valeurs des paramètres du modèle, notés $\theta = \left\{W^{(i)}, b^{(i)}\right\}_{i \in [\![1,D]\!]}$, pour approcher la fonction $F : x \mapsto P(y = 1|x)$. Dans l'exemple précédent, la taille de θ vaut $6 \times 64 + 6 + 4 \times 6 + 4 + 1 \times 4 + 1 = 423$.

La profondeur du réseau et les largeurs de chaque couche sont des hyper-paramètres. Le bon choix d'hyper-paramètres n'est pas universel et dépend de la structure du problème considéré.

Lorsqu'il n'y a pas de couche cachée et que la fonction d'activation de la couche de sortie est une sigmoïde, on parle de régression logistique. C'est en fait le réseau de neurones le moins profond possible.

La structure globale d'un perceptron multicouche (on parle d'architecture) est en fait assez simple : c'est une succession d'applications linéaires et de fonctions non-linéaires simples (comme la fonction ReLU). L'idée sous-jacente est qu'une succession d'opérations simples

peut très bien approcher des fonctions complexes, tout comme des neurones simples forment un cerveau humain. Dans la reconnaissance d'images par exemple, dans le cas où on veut prédire s'il y a une voiture dans l'image à partir des pixels (qui est une opération assez complexe pour une machine, quoique simple pour l'humain), les neurones de la première couche vont détecter des éléments basiques comme les bords ou les contours, ceux de la deuxième des éléments un peu moins basiques comme des formes (circulaires, angulaires, droites, etc.), puis ceux de la couche suivante des éléments plus complexes comme des parties de voiture (roues, capot, fenêtres, etc.) pour finalement, à la dernière couche, donner une prédiction de la présence d'une voiture ou non dans l'image.

D'autres architectures sont également très largement utilisées dans la littérature, les deux exemples les plus connus étant les réseaux de neurones récurrents ou RNN [51], utilisés le plus souvent pour les données séquentielles comme des signaux temporels, et les réseaux de neurones convolutifs ou CNN [39, 40], créés à la base pour traiter des données ayant une structure de grille similaire à celle des images, très performants pour la reconnaissance d'image.

2.1.4 La fonction de perte

La fonction de perte permet d'évaluer l'erreur commise par f par rapport à la fonction recherchée $F : x \mapsto P(y = 1|x)$, qui est toutefois inconnue. On n'a accès qu'au label y associé à x pour un échantillon donné de valeurs de x. En effet, pour un x donné, y est seulement une réalisation de la loi de Bernoulli (dans le cas des labels binaires) caractérisée par $P(y = 1|x)$. Toutefois, en appliquant la fonction de perte entre la sortie $\hat{y} = f(x)$ et le label y à de nombreux exemples et en en calculant la moyenne, statistiquement, on se rapproche de l'espérance de l'erreur réelle entre $f(x) = \hat{y}$ et $F(x)$. C'est donc ceci qu'on va faire à la sortie du perceptron multicouche.

Théorie de l'information

Une fonction de perte intuitive est l'erreur quadratique moyenne, mais en apprentissage machine, on utilise couramment l'entropie croisée binaire dans les problèmes de classification binaire. La raison de ce choix s'explique par la théorie de l'information [10, 44]. En effet, lorsqu'on calcule l'erreur quadratique moyenne, on accorde peu d'importance à l'erreur commise sur des événements rares. Or, en apprentissage, intuitivement, apprendre qu'un événement rare s'est réalisé donne plus d'information que d'apprendre qu'un événement très probable s'est réalisé. Par ailleurs, l'information qu'apporte la réalisation de deux événements indépendants est la somme des informations qu'ont apportées chacun des événements. Cela nous

amène à définir la fonction $I(z) = -\log[P(z)]$ qui représente l'information apportée par l'événement $Z = z$, où Z est une variable aléatoire et z est une de ses réalisations. On peut ensuite définir l'entropie de Shannon $H(P)$ d'une distribution P de la variable aléatoire Z comme étant l'information moyenne qu'apporte une réalisation de Z suivant la distribution P : $H(P) = -\mathbb{E}_{z\sim P}[\log(P(z))]$. Puis, on peut définir une distance entre deux distributions de probabilités P et Q, appelée la divergence de Kullback-Leibler :

$$D_{KL}(P||Q) = \mathbb{E}_{z\sim P}\left[\log \frac{P(z)}{Q(z)}\right] \tag{2.1}$$

$$= \mathbb{E}_{z\sim P}[\log P(z) - \log Q(z)] \tag{2.2}$$

$$= -H(P) - \mathbb{E}_{z\sim P}[\log Q(z)] \tag{2.3}$$

L'entropie croisée $H(P, Q)$ est définie comme étant la divergence de Kullback-Leibler sans son terme d'entropie de Shannon :

$$H(P, Q) = -\mathbb{E}_{z\sim P}[\log Q(z)] \tag{2.4}$$

Dans le cas de l'apprentissage, P est la vraie distribution de probabilité d'une variable aléatoire, et Q correspond à l'approximation qu'on en fait. Ainsi choisir Q pour minimiser l'entropie croisée revient à minimiser la divergence de Kullback-Leibler, ce qui revient à choisir Q le plus proche possible de P.

L'entropie croisée binaire ou *binary cross entropy (BCE)*

Si on revient au cas de la classification binaire en apprentissage machine, on veut approcher F avec f. Comme on n'a accès qu'aux valeurs des labels y pour certaines valeurs de x, la fonction de perte qu'on va utiliser est :

$$Loss(f) = -\frac{1}{N}\sum_{j=1}^{N}\left[y^{(j)} \log f(x^{(j)}) + (1 - y^{(j)}) \log\left[1 - f(x^{(j)})\right]\right] \tag{2.5}$$

ou encore :

$$Loss(\hat{y}) = -\frac{1}{N}\sum_{j=1}^{N}\left[y^{(j)} \log \hat{y}^{(j)} + (1 - y^{(j)}) \log\left(1 - \hat{y}^{(j)}\right)\right] \tag{2.6}$$

avec N le nombre total d'exemples de valeurs de x et y, les $x^{(j)}$ et $y^{(j)}$ pour $j \in [\![1, N]\!]$

les différents exemples dans l'ensemble de données de taille N, et $\hat{y}^{(j)} = f(x^{(j)})$. On prendra comme convention : $0 \log 0 = 0$. Cette fonction de perte est aussi appelée la log-vraisemblance négative (*negative log-likelihood*). Choisir les paramètres du modèle qui maximisent la vraisemblance (i.e. qui minimisent l'entropie croisée) garantit le meilleur estimateur asymptotiquement, lorsque le nombre d'exemples de données tend vers l'infini. Autrement dit, c'est un estimateur consistent [27]. Par ailleurs, c'est aussi un estimateur efficace statistiquement, dans le sens où il est l'estimateur consistent ayant la plus petite erreur quadratique moyenne. C'est pour ces raisons que l'entropie croisée est souvent utilisée en apprentissage machine.

Plus de détails sur la théorie de l'information, sur l'entropie croisée binaire et sur l'estimateur du maximum de vraisemblance sont présentés dans le livre de Goodfellow et al. [27].

2.1.5 L'optimiseur

L'optimiseur usuel en apprentissage machine est la descente de gradient stochastique ou *stochastique gradient descent* (SGD) [6]. À chaque itération, les valeurs des paramètres qu'on souhaite optimiser se déplacent d'un pas dans la direction de plus forte pente de la fonction de perte. La direction du pas est donnée par le gradient de la fonction de perte par rapport aux paramètres du modèle θ. La taille du pas est déterminée par ce gradient et par la valeur d'un hyper-paramètre : la vitesse d'apprentissage ou le *learning rate* ou *lr* en anglais, noté ε, un scalaire positif. Le pas de descente est alors égal au gradient multiplié par ce scalaire.

L'aspect stochastique de cette méthode vient des *minibatchs*. En effet, pour des raisons de mémoire, il n'est pas possible de stocker toutes les données d'entraînement (ainsi que les gradients associés) en une seule fois. Il peut arriver que les données d'entraînement contiennent des milliers ou des millions d'exemples. Pour pallier ce problème, on divise l'ensemble des données en *minibatchs*, d'une taille fixe m', qui peuvent être composés de quelques dizaines d'exemples chacun, jusqu'à environ une centaine. Les paramètres sont alors mis à jour entre chaque *minibatch*, et le gradient est calculé seulement sur les exemples du *minibatch*. On a alors uniquement une estimation du gradient.

Pour un *minibatch* $\mathbb{B} = \{x^{(1)}, x^{(2)}, ..., x^{(m')}\}$ de taille m', et en notant $\hat{y}^{(1)}, \hat{y}^{(2)}, ..., \hat{y}^{(m')}$ les prédictions correspondantes, la mise à jour du gradient se fait suivant les équations suivantes :

$$\hat{Grad} = \frac{1}{m'} \sum_{j=1}^{m'} \nabla_\theta Loss(\hat{y}^{(j)})$$

$$et \qquad \theta \leftarrow \theta - \varepsilon \, \hat{Grad}$$

où on note $Loss(\hat{y}^{(j)}) = -\left[y^{(j)} \log \hat{y}^{(j)} + (1 - y^{(j)}) \log\left(1 - \hat{y}^{(j)}\right)\right]$ par commodité, ce qui représente la contribution d'un échantillon dans l'équation (2.6) avant de diviser par N.

L'avantage de la descente de gradient stochastique est que le temps de chaque itération de *minibatch* ne dépend pas du nombre d'exemples dans l'ensemble de données, mais seulement de la taille des *minibatchs*. Par ailleurs, même s'il n'y a pas de garantie de convergence vers un minimum global, ou même local, en un temps raisonnable, dans la pratique, la descente de gradient stochastique donne de bons résultats assez rapidement.

Les recommandations usuelles sont :

— de mélanger les données avant de choisir les *minibatchs* pour éviter que l'ordre des données n'influe sur la qualité de l'estimateur du gradient,

— d'effectuer une partition des données en choisissant une taille de *minibatch* qui soit une puissance de 2 pour optimiser la mémoire sur la carte graphique lors de l'apprentissage,

— et de passer plusieurs fois par l'ensemble des données. Un passage par toutes les données s'appelle une époque.

Ainsi, si on a 100 exemples et qu'on choisit une taille de *minibatch* de 16, on obtient 7 *minibatchs*, sachant que le dernier est seulement de taille 4. Par abus de langage, les *minibatchs* sont souvent appelés des *batchs*.

L'Algorithme 1 présente l'algorithme de la descente de gradient stochastique. Comme les prédictions $\hat{y}$ dépendent des paramètres θ, on utilisera les notations $Loss(\theta)$ et $Loss(\hat{y})$ indifféremment.

Algorithme 1: Descente de gradient stochastique

Paramètre: *vitesse d'apprentissage* ε

Initialiser θ

Tant que *critère d'arrêt non vérifié* **faire**

  Génération d'un *minibatch* $\mathbb{B} = \{x^{(1)}, x^{(2)}, ..., x^{(m')}\}$

  Calculer $\hat{y}^{(1)}, \hat{y}^{(2)}, ..., \hat{y}^{(m')}$, les prédictions correspondantes

  Calculer $Loss(\theta)$, l'erreur du modèle sur le *minibatch* $\mathbb{B}$

  Calculer son gradient $\hat{Grad} = \nabla_\theta Loss(\theta)$ sur le *minibatch* $\mathbb{B}$

  Mettre à jour $\theta : \theta \leftarrow \theta - \varepsilon \hat{Grad}$

fin

2.1.6 La rétropropagation du gradient

À chaque exemple x, on peut propager l'information à travers toutes les couches jusqu'à la sortie $\hat{y} = f(x)$: c'est la propagation en avant, comme vu dans l'exemple de la section

2.1.3, qui s'effectue à la ligne 4 de l'Algorithme 1. Ceci étant fait sur tous les exemples du *minibatch*, on peut calculer l'erreur commise par le modèle $Loss(\theta)$ sur le *minibatch* (ligne 5 de l'Algorithme 1). L'étape suivante est de calculer le gradient de la fonction de perte par rapport à θ (ligne 6 de l'Algorithme 1).

L'algorithme de rétropropagation du gradient (*backpropagation*, ou *backprop*) [51] permet de calculer ce gradient de manière efficace en propageant les informations sur le gradient depuis la couche de sortie jusqu'à la 1re couche cachée. Les algorithmes de propagation avant et de rétropropagation sont présentés ci-dessous dans les Algorithmes 2 et 3 [27]. Pour des raisons de lisibilité, les algorithmes décrivent les opérations sur un seul exemple (x, y). Il suffira ensuite d'appliquer la propagation avant à tous les exemples du *batch* courant, et de faire la moyenne sur tous les exemples du *batch* dans les calculs de gradient de la rétropropagation. On prendra les mêmes notations que dans la section 2.1.3 :

- x l'entrée, y son label,
- D le nombre de couches,
- $\forall i \in [\![1, D]\!], (W^{(i)}, b^{(i)})$ les paramètres de poids et de biais de la couche i,
- $\forall i \in [\![1, D]\!], g^{(i)}$ les fonctions d'activation (non-linéaires) de la couche i,
- $Loss$ la fonction de perte,
- $h^{(0)} = x$,
- $\forall i \in [\![1, D]\!], h^{(i)} = g^{(i)}(W^{(i)}h^{(i-1)} + b^{(i)})$ l'état intermédiaire de la couche i,
- $\hat{y} = h^{(D)}$ la prédiction de sortie,
- et aussi $\forall i \in [\![1, D]\!], a^{(i)} = W^{(i)}h^{(i-1)} + b^{(i)}$,
  et donc $\forall i \in [\![1, D]\!], h^{(i)} = g^{(i)}(a^{(i)})$.

L'opérateur $\odot$ représente la multiplication terme à terme.

Algorithme 2: Propagation avant

1 **Pour** $i = 1, 2, ..., D$ **faire**
2 $\quad a^{(i)} = W^{(i)}h^{(i-1)} + b^{(i)}$
3 $\quad h^{(i)} = g^{(i)}(a^{(i)})$
4 **fin**
5 $\hat{y} = h^{(D)}$
6 Calculer la perte : $Loss(\hat{y})$

Algorithme 3: Rétropropagation du gradient

$grad \leftarrow \nabla_{\hat{y}} Loss(\hat{y})$
Pour $i = D, D-1, ..., 1$ **faire**
  ** $grad$ vaut $\nabla_{h^{(i)}} Loss$ **
  $grad \leftarrow \nabla_{a^{(i)}} Loss = grad \odot g^{(i)\prime}(a^{(i)})$ // car $h^{(i)} = g^{(i)}(a^{(i)})$
  $\nabla_{b^{(i)}} Loss = grad$ // car $a^{(i)} = W^{(i)} h^{(i-1)} + b^{(i)}$ et $grad = \nabla_{a^{(i)}} Loss$
  $\nabla_{W^{(i)}} Loss = grad \times h^{(i-1)\mathsf{T}}$ // pour les mêmes raisons
  $grad \leftarrow \nabla_{h^{(i-1)}} Loss = W^{(i)\mathsf{T}} \times grad$ // pour les mêmes raisons
fin

L'algorithme de rétropropagation se base sur le théorème de dérivation des fonctions composées. Le cas de la rétropropagation est un peu plus simple que le cas général, car les fonctions utilisées sont soit des applications linéaires, soit des fonctions scalaires appliquées terme à terme sur les composants d'un vecteur (d'où la multiplication terme à terme avec $\odot$).

On remarque qu'on a remonté les couches du réseau depuis la sortie $\hat{y}$ jusqu'à la couche d'entrée. De cette façon, les calculs de gradient utilisent les informations sur les gradients précédents sans devoir les recalculer. Ainsi, le temps de calcul de la rétropropagation est du même ordre de grandeur que celui de la propagation avant, c'est-à-dire que calculer $Loss(\theta)$ prend un temps proportionnel au calcul de $\nabla_\theta Loss$. Cette méthode du calcul du gradient est en fait un cas particulier de différentiation automatique [2].

2.1.7 La généralisation

Le but de l'apprentissage machine est d'appliquer ce que la machine a appris à la prédiction de données futures et inconnues. On appelle cela la généralisation. On divise alors l'ensemble des données en deux : une partie pour l'apprentissage, et l'autre pour tester la capacité du modèle à généraliser. On appelle erreur d'entraînement l'erreur du modèle sur les données d'entraînement, et de la même manière, on parle d'erreur de test sur les données de test. Ces erreurs sont calculées à partir de la fonction de perte. L'optimiseur (comme la descente de gradient stochastique) s'occupe de minimiser l'erreur d'entraînement, mais contrairement à l'optimisation classique, on accorde une très grande importance à ce que l'erreur de généralisation soit minimale également.

Les données de test seront totalement cachées pendant la phase d'apprentissage pour pouvoir approcher au mieux l'erreur de généralisation. En effet, l'erreur de test est biaisée par rapport à l'erreur de généralisation puisqu'elle est calculée sur les données limitées dont on dispose. La vraie erreur de généralisation impliquerait une infinité d'exemples de test. Par abus de

langage, les deux termes sont utilisés indistinctement. Souvent, l'ensemble de test représente 20% des données totales, et l'ensemble d'entraînement 80%, le tout étant choisi aléatoirement.

Sur-apprentissage et sous-apprentissage

Il y a en réalité deux enjeux : minimiser l'erreur d'entraînement et minimiser la différence entre celle-ci et l'erreur de test. Ces deux enjeux correspondent à deux écueils : le sur-apprentissage (ou *overfitting*) et le sous-apprentissage (*underfitting*). Minimiser seulement l'erreur d'entraînement conduit souvent à "apprendre les données d'entraînement par cœur" (i.e. très faible erreur d'entraînement) et à un grand écart avec l'erreur de test, donc potentiellement une erreur de test importante. D'autre part, minimiser la différence entre les deux conduit à sélectionner aléatoirement un modèle pour ainsi avoir une différence nulle, mais une erreur importante sur les deux ensembles.

Pour pallier le problème de sous-apprentissage, il suffit de choisir un modèle qui couvre un espace suffisamment large de fonctions (on parle de capacité d'un modèle) et d'en optimiser les paramètres en fonction des données d'entraînement. Il suffira souvent de prendre un modèle avec beaucoup de paramètres.

Toutefois, les modèles ayant une trop grande capacité vont souvent conduire au sur-apprentissage. Pour éviter cela, il existe plusieurs techniques de régularisation. On appelle régularisation toute modification de l'algorithme d'apprentissage visant à diminuer l'erreur de généralisation mais pas l'erreur d'entraînement. Une méthode classique, appelée dégradation des pondérations ou *weight decay* en anglais, consiste à ajouter à la fonction de perte un terme qui pénalise les grandes valeurs des paramètres et favorise les matrices creuses, en utilisant des normes L^2 ou L^1 par exemple. Cela suit le principe du rasoir d'Ockham (*Occam's razor*) [60] qui est un principe de parcimonie selon lequel, parmi plusieurs modèles "équivalents" (équivalence que l'utilisateur va lui-même définir), il vaut mieux choisir le modèle le plus "simple".

On notera également que l'usage des *minibatchs* a aussi un effet de régularisation, car les *minibatchs* contiennent du bruit par rapport à la distribution de l'ensemble d'entraînement.

Arrêt anticipé ou *early stopping*

Une autre régularisation courante est l'arrêt anticipé ou *early stopping*. Pour cela, on enlève une partie de l'ensemble d'entraînement, qu'on appelle ensemble de validation. Cet ensemble servira à optimiser le nombre d'époques. Plus généralement, l'ensemble de validation est utilisé pour optimiser tous les hyper-paramètres du modèle. Ici, on ne considère que le nombre

d'époques. Ainsi, l'arrêt anticipé consiste à arrêter l'algorithme d'apprentissage lorsque l'erreur de validation augmente p fois consécutivement, où p est un entier qui définit la "patience".

L'algorithme d'arrêt anticipé est présenté dans l'Algorithme 4. À noter qu'en pratique, on se donne également un nombre maximal d'époques.

L'arrêt anticipé est une méthode de régularisation car des erreurs de validation consécutives de plus en plus importantes sont le signe de sur-apprentissage. Elle est couramment utilisée car elle est non-invasive : elle donne simplement un critère d'arrêt sans ajouter plus de calculs pendant la propagation avant, sans non plus ajouter de terme à la fonction de perte qui rajouterait des calculs dans le gradient.

Algorithme 4: L'arrêt anticipé ou *early stopping*

Paramètre: nombre de minibatchs par époque N^{batch}
Paramètre: fonction de perte $Loss$
Paramètre: vitesse d'apprentissage ε
Paramètre: patience p

Initialiser $\theta \leftarrow \theta_0$
$j = 0$ // Nombre d'augmentations consécutives de l'erreur de validation
$erreur\_precedente = \infty$
Tant que $j < p$ **faire**
  Pour $l = 1, 2, ..., N^{batch}$ **faire**
    Génération du l-ième *minibatch* $\mathbb{B}$
    Calculer l'erreur du modèle sur le *minibatch* $Loss_{(\mathbb{B})}(\theta)$ // Propagation avant
    Mise à jour $\theta : \theta \leftarrow \theta - \varepsilon \nabla_\theta Loss_{(\mathbb{B})}(\theta)$ // Rétropropagation du gradient
  fin
  Calculer l'erreur sur l'ensemble de validation $erreur\_courante$
  si $erreur\_courante < erreur\_precedente$ **alors**
    $j = 0$
  sinon
    $j = j + 1$
  fin
  $erreur\_precedente = erreur\_courante$
fin

2.1.8 Décrochage ou *dropout*

Le décrochage ou *dropout* en anglais [56], est aussi une technique courante pour éviter le sur-apprentissage. Cela consiste à appliquer (multiplier terme à terme) un masque binaire sur tous les nœuds du réseau de neurones pendant la propagation avant, où le masque est tiré aléatoirement et indépendamment entre chaque exemple de donnée, avec une probabilité définie par l'utilisateur pour chaque nœud. Ainsi certaines parties du réseau seront totalement ignorées, et entre chaque exemple, l'optimiseur va considérer un réseau de neurones différent. Les recommandations usuelles sont de donner une probabilité 0.5 d'être masqué aux unités cachées, et 0.2 pour les unités d'entrée. Ces masques permettent d'éviter d'apprendre les données "par cœur" puisque l'architecture change constamment entre chaque exemple de données. En fait, cela permet de s'entraîner sur plusieurs modèles, car implicitement tous les sous-réseaux du réseau initial sont considérés. Par ailleurs, comme chaque neurone est susceptible d'être masqué, les neurones doivent pouvoir s'adapter à ce manque d'information. Par conséquent, le réseau de neurones est plus robuste et évite le sur-apprentissage. Un autre avantage de cette méthode est que cela coûte peu cher à mettre en place dans l'algorithme : ni la propagation avant, ni la rétropropagation en sont grandement affectées. C'est pourquoi elle est souvent utilisée.

Le décrochage est un cas particulier d'un ensemble de méthodes appelées *bootstrap aggregation* ou *bagging* [27].

2.1.9 La mesure de performance

Jusqu'à présent, on a considéré que la fonction de perte évalue la qualité du modèle. On l'utilise alors pour calculer le gradient par rapport à tous les paramètres et dans le critère d'arrêt de l'arrêt anticipé (*early stopping*). Toutefois, dans bien des cas, la fonction de perte ne reflète pas la performance du modèle, souvent parce que la performance est difficile à calculer et non-dérivable. Parfois même, la mesure de performance d'un modèle se calcule grâce à une boîte noire extérieure et indépendante, avec un temps de calcul plus ou moins long. Or, la régularité de la fonction de perte est une caractéristique essentielle pour utiliser la descente de gradient stochastique, et la fonction de perte doit être rapide à calculer car elle est appelée très souvent. C'est pourquoi la vraie fonction qui mesure la performance d'un modèle peut être inutilisable pour optimiser les paramètres d'un modèle dans la descente de gradient. C'est en fait une caractéristique de l'apprentissage machine qui le distingue de l'optimisation classique : cette dernière se concentre uniquement sur la minimisation de la fonction de perte.

Ainsi, la descente de gradient minimise la valeur de la fonction de perte sur l'ensemble de test pour indirectement maximiser la performance du modèle. Cependant, on peut utiliser la vraie mesure de performance dans le calcul de l'erreur de validation dans l'arrêt anticipé : dès que la performance sur l'ensemble de validation cesse de s'améliorer, c'est un signe de sur-apprentissage donc on s'arrête et on sélectionne le meilleur modèle rencontré jusqu'à présent.

2.1.10 Vue générale d'un code d'apprentissage machine classique

Un code d'apprentissage machine se présente généralement comme dans l'Algorithme 5.

Algorithme 5: Pseudo-code classique en apprentissage

Paramètre: données d'entraînement de taille N
Paramètre: vitesse d'apprentissage ε
Paramètre: nombre de *minibatchs* par époque N^{batch}
Paramètre: modèle perceptron multicouche *modele*
Paramètre: fonction de perte BCE *Loss*
Paramètre: nombre maximal d'époques N^{epoque}
Paramètre: évaluation de la performance toutes les n^{perf} époques

Faire
  Pour $epoque = 1, 2, ..., N^{epoque}$ **faire**
    Pour $l = 1, 2, ..., N^{batch}$ `// Entraînement`
    faire
      Génération du l-ième *minibatch* $\mathbb{B}$
      Calculer les prédictions $\hat{y}^{(\mathbb{B})} = modele(\mathbb{B})$ `// Propagation avant`
      Calculer l'erreur du modèle sur le *minibatch* $Loss_{(\mathbb{B})}(\theta)$
      Calculer $\nabla_\theta Loss_{(\mathbb{B})}(\theta)$ `// Rétropropagation du gradient`
      Mise à jour $\theta : \theta \leftarrow \theta - \varepsilon \nabla_\theta Loss_{(\mathbb{B})}(\theta)$ `// Descente de gradient stochastique`
    fin
    si $epoque \mod n^{perf} = 0$ `// Test`
    alors
      Calculer la performance du modèle sur les données de validation : $perf$
    fin
  fin
tant que *perf ne vérifie pas le critère d'arrêt* `// Arrêt anticipé`

2.2 Intégration d'apprentissage machine en recherche opérationnelle

2.2.1 Contexte général d'intégration d'apprentissage machine en optimisation combinatoire

Bengio et al. [4] présentent une étude récente de l'apprentissage machine en optimisation combinatoire. Deux cas y sont distingués. Tout d'abord, il y a l'usage de l'apprentissage machine de bout-en-bout, c'est-à-dire que l'objectif de l'apprentissage est de résoudre complètement le problème d'optimisation. Par exemple, le problème du voyageur de commerce (*travelling salesman problem* ou TSP) a été beaucoup étudié en apprentissage, que ce soit avec des réseaux neuronaux récurrents [61] ou bien avec de l'apprentissage par renforcement [3, 18, 34]. L'avantage de l'apprentissage par renforcement est son aspect non-supervisé. Cela implique alors que l'apprentissage n'est pas limité par la performance d'un expert comme en apprentissage supervisé : il peut faire mieux. Néanmoins, cela nécessite de définir une fonction de récompense pertinente, ce qui peut être non trivial. Un désavantage de l'apprentissage machine de bout-en-bout est son aspect lourd à implémenter, quoique les auteurs mentionnent la possibilité de commencer par de l'apprentissage supervisé. Mais le plus gros désavantage est la perte des garanties de réalisabilité et d'optimalité qu'ont les algorithmes d'optimisation. En effet, la convergence en apprentissage machine est encore un domaine actif de recherche, et la réalisabilité d'une solution obtenue par apprentissage dépend de la qualité de la modélisation du problème dans le cadre de l'apprentissage. C'est un autre désavantage de l'apprentissage de bout-en-bout. En effet, des aspects simples en optimisation combinatoire, comme par exemple une permutation, sont parfois complexes à modéliser avec des réseaux de neurones, où des propriétés de différentiabilité sont requises.

Ce qui est plus courant, et recommandé par les auteurs de cet article, est d'intégrer l'apprentissage à l'intérieur des algorithmes d'optimisation, soit en l'intégrant seulement au début, soit en y faisant des appels répétés, mais toujours en terminant par l'optimisation. Ainsi, l'algorithme global conserve certaines propriétés importantes de l'optimisation : si l'algorithme d'optimisation est exact, ce sera conservé, s'il ne propose que des solutions réalisables, ce sera également conservé. Le plus souvent, la partie apprentissage se fera de manière déconnectée de l'optimisation, c'est-à-dire que la phase d'apprentissage se fait au début avec des données disponibles au préalable, mais lors de l'intégration avec l'optimisation, la machine n'apprend plus et reste en l'état. L'apprentissage supervisé est courant dans ce cas, par exemple pour décider s'il est intéressant de faire une décomposition de Dantzig-Wolfe pour de l'optimisation linéaire en nombres entiers mixtes (*mixed integer linear programming* ou MILP) [38]. Un autre exemple est de prendre des décisions dans l'arbre de séparation et d'évaluation

(*branch and bound* ou B&B) pour des MILP. Cela peut être pour la sélection des nœuds de l'arbre [30] ou encore pour approximer des algorithmes de branchement très performants mais coûteux [35, 45]. Lodi et Zarpellon [42] donnent pour cela une étude des méthodes d'apprentissage pour les décisions de branchement. D'un point de vue plus appliqué, Lodi et al. [43] proposent d'intégrer des techniques d'apprentissage à l'optimisation combinatoire dans le problème d'emplacement d'installations (*facility location problem*), et Fischetti et Fraccaro [23] dans le domaine de la prédiction de la production d'un parc éolien offshore.

Par ailleurs, l'apprentissage supervisé utilise souvent une fonction de perte qui est déconnectée de l'algorithme d'optimisation. Mais une bonne performance d'apprentissage n'est pas équivalente à une bonne performance de l'intégration avec l'algorithme d'optimisation. On peut supposer que la première implique la deuxième. Toutefois, dans bien des cas, surtout quand il existe de nombreuses solutions équivalentes, un mauvais apprentissage peut tout de même conduire à une bonne solution globale. Il y a un écart entre ce que fait l'apprentissage, à savoir minimiser la fonction de perte, et ce qui compte vraiment : améliorer la performance de l'algorithme d'optimisation combinatoire.

2.2.2 Intégration de l'apprentissage machine dans le problème des rotations d'équipages aériens

Soumis et al. [55] présentent une méthode intégrant des techniques d'apprentissage dans l'optimisation des rotations en planification d'équipages aériens (l'étape juste avant celle des blocs dans la Figure 1.1).

Le problème des rotations se présente comme un problème de partition, où chaque vol doit être couvert par une rotation, avec de nombreuses contraintes de conventions collectives. La génération de colonnes et la séparation et l'évaluation sont souvent utilisées pour ces problèmes. Toutefois, la taille des problèmes augmente de plus en plus, et les temps de calcul sont devenus de plus en plus importants, que ce soit à cause du nombre d'itérations de la génération de colonnes, du temps consacré à une résolution du problème maître restreint, ou bien du nombre de nœuds dans l'arbre de séparation et d'évaluation. D'autres méthodes ont été alors introduites pour accélérer la génération de colonnes dans les problèmes de grande taille, notamment l'agrégation dynamique de contraintes de Elhallaoui et al. [21]. Cela consiste à regrouper les vols : on parle de groupes ou grappes, ou *clusters* en anglais. Les vols dans un groupe doivent avoir une forte probabilité de se suivre dans la même rotation, puis ces groupes sont donnés au problème maître qui va regrouper les contraintes concernant chacun des vols en une seule concernant le groupe. Les groupes pourront ensuite être brisés si besoin. Les auteurs de [21] montrent que les temps de calcul sont significativement réduits,

que le nombre d'itérations de la génération de colonnes ainsi que le nombre de variables fractionnaires dans les solutions des relaxations linéaires résolues dans l'arbre de séparation et d'évaluation diminuent.

C'est dans ce cadre d'agrégation dynamique de contraintes que Soumis et al. [55] interviennent : le but de l'apprentissage machine est de prédire, grâce aux données historiques, les vols ayant une forte probabilité de se suivre dans la même rotation. Autrement dit, il s'agit de créer les groupes. L'apprentissage est supervisé, et on considère en entrée les caractéristiques d'un vol pour prédire le suivant. Les vols antérieurs sont ignorés pour éviter la propagation d'erreurs. Les vols suivants envisagés sont sélectionnés après avoir appliqué un masque, qui s'assure que les vols suivants soient cohérents : ils doivent décoller dans les 48 heures suivant l'arrivée du vol courant, on n'en considère pas plus que 20, le type d'avion doit être le même, etc. Ces vols suivants sont ensuite triés dans l'ordre chronologique de départ, et l'apprentissage prédit le rang du vol suivant dans ce classement. C'est donc traité comme un problème de classification à 20 classes. Les auteurs utilisent un réseau neuronal convolutif, optimisent les hyper-paramètres, notamment avec la validation croisée, et obtiennent des résultats très satisfaisants. La précision du modèle d'apprentissage est très élevée, et lorsque ces résultats sont intégrés à la génération de colonnes et à l'agrégation dynamique de contraintes, les résultats sont d'aussi bonne qualité voire meilleurs, les temps de calcul diminuent significativement, le nombre d'itérations de la génération de colonnes et le nombre de nœuds dans la séparation et l'évaluation diminuent. La précision peut encore être augmentée grâce à des techniques d'abstention [62] qui permettent à la machine de s'abstenir de prédire au lieu de mal prédire.

Ainsi, l'apprentissage machine est ici utilisé pour donner de l'information pertinente (i.e. les groupes) à un optimiseur exact du problème des rotations (i.e. la génération de colonnes et l'agrégation dynamique de contraintes), qui se charge ensuite de traiter le problème dans les détails (notamment avec les contraintes et les coûts exacts). On conserve alors toutes les garanties d'optimalité et de réalisabilité données par l'optimiseur initial. Pour accélérer la résolution des problèmes de recherche opérationnelle, Soumis et al. [55] valorisent donc l'intégration de l'apprentissage machine au début seulement, de la même manière que Bengio et al. [4].

CHAPITRE 3 ANALYSE DE DONNÉES

Ce chapitre présente les données reçues de la part d'AD OPT contenant les données réelles d'une grande compagnie aérienne du Moyen-Orient. Il y a 7 types de fichiers qu'on va utiliser dans ce projet. Chaque fichier contient les informations sur un mois. On dispose des données sur 3 mois : septembre 2018, octobre 2018, novembre 2018. Donc il y a en tout 21 fichiers. La plupart des informations dans ce chapitre concernent un seul mois, les autres mois ayant des caractéristiques très similaires. Toutes les analyses de données de cette partie ont été réalisées avec Python et la librairie `pandas`. Dans ce chapitre, certaines données confidentielles ont été cachées, et parfois remplacées par des sigles arbitraires dans un ordre aléatoire.

3.1 Les solutions

La Figure 3.1 présente la table de données des solutions.

	Employee	Position	Assigner	Activity Type	Pairing Code	Start DateTime	End DateTime
0	E_A		2	DO	LVE	2018-09-27 20:00:00	2018-09-28 20:00:00
1	E_B		2	DO	LVE	2018-09-28 20:00:00	2018-09-29 20:00:00
2	E_C		2	DO	LVE	2018-09-29 20:00:00	2018-09-30 20:00:00
3	E_D		2	DO	OFF	2018-09-30 20:00:00	2018-10-01 20:00:00
4	E_E		2	TRP	AAA	2018-10-02	2018-10-04
5	E_F		2	DO	OFF	2018-10-04 20:00:00	2018-10-05 20:00:00

Figure 3.1 Extrait de la table de données des solutions de novembre 2018

La table de données des solutions (pour le mois de novembre ici) a 84791 lignes, où chaque ligne correspond à un poste dans une rotation (les valeurs de *Position* sont uniques). Dans ce mois, il y a 1861 employés (caractérisés par les numéros d'employé uniques). Il y a également 5 types d'activité : DO pour *day off* (jour de congé), TRP pour *trip* (une rotation), GND pour *ground* (activité au sol), RSV pour *reserve* (activité de réserve), et TRN pour *training* (activité de formation). Nous allons seulement nous intéresser aux activités de type TRP. Le *Pairing Code* correspond au code de la rotation, qui n'est pas unique. Ce qui caractérise une rotation de manière unique est le code de rotation avec la date et l'heure de départ. On remarque également que les activités contenues dans le fichier du mois de novembre ne se déroulent pas uniquement en novembre. En fait, dans ce cas-ci, les activités s'étalent du 27 septembre au 15 décembre, probablement pour des raisons internes au fonctionnement de l'entreprise, sûrement pour traiter les chevauchements de certaines rotations sur deux mois

consécutifs. Nous allons nous concentrer sur les activités du mois de novembre uniquement. La colonne *Assigner* contient un entier, qui signifie lorsqu'il est positif que l'activité a été pré-assignée par le solveur (c'est le cas des activités qui se passent en septembre et octobre entre autres). Nous ne traiterons que les activités ayant un *assigner* négatif.

Après avoir appliqué tous ces filtres, on obtient pour le mois de novembre des données de solutions avec 10170 lignes, 1778 employés, 195 codes de rotation et 5082 rotations uniques. Les rotations sont réparties uniformément dans le mois et durent en moyenne un jour et demi. La rotation la plus courte dure 5 heures et la plus longue dure plus de 5 jours. Chaque employé a en moyenne 6 rotations (qui ne sont pas pré-assignées) dans son bloc mensuel, au minimum 1 et au maximum 12. Les mois de septembre et d'octobre ont des caractéristiques quasi-identiques.

3.2 Les employés

La Figure 3.2 présente la table de données du fichier des employés après en avoir extrait les informations les plus importantes.

Employee ID	Employee	Qualifications	Languages
245	E_A	[1051, 1079, 1121, 1171, 1172, 169, 31, 330, 3...	[]
246	E_B	[1051, 1079, 1121, 1171, 1172, 169,	[L1]
247	E_C	[1051, 1079, 1121, 1171, 1172, 169,	[L5]
248	E_D	[1051, 1079, 1121, 1171, 1172, 169, 320, 321, ...	[L3, L6]
249	E_E	[1051, 1079, 1121, 1171, 1172, 169, 320, 321, ...	[L2]

Figure 3.2 Extrait de la table de données des employés de novembre 2018

Il y a en tout 1861 employés qui parlent 58 langues différentes, avec une moyenne de 1.4 langues parlées par employé (en plus de l'anglais qui est parlé par tous). Il y a 118 qualifications différentes, avec en moyenne 19 qualifications par employé. Les Figures 3.3 et 3.4 décrivent les langues et qualifications les plus courantes parmi les employés.

Il y a deux types d'identificateur unique pour chaque employé. Celui de la colonne *Employee* est utilisé dans les fichiers des solutions et de l'équité, et celui dans la colonne *Employee ID* dans le fichier des préférences.

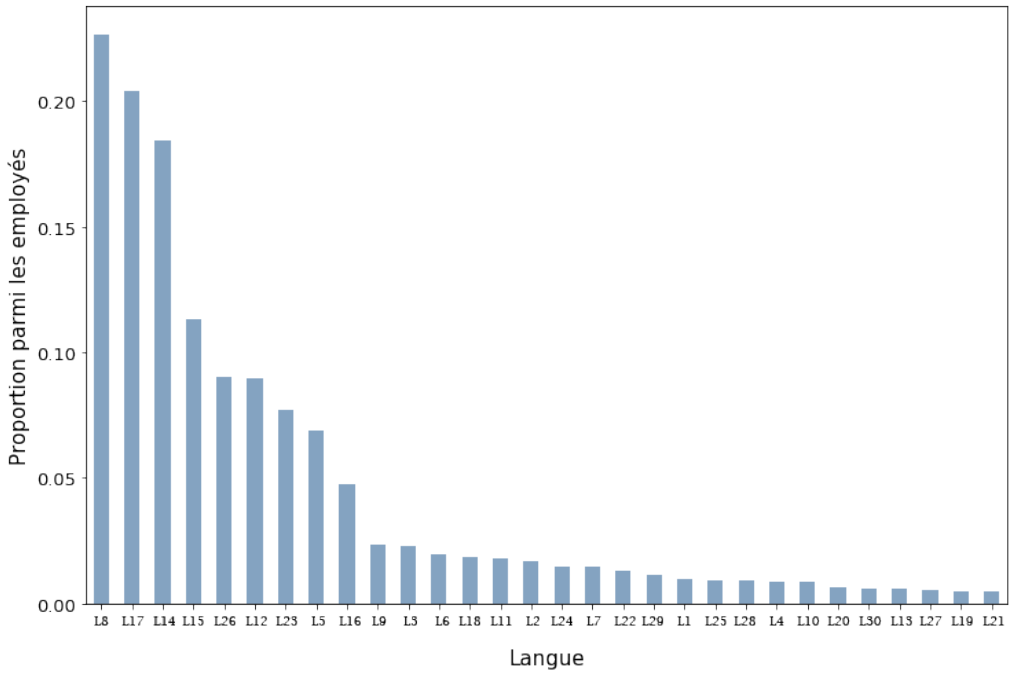


Figure 3.3 Proportion des employés parlant les 30 langues les plus courantes (sans compter l'anglais)

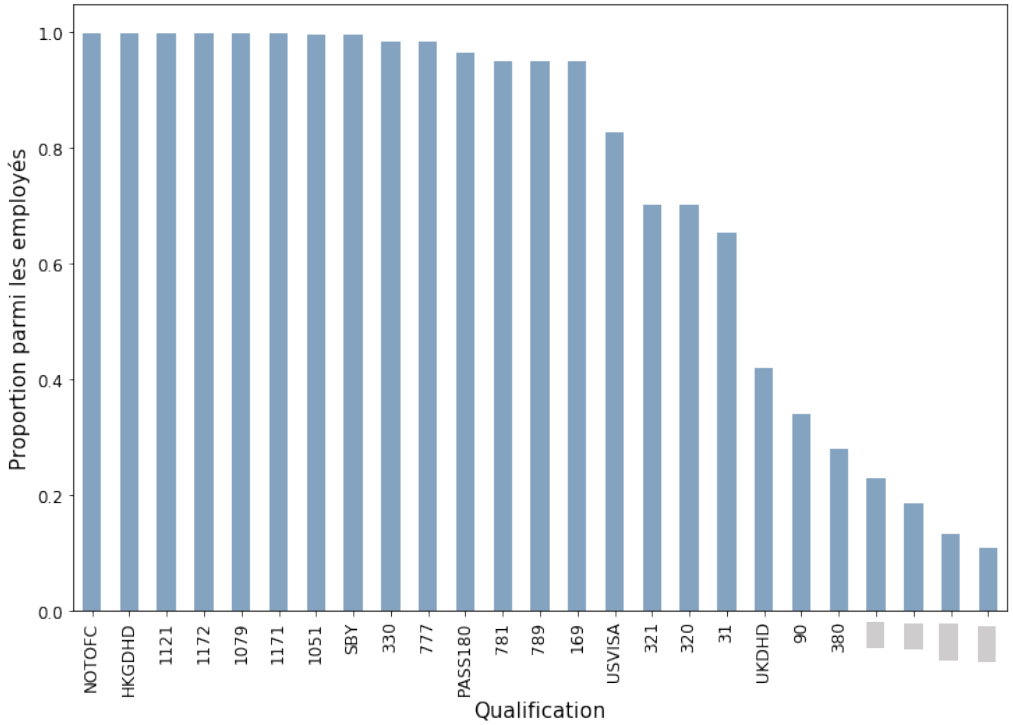


Figure 3.4 Proportion des employés ayant les 25 qualifications les plus courantes

3.3 Les activités

La Figure 3.5 présente le fichier des activités sous la forme d'une table de données.

	Activity Type	Pairing Code	Base	Start DateTime	End DateTime	Positions	Month Time Credit	Total Time Credit	Month Time Block	Total Time Block	Min Rest Before	Alt Rest Before	rgsz Extra Qual	Qualifications	Cap Reduce	Activity Attributes
0	TRP	"P1"		2018-11-01	2018-11-03	[443170667, 443170668, 443170669, 443170670, 4...	31:25	31:25	27:25	27:25	0:00	0:00	0	[1051, PASS180, USVISA]	0	[1051, PASS180, ULRU, ULR_ALL, USVISA]
1	TRP	"P2"		2018-11-02	2018-11-04	[443170686, 443170687, 443170688, 443170689, 4...	31:25	31:25	27:25	27:25	0:00	0:00	0	[1051, PASS180, USVISA]	0	[1051, PASS180, ULRU, ULR_ALL, USVISA]

Figure 3.5 Extrait de la table de données des activités de novembre 2018

Le fichier des activités contient 11407 lignes, où une ligne correspond à une activité. De la même manière que dans le fichier des solutions, il y a les 5 mêmes types d'activités et on va se concentrer uniquement sur les rotations TRP. La base est unique donc il n'y a pas besoin de séparer par base. De même qu'avant, les activités s'étalent sur deux mois et demi (au lieu d'un mois). Chaque rotation a une liste de qualifications, avec au total 16 types de qualification possibles. En moyenne, une rotation possède une qualification, sachant que 44% des rotations ne demandent aucune qualification. Le fichier contient également des informations sur les attributs des rotations. Le Tableau 3.1 décrit les différents attributs et qualifications possibles : en tout il y a 16 qualifications et 14 attributs possibles.

Tableau 3.1 Fréquence d'apparition des qualifications et des attributs des rotations dans le fichier des activités

Qualifications	**Nombre d'occurences**
90	3566
789	2718
1051	2147
PASS180	1008
31	770
USVISA	639
1079	380
169	128
XX	25
YY	18
ZZ	7
UKDHD	6
FJ	4
FB	1
CS	1
FC	1

Attributs	**Nombre d'occurences**
90	3566
789	2718
1051	2147
ULR_ALL	1632
ULRU	1627
PASS180	1008
31	770
USVISA	639
1079	380
169	128
XX	25
YY	18
ZZ	7
UKDHD	6

On remarque que seuls les attributs ULR_ALL et ULRU n'apparaissent pas dans les qualifications. Ce sont des attributs qui indiquent si la rotation contient des vols très long courrier (*ultralong range*), ce qui n'est pas une information pertinente dans notre cas. Par ailleurs, tous les autres attributs sont en fait déjà décrits dans les qualifications. Ainsi, seules les informations sur les qualifications seront retenues pour le projet. La signification des qualifications décrites dans le Tableau 3.1 sont inconnues, mais cela ne nuira pas dans la suite du projet.

Il y a au total 11361 rotations dans le mois de novembre, 268 codes de rotation et 22018 postes, avec une moyenne de 2 postes par rotation. À noter que toutes les rotations des solutions sont aussi dans les fichiers des activités.

Toutes les informations sur les différentes durées décrites dans ce fichier seront ignorées.

En croisant les informations contenues dans les fichiers des solutions, des employés et des activités, il apparaît que, dans la solution, les contraintes des qualifications dans les rotations sont presque toujours satisfaites. Plus précisément, dans le cas de novembre 2018, si on considère qu'une rotation sans qualification requise est satisfaite à 100%, alors dans 99.9% des cas, les rotations décrites dans la solution ont 100% de leurs contraintes de qualification satisfaites. Dans le dernier 0.1%, les rotations ont 50% ou 67% de leurs contraintes de qualification satisfaites.

3.4 Les rotations

La Figure 3.6 présente les informations sur une rotation sous la forme d'une table de données.

	Pairing Code	Start DateTime	End DateTime	Min Rest Before	Flown Time	Worked Time	Leg Code	Leg DateTime	Engine	Destination	Layover Station	Layover DateTime	Layover Duration	fDayOff
0	AAA	2018-11-01	2018-11-03	0 days	1 days	1 days	AB123	2018-11-01	380	YYZ	None	NaT	NaT	None
1	BBB	2018-11-01	2018-11-03	0 days	1 days	1 days	None	NaT	None	None	YYZ	2018-11-01	1 days	0
2	CCC	2018-11-01	2018-11-03	0 days	1 days	1 days	AC345	2018-11-03	380	JFK	None	NaT	NaT	None

Figure 3.6 Extrait de la table de données des rotations de novembre 2018

Dans cette table de données, une ligne correspond à un segment de vol ou à une escale, avec en tout 31401 lignes. Il y a donc les informations sur les codes de vol, leur date et heure de début, le type d'avion et leur aéroport de destination. De même, il y a les informations sur les aéroports d'escale, la date et l'heure de début, ainsi que la durée de l'escale. Les escales font en moyenne 28 heures, et il y a 5 types d'avion : 320, 789, 330, 777, 380. Il y a en tout 11363 rotations, 269 codes de rotation, 304 codes de vol, 57 aéroports d'escale et 88 aéroports de destination.

3.5 Les langues

La Figure 3.7 présente les données du fichier des langues sous la forme d'une table de données.

	Pairing Code	Language	Start Date	Demand	Other Cat	Cat Preassign	Remaining Req	Cat Target	Cat Coverage	Remain
0	P1	L1	2018-11-01	1	0	0	1	1	1	0
1	P2	L2	2018-11-01	2	0	0	2	0	1	1
2	P3	L3	2018-11-02	1	0	0	1	1	1	0

Figure 3.7 Extrait de la table de données des langues de novembre 2018

Ici, une ligne correspond à une langue requise dans une rotation, avec en tout 7444 lignes. Le Tableau 3.2 présente la signification des colonnes du fichier des langues ainsi que la distribution des valeurs possibles pour chaque colonne.

Tableau 3.2 Signification des différentes colonnes du fichier des langues

Colonne	**Signification**	**Valeurs possibles**
Demand	Nombre total de personnes requises pour la contrainte à travers toutes les catégories. Calculé comme étant le maximum sur tous les segments de vol de la rotation.	1 (90%), 2
Other Cat	Nombre de personnes couvrant la contrainte dans toutes les autres catégories.	0 (99%), 1 (1%), 2
Cat Preassign	Nombre de personnes pré-affectées couvrant la contrainte dans la catégorie courante	0 (100%), 1
Remaining Req	Nombre restant de personnes requises pour la catégorie courante = *Demand* − *Other Cat* − *Cat Preassign*	0 (89%), 1 (10%), 2
Cat Target	Cible calculée pour résoudre la catégorie courante pour cette contrainte	0 (16%), 1 (84%)
Cat Coverage	Nombre de personnes couvrant la contrainte dans la solution de la catégorie courante	0 (64%), 1 (32%), 2 (3%), 3, 4
Remain	Nombre de personnes requises pour la contrainte après la solution de la catégorie courante	0 (54%), 1 (40%), 2

Le fichier des langues décrit 5091 rotations qui s'étalent sur le mois de novembre seulement (pour le fichier de novembre), avec 197 codes de rotation uniques. 20 langues différentes sont demandées dans les rotations, sachant que L8 est quasiment toujours demandé (voir Figure 3.8). La demande d'une langue dans une rotation n'excède jamais 2. Elle vaut 2 pour 7 langues. Il y a 0.5% des cas où une contrainte de langue a été satisfaite par pré-affectation d'un membre d'équipage de la catégorie courante (colonne *Cat Preassign*). Il s'agira toujours de L8, et dans ces cas-là, la *Cat Target* sera égale à 0. Les affectations dans les catégories

de membre d'équipage précédentes couvrent une contrainte de langue dans 0.8% des cas (colonne *Other Cat*), dans lesquels il s'agira presque toujours de L8. La valeur de *Cat Target* est toujours plus faible que la valeur de *Remaining Req* pour des raisons de faisabilité.

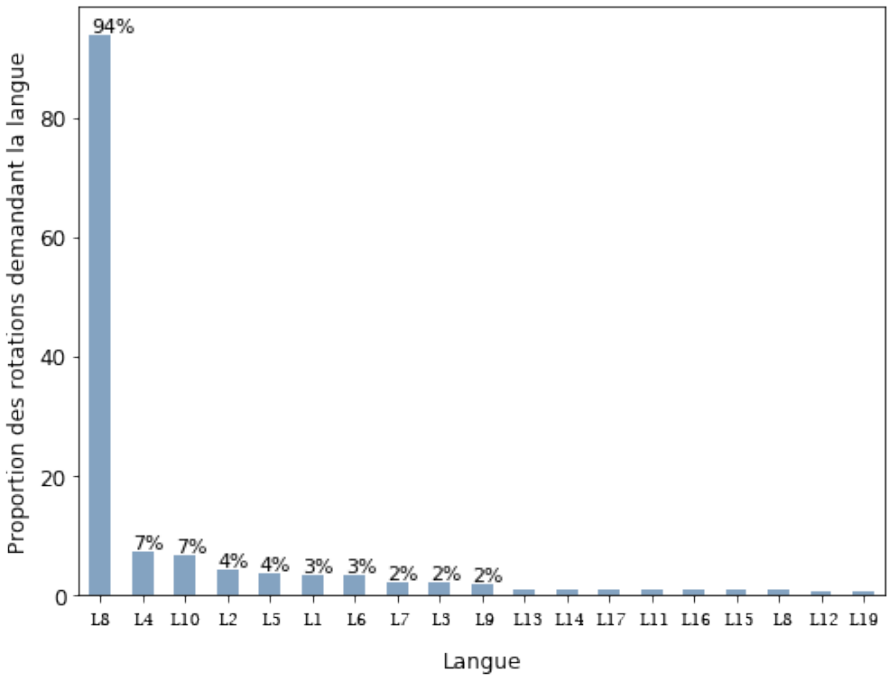


Figure 3.8 Proportion des rotations demandant chaque langue pour novembre 2018

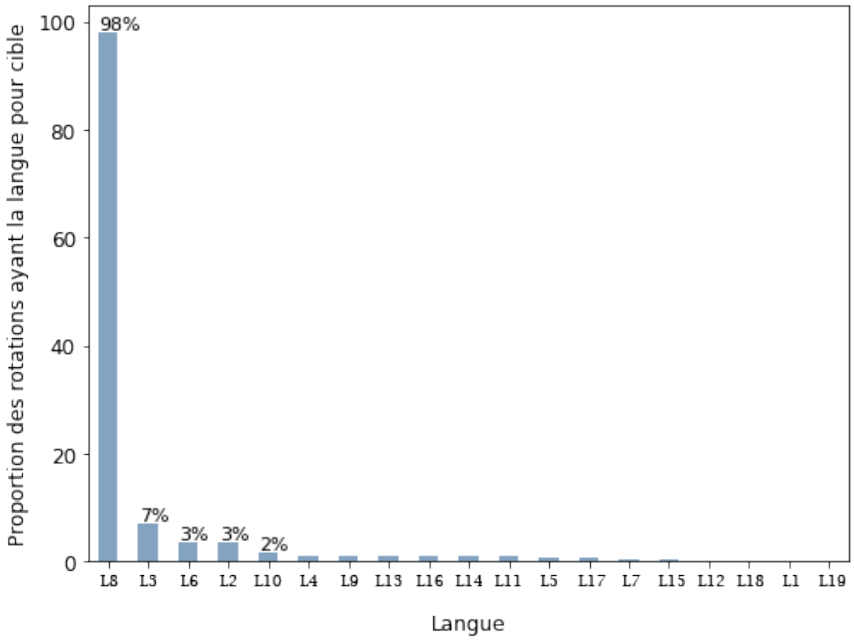


Figure 3.9 Proportion des rotations ciblant chaque langue pour novembre 2018

La colonne qui nous intéresse est *Cat Target*, car c'est celle qui sera utilisée lors de l'apprentissage. D'après le Tableau 3.2, dans 84% des cas, la cible sera égale à 1. La Figure 3.9 présente pour chaque langue la proportion des rotations qui la contiennent comme cible. Encore une fois, L8 est dominant.

La colonne *Cat Coverage* donne le nombre de personnes couvrant la contrainte dans la solution de la catégorie courante. Elle doit se rapprocher de la valeur de *Cat Target* dans le meilleur des cas.

Il y a plusieurs cas de sur-satisfaction de la demande. Dans la solution actuelle, si on considère que les contraintes sur-satisfaites sont satisfaites à 100%, et si on ignore les contraintes ayant des cibles nulles, le taux de couverture moyen d'une contrainte est de 40%. La Figure 3.10 présente les taux de couverture par langue. On remarque que les contraintes de langues ne sont pas toutes satisfaites après avoir résolu le problème des agents de bord. Les contraintes de langue sont couvertes à 42% en moyenne. Il est probable que les contraintes de langue soient trop restrictives car le personnel de bord n'est pas capable de toutes les satisfaire.

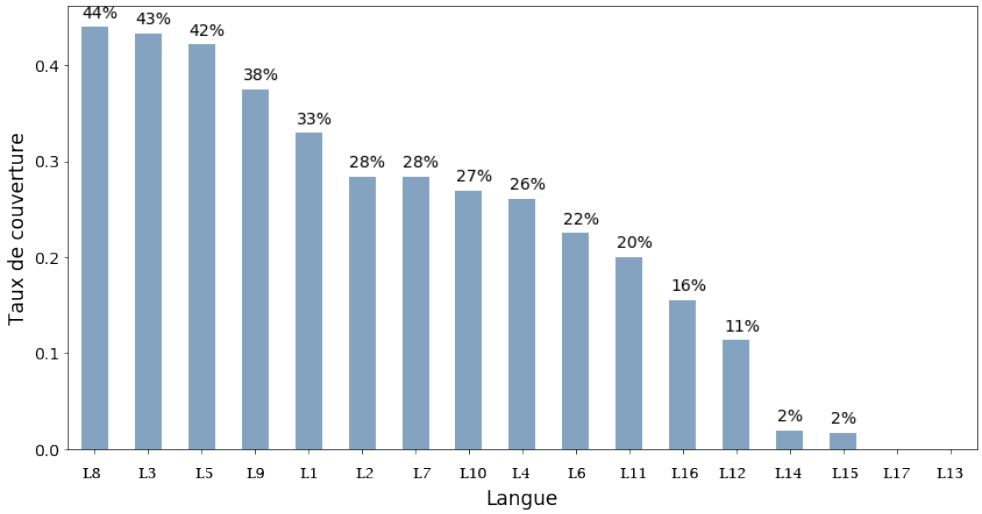


Figure 3.10 Taux de couverture par langue

3.6 Les préférences

La Figure 3.11 présente les données du fichier des préférences sous la forme d'une table de données.

Employee ID	Ladderlevel	Weight	Criterion Code	Wish Type	Relop	Layover Station	Layover Duration	Pairing Duration	Leg Code	Pairing Code	Start Date	End Date	Specific Date	WeekDay 1	WeekDay 2
132	1	10000	C216	AWARD	EQUAL	None	NaT	NaT	None	None	2018-11-	2018-11-	NaT	NaN	NaN
132	2	5500	C108	AWARD	EQUAL	None	NaT	NaT	None	None	2018-11-	2018-11-	NaT	NaN	NaN
132	3	1000	C216	AWARD	EQUAL	None	NaT	NaT	None	None	2018-11-	2018-11-	NaT	NaN	NaN
132	4	850	C216	AWARD	EQUAL	None	NaT	NaT	None	None	2018-11-	2018-11-	NaT	NaN	NaN
132	5	400	C216	AWARD	EQUAL	None	NaT	NaT	None	None	2018-11-	2018-11-	NaT	NaN	NaN

Figure 3.11 Extrait de la table de données des préférences de novembre 2018

Chaque ligne correspond à une préférence pour un employé. Il y a 8796 lignes, et il en reste 7867 après avoir fusionné les doublons (pour un mois). Le fichier contient les préférences de 1861 employés (décrit par leur *Employee ID*). Les préférences sont de 18 types (donnés par le *Criterion Code*), concernent 85 aéroports d'escale (colonne *Layover Station*), 69 codes de rotation (colonne *Pairing Code*) et 100 codes de vol (colonne *Leg Code*). Chaque employé a en moyenne 4 préférences (de 0 à 6 préférences). Les poids donnés à chaque préférence sont très variables, allant de 300 à 18000, avec une moyenne de 3200. Les escales souhaitées ont une durée moyenne de 1.8 jours. La colonne *Ladderlevel* donne l'ordre des préférences pour chaque employé et la colonne *Wish Type* vaut toujours *AWARD*. La colonne *Relop* peut prendre les valeurs *EQUAL*, *GREATER* ou *SMALLER*. Cela indique les conditions de satisfaction de la préférence : par exemple, lorsque *Relop* vaut *GREATER* pour une préférence d'escale, une escale plus longue que celle souhaitée correspond aussi à une préférence satisfaite. Le Tableau 3.3 donne une vue d'ensemble sur les 18 sortes de préférences. Ce tableau décrit les différentes catégories de préférence possibles, où pour chaque catégorie, les champs pour lesquels il y a de l'information sont indiqués avec "——-". Par exemple la préférence *C160* est une préférence d'aéroport d'escale, caractérisée par le code de l'aéroport d'escale souhaité. La préférence *C216* concerne une période de congés indivisible, caractérisée par une date de début et de fin de congé. Elle est indivisible dans le sens où, si la période complète n'est pas accordée en congé, l'employé préfère travailler durant cette période au complet et cette préférence est entièrement insatisfaite. Quant à la préférence *C108*, elle concerne également une période de congés souhaitée, caractérisée aussi par une date de début et de fin, mais si une partie seulement de la période est accordée en congé, l'employé sera en partie satisfait.

Tableau 3.3 Présentation des 18 différentes catégories de préférence

	Relop	Layover Station	Layover Duration	Pairing Duration	Leg Code	Pairing Code	Start Date	End Date	Specific Date	WeekDay 1	WeekDay 2
Station Layover C160	Equal	——-									
Period of Dates Off C216	Equal						——-	——-			
Station Layover Duration C267	Greater(99%)/ Smaller (1%)	——-	——-								
String of Dates Off C108	Equal						——-	——-			
Station Layover Duration on Date C385	Greater(98%)/ Smaller (2%)	——-	——-						——-		
Flight on Dates C702	Equal				——-		——	——			
Pairing Length in Duration C299	Smaller			——-							
Specific Flight C128	Equal				——-						
Station Layover on Dates C404	Equal	——-					——-	——-			
Specific Date Off C217	Equal								——-		
Weekends Off C214	Equal										
Specific Pairing on Dates C234	Equal					——-	——	——			
Specific Pairing C131	Equal					——-					
Day of Week Off C215	Equal									——-	
Flight on Days C703	Equal				——-					——-	——-
Period of Days Off C650	Equal									——-	——-
String of Days Off C503	Equal									——-	——-
Check-In on Date(s) C607	Equal						——-	——-			

Les Figures 3.12 et 3.13 présentent les popularités des différentes sortes de préférence, ainsi que les grandes tendances des employés.

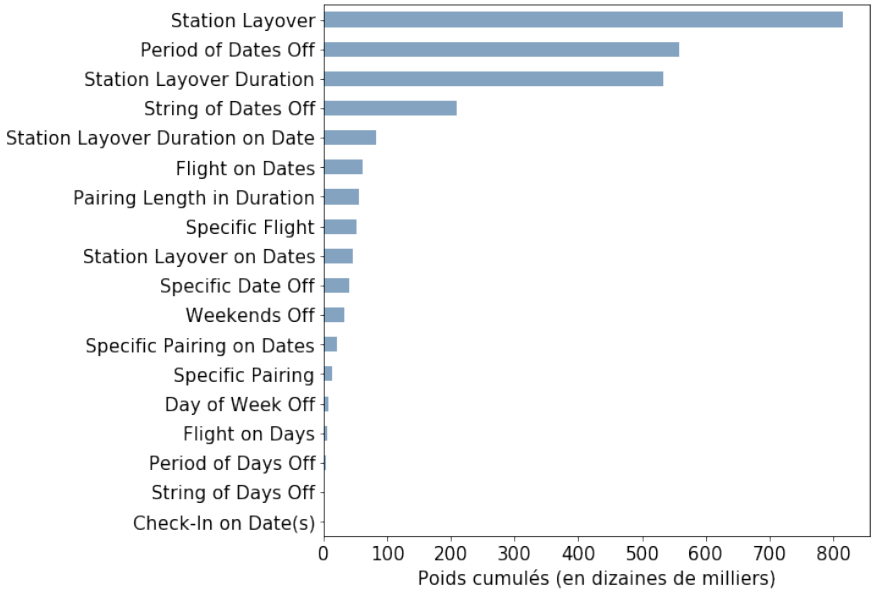


Figure 3.12 Poids cumulés par type de préférence pour novembre 2018

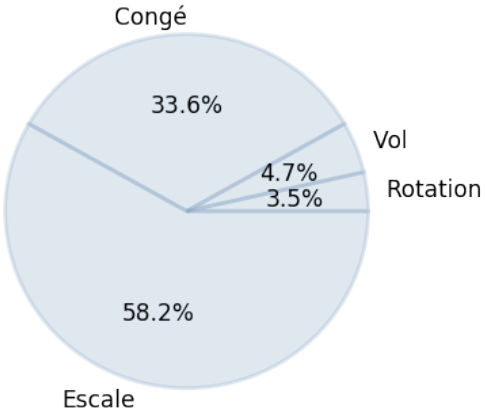


Figure 3.13 Tendances générales des préférences pour novembre 2018

3.7 L'équité

La Figure 3.14 présente les données du fichier de l'équité sous la forme d'une table de données.

	Employee	Alpha	Best	Worst	Satisfaction
0	E_A	27.39	86223.000000	-0.000000	0.871157
1	E_B	55.11	4135.000000	-0.000000	0.999075
2	E_C	17.35	48678.000000	-0.000000	0.386956
3	E_D	22.30	25287.000000	-0.000000	0.583386
4	E_E	1.00	()	()	()

Figure 3.14 Extrait de la table de données de l'équité de novembre 2018

Ce fichier contient 1861 lignes, où une ligne correspond à un employé. La colonne *Alpha* indique la valeur d'un facteur de multiplication de la satisfaction de l'employé suivant les blocs mensuels précédents. Les colonnes $Best$ et $Worst$ donnent les meilleurs et les pires scores de satisfaction que l'employé peut obtenir sur un mois. La colonne $Satisfaction$ donne la satisfaction de l'employé pour son bloc mensuel (calculée en interne par l'entreprise AD OPT) dans la solution fournie par le fichier des solutions. $Best$ vaut environ 52000 en moyenne et $Worst$ vaut 0 dans 98% des cas. La satisfaction est un réel entre 0 et 1, avec une moyenne de 0.7. Il y a quelques cas où $Best = Worst = 0$, auquel cas la satisfaction vaut 1. La Figure 3.15 présente la distribution des valeurs de satisfaction pour novembre 2018, et la Figure 3.16 présente celle des valeurs de $Best$.

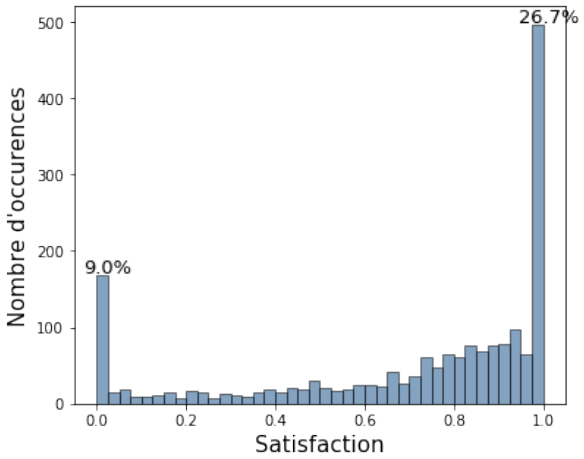


Figure 3.15 Histogramme des satisfactions pour novembre 2018

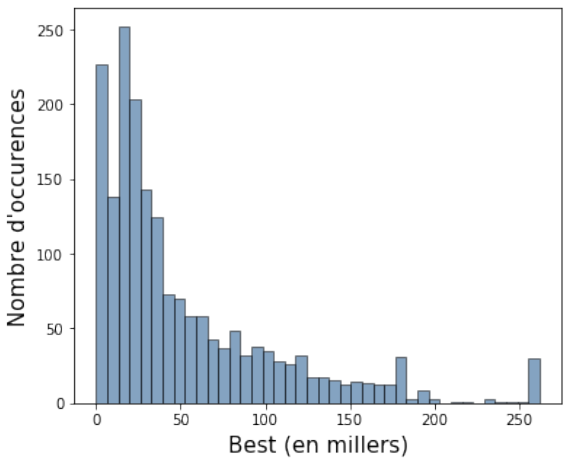


Figure 3.16 Histogramme des valeurs de $Best$ pour novembre 2018

3.8 Informations générales résumées

Les Tableaux 3.4 et 3.5 présentent les informations générales résumées de tous les fichiers de données.

Tableau 3.4 Informations sur les lignes des différents fichiers de données de novembre 2018

	Nombre de lignes	**Signification d'une ligne**
Solution	10.170 lignes	1 ligne = 1 poste
Employés	1.861 lignes	1 ligne = 1 employé
Activités	11.407 lignes	1 ligne = 1 rotation
Rotations	31.401 lignes	1 ligne = 1 segment de vol ou 1 escale
Langues	7.444 lignes	1 ligne = 1 langue dans une rotation
Préférences	7.867 lignes	1 ligne = 1 préférence
Équité	1.861 lignes	1 ligne = 1 employé

Tableau 3.5 Nombre d'éléments uniques dans les différents fichiers de données de novembre 2018

	Rotations	**Codes de rotation**	**Employés**	**Postes**	**Codes de vol**	**Aéroports d'escale**	**Qualifications**	**Langues**	**Destinations**
Solution	5.082	195	1.778	10.170					
Employés			1.861				118	58	
Activités	11.361	268		22.018			16		
Rotations	11.363	269			304	57			88
Langues	5.091	197						20	
Préférences		69	1.861		100	85			
Équité			1.861						

CHAPITRE 4 MODÉLISATION ET RÉSULTATS

Ce chapitre présente les différents modèles utilisés ainsi que leurs résultats. Il est composé de cinq sections. La première décrit le modèle basique d'apprentissage profond utilisé dans le projet. Elle commence par décrire le traitement des données effectué, puis la mesure de performance utilisée, explique la normalisation de *batch*, décrit les conditions de l'implémentation des expériences, et enfin présente et commente les résultats obtenus. Tout ce qui est expliqué dans la première section est valable dans toutes les suivantes, car les quatre sections suivantes présentent quatre améliorations différentes du modèle basique. La dernière résume et commente l'ensemble des résultats de ce chapitre.

4.1 Résultats préliminaires

Cette section présente les conditions et les détails d'un modèle basique d'apprentissage profond, reprenant le modèle décrit dans la section 2.1. Les résultats serviront de référence de comparaison pour les modèles suivants.

4.1.1 Les éléments en entrée

Pour apprendre à approximer la fonction F par apprentissage machine, l'approche choisie pour ce projet est d'énumérer toutes les combinaisons de (rotation, employé) possibles. Sachant qu'en un mois, il y a environ 1800 employés et 5000 rotations, avec des données sur 3 mois, environ 27 millions de combinaisons sont énumérées. Cela fait beaucoup de données sur lesquelles apprendre, mais cette méthode a l'avantage d'être simple et directe.

L'entrée x de la fonction F comprend donc deux types de caractéristiques (ou *features*) : celles concernant les rotations, et celles des employés.

Concernant les rotations, nous disposons des informations sur les dates de début et de fin de la rotation, les durées, les dates de début et de fin des segments de vol, les jours de la semaine correspondants, les destinations des segments de vol (et donc les aéroports d'escale), les durées d'escale, les langues requises ainsi que d'autres qualifications requises sur la rotation, par exemple de sécurité, de formation, ou bien de visa.

Concernant les employés, nous disposons des informations sur les langues parlées et les autres qualifications, ainsi que de beaucoup d'informations sur les préférences. Il y en a de plusieurs sortes : celles sur les jours de congé (date ou jour de la semaine), sur les dates, durées, et lieux d'escales, sur les dates, durées, et codes de vol, sur les dates, durées et types de rotation, etc.

4.1.2 La fonction de satisfaction

Pour des raisons de complexité des informations sur les préférences des employés, nous avons choisi d'écrire une fonction de satisfaction qui reflète la contribution d'une rotation à la satisfaction de l'employé pour son bloc mensuel. Ainsi, cette fonction prend en entrée une rotation et un employé donné, et donne en sortie un scalaire qui a été mis à l'échelle des informations contenues dans le fichier d'équité. L'entreprise AD OPT a fourni des indications sur le fonctionnement du calcul du score de satisfaction sur un bloc mensuel complet, mais ils ne disposaient pas de fonction de satisfaction d'un employé pour une unique rotation, c'est pourquoi des ajustements ont dû être effectués.

Si l'employé a une préférence de segment de vol, de rotation ou d'escale, et que la rotation la satisfait (ce qui est facile à vérifier grâce au fichier des rotations), la fonction va simplement sommer les poids associés aux préférences satisfaites. Dans le cas des préférences de jour de congé, la fonction va ajouter le poids de la préférence par défaut, mais si la rotation empiète sur un jour de congé demandé par l'employé, le poids de la préférence sera retranché, soit au complet si c'est une préférence de période de jour de congé, soit proportionnellement au nombre de jours de congé empiétés sur le nombre total de jours de congé demandés.

Par exemple, si l'employé a une préférence $C216$ *Period of Dates Off* (période de congés indivisible), ayant un poids de 2000 pour la période du 3 au 5 novembre, et que la rotation empiète sur au moins un de ces 3 jours, alors la fonction n'ajoutera rien ; mais si la rotation n'empiète pas sur ces 3 jours, alors la fonction va ajouter 2000.

Autre exemple : pour un employé ayant une préférence $C108$ *String of Dates Off*, avec un poids de 5000 pour la période du 10 au 13 novembre, si la rotation empiète sur les 10 et 11 novembre, alors la fonction va ajouter seulement 2500.

Dans les cas où les jours de congé demandés sont des jours de la semaine ou des fins de semaines, alors le nombre de jours de congé total demandés sera calculé comme étant le nombre de fois que ces jours de la semaine apparaissent dans le mois. Par exemple, si l'employé demande à avoir les lundis de congé, comme il y en a 4 en novembre 2018, on comptera 4 jours de congé demandés au total, et si la rotation empiète sur un seul lundi, la fonction ajoutera les trois quarts du poids associé à la préférence.

Après avoir considéré toutes les préférences de l'employé et fait les additions décrites plus haut, la fonction de satisfaction remet à l'échelle ce score avec les *Best* et *Worst* de l'employé. Il arrive que le score obtenu soit plus faible que le *Worst* ou plus grand que le *Best*, car les *Best* et *Worst* sont calculés sur tous les blocs mensuels réalisables pour l'employé. Dans ces cas-là, on remettra le score à 0 ou 1 suivant le cas.

Ainsi, les informations sur les détails des rotations (segments de vol et escales) et les détails des préférences des employés ont été mises dans un scalaire unique compris entre 0 et 1 pour chaque paire (employé, rotation). Les caractéristiques d'apprentissage restantes sont les informations sur les langues et qualifications requises sur les rotations, et les langues et qualifications des employés. Ce choix de caractéristiques permet de généraliser l'apprentissage à d'autres mois au sein de la même compagnie. En effet, d'un mois sur l'autre, ces caractéristiques sont sensiblement les mêmes : les rotations se ressemblent beaucoup, ce sont les mêmes employés, donc les distributions des langues et autres qualifications sont presque inchangées. Ils ont aussi des préférences très similaires. Toutes les informations concernant un mois spécifique ont été mises dans ce score de satisfaction : par exemple les dates précises, les codes de vol, les codes de rotation, etc.

4.1.3 Le prétraitement des données

Comme le montre le Tableau 3.5, les différents fichiers n'ont pas exactement le même nombre d'employés, de rotations, etc.

Par exemple, on remarque que certaines rotations sont dans les solutions mais pas dans le fichier des langues, et réciproquement. Par contre, toutes les rotations des solutions sont dans les fichiers des activités et des rotations. Ainsi, pour construire les données qui seront utilisées pour l'apprentissage, on ne va retenir que les rotations qui sont à la fois dans le fichier des langues et des solutions, ce qui fait 4978 rotations pour le mois de novembre 2018.

Par ailleurs, tous les employés dans la solution sont décrits dans le fichier des employés, des préférences et de l'équité. On ne retiendra donc que les employés du fichier des solutions.

Aussi, en croisant les informations, on remarque également que les qualifications "FB", "FC", "FJ", et "CS", très peu présentes dans le fichier des rotations, ne sont en fait jamais présentes dans le fichier des employés (alors que toutes les autres le sont). On peut donc réduire le nombre de qualifications possibles à 12 (au lieu des 16 du Tableau 3.5). Les autres qualifications des employés qui ne sont pas parmi ces 12-là seront ignorées. Toutes les 20 langues requises dans une rotation (dans le fichier des langues) ont des employés qui les parlent (dans le fichier des employés). Les autres langues parlées par les employés seront ignorées.

Ainsi, les données d'apprentissage seront constituées de 65 colonnes (les caractéristiques, ou *features* en anglais), où chaque ligne correspondra à une paire (employé, rotation) possible. Dans ces 65 colonnes, 12 seront pour les qualifications de l'employé, 20 seront pour les langues parlées par l'employé, 12 pour les qualifications requises par la rotation, 20 pour les langues requises par la rotation, et 1 sera dédiée au score de satisfaction de la rotation pour l'employé.

Les valeurs des 64 premières colonnes seront binaires, indiquant la présence ou l'absence de la caractéristique. La dernière sera un scalaire entre 0 et 1. À cela est ajoutée une colonne pour le label binaire, qui vaut 1 si la rotation est dans le bloc de l'employé, et 0 sinon.

4.1.4 La mesure de performance

Pour le problème de prédiction des rotations dans les blocs des employés, la performance doit être mesurée sur un mois au complet, et ne peut être mesurée au niveau d'un *batch*, qui est souvent de taille réduite (128 dans notre cas), et contient les combinaisons ayant des employés et des rotations différentes. En effet, il faut connaître les prédictions sur tout un mois pour pouvoir classer pour chaque employé les rotations par ordre de valeur de prédiction, avant d'en sélectionner les 500 meilleures, les 1000 meilleures, etc. La performance "unitaire" du modèle vis-à-vis d'une combinaison (employé, rotation) (e, r) ne peut pas s'évaluer sans connaître les prédictions de toutes les autres combinaisons (e, r') pour le même employé e et les 5000 autres rotations possibles r'. On remarque donc que le vrai but de l'apprentissage est de classer correctement les rotations dans l'ordre de probabilité de présence dans le bloc de chaque employé, et la mesure de performance, calculée sur un mois, correspond à cet objectif. Mais la fonction de perte utilisée pour la descente de gradient ne prend pas en compte les dépendances entre les combinaisons (employé, rotation). Ainsi, par rapport à un cas standard de classification, le décalage entre la fonction de perte et la performance est un peu plus important.

Dans un cas idéal, pour mesurer la performance, il faudrait donner les 500 rotations les plus probables aux sous-problèmes de chaque employé dans l'algorithme de génération de colonnes, puis lancer la génération de colonnes, examiner la *nouvelle* solution, puis regarder la proportion des nouveaux labels 1 qui sont dans les tops 500 donnés initialement par l'apprentissage machine, en moyennant sur tous les employés. Toutefois, cette démarche prendrait trop de temps et serait difficile à implémenter, car l'algorithme de génération de colonnes d'AD OPT prend du temps et son accès n'est pas fourni dans tous les cas.

Ainsi, en pratique, l'apprentissage machine n'est pas intégré à la génération de colonnes. Au lieu de regarder les *nouveaux* labels 1, on regardera les anciens labels 1 (ceux fourni par le fichier des solutions). Si une nouvelle solution avait été fournie par la génération de colonnes intégrant les résultats de l'apprentissage, elle contiendrait plus de combinaisons (employé, rotation) parmi celles fournies par l'apprentissage. La performance du modèle est donc sous-estimée.

Par ailleurs, ce sera bien une sous-estimation, car le problème des blocs mensuels a beaucoup de solutions équivalentes en termes de coût et de qualité, et ces mêmes solutions peuvent être

très différentes. En effet, la solution fournie (pour chaque mois) n'est pas la solution optimale, mais seulement une solution qui a été jugée satisfaisante dans un contexte industriel. Ainsi, en essayant de reproduire une seule solution (celle dont on dispose), on vise une cible beaucoup plus petite que celle qu'on voudrait viser en réalité. Idéalement, il faudrait apprendre à partir de plusieurs solutions de même qualité mais significativement différentes pour avoir un apprentissage plus pertinent. Ce sera l'objet de travaux futurs pour la poursuite du projet.

4.1.5 La normalisation de *batch*

La normalisation de *batch* [31] est une technique introduite suite à la remarque suivante. Dans les réseaux de neurones profonds, plusieurs fonctions sont composées successivement, et chaque fonction contient des paramètres qu'on optimise par descente de gradient. Or, les gradients par rapport à chacun des paramètres sont calculés en considérant que tous les autres paramètres sont constants, alors qu'en pratique, on calcule tous les gradients en une fois et on met à jour les paramètres de toutes les couches en même temps. La structure de couches propage en avant les informations sur les pas de chacun des paramètres, et à la sortie, le pas effectué par la prédiction $\hat{y}$ peut dépendre des valeurs des paramètres des couches précédentes à des ordres 2 ou plus en la vitesse d'apprentissage ε [27]. Puisque plusieurs fonctions se composent successivement, l'effet sur le pas de $\hat{y}$ de la mise à jour des paramètres d'une couche dépend fortement de celle des couches inférieures.

Pour pallier cet effet, la normalisation de *batch* remplace la sortie $h^{(i)}$ de la couche i par sa version renormalisée $h^{(i)} \leftarrow \dfrac{h^{(i)} - \mu^{(i)}}{\sigma^{(i)}}$, où $h^{(i)}$ est un vecteur de taille $K \times 1$, avec K la largeur de la couche i ; $\mu^{(i)}$ est la moyenne de $h^{(i)}$ sur tous les exemples du *batch* ; et $\sigma^{(i)}$ sa variance. Ainsi, en notant m la taille du *batch*, $h_k^{(i)}$ la valeur de $h^{(i)}$ pour l'exemple k du *batch* :

$$\mu^{(i)} = \frac{1}{m} \sum_{k=1}^{m} h_k^{(i)} \tag{4.1}$$

et

$$\sigma^{(i)} = \sqrt{\frac{1}{m} \sum_{k=1}^{m} \left(h_k^{(i)} - \mu^{(i)} \right)^2} \tag{4.2}$$

sachant que toutes les opérations dans l'équation (4.2) sont effectuées terme à terme.

Toutefois, pour ne pas réduire la capacité du modèle, on introduit encore des paramètres $\alpha^{(i)}$ et $\beta^{(i)}$ pour chaque couche renormalisée, et on remplace le nouveau $h^{(i)}$ par $\alpha^{(i)} h^{(i)} + \beta^{(i)}$, pour que $h^{(i)}$ puisse avoir n'importe quelle moyenne $\beta^{(i)}$ et n'importe quelle variance $(\alpha^{(i)})^2$ sur le *batch*. L'intérêt de cette démarche, qui pourrait sembler inutile de prime abord, réside

dans le fait que la moyenne et la variance sur le *batch* sont devenues des paramètres à part entière, qu'on apprend en faisant des pas de descente dédiés. Avant cette manipulation, la moyenne et la variance dépendaient intrinsèquement du *batch* ainsi que des paramètres des couches courante et précédentes, et ce, de manière complexe. Maintenant, lorsqu'il conviendra de changer la moyenne et la variance de la sortie de la couche i, il suffira de mettre à jour $\alpha^{(i)}$ et $\beta^{(i)}$ sans que cela soit affecté par les mises à jour des couches précédentes, et sans que cela n'affecte les mises à jour des couches suivantes. Ioffe et Szegedy [31] présentent plus d'informations sur la normalisation de *batch*.

4.1.6 Implémentation des expériences

Cette section décrit les conditions générales des expériences, valables également pour les expériences des sections 4.2, 4.3, 4.4, et 4.5.

Les expériences ont toutes été implémentées avec Python et la librairie PyTorch. Comme nous disposons des données sur trois mois, on utilisera les données de deux mois pour l'entraînement et les données du troisième pour la phase de test. Par manque de données et de mémoire, la performance utilisée dans l'arrêt anticipé sera la performance de test. La taille des *minibatchs* est 128.

Pour l'ensemble des expériences qui seront réalisées dans ce projet, les commentaires sur les résultats porteront toujours sur la performance en extrayant les 500 plus probables par employé (lignes grisées), mais les résultats afficheront toujours les performances pour les 1000, 2000, 3000, 4000 et 5000 plus probables à titre indicatif. La performance du Top 500 servira de référence car c'est celle qui offre les plus grandes perspectives d'accélération de l'algorithme de génération de colonnes.

Le modèle basique décrit au chapitre 3 a été testé pour trois architectures différentes : la régression logistique (sans couche cachée) ; le perceptron multicouche à 2 couches cachées ayant chacune 130 unités cachées ; et le perceptron multicouche à 5 couches cachées, ayant successivement 1000, 750, 500, 250 et 50 unités cachées. Les fonctions d'activation des couches cachées sont toutes des unités de rectification linéaires, et celle de la couche de sortie est la fonction sigmoïde. Ces trois architectures seront utilisées dans les modèles des sections qui suivent pour pouvoir les comparer. Il conviendrait d'explorer plus d'architectures, avec plus ou moins de couches et de neurones, mais pour des raisons pratiques, seules celles-ci ont été testées.

Pour chacune de ces trois architectures, on a aussi testé l'effet du décrochage (*dropout*) sur les unités cachées (ne s'applique pas pour la régression logistique), avec la même probabilité

pour toutes les unités cachées d'être masquées. Quatre valeurs ont été testées : 0.0, 0.3, 0.5 et 0.7. L'effet de la normalisation de *batch* a aussi été testée. Enfin, quatre valeurs de la vitesse d'apprentissage (*learning rate*) ont été testées : 10^{-1}, 10^{-2}, 10^{-3}, et 10^{-4}. Cette échelle logarithmique avait pour but de couvrir un spectre assez large de valeurs de vitesse d'apprentissage. Idéalement, il faudrait explorer plus en détails les valeurs des hyper-paramètres, surtout la vitesse d'apprentissage qui semble être cruciale, mais aussi les hyper-paramètres concernant l'architecture : le nombre d'unités cachées, le nombre de couches cachées, les fonctions d'activation, les probabilités de décrochage etc. Pour des raisons de temps de calcul, l'exploration des valeurs d'hyper-paramètres a été limitée.

Pour pouvoir suivre l'évolution de l'algorithme, les performances sur l'ensemble d'entraînement et de test seront calculées seulement toutes les 10 époques pour des raisons de temps de calcul (voir section 4.1.4). Par ailleurs, l'arrêt anticipé est utilisé dans toutes les expériences, avec un maximum de 80 époques pour avoir des temps de calcul raisonnables. Ce critère est testé toutes les 10 époques vu qu'il porte sur la performance du modèle. Si la performance mesurée décroît deux fois consécutivement (donc sur 20 époques), l'apprentissage s'arrête.

4.1.7 Résultats

Les Tableaux 4.1, 4.2, 4.3, et 4.4 présentent les résultats pour le modèle basique. Dans cette section et dans toute la suite, les résultats affichés concerneront seulement les performances de test. Les performances d'entraînement ne sont pas affichées par souci de concision.

Tableau 4.1 Performances de test pour le modèle basique, vitesse d'apprentissage = 10^{-4}

	Rég. log.	**MLP 2 couches**								**MLP 5 couches**							
		Batch norm 0				*Batch norm* 1				*Batch norm* 0				*Batch norm* 1			
Dropout		0.0	0.3	0.5	0.7	0.0	0.3	0.5	0.7	0.0	0.3	0.5	0.7	0.0	0.3	0.5	0.7
Top 500	13.4	15.86	11.9	11.37	11	26.66	18.39	15.18	12.76	20.26	13.06	12.14	10.32	30.22	20.34	15.76	10.61
Top 1000	21.07	25.1	23.31	21.96	21.33	42.46	31.61	27.95	24.45	35.82	25.27	23.05	20.65	41.73	35.39	28.31	19.95
Top 2000	36.25	40.48	45.54	42.51	41.73	62.68	53.4	49.91	45.54	60.38	47.93	43.94	41.65	59.21	57.32	50.12	40.16
Top 3000	64.83	67.71	64.95	63.81	62.19	77.67	70.89	68.98	64.98	76	67.93	64.7	61.84	76.2	74.34	69.25	59.84
Top 4000	85.36	88.83	84.36	82.99	82.06	91.27	87.69	86.58	84.01	90.33	85.71	83.44	80.75	89.31	89.37	86.4	79.83
Top 5000	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
Early stop	30	80	80	80	80	80	80	80	80	80	80	80	40	80	80	80	30
Temps (h)	3	9.2	10	10.1	11.3	12.7	12.6	10.4	10.4	13.6	15	14.6	6.7	18.5	14.3	18.9	7.2

Tableau 4.2 Performances de test pour le modèle basique, vitesse d'apprentissage = 10^{-3}

	Rég. log.	**MLP 2 couches**								**MLP 5 couches**							
		Batch norm 0				*Batch norm* 1				*Batch norm* 0				*Batch norm* 1			
Dropout		0.0	0.3	0.5	0.7	0.0	0.3	0.5	0.7	0.0	0.3	0.5	0.7	0.0	0.3	0.5	0.7
Top 500	19.15	25.46	17.62	15.77	14.32	34.73	29.18	26.19	19.79	30.16	18.66	18.58	11.16	29.32	27.99	23.27	15.58
Top 1000	33	43.37	33.15	29.24	27.88	44.6	41.51	38.79	32.88	48.65	34.59	33.75	21.67	38.63	41.96	37.54	30.41
Top 2000	49.28	69.58	59.4	54.75	53.21	62.6	61.09	59.14	54.38	73.43	62.79	59.05	42.31	56.89	61.89	59.21	54.12
Top 3000	68.63	88.81	82.3	76.55	75.28	81.98	77.67	75.9	72.64	91.3	84.92	81.03	61.98	76.2	77.9	75.49	72.68
Top 4000	88.9	95.79	94.93	92.57	91.85	92.92	91.12	90.33	88.41	97.31	96	93.27	81.55	89.18	91.51	89.81	89.47
Top 5000	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
Early stop	80	80	80	80	80	80	80	80	80	80	80	80	40	60	80	80	80
Temps (h)	7.6	12.8	13.9	9.1	11.1	12.6	13.2	11.6	11.3	13.2	14.2	14.5	6.6	13.7	18.4	74.7	19.6

Tableau 4.3 Performances de test pour le modèle basique, vitesse d'apprentissage = 10^{-2}

	Rég. log.	**MLP 2 couches**								**MLP 5 couches**							
		Batch norm 0				*Batch norm* 1				*Batch norm* 0				*Batch norm* 1			
Dropout		0.0	0.3	0.5	0.7	0.0	0.3	0.5	0.7	0.0	0.3	0.5	0.7	0.0	0.3	0.5	0.7
Top 500	22.7	30.49	28.17	27.17	26.39	34.76	33.2	33.69	31.64	30.85	29.94	29.4	24.51	32.94	37.25	36.77	32.34
Top 1000	40.58	47.19	46.41	45.61	44.6	44.1	43.95	44.58	43.87	47.6	47.61	47.56	43.85	42.3	48.11	48.92	45.6
Top 2000	61.22	72.51	71.05	70.07	70.04	61.41	64.58	64.2	62.91	69.94	71.81	72.24	69.56	60.05	66.02	67.33	65.35
Top 3000	76.91	91.68	90.3	89.59	89.92	79.61	81.1	80.78	79.1	88	90.63	90.66	89.33	77.47	83.31	82.92	80.99
Top 4000	90.76	97.57	97.02	96.94	96.98	90.81	92.89	92.74	91.93	95.97	97.26	97.26	96.91	88.51	93.84	93.94	92.91
Top 5000	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
Early stop	80	80	80	80	80	80	80	80	80	70	80	80	80	50	80	80	80
Temps (h)	6.5	9.8	10.8	8.3	10.8	12.2	12.9	12.7	11.7	12	13.8	14.2	10.1	11.3	20.1	19.3	18.8

Tableau 4.4 Performances de test pour le modèle basique, vitesse d'apprentissage = 10^{-1}

	Rég. log.	**MLP 2 couches**								**MLP 5 couches**							
		Batch norm 0				*Batch norm* 1				*Batch norm* 0				*Batch norm* 1			
Dropout		0.0	0.3	0.5	0.7	0.0	0.3	0.5	0.7	0.0	0.3	0.5	0.7	0.0	0.3	0.5	0.7
Top 500	23.8	30.12	27.79	28.02	26.19	36.04	33.55	33.14	32.6	29.96	28.86	29.41	27.14	35.38	37.39	36.87	36.77
Top 1000	39.52	45.81	43.61	45.26	43.78	45.28	43.99	43.19	43.94	45.22	44.05	45.01	45.51	44.3	47.46	46.48	48.3
Top 2000	62.31	67.14	66.17	68.59	67.57	62.31	63.12	61.76	62.78	66.92	65.86	67.11	70.18	61.54	64.92	63.85	66.76
Top 3000	78.14	85.7	85.62	87.8	87.44	80.18	81.59	78.88	79.75	85	85.75	86.63	90.05	79.08	81.15	81.11	82.68
Top 4000	90.59	94.66	95.51	96.25	95.86	91.07	92.52	91.05	92.18	94.28	95.29	95.57	97.04	88.54	90.68	91.42	92.51
Top 5000	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
Early stop	30	80	40	80	80	80	70	80	70	30	80	80	80	60	80	80	80
Temps (h)	3.5	9.9	5.3	8.3	10.8	10.8	11.3	9.4	10.2	5.1	13.8	14.2	14	13.5	19.4	19.6	18.5

"Rég. log." signifie "régression logistique", et les "MLP" 2 et 5 couches sont les perceptrons multicouches décrits dans la section précédente. "*Batch norm*" indique la présence ou non de normalisation de *batch*. La ligne *Dropout* indique la valeur de la probabilité de masquer une unité cachée dans les perceptrons multicouches. Les lignes "Top 500", ..., "Top

5000" contiennent les meilleures performances pendant l'apprentissage (mesurée toutes les 10 époques). La performance est le pourcentage des labels 1 qui ont été correctement prédits dans les 500, ..., 5000 meilleures rotations. À titre de comparaison, si on avait choisi 500 rotations au hasard, on aurait eu une performance d'environ 10%.

La ligne *Early stop* indique l'époque où le critère d'arrêt a été satisfait. Lorsque cela vaut 80, cela signifie que le nombre d'époques maximal a été atteint sans pour autant que le critère d'arrêt ait été vérifié. La valeur minimale est 30, puisqu'on décide d'arrêter après deux décroissances consécutives de la performance du Top 500. La ligne "Temps" indique le temps de calcul de l'apprentissage en heures. Le temps total de calcul est de 890 heures.

On remarque que les résultats de la régression logistique sont globalement moins bons que ceux des deux autres architectures, qui sont comparables. La normalisation de *batch* augmente significativement les résultats, tandis que le décrochage (ou *dropout*) fait presque toujours diminuer la performance. Le décrochage n'est sûrement pas efficace ici car le sur-apprentissage n'est pas atteint. C'est sûrement dans les cas de léger sur-apprentissage que le décrochage améliore la performance, ce qui correspond en pratique à des performances d'entraînement supérieures à 40% (pas affichées ici par souci de concision).

Les résultats supérieurs à 35% ont été mis en valeur car ce sont les meilleurs performances. On remarque ainsi que les meilleurs résultats ont été obtenus avec le perceptron à 5 couches, la normalisation de *batch*, une probabilité de décrochage de 0.3, et une vitesse d'apprentissage de 10^{-2} ou 10^{-1}. La meilleure performance atteinte ici pour le Top 500 est d'environ 37%.

Nous avons aussi regardé le pire parmi les meilleurs rangs des rotations, sachant que le meilleur rang d'une rotation est choisi parmi ses combinaisons avec tous les employés possibles. On veut que cette valeur soit basse (en-dessous de 500 idéalement), car cela indique que la génération de colonnes aura la garantie d'une solution réalisable avec les réseaux réduits de taille 500. En effet, si on sélectionne les 500 rotations les plus probables pour chaque employé, mais qu'il y a une rotation qui n'apparaît dans aucun des Top 500 par employé, alors cette rotation ne sera pas présente dans les réseaux réduits alors qu'elle doit être couverte dans le problème maître, ce qui conduit à une solution non-réalisable.

Ainsi, parmi les 7 meilleures performances, dans seulement 3 cas, le pire des meilleurs rangs est convenable : 499, 578 et 393. Dans les autres cas, il est au-dessus de 2000. Parmi les trois cas convenables, c'est le cas surligné en orange (Tableau 4.3, MLP 5 couches, *batch norm* 1, *Dropout* 0.3) qui a la meilleure performance, pour un pire des meilleurs rangs de 499. Ainsi, la contrainte globale concernant la présence de toutes les rotations dans les réseaux réduits est satisfaite dans cette configuration. Le Tableau 4.5 présente les informations des meilleurs rangs dans ce cas.

Tableau 4.5 Informations sur la distribution des meilleurs rangs des rotations dans la configuration retenue pour le modèle basique

Moyenne	Écart type	Minimum	1er quartile	Médiane	3e quartile	Maximum
20.02	32.45	0	1	9	27	499

Enfin on remarque que l'arrêt anticipé n'est effectif que dans 13 cas sur 68, souvent avec le perceptron à 5 couches. Le nombre total d'époques effectuées est de 5020.

4.2 Une fonction de perte pondérée

Cette section présente un modèle amélioré par rapport à celui de la section précédente.

4.2.1 Des données déséquilibrées

La raison des faibles résultats du modèle basique vient sûrement du déséquilibre des données entre les labels 1 et les labels 0.

En effet, les données ont été générées en énumérant toutes les paires (employé, rotation) possibles. Il y en a un nombre combinatoire, de l'ordre de 9 millions pour un mois. Parmi celles-ci, seulement 0.1% ont un label 1 (i.e. apparaissent dans la solution fournie). Le modèle aura alors tendance à tout prédire à 0 et tout en conservant une perte très faible. Mais la performance restera mauvaise, car la fonction de perte ne prend pas en compte les prédictions des autres combinaisons de (employé, rotation), tandis que la performance oui.

Le Tableau 4.6 présente la matrice de confusion et les notions de vrai positif, vrai négatif, faux positif, faux négatif.

Tableau 4.6 Matrice de confusion

Classe réelle \ Classe prédite	Prédit à 0	Prédit à 1
Réellement 0	Vrai négatif	Faux positif
Réellement 1	Faux négatif	Vrai positif

Il est crucial de bien prédire les labels 1. En effet, lorsqu'on trie les rotations par ordre de valeur de prédiction pour chaque employé, puis qu'on en extrait les 500 premiers, ces 500 auront nécessairement au moins 488 combinaisons avec un label 0. En effet, un employé a en moyenne 6 rotations dans son bloc mensuel, avec un maximum de 12 (voir section 3.1). Il est donc peu important de minimiser le nombre de faux positifs, vu que c'est inévitable, et qu'ils sont en large majorité dans le Top 500. Par contre, il est essentiel d'éviter les faux négatifs, c'est-à-dire qu'il faut que les rotations présentes dans le bloc de l'employé dans la solution soient présentes dans le Top 500 de cet employé.

4.2.2 *Accuracy* et rappel

L'entropie croisée binaire va avoir tendance à minimiser l'*accuracy* du modèle, c'est-à-dire le nombre de vrais positifs et de vrais négatifs divisé par le nombre total d'échantillons, alors qu'il faut plutôt minimiser le rappel (ou *recall*) qui est le rapport entre le nombre de vrais positifs sur le nombre total de positifs. La Figure 4.1 ci-dessous illustre les éléments de la matrice de confusion. L'*accuracy* tel que défini ici est appelé le plus souvent la performance, et parfois la justesse. Par souci de clarté, l'appellation *accuracy* sera conservée.

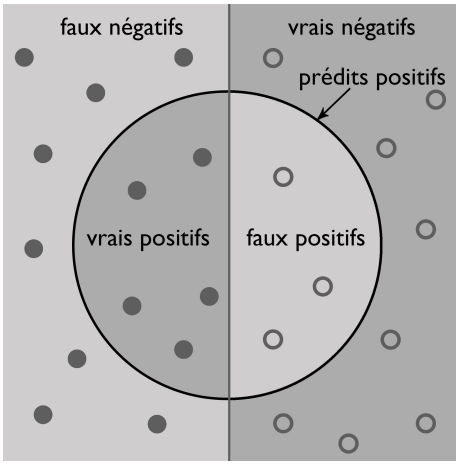
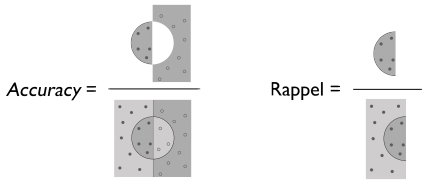


Figure 4.1 Illustration des éléments de la matrice de confusion

– Les cercles pleins représentent des échantillons avec un label 1.
– Les cercles vides représentent ceux avec un label 0.



Prédire les labels 1 est le plus important, alors que les labels 1 sont en très forte minorité. Or, l'entropie croisée binaire traite les erreurs sur les labels 0 et 1 équitablement. Cela conduit à un apprentissage peu performant. Il est alors naturel de pondérer l'*accuracy* et le rappel dans la fonction de perte, et c'est ce qui a été fait dans les expériences qui suivent. La pondération utilisée est celle suggérée pour faire croire à l'optimiseur que les données sont équilibrées : nombre de labels 0 divisé par le nombre de labels 1, ce qui vaut environ 900.

4.2.3 Résultats

Les Tableaux 4.7, 4.8, 4.9, 4.10 présentent les résultats obtenus avec une fonction de perte pondérée. Le temps total de calcul est d'environ 759 heures.

Tableau 4.7 Performances de test pour la fonction de perte pondérée, vitesse d'apprentissage $= 10^{-4}$

	Rég. log.	**MLP 2 couches**								**MLP 5 couches**							
		Batch norm 0				*Batch norm* 1				*Batch norm* 0				*Batch norm* 1			
Dropout		0.0	0.3	0.5	0.7	0.0	0.3	0.5	0.7	0.0	0.3	0.5	0.7	0.0	0.3	0.5	0.7
Top 500	23.43	29.01	27.36	26.41	25.24	35.12	34.17	33.54	31.01	29.55	29.33	28.41	27.95	34.45	36.4	33.29	34.33
Top 1000	38.97	46.64	44.9	44.32	42.9	44.58	46.18	45.61	42.59	45.23	46.18	46.82	46.86	43.25	47.26	46.42	47.88
Top 2000	62.2	69.45	68.72	68.21	68.34	62.85	66.05	66.02	62.02	65.78	69.36	71.78	73.37	60.77	64.48	66.19	67.21
Top 3000	78.11	87.08	87.35	87.58	87.2	79.99	82.83	82.92	78.36	84.79	87.57	90.93	91.55	77.89	81.03	83.07	83.13
Top 4000	91.36	95.36	96.04	96.35	95.87	90.28	93.63	93.54	89.96	94.91	95.88	97.37	97.23	87.96	92.21	93.48	93.68
Top 5000	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
Early stop	50	70	80	80	80	80	80	80	80	80	80	80	80	50	80	60	80
Temps (h)	4.8	9.7	11.4	11.7	11.6	13.1	13.6	12.9	12.8	12.3	15	14.6	15.2	12.1	19	13.9	18.9

Tableau 4.8 Performances de test pour la fonction de perte pondérée, vitesse d'apprentissage $= 10^{-3}$

	Rég. log.	**MLP 2 couches**								**MLP 5 couches**							
		Batch norm 0				*Batch norm* 1				*Batch norm* 0				*Batch norm* 1			
Dropout		0.0	0.3	0.5	0.7	0.0	0.3	0.5	0.7	0.0	0.3	0.5	0.7	0.0	0.3	0.5	0.7
Top 500	23.73	29.08	28.05	26.71	24.83	34.92	35.63	33.67	31.9	29.06	28.55	27.95	27.67	36.34	37.5	37.35	35.75
Top 1000	35.85	44.57	43.86	44.4	42.67	44.67	45.39	43.8	43.79	44.36	44.53	44.61	45.36	45.26	48.07	48.24	46.6
Top 2000	62.33	65.78	66.61	68.98	69.84	62.55	63.16	62.45	63.88	65.87	66	67.19	70.19	61.76	64.72	65.99	65.02
Top 3000	78.19	84.3	85.26	87.81	90.55	79.27	81.17	79.12	80.83	83.26	84.72	85.03	89	79.52	80.6	82.19	81.6
Top 4000	90.26	93.98	94.58	96.04	97.38	89.82	92.02	90.62	91.69	94.24	94.49	94.62	96.58	89.84	90.19	92.72	92.45
Top 5000	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
Early stop	80	50	80	80	80	80	80	80	70	40	80	80	80	30	80	80	80
Temps (h)	8	7	11.5	11.6	11.6	11.5	13.1	13.1	10.4	6.4	15.1	14.6	15.1	4.7	16.5	19.2	19.5

Tableau 4.9 Performances de test pour la fonction de perte pondérée, vitesse d'apprentissage $= 10^{-2}$

	Rég. log.	**MLP 2 couches**								**MLP 5 couches**							
		Batch norm 0				*Batch norm* 1				*Batch norm* 0				*Batch norm* 1			
Dropout		0.0	0.3	0.5	0.7	0.0	0.3	0.5	0.7	0.0	0.3	0.5	0.7	0.0	0.3	0.5	0.7
Top 500	20.66	28.91	25.79	23.79	19.97	35.19	29.19	27.36	22.39	28.95	24.35	10.68	10.72	36.89	36.27	34.76	35.52
Top 1000	33.66	43.71	42.07	40.65	31.51	43.8	40.88	38.75	33.73	45.29	42.43	20.81	20.78	46.09	46.57	45.15	46.15
Top 2000	51.31	65.33	64.47	65.32	49.99	61.36	62.03	58.97	53.95	66.97	70.4	40	39.97	63.69	63.89	63.72	63.68
Top 3000	70.49	83.5	84.6	87.08	68.92	79.8	80.64	75.83	71.77	84.42	89.58	60.51	60.48	79.62	80.48	82.76	80.59
Top 4000	89.9	94.17	94.4	96.39	87.73	89.35	91.69	89.38	86.75	93.98	96.48	80.27	80.27	87.78	90.73	92.43	94.3
Top 5000	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
Early stop	80	40	80	50	60	80	60	50	40	70	70	80	80	40	50	80	80
Temps (h)	8.1	5.6	11.5	7.3	8.7	12.8	9.8	8.4	6.6	11.8	13.1	15.2	14.6	8.3	12.5	19.5	19.6

Tableau 4.10 Performances de test pour la fonction de perte pondérée, vitesse d'apprentissage $= 10^{-1}$

	Rég. log.	**MLP 2 couches**								**MLP 5 couches**							
		Batch norm 0				*Batch norm* 1				*Batch norm* 0				*Batch norm* 1			
Dropout		0.0	0.3	0.5	0.7	0.0	0.3	0.5	0.7	0.0	0.3	0.5	0.7	0.0	0.3	0.5	0.7
Top 500	19.53	10.68	10.68	10.68	10.68	10.68	10.68	10.68	10.68	10.71	10.76	10.71	10.68	10.7	37.75	10.68	10.73
Top 1000	33.91	20.79	20.79	20.79	20.79	20.79	20.79	20.79	20.79	20.8	20.81	20.83	20.79	20.76	45.15	20.79	20.78
Top 2000	49.07	39.95	39.94	39.95	39.95	39.95	39.95	39.95	39.95	39.95	39.92	39.97	39.95	39.96	58.46	39.95	39.97
Top 3000	71.6	60.47	60.46	60.48	60.47	60.47	60.47	60.47	60.47	60.47	60.59	60.46	60.47	60.48	72.77	60.47	60.49
Top 4000	89.72	80.29	80.25	80.29	80.29	80.29	80.29	80.29	80.29	80.28	80.32	80.25	80.29	80.29	86.16	80.29	80.29
Top 5000	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
Early stop	80	40	40	40	30	30	30	30	30	40	80	30	30	80	80	30	40
Temps (h)	8.1	5.6	5.7	5.9	4.3	3.4	4.7	5	3.7	6.8	15.1	5.6	5.6	17.5	19	7.5	9.9

De même que précédemment, la régression logistique donne de moins bons résultats que pour les perceptrons à 2 et 5 couches, la normalisation de *batch* augmente la performance, tandis que le décrochage a tendance à la réduire (c'est beaucoup moins flagrant que dans le modèle basique). Les moments où le décrochage améliore la performance doivent probablement correspondre à un léger sur-apprentissage, avec des performances d'entraînement de plus de 38% en pratique.

On remarque aussi que les hautes performances (surlignées en jaune) sont plus nombreuses qu'avant : 12 au lieu de 7. De nouveau, le perceptron à 5 couches donne les meilleures performances, avec un maximum d'environ 38%. C'est légèrement mieux que pour le modèle basique. Les vitesses d'apprentissage de 10^{-3} et 10^{-2} sont les plus efficaces. Le temps de calcul total est inférieur à celui du modèle basique, car l'arrêt anticipé a été effectif dans 30 cas sur 68 (au lieu de 13), avec au total 4350 époques (au lieu de 5020).

De la même manière que précédemment, les pires des meilleurs rangs des rotations ont été

calculés. Parmi les 12 meilleures configurations, seules 5 ont un pire des meilleurs rangs convenable : 606, 445, 317, 201, et 337 (les autres ont un pire des meilleurs rangs supérieur à 2300). Parmi ces 5 cas, deux configurations ont été retenues (surlignées en orange) : le perceptron à 5 couches, avec normalisation de *batch*, une probabilité de décrochage de 0.5 et 0.3, et une vitesse d'apprentissage de 10^{-3} et 10^{-1} respectivement.

Le premier cas est toutefois plus rapide et plus robuste aux changements de valeurs de probabilité de décrochage (Tableau 4.8, MLP 5 couches, *batch norm* 1, *dropout* 0.5), avec un pire des meilleurs rangs de 445. Le deuxième cas (Tableau 4.10, MLP 5 couches, *batch norm* 1, *dropout* 0.3, pire des meilleurs rangs 337) est moins robuste, mais les performances par rapport aux autres architectures, aux autres valeurs de probabilité de décrochage, et à l'absence de normalisation de *batch* sont mauvaises, avec des arrêts anticipés fréquents. C'est peut-être plus avantageux lors de l'exploration des différentes valeurs d'hyper-paramètres, car grâce à l'arrêt anticipé, les mauvaises valeurs d'hyper-paramètres sont écartées plus rapidement, ce qui permet de passer plus de temps à chercher autour des meilleures valeurs.

Le Tableau 4.11 présente les informations des meilleurs rangs dans le cas retenu avec une vitesse d'apprentissage de 10^{-3}, et le Tableau 4.12 avec une vitesse d'apprentissage de 10^{-1}.

Tableau 4.11 Informations sur la distribution des meilleurs rangs des rotations dans la première configuration retenue pour la fonction de perte pondérée

Moyenne	Écart type	Minimum	1er quartile	Médiane	3e quartile	Maximum
12.67	22.94	0	1	5	15	445

Tableau 4.12 Informations sur la distribution des meilleurs rangs des rotations dans la deuxième configuration retenue pour la fonction de perte pondérée

Moyenne	Écart type	Minimum	1er quartile	Médiane	3e quartile	Maximum
23.84	38.57	0	1	8	30	337

4.3 Sur-échantillonnage aléatoire pondéré (*weighted random oversampling*)

Cette section présente une autre méthode pour pallier le déséquilibre des données.

4.3.1 Principe

Dans le cas de données très déséquilibrées, il est aussi possible d'utiliser des techniques de sur-échantillonnage. Ce sont des méthodes qu'on applique sur l'ensemble des données avant même

la phase d'entraînement. Le but est de transformer artificiellement l'ensemble de données déséquilibré en un ensemble équilibré de manière cohérente. Après cela, l'apprentissage doit être plus performant, car il a l'impression d'un équilibre entre les différentes classes (une classe étant un ensemble d'échantillons partageant le même label).

Le sur-échantillonnage aléatoire pondéré consiste à tirer aléatoirement avec remise les échantillons de données, de telle sorte que toutes les classes aient la même probabilité d'être tirée. Dans notre cas, s'il y a n_0 échantillons avec un label 0 et n_1 avec un label 1, alors il suffit de choisir les probabilités $\frac{r}{n_0}$ pour les échantillons de la classe 0 et $\frac{1-r}{n_1}$ pour ceux de la classe 1, où $r = \frac{n_1}{n_0}$. Ainsi, les données ré-échantillonnées seront statistiquement équilibrées, avec quasiment autant de labels 1 que de labels 0.

Si on garde le même nombre d'échantillons, cela implique donc que les échantillons de la classe minoritaire seront souvent répétés, et que des échantillons de la classe majoritaire (environ la moitié dans ce cas) seront ignorés complètement durant l'apprentissage. On parle parfois de sous-échantillonnage de la classe majoritaire. Ici, cela importe peu car les échantillons ignorés n'apportent que peu d'information par rapport à ceux qui ont été ré-échantillonnés, et parce que l'apprentissage met l'accent sur les labels 1.

La Figure 4.2 illustre les cas de sous-échantillonnage, de sur-échantillonnage et d'une combinaison des deux.

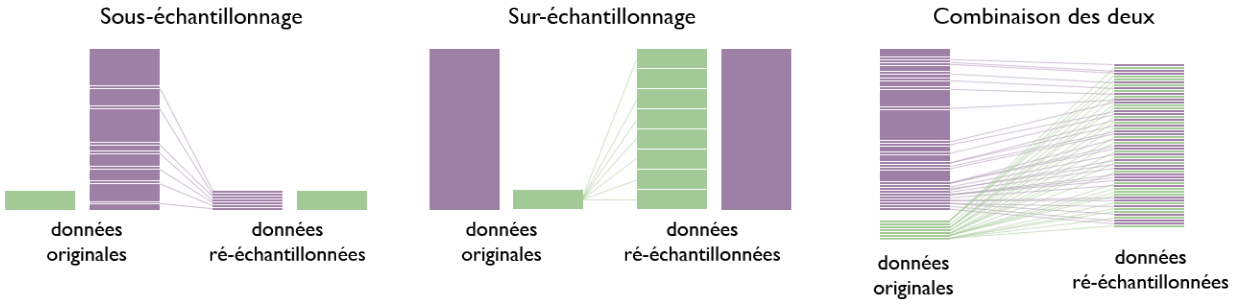


Figure 4.2 Sous-échantillonnage et sur-échantillonnage

4.3.2 Résultats

Les Tableaux 4.13, 4.14, 4.15, 4.16 présentent les résultats obtenus avec le sur-échantillonnage aléatoire pondéré. Le temps total de calcul est d'environ 649 heures.

Tableau 4.13 Performances de test pour le modèle de sur-échantillonnage aléatoire pondéré, vitesse d'apprentissage = 10^{-4}

	Rég. log.	**MLP 2 couches**								**MLP 5 couches**							
		Batch norm 0				*Batch norm* 1				*Batch norm* 0				*Batch norm* 1			
Dropout		0.0	0.3	0.5	0.7	0.0	0.3	0.5	0.7	0.0	0.3	0.5	0.7	0.0	0.3	0.5	0.7
Top 500	22.8	33.5	32.84	30.2	27.26	23.73	23.28	23.01	20.71	33.99	31.66	29.67	21.51	18.58	22.55	21.32	20.81
Top 1000	41.69	51.68	50.84	49.07	46.76	38.56	39.63	39.27	37.43	51.32	49.58	48.35	38.8	31.41	38.25	38.1	36.96
Top 2000	62.19	75.31	73.85	73.57	71.74	60.26	59.45	60.29	59.13	73.98	73.46	73.05	65.44	48.9	58.69	59.57	58.1
Top 3000	78.02	92.48	91.31	90.84	90.44	77.44	76.95	77.68	76.17	91.06	90.94	90.97	86.9	66.79	76.19	76.85	75.7
Top 4000	91.45	97.58	97.4	97.51	97.28	90.36	89.51	89.79	89.29	97.27	97.33	97.35	96.15	84.45	89.78	89.83	89.69
Top 5000	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
Early stop	40	80	80	80	80	30	80	80	40	50	80	80	80	80	70	70	80
Temps (h)	4	10.5	12.3	11.6	8.3	3.4	13.2	11.2	6.5	8.7	15.4	14.9	14.4	6.9	16.6	16.1	20.2

Tableau 4.14 Performances de test pour le modèle de sur-échantillonnage aléatoire pondéré, vitesse d'apprentissage = 10^{-3}

	Rég. log.	**MLP 2 couches**								**MLP 5 couches**							
		Batch norm 0				*Batch norm* 1				*Batch norm* 0				*Batch norm* 1			
Dropout		0.0	0.3	0.5	0.7	0.0	0.3	0.5	0.7	0.0	0.3	0.5	0.7	0.0	0.3	0.5	0.7
Top 500	24.07	34.86	37.8	36.83	34.62	15.57	23.89	22.72	21.92	34.37	36.73	36.91	35.59	15.39	21.62	21.7	21.6
Top 1000	38.74	51.55	54.46	53.14	52.05	33.03	38.78	38.86	38.58	49.77	51.23	52.01	51.96	29.25	38.57	38.19	38.36
Top 2000	62.26	72.09	75.46	75.04	73.89	51.11	58.3	60.41	59.57	69.98	71.44	72.98	73.77	49.49	57.8	59.76	59.19
Top 3000	78.11	90.03	92.07	91.86	91.2	72.76	76.19	77.34	76.45	89	89.5	89.93	91.27	65.81	74.77	77.05	76.76
Top 4000	91.55	97.1	97.45	97.7	97.43	87.81	90.09	89.35	89.76	96.81	96.75	96.87	97.19	81.68	88.77	90.02	89.98
Top 5000	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
Early stop	50	80	80	80	80	30	50	30	30	30	50	80	80	80	70	30	40
Temps (h)	5.2	10.1	11.3	11.6	8.4	4.7	8.5	4.8	4.8	5.4	9.8	15	14.3	9	16.6	7.1	10

Tableau 4.15 Performances de test pour le modèle de sur-échantillonnage aléatoire pondéré, vitesse d'apprentissage = 10^{-2}

	Rég. log.	**MLP 2 couches**								**MLP 5 couches**							
		Batch norm 0				*Batch norm* 1				*Batch norm* 0				*Batch norm* 1			
Dropout		0.0	0.3	0.5	0.7	0.0	0.3	0.5	0.7	0.0	0.3	0.5	0.7	0.0	0.3	0.5	0.7
Top 500	24.02	36.17	37.04	37.45	34.54	20.02	21.23	22.74	21.35	30.96	34.86	35.5	33.97	15.99	20.03	21.48	21.37
Top 1000	38.62	50.2	53.27	53.05	51.69	34.38	38.16	39.04	37.56	43.62	49.48	50.03	50.05	26.54	35.8	38.06	38.08
Top 2000	62.29	68.75	75.32	74.33	73.75	51.61	57.86	58.28	56.58	61.62	69.42	70.05	71.49	43.72	54.39	58.28	58.43
Top 3000	78.11	88.35	92.01	91.17	91.23	71.93	75.76	76.08	74.16	85.66	88.06	88.87	89.24	65.22	71.6	75.05	75.83
Top 4000	91.45	96.65	97.52	97.15	97.34	88.99	89.94	89.67	88.13	95.68	96.6	96.62	96.74	86.61	87.91	89.02	89.43
Top 5000	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
Early stop	50	80	80	80	80	30	50	60	80	30	50	50	60	80	30	40	80
Temps (h)	5.1	10.6	11.4	11.9	11.4	4.6	8.5	9.8	12.8	5.3	9.4	10.8	10.8	18.3	6.9	10.1	20

Tableau 4.16 Performances de test pour le modèle de sur-échantillonnage aléatoire pondéré, vitesse d'apprentissage = 10^{-1}

	Rég. log.	**MLP 2 couches**								**MLP 5 couches**							
		Batch norm 0				*Batch norm* 1				*Batch norm* 0				*Batch norm* 1			
Dropout		0.0	0.3	0.5	0.7	0.0	0.3	0.5	0.7	0.0	0.3	0.5	0.7	0.0	0.3	0.5	0.7
Top 500	23.66	32.45	37.17	36.68	36.59	20.39	21.44	22.14	21.19	29.29	32.64	34.76	34.46	17.48	13.78	18.09	18.71
Top 1000	38.24	47.25	52.53	52.5	53.12	31.42	37.21	37.42	38.03	41.38	46.6	50.03	50.66	29.41	26.8	33.29	32.6
Top 2000	62.34	66.36	73.53	74.2	74.22	48.95	55.92	56.71	57.51	59.83	66.34	70.63	71.87	46.41	49.16	54.24	53.21
Top 3000	78.11	87.99	90.79	90.86	91.49	71.89	74.74	74.85	75.56	84.05	87.11	89	89.83	67.18	69.32	72.23	71.88
Top 4000	91	96.39	97.06	97.31	97.4	88.85	89.58	89.63	88.93	95.61	96.1	96.71	96.6	85.6	86.9	87.97	88.2
Top 5000	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
Early stop	60	30	70	40	80	30	30	30	40	30	40	60	70	30	30	30	30
Temps (h)	5.8	4	9.6	5.8	11.4	4.4	5.2	4.6	6.6	5.5	7.6	11.8	12.2	6.9	7.1	5.8	7.3

On remarque ici que la normalisation de *batch* diminue la performance. Une possible explication est la faible profondeur des réseaux, car la normalisation de *batch* a été conçue pour éviter les effets de la propagation du gradient à travers les couches. La régression logistique donne d'ailleurs des résultats comparables voire meilleurs que ceux des perceptrons avec normalisation de *batch*. Mais les résultats de la régression logistique sont moins bons que ceux des perceptrons multicouches lorsqu'il n'y a pas de normalisation de *batch*.

Comparés aux cas précédents, le décrochage a encore moins tendance à diminuer les performances. Les moments où le décrochage améliore la performance correspondent à des performances d'entraînement supérieures à 40%, ce qui indique probablement le sur-apprentissage. On remarque aussi que les hautes performances (surlignées en jaune) sont aussi nombreuses que pour le modèle de la fonction de perte pondérée, avec 12 performances supérieures à 35%. La performance maximale est d'environ 38%, ce qui est aussi similaire à celle du modèle précédent. Cette fois-ci, le perceptron à 2 couches donne de meilleurs résultats que celui à 5 couches. Les vitesses d'apprentissage de 10^{-3} et 10^{-2} sont les plus efficaces. Le temps de calcul total est encore inférieur à celui du modèle de la fonction de perte pondérée, car l'arrêt anticipé a été effectif dans 42 cas sur 68 (au lieu de 30 et 12), avec un total de 3890 époques (au lieu de 5020 et 4350).

De la même manière que précédemment, les pires des meilleurs rangs des rotations ont été calculés. Parmi les 12 meilleures configurations, 8 ont un pire des meilleurs rangs convenable : 364, 330, 324, 347, 364, 248, 254 et 261 (les autres ont un pire des meilleurs rangs supérieur à 1900). Parmi ces 8 cas, deux configurations ont été retenues (surlignées en orange) : le perceptron à 2 couches, sans normalisation de *batch*, une probabilité de décrochage de 0.3 et 0.5 respectivement, une vitesse d'apprentissage de 10^{-3} et 10^{-2}, et un pire des meilleurs rangs de 364 dans les deux cas. Les deux cas semblent équivalents, même si on remarque que

les performances pour une vitesse d'apprentissage de 10^{-3} (Tableau 4.14) sont légèrement supérieures.

Le Tableau 4.17 présente les informations des meilleurs rangs dans le cas retenu avec une vitesse d'apprentissage de 10^{-3}, et le Tableau 4.18 avec une vitesse d'apprentissage de 10^{-2}.

Tableau 4.17 Informations sur la distribution des meilleurs rangs des rotations dans la première configuration retenue pour l'échantillonnage aléatoire pondéré

Moyenne	Écart type	Minimum	1er quartile	Médiane	3e quartile	Maximum
59.89	75.06	0	2	16.5	114.75	364

Tableau 4.18 Informations sur la distribution des meilleurs rangs des rotations dans la deuxième configuration retenue pour l'échantillonnage aléatoire pondéré

Moyenne	Écart type	Minimum	1er quartile	Médiane	3e quartile	Maximum
35.62	48.59	0	2	11	56	364

4.4 SMOTE (*Synthetic Minority Oversampling Technique*)

Cette section présente encore un autre moyen de pallier le déséquilibre des données.

4.4.1 Principe

SMOTE est une autre méthode de sur-échantillonnage, mais qui va générer artificiellement des nouveaux échantillons de la classe minoritaire, alors que la méthode précédente répétait des échantillons de la classe minoritaire déjà existants.

Cette méthode est décrite dans l'article de Bowyer et al. [7]. D'après les auteurs, cette méthode a été inspirée des problèmes de reconnaissance de chiffres manuscrits, où de nouveaux échantillons étaient générés en appliquant des opérations simples comme des rotations ou des translations de l'image, puis en donnant le même label que celui de l'échantillon original à ces nouveaux échantillons. C'est une méthode de génération de nouveaux échantillons assez naturelle. Avec SMOTE, les nouveaux échantillons de la classe minoritaire sont définis comme étant des points sur des segments reliant 2 vrais échantillons de la classe minoritaire assez proches l'un de l'autre (parmi les k plus proches voisins de l'un ou de l'autre), où le segment est défini dans l'espace des caractéristiques (*features*). C'est une méthode d'interpolation.

La démarche est la suivante. On se donne un entier k, par exemple $k = 6$. L'objectif est de multiplier la population de la classe minoritaire par p. Alors pour chaque échantillon de

la classe minoritaire, on choisit au hasard un de ses $k = 6$ plus proches voisins de la classe minoritaire (distance définie dans l'espace des caractéristiques). Puis, on choisit au hasard un point entre ces deux échantillons, on lui donne le label minoritaire, et c'est le nouvel échantillon de la classe minoritaire. On répète cette procédure p fois pour chaque échantillon de la classe minoritaire. Les extrémités des segments considérés sont toujours des échantillons réels (et pas artificiellement créés au fur et à mesure). Si p est un réel non entier, il suffira de choisir au hasard des éléments de la classe minoritaire pour compléter la partie décimale de p.

La Figure 4.3 illustre le sur-échantillonnage SMOTE sur un exemple de données, et a été tirée de [41]. Ainsi on peut distinguer les segments reliant des points de la classe minoritaire dans les données ré-échantillonnées.

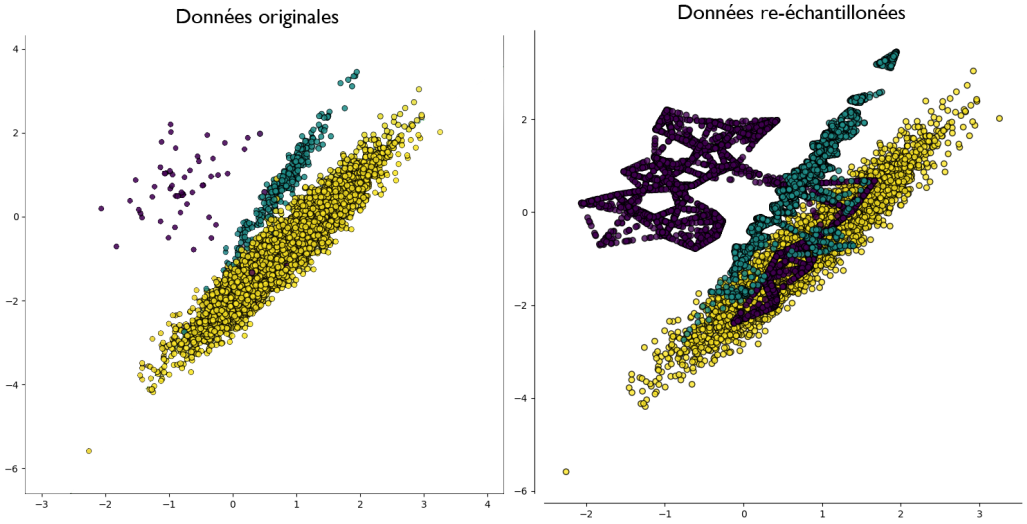


Figure 4.3 SMOTE

4.4.2 Résultats

On remarque que cette méthode ne traite pas du tout les échantillons de la classe majoritaire. Ainsi, dans le cadre de ce projet, cela implique de doubler la taille des données d'entraînement, ce qui a soulevé des défis de mémoire, et cela a bien sûr augmenté les temps de calculs. Ainsi, la performance du modèle toutes les 10 époques a été calculée sur des CPU, au lieu d'être sur GPU. Les expériences présentées ci-dessous ont été réalisées avec la librairie Python `imblearn`.

Les Tableaux 4.19, 4.20, 4.21, 4.22 présentent les résultats obtenus avec SMOTE. Le temps total de calcul est d'environ 1270 heures, avec en tout 4070 époques. Ainsi, on passe environ

19 minutes par époque au lieu de 11 minutes environ pour les 3 modèles précédents. C'est cohérent car la taille des données a doublé.

Tableau 4.19 Performances de test pour SMOTE, vitesse d'apprentissage = 10^{-4}

	Rég. log.	**MLP 2 couches**								**MLP 5 couches**							
		Batch norm 0				*Batch norm* 1				*Batch norm* 0				*Batch norm* 1			
Dropout		0.0	0.3	0.5	0.7	0.0	0.3	0.5	0.7	0.0	0.3	0.5	0.7	0.0	0.3	0.5	0.7
Top 500	22.59	34.61	34.76	34.15	30.81	19.66	22.51	22.55	21.49	33.05	33.93	31.1	30.03	8.98	20.92	21.01	20.55
Top 1000	39.45	50.38	51.58	50.77	48.51	30.53	38.1	38.64	37.33	46.14	49.1	48.13	47.58	17	35.87	36.74	35.69
Top 2000	61.09	70.43	74.06	73.16	72.24	48.98	58.42	58.6	57.92	65.03	70.04	71.46	71.22	36.9	56.03	56.96	56.38
Top 3000	76.81	89.1	91.21	91.01	90.74	71.38	75.98	76.15	76.09	87.18	89.58	89.97	90.06	64.16	73.16	74.66	74.83
Top 4000	90.96	96.74	97.5	97.32	97.03	88.25	90.01	89.88	89.53	96.55	97	97.25	97.06	81.3	89.3	89.56	90.1
Top 5000	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
Early stop	80	80	80	80	80	30	40	50	50	80	80	80	80	80	50	40	80
Temps (h)	14.5	19.7	20	20.2	19.9	8.4	11.8	14.9	14.4	25.2	26.3	26.6	26.7	33.4	22.1	18.1	35.6

Tableau 4.20 Performances de test pour SMOTE, vitesse d'apprentissage = 10^{-3}

	Rég. log.	**MLP 2 couches**								**MLP 5 couches**							
		Batch norm 0				*Batch norm* 1				*Batch norm* 0				*Batch norm* 1			
Dropout		0.0	0.3	0.5	0.7	0.0	0.3	0.5	0.7	0.0	0.3	0.5	0.7	0.0	0.3	0.5	0.7
Top 500	22.4	36.11	36.25	35.8	35.36	8.88	18.99	19.55	19.72	33.47	34.86	35.06	33.65	9.33	17.95	19.4	19.2
Top 1000	41.22	49.25	51.28	51.63	50.7	17.07	34.82	35.36	36.04	46.77	48.31	49.26	49.03	17.15	32.42	34.05	34.77
Top 2000	61.1	68.36	73.07	72.77	72.48	36.25	54.79	55.62	57.84	66.02	68.49	69.62	69.78	36.06	51.31	54.2	55.29
Top 3000	76.43	88.51	91.13	90.53	90.46	65.05	73.26	74.17	75.82	87.6	88.83	89.12	88.54	62.87	71.83	73.15	74.6
Top 4000	90.67	96.93	97.59	97.18	97.01	84.57	88.85	89.94	89.88	96.59	96.95	96.92	96.65	80.72	89	88.17	90.05
Top 5000	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
Early stop	80	80	60	80	80	80	30	30	80	40	80	50	70	40	50	80	30
Temps (h)	14.6	20	14.8	20.5	20.3	22.7	8.8	8.8	23.4	12.4	26.9	17	23.2	16.9	22.5	35.9	13.3

Tableau 4.21 Performances de test pour SMOTE, vitesse d'apprentissage = 10^{-2}

	Rég. log.	**MLP 2 couches**								**MLP 5 couches**							
		Batch norm 0				*Batch norm* 1				*Batch norm* 0				*Batch norm* 1			
Dropout		0.0	0.3	0.5	0.7	0.0	0.3	0.5	0.7	0.0	0.3	0.5	0.7	0.0	0.3	0.5	0.7
Top 500	22.38	34.61	35.53	35.76	35.02	9.59	15.8	17.45	19.27	32.34	33.16	34.36	33.98	10.24	14.03	16.64	16.63
Top 1000	41.16	48.12	50.98	51.45	51.06	18.13	27.93	28.9	34.31	44.66	46.27	48.9	48.44	17.59	28.33	30.67	31.15
Top 2000	61.1	67.19	71.9	73.09	73.18	37.29	48.23	47.84	54.43	63.86	66.64	69.15	68.72	35.87	50.51	52.95	50.85
Top 3000	76.46	88.71	90.18	90.26	90.57	64.85	70.4	68.91	73.06	87	87.73	88.45	88.61	64.89	68.7	71.92	69.2
Top 4000	90.66	96.98	97.2	97.06	97.18	84.69	86.3	85.84	87.83	96.53	96.62	96.72	96.71	85.74	85	87.85	87.72
Top 5000	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
Early stop	70	50	60	50	50	30	80	60	30	30	60	30	80	40	30	30	30
Temps (h)	12.9	12	15.1	12.9	12.5	8.5	23	17.7	8.8	9.6	20	10.2	26.8	17.1	13.6	13.6	13

Tableau 4.22 Performances de test pour SMOTE, vitesse d'apprentissage = 10^{-1}

	Rég. log.	**MLP 2 couches**								**MLP 5 couches**							
		Batch norm 0				*Batch norm* 1				*Batch norm* 0				*Batch norm* 1			
Dropout		0.0	0.3	0.5	0.7	0.0	0.3	0.5	0.7	0.0	0.3	0.5	0.7	0.0	0.3	0.5	0.7
Top 500	22.91	34.22	35.4	35.66	34.23	8.93	17.67	17.05	18.57	31.03	32.93	34.21	34.13	12.15	13.02	15.88	15.66
Top 1000	40.77	47.68	51.12	51.39	50.33	17.64	28.75	28.73	32.88	42.5	46.48	48.16	48.65	21.23	26.68	29.54	28.1
Top 2000	61.09	66.48	72.01	72.18	72.53	38.07	46.84	48.8	52.18	61.83	66.98	68.63	68.98	40.99	47.89	48.75	47.27
Top 3000	76.53	88.95	90.23	90.06	90.71	66.63	68.45	71.67	71.59	85.66	87.57	88.63	88.71	67.5	65.74	66.79	65.62
Top 4000	90.75	97.04	97.09	97.37	97.33	86.3	86.13	88.39	88.51	95.34	96.47	96.68	96.63	85.11	83.06	85.63	84.91
Top 5000	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
Early stop	50	80	80	60	60	60	80	80	50	70	30	80	70	40	80	80	30
Temps (h)	12.9	12	15.1	12.9	12.5	8.5	23	17.7	8.8	9.6	20	10.2	26.8	17.1	13.6	13.6	13

On remarque ici que la normalisation de *batch* dégrade la performance. De même qu'avant, c'est peut-être dû à la faible profondeur des réseaux. La régression logistique donne d'ailleurs des résultats meilleurs que ceux des perceptrons avec normalisation de *batch*. Mais les résultats de la régression logistique sont moins bons que ceux des perceptrons multicouches lorsqu'il n'y a pas de normalisation de *batch*.

Comparés aux cas précédents, le décrochage améliore les performances. Le sur-apprentissage semble plus fréquent, et lorsque la performance d'entraînement est haute sans décrochage, le décrochage fait baisser celle-ci, et fait potentiellement augmenter la performance de test. Le décrochage sert donc bien à régulariser l'apprentissage. Par exemple, pour le perceptron à 2 couches, sans normalisation de *batch*, et une vitesse d'apprentissage de 10^{-2}, sans décrochage, la performance d'entraînement atteint 41%, et avec une probabilité de décrochage de 0.3, 0.5, 0.7, elle est de 37.5%, 36% et 34.8%. D'autre part, dans le Tableau 4.21, on constate que la performance de test a tendance à augmenter dans ces cas.

Cette fois-ci aussi, le perceptron à 2 couches donne de meilleurs résultats que celui à 5 couches. Les vitesses d'apprentissage de 10^{-3} et 10^{-2} sont les plus efficaces. La performance maximale est d'environ 36%, ce qui est moins bien que celle des deux modèles précédents, et même que celle du modèle basique. Peut-être que cette méthode de sur-échantillonnage marche moins bien, car l'interpolation qui génère des nouveaux échantillons de la classe minoritaire n'est pas très pertinente. En effet, la distance utilisée pour l'interpolation est une distance dans l'espace des caractéristiques. Ici, 64 des 65 caractéristiques sont binaires, donc la distance associée à celles-ci prendra des valeurs discrètes, ce qui rend la notion de plus proches voisins entre échantillons moins pertinente. Les nouveaux échantillons ont aussi potentiellement plus de caractéristiques réelles, et la perte de la structure binaire rend peut-être l'apprentissage difficile.

Par ailleurs, on observe 10 cas où la performance est supérieure à 35%. Les pires des meilleurs rangs des rotations ont aussi été calculés, et seules deux configurations d'hyper-paramètres ont un pire des meilleurs rangs convenable : 237, et 193 (les autres ont un pire des meilleurs rangs supérieur à 2800). Celles-ci sont surlignées en orange dans les tableaux ci-dessus, avec un perceptron à 2 couches, sans normalisation de *batch*, une probabilité de décrochage de 0.7, et une vitesse d'apprentissage de 10^{-3} et 10^{-2}. Ces deux cas semblent équivalents, même si on remarque que les performances pour une vitesse d'apprentissage de 10^{-3} (Tableau 4.20) sont légèrement supérieures, quoique l'apprentissage dans le deuxième cas est plus rapide. On remarque à nouveau que le modèle avec SMOTE est moins intéressant que les trois modèles précédents.

Le Tableau 4.23 présente les informations des pires des meilleurs rangs dans le cas retenu avec une vitesse d'apprentissage de 10^{-3}, et le Tableau 4.24 avec une vitesse d'apprentissage de 10^{-2}.

Tableau 4.23 Informations sur la distribution des meilleurs rangs des rotations dans la première configuration retenue pour SMOTE

Moyenne	Écart type	Minimum	1er quartile	Médiane	3e quartile	Maximum
28.58	37.13	0	1	9	50	237

Tableau 4.24 Informations sur la distribution des meilleurs rangs des rotations dans la deuxième configuration retenue pour SMOTE

Moyenne	Écart type	Minimum	1er quartile	Médiane	3e quartile	Maximum
25.59	34.98	0	1	8	41	193

4.5 AdaM : Optimiseur à moments adaptatifs

4.5.1 Principe

AdaM (moments adaptatifs ou *adaptive moments*) est un autre optimiseur, souvent utilisé en apprentissage, et un peu différent de la descente de gradient stochastique. Il combine deux méthodes : la méthode des moments et RMSProp. Le contenu de cette section a été largement repris de [27] et de [49].

Les moments

La méthode des moments a été introduite pour éviter les oscillations de la descente de gradient dans les cas où la fonction de perte ressemble à une fonction quadratique ayant une hessienne mal conditionnée. La Figure 4.4 en présente un exemple en dimension 2 (i.e. avec 2 paramètres dans le modèle d'apprentissage). Dans une direction, la fonction décroît rapidement, et dans une autre direction, elle décroît lentement. Le gradient va donc être très proche de la direction de forte décroissance, et la descente de gradient se retrouve à faire des va-et-vient au lieu de descendre rapidement vers le minimum. Pour pallier ce problème, on peut utiliser la méthode de Newton, classique en optimisation, qui utilise la matrice hessienne. Toutefois, en apprentissage profond, les paramètres sont très nombreux, et donc la hessienne est de très grande taille, ce qui pose des défis de calculs et de mémoire.

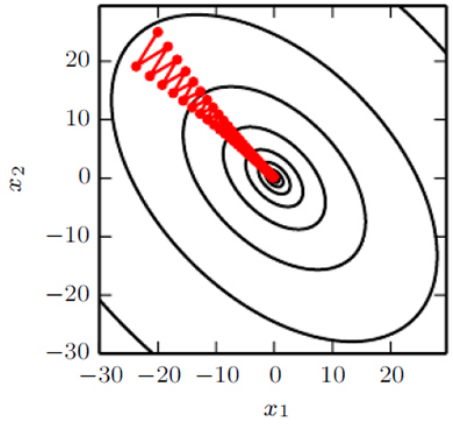


Figure 4.4 Descente de gradient sur une fonction quadratique mal conditionnée

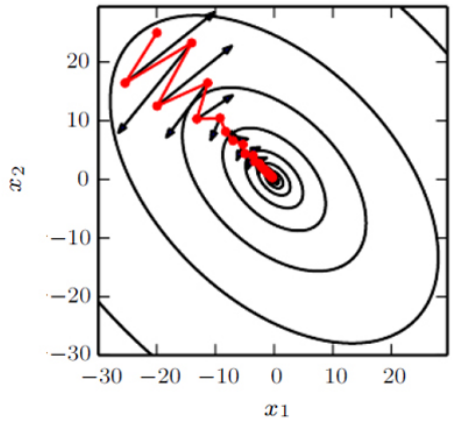


Figure 4.5 Descente de gradient avec un moment sur la même fonction quadratique mal conditionnée

Images tirées de [27]

La méthode des moments consiste à accumuler dans une variable v la moyenne mobile pondérée exponentielle (*exponentially weighted moving average*) des gradients des itérations passées. Pour cela, on définit un hyper-paramètre $\beta_1 \in]0, 1[$ pour la moyenne mobile. Si on note g_t le gradient de la fonction de perte par rapport à tous les paramètres à l'itération t, θ les paramètres du modèle, et ε la vitesse d'apprentissage (*learning rate*), alors la mise à jour à l'itération t pour la méthode des moments sera :

$$v \leftarrow \beta_1 v + (1 - \beta_1) g_t \tag{4.3}$$

$$\theta \leftarrow \theta - \varepsilon v \tag{4.4}$$

Ainsi, la descente ne se fait plus selon le gradient, qui est la direction de la plus forte pente. Désormais, le pas prend en compte les gradients des itérations passées, sachant que les itérations les plus anciennes ont de moins en moins d'influence sur le pas, et que les itérations les plus récentes ont beaucoup plus de poids.

Par conséquent, si les gradients des itérations récentes sont tous dans la même direction, la descente se fera dans cette même direction. Dans le cas où les itérations récentes ont des directions presque opposées, par exemple au pic d'une oscillation (cf Figure 4.4), le prochain pas sera ralenti dans la direction de l'oscillation, pour se rapprocher peu à peu de la direction du minimum local. La Figure 4.5 applique cette méthode au cas de la Figure 4.5. Les flèches noires représentent les directions de plus forte descente, celles qui auraient été choisies sans la méthode des moments.

L'appellation des "moments" vient de l'analogie avec le mouvement d'une particule. Le moment ici ressemble à une force de frottement visqueuse en $-v(t)$, où $v(t)$ est la vitesse de la particule. Cette force de frottement visqueuse va faire perdre de l'énergie à la particule lorsqu'elle fait les va-et-vient de la Figure 4.4, pour arriver plus vite sur des points de l'espace des paramètres où le gradient pointe en direction du minimum local.

RMSProp

RMSProp (*Root Mean Square Propagation*) est une autre méthode qui améliore la descente de gradient usuelle en adaptant la vitesse d'apprentissage pour chacun des paramètres. En effet, la vitesse d'apprentissage est un hyper-paramètre connu pour avoir une grande influence sur les performances des modèles d'apprentissage. Or, dans la descente de gradient usuelle, il est le même pour tous les paramètres. L'idée RMSProp est de diminuer la vitesse d'apprentissage pour les paramètres faisant beaucoup osciller la fonction de perte, et de l'augmenter pour les autres. Pour cela, on accumule les carrés des gradients des itérations précédentes dans une variable s avec une moyenne mobile pondérée exponentielle (*exponentially weighted moving average*), le carré d'un gradient étant le vecteur contenant les carrés des composants du gradient (carré terme à terme). Cela requiert donc un hyper-paramètre $\beta_2 \in]0, 1[$. En prenant les mêmes notations que dans les équations (4.3) et (4.4), la mise à jour à l'itération t pour la méthode RMSProp sera :

$$s \leftarrow \beta_2 s + (1 - \beta_2) g_t \odot g_t \tag{4.5}$$

$$\theta \leftarrow \theta - \frac{\varepsilon}{\sqrt{s}} \odot g_t \tag{4.6}$$

La division et la racine calculée dans l'équation (4.6) sont des opérations effectuées terme à terme.

L'idée derrière cette méthode est la suivante : prenons l'exemple en dimension 2, avec les paramètres x_1 et x_2, où l'oscillation se fait en direction de x_1 alors que le minimum local est en direction de x_2. Alors le gradient suivant x_1 est grand tandis que celui suivant x_2 est petit. Alors $g_t \odot g_t$ suivant x_1 est grand, donc s suivant x_1 est grand, donc $\frac{\varepsilon}{\sqrt{s}}$ suivant x_1 est petit, c'est-à-dire que la vitesse d'apprentissage suivant x_1 est petit. De même, la vitesse d'apprentissage suivant x_2 est plus grand. Ainsi on a atténué les oscillations indésirables et accéléré la progression vers le minimum local.

À noter qu'en pratique il est important d'introduire un autre hyper-paramètre δ très petit, et l'équation (4.6) devient :

$$\theta \leftarrow \theta - \frac{\varepsilon}{\delta + \sqrt{s}} \odot g_t \tag{4.7}$$

Ce δ sert à la stabilité numérique, et sa valeur suggérée est 10^{-8}.

AdaM

AdaM combine les moments et RMSProp. On dispose de deux hyper-paramètres β_1 et β_2 dans $]0, 1[$. On note v et s les deux variables qui accumulent les moyennes mobiles pondérées exponentielles des gradients et des carrés des gradients, qui sont toutes les deux initialisées à 0. À l'itération t, on fait les mises à jour suivantes :

$$v \leftarrow \beta_1 v + (1 - \beta_1) g_t \tag{4.8}$$

$$s \leftarrow \beta_2 s + (1 - \beta_2) g_t \odot g_t \tag{4.9}$$

AdaM va en plus corriger les biais de v et s induits par les moyennes mobiles (pour $t > 0$) :

$$v \leftarrow \frac{v}{1 - \beta_1^t} \tag{4.10}$$

$$s \leftarrow \frac{s}{1 - \beta_2^t} \tag{4.11}$$

Et enfin, la mise à jour des paramètres se fait selon l'équation (4.7) en remplaçant g_t par v

$$\theta \leftarrow \theta - \frac{\varepsilon}{\delta + \sqrt{s}} \odot v \tag{4.12}$$

où $\delta = 10^{-8}$ usuellement. Les valeurs recommandées pour les hyper-paramètres β_1 et β_2 (appelés *decay rates*) sont respectivement 0.9 et 0.999. Ces trois hyper-paramètres sont rarement optimisés.

4.5.2 Résultats

Le Tableau 4.25 présente sous la forme d'une carte de température, les résultats des modèles des 4 sections précédentes en utilisant l'optimiseur AdaM, et le Tableau 4.26 résume ceux avec l'optimiseur SGD. Seules les performances pour le Top 500 ont été présentées. *lr* signifie *learning rate* (vitesse d'apprentissage). Les abréviations de la première colonne signifient dans l'ordre : "Basique", "Perte pondérée", et "Échantillonnage aléatoire pondéré". Cela correspond aux modèles vus précédemment. Plus la couleur est foncée, plus la performance est haute. 10.7% revient souvent dans les tableaux, car ce nombre correspond au cas où toutes les paires (employé, rotation) ont la même prédiction, et dans ce cas, le classement des rotations par employé se fait selon un ordre par défaut, et il se trouve que la performance vaut 10.7%. Ce sont des cas où la machine n'apprend rien.

Tableau 4.25 Carte de température suivant les performances de test avec AdaM

		AdaM															
		Rég.	MLP 2 couches								MLP 5 couches							
		log.	*Batch norm* 0				*Batch norm* 1				*Batch norm* 0				*Batch norm* 1			
	Dropout		0.0	0.3	0.5	0.7	0.0	0.3	0.5	0.7	0.0	0.3	0.5	0.7	0.0	0.3	0.5	0.7
Basiq.	*lr* 10^{-4}	19.5	26.9	20.9	16.6	12.1	35.3	30.7	29.9	28.6	26.1	12.9	10.8	10.7	33.5	31.4	30.6	13.2
	lr 10^{-3}	18.0	13.9	11.7	11.2	10.8	33.2	30.1	26.2	16.3	10.7	10.7	10.7	10.8	34.7	32.3	29.4	26.8
	lr 10^{-2}	19.1	11.9	11.2	10.7	10.8	33.8	10.7	10.7	10.7	10.7	10.7	10.7	10.7	33.5	26.8	10.7	10.7
	lr 10^{-1}	10.7	10.7	10.7	10.7	10.8	10.7	10.7	10.7	10.7	10.7	10.7	10.7	10.7	10.7	10.7	10.7	10.7
Perte	*lr* 10^{-4}	18.7	25.5	21.1	15.8	11.5	34.8	31.7	31.0	28.9	10.7	12.5	10.8	11.5	33.0	33.7	30.8	20.3
pond.	*lr* 10^{-3}	18.0	14.0	11.6	10.6	10.7	33.6	30.2	24.9	12.8	10.7	10.7	10.3	10.4	34.2	31.6	29.9	12.9
	lr 10^{-2}	17.9	10.7	12.1	10.7	10.9	33.5	10.7	10.7	10.7	10.7	10.7	10.7	10.7	34.8	27.2	10.7	10.7
	lr 10^{-1}	16.8	10.7	10.7	10.7	10.7	10.7	10.7	10.7	10.7	10.7	10.7	10.7	10.7	10.7	10.7	10.7	10.7
Échant.	*lr* 10^{-4}	24.1	37.1	36.8	37.1	36.8	15.1	20.5	21.9	20.4	26.5	32.9	35.5	36.8	13.8	12.2	17.3	19.0
aléat.	*lr* 10^{-3}	23.7	31.9	36.7	37.3	35.8	18.3	20.4	20.0	21.1	27.7	34.0	36.2	36.0	10.7	13.4	15.9	18.4
pond.	*lr* 10^{-2}	24.0	39.0	31.4	26.0	19.9	17.5	20.4	19.2	18.8	10.7	10.7	10.7	10.7	19.0	14.8	18.4	18.0
	lr 10^{-1}	23.3	10.7	10.7	10.7	10.7	18.7	14.9	12.8	10.7	10.7	10.7	10.7	10.7	14.2	17.8	12.5	10.7
SMOTE	*lr* 10^{-4}	23.2	35.4	36.2	36.2	36.0	9.8	18.0	17.7	19.6	30.8	33.6	34.6	35.7	14.2	9.8	12.7	15.2
	lr 10^{-3}	23.3	33.8	35.9	35.8	35.4	8.5	17.2	16.5	18.7	30.2	33.4	34.8	24.2	16.5	9.3	12.0	14.5
	lr 10^{-2}	23.4	37.8	33.8	24.4	16.0	9.1	17.6	16.1	17.3	13.9	10.7	10.7	10.7	12.3	11.0	12.3	14.2
	lr 10^{-1}	23.7	10.7	10.7	10.7	10.7	13.7	12.9	12.8	11.15	10.7	10.7	10.7	10.7	10.9	11.2	18.2	15.1

Tableau 4.26 Carte de température suivant les performances de test avec SGD

		SGD																
		Rég.	MLP 2 couches								MLP 5 couches							
		log.	*Batch norm* 0				*Batch norm* 1				*Batch norm* 0				*Batch norm* 1			
	Dropout		0.0	0.3	0.5	0.7	0.0	0.3	0.5	0.7	0.0	0.3	0.5	0.7	0.0	0.3	0.5	0.7
Basiq.	lr 10^{-4}	13.4	15.9	11.9	11.4	11.0	26.7	18.4	15.2	12.8	20.3	13.1	12.1	10.3	30.2	20.3	15.8	10.6
	lr 10^{-3}	19.2	25.5	17.6	15.8	14.3	34.7	29.2	26.2	19.8	30.2	18.7	18.6	11.2	29.3	28.0	23.3	15.6
	lr 10^{-2}	22.7	30.5	28.2	27.2	26.4	34.8	33.2	33.7	31.6	30.9	29.9	29.4	24.5	32.9	37.3	36.8	32.3
	lr 10^{-1}	23.8	30.1	27.8	28.0	26.2	36.0	33.6	33.1	32.6	30.0	28.9	29.4	27.1	35.4	37.4	36.9	36.8
Perte	lr 10^{-4}	23.4	29.0	27.4	26.4	25.2	35.1	34.2	33.5	31.0	29.6	29.3	28.4	28.0	34.5	36.4	33.3	34.3
pond.	lr 10^{-3}	23.7	29.1	28.1	26.7	24.8	34.9	35.6	33.7	31.9	29.1	28.6	28.0	27.7	36.3	37.5	37.4	35.8
	lr 10^{-2}	20.7	28.9	25.8	23.8	20.0	35.2	29.2	27.4	22.4	29.0	24.4	10.7	10.7	36.9	36.3	34.8	35.5
	lr 10^{-1}	19.5	10.7	10.7	10.7	10.7	10.7	10.7	10.7	10.7	10.7	10.8	10.7	10.7	10.7	37.8	10.7	10.7
Échant.	lr 10^{-4}	22.8	33.5	32.8	30.2	27.3	23.7	23.3	23.0	20.7	34.0	31.7	29.7	21.5	18.6	22.6	21.3	20.8
aléat.	lr 10^{-3}	24.1	34.9	37.8	36.8	34.6	15.6	23.9	22.7	21.9	34.4	36.7	36.9	35.6	15.4	21.6	21.7	21.6
pond.	lr 10^{-2}	24.0	36.2	37.0	37.5	34.5	20.0	21.2	22.7	21.4	31.0	34.9	35.5	34.0	16.0	20.0	21.5	21.4
	lr 10^{-1}	23.7	32.5	37.2	36.7	36.6	20.4	21.4	22.1	21.2	29.3	32.6	34.8	34.5	17.5	13.8	18.1	18.7
SMOTE	lr 10^{-4}	22.6	34.6	34.8	34.2	30.8	19.7	22.5	22.6	21.5	33.1	33.9	31.1	30.0	9.0	20.9	21.0	20.6
	lr 10^{-3}	22.4	36.1	36.3	35.8	35.4	8.9	19.0	19.6	19.7	33.5	34.9	35.1	33.7	9.3	18.0	19.4	19.2
	lr 10^{-2}	22.4	34.6	35.5	35.8	35.0	9.6	15.8	17.5	19.3	32.3	33.2	34.4	34.0	10.2	14.0	16.6	16.6
	lr 10^{-1}	22.9	34.2	35.4	35.7	34.2	8.9	17.7	17.1	18.6	31.0	32.9	34.2	34.1	12.2	13.0	15.9	15.7

On remarque qu'AdaM est globalement moins performant avec le modèle basique et celui de la perte pondérée, comparés aux modèles de sur-échantillonnage. On remarque également que les hautes performances sont beaucoup moins fréquentes avec AdaM qu'avec SGD, et de manière générale, AdaM donne des moins bons résultats que SGD. Les meilleurs résultats ne correspondent plus aux mêmes valeurs de vitesse d'apprentissage qu'avec SGD, mais gardent les mêmes choix de normalisation de *batch* (avec normalisation pour "Basique" et "Perte pondérée", et sans pour les deux autres). L'effet du décrochage avec AdaM est similaire à celui avec SGD et est plus flagrant : dans les cas "Basique" et "Perte pondérée", le décrochage a un effet négatif, tandis que dans les deux autres cas, il a plutôt un effet positif. La régression logistique donne de faibles résultats par rapport aux perceptrons, et les deux types de perceptron ont des performances très similaires.

Le Tableau 4.27 présente les moyennes, écarts-types et maxima des performances du Top 500 sur l'ensemble des trois architectures, sur les quatre valeurs de probabilité de décrochage, les quatre valeurs de vitesse d'apprentissage, la présence ou non de normalisation de *batch*, et le choix de l'optimiseur.

Tableau 4.27 Informations sur la distribution de la performance du Top 500 avec AdaM et SGD

	AdaM			**SGD**		
	Moyenne	Écart-type	Maximum	Moyenne	Écart-type	Maximum
Basique	17.10	7.76	35.33	25.02	7.16	37.39
Perte pondérée	16.78	7.53	34.83	25.25	8.00	37.75
Échant. aléat. pond.	21.05	7.75	39.01	26.96	6.64	37.80
SMOTE	19.48	8.34	37.84	25.13	8.39	36.25
Tous les cas	18.60	8.00	39.01	25.59	7.48	37.75

On constate que la performance avec AdaM est en moyenne inférieure de 7% à celle avec SGD, même si AdaM est équivalent à SGD voire meilleur dans les cas de sur-échantillonnage. Que ce soit avec AdaM ou avec SGD, c'est l'échantillonnage aléatoire pondéré qui donne les meilleurs résultats en moyenne et les meilleures performances maximales.

On remarque aussi que les performances avec AdaM sont plus dispersées qu'avec SGD. D'ailleurs le Tableau 4.25 montre que, bien plus souvent qu'avec SGD, AdaM donne des performances très mauvaises de 10.7%. Toutefois, ces mauvaises performances correspondent à des cas où l'arrêt anticipé agit au bout de 30 époques (donc le minimum), comme le montre le Tableau 4.28, qui est une carte de température, où les arrêts les plus anticipés sont les plus foncés. Le Tableau 4.29 présente la même carte de température avec SGD.

Tableau 4.28 Carte de température suivant l'époque d'arrêt anticipé avec AdaM

		AdaM																
		Rég.	MLP 2 couches								MLP 5 couches							
		log.	*Batch norm* 0				*Batch norm* 1				*Batch norm* 0				*Batch norm* 1			
	Dropout		0.0	0.3	0.5	0.7	0.0	0.3	0.5	0.7	0.0	0.3	0.5	0.7	0.0	0.3	0.5	0.7
Basiq.	*lr* 10^{-4}	80	30	60	60	70	80	40	80	80	30	30	80	80	80	80	80	60
	lr 10^{-3}	80	80	30	30	80	30	80	70	30	30	50	80	80	60	80	30	80
	lr 10^{-2}	80	80	80	40	60	80	30	30	30	30	30	40	30	80	30	30	30
	lr 10^{-1}	40	30	80	30	80	30	30	80	30	30	30	30	30	30	30	30	30
Perte	*lr* 10^{-4}	80	30	40	30	80	60	50	80	40	30	80	30	30	50	80	80	80
pond.	*lr* 10^{-3}	60	50	30	80	80	80	80	30	80	30	30	80	50	50	30	50	80
	lr 10^{-2}	30	30	40	30	80	60	30	30	30	30	30	30	30	80	30	80	80
	lr 10^{-1}	30	30	30	30	30	30	30	30	30	30	30	30	30	30	80	30	30
Échant.	*lr* 10^{-4}	80	30	30	30	40	40	80	30	30	30	30	30	30	30	80	40	40
aléat.	*lr* 10^{-3}	40	30	60	70	50	40	30	50	30	60	30	50	30	30	30	30	30
pond.	*lr* 10^{-2}	30	60	30	40	30	70	30	80	80	30	30	30	30	50	30	30	30
	lr 10^{-1}	80	30	30	30	30	70	70	80	30	30	30	30	30	40	30	80	30
SMOTE	*lr* 10^{-4}	80	40	80	80	30	60	40	30	60	70	30	30	80	40	50	80	50
	lr 10^{-3}	80	30	50	80	40	30	30	80	80	30	30	30	30	40	40	30	30
	lr 10^{-2}	70	30	30	40	70	30	40	70	50	30	30	30	30	30	70	30	30
	lr 10^{-1}	50	30	30	30	30	80	30	50	30	30	30	30	30	80	30	30	30

Tableau 4.29 Carte de température suivant l'époque d'arrêt anticipé avec SGD

		SGD															
		Rég.	MLP 2 couches								MLP 5 couches							
		log.	*Batch norm* 0				*Batch norm* 1				*Batch norm* 0				*Batch norm* 1			
	Dropout		0.0	0.3	0.5	0.7	0.0	0.3	0.5	0.7	0.0	0.3	0.5	0.7	0.0	0.3	0.5	0.7
Basiq.	*lr* 10^{-4}	30	80	80	80	80	80	80	80	80	80	80	80	40	80	80	80	30
	lr 10^{-3}	80	80	80	80	80	80	80	80	80	80	80	80	40	60	80	80	80
	lr 10^{-2}	80	80	80	80	80	80	80	80	80	70	80	80	80	50	80	80	80
	lr 10^{-1}	30	80	40	80	80	80	70	80	70	30	80	80	80	60	80	80	80
Perte	*lr* 10^{-4}	50	70	80	80	80	80	80	80	80	80	80	80	80	50	80	60	80
pond.	*lr* 10^{-3}	80	50	80	80	80	80	80	80	70	40	80	80	80	30	80	80	80
	lr 10^{-2}	80	40	80	50	60	80	60	50	40	70	70	80	80	40	50	80	80
	lr 10^{-1}	80	40	40	40	30	30	30	30	30	40	80	30	30	80	80	30	40
Échant.	*lr* 10^{-4}	40	80	80	80	80	30	80	80	40	50	80	80	80	80	70	70	80
aléat.	*lr* 10^{-3}	50	80	80	80	80	30	50	30	30	30	50	80	80	80	70	30	40
pond.	*lr* 10^{-2}	50	80	80	80	80	30	50	60	80	30	50	50	60	80	30	40	80
	lr 10^{-1}	60	30	70	40	80	30	30	30	40	30	40	60	70	30	30	30	30
SMOTE	*lr* 10^{-4}	80	80	80	80	80	30	40	50	50	80	80	80	80	80	50	40	80
	lr 10^{-3}	80	80	60	80	80	80	30	30	80	40	80	50	70	40	50	80	30
	lr 10^{-2}	70	50	60	50	50	30	80	60	30	30	60	30	80	40	30	30	30
	lr 10^{-1}	50	80	80	60	60	60	80	80	50	70	30	80	70	40	80	80	30

On remarque que les faibles performances d'AdaM ont un léger avantage de temps, avec un nombre total d'époques beaucoup plus faible. Ainsi, l'exploration des hyper-paramètres s'attarde moins sur les cas inutiles, ce qui pourrait permettre de se concentrer sur les hautes performances, qui sont plus localisées, tout en étant comparables à celles de SGD.

Le Tableau 4.30 présente les informations sur les temps de calcul totaux avec AdaM et SGD, les nombres d'époques totaux ainsi que le temps de calcul par époque en moyenne.

Tableau 4.30 Informations sur les temps de calcul moyens avec AdaM et SGD

	AdaM			**SGD**		
	Temps (h)	Nb. époques	Temps par époque (min)	Temps (h)	Nb. époques	Temps par époque (min)
Basique	11.1	53	12.7	13.1	74	10.6
Perte pondérée	10.6	48	13.4	11.1	64	10.5
Échant. aléat. pond.	9.1	42	13.1	9.5	57	10
SMOTE	18.5	45	24.7	18.7	60	18.7
Tous les cas	12.3	47	15.8	13.1	63.8	12.3

On remarque donc que les temps de calcul et les nombres d'époques avec AdaM sont systématiquement inférieurs à ceux avec SGD. AdaM économise environ 6% de temps de calcul et 27% d'époques. Cette économie d'époque se fait au prix d'une augmentation du temps

consacré à chaque époque : cela rajoute en moyenne environ 3 minutes par époque sur les 12 minutes de la SGD.

Le Tableau 4.31 présente les résultats de AdaM et SGD avec le bon choix de normalisation de *batch* (*bn* dans le tableau). Les hautes performances (supérieures à 35%) ont été surlignées en jaune, celles ayant en plus un pire des meilleurs rangs des rotations en-dessous de 606 sont en orange clair, et les meilleures performances parmi ces dernières sont en orange foncé. Celles en jaune ont presque toujours un pire des meilleurs rangs supérieur à 1950, ce qui n'est pas convenable. Celles en orange foncé ont toujours un pire des meilleurs rangs inférieur à 500. Les cas en orange foncé sont ceux qu'on retiendra.

Tableau 4.31 Synthèse des résultats avec AdaM et SGD (Top 500 affichés)

		AdaM									**SGD**								
		Rég. log.	MLP 2 couches				MLP 5 couches				Rég. log.	MLP 2 couches				MLP 5 couches			
	Dropout		0.0	0.3	0.5	0.7	0.0	0.3	0.5	0.7		0.0	0.3	0.5	0.7	0.0	0.3	0.5	0.7
Basiq.	*lr* 10^{-4}	19.5	35.3	30.7	29.9	28.6	33.5	31.4	30.6	13.2	13.4	26.7	18.4	15.2	12.8	30.2	20.3	15.8	10.6
bn 1	*lr* 10^{-3}	18	33.2	30.1	26.2	16.3	34.7	32.3	29.4	26.8	19.2	34.7	29.2	26.2	19.8	29.3	28	23.3	15.6
	lr 10^{-2}	19.1	33.8	10.7	10.7	10.7	33.5	26.8	10.7	10.7	22.7	34.8	33.2	33.7	31.6	32.9	37.3	36.8	32.3
	lr 10^{-1}	10.7	10.7	10.7	10.7	10.7	10.7	10.7	10.7	10.7	23.8	36	33.6	33.1	32.6	35.4	37.4	36.9	36.8
Perte	*lr* 10^{-4}	18.7	34.8	31.7	31	28.9	33	33.7	30.8	20.3	23.4	35.1	34.2	33.5	31	34.5	36.4	33.3	34.3
pond.	*lr* 10^{-3}	18	33.6	30.2	24.9	12.8	34.2	31.6	29.9	12.9	23.7	34.9	35.6	33.7	31.9	36.3	37.5	37.4	35.8
bn 1	*lr* 10^{-2}	17.9	33.5	10.7	10.7	10.7	34.8	27.2	10.7	10.7	20.7	35.2	29.2	27.4	22.4	36.9	36.3	34.8	35.5
	lr 10^{-1}	16.8	10.7	10.7	10.7	10.7	10.7	10.7	10.7	10.7	19.5	10.7	10.7	10.7	10.7	10.7	37.8	10.7	10.7
Échant.	*lr* 10^{-4}	24.1	37.1	36.8	37.1	36.8	26.5	32.9	35.5	36.8	22.8	33.5	32.8	30.2	27.3	34	31.7	29.7	21.5
aléat.	*lr* 10^{-3}	23.7	31.9	36.7	37.3	35.8	27.7	34	36.2	36	24.1	34.9	37.8	36.8	34.6	34.4	36.7	36.9	35.6
pond.	*lr* 10^{-2}	24	39	31.4	26	19.9	10.7	10.7	10.7	10.7	24	36.2	37	37.5	34.5	31	34.9	35.5	34
bn 0	*lr* 10^{-1}	23.3	10.7	10.7	10.7	10.7	10.7	10.7	10.7	10.7	23.7	32.5	37.2	36.7	36.6	29.3	32.6	34.8	34.5
SMOTE	*lr* 10^{-4}	23.2	35.4	36.2	36.2	36	30.8	33.6	34.6	35.7	22.6	34.6	34.8	34.2	30.8	33.1	33.9	31.1	30
bn 0	*lr* 10^{-3}	23.3	33.8	35.9	35.8	35.4	30.2	33.4	34.8	24.2	22.4	36.1	36.3	35.8	35.4	33.5	34.9	35.1	33.7
	lr 10^{-2}	23.4	37.8	33.8	24.4	16	13.9	10.7	10.7	10.7	22.4	34.6	35.5	35.8	35	32.3	33.2	34.4	34
	lr 10^{-1}	23.7	10.7	10.7	10.7	10.7	10.7	10.7	10.7	10.7	22.9	34.2	35.4	35.7	34.2	31	32.9	34.2	34.1

Parmi les cas retenus, AdaM donne une performance maximale similaire à celle de SGD, d'environ 37%. Même après avoir sélectionné les configurations ayant un faible pire des meilleurs rangs de rotation, c'est toujours l'échantillonnage aléatoire pondéré, avec le perceptron à deux couches sans normalisation de *batch*, qui donne les meilleurs résultats. Aucune configuration d'hyper-paramètre n'a été retenue avec les modèles "Basique" et "Perte pondérée".

On remarque également que tous les cas retenus ont du décrochage. Le décrochage semble avoir un effet positif sur les pires des meilleurs rangs.

Le Tableau 4.32 présente les informations sur la distribution des meilleurs rangs des rotations avec AdaM, un échantillonnage aléatoire pondéré, un perceptron à 2 couches, une probabilité de décrochage de 0.5, sans normalisation de *batch*, et avec une vitesse d'apprentissage de

10^{-4}. Le pire des meilleurs rangs est 361 dans ce cas.

Tableau 4.32 Informations sur la distribution des meilleurs rangs des rotations dans la meilleure configuration avec AdaM

Moyenne	Écart type	Minimum	1er quartile	Médiane	3e quartile	Maximum
40.98	57.45	0	2	9	68	361

CHAPITRE 5 CONCLUSION ET RECOMMANDATIONS

5.1 Synthèse des travaux

Le but de ce projet est d'utiliser des techniques d'apprentissage machine dans l'optique d'accélérer la génération de colonnes du problème des blocs mensuels d'équipages aériens au niveau des sous-problèmes. L'apprentissage machine a pour objectif de prédire la probabilité qu'une rotation soit dans le bloc mensuel d'un employé. Par la suite, les meilleures rotations pour chaque employé doivent être données au sous-problème correspondant à l'employé, réduisant ainsi la taille du réseau d'un facteur 10 (on parle alors de réseau réduit).

Les caractéristiques d'apprentissage sélectionnées, pour un employé et une rotation donnés, sont : les langues parlées par l'employé ainsi que ses qualifications, les langues et les qualifications requises dans la rotation, et enfin un score de satisfaction de l'employé vis-à-vis de la rotation. Grâce à la collaboration de l'entreprise AD OPT, ce projet porte sur une étude de cas réel.

Plusieurs modèles d'apprentissage ont été testés. Pour chaque modèle, nous avons considéré la régression logistique ainsi que 2 architectures de perceptron multicouche, 4 valeurs de probabilité de décrochage, 4 valeurs de vitesse d'apprentissage, et enfin, l'effet de la normalisation de *batch*. Dans tous les cas, un critère d'arrêt anticipé a été appliqué.

Le premier modèle reste dans le cadre générique d'un perceptron multicouche, utilise l'entropie croisée binaire et la descente de gradient stochastique. Dans le deuxième modèle, nous avons voulu compenser le déséquilibre des données entre les labels 1 et les labels 0 en ajoutant un poids dans la fonction de perte qui favorise la minimisation du rappel plutôt que de l'*accuracy*. En effet, la performance du modèle dépend plus fortement des bonnes prédictions sur les échantillons labellisés 1 que sur ceux labellisés 0. Dans le troisième modèle, un échantillonnage en amont de l'apprentissage a été effectué pour rééquilibrer les données, en répétant aléatoirement les échantillons labellisés 1 quitte à ignorer des échantillons labellisés 0. Le quatrième modèle ré-échantillonne aussi les données, mais en générant des données artificielles par interpolation. Par rapport au modèle de base, ces trois derniers modèles ont amélioré les résultats et la robustesse vis-à-vis des hyper-paramètres considérés.

En outre, plutôt que d'employer la descente de gradient stochastique (SGD), la méthode des moments adaptatifs (AdaM) a été testée. En comparant les résultats de ces deux optimiseurs, il apparaît que AdaM n'apporte pas d'amélioration notable et dégrade en moyenne la performance. En effet, les mauvaises performances sont beaucoup plus nombreuses, les

hautes performances le sont beaucoup moins et sont beaucoup plus localisées. Cependant, cette forte dispersion a l'avantage de donner des temps de calcul plus faibles lors de l'exploration de l'espace des hyper-paramètres. En effet, AdaM permet d'économiser environ 6% de temps de calcul avec 27% d'époques en moins, au prix d'une augmentation du temps de calcul par époque d'environ 28%. Ce temps économisé permettrait alors d'explorer plus de valeurs d'hyper-paramètres, et notamment la vitesse d'apprentissage, qui semble être un hyper-paramètre crucial. Ainsi, SGD est plus robuste, mais AdaM peut plus explorer les valeurs d'hyper-paramètres.

Dans toutes les expériences, une sélection suivant la valeur du pire des meilleurs rangs des rotations a été effectuée. En effet, pour des raisons de réalisabilité, il est nécessaire que toutes les rotations soient présentes dans l'ensemble des réseaux réduits des employés. On remarque alors que le décrochage semble être un bon moyen de diminuer le pire des meilleurs rangs.

Au terme de ces travaux, la meilleure configuration que nous sélectionnons est celle avec un échantillonnage aléatoire pondéré, qui a une performance d'environ 37% sur l'ensemble de test, que ce soit avec AdaM ou avec SGD. Il faut cependant noter que la mesure de performance choisie ici est loin d'être idéale, car elle est calculée par rapport à une unique solution, quand il en existe de nombreuses autres tout aussi bonnes. On espère que la performance, mesurée par rapport à une solution donnée par l'algorithme de génération de colonnes intégrant les informations obtenues par apprentissage, soit meilleure.

5.2 Limites

L'apprentissage machine des données historiques présente quelques limites dans notre cas, car il arrive que certaines affectations de rotation ne doivent pas être reproduites dans le futur. En effet, il existe des rotations indésirables pour tous les employés, souvent sans qualification, et des employés ayant peu de compatibilité avec l'ensemble des rotations possibles, par exemple parce qu'ils parlent peu de langues, ont moins de compétences, et peu de préférences (ou bien des préférences inadaptées). Pour ces rotations et ces employés-là, les affectations peuvent être dues seulement à des obligations de couverture. Ainsi, non seulement les caractéristiques d'apprentissage deviennent non pertinentes, mais aussi ces affectations ne doivent pas être reproduites. Les rotations indésirables doivent être données à des personnes différentes d'un mois sur l'autre pour des raisons d'équité. Or l'apprentissage machine est seulement fait pour reproduire au mieux les données à disposition, donc il peut seulement perpétuer l'iniquité. Ainsi, il conviendrait de coupler l'apprentissage à d'autres techniques complémentaires pour pallier ce problème.

5.3 Travaux futurs

5.3.1 Agrandir la cible d'entraînement

Comme mentionné en 4.1.4, la cible visée est plus petite que celle qu'il faudrait viser dans l'idéal. La solution dont on dispose est une bonne solution, mais il existe de nombreuses autres solutions aussi bonnes et très différentes. Pour agrandir la cible d'entraînement et être plus proche d'une mesure de performance idéale, on peut se procurer plusieurs solutions équivalentes, par exemple 10, d'un même mois avec exactement les mêmes employés et les mêmes rotations. En pratique, on peut obtenir ces 10 solutions pour chaque mois en changeant par exemple l'ordre des employés dans la génération de colonnes. C'est une manière simple et cohérente, et cela suffit à obtenir des solutions complètement différentes. Il est à noter que cette forte augmentation de données n'affectera pas le temps de calcul lors de l'intégration avec la génération de colonnes, puisque l'apprentissage se ferait séparément de l'optimisation.

Ces 10 solutions d'un même mois pourraient être utilisées à la fois pendant l'entraînement et pendant le test. En effet, pendant l'entraînement, on pourrait considérer qu'une combinaison (employé, rotation) a un label 1 si cette combinaison est présente dans une des 10 solutions, ce qui donnerait vraisemblablement 10 fois plus de labels 1 environ, et apporterait plus d'informations pertinentes sur ce qu'est une "bonne" combinaison. Pour aller plus loin, on pourrait même considérer un problème de classification à 11 classes (au lieu de 2), en donnant le label $k \in [\![0, 10]\!]$ à une combinaison (employé, rotation), suivant sa fréquence d'apparition dans les 10 solutions, puis donner plus de poids aux erreurs faites sur les combinaisons apparaissant fréquemment dans la fonction de perte. Ainsi, pendant l'apprentissage, on incite la machine à mieux apprendre sur les combinaisons qui reviennent souvent dans les solutions.

Par ailleurs, ces 10 bonnes solutions peuvent servir pour mieux évaluer la mesure de performance. En effet, on pourrait mesurer la performance d'un modèle en choisissant la meilleure des performances parmi les 10 solutions. On pourrait aussi, de la même manière que précédemment, pondérer la mesure de performance en fonction de la fréquence d'apparition des combinaisons dans les 10 solutions. C'est-à-dire qu'on pourrait donner plus d'importance à la performance concernant les combinaisons (employé, rotation) apparaissant souvent dans les 10 solutions. En effet, si certaines combinaisons apparaissent dans 8 des 10 solutions, il est plus important de bien les prédire comparées à celles apparaissant une seule fois.

5.3.2 Décomposer le score de satisfaction

Une autre piste d'amélioration envisageable est de décomposer le score de satisfaction en 2. En effet, les spécialistes du domaine savent que les préférences concernant les jours de congé

sont très importantes par rapport aux autres types de préférences. Ainsi, au lieu d'avoir un seul scalaire reflétant la satisfaction d'une combinaison (employé, rotation), il y en aurait deux : un concernant les jours de congé, et un autre concernant le reste, ce qui rajouterait une caractéristique d'apprentissage.

5.3.3 Intégration avec la génération de colonnes

Enfin, pour les travaux futurs, il conviendrait d'intégrer les prédictions de l'apprentissage machine à l'algorithme de génération de colonnes de chez AD OPT pour avoir une mesure plus proche de la mesure de performance idéale décrite dans la section 4.1.4. Pour s'assurer du bon fonctionnement de cette intégration, il faut également s'assurer que les contraintes globales mentionnées dans la section 1.2.2 soient respectées : le réseau réduit de chaque employé doit contenir des rotations réparties à peu près uniformément sur le mois, et chaque rotation doit apparaître dans les réseaux réduits de suffisamment d'employés. Ceci n'a pas pu être considéré dans ce projet par manque de temps.

RÉFÉRENCES

[1] Ranga Anbil, John J Forrest, et William R Pulleyblank. Column generation and the airline crew pairing problem. *Documenta Mathematica*, 3(1) :677, 1998.

[2] Atilim Gunes Baydin, Barak A. Pearlmutter, Alexey Andreyevich Radul, et Jeffrey Mark Siskind. Automatic differentiation in machine learning : a survey. *Journal of Machine Learning Research*, 18(153) :1–43, 2018. URL `http://jmlr.org/papers/v18/17-468.html`.

[3] Irwan Bello, Barret Zoph, Vijay Vasudevan, et Quoc V Le. Neural optimizer search with reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 459–468. JMLR. org, 2017.

[4] Yoshua Bengio, Andrea Lodi, et Antoine Prouvost. Machine learning for combinatorial optimization : a methodological tour d'horizon. *arXiv preprint arXiv :1811.06128*, 2018.

[5] Christopher M Bishop. *Pattern recognition and machine learning*. Springer Science+ Business Media, 2006.

[6] Léon Bottou. Online learning and stochastic approximations. *On-line learning in neural networks*, 17(9) :142, 1998.

[7] Kevin W. Bowyer, Nitesh V. Chawla, Lawrence O. Hall, et W. Philip Kegelmeyer. SMOTE : synthetic minority over-sampling technique. *CoRR*, abs/1106.1813, 2011. URL `http://arxiv.org/abs/1106.1813`.

[8] Amy Mainville Cohn et Cynthia Barnhart. Improving crew scheduling by incorporating key maintenance routing decisions. *Operations Research*, 51(3) :387–396, 2003.

[9] Jean-François Cordeau, Goran Stojković, François Soumis, et Jacques Desrosiers. Benders decomposition for simultaneous aircraft routing and crew scheduling. *Transportation science*, 35(4) :375–388, 2001.

[10] Thomas M Cover et Joy A Thomas. *Elements of information theory*. John Wiley & Sons, 2012.

[11] Yann Dauphin, Harm De Vries, et Yoshua Bengio. Equilibrated adaptive learning rates for non-convex optimization. In *Advances in neural information processing systems*, pages 1504–1512, 2015.

[12] YN Dauphin, H De Vries, J Chung, et Y Bengio. Rmsprop and equilibrated adaptive learning rates for non-convex optimization. arxiv 2015. *arXiv preprint arXiv :1502.04390*, 2015.

[13] G Desaulniers et F Soumis. Airline crew scheduling by column generation. *CIRRELT Spring School on Combinatorial Optimization in Logistics*, 2010.

[14] Guy Desaulniers, Jacques Desrosiers, et Marius M Solomon. *Column generation*, volume 5. Springer Science & Business Media, 2006.

[15] Martin Desrochers et François Soumis. A column generation approach to the urban transit crew scheduling problem. *Transportation science*, 23(1) :1–13, 1989.

[16] Jacques Desrosiers, François Soumis, et Martin Desrochers. Routing with time windows by column generation. *Networks*, 14(4) :545–565, 1984.

[17] Jacques Desrosiers, Yvan Dumas, Marius M Solomon, et François Soumis. Time constrained routing and scheduling. *Handbooks in operations research and management science*, 8 :35–139, 1995.

[18] Michel Deudon, Pierre Cournut, Alexandre Lacoste, Yossiri Adulyasak, et Louis-Martin Rousseau. Learning heuristics for the tsp by policy gradient. In *International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 170–181. Springer, 2018.

[19] Muhammet Deveci et Nihan Cetin Demirel. Evolutionary algorithms for solving the airline crew pairing problem. *Computers & Industrial Engineering*, 115 :389–406, 2018.

[20] Muhammet Deveci et Nihan Cetin Demirel. A survey of the literature on airline crew scheduling. *Engineering Applications of Artificial Intelligence*, 74 :54–69, 2018.

[21] Issmail Elhallaoui, Daniel Villeneuve, François Soumis, et Guy Desaulniers. Dynamic aggregation of set-partitioning constraints in column generation. *Operations Research*, 53(4) :632–645, 2005.

[22] Issmail Elhallaoui, Abdelmoutalib Metrane, François Soumis, et Guy Desaulniers. Multiphase dynamic constraint aggregation for set partitioning type problems. *Mathematical Programming*, 123(2) :345–370, 2010.

[23] Martina Fischetti et Marco Fraccaro. Machine learning meets mathematical optimization to predict the optimal production of offshore wind parks. *Computers & Operations Research*, 106 :289–297, 2019.

[24] M Gamache. Fabrication d'horaires mensuels pour les membres d'equipages en transport aerien (french and english text). 1997.

[25] Michel Gamache et François Soumis. A method for optimally solving the rostering problem. In *Operations Research in the Airline Industry*, pages 124–157. Springer, 1998.

[26] Chunhua Gao, Ellis Johnson, et Barry Smith. Integrated airline fleet and crew robust planning. *Transportation Science*, 43(1) :2–16, 2009.

[27] Ian Goodfellow, Yoshua Bengio, et Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[28] Balaji Gopalakrishnan et Ellis L Johnson. Airline crew scheduling : state-of-the-art. *Annals of Operations Research*, 140(1) :305–337, 2005.

[29] Yufeng Guo, Taieb Mellouli, Leena Suhl, et Markus P Thiel. A partially integrated airline crew scheduling approach with time-dependent crew capacities and multiple home bases. *European Journal of Operational Research*, 171(3) :1169–1181, 2006.

[30] He He, Hal Daume III, et Jason M Eisner. Learning to search in branch and bound algorithms. In *Advances in neural information processing systems*, pages 3293–3301, 2014.

[31] Sergey Ioffe et Christian Szegedy. Batch normalization : Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv :1502.03167*, 2015.

[32] Stefan Irnich et Guy Desaulniers. Shortest path problems with resource constraints. In *Column generation*, pages 33–65. Springer, 2005.

[33] Atoosa Kasirzadeh, Mohammed Saddoune, et François Soumis. Airline crew scheduling : models, algorithms, and data sets. *EURO Journal on Transportation and Logistics*, 6 (2) :111–137, 2017.

[34] Elias Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, et Le Song. Learning combinatorial optimization algorithms over graphs. In *Advances in Neural Information Processing Systems*, pages 6348–6358, 2017.

[35] Elias B Khalil. Machine learning for integer programming. In *IJCAI*, pages 4004–4005, 2016.

[36] Diederik P Kingma et Jimmy Ba. Adam : A method for stochastic optimization. *arXiv preprint arXiv :1412.6980*, 2014.

[37] Niklas Kohl et Stefan E Karisch. Airline crew rostering : Problem types, modeling, and optimization. *Annals of Operations Research*, 127(1-4) :223–257, 2004.

[38] Markus Kruber, Marco E Lübbecke, et Axel Parmentier. Learning when to use a decomposition. In *International Conference on AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 202–210. Springer, 2017.

[39] Yann LeCun, Yoshua Bengio, et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10) :1995, 1995.

[40] Yann LeCun et al. Generalization and network design strategies. In *Connectionism in perspective*, volume 19. Citeseer, 1989.

[41] Guillaume Lemaître, Fernando Nogueira, et Christos K. Aridas. Imbalanced-learn : A python toolbox to tackle the curse of imbalanced datasets in machine learning. *Journal of Machine Learning Research*, 18(17) :1–5, 2017. URL `http://jmlr.org/papers/v18/16-365.html`.

[42] Andrea Lodi et Giulia Zarpellon. On learning and branching : a survey. *Top*, 25(2) : 207–236, 2017.

[43] Andrea Lodi, Luca Mossina, et Emmanuel Rachelson. Learning to handle parameter perturbations in combinatorial optimization : an application to facility location. *arXiv preprint arXiv :1907.05765*, 2019.

[44] David JC MacKay et David JC Mac Kay. *Information theory, inference and learning algorithms*. Cambridge university press, 2003.

[45] Alejandro Marcos Alvarez, Louis Wehenkel, et Quentin Louveaux. Online learning for strong branching approximation in branch-and-bound. 2016.

[46] Anne Mercier et François Soumis. An integrated aircraft routing, crew scheduling and flight retiming model. *Computers & Operations Research*, 34(8) :2251–2265, 2007.

[47] Anne Mercier, Jean-François Cordeau, et François Soumis. A computational study of benders decomposition for the integrated aircraft routing and crew scheduling problem. *Computers & Operations Research*, 32(6) :1451–1476, 2005.

[48] Kevin P Murphy. *Machine learning : a probabilistic perspective*. MIT press, 2012.

[49] Andrew Ng. Improving deep neural networks : Hyperparameter tuning, regularization and optimization. *Deeplearning. ai on Coursera*, 2017.

[50] Nikolaos Papadakos. Integrated airline scheduling. *Computers & Operations Research*, 36(1) :176–195, 2009.

[51] David E Rumelhart, Geoffrey E Hinton, Ronald J Williams, et al. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3) :1, 1988.

[52] Mohammed Saddoune, Guy Desaulniers, Issmail Elhallaoui, et François Soumis. Integrated airline crew pairing and crew assignment by dynamic constraint aggregation. *Transportation Science*, 46(1) :39–55, 2012.

[53] Rivi Sandhu et Diego Klabjan. Integrated airline fleeting and crew-pairing decisions. *Operations Research*, 55(3) :439–456, 2007.

[54] Nadia Souai et Jacques Teghem. Genetic algorithm based approach for the integrated airline crew-pairing and rostering problem. *European Journal of Operational Research*, 199(3) :674–683, 2009.

[55] François Soumis, Yassine Yaakoubi, et Simon Lacoste-Julien. Machine learning feeding mathematical programming for air crew scheduling. In *TRISTAN*, 2019.

[56] Nitish Srivastava. Improving neural networks with dropout. *University of Toronto*, 182 : 566, 2013.

[57] Richard S Sutton, Andrew G Barto, et al. *Introduction to reinforcement learning*, volume 2. MIT press Cambridge, 1998.

[58] Tijmen Tieleman et Geoffrey Hinton. 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, 2012.

[59] Roman Václavík, Antonín Novák, Přemysl Šůscha, et Zdeněk Hanzálek. Accelerating the branch-and-price algorithm using machine learning. *European Journal of Operational Research*, 271(3) :1055–1069, 2018.

[60] Vladimir Vapnik. *The nature of statistical learning theory*. Springer science & business media, 2013.

[61] Oriol Vinyals, Meire Fortunato, et Navdeep Jaitly. Pointer networks. In *Advances in Neural Information Processing Systems*, pages 2692–2700, 2015.

[62] Yassine Yaakoubi, François Soumis, et Simon Lacoste-Julien. Flight-connection prediction for airline crew scheduling to construct initial clusters for OR optimizer. *Les Cahiers du GERAD*, G–2019(26), 2019.