TECHNIQUES D'AMPLIFICATION DES DONNÉES TEXTUELLES POUR L'APPRENTISSAGE PROFOND.

Thèse présentée comme exigence partielle
du doctorat en informatique cognitive

Par Claude COULOMBE

Janvier 2020

À la mémoire de ma soeur Dominique. Pour son amour de la langue française et sa passion pour notre pays, le Québec.

Remerciements

Un gros merci à ma conjointe Isabelle Harpin et à mes enfants Alice et Sophie pour leur affection et leur support tout au long de mon interminable quête scientifique jonchée d'obstacles, d'épreuves familiales, d'absences, d'essais et d'erreurs, de petites et de grandes victoires.

Merci à mes directeurs de recherche, Gilbert Paquette (IA et éducation) et Neila Mezghani (apprentissage automatique) de la TÉLUQ où j'ai bénéficié de conseils, de support, d'une grande liberté et de beaucoup de flexibilité dans le cadre du programme de Doctorat en informatique cognitive.

Un merci particulier à mon co-directeur de recherche et ami Michel Gagnon (TLN) de la Polytechnique Montréal qui aura réussi l'exploit de me faire renouer avec le traitement de la langue naturelle et surtout de m'aider à terminer mon doctorat.

Un clin d'oeil à Antoine Saucier Polytechnique Montréal et à Marie Bourdon, linguiste informaticienne, chez Coginov inc Montréal et mon grand ami François Dubé, médecin interniste au CHUL de Québec, pour le prêt du chalet familial à Notre-Dame-du-Portage.

Merci également à MITACS et Coginov pour mon stage en entreprise et à Calcul Québec pour l'accès à ses infrastructures de calcul.

Je suis reconnaissant à mes anciens professeurs. Je pense à madame Perreault de l'école primaire Roy de Rivière-du-Loup, Jean-Pierre Derat, Paul Darveau, Christiane Bourque, Denis Chouinard et mes amis du Club science de l'École polyvalente St-Pierre, Jean-Marc Rousseau et Wolfgang Blach du Conseil de la jeunesse scientifique (CJS), Fernand Gagné et Raymond Gagnon du CÉGEP, Michel Moisan, Gilles Fontaine, Bernard Goulard, Bernard Lorazo et Paul Lorrain du département de physique de l'UdeM, Yoshua Bengio, Philippe Langlais, Jean Vaucher, Guy Lapalme, Paul Bratley et Gilles Brassard du DIRO de l'UdeM, Renato De Mori de l'Université McGill, et Igor Mel'čuk du département de linguistique de l'UdeM dont le génie n'est pas assez reconnu.

Enfin, merci à mes parents, Viateur et Denise, mes amis et à toutes les personnes qui se sont intéressées à mon travail.

Résumé

Cette thèse a pour objectif d'étudier la faisabilité de différentes techniques d'amplification des données textuelles (ADT)[1] basées sur le traitement de la langue naturelle (TLN)[2] et l'apprentissage automatique[3] afin de pallier à l'insuffisance de données pour l'entraînement de modèles en apprentissage profond[4]. Pour y arriver, nous montrerons une différence de performance entre les mêmes modèles entraînés avec et sans données amplifiées.

Sur le plan pratique, il est fréquent de se retrouver avec des quantités insuffisantes de données pour entraîner de gros modèles statistiques. Ce «*mur des données massives»* représente un défi à la fois pour les communautés linguistiques minoritaires sur la Toile, les organisations, les laboratoires et les entreprises qui rivalisent avec les géants du GAFAM[5].

Alors que le gros de l'effort de recherche en amplification textuelle se concentre sur des solutions d'apprentissage de bout-en-bout[6], le présent travail s'oriente plutôt vers l'emploi de techniques de prétraitement des données pratiques, robustes, capables de monter en charge et simples à mettre en oeuvre. Ces techniques sont inspirées par les techniques d'amplification utilisées avec succès en vision artificielle.

L'idée est de créer de nouvelles données à partir des données existantes. Par abus de langage on parle d'augmentation des données, mais il s'agit plutôt d'une amplification, puisque l'on crée de nouvelles données en préservant le sens qui demeure invariant. On parle aussi de données synthétiques, de données générées ou de données artificielles.

Bien que les techniques étudiées s'appliquent à tout genre de textes, cette thèse se concentre sur l'amplification de phrases. Nous verrons que le concept de «transformation sémantiquement invariante» est au coeur du processus d'amplification des données.

Plusieurs techniques d'ADT ont été expérimentées. Certaines furent testées pour fins de comparaison comme l'injection de bruit ou l'emploi d'expressions régulières. D'autres ont été améliorées comme la substitution lexicale. D'autres techniques plus innovatrices font appel à des services en ligne robustes et capables de monter en charge comme la rétrotraduction[7] et la génération de paraphrases par la transformation d'arbres syntaxiques.
Les techniques d'amplification textuelle étudiées ont permis d'accroître l'exactitude des résultats dans une fourchette 0.5 à 8.8%, sur une tâche normalisée de prédiction de la

[1] En anglais: «data augmentation», plus rarement «data amplification»
[2] En anglais: «natural language processing», «NLP», «computational linguistic», «linguistic engineering». En français: «traitement automatique de la langue naturelle», «TALN», «traitement automatique de la langue», «TAL», «linguistique computationnelle», «ingénierie linguistique»
[3] En anglais: «machine learning». En français: «apprentissage machine», «apprentissage artificiel».
[4] En anglais «deep learning»
[5] Note: GAFAM: Google, Amazon, Facebook, Apple, Microsoft
[6] En anglais: «end-to-end learning». Se dit d'un système d'IA qui apprend tout à partir de données.
[7] En anglais:«back-translation»,«backtranslation»,«round trip translation». En français: «rétroversion»

polarité de textes de critiques de film de la base de données IMDB. Différentes architectures de réseaux profonds de neurones ont été testées: le perceptron multicouche (PMC), le réseau de neurones convolutif 1D (RNC 1D), le réseau récurrent à longue mémoire court terme (LMCT) et le réseau récurrent LMCT bidirectionnel (biLMCT).

Les techniques de transformation de surface (bruit lexical, fautes d'orthographe, expressions régulières) s'avèrent les plus performantes compte tenu de leur rapidité et de leur faible besoin en calcul. La substitution lexicale avec des vecteur-mots AdaGram qui traitent la polysémie est meilleure que la substitution lexicale simple avec WordNet. La génération de paraphrases basée sur la rétrotraduction s'illustre particulièrement sur certains jeux de données et mériterait d'être approfondie.

La principale limite de ce travail est que l'expérimentation ne réalise qu'une seule tâche, soit la prédiction de la polarité de textes. Aussi l'objectif se limitait à montrer la faisabilité des différentes techniques d'ADT. Par exemple, les expériences se sont limitées à un facteur d'amplification de cinq (5), soient cinq nouveaux textes par texte original ce qui ne représente, dans bien des cas, qu'une infime partie des combinaisons possibles.

Malgré leur côté simple et pratique qui plaira aux praticiens, le principal inconvénient de certaines des techniques d'ADT explorées est la quantité de calculs requis. Notons aussi la dépendance envers les services de traduction et d'analyse syntaxique de fournisseurs privés. Nous proposons des alternatives pour mitiger cette dépendance.

En conclusion, nous faisons quelques propositions de futurs travaux, des pistes de valorisation et discutons de l'émergence d'un nouveau défi, le «*mur des modèles préentrainés*», qui combine à la fois la masse des données et la masse des calculs.

Table des matières

Chapitre 1 - Problématique

1.1 Quantité insuffisante de données d'entraînement

Avoir des données d'entraînement en quantité suffisante est probablement le défi en tête de liste dans la pratique des statistiques et de la science des données.

Faire des statistiques implique que l'on dispose de suffisamment de données en quantité et en qualité pour être capable d'en tirer des informations statistiquement significatives.

Pour fonctionner, un algorithme d'apprentissage automatique doit disposer de suffisamment de données pour être capable d'identifier des régularités statistiques. On parle aussi de détection de signaux ou de reconnaissance de formes[8].

Pour une introduction aux grands principes de l'apprentissage automatique, voir l'annexe 1

Postulat fondamental de l'apprentissage automatique (fig. 1.1)
Un algorithme d'apprentissage automatique doit disposer de suffisamment de données pour être capable d'identifier des régularités statistiques dans les données. On parle aussi de détection de signaux ou de reconnaissance de formes.

De même, en science des données, on utilise l'expression consacrée «Foutaises en entrée, foutaises en sortie.»[9] pour insister sur l'importance des données en quantité et en qualité. On ne le répétera jamais assez!

Postulat fondamental de la science des données ;-) (fig. 1.2)
Foutaises en entrée, foutaises en sortie!

Il faut suffisamment de données pour entraîner des modèles statistiques avec des algorithmes classiques d'apprentissage automatique, usuellement des milliers d'exemples. Il faut des quantités encore plus considérables de données, souvent des millions d'exemples, pour entraîner des modèles à base de réseaux de neurones profonds[10] [Goodfellow, Bengio & Courville, 2016], [LeCun, Bengio & Hinton, 2015]. Nous verrons plus loin pourquoi. Pour le moment, contentons-nous de voir un réseau de neurones profond comme un réseau de neurones organisé selon plusieurs couches successives.

[8] En anglais: «statistical regularities detection», «signal detection», «pattern matching», «pattern recognition»
[9] En anglais: «Garbage In, Garbage Out», «GIGO»
[10] En anglais: «deep neural networks»

1.1.1 Motivation - Événement déclencheur

Cette thèse cherche à résoudre un problème rencontré lors de la mise au point d'un outil de correction automatique de réponses à des questions ouvertes dans le cadre de cours en ligne ouverts et massifs (CLOM)[11] offerts à la Téléuniversité du Québec (projet Ulibre) [Coulombe, Paquette & Mezghan, 2016].

Plus précisément la tâche était de noter automatiquement des réponses à des questions ouvertes en français. L'idée était d'entraîner un modèle statistique avec un faible nombre de réponses préalablement corrigées par un professeur ou un tuteur humain (typiquement de l'ordre d'une centaine), puis d'étendre l'étiquetage à quelques centaines de textes en utilisant l'apprentissage semi-supervisé[12] [Zhu, 2005], enfin d'entraîner un modèle de correction automatique. Une fois entraîné, le modèle de correction devait être capable d'évaluer automatiquement un nombre quasi illimité de copies. Pour un résumé de l'état de l'art des logiciels de correction automatique d'essais écrits[13], voir [Shermis & Burstein, 2013].

Entre autres, nous voulions entraîner le modèle prédictif directement à partir des textes bruts des étudiants avec le moins possible d'ingénierie des attributs[14]. Il était donc naturel d'opter pour des réseaux de neurones profonds réputés capables d'apprendre les attributs directement à partir des données [LeCun, Bengio & Hinton, 2015]. Or, il faut savoir que l'apprentissage des attributs par un réseau de neurones exige davantage de données. Pour plus de détails sur l'apprentissage profond, consultez l'annexe 7.

Après quelques essais, il est devenu évident que les modèles profonds obtenus étaient en fort régime de surajustement[15]. Ils avaient des performances médiocres sur les données de test ce qui montrait une faible capacité de généralisation. Avec seulement 400 à 700 réponses par question et une échelle d'évaluation à plusieurs niveaux (problème de classification multiclasse), le projet de correcteur de questions ouvertes à base d'apprentissage profond manquait de données d'entraînement d'abord pour apprendre les attributs et ensuite pour prédire les notes des étudiants.

Rétrospectivement, il se peut que les données récoltées dans le cadre des CLOM Ulibre aient été à la fois en quantité insuffisante, de mauvaise qualité, et bruitées. L'amélioration était à peine perceptible après avoir nettoyé les données avec des outils de correction de textes. Aussi, la tâche était complexe, avec plusieurs classes[16] mal-équilibrées[17]. Une

[11] En anglais: «Massive Open Online Course», «MOOC»
[12] En anglais: «semi-supervised learning»
[13] En anglais: «Automated Essay Scoring», «AES»,«Automatic Essay Evaluation»,«AEE»,«Automatic Essay Grading»,«AEG»,«Automated Student Assessment»,«ASA», «Automatic Short Answer Grading»,«ASAG». En français: «Correction automatique d'essais écrits»,«CA2E»,«Correction automatique de réponses courtes»,«CARC»
[14] En anglais: «features engineering»
[15] En anglais: «overfitting», «overtraining».
[16] En anglais: «multiclass»

possibilité aurait été de ramener l'échelle de correction à deux niveaux (bonne ou mauvaise réponse). Un réseau de neurones, même complexe, peinait à apprendre les relations entre les données d'entraînement et l'étiquette de classe ce qui se traduisait par une erreur d'entraînement élevée (biais élevé) caractéristique d'une situation de sous-apprentissage. Pour en apprendre davantage, consulter l'annexe 2 qui traite du compromis biais-variance. Enfin, il n'était pas du tout certain qu'un être humain aurait obtenu de bien meilleurs résultats en corrigeant les essais à partir de ces seules données.

1.1.2 Le problème des langues minoritaires sur la Toile

Avec l'expansion de la Toile où l'anglais prédomine, avec un peu plus de 50% des contenus [W3CTechs, 2018], on assiste au développement accéléré de technologies de traitement de la langue naturelle à base de statistiques qui utilisent de larges corpus[18] extraits de la Toile. Il en découle que la grande majorité des technologies traite uniquement la langue anglaise.

L'insuffisance de données et de ressources linguistiques est loin d'être un problème exclusif à la langue française. Le problème se retrouve avec plus ou moins d'importance pour toutes les langues moins dotées[19] ou plus rares sur la Toile. Avec plus de 4% des contenus [W3CTechs, 2018], la langue française fait même bonne figure en se classant dans les 10 premières langues sur la Toile. La position de la langue française varie du 4e au 8e rang selon les différents palmarès [Pimienta, 2017].

La plupart des 7000 langues parlées sur notre planète n'ont pas cette présence. Seulement quelques centaines de langues se retrouvent sur la Toile. Presque toutes les langues ne disposent pas d'assez de corpus de textes en format électronique pour entraîner de gros modèles statistiques comme les réseaux profonds de neurones qui sont à la base des nouvelles applications d'intelligence artificielle. Par exemple, la qualité de la traduction automatique neuronale[20] dépend fortement de la disponibilité de larges corpus alignés[21]. Des possibilités de transfert interlinguistique [Johnson et al., 2017] existent, par exemple la traduction en anglais de textes arabes pour l'analyse de sentiment [Salameh et al, 2015]. Mais à part l'anglais, une poignée seulement de langues dispose d'assez de corpus généraux et le problème s'aggrave pour les corpus spécialisés.

---

[17] en anglais: «unbalanced classes»
[18] Note: L'usage veut que le pluriel des mots étrangers suive les règles pour le français. Ici le mot «corpus» d'origine latine dont le pluriel est «corpora» en latin, serait au pluriel «corpus», mais la forme «corpora» pourrait être acceptée.
[19] En anglais «low-resource language», «resource-poor language». En français: «langue moins bien informatisée».
[20] En anglais: «neuronal machine translation», «NMT»
[21] En anglais «parallel corpora». En français: «corpus parallèles».

1.1.3 Inforiches et infopauvres

Évidemment, en dehors du contexte particulier des communautés linguistiques minoritaires, l'insuffisance de données se pose également pour les entreprises qui doivent compétitionner avec les géants de la Toile, les fameux GAFAM: Google, Amazon, Facebook, Apple et Microsoft. En effet, ces géants disposent d'un accès privilégié aux données sur la Toile en plus d'immenses fermes de serveurs pour leurs calculs.

De l'espoir et de nouvelles menaces, selon le cas, apparaissent du côté de l'apprentissage par transfert[22] qui réutilise de gros modèles préentraînés. Pour le moment, ces modèles qui émergent tout juste des laboratoires d'entreprises privées existent presque uniquement pour la langue anglaise. Nous reparlerons des enjeux les concernant à la section 7.4.

Après une discussion du contexte particulier de l'apprentissage profond, différentes solutions seront examinées. Pour une introduction aux réseaux de neurones artificiels[23], consultez l'annexe 5 et l'annexe 7 qui fait un survol de l'apprentissage profond.

1.2 Quantité de données et apprentissage profond

Sans le vouloir, notre projet d'outil de correction de questions ouvertes s'était heurté au «*mur des données massives*»[24] de l'apprentissage profond.

1.2.1 Le mur des données massives

L'entraînement de modèles d'apprentissage profond exige énormément de données pour premièrement résoudre une tâche complexe, deuxièmement pour apprendre les attributs directement à partir des données, et troisièmement pour surmonter le problème de surajustement qui se manifeste quand un modèle possède beaucoup de paramètres.

Il s'agit d'un obstacle majeur à l'emploi de l'apprentissage profond, quand de façon imagée on se heurte au «*mur des données massives*», un peu comme un avion supersonique se heurte au mur du son.

Ce «*mur des données massives*» est un secret bien gardé, tout comme l'énorme quantité de calculs pour bien entraîner les modèles profonds et le travail de mise au point des architectures de réseaux de neurones qui requiert un savoir-faire empirique mal maîtrisé et documenté.

---

[22] En anglais: «transfer learning». En français: «apprentissage par transfert», bien que ce soit plutôt un «transfert d'apprentissage», pour des questions d'uniformité avec les expressions «apprentissage supervisé», «apprentissage non supervisé», «apprentissage par renforcement»
[23] En anglais: «artificial neural network», «ANN»
[24] En anglais: «big data wall»

Le mur des données massives (fig. 1.3)
Pas de données massives => Pas d'apprentissage profond

1.2.2 Un secret bien gardé de l'apprentissage profond

Alors que les algorithmes d'apprentissage automatique classiques plafonnent avec l'accroissement de la quantité de données, la performance des réseaux profonds de neurones continue d'augmenter [Ng, 2917]. Pour obtenir les extraordinaires performances dont sont capables les réseaux profonds il faut une quantité considérable de données.

L'apprentissage supervisé[25] de réseaux profonds, la technique à privilégier pour les applications, n'est devenu praticable que dans les dernières années avec l'accroissement spectaculaire des capacités de traitement des architectures de calcul distribué et les quantités considérables de données disponibles sur la Toile.

Il reste beaucoup d'éducation à faire au niveau des non-spécialistes. Cette relative ignorance du «mur des données massives» est en partie masquée par la possibilité bien réelle d'utiliser des réseaux de neurones profonds avec de faibles quantités de données sur des problèmes jouets. Cela se fait en appliquant beaucoup de régularisation (voir 1.3.4). En effet, les réseaux de neurones sont très flexibles. Aussi, plusieurs ont l'illusion qu'il suffit de se baisser pour récolter des données de qualité sur la Toile. Enfin, il est vrai qu'on peut facilement louer des infrastructures infonuagiques[26] pour traiter des données massives.

Tout cela sans compter les armées d'annotateurs humains qu'il faut rémunérer [Casili, 2019], afin d'étiqueter les données pour faire de l'apprentissage supervisé. En raison des quantités considérables de données requises par l'apprentissage profond et des coûts élevés de l'étiquetage manuel des données, il peut être difficile d'obtenir suffisamment de données pour entraîner efficacement ces modèles.

On parle généralement de données étiquetées pour l'apprentissage supervisé et éventuellement de données non étiquetées mais exploitables par apprentissage semi-supervisé [Zhu, 2005]. Remarquez qu'on peut faire de l'apprentissage profond non supervisé[27] ou de l'apprentissage profond par renforcement[28] qui sont des sujets de recherche très actifs mais encore peu utilisés dans les applications.

Afin de traiter des données massives en un temps raisonnable, les algorithmes d'apprentissage profond exigent une infrastructure de calcul importante. On utilise des

[25] En anglais: «supervised learning»
[26] En anglais: «cloud computing»
[27] En anglais: «unsupervised learning»
[28] En anglais: «reinforcement learning»

processeurs spécialisés dans les calculs parallèles qui sont le plus souvent des processeurs graphiques ou GPU[29].

L'entraînement de modèles profonds requiert beaucoup de savoir-faire. Assurément davantage que certains algorithmes d'apprentissage classiques qu'on pourrait qualifier de prêts à l'emploi. Des algorithmes comme les arbres à dopage de gradient[30], les forêts d'arbres aléatoires[31] et les séparateurs à vaste marge[32] donnent souvent de bons résultats directement avec leurs paramètres par défaut.

Dans le cas des réseaux de neurones, de petites modifications des hyperparamètres (initialisation des poids, taille des lots, nombre d'itérations, pas de gradient, constantes de régularisation, etc.) peuvent résulter en d'importantes différences de performance. À cela s'ajoute l'énorme quantité de calculs nécessaires pour mener des expériences exhaustives et le temps nécessaire à attendre des résultats.

Aussi, il faut maîtriser l'architecture des réseaux de neurones qui joue un rôle très important. Beaucoup des travaux en pointe dans le domaine de l'apprentissage profond relèvent de la conception de nouveaux modèles et architectures de réseaux de neurones.

Un secret bien gardé de l'apprentissage profond (fig. 1.4)
L'apprentissage profond exige d'énormes quantités de données, l'accès à d'importantes infrastructures de calcul avec des processeurs graphiques et une bonne dose de savoir-faire pour préparer les données, bâtir les architectures et entraîner les réseaux de neurones profonds.

1.2.3 Qu'est-ce que la généralisation?

La généralisation est la capacité d'un modèle statistique de faire de bonnes prédictions sur des données / observations qu'il n'a jamais traitées auparavant.

En effet, faire de bonnes prédictions sur le jeu de données d'entraînement[33] est nécessaire, mais cela est insuffisant. Le véritable défi est de faire de bonnes prédictions à partir de données jamais vues. De telles données forment le jeu de données de test[34]. L'erreur de prédiction sur les données de test est justement appelée l'erreur de généralisation[35].

[29] En anglais: «graphical processor unit», «GPU»
[30] En anglais: «gradient boosting». Aussi en français: «arbres à renforcement de gradient»
[31] En anglais: «random forests»
[32] En anglais: «support vector machine», «SVM». En français, on préfère le terme «séparateur à vaste marge» qui conserve l'acronyme «SVM», aussi «machine à vecteurs de support», «classificateur à vaste marge», «classificateur à large marge»
[33] En anglais: «training dataset»
[34] En anglais: «test dataset»
[35] En anglais: «generalization error»

La généralisation est la différence entre mémoriser des résultats pour les retrouver au besoin ou entraîner un modèle statistique sur un ensemble de données pour faire des prédictions.

1.2.4 Le fléau du surajustement

En situation de surajustement[36], on peut dire que le modèle statistique a appris par coeur. Typiquement, il performe parfaitement sur les données d'entraînement mais très mal sur des données fraîches ou données de test. Le modèle peine à généraliser.

Le surajustement s'observe habituellement lorsque le modèle statistique possède beaucoup de paramètres par rapport à la quantité de données d'entraînement. On dit aussi que le modèle est trop complexe pour la quantité de données d'entraînement dont il dispose.

S'il y a peu de données, un modèle statistique de grande capacité comme un réseau de neurones profond va «apprendre par coeur» des régularités statistiques[37] insignifiantes ou fortuites qui seront absentes dans de nouvelles données. Un modèle de grande capacité peut même apprendre du bruit comme expliqué dans [Zhang, et al,, 2016] ou encore des fluctuations statistiques entièrement dues au hasard.

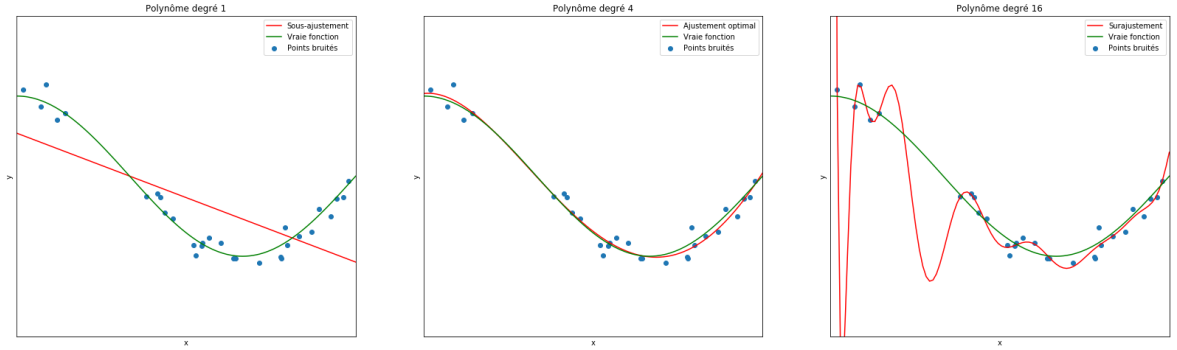
Pour bien comprendre, à cette étape nous devons introduire la notion de sous-ajustement[38] qui est simplement l'inverse du surajustement. Le sous-ajustement se produit lorsque le modèle est trop simple et ne possède pas suffisamment de paramètres pour apprendre le signal ou les régularités statistiques contenues dans les données d'entraînement. Un symptôme évident de sous-ajustement est que les prédictions sur les données d'entraînement sont mauvaises donc l'erreur d'entraînement[39] est importante.

En résumé, lorsque le nombre de paramètres d'un modèle est égal ou supérieur au nombre de données d'entraînement, le modèle peut éventuellement mémoriser les données et faire des prédictions parfaites. Par contre un tel modèle échouera lors de la prévision de données nouvelles (données de test). Il souffre de surajustement.

[36] En anglais: «overfitting», parfois «overtraining»
[37] Note: reconnaître des formes ou identifier un signal
[38] En anglais: «underfitting»
[39] En anglais: «training error»

Sous-ajustement, ajustement optimal, surajustement (fig. 1.5)

Démonstration des problèmes de sous-ajustement et de surajustement. Cet exemple utilise des attributs polynomiaux de degré 1, 4 et 16 et un modèle de régression linéaire pour approximer (en rouge) une fonction non-linéaire à partir de données générées (points bleus) à partir de la vraie fonction (un bout de cosinus en vert) en y ajoutant du bruit[40].



À gauche, on constate qu'un polynôme de degré 1 est incapable d'ajuster les données (sous-ajustement). À droite, un polynôme de degré 16 tente d'ajuster toutes les données incluant le bruit (surajustement). Au centre un polynôme de degré 4 approxime la fonction presque parfaitement (ajustement optimal) [scikit-learn, Underfitting vs. Overfitting]

Attention! Il est simpliste de ramener la complexité d'un modèle à son seul nombre de paramètres, particulièrement lorsqu'on compare des familles de fonctions mathématiques très différentes. Par exemple, il est tout à fait envisageable de concevoir une fonction mathématique, un peu pathologique, ayant une grande capacité de modélisation mais avec un seul paramètre pourvu que la précision (nombre de décimales) de ce paramètre soit arbitrairement grande [Piantadosi, 2018]. Il s'agit d'une astuce où l'on encode toute l'information dans les décimales d'un unique paramètre.

Pour les réseaux profonds de neurones, leur nombre de paramètres réflète généralement assez bien leur capacité de modélisation. Grâce à leur très grand nombre de paramètres, les réseaux profonds de neurones sont capables de mémoriser ou d'apprendre par coeur les moindres détails des données d'entraînement. C'est précisément ce que l'on appelle du surajustement. Notez, que le surajustement n'est pas limité aux réseaux de neurones ou à l'apprentissage supervisé. Chaque fois qu'il y a un processus d'apprentissage, il est possible d'apprendre en mémorisant chaque détail sans aucune généralisation, incluant en apprentissage non supervisé.

De manière étonnante, on a observé que les réseaux de neurones profonds sont capables de généraliser même lorsque leur nombre de paramètres est significativement supérieur au nombre de données d'entraînement [Zhang, et al,, 2016].

Cela dit, il reste que les gros modèles d'apprentissage profond capables de résoudre des tâches très complexes ont des millions (parfois des milliards) de paramètres. Il faut donc

[40] Note: puisque les données textuelles seront numérisées cela s'applique.

beaucoup de données pour bien les entraîner afin d'éviter le surajustement. Sinon, ces réseaux complexes peuvent donner de bons résultats, mais seulement sur le jeu de données d'entraînement. Dans ce cas il n'y a pas de généralisation.

1.2.5 Règles empiriques sur les quantités de données en apprentissage profond

Une première règle empirique en apprentissage profond est d'avoir environ une donnée d'entraînement pour chaque paramètre du réseau neuronal [Sutskever, 2013]. Une deuxième règle empirique est qu'un algorithme d'apprentissage profond supervisé[41] entraîné pour la classification atteindra généralement des performances acceptables avec 5000 exemples étiquetés par classe / catégorie. Le même algorithme égalera ou dépassera la performance humaine lorsqu'il sera entraîné avec un ensemble de données contenant au moins 10 millions d'exemples étiquetés [Goodfellow, Bengio & Courville, 2016]. Plus récemment, Jeff Dean, qui est à tête de Google Brain, affirmait qu'il fallait de l'ordre 100 000 données d'entraînement pour réussir un apprentissage profond [Frank, 2017].

Règles empiriques sur la quantité de données en AP (fig. 1.6)
Number of training cases ~ number of trainable parameters [Sutskever, 2013]
… a supervised deep learning algorithm will generally achieve acceptable performance with around 5,000 labeled examples per category [Goodfellow, Bengio & Courville, 2016]
… and will match or exceed human performance when trained with a dataset containing at least 10 million labeled examples [Goodfellow, Bengio & Courville, 2016]
If you only have 10 examples of something, it's going to be hard to make deep learning work. If you have 100,000 things you care about, records or whatever, that's the kind of scale where you should really start thinking about these kinds of techniques. Jeff Dean, Google Brain [Frank, 2017]

Il est important de rappeler qu'aucun algorithme d'apprentissage n'est meilleur que tous les autres. Cela découle du célèbre théorème «Rien n'est gratuit!»[42] formalisé par [Wolpert & Macready, 1997] que l'on peut résumer de manière non-rigoureuse comme «Il n'y a pas d'algorithme d'apprentissage automatique qui soit meilleur que tous les autres quand on considère tous les problèmes et tous les jeux de données possibles.»

Dans le même ordre d'idées, aucune règle empirique n'est capable de couvrir tous les cas dans un domaine en évolution rapide comme l'apprentissage profond.

[41] En anglais «supervised learning». On apprend sur des données étiquetées.
[42] En anglais:«no free-lunch theorem». Aussi en français la traduction littérale: «Pas de repas gratuit!»

En conclusion, il est préférable d'avoir d'énormes quantités de données, c'est à dire des centaines de milliers, voire des millions de données d'entraînement, avant de s'aventurer sur les sentiers de l'apprentissage profond avec une tâche complexe. Un conseil précieux pour éviter toute déception aux personnes n'ayant pas accès à de gros jeux de données.

Remarquez qu'on a toujours le loisir de bidouiller avec des outils d'apprentissage profond sur un problème-jouet, car la grande flexibilité des réseaux de neurones combinée à beaucoup de régularisation (voir 1.3.4) permet de traiter même de petites quantités de données, mais cela n'est clairement pas de l'apprentissage profond sur une tâche complexe.

1.2.6 Apprendre une tâche complexe implique beaucoup de données

Le besoin d'une grande quantité de données n'est pas particulier à l'apprentissage profond, il est en fait lié à la complexité de la tâche à résoudre. Sans surprise, apprendre une tâche complexe exige un modèle complexe et une grande quantité de données d'entraînement.

Des exemples de tâches complexes sont la reconnaissance d'images[43], la reconnaissance de la parole[44] et la traduction automatique[45]. Tous ces tâches requièrent des données massives[46] et d'importantes capacités de calcul.

Une autre idée forte popularisée par Peter Norvig, directeur en chef des technologies chez Google, est que les données sont plus importantes que les algorithmes pour résoudre des problèmes complexes. Cela se résume assez bien par le titre de l'article «The unreasonable effectiveness of data» [Halevy, Norvig & Pereira, 2009]. Dans le même ordre d'idées, les travaux de Banko et Brill chercheurs à Microsoft ont montré que différents algorithmes d'apprentissage classiques, même très simples, pouvaient donner de bons résultats sur des tâches complexes comme la désambiguïsation de mots[47] en langue naturelle, s'ils disposaient de vastes jeux de données d'entraînement [Banko & Brill, 2001].

Le principal avantage de l'apprentissage profond par rapport à l'apprentissage automatique classique est que l'apprentissage profond facilite la construction de modèles statistiques complexes, avec peu ou pas d'ingénierie des attributs. En effet, l'apprentissage profond est capable d'apprendre directement les attributs à partir de données de manière hiérarchique et distribuée, même avec des données en haute dimension.

---

[43] En anglais: «image recognition»
[44] En anglais: «speech recognition», «automated speech recognition», «ASR», «voice recognition». Aussi en français: «reconnaissance de la parole», «reconnaissance automatique de la parole» «RAP», «reconnaissance vocale» «reconnaissance de la voix»
[45] En anglais: «machine translation»
[46] En anglais: «big data»
[47] En anglais: «Word Sense Disambiguation», «WSD»

1.3 Solutions au surajustement sans ajouter de données

Nous allons examiner différentes solutions au problème de surajustement des modèles statistiques, en commençant par les solutions qui ne demandent pas d'ajouter de nouvelles données d'entraînement. Remarquez que l'ajout de données demeure le moyen le plus simple et le plus fiable de combattre le surajustement. Évidemment cette solution est au coeur même de cette étude. Nous y reviendrons plus loin.

1.3.1 Simplifier le modèle

La première solution consiste à choisir un modèle plus simple qui comporte moins de paramètres. Par exemple, on décidera de prendre un modèle linéaire plutôt qu'un modèle polynomial de degré plus élevé.

Pour les réseaux de neurones, la simplification passe par la réduction du nombre de couches cachées[48] et / ou du nombre de neurones par couche. Dans la pratique, la simplification du modèle est surtout utilisée pour accélérer le temps de traitement [Ng, 2017], beaucoup moins fréquemment pour contrer le surajustement.

1.3.2 Réduire le nombre d'attributs et le bruit

Réduire le nombre ou le type d'attributs dans les données peut aider à résoudre les problèmes de surajustement (réduire la variance) sans exclure trop d'attributs utiles [Géron, 2017a].

Lorsque les données sont abondantes, la pratique en apprentissage profond est de donner tous les attributs dont nous disposons au réseau de neurones et le laisser découvrir les attributs utiles directement dans les données [Goodfellow, Bengio & Courville, 2016], [Ng, 2017]. Quand le jeu de données d'entraînement est petit, la réduction / sélection des attributs peut être utile. Cela dit, il est toujours utile de réduire le bruit, retirer les attributs inutiles, corriger les erreurs et supprimer les données aberrantes.

1.3.3 Arrêt précoce de l'entraînement

Les algorithmes d'apprentissage automatique procèdent par itérations. Les premières itérations améliorent le modèle mais souvent, passé un certain nombre d'itérations, le modèle commence à surajuster sur les données d'entraînement. Il perd alors en généralisation.

[48] En anglais: «hidden layers»

L'arrêt précoce[49] consiste à stopper l'algorithme avant que le modèle ne souffre trop de surajustement [Goodfellow, Bengio & Courville, 2016].

L'idée est d'arrêter l'entraînement lorsque l'erreur de validation atteint un minimum. Pratiquement, on attendra une à quelques itérations supplémentaires pour s'assurer de la tendance [Chollet, 2017], [Coulombe, 2018b]. Généralement on considère l'arrêt précoce comme une technique de régularisation [Géron, 2017a], [Ng, 2017].

1.3.4 Introduire de la régularisation

Les praticiens de l'apprentissage profond utilisent surtout la régularisation pour réduire le surajustement.

Les techniques de régularisation[50] imposent des contraintes au modèle. Traditionnellement, on ajoute un terme dit de régularisation à la fonction de coût[51] dont l'algorithme d'apprentissage cherche à minimiser les erreurs. L'emploi de la régularisation aide à diminuer le surajustement et à produire des modèles statistiques qui généralisent mieux.

La régularisation s'avère particulièrement utile avec les modèles complexes comme les réseaux de neurones profonds. Pour en apprendre davantage sur les techniques de régularisation et de normalisation, veuillez consulter l'annexe 3.

1.3.5 Ajouter des connaissances a priori et des attributs plus riches

Une approche générale en apprentissage automatique est d'enrichir les modèles avec des connaissances a priori. Le plus souvent, on ajoute des attributs[52] à plus grande valeur informative que les données brutes [Goodfellow, Bengio & Courville, 2016]. On peut aussi ajouter des attributs pour corriger des erreurs que l'on a observées [Ng, 2017].

Lorsqu'un algorithme d'apprentissage profond ne dispose pas de suffisamment de données pour découvrir ou apprendre par lui-même les attributs des données, on peut très bien les lui fournir directement [Goodfellow, Bengio & Courville, 2016]. Il existe un compromis à établir entre les connaissances préalables[53] et la quantité de données d'entraînement.

Justement, il existe des approches intéressantes pour apprendre à partir de très peu de données que l'on appelle apprentissage à partir de peu d'exemples, apprentissage à partir

[49] En anglais: «early stopping»
[50] En anglais: «regularization»
[51] En anglais: «error function», «cost function», «loss function», «objective function». En français, «fonction de coût», «fonction de perte», «fonction d'erreur», «fonction objective». En apprentissage automatique, on utilisera les termes «erreur», «coût», et «perte» d'une manière interchangeable.
[52] En anglais: «features»
[53] En anglais: «prior knowledge»

d'un seul exemple et même apprentissage à partir de zéro exemple [Ravi & Larochelle, 2016]. Ces approches qualifiées de méta-apprentissage reposent sur des connaissances a priori (ou méta-connaissances) apprises sur plusieurs autres tâches apparentées [Finn et al, 2018]. Techniquement, on peut ranger ces approches dans l'apprentissage par transfert discuté à plusieurs endroits dans cette thèse dont la section 1.3.7.

Depuis 2013, la représentation des mots par vecteurs-mots[54] ou vecteurs contextuels s'impose en TLN pour remplacer l'encodage à un bit discriminant[55] et les sacs de mots[56]. L'emploi de vecteurs-mots est une technique d'enrichissement par des connaissances a priori. Les vecteurs-mots viennent enrichir les données d'entraînement des algorithmes d'apprentissage aussi bien classiques que profonds. Même si ce ne sont pas des modèles proprement dits, les vecteurs-mots sont considérés par plusieurs comme une forme d'apprentissage par transfert. Pour plus de détails, lire A8.8.1 qui traite des représentations par vecteurs-mots.

1.3.6 Utiliser un modèle génératif

Avant de poursuivre, il serait bon de rappeler la différence entre un modèle génératif[57] et un modèle prédictif ou discriminant[58] en apprentissage automatique. Pour l'amplification de données, l'idée serait d'utiliser un modèle génératif pour produire de nouvelles données.

Grossièrement, un modèle génératif apprend la distribution de probabilités des données alors qu'un modèle prédictif apprend les frontières de décision[59] entre les classes (ou catégories) des données. Avec un modèle génératif on peut générer un exemplaire de donnée qui est similaire à ceux d'une distribution statistique apprise par le modèle alors qu'avec un modèle prédictif, on peut prédire l'étiquette (ou la classe) à partir des attributs de la donnée. Il faut savoir qu'en général les modèles prédictifs performent mieux que les modèles génératifs sur les tâches de classification [Ng & Jordan 2002].

En langage statistique, un modèle génératif apprend la distribution de probabilité conjointe $P(x,y)$, c'est à dire une fonction qui donne la probabilité d'observer x et y en même temps. Un bon exemple de modèle génératif est un modèle de Bayes naïf. Un modèle prédictif apprend plutôt la distribution de probabilité conditionnelle $P(y|x)$, c'est à dire une fonction qui donne la probabilité d'observer y si l'on a observé x. On peut toujours transformer un modèle génératif en un modèle prédictif en appliquant la règle de Bayes [Ng, 2016]. En effet, ces deux probabilités sont reliées par la règle de Bayes, $P(x,y) = P(x)\,P(y|x)$.

---

[54] En anglais: «word-vectors», «word embeddings», «embeddings». En français: «vecteurs contextuels», «vecteur de contexte», «vecteurs d'enrichissement», «plongements lexicaux», «plongements»
[55] En anglais: «one-hot-encoder»
[56] En anglais: «bag-of-words», «BOW»
[57] En anglais: «generative model»
[58] En anglais: «predictive model», «discriminative model»
[59] En anglais: «decision boundaries»

Tout à fait en dehors de l'apprentissage profond, mais dans l'idée d'utiliser des modèles génératifs pour l'amplification de données, des chercheurs du MIT [Patki, Wedge & Veeramachaneni, 2016] travaillent sur une voûte de données synthétiques (Synthetic Data Vault) afin de créer des modèles génératifs à partir de bases de données relationnelles.

En apprentissage profond, des réseaux antagonistes génératifs[60] ont été employés en vision artificielle pour générer automatiquement des images afin d'amplifier des jeux de données. Bien que ces images ne soient toujours très réalistes (comme les fameux oiseaux sans pattes ou avec une seule patte) et habituellement de faible résolution, avec quelques astuces on arrive à créer des données utilisables. Par exemple, un groupe de chercheurs de l'Université de Toronto a réussi à accroître l'exactitude d'un algorithme de classification d'images rayons-X de 20% avec des images générées par un réseau antagoniste génératif profond convolutif[61] [Salehi Nejad et al, 2018].

Différentes tentatives ont été faites pour développer des réseaux antagonistes génératifs (RAG) ou des réseaux neuronal récurrents (RNR) génératifs pour les données textuelles. Cette approche est discutée à la section 6.5.2.

1.3.7 L'apprentissage par transfert

Dans cet ordre d'idées, mais à une autre échelle, l'apprentissage par transfert[62] se répand rapidement au point d'annoncer un futur état de l'art[63] pour le TLN. L'apprentissage par transfert consiste à utiliser de gros modèles préentraînés[64].

L'idée sous-jacente à l'apprentissage par transfert est assez simple. On prend un gros modèle préentraîné pour une tâche générique sur un très gros jeu de données et on l'applique à une nouvelle tâche en l'adaptant. On parle généralement de gros modèles prédictifs entraînés selon une approche d'apprentissage supervisé sur de gros corpus étiquetés, typiquement sur des millions d'exemples, parfois même des milliards. Comme d'autres l'ont démontré, la puissance prédictive de ces modèles augmente avec leur taille [Halevy, Norvig & Pereira, 2009], [Banko & Brill, 2001].

Plus précisément, les modèles entraînés pour un problème précis sur des données d'un type spécifique (images, voix, textes, etc.) apprennent toutes sortes de régularités statistiques propres à ce type de données qui peuvent être utilisées pour résoudre d'autres problèmes faisant appel au même type de données. Une certaine adaptation reste habituellement nécessaire par l'ajout de données spécifiques à la nouvelle tâche.

---

[60] En anglais: «generative adversarial network», «GAN». En français:, «réseau adversaire génératif»
[61] En anglais: «deep convolutional generative adversarial Network», «DCGAN»
[62] En anglais: «learning transfer»
[63] En anglais: «state of the art», «SOTA»
[64] Note: Initialement, je fus séduit par l'idée d'orienter ma thèse vers l'apprentissage par transfert en TLN, mais devant l'absence de gros modèles préentraînés, la perspective de les créer à partir de zéro sembla hors de ma portée. Entre autres, réunir les corpus et surtout la quantité de calcul nécessaire.

Typiquement, les premières couches d'un réseau profond de neurones, celles qui sont plus près de l'entrée des données, apprennent des relations plus primitives (comme les textures et les contours dans les images) puis les couches suivantes apprennent des représentations de plus en plus abstraites (assemblages, formes complexes, visages) au fur et à mesure que l'on s'approche de la sortie du réseau. En fait, seule la dernière couche avant la sortie (parfois quelques unes de plus) apprend des représentations spécifiques au problème à résoudre. Ces couches aux savoirs très spécifiques ne sont pas utiles pour le transfert. Par contre les couches profondes, les plus proches de l'entrée, peuvent contenir un savoir transférable.

Des chercheurs ont étudié à quel point des représentations et attributs appris par un réseau de neurones étaient transférables [Yosinski et al, 2014]. Pour un tour d'horizon de l'apprentissage par transfert, voir [Weiss, Khoshgoftaar & Wang, 2016].

1.3.7.1 L'apprentissage par transfert en vision par ordinateur

Les chercheurs en vision par ordinateur[65] ont rapidement réalisé que les paramètres appris dans les modèles entraînés sur les données de ImageNet (14 millions d'images annotées manuellement en 20 000 catégories) pouvaient être utilisés pour initialiser des modèles pour d'autres ensembles de données, réduire considérablement le temps d'entraînement et améliorer les performances [Deng et al, 2009].

Dans le domaine de la vision par ordinateur, plusieurs modèles préentrainés généralement sur le corpus ImageNet et disponibles gratuitement existent comme le VGG du Visual Graphics Group à Oxford, AlexNet et MobileNet de Google. Plusieurs de ces modèles préentraînés se trouvent en accès libre [ModelZoo, 2018]. Il existe également quelques modèles préentraînés pour la reconnaissance de la parole comme le modèle DeepSpeech de la Fondation Mozilla [Mozilla Foundation, 2016].

Sans les données d'ImageNet, il n'y aurait peut-être pas la révolution de l'apprentissage profond d'aujourd'hui. Malheureusement, depuis l'abandon du défi ImageNet en 2017, les modèles de vision par ordinateur préentraînés et partagés librement stagnent ou deviennent désuets.

1.3.7.2 L'apprentissage par transfert en traitement de la langue naturelle

Depuis 2013, malgré que ce ne soient pas des modèles proprement dits, les représentations par vecteurs-mots ou vecteurs contextuels sont considérées par plusieurs comme une première forme d'apprentissage par transfert en langue naturelle. Les vecteurs-mots sont utilisés pour enrichir les données textuelles alimentant les réseaux neuronaux profonds ou

[65] En anglais: «computer vision». Aussi en français: «vision artificielle».

d'autres algorithmes d'apprentissage. Certaines ressources en vecteurs-mots sont proposées gratuitement en téléchargement par des joueurs du GAFAM.

Bon nombre de ces ressources gratuites sont de piètre qualité, particulièrement pour les langues autres que l'anglais. Par exemple, la plupart des ressources en vecteurs-mots font abstraction de la morphologie riche du français et de l'existence d'unités nominales complexes comme «carte à puce» et «effet de serre». De plus, elles sont rarement mises à jour. Pour plus de détails sur les représentations par vecteurs-mots, lire la section A8.8.1.

Récemment (2018), de gros modèles de langue[66] préentraînés ont été utilisés avec des performances conformes à l'état de l'art sur un large éventail de tâches en TLN. Un modèle de langue est un modèle prédictif, dans le sens de la prédiction du prochain mot dans un texte. Apparentés aux vecteurs-mots, les modèles de langue sont de vrais modèles avec des paramètres et une architecture que l'on peut adapter à d'autres tâches par entraînement.

Pour le moment, ces modèles, qui émergent tout juste des laboratoires, existent souvent uniquement pour la langue anglaise. Il existe bien quelques modèles multilingues, comme une version de BERT multilingue [GOOGLE, 2018e], mais ils sont assez pauvres[67]. Malgré certaines possibilités du côté du transfert interlinguistique [Johnson et al., 2017], une poignée seulement de langues disposent d'assez de corpus généraux pour entraîner de tels modèles, mais le problème persiste pour les corpus spécialisés. Aussi, personne n'est assuré de la qualité de ces ressources, leur maintenance, et encore moins de leur pérennité.

Pour comprendre comment on réalise techniquement l'apprentissage par transfert, consultez l'annexe 9 qui présente un guide pratique de l'apprentissage par transfert [Géron, 2017a].

Nous verrons dans la discussion, section 6.7 quel rôle incombe à l'amplification de données textuelles dans un contexte où l'apprentissage par transfert est appelé à se généraliser.

1.3.7.3 La disponibilité de gros modèles préentraînés

L'utilisation pratique de l'apprentissage par transfert suppose que l'on dispose de gros modèles génériques préentraînés sur d'énormes jeux de données. L'idée de base est que le modèle servant au transfert doit être suffisamment gros, suffisamment générique et avoir été entraîné sur suffisamment de données pour avoir appris des régularités statistiques utiles dans les données (textures et formes dans des images, phonèmes dans la parole, relations lexicales et grammaticales dans des textes) et transférables afin de résoudre d'autres problèmes. Nous y reviendrons à la section 7.4.1 Le mur des modèles préentrainés

[66] En anglais: «language model»
[67] Note: Deux modèles de langue en français inspirés de BERT sont apparus à la fin de 2019, CamemBERT le 10 novembre 2019 et FlauBERT le 12 décembre 2019.

1.3.8 Combinaison d'algorithmes

Une technique qui relève davantage du méta-apprentissage consiste à combiner les résultats de plusieurs modèles différents, selon ce qu'on appelle les méthodes ensemblistes d'apprentissage[68].

Les deux méthodes ensemblistes les plus utilisées sont la simple mise en commun[69] et le dopage[70]. L'algorithme ensembliste de mise en commun se contente de combiner les résultats de modèles au moyen de stratégies de vote, par exemple par un vote majoritaire. De son côté, l'algorithme de dopage ensembliste donne un coup de pouce aux modèles les plus performants sur la base des données précédemment mal évaluées parmi un ensemble de modèles plus simples.

Dans le cas général qui nous intéresse, l'approche de mise en commun simple, par votation, serait la plus simple à mettre en oeuvre pour réduire le surajustement.

1.4 Solutions au surajustement impliquant des données supplémentaires

Pour contrer le surajustement et l'insuffisance de données afin de bien entraîner un modèle statistique, la solution évidente consiste à récolter davantage de données ou à amplifier les données dont on dispose.

1.4.1 Simulateurs et capteurs

Pour renchérir sur l'idée d'acquisition de données, on peut créer une simulation réaliste ou munir un dispositif physique de capteurs. Voilà une approche fort répandue en apprentissage par renforcement[71] pour les jeux, les voitures autonomes et en robotique.

1.4.2 Annotation participative

Si l'on dispose de masses de données non étiquetées et qu'on a des budgets conséquents, on peut envisager une campagne d'annotation participative[72] sur une plateforme de production participative[73] comme Amazon Mechanical Turk, Clickworker ou Crowdflower.

---

[68] En anglais: «ensemble learning». Aussi en français, «apprentissage ensembliste», «apprentissage par combinaison de modèles».
[69] En anglais: «bagging»
[70] En anglais: «boosting». Il s'agit d'un principe général qui consiste à doper, renforcer certains algorithmes faisant partie d'un ensemble. Les arbres à renforcement de gradient (en anglais: «gradient boosting trees») sont des «arbres à dopage de gradient» ou «arbres à gradient dopé»
[71] En anglais: «reinforcement learning».
[72] En anglais: «crowdsourcing labeling»
[73] En anglais: «crowdsourcing». Aussi en français: «externalisation ouverte», «externalisation par des foules»

D’autres créent des applications ludiques, des plateformes sociales en ligne ou des applications pour téléphones intelligents afin que le grand public annote des données gratuitement.

Notez que le travail d’annotation, réalisé par de la main-d’oeuvre bon marché, est de plus en plus dénoncé comme une forme grandissante d’exploitation des travailleurs et de précarisation de l’emploi [Casili, 2019].

1.4.3 Apprentissage semi-supervisé

Le cas échéant, on peut utiliser l’apprentissage semi-supervisé sur des données partiellement étiquetées.

Ces algorithmes exploitent la similarité entre les données pour leur attribuer des étiquettes. Par exemple, un algorithme non-supervisé de groupage[74] identifie des groupes, puis attribue une étiquette à chacun des groupes, puis étiquette tous les autres membres de chacun de ces groupes [Zhu, 2005]. Dans la pratique, il n’y a pas toujours des groupes bien séparés et un nettoyage manuel des données s’imposera.

1.4.4 Augmentation, amplification des données

L’augmentation ou plutôt l’amplification est une technique utilisée dans de nombreuses tâches d'apprentissage automatique pour agrandir la taille du jeu de données d'entraînement. L’amplification de données consiste à générer de nouvelles données à partir des données existantes. Nous consacrerons le prochain chapitre à cette idée.

[74] En anglais: «clustering». Aussi en français: «agrégation»

Chapitre 2 - L’amplification des données

Ce chapitre passera en revue les travaux dans la littérature scientifique sur l’amplification des données. Avant d’aller plus loin, il serait bon de rafraîchir vos connaissances avec les deux principales sources d’erreurs en apprentissage automatique qui sont le biais et la variance. L’annexe 2 traite du compromis biais-variance. Il serait également bon de s’informer sur l’usage des courbes d’entraînement[75] à l’annexe 4.

L’objectif est d'améliorer les performances d'un algorithme d'apprentissage automatique en augmentant la quantité de données d'entraînement. Le deuxième résultat recherché est de combattre le surajustement.

L’idée consiste à générer de nouvelles données à partir des données existantes et à les ajouter à l'ensemble des données d’entraînement. Pour certaines tâches d’apprentissage automatique, il est relativement simple de créer de nouvelles données [Goodfellow, Bengio & Courville, 2016].

Sur le plan terminologique on utilise les termes: amplification de données, données amplifiées, augmentation de données, données augmentées, synthèse de données, données synthétiques, données artificielles, génération de données, données générées, données simulées, et même parfois fausses données[76].

2.1 Amplification des données plutôt qu’augmentation des données

Par abus de langage ou pour faire simple on parle d’augmentation des données[77], mais il serait plus juste de désigner le processus comme une amplification de données[78]. Rigoureusement, il s’agit d’une transformation invariante où l’invariant est le sens.

On peut assimiler une transformation invariante à une amplification. Cette idée d’un accroissement par une transformation qui respecte un invariant se retrouve dans la définition usuelle du mot amplification où l’on trouve la notion d’accroissement et la notion d’invariant. Ces notions sont présentes dans le cas de l’amplification de signaux électroniques et dans l’amplification de matériel génétique. Pour les données, la sémantique de la donnée est la partie invariante qui est conservée et donc amplifiée par la transformation. Par contre, la notion d’invariance n’est pas du tout sous-jacente au mot augmentation.

Par analogie, l’amplification de données renforce le signal ou si l’on préfère les régularités statistiques que l’on cherche à apprendre avec un algorithme d’apprentissage statistique.

---

[75] En anglais: «learning curves»
[76] En anglais: «fake data»
[77] En anglais: «data augmentation»
[78] En anglais: «data amplification»

L'amplification présente un très gros avantage sur l'augmentation ou la génération «ex nihilo». L'idée est de partir de données qui font déjà du sens et qui sont pertinentes au problème que l'on cherche à résoudre.

Il existe un autre usage du terme anglais «data augmentation» en lien avec les méthodes de Monte-Carlo par chaînes de Markov[79] qui ne correspond pas à la définition de cette étude. En gros, ce sont des méthodes d'échantillonnage[80] à partir de distributions de probabilités utilisées en simulation [Van Dyk & Meng, 2001]. L'usage du terme amplification de données permettrait d'éviter cette confusion.

Dans cette étude, nous nous intéressons plutôt à des techniques d'amplification des données textuelles (ADT) pour générer des données sémantiquement similaires à des données textuelles existantes par des transformations de celles-ci.

2.2 Historique

Les premiers travaux relatifs à une forme d'amplification des données remontent aux années 80 autour de l'enrichissement des requêtes[81] dans le cadre de la recherche d'information[82], de l'indexation des textes et des moteurs de recherche [Rosario, 2017]. Il s'agissait le plus souvent d'ajouter des mots à une requête pour en élargir ou en préciser la portée ou encore pour la normaliser [Carpineto & Romano, 2012].

La normalisation d'une requête consiste le plus souvent en la réduction du nombre de variantes morphologiques ou syntaxiques par la troncature[83] ou la lemmatisation[84]. Par exemple, ramener les mots «transformer», «transforme», «transforment», «transformais», «transformateur» et «transformation» à la racine «transform» par troncature ou la liste «transformer», «transforme», «transforment», «transformais» au lemme «transformer» par lemmatisation. Dans le jargon de la recherche d'information l'enrichissement des requêtes affectera le rappel[85] et la précision[86].

En 1997, [Ha & Bunke, 1997] créent des jeux d'images supplémentaires en appliquant des transformations sur des images dans le but d'aider à la reconnaissance de caractères en écriture manuscrite.

En marge des progrès de l'apprentissage profond et de son insatiable besoin en données, plusieurs techniques d'amplification des données ont été développées dans les dernières

[79] En anglais: «Markov chain Monte Carlo», «MCMC»
[80] En anglais: «sampling»
[81] En anglais: «query expansion»
[82] En anglais: «information retrieval», «IR»
[83] En anglais: «stemming»
[84] En anglais: «lemmatization»
[85] En anglais: «recall»
[86] En anglais: «precision»

années. Les premiers succès de l'apprentissage profond ont été pour l'essentiel dans les applications de la perception: vision par ordinateur et reconnaissance de la parole. C'est pourquoi, les premières avancées en amplification des données ont été dans ces domaines.

Dans l'article séminal de [Krizhevsky, Sutskever & Hinton, 2012], l'amplification de données a été utilisée pour combattre le surajustement d'un réseau de neurones profond qui comportait plus de 60 millions de paramètres. Utilisées pour remporter la compétition ImageNet de classification supervisée d'images, il s'agissait de transformations qui généraient de nouvelles images tout en préservant l'étiquette ou la classe de ces images.

En vision par ordinateur, il est d'usage de créer de nouvelles images par des transformations géométriques qui préservent la similitude. On parle d'isométries (conservation des distances) comme les translations, les rotations, les réflexions ou retournements[87] selon un axe orthogonal. Les homothéties (changement d'échelle) comme les agrandissements / réductions[88] et même des déformations élastiques [Simard, Steinkraus & Platt, 2003] ont aussi été utilisées. On peut également altérer l'intensité des canaux RGB (codification des couleurs) en utilisant l'analyse en composantes principales pour générer de nouvelles images [Krizhevsky, Sutskever & Hinton, 2012].

L'amplification de données est une technique tellement utile en vision par ordinateur qu'il existe maintenant des modules d'amplification de données dans les bibliothèques logicielles d'apprentissage profond les plus populaires comme Tensorflow / Keras et PyTorch.

En reconnaissance de la parole, l'amplification de données est réalisée par la manipulation du signal pour le ralentir ou l'accélérer [Ko et al, 2015], l'injection de bruit et la modification du spectrogramme [Jaitly & Hinton, 2013].

L'injection de bruit à l'entrée d'un réseau de neurones peut également être considérée comme une forme d'amplification des données [Goodfellow, Bengio & Courville, 2016]. En reconnaissance vocale, les modèles obtenus ont non seulement fracassé des records [Dahl et al, 2012] mais ils ont été rapidement incorporés dans des produits comme Siri d'Apple, Alexa d'Amazon et le Google Assistant.

Enfin, lors de la comparaison d'algorithmes d'apprentissage automatique, il est recommandé de prendre en compte l'effet de l'amplification du jeu de données d'entraînement [Goodfellow, Bengio & Courville, 2016].

[87] En anglais: «flipping»
[88] En anglais; «rescaling»

Amplification des données - techniques selon le type de données (fig. 2.1)			
	_____ Type de données _____		
	images	voix	texte
Techniques d'amplification	translation, rotation, réflexion, redimension, rognage, extraction de parties, altération des couleurs	manipulation du signal, manipulation du spectrogramme, injection de bruit	substitution lexicale par des synonymes, injection de bruit, transformation de surface (expressions régulières)
			Nouvelles techniques: transformation d'arbres syntaxiques, rétrotraduction

2.3 L'amplification textuelle est peu répandue

Après quelques recherches sur la Toile, il est devenu évident que contrairement à la vision par ordinateur et la reconnaissance de la parole, le traitement de la langue naturelle écrite ne semblait pas disposer de beaucoup de techniques pour l'amplification de données.

Au début de 2015, la seule technique d'amplification textuelle connue était la substitution lexicale qui consiste à remplacer un mot par son synonyme en utilisant un thésaurus ou un dictionnaire de synonymes [Zhang & LeCun, 2015].

Comme le mentionne [Kobayashi, 2018], l'amplification de données n'est pas encore courante pour les tâches de traitement du langage naturel .

Citation - L'amplification textuelle est peu répandue (fig. 2.2)
However, usage of data augmentation for NLP has been limited. [Kobayashi, 2018]

2.3.1 Les données textuelles sont difficiles à traiter

Il est légitime de se demander pourquoi? Premièrement, parce que les données textuelles sont difficiles à traiter. Selon [Goldberg, 2017], les données textuelles sont difficiles à traiter car elles sont symboliques, discrètes, compositionnelles et éparses[89]. À cela, on peut ajouter que les données textuelles sont hiérarchiques, bruitées, pleines d'exceptions et ambigües.

[89] En anglais: «sparse». En français, l'adjectif «épars» est préféré à cause de sa proximité avec le mot anglais «sparse». En français, les mots «dilué» ou« dispersé» sont aussi de bons candidats, aussi «clairsemé» et «disséminé».

2.3.1.1 Données symboliques

L'élément de base de la langue écrite est le caractère, un symbole discret, qui ne renseigne pas beaucoup sinon sur la langue ou la famille de langues concernée via l'encodage.

Les caractères forment les mots, mais les mots sont souvent des symboles discrets sans relation entre eux. Pour illustrer son propos, [Goldberg, 2017] utilise les mots «hamburger» et «pizza». Très bien, mais il néglige la riche morphologie lexicale de beaucoup de langues, y compris la langue anglaise. Nous y reviendrons quand nous décrirons les différents niveaux d'organisation de la langue naturelle.

2.3.1.2 Données discrètes

[Goldberg, 2017] rappelle que les données textuelles sont discrètes par contraste avec les données continues que l'on retrouve dans les images et les signaux acoustiques. Il précise qu'il n'existe pas d'opération simple et continue qui permette par exemple de passer du mot «red» au mot «pink» sans faire intervenir un dictionnaire ou une table de conversion[90].

Les techniques d'apprentissage à base de descente de gradient ne s'appliquent pas directement à des données discrètes comme l'explique Ian Goodfellow dans son commentaire sur la difficulté de réaliser des réseaux antagonistes génératifs pour du texte [Goodfellow, 2016].

2.3.1.3 Données composées et hiérarchiques

La structure hiérarchique et la compositionalité[91] de la langue sont évidentes. Cette structure implique que le sens d'une expression est une fonction de l'interprétation de ses parties en interaction avec le contexte.

On part du caractère, puis le mot, le groupe de mots ou syntagme, la phrase simple ou la proposition, la phrase complexe, le paragraphe, le texte complet, et d'autres niveaux comme l'organisation en introduction, développement et conclusion, le découpage en chapitres, le livre, beaucoup d'autres regroupements textuels, pour finir avec les corpus.

Le premier niveau d'organisation de la langue écrite est le caractère. Les caractères forment les mots, le second niveau d'organisation. Les mots sont souvent des symboles discrets sans relation entre eux [Goldberg, 2017]. Il reste que même au niveau du mot, la morphologie lexicale peut donner certaines indications utiles. On s'intéresse ici aux

[90]En anglais: «look-up table»
[91]En anglais: «compositionality». La compositionalité c'est le fait de pouvoir être composé, la composition c'est le résultat ou l'action selon Le Grand Robert.

morphèmes dont le radical ou racine[92] et les affixes. Parmi les affixes, les plus utiles sont le préfixe[93] qui se trouve en début de mot et le suffixe[94] qui se trouve en fin de mot. Ainsi, les mots «révolution» et «résolution» ne diffèrent que par une lettre mais ont des significations totalement différentes. Malgré ce fait, le suffixe «tion» nous indique qu'il s'agit de noms communs probablement dérivés du latin qui désignent une action ou le résultat d'une action. Mais il y a des exceptions, c'est-à-dire des mots qui se terminent en «tion» et qui ne sont ni dérivés du latin, ni reliés à un verbe, comme le mot anglais «gumption». Enfin, il y a toujours la possibilité d'une erreur de frappe ou d'orthographe.

En apprentissage automatique, le traitement de la morphologie lexicale (les différentes formes fléchies) est variable en fonction de l'application et de la quantité de données. Souvent, le mot est simplement tronqué, on parle alors de troncature[95] et l'algorithme s'appelle un troncateur[96]. On peut aussi produire un lemme (ou la forme canonique d'un mot), soit l'équivalent de l'entrée d'un mot dans un dictionnaire par la lemmatisation[97]. En français, la lemmatisation d'un verbe retourne ce verbe à l'infinitif et pour les autres mots, la lemmatisation retourne ce mot au masculin singulier. L'algorithme de lemmatisation s'appelle aussi un lemmatiseur[98] [Coulombe, 2018a]. Dans la pratique, l'analyse morphologique d'un mot produit rarement un résultat unique, mais plutôt un ensemble de résultats possibles.

Traitement de la morphologie lexicale en apprentissage profond (fig. 2.3)
Pour le traitement de la morphologie lexicale, la tendance observée en apprentissage profond consiste à conserver toutes les formes fléchies et à entraîner le modèle sur les caractères plutôt que les mots. L'idée est qu'avec suffisamment de données et en allongeant le temps de calcul, le modèle devrait être capable d'apprendre par lui-même à traiter la morphologie directement à partir des données.

Le niveau d'organisation suivant, le troisième, est le groupe de mots ou syntagme[99]. À ce niveau, l'analyse lexicale permet de réduire l'ensemble des étiquettes lexicales résultant de l'analyse morphologique à une seule étiquette en tenant compte du contexte du mot. En anglais, on a l'habitude de désigner l'étiquette lexicale par le terme «part-of-speech» (POS), en français «partie du discours».

[92] En anglais: «root»
[93] En anglais «prefix»
[94] En anglais «suffix»
[95] En anglais «stemming». Un algorithme qui tronque la fin des mots à partir d'une liste de suffixes.
[96] En anglais: «stemmer»
[97] En anglais «lemmatization». Un algorithme qui retourne le lemme d'un mot ou plus simplement son entrée courante dans un dictionnaire.
[98] En anglais: «lemmatizer»
[99] En anglais: «phrase»

Note sur les différents niveaux d'analyse lexicale (fig. 2.4)

Le premier niveau d'étiquetage est l'étiquetage lexical, l'analyse lexicale ou catégorisation lexicale (en anglais: POS tagging, lexical tagging) qui résulte de l'analyse lexicale ou morpho-lexicale (lexical analysis). Ce premier niveau d'analyse lexicale retourne une liste d'étiquettes lexicales qui comporte une ou plusieurs étiquettes car les mots isolés peuvent être ambigus. Par exemple: «note» peut être un nom ou le verbe «noter».

Un deuxième niveau d'analyse lexicale consiste en la désambiguïsation lorsqu'il y a plusieurs étiquettes possibles. Ce deuxième niveau d'analyse lexicale utilise le contexte du mot pour déterminer la bonne étiquette parmi les étiquettes lexicales possibles.

Parfois l'ambiguïté ne peut être résolue que par référence à la syntaxe ou la connaissance du monde.

Le groupe de mots ou syntagme représente un constituant de la phrase comme un groupe verbal ou un groupe nominal. À ce niveau, on trouve également des expressions figées[100] et des collocations. Encore là, le bruit, les erreurs de syntaxe, les phénomènes propres à la langue parlée comme les hésitations, interruptions, redites, précisions et corrections, finalement les nombreuses exceptions compliquent le traitement des données.

Des algorithmes d'analyse linguistique, appelés analyseurs de fragments[101], se spécialisent dans le traitement des groupes de mots ou syntagmes, sans nécessairement s'intéresser au niveau syntaxique de la phrase complète. À partir de ce niveau, l'ordre des mots (ou la séquence des mots) devient important et les modèles qui traitent la langue comme un simple sac de mots montrent alors leurs limites.

Le quatrième niveau d'organisation est celui la phrase simple ou de la proposition dans le cas de phrases complexes. Au niveau de la phrase, il peut s'avérer important de considérer la division de la phrase en parties par le mécanisme de ponctuation. Cela dit, l'exercice ou plutôt l'art de la ponctuation est difficile et généralement mal maîtrisé. À ce niveau, l'analyse syntaxique le plus souvent basée sur le formalisme des grammaires de dépendance[102] détermine le rôle syntaxique ou grammatical de chaque groupe de mots ou syntagme. Par exemple, le rôle d'un groupe sera «sujet» ou «objet».

Le traitement de la syntaxe en apprentissage profond (fig. 2.5)

En général le traitement au niveau du groupe de mots (syntagme) et de la phrase en apprentissage profond se fait en tenant compte de la séquence des mots. Deux architectures de réseaux de neurones sont principalement employées, le réseau convolutif qui traite la séquence au niveau spatial et le réseau récurrent qui traite la séquence au niveau temporel.

[100] En anglais: «idioms»
[101] En anglais: «chunker»
[102] En anglais: «dependency grammar»

Le prochain niveau d'organisation concerne la division en paragraphes, avant tout sémantique selon la maxime: «Une idée, un paragraphe.». Encore là, le découpage en paragraphes est souvent mal maîtrisé par les usagers de la langue. S'enchaînent ensuite, les niveaux du texte puis du document. Typiquement un document regroupe plusieurs textes qui sont organisés selon une structure type. Par exemple, un livre est un document avec des textes organisés en chapitres, avec une préface, une table des matières, un index, etc.

2.3.1.4 Dispersion des données linguistiques

Une caractéristique importante des données linguistiques est qu'elles sont éparses[103]. Cette dispersion est directement liée à l'énorme productivité combinatoire introduite par la structure composée et hiérarchique de la langue, l'étendue du lexique, les différentes catégories de mots et les multiples niveaux d'organisation de la langue naturelle.

Théoriquement, la combinatoire de la langue naturelle est infinie, car on peut toujours allonger une phrase comme on peut toujours ajouter un nombre à la suite des entiers naturels. En pratique, on s'épuisera bien à un moment, car l'être humain ne dispose ni d'assez de temps ou de patience, ni d'un cerveau infini. On n'a qu'à penser aux interminables phrases de Proust qui ont fait la renommée de l'écrivain, dont une qui fait neuf cent quarante-quatre (944) mots [Van der Velde, van der Voort van der Kleij & de Kamps, 2004].

Dans un contexte de données éparses, un phénomène linguistique de basse fréquence sera difficile à identifier statistiquement.

2.3.1.5 Les données linguistiques sont ambigües

L'ambiguïté des données linguistiques est probablement le pire cauchemar du traitement de la langue naturelle. Un mot ou un énoncé est dit ambigu s'il est susceptible d'avoir plusieurs sens ou interprétations. L'ambiguïté et la polysémie (multiples sens des mots) sont au coeur de la problématique du TLN.

On parle d'ambiguïté lexicale quand un mot peut recevoir plusieurs catégories lexicales ou plusieurs sens (polysémie). Pour être plus précis, quand les sens associés à un même mot n'ont pas d'éléments communs, on parle alors d'homonymie, quand ils en ont, on parle alors de polysémie. L'homonymie désigne le lien de similitude ou de ressemblance entre des mots ayant des sens différents. L'homonymie peut être au niveau de la forme orale (mots ayant le même son), auquel cas on parle d'homophonie (par exemple, «perd» et «paire» sont homophones), ou les fameux «vert», «verre», «vers» et «ver».

[103] En anglais: «sparse»

Un cas d'ambiguïté homophone célèbre en français (fig. 2.6)
Si six scies scient six cyprès, six cents scies scient six cents cyprès.

L'homonymie peut être au niveau de la forme écrite (mot ayant la même orthographe), on parle d'homographie. Par exemple, le fameux «avocat» (homme de loi) qui plaide versus le délicieux «avocat» (fruit) que l'on mange sont des homographes.

Les homographes se divisent en deux groupes : les homographes de mots ayant des catégories grammaticales différentes, qualifiés d'homographes faciles, et les homographes de mots qui ont des catégories grammaticales identiques, les homographes difficiles comme nos «avocats» qui sont tous deux des noms communs.

Un cas d'ambiguïté homographe facile en français (fig. 2.7)
Les poules du couvent couvent.

La désambiguïsation lexicale[104] est la tâche consistant à déterminer le sens d'un mot polysémique dans un contexte donné. Ici, on se réfère implicitement à un inventaire de sens prédéfini. La désambiguïsation lexicale est très utile dans un grand nombre d'applications en traitement automatique des langues naturelles.

Il est possible de lever l'ambiguïté d'un grand nombre de cas d'homographes faciles à l'aide d'une analyse morphosyntaxique. Dans l'exemple ci-haut, une analyse syntaxique permettra de trancher entre le nom «couvent» et le verbe «couver».

Un cas d'ambiguïté homographe difficile en français (fig. 2.8)
Comme il faisait très chaud le jour de l'enterrement, on sortit la bière. bière: cercueil ou boisson?

Il subsiste toutefois des exemples d'homographes faciles qui demeurent ambigus dans des phrases qui ont plusieurs analyses syntaxiques. Dans certains cas, il s'agit même d'ambiguïtés réelles qui resteront ambiguës pour un humain à moins qu'un contexte plus large ou des connaissances externes ne soient utilisés.

Le terme contexte signifie "ce qui entoure un élément". Il peut désigner suivant les cas, le contexte lexical (les mots qui entourent un élément), le contexte syntaxique (les catégories grammaticales qui entourent un élément) ou encore le contexte sémantique (des catégories sémantiques qui entourent un élément).

[104] en anglais: «word sense disambiguation (WSD)»

Un cas d'ambiguïté homographe facile en français (fig. 2.9)
La belle ferme le voile. ferme et voile: verbe ou nom?

En anglais, la situation est pire... Pourtant un mythe répandu veut que l'anglais, dont la grammaire est moins complexe que celle du français, serait plus facile à traiter par un programme informatique. C'est faux! Le traitement automatique d'une langue n'est pas facilité par une grammaire rudimentaire, au contraire il devient plus difficile à cause des ambiguïtés. La langue anglaise possède une syntaxe et une grammaire moins élaborée, surtout au niveau morphologique, que beaucoup de langues à déclinaisons comme le français, ce qui en fait une langue peu précise et fortement polysémique. Or la morphologie peut lever beaucoup d'ambiguïtés.

Par exemple, en anglais, il est usuel qu'un même mot puisse servir de nom, de verbe ou d'adjectif. En anglais, un mot fréquent comme «but», peut être à la fois un nom, un verbe, un adjectif, un adverbe, une conjonction, une préposition, et un pronom. L'ambiguïté lexicale que cela provoque est un grave obstacle à la véritable compréhension d'un texte par un programme informatique. L'effet multiplicatif de l'ambiguïté lexicale et de l'ambiguïté syntaxique peut amener une simple phrase anglaise de quelques mots à avoir des centaines d'interprétations différentes, même si la plupart de ces interprétations sont absurdes[105]. Cela dit, l'emploi de statistiques d'usage permet d'atténuer le problème.

Un cas d'ambiguïté homographe facile en anglais (fig. 2.10)
Empty glasses empty: adjectif, verbe ou nom?

2.3.2 Les données textuelles sont plus difficiles à amplifier

Enfin, il est sans doute plus difficile de générer des données réalistes pour les données textuelles. En effet, alors qu'il est relativement aisé de transformer des données perceptuelles comme des images et des sons, il est plus compliqué de le faire avec du texte tout en demeurant compréhensible.

La difficulté d'amplifier les données textuelles est soulignée dans l'article «Text Understanding from Scratch» [Zhang & LeCun, 2015] qui avance que la reformulation de phrases en paraphrases serait idéal, mais n'envisageait pas pouvoir le faire d'une manière automatique, si ce n'est par la technique de remplacement de mots avec un thésaurus.

[105] Note: D'où la fameuse phrase «Buffalo buffalo Buffalo buffalo bufallo buffalo Buffalo buffalo.», plus clairement «Bison from New York, who are intimidated by bison from New York, intimidate bison from New York.». http://bit.ly/2IpuivM

Récemment, [Kobayashi, 2018] émettait des doutes quant à la possibilité de créer des règles universelles de transformations de textes applicables à différents domaines

La génération de paraphrases, jugée irréaliste et coûteuse (fig. 2.11)
Therefore, the best way to do data augmentation would have been using human rephrases of sentences, but this is unrealistic and expensive due the large volume of samples in our datasets. As a result, the most natural choice in data augmentation for us is to replace words or phrases with their synonyms. [Zhang & LeCun, 2015]

La difficulté de créer des règles de transformations universelles (fig. 2.12)
In natural languages, it is very difficult to obtain universal rules for transformations which assure the quality of the produced data and are easy to apply automatically in various domains. [Kobayashi, 2018]

Pourquoi l'amplification textuelle est si peu répandue? (fig. 2.13)
Les données textuelles sont difficiles à traiter car elles sont symboliques, discrètes, composées, hiérarchiques, éparses, bruitées, pleines d'exceptions et ambigües;
La descente de gradient[106] ne s'applique pas à des données discontinues comme le texte;
Il est plus difficile de générer des données textuelles réalistes, c'est à dire de trouver des transformations sémantiquement invariantes;
Les techniques classiques de traitement de la langue naturelle et surtout celles qui impliquent une intervention humaine à l'exception de l'annotation de corpus sont négligées;

2.3.3 Autres raisons

D'autres raisons plus sociologiques que techniques viennent à l'esprit pour expliquer l'état de sous-développement de l'amplification des données textuelles.

1) Ce n'est pas un vrai problème puisque la Toile regorge de données textuelles. Il suffit de se pencher pour en ramasser;
2) L'effort de recherche se concentre sur la tâche de créer des solutions d'apprentissage bout-en-bout[107] dans une philosophie : «Créer des réseaux de neurones pour nourrir des réseaux de neurones»;

[106] En anglais «gradient descent». La descente de gradient est un procédé d'optimisation des fonctions.
[107] en anglais: «end-to-end learning»

3) Les techniques classiques de traitement de la langue naturelle et surtout tout ce qui implique une intervention humaine à l'exception de l'annotation de corpus sont négligées;
4) Plusieurs le font déjà, mais personne n'en parle car c'est un secret industriel;
5) Par désintérêt, sous le couvert d'arguments du genre: «Cela n'est pas un sujet de recherche important ni intéressant.».

2.4 État de l'art de l'amplification de données textuelles

En raison de la difficulté de la tâche et pour toutes les raisons que nous avons énumérées précédemment, la littérature sur l'amplification ou l'augmentation des données textuelles (ADT) est peu abondante.

2.4.1 La substitution lexicale

Jusqu'à tout récemment, une seule technique d'amplification de données textuelles était assez répandue pour être considérée comme faisant partie de l'état de l'art. Il s'agit de la substitution lexicale (ou remplacement lexical)[108] qui consiste à remplacer un mot par un mot synonyme [Zhang & LeCun, 2015].

Généralement, on ne fait pas la substitution de mots grammaticaux. Voici en ordre de difficulté croissante les types de mots candidats à la substitution lexicale: les adverbes, les adjectifs, les noms et les verbes. La substitution des verbes représente un défi à cause des différents arguments qui accompagnent les verbes (le régime des verbes). Dans beaucoup de situations, on se limitera à remplacer les adverbes et les adjectifs, parfois on ajoutera les noms, très rarement les verbes.

2.4.1.1 Utilisation de dictionnaires

Une première approche pour la substitution lexicale utilise un thésaurus ou un dictionnaire de synonymes du genre Wordnet [Zhang & LeCun, 2015]. Les ressources linguistiques de type WordNet [Miller & al, 1990] ont été laborieusement confectionnées à la main. Il faut donc composer avec l'inventaire des sens particuliers choisis par ceux qui ont créé ces ressources. Rappelons que la substitution lexicale était jusqu'à récemment la seule technique d'amplification textuelle vraiment répandue.

En règle générale, l'algorithme génère tous les synonymes possibles, puis les filtre selon différents critères.

La principale difficulté de la substitution lexicale provient de l'ambiguïté de la langue naturelle. Lorsqu'un mot a plusieurs sens, il a plusieurs synonymes différents.

[108] En anglais: «word replacement»

2.4.1.2 Utilisation de vecteurs-mots

L'idée est basée sur l'hypothèse de distribution en linguistique, selon laquelle les mots ayant des contextes similaires ont des sens proches [Firth, 1957], [Harris, 1954].

Un vecteur-mot ou vecteur contextuel[109] est la représentation d'un mot par un vecteur de faible dimension de nombres réels qui capture des informations syntaxiques et sémantiques à partir de l'examen du voisinage du mot. On explique à l'annexe A8.8.5 que cela revient à l'établissement de la matrice des cooccurrences des mots et à sa réduction en dimension. Deux méthodes sont utilisées: la factorisation matricielle ou l'entraînement d'un réseau de neurones à une couche cachée.

Une fois les vecteurs-mots calculés, l'algorithme trouve les synonymes d'un mot donné en trouvant ses plus proches voisins par évaluation de la similarité. Par exemple, en utilisant la distance entre les vecteurs (similarité cosinus) [Melamud, Levy & Dagan, 2015].

L'emploi de vecteurs-mots est affecté du problème de polysémie. Le vecteur-mot associé à un mot polysémique est une combinaison de tous ses sens. Un autre problème connu des modèles de similarité distributionnelle est qu'ils ont tendance à extraire non seulement des mots qui se ressemblent (synonymes), mais parfois des mots qui s'opposent (antonymes) [Mohammad et al, 2013]. Il arrive qu'un mot et son antonyme soient interchangeables syntaxiquement ce qui fait que les techniques qui identifient des mots semblables par cooccurrence retournent souvent pêle-mêle des synonymes et des antonymes.

2.4.1.2 Autres techniques de substitution lexicale

D'autres techniques de substitutions lexicales ont été développées, celles-ci essentiellement basées sur des réseaux de neurones profonds.

Certains chercheurs comme [Kobayashi, 2018] et [Wang & Yang, 2015] ont proposé des améliorations à la technique du remplacement lexical en tenant compte davantage du contexte. Par exemple, utiliser un modèle de langue[110] entraîné avec un réseau récurrent à longue mémoire court terme (LMCT)[111] pour prédire des mots en fonction du contexte du mot à remplacer [Kobayashi, 2018].

[Fadaee, Bisazza & Monz, 2017] utilisent des réseaux de neurones pour faire des substitutions lexicales en ciblant des mots rares afin d'augmenter le corpus destiné à entraîner des modèles de traduction pour des langues moins dotées. Ils partent d'un corpus aligné et sélectionnent un ensemble de mots rares dans la langue cible. Puis ils utilisent des modèles de langue entraînés avec des réseaux récurrents à longue mémoire court terme (LMCT) pour trouver des mots communs pouvant être substitués par un mot rare dans la

[109] En anglais: «word-vector», «word embedding», ou «embedding». Aussi en français: «vecteur-mot dense», «vecteur d'enrichissement», «plongement», «plongement lexical»
[110] En anglais: «language model (LM)», «LM»
[111] En anglais: «Long Short-Term Memory», «LSTM»

langue cible. Des phrases sont construites en remplaçant le mot commun par le mot rare. Puis grâce à un algorithme d'alignement les phrases sont modifiées dans la langue source. Comme les auteurs l'écrivent, cette méthode de substitution lexicale ne garantit pas la préservation du sens, mais cela n'est pas une nécessité pour l'application visée.

En traduction automatique, [Wang et al, 2018] propose de remplacer de manière aléatoire des mots dans une phrase source et dans la phrase correspondante dans la langue cible par d'autres mots dans les vocabulaires correspondants. Pour cela, ils utilisent des réseaux de neurones profonds en définissant le problème d'amplification textuelle comme un problème d'optimisation.

2.4.2 Les transformations de surface

Une technique souvent mentionnée consiste à transformer les textes localement et en surface à l'aveugle par injection de bruit textuel, au moyen de règles simples ou d'expressions régulières. Ainsi [Kobayashi, 2018] mentionne qu'il y aurait d'autres techniques d'amplification de données textuelles pour des domaines spécifiques avec des règles codées à la main, mais sans citer de sources ni donner d'exemples précis.

Règles codées à la main (fig. 2.14)
Other augmentation methods are known but are often developed for specific domains with handcrafted rules or pipelines, with the loss of generality. [Kobayashi, 2018]

Il est probable que des transformations de surface analogues à nos expériences avec des expressions régulières soient d'usage courant. C'est un secret de polichinelle que l'utilisation de règles codées à la main, incluant l'injection de bruit, est assez répandue pour amplifier les données textuelles. Un exemple pratique est la bibliothèque Python NoiseMix [Bittlingmayer, 2018] qui procède par injection de bruit textuel.

2.4.3 Transformation par rétrotraduction

Une technique émergente, employée par [Prabhumoye et al., 2018], [Edunov & al, 2018] et [Wieting, Mallinson & Gimpel, 2017] est la rétrotraduction[112].
La rétrotraduction est un vieille astuce utilisée pour tester la qualité d'un logiciel de traduction automatique. La rétrotraduction consiste à traduire vers la langue d'origine un texte déjà traduit depuis cette langue.

Historiquement, la première mention de l'emploi de la rétrotraduction, sous le terme «round trip machine translation», pour introduire des variantes dans des données textuelles se trouve dans un article d'une équipe du King's College London présenté à la conférence ISCOL en juin 2015 [Lau, Clark & Lappin, 2015].

[112] En anglais: «back-translation», «backtranslation» ou «round trip translation». En français, on rencontre parfois le terme «rétroversion».

Chapitre 3 - Contribution à l'amplification des données textuelles

Dans ce chapitre nous allons définir notre hypothèse de recherche en réponse à la problématique que nous avons identifiée.

Selon [Zhang & LeCun, 2015] l'emploi de paraphrases serait idéal mais les auteurs n'envisageaient pas pouvoir le faire d'une manière automatique, si ce n'est par la technique de substitution lexicale avec un thésaurus.

Voilà une affirmation qui lance un défi, d'autant plus que le domaine du traitement de la langue naturelle écrite semblait «orphelin» face aux percées de l'amplification de données dans les domaines de la vision par ordinateur et de la reconnaissance de la parole.

3.1 Objectif de cette thèse / hypothèse scientifique de base

L'hypothèse scientifique de base est que l'ajout de données textuelles amplifiées par différentes techniques de génération de paraphrases va faciliter l'entraînement de gros modèles statistiques, plus particulièrement des réseaux de neurones profonds.

Sur le plan pratique, nous voulons développer des techniques d'amplification des données textuelles (ADT), issues du traitement de la langue naturelle et de l'apprentissage automatique, qui soient simples, pratiques et faciles à mettre en oeuvre.

En cela nous cherchons à recréer pour la langue naturelle les techniques d'amplification de données similaires à celles utilisées avec succès en vision artificielle, lesquelles consistent en des transformations des données principalement au niveau du prétraitement.

Nous faisons l'hypothèse que l'amplification de données textuelles par des manipulations textuelles de surface avec des expressions régulières mais également plus profondes par des transformations d'arbres syntaxiques et par rétrotraduction est faisable et utile.

Bien que certaines techniques étudiées s'appliquent à tout genre de textes, nous nous concentrons sur l'amplification de phrases[113]. En effet, on pourrait amplifier des textes définis de façon arbitraire sans tenir compte des frontières de phrases; par exemple des textes entiers ou des fragments de phrases.

[113] En anglais: «sentences»

Objectif (fig. 3.1)
Montrer la faisabilité de techniques d'amplification des données textuelles (ADT) pratiques, simples et faciles à mettre en oeuvre.
Reproduire en TLN des techniques d'amplification des données, similaires à celles utilisées avec succès en vision artificielle, qui consistent en des transformations des données qui respectent des invariants et ce principalement au stade du prétraitement.

Hypothèse scientifique de recherche (fig. 3.2)
L'ajout de données textuelles amplifiées, par la génération de paraphrases respectant l'invariance sémantique, va faciliter l'entraînement de modèles statistiques profonds.
Les paraphrases sont obtenues au stade du prétraitement des données, par l'injection de bruit textuel, l'injection de fautes d'orthographe, des transformations de surface avec des expressions régulières, la substitution lexicale (avec thésaurus et avec des vecteurs-mots), la transformation d'arbres syntaxiques et la rétrotraduction.

Hypothèse d'ingénierie (fig. 3.3)
L'amplification de données textuelles par la transformation d'arbres syntaxiques et par la rétrotraduction est réalisable et pratique en utilisant des services en ligne robustes, capables de monter en charge, faciles à mettre en oeuvre et peu coûteux.

Pour démontrer ces affirmations et valider notre hypothèse de recherche, nous allons réaliser un certains nombre d'expériences qui constituent une preuve de concept. Dans l'esprit de partage de la science et du logiciel libre, l'essentiel des codes et des données nécessaires pour reproduire ces expériences seront fournis dans des carnets IPython pratiques et faciles à installer.[114]

3.2 Définition de l'amplification de données

En vision artificielle, on affirme que si les données sont rares et la distribution des données originales possède des propriétés d'invariance de transformation, la génération de données supplémentaires à l'aide de transformations géométriques peut améliorer les performances [Simard, Steinkraus & Platt, 2003], [Yaeger, Lyon, & Webb, 1997].

Sur le plan statistique, les données supplémentaires générées doivent autant que possible suivre la même distribution statistique que les données originales. Le plus important étant que la distribution statistique des données amplifiées diffère le moins possible des données

[114] https://github.com/ClaudeCoulombe/TextDataAugmentation

que le modèle rencontrera quand il sera mis en production. On parle ici de la capacité de généralisation, c'est à dire la qualité de prédiction du modèle face à de nouvelles données (i.e. des données avec lesquelles le modèle n'a jamais été entraîné). Le même genre de contrainte s'applique aux données de l'ensemble de test qui servent à mesurer la capacité de généralisation du modèle. Pour bien généraliser, il faut que les données amplifiées soient représentatives des nouvelles données auxquelles le modèle sera éventuellement soumis. Cela suggère des améliorations pour raffiner les techniques d'amplification textuelle.

Règle du respect de la distribution statistique (fig. 3.4)
Les données amplifiées doivent suivre une distribution statistique similaire à celle des données originales.

Sur le plan sémantique, l'idée est de trouver des transformations qui n'affectent pas le sens de la donnée mais qui contribuent à l'apprentissage de «nouvelles formes» au sens de la reconnaissance de formes[115]. Cela se complique quand on considère la règle d'or qu'un être humain doit juger «plausible» la donnée amplifiée.

Dans beaucoup de contextes, comme celui de l'apprentissage de phénomènes linguistiques de surface sans tenir compte du sens, la règle de plausibilité n'est pas nécessaire. Cela simplifiera le choix des transformations qui pourront alors être plus facilement automatisées.

Règle d'or de plausibilité (fig. 3.5)
Un être humain ne devrait pas pouvoir distinguer entre les données amplifiées et les données originales. [Géron, 2017b]

Choisir automatiquement une transformation qui «préserve la plausibilité» représente un défi car il faut définir statistiquement une mesure de «plausibilité humaine».

Une première idée serait d'entraîner un modèle supervisé avec des étiquettes sémantiques pour des domaines restreints. Pour cela, on constitue un corpus de paraphrases simples que l'on fait correspondre à des thèmes[116] ou des intentions[117] parmi un ensemble préalablement identifié, et on demande à des annotateurs humains de leur attribuer une étiquette de sens (peut être même plusieurs étiquettes avec une pondération). En pratique, cette technique est utilisée pour annoter des «intentions» dans les systèmes de dialogue. On peut également imaginer associer les paraphrases à des gabarits plus structurés qu'une étiquette unique [Zhang & Wang, 2016].

[115] En anglais: «pattern matching», «pattern recognition»
[116] En anglais: «topics»
[117] En anglais: «intents»

En ce qui nous concerne, l'intelligence humaine sera mise à contribution pour choisir les transformations qui donneront des «données amplifiées plausibles». Par exemple, à un niveau superficiel, on déterminera que le remplacement de la forme «can not» par «can't» ou «cannot» ne change pas le sens. À un niveau plus profond, on acceptera la transformation d'une phrase de la forme active à la forme passive.

Une fois choisie une transformation plausible, l'appliquer à une donnée est simple, mais le problème inverse (l'invariance de transformation) qui consiste à retrouver le sens de la donnée à partir de la donnée transformée peut être très difficile. C'est là qu'intervient l'apprentissage automatique car justement les algorithmes d'apprentissage automatique sont très efficaces pour résoudre les problèmes inverses[118] [Simard, Steinkraus & Platt, 2003].

L'amplification de données est plus facile dans les situations de classification supervisée impliquant des étiquettes cibles simples. Un classificateur prend en entrée un vecteur **X** de grande dimension et l'associe avec une étiquette unique d'une classe cible **y** de faible dimension. Cela signifie que le classificateur doit être invariant à une grande variété de transformations sur les données d'entrée **X** mais l'étiquette cible **y** demeure simple. Il est potentiellement facile de générer de nouvelles paires (**X**, **y**) en transformant les entrées **X** de l'ensemble de données d'entraînement tout en conservant l'étiquette cible **y** [Goodfellow, Bengio & Courville, 2016].

Les applications les plus abouties de l'apprentissage automatique en TLN utilisent l'apprentissage supervisé. Dans une tâche de classification supervisée, on applique des transformations sémantiquement invariantes pour générer des données supplémentaires et on laisse l'algorithme d'apprentissage prédire la classe[119]. La correspondance entre chacune des classes et les différentes paraphrases est apprise et mémorisée dans les paramètres du modèle au moment de l'entraînement, ce qui revient à apprendre une transformation invariante sur la classe. La mise en correspondance est en quelque sorte gratuite et réalisée au moment de l'utilisation du modèle entraîné (au moment de l'inférence) sur de nouvelles données puisque les paramètres du modèle l'ont appris [Simard, Steinkraus & Platt, 2003].

Dans un effort de formalisation, on peut énoncer ce que nous appellerons la règle d'invariance sémantique. L'amplification des données est une transformation invariante sur le sens, ou une transformation invariante pour le sens ou plus simplement une **transformation sémantiquement invariante**. C'est le terme que nous retiendrons au final.

Formulation mathématique de l'invariance sémantique (fig. 3.6)
Mathématiquement, un sous-ensemble E du domaine U d'une transformation $T : U \rightarrow U$ est un ensemble invariant pour cette transformation lorsque $x \in E \implies T(x) \in E$

[118] En anglais: «inverse problem». Résoudre un problème inverse consiste à partir des effets pour trouver les causes. Ici, on partira d'une donnée transformée pour retrouver sa classe.
[119] Note: ou la catégorie, ou l'étiquette associée à la classe ou à la catégorie

Règle d'invariance sémantique (fig. 3.7)
L'amplification des données implique des transformations sémantiquement invariantes.

En apprentissage supervisé, les transformations permises sont celles qui ne changeront pas l'étiquette de classe de la nouvelle donnée générée[120]. Par exemple, pour bien différencier un «b» et un «d» ou un «6» et un «9», la réflexion horizontale et la rotation de 180° ne seraient pas des transformations autorisées pour la reconnaissance de caractères [Goodfellow, Bengio & Courville, 2016]. De manière équivalente avec des données textuelles, on ne pourra sans risque retrancher ou ajouter une négation à une phrase.

Règle d'invariance sémantique en apprentissage supervisé (fig. 3.8)
En apprentissage supervisé, les transformations permises pour l'amplification de données sont celles qui ne modifient pas l'étiquette de classe de la nouvelle donnée générée. [Goodfellow, Bengio & Courville, 2016]

Du point de vue plus de la théorie de l'apprentissage statistique, l'amplification des données est une forme de régularisation car elle permet de réduire le surajustement. On peut aller jusqu'à dire que l'amplification des données est similaire aux méthodes d'apprentissage ensemblistes[121] [Bouthillier et al, 2015].

3.3 Technique 1 - Injection de «bruit textuel»[122]

Ce qui s'approche le plus d'une modification de nature continue dans un texte est l'injection de bruits textuels faibles: modifications, ajouts, retraits de lettres dans les mots, modification de la casse, modification de la ponctuation.

Injection de bruit textuel (fig. 3.9)
L'injection de bruit textuel faible est une transformation sémantiquement invariante.
L'injection de bruit textuel fort n'est pas une transformation sémantiquement invariante.

Nous avons vu que l'injection de bruit dans un réseau de neurones peut être considérée comme une forme d'amplification des données. Cela dit, l'ajout de bruit contribue à la robustesse de l'apprentissage [Xie et al, 2017] mais pas beaucoup à la reconnaissance de nouvelles formes dans les données. On peut très bien en débattre, mais puisque la règle

[120] En anglais: «label-preserving transformations»
[121] En anglais: «ensemble learning». En français: «apprentissage ensembliste», «apprentissage par combinaison de modèles».
[122] En anglais: «textual noise»

d'invariance s'applique, l'injection de bruit faible sera donc considérée comme une technique d'amplification de données textuelles (ADT) et soumise à l'expérience.

Évidemment, la génération de bruit textuel fort comme le changement de mots sans restriction et la génération de texte aléatoire ou sans contrainte sémantique ne passe pas le test d'invariance de sens. Le chaos n'est pas propice à la découverte de régularité statistique.

3.4 Technique 2 - Injection de fautes d'orthographe

L'examen des textes des critiques de films montre qu'ils contiennent un bon nombre de fautes d'orthographe[123]. L'idée est de générer des textes contenant des fautes d'orthographe usuelles afin d'entraîner nos modèles qui deviendront ainsi plus robustes à ce type particulier de bruit textuel et même reconnaître de nouvelles formes dans les données.

L'algorithme d'injection de fautes d'orthographe se base sur une liste des erreurs d'orthographe les plus usuelles[124] en anglais. Cette liste a été inspirée de la liste compilée par l'éditeur des dictionnaires Oxford [Oxford Dictionaries, 2018].

Injection de fautes d'orthographe (fig. 3.10)
L'injection de fautes d'orthographe est une transformation sémantiquement invariante.

Transformations par des fautes d'orthographe - exemples (fig. 3.11)
cc => c, mm => m, ie => ei, gg => g, en => an, ss => s, ally => ly, nn => n, => r, si => is, ar => er, r => rr, bb => b, ery => ary, eur => r, ea => e, m => mm, tt => t, ely => ly, sc => c, os => ous, ite => ate, mm => mn, s => ss, pp => p, sy => cy, nm => m, ahr => ar, iar => ar, ll => l, uo => o, ei => ie, ore => or, or => ur, orw => ow, g => j, rnm => rm, ua => au, ed => d, asy => acy, ely => ly, ally => ly, ible => able, edg => eg, iai => ia, li => ly, hal => al, ea => a, l => ll, aoh => oah, ia => a, gue => ge, ly => ally, iou => ou, mem => me, se => ce, ar => er, gue => ge, ore => or, ely => ly, re => ree, wh => w, pa => pe, un => oun

3.5 Amplification textuelle par substitution lexicale

Nous avons expérimenté différentes améliorations de la substitution lexicale[125] avant de nous orienter davantage vers la recherche de nouvelles techniques d'amplification de données textuelles (ADT).

[123] En anglais: «spelling error», «misspelling»
[124] En anglais: «common misspellings»
[125] En anglais: «lexical replacement». Aussi en français: «remplacement lexical».

Rappelons que l'amplification textuelle par substitution lexicale et l'emploi de règles de transformations écrites à la main sont probablement les deux techniques d'ADT les plus répandues. Ce type d'amplification des données devrait contribuer à la reconnaissance de nouvelles formes dans les données.

Une première approche d'amplification textuelle consiste tout simplement à mettre un mot à la place d'un autre [Zhang & LeCun, 2015]. On parle ici de «substitution lexicale» ou encore de «remplacement lexical», ou «remplacement par thésaurus»[126]. Nous préférons le terme «substitution lexicale» pour des raisons esthétiques afin de respecter le suffixe en «tion» commun aux autres techniques d'ADT étudiées.

Pour la substitution lexicale, on favorise l'emploi d'hyperonymes (mot plus général, tulipe => fleur) et on évitera l'emploi d'hyponymes (mot plus précis, fleur => tulipe).

Règles pour la substitution lexicale (fig. 3.12)
La substitution d'un mot par un vrai synonyme est une transformation sémantiquement invariante.
La substitution d'un mot par un hyperonyme (mot plus général) pas trop éloigné[127] *est une transformation sémantiquement invariante.*
La substitution d'un mot par un hyponyme (mot plus spécifique) n'est généralement pas une transformation sémantiquement invariante.
La substitution d'un mot par un antonyme n'est pas une transformation sémantiquement invariante.

3.5.1 Technique 3 - Substitution lexicale par l'emploi d'un dictionnaire

La substitution lexicale consiste à proposer un ou plusieurs mots qui pourront se substituer à un mot donné. Ces mots sont typiquement de vrais synonymes du mot à remplacer.

En réalité, la notion de «synonymes» est floue car il n'existe pas vraiment deux mots distincts qui soient parfaitement équivalents. Il y a toujours des nuances, parfois subtiles, par exemple entre les mots «vrai», «véridique» et «véritable». L'usage se satisfaisant de moins de rigueur, on considère que deux mots sont synonymes ou plutôt similaires si on peut les remplacer l'un pour l'autre sans changer la valeur de vérité de la phrase.

Les dictionnaires comme WordNet sont organisés de façon à associer à chaque entrée du dictionnaire une liste qui contient plusieurs ensembles de synonymes. Chacun de ces

[126] En anglais: «word replacement using thesaurus»
[127] Note: Dans la mesure où l'hyperonyme n'est pas un concept trop général. C'est pourquoi les classes très générales comme «object», «spirit» et «substance» doivent être filtrées dans WordNet,

ensembles de synonymes ou ensyns[128] correspond à un sens particulier. Le nombre de sens est variable et dépend le plus souvent d'une annotation humaine qui peut être plus ou moins exhaustive et justifiée.

Le défi consiste à choisir le bon ensemble de synonymes ou ensyn. Plusieurs stratégies peuvent être employées pour trouver le bon ensyn. Par exemple, on peut choisir le sens le plus répandu en se basant sur le nombre d'occurrences dans un corpus. On peut aussi s'aider du contexte du mot. Pour cela, on utilise les textes qui accompagnent chaque ensyn dans WordNet comme des définitions, des exemples ou des phrases types, puis on calcule la similarité entre le contexte du mot que l'on cherche à remplacer et ces textes d'accompagnement. Au final, on choisira l'ensyn qui montre la plus forte similarité.

Le calcul de similarité repose sur les mots communs trouvés dans le contexte du mot et les textes associés à chacun des ensyns. La métrique utilisée est appelée la similarité cosinus [129]. La similarité cosinus part du fait que les textes sont représentés par de longs vecteurs de cooccurrences et que mathématiquement la distance entre deux vecteurs est déterminée par le cosinus de l'angle qu'ils font entre eux [Manning & Schütze, 1999], [WIKIPÉDIA, Cosine similarity]. Le calcul de similarité repose plus concrètement sur le calcul de la fréquence des mots communs trouvés dans le contexte du mot et les textes associés à chacun des ensyns.

Formulation mathématique - similarité cosinus (fig. 3.13)

Soient $Texte_A$ et $Texte_B$ représentés par les vecteurs de fréquences de mots A et B

$$similarité(A,B) = \cos(\theta_{A,B}) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|}$$

$$similarité(A,B) = \cos(\theta_{A,B}) = \frac{\sum_{mot=1}^{n} A_{mot} B_{mot}}{\sqrt{\sum_{mot=1}^{n} A_{mot}^2} \sqrt{\sum_{mot=1}^{n} B_{mot}^2}}$$

Parfois, il peut arriver qu'en naviguant dans le réseau des synonymes de WordNet on aboutisse à des mots antonymes. Dans ce cas, on filtre la sortie à l'aide de l'IPA[130] de WordNet pour les antonymes.

[128] En anglais: «synsets»
[129] En anglais: «cosine similarity»
[130] En anglais: «API», «application programming interface»

3.5.2 Technique 4 - Substitution lexicale par l'emploi de vecteurs-mots

L'idée repose sur l'hypothèse distributionnelle en linguistique qui dit que les mots qui ont des contextes similaires ont des significations proches [Firth, 1957], [Harris, 1954] .

3.5.2.1 Mots linguistiquement liés

Dans le jargon linguistique, deux mots qui peuvent être substitués l'un pour l'autre sont en relation paradigmatique[131]. Par exemple, les mots «chien» et «chat», ou encore «lundi» et «mardi». Il est important de distinguer la relation paradigmatique de la relation syntagmatique[132] qui décrit plutôt une relation sémantique plus éloignée du genre «chien» et «aboyer» ou «avion» et «piloter».

Les mots en relation paradigmatique (qui appartiennent à la même classe sémantique de mots) ont tendance à se retrouver aux mêmes endroits dans une phrase (ou dans une séquence de mots). Ils ont des contextes similaires. Au contraire les unités en relation syntagmatique ont tendance à se retrouver simplement en cooccurrence.

3.5.2.2 Collocation ou cooccurrence

Les informaticiens et statisticiens préfèrent le terme «cooccurrence» et les linguistes utilisent davantage le terme «collocation». Il est difficile de distinguer les notions de collocation et de cooccurrence que beaucoup d'auteurs considèrent comme des synonymes.

Toutefois, il existe une nuance. La cooccurrence est une relation qui existe entre des choses apparentées, par exemple la relation entre le mot «juge» et le mot «avocat». Autrement dit, on considère qu'il y a cooccurrence lorsque la présence d'un mot dans un texte donne une indication sur la présence d'un autre mot. Les mots qui figurent aux côtés de l'occurrence d'un mot-cible dans un texte forment des cooccurrences, et sont appelés ses cooccurrents.

Pour les linguistes, la notion de collocation implique une relation syntaxique habituellement localisée dans un syntagme alors que la cooccurrence est une notion plus générale et souvent à plus grande distance et où l'ordre d'apparition des mots est sans importance (hypothèse du sac de mots). La collocation est une relation plus forte caractérisée par une unité syntaxique et sémantique et dont le sens ne peut pas être dérivé du sens de chacune de ses composantes individuelles. Typiquement, c'est un groupe de deux ou plusieurs mots qui correspondent à une façon conventionnelle de dire les choses. Par exemple «déclaration sous serment» ou «conseiller juridique», «faim de loup». Cela dit, la collocation est souvent très proche d'une expression figée et peut éventuellement devenir une unité lexicale comme l'expression «au fur et à mesure».

[131] En anglais: «paradigmatic relation»
[132] En anglais: «syntagmatic relation»

3.5.2.3 Mots statistiquement liés

Plusieurs mesures statistiques permettent de quantifier la tendance de deux mots à apparaître en même temps. Une mesure souvent utilisée est l'information mutuelle.
Par exemple, si le mot «boson» et le mot «Higgs» étaient indépendants, alors connaître le mot «boson» ne donnerait aucune information sur le mot «Higgs» et vice versa. Leur information mutuelle serait donc nulle. À l'autre extrême, si le mot «boson» est fortement lié au mot «Higgs» et que le mot «Higgs» est fortement lié au mot «boson», alors toute information véhiculée par le mot «boson» serait partagée avec le mot «Higgs». La connaissance du mot «boson» déterminerait alors la valeur du mot «Higgs» et vice versa.

Les vecteurs-mots ou vecteurs contextuels proposent essentiellement des mots "liés" qui ont tendance à apparaître statistiquement dans le même contexte, mais pas nécessairement de vrais synonymes.

En gros, un algorithme de création de vecteurs-mots rapproche les mots qui apparaissent dans des ”contextes" similaires et éloigne les mots qui se trouvent dans des contextes différents en ajustant des nombres dans un vecteur. On dit aussi que l'algorithme retourne des vecteurs de mots qui cooccurrent dans des contextes similaires.

Pour plus de détails, consultez l'annexe A8.8.1

3.5.2.4 Problème des antonymes

Un problème notoire des algorithmes de substitution lexicale basés sur la similarité de distribution est que ces algorithmes récupèrent non seulement des mots qui se ressemblent fortement (des synonymes), mais aussi des mots qui diffèrent fortement au niveau du sens (des antonymes).

Ce comportement est en grande partie dû à la similarité distributionnelle des synonymes et des antonymes [Mohammad et al, 2013]. Souvent un mot et son antonyme sont interchangeables syntaxiquement ce qui fait que les techniques qui identifient des mots similaires par cooccurrence retournent pêle-mêle des synonymes et des antonymes.

Pour solutionner ce problème, nous avons opté pour le filtrage des mots en sortie avec un dictionnaire des antonymes. Pour cela nous utilisons l'IPA de WordNet pour les antonymes.

3.5.2.5 Problème de polysémie

Le problème le plus important avec l'emploi de vecteurs-mots pour la substitution lexicale est la polysémie. Les algorithmes d'extraction de vecteurs-mots comme Word2Vec produisent des vecteurs-mots polysémiques qui mélangent leurs multiples sens. En fait, les techniques de génération de vecteurs-mots opèrent sur un mot sans considérer qu'il peut être ambigu

ou polysémique. Par exemple, toute l'information pour le mot «java» se retrouvera condensée dans un unique vecteur-mot associé à «java» que ce soit le lieu géographique, le café ou le langage de programmation.

3.5.2.6 Solutions au problème de polysémie

Pour résoudre le problème de polysémie, une bonne idée serait de modifier l'algorithme de génération de vecteur-mot (par exemple, Word2Vec) afin qu'il sépare les différents sens. Par exemple, en y intégrant un algorithme de groupage[133] du genre K-moyennes[134] ou espérance-maximisation (EM)[135] ou encore l'algorithme des k plus proches voisins[136].

Deux travaux de recherche se sont imposés dans la littérature scientifique. D'abord le travail de chercheurs de l'Université Hébraïque de Bar-Ilan en Israël [Melamud, Levy & Dagan, 2015] et le travail d'un groupe de recherche sur les méthodes bayésiennes à Moscou en Russie [Bartunov et al, 2016].

Melamud, Levy et Dagan ont eu l'idée de modifier la fonction de similarité dans Word2Vec pour la rendre sensible au contexte. Le nouvel algorithme baptisé Word2Vecf traite des contextes arbitraires, pas seulement des mots dans une fenêtre. Ainsi si on interroge Word2Vecf avec le mot «java» et le contexte «programs written in java», il retournera «Python» ou «C++» mais pas «indonesia» ou «coffee» [Melamud, Levy & Dagan, 2015].

De leur côté, Bartunov, Kondrashkin, Osokin et Vetrov proposent AdaGram (Adaptive Skip-gram), une extension de Word2Vec. AdaGram intègre un algorithme de groupage semblable à l'algorithme EM (espérance-maximisation) [Bartunov et al, 2016].

Notons également au passage les travaux de [Wang & Yang, 2015] qui proposent d'améliorer le choix des mots de substitution par un calcul de similarité avec un algorithme d'apprentissage du genre K plus proches voisins et des vecteurs-mots.

3.5.2.7 L'algorithme AdaGram

Après analyse, nous avons décidé d'utiliser l'algorithme AdaGram (Adaptive Skip-gram) [Bartunov et al, 2016], [Bartunov, 2015]. Ce qui a motivé notre choix était 1) AdaGram résout le problème de polysémie des vecteurs-mots en générant des vecteurs-mots séparés selon les différents sens 2) AdaGram offre des modèles préentraînés sur de gros corpus 3) AdaGram offre également une interface de programmation (IPA) avec la fonction `disambiguate(...)` qui à partir d'un mot et d'un contexte retourne une liste de vecteurs-mots regroupés par sens. De plus, chaque vecteur-mot retourné est accompagné

[133] En anglais: «clustering». Aussi en français: «algorithme de partitionnement».
[134] En anglais: «k-means»
[135] En anglais: «expectation–maximization», «EM»
[136] en anglais: «k-nearest neighbors», «KNN»

d'un coefficient de similarité avec le contexte. On peut ensuite extraire les mots voisins du mot cherché dans l'espace du vecteur-mot [Coulombe, 2017a], [Coulombe, 2017b].

AdaGram[137] est une extension de la variante grammes-voisins (Skip-gram) de Word2Vec qui cherche à prédire les voisins d'un mot. Skip-gram en français devient l'algorithme des grammes-voisins, donc un algorithme qui prédit des voisins d'un mot ou grammes[138].

Le principal problème de l'algorithme des grammes-voisins (Skip-Gram) est le calcul du terme au dénominateur qui assure la normalisation. Cela requiert un calcul de l'ordre de la taille du lexique $O(V)$ pour chaque mot. Word2Vec résout le problème avec une approximation astucieuse. De son côté, AdaGram modifie l'échantillonnage de l'algorithme des grammes-voisins en utilisant une fonction softmax hiérarchique qui construit un arbre de Huffman pour le lexique ce qui fait passer la complexité à $O(log(V))$ [Bartunov et al, 2016].

De plus, AdaGram ajoute à Word2Vec un algorithme de groupage apparenté à l'algorithme EM (espérance-maximisation)[139] [Bartunov et al, 2016] afin de résoudre le problème des vecteurs-mots polysémiques.

Rappelons que l'algorithme espérance-maximisation (EM) est un algorithme d'apprentissage non-supervisé qui permet d'estimer les paramètres d'un modèle statistique qui comporte des variables latentes (non observables) par l'application de la méthode du maximum de vraisemblance[140] sur des données. C'est un problème d'optimisation [Do & Batzoglou, 2008].

Comme son nom l'indique, une fois les paramètres θ initialisés aléatoirement, l'algorithme EM comporte deux étapes: 1) Une étape d'évaluation de l'espérance (E) de la fonction de vraisemblance. Cette étape pondère dans quelle mesure chaque donnée contribue à l'estimation de la vraisemblance maximale. 2) Une étape de maximisation (M) de la fonction de vraisemblance trouvée à l'étape E où les paramètres sont ajustés en fonction des données qui ont été repondérées.

Les paramètres mis-à-jour à l'étape M sont réinjectés à l'étape E et on itère ainsi jusqu'à convergence (les paramètres ne changent plus).

Puisque l'algorithme EM converge vers un maximum local, pour trouver un maximum global il faut reprendre le calcul un grand nombre de fois avec des paramètres initiaux différents. Ci-dessous, une description de l'algorithme après quelques optimisations [Bartunov et al, 2016].

---

[137]Note: En français on pourrait qualifier AdaGram d'algorithme grammes-voisins adaptatif
[138]En anglais: «Skip-gram». En français: «algorithme de prédiction des grammes-voisins», «algorithme grammes-voisins», «GV». Un algorithme qui prédit les mots (grammes) voisins d'un mot (gramme).
[139] En anglais: «expectation maximization», «EM»
[140] En anglais: «maximum likelihood estimation», «MLE». Aussi, en français: «estimation du maximum de vraisemblance».

Algorithme AdaGram (fig. 3.14)

- Construction d'un arbre de Huffman (softmax hiérarchique) pour le dictionnaire
- Initialisation
  - approximations des paramètres
  - groupes de sens déterminés par un processus «rupture de bâton»
- Une seule passe dans l'ensemble des données d'entraînement
  - Étape E (espérance) - calcul des probabilités des sens y_i pour chaque donnée x_i et la variable latente z_i et $K(x_i)$ est le nombre total de sens pour le mot x_i

$$p(z_j|x_i, y_i) = \frac{p(y_i|x_i, z_i)\,p(z_i|x_i)}{\sum_{k=1}^{K(x_i)} p(y_i|x_i, k)\,p(z_i = k|x_i)}$$

  - Étape M (maximisation) - une itération de descente de gradient stochastique pour mettre à jour les paramètres θ du modèle:

$$\theta_{nouveau} = \theta_{ancien} + \alpha_i \sum_{k=1}^{K(x_i)} p(z_i = k|x_i, y_i)\nabla_\theta \log p(y_i|x_i, k)$$

Pour son algorithme de groupage AdaGram met en jeu des techniques statistiques bayésiennes non-paramétriques[141] appelées processus de Dirichlet qui permettent au nombre de variables latentes (ici le nombre de groupes) d'augmenter au besoin. C'est ce qui donne le caractère «adaptatif» et son nom à l'algorithme, AdaGram, pour «Adaptive Skip-gram». Notez que le plus souvent ces variables continuent à suivre des distributions paramétriques comme des gaussiennes.

Dans le cas qui nous intéresse, le nombre de groupes[142] n'est pas fixe et il peut augmenter avec la quantité de données traitée. Un paramètre α, appelé paramètre de dispersion, contrôle le nombre de nouveaux groupes formés. Plus α est grand, plus il y a de groupes.

L'assignation a priori de groupes de sens (ensyn) pour chaque mot suit un processus dit de «rupture de bâton»[143] [Chen, 2012], [Coulombe, 2019]. On part avec un bâton de longueur un (1) qui représente la probabilité d'assigner une donnée à un groupe unique. La longueur résiduelle résultant de la rupture du bâton est gouvernée par une distribution statistique $Beta(1, \alpha)$ qui retourne un nombre réel β_1 entre 0 et 1 avec une espérance de $1/(1+\alpha)$ [WIKIPÉDIA, Beta distribution]. La longueur $l_1 = \beta_1 * 1$ représente la longueur du nouveau bâton (groupe) formé disons à gauche. On reprend le processus de formation d'un nouveau groupe en brisant le bâton résiduel de droite (de longueur $1 - l_1$) avec un nombre réel

[141] Note: Le nombre de paramètres n'est pas fixe, il est potentiellement infini et dépend de la quantité de données traitée. Aussi, le modèle n'est pas fixe et grossit avec la complexité des données.
[142] En anglais: «clusters»
[143] En anglais: «stick-breaking process». Il s'agit bien de diviser un groupe comme si l'on brisait un bâton. L'expression «à bâtons rompus» serait intéressante, mais elle ne s'utilise plus aujourd'hui que pour qualifier une discussion.

généré aléatoirement β_2 ce qui laisse un bâton de longueur $l_2 = \beta_2(1 - l_1)$. Et ainsi de suite. Dans le cas d'AdaGram, le maximum de groupes (ou de sens) a été fixé à 30. On obtient ainsi une série de probabilités d'assigner un mot à chacun des groupes ce qui permet d'assigner les différents sens d'un mot à priori [Bartunov et al, 2016].

3.6 Amplification textuelle par génération de paraphrases

En continuant d'explorer les techniques d'amplification de données textuelles (ADT), nous en sommes venu à concevoir, comme [Zhang & LeCun, 2015], que l'amplification des données textuelles passe idéalement par la génération de paraphrases.

3.6.1 Définition de paraphrase

Par définition, une paraphrase est une forme de surface alternative dans la même langue qui exprime le même contenu sémantique que la forme originale [Madnani & Dorr, 2010]. [Chen & Dolan, 2011] proposent la paraphrase idéale.

Définition de la paraphrase idéale (fig. 3.15)
In addition to being meaning-preserving, an ideal paraphrase must also diverge as sharply as possible in form from the original while still sounding natural and fluent.[144] [Chen & Dolan, 2011]

Les paraphrases peuvent apparaître à différents niveaux. Par exemple, les mots ayant le même sens, communément appelés synonymes, peuvent également être considérés comme des paraphrases lexicales. Il existe des paraphrases au niveau du groupe de mots ou syntagme[145] comme avec «take over» et «assume control of», ainsi qu'au niveau de la phrase complète[146], par exemple «I finished my work.» et «I completed my assignment.» [Madnani & Dorr, 2010].

Il est important de noter que le problème de génération de paraphrases est similaire au problème de l'inférence textuelle[147]. Une inférence textuelle est définie comme une implication logique entre textes. Quelqu'un lisant le premier texte, dit texte de référence «R», peut raisonnablement inférer que le second texte, dit texte hypothèse «H», est vrai [Gleize, 2016]. Par exemple, à la lecture du texte de référence: «Au cours d'histoire, j'ai appris que Jacques Cartier, Samuel de Champlain, le Sieur de Laviolette, et Pierre-Lemoyne D'Iberville étaient de grands explorateurs.», on peut inférer que l'hypothèse «Champlain était un

[144] Note: En plus de préserver le sens, une paraphrase idéale doit aussi diverger aussi radicalement que possible de la phrase originale tout en restant naturelle et fluide.
[145] En anglais: «phrasal paraphrase»
[146] En anglais: «sentential paraphrase»
[147] En anglais: «text entailment»

explorateur.» est vraie. Par contre on ne peut rien inférer avec l'hypothèse «Laviolette fonda Trois-Rivières.» La relation entre une phrase et sa paraphrase peut ainsi être vue comme une inférence textuelle bidirectionnelle [Androutsopoulos & Malakasiotis, 2010].

Notez que la typologie des paraphrases établie par [Vila, Martí & Rodríguez, 2014] a été un guide important dans notre recherche.

Il est intéressant d'évaluer la combinatoire de toutes les paraphrases possibles. En y réfléchissant un peu, on ne peut l'évaluer dans l'absolu à cause des subtilités de la sémantique. Cela dit, on peut facilement l'évaluer quand on connaît le mécanisme de génération des paraphrases.

En considérant un mécanisme de génération par combinaison aléatoire des différentes variantes des parties d'une phrase, par des substitutions lexicales ou des transformations simples, on arrive au simple produit du nombre de variantes pour chaque partie. En limitant les parties à des mots, on arrive au produit du nombre des différents mots possibles.

Calcul de la combinatoire d'une paraphrase (fig. 3.16)

En ne considérant que des substitutions lexicales ou des transformations simples sur des parties d'une phrase

$$NbreParaphrases(phrase) = NbreParaphrases(partie_1, \ldots, partie_n)$$

$$NbreParaphrases(partie_1, \ldots, partie_n) = NbreVariantes(partie_1)\times, \ldots, \times NbreVariantes(partie_n)$$

$$NbreParaphrases(partie_1, \ldots, partie_n) = \prod_{i=1}^{n} NbreVariantes(partie_i)$$

Si $partie_i$ = $mots_i$ alors:

$$NbreParaphrases(mot_1, \ldots, mot_n) = \prod_{i=1}^{n} NbreVariantes(mot_i)$$

3.6.2 Définition de paratexte

Définissons un paratexte comme un texte obtenu par le remplacement de phrases par des paraphrases.

Définition de paratexte (fig. 3.17)

Un paratexte est un texte obtenu par le remplacement de phrases par des paraphrases.

Notez que le mot paratexte est déjà utilisé en théorie littéraire mais dans un sens très différent. Il désigne l'ensemble hétéroclite de tout ce qui accompagne et met en valeur un texte comme les titres, sous-titres, noms d'auteur, illustrations, notes de bas de pages, etc. [Genette, 1997].

Maintenant, intéressons-nous au calcul de la combinatoire des paratextes. Le nombre de paratextes différents pouvant être générés est le simple produit du nombre de paraphrases possibles pour chacune des phrases qu'il contient.

Calcul de la combinatoire d'un paratexte (fig. 3.18)

$$NbreParatextes(texte) = NbreParatextes(phrase_1, \ldots, phrase_k)$$

$$NbreParatextes(phrase_1, \ldots, phrase_k) = NbreParaphrases(phrase_1)\times, \ldots, \times NbreParaphrases(phrases_k)$$

$$NbreParatextes(phrase_1, \ldots, phrase_k) = \prod_{j=1}^{k} NbreParaphrases(phrase_j)$$

En intégrant le calcul de la combinatoire des paraphrases:

$$NbreParatextes(texte) = \prod_{j=1}^{k} \prod_{i=1}^{n} NbreVariantes(phrase_j, mot_i)^*$$

**Note : comporte un algorithme de division d'un texte en phrases et en mots*

3.6.3 Techniques de génération de paraphrases

Quiconque familier avec le traitement de la langue naturelle (TLN) sait qu'il devrait être possible de générer des paraphrases à partir de grammaires. Le premier type de transformations qui vient à l'esprit concerne les transformations de surface. Une transformation de surface désigne une transformation qui ignore la syntaxe et qui s'effectue avec de simples règles de reconnaissance de formes[148].

Trois techniques sont étudiées: 1) des transformations lexicales ou syntaxiques superficielles par la substitution de chaînes de caractères au moyen d'expressions régulières[149], 2) des transformations profondes qui sont basées sur des manipulations d'arbres syntaxiques, 3) la génération de paraphrases en utilisant la rétrotraduction.

Nous pensons que ces techniques d'ADT devraient contribuer à la reconnaissance de nouvelles formes dans les données.

[148] En anglais: «pattern matching», «pattern recognition»
[149] En anglais: «regular expressions», «regex»

3.6.4 Technique 5 - Génération de paraphrases par des expressions régulières

Dans un premier temps, les transformations de surface qui peuvent être réalisées avec des expressions régulières sont à privilégier car elles sont simples, une fois surmonté l'aspect rébarbatif des expressions régulières, et vraiment très performantes en terme de calculs.

Une expression régulière est un formalisme qui décrit la forme générale d'une chaîne de caractères. Les expressions régulières sont utilisées pour rechercher des chaînes de caractères d'une certaine forme dans un texte. Une fois les chaînes trouvées, on peut leur appliquer un traitement, comme un ajout, une modification ou une suppression [WIKIPÉDIA, Regular expression].

Parmi les transformations de surface, beaucoup sont dépendantes de la langue traitée; comme les contractions en anglais. Il existe cependant des transformations de surface communes à plusieurs langues comme l'injection de bruit, l'injection de certains types de fautes d'orthographe déjà traitées précédemment, des transformations sur des objets culturellement partagés comme les dates, les noms de lieux, les unités de mesure.

Dans le même ordre d'idées, il y a toutes les transformations au niveau des abréviations, sigles, acronymes, notations et variantes orthographiques. On a potentiellement des milliers de règles que l'on peut en partie rendre plus générales en factorisant leurs comportements.

Par exemple, en anglais, la transformation d'une forme verbale vers une forme contractée (contraction) et son inverse (expansion) est considérée comme une transformation sémantiquement invariante à la condition qu'on ne vienne pas résoudre une ambiguïté qui entraînerait un potentiel contresens. Évidemment cela se produit dans le contexte de transformations mécaniques et locales où il n'y a aucun moyen de trancher les ambiguïtés.

Exemples d'une transformation textuelle de surface (fig. 3.18)
Le passage d'une forme verbale vers une forme contractée et vice versa est une transformation de surface sémantiquement invariante à la condition que l'on préserve les ambiguïtés.
`I am => I'm, you are => you're, he is => he's, it is => it's, she is => she's, we are => we're, they are => they're,, I have => I've, you have => you've, we have => we've, they have => they've, he has => he's, it has => it's, she has => she's, I will => I'll, you will => you'll, he will => he'll, are not => aren't, is not => isn't, was not => wasn't, ..., can't => cannot, won't => will not, ...`

Pour préserver la propriété d'invariance sémantique, il est permis d'introduire des ambiguïtés mais il est interdit de résoudre des ambiguïtés qui pourraient entraîner un contresens. Par exemple les transformations «he is» vers «he's» et «he has» vers «he's» seront permises même si elles introduisent des phrases ambiguës. Mais les transformations inverses «he's» à «he is» et «he's» vers «he has» sont interdites car elles risquent

d'introduire un contresens. En fait, un traitement local de surface ne peut lever ce genre d'ambiguïté. Pour traiter le cas général, il faudrait manipuler des structures syntaxiques.

Exemple de transformations interdites (fig. 3.19)
she's => she is *she's => she has*

Règle empirique du «respect de l'ambiguïté» (fig. 3.20)
Une transformation qui génère une ambiguïté ou une imprécision peut généralement être considérée comme sémantiquement invariante.
Une transformation qui lève une ambiguïté, en précisant une information, ne peut pas être considérée comme une transformation sémantiquement invariante, à moins que l'information précisée ne soit motivée par le contexte.

Il faut se méfier car certaines transformations, en apparence simples, exigent une manipulation plus profonde, par exemple pour respecter les accords grammaticaux ou parce qu'il y a des obstacles. Ces transformations sont impossibles à faire avec de simples expressions régulières à partir d'une forme de surface. Il faut travailler plus profondément au niveau syntaxique.

Par exemple, pour la transformation «must» vers «have to», l'accord avec le sujet du verbe «have» demande une manipulation plus profonde. Beaucoup de ces transformations peuvent être réalisées avec des manipulations d'arbres syntaxiques que nous aborderons dans la prochaine section.

3.6.5 Technique 6 - Génération de paraphrases par transformation d'arbres

La génération de paraphrases par la transformation d'arbres syntaxiques a été le point de départ de cette étude, étant vue comme une méthode originale d'amplification textuelle. La technique est directement inspirée des travaux de Michel Gagnon de Polytechnique Montréal qui a codirigé cette recherche [Gagnon & Da Sylva, 2005], [Zouaq, Gagnon & Ozell, 2010]. Au cours de ses recherches, Michel Gagnon a montré comment condenser des textes en transformant des arbres de dépendance issus d'un analyseur morpho-syntaxique à large couverture. L'analyseur, à base de règles, constituait le cœur du «Correcteur 101», un correcteur grammatical du français qui a connu un grand succès commercial [Bourdon et al, 1998], [Doll, Drouin, & Coulombe, 2005] et auquel a participé l'auteur de cette thèse.

Dans le même domaine, citons les travaux de [Rascu & Schmidt, 2005] qui utilisaient une grammaire d'arbres adjoints[150] pour générer des paraphrases. Profitons de l'occasion pour rappeler les travaux pionniers du linguiste Igor Mel'čuk de l'Université de Montréal sur les grammaires de dépendance [Bourdon et al, 1998], [Mel'cuk, 1988b].

Le résultat de l'analyse d'une phrase selon un formalisme en grammaire de dépendance est généralement un arbre dont les nœuds sont les mots de la phrase et les arêtes (liens) les dépendances syntaxiques entre les mots.

Exemple d'arbres de dépendance produite par SyntaxNet (fig. 3.21)

A man eats an apple in the kitchen.

```
root eats/eat/verb/proper_unknown/present/third/gender_unknown/singular
    nsubj man/man/noun/proper_unknown/tense_unknown/person_unknown/gender_unknown/singular
        det A/A/det/proper_unknown/tense_unknown/person_unknown/gender_unknown/number_unknown
    dobj apple/apple/noun/proper_unknown/tense_unknown/person_unknown/gender_unknown/singular
        det an/an/det/proper_unknown/tense_unknown/person_unknown/gender_unknown/number_unknown
    prep in/in/adp/proper_unknown/tense_unknown/person_unknown/gender_unknown/number_unknown
        pobj kitchen/kitchen/noun/proper_unknown/tense_unknown/person_unknown/gender_unknown/singular
            det the/the/det/proper_unknown/tense_unknown/person_unknown/gender_unknown/number_unknown
    p ././punct/proper_unknown/tense_unknown/person_unknown/gender_unknown/number_unknown
```

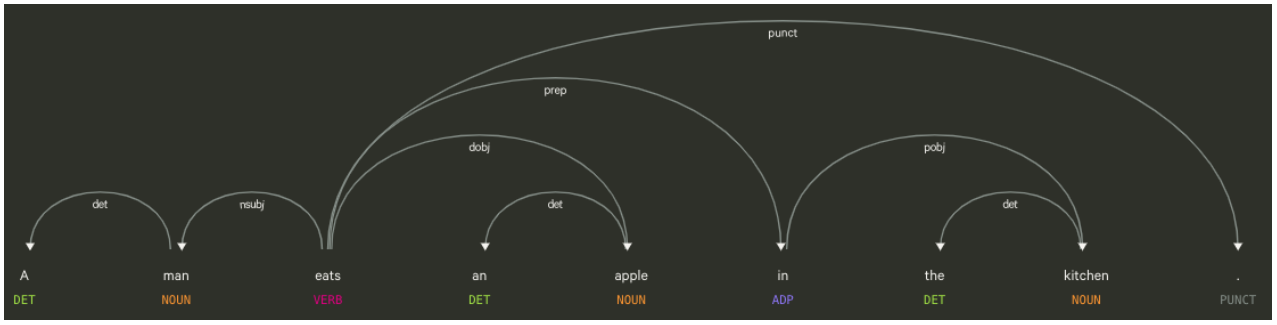


Schéma dessiné avec l'aide de spaCy [Honnibal & Montani, 2017]

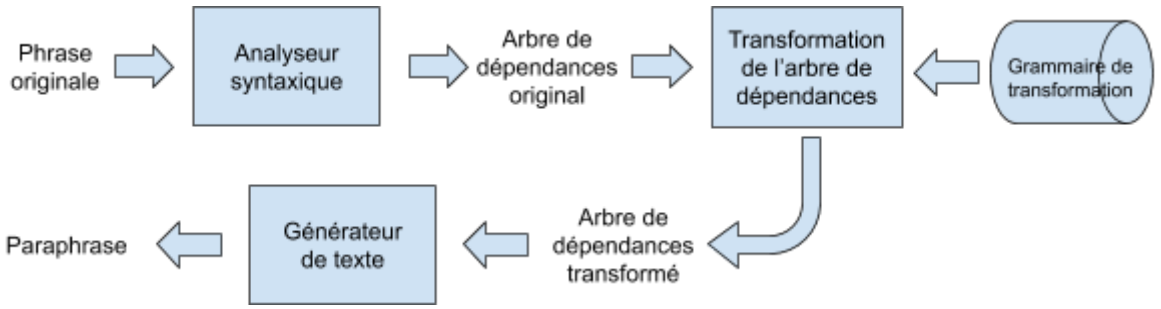
Dans un arbre de dépendance, on pourra trouver la dépendance entre le verbe et le nom qui est à la tête du groupe nominal (ou syntagme) sujet, ou bien la dépendance entre un nom et l'adjectif qui le modifie.

Pour fonctionner, le générateur de paraphrases a besoin d'un analyseur morphosyntaxique à large couverture qui produit des analyses de phrases sous la forme d'arbres de dépendance syntaxiques[151]. De nos jours, il existe de puissants analyseurs pour plus d'une cinquantaine de langues qui sont disponibles soit en code source libre soit sous la forme de services en ligne. Pour réaliser nos expériences, nous avons utilisé le service en ligne Language Cloud API de Google [Google, 2018a] basé sur SyntaxNet [Petrov, 2016], [Kong et al, 2017].

[150] En anglais: «tree-adjoining grammar», «TAG»
[151] En anglais: «dependency syntax trees»

L’analyseur syntaxique reçoit en entrée la phrase originale et produit en sortie un arbre de dépendance. Ensuite, le générateur de paraphrases transforme cet arbre de dépendance en un nouvel arbre de dépendance en utilisant une grammaire de transformations sémantiquement invariantes. L’arbre syntaxique ainsi transformé sert à générer une nouvelle forme de surface, c’est-à-dire une paraphrase.

Génération de paraphrases par transformation d’arbres (fig. 3.22)



Le code à base de règles du générateur de paraphrases a été originalement écrit en Prolog puis traduit en Python dans un outil appelé PolyPhrases.

Comparaison d’outils de génération de paraphrases (fig. 3.23)

A man eats an apple in the kitchen. To be or not to be. I took a train yesterday.			
A man eats an apple in the kitchen. to be or not to be. I caught a train last night.	A man eats an apple in the kitchen. Regarding life, what to think about it. I took a prepare yesterday.	A man fare Associate in Nursing apple within the room. To be or to not be. I took a train yesterday.	An apple is eaten by a man in the kitchen. To be or not to be. I took it yesterday.
outil commercial 1	outil commercial 2	outil commercial 3	PolyPhrases

Pour le prototype, les règles de transformations, assez générales, sont construites manuellement, en se basant sur la typologie des paraphrases établie par [Vila, Martí & Rodríguez, 2014]. L'objectif était de s’occuper en priorité du 20% des règles qui couvre 80% des cas, conformément au principe d'ingénierie de Pareto 20/80. Marie Bourdon, linguiste informaticienne chez Coginov inc Montréal, a apporté une aide précieuse [Bourdon et al, 1998].

Pour l’identification de règles et de gabarits[152] supplémentaires, il est possible d'imaginer l’emploi de techniques d'extraction par autorenforcement[153] comme celles décrites par [Androutsopoulos & Malakasiotis, 2010].

[152] En anglais: «templates»
[153] En anglais: «bootstrapping»

Transformations sémantiquement invariantes d'arbres (fig. 3.24)
Le passage de la forme passive à la forme active et vice versa est une transformation sémantiquement invariante.
Le remplacement d'un nom ou d'un syntagme nominal par un pronom est une transformation sémantiquement invariante.
Le retrait d'un adjectif, d'un adverbe, d'un groupe adjectival ou d'un groupe adverbial est une transformation sémantiquement invariante.

Illustration du passage de la voix active à la voix passive (fig. 3.25)

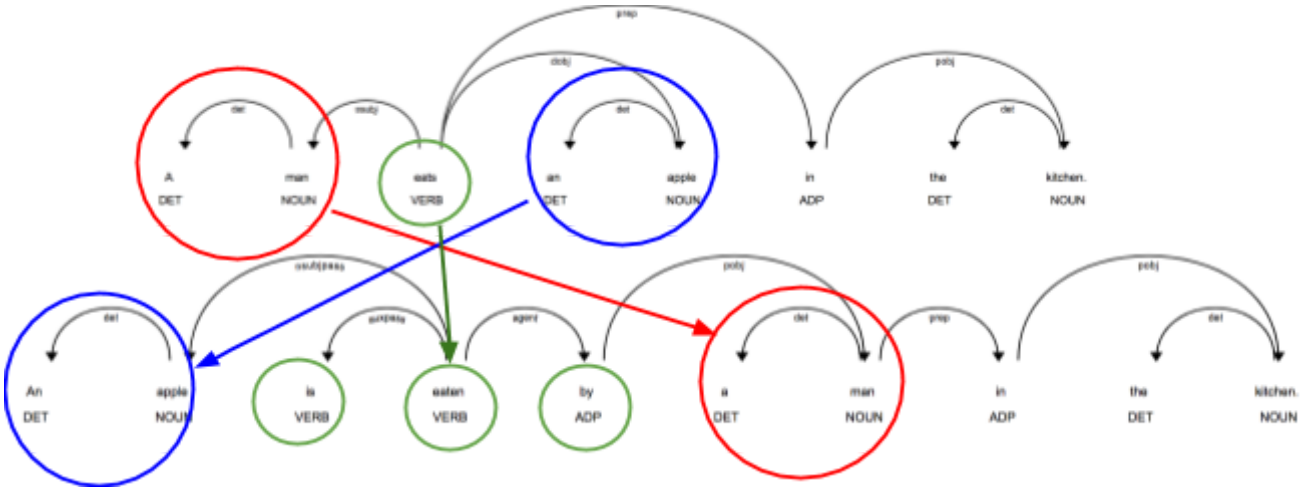


Schéma produit avec l'aide de spaCy [Honnibal & Montani, 2017]

Passage à la voix passive de la phrase «A man eats an apple in the kitchen.» La tête de la structure de dépendance est le verbe «eat». La règle de transformation commence par échanger le groupe sujet «man» (en rouge) et le groupe objet «apple» (en bleu) puis l'accord du verbe «eat» est modifié (en vert) pour donner une nouvelle structure de dépendance qui, une fois aplatie, génère la phrase «An apple is eaten by a man in the kitchen.».

Cette technique de génération de paraphrases basée sur la transformation d'arbres syntaxiques est la plus innovatrice, au sens de la plus exclusive à ce travail de recherche. Elle est également plus précise car elle permet des interventions ciblées, voire chirurgicales. Son principal inconvénient est quelle nécessite un travail manuel assez important pour construire des régles.

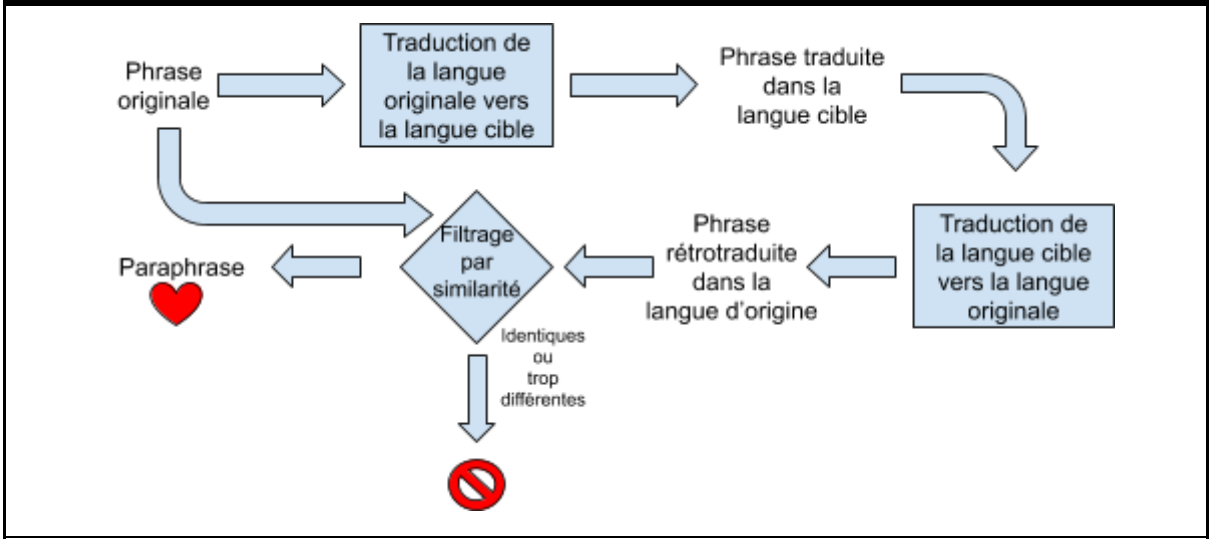
3.6.6 Technique 7 - Génération de paraphrases par rétrotraduction[154]

La rétrotraduction est une vieille astuce utilisée pour tester la qualité d'un logiciel de traduction automatique. Elle consiste à traduire vers la langue d'origine un texte déjà traduit dans une autre langue, d'où le nom rétrotraduction.

À vrai dire, il n'y a pas de traduction correcte et unique d'une phrase dans une langue donnée. En fait, il existe toujours un grand nombre de traductions équivalentes à cause de l'immense productivité combinatoire de la langue naturelle et de la complexité du monde réel. Par définition, toutes ces traductions équivalentes sont des paraphrases. Sur le plan opérationnel, on considérera comme une bonne paraphrase toute rétrotraduction non identique mais suffisamment similaire à la phrase d'origine.

Grâce aux progrès de la traduction automatique neuronale[155] basée sur l'apprentissage profond, l'emploi de la rétrotraduction est devenu une technique intéressante à exploiter.[156]

Générateur de paraphrases basé sur la rétrotraduction (fig. 3.27)



Rappelons que les avancées récentes en traduction automatique découlent en grande partie des travaux pionniers du MILA de l'Université de Montréal dirigé par Yoshua Bengio sur la traduction neuronale qui ont été perfectionnés et industrialisés par Google. Par exemple, les modèles neuronaux du langage [Bengio et al, 2003] et l'idée de vecteur-mot à l'origine de Word2Vec, les architectures encodeur/décodeur [Cho et al, 2014a], [Cho et al, 2014b] et le

[154] En anglais: «back-translation», «backtranslation» ou «round trip translation». En français on dit parfois «rétroversion».
[155] En anglais: «neural machine translation», «NMT»
[156] Note: L'emploi de la rétrotraduction m'a été suggérée par le physicien Antoine Saucier à l'occasion d'une «discussion de corridor» à la Polytechnique de Montréal au printemps 2015.

mécanisme d'attention [Bahdanau, Cho & Bengio, 2014] qui est au coeur de toutes les architectures qui entraînent aujourd'hui les gros modèles de langue.

Un traducteur automatique neuronal reçoit une phrase originale en entrée et retourne une phrase traduite dans la langue cible. Puis cette phrase en langue cible est traduite à son tour (rétrotraduite) vers la langue d'origine par un autre appel au traducteur neuronal.

Le résultat de la rétrotraduction est filtré afin de récupérer les bonnes paraphrases. Si la rétrotraduction est identique à la phrase originale, elle est immédiatement rejetée. Le cas échéant on procède à une mesure de similarité entre le texte de la rétrotraduction et le texte original. Pour être tout à fait rigoureux, il s'agit plutôt d'identifier si la rétrotraduction peut être considérée comme une paraphrase du texte original. On part de l'observation que la paraphrase est souvent similaire au texte original [Dolan et al, 2004]. Mais ce n'est pas toujours le cas et de bonnes paraphrases seront écartées. À la fin, la rétrotraduction est conservée lorsque sa similarité avec le texte original est supérieure à un certain seuil qui a d'abord été fixé empiriquement à 25%.

Amplification de données textuelles avec la rétrotraduction (fig. 3.26)
La rétrotraduction de bonne qualité est une transformation sémantiquement invariante.
La rétrotraduction de mauvaise qualité n'est pas une transformation sémantiquement invariante.

Il existe un grand nombre de méthodes pour mesurer la similarité entre deux textes [Bär, Zesch & Gurevych, 2015], du simple compte du nombre de mots en commun, en passant par le cosinus dans un espace de vecteurs [Manning & Schütze, 1999], jusqu'au plus récentes méthodes à base de réseaux de neurones capables d'identifier des paraphrases [Lan & Xu, 2018]. Dans un premier temps, our des raisons de mise en place rapide et de performance de calcul, nous avons opté pour un simple mesure de la différence de longueur entre la rétrotraduction et le texte original. Une technique similaire est utilisée par [Wieting, Mallinson & Gimpel, 2017]. Une mesure de similarité grossière entre un texte original et sa rétrotraduction consiste à calculer leur nombre de mots respectif. Le seuil a été fixé rapidement par essai et erreur à 25%, mais cela pourrait faire l'objet d'ajustements plus fins.

Dans un second temps, un petit modèle de classification logistique a été entraîné avec l'idée de remplacer l'heuristique précédente. La tâche est de déterminer par apprentissage supervisé si l'on doit conserver ou écarter la rétrotraduction proposée. Pour cela quelques 1200 rétrotraductions ont été étiquetées manuellement. Les attributs des données d'entraînement sont: la phrase source, la rétrotraduction, la langue cible, la longueur de la phrase source, la longueur de la rétrotraduction, la métrique BLEU[157] de la rétrotraduction et

[157] En anglais: «BLEU score» pour «bilingual evaluation understudy». L'algorithme BLEU permet de comparer un texte à un corpus de référence en considérant le recouvrement des n-grammes.

de la phrase originale et une étiquette booléenne de classe pour indiquer si l'on conserve ou l'on écarte la rétrotraduction. L'exactitude du modèle basé sur ce petit classificateur logistique[158] est de 66% sur des données test. Il serait intéressant de poursuivre dans cette voie en exploitant davantage les textes de la phrase source et de la rétrotraduction.

À la différence de [Edunov & al, 2018] ou [Wieting, Mallinson & Gimpel, 2017], qui entraînent leurs propres modèles de traduction automatique neuronale, ce qui nécessite des ressources informatiques considérables, il est apparu plus pratique d'utiliser les services de traduction en ligne de Google via Google Translate API. Rappelons que Google Translate, le service de traduction en ligne de Google, utilise la traduction automatique neurale (NMT) basée sur l'apprentissage en profondeur. Une fois dûment enregistré et la clé d'accès au service Google Translate obtenue, tout le code requis tient dans une petite fonction Python.

3.7 Combinaison de transformations

On peut combiner les différentes transformations pour générer davantage de données. La bonne nouvelle est qu'en général la combinaison de transformations sémantiquement invariantes est elle-même une transformation sémantiquement invariante.

Combinaison de transformations sémantiquement invariantes (fig. 3.28)
En général la combinaison de transformations sémantiquement invariantes est une transformation sémantiquement invariante et cela jusqu'à ce que le bruit introduit par chaque transformation ne devienne trop important.

Attention! Il y a un ordre à respecter. On commence par les transformations profondes, puis on applique les transformations de surface, comme l'injection de bruit textuel. Autrement le bruit introduit fait obstacle aux traitements plus profonds.

Ordre d'application des transformations sem. invariantes (fig. 3.29)	
1	*Génération de paraphrases par transformation d'arbre syntaxique*
2	*Génération de paraphrases par rétrotraduction*
3	*Génération de paraphrases au moyen d'expressions régulières*
4	*Substitutions lexicales*
5	*Injection de fautes d'orthographe*
6	*Injection de bruit textuel*
Note: Les transformations 1, 2, 3, 4 sont interchangeables et peuvent même se répéter avec le risque que certaines transformations s'annulent. Une certaine dérive du sens se produira par application successive de substitutions lexicales. Enfin, les transformation 5 et 6 produiront rapidement des textes trop bruités.	

[158] En anglais: «logistic classifier». Parfois improprement appelé «régression logistique».

Le jeu du téléphone consiste à faire circuler une phrase de bouche à oreille et à voix basse à travers une file de joueurs. Le premier joueur dans la file invente une phrase et le dernier de la file récite la phrase qu'on lui a communiquée. L'intérêt du jeu est de comparer la version finale de la phrase à sa version initiale.

En combinant avec soin les transformations proposées, on peut à la fois obtenir des paraphrases de plus en plus éloignées de la phrase de départ tout en conservant le sens. Toutefois, il faut rester vigilant car à chaque transformation on court le risque de s'éloigner de plus en plus du sens original. D'où la règle empirique du jeu du téléphone.

Règle empirique dite du jeu du téléphone (fig. 3.30)
Afin de respecter l'invariance sémantique, il faut limiter le nombre de transformations successives ou combinées.

3.8 Une combinatoire explosive!

Maintenant, intéressons-nous à la combinatoire des transformations. Par exemple, évaluons le nombre de transformations possibles pour le remplacement des mots dans une phrase. Le nombre de phrases possibles augmente très rapidement avec la longueur de la phrase originale et le nombre de mots qu'on peut y remplacer. Soit N le nombre de mots que l'on peut remplacer dans une phrase. En supposant que l'on a K synonymes pour chaque mot, on arrive à un total de K^N phrases possibles.

Une phrase moyenne en anglais compte 14 mots[159] [Sichel, 1974] dont 4 sont des mots grammaticaux (25%) [Ramos, 2017], cela laisse tout de même 10 mots à remplacer et 3 choix de mots (le mot original lui-même et 2 synonymes en moyenne [Latour, 2011]). Au total cela fait $3^{10} = 59\,049$, soit près de 60 000 phrases possibles. Une phrase de 20 mots, assez fréquente, générera potentiellement plus de 14 millions de paraphrases (3^{15} = 14 348 907) et cela en utilisant uniquement la technique de remplacement de mots. Donc, il semble bien qu'on ne manquera pas de données!

Une question légitime à se poser est de déterminer le facteur d'amplification[160] optimal. Par exemple, en vision artificielle, une règle empirique parle de la création d'environ 1000 images par image originale. Toujours avec les images, il ne semble pas intéressant de créer une nouvelle image en ne modifiant qu'un seul pixel.

[159] Note: En 2019, le nombre de mots est probablement plus près de 10. Source:http://bit.ly/2Xo8B2l
[160] En anglais: «amplification factor»

Facteur d'amplification (fig. 3.31)

Le facteur d'amplification est égal au nombre de données amplifiées sur le nombre de données originales.

$$f_{amp} = \frac{\#\ données\ amplifiées}{\#\ données\ originales}$$

A priori, il semblerait que l'on puisse obtenir des facteurs d'amplification beaucoup plus élevé pour les données textuelles. Il semble raisonnable de mettre dans la balance l'allongement du temps de traitement, la quantité de variabilité à ajouter à chaque nouvelle données et l'amélioration des performances du modèle profond. Cela demanderait de nouvelles expériences avec des moyens de calcul plus importants.

Puisque cette étude ne vise qu'à montrer la faisabilité des différentes techniques d'amplification, les expériences se sont limitées à un facteur d'amplification de cinq (5), soient cinq paratextes par texte original, ce qui représente une infime partie de toutes les combinaisons possibles.

3.9 Échantillonnage aléatoire

Face à l'explosion combinatoire, il s'avère impossible de générer toutes les paraphrases ou tous les paratextes, que ce soit pour les utiliser directement, ni même pour en échantillonner un sous-ensemble après coup.

Nous devons donc coder un algorithme capable de générer à la demande un échantillon[161] de paratextes ou de paraphrases sans avoir à tous les générer puis procéder au tirage d'un échantillon. De toute façon cela serait impossible à cause de l'énorme combinatoire sous-jacente dont nous avons déjà discuté. De plus, nous aimerions utiliser le même algorithme pour générer des phrases à partir de mots ou des textes à partir de phrases. Pour les phrases nous parlons de paraphrases (combinaisons de mots) et pour les textes nous utilisons le terme paratextes (combinaisons de phrases).

Nous ne proposons pas une technique générale car elle reste fortement liée aux techniques d'amplification textuelle étudiées, mais elle peut être utile pour de nombreuses applications.

3.9.1 Dictionnaires de variantes

Deux structures de données sont définies: une première structure de données pour les paraphrases (ParaPhrasesDict) et une seconde pour les paratextes (ParaTextesDict). Techniquement, des dictionnaires Python sont utilisés.

[161] En anglais: «sample»

Ces structures de données sont utilisées pour décrire les différentes variantes possibles de chaque partie d'un texte. Il s'agit d'une table associative ou dictionnaire qui comporte un index pour chacune des parties (partie_1, partie_2, …, partie_n) auquel est associée une liste des variantes avec une information complémentaire sur la fréquence ou un poids heuristique. Pour nos expériences, nous n'avons utilisé que la fréquence, mais d'autres pondérations et choix d'heuristiques pourraient s'avérer utiles.

Structure de donnée ParaPhrasesDict (fig. 3.32)

```
paraphrases_dict = {
    "index_mot_1":{"mot_1":poids_1_0,"variante_1_1":poids_1_1,...,"variante_1_i":poids_1_i},
    "index_mot_2":{"mot_2":poids_2_0,"variante_2_1":poids_2_1,...,"variante_2_j":poids_2_j},
    …,
    "index_mot_n":{"mot_n":poids_n_0,"variante_n_1":poids_n_1,"...,"variante_n_m":poids_n_m},
}
```

```
exemple_paraphrases_dict = {
    "1":{"le":8,"un":4,"ce":2},
    "2":{"petit":3,"gros":2,"beau":10},
    "3":{"chien":10,"pitou":2},
    "4":{"aboie":3,"jappe":2,"mange":1},
    "5":{".":10}
}
```

Structure de donnée ParaTextesDict (fig. 3.33)

```
paratextes_dict = {
    "index_phrase_1":{"phrase_1_0":poids_1_0,"phrase_1_1":poids_1_1,...,"phrase_1_i":poids_1_i},
    "index_phrase_2":{"phrase_2_0":poids_2_0,"phrase_2_1":poids_2_1,...,"phrase_2_j":poids_2_j},
    …,
    "index_phrase_n":{"phrase_n_0":poids_n_0,"phrase_n_1":poids_n_1,"...,"phrase_n_m":poids_n_m}
}
```

```
exemple_paratextes_dict = {
    "1":{"un chien aboie":4,"un petit pitou jappe":1,"ce gros chien jappe":3},
    "2":{"la caravane passe":3,"le convoi passe":2},
    "3":{"le soleil brille":10,"l'astre du jour rayonne":2}
}
```

3.9.2 Algorithme d'échantillonnage aléatoire de variantes textuelles

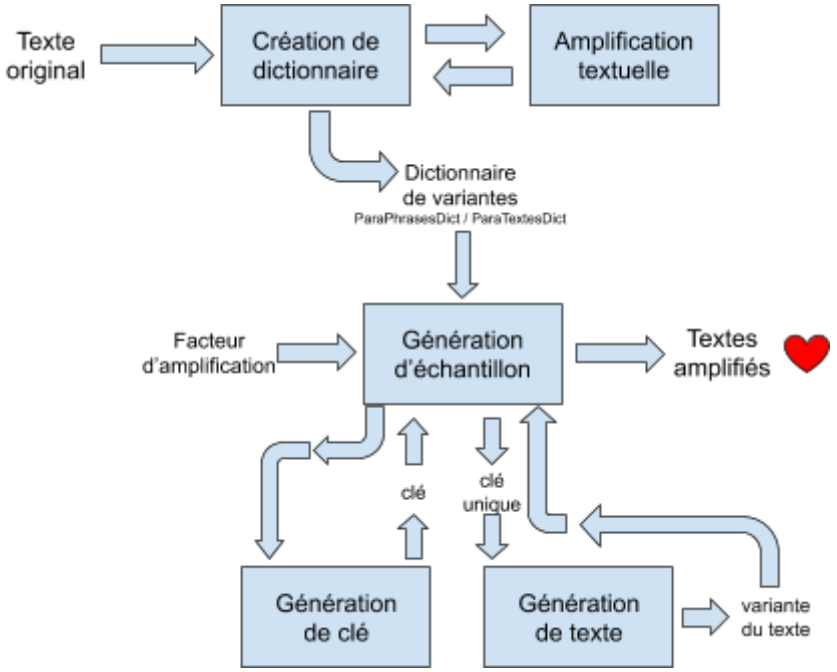
Le même algorithme d'échantillonnage aléatoire est utilisé pour générer les différentes paraphrases possibles (combinaisons de mots) et les différents paratextes possibles (combinaisons de phrases).

Le générateur reçoit en entrée le texte à amplifier et le facteur d'amplification. En sortie, le générateur produit le nombre de textes (paraphrases ou paratextes) désirés.

L'algorithme d'échantillonnage comporte: 1) un module de création de dictionnaire 2) un module d'amplification de données textuelles 3) un module de génération de clé 4) un module de génération d'échantillon 5) un module de génération de texte.

Le module de création de dictionnaire reçoit une phrase ou un texte et produit un dictionnaire de variantes, soit une structure de données de type ParaPhrasesDict ou ParaTextesDict. Rappelons qu'un dictionnaire de variantes décrit les différentes variantes pour chaque partie d'un texte (mots ou phrases), incluant le mot ou la phrase originale. Pour enrichir le texte original en variante, le module de création de dictionnaire fait appel au module d'amplification textuelle.

Algorithme d'échantillonnage aléatoire (fig. 3.34)



Création d'un dictionnaire de paraphrases - exemple (fig. 3.35)

```
Entrée phrase originale: "Le mignon chiot jappe."

Après enrichissement par le module d'amplification textuelle, le module de création de dictionnaires
retourne un dictionnaire de paraphrases (ParaPhrasesDict)

exemple_paraphrases_dict = {
    "1":{"le":8,"un":4,"ce":2},
    "2":{"joli":3,"mignon":2,"beau":10},
    "3":{"chiot":2,"chien":10,"clébard":1},
    "4":{"aboie":3,"jappe":2,"hurle":1},
    "5":{".":10}
}
```

Le module de génération de clé retourne une clé d'échantillonnage aléatoire qui encode une combinaison de variantes parmi les différentes combinaisons possibles du dictionnaire de variantes. La clé est formée par la concaténation de choix aléatoires pour chacune des variantes possibles dans le dictionnaire de variantes (ParaPhrasesDict ou ParaTextesDict).

Le module de génération de clé joue un rôle déterminant (clé!) dans la stratégie d'échantillonnage des résultats de l'amplification textuelle. Cette stratégie est hybride, elle combine le hasard et une heuristique. Trois paramètres déterminent la clé d'échantillonnage.

Un premier tirage aléatoire (basé sur une loi normale) détermine le nombre d'éléments variables, `nbr_variantes`, d'un dictionnaire feront l'objet d'une amplification textuelle.

Clé d'échantillonnage aléatoire - nombre d'éléments (fig. 3.36)

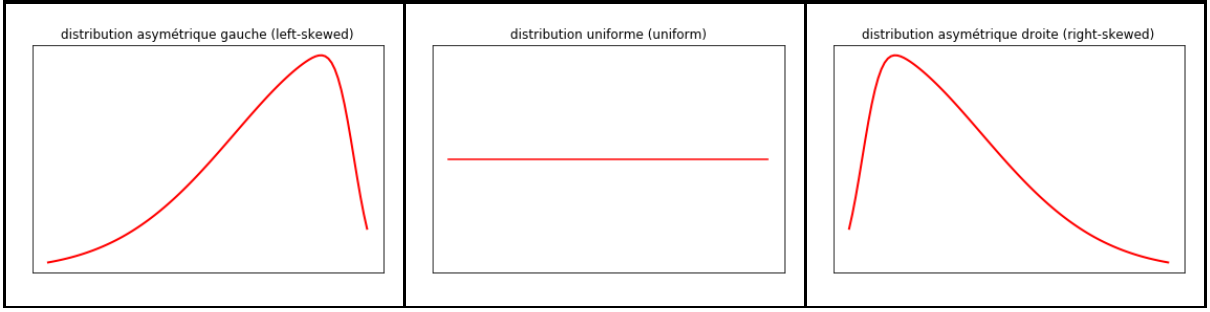
Soit le exemple_paraphrases_dict ci-dessous, qui comporte quatre (4) éléments variables, 1, 2, 3 et 4, le 5e étant fixe (ayant un seul choix possible).

```
exemple_paraphrases_dict = {
    "1":{"le":8,"un":4,"ce":2},
    "2":{"joli":3,"mignon":2,"beau":10},
    "3":{"chiot":2,"chien":10,"clébard":1},
    "4":{"aboie":3,"jappe":2,"hurle":1},
    "5":{".":10} }
```

Un premier tirage aléatoire détermine que trois (3) éléments variables seront amplifiés. Donc nbr_variantes contient 3.

Un deuxième tirage choisit `nbr_variantes` éléments parmi les éléments variables du dictionnaire. Ce tirage est pondéré par une liste de poids que l'on peut assimiler à une distribution statistique discrète. Un poids est associé à chacun des éléments. Un élément associé à un poids élevé sera davantage susceptible d'être choisi qu'un élément associé à un faible poids. Trois distributions statistiques sont prédéfinies: une distribution asymétrique gauche[162] qui favorise les éléments en fin de dictionnaire, une distribution asymétrique droite [163] qui favorise les éléments en début de dictionnaire et une distribution uniforme. Le code a été conçu pour que l'usager puisse aussi définir une distribution de son choix.

Distributions statistiques prédéfinies pour l'échantillonnage (fig. 3.37)



[162] En anglais: «left skewed»
[163] En anglais: «right skewed»

Clé d'échantillonnage aléatoire - choix des éléments variables (fig. 3.38)

Un second tirage aléatoire choisit `nbr_variantes` éléments variables à amplifier selon une distribution statistique prédéfinie. Par exemple, une distribution asymétrique gauche pourrait retourner la liste de 3 éléments `[2,3,4]`.

```
exemple_paraphrases_dict = {
    "1":{"le":8,"un":4,"ce":2},
    "2":{"joli":3,"mignon":2,"beau":10},
    "3":{"chien":10,"chiot":2,"clébard":1},
    "4":{"aboie":3,"jappe":2,"hurle":1},
    "5":{".":10} }
```

Troisièmement, un poids heuristique associé à chacune des variantes participe au choix la variante qui sera utilisée pour l'amplification.

Clé d'échantillonnage aléatoire - choix des variantes (fig. 3.39)

Des variantes sont choisies pour chacun des éléments ciblés. Ces choix sont influencés par les poids heuristiques associés à chaque variante. Par exemple, pour l'élément 2, la variante "beau" d'indice 2, qui est associée au poids 10 est choisie.

```
exemple_paraphrases_dict = {
    "1":{"le":8,"un":4,"ce":2},
    "2":{"joli":3,"mignon":2,"beau":10},
    "3":{"chien":10,"chiot":2,"clébard":1},
    "4":{"aboie":3,"jappe[164]":2,"hurle":1},
    "5":{".":10} }
```

clé d'échantillonnage aléatoire générée: `1-0_2-2_3-1_4-1_5-0`

Comme son nom l'indique le module de génération d'échantillon accumule et orchestre la génération des variantes textuelles (mots ou paraphrases). Il reçoit en entrée un dictionnaire de variantes (ParaPhrasesDict ou ParaTextesDict) et un facteur d'amplification. Le module de génération d'échantillon commence par demander une clé au module de génération de clé, s'assure que la clé obtenue est nouvelle et unique (pas déjà générée), le cas échéant il demande une nouvelle clé, enfin il demande au générateur de texte de produire un texte à partir de la clé. Le module de génération d'échantillon accumule les textes générés jusqu'au moment où le facteur d'amplification est atteint ou qu'il soit impossible de l'atteindre après un certain nombre maximum d'essais.

Génération d'un échantillon de texte - exemple (fig. 3.40)

En retour de la clé d'échantillonnage aléatoire `1-0_2-2_3-1_4-1_5-0` et du dictionnaire `exemple_paraphrases_dict`, le module de génération de texte produira la phrase: *"Le beau chien jappe."*

[164] Note: Ici on comprend que le choix est aléatoire...

Pour traiter de grandes quantités de données, impossible à mettre en mémoire vive, on pourrait facilement virtualiser l'algorithme en stockant les structures de données sur disque. Le prix à payer serait de ralentir le traitement dans le cas d'un processeur unique. Pour corriger ce problème, on pourrait distribuer le calcul sur plusieurs serveurs et mettre en place des flux pour partager les données.

3.10 Limites de l'amplification de données

Nous avons vu qu'en combinant des transformations sémantiquement invariantes, on s'éloigne de plus en plus du sens original en ajoutant de plus en plus de bruit. À un moment donné, les données amplifiées ne sont que du bruit. D'où la règle empirique du jeu du téléphone qui limite le nombre de transformations et la quantité de données amplifiées.

L'ajout de données peut conduire à un meilleur modèle si les informations et attributs ajoutés sont pertinents au problème (signaux pertinents), mais cela peut également conduire à des résultats pires si les informations ajoutées ne sont pas suffisamment pertinentes (bruit) ou carréent contradictoires. Aussi, l'ajout d'information, même pertinente, peut introduire une variance qui vient annuler la diminution de biais dans le délicat équilibre biais-variance.

De plus, le mécanisme d'amplification des données pourrait être affecté par des biais de différentes sources que nous regroupons sous le nom de «biais d'amplification» tout à fait analogue à un biais d'échantillonnage. Il faudrait pouvoir mieux identifier, catégoriser mesurer et prévenir le biais d'amplification. Ce travail reste à faire.

Chapitre 4 - Expérimentation / méthodologie

4.1 Choix de la tâche - prédiction de la polarité d'un texte

Pour tester notre hypothèse scientifique, nous avons décidé de simplifier le problème. L'idée est de choisir un problème simple qui fait intervenir un jeu de données normalisé et des architectures courantes de réseaux de neurones profonds. On espère ainsi arriver plus facilement à isoler l'effet de l'amplification des données textuelles (ADT) à des fins de comparaison des résultats avec des données amplifiées et sans données amplifiées.

Nous avons opté pour une tâche d'analyse des sentiments[165], également appelée analyse des opinions ou analyse des émotions, qui consiste à identifier et caractériser automatiquement les états affectifs et les opinions à partir de textes ou de transcriptions de la parole. L'analyse des sentiments peut être appliquée aux avis de clients d'un commerce, aux billets de blogue, ou micro-blogues (gazouillis Twitter), aux réponses à des enquêtes ou sondages, au traitement des questions ouvertes et aux médias sociaux.

Dans sa forme la plus simple, la tâche d'analyse des sentiments consiste en la prédiction de la polarité[166], positive ou négative, d'une opinion. Il s'agit d'une tâche d'apprentissage supervisé où le jeu de données d'entraînement a été préalablement étiqueté.

Plus précisément, nous avons opté pour la tâche de prédiction de la polarité de critiques de film de la base IMDB (Internet Movie Database) [Pang, Lee & Vaithyanathan, 2002]. Le but du modèle est de prédire la polarité (opinion positive ou négative) de critiques de films.

Pour cette tâche et avec ce corpus précis, les performances s'échelonnent de 70% à 85 % pour les algorithmes d'apprentissage classiques jusqu'à plus de 90% pour des réseaux de neurones profonds finement ajustés [Brownlee, 2018b]. Rappelons, que l'objectif n'est pas de montrer que les différentes techniques d'amplification de données textuelles expérimentées donnent des modèles supérieurs aux meilleures approches, mais plus modestement de démontrer que l'ADT peut faciliter l'apprentissage.

Il est donc essentiel de valider la faisabilité des différentes techniques d'ADT proposées. D'abord pour lever les doutes soulevés par l'article «Text Understanding from Scratch» [Zhang & LeCun, 2015] quant à la faisabilité de la génération automatique de paraphrases en dehors du contexte restreint de la substitution lexicale. Aussi, nos premières expériences avec des questions ouvertes, d'abord en français puis en anglais, n'étaient pas concluantes. Parfois, les résultats étaient améliorés, parfois ils étaient pires. En gros, il y avait un risque que les paraphrases générées par amplification textuelle introduisent davantage de bruit que de signaux utiles.

[165] En anglais: «sentiment analysis»
[166] En anglais: «polarity»

4.2 Données - Critiques de films IMDB

Les données traitées proviennent de la base de critiques de film IMDB (Internet Movie Database) que l'on peut qualifier de norme car largement utilisée dans la communauté scientifique pour la prédiction de la polarité d'opinions. De plus, la tâche de prédiction des sentiments[167] est un exemple bien connu d'utilisation des réseaux de neurones profonds.

Plus spécifiquement nous avons utilisé le jeu de données «polarity dataset v2.0» qui comporte 1000 critiques positives et 1000 critiques négatives extraites de la base de données IMDB. Ce jeu de données a été rendu disponible en juin 2004 par [Pang & Lee 2004]. Le fichier a été téléchargé sur la Toile [Cornel, 2004].

4.2.1 Description des données

Une description du corpus de critiques de films IMDB se trouve à l'annexe 12.

4.3 Protocole expérimental - évaluer les techniques d'ADT

L'expérimentation consiste à entraîner différentes architectures de réseau profond de neurones sur les données textuelles originales (sans amplification) puis sur les données textuelles amplifiées en utilisant différentes techniques d'amplification textuelles.

L'expérimentation comporte deux (2) phases: 1) une phase de prétraitement où on réalise l'amplification des données textuelles selon différentes techniques 2) la phase d'entraînement des modèles avec différentes architectures de réseaux de neurones profonds sur les données originales non amplifiées et sur les données amplifiées selon différentes techniques d'ADT.

Les modèles sont ensuite comparés pour voir s'il y a amélioration ou dégradation des performances de prédiction (exactitude) des modèles sur des données de test. Les erreurs d'entraînement et les mesures F1[168] ont également été calculées.

Notre jeu de données de test n'a que 200 exemples (observations) soit 10 % des données initiales (2000). Cela apparaît faible en considérant que la recommandation générale est plutôt de 20% [Géron 2017a] ou 30 % [Ng, 2017]. En fait, la motivation de ce choix était pour se comparer plus facilement avec d'autres auteurs qui ont utilisés les mêmes données de test [Brownlee, 2018b].

---

[167] En anglais: «sentiment prediction»
[168] En anglais: «F1 score». La mesure F1 est une mesure de performance des classificateurs qui combine la précision et le rappel. Aussi en français: «moyenne harmonique».

Compte-tenu de nos capacités de calcul, nos expériences ont été restreinte à la seule preuve de concept. C'est-à-dire, montrer que les modèles profonds entraînés avec des données textuelles amplifiées donnent des résultats meilleurs et statistiquement significatifs par rapport aux mêmes modèles entraînés avec des données non amplifiées.

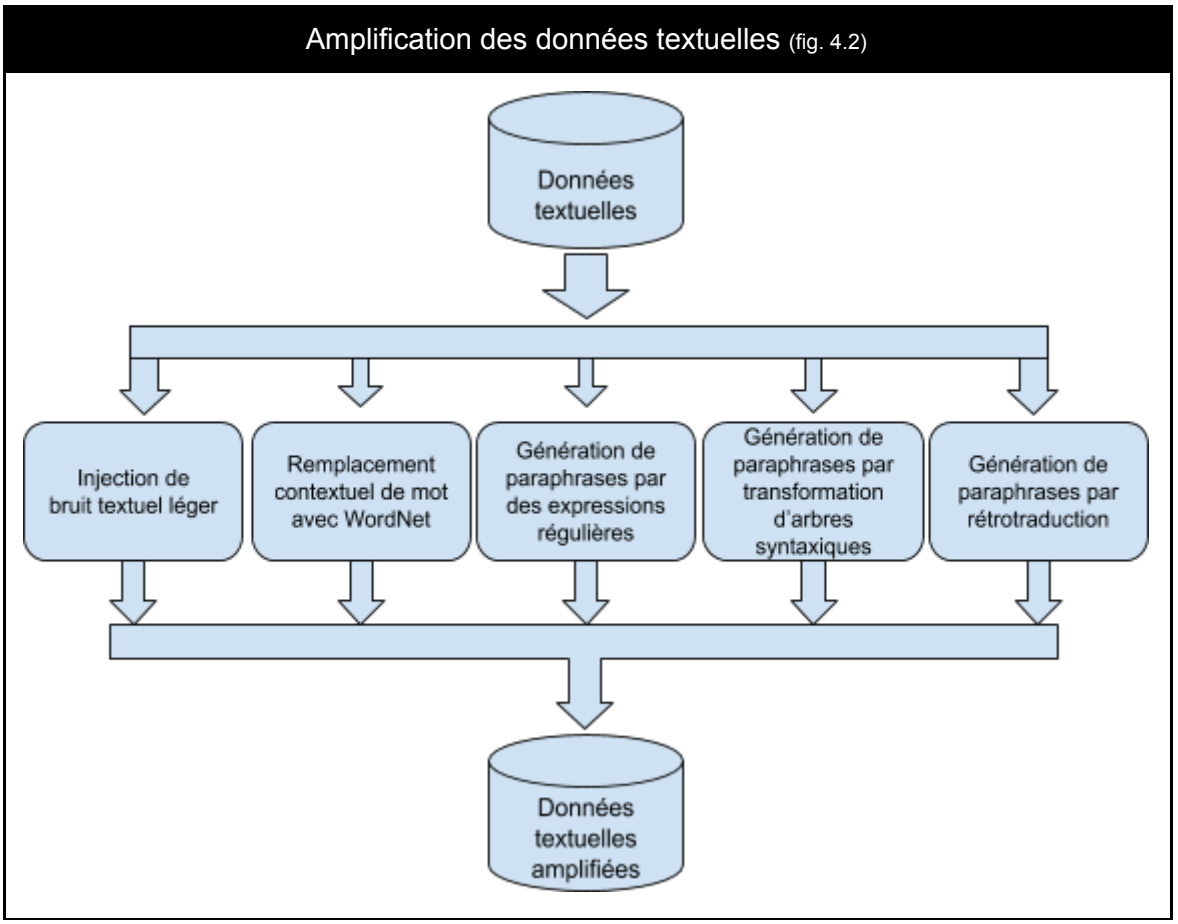
4.3.1 Processus d'amplification des données textuelles

Au début de nos expériences, la génération des paraphrases devait se faire exclusivement à partir de la transformation d'arbres syntaxiques. Après les premières expériences, nous avons convenu de séparer les paraphrases en deux types, celles que l'on pouvait obtenir par de transformations de surface au moyen d'expressions régulières et celles qui nécessitaient des transformations plus profondes et qui se baseraient sur la transformation d'arbres syntaxiques et la rétrotraduction.

Toutes les techniques d'ADT étudiées modifient les textes phrase par phrase. Nous allons créer un objet paratexte capable de générer au besoin le nombre désiré de variantes du texte original.

Il est important de noter que pour ces expériences, qui ne visaient qu'à montrer la faisabilité, nous nous sommes limité à la génération de 5 textes par textes originaux, ce qui représente dans bien des cas qu'une petite fraction de toutes les combinaisons possibles.

Amplification des données textuelles (fig. 4.2)



4.3.2 Injection de «bruit textuel»

L'injection de bruit textuel consiste à retirer, remplacer ou ajouter au hasard un caractère alphabétique (appartenant à l'ensemble string.ascii_lowercase) ou à une table d'équivalences préétablies. Deux tables d'équivalences ont été établies soient les erreurs les plus fréquentes en reconnaissance optique de caractères (ROC)[169] et les erreurs de frappe au clavier dues à la proximité des touches.

L'algorithme d'injection de bruit textuel procède en trois étapes. Il débute par le choix au hasard d'un caractère à remplacer dans le texte. Puis il procède à un deuxième tirage aléatoire pour choisir la transformation à effectuer dans une liste d'opérateurs qui sont: le retrait, le remplacement, l'ajout ou l'équivalence. On peut jouer sur la fréquence d'utilisation d'un opérateur en dupliquant au besoin des entrées dans la liste d'opérateurs. Ce processus est équivalent à constituer une distribution aléatoire de l'utilisation des différents opérateurs. Une fois l'opérateur déterminé, l'algorithme procède à un troisième choix aléatoire pour déterminer le caractère ou la séquence de remplacement [Roquette, 2018].

Injection de bruit textuel - exemples (fig. 4.3)

```
input_sentence = "A man eats a red apple in the kitchen."
injector = NoiseInjector()
injector.inject_noise(input_sentence)
=>
["A man eats a red apple in the kitbhen.", "A man ets a red apple in the
kitchen.", "A man eaits a red appae in the kitchen.", "A man heats a red
apple in the kitchen."]
```

La quantité de bruit à ajouter aux données amplifiées a été déterminé sur la base de l'appréciation de l'expérimentateur par essais et erreurs tout en restant vraisemblable.

4.3.3 Injection de fautes d'orthographe

L'injection de fautes d'orthographe est analogue à l'injection de bruit textuel.

Injection de fautes d'orthographe - exemples (fig. 4.4)

```
input_sentence = "A man eats a red apple in the kitchen."
injector = MispellingInjector()
injector.inject_mispellings(input_sentence)
=>
['A man ets a red apple in the kitchen.', 'A mman ats a red apple in the
kitchen.', 'A man eats a red aple in the kitchan.', 'A man ate a red apple
in the kitchen.', 'A man eats a rd apple in the kitchen.', 'A man eatss a
red apple in the kitchen.', 'A man eat a rad apple in the kitchen.']
```

[169] En anglais: «optical character recognition», «OCR»

L'implémentation de l'algorithme d'injection de fautes d'orthographe a été réalisée avec des expressions régulières mais aurait pu être faite en ajoutant une table d'équivalences à l'algorithme d'injection de bruit textuel.

Dans une étude plus détaillée, il serait intéressant d'étudier la performance des modèles en fonction de la quantité de bruit ou de fautes d'orthographe ajoutée.

4.3.4 Amplification textuelle par substitution lexicale

L'amplification textuelle par substitution lexicale consiste simplement à mettre un mot à la place d'un autre. On parle aussi de «substitution lexicale» ou encore de «remplacement lexical».

Deux techniques d'amplification textuelle par substitution lexicale ont été implémentées. Une première se base sur la ressource lexicale WordNet et une seconde sur des vecteurs-mots regroupés par sens produit par l'algorithme AdaGram. Elles seront davantage expliquées dans les sections suivantes.

4.3.4.1 Substitution lexicale avec le dictionnaire WordNet

Le traitement de chaque texte procède phrase par phrase. La première étape est une analyse lexicale qui filtre les résultats de l'analyse de SyntaxNet pour ne retenir que les mots et leurs étiquettes lexicales[170].

L'algorithme de substitution lexicale ne s'intéresse qu'aux synonymes des noms, adverbes et adjectifs (étiquettes lexicales 'a', 'n' et 'r'). Les fonctions `get_synonyms` et `get_all_synonyms` qui étend la recherche aux synonymes des synonymes interrogent l'IPA NLTK de WordNet avec le mot et son étiquette lexicale [Bird, Klein & Loper, 2009].

Un dictionnaire de synonymes comme WordNet associe à chaque entrée de dictionnaire une liste contenant plusieurs ensembles de synonymes ou ensyns[171]. Chaque ensyn correspond à un sens particulier du mot en entrée. Le nombre de sens est variable et dépend de la qualité des annotations.

Génération de synonymes - exemple (fig. 4.5)

```
get_synonyms('dog','n')
=>
['domestic_dog', 'canis familiaris', 'frump', 'cad', 'bounder', 'blackguard',
'hound', 'heel', 'frank', 'frankfurter', 'hotdog', 'hot dog', 'wiener',
'wienerwurst', 'weenie', 'pawl', 'detent', 'click', 'andiron', 'firedog',
'dog-iron']
```

[170] En anglais: «part of speech tag», «POS tag»
[171] En anglais: «synonyms set», «synset»

```
get_all_synonyms('dog','n')
=>
['hound_dog', 'hound', 'detent', 'chink', 'frankfurter', 'hotdog', 'pawl',
'frank', 'mouse click', 'domestic_dog', 'cad', 'click', 'computer-aided
design', 'canis familiaris', 'bounder', 'clink', 'hot dog', 'andiron',
'leaper', 'norbert wiener', 'frump', 'suction_stop', 'weenie', 'dog-iron',
'wienerwurst', 'heel', 'wiener', 'red hot', 'blackguard', 'firedog']
```

Les différents sens du mot «dog» sont ci-haut retournés pêle-mêle. On peut demander à WordNet de les retourner regroupés par ensembles de synonymes ou ensyns.

Si la liste de synonymes comporte moins qu'un nombre prédéterminé de synonymes (fixé à 5), notre algorithme interroge WordNet pour élargir la recherche de synonymes, puis si le nombre de synonymes est toujours inférieur à ce seuil, l'algorithme appelle la fonction `get_hypernyms` codée à partir de la fonction `hypernyms()` de l'IPA NLTK WordNet afin d'obtenir une liste d'hyperonymes (mots plus généraux) [Bird, Klein & Loper, 2009].

Génération d'hyperonymes - exemple (fig. 4.6)

```
get_hypernyms_lemmas(wn.synset('dog.n.01'),'dog','n')
```

retourne 18 candidats =>

```
['canine', 'canid', 'domestic animal', 'domesticated animal', 'carnivore',
'animal', 'beast', 'brute', 'fauna', 'placental', 'placental mammal',
'eutherian', 'eutherian mammal', 'mammal', 'mammalian', 'vertebrate',
'craniate', 'chordate']
```

À cette étape, le défi consiste à choisir le bon ensemble de synonymes ou ensyn. Plusieurs stratégies peuvent être utilisées. On peut choisir le sens le plus fréquent en fonction du nombre d'occurrences dans un corpus de référence. On peut utiliser les informations qui accompagnent chaque ensyn dans le dictionnaire WordNet comme des définitions, des exemples, etc. L'algorithme calcule une mesure de similarité entre les informations associées à chaque ensyn et le contexte du mot (typiquement la phrase d'où il provient). Enfin, l'algorithme choisit l'ensyn qui a l'information la plus similaire au contexte du mot.

Nous avons opté pour cette stratégie avec la fonction `get_synonyms_in_context` qui reçoit en entrée le mot, son étiquette lexicale, un contexte local (typiquement la phrase d'origine) et un contexte élargi tiré du texte au complet. La fonction `get_synonyms_in_context` extrait les mots des définitions et les exemples associés à chaque ensyn dans WordNet (fonctions `syn.definition()` et `syn.examples()`. Puis `get_synonyms_in_context` calcule la similarité cosinus entre les informations contenues dans WordNet et les contextes fournis [Manning & Schütze, 1999]. Finalement, la fonction retourne une liste triée selon le degré de similarité.

Génération de synonymes en contexte - exemple (fig. 4.7)

```
sentence_context = "The small dog is eating meat."
text_context = "The domestic dog is a member of the genus Canis (canines), which forms
part of the wolf-like canids, and is the most widely abundant terrestrial carnivore."

get_synonyms_in_context('dog','n',local_context=sentence_context,extended_context=text_c
ontext)
=>
[('dog.n.01', 0.24056226923462665, ['canine', 'canid', 'domestic animal', 'domesticated
animal', 'carnivore', 'animal', 'beast', 'brute', 'fauna', 'placental', 'placental
mammal', 'eutherian', 'eutherian mammal', 'mammal', 'mammalian', 'vertebrate',
'craniate', 'chordate', 'domestic dog', 'canis familiaris']),
('frank.n.02', 0.15249857033260467, ['sausage', 'meat', 'food', 'solid food', 'solid',
'frank', 'frankfurter', 'hotdog', 'hot dog', 'wiener', 'wienerwurst', 'weenie']),
('cad.n.01', 0.11179032745879261, ['villain', 'scoundrel', 'unwelcome person', 'persona
non grata', 'person', 'individual', 'someone', 'somebody', 'cad', 'bounder',
'blackguard', 'hound', 'heel']),
('frump.n.01', 0.10414662708093994, ['unpleasant woman', 'disagreeable woman',
'unpleasant person', 'disagreeable person', 'unwelcome person', 'persona non grata',
'person', 'individual', 'someone', 'somebody', 'frump']),
('dog.n.03', 0.10042589625609066, ['fellow', 'feller', 'fella', 'lad', 'gent',
'blighter', 'bloke', 'male', 'male person', 'person', 'individual', 'someone',
'somebody']), ('pawl.n.01', 0.0, ['restraint', 'constraint', 'device',
'instrumentality', 'instrumentation', 'pawl', 'detent', 'click']),
('andiron.n.01', 0.0, ['device', 'instrumentality', 'instrumentation', 'andiron',
'firedog', 'dog-iron'])]
```

Il arrive que les dictionnaires de synonymes à la WordNet retournent des antonymes lorsque l'on prolonge la chaîne de synonymie (i.e. les synonymes de synonymes). La dernière étape de la recherche de synonymes consiste alors à filtrer la liste des candidats avec l'IPA des antonymes NLTK WordNet pour en retirer les antonymes [Coulombe, 2017d].

Pour chaque phrase l'algorithme produit un objet paraphrase qui recense pour chaque mot, tous les synonymes retournés par `get_synonyms_in_context`. Les différents objets paraphrases associés à chaque phrase d'un texte serviront à produire un certain nombre de variantes qui seront rassemblées dans un objet paratexte. Finalement l'objet paratexte génèrera le nombre de variantes du texte original que l'on désire.

4.3.4.2 Substitution lexicale avec des vecteurs-mots AdaGram

Le code original d'AdaGram écrit en langage Julia est disponible sur GitHub [Bartunov, 2015], [Coulombe, 2017a], [Coulombe, 2017c]. Il existe également une version partielle en Python [Lopuhin, 2017], [Coulombe, 2017b].

L'exécution de l'algorithme AdaGram encapsulé en Python est très lent. Il faut compter environ vingt minutes par texte sur un ordinateur portable donc plusieurs jours pour le corpus au complet.

Le traitement par lots a été divisé en trois phases. Dans la première phase qui est rapide, l'algorithme procède à l'analyse lexicale de chacun des textes, phrase par phrase, à la recherche de mots candidats pour la substitution. Il en résulte un fichier de mots-candidats pour la substitution lexicale. Chaque ligne du fichier comporte le numéro de la phrase, la phrase, le mot proprement dit, la position du mot dans la phrase, son étiquette lexicale.

La deuxième phase du traitement qui implique AdaGram est la plus longue. Le traitement consiste à invoquer un vecteur-mot AdaGram afin d'obtenir les différents ensyns (ensemble de synonymes) qui correspondent à un mot et son contexte. Le mot et son contexte, qui est la phrase dans laquelle le mot se trouve, sont lus dans le fichier produit à la phase précédente. Pour éviter les problèmes avec les mots hors vocabulaire[172], les mots du contexte sont filtrés. Ensuite, une recherche des plus-proches-voisins retourne les ensyns les plus prometteurs. Le résultat est versé dans un autre fichier qui contient le numéro de phrase, le mot original, sa position, son étiquette lexicale, le degré de polysémie du mot, une probabilité de substitution et une liste de tous les mots de substitution.

La troisième phase utilise le fichier des mots de substitutions d'AdaGram pour produire les paraphrases et un objet paratexte en reprenant l'algorithme de substitution lexicale vu précédemment, mais en remplaçant WordNet par une table de substitutions issue d'AdaGram.

Substitutions lexicales AdaGram - exemples (fig. 4.8)

```
# Chargement du modèle
vm, dict = load_model(MODEL_PATH)
# Chercher les vecteurs_mots qui correspondent au mot et son contexte
vecteurs_mots = disambiguate(vm, dict, mot, contexte)
# Trier les vecteurs-mots selon leur similarité avec le contexte
index_max = sortperm(results,rev=true)[1]
# Retourner des synonymes dans le voisinage du mot
knn = nearest_neighbors(vm,dict,word,index_max,size_synset)
# Afficher chaque synonyme qui a une probabilité plus grande qu'un seuil
for syn in knn:
    if (!(syn[1]==word)) && (syn[3] >= min_prob)
        println("* Mot: ", mot, " * Syn: ", syn[1], " * Prob: ", syn[3])
```

```
=>
* Mot:  party      * Syn: dinner          * Prob: 0.730929
* Mot:  party      * Syn: wedding         * Prob: 0.721681
* Mot:  girlfriend * Syn: wife            * Prob: 0.823606
* Mot:  girlfriend * Syn: fiancée         * Prob: 0.783926
* Mot:  life       * Syn: father          * Prob: 0.735282
* Mot:  nightmares * Syn: hallucinations  * Prob: 0.727164
...
```

[172] En anglais: «Out-of-Vocabulary», «OOV»

À la fin, l'objet paratexte créé avec les phrases contenant des substitutions proposées par AdaGram génère le nombre de variantes du texte original que l'on désire.

Le plus gros obstacle technique était le code d'AdaGram écrit en Julia. Nous avons opté pour une interface Python qui "enrobe" simplement un petit code Julia qui interroge l'IPA d'AdaGram, mais cela au prix d'une performance dégradée [Coulombe, 2017c].

4.3.5 Transformation textuelle de surface au moyen d'expressions régulières

La transformation textuelle de surface procède également phrase par phrase. Le traitement est rapide. Il consiste à appliquer séquentiellement dans une boucle un ensemble de règles de transformation, écrites selon le formalisme des expressions régulières, à chacune des phrases du texte original.

Chaque expression régulière s'applique directement par reconnaissance de formes[173] sur la chaîne de texte de la phrase originale. L'expression régilière retourne alors une chaîne de texte qui correspond à une paraphrase. Les résultats de chaque transformation sont accumulés au fur et à mesure et sont retournés dans une liste de paraphrases (ou variantes) et ceci pour chacune des phrases du texte original.

Cette liste de paraphrases sert ensuite à construire un objet paratexte qui lui-même sera utilisé pour générer le nombre de variantes du texte original que l'on désire obtenir.

Transformations de surface par expressions régulières - exemples (fig. 4.8)

```
replacer = RegexpReplacer()
input_sentence = "They will not play hockey since they can't skate."
replacer.replace(input_sentence)
=>
["They'll not play hockey since they can't skate.", "They won't play hockey
since they can't skate.", 'They will not play hockey since they can not
skate.', 'They will not play hockey since they cannot skate.', "They shall not
play hockey since they can't skate."]
```

4.3.6 Transformation textuelle profonde par manipulation d'arbres syntaxiques

Une transformation textuelle profonde s'apparente à une transformation textuelle de surface à la différence importante que les règles de transformations s'appliquent non pas à du texte brut mais à une structure qui résulte de l'analyse morphosyntaxique de la phrase selon le formalisme des grammaires de dépendance.

[173] En anglais: «pattern matching», «pattern recognition»

Le traitement procède phrase par phrase. Chaque phrase est soumise à l'analyse du logiciel SyntaxNet, un analyseur syntaxique à large couverture en code libre de Google, qui utilise des techniques d'apprentissage profond et la bibliothèque TensorFlow [Petrov, 2016], [Kong et al, 2017]. Pour être efficace, le code est exécuté dans l'infrastructure infonuagique de Google Cloud en utilisant le service web Cloud Natural Language [Google, 2018a].

La structure syntaxique de dépendance retournée par SyntaxNet est ensuite manipulée avec des règles de transformations écrites en Python selon un formalisme développé par Michel Gagnon de Polytechnique Montréal [Gagnon & Da Sylva, 2005], [Zouaq, Gagnon & Ozell, 2010] dans l'outil PolyPhrases. La structure transformée sert à produire un texte qui correspond à une transformation profonde de la phrase d'origine.

Les résultats de chaque transformation sont accumulés au fur et à mesure et sont retournés dans une liste de paraphrases (ou variantes) et ceci pour chacune des phrases du texte original. Ensuite, l'algorithme construit un objet paratexte à partir de la liste des paraphrases. Finalement, le paratexte est utilisé pour générer le nombre de variantes du texte original que l'on désire obtenir.

Transformations par manipulation d'arbres - exemples (fig. 4.9)

```
input_sentence = "A man eats a red apple in the kitchen."
paraphrase.convert(input_sentence)

=>

["A man eats an apple in the kitchen.", "A red apple is eaten by a man in
the kitchen.", "He eats a red apple in the kitchen.", "A man eats it in the
kitchen."]
```

4.3.7 Transformation textuelle par rétrotraduction

Comme toutes les autres transformations, la transformation par rétrotraduction procède phrase par phrase.

Chaque phrase originale en anglais est traduite une première fois en plusieurs langues cibles en invoquant l'IPA Google Translate. Une seconde traduction est demandée à Google Translate pour chaque traduction produite mais cette fois vers l'anglais, d'où le concept de rétrotraduction. Ces phrases «rétrotraduites» sont des paraphrases potentielles qui sont filtrées selon différents mécanismes pour être retenues ou éliminées.

Un premier mécanisme de filtrage consiste à ne conserver que les paraphrases non-identiques qui diffèrent en longueur de moins de 25%. Rappelons que Google Translate, le service de traduction en ligne de Google, utilise la traduction neuronale.

Transformations textuelles par rétrotraduction - exemples (fig. 4.10)

```
input_sentence = "A man eats a red apple in the kitchen."
paraphrase.convert(input_sentence)

=>

["Man eating a red apple in the kitchen.", "One ate a red apple in the
kitchen.", "A man is eating a red apple in the kitchen.", "The man eats a
red apple in the kitchen.", "In the kitchen , a man eats a red apple."]
```

4.4 Combinaisons de techniques d'ADT testées

En plus de tester chaque technique d'ADT individuellement, nous avons également entraîné pour quelques combinaisons de techniques car il était impossible de toutes les essayer en raison de nos ressources de calcul[174]. Nous avons choisi parmi les 128 combinaisons possibles des 7 techniques d'amplification (2^7 = 128) quelques-unes qui nous semblaient les plus informatives. Par exemple: la combinaison 1 + 2 (injection de bruit et de fautes d'orthographe), la combinaison 3 + 4 (les substitutions lexicales), 1 + 2 + 5 (les techniques de surface: bruit, orthographe et expressions régulières), la combinaison 3 + 4 + 6 + 7 (les techniques profondes: substitutions lexicales, transformations d'arbres et rétrotraduction), et enfin toutes les combinaisons, pour observer comment elles pouvaient s'additionner.

4.5 Architectures de réseau de neurones testées

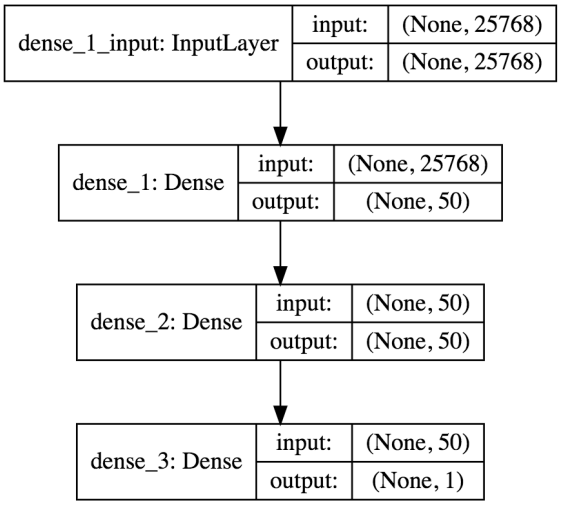
Les architectures de réseau profond de neurones testées recouvrent les trois types les plus couramment utilisés: le perceptron multicouche[175], le réseau convolutif[176] et le réseau récurrent[177].

Plus précisément, nous avons expérimenté avec le perceptron multicouche à deux couches cachées (PM2C)[178] le réseau convolutif 1D[179], le réseau récurrent à longue mémoire court terme (LMCT)[180], le réseau récurrent à longue mémoire court terme bidirectionnel (biLMCT)[181]. Notez que nous utilisons un réseau convolutif 1D car les textes sont invariants selon une seule dimension, leur séquence.

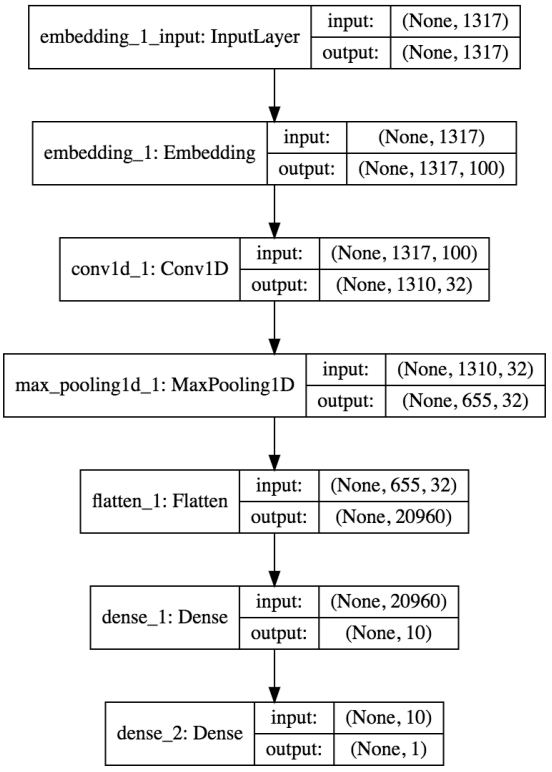
---

[174] Note: L'entraînement des 128 combinaisons prend 8 jours pour le perceptron.
[175] En anglais: «multilayer perceptron», «MLP»
[176] En anglais: «convolutional neural network», «CNN», «convolutional network», «ConvNet». Aussi en français, «réseau de neurones convolutif», «réseau de neurones à convolutions», «réseau à convolutions». Note: c'est le réseau qui est convolutif pas les neurones.
[177] En anglais: «recurrent neural network», «RNN». En français, on dit aussi «réseau de neurones récurrent», «réseau neuronal récurrent»,. C'est le réseau qui est récurrent pas les neurones.
[178] En anglais: «two hidden layers», «MLP»
[179] En anglais: «convolutional network 1D», «CNN 1D»
[180] En anglais: «long short-term memory», «LSTM»
[181] En anglais: «bidirectional LSTM», «biLSTM»

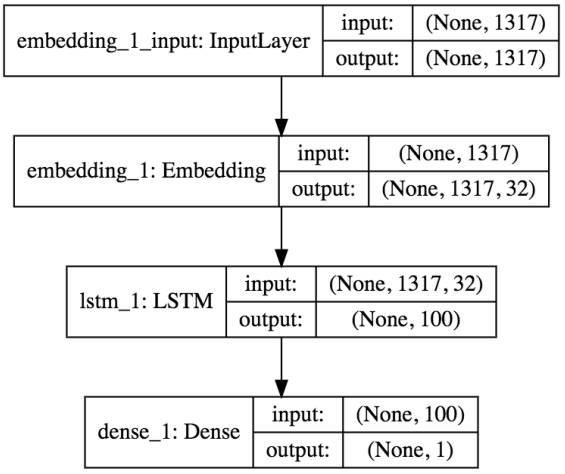
Architectures de réseaux de neurones testées (fig. 4.11)



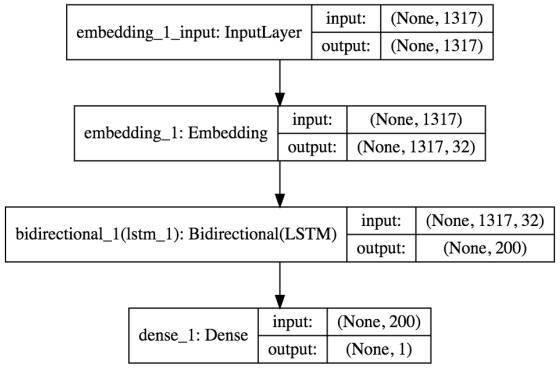
Perceptron multicouche
à deux couches cachées (PM2C)



Réseau convolutif 1D



Réseau récurrent à
longue mémoire court terme (LMCT)



Réseau récurrent à longue mémoire court terme bidirectionnel (biLMCT)

Pour les réseaux convolutifs, nous aurions pu empiler plusieurs couches convolutives 1D pour construire des réseaux plus profonds, mais pour les fins d'une preuve de concept, nous avons opté pour le réseau le plus simple et le plus rapide à entraîner. Rappelons que les couches de convolution 1D sont invariantes en translation dans le sens où parce que la même transformation d'entrée est effectuée sur chaque fenêtre de traitement, une forme ou

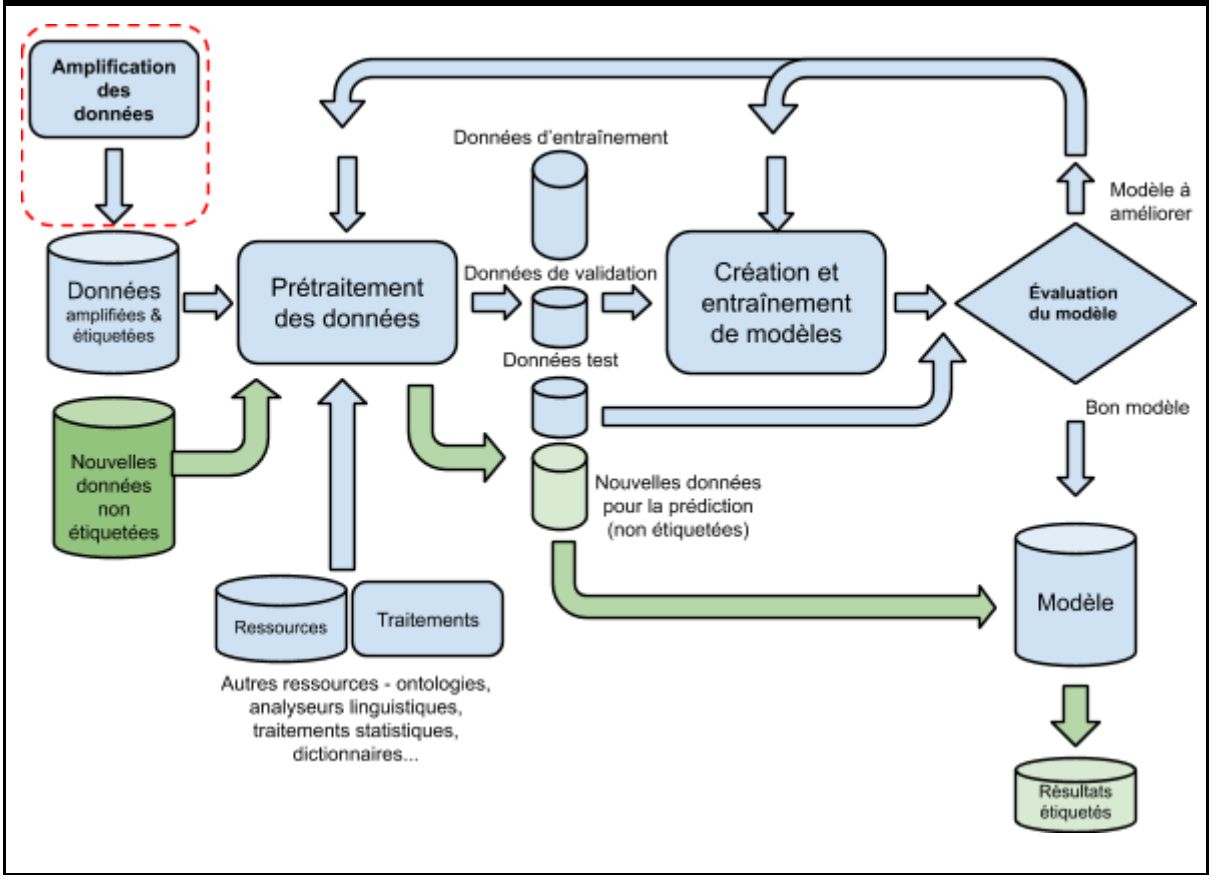
motif appris à une certaine position dans une phrase peut être reconnu ultérieurement à une position différente. La convolution sert à extraire des attributs à partir d'une série de filtres ou noyaux dont les paramètres sont appris par rétropropagation. En travaillant au niveau du caractère, un réseau convolutif 1D peut apprendre la morphologie [Cholet, 2017]. D'autres avantages des convolutions sont le partage des paramètres et la réduction du nombre de connexions.

Nos réseaux neuronaux sont régularisés uniquement par l'ajout de données. Ces architectures ne comportent donc pas d'enrichissement par vecteurs-mots externes, pas de régularisation L_1 ou L_2 et pas d'extinction de neurones[182].

4.6 Chaîne de traitement

Ci-dessous un diagramme qui illustre le processus itératif d'entraînement et de sélection de modèles statistiques à partir de données. Plus précisément, il s'agit de la chaîne de traitement[183] d'une tâche d'apprentissage supervisé.

Chaîne de traitement d'une tâche d'apprentissage supervisé (fig. 4.11)



[182] En anglais: «dropout»
[183] en anglais: «data pipeline»

La première étape est le prétraitement des données qui se divise lui-même en deux parties: 1) D'abord l'amplification des données réalisée au préalable (zone en pointillé rouge de la fig. 4.11) 2) Les opérations de prétraitement usuelles où les données sont lues, transformées, normalisées, filtrées et formatées en vecteurs de nombres réels qui sont fournis en entrée à l'algorithme qui entraîne un modèle.

La deuxième étape, le cycle de mise au point d'un modèle. Les données sont d'abord partagées en trois ensembles: le jeu de données d'entraînement, le jeu de données de validation et le jeu de données de test. Pour construire un modèle d'apprentissage automatique performant, il faut autant que possible entraîner le modèle, le valider et le tester avec des données provenant de la même distribution statistique. En effet, l'algorithme d'apprentissage se base essentiellement sur les régularités statistiques dans les données, si les distributions statistiques des jeux de données sont trop différentes, l'algorithme sera perdu. Pour éviter ce problème et lorsque cela est possible, toutes nos données proviennent du même jeu de données initial. En principe, cela nous assure que toutes les données ont la même distribution statistique [Ng, 2017].

Sinon, les différences statistiques entre les données d'entraînement, les données de validation et les données de test peuvent réserver des surprises. Faire fonctionner un modèle sur des données ayant une distribution statistique différente de la distribution des données avec lesquelles il a été entraîné est possible, mais cela représente un défi[184].

Les données de validation, parfois appelées données de validation croisée[185] ou données de mise-au-point, servent au réglage des hyperparamètres, au contrôle du déroulement du processus d'entraînement, mais surtout au choix du meilleur modèle.

Le jeu de données de test, qui contient des données fraîches jamais utilisées pour entraîner le modèle, vient mesurer la performance finale du modèle. Il est important de ne prendre aucune décision en se basant sur les données de test, que ce soit au niveau du prétraitement des données ou de l'algorithme d'apprentissage. L'objectif est d'éviter la contamination du processus par les données de test[186].

Les données de test serviront à évaluer la performance finale du modèle une fois celui-ci entraîné. Le taux d'erreur sur les données de test correspond à l'erreur de généralisation du modèle qui mesure sa capacité de prédiction sur de nouvelles données.

Afin de ne pas nous éparpiller, nous avons opté pour une seule métrique, soit l'exactitude, pour mesurer la performance de nos algorithmes. Rappelons que l'exactitude est le pourcentage de bonnes prédictions sur les données de test.

---

[184] Note: On peut toujours s'arranger si les données de validation et de test ont la même distribution.
[185] En anglais: «cross-validation», «CV»
[186] En anglais: «data snooping»

Le schéma de la chaîne de traitement (fig. 4.11) montre bien qu'il s'agit d'un processus itératif. On part d'une hypothèse qu'on l'implémente d'abord au niveau du prétraitement des données. Parfois on va jusqu'à chercher de nouvelles sources de données. À l'étape suivante, on extrait, transforme et normalise les attributs. Puis on entraîne un modèle. Les résultats et la mesure de la performance du modèle permettent de comprendre la validité de notre hypothèse et de savoir si l'on se dirige dans la bonne direction. Le cas échéant on retourne en arrière pour essayer une nouvelle hypothèse. À la fin, une fois les paramètres du modèle déterminés, il est d'un bon usage d'entraîner le modèle final avec toutes les données disponibles.

4.7 Entraînement des modèles

Les architectures de réseau neuronal profond testées couvrent les trois types les plus couramment utilisés, à savoir les perceptrons multicouches (PMC), les réseaux convolutifs et les réseaux récurrents. Précisément, nous avons expérimenté un perceptron multicouche à deux (2) couches cachées, un réseau convolutif à une dimension (CNN 1D), un réseau récurrent à longue mémoire court terme (LMCT), un LMCT bidirectionnel (biLMCT).

Les réseaux neuronaux sont régularisés uniquement en ajoutant plus de données et par arrêt précoce. Donc pas d'enrichissement par des vecteurs-mots externes, pas de régularisation L_1, L_2, pas d'extinction de neurones et aucun ajustement d'hyperparamètres.

Le fait d'entraîner les modèles sur les données originales sans amplification et de comparer ces résultats avec les mêmes modèles entraînés sur des données amplifiées cache potentiellement des biais. Par exemple, dans le cas des données amplifiées, on compare un modèle qui passe n_epochs fois dans l'ensemble des données avec des modèles qui passent n_epochs fois mais avec des données amplifiées K fois[187], de ce qui reviendrait à passer K*n_epochs fois dans les données originales. Or, comparativement on ne passe que n_epochs fois dans les données originales.

Pour corriger cet éventuel biais, tous les modèles sont entraînés avec un nombre d'itérations suffisant (n_epochs = 100) pour que l'algorithme s'arrête soit par arrêt précoce ou parce qu'il a atteint un plateau. En fait, l'algorithme ne se rend jamais à la limite des 100 itérations. En principe, poursuivre les itérations ne donnerait rien et cela pourrait même dégrader les résultats. Une autre approche aurait été de recopier les données non-amplifiées K fois.

Dans la pratique, il est souvent recommandé d'utiliser une technique de régularisation comme la régularisation L_2 et d'entraîner le réseau de neurones aussi longtemps que possible mais cela demande d'essayer un grand nombre de valeurs différentes pour l'hyperparamètre λ qui gouverne la régularisation L_2 [Ng, 2017].

[187] Note: K est le facteur d'amplification

Pour bien isoler l'effet de l'ajout de données textuelles amplifiées, nous avons délibéré choisi de ne pas faire de régularisation de type L_1 ou L_2.

Pour tenir compte de la variabilité des résultats et faciliter un test de signification statistique [188], nous avons calculé une moyenne sur plusieurs exécutions (20) et effectué de la validation croisée avec 10 plis[189]. Cette variabilité a probablement ralenti les calculs. Pour une discussion des effets des données corrélées sur la variance voir l'annexe 11.

Finalement, nous avons retenu les résultats où concordaient les calculs de la signification statistique pour la validation croisée 10 plis et la répétition pour 20 exécutions.

Nous avons également implémenté différents mécanismes comme des procédures d'arrêt précoce et la diminution du pas du gradient[190] disponible dans l'outil Keras [Coulombe, 2018b].

Pour fins de comparaison, nous avons également entraîné un modèle avec un algorithme classique. L'algorithme XGBoost s'est naturellement imposé. Rappelons que XGBoost est une implémentation très performante d'un algorithme de régression et de classification de type ensembliste qui fait partie de la famille des arbres à dopage de gradient[191]. XGBoost utilise le principe du dopage[192] pour renforcer certains arbres en utilisant une fonction de coût optimisée par descente de gradient [Chen & Guestrin, 2016].

4.8 Outils

Pour nos expériences, nous avons essentiellement utilisé des outils Python disponibles en logiciel libre[193] ou en code source ouvert[194].

Développée à l'INRIA en France, Scikit-Learn est une bibliothèque spécialisée en apprentissage statistique classique qui se marie bien avec l'écosystème Python dont NumPy et SciPy [Pedregosa et al,, 2011]. Pandas a été utilisé pour le prétraitement et la manipulation des données [McKinney, 2010] et Matplotlib pour l'affichage de graphiques.

Keras est une bibliothèque Python de haut niveau pour l'apprentissage profond bâtie au dessus de TensorFlow, Theano ou CNTK. Keras est publié en code source ouvert sous la licence MIT. Keras est développé par François Chollet, un ingénieur de Google, pour faciliter la création et la mise au point de modèles d'apprentissage profond. Keras fonctionne avec

---

[188] En anglais: «statistical significance»
[189] en anglais: «folds»
[190] En anglais: «learning rate reduction». En français, «gain du gradient» ou «taux d'apprentissage»
[191] En anglais: «gradient boosting tree». Aussi en français: «arbres à renforcement de gradient»
[192] En anglais: «boosting»
[193] En anglais: «free software»
[194] En anglais: «open source code»

Python 2.7.X ou 3.X et s'exécute de manière transparente sur des processeurs graphiques [195] et des processeurs classiques[196] [Chollet, 2015], [Chollet, 2017].

Pour le traitement de la langue naturelle (TLN), nous avons choisi la bibliothèque NLTK (Natural Language ToolKit) en Python [Bird, Klein & Loper, 2009] qui offre une interface de programmation d'applications ou IPA[197] de haut niveau pour Wordnet. Nous avons également utilisé la bibliothèque spaCy qui offre plusieurs IPA de calibre industriel, des vecteurs-mots, des modèles de langues préentraînés et des outils pour l'apprentissage profond [Honnibal & Montani, 2017].

À cela s'ajoute des ressources linguistiques comme le thésaurus WordNet [Miller et al, 1990] ou WOLF (Wordnet Libre du Français) [Sagot & Fišer, 2008][198], des lexiques sémantiques, ou des dictionnaires comme le dictionnaire de synonymes DES (Dictionnaire Électronique des Synonymes) [Ploux & Victorri 1998], les banques de n-grammes de Google [Franz & Thorsten, 2006].

Nous avons utilisé SyntaxNet de Google, un analyseur à large couverture à base de grammaires de dépendance [Petrov, 2016], [Kong et al, 2017]. Disponible en code source ouvert et écrit avec TensorFlow également en code source ouvert, SyntaxNet utilise le formalisme CoNLL (Computational Natural Language Learning) et la norme «Universal dependencies» (UD) [Batista, 2017].

Pour des raisons pratiques, nous avons opté pour l'emploi de services de la plateforme infonuagique Google Cloud Platform [Google, 2018b]. Plus précisément, nous utilisons le service d'analyse syntaxique[199], dérivé de SyntaxNet, de l'IPA de langage naturel (Cloud Natural Language API) [Google, 2018c] et le service de traduction automatique (Cloud Translation API), basé sur Google Translate [Google, 2018d].

[195] En anglais: «graphical processor unit», «GPU»
[196] En anglais: «central processing unit», «CPU»
[197] En anglais «application programming interface», «API». Aussi en français: «interface de programmation applicative», «IPA»
[198] Note; Utilisé uniquement dans des travaux préliminaires
[199] En anglais: «syntax analysis»

Chapitre 5 - Résultats

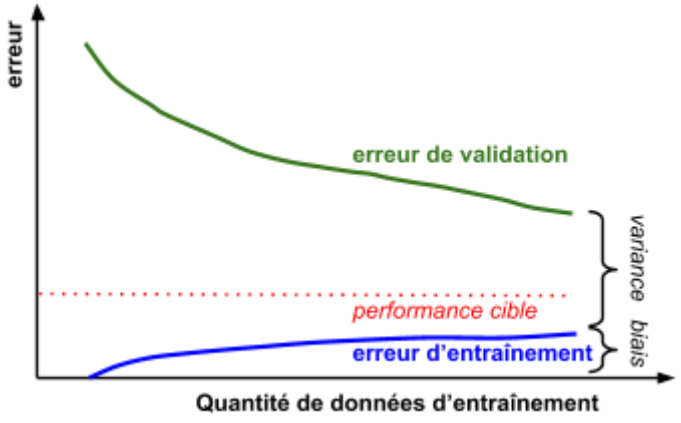
Les expériences mesurent le pourcentage des critiques de film correctement classées, soit l'exactitude[200]. L'exactitude est le critère généralement utilisé pour les tâches de classification supervisée.

La base de référence[201] qui s'impose naturellement est la performance mesurée sur les données originales non amplifiées. Ce seuil varie en fonction des architectures de réseau de neurones. Pour le perceptron multicouche, le seuil de performance à dépasser a été mesuré autour de 89%. L'amplification textuelle a permis d'accroître l'exactitude des résultats dans une fourchette 0.5 à 8.8% sur une tâche normalisée de prédiction de la polarité de textes de critiques de film de la base de données IMDB.

5.1 Davantage de données

En examinant les courbes d'entraînement, nous observons que l'erreur sur les données d'entraînement augmente progressivement avec la quantité de données d'entraînement. Cela s'explique ainsi: plus il y a de données d'entraînement, plus le modèle a de la difficulté à apprendre de nouvelles données car le modèle a une capacité fixe. Au contraire, l'erreur de validation, qui est l'erreur sur de nouvelles données, diminue avec la quantité de données d'entraînement [Ng, 2017].

Test des courbes d'entraînement (fig. 5.1)



En extrapolant la courbe d'erreur de validation (en vert), on voit que l'ajout de données d'entraînement fera diminuer l'erreur.

En fait, nous avons entraîné le même modèle avec des quantités croissantes de données. Si on extrapole la courbe de l'erreur de validation, on voit qu'en ajoutant des données, elle devrait éventuellement croiser la courbe d'erreur d'entraînement.

[200] En anglais: «accuracy»
[201] En anglais: «baseline». Aussi en français, «seuil de référence», «référence»

5.2 Résultats

Exactitude de la prédiction de la polarité (fig. 5.2)					
Technique d'amplification textuelle	**Classique**	**Architecture du réseau de neurones**			
	XGBoost	Perceptron multicouche	Réseau convolutif 1D	Réseau récurrent LMCT	Réseau récurrent biLMCT
0- Aucune base de référence	T:**82.50** E: 94.67 F1: 0.83	T:**88.95±0.80** E: 96.82±0.11 F1: 0.89±0.01	T:**77.58±2.07** E: 92.19±0.57 F1: 0.74±0.03	T:**74.28±4.51** E: 91.52±2.66 F1: 0.69±0.08	T:**72.62±4.51** E: 91.92±2.18 F1: 0.67±0.07
1- Bruit textuel	T:**79.50** E: 94.27 F1: 0.80	T:**90.88±0.44** E:100.00±0.00 F1: 0.91±0.00	T:**85.97±2.02** E:100.00±0.00 F1: 0.86±0.02	T:**81.25±2.49** E:99.96±0.08 F1: 0.81±0.03	T:**80.83±3.10** E: 99.80±0.49 F1: 0.80±0.03
2- Fautes d'orthographe	T:**80.00** E: 94.31 F1: 0.81	T:**90.58±0.36** E:100.00±0.00 F1: 0.90±0.00	T:**84.78±1.36** E:100.00±0.00 F1: 0.85±0.01	T:**81.08±2.74** E: 99.87±0.13 F1: 0.81±0.04	T:**80.97±2.38** E: 99.88±0.12 F1: 0.81±0.04
3- Subst. lex. AdaGram	T:**82.50** E: 94.82 F1: 0.83	T:**91.08±0.33** E: 99.95±0.01 F1: 0.91±0.00	T:**86.38±1.18** E:100.00±0.00 F1: 0.87±0.01	T:**77.80±7.26** E:97.30±10.98 F1: 0.75±0.18	T:**79.35±3.13** E: 99.93±0.08 F1: 0.78±0.04
4- Subst. lex. WordNet	T:**82.00** E: 94.79 F1: 0.82	T:**90.25±0.37** E:100.00±0.00 F1: 0.90±0.00	T:**83.08±2.57** E:100.00±0.00 F1: 0.83±0.02	T:**80.33±2.78** E: 99.84±0.34 F1: 0.80±0.03	T:**78.80±3.01** E: 99.85±0.32 F1: 0.78±0.04
5- Paraphrases exp. rég.	T:**80.50** E: 95.19 F1: 0.81	T:**90.67±0.36** E:100.00±0.00 F1: 0.91±0.00	T:**84.80±1.46** E:100.00±0.00 F1: 0.85±0.01	T:**81.22±2.38** E: 99.95±0.16 F1: 0.81±0.03	T:**80.92±1.75** E: 99.99±0.03 F1: 0.81±0.02
6- Paraphrases arbres syntax.	T:**80.00** E: 93.89 F1: 0.81	T:**90.75±0.43** E:100.00±0.00 F1: 0.91±0.00	T:**85.15±2.17** E:100.00±0.00 F1: 0.85±0.02	T:**81.10±1.48** E: 99.95±0.04 F1: 0.81±0.01	T:**79.88±3.84** E: 99.65±1.05 F1: 0.79±0.04
7- Rétro-traduction	T:**80.50** E: 95.62 F1: 0.82	T:**91.05±0.15** E:100.00±0.00 F1: 0.91±0.00	T:**83.78±1.86** E:100.00±0.00 F1: 0.84±0.02	T:**80.38±2.21** E: 99.93±0.25 F1: 0.80±0.03	T:**79.50±2.14** E: 99.94±0.17 F1: 0.79±0.03
1 + 2	T:**81.00** E: 93.94 F1: 0.82	T:**91.10±0.49** E:100.00±0.00 F1: 0.91±0.01	T:**85.12±2.05** E:100.00±0.01 F1: 0.85±0.02	T:**81.67±2.35** E: 99.80±0.07 F1: 0.81±0.03	T:**79.25±2.36** E: 99.80±0.12 F1: 0.78±0.03
3 + 4	T:**77.50** E: 94.57 F1: 0.78	T:**90.90±0.30** E:100.00±0.00 F1: 0.91±0.00	T:**85.12±2.02** E:100.00±0.00 F1: 0.85±0.02	T:**78.83±4.46** E: 99.83±0.28 F1: 0.78±0.05	T:**79.12±2.77** E: 99.92±0.13 F1: 0.78±0.03
1 + 2 + 5	T:**82.50** E: 94.47 F1: 0.83	T:**91.28±0.29** E:100.00±0.00 F1: 0.91±0.00	T:**84.65±1.73** E:100.00±0.00 F1: 0.85±0.02	T:**79.08±2.93** E: 99.96±0.08 F1: 0.78±0.03	T:**79.85±1.99** E: 99.99±0.01 F1: 0.79±0.03
3 + 4 + 6 + 7	T:**83.00** E: 93.28 F1: 0.83	T:**90.60±0.41** E:100.00±0.00 F1: 0.90±0.00	T:**83.67±2.08** E:100.00±0.00 F1: 0.84±0.02	T:**77.35±3.34** E: 99.92±0.06 F1: 0.76±0.04	T:**77.53±2.15** E: 99.74±0.50 F1: 0.77±0.02
1+2+3+4+5+6+7	T:**81.00** E: 93.47 F1: 0.82	T:**90.75±0.43** E:100.00±0.00 F1: 0.91±0.00	T:**84.42±1.39** E:100.00±0.00 F1: 0.84±0.01	T:**77.25±2.68** E:99.95±0.04 F1: 0.76±0.03	T:**76.95±2.78** E: 99.94±0.07 F1: 0.75±0.04

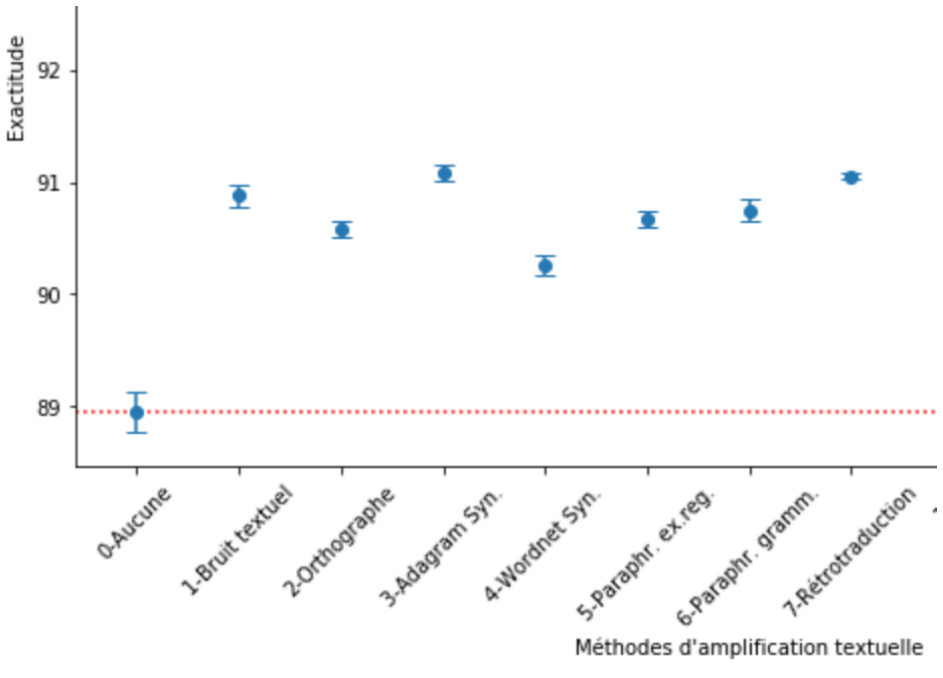
Le tableau précédent (fig. 5.2) répertorie les résultats d'expériences répétées 20 fois pour chacune des différentes techniques d'amplification textuelle (lignes) avec différentes architectures courantes de réseaux de neurones (colonnes)[202]. Avec T: données test, E: données d'entraînement, mesure d'exactitude avec l'écart-type, F1: mesure F1 avec l'écart-type. Les résultats montrent que les différentes techniques d'amplification textuelle étudiées améliorent les performances des modèles, dans une fourchette 0.5 à 8.8%, par rapport à la base de référence qui est le jeu de données initial, sans amplification des données. On constate également d'importantes fluctuations statistiques.

Pour une preuve de concept, l'exactitude maximale atteinte n'est pas très importante, ce qui compte surtout c'est de démontrer l'amélioration des résultats par l'emploi des différentes techniques d'ADT et que l'amélioration est statistiquement significative.

5.3 Choix d'une représentation graphique des résultats

Les résultats sont présentés dans des graphiques avec des barres d'erreur[203] [WIKIPÉDIA, Error bar] qui représentent l'erreur type de la moyenne (pas l'écart-type). Nous voulons montrer l'effet de chaque technique d'amplification par rapport à la base de référence. Rappelons que cette référence correspond à l'exactitude de la prédiction de polarité sans amplification des données (la ligne rouge en pointillé).

Représentation par graphique avec barres-d'erreur et référence (fig. 5.3)



[202] Note: Des expériences avec de la validation croisée 10 plis ont donné des résultats similaires.
[203] En anglais: «error bars». Une représentation graphique de la variabilité des données.

Le choix de l'erreur-type s'est imposé car nous voulions illustrer la signification statistique plutôt que la dispersion des données [Brown, 1982]. Plus précisément, il s'agit de l'erreur type de la moyenne[204] qui mesure l'écart de la moyenne des échantillons par rapport à la moyenne de la population [WIKIPÉDIA, Standard error]. Cette façon de présenter l'information a l'avantage de faire ressortir l'effet comparé des différentes techniques d'amplification.

Il est utile de rappeler que l'écart-type est une mesure de la variabilité d'une donnée individuelle, tandis que l'erreur type de la moyenne est une mesure de la variabilité de la moyenne de toutes les données d'un échantillon.

Formulation mathématique - Erreur type de la moyenne (fig. 5.4)
$$ETM \approx \frac{S_\sigma}{\sqrt{n}}$$
ETM: Erreur type de la moyenne, S_σ l'estimation de l'écart-type et n la taille de l'échantillon

5.4 Signification statistique des résultats

Attention! Une fausse conception, malheureusement très répandue, consiste à croire qu'on peut conclure qu'il existe ou non une différence statistiquement significative en vérifiant simplement que les barres d'erreur se chevauchent ou pas. Ce n'est pas le cas. Il convient ici de rappeler que si deux barres d'erreur (affichant l'erreur type) se chevauchent alors on peut conclure que la différence n'est pas statistiquement significative. Par contre, l'inverse n'est pas vrai. Si les barres d'erreur ne se chevauchent pas, comme dans le graphique précédent, on ne peut rien conclure [Motulsky, 1995], [Belia at al, 2005].

Pour déterminer si l'écart observé entre les moyennes des mesures est statistiquement significatif, nous utilisons un test de signification statistique[205] [WIKIPÉDIA, Statistical significance]. Plus précisément, nous utilisons un test conçu pour comparer des moyennes, le test de signification statistique de Student ou test T de Student[206] [WIKIPÉDIA, Student's t-test]. Le test T de Student mesure si les valeurs moyennes (espérées) diffèrent significativement d'un échantillon à l'autre.

[204] En anglais: «standard error», «SE», «standard error of the mean», «SEM»
[205] En anglais: «statistical significance»
[206] En anglais: «Student's t-test»

Test de signification statistique T de Student (fig. 5.5)

$$t = \frac{\bar{X}_{ampl} - \bar{X}_{réf}}{\sqrt{\frac{s^2_{ampl}}{n_{ampl}} + \frac{s^2_{réf}}{n_{réf}}}} \quad or \quad n_{ampl} = n_{réf} = n \quad d'où \quad t = \frac{\bar{X}_{ampl} - \bar{X}_{réf}}{\sqrt{\frac{s^2_{ampl} + s^2_{réf}}{n}}}$$

t : mesure T de Student, $\overline{X}ampl$: moyenne avec données amplifiées, $\overline{X}réf$: moyenne sans données amplifiées, S^2_{ampl} estimation de la variance avec données amplifiées, $S^2_{réf}$ estimation de la variance sans données amplifiées, n_{ampl} : taille de l'échantillon avec données amplifiées, $n_{réf}$: taille de l'échantillon sans données amplifiées,

Il faut s'assurer que la différence que nous observons n'est pas le fruit du hasard. L'objectif est donc de comparer la moyenne des mesures d'exactitude entre la base de référence (aucune amplification) avec la moyenne des mesures d'exactitude pour chaque technique d'amplification afin de déterminer si la différence observée est statistiquement significative.

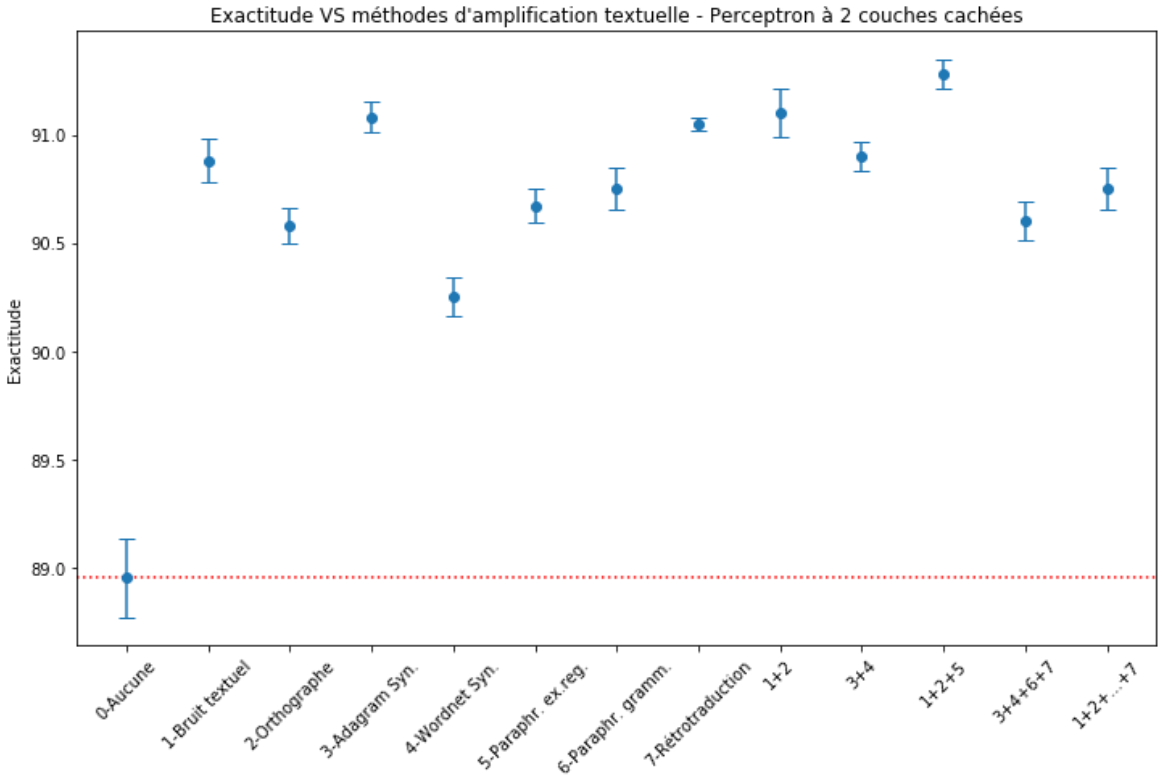
Nous faisons l'hypothèse que les deux mesures sont distribuées selon des lois normales. Ensuite, nous appliquons des tests de signification selon deux hypothèses complémentaires: les variances sont identiques (test T de Student classique) puis l'hypothèse que les variances sont différentes (test T de Welch) [WIKIPÉDIA,Welch's t-test]. Avec 20 répétitions (n = 20), nos expériences se situent dans le cas d'échantillons de petite taille (n < 30). Enfin, la probabilité de rejeter l'hypothèse nulle a été fixé à 0.05 ($\alpha = 0.05$).

Nous désirons comparer les deux distributions en partant de l'hypothèse nulle H_0 que les deux échantillons sont issus d'une même distribution et que la moyenne des résultats avec des données augmentées est la même que la moyenne sans augmentation, soit la moyenne de référence $\mu_{aug} = \mu_{réf}$. Le cas échéant, la différence sera considérée comme statistiquement significative. Le signe du résultat du test `t_score` indique si la moyenne des résultats pour les données amplifiées est supérieure à la moyenne de référence (sans amplification des données) et `p_score`[207] indique le degré de signification. s^2_{aug} et $s^2_{réf}$ ci-dessous sont des estimateurs de la variance. Notons également que les tailles d'échantillon $n_{aug} = n_{réf}$ sont les mêmes. En Python, le calcul est réalisé par la fonction `ttest_ind` de la bibliothèque `scipy.stats`. La même fonction calcule le test T de Welch, il suffit de lui indiquer par un paramètre que les variances sont différentes. Pour cette taille d'échantillons, les deux valeurs sont très proches. Nous avons également utilisé la fonction `shapiro` de la bibliothèque Python `scipy.stats`. pour tester la normalité des échantillons.

[207] En anglais: «p value». En français: «valeur p», «valeur de p», «degré de signification statistique»

5.5 Effet de l'amplification des données - perceptron multicouche

Amplification des données textuelles - perceptron multicouche (fig. 5.6)



Avec le perceptron multicouche, toutes les techniques d'amplification de données textuelles (ADT) montrent une amélioration statistiquement significative de l'exactitude de prédiction du modèle. Cela dans une fourchette comprise entre +1.3% (substitution lexicale avec WordNet) jusqu'à +2.3% (1+2+5, bruit textuel, orthographe, exp. rég.). Ces résultats demeurent significatifs quand on considère la mesure F1, avec un F1 de 0.91 pour la plupart des techniques d'ADT contre un F1 de 0.89 pour la base de référence (sans amplification).

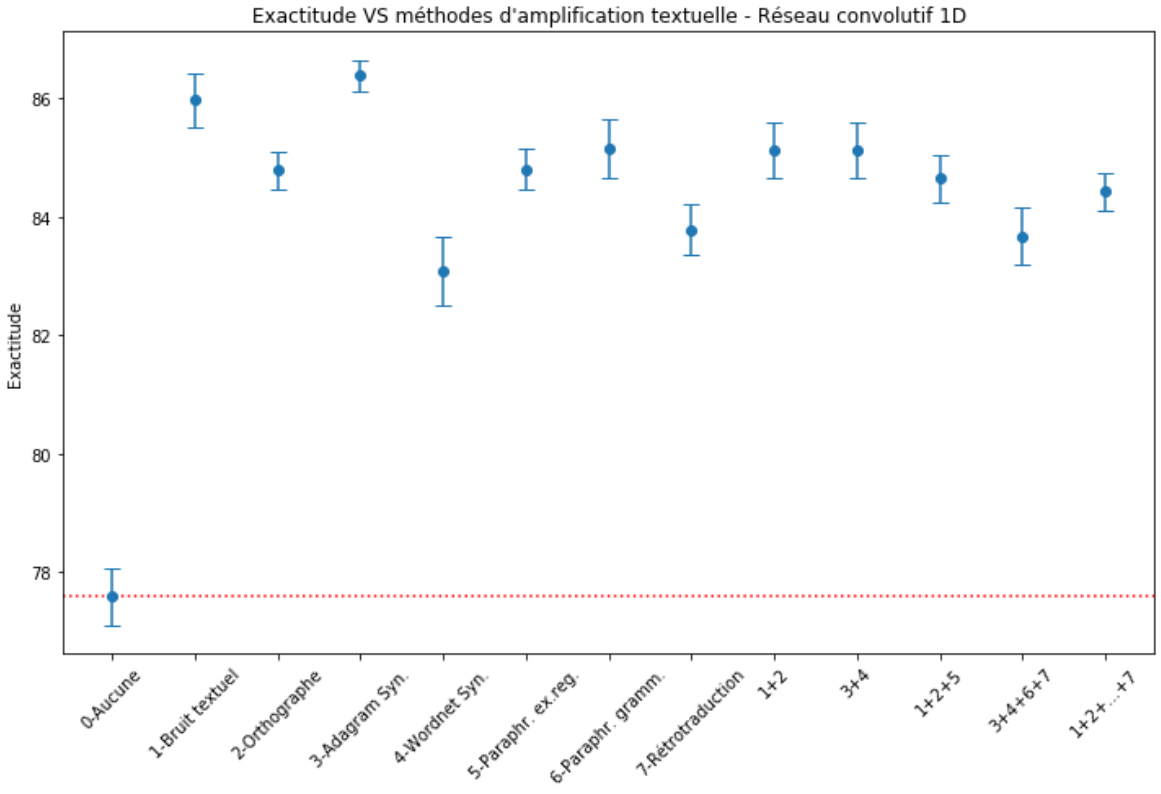
Le meilleur score mesuré est de 91.28 % avec l'addition des jeux de données des techniques 1, 2 et 5 (injection de bruit textuel, l'injection de fautes d'orthographe et transformation de surface au moyen d'expressions régulières). C'est également le meilleur résultat de toutes les architectures de réseaux de neurones testées.

Un autre résultat intéressant est que la substitution lexicale avec AdaGram (91.08%, F1 à 0.91) qui sépare les sens est significativement plus performante que la substitution lexicale simple avec WordNet (90.25%, F1 de 0.90). En fait, la substitution lexicale avec WordNet, que l'on pourrait qualifier de technique classique d'ADT, affiche les pires performances parmi toutes les techniques testées. Au contraire, la substitution lexicale avec AdaGram est la

meilleure technique d'ADT considérée sur une base individuelle pour le perceptron multicouche (91.08%, F1 de 0.91). La substitution lexicale avec AdaGram n'est égalée que par l'addition des jeux de données produits par les techniques 1, 2 et 5 (bruit textuel, fautes d'orthographe, transformations de surface) à 91.28% (F1 de 0.91).

5.6 Effet de l'amplification des données - réseau convolutif

Amplification des données textuelles - réseau convolutif (fig. 5.7)

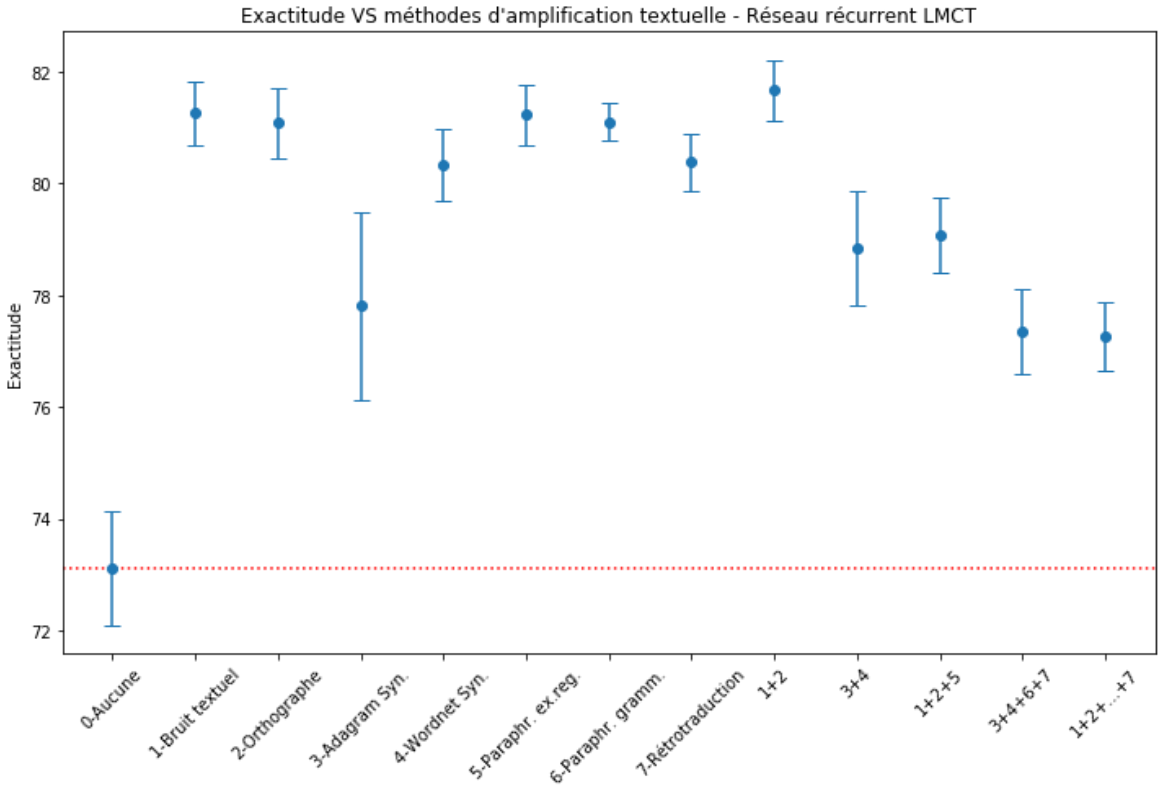


Nous voyons ci-dessus que pour un réseau convolutif 1D, qui comporte 5 couches cachées, l'amplification des données textuelles a également des effets positifs comparativement à la référence (aucune amplification). La fourchette se situe entre +5.5% et +8.8 %. Ces effets sont statistiquement significatifs (test T de Student) autant pour l'exactitude que pour la mesure F1.

Ici, la substitution lexicale avec AdaGram (86.38%, F1 de 0.87) est la technique d'ADT la plus performante à la fois sur le plan individuel et globalement. À nouveau, la substitution lexicale simple avec WordNet est la moins performante des techniques d'ADT.

5.7 Effet de l'amplification des données - réseaux récurrents

Amplification des données textuelles - réseau récurrent LMCT (fig. 5.8)



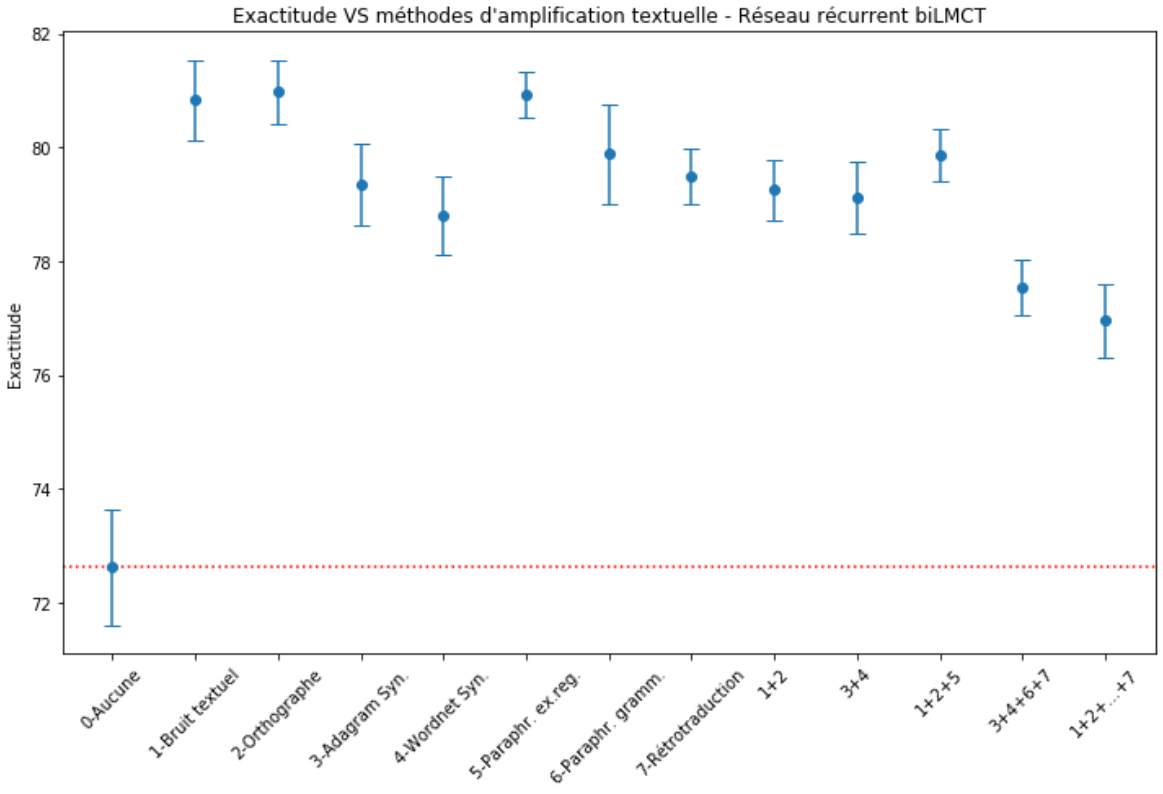
Pour un réseau de type LMCT (réseau récurrent à longue mémoire court terme), on augmente la performance du modèle de presque 10% en injectant du bruit textuel. La fourchette d'amélioration observée se situe entre +4.7% (min 77.80%, F1 0.75) et +8.5 % (max. 81.67%, F1 0.78).

Contrairement aux autres architectures de réseaux de neurones, la substitution lexicale AdaGram affiche ici la pire performance individuelle (77.80%, F1 0.80). Cela est associé à une très grande variabilité des résultats pour ce jeu de données (± 7.26% d'écart-type ou ±1.67% d'erreur type de la moyenne). Il s'agit probablement d'un problème dû à la stochasticité des réseaux de neurones profonds. Malheureusement, la durée et les coûts de l'entraînement d'un tel réseau récurrent, de l'ordre d'une trentaine heures pour une seule exécution avec un processeur graphique, nous a empêché de reprendre l'expérience pour vérifier les causes. Une discussion de la stochasticité des résultats des réseaux de neurones se trouve à l'annexe 11.

L'addition de plusieurs jeux de données amplifiées (3+4), (1+2+5), (3+4+6+7), (1+2+...+6+7) résulte en une détérioration de la performance du modèle LMCT, sauf dans le cas des

techniques d'injection de bruits (1+2) qui affichent la meilleure performance (81.67%, F1 0.81) avec ce type de réseau.

Amplification des données textuelles - réseau récurrent biLMCT (fig. 5.9)



Le réseau récurrent biLMCT offre des performances similaires à celles du réseau récurrent LMCT à l'exception de la substitution lexicale AdaGram qui surpasse la substitution lexicale avec WordNet comme avec les autres architectures de réseaux. La fourchette d'amélioration observée se situe entre +6.2% (min 78.8%, F1 0.78) et +8.3% (max. 80.97%, F1 0.81).

Les méthodes 1, 2 et 5 associées à l'injection de fautes d'orthographe (80.97%), la transformation de surface par des expressions régulières (80.92%) et de bruit textuel (80.83%) sont dans l'ordre les trois méthodes les plus performantes avec le réseau biLMCT.

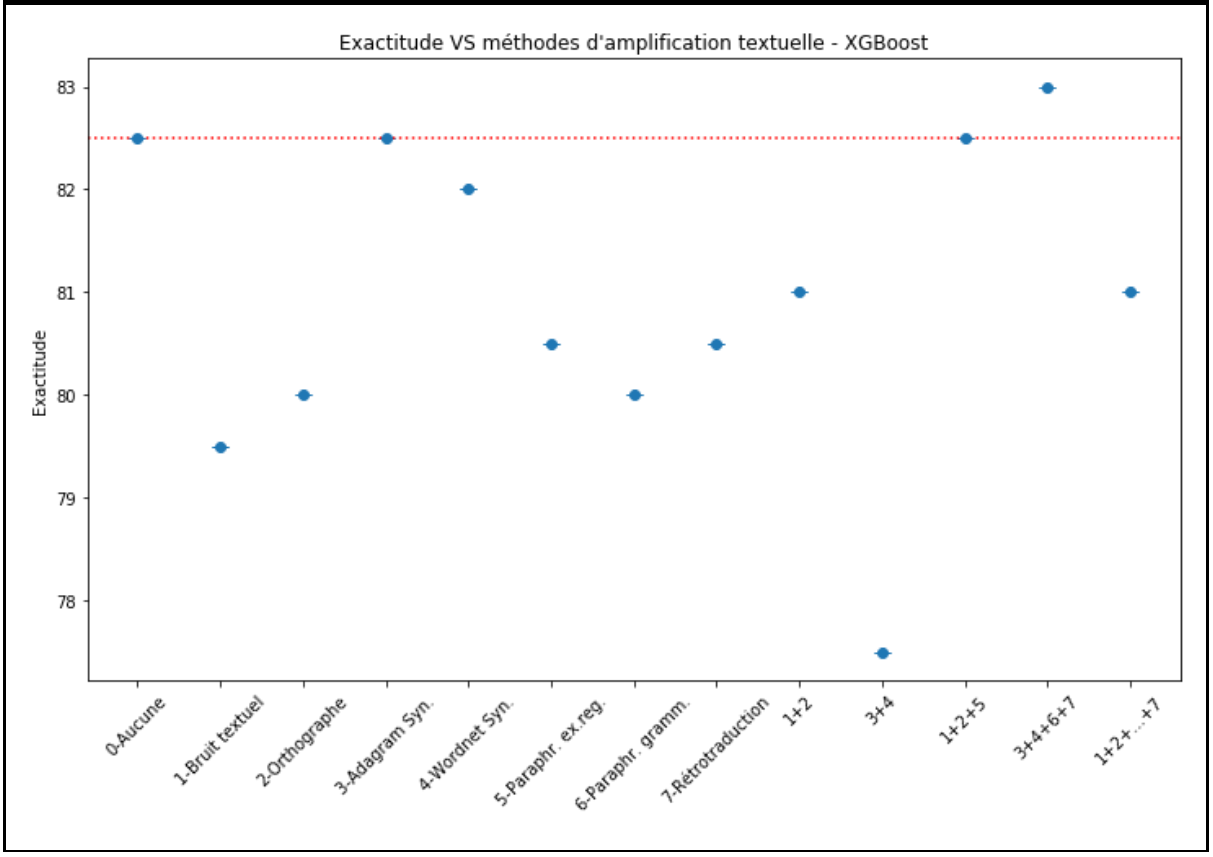
On observe également une détérioration des performances avec l'addition d'un plus grand nombre de jeux de données amplifiées (3+4+6+7) à 77.53%, et (1+2+...+6+7) à 76.95%, les deux pires performances observées avec des données amplifiées pour ce type d'architecture.

5.8 Effet de l'amplification des données avec l'algorithme classique XGBoost

Nous avons également testé avec l'algorithme classique XGBoost qui est très efficace dans les situations ne nécessitant pas l'emploi de l'apprentissage profond, en général parce qu'on ne dispose pas d'une quantité suffisante de données.

Il est intéressant de constater qu'un excellent algorithme d'apprentissage classique comme XGBoost donne un résultat intéressant à partir des données originales, sans amplification. En terme de performance de base, à 82.50 % d'exactitude, XGBoost bat les réseaux récurrents testés (LMCT et biLMCT). On ne s'étonne pas que l'algorithme XGBoost soit devenu l'algorithme de référence pour les petits jeux de données.

Amplification des données textuelles - algorithme XGBoost (fig. 5.10)



En réalité, XGBoost peine à exploiter l'ajout de données amplifiées. Au contraire on observe une dégradation de ses performances, jusqu'à -5% pour la combinaison de la substitution lexicale avec AdaGram et WordNet. Sur le graphique, XGBoost semble montrer une petite amélioration de 0.5% avec le groupe de données amplifiées (3+4+6+7) qui s'avère non statistiquement significative quand on considère la mesure F1 stable à 0.83. Cela est peut être relié au fait que les algorithmes d'arbres de décision dopés sont robustes aux données corrélées [Chen, Benesty & He, 2018].

5.9 Expérimentation avec un autre jeu de données

Afin de valider notre méthodologie et contrevérifier nos résultats, une série d'expériences a été réalisée avec un autre jeu de données bien connu. Nous avons utilisé des données de YELP sur des appréciations de commerces et restaurants. L'annexe 13 contient davantage de détails sur ces données.

Globalement, les résultats obtenus confirment l'hypothèse d'un effet positif de l'amplification des données sur la prédiction de la polarité des textes mais l'effet est moins prononcé, de l'ordre de 3% maximum que pour les données IMDB. On remarque également que pour plusieurs techniques et architectures de réseaux de neurones profonds, l'amélioration n'est souvent pas statistiquement significatives, voir le tableau à l'annexe A13.2.

Ici, la substitution lexicale basée sur les synonymes fournis par WordNet détériore carrément les performances pour le jeu de données YELP. Par contre la génération de paraphrases par rétrotraduction se distingue positivement et donne même les meilleures performances observées pour le réseau convolutif et les deux réseaux récurrents. Sur un plan plus global, nous notons une relative sous-performance des réseaux récurrents et au contraire de meilleurs résultats du réseau convolutif.

Ces observations peuvent toutes s'expliquer par la même particularité du jeu de données YELP, soit la taille réduite des textes et des phrases. Les textes du corpus YELP (détails dans l'annexe 13) sont en effet beaucup plus courts avec une taille médiane de 94 mots ou 6 phrases comparativement à 602 mots ou 31 phrases pour le corpus IMDB (détails dans l'annexe 12). Aussi la taille médiane des phrases est de 13 mots pour YELP contre 18 mots pour IMDB.

La piètre performance de la substitution lexicale WordNet s'expliquerait par la taille réduite des contextes (contexte de la phrase, et contexte du texte entier) fournis à l'algorithme de filtrage des synonymes. Disposant de contextes moins riches, il est normal que l'algorithme de sélection des synonymes fasse davantage d'erreur au point d'empirer les résultats.

Au contraire, la bonne performance de la génération de paraphrases par rétrotraduction s'expliquerait justement par la taille réduite des phrases. En effet, la qualité de la rétrotraduction est particulièrement sensible à la longueur des phrases, de plus courtes phrases donnent de meilleurs résultats pour la rétrotraduction.

De même, la sous-performance des réseaux récurrents s'expliquerait par la taille réduite des données traitées, surtout des dépendances à plus longue portée qui font leur force. Enfin, la meilleure performance relative du réseau convolutif 1D s'expliquerait par sa capacité de justement mieux exploiter les contextes plus locaux des phrases courtes.

D'où émerge l'idée de choisir les techniques d'ADT en fonction des types de textes.

Chapitre 6 - Discussion des résultats

Toutes les techniques d'amplification de données textuelles (ADT) testées ont permis d'accroître l'exactitude des résultats sur une tâche normalisée de prédiction de la polarité de critiques de film et cela pour différentes architectures de réseaux de neurones.

Notre hypothèse de départ est donc vérifiée et les techniques d'ADT que nous proposons améliorent les résultats dans une fourchette de 0.5 à 8.8% sur une tâche normalisée de prédiction de la polarité de textes de critiques de film de la base de données IMDB.

6.1 Analyse des résultats pour chaque technique

L'injection de bruit textuel et d'erreurs de frappe et de fautes d'orthographe posent la question de la frontière entre bruit léger et chaos. Il pourrait être intéressant d'investiguer sur ce sujet. Nous pourrions expérimenter en contrôlant la quantité et le type de bruit textuel injecté et observer les résultats. De même, il serait intéressant de mesurer l'effet de l'injection graduelle de plus en plus de fautes d'orthographe.

La substitution lexicale en utilisant des vecteurs-mots AdaGram, qui rappelons-le sépare les synonymes des mots polysémiques entre leurs différents sens, a mieux performé que la substitution lexicale simple avec WordNet.

À 91.28%, notre meilleur résultat obtenu[208] se compare aux scores de 90% obtenus pour des réseaux de neurones profonds finement ajustés [Brownlee, 2018b]. Cela est d'autant plus remarquable que nous n'utilisons aucune technique de régularisation ni d'enrichissement des données par des vecteurs-mots externes.

L'ajout de données amplifiées, issues des techniques d'ADT prises isolément, améliore (+4% à +8.55%) la performance des réseaux récurrents. Par contre, les réseaux récurrents peinent à exploiter l'addition des données amplifiées de plusieurs techniques d'ADT. En effet, avec l'addition de plusieurs jeux de données amplifiées (3+4+6+7) on voit plonger les performances des modèles à base de réseaux récurrents. L'effet est encore plus visible lorsque l'on additionne l'ensemble des données amplifiées (1+2+3+4+5+6+7). Il en résulte les résultats les plus faibles observés pour les réseaux récurrents.

XGBoost qui est considéré comme le meilleur algorithme classique d'apprentissage automatique n'arrive pas à bien exploiter l'ajout de données amplifiées.

[208] Note: avec le perceptron multicouche (PMC)

6.2 Tendances observées

L'analyse des erreurs[209] est l'examen manuel des exemples mal classés dans le but de découvrir les causes sous-jacentes. Il s'en dégage que certains des exemples mal classés devraient être classés dans une troisième catégorie, la catégorie «neutre» car ils ne sont ni clairement positifs, ni clairement négatifs.

Les techniques d'ADT 1, 2 et 5 (bruit textuel, fautes d'orthographe, transformations de surface) se montrent particulièrement efficaces avec les réseaux récurrents (LMCT et biLCTM) et le perceptron multicouche. L'injection de bruit contribue généralement davantage à la robustesse de l'apprentissage [Xie et al, 2017]. Une partie des bonnes performances observées par ces techniques seraient dues à l'effet de régularisation par le bruit des réseaux de neurones [Goodfellow, Bengio & Courville, 2016].

Nous nous attendions à observer un effet cumulatif des différentes techniques d'ADT avec une performance qui augmenterait jusqu'à plafonner avec l'addition des différents jeux de données amplifiées. Cet effet attendu est partiellement observé pour les techniques impliquant l'injection de bruit textuel, de fautes d'orthographe et les transformations de surface (1+2+5), mais globalement, on assiste plutôt à une dégradation des performances à mesure que l'on additionne de nouveaux jeux de données amplifiées.

Cette dégradation des performances avec l'addition de nouveaux jeux de données amplifiées est particulièrement évidente pour les architectures de réseaux de neurones récurrents. À la défense des réseaux récurrents, ils n'ont fait l'objet d'aucun réglage fin. Ces réseaux sont régularisés uniquement par l'ajout de données et l'arrêt précoce. Donc, pas d'enrichissement par vecteur-mots externes, pas de régularisation L1, L2, pas d'extinction de neurones[210], ni d'ajustement des hyperparamètres. Rappelons que la procédure était délibérée afin de ne faire ressortir que le seul effet de l'amplification des données.

Aussi, les techniques d'ADT qui produisent des paraphrases davantage différentes des phrases de départ comme la transformation d'arbres syntaxiques et la rétrotraduction semblent se nuire mutuellement. Logiquement, ces techniques d'ADT devraient contribuer davantage à la reconnaissance de nouvelles formes dans les données et corriger des erreurs de classification. L'examen des exemples mal classés montre qu'à partir d'un certain seuil, les effets semblent s'annuler plutôt que de se combiner.

Une explication est peut être à rechercher du côté des différences dans les distributions statistiques entre les différents jeux de données amplifiées. Particulièrement entre la distribution statistique des données qui résultent de l'addition des jeux de données amplifiées et la distribution statistique des données de test.

---

[209] En anglais: «error analysis»
[210] En anglais: «dropout»

Il se peut également que ce soit simplement un problème d'affinement des techniques d'ADT afin quelles ajoutent des informations davantage pertinentes. Par exemple, au départ la technique de substitution lexicale AdaGram performait moins bien que la technique qui se basait uniquement sur WordNet. Or, avec un effort de mise au point, la technique de substitution lexicale AdaGram donne maintenant des résultats supérieurs.

On peut aussi légitimement se demander si les paraphrases générées introduisent davantage de bruit que de signaux utiles (malédiction de la haute dimension), mais nos expériences ne sont pas assez affinées ni exhautives pour se prononcer.

Une solution serait de retirer les données trop fortement corrélées ouencore de réduire la dimension[211] des données avec des algorithmes comme l'analyse en composante principale (ACP)[212], la décomposition en valeurs singulières (DVS)[213], la normalisation des lots[214][Ioffe & Szegedy, 2015] ou en modifiant l'architecture des réseaux de neurones (introduire un étranglement comme pour un autoencodeur). Notre budget calcul limité n'a pas permis d'explorer ces avenues. Pour une discussion sur les données corrélées lire l'annexe 11.

Analyse des résultats - grandes tendances (fig. 6.1)
• *Toutes les techniques d'ADT testées ont permis d'accroître l'exactitude des résultats dans une fourchette de 0.5% à 8.8% sur le corpus IMDB;* • *Avec* 91.28%, le *perceptron multicouche s'avère l'architecture la plus performante;* • *Contrairement aux attentes, on assiste parfois à une dégradation des performances à mesure que l'on additionne de nouveaux jeux de données amplifiées, particulièrement avec les architectures de réseaux récurrents;* • *Les techniques d'ADT qui produisent des paraphrases très différentes des phrases originales comme la transformation d'arbres syntaxiques et la rétrotraduction qui devraient contribuer à la reconnaissance de nouvelles formes dans les données et corriger des erreurs de classification semblent s'annuler plutôt que se combiner;* • *Les techniques de transformation de surface (bruit lexical, fautes d'orthographe, expressions régulières) s'avèrent les plus performantes compte tenu de leur rapidité et de leur faible intensité en calcul;* • *La substitution lexicale avec AdaGram qui traite la polysémie est meilleure que la substitution lexicale simple avec WordNet;* • *La rétrotraduction donne d'excellents résultats sur certains jeux de données, avec des résultats plus bruités mais elle est automatique;* • *La transformation d'arbres syntaxiques est plus précise, voire chirurgicale, mais exige beaucoup de travail manuel;* • *L'algorithme classique XGBoost peine à exploiter les données amplifiées.*

[211] En anglais: «dimension reduction, dimensionality reduction (sic)». Le mot «dimensionnalité» en anglais «dimensionality» est inutile puisque le mot dimension existe déjà.
[212] En anglais: «principal component analysis», «PCA»
[213] En anglais: «singular value decomposition», «SVD»
[214] En anglais: «batch normalization»

6.3 Analyse théorique

Il aurait été intéressant de faire une analyse mathématique sur la base de la théorie PAC de l'apprentissage probablement approximativement correct[215] [Valiant, 1984] ou de la théorie de l'apprentissage VC et de la dimension de Vapnik-Chervonenkis [Vapnik, 1999]. Une telle analyse aurait permis de fixer une limite (epsilon) pour l'erreur d'apprentissage sous une certaine probabilité (delta). Cette erreur epsilon contrôle le nombre d'exemplaires m nécessaire pour apprendre.

En réalité, on ne comprend pas pourquoi l'apprentissage en profondeur fonctionne aussi bien et c'est un sujet encore chaudement débattus par les théoriciens du domaine [Daniely, 2018]. L'extension de la dimension VC aux réseaux de neurones profonds se heurte à nombreuses difficultés. En fait la thérie l'apprentissage VC donne des bornes très supérieures[216] aux quantités de données pour lesquelles on constate une bonne généralisation avec les réseaux de neurones [Zhang, et al, 2016].

Une telle analyse mathématique appliquée aux algorithmes d'apprentissage profond est un travail théorique qui va bien au-delà du périmètre de cette thèse.

6.4 Avantages

Dans plusieurs situations pratiques, l'amplification de données textuelles est plus efficace, plus rapide et moins coûteuse que la collecte de «vraies» données supplémentaires, particulièrement s'il faut payer des foules pour annoter manuellement les données. Il existe également des cas où il est tout simplement impossible d'obtenir de nouvelles données.

L'amplification de données textuelles peut aider à adapter des données existantes à de nouveaux besoins. On peut aussi amplifier des données pour traiter davantage de cas . Une autre possibilité est de profiter de l'amplification des données pour dissimuler des informations confidentielles ou pour protéger la vie privée. Ceci peut représenter un avantage dans le cas de données médicales, juridiques ou financières.

6.4.1 Solution multilingue, de qualité satisfaisante

Les services en ligne utiles aux algorithmes l'amplification textuelle sont offerts dans un grand nombre de langues. La qualité des analyses syntaxiques ou traductions est satisfaisantes dans beaucoup de langues. En cela, des services en ligne comme SyntaxNet pour l'analyse linguistique et Google translate pour la traduction répondent plutôt bien aux besoins de l'amplification textuelle.

---

[215] En anglais: «Probably Approximately Correct learning», «PAC»
[216] Note: De l'ordre du nombre de neurones O(N) ou même du nombre d'arêtes

6.4.2 Solution robuste, fiable et capable de monter en charge

Du point de vue pratique et du génie logiciel, les services en ligne des fournisseurs commerciaux importants sont robustes, fiables et capables de monter en charge.

6.4.3 Solution simple et pratique

Les solutions faisant appel aux services en ligne sont faciles à mettre en oeuvre. Quelques lignes de code suffisent pour appeler un service en ligne et récupérer les résultats.

6.4.4 Solution peu onéreuse

Dans plusieurs situations pratiques, l'amplification de données textuelles est moins coûteuse que la collecte et l'étiquetage de «vraies» données supplémentaires.

Par exemple, dans nos expériences, les prétraitements nécessaires à l'amplification des données textuelles n'ont coûté que quelques dizaines de dollars canadiens. Par contre l'entraînement des modèles profonds a parfois dépassé les 1000 dollars sur une infrastructure commerciale louée comme Amazon AWS avant de bénéficier de l'infrastructure de Calcul Québec.

6.4.5 Possibilité d'automatisation

Nous l'avons vu précédemment, l'écriture de règles de transformations sémantiquement invariantes exige une connaissance du sens commun qui rend l'automatisation complète de cette tâche difficile. Cela dit, il est tout à fait envisageable d'assister ce travail pour le rendre beaucoup moins laborieux.

Premièrement, il existe des ressources comme des dictionnaires, des lexiques et des services linguistiques accessibles via des IPA qui pourraient être mis à contribution. Deuxièmement, on peut utiliser des outils de dépouillement de corpus comme les concordanciers[217] pour dépister et collecter les variantes et des exemples intéressants pouvant faire l'objet de règles. On peut également utiliser des moteurs de recherche pour identifier des gabarits de règles [Androutsopoulos & Malakasiotis, 2010]. Troisièmement, on pourrait générer automatiquement des règles, particulièrement des expressions régulières à partir de gabarits et d'exemples. Les transformations d'arbres syntaxiques pourraient également faire l'objet d'un formalisme plus facile à coder.

[217] En anglais: «concordancer». Le concordancier est un outil logiciel qui permet de faire la recherche de concordances (un mot accompagné de son contexte) dans un corpus.

6.4.6 Possibilité de traitement distribué

Pour le moment le traitement proposé se fait par lots en deux phases. La première phase s'occupe de l'amplification des données proprement dite. La deuxième phase consomme les données amplifiées pour entraîner des modèles qui sont essentiellement des réseaux de neurones profonds.

Afin de réaliser les calculs impliqués, il est avantageux de disposer de processeurs graphiques et de bonnes capacités de stockage de données. Les expérimentations de cette étude ont été réalisées en grande partie sur des infrastructures infonuagiques en location chez Google (Google Cloud) ou Amazon (Amazon Web Services, AWS) puis sur l'infrastructure de Calcul Québec. Nos évaluations montrent, qu'avec des processeurs graphiques, les calculs sont environ 50 fois plus rapides.

La preuve de concept a été développée avec des carnets interactifs IPython sur un ordinateur portable. Les traitements plus massifs étaient réalisés sur une infrastructure infonuagique avec ces mêmes carnet, mais le plus souvent avec des codes Python directement tirés de ces carnets et lancés à partir d'une ligne de commande.

Les tâches se basant sur les phrases individuelles peuvent être facilement découpées à ce niveau pour un traitement distribué selon une architecture diviser-regrouper[218], la version ouverte de la technologie propriétaire MapReduce de Google. Hadoop est une implémentation en code source libre de l'architecture diviser-regrouper.

Pour économiser de l'espace de stockage, il serait possible de générer les données au fur et à mesure et entraîner les modèles à partir de flux de données. Ce serait alors de l'apprentissage en flux continu[219].

Une solution basée sur la plateforme de traitement distribuée Spark serait appropriée. Des essais ont été menés mais en traitement par lots et uniquement sur la partie amplification de données afin de se familiariser avec la plateforme Spark en utilisant l'IPA PySpark sur des serveurs Databricks (Community Edition). L'amplification de données pourrait être fortement accélérée, de l'ordre de dix fois, dépendant du nombre de serveurs mobilisés pour la tâche. Pour un déploiement industriel, le traitement serait distribué sur plusieurs serveurs (emploi de Spark de Hadoop).

En entreprise, le déploiement se fera en interne lorsqu'elle dispose de l'infrastructure nécessaire. Dans un premier temps ou pour un projet pilote, le moins risqué serait de louer

[218] en anglais: «map-reduce»
[219] en anglais; «online training». Aussi en français: «apprentissage en flux», «apprentissage au fur et à mesure», «apprentissage par flux de données», «apprentissage progressif».

l'infrastructure et les services linguistiques à des fournisseurs comme Google Cloud Engine [Google, 2018b] ou Amazon Web Services (AWS).

Au final, on réalise qu'il y a de véritables possibilités d'industrialisation des processus.

Amplification de données textuelles - avantages (fig. 6.2)
● *Solution plus efficace, plus rapide et moins coûteuse que la collecte et l'annotation de «vraies» données supplémentaires;* ● *Solution robuste, fiable et capable de monter en charge;* ● *Solution simple et pratique;* ● *Solution peu onéreuse;* ● *Possibilité d'automatisation;* ● *Possibilité de traitement distribué.*

6.5 Inconvénients

6.5.1 Quantité massive de traitements et calculs

Le principal inconvénient de certaines des techniques d'amplification textuelle explorées demeure la quantité de traitement et leur aspect massif qui impose pratiquement l'emploi d'infrastructure infonuagiques.

La distribution des calculs et les contraintes de temps alloué exigent de découper les tâches au maximum. Même sur une architecture distribuée de grande envergure comme celle de Calcul Québec, on parle de plusieurs centaines de tâches (~500) et de plusieurs semaines de calcul. Par exemple, avec un perceptron multicouche qui est la seule architecture à pouvoir être entraînée avec toutes les combinaisons des 7 techniques d'amplification (2^7 = 128), le calcul prenait environ 8 jours.

Il faut toutefois mettre dans la balance que certaines solutions alternatives basées sur l'entraînement de réseaux de neurones spécialisés peuvent s'avérer encore plus coûteuses en terme de calcul sans les avantages d'une solution plus facile à mettre en oeuvre.

6.5.2 Dégradation des performances avec la longueur des phrases

Les techniques d'amplification de données textuelles basées sur la transformation d'arbres syntaxiques et la rétrotraduction se dégradent avec la longueur et la mauvaise qualité des phrases traitées.

En fait, pour éviter des calculs inutiles, on peut restreindre les analyses à de courtes phrases. La détermination du seuil de coupure est basé sur une heuristique ou par

entraînement sur des données. Il s'agit d'un petit problème de classification binaire supervisée avec un classificateur logistique où les données sont annotées à la main sur quelques milliers d'observations. Les attributs sont la longueur de la phrase d'entrée et une étiquette de classe, bonne_paraphrase (on conserve) versus mauvaise_paraphrase (on rejette).

Le point de coupure devrait varier selon la qualité du texte à amplifier. Par exemple, un texte bien rédigé par un rédacteur professionnel devrait être plus facilement «analysable» et le point de coupure sera plus élevé qu'un simple commentaire rédigé négligemment. Nous avons constaté le problème mais des expériences plus fines impliqueraient une mesure de la qualité du texte, par exemple sa grammaticalité. Une idée serait d'évaluer la qualité des phrases par rapport à un corpus de référence avec une métrique du genre BLEU[220].

6.5.3 Effet de masque

Les données amplifiées sont susceptibles de masquer ou de diluer certains signaux faibles présents dans les données originales. Par exemple, on peut se questionner sur l'effet d'ajouter d'importantes quantité de données similaires, fortement corrélées et potentiellement redondantes (malédiction de la haute dimension), voir l'annexe 11.

La présence de beaucoup de données redondantes ou dans l'ensemble de données d'entraînement pourrait biaiser l'algorithme d'apprentissage en donnant plus d'importance statistique aux données plus fréquentes ce qui aurait pour effet de masquer l'apprentissage de données moins fréquentes.

6.5.4 Dépendance envers des services de fournisseurs privés

Certaines des méthodes d'amplification proposées dépendent de services linguistiques en ligne comme les services de traduction et d'analyse syntaxique qui sont loués à des entreprises privées. Des solutions de mitigation sont discutées à la section suivante.

6.5.5 Alternatives aux ressources en ligne de fournisseurs privés

Pour des raisons pratiques, nous avons opté pour l'emploi de services de la plateforme infonuagique Google Cloud Platform [Google, 2018b]. Plus précisément, nous utilisons le service d'analyse syntaxique[221] basé sur SyntaxNet [Petrov, 2016], [Kong et al, 2017], avec l'IPA de langage naturel (Cloud Natural Language API) [Google, 2018c] et le service de traduction automatique, basé sur Google Translate, avec l'IPA de traduction (Cloud Translation API) [Google, 2018d].

---

[220] En anglais: «BLEU score» pour «bilingual evaluation understudy». L'algorithme BLEU permet de comparer un texte à un corpus de référence en considérant le recouvrement des n-grammes.
[221] En anglais: «syntax analysis»

D'un point de vue pratique il faut magasiner ces services auprès de plusieurs fournisseurs. Par exemple, Amazon offre la solution de traitement de la langue naturelle «Comprehend» [Amazon, 2018] et Microsoft ses «Cognitives services» qui incluent un service de traduction [Microsoft, 2018]. Les différents fournisseurs offrent des quotas d'utilisation gratuite et des tarifs plus ou moins avantageux. On peut s'attendre qu'à moyen terme, tous ces services en ligne deviendront des commodités.

Si l'on dispose d'une bonne infrastructure informatique à l'interne ou en location, une deuxième solution est d'entraîner son propre modèle de traduction neuronal et son propre analyseur morphosyntaxique. Ici, le principal obstacle est de réunir suffisamment de corpus de bonne qualité.

Malgré que SyntaxNet soit plutôt difficile à installer sur un serveur ou un poste local, la licence en code source ouvert de Google permet de le faire. Trois installations sont possibles: installation via une image docker fournies par Google [Docker, 2016], installation via une image de serveur préconfigurée (du genre AMI: Amazon Machine image) ou l'installation sur des serveurs loués sur une infrastructure infonuagique commerciale à partir du code source de Google [Google, 2017], [Poddutur, 2018]. Une alternative intéressante à SyntaxNet est d'utiliser spaCy [Honnibal & Montani, 2017] qui offre des analyseurs syntaxiques préentraînés de bonne qualité pour un bon nombre de langues.

Pour remplacer Google Translate, il demeure toujours possible d'entraîner son propre modèle de traduction neuronal à partir de codes sources ouverts comme l'ont fait [Edunov & al, 2018] et [Wieting, Mallinson & Gimpel, 2017].

Amplification de données textuelles - inconvénients (fig. 6.3)
● *Importante quantité de calcul;* ● *Dégradation avec la longueur et la mauvaise qualité des phrases;* ● *Effet de masque;* ● *Dépendance envers des services de fournisseurs privés.*

6.6 Comparaison avec d'autres approches

6.6.1 Créer des réseaux de neurones pour nourrir des réseaux de neurones

L'essentiel de l'effort de recherche en amplification textuelle se concentre sur la mise au point de solutions d'apprentissage de bout-en-bout [Edunov & al, 2018], [Kobayashi, 2018], [Prabhumoye et al., 2018], [Wang et al, 2018], [Fadaee, Bisazza & Monz, 2017], [Wieting, Mallinson & Gimpel, 2017], [Zhang et al, 2017]. Il s'agit un objectif à long terme qui revient à utiliser des réseaux de neurones pour nourrir des réseaux de neurones.

Ces techniques expérimentales sont très difficiles à mettre en oeuvre. Elles requièrent une maîtrise des arcanes des réseaux de neurones car les modèles génératifs sont très sensibles, particulièrement au niveau de leurs hyperparamètres. Elles impliquent également des calculs particulièrement longs et fastidieux.

Au contraire, les techniques d'amplification des données textuelles (ADT) proposées dans cette étude s'apparentent davantage à celles utilisées en vision par ordinateur qui consistent à amplifier les données par des transformations simples qui préservent la similitude. Aussi, dans la chaîne de traitement, l'amplification des données se situe à l'étape du prétraitement.

Rappelons que les techniques d'ADT que nous avons expérimentées sont robustes, peu coûteuses et faciles à mettre en oeuvre.

6.6.2 Génération de texte à partir de réseaux antagonistes génératifs

Plusieurs tentatives ont été faites pour développer des réseaux antagonistes génératifs (RAG) ou des réseaux récurrents de neurones génératifs (RRNG) pour les données textuelles comme TextGAN [Zhang et al, 2017]. Il existe d'ailleurs une plateforme, appelée TexyGen, pour comparer les différentes approches [Zhu et al, 2018], [Texygen, 2018].

A priori, elles se heurtent à la nature discontinue de la langue naturelle car les RAG ont été conçus pour traiter des données continues. Dans le forum Reddit, Ian Goodfellow, l'inventeur des RAG, explique pourquoi un réseau antagoniste génératif textuel n'est pas réalisable en utilisant la descente de gradient [Goodfellow, 2016].

Pour le moment, les résultats sont mitigés ou produisent des textes sans grande cohérence. Qualitativement, le résultat de la génération de textes à partir de modèles génératifs comme les réseaux antagonistes génératifs[222] ou de réseaux récurrents de neurones génératifs (RRNG) (generative recurrent neural networks, GRNN) s'apparente à de la génération automatique de textes ou à la sortie de modèles de chaînes de Markov. La génération de textes par réseaux neuronaux profonds souffre d'une absence quasi complète de contrainte sémantique et passe difficilement le test de la règle d'invariance sémantique. Pour le moment, il peuvent générer de la poésie mais demeurent d'une utilité douteuse pour l'amplification textuelle lorsque le sens est important. Par conséquent à moins d'introduire des contraintes sémantiques ou des gabarits structurants comme des scripts logiques, ils ne semblent pas utilisables pour l'amplification de données textuelles dans l'immédiat.

Lors d'expériences en marge de cette thèse, nous avons comparé les résultats de la génération de texte par des RNN et de simples modèles de Markov. En gros, les résultats sont comparables mais au prix de très long calculs pour les RNN alors que les modèles de Markov sont quasi instantanés à entraîner.

[222] En anglais: «generative adversarial network», «GAN»

Un expert en apprentissage profond affirmera, qu'en théorie, un réseau est capable de apprendre s'il a suffisamment de données pertinentes. Mais pour le moment, la génération de phrases à partir de réseaux récurrents tarde à produire des phrases sensées, même en considérant les progrès annoncés récemment par OpenAI avec son modèle génératif GPT-2 (Generative Pre-training Transformers) [Radford et al, 2019].

Nous nous risquons à donner une explication. Une petite transformation d'un signal ou une légère modification du gradient sur une donnée continue comme une image ou un signal sonore a une faible probabilité de changer radicalement le sens de cette donnée. Donc on a de forte chance d'avoir une transformation sémantiquement invariante. Par contre, la transformation discrète d'un mot, d'une partie d'une phrase ou d'un bout de texte dans une donnée textuelle, parfois seulement changer une lettre, si elle n'est pas finement contrôlée, peut avoir une forte incidence sur le sens de cette donnée textuelle transformée. Elle a une plus forte probabilité de ne pas être une transformation sémantiquement invariante. Ceci expliquerait en partie les problèmes des réseaux antagonistes génératif pour le texte.

L'apprentissage par renforcement[223] non soumis aux contraintes de continuité des systèmes d'apprentissage basés sur la descente de gradient présente davantage d'intérêt, mais il n'a pas été étudié dans cette thèse.

L'avantage principal des techniques proposées est que l'on n'essaie pas de générer des phrases ex-nihilo à partir d'un modèle appris de phrases sensées, mais plutôt par modification de phrases déjà sensées au moyen de transformations sémantiquement invariantes.

6.7 La place de l'amplification de données dans la chaîne de traitement

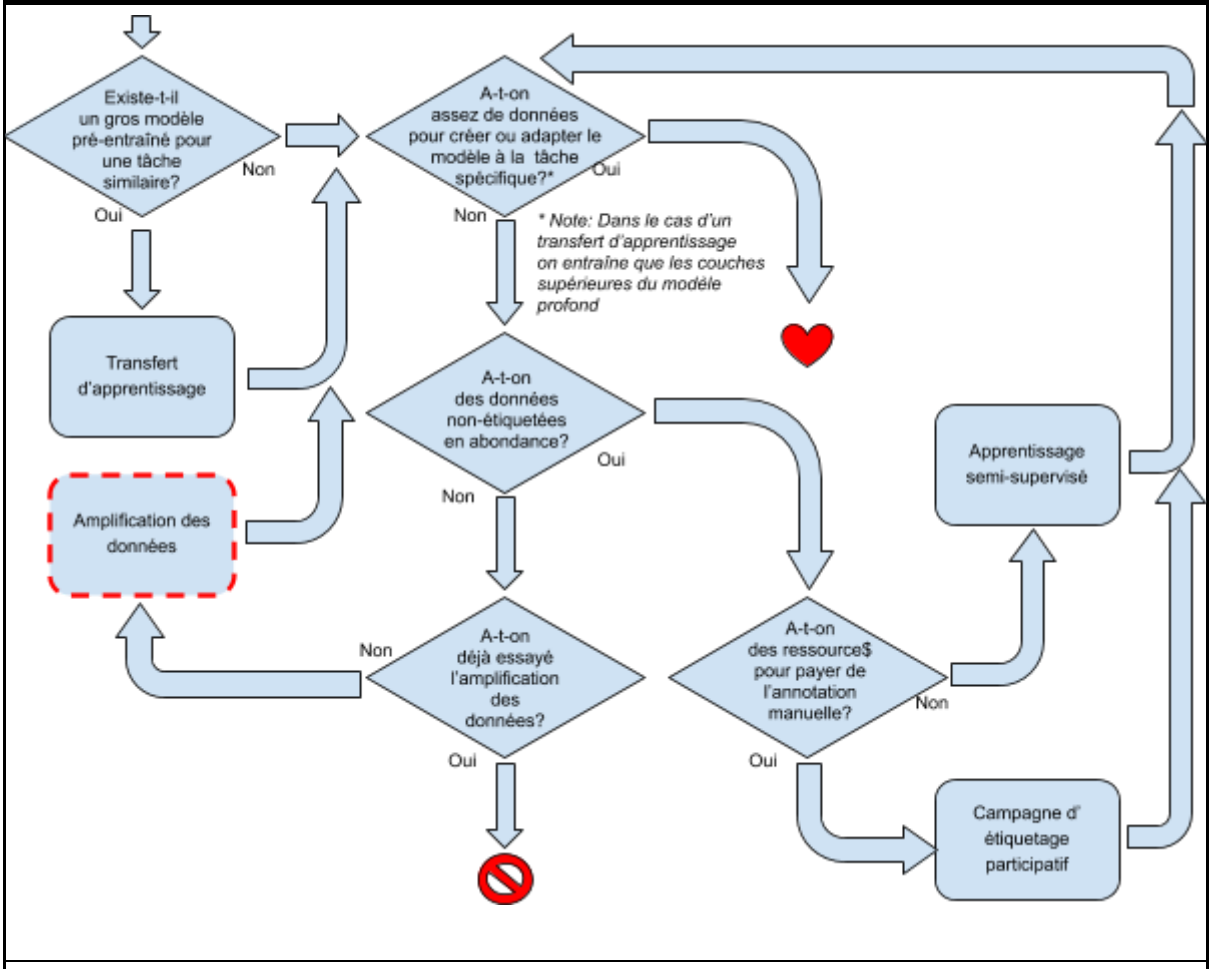
En 2018, nous avons assisté à l'émergence de plusieurs gros modèles de langue comme BERT [Devlin et al, 2018], ELMo [Peters et al, 2018], ULMFiT [Howard & Ruder, 2018] et Transformer [Radford et al, 2018]. On peut raisonnablement envisager que l'apprentissage par transfert devienne une pratique courante pour les applications complexes en traitement de la langue naturelle [Ruder, 2018], [Ruder, 2019]. C'est déjà le cas dans le domaine de la vision artificielle.

Rappelons qu'en apprentissage par transfert, on part d'un gros modèle[224] de langue préentrainé sur un vaste corpus auquel on superpose de nouvelles couches de neurones entraînées sur des données spécifiques à la tâche. Les techniques d'ADT peuvent aider à deux niveaux. Au niveau de l'enrichissement des corpus pour la construction de gros modèles génériques. Mais surtout, pour la mise au point d'applications dans des domaines spécifiques où l'amplification textuelle peut jouer un rôle clé.

---

[223] En anglais: «reinforcement learning»
[224] Note: Voir la section 1.37, les annexes A8.13 et A9.2

Place de l'amplification de données dans la chaîne de traitement (fig. 6.4)



L'amplification de données peut s'avérer essentielle, même dans des situations où l'on utilise l'apprentissage par transfert avec un gros modèle préentraîné. Il s'agit des nombreux cas où l'on ne dispose pas de suffisamment de données spécifiques pour entraîner ou adapter le modèle pour une nouvelle tâche.

Il existe des situations où l'amplification des données est l'unique recours. Il s'agit de cas assez répandus où malgré l'utilisation de gros modèles génériques préentraînés pour l'apprentissage par transfert, on ne dispose pas d'assez de données pour adapter le modèle pour une tâche spécifique.

Nous discuterons à la section 7.4 des défis que cela pose, particulièrement au niveau du partage en libre accès de ces gros modèles génériques préentraînés.

6.8 Limites du présent travail

La principale limite et critique de ce travail est que l'expérimentation n'a été réalisée que sur une seule tâche, de surcroît très simple de prédiction de la polarité d'un texte. Cela dit, l'approche a été validée sur un deuxième jeu de données.

Pour avoir une meilleure idée de la contribution effective des techniques d'amplification textuelle proposées, il faudrait répéter nos expériences avec d'autres jeux de données et pour d'autres tâches en variant les paramètres.

Rappelons que l'objectif était de prouver que les techniques d'amplification textuelle proposées pouvaient fonctionner. Beaucoup de travail reste à faire pour explorer chaque technique d'ADT de manière plus détaillée, faire varier les paramètres d'amplification et les combiner.

Par exemple, il serait intéressant de déterminer la quantité de variabilité optimale à ajouter à chaque nouvelle donnée textuelle. Cela exigerait de nombreuses expériences et des ressources de calcul importantes.

En nous concentrant sur la faisabilité, nous avons évité la mise au point optimale des modèles. La recherche d'une architecture optimale et le réglage fin des hyperparamètres[225] afférents peut s'avérer longue, fastidieuse et coûteuse en ressources de calcul. Il peut arriver que les modèles testés souffrent toujours de surajustement, mais l'amplification des données aura permis de le réduire.

[225] En anglais: «hyperparameters tuning»

Chapitre 7 - Conclusion et perspectives

Dans cette section qui se veut plus libre, nous tirerons des conclusions, puis lancerons des idées à la fois sur une nouvelle approche basée sur l'apprentissage par renforcement ou le paradigme «générer tester»[226], les limites de la Toile comme corpus en TLN, le potentiel des paracorpus, l'émergence d'un nouveau défi qui est le «mur des modèles préentrainés», des travaux futurs et des pistes de valorisation.

7.1 Conclusion

Pour des problèmes complexes, le fait de ne pas disposer d'assez de données constitue un obstacle majeur dans l'utilisation de l'apprentissage profond. C'est le *mur des données massives*.

Ce travail empirique, mené avec des ressources de calcul limitées, a permis de démontrer que l'utilisation de différentes techniques d'ADT est susceptible d'aider à franchir ce mur. Ces techniques sont l'injection de bruit textuel, l'injection de fautes d'orthographe, la substitution de mots et la génération de paraphrases au moyen de grammaires d'expressions régulières, la transformation d'arbres syntaxiques et la rétrotraduction

Les expériences avec ces techniques d'amplification textuelle ont permis d'accroître l'exactitude des résultats dans une fourchette 0.5 à 8.8% sur une tâche simple (classification binaire supervisée) normalisée de prédiction de la polarité de textes de critiques de film de la base IMDB.

Les techniques de transformation de surface (bruit lexical, fautes d'orthographe, expressions régulières) s'avèrent les plus performantes compte tenu de leur rapidité et de leur faible besoin en calcul. La substitution lexicale avec les vecteurs-mots AdaGram qui traitent la polysémie est meilleure que la substitution lexicale simple avec WordNet. La génération de paraphrases basée sur la rétrotraduction donne d'excellents résultats sur certains jeux de données et mériterait d'être approfondie.

Les réseaux profonds de neurones se sont montrés capables d'exploiter les données textuelles amplifiées, alors qu'un algorithme classique d'apprentissage automatique comme XGBoost peine à le faire.

7.2 Amplificateur de données textuelles par renforcement

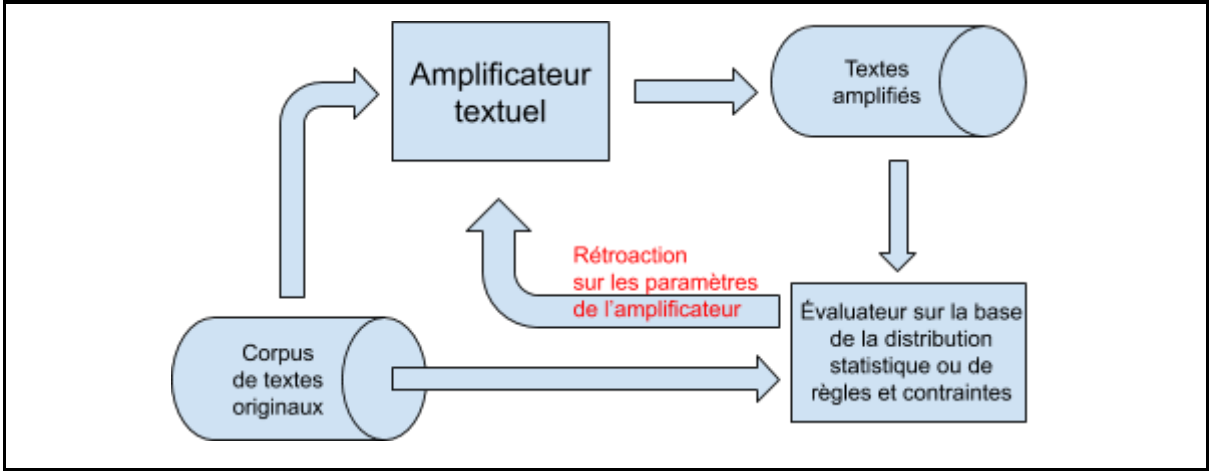
Une idée à développer serait d'introduire une boucle de rétroaction entre l'amplificateur textuel et les textes amplifiés afin de mieux contrôler l'amplification de données textuelles.

[226] En anglais: «generate and test»

La rétroaction serait de type apprentissage par renforcement[227] ou peut-être selon le vieux principe d'IA «générer-tester» qui agirait sur les paramètres de l'amplificateur / générateur de données textuelles.

On pourrait reprendre le concept du réseau antagoniste génératif [Goodfellow et al, 2014], mais sans la contrainte de la descente de gradient qui exige une fonction de coût continue et dérivable. Ici, on a pas besoin que les générateurs fonctionnent par descente de gradient. Un mécanisme de renforcement sur différents paramètres des algorithmes de génération pourrait suffire. Ou bien, on pourrait utiliser des gabarits et des contraintes selon le vieux paradigme «générer-tester».

Amplificateur textuel par renforcement et/ou «générer-tester» (fig. 7.1)



La fonction de comparaison des textes pourrait se baser sur les différences ou les similarités entre les textes originaux et les textes amplifiés. Par exemple sur la distribution statistique selon différents critères, ou avec des règles ou contraintes apprises ou codées manuellement. On pourrait aussi envisager de mettre des humains dans la boucle pour renforcer ou inhiber des comportements de l'amplificateur.

Rappelons en passant que l'usage de l'apprentissage par renforcement est en forte progression dans le traitement de la langue naturelle. Il s'avère intéressant de combiner l'apprentissage par renforcement et l'apprentissage en profondeur pour résoudre de nombreux problèmes touchant la linguistique. Par exemple, des chercheurs de l'UdeM ont utilisé l'apprentissage par renforcement pour créer un agent conversationnel[228], le MILABOT [Serban et al,, 2017].

[227] En anglais: «reinforcement learning»
[228] En anglais: «chatbots»

7.3 Les limites de l’usage de la Toile comme corpus en TLN

Un corpus est une collection de données linguistiques qui sont sélectionnées et organisées selon différents critères pour servir d'échantillon du langage. Dans notre étude ce sont des textes.

On distingue entre corpus de textes complets et corpus d'échantillons, entre corpus monolingues et corpus multilingues, entre corpus spécialisés et corpus de langue générale (ou corpus de référence), on distingue aussi entre langue écrite et langue parlée (i.e. des transcriptions), enfin on distingue parfois au niveau de la période de temps couverte.

Un corpus s'évalue selon trois critères : sa taille, sa diversité et ses annotations qui est le critère le plus important.

La Toile est considérée comme un fantastique corpus de textes. Les moteurs de recherche comme Google Search sont des outils efficaces pour chercher des exemples linguistiques pertinents dans cette grande quantité de textes.

En réalité, la Toile est loin d’être le corpus idéal. Certes sa taille est imposante (estimée à plus de 52 milliards de pages indexées [Van den Bosch, Bogers, & De Kunder, 2016]), mais il faut d’abord trouver le contenu pertinent. Beaucoup du contenu sur la Toile est insignifiant, redondant ou de qualité douteuse. Par exemple dans les réseaux sociaux où les utilisateurs expriment davantage leurs émotions que des faits objectifs. Aussi, beaucoup d’utilisateurs ne respectent pas la grammaire et l’orthographe dans les contenus qu’ils rédigent. Il y a aussi beaucoup de redondance, de redites et de copier-coller sur la Toile. Il s’agit d’une évidence partagée par des chercheurs de Google dans l’article «Unreasonable Effectiveness of Data» [Halevy, Norvig & Pereira, 2009].

Qualité des données moissonnées sur la Toile - Citation (fig. 7.2)
it’s taken from unfiltered Web pages and *thus contains incomplete sentences, spelling errors,* *grammatical errors, and all sorts of other errors.* [Halevy, Norvig & Pereira, 2009]

Ensuite, il faut extraire, nettoyer et transformer cette masse de données pour en tirer des données utiles. Le processus s’apparente beaucoup à celui de l’extraction minière d’où la métaphore du forage de données[229].

L’absence d’annotation est la principale carence de l’information trouvée sur la Toile. On peut certes faire de l’apprentissage non-supervisé ou de l’apprentissage semi-supervisé,

[229] En anglais: «data mining». Aussi en français: «forage des données», «fouille des données»

mais pour l'apprentissage supervisé qui est à la base de la plupart succès de l'IA, on fait appel à des services de production participative[230] pour annoter les données extraites.

Qu'en est-il de la diversité du corpus que représente la Toile? Mettons de côté le problème des langues peu informatisées déjà discuté à la section 1.1.2. En effet, seulement une poignée des 7000 langues parlées dans le monde disposent d'assez de corpus en format électronique pour entraîner de gros modèles à la base des nouvelles applications d'IA.

Plus fondamentalement, sur le plan linguistique, la diversité n'est pas toujours au rendez-vous. Une autre observation intéressante tirée de l'étude [Halevy, Norvig & Pereira, 2009] est qu'il est important de considérer les événement rares dans les corpus.

Problème des phénomènes linguistiques rares - Citations (fig. 7.3)
all the experimental evidence from the last decade suggests that throwing away rare events is almost always a bad idea, because much Web data consists of individually rare but collectively frequent events. [Halevy, Norvig & Pereira, 2009]
… even with a huge example set we are very likely to observe events that never occurred in the example set, and that are very different than all the examples that did occur in it. [Goldberg, 2017]

Plusieurs chercheurs dans le domaine du traitement des langues naturelles reconnaissent que la dilution des données[231] en langue naturelle représente un défi important. Ce problème de dilution des données a d'ailleurs donné naissance aux techniques de lissage[232] pour les modèles de langue qui consistent à ajouter un peu de probabilité aux observations rares pour éviter que leur probabilité ne soit de zéro [Xie et al, 2017].

Problème de la dilution des données en TLN - Citation (fig. 7.4)
*A key challenge when performing estimation in language modeling is the **data sparsity problem**: due to large vocabulary sizes and the exponential number of possible contexts, the majority of possible sequences are rarely or never observed, even for very short subsequences.* [Xie et al, 2017]

En fait cette dilution des données résulte directement de l'énorme productivité combinatoire de la langue naturelle. L'ensemble des productions possibles de la langue naturelle est infini quand on considère qu'il n'y a pas de limite à la longueur d'une phrase et aux différentes compositions des parties. Au pire, on s'épuise comme dans la lecture des interminables

[230] En anglais: «crowdsourcing»
[231] En anglais: «data sparsity», «sparseness»
[232] En anglais: «smoothing»

phrases de Proust. D'ailleurs, l'explosion combinatoire est un problème bien connu de ceux qui codent des logiciels d'analyse de la langue naturelle [Bourdon et al, 1998].

Ce qui cause la dispersion des données linguistiques (fig. 7.5)
Il est important de rappeler que la cause de la dispersion des données linguistiques est l'énorme productivité combinatoire de la langue naturelle.

À travers nos expériences d'ADT et une toute petite expérience qui consistait a générer des requêtes automatiquement au moteur de recherche Google Search à partir de petits paratextes, on découvre que tout corpus est nécessairement parcimonieux[233] , même l'immense Toile. Cela à cause de l'énorme productivité combinatoire de la langue naturelle.

Une petite expérience de recherche sur la Toile (fig. 7.6)

Soit un petit dictionnaire de variantes capable de générer 432 paraphrases différentes:
 	
`paraphrases_dict = {"1":["","a","the",],`
`"2":["","small","domestic","tiny","black","nice"],`
`"3":["cat","dog"],`
`"4":["eating","is eating","eats"],`
`"5":["","food","meat","pet food"]}`
 	
On ne retrouve sur la Toile, en utilisant l'IPA de Google Search, que 82 (soit 19%) des paraphrases de 2 à 7 mots générées aléatoirement par notre petit dictionnaire. Évidemment, les phrases les plus courtes (courts n-grammes) sont plus fréquemment trouvées.

Il en découle que tout corpus ne peut contenir qu'une faible partie de toutes les productions possibles de la langue naturelle. Les seules vraies limites sont les contraintes sémantiques et au-delà, l'imagination ou le hasard.

Dans son livre «L'instinct du langage», le célèbre linguiste Stephen Pinker, qui est né et a étudié à Montréal, estimait la productivité combinatoire du langage naturel à 10^{20} phrases au minimum sur la base d'un lexique de 10^5 mots [Pinker, 1995]. Il faudrait environ 2 milliards de phrases différentes sur chacune des pages de la Toile pour contenir un tel corpus.

Même le contenu textuel de la Toile en son entier est parcimonieux en comparaison de la production potentielle de la langue naturelle. Une grammaire générative est capable de produire exponentiellement plus d'exemples que ce qu'on retrouve dans un corpus. La combinatoire d'une grammaire est explosive. Tout le contenu textuel de la Toile n'est qu'une goutte d'eau dans l'océan des productions potentielles de la langue naturelle.

[233] En anglais: «parcimonious», «sparse». En français: dans le sens de frugal, modique, restreint, minime, voire négligeable, insignifiant, dérisoire. L'adjectif «parcimonieux» est le favori à cause de sa proximité avec les mots «parcimonious» et «sparse» en anglais.

Limites des corpus en langue naturelle (fig. 7.7)
En raison de l'énorme productivité combinatoire de la langue naturelle, tout corpus est nécessairement parcimonieux.
Tout le contenu textuel sur la Toile est un échantillon parcimonieux de la production potentielle des langues naturelles.

Les systèmes résultant d'une approche par apprentissage de bout-en-bout sont en principe capables de gérer les régularités et les formes qu'ils ont rencontrées suffisamment souvent dans les données d'apprentissage, mais pas au-delà. Par conséquent, un phénomène linguistique de basse fréquence sera difficile à identifier statistiquement.

7.4 Vers la création de paracorpus

Les limites bien réelles de la Toile-corpus, la déraisonnable efficacité des données et l'importance des événements rares militent pour la mise à l'essai de grammaires et de générateurs de contenus textuels. En effet, la langue est tellement vaste que l'on se retrouve la plupart du temps en situation de données parcimonieuses.

Une grammaire générative d'une langue, avec un ensemble fini de règles, a la capacité de générer un nombre exponentiel de phrases grammaticalement correctes et potentiellement un nombre infini si on ne limite pas la longueur des phrases.

Générer du Molière[234] (fig. 7.8)

Dans le «Bourgeois gentilhomme» de Molière, la phrase «Belle Marquise, vos beaux yeux, me font mourir d'amour.» peut s'écrire: «D'amour mourir me font, belle Marquise, vos beaux yeux.», «Vos yeux beaux d'amour me font, belle Marquise, mourir.», «Mourir vos beaux yeux, belle Marquise, d'amour me font.», «Me font vos yeux beaux mourir, belle Marquise, d'amour.», etc.

```
import random
import itertools
belles_marquises = list(itertools.permutations(["vos beaux yeux","me font",
"mourir","d'amour","belle marquise"])) + list(itertools.permutations(["vos yeux beaux","me font",
"mourir","d'amour","belle marquise"]))
random.shuffle(belles_marquises)
belles_marquises

[('me font', 'belle marquise', 'vos yeux beaux', 'mourir', "d'amour"),
 ("d'amour", 'vos beaux yeux', 'belle marquise', 'mourir', 'me font'),
 ('mourir', "d'amour", 'belle marquise', 'me font', 'vos yeux beaux'),
 ('belle marquise', 'me font', "d'amour", 'mourir', 'vos yeux beaux'),
 ('mourir', 'me font', 'vos yeux beaux', 'belle marquise', "d'amour"),
 ('vos yeux beaux', "d'amour", 'me font', 'belle marquise', 'mourir'),
```

[234] Note; Plusieurs des phrases générées ne sont pas syntaxiquement valides. Ce n'est qu'une illustration imparfaite de la productivité de la langue naturelle.

```
 ("d'amour", 'vos yeux beaux', 'mourir', 'belle marquise', 'me font'),
 ('mourir', 'vos beaux yeux', 'me font', 'belle marquise', "d'amour"),
 ('mourir', 'me font', 'vos beaux yeux', 'belle marquise', "d'amour"),
 ("d'amour", 'me font', 'mourir', 'vos yeux beaux', 'belle marquise'),
...
```

7.5 Le partage en libre accès des gros modèles génériques préentraînés

Nous entrons dans une économie des données. La richesse du monde n'est plus dans les gisements de pétrole mais dans les gisements de données.

Cela représente un énorme défi à la fois pour les communautés linguistiques minoritaires sur la Toile et les entreprises qui doivent rivaliser avec les géants du GAFAM (Google, Amazon, Facebook, Apple, Microsoft).

La disponibilité des grands corpus et des gros modèles préentraînés est au coeur de la démocratisation de l'intelligence artificielle. L'accès aux applications de l'intelligence artificielle au plus grand nombre est lié à l'existence de ressources linguistiques et de modèles préentrainés dans un maximum de langues possibles. Ces ressources peuvent contribuer au maintien et au développement de ces langues et des cultures qui y sont associées. Par exemple, pour créer des outils de traduction neuronale performants afin de traduire des documents dans les domaines comme la santé, l'agriculture, l'éducation, les politiques gouvernementales, les lois et réglements, qui concernant les populations appartenant à des groupes linguistiques minoritaires.

L'apprentissage par transfert qui réutilise de gros modèles génériques préentraînés représente à la fois un espoir et un nouvel obstacle. Pour le moment, ces modèles émergent tout juste des laboratoires d'entreprises privées.

7.4.1 Le mur des modèles préentrainés

Un nouveau mur, «le mur des modèles préentraînés», se pointe à l'horizon. Le mur des modèles préentraînés est plus difficilement franchissable que le seul mur des données massives car il combine à la fois la masse des données et la masse des calculs.

Il faut bien comprendre que la collecte, le traitement et surtout l'entraînement de modèles linguistiques sur des milliards de mots représente une tâche colossale. La mise au point de gros modèles préentraînés est hors de portée de quiconque ne dispose pas d'énormes corpus (milliards de mots) et d'importantes infrastructures de calcul. L'entraînement de ces très gros modèles peut prendre des jours, des semaines, voire parfois des mois sur des architectures de calcul distribuées équipés de processeurs graphiques.

Cette tâche colossale demande des moyens de calculs que seuls les géants du GAFAM et peut-être quelques grands laboratoires publics possèdent. Cela dit, les laboratoires publics n'ont pas investi le domaine. Dans une déclaration à l'été 2019, Yoshua Bengio indiquait que son laboratoire le MILA ne disposait pas des ressources pour mettre au point ces énormes modèles [SYNCED, 2019].

Pour le moment, la stratégie des géants du GAFAM est de donner accès gratuitement à certains modèles préentraînés à l'occasion d'évènement ponctuels comme les compétitions ouvertes de type ImageNet ou lors de la publication d'articles scientifiques. Cela a pour effet de soutenir la recherche mais de ralentir la création de ressources de haute qualité ouvertes et publiques nécessaires à la démocratisation de l'IA.

De plus, les gros modèles linguistiques préentraînés, à de rares exceptions près, n'existent que pour la langue anglaise. Les rares modèles multilingues sont plus pauvres et moins performants. L'importance de disposer de gros modèles génériques préentraînés pour un maximum de langues et distribués selon des licences libres est un enjeu stratégique pour toutes les communautés linguistiques minoritaires[235]. C'est également un moyen de lutter contre l'extinction massive des langues minoritaires et des cultures associées qui représente un appauvrissement de la culture mondiale.

À ce sujet, il faut souligner la controverse soulevée par le refus de OpenAI de rendre disponible le modèle génératif GPT-2 (Generative Pre-training Transformers) sous prétexte qu'il pouvait être utilisé par des personnes malveillantes pour produire des fausses nouvelles [Pilipiszyn, 2019]. GPT-2 est une version améliorée [Radford et al, 2019], de GPT (ou Transformers) [Radford et al, 2018].

GPT-2 a été entraîné sur un corpus énorme (40 gigaoctets de textes) pendant une semaine sur une architecture comportant 32 processeurs spécialisés dans les calculs matriciels (TPU: Tensor Processor Unit). On estime les coûts de calcul de la plus grosse version de GPT-2 qui comporte 1.5 milliards de paramètres à environ 56 000 dollars canadiens [HACKER NEWS, 2019]. Ce n'est que la pointe de l'iceberg, car on néglige tous les calculs ayant menés à la mise au point du modèle, au moins dix fois plus. On parle potentiellement de millions de dollars [SYNCED, 2019].

7.6 Futurs travaux

En terminant, traçons quelques pistes pour de futurs travaux. Par manque de ressources, ce travail de recherche a volontairement été limité à une preuve de concept afin de prouver que les techniques d'amplification textuelle proposées fonctionnaient.

[235] Note: Soulignons les contributions récentes de la jeune pousse Hugging Face fondée en 2016 par des informaticiens français dont la mission est de démocratiser les nouvelles techniques de TLN.

Pour avoir une meilleure idée de la contribution effective des techniques d'amplification textuelle proposées, il faudrait répéter ces expériences avec d'autres jeux de données et pour d'autres tâches.

Par exemple, il serait intéressant de mesurer et déterminer la variabilité optimale à ajouter à aux données textuelles amplifiées. Une première phase semble un travail essentiellement empirique qui demanderait des expériences systématiques et des ressources de calcul importantes. En parallèle, un travail plus théorique devrait s'amorcer avec probablement pour toile de fond la théorie de l'information.

Aussi, il faudrait mieux comprendre pourquoi les techniques d'ADT qui produisent des paraphrases très différentes des phrases de départ comme la transformation d'arbres syntaxiques et la rétrotraduction qui devraient contribuer à la reconnaissance de nouvelles formes dans les données et corriger des erreurs de classification semblent s'annuler plutôt que se combiner. On peut légitimement se demander si les paraphrases générées introduisent davantage de bruit que de signaux utiles (malédiction de la haute dimension). Pour mieux comprendre ce phénomène, il faudrait affiner et élargir les expériences, entre autres en expérimentant des techniques de réduction de la dimension des données comme l'ACP (analyse en composantes principales) ou en modifiant l'architecture des réseaux de neurones (introduire un étranglement comme pour un autoencodeur).

Beaucoup de travail reste à faire pour explorer individuellement chaque technique d'amplification de manière plus détaillée, faire varier les paramètres d'amplification et aussi les combiner. Dans le cas des techniques impliquant de l'injection de bruit textuel, il serait intéressant d'explorer les effets du passage d'un bruit faible à un bruit fort sur l'invariance sémantique. Entre autres, la génération de paraphrases par la transformations d'arbres syntaxique mériterait à être étendue avec une grammaire plus importante qui couvre davantage de cas. Enfin, la rétrotraduction, pourtant très prometteuse, a été à peine effleurée. On ignore pratiquement tout des meilleures pratiques, des meilleurs couples de langue et des techniques de sélection et de filtrage des paraphrases. Des travaux intéressants dans cette direction ont été publiés récemment dans l'article «Understanding Back-Translation at Scale» par une équipe réunissant des chercheurs de Facebook et Google [Edunov & al, 2018].

Aussi, il faudrait mieux préciser la longueur de la chaîne de transformations optimale (le jeu du téléphone) et les conditions de l'application de multiples transformations successives. Une autre piste intéressante serait d'approfondir les différences observées entre les algorithmes d'apprentissage classiques et les réseaux profonds de neurones dans leur capacité à exploiter les données amplifiées.

Peu importe, il semble bien que la poursuite de ces travaux exigera l'accès à une importante infrastructure de calcul de type infonuagique (cloud) équipés de processeurs graphiques et d'une capacité de stockage afférente.

7.7 Valorisation et applications

Sans disposer d'une étude de marché, il semble y avoir des possibilités commerciales du côté de services en ligne pour l'amplification de données textuelles..

Le créneau le plus intéressant, du moins à court terme, serait l'amplification de données textuelles pour faciliter l'entraînement de robots conversationnels[236]. Entre autres parce que les techniques d'amplification textuelle sont plus performantes sur de courtes phrases. En plus de l'amplification textuelle plus conventionnelle, on pourrait générer des bouts de dialogue avec des hésitations, interruptions, (Euh, Uhm, voyez-vous, ok,), des redites ou répétitions (je voudrais savoir, j'aimerais connaître, le prix, le coût) et des corrections, précisions, (italiens, non mexicains) [Eshghi, Shalyminov & Lemon,2017].

Un projet plus ambitieux et plus risqué serait de créer une entreprise dont la mission serait d'aider à préserver les langues peu informatisées[237]. Par exemple, créer des corpus, des modèles et des outils pour les langues des peuples autochtones ou aborigènes. L'idée serait de combiner de gros modèles préentraînés et l'amplification textuelle en exploitant l'apprentissage par transfert et de propriétés interlinguistiques [Fadaee, Bisazza & Monz, 2017]. Les progrès récents de la reconnaissance de la parole pourraient probablement aider à mettre à contribution des corpus oraux.

La détection du plagiat, particulièrement en langue française, et dans l'optique de dépister d'éventuels emprunts interlinguistiques serait également un créneau d'application à évaluer. Par souci d'éthique, l'offre de ces produits devrait être orientée vers les institutions qui luttent contre le plagiat et non vers ceux qui veulent rendre leur plagiat indétectable.

Plus globalement, le traitement de la langue française pourrait s'avérer une niche intéressante. Un projet mobilisateur serait la mise en place de corpus, vecteurs-mots, et modèles préentraînés en langue française[238]. Idéalement ces modèles devraient être disponibles selon des licence libres.

---

[236] En anglais: «chatbots»

[237] En anglais: «low-resource language», «resource-poor language». En français, on trouve aussi: «langue moins dotée», «langue moins informatisée».

[238] Note: Deux modèles de langue en français inspirés de BERT sont apparus à la fin de 2019, CamemBERT le 10 novembre 2019 et FlauBERT le 12 décembre 2019.

Bibliographie

[Altay, 2018] Altay, G. (2018). Word Vectors from Decomposing a Word-Word Pointwise Mutual Information Matrix. En Python. Sur GitHub. En ligne: http://bit.ly/2OfIvum

[AMAZON, 2018] (2018). Amazon Comprehend - Natural Language Processing (NLP) and Machine Learning (ML). En ligne: https://aws.amazon.com/comprehend/

[Androutsopoulos & Malakasiotis, 2010] Androutsopoulos, I., & Malakasiotis, P. (2010). A survey of paraphrasing and textual entailment methods. Journal of Artificial Intelligence Research, 135–187. En ligne: https://arxiv.org/pdf/0912.3747.pdf

[Arora et al, 2018] Arora, S., Ge, R., Neyshabur, B., & Zhang, Y. (2018). Stronger generalization bounds for deep nets via a compression approach. arXiv preprint arXiv:1802.05296. En ligne: https://arxiv.org/pdf/1802.05296.pdf

[Bahdanau, Cho & Bengio, 2014] Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. ArXiv Preprint ArXiv:1409.0473. En ligne: http://arxiv.org/pdf/1409.0473.pdf

[Banko & Brill, 2001] Banko, M., & Brill, E. (2001). Scaling to very very large corpora for natural language disambiguation. In Proceedings of the 39th annual meeting on association for computational linguistics (p. 26–33). Association for Computational Linguistics. En ligne:http://www.aclweb.org/anthology/P01-1005

[Bär, Zesch & Gurevych, 2015] Bär, D., Zesch, T., & Gurevych, I. (2015). Composing Measures for Computing Text Similarity. En ligne: http://bit.ly/2O7dBEi

[Baroni et al, 2009] Baroni, M., Bernardini, S., Ferraresi, A., & Zanchetta, E. (2009). The WaCky wide web: a collection of very large linguistically processed web-crawled corpora. Language Resources and Evaluation, 43 (3), 209–226. En ligne: https://goo.gl/5K1bcZ

[Bartunov et al, 2016] Bartunov, S., Kondrashkin, D., Osokin, A., & Vetrov, D. (2016). Breaking sticks and ambiguities with adaptive skip-gram. In Artificial Intelligence and Statistics (pp. 130–138). En ligne: http://proceedings.mlr.press/v51/bartunov16.pdf

[Bartunov, 2015] Bartunov, S. (2015). Adaptive Skip-gram implementation in Julia. En Julia, Sur GitHub. En ligne: https://github.com/sbos/AdaGram.jl

[Batista, 2017] Batista, D. S. (2017, 25 mars). Google's SyntaxNet in Python NLTK. En ligne: http://www.davidsbatista.net/blog/2017/03/25/syntaxnet/

[Belia at al, 2005)] Belia, S., Fidler, F., Williams, J., & Cumming, G. (2005). Researchers Misunderstand Confidence Intervals and Standard Error Bars. Psychological Methods, 10(4), 389–396. En ligne: http://bit.ly/2T5Tqg1

[Bengio & LeCun, 2007] Bengio, Y., & LeCun, Y. (2007). Scaling learning algorithms towards AI. Large-Scale Kernel Machines, 34(5). En ligne: http://bit.ly/2x9wC1H

[Bengio et al, 2003] Bengio, Y., Ducharme, R., Vincent, P., & Jauvin, C. (2003). A neural probabilistic language model. Journal of machine learning research, 3(Feb), 1137−1155. En ligne: http://www.jmlr.org/papers/volume3/bengio03a/bengio03a.pdf

[Bird, Klein & Loper, 2009] Bird, S., Klein, E., & Loper, E. (2009). Natural language processing with Python. O'Reilly Media, Inc. En ligne: http://www.nltk.org/book

[Bittlingmayer, 2018] Bittlingmayer, A. (2018). NoiseMix: data generation for natural language, En Python, Sur GitHub. En ligne: https://github.com/noisemix/noisemix

[Bojanowski et al, 2017] Bojanowski, P., Grave, E., Joulin, A., & Mikolov, T. (2017). Enriching word vectors with subword information. Transactions of the Association for Computational Linguistics, 5, 135–146. En ligne: https://arxiv.org/pdf/1607.04606.pdf

[Bourdon et al, 1998] Bourdon, M., Da Sylva, L., Gagnon, M., Kharrat, A., Knoll, S., & Maclachlan, A. (1998). A case study in implementing dependency-based grammars. Processing of Dependency-Based Grammars. En ligne: http://bit.ly/2HpxBSj

[Bouthillier et al, 2015] Bouthillier, X., Konda, K., Vincent, P., & Memisevic, R. (2015). Dropout as data augmentation. arXiv preprint arXiv:1506.08700. En ligne: https://arxiv.org/pdf/1506.08700.pdf

[Bowman et al, 2015] Bowman, S. R., Vilnis, L., Vinyals, O., Dai, A. M., Jozefowicz, R., & Bengio, S. (2015). Generating sentences from a continuous space. arXiv preprint arXiv:1511.06349. En ligne: https://arxiv.org/pdf/1511.06349.pdf

[Brown, 1982] Brown, G. W. (1982). Standard deviation, standard error: Which'standard'should we use? American Journal of Diseases of Children, 136(10), 937–941. En ligne: http://bit.ly/2YBRQl6

[Brownlee, 2018a] Brownlee, J. (2018). Deep Learning With Python: Develop Deep Learning Models on Theano and TensorFlow using Keras (v1.14). En ligne: http://bit.ly/2HAm8hS

[Brownlee, 2018b] Brownlee, J. (2018). Deep Learning for Natural Language Processing: Develop Deep Learning Models Natural Language Processing in Python (v1.2). En ligne: http://bit.ly/2HAm8hS

[Brownlee, 2018c] Brownlee, J. (2018). Long Short-Term Memory Networks With Python: Develop Sequence Prediction Models With Deep Learning (v1.4). En ligne: http://bit.ly/2HAm8hS

[Brownlee, 2018d] Brownlee, J. (2018). XGBoost With Python: Gradient Boosted Trees With XGBoost and scikit-learn (v1.10). En ligne: http://bit.ly/2HAm8hS

[Bryson & Ho, 1969] Bryson, A. E., & Ho, Y.-C. (1969). Applied optimal control. 1969. Blaisdell, Waltham, Mass, 8, 72.

[Carpineto & Romano, 2012] Carpineto, C., & Romano, G. (2012). A survey of automatic query expansion in information retrieval. ACM Computing Surveys (CSUR), 44(1), 1. En ligne: http://bit.ly/2Qoyw6w

[Casili, 2019] Casilli, A. A. (2019). En attendant les robots: Enquête sur le travail du clic. Seuil. http://bit.ly/2NtCTMF, http://bit.ly/2H38fss

[Chen, Benesty & He, 2018] Chen T., Benesty M., He T (2018). Understand Your Dataset with Xgboost, Sur cran.r-project.org, The Comprehensive R Archive Network, En ligne: http://bit.ly/36wd9b1

[Chen & Guestrin, 2016] Chen, T., & Guestrin, C. (2016). Xgboost: A scalable tree boosting system. In Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining (pp. 785–794). ACM. En ligne: http://bit.ly/2T1HHyF

[Chen, 2012] Chen, E. (2012, March 20). Infinite Mixture Models with Nonparametric Bayes and the Dirichlet Process. En ligne: http://bit.ly/2TeHSCa

[Chen & Dolan, 2011] Chen, D. L., & Dolan, W. B. (2011). Collecting highly parallel data for paraphrase evaluation. In Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1 (p. 190–200). Association for Computational Linguistics. En ligne: http://bit.ly/2Fiowly

[Cho et al, 2014b] Cho, K., Van Merriënboer, B., Bahdanau, D., & Bengio, Y. (2014). On the properties of neural machine translation: Encoder-decoder approaches. arXiv preprint arXiv:1409.1259. En ligne: https://arxiv.org/pdf/1409.1259.pdf

[Cho et al, 2014a] Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation. arXiv preprint arXiv:1406.1078. En ligne: https://arxiv.org/pdf/1406.1078.pdf

[Chollet, 2017] Chollet, F. (2017). Deep learning with python. Manning Publications Co. En ligne: https://www.manning.com/books/deep-learning-with-python

[Chollet, 2016] Chollet, F. (2016, juin 5). Building powerful image classification models using very little data. En ligne: http://bit.ly/2HEZBQT

[Chollet, 2015] Chollet, F. (2015). Keras. En ligne: https://keras.io

[Chung et al, 2014] Chung, J., Gulcehre, C., Cho, K., & Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. arXiv preprint arXiv:1412.3555. En ligne: https://arxiv.org/pdf/1412.3555

[Cornel, 2004] Cornel. (2004). *Data - sentiment polarity dataset v2.0*. En ligne: http://www.cs.cornell.edu/people/pabo/movie-review-data/review_polarity.tar.gz

[Coulombe, 2019] Coulombe, C. (8 mars 2019). Coding a simple Stick-Breaking Process in Python. En ligne: http://bit.ly/2NRQmhB

[Coulombe, 2018b] Coulombe, C. (2018). *Open source contribution to the Jupyter notebooks for the code samples of the book «Deep Learning with Python»*. Jupyter Notebook en Python, Sur GitHub. En ligne: http://bit.ly/2JulOzt

[Coulombe, 2018a] Coulombe, C. (avril, 2018). A French Lemmatizer in Python based on LEFFF, a large-scale morphological and syntactic lexicon for French.: FrenchLefffLemmatizer. En Python. Sur GitHub. En ligne: http://bit.ly/2HDISxB

[Coulombe, 2017d] Coulombe C. (22 novembre 2017). How to get a list of antonyms lemmas using Python, NLTK and WordNet?, Questions / Solutions, Sur StackOverflow. En ligne: http://bit.ly/2FfzWN3

[Coulombe, 2017c] Coulombe C. (17 août 2017). Calling a Julia 0.4 function from Python. How i can make it work?, Solutions, Sur StackOverflow. En ligne: http://bit.ly/2JdMYiu

[Coulombe, 2017b] Coulombe C. (18 juillet 2017). AdaGram (adaptive skip-gram) for Python. Commentaires. Sur GitHub, En ligne: http://bit.ly/2EOecXs

[Coulombe, 2017a] Coulombe C. (2017). Adaptive Skip-gram implementation in Julia. Commentaires. Sur GitHub, En ligne: http://bit.ly/2VR1C0s

[Coulombe, Paquette & Mezghani, 2016] Coulombe, C., Paquette, G., & Mezghani, N. (2016). Improving MOOCs' Perseverance and Completion Rates Using Best Practice Design Principles. En ligne: http://www.iiis.org/CDs2016/CD2016Summer/papers/EA251SA.pdf

[Dahl et al, 2012] Dahl, G. E., Yu, D., Deng, L., & Acero, A. (2012). Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition. IEEE Transactions on audio, speech, and language processing, 20(1), 30–42. En ligne: http://bit.ly/2CJwI5v

[Daniely, 2018] Daniely, A. (2018, juin). PAC Analysis of Deep Learning Algorithms. STOC 2018 Deep Learning Workshop, Duke University, North-Carolina, USA. En ligne: http://bit.ly/2FrAPlj

[Dean & Ng, 2012] Dean, J., & Ng, A. (26 juin 2012). Using large-scale brain simulations for machine learning and A.I. En ligne: http://bit.ly/2O938Z4

[DEEPLEARNING4J, 2016] Word2Vec, En ligne: https://deeplearning4j.org/word2vec.html

[Deerwester et al, 1990] Deerwester, S., Dumais, S. T., Furnas, G. W., Landauer, T. K., & Harshman, R. (1990). Indexing by latent semantic analysis. Journal of the American society for information science, 41(6), 391−407. En ligne: http://bit.ly/2GAXkrp

[Deng et al,2009] Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., & Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on (p. 248−255). Ieee. En ligne: http://bit.ly/2N7VTTX

[Devlin et al, 2018] Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2018). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv preprint arXiv:1810.04805. En ligne: https://arxiv.org/pdf/1810.04805.pdf

[Do & Batzoglou, 2008] Do, C. B., & Batzoglou, S. (2008). What is the expectation maximization algorithm? Nature Biotechnology, 26(8), 897. En ligne: http://bit.ly/2VMjo4K

[DOCKER, 2016] (s. d.). Code informatique, *tensorflow/syntaxnet - Docker Hub*. En ligne: https://hub.docker.com/r/tensorflow/syntaxnet/

[Dolan et al, 2004] Dolan, B., Quirk, C., & Brockett, C. (2004). Unsupervised construction of large paraphrase corpora: Exploiting massively parallel news sources. In Proceedings of the 20th international conference on Computational Linguistics (p. 350). Association for Computational Linguistics. En ligne: http://www.aclweb.org/anthology/C04-1051.pdf

[Doll, Drouin, & Coulombe, 2005] Doll, F., Drouin, P., & Coulombe, C. (2005). Intégration d'un analyseur syntaxique à large couverture dans un outil de langage contrôlé en français. *Linguisticae investigationes: Revue internationale de linguistique française et de linguistique générale*, *28*(1), 19−36.

[Domingos, 2015] Domingos, P. (2015). The master algorithm: How the quest for the ultimate learning machine will remake our world. Basic Books. En ligne: http://bit.ly/2Cl49bO

[Dos Santos & Gatti, 2014] Dos Santos, C., & Gatti, M. (2014). Deep convolutional neural networks for sentiment analysis of short texts. In Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers (p. 69–78). En ligne: http://www.aclweb.org/anthology/C14-1008

[Edunov & al, 2018] Edunov, S., Ott, M., Auli, M., & Grangier, D. (2018). Understanding back-translation at scale. *arXiv preprint arXiv:1808.09381*. En ligne: http://bit.ly/2XYfTKF

[Eshghi, Shalyminov & Lemon,2017] Eshghi, A., Shalyminov, I., & Lemon, O. (2017). Bootstrapping incremental dialogue systems from minimal data: the generalisation power of dialogue grammars. arXiv preprint arXiv:1709.07858. En ligne: http://bit.ly/2ucEMV7

[Fadaee, Bisazza & Monz, 2017] Fadaee, M., Bisazza, A., & Monz, C. (2017). Data augmentation for low-resource neural machine translation. *arXiv preprint arXiv:1705.00440*. En ligne: https://arxiv.org/pdf/1705.00440.pdf

[Finn et al, 2018] Finn, C., Xu, K., & Levine, S. (2018). Probabilistic model-agnostic meta-learning. Advances in Neural Information Processing Systems, 9516–9527. En ligne: http://bit.ly/2N5DyUC

[Firth, 1957] Firth, J. R. (1957). A synopsis of linguistic theory, 1930-1955. Studies in Linguistic Analysis.

[Frank, 2017] Frank, B. H. (2017, octobre 23). Google Brain chief: Deep learning takes at least 100,000 examples. En ligne: http://bit.ly/2FgHUb7

[Franz & Thorsten, 2006]. Franz, A., & Thorsten, B. (2006, août 3). All Our N-gram are Belong to You. En ligne: http://bit.ly/2TXT6iG

[Gagnon & Da Sylva, 2005] Gagnon, M., & Da Sylva, L. (2005). Text summarization by sentence extraction and syntactic pruning. En ligne: http://bit.ly/2MpVCXg

[Genette, 1997] Genette, G. (1997). Paratexts: Thresholds of interpretation (Vol. 20). Cambridge University Press.

[Géron, 2017a] Géron, A. (2017). Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems. O'Reilly Media, Inc. En ligne: http://shop.oreilly.com/product/0636920052289.do

[Géron, 2017b] Géron, A. (2017). Deep Learning avec TensorFlow: Mise en oeuvre et cas concrets. Dunod. En ligne: http://bit.ly/2QqH9O3

[Gleize, 2016] Gleize, M. (2016). Textual Inference for Machine Comprehension. Université Paris-Saclay. En ligne: https://tel.archives-ouvertes.fr/tel-01317577/document

[Glorot, Bordes & Bengio, 2011] Glorot, X., Bordes, A., & Bengio, Y. (2011). Deep sparse rectifier neural networks. In Proceedings of the fourteenth international conference on artificial intelligence and statistics (p. 315–323). En ligne: http://bit.ly/2Fih3JC

[Goldberg & Levy, 2014] Goldberg, Y., & Levy, O. (2014). word2vec Explained: deriving Mikolov et al.'s negative-sampling word-embedding method. arXiv Preprint arXiv:1402.3722. En ligne: https://arxiv.org/pdf/1402.3722.pdf

[Goldberg, 2017] Goldberg, Y. (2017). Neural Network Methods for Natural Language Processing (Morgan & Claypool Publishers). Coll. Synthesis Lectures on Human Language Technologies, Editor: Graeme Hirst, University of Toronto. En ligne: http://bit.ly/2ObWUrp

[Golub & Reinsch, 1970] Golub, G. H., & Reinsch, C. (1970). Singular value decomposition and least squares solutions. Numerische mathematik, 14(5), 403–420. En ligne: http://bit.ly/2BpkyuI

[Goodfellow, 2016] Goodfellow, I., Reddit MachineLearning - Generative Adversarial Networks for Text. En ligne: http://bit.ly/2ES5Elq

[Goodfellow, Bengio & Courville, 2016] Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep learning. MIT Press. En ligne: http://www.deeplearningbook.org/

[Goodfellow et al, 2014] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., … Bengio, Y. (2014). Generative adversarial nets. In Advances in neural information processing systems (p. 2672–2680). En ligne: http://bit.ly/2HDliRs

[GOOGLE, 2018a] Google (2018). SyntaxNet Models and examples built with TensorFlow. En Python. Sur GitHub. (Travail original publié en 2016) En ligne: http://bit.ly/2Y4AtZD

[GOOGLE, 2018b] Google (2018). Services de cloud computing, d'hébergement et d'API de Google. En ligne: https://cloud.google.com/gcp/?hl=fr

[GOOGLE, 2018c] Google (2018). Cloud Natural Language API. En ligne: https://cloud.google.com/natural-language/?hl=fr

[GOOGLE, 2018d] Google (2018). API Cloud Translation – Traduction instantanée | Translation API. En ligne: https://cloud.google.com/translate/?hl=fr

[GOOGLE, 2018e] BERT - multilingua model. Sur GitHub. En ligne: https://github.com/google-research/bert/blob/master/multilingual.md

[GOOGLE, 2017] Google (2017). Models and examples built with TensorFlow. En Python, tensorflow. Sur GitHub. En ligne: https://github.com/tensorflow/models

[Griewank, 2012] Griewank, A. (2012). Who invented the reverse mode of differentiation. Documenta Mathematica, Extra Volume ISMP, 389–400.

[Ha & Bunke, 1997] Ha, T. M., & Bunke, H. (1997). Off-line, handwritten numeral recognition by perturbation method. IEEE Transactions on Pattern Analysis & Machine Intelligence, (5), 535–539.

[HACKER NEWS, 2019] Ask HN: How much would it cost to replicate GPT-2's model ? | Hacker News. (s. d.). En ligne: https://news.ycombinator.com/item?id=19402666

[Halevy, Norvig & Pereira, 2009] Halevy, A., Norvig, P., & Pereira, F. (2009). The unreasonable effectiveness of data. IEEE Intelligent Systems, 24(2), 8–12. En ligne: http://bit.ly/2VavRio

[Harris, 1954] Harris, Z. S. (1954). Distributional structure. Word, 10(2–3), 146–162. En ligne: http://www.tandfonline.com/doi/pdf/10.1080/00437956.1954.11659520

[Hastie, Tibshirani & Friedman, 2008] Hastie, T., & Tibshirani, R. (s. d.). & Friedman, J.(2008). The Elements of Statistical Learning; Data Mining, Inference and Prediction. Springer, New York. En ligne: https://stanford.io/37Mo5l9

[Hinton, Osindero & Teh., 2006] Hinton, G., Osindero, S., & Teh, Y.-W. (2006). A fast learning algorithm for deep belief nets. Neural Computation, 18 (7), 1527–1554. En ligne http://www.cs.toronto.edu/~hinton/absps/fastnc.pdf

[Hinton, 1984] Hinton, G. E. (1984). Distributed representations.

[Honnibal & Montani, 2017] Honnibal, M., & Montani, I. (2017). spaCy 2: Natural language understanding with bloom embeddings, convolutional neural networks and incremental parsing. To appear. En ligne: https://spacy.io/

[Hou et al, 2018] Hou, Y., Liu, Y., Che, W., & Liu, T. (2018). Sequence-to-Sequence Data Augmentation for Dialogue Language Understanding. arXiv preprint arXiv:1807.01554. En ligne: https://arxiv.org/pdf/1807.01554.pdf

[Howard & Ruder, 2018] Howard, J., & Ruder, S. (2018). Universal language model fine-tuning for text classification. In Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers) (Vol. 1, p. 328–339). En ligne: http://www.aclweb.org/anthology/P18-1031

[Hu et al, 2017] Hu, Z., Yang, Z., Liang, X., Salakhutdinov, R., & Xing, E. P. (2017). Toward controlled generation of text. arXiv preprint arXiv:1703.00955. En ligne: https://arxiv.org/pdf/1703.00955.pdf

[Karpathy, 2015] Karpathy, A. (2015, May 21). The Unreasonable Effectiveness of Recurrent Neural Networks. Andrej Karpathy blog website: En ligne: http://bit.ly/2FWUnPp

[Lyyer et al, 2018] Lyyer, M., Wieting, J., Gimpel, K., & Zettlemoyer, L. (2018). Adversarial example generation with syntactically controlled paraphrase networks. *arXiv preprint arXiv:1804.06059*. En ligne: https://arxiv.org/pdf/1804.06059.pdf

[IMT, 2017] IMT: Institut de Mathématiques, U. P. S. (s. d.). Explicabilité des décisions algorithmiques. En ligne: http://bit.ly/2CdfNGH

[Ioffe & Szegedy, 2015] Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. ArXiv Preprint ArXiv:1502.03167. En ligne: https://arxiv.org/pdf/1502.03167.pdf

[Jaitly & Hinton, 2013] Jaitly, N., & Hinton, G. E. (2013). Vocal tract length perturbation (VTLP) improves speech recognition. In Proc. ICML Workshop on Deep Learning for Audio, Speech and Language (Vol. 117). En ligne: http://bit.ly/2CopqBE

[James et al, 2013] James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). *An introduction to statistical learning* (Vol. 112). Springer. En ligne: http://bit.ly/2F939s1

[Johnson et al., 2017] Johnson, M., Schuster, M., Le, Q. V., Krikun, M., Wu, Y., Chen, Z., Thorat N., Viégas F., Wattenberg M., Corrado G., Hughes M., Dean J. (2017). Google's Multilingual Neural Machine Translation System: Enabling Zero-Shot Translation. *ArXiv Preprint ArXiv:1611.04558*. En ligne: https://arxiv.org/pdf/1611.04558.pdf

[Jurafsky & Martin, 2018] Jurafsky, D., & Martin, J. (2018). Speech & language processing (Third Edition draft). Pearson Education India. En ligne: https://stanford.io/2HoGmff

[Ko et al, 2015] Ko, T., Peddinti, V., Povey, D., & Khudanpur, S. (2015). Audio augmentation for speech recognition. In Sixteenth Annual Conference of the International Speech Communication Association. En ligne: http://bit.ly/2W3OTHR

[Kong et al, 2017] Kong, L., Alberti, C., Andor, D., Bogatyy, I., & Weiss, D. (2017). Dragnn: A transition-based framework for dynamically connected neural networks. arXiv Preprint arXiv:1703.04474. En ligne: https://arxiv.org/pdf/1703.04474.pdf

[Kobayashi, 2018] Kobayashi, S. (2018). Contextual Augmentation: Data Augmentation by Words with Paradigmatic Relations. arXiv preprint arXiv:1805.06201. En ligne: https://arxiv.org/pdf/1805.06201.pdf

[Krizhevsky, Sutskever & Hinton, 2012] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems (p. 1097–1105). En ligne: http://bit.ly/2UFB6qJ

[Lan & Xu, 2018] Lan, W., & Xu, W. (2018). Neural Network Models for Paraphrase Identification, Semantic Textual Similarity, Natural Language Inference, and Question Answering. arXiv preprint arXiv:1806.04330. En ligne: https://arxiv.org/pdf/1806.04330.pdf

[Latour, 2011] Latour, J. (s. d.). How many synonyms are there on an average for words in English language? - Quora. En ligne: http://bit.ly/2uc7gyg

[Lau, Clark & Lappin, 2015] Lau, J. H., Clark, A., & Lappin, S. (2015). Predicting Acceptability Judgements with Unsupervised Language Models. In ISCOL 2015. The Open University of Israel. En ligne: http://bit.ly/2TJ9y7r

[LeCun, Bengio & Hinton, 2015] LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. Nature, 521(7553), 436–444. En ligne: http://bit.ly/2NBZby9

[LeCun, 1985] LeCun, Y. (1985). Une Procedure d'apprentissage pour réseau à seuil assymétrique, Proceedings of Cognitiva 85, Paris, 1985.

[Levy, Goldberg & Dagan, 2015] Levy, O., Goldberg, Y., & Dagan, I. (2015). Improving distributional similarity with lessons learned from word embeddings. Transactions of the Association for Computational Linguistics, 3, 211-225. En ligne: http://bit.ly/2Y06VfQ

[Levy & Goldberg, 2014] Levy, O., & Goldberg, Y. (2014). Dependency-Based Word Embeddings. In ACL (2) (pp. 302–308). En ligne: http://bit.ly/2QjnVKh

[Li & Li, 2018] Li, J., & Li, F.-F. (2018). Cloud AutoML: Making AI accessible to every business. En ligne: http://bit.ly/2CiAvEX

[Li et al, 2015] Li, J., Galley, M., Brockett, C., Gao, J., & Dolan, B. (2015). A diversity-promoting objective function for neural conversation models. arXiv Preprint arXiv:1510.03055. En ligne: https://arxiv.org/pdf/1510.03055.pdf

[Lopuhin, 2017] Lopuhin, K. (2017). AdaGram (adaptive skip-gram) for Python. En Python. Sur Github, En ligne: https://github.com/lopuhin/python-adagram

[Lyyer et al, 2018] Lyyer, M., Wieting, J., Gimpel, K., & Zettlemoyer, L. (2018). Adversarial example generation with syntactically controlled paraphrase networks. arXiv preprint arXiv:1804.06059. En ligne: https://arxiv.org/pdf/1804.06059.pdf

[Madnani & Dorr, 2010] Madnani, N., & Dorr, B. J. (2010). Generating phrasal and sentential paraphrases: A survey of data-driven methods. Computational Linguistics, 36(3), 341–387. En ligne: http://www.mitpressjournals.org/doi/pdf/10.1162/coli_a_00002

[Manning & Schütze, 1999] Manning, C. D., & Schütze, H. (1999). Foundations of statistical natural language processing. MIT press. En ligne: http://bit.ly/2TscbcU

[McKinney, 2010] McKinney, W. (2010). Data structures for statistical computing in Python. In Proceedings of the 9th Python in Science Conference (Vol. 445, pp. 51–56). En ligne: http://204.236.236.243/proceedings/scipy2010/pdfs/mckinney.pdf

[Mel'cuk, 1988b] Mel'cuk, I. A., Mel'čuk, I. A., & Mel'čuk, I. A. (1988). Dependency syntax: theory and practice. SUNY press.

[Mel'cuk, 1988a] Mel'cuk, I. (1988). Paraphrase et lexique dans la théorie linguistique Sens-Texte in Lexique et paraphrase. Lexique, (6), 13–54.

[Melamud, Levy & Dagan, 2015] Melamud, O., Levy, & O., Dagan, I., I. (2015). A simple word embedding model for lexical substitution. In Proceedings of the 1st Workshop on Vector Space Modeling for Natural Language Processing (pp. 1–7). En ligne: http://bit.ly/2p8RZfn

[Merity, 2016] Merity, S. (2016). In deep learning, architecture engineering is the new feature engineering. En ligne: http://bit.ly/2CpWxVp

[MICROSOFT, 2018], (2018) Cognitive Services Directory | Microsoft Azure. En ligne: :https://azure.microsoft.com/en-ca/services/cognitive-services/directory/lang/

[Mikolov et al, 2013a] Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. arXiv Preprint arXiv:1301.3781. En ligne: https://arxiv.org/pdf/1301.3781.pdf

[Mikolov et al, 2013b] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In Advances in neural information processing systems (pp. 3111–3119). En ligne: http://bit.ly/2TExkRM

[Miller et al, 1990] Miller, G. A., Beckwith, R., Fellbaum, C., Gross, D., & Miller, K. J. (1990). Introduction to Wordnet: An on-line lexical database. International journal of lexicography, 3(4), 235–244. En ligne: http://www.cfilt.iitb.ac.in/archives/english_wordnet_5papers.pdf

[Mitchell, 1997] Mitchell, T. M. (1997). Machine Learning. Boston, USA: McGraw-Hill. En ligne: http://www.cs.cmu.edu/~tom/mlbook.html

[MODELZOO, 2018] ModelZoo (2018). Model Zoo - Deep learning code and pretrained models for transfer learning, educational purposes, and more. En ligne: https://modelzoo.co/

[Mohammad et al, 2013] Mohammad, S. M., Dorr, B. J., Hirst, G., & Turney, P. D. (2013). Computing lexical contrast. Computational Linguistics, 39(3), 555–590. En ligne: https://www.mitpressjournals.org/doi/pdf/10.1162/COLI_a_00143

[Moody, 2017] Moody, C. (2017, October 18). Stop Using word2vec, Billet de blogue. En ligne: https://multithreaded.stitchfix.com/blog/2017/10/18/stop-using-word2vec/

[Motulsky, 1995] Motulsky, H. (1995). The link between error bars and statistical significance. En ligne: http://bit.ly/2GOSXrG

[MOZILLA FOUNDATION, 2016] Mozilla Foundation. (2016). A TensorFlow implementation of Baidu's DeepSpeech architecture: mozilla/DeepSpeech. C++, Mozilla. En ligne: https://github.com/mozilla/DeepSpeech

[Mukherjee, 2017] Mukherjee, U. (2017). How to handle Imbalanced Classification Problems in machine learning? En ligne: http://bit.ly/2CoBfrf

[Navigli & Ponzetto, 2012] Navigli, R., & Ponzetto, S. P. (2012). BabelNet: The Automatic Construction, Evaluation and Application of a Wide-Coverage Multilingual Semantic Network. Artificial Intelligence, 193, 217–250. En ligne: http://bit.ly/2VZu7sK

[Ng, 2017] Ng, A. (2017). Machine Learning Yearning. En ligne: http://www.mlyearning.org, http://bit.ly/2W0PJ9o

[Ng, 2016] Ng, A. (2016). Generative Learning Algorithms. In CS229 Lecture notes. En ligne: http://cs229.stanford.edu/notes/cs229-notes2.pdf

[Ng & Jordan 2002] Ng, A. Y., & Jordan, M. I. (2002). On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. In Advances in neural information processing systems (p. 841–848). En ligne: https://arxiv.org/pdf/1709.02349.pdf

[Nielsen, 2015] Nielsen, M. A. (2015). Neural Networks and Deep Learning. Determination Press. En ligne: http://neuralnetworksanddeeplearning.com

[OXFORD DICTIONARIES, 2018] (s. d.). Common misspellings | Oxford Dictionaries. En ligne: https://en.oxforddictionaries.com/spelling/common-misspellings

[Pang & Lee 2004] Pang, B., & Lee, L. (2004). A sentimental education: Sentiment analysis using subjectivity summarization based on minimum cuts. In Proceedings of the 42nd annual meeting on Association for Computational Linguistics (p. 271). Association for Computational Linguistics. En ligne: https://arxiv.org/pdf/cs/0409058.pdf

[Pang, Lee & Vaithyanathan, 2002] Pang, B., Lee, L., & Vaithyanathan, S. (2002). Thumbs up?: sentiment classification using machine learning techniques. In Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10 (p. 79–86). Association for Computational Linguistics. En ligne: https://arxiv.org/pdf/cs/0205070.pdf

[Park, 2010] Park, E. (2010, décembre 24). The Book-Length Sentence - Essay. The New York Times. En ligne: https://www.nytimes.com/2010/12/26/books/review/Park-t.html

[Patki, Wedge & Veeramachaneni, 2016] Patki, N., Wedge, R., & Veeramachaneni, K. (2016). The synthetic data vault. In *Data Science and Advanced Analytics (DSAA), 2016 IEEE International Conference on* (p. 399–410). IEEE. En ligne: http://bit.ly/2FgUcxP

[Pedregosa et al, 2011] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., … Dubourg, V. (2011). Scikit-learn: Machine learning in Python. The Journal of Machine Learning Research, 12, 2825–2830. En ligne: http://arxiv.org/pdf/1201.0490.pdf

[Pennington, Socher, Manning, 2014] Pennington, J., Socher, R., & Manning, C. D. (2014). Glove: Global Vectors for Word Representation. In EMNLP (Vol. 14, pp. 1532–43). En ligne: http://nlp.stanford.edu/pubs/glove.pdf

[Peters et al, 2018] Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., & Zettlemoyer, L. (2018). Deep contextualized word representations. arXiv preprint arXiv:1802.05365. En ligne: https://arxiv.org/pdf/1802.05365.pdf

[Petrov, 2016] Petrov, S. (2016). Announcing syntaxnet: The world's most accurate parser goes open source. Google Research Blog. En ligne: http://bit.ly/2UHgEWa

[Piantadosi, 2018] Piantadosi, S. T. (2018). One parameter is always enough. *AIP Advances*, *8*(9), 095118. En ligne: https://aip.scitation.org/doi/pdf/10.1063/1.5031956?class=pdf

[Pilipiszyn, 2019] Pilipiszyn, A. (2019, 14 février). Better Language Models and Their Implications. OpenAI Blog. En ligne: http://bit.ly/2UACv5b

[Pinker, 1995] Pinker, S. (1995). The language instinct: The new science of language and mind (Vol. 7529). Penguin UK.

[Pimienta, 2017] Pimienta, D. (2017). Une approche alternative à la mesure des langues dans l'Internet et recommandations pour la réappropriation du thème par les chercheurs (p. 49). Observatoire des langues et culture dans l'Internet. En ligne: http://bit.ly/2TR0nRC

[Poddutur, 2018] Poddutur, S. (2018). Syntaxnet Parsey McParseface wrapper for POS tagging and dependency parsing: spoddutur/syntaxnet. Python. En ligne: https://github.com/spoddutur/syntaxnet

[Prabhumoye et al., 2018] Prabhumoye, S., Tsvetkov, Y., Salakhutdinov, R., & Black, A. W. (2018). Style Transfer Through Back-Translation. *arXiv preprint arXiv:1804.09000*. En ligne: https://arxiv.org/pdf/1804.09000.pdf

[Radford et al, 2019] Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., & Sutskever, I. (2019). Language models are unsupervised multitask learners. Technical report, OpenAi. En ligne:: http://bit.ly/2VdXd7W

[Radford et al, 2018] Radford, A., Narasimhan, K., Salimans, T., & Sutskever, I. (2018). Improving language understanding by generative pre-training. En ligne: http://bit.ly/2p28V6U

[Rajeswar et al, 2017] Rajeswar, S., Subramanian, S., Dutil, F., Pal, C., & Courville, A. (2017). Adversarial generation of natural language. arXiv preprint arXiv:1705.10929. En ligne: https://arxiv.org/pdf/1705.10929.pdf

[Ramos, 2017] Ramos, H. (2017, mars 28). Google Stop Words: The Definitive Guide (+ List of SEO Stop Words 2018). En ligne: https://cseo.com/blog/google-stop-words/

[Raschka & Mirjalili, 2017] Raschka, S., & Mirjalili, V. (2017). Python machine learning (2nd edition). Packt Publishing Ltd. E ligne: http://bit.ly/2NCAO3u

[Raschka, 2015] Raschka, S. (2015). Python machine learning. Packt Publishing Ltd. En ligne: http://bit.ly/2NJaUbV

[Rascu & Schmidt, 2005] Carl, M., Rascu, E., & Schmidt, P. (2005). Using template grammars for shake & bake paraphrasing. In Proceedings of eamt. En ligne: http://www.aclweb.org/anthology/W05-1605

[Ravi & Larochelle, 2016] Ravi, S., & Larochelle, H. (2016). Optimization as a model for few-shot learning. Consulté à l'adresse http://bit.ly/2pD1EhA

[Řehůřek & Sojka, 2010] Řehůřek, R., & Sojka, P. (2010). Software Framework for Topic Modelling with Large Corpora. In Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks (p. 45–50). Valletta, Malta: ELRA. En ligne: https://radimrehurek.com/gensim/lrec2010_final.pd

[Roquette, 2018] Roquette S. (2018). *Spelling corrector project based on Deep Learning. On GitHub*. Langage Python sur GitHub. En ligne: https://github.com/simonroquette/CORAP

[Rosario, 2017] Rosario, R. R. (2017). A Data Augmentation Approach to Short Text Classification. Thèse , UCLA. En ligne: http://bit.ly/2FbXp0m

[Ruder, 2018] Ruder, S. (2018, juillet 8). NLP's ImageNet moment has arrived. En ligne: https://thegradient.pub/nlp-imagenet/

[Ruder, 2019] Ruder, S. (2019, 18). The State of Transfer Learning in NLP [Blogue]. Consulté 23 octobre 2019. En ligne: https://ruder.io/state-of-transfer-learning-in-nlp/

[Rumelhart, Hinton & Williams, 1985] Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1985). Learning internal representations by error propagation. California Univ San Diego La Jolla Inst for Cognitive Science. En ligne: https://apps.dtic.mil/dtic/tr/fulltext/u2/a164453.pdf

[Salameh et al, 2015] Salameh, M., Mohammad, S., & Kiritchenko, S. (2015). Sentiment after translation : A case-study on arabic social media posts. Proceedings of the 2015 conference of the North American chapter of the association for computational linguistics: Human language technologies, 767−777. En ligne: http://bit.ly/2MzgKO3

[Salehinejad et al, 2018] Salehinejad, H., Valaee, S., Dowdell, T., Colak, E., & Barfett, J. (2018). Generalization of deep neural networks for chest pathology classification in x-rays using generative adversarial networks. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (p. 990−994). IEEE. En ligne: http://bit.ly/2UHgw9c

[Sanchez Perez, 2018] Sanchez Perez, M. A. (2018). Plagiarism Detection Through Paraphrase Recognition (PhD Computer Science). Instituto Politecnico Nacional, Mexico. En ligne: http://bit.ly/2x4OJ99

[Scheible, Walde & Springorum, 2013] Scheible, S., Im Walde, S. S., & Springorum, S. (2013). Uncovering Distributional Differences between Synonyms and Antonyms in a Word Space Model. In IJCNLP (p. 489−497). En ligne: http://bit.ly/2HzEFuO

[Schmidhuber, 1992] Schmidhuber, J. (1992). Learning complex, extended sequences using the principle of history compression. Neural Computation, 4(2), 234−242. En ligne: https://mediatum.ub.tum.de/doc/814767/file.pdf

[SCIKIT-LEARN, Underfitting vs. Overfitting] Underfitting vs. Overfitting — scikit-learn 0.15-git documentation. (s. d.). En ligne: http://bit.ly/2u9vucD

[Serban et al,, 2017] Serban, I. V., Sankar, C., Germain, M., Zhang, S., Lin, Z., Subramanian, S., … Ke, N. R. (2017). A deep reinforcement learning chatbot. arXiv preprint arXiv:1709.02349. En ligne: https://arxiv.org/pdf/1709.02349.pdf

[Shabda, 2009] Shabda AGILIQ (2009). Generating pseudo random text with Markov chains using Python. En Python. En ligne: http://bit.ly/2FYgwNd

[Shermis & Burstein, 2013] Shermis, M. D., & Burstein, J. (2013). Handbook of automated essay evaluation: Current applications and new directions. New York, USA: Routledge.

[Shwartz-Ziv & Tishby, 2017] Shwartz-Ziv, R., & Tishby, N. (2017). Opening the black box of deep neural networks via information. arXiv preprint arXiv:1703.00810. En ligne: https://arxiv.org/pdf/1703.00810.pdf

[Sichel, 1974] Sichel, H. S. (1974). On a distribution representing sentence-length in written prose. Journal of the Royal Statistical Society. Series A (General), 25–34. En ligne: http://bit.ly/2xkJSR3

[Simard, Steinkraus & Platt, 2003] Simard, P. Y., Steinkraus, D., & Platt, J. C. (2003). Best practices for convolutional neural networks applied to visual document analysis. In 12th International Conference on Document Analysis and Recognition (Vol. 2, p. 958–958). IEEE Computer Society. En ligne: *http://bit.ly/2DP3xLp*

[Sutskever, 2013] Sutskever, I. (2013). *Neural Networks for Machine Perception*. Vimeo - Meetup SF Bay Area Machine Learning. En ligne: https://vimeo.com/77050653

[SYNCED, 2019] Synced. (2019, juin 27). The Staggering Cost of Training SOTA AI Models, billet de blogue. En ligne: http://bit.ly/2oeRenY

[TEXYGEN, 2018] Texygen. (s. d.). *A text generation benchmarking platform. Contribute to geek-ai/Texygen development by creating an account on GitHub*. (2018). Python, Geek.AI Organization. En ligne: https://github.com/geek-ai/Texygen

[Torrey & Shavlik, 2009] Torrey, L., & Shavlik, J. (2009). Transfer learning. Handbook of Research on Machine Learning Applications and Trends: Algorithms, Methods, and Techniques, 1, 242. En ligne: http://bit.ly/2ObXPIn

[Valiant, 1984] Valiant, L. G. (1984). A theory of the learnable. Proceedings of the sixteenth annual ACM symposium on Theory of computing, 436–445. En ligne: http://bit.ly/2tkGuql

[Van den Bosch, Bogers, & De Kunder, 2016] Van den Bosch, A., Bogers, T., & De Kunder, M. (2016). Estimating search engine index size variability: a 9-year longitudinal study. Scientometrics, 107(2), 839–856. En ligne: http://bit.ly/2Xj3elb, site web associé: https://www.worldwidewebsize.com/

[Van der Velde, van der Voort van der Kleij & de Kamps, 2004] van der Velde, F., van der Voort van der Kleij, G. T., & de Kamps, M. (2004). Lack of combinatorial productivity in language processing with simple recurrent networks. Connection Science, 16(1), 21–46. En ligne: https://mediatum.ub.tum.de/doc/1292326/file.pdf

[Van Dyk & Meng, 2001] Van Dyk, D. A., & Meng, X.-L. (2001). The art of data augmentation. Journal of Computational and Graphical Statistics, 10(1). En ligne: http://bit.ly/2Ob8DXh

[Vapnik, 1999] Vapnik, V. (1999). The nature of statistical learning theory. First edition, Springer science & business media.

[Vila, Martí & Rodríguez, 2014] Vila, M., Martí, M. A., & Rodríguez, H. (2014). Is this a paraphrase? What kind? Paraphrase boundaries and typology. Open Journal of Modern Linguistics, 4(01), 205. En ligne: http://file.scirp.org/pdf/OJML_2014031817224144.pdf

[W3CTECHS, 2018] W3CTechs. (s. d.). Historical yearly trends in the usage of conten languages, Novembre 2018. En ligne: http://bit.ly/2FipwfT

[Wang et al, 2018] Wang, X., Pham, H., Dai, Z., & Neubig, G. (2018). Switchout: an efficient data augmentation algorithm for neural machine translation. *arXiv preprint arXiv:1808.07512*. En ligne: https://arxiv.org/pdf/1808.07512.pdf

[Wang & Yang, 2015] Wang, W. Y., & Yang, D. (2015). That's So Annoying!!!: A Lexical and Frame-Semantic Embedding Based Data Augmentation Approach to Automatic Categorization of Annoying Behaviors using# petpeeve Tweets. In Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing (p. 2557–2563). En ligne: http://www.aclweb.org/anthology/D15-1306

[Weiss, Khoshgoftaar & Wang, 2016] Weiss, K., Khoshgoftaar, T. M., & Wang, D. (2016). A survey of transfer learning. Journal of Big Data, 3(1), 9. En ligne: http://bit.ly/2x2v3Th

[Werbos, 1974] Werbos, P. (1974). Beyond Regression:" New Tools for Prediction and Analysis in the Behavioral Sciences. Ph. D. dissertation, Harvard University.

[Wieting, Mallinson & Gimpel, 2017] Wieting, J., Mallinson, J., & Gimpel, K. (2017). Learning paraphrastic sentence embeddings from back-translated bitext. *arXiv preprint arXiv:1706.01847*. En ligne: https://arxiv.org/pdf/1706.01847.pdf

[WIKIPÉDIA, Beta distribution] Wikipédia. (s. d.). Beta distribution. Dans Wikipédia, l'encyclopédie libre. En ligne: https://en.wikipedia.org/wiki/Beta_distribution

[WIKIPÉDIA, Bootstrap] Wikipédia. (s. d.). Bootstrap (statistiques). Dans Wikipédia, l'encyclopédie libre. En ligne: https://fr.wikipedia.org/wiki/Bootstrap_(statistiques)

[WIKIPÉDIA,Cosine similarity] Wikipédia. (s. d.). Cosine similarity. Dans Wikipédia, l'encyclopédie libre. En ligne: https://en.wikipedia.org/wiki/Cosine_similarity

[WIKIPÉDIA, Error bar] Wikipédia. (s. d.). Error bar. Dans Wikipédia, l'encyclopédie libre. En ligne: https://en.wikipedia.org/wiki/Error_bar

[WIKIPÉDIA, Igor Mel'čuk] Wikipédia. (s.d.). Igor Mel'čuk. Dans *Wikipédia,* l'encyclopédie libre. En ligne: http://bit.ly/2QdLSpd

[WIKIPÉDIA, Multicollinearity] Wikipédia. (s.d.). Multicollinearity. Dans *Wikipédia,* l'encyclopédie libre. En ligne: https://en.wikipedia.org/wiki/Multicollinearity

[WIKIPÉDIA, Mutual information] Wikipédia. (s.d.). Mutual information. Dans *Wikipédia,* l'encyclopédie libre. En ligne: https://en.wikipedia.org/wiki/Mutual_information

[WIKIPÉDIA, Plagiarism detection] Wikipédia. (s. d.). Plagiarism detection. Dans Wikipédia, l'encyclopédie libre. En ligne: https://en.wikipedia.org/wiki/Plagiarism_detection#Software

[WIKIPÉDIA, Pointwise Mutual information] Wikipédia. (s.d.). Pointwise Mutual information. Dans *Wikipédia,* l'encyclopédie libre. En ligne: http://bit.ly/2F7QW6X

[WIKIPÉDIA, Regular expression] Wikipédia. (s.d.). Regular expression. Dans *Wikipédia,* l'encyclopédie libre. En ligne: http://bit.ly/32a4A35

[WIKIPÉDIA, Shapiro-Wilk's test] Wikipédia. (s. d.). Shapiro-Wilk's test. Dans Wikipédia, l'encyclopédie libre. En ligne: https://en.wikipedia.org/wiki/Shapiro%E2%80%93Wilk_test

[WIKIPÉDIA, Standard error] Wikipédia. (s. d.). Standard error. Dans Wikipédia, l'encyclopédie libre. En ligne:https://en.wikipedia.org/wiki/Standard_error

[WIKIPÉDIA, Student's t-test] Wikipédia. (s. d.). Student's *t*-test. Dans Wikipédia, l'encyclopédie libre. En ligne: https://en.wikipedia.org/wiki/Student%27s_t-test

[WIKIPÉDIA, tf-idf] Wikipédia. (s. d.). tf-idf. Dans Wikipédia, l'encyclopédie libre. En ligne: https://en.wikipedia.org/wiki/Tf%E2%80%93idf

[WIKIPÉDIA, Variance] Wikipédia. (s. d.). Variance. Dans Wikipédia, l'encyclopédie libre. En ligne: https://en.wikipedia.org/wiki/Variance

[WIKIPÉDIA,Welch's t-test] Wikipédia. (s. d.). Welch's t-test. Dans Wikipédia, l'encyclopédie libre. En ligne: https://en.wikipedia.org/wiki/Welch%27s_t-test

[Wolpert & Macready, 1997] Wolpert, D. H., & Macready, W. G. (1997). No free lunch theorems for optimization. IEEE transactions on evolutionary computation, 1(1), 67–82. En ligne: https://ti.arc.nasa.gov/m/profile/dhw/papers/78.pdf

[Xie et al, 2017] Xie, Z., Wang, S. I., Li, J., Lévy, D., Nie, A., Jurafsky, D., & Ng, A. Y. (2017). Data noising as smoothing in neural network language models. arXiv preprint arXiv:1703.02573. En ligne: https://arxiv.org/pdf/1703.02573.pdf

[Yaeger, Lyon, & Webb, 1997] Yaeger, L. S., Lyon, R. F., & Webb, B. J. (1997). Effective training of a neural network character classifier for word recognition. In Advances in neural information processing systems (p. 807–816). En ligne: http://bit.ly/2EhzhdE

[Yang et al, 2019] Yang, Z., Dai, Z., Yang, Y., Carbonell, J., Salakhutdinov, R., & Le, Q. V. (2019). XLNet : Generalized Autoregressive Pretraining for Language Understanding. arXiv:1906.08237 [cs]. En ligne: http://arxiv.org/abs/1906.08237

[Yosinski et al, 2014] Yosinski, J., Clune, J., Bengio, Y., & Lipson, H. (2014). How transferable are features in deep neural networks? In Advances in neural information processing systems (pp. 3320–3328). En ligne: https://arxiv.org/pdf/1411.1792.pdf

[Yu et al, 2017] Yu, L., Zhang, W., Wang, J., & Yu, Y. (2017). SeqGAN: Sequence Generative Adversarial Nets with Policy Gradient. In AAAI (p. 2852–2858). En ligne: http://www.aaai.org/ocs/index.php/AAAI/AAAI17/paper/download/14344/14489

[Zhang et al, 2017] Zhang, Y., Gan, Z., Fan, K., Chen, Z., Henao, R., Shen, D., & Carin, L. (2017). Adversarial feature matching for text generation. *arXiv preprint arXiv:1706.03850*. En ligne: https://arxiv.org/pdf/1706.03850.pdf

[Zhang & Wang, 2016] Zhang, X., & Wang, H. (2016). A Joint Model of Intent Determination and Slot Filling for Spoken Language Understanding. In IJCAI (pp. 2993–2999). En ligne: http://bit.ly/2X9emRl

[Zhang, et al,, 2016] Zhang, C., Bengio, S., Hardt, M., Recht, B., & Vinyals, O. (2016). Understanding deep learning requires rethinking generalization. arXiv preprint arXiv:1611.03530. En ligne: https://arxiv.org/pdf/1611.03530.pdf

[Zhang & LeCun, 2015] Zhang, X., & LeCun, Y. (2015). Text Understanding from Scratch. ArXiv Preprint ArXiv:1502.01710. En ligne: http://arxiv.org/pdf/1502.01710.pdf

[Zhu et al, 2018] Zhu, Y., Lu, S., Zheng, L., Guo, J., Zhang, W., Wang, J., & Yu, Y. (2018). Texygen: A Benchmarking Platform for Text Generation Models. *arXiv preprint arXiv:1802.01886*. En ligne: https://arxiv.org/pdf/1802.01886.pdf

[Zhu, 2005] Zhu, X. (2005). Semi-supervised learning literature survey. En ligne: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.99.9681&rep=rep1&type=pdf

[Zouaq, Gagnon & Ozell, 2010] Zouaq, A., Gagnon, M., & Ozell, B. (2010). Semantic analysis using dependency-based grammars and upper-level ontologies. International Journal of Computational Linguistics and Applications, 1(1-2), 85–101. En ligne: http://site.cicling.org/2010/IJCLA-2010.pdf#page=85

Annexe 1 - Qu’est-ce qu’un algorithme d’apprentissage?

L’apprentissage automatique consiste à programmer un algorithme capable d’apprendre par lui-même, à partir de données sans avoir à coder explicitement son comportement.

Algorithme d’apprentissage automatique - définition plus formelle (fig. A1.1)
Soient une tâche T et une mesure de performance P, on dit qu’un algorithme apprend à partir d’une expérience E, si les résultats obtenus sur T, mesurés par P, s’améliorent avec l’expérience E. Tom Mitchell [Mitchell, 1997]

Un algorithme d’apprentissage automatique comporte quatre parties[239]: données, modèle, fonction de coût et algorithme d’optimisation.

Les composantes d’un algorithme d’apprentissage (fig. A1.2)
1. Des données 2. Un modèle 3. Une fonction de coût[240] dérivable 4. Un algorithme d’optimisation

Le cycle de vie d’un modèle statistique comporte deux phases: l’entraînement et l’exécution.

Cycle de vie d’un modèle statistique (fig. A1.3)
1. L’algorithme d’apprentissage calcule un modèle statistique à partir de données, c’est la phase d’entraînement du modèle. 2. Une fois entraîné, le modèle est appliqué sur de nouvelles données, c’est la phase d’exécution (ou d’inférence) du modèle.

Un algorithme d’optimisation mathématique (ou optimiseur) est utilisée dans l'apprentissage automatique pendant l’entraînement d’un modèle sur les données. L’algorithme cherche à

[239] Note: Cette recette s'applique aussi bien à un algorithme d'apprentissage supervisé où les données sont des couples (attributs, étiquette) qu’à un algorithme d'apprentissage non-supervisé où on aura uniquement des attributs sans étiquette. Bien entendu, il faut un modèle et une fonction de coût appropriés.

[240] En anglais: «cost function», «loss function», «error function», «objective function». En français: «fonction de coût», «fonction de perte», «fonction d’erreur» ou «fonction objective.» Souvent en apprentissage automatique, on utilisera les termes erreur, coût, perte et objectif d'une manière interchangeable.

minimiser le coût des erreurs (i.e. la fonction de coût) entre les prédictions du modèle et chacune des données d'entraînement.

Grossièrement, entraîner consiste à optimiser en calculant itérativement la valeur des paramètres qui minimisent l'erreur commise par un modèle sur un jeu de données d'entraînement. L'entraînement consiste à réduire progressivement les erreurs que fait l'algorithme.

Le plus souvent, l'algorithme d'optimisation sera une variante de la descente de gradient. Par exemple l'optimiseur Adam. L'algorithme d'optimisation modifie petit à petit (itérativement) les paramètres dans le but de minimiser la fonction de coût.

Dans le cas de l'apprentissage profond, le modèle comporte l'architecture du réseau de neurones, ses paramètres (ou poids) et ses hyperparamètres. La fonction de coût dépend du type de sortie. Par exemple on utilise l'entropie croisée[241], qui mesure l'écart entre deux distributions de probabilités, pour une classification binaire.

Dans la pratique, le choix d'un algorithme d'apprentissage dépend du jeu de données et de la tâche à accomplir.

Inspiré de la section 5.10 du livre «Deep Learning» [Goodfellow, Bengio & Courville, 2016]

[241] En anglais: «binary crossentropy», «binary cross-entropy»

Annexe 2 - Biais et variance en apprentissage automatique

A2.1 Définition

Il y a deux grandes sources d'erreurs en apprentissage automatique: le biais[242] et la variance[243].

Le jeu de dards (fléchettes) permet de bien conceptualiser le biais et la variance [Domingos, 2015]. La distance moyenne qui sépare les fléchettes du centre de la cible représente le biais. Si les fléchettes sont près de la centre de la cible alors le biais est faible, mais lorsque les fléchettes sont éloignées du centre alors le biais augmente. La dispersion des fléchettes à la surface de la cible correspond à la variance. Si les fléchettes sont regroupées alors la variance est faible. Par contre si les fléchettes sont dispersées alors la variance est élevée.

Illustration des concepts de biais-variance (fig. A2.1)

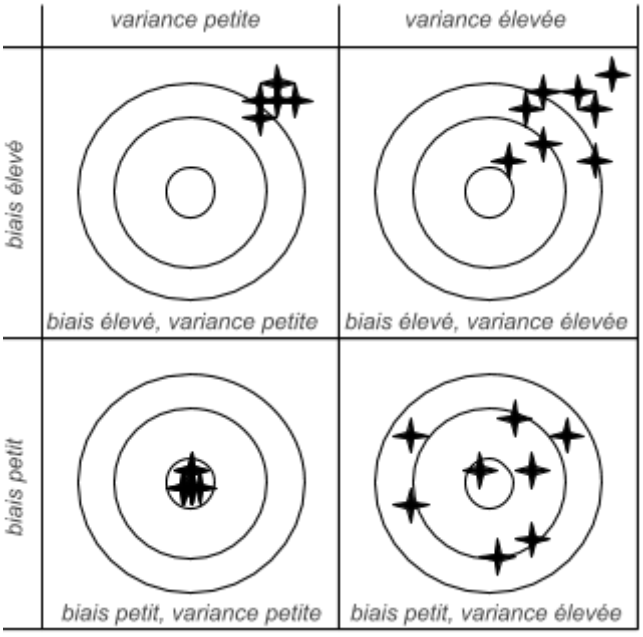


Diagramme inspiré de [Domingos, 2015]

[242] En anglais: «bias»
[243] En anglais: «variance»

A2.2 Compromis biais-variance

L'idée du compromis biais-variance[244] est de trouver un équilibre entre l'ajustement aux détails des données (variance élevée) et la simplicité du modèle (biais élevé) de manière à maximiser la généralisation. Un modèle avec un biais élevé aura tendance à faire des prévisions incorrectes alors qu'un modèle avec une variance élevée fera des prévisions erratiques ou plus variables.

Le principe du rasoir d'Occam, largement utilisé en science, nous enseigne de préférer le modèle le plus simple possible capable d'expliquer les observations. La courbe de l'erreur de généralisation (ou erreur de test) fait un «U» caractéristique et le point d'ajustement optimal correspond au creux de ce «U».

A2.3 Biais-variance et entraînement d'un modèle statistique

Ci-dessous un diagramme qui explique sommairement la procédure d'entraînement d'un modèle statistique sur la base de la mesure de l'erreur d'entraînement (erreur sur les données d'entraînement) et de l'erreur de validation[245] (erreur sur les données de validation) [Ng, 2017].

Procédure d'entraînement d'un modèle (fig. A2.2)

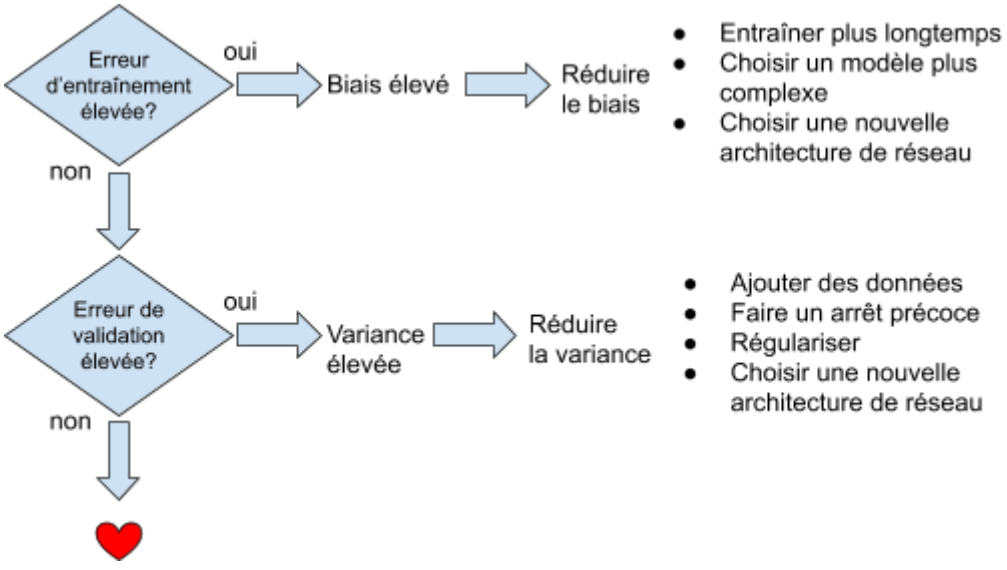


Diagramme inspiré de [Ng, 2017]

[244] En anglais: «bias-variance tradeoff»
[245] Note: En apprentissage automatique les expressions «erreur de validation», «erreur de généralisation» et «erreur de test» peuvent être considérées comme des synonymes.

On commence par s'occuper du biais. Dans le cas d'une erreur élevée sur les données d'entraînement (erreur d'entraînement élevée) ce qui traduit un biais élevé, on a le choix entre créer un modèle plus complexe, changer d'architecture ou entraîner plus longtemps le modèle.

Par contre, si l'erreur d'entraînement est modérée, on s'intéresse alors à la variance (l'ajustement) du modèle. Si l'erreur de validation / généralisation / test est élevée, on se trouve typiquement dans une situation de surajustement que l'on cherchera à corriger en augmentant la quantité de données, en introduisant de la régularisation ou en choisissant une nouvelle architecture moins complexe.

Ci-dessous un diagramme qui explique la relation entre la capacité / complexité d'un modèle statistique, les erreurs d'entraînement et les erreurs de validation (généralisation, test).

Relation entre la capacité / complexité d'un modèle et l'erreur (fig. A2.3)

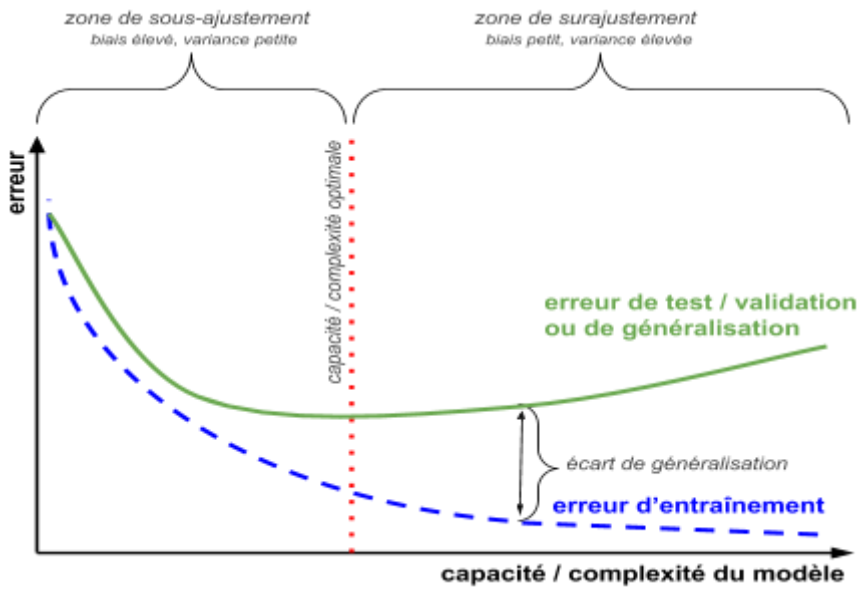


Diagramme inspiré de [Goodfellow, Bengio & Courville, 2016]

On corrigera le sous-ajustement en choisissant un modèle plus complexe ou en réduisant la régularisation. Il faut un modèle suffisamment complexe pour obtenir une erreur d'entraînement faible. Ici, l'ajout de données supplémentaires ne corrigera pas le problème de sous-ajustement.

Augmenter la complexité du modèle réduit l'erreur de validation (ou généralisation, test) jusqu'à un point qui correspond à la complexité optimale au delà duquel nous ajoutons simplement des détails inutiles (du bruit). L'erreur d'entraînement continuera à diminuer avec la complexité grandissante, mais l'erreur de validation (généralisation, test) commencera à augmenter. On entre alors en régime de surajustement.

Pour obtenir un modèle optimal, il faut trouver un bon équilibre entre le biais et la variance de façon à minimiser l'erreur totale.

A2.4 Différentes tendances observées pour un modèle statistique

Le diagramme ci-dessous résume les différentes tendances que l'on peut observer lors de l'entraînement d'un modèle statistique.

En gros on voit que le modèle optimal auquel on aspire se retrouve quelque part entre un modèle sous-optimal (trop simple) où le biais est élevé et un modèle sur-optimal (trop complexe) où la variance est élevée et qui présente des signes de surajustement.

Tendances observées pour un modèle statistique (fig. A2.4)		
modèle trop simple (peu de paramètres)	modèle optimal (assez de paramètres)	modèle trop complexe (beaucoup de paramètres)
biais élevé (peu sensible aux signaux faibles dans les données)	équilibre biais-variance	variance élevée (sensible à des signaux faibles dans les données)
sous-ajustement (ajustement grossier aux données)	ajustement optimal aux données	surajustement (ajustement détaillé aux données)
erreur élevée sur les données d'entraînement	erreur petite sur les données d'entraînement	erreur petite sur les données d'entraînement
erreur élevée sur les données de test (erreur de généralisation / validation élevée)	erreur petite sur les données de test (faible erreur de généralisation)	erreur élevée sur les données de test (erreur de généralisation/ validation élevée)

A2.5 Décomposition de l'erreur

L'erreur de généralisation, c'est à dire l'erreur sur des données nouvelles, d'un modèle est la somme de trois erreurs: le biais qui dépend du choix de modèle, mathématiquement de la famille de fonctions (tendance au sous-ajustement), la variance qui est la sensibilité à de petites variations dans les données d'entraînement (tendance au surajustement) et le bruit, typiquement des erreurs dans les données, des anomalies et des données aberrantes.

Erreur de généralisation = Biais + Variance + Bruit (fig. A2.5)

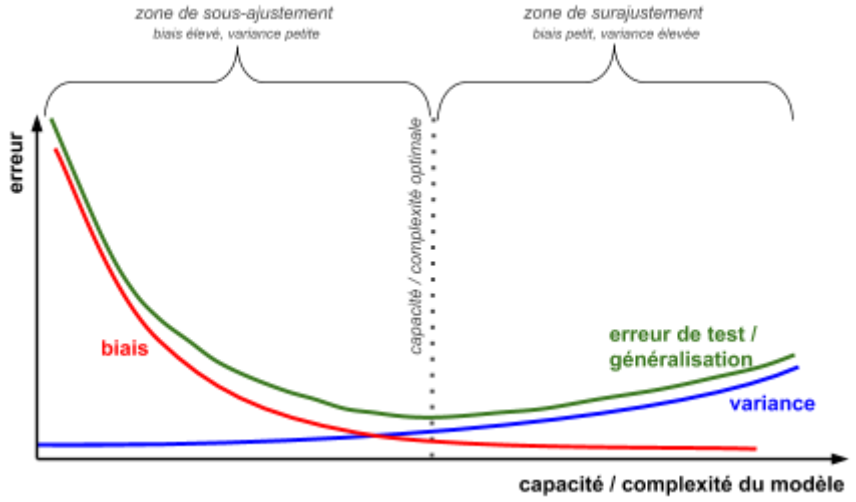


Diagramme inspiré de [Goodfellow, Bengio & Courville, 2016]

Comme le montre le graphique ci-haut, augmenter la complexité d'un modèle va accroître sa variance et diminuer son biais. Inversement, diminuer la complexité d'un modèle va accroître son biais et réduire sa variance.

Grossièrement on a: Erreur de généralisation = Biais + Variance + Bruit

Plus formellement, rappelons que l'apprentissage statistique a pour but de trouver une fonction $\hat{f}(x)$ qui approxime une fonction $f(x)$ inconnue qui décrirait parfaitement la distribution statistique des données [Hastie, Tibshirani & Friedman, 2008]. On dit que $f(x)$ est la «vraie fonction» et $\hat{f}(x)$ une fonction d'approximation qui tente de s'approcher de $f(x)$. Pour y arriver on utilise un algorithme d'apprentissage et des données pour ajuster les paramètres de la fonction $\hat{f}(x)$ [Hastie, Tibshirani & Friedman, 2008].

C'est pourquoi certains emploient le terme d'erreur d'approximation pour désigner le biais, dans le sens du choix d'une fonction $\hat{f}(x)$ qui approxime la vraie fonction $f(x)$. L'erreur de biais concerne le choix d'une fonction qui n'est pas parfaite parmi toutes les fonctions possibles. Du côté de la variance, on parle plutôt d'une erreur d'estimation. C'est à dire une erreur due au processus d'entraînement de la fonction à partir des données. Enfin le bruit est aussi parfois appelé erreur irréductible[246]. Ces trois termes additionnés sont non négatifs de façon à constituer une borne inférieure à l'erreur de généralisation.

[246] En anglais: «irreductible error»

Mathématiquement, cela donne:

Décomposition de l'erreur de généralisation - Formulation math. (fig. A2.6)
$$E\left[\left(y-\hat{f}(x)\right)^2\right] = \left(biais[\hat{f}(x)]\right)^2 + variance[\hat{f}(x)] + \sigma^2$$

Une courte explication s'impose. L'erreur totale ou erreur de généralisation $E\left[\left(y-\hat{f}(x)\right)^2\right]$ est l'espérance au sens statistique (i.e. la moyenne) pour une donnée nouvelle x (donnée de test) d'une fonction $\hat{f}(x)$ qui cherche à approximer la vraie fonction $f(x)$ plus du bruit ε, soit $y = f(x) + \varepsilon$ (ε est une variable aléatoire de moyenne 0 et de variance σ^2. Ici, nous minimisons l'erreur quadratique moyenne[247] pour éviter l'annulation des écarts positifs et négatifs.

Le biais, $biais[\hat{f}(x)] = E\left[\hat{f}(x) - f(x)\right]$ est précisément l'espérance de l'erreur entre la fonction d'approximation $\hat{f}(x)$ et la vraie fonction $f(x)$. C'est l'erreur reliée au choix d'une fonction précise parmi la multitude des fonctions possibles. Une fois de plus on élève le biais au carré pour éviter l'annulation des écarts positifs et négatifs, $\left(biais[\hat{f}(x)]\right)^2$.

La variance, $variance[\hat{f}(x)] = E\left[\left(\hat{f}(x) - \mu\right)^2\right]$ exprime l'espérance (i.e. la moyenne) du carré de la déviation entre la fonction $\hat{f}(x)$ et sa moyenne $\mu = E[\hat{f}(x)]$. Intuitivement, cela décrit le comportement plus ou moins variable de la fonction $\hat{f}(x)$.

Enfin, l'erreur irréductible σ^2 sur les données qui correspond à la variance σ^2 du bruit ε introduit plus tôt.

Pour une introduction à l'apprentissage automatique, consultez l'annexe 1.

[247] En anglais: «mean squared error»

Annexe 3 - Techniques de régularisation et de normalisation

La régularisation[248] impose des contraintes “mathématiques” à un modèle statistique afin de lutter contre le surajustement.

Pour revenir au dilemme biais / variance, les techniques de régularisation augmentent le biais afin de lutter contre le surajustement. On peut voir cela comme le retrait de degrés de liberté. L’emploi de la régularisation aide à produire des réseaux qui généralisent mieux.

Le contrôle algorithmique de la régularisation est confié à des hyperparamètres [Géron, 2017a].

A3.1 Techniques de pénalisation des poids

Parmi les techniques de régularisation largement utilisées, on trouve les techniques de pénalisation des poids[249] qui consistent à ajouter un terme dit de régularisation à la fonction de coût[250] J dont l’algorithme d’apprentissage cherche à minimiser le coût des erreurs.

Soit la fonction de coût J avec pour paramètres w et b qui est égale à la somme des erreurs:

$$J(w^{[l]}, b^{[l]}) = \frac{1}{m} \sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)})$$

La régularisation se fait par l’ajout d’un terme de régularisation à cette fonction de coût J mais uniquement pendant l’entraînement du modèle.

$$J(w^{[l]}, b^{[l]}) = \frac{1}{m} \sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)}) + Régularisation(w^{[l]})$$

A3.1.1 Régularisation L2 ou régression de Ridge

La régularisation de L_2 ou régression de Ridge[251] consiste à ajouter un terme de régularisation à la fonction de coût selon la norme vectorielle L_2 ou norme euclidienne.

---

[248] En anglais: «regularization»
[249] En anglais: «weights decay»
[250] En anglais: «cost function», «loss function», «error function», «objective function». En français: «fonction de coût», «fonction de perte», «fonction d’erreur» ou «fonction objective.»
[251] Note: Appelée aussi régularisation de Tikhonov ou régression de crête

$$J(w^{[l]}, b^{[l]}) = \frac{1}{m} \sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m} \sum_{l=1}^{L} \left|\left| w^{[l]} \right|\right|_F^2$$

Le deuxième terme influencé par l'hyperparamètre λ , appelé terme de régularisation L_2 pénalise la matrice de poids $w^{[l]}$. Intuitivement si on augmente fortement λ il faudra pour minimiser la fonction de perte J que la matrice de poids $w^{[l]}$ soit aussi petite que possible (tende vers zéro). Une petite matrice de poids avec de nombreux zéros est équivalent à avoir un réseau de plus petite taille.

A3.1.2 Régularisation L1 ou régression Lasso

On parle de régularisation L_1 ou de régression LASSO (Least Absolute Shrinkage and Selection Operator) lorsque le terme ajouté correspond à la somme des valeurs absolues des éléments d'un vecteur ce qui correspond à la norme vectorielle L_1

$$J(w^{[l]}, b^{[l]}) = \frac{1}{m} \sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m} \sum_{l=1}^{L} \left| w^{[l]} \right|$$

A3.1.2 Régression élastique

La régression élastique[252] est une combinaison astucieuse des régressions LASSO et Ridge [Hastie, Tibshirani & Friedman, 2008].

A3.2 Normalisation

La normalisation des lots[253] au moment de la descente de gradient} réduit le surajustement et peut être considérée comme une technique de régularisation [Goodfellow, Bengio & Courville, 2016]. La normalisation des lots est surtout utile pour réduire le temps de calcul.

A3.3 Extinction des neurones (dropout)

L'extinction de neurones[254] est une puissante technique de régularisation qui consiste à déconnecter une fraction des neurones des couches cachées du réseau. Elle peut également être considérée comme une injection de bruit, mais cette fois appliquée aux couches cachées de neurones plutôt qu'à l'entrée du réseau [Goodfellow, Bengio & Courville, 2016][255].

---

[252] En anglais: «elastic net»
[253] En anglais: «batch normalization»
[254] En anglais: «dropout»
[255] Note: It is important to understand that a large portion of the power of dropout arises from the fact that the masking noise is applied to the hidden units. [Goodfellow, Bengio & Courville, 2016]

Annexe 4 - Du bon usage des courbes d'entraînement

Les courbes d'entraînement[256] décrivent le processus d'entraînement d'un modèle statistique. Elles montrent l'évolution de l'erreur d'entraînement (erreur sur les données d'entraînement) et de l'erreur de validation (erreur sur les données de validation) d'un modèle en fonction de la taille du jeu de données d'entraînement. Cet outil permet de déterminer dans quelle mesure il serait avantageux d'ajouter plus de données d'entraînement. Les courbes d'entraînement aident au diagnostic des erreurs de variance (surajustement) ou de biais (sous-ajustement) [Géron, 2017a], [Ng, 2017].

Prenons une tâche de prédiction basée sur l'apprentissage supervisé. Ce cas recoupe un grand nombre des applications usuelles de l'apprentissage automatique. À partir d'un certain nombre d'observations (ou exemples) traités par l'algorithme d'apprentissage, ce dernier doit être capable de faire de bonnes prédictions sur des exemples qu'il n'a jamais vus auparavant. C'est de qu'on appelle sa capacité de généralisation.

En effet, faire de bonnes prédictions sur le jeu de données d'entraînement[257] est une bonne chose, mais cela est insuffisant, le véritable objectif est de faire de bonnes prédictions sur de nouvelles données qui appartiennent au jeu de données de validation[258]. L'erreur de prédiction sur les données de validation peut être considérée comme un bon indicateur de l'erreur de généralisation car le modèle n'a pas été entraîné sur ces données[259].

L'examen de la courbe d'apprentissage déterminera si l'ajout de données peut aider à améliorer la performance de notre modèle.

On débute en se fixant une performance cible[260] avec une ligne rouge horizontale. Puis on trace la courbe d'apprentissage sur les données de validation [Ng, 2017].

---

[256] Note; Le terme «courbe d'entraînement» (ou courbe de l'entraînement) est préféré à «courbe d'apprentissage» pour éviter la confusion avec l'expression déjà répandue «courbe d'apprentissage» qui désigne l'effort que doit déployer un individu pour apprendre quelque chose. Aussi, «courbe d'entraînement» est en harmonie avec l'expression «erreur d'entraînement» plutôt qu'«erreur d'apprentissage» et «données d'entraînement», plutôt que«données d'apprentissage». Une «courbe d'entraînement» (en anglais: «learning curve») montre l'«erreur d'entraînement» et l'«erreur de validation» (erreur sur les données de validation) d'un modèle en fonction de la taille du jeu de données d'entraînement.

[257] En anglais: «training dataset»

[258] En anglais: «validation dataset»

[259] Note: C'est aussi le cas pour les données de test (en anglais «test dataset»), mais le jeu de données de test sert uniquement à la toute fin pour connaître la performance finale du modèle.

[260] En anglais: «target performance»

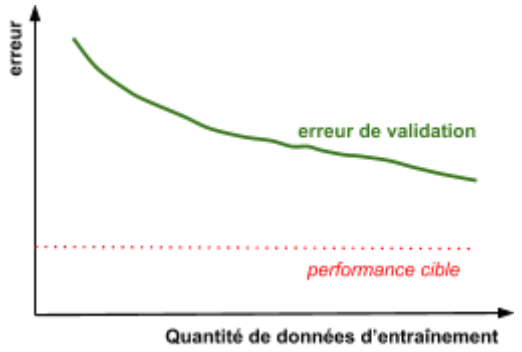
<table>
<tr><th colspan="2">Test des courbes de l'entraînement (fig. A3.1)</th></tr>
<tr><td> 

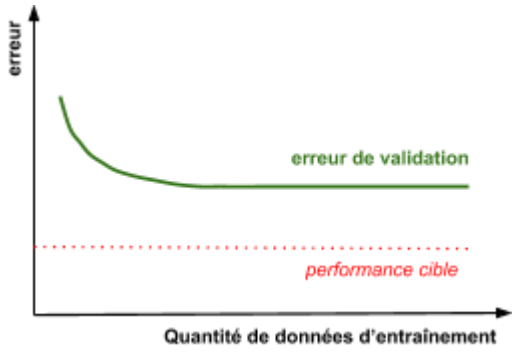
Diagramme inspiré de [Ng, 2017]</td><td> 

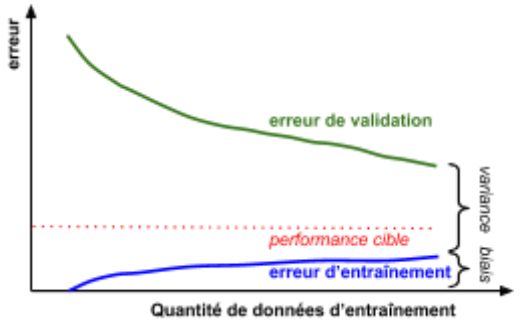
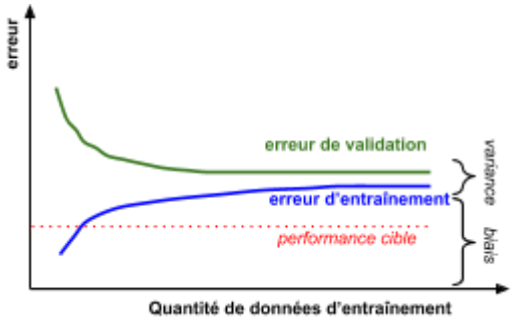
Diagramme inspiré de [Ng, 2017]</td></tr>
<tr><td>En extrapolant la courbe de l'erreur de validation (en vert), on voit que l'ajout de données d'entraînement devrait permettre d'atteindre la performance cible (en rouge).</td><td>Ici la courbe de l'erreur de validation (en vert) atteint un plateau, l'ajout de données d'entraînement ne permettra jamais d'atteindre la performance cible (en rouge).</td></tr>
<tr><td> 

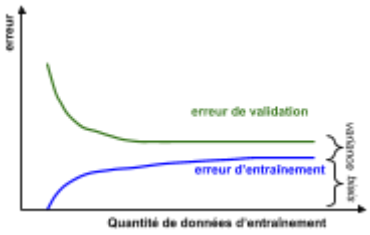
Diagramme inspiré de [Ng, 2017]</td><td> 
</td></tr>
<tr><td>L'ajout de la courbe de l'erreur d'entraînement permet de constater que «normalement» 1) l'algorithme performe mieux sur les données d'entraînement (plus faible erreur) que sur les données de validation. 2) L'erreur d'entraînement augmente avec le nombre de données d'entraînement.</td><td>Avec les deux courbes, il devient plus évident que l'ajout de plus de données ne sera pas suffisant pour atteindre la performance cible. Ici, l'erreur atteint un plateau ce qui signifie que le modèle sous-ajuste.</td></tr>
</table>

L'examen des courbes d'entraînement nous permet de décider s'il est utile de collecter davantage de données ou de plutôt envisager de modifier notre modèle.

Si l'erreur d'entraînement et l'erreur de validation convergent vers une valeur qui est égale ou plus basse que la performance cible avec la taille croissante du jeu de données d'entraînement, il serait alors utile d'augmenter la quantité de données d'entraînement.

Au contraire, si l'erreur d'entraînement et l'erreur de validation convergent vers une valeur qui est plus élevée que l'erreur désirée avec la taille croissante de l'ensemble des données d'entraînement, il serait inutile d'augmenter la quantité de données d'entraînement. Il faut plutôt prendre un modèle qui nous permettra d'apprendre des concepts plus complexes (c.-à-d. avec un biais plus faible et une variance plus élevée).

L'erreur d'entraînement, donnée par la hauteur de la courbe bleue, nous renseigne sur le biais du modèle. D'une manière analogue, l'écart entre les courbes de l'erreur d'entraînement (bleue) et de l'erreur de validation (verte) nous renseigne sur la variance du modèle avec un indice de sous-apprentissage. Plus l'écart est grand plus la variance est grande avec un risque accru de surajustement. [Ng, 2017]

Interprétation des courbes d'apprentissage (fig. A3.2)		
		
biais élevé sous-apprentissage	équilibre biais / variance ajustement optimal	variance élevée surapprentissage

On peut également tracer une courbe d'apprentissage pour l'erreur d'entraînement et l'erreur de validation en fonction du nombre d'itérations à travers le jeu de données ou époques[261]. Cette fois, il s'agira de déterminer s'il serait avantageux d'effectuer de nouvelles itérations d'entraînement, d'arrêter l'entraînement (arrêt précoce) et de voir si le modèle souffre davantage d'une erreur de variance (surajustement) ou d'une erreur de biais (sous-ajustement).

Effet de l'ajout de données (fig. A3.3)
Ajouter des données à un modèle en sous-ajustement ne servira à rien. *[Géron 2017a]*
Ajouter des données à un modèle en surajustement fera se rapprocher les courbes d'erreur d'entraînement et d'erreur de validation. *[Géron 2017a]*

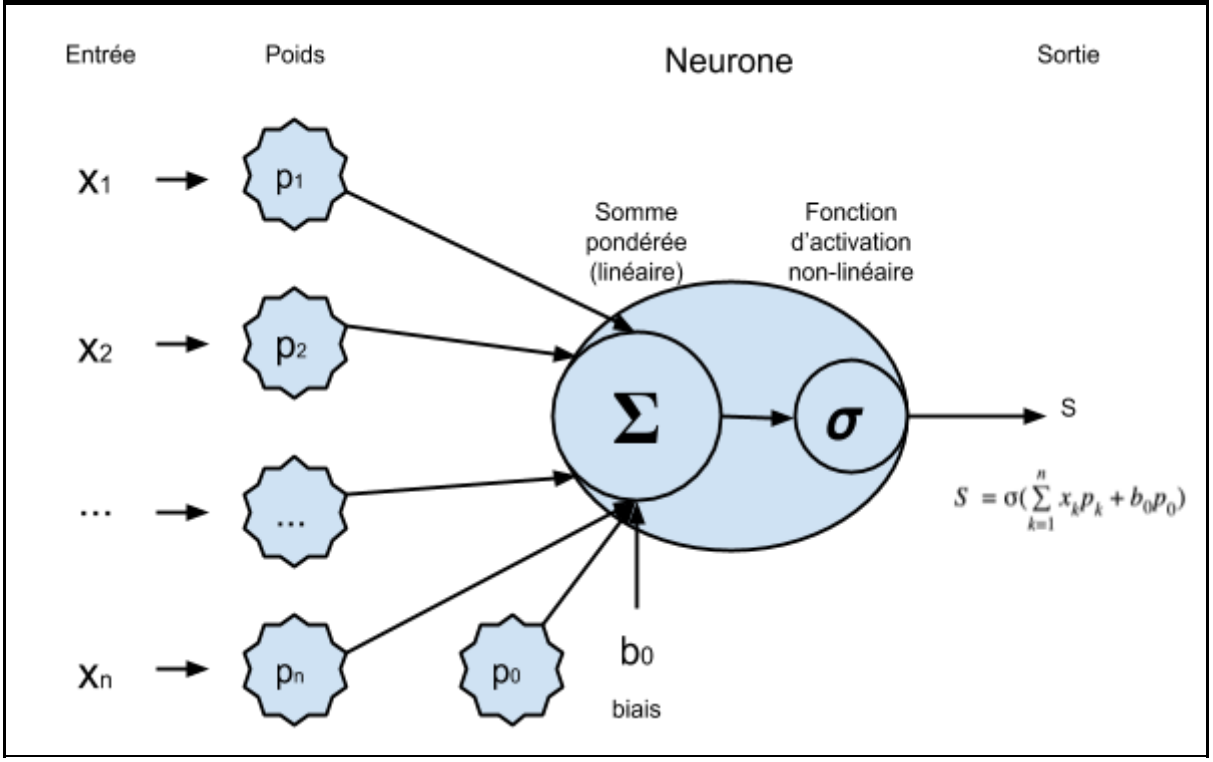
[261] En anglais: «epoch»

Annexe 5 - Généralités sur les réseaux de neurones artificiels

A5.1 Le neurone artificiel

Le neurone artificiel calcule la somme (Σ) pondérée par les poids p_k de ses attributs d'entrée x_k puis passe le résultat à travers une fonction d'activation non-linéaire (σ) pour produire sa sortie.

Neurone artificiel (fig. A5.1)



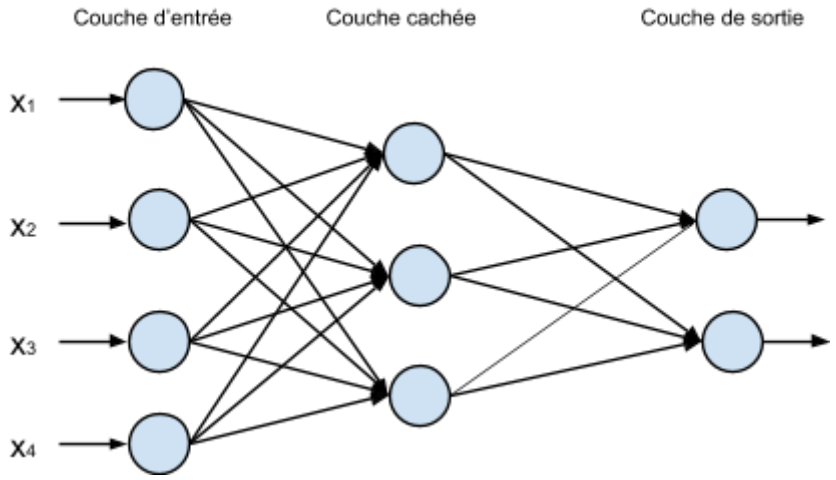
A5.2 Les réseaux de neurones artificiels

D'abord quelques généralités utiles à connaître sur les réseaux de neurones artificiels[262].

Grossièrement inspiré par le fonctionnement du cerveau biologique, comme le cerveau humain qui comporte quelques 100 milliards de neurones, où l'apprentissage modifie les connexions entre les neurones et leur intensité, un réseau de neurones artificiels est un ensemble organisé de neurones artificiels interconnectés, créé dans le but d'apprendre à partir de données. Les réseaux de neurones sont structurés en couches multiples d'où proviennent les notions de profondeur et d'apprentissage profond.

[262] En anglais: ««artificial neural network». «ANN»

Réseau de neurones à une couche cachée (fig. A5.2)



Le calcul fait par un réseau de neurones revient à évaluer une grosse fonction comportant des parties linéaires et des parties non-linéaires.

L'optimisation mathématique est utilisée pour l'entraînement du modèle sur les données afin de minimiser le coût des erreurs entre le modèle et chacune des données. L'optimisation a pour but d'estimer les paramètres de cette grosse fonction comportant des parties linéaires et des parties non-linéaires. Donc apprendre avec un réseau de neurones revient à approximer et optimiser une grosse fonction. Jusqu'à récemment on n'arrivait à bien entraîner que les réseaux de neurones constitués d'une seule couche de neurones cachée.

Les paramètres de la fonction correspondent aux poids du réseau de neurones que l'algorithme de rétropropagation du gradient[263] calcule à partir des données. Un réseau de neurones peut comporter un très grand nombre de paramètres mais demeure un modèle paramétrique puisque le nombre de paramètres est fini.

La fonction de coût d'un réseau de neurones profond n'est pas une fonction convexe[264], par conséquent rien ne garantit la convergence vers un minimum global. Cela dit, dans le cas des réseaux de neurones profonds entraînés sur de grandes quantités de données, la nature semble bien faire les choses et le minimum trouvé est souvent très convenable. En gros, tous les minima locaux se valent [LeCun, Bengio & Hinton, 2015].

Un modèle de réseau de neurones comporte trois parties importantes: 1) une architecture, 2) des paramètres ou poids qui sont appris à partir des données, et 3) des hyperparamètres qui précisent l'architecture (comme le nombre de couches et le nombre de neurones par couche) et d'autres hyperparamètres qui sont des paramètres de l'algorithme d'entraînement comme le pas du gradient[265] qui sont déterminés expérimentalement par essais et erreurs.

[263] En anglais: «gradient backpropagation»
[264] Note: Une fonction fortement convexe a un seul minimum global.
[265] En anglais: «learning rate». Aussi français, «gain du gradient» ou «taux d'apprentissage»

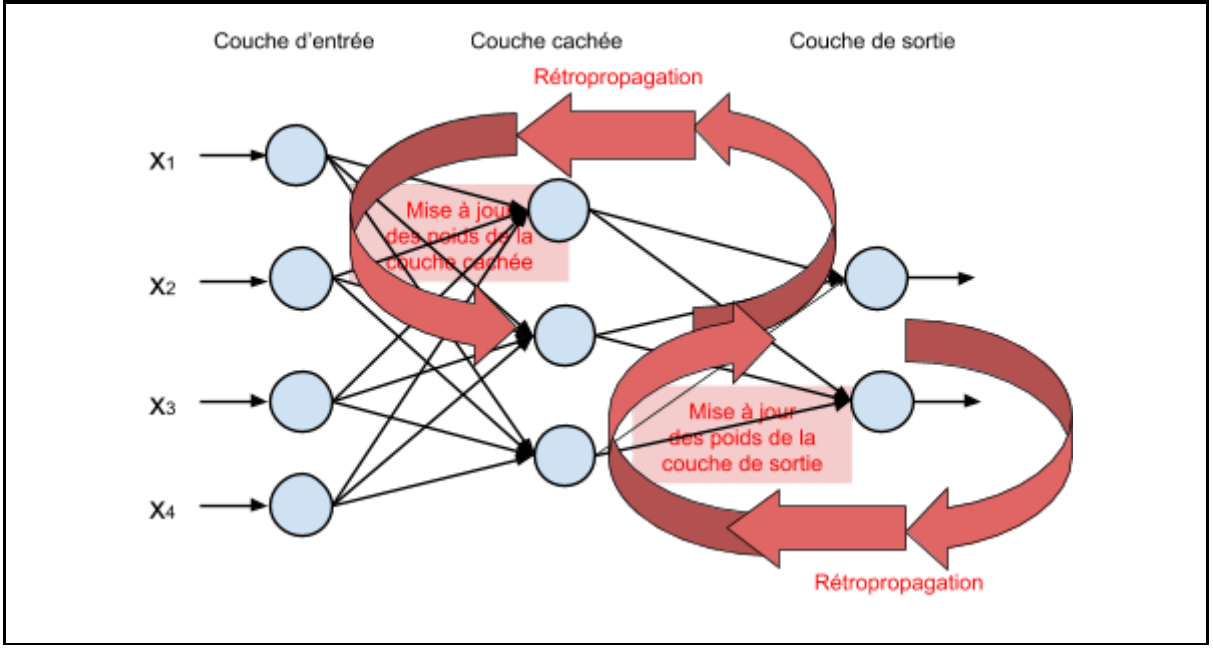
Annexe 6 - Note historique: Qui a inventé la rétropropagation?

Expliquons différents concepts et relations qui sont assez confus pour beaucoup de gens.

1) La rétropropagation[266] est l'idée générale de propager l'erreur de sortie vers l'arrière dans le réseau afin de mettre à jour les poids et ainsi minimiser l'erreur de prédiction. Elle trouve sa source dans la théorie du contrôle (algorithme de Kelly-Bryson) et de la cybernétique (boucle de rétroaction) des années 50 et 60 [Bryson & Ho, 1969].

L'algorithme de rétropropagation fut redécouvert et appliqué aux réseaux de neurones par Rumelhart en 1982 sans le faire fonctionner, par LeCun à partir de 1981 ou 1982 sous la forme de la répropagation des sorties binaires désirées qui fut présentée à Strasbourg en 1984 au congrès «Neurosciences et sciences de l'ingénieur» et publié en français dans les actes du congrès Cognitiva en 1985 à Paris [LeCun, 1985]. En fait, la rétropropagation peut se faire de différentes manières, certaines plus efficaces que d'autres, par exemple en propageant une perturbation.

Rétropropagation (fig. A6.1)



2) Le mécanisme d'autodifférentiation ou différentiation automatique[267] qui combine la différentiation à la rétropropagation (1) en se basant sur la règle de chaînage de la dérivée (6) fut inventé par le finlandais Seppo Linnainmaa vers 1970 [Griewank, 2012]. La différentiation automatique est utilisée entre autres pour l'analyse d'erreurs en calcul numérique, la résolution d'équations différentielles et le calcul analytique du gradient.

[266] En anglais: «backpropagation»
[267] En anglais «automatic differentiation» ou «reverse mode automatic differentiation»

3) La rétropropagation du gradient est l'application du mécanisme d'autodifférentiation (2) à l'entraînement des réseaux de neurones multicouches. Elle est devenue par la suite la définition usuelle de la rétropropagation pour les réseaux de neurones.

La rétropropagation du gradient pour l'entraînement des réseaux de neurones aurait été suggérée par l'américain Paul Werbos dans sa thèse [Werbos, 1974], sans être testée. La rétropropagation du gradient fut redécouverte par Rumelhart et Hinton et publiée en 1985 dans un rapport de l'University of California San Diego [Rumelhart, Hinton & Williams, 1985] puis largement diffusée dans le livre «Parallel distributed processing» de 1986.

La rétropropagation du gradient peut être utilisée partout où l'on a besoin de calculer le gradient dans un graphe de calcul, y compris dans d'autres contextes d'optimisation (4).

4) La descente de gradient qui est une famille de méthodes d'optimisation inventées par Cauchy en 1847. Ces algorithmes d'optimisation peuvent utiliser l'autodifférentiation (3) pour calculer le gradient si nécessaire. La descente de gradient qui a de nombreuses extensions et variantes est utilisée en apprentissage automatique pour optimiser une fonction de coût différentiable.

5) La descente de gradient stochastique[268], DGS, inventée par les statisticiens américains Herbert Robbins et Sutton Monro en 1951 est la variante la plus usuelle de la descente de gradient (4) pour l'entraînement de réseaux de neurones.

6) La règle de chaînage[269] des dérivées inventée par Leibniz en 1676. La règle de chaînage est utilisée par l'auto-différenciation ou la différenciation automatique (2).

[268] En anglais «stochastic gradient descent», «SGD»
[269] En anglais: «chain rule»

Annexe 7 - Qu’est-ce que l’apprentissage profond?

A7.1 L’émergence de l’apprentissage profond

Depuis quelques années, nous assistons émerveillés aux exploits des réseaux profonds de neurones que ce soit en classification d’images, en reconnaissance de la parole, en traduction automatique ou pour vaincre le champion du monde du jeu de Go.

Pourtant les statisticiens s’amusent à répéter que l’apprentissage profond n’est que le simple résultat de l’approximation optimisée par un réseau de neurones d’une fonction paramétrique différentiable de haute dimension à partir de données.

D’un autre point de vue, on peut voir l’apprentissage profond comme le résultat de l’entraînement de plusieurs couches successives d’unités non-linéaires de traitement appelées neurones sur un gros jeu de données qui associe des entrées à des sorties afin d’extraire des attributs et transformer les données d’entrée pour aboutir à la prédiction des données de sortie [Goldberg 2017], [Cholet 2017].

Pourquoi n'avons-nous pas découvert plus tôt que les réseaux profonds de neurones étaient si performants?

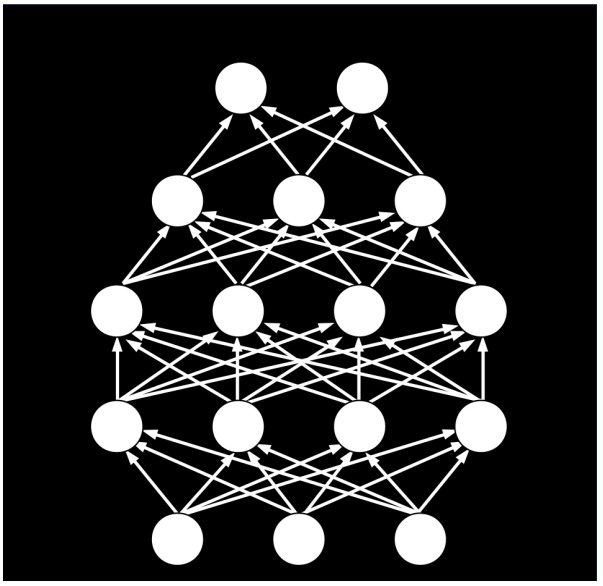
Il y a plusieurs raisons, mais surtout parce que les réseaux profonds de neurones ne brillent vraiment si vous avez suffisamment de données et d’importantes capacités de calcul pour les entraîner. Ce n'est que ces dernières années, avec l’avènement des données massives récoltées sur la Toile, les énormes capacités de calcul offertes par les processeurs graphiques et l’infonuagique que les conditions gagnantes ont été réunies pour entraîner des réseaux de neurones profonds.

Aussi, jusqu’au début des années 1990, on ne savait pas entraîner les réseaux de neurones à plusieurs couches (i.e. comportant plus d’une seule couche cachée) en raison des phénomènes d’explosion et d’évanescence du gradient. Malgré une percée par Schmidhuber en 1992 [Schmidhuber, 1992] et les travaux pionniers de LeCun sur les réseaux convolutifs [LeCun et al,, 1998], il fallut attendre en 2006 pour qu’une solution proposée par Geoffrey Hinton ne s’impose [Hinton, Osindero & Teh., 2006], [Bengio et al, 2007]. Il s’agissait d’un apprentissage non-supervisé couche par couche suivie d’une rétropropagation avec descente de gradient. Depuis, on a constaté qu’avec de bonnes fonctions d’activation (comme la ReLU), de plus grandes quantités de données et beaucoup de calculs, on pouvait se passer de cette astuce et entraîner directement des réseaux de neurones profonds [LeCun, Bengio & Hinton, 2015]. Il faut également souligner la contribution de Andrew Ng, chercheur à la Stanford University, et Jeff Dean de Google à l’industrialisation de cette technologie [Dean & Ng, 2012].

A7.2 L’apprentissage de représentations hiérarchiques

L’apprentissage profond utilise de multiples niveaux de représentation et de multiples couches successives d'unités de traitement non linéaires (ou neurones) sur des données en haute dimension. On parle ici de réseaux de neurones à plusieurs couches dans une hiérarchie. La profondeur d’un réseau de neurones correspond au nombre de couches cachées ou couches intermédiaires. Les données sortant de la couche inférieure viennent alimenter la couche supérieure qui transforme les données pour produire à son tour une nouvelle représentation. Les représentations produites par les couches inférieures de neurones sont combinées / transformées pour former des représentations de plus haut niveau (considérées plus abstraites) dans la couche suivante et ainsi de suite. L'apprentissage dans un réseau de neurones profond procède donc par transformation successive des données d'entrée jusqu'à une transformation finale qui produit la représentation des résultats. Les paramètres (ou poids) qui caractérisent les transformations produites par le réseau sont appris à partir des données par rétropropagation de l’erreur, entre l’étiquette prédite et l'étiquette souhaitée.

Illustration d’une architecture de réseau de neurones profond (fig. A7.1)



Un réseau de neurones à trois couches cachées.

Il a donc semblé normal de décrire ces réseaux comme étant profonds. Ensuite, les médias et les réseaux sociaux ont décidé d'utiliser le terme «apprentissage profond»[270] et nous sommes maintenant coincés avec lui. L’apprentissage profond est en quelque sorte la nouvelle image de marque des réseaux de neurones. C’est un nom qui fait rêver…

[270] En anglais: «deep learning»

Les premiers réseaux neuronaux, qui n'avaient pas la capacité d'apprentissage des réseaux neuronaux profonds d'aujourd'hui, n'avaient que quelques couches cachées, le plus souvent une seule. Il s'agissait donc de réseaux peu profonds, renforçant ainsi l'utilisation du terme «apprentissage profond» pour les réseaux profonds actuels.

Un terme plus exact serait «apprentissage de représentations»[271] ou «apprentissage hiérarchique»[272]. Il peut être utile de rappeler que la Conférence ICLR (International Conference on Learning Representations) a été créée en 2013 au début de la vague de l'apprentissage profond. L'apprentissage profond implique aussi la notion de représentation distribuée[273]. Une représentation distribuée encode chaque objet à l'aide d'un ensemble d'attributs qui sont considérés simultanément selon un profil d'activation[274] [Hinton, 1984].

A7.3 L'apprentissage automatique des attributs

De plus, les réseaux profonds de neurones sont réputés capables d'apprendre les représentations et les attributs[275] directement à partir des données brutes ou très peu raffinées avec peu ou pas de connaissances préalables [LeCun, Bengio & Hinton, 2015].

Les réseaux profonds de neurones requièrent peu ou pas d'ingénierie des attributs généralement réalisée avec du code explicite qui consiste à définir, sélectionner et extraire des attributs à partir de données.

En contrepartie, l'apprentissage automatique des attributs requiert davantage de données. Il arrive même qu'on utilise l'apprentissage profond seulement pour extraire des attributs plus riches ou pour condenser (réduire la dimension) de données. Par la suite ces attributs enrichis seront utilisés pour entraîner des modèles avec des algorithmes classiques d'apprentissage automatique ou d'autres algorithmes d'apprentissage profond.

Certaines personnes ont donc proposé le terme "apprentissage des attributs"[276] qui promet la fin de l'ingénierie des attributs[277], ce qui contraste avec les approches plus classiques de l'apprentissage automatique.

On a constaté que certains attributs appris par les réseaux neuronaux sont souvent des régularités statistique qui ne sont pas intuitives pour un être humain. Ces attributs passeraient totalement inaperçus. Ces attributs ne peuvent donc qu'être appris par un algorithme. Ceci s'avère particulièrement utile dans les domaines où les régularités statistiques (ou formes) ne sont pas faciles à identifier par un humain.

---

[271] En anglais: «representation learning»
[272] En anglais: «hierarchical learning»
[273] En anglais: «distributed representation»
[274] En anglais: «activation pattern», «pattern of activations». En français: «profil d'activation», «code d'activation», «motif d'activation»
[275] En anglais:«features». Aussi en français: «descripteur»,«caractéristique»,«propriété»,«caractère»
[276] En anglais: «features learning»
[277] En anglais: «features engineering»

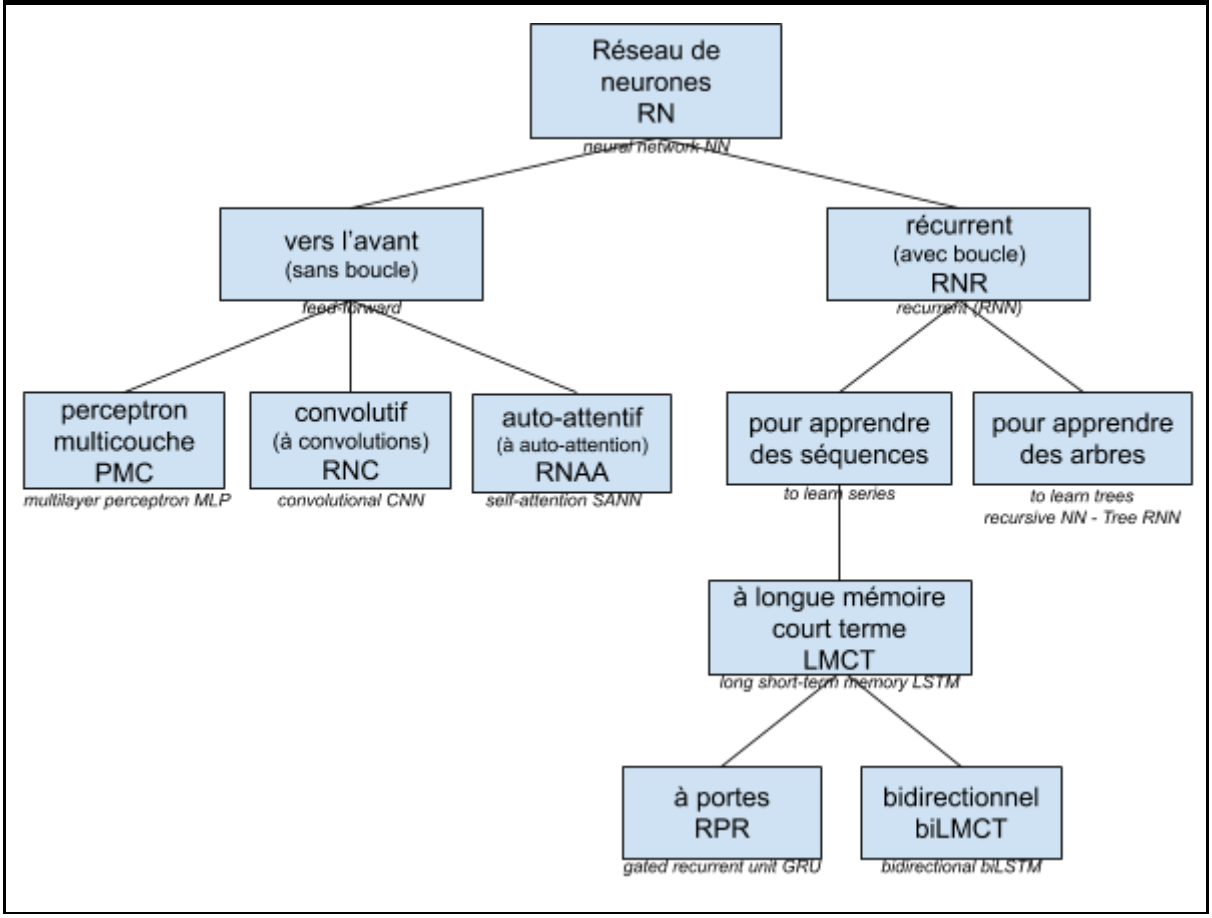
Il s'agit d'une vision un peu simpliste. En réalité, on ne peut pas se passer complètement de la phase d'ingénierie des attributs, ne serait ce que pour trouver des attributs suffisamment riches et adaptés à l'apprentissage profond. Il y a toujours un compromis à établir entre les connaissances préalables[278], la quantité de données d'entraînement, la capacité de généralisation des modèles et la difficulté d'entraîner ces modèles.

On peut toutefois affirmer qu'avec l'apprentissage profond l'ingénierie des attributs est réduite à sa plus simple expression. Cela dit, la recherche d'attributs et de connaissances génériques a priori qui soient capables de résoudre des classes importantes de problèmes demeure un domaine très actif [Bengio & LeCun, 2007].

A7.4 L'importance de l'architecture

L'architecture des réseaux de neurones joue un rôle très important en offrant une flexibilité non moins importante qu'il faut maîtriser. C'est un sujet de recherche très actif et beaucoup de publications font état de la création de nouvelles architectures ou d'améliorations à des architectures existantes. La mise au point de nouvelles architectures de réseaux de neurones est devenu le paradigme principal de la recherche actuelle.

Taxonomie des architectures usuelles de réseaux de neurones (fig. A7.2)



[278] En anglais: «prior knowledge»

Avec l'apprentissage profond, l'ingénierie des architectures de réseaux de neurones a remplacé l'ingénierie des attributs [Merity, 2016]. Cela représente une approche totalement nouvelle par rapport à l'ingénierie des attributs, car il faut mettre beaucoup d'effort du côté de la conception des architectures ou des modèles sous-jacent.

Importance de l'architecture dans les RN profonds - Citation (fig. A7.3)

In deep learning, architecture engineering is the new feature engineering. [Merity, 2016]

La maîtrise des architectures de réseaux profonds est difficile et constitue un goulot d'étranglement pour le développement d'applications. Le conseil pour un débutant est de partir d'une architecture qui a déjà résolu un problème similaire pour la modifier.

Il existe des services infonuagiques pour les architectures ou des applications simples comme Cloud AutoML de Google [Li & Li, 2018]. À un niveau plus avancé, plusieurs équipes de recherche tentent d'assister voire de découvrir automatiquement la bonne architecture de réseaux de neurones pour un problème donné. Pour le moment, il s'agit surtout d'utiliser des algorithmes génétiques ou évolutionnaires[279], l'apprentissage par renforcement[280] et d'importantes ressources en calcul pour explorer les possibilités. Citons les travaux de Google Brain qui ont découvert des algorithmes de reconnaissance d'images plus performants que ceux conçus «manuellement» [Real et al,, 2018].

A7.5 Une performance qui augmente avec la quantité de données

Alors que les algorithmes d'apprentissage automatique classiques semblent plafonner avec l'accroissement de la quantité de données, la performance des réseaux profonds de neurones augmente [Ng, 2917].

Performance de l'apprentissage profond et quantité de données (fig. A7.4)

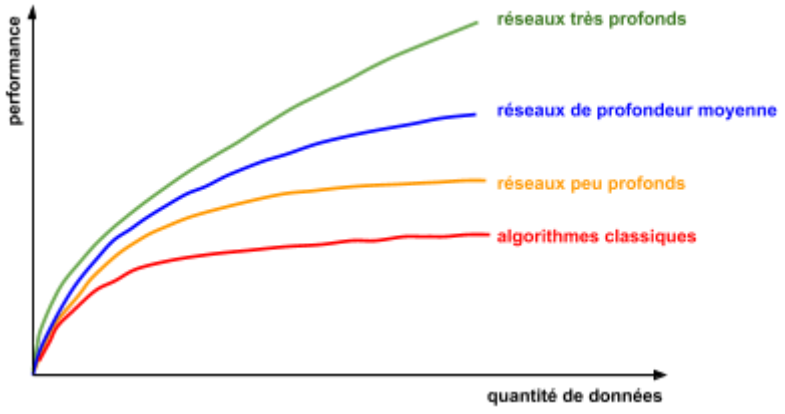


Diagramme inspiré de [Ng, 2017]

[279] En anglais: «genetic algorithms», «evolutionary algorithms»
[280] En anglais: «reinforcement learning»

A7.6 Une capacité de généralisation encore inexpliquée...

On a observé que les réseaux de neurones profonds généralisent bien (font de bonnes prédictions sur des données jamais vues auparavant) en dépit d'avoir beaucoup plus de paramètres que le nombre de données d'entraînement.

Plusieurs explications ont été avancées pour élucider ce mystère, mais aucune ne fait actuellement consensus.

L'article de [Zhang, et al,, 2016] qui s'est mérité le prix du meilleur article à la conférence ICLR 2017 affirme que pour comprendre l'apprentissage profond, il faut repenser la notion même de généralisation. Les auteurs ont observé que certaines architectures de réseaux de neurones profonds étaient capables d'apprendre sans erreur des données d'entraînement avec des étiquettes attribuées aléatoirement avec tout juste un léger accroissement du temps de calcul. Évidemment, l'erreur de généralisation sur le jeu de données de test était catastrophique puisque le réseau n'avait rien appris sur les vraies étiquettes. Ils ont également essayé avec des données qui n'étaient que du bruit aléatoire pour constater que ces réseaux étaient capables d'apprendre ces données d'entraînement sans aucune erreur. Appliqué aux données de test, ils ont observés une détérioration graduelle de l'erreur de généralisation à mesure qu'augmentait le bruit. Les réseaux de neurones profonds sont capables de reconnaître le signal résiduel dans les données tout en apprenant la partie bruitée.

Une théorie intéressante de chercheurs de l'Université de Jérusalem revisite l'idée de «compression de données en tant qu'apprentissage» comme avec les autoencodeurs. Le réseau de neurones supprimerait les détails de données bruitées en les comprimant au travers un «goulot d'étranglement de l'information»[281] qui ne conserverait que les caractéristiques les plus pertinentes des concepts généraux [Shwartz-Ziv & Tishby, 2017].

Ce qui nous amène à la notion de paramètres effectifs. On cherche à estimer le nombre de paramètres minimal en fonction du nombre de données d'entraînement pour une certaine architecture de réseau de neurones. En général ces estimations demeurent plusieurs ordres de grandeur au dessus de ce qui est observé expérimentalement. La meilleure estimation actuelle (mars 2018) est celle d'une équipe de Princeton University [Arora et al, 2018].

A7.7 Une capacité intrinsèque à lutter contre le surajustement

Parfois, l'apprentissage profond donne de bons résultats avec de petits jeux de données. Par exemple, le défi Kaggle Merck la synthèse de molécules ne comportait que quelques milliers de données et il a été remporté par un algorithme d'apprentissage profond.

[281] En anglais: «information bottleneck»

Avec beaucoup de régularisation par extinction de neurones[282] et en exerçant certaines contraintes au niveau de l'architecture du réseau de neurones, il est possible de réduire drastiquement le surajustement. Il semble bien que l'architecture du réseau de neurones et même le processus de descente de gradient joueraient un rôle de régularisation [Zhang, et al,, 2016]. En fait, toutes ces astuces pénalisent probablement la complexité et simplifient le modèle sous-jacent.

Aussi, on peut injecter des attributs plus riches ou «augmenter» les données en interne et à l'externe sans trop s'en apercevoir. En fait, cela revient soit à ajouter des connaissances a priori soit à accroître la quantité de données qui sert à l'apprentissage.

Il est probable que certaines de ces tâches Kaggle sont plus faciles qu'elles ne le paraissent au premier abord. Elles sont bien en deçà du niveau de complexité qui requiert l'apprentissage profond ou bien le jeu de données renferme des connaissances préalables que le réseau a réussi à apprendre. Parfois un simple réseau de neurones à une couche cachée suffit à résoudre le problème.

Cela reste un grand mystère des réseaux de neurones profonds d'être capable de généraliser (faire de bonnes prédictions sur des données jamais traitées) en dépit d'avoir beaucoup plus de paramètres que le nombre de données d'entraînement [Arora et al, 2018].

A7.8 Limites de l'apprentissage profond

Dans beaucoup de situations, on peut toujours appliquer des outils d'apprentissage profond à des problèmes simples avec de petits jeux de données, mais dans la plupart des cas l'apprentissage automatique avec des algorithmes classiques surpassera l'apprentissage profond avec beaucoup moins d'efforts; En d'autres termes, il est inutile d'utiliser un marteau pour écraser une mouche.

L'apprentissage profond brille quand il s'agit de problèmes complexes tels que la vision artificielle ou la reconnaissance de la parole, donc principalement des tâches de perception ou de reconnaissance de formes. On parle également d'apprentissage statistique[283] qui consiste en l'identification de régularités statistiques[284] [Hastie, Tibshirani & Friedman, 2008].

À la base, l'apprentissage profond est une algorithme de reconnaissance de formes sophistiqué et ultra-performant. C'est pourquoi, l'apprentissage profond semble plus limité du côté du raisonnement, de la logique, de la planification et de l'application du sens commun, qui relèvent traditionnellement des techniques d'IA symboliques. Une solution pourrait être d'hybrider les approches connexionnistes et symboliques ou encore d'implanter (simuler) des mécanismes de raisonnement dans les réseaux de neurones.

[282] En anglais: «dropout»
[283] En anglais: «statistical learning»
[284] En anglais: «statistical regularities»

Il est intéressant de noter que la plupart des applications pratiques comme les voitures autonomes, le joueur de Go AlphaGo, le moteur de recherche de Google et le traducteur DeepL sont des systèmes hybrides.

Aussi, les réseaux de neurones profonds sont essentiellement des boîtes noires. L'explicabilité[285] et l'interprétabilité[286] des résultats d'un modèle profond par un humain demeurent de beaux défis pour la recherche. Un algorithme est explicable s'il est possible de rendre compte de ses résultats explicitement à partir des données et attributs d'une situation. Autrement dit, s'il est possible de mettre en relation les données d'une situation et leurs conséquences sur les résultats de l'algorithme. Un algorithme est interprétable s'il est possible d'identifier, ou mieux de mesurer, les données ou les attributs qui participent le plus aux résultats de l'algorithme [IMT, 2017].

Enfin, les systèmes à base d'apprentissage profond sont très spécialisés, ce sont des idiots savants incapables d'accomplir d'autres tâches que celles prévues à l'origine. On est très loin d'une intelligence artificielle générale[287].

Pour une bonne introduction sur l'apprentissage profond, un article paru dans Nature [LeCun, Bengio & Hinton, 2015].

[285] En anglais: «explicability»
[286] En anglais: «interpretability»
[287] En anglais «Artificial General Intelligence», «AGI»

Annexe 8 - La représentation de données textuelles

Historiquement on peut constater une évolution dans la manière de représenter ou modéliser les données textuelles. L'idée est de passer de représentations textuelles à des représentations numériques. Ici, les termes représentation et modélisation sont pratiquement interchangeables bien que la notion de modèle soit plus complète.

A8.1 Représentation à un bit discriminant

D'abord, les vecteurs creux[288] de mots où les mots sont encodés avec un seul bit discriminant[289] qui ont la taille du lexique (une dimension est mise à 1 pour représenter le mot et toutes les autres dimensions à 0 pour représenter les autres mots), avec communément des dimensions de plusieurs milliers jusqu'à plusieurs centaines de milliers. Cette représentation volumineuse ne fournit pas d'informations syntaxiques, ni sémantiques.

Pour illustrer, nous allons utiliser un petit exemple inspiré de [Raschka, 2015].

Représentation à un bit discriminant (one-hot) (fig. A8.1)

```
from numpy import array
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder
corpus = [ "Le soleil brille",
           "Le temps est bon",
           "Le soleil brille le temps est bon et le ciel est bleu"]
corpus_mots = []
for document in corpus: corpus_mots += document.lower().split()
print("Le corpus:\n",corpus_mots)
# Encodeur étiquette vers nombre entier
encodeur_entiers = LabelEncoder()
corpus_entiers = encodeur_entiers.fit_transform(array(corpus_mots))
print("Le corpus encodé en nombres entiers:\n",corpus_entiers)
# Encodeur à un bit discriminant (One Hot Encoding)
encodeur_unbit = OneHotEncoder(sparse=False,categories='auto')
corpus_entiers = corpus_entiers.reshape(len(corpus_entiers), 1)
corpus_unbit = encodeur_unbit.fit_transform(corpus_entiers)
print("Une partie du corpus en encodage à un bit discriminant:\n",corpus_unbit[:10])
nouv_doc = "le soleil est bon"
nouv_doc_mots = nouv_doc.lower().split()
print("Un nouveau document:\n",nouv_doc_mots)
nouv_doc_entiers = encodeur_entiers.transform(array(nouv_doc_mots))
print("Nouveau document encodé en nbr entiers:\n", nouv_doc_entiers)
nouv_doc_mots = encodeur_entiers.inverse_transform(nouv_doc_entiers)
print("Nouveau document encodé en nbr entiers puis décodé:\n", nouv_doc_mots)
nouv_doc_unbit = encodeur_unbit.transform(nouv_doc_entiers.reshape(len(nouv_doc_entiers), 1))
print("Le nouveau document en encodage à un bit discriminant:\n",nouv_doc_unbit)
print("Le nouveau document en encodage à un bit discriminant affiché mot par mot:")
lexique = list(encodeur_entiers.inverse_transform(range(0,len(encodeur_entiers.classes_))))
afficher_document_un_bit_discriminant(lexique,nouv_doc_unbit)
```

[288] En anglais: «sparse vectors»

[289]En anglais: «one-hot encoding» où on a un vecteur en haute dimension avec la valeur 1 sur une seule dimension spécifique et 0 partout ailleurs.

```
Le corpus:
 ['le', 'soleil', 'brille', 'le', 'temps', 'est', 'bon', 'le', 'soleil', 'brille', 'le', 'temps',
'est', 'bon', 'et', 'le', 'ciel', 'est', 'bleu']
Le corpus encodé en nombres entiers:
 [6 7 2 6 8 4 1 6 7 2 6 8 4 1 5 6 3 4 0]
Une partie du corpus en encodage à un bit discriminant:
 [[0. 0. 0. 0. 0. 0. 1. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 1. 0.]
 [0. 0. 1. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 1. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [0. 0. 0. 0. 1. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 1. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 1. 0.]
 [0. 0. 1. 0. 0. 0. 0. 0. 0.]]
Un nouveau document:
 ['le', 'soleil', 'est', 'bon']
Nouveau document encodé en nbr entiers:
 [6 7 4 1]
Nouveau document encodé en nbr entiers puis décodé:
 ['le' 'soleil' 'est' 'bon']
Le nouveau document en encodage à un bit discriminant:
 [[0. 0. 0. 0. 0. 0. 1. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 1. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0. 0. 0. 0.]]
Le nouveau document en encodage à un bit discriminant affiché mot par mot:
       [bleu    bon     brille  ciel    est     et      le      soleil  temps    ]
le     [0.0     0.0     0.0     0.0     0.0     0.0     1.0     0.0     0.0      ]
soleil [0.0     0.0     0.0     0.0     0.0     0.0     0.0     1.0     0.0      ]
est    [0.0     0.0     0.0     0.0     1.0     0.0     0.0     0.0     0.0      ]
bon    [0.0     1.0     0.0     0.0     0.0     0.0     0.0     0.0     0.0      ]
```

A8.2 Représentation par sac de mots

Le modèle le plus simple qui ne tient pas compte de l'ordre des mots est le sac de mots[290]. Le sac de mots ne renseigne pas sur la position relative des mots et ne donne aucune information linguistique.

Le sac de mot le plus simple consiste en un vecteur de la taille du lexique, dans lequel on affecte un 1 à la dimension qui correspond à chacun des mots présents dans le texte, les autres étant à 0. Ce sac de mot est dit binaire car il ne renseigne que sur la présence ou l'absence d'un mot dans le document.

Sac de mots binaire (fig. A8.2)

```
import numpy as np
from sklearn.feature_extraction.text import CountVectorizer
corpus = [ "Le soleil brille",
           "Le temps est bon",
           "Le soleil brille, le temps est bon et le ciel est bleu"]
# création d'un sac de mots binaire
compteur_bin = CountVectorizer(binary=True)
sac_de_mots_bin = compteur_bin.fit_transform(np.array(corpus))
```

[290] En anglais: «bag of words», «BOW»

```
import operator
# création et tri du lexique iems
lexique_trie = sorted(compteur_bin.vocabulary_.items(), key=operator.itemgetter(1))
# affichage du contenu du sac de mots binaire pour chaque document
afficher_sac_de_mots_binaire(lexique_trie,sac_de_mots_bin)
```

```
Le soleil brille
[       bleu    bon     brille  ciel    est     et      le      soleil  temps   ]
[       0       0       1       0       0       0       1       1       0       ]
Le temps est bon
[       bleu    bon     brille  ciel    est     et      le      soleil  temps   ]
[       0       1       0       0       1       0       1       0       1       ]
Le soleil brille, le temps est bon et le ciel est bleu
[       bleu    bon     brille  ciel    est     et      le      soleil  temps   ]
[       1       1       1       1       1       1       1       1       1       ]
```

Le sac de mots avec compteur ou sac de mots avec fréquence contient pour chacun des mots dans le lexique le nombre d'occurrences de ce mot dans le document ce qui correspond à la fréquence du terme ou FT.

Sac de mots avec compteur (fig. A8.3)

```
import numpy as np
from sklearn.feature_extraction.text import CountVectorizer
# Création d'un sac de mots avec compteur
compteur = CountVectorizer()
sac_de_mots_freq = compteur.fit_transform(np.array(corpus))
import operator
# création et tri du lexique
lexique_trie = sorted(compteur.vocabulary_.items(), key=operator.itemgetter(1))
# affichage du contenu du sac de mots avec fréquence pour chaque document
afficher_sac_de_mots_avec_freq(lexique_trie,sac_de_mots_freq)
```

```
Le soleil brille
[       bleu    bon     brille  ciel    est     et      le      soleil  temps   ]
[       0       0       1       0       0       0       1       1       0       ]
Le temps est bon
[       bleu    bon     brille  ciel    est     et      le      soleil  temps   ]
[       0       1       0       0       1       0       1       0       1       ]
Le soleil brille, le temps est bon et le ciel est bleu
[       bleu    bon     brille  ciel    est     et      le      soleil  temps   ]
[       1       1       1       1       2       1       3       1       1       ]
```

A8.3 Représentation par un sac de mots FT-IFC

Pour tenir compte de l'importance relative d'un terme (mot ou expression) dans un document on utilise le sac de mots FT-IFC[291] où l'importance d'un terme dans un texte est directement proportionnel à sa fréquence d'apparition dans un texte (fréquence du terme, FT) et inversement proportionnel à sa fréquence d'apparition dans un corpus général (inverse de la fréquence dans le corpus, IFC) [WIKIPÉDIA, tf-idf]. L'IFC (en anglais IDF) est une mesure de la spécificité (rareté) du terme, car plus un terme est fréquent dans un corpus général (une large collection de textes), moins il est discriminant et son

[291] En anglais: «term-frequency inverse document frequency», «TF-IDF». En français, «produit de la fréquence d'un terme par l'inverse de sa fréquence dans un corpus» , «FT-IFC»

importance relative s'en trouve diminuée. Par exemple, des mots très fréquents comme «le», «la», «les» sont peu discriminants. Cette représentation repose sur des vecteurs creux[292] qui peuvent être de grande dimension.

Formulation mathématique - FT-IFC (fig. A8.4)

$$FT\text{-}IFC(t, doc) = FT(t, doc) * IFC(t, doc)$$

$$FT\text{-}IFC(t, doc) = log_{10}(1 + CompterTerme(t, doc)) * log_{10}(\frac{CompterDoc(doc, Corpus))}{1 + CompterDocContenantTerme(t, Corpus)})$$

Sac de mots FT-IFC (en anglais TF-IDF) (fig. A8.5)

```
import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer
corpus = [ "Le soleil brille",
           "Le temps est bon",
           "Le soleil brille, le temps est bon et le ciel est bleu"]
# création d'un compteur FT-IFC
compteur_ftifc = TfidfVectorizer()
sac_de_mots_ftifc = compteur_ftifc.fit_transform(np.array(corpus))
import operator
# création et tri du lexique
lexique_trie = sorted(compteur_ftifc.vocabulary_.items(), key=operator.itemgetter(1))
# affichage du sac de mots avec indice FT-IFC pour chaque document du corpus
afficher_sac_de_mots_avec_ftifc(lexique_trie,sac_de_mots_ftifc)
```

```
Le soleil brille
[       bleu    bon     brille  ciel    est     et      le      soleil  temps   ]
[       0.00    0.00    0.62    0.00    0.00    0.00    0.48    0.62    0.00    ]
Le temps est bon
[       bleu    bon     brille  ciel    est     et      le      soleil  temps   ]
[       0.00    0.53    0.00    0.00    0.53    0.00    0.41    0.00    0.53    ]
Le soleil brille, le temps est bon et le ciel est bleu
[       bleu    bon     brille  ciel    est     et      le      soleil  temps   ]
[       0.30    0.23    0.23    0.30    0.46    0.30    0.54    0.23    0.23    ]
```

A8.4 Représentation par un modèle à base d'information mutuelle

Une approche statistique alternative se base sur l'hypothèse distributionnelle en linguistique qui rappelons-le stipule que les mots qui apparaissent dans des contextes similaires partagent des sens similaires [Firth, 1957], [Harris, 1954].

Plusieurs mesures permettent de mesurer la tendance de deux mots à apparaître en même temps. Une mesure souvent utilisée est l'information mutuelle spécifique [Manning & Schütze, 1999], [WIKIPÉDIA, Pointwise Mutual information]. On dit spécifique car on se réfère à des observations spécifiques alors que l'information mutuelle IM s'intéresse à des moyennes sur toutes les informations [WIKIPÉDIA, Mutual information]. Intuitivement, l'information mutuelle mesure l'information que le mot_1 et le mot_2 partagent. La

[292] En anglais: «sparse vectors»

connaissance d'un mot réduit l'incertitude au sujet de l'autre mot. L'information mutuelle mesure la réduction de l'incertitude sur le mot_1 après avoir observé le mot_2 et vice versa.

Dépendant du problème à résoudre, on fixera l'étendue de la recherche de liens à l'intérieur d'une fenêtre de proximité de part et d'autre d'un mot (de 1 à 5 mots à gauche et à droite), à l'intérieur d'une même phrase, à l'intérieur d'une structure syntaxique (groupe de mots) à l'intérieur d'un paragraphe, à l'intérieur d'un document et même à l'intérieur d'une collection de documents. De plus, on a le choix entre seulement indiquer la présence du mot dans fenêtre fixée (mesure booléenne), ou d'en mesurer la fréquence qui correspond au nombre d'occurrences dans la fenêtre fixée, ou même de pondérer la fréquence en fonction de la distance plus ou moins grande entre les mots [Jurafsky & Martin, 2018].

Par exemple, si le mot «boson» et le mot «Higgs» étaient indépendants, alors connaître le mot «boson» ne donnerait aucune information sur le mot «Higgs» et vice versa. Leur information mutuelle serait donc nulle. À l'autre extrême, si le mot «boson» est fortement lié au mot «Higgs» et que le mot «Higgs» est fortement lié au mot «boson», alors toute information véhiculée par le mot «boson» serait partagée avec le mot «Higgs». La connaissance du mot «boson» détermine alors la valeur du mot «Higgs» et vice versa.

La cooccurrence de mots peut être définie par documents, par exemple dans une matrice où les mots du lexique sont les colonnes et les lignes différents documents, paragraphes, phrases, fenêtre, etc.

Pour fins de comparaisons, voici une implémentation d'une représentation de notre petit corpus sous la forme d'une matrice de cooccurrences mot-par-mot pour une fenêtre de 2 mots de part et d'autres. Le code ci-dessous produit une matrice de cooccurrences mot-par-mot. L'exemple s'inspire d'un carnet iPython partagé sur le site Kaggle [Altay, 2018].

Matrice de cooccurrences (fig. A8.6)

```
from collections import Counter
import itertools
import numpy as np

corpus = [ "Le soleil brille",
           "Le temps est bon",
           "Le soleil brille le temps est beau et le ciel est bleu"]

def creerLexiqueIndexCompteur(corpus):
    lexique = dict()
    compteur_freq = Counter()
    for i_doc, document in enumerate(corpus):
        for i_mot, mot in enumerate(document.lower().split()):
            compteur_freq[mot] += 1
            if mot not in lexique:
                lexique[mot] = len(lexique)
    index = dict([(mot,index) for mot,index in enumerate(lexique)])
    return lexique, index, compteur_freq

TAILLE_DU_CONTEXTE = 5
```

```
def genererGrammesVoisins(corpus):
    borne_fenetre = int(TAILLE_DU_CONTEXTE/2)
    compteur_grammesvoisins = Counter()
    for index_doc, document in enumerate(corpus):
        mots = document.lower().split()
        # Pour chaque mot trouver ses voisins dans le contexte
        for index_mot, mot in enumerate(mots):
            contexte_min = max(0, index_mot - borne_fenetre)
            contexte_max = min(len(mots) - 1, index_mot + borne_fenetre)
            index_contextes = [i for i in range(contexte_min, contexte_max + 1) if i != index_mot]
            for index_contexte in index_contextes:
                gramme_voisin = (mots[index_mot], mots[index_contexte])
                compteur_grammesvoisins[gramme_voisin] += 1
    return compteur_grammesvoisins

from scipy import sparse
def crerMatriceCooccurrences(compteur_grammesvoisins,lexique):
    lignes = []
    colonnes = []
    frequences = []
    for (mot1, mot2), frequence in compteur_grammesvoisins.items():
        index_mot1 = lexique[mot1]
        index_mot2 = lexique[mot2]
        lignes.append(index_mot1)
        colonnes.append(index_mot2)
        frequences.append(frequence)
    return sparse.csr_matrix((frequences, (lignes, colonnes)))

# Créer un lexique, un index du lexique et un compteur
lexique, index_lexique, compteur_freq = creerLexiqueIndexCompteur(corpus)
compteur_grammesvoisins = genererGrammesVoisins(corpus)
matrice_cooccurrences = crerMatriceCooccurrences(compteur_grammesvoisins,lexique)
afficher_matrice_cooccurences(matrice_cooccurrences,lexique,index_lexique)
```

```
       [        le      soleil  brille  temps   est     bon     beau    et      ciel    bleu    ]
    le [        0       3       3       2       3       0       1       1       1       0       ]
soleil [        3       0       2       0       0       0       0       0       0       0       ]
brille [        3       2       0       1       0       0       0       0       0       0       ]
 temps [        2       0       1       0       2       1       1       0       0       0       ]
   est [        3       0       0       2       0       1       1       1       1       1       ]
   bon [        0       0       0       1       1       0       0       0       0       0       ]
  beau [        1       0       0       1       1       0       0       1       0       0       ]
    et [        1       0       0       0       1       0       1       0       1       0       ]
  ciel [        1       0       0       0       1       0       0       1       0       1       ]
  bleu [        0       0       0       0       1       0       0       0       1       0       ]
```

À partir de la matrice des cooccurrences, nous pouvons calculer des représentations qui exploitent l'information mutuelle. L'information mutuelle spécifique IMS est le rapport de la probabilité de la cooccurrence des deux mots, $p(mot_1, mot_2)$ et du produit de la probabilité d'apparition de chacun d'eux séparément: $p(mot_1)\ p(mot_2)$.[293]

Information mutuelle spécifique - formulation math. (fig. A8.7)

$$IMS(mot_1, mot_2) = log_2(\frac{p(mot_1, mot_2)}{p(mot_1)p(mot_2)})$$

[293] Note: Rappelons que la probabilité que deux événement indépendants se réalisent ensemble est le produit des probabilités de chacun des événements. C'est la probabilité de cooccurrence fortuite.

La probabilité $p(mot_1, mot_2)$ est estimée par simple comptage des mots dans une fenêtre, plus précisément $p(mot_1, mot_2) = fréquence(mot_1, mot_2)/totalDeMots$. Il en va de même pour $p(mot_1)$ et $p(mot_2)$. Le logarithme ajouté permet de "réduire" la dispersion du résultat et d'exploiter le fait que le logarithme d'un quotient est la différence des logarithmes.

Bien que l'information mutuelle spécifique[294] IMS puisse être négative, par exemple dans le cas où deux mots cooccurrent moins souvent que le hasard ne le permet, pour des raisons pratiques[295] on utilise l'information mutuelle spécifique positive[296], $IMSP$ en remplaçant les valeurs négatives par zéro (0)[297].

Information mutuelle spécifique positive - formulation math. (fig. A8.8)

$$\mathrm{IMS}(mot, contexte) = log_2 \frac{p(mot, contexte)}{p(mot)p(contexte)}$$

$$p(mot_i, contexte_j) = \frac{\#(mot_i, contexte_j)}{\sum_k^N \sum_l^N \#(mot_k, contexte_l)}, \quad p(mot_i) = \frac{\sum_j^N \#(mot_i, contexte_j)}{\sum_k^N \sum_l^N \#(mot_k, contexte_l)}, \quad p(contexte_j) = \frac{\sum_i^N \#(mot_i, contexte_j)}{\sum_k^N \sum_l^N \#(mot_k, contexte_l)}$$

où $\#(mot_i, contexte_j)$ est le décompte du nombre d'occurrences du mot mot_i dans le $contexte_j$

Enfin, on peut définir l'information mutuelle spécifique positive comme étant:

$$\mathrm{IMSP}(mot, contexte) = \max\left[\mathrm{IMS}(\mathrm{mot}, \mathrm{contexte}), 0\right]$$

Matrices d'information mutuelle spécifique (fig. A8.9)

```
def creerMatricesIMS(matrice_cooccurrences,lexique):
    # Listes servant à construire des matrices
    lignes = []
    colonnes = []
    # IMS: Information Mutuelle Spécifique
    IMS_val = []
    # IMSP: Information Mutuelle Spécifique Positive
    IMSP_val = []
    # IMSL: Information Mutuelle Spécifique Lisse
    IMSL_val = []
    # IMSL: Information Mutuelle Spécifique Lisse Positive
    IMSLP_val = []
    nbre_total_grammes_voisins = matrice_cooccurrences.sum()
    # Somme sur les mots et les contextes
```

[294] En anglais: «pointwise mutual information», «PMI»
[295] Note: jugée peu fiable à moins de corpus énorme et difficile à valider
[296] En anglais: «positive pointwise mutual information», «PPMI»
[297] Note: avec la fonction max

```
    somme_sur_les_mots = np.array(matrice_cooccurrences.sum(axis=0)).flatten()
    somme_sur_les_contextes = np.array(matrice_cooccurrences.sum(axis=1)).flatten()
    # Lissage
    alpha = 0.75
    nbre_total_contextes_lisse = np.sum(somme_sur_les_contextes**alpha)
    somme_sur_les_contextes_lisse = somme_sur_les_contextes**alpha
    for (mot1, mot2), occurrences_grammesvoisins in compteur_grammesvoisins.items():
        mot1_index = lexique[mot1]
        mot2_index = lexique[mot2]
        prob_mot_contexte = occurrences_grammesvoisins / nbre_total_grammes_voisins
        occurrences_mot = somme_sur_les_contextes[mot1_index]
        prob_mot = occurrences_mot / nbre_total_grammes_voisins
        occurrences_contexte = somme_sur_les_mots[mot2_index]
        prob_contexte = occurrences_contexte / nbre_total_grammes_voisins
        IMS = np.log2(prob_mot_contexte/(prob_mot*prob_contexte))
        IMSP = max(IMS, 0)
        nbre_contextes_lisse = somme_sur_les_contextes_lisse[mot2_index]
        prob_contextes_lisse = nbre_contextes_lisse / nbre_total_contextes_lisse
        IMSL = np.log2(prob_mot_contexte/(prob_mot*prob_contextes_lisse))
        IMSLP = max(IMSL, 0)
        # Accumulation dans des tableaux des valeurs d'information mutuelle
        lignes.append(mot1_index)
        colonnes.append(mot2_index)
        IMS_val.append(IMS)
        IMSP_val.append(IMSP)
        IMSL_val.append(IMSL)
        IMSLP_val.append(IMSLP)
    # Création de matrices creuses
    IMS_mat = sparse.csr_matrix((IMS_val, (lignes, colonnes)))
    IMSP_mat = sparse.csr_matrix((IMSP_val, (lignes, colonnes)))
    IMSL_mat = sparse.csr_matrix((IMSL_val, (lignes, colonnes)))
    IMSLP_mat = sparse.csr_matrix((IMSLP_val, (lignes, colonnes)))
    return IMS_mat, IMSP_mat, IMSL_mat, IMSLP_mat

IMS_mat, IMSP_mat, IMSL_mat, IMSLP_mat = creerMatricesIMS(matrice_cooccurrences,lexique)
afficher_matrice_IMS(IMS_mat,lexique,index_lexique)
```

```
       [        le      soleil  brille  temps   est     bon     beau    et      ciel    bleu    ]
    le [        0.0     1.314   1.051   0.243   0.314   0.0     0.051   0.051   0.051   0.0     ]
soleil [        1.314   0.0     1.951   0.0     0.0     0.0     0.0     0.0     0.0     0.0     ]
brille [        1.051   1.951   0.0     0.466   0.0     0.0     0.0     0.0     0.0     0.0     ]
 temps [        0.243   0.0     0.466   0.0     0.729   2.051   1.051   0.0     0.0     0.0     ]
   est [        0.314   0.0     0.0     0.729   0.0     1.536   0.536   0.536   0.536   1.536   ]
   bon [        0.0     0.0     0.0     2.051   1.536   0.0     0.0     0.0     0.0     0.0     ]
  beau [        0.051   0.0     0.0     1.051   0.536   0.0     0.0     1.858   0.0     0.0     ]
    et [        0.051   0.0     0.0     0.0     0.536   0.0     1.858   0.0     1.858   0.0     ]
  ciel [        0.051   0.0     0.0     0.0     0.536   0.0     0.0     1.858   0.0     2.858   ]
  bleu [        0.0     0.0     0.0     0.0     1.536   0.0     0.0     0.0     2.858   0.0     ]
```

Il a été établi que les matrices de cooccurrences à l'état brut donnent de piètres modèles, mais l'application de techniques de réduction de dimension augmente la performance des modèles. Sur les matrices d'information mutuelle spécifique, nous appliquons la décomposition en valeurs singulières (DVS)[298] pour produire des vecteurs denses à partir de matrices creuses.

A8.5 Représentation par un modèle n-gramme

Les modèles décrits jusqu'ici ne prenaient pas en compte l'ordre des différents mots dans la phrase, mais seulement l'occurrence de chaque mot séparément. Une première approche

[298] En anglais: «Singular Value Decomposition», «SVD»

statistique pour tenir compte de l'ordre des mots et de leur voisinage est le modèle n-gramme[299] qui examine des séries de mots consécutifs de longueur «n» et traite chaque série de «n» mots comme un mot différent. Par exemple, un modèle bigramme ou trigramme sera capable de tenir compte de la présence d'une négation.

Représentation n-gramme (fig. A8.10)

```
import re
from nltk.util import ngrams

def generer_ngrammes(phrase,n):
    phrase = re.sub(r'[^a-zA-Z0-9\s]', ' ', phrase.lower())
    jetons = [jeton for jeton in phrase.split(" ") if jeton != ""]
    return list(ngrams(jetons, n))

corpus = [ "Le soleil brille",
           "Le temps est bon",
           "Le soleil brille le temps est beau et le ciel est bleu"]

for document in corpus:
    print(generer_ngrammes(document,2))
```

```
[('le', 'soleil'), ('soleil', 'brille')]
[('le', 'temps'), ('temps', 'est'), ('est', 'bon')]
[('le', 'soleil'), ('soleil', 'brille'), ('brille', 'le'), ('le', 'temps'), ('temps', 'est'),
('est', 'beau'), ('beau', 'et'), ('et', 'le'), ('le', 'ciel'), ('ciel', 'est'), ('est', 'bleu')]
```

Maintenant, nous allons créer un sac de bigrammes sur le modèle des sacs de mots vus précédemment.

Sac de bigrammes (fig. A8.11)

```
from sklearn.feature_extraction.text import CountVectorizer
corpus = [ "Le soleil brille",
           "Le temps est bon",
           "Le soleil brille le temps est beau et le ciel est bleu"]
# création d'un compteur de bigrammes
compteur_bigrammes = CountVectorizer(analyzer = "word", binary = True,ngram_range=(2,2))
sac_de_bigrammes = compteur_bigrammes.fit_transform(np.array(corpus))
import operator
# création et tri du lexique de bigrammes
lexique_bigrammes_trie = sorted(compteur_bigrammes.vocabulary_.items(), key=operator.itemgetter(1))
print(lexique_bigrammes_trie,"\n")
# affichage du contenu du sac de bigrammes avec fréquence pour chaque document
afficher_sac_de_bigrammes(sac_de_bigrammes,lexique_bigrammes_trie)
```

```
[('beau et', 0), ('brille le', 1), ('ciel est', 2), ('est beau', 3), ('est bleu', 4), ('est bon',
5), ('et le', 6), ('le ciel', 7), ('le soleil', 8), ('le temps', 9), ('soleil brille', 10), ('temps
est', 11)]

Le soleil brille
          Bigramme                      Fréquence
        «beau et»                          0
        «brille le»                        0
        «ciel est»                         0
        «est beau»                         0
        «est bleu»                         0
```

[299] En anglais: «n-gram»

```
        «est bon»                           0
        «et le»                             0
        «le ciel»                           0
        «le soleil»                         1
        «le temps»                          0
        «soleil brille»                     1
        «temps est»                         0

Le temps est bon
          Bigramme                      Fréquence
        «beau et»                           0
        «brille le»                         0
        «ciel est»                          0
        «est beau»                          0
        «est bleu»                          0
        «est bon»                           1
        «et le»                             0
        «le ciel»                           0
        «le soleil»                         0
        «le temps»                          1
        «soleil brille»                     0
        «temps est»                         1

Le soleil brille le temps est beau et le ciel est bleu
          Bigramme                      Fréquence
        «beau et»                           1
        «brille le»                         1
        «ciel est»                          1
        «est beau»                          1
        «est bleu»                          1
        «est bon»                           0
        «et le»                             1
        «le ciel»                           1
        «le soleil»                         1
        «le temps»                          1
        «soleil brille»                     1
        «temps est»                         1
```

A8.6 Modèle de langue

L'idée d'un modèle de langue est d'être capable de prédire le prochain mot dans un texte. Cette capacité de prédiction du prochain mot est utile dans plusieurs tâches comme la reconnaissance vocale, la reconnaissance d'écriture manuscrite ou la génération de texte.

Dans l'exemple suivant, nous allons utiliser un petit modèle trigramme pour générer des textes sur le modèle de la Génèse, inspiré de [Shabda, 2009]. Les trigrammes sont mémorisés dans une cache (dictionnaire Python) et choisis au hasard au moment de rallonger le texte.

L'intérêt pédagogique de ce petit bout de code est de montrer un générateur de texte basé sur un modèle n-gramme peut produire en quelques secondes des résultats «bluffants mais absurdes» comparables à des modèles à base de réseaux récurrents (typiquement LMCT) qui prennent des heures à entraîner [Karpathy, 2015].

Modèle trigramme pour la genèse d'un texte (fig. A8.12)

```
import random
from nltk.util import ngrams
import re

class ChaineDeMarkov(object):

    def __init__(self, corpus):
        self.cache = {}
        self.lexique = corpus
        self.taille_lexique = len(self.lexique)
        self.cache_de_trigrammes()

    def generer_ngrammes(self, n):
        return list(ngrams(self.lexique, n))

    def cache_de_trigrammes(self):
        for gramme1, gramme2, gramme3 in self.generer_ngrammes(3):
            cle = (gramme1, gramme2)
            if cle in self.cache:
                # On autorise la copie multiple pour tenir compte
                # de la fréquence
                self.cache[cle].append(gramme3)
            else:
                self.cache[cle] = [gramme3]

    def generation_texte(self, taille=25):
        germe = random.randint(0, self.taille_lexique-3)
        gramme1, gramme2 = self.lexique[germe], self.lexique[germe+1]
        mots_generes = []
        for i in range(taille):
            mots_generes.append(gramme1)
            gramme1, gramme2 = gramme2, random.choice(self.cache[(gramme1, gramme2)])
        mots_generes.append(gramme2)
        texte_genere = ' '.join(mots_generes)
        texte_genere = texte_genere[0:texte_genere.rfind(".")+1]
        texte_genere = texte_genere[0].upper() + texte_genere[1:]
        texte_genere = re.sub(r"\s([.!?,;:])|\s(['-])\s", r'\g<1>\g<2>', texte_genere)
        return texte_genere

print("Code chaine de Markov!")

texte_genese_multilingue = nltk.Text(nltk.corpus.genesis.words())
# Extraction du corpus NLTK de la version française du texte de la Genèse
texte_genese_francais = texte_genese_multilingue.tokens[121338:167454]
markov = ChaineDeMarkov(texte_genese_francais)
texte_genere = markov.generation_texte(150)
print(texte_genere)
```

```
Et tout ce qu'on abreuve les brebis étaient chétives, il ne fut plus, il l'apprit. Les brebis
entraient en chaleur, je vous donne tout cela comme l'appela du nom de Peniel: car elle disait:
C'est en bon état; et elle habita dans le pays d'égypte des chars et des filles. Tous les jours
d'Issacar. Léa devint encore enceinte, et partirent. L'éternel lui apparut dans la maison, et que
les fils d'un petit nombre d'Israël étaient appesantis par la fenêtre qu'ils se rassembleront contre
moi. Et Pharaon s'éloigna de la hanche de Jacob, en te tuant.
```

Cela dit, il semble possible qu'en multipliant les niveaux de représentations (la profondeur des modèles), la taille et la richesse des corpus, on puisse entraîner de gros modèles de langue à générer des textes nettement plus cohérents. Par exemple, le modèle génératif GPT-2 (Generative Pre-training Transformers) qui a été entraîné sur un corpus énorme (40 Go de textes) pendant une semaine sur une architecture comportant 32 processeurs

spécialisés dans les calculs matriciels (TPU: Tensor Processor Unit). On estime les coûts de calcul à environ 58 K$ can,est une autre preuve de la déraisonnable efficacité des données [Radford et al, 2019].

A8.7 Modèles à base de connaissances linguistiques

A8.7.1 Représentation par un modèle lexical

Évidemment, il est possible de représenter un texte par une analyse linguistique de son contenu.

Ci-dessous, nous allons générer une série d'attributs lexicaux en soumettant les phrases de notre petit corpus à l'analyseur SyntaxNet [Google, 2018a], [Petrov, 2016], [Kong et al, 2017] encapsulé dans la bibliothèque `paraphrase` et l'outil PolyPhrases développés à Polytechnique Montréal.

Typiquement une balise lexicale comporte le mot, sa position dans la phrase et son étiquette lexicale. On peut éventuellement enrichir la balise d'informations supplémentaires comme le lemme, le genre, le nombre, etc.

Représentation lexicale (fig. A8.13)

```
import paraphrase.google_api

def get_pos_tag_strings(phrase):
    pos_tag_strings = []
    syntaxnet_parsing = paraphrase.google_api.parse(phrase,"fr")
    pos_tags = paraphrase.google_api.get_POS_tags(syntaxnet_parsing)
    for pos_tag_i, pos_tag in enumerate(pos_tags):
        pos_tag_string = pos_tag[0]+"_"+str(pos_tag_i)+"_"+pos_tag[1]
        pos_tag_strings.append(pos_tag_string)
    return(pos_tag_strings)

for phrase in corpus:
    print(phrase)
    print(get_pos_tag_strings(phrase),"\n")
```

```
Le soleil brille
['Le_0_DET', 'soleil_1_NOUN', 'brille_2_VERB']

Le temps est bon
['Le_0_DET', 'temps_1_NOUN', 'est_2_VERB', 'bon_3_ADJ']

Le soleil brille le temps est beau et le ciel est bleu
['Le_0_DET', 'soleil_1_NOUN', 'brille_2_VERB', 'le_3_DET', 'temps_4_NOUN', 'est_5_VERB',
'beau_6_ADJ', 'et_7_CONJ', 'le_8_DET', 'ciel_9_NOUN', 'est_10_VERB', 'bleu_11_ADJ']
```

A8.7.2 Représentation par un modèle syntaxique

Nous allons maintenant générer une série d'attributs syntaxiques. Une balise syntaxique comporte le mot, sa position dans la phrase et son étiquette syntaxique (racine, sujet, complément, etc.) ainsi que le mot dont il dépend avec sa position.

Représentation syntaxique (fig. A8.14)

```
import paraphrase.google_api

for phrase in corpus:
    syntaxnet_parsing = paraphrase.google_api.parse(phrase,"fr")
    dep_tags = paraphrase.google_api.get_dep_tags(syntaxnet_parsing)
    print(phrase)
    print(dep_tags,"\n")
```

```
Le soleil brille
['Le_0_DET_soleil_1', 'soleil_1_NSUBJ_brille_2', 'brille_2_ROOT']

Le temps est bon
['Le_0_DET_temps_1', 'temps_1_NSUBJ_est_2', 'est_2_ROOT', 'bon_3_ACOMP_est_2']

Le soleil brille le temps est beau et le ciel est bleu
['Le_0_DET_soleil_1', 'soleil_1_NSUBJ_brille_2', 'brille_2_NSUBJ_est_5', 'le_3_DET_temps_4',
'temps_4_DOBJ_brille_2', 'est_5_ROOT', 'beau_6_ACOMP_est_5', 'et_7_CC_est_5', 'le_8_DET_ciel_9',
'ciel_9_NSUBJ_est_10', 'est_10_CONJ_est_5', 'bleu_11_ACOMP_est_10']
```

A8.8 Modèles neuronaux de la langue

L'idée de la représentation par vecteur-mot[300] remonte aux travaux pionniers de l'équipe de Yoshua Bengio [Bengio et al, 2003] au MILA sur les premiers modèles neuronaux de la langue. L'objectif était de contrer le problème de la dilution[301] des données linguistiques et de lutter contre la malédiction de la haute dimension[302] en TLN où les matrices sont de très grande dimension, très éparses et comportent beaucoup d'éléments à zéro.

A8.8.1 Vecteur-mot

Le vecteur-mot ou vecteur contextuel reprend l'hypothèse distributionnelle mais cette fois selon une approche prédictive. L'idée de base du vecteur-mot consiste à représenter chaque mot en utilisant son contexte, par un vecteur sémantiquement informatif, dans un espace vectoriel de dimension réduite et plus dense. L'idée est donc de regrouper les mots contextuellement proches. Ces représentations permettent de capturer des similitudes entre des mots, des phrases ou des documents et de définir au besoin des opérations linéaires.

---

[300] En anglais: «word vectors», «word embeddings», «embeddings». Aussi en français, «vecteur de contexte», «vecteur d'enrichissement», «plongement lexical», «plongement»
[301] en anglais: «sparsity», «data sparseness». En français, on dit aussi «dispersion des données».
[302] en anglais: «curse of dimensionality». On peut se questionner sur l'intérêt d'inventer le terme «dimensionality» alors que le mot «dimension» existe déjà.

Typiquement, un vecteur-mot a une taille comprise entre 50 et 500 (plus souvent 200 ou 300) dimensions.

Représentation par vecteur-mot - intuition (fig. A8.15)
L'idée de base de la représentation par vecteur-mot ou vecteur contextuel est d'enrichir en exploitant le contexte (typiquement via la cooccurrence) et densifier (réduire la dimension) la représentation d'information linguistique la faisant passer d'une information fortement diluée et en très haute dimension comme la représentation à un bit discriminant (one hot encoder) vers une représentation vectorielle de dimension réduite capable de capturer les similitudes entre des régularités statistiques associées à des propriétés linguistiques.

Certains voient la création d'un vecteur-mot comme analogue à une opération de plongement[303] d'où le terme anglais «embedding». Or en mathématique, une opération de plongement implique que l'un des objets est un sous-objet de l'autre. Ici, il semble davantage question d'une opération de réduction de dimension ou d'une approximation où l'on passe d'un espace vectoriel très vaste vers un espace vectoriel de dimension réduite.

A8.8.2 Word2Vec

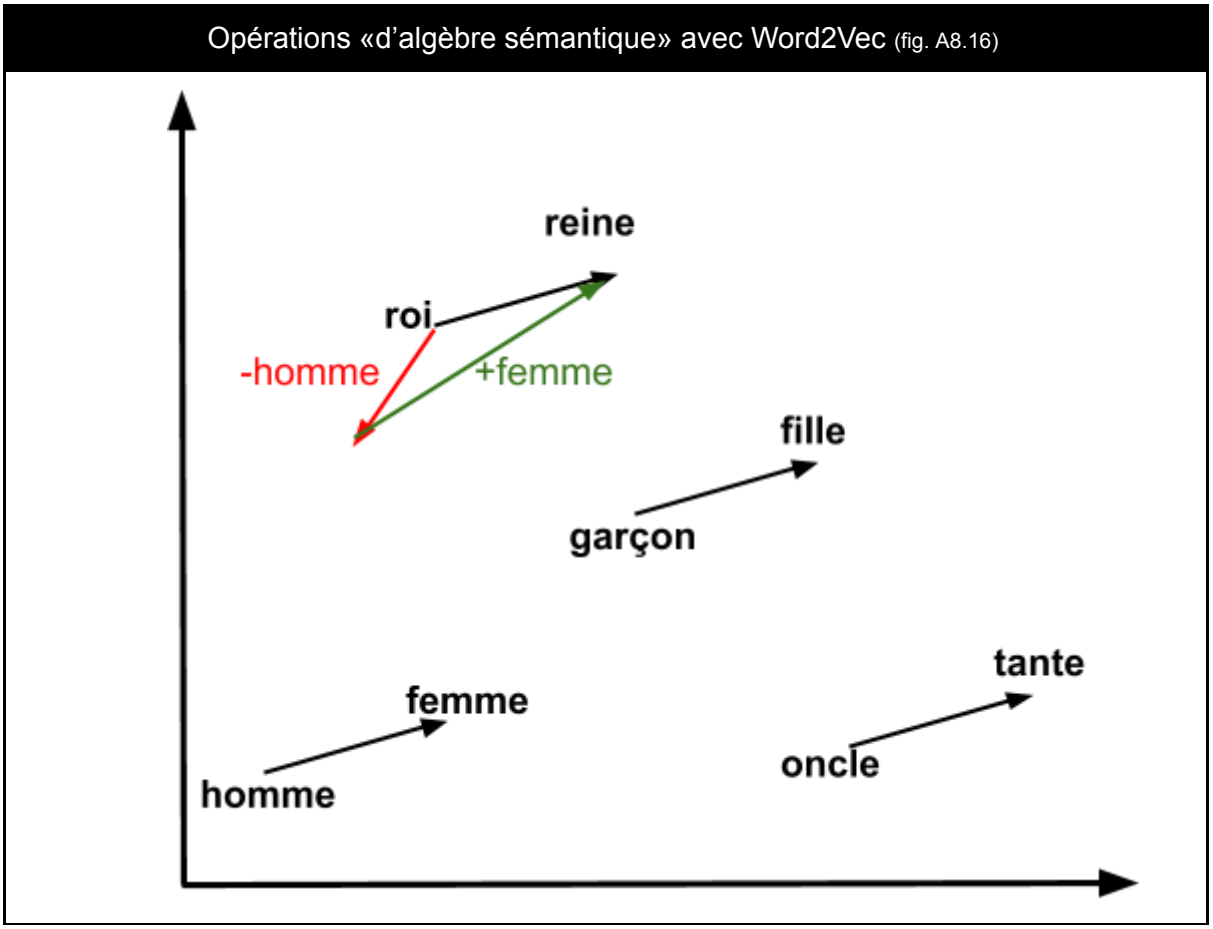
L'idée des vecteurs-mots a été améliorée par Tomáš Mikolov qui a fait un stage au laboratoire MILA de l'UdeM en 2011, puis est allé travailler chez Google. Il en a résulté une famille d'algorithmes communément appelée «Word2Vec» qui est sortie des labos de Google en 2013 [Mikolov et al, 2013a], [Mikolov et al, 2013b]. La principale innovation a été d'accélérer l'extraction des vecteurs-mots, entre autres par l'emploi astucieux d'approximations. Ceci allait permettre la production de vecteurs-mots à large échelle.

En gros, Word2Vec rapproche les mots qui apparaissent dans des ”contextes" similaires et éloigne les mots qui se trouvent dans des contextes différents en ajustant des nombres dans le vecteur. Word2Vec retourne les mots qui cooccurrent dans des contextes similaires.

Le but de Word2Vec est de rapprocher les vecteurs de mots similaires dans l'espace vectoriel. Word2Vec crée des vecteurs qui sont des représentations distribuées des mots, à partir de caractéristiques telles que le contexte de chaque mot individuel. Les représentations vectorielles captent des similitudes à différents niveaux (morphologiques, syntaxiques, sémantiques) sans intervention humaine [Deeplearning4J, 2016], [Goldberg & Levy, 2014], [Levy, Goldberg & Dagan, 2015].

[303] En anglais: «embedding»

Opérations «d'algèbre sémantique» avec Word2Vec (fig. A8.16)



Ces vecteurs-mots peuvent être utilisés pour établir la relation d'un mot avec d'autres mots. Par exemple, en représentation interne, «homme» est à «femme» ce que «garçon» est à «fille», ou «oncle» à «tante». On peut également faire des opération algébrique entre les vecteurs-mots en représentation interne comme les additionner et les soustraire. Par exemple, «roi» - «homme» + «femme» ≈ «reine» [Deeplearning4J, 2016]. Aussi, la similarité entre deux vecteurs peut être mesurée par différents métriques dont le cosinus de l'angle entre les deux vecteurs [Manning & Schütze, 1999]. Word2Vec, ainsi que d'autres représentations à base de vecteurs-mots, semblent capturer des notions comme le genre mais aussi la géographie et d'autres. On aura , «Paris» - «France» + «Italie» ≈ «Rome». Cette algèbre «sémantique» est aujourd'hui fortement contestée.

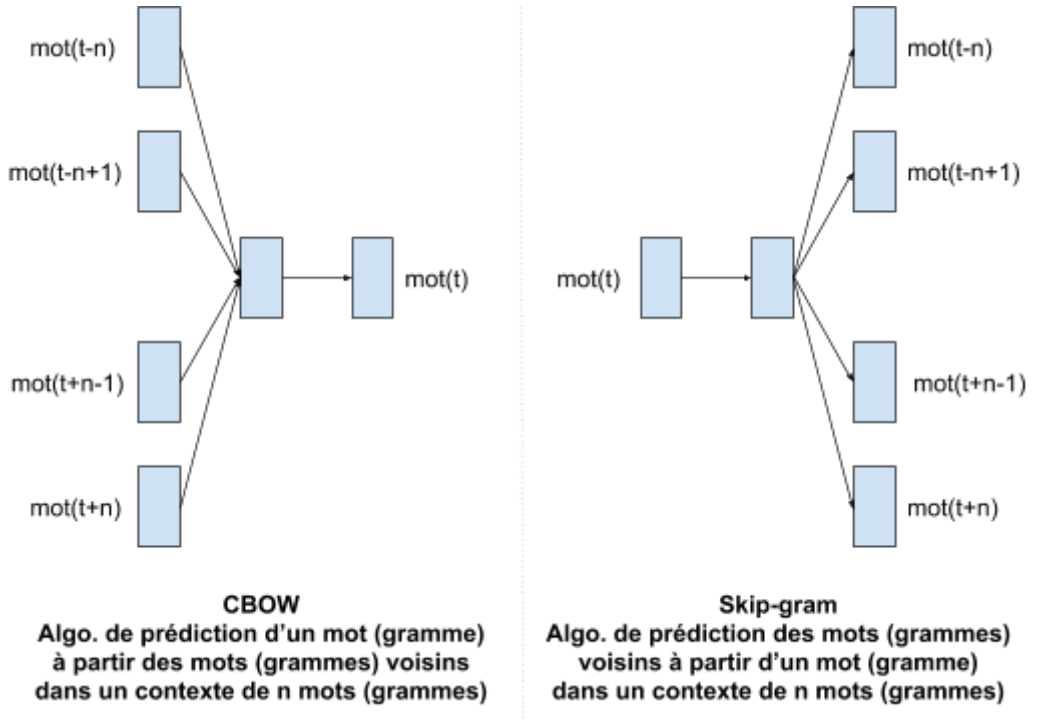
Il y a deux variantes de Word2Vec: un premier algorithme qui prédit un mot à partir de ses voisins situés dans une fenêtre de part et d'autres du mot appelé algorithme de prédiction du gramme[304], et un second algorithme que nous appelons algorithme des grammes-voisins[305] qui prédit les voisins d'un mot ou gramme (Skip-gram).

[304] En anglais: «continuous bag of words», «CBOW». Aussi en français:« algorithme du gramme» pour un algorithme de prédiction d'un mot ou gramme à partir des mots ou grammes voisins.
[305] En anglais: «Skip-gram». Aussi en français: «algorithme de prédiction des grammes-voisins»

Les vecteurs-mots sont générés par des réseaux de neurones classiques ne comportant qu'une seule couche cachée.

- Algorithme de prédiction d'un gramme à partir des grammes voisins[306].
- Algorithme de prédiction des grammes voisins à partir d'un gramme (Skip-gram)

Algorithme Word2Vec (fig. A8.17)



Dans le cas de l'algorithme des grammes-voisins, exprimons mathématiquement la prédiction d'un mot dans le contexte (t+n) étant donné l'observation du mot en position (t), $Rep_{interne}$ et Rep_{sortie} sont les vecteurs de représentation interne et en sortie.

Prédiction d'un mot avec l'algorithme des grammes-voisins (fig. A8.18)

$$p(mot(t+n) \mid mot(t)) = \frac{\exp {Rep_{interne}(mot(t))}^T \, Rep_{sortie}(mot(t+n))}{\sum_{mot'(t+n) \, \epsilon \, V} \exp {Rep_{interne}(mot(t))}^T \, Rep_{sortie}(mot'(t+n))}$$

Le calcul du dénominateur qui sert pour la normalisation est un problème car il requiert un calcul de l'ordre de la taille du lexique $O(|V|)$ pour chaque mot.

[306] En anglais: «Continuous Bag Of Words», «CBOW»

Plusieurs solutions ont été proposées pour résoudre ce problème. Deux approches ont été essayées, une première consiste à optimiser le calcul du dénominateur, la seconde repose sur l'idée d'approximer le calcul plutôt que le faire au complet.

Mikolov eut l'idée de modifier la fonction objectif (ou fonction de coût) à optimiser en échantillonnant des exemples (mot, contexte) effectivement observés et des exemples jamais observés puisque générés aléatoirement ce qu'il a appelé un échantillonnage négatif [307]. L'optimisation de cette nouvelle fonction objective permet de regrouper les exemples observés et disperser les exemples non observés ce qui accélère beaucoup l'algorithme.

A8.8.3 GloVe et fastText

Il existe plusieurs variantes de vecteurs-mots et d'algorithmes pour les produire. Une représentation par vecteurs-mots largement utilisée est GloVe (Global Vectors) qui se base sur le calcul de matrices de cooccurrences [Pennington, Socher, Manning, 2014].

FastText est un outil développé par Facebook pour la génération de vecteurs-mots qui fait appel à de l'information sublexicale[308], c'est à dire sur les parties du mot, ce qui facilite le traitement de la morphologie. fastText est disponible en 294 langues [Bojanowski et al, 2017].

A8.8.4 Du vecteur-mot au vecteur de pensée

On peut étendre le concept de vecteur de contexte à partir du mot (vecteur-mot) jusqu'à l'échelle de la phrase (vecteur-phrase), du paragraphe (vecteur-paragraphe) et du document (vecteur-document)[309]. On parle ici de représentations vectorielles riches et continues. Certains ont poussé l'analogie juqu'à parler de vecteur de pensée[310].

A8.8.5 Équivalence approches distributionnelles classiques et vecteurs-mots

Grâce aux travaux de [Levy, Goldberg & Dagan, 2015], nous savons que les modèles distributionnels traditionnels qui se basent sur la cooccurrence de mots (le comptage) sont en fait équivalents aux modèles à base de vecteurs-mots [Moody, 2017].

Les différences observées en faveur des vecteurs-mots sont le résultat de détails d'implémentation et d'optimisation des hyperparamètres. En tenant compte de ces différences, [Levy, Goldberg & Dagan, 2015] ont montré que les performances des deux approches sont tout à fait similaires.

---

[307] En anglais: «negative sampling»
[308] En anglais: «subword information», «sublexical information». Aussi en français: «sous-lexical»
[309] En anglais: «sentence vector», «paragraph vector», «document vector»
[310] En anglais: «thought vector». Aussi en français: «pensée vectorielle».

Un algorithme comme Word2Vec[311] peut être assimilé à une technique de réduction de la dimension car il permet de passer d'un encodage de type à un bit discriminant[312] impliquant un vecteur de très haute dimension (10^4 à 10^5 dimensions) vers un encodage beaucoup plus compact (10^2 à 10^3 dimensions). Le principe sous-jacent est de fusionner les attributs (les dimensions) fortement corrélées.

Aussi, la décomposition en valeurs singulières (DVS)[313] est une méthode générale de factorisation de matrice par approximation numérique qui peut être utilisée pour obtenir une matrice de plus faible dimension mais plus dense [Golub & Reinsch, 1970]. La DVS est une méthode de réduction de la dimension très répandue. Rappelons que la décomposition en valeurs singulières est à la base de la l'indexation sémantique latente[314] et de l'analyse sémantique latente[315]. L'analyse sémantique latente fut l'objet d'un dépôt de brevet en 1988 qui est maintenant dans le domaine public [Deerwester et al, 1990].

La plupart des algorithmes qui crée des vecteurs-mots peuvent être formulés sous la forme d'approximations de matrices de rang réduit, soit explicitement ou implicitement [Li et al, 2015]. Par exemple, [Goldberg & Levy, 2014] ont montré que Word2Vec fait implicitement la factorisation d'une matrice d'information mutuelle spécifique[316].

A8.9 Ne pas confondre vecteurs-mots et apprentissage profond

Beaucoup de gens confondent vecteurs-mots et apprentissage profond et cela mérite une explication.

Premièrement, les vecteurs-mots sont davantage des enrichissements statiques que de véritables modèles dans le sens prédictif ou génératif du terme. Par définition un modèle de langue[317] sert à prédire, par exemple le prochain mot dans un texte (modèle prédictif) ou à générer un exemple (modèle génératif).

Deuxièmement, les vecteurs-mots sont produits par des réseaux de neurones classiques ne comportant qu'une seule couche cachée ou encore avec des algorithmes de factorisation matricielle [Goldberg & Levy, 2014]. On parle ici d'apprentissage non supervisé.

Les vecteurs-mots n'ont rien à voir avec l'apprentissage profond si ce n'est qu'ils servent à enrichir les entrées de ces modèles et surtout à réduire la dimension des données à l'entrée des modèles. En effet, la construction de vecteurs-mots peut être assimilée à une technique de réduction de la dimension des données linguistiques. Malgré que ce ne soient pas des

[311] Note: Word2Vec n'est pas un modèle de langue capable de prédire le prochain mot dans un texte.
[312] En anglais: «one-hot»
[313] En anglais: «singular value decomposition», «SVD»
[314] En anglais: «latent semantic indexing», «LSI»
[315] En anglais: «latent semantic analysis», «LSA»
[316] En anglais: «pointwise mutual information», «PMI»
[317] En anglais: «language model», «LM»

modèles proprement dits, les vecteurs-mots sont considérés comme une forme d'apprentissage par transfert (voir section 1.3.7 et l'Annexe A.8.8.1). En fait, les vecteurs-mots sont utilisés pour enrichir les données textuelles pour l'entraînement de modèles par des algorithmes d'apprentissage classiques ou profonds.

Les vecteurs-mots sont utilisés pour représenter les mots en entrée dans les réseaux de neurones profonds mais leur création comme telle se base sur des réseaux de neurones peu profonds. La représentation Word2Vec se calcule avec un réseau neuronal à deux couches qui traite en entrée un gros corpus de texte et génère en sortie un ensemble de vecteurs-mots caractéristiques des mots du corpus.

On peut également produire des vecteurs-mots par la factorisation en une matrice de dimension inférieure de la matrice de cooccurrences des mots, une forme de réduction de la dimension en utilisant la descente de gradient stochastique.

A8.10 Avantages des vecteurs-mots

Les vecteurs-mots simplifient grandement la chaîne de traitement. Les vecteurs-mots ne nécessitent pas d'annotation de corpus car ils font appel à des algorithmes non-supervisés. Les vecteurs-mots préentraînés sont directement utilisables dans des applications. .

Avantages des vecteurs-mots - Citation(fig. A8.19)
"The main benefit of the dense representations is generalization power" … "One benefit of using dense and low-dimensional vectors is computational: the majority of neural network toolkits do not play well with very high-dimensional, sparse vectors. [Goldberg, 2017]

A8.11 Inconvénients des vecteurs-mots

Les vecteurs-mots ne tiennent généralement pas compte de la morphologie des mots. Ceci peut entraîner une incapacité à reconnaître que deux mots sont proches. Par exemple, «va» et «allons». Une solution serait la lemmatisation. Aussi, les vecteurs-mots ne traitent pas les unités nominales complexes (UNC) ou expressions nominales[318] comme «carte à puce». Une solution serait d'utiliser un extracteur d'UNC. Par contre, l'emploi de la lemmatisation et l'extraction d'unités nominales complexes sur la masse des données ralentiraient les traitements, en ajoutant au temps d'exécution.

L'absence de traitement des mots rares et les mots hors vocabulaire. Typiquement, ces mots sont affectés de l'étiquette «inconnu» (unknown, UNK) et se voient tous attribuer le même vecteur, ce qui fragilise le modèle si le nombre de mots hors vocabulaire est important.

[318] En anglais: «multiwords expressions»

Mais, l'inconvénient le plus important des vecteurs-mots est qu'ils ne tiennent pas compte de la polysémie. La plupart des formalismes de vecteurs-mots mélangent les différents sens possibles. C'est un problème connu et plusieurs travaux récents tentent d'y remédier. Une solution au problème de polysémie consisterait à utiliser un algorithme de groupage de données sur les résultats de Word2Vec. Avec Word2Vecf [Melamud, Levy & Dagan, 2015] modifie la fonction de similarité dans Word2Vec pour la rendre sensible au contexte. AdaGram [Bartunov et al, 2016] modifie Word2Vec au niveau de l'échantillonnage et par l'ajout d'un algorithme de groupage apparenté à EM (espérance-maximisation). Cela permet d'apprendre plusieurs représentations par mot capturant ainsi différents sens.

Nous en venons à un problème récurrent tout au long de la présente étude, soit la difficulté de traiter des données massives. En effet, pour bien fonctionner, les vecteurs-mots requièrent d'énorme corpus d'entraînement, de l'ordre du milliard de mots. Il existe quelques corpus libres et gratuits comme Wikipédia et l'initiative WaCKy (The Web-As-Corpus Kool Ynitiative) [Baroni et al, 2009].

La vectorisation des mots, des phrases, des paragraphes et des documents élargit l'usage des vecteurs-mots comme entrées enrichies pour les tâches en TLN. Ces vecteurs sont appris sur de grands corpus non étiquetés. Ceci injecte des connaissances a priori dans des tâches d'apprentissage supervisé qui nécessitent ainsi moins de données.

A8.12 Vecteurs-mots préentraînés

Pour nos expériences, nous avons utilisé l'implantation Gensim en Python [Řehůřek & Sojka, 2010][319]. Nous avons également utilisé des vecteurs-mots préentrainés. C'est la solution à privilégier si l'on ne dispose pas d'une grosse infrastructure de calcul.

Vecteurs-mots préentraînés (fig. A8.20)	
Ressource	URL
fastText - Facebook	https://github.com/facebookresearch/fastText/blob/master/docs/crawl-vectors.md
Word2Vec - Google	https://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTlSS21pQmM/edit
GloVe - Stanford U.	https://nlp.stanford.edu/projects/glove/
Word2Vec français - J.P. Fauconnier	http://fauconnier.github.io/#data
word2vec / fastText - spaCy	https://spacy.io/usage/vectors-similarity
word2vec/ fastText / doc2vec - gensim	https://radimrehurek.com/gensim/models/word2vec.html

[319] Note: Code source en licence libre disponible: https://github.com/RaRe-Technologies/gensim

D’autres ressources apparentés aux vecteurs-mots mais beaucoup plus riches, prenant en compte la polysémie et entraînés sur de plus grandes masses de données sont apparus en 2018 et 2019. Citons les travaux du Allen Institute for Artificial Intelligence (A2I) avec ELMo [Peters et al, 2018], et de Fast.ai avec ULMFiT (Universal Language Model Fine-tuning for Text - classification) [Howard & Ruder, 2018].

A8.13 Modèles de langue préentraînés

Encore plus, récemment, de gros modèles de langue[320] préentraînés on été utilisés avec des performances conformes à l’état de l’art sur un large éventail de tâches en traitement de la langue naturelle (TLN). Un modèle de langue est un modèle génératif, dans le sens de la prédiction du prochain mot dans un texte.

À la fin de 2018 et en 2019, une pléthore de modèles préentraînés sont apparus à chaque fois plus gros et plus performants. Par exemple, BERT (Bidirectional Encoder Representations from Transformers) annoncé en octobre 2018 par l’équipe de Google AI Language [Devlin et al, 2018] et encore plus récemment XLNet [Yang et al, 2019]. Citons également OpenAI avec GPT (Generative Pre-training Transformers) et GPT-2 aussi connus sous le nom de modèle Transformers [Radford et al, 2018], [Radford et al, 2019].

Ces gros modèles préentraînés ambitionnent jouer le même rôle avec l’apprentissage par transfert en TLN que le rôle joué par les modèles préentraînés en vision. Deux billets de blogue de Sebastian Ruder, maintenant chercheur à DeepMind, explorent ce paysage en évolution [Ruder, 2018], [Ruder, 2019]. On peut s’attendre à de nombreux nouveaux développements dans les prochains mois et années.

[320] En anglais: «language model», «LM»

Annexe 9 - Guide pratique de l'apprentissage par transfert

A9.1 Idée générale

L'idée derrière l'apprentissage par tranfert[321] est de se baser sur des modèles préentraînés ou d'autres enrichissements produits par des algorithmes d'apprentissage pour facilliter l'accomplissement de tâches similaires par un transfert d'apprentissage ou transfert de connaissances.

Bien qu'il existe plusieurs formes d'apprentissage par transfert appliquées aux tâches de traitement de la langue naturelle, deux formes principales se dégagent:

1. l'enrichissement par vecteurs-mots[322]
2. l'utilisation de gros modèles préentraînés

On peut même combiner les deux.

Malgré que ce ne soient pas des modèles proprement dits, les vecteurs-mots sont considérés par plusieurs comme une forme d'apprentissage par transfert. Mais il s'agit essentiellement d'enrichissement par ajout d'attributs statiques. Les vecteurs-mots sont discutés à la section A8.8 Modèles neuronaux de la langue.

A9.2 Modèles préentraînés

L'emploi d'un gros modèle préentraîné s'avère utile lorsqu'on a peu de données. Rappelons que le résultat de l'entraînement d'un modèle d'apprentissage profond est une architecture de réseau de neurones à plusieurs couches avec tous les paramètres (ou poids) associés. Ici, on ne parle plus seulement d'ajouter des attributs statiques comme avec les vecteurs-mots, mais bien de se baser sur un modèle neuronal complet et capable de s'adapter par entraînement.

Ici, l'apprentissage par transfert est une technique où, au lieu d'entraîner un modèle à partir de zéro, on réutilise un gros modèle préentraîné et on ajuste ensuite ce modèle pour réaliser une tâche similaire. Le degré de similarité entre les tâches est laissé à l'appréciation de l'utilisateur.

---

[321] En anglais: «transfer learning». Sémantiquement on est plus près d'un «transfert d'apprentissage» ou «transfert de connaissances» mais on utilise «apprentissage par transfert» pour une raison d'uniformité terminologique avec des termes tels que «apprentissage par renforcement» ou «apprentissage par imitation», «apprentissage par fonction état-action», etc.

[322] En anglais: «embeddings». Aussi en français: «vecteurs continus», «représentations vectorielles continues», «plongements», «plongements vectoriels» et «vecteurs-mots» quand l'enrichissement porte sur un mot spécifique.

Typiquement, on spécialisera et raffinera un gros modèle préentrainé en modifiant par entraînement les paramètres de ses dernières couches avec des données d'un corpus spécifique à la nouvelle tâche ou corpus applicatif[323]. Le gros modèle est préentrainé avec un corpus de textes de plusieurs gigaoctets, le corpus de référence[324], sur une tâche générique.

Une tâche générique type pour le TLN est la prédiction du prochain mot dans une séquence, par exemple prédire le prochain mot dans une phrase ou plus généralement le prochain mot dans un texte. Un tel modèle est appelé un «modèle de langue»[325]. Non seulement ces modèles sont capables de prédire le prochain mot dans un texte mais ils capturent également des connaissances linguistiques utiles à diifférentes autres tâches.

Modèle de langue - définition simple (fig. A9.1)
Un modèle de langue est juste un modèle capable de générer le mot suivant.

Un modèle de langue est un modèle génératif [326]probabiliste, il produit une distribution de probabilité sur les différents mots possibles. À partir d'une séquences de mots, $m_1, m_2, \ldots, m_n$ un modèle de langue sera capable de lui assigner une probabilité $P(m_1, m_2, \ldots, m_n)$. Une fois entraîné, un modèle de langue qui est juste un décodeur n'a pas besoin de données d'entrée proprement dites, juste d'un germe et il produira les suites probables.

Le gros avantage d'un modèle de langue est qu'on a pas besoin d'annoter le corpus manuellement pur le créer. L'annotation est implicite car il suffit d'entraîner le modèle en cachant des mots dans des textes existants. On entraîne également ces gros modèles à prédire des mots dans des phrases à trous. Rappelons que l'entraînement d'un modèle de référence exige des ressources de calcul considérables.

En contraste, l'adaptation d'un modèle préentraîné requiert normalement beaucoup moins de calcul, typiquement de l'ordre de quelques heures à un jour ou deux avec un seul GPU.

A9.3 Petit guide pratique de l'apprentissage par transfert

Ci-dessous, un petit guide pratique de l'apprentissage par transfert inspiré de [Géron, 2017b] et [Chollet, 2016]. Pour plus de détails techniques et des exemples avec TensorFlow, consulter [Géron, 2017a] ou [Géron, 2017b], pour Keras [Chollet, 2016].

---

[323] En anglais «target corpus», «application corpus», «target dataset», «application dataset»
[324] En anglais: «reference corpus», «reference dataset», «source corpus», «source dataset»
[325] En anglais: «language model», «LM»
[326] Note: Nous avons un modèle qui prédit (donc prédictif) mais également génératif, ce qui peut sembler contradictoire. En fait, on peut créer un modèle prédictif à partir d'un modèle génératif. Par contre l'inverse n'est pas vrai.

Attention! Ce guide a été conçu pour les tâches de vision artificielle. L'apprentissage par transfert en langue naturelle semble quelque peu différent, mais il est encore trop tôt pour se prononcer définitivement. En gros, il semblerait que les modèles linguistiques soient moins profonds (moins de couches). Aussi, on comprend moins bien la nature des attributs extraits automatiquement par le réseau et la progression des niveaux d'abstraction d'une couche de représentation à la suivante.

1) Objectif: Nous voulons réaliser un modèle spécialisé pour une application ou une tâche précise en nous basant sur un gros modèle préentrainé que l'on appellera le modèle de référence[327];
2) Avant tout, il faut que les tâches se ressemblent, du moins partagent des connaissances au bas niveau;
3) Puis, il faut s'assurer que les entrées des deux réseaux soient compatibles. Au besoin, il faudra mettre en place le prétraitement nécessaire pour convertir les données d'entrée du modèle spécialisé vers un format accepté par le modèle générique;
4) Typiquement, il faut retrancher la dernière couche de neurones (parfois les deux dernières) du modèle générique près de la sortie du réseau en conservant et en gelant les poids des premières couches qui sont plus près de l'entrée des données[328] . Ceci est conforme avec l'hypothèse que les premières couches du modèle de référence préentraîné ont acquis des connaissances de bas niveau plutôt génériques et transférables à d'autres tâches du même type;
5) Remplacer les couches retranchées, par de nouvelles couches de neurones et un classificateur de sortie spécifique à la nouvelle tâche;
6) Initialiser aléatoirement ces nouvelles couches tout en gardant intactes les couches inférieures du modèle générique (les paramètres y sont gelés);
7) Entraîner le modèle résultant sur un petit ensemble de données spécifiques à l'application spécialisée, le corpus applicatif.

[327] En anglais: «reference model», «generic model». Aussi en français, «modèle générique».
[328] Note: Il peut être intéressant de réentraîner tout le modèle avec plus ou moins de données applicatives, mais cela au risque que le modèle oublie trop de résultats de ses apprentissages précédents. Pour en décider, on s'assurera d'avoir une copie du réseau de référence non-modifié, puis on procédera empiriquement par essais et erreurs.

Annexe 10 - Quelques contributions montréalaises en apprentissage profond

Quand on parle de traduction automatique, il serait de bonne guerre de préciser que les avancées récentes en traduction automatique neuronale originent en grande partie des travaux pionniers du MILA et de l'équipe de Yoshua Bengio de l'UdeM qui ont été perfectionnés et industrialisés par Google.

Par exemple, les modèles de langue neuronaux[329] [Bengio et al, 2003] et l'idée de vecteur dense à l'origine de Word2Vec, des améliorations aux architectures encodeur/décodeur [Cho et al, 2014a], [Cho et al, 2014b] et particulièrement le mécanisme d'attention [Bahdanau, Cho & Bengio, 2014] qui est à la base de tous les gros modèles de langue préentraînés qui s'imposent depuis quelques temps.

Aussi la fonction ReLU qui a contribué aux succès de l'apprentissage profond dont une équipe de l'UdeM a démontré la supériorité pour l'entraînement de réseaux profonds comparativement aux fonctions d'activation couramment utilisées avant 2011, soient la sigmoïde et la tangente hyperbolique [Glorot, Bordes & Bengio, 2011].

Les réseaux récurrents à portes[330] similaires aux réseaux récurrents à longue mémoire court terme (LMCT)[331] mais plus simples et plus performants sur de petits jeux de données ont été créés en 2014 au laboratoire MILA [Chung et al, 2014].

Enfin, les réseaux antagonistes génératifs (RAG)[332] qui ont été inventés à Montréal par Ian Goodfellow alors qu'il était étudiant au doctorat à l'UdeM [Goodfellow et al, 2014].

[329] En anglais: «neural language models»
[330] En anglais: «gated recurrent unit», «GRU»
[331] En anglais: «long short-term memory», «LSTM»
[332] En anglais: «generative adversarial network», «GAN». Aussi en français: «réseau génératif antagoniste»

Annexe 11 - Stochasticité, variance et données corrélées

Une observation importante avec les réseaux de neurones profonds est la grande variabilité des résultats qu'ils produisent. Le même réseau entraîné avec les mêmes données peut produire des résultats passablement différents d'un entraînement à l'autre. Cette stochasticité[333] des réseaux de neurones est nécessaire à leur bon fonctionnement car elle leur permet entre autres, d'essayer différentes possibilités en cours d'entraînement et d'éviter de rester coincé dans un minimum local. Il est important de conserver cette variabilité au moment de la conception et de la mise au point d'un réseau de neurones [Brownlee, 2018a].

La plus grande partie de cette variabilité s'explique par le fait que les poids du réseau de neurones sont initialisés de manière aléatoire à chaque fois que débute l'entraînement [Brownlee, 2018a]. On pourrait même parler de sensibilité aux conditions initiales ce qui permettrait peut-être de qualifier le réseau de neurones de système chaotique. De plus, il est bien connu que l'entraînement d'un réseau par la descente de gradient stochastique[334] conduit rarement au même résultat à chaque fois. À cela il faut ajouter le comportement aléatoire de certaines formes de régularisation comme l'extinction de neurones.

Avec la bibliothèque Keras, il est possible de réduire la variabilité des résultats en fixant un germe aléatoire «`random_seed`» au démarrage de la fonction «`random`» de la bibliothèque Numpy (`np.random.seed(random_seed)`) utilisée par Keras [Chollet, 2017].

Attention! Il ne faut jamais fixer le germe aléatoire au moment de la mise au point des modèles, sinon peut-être à l'occasion pour vérifier si un changement de performance est dû au hasard ou à une modification du modèle ou des données. Toutefois, on pourra le faire une fois le comportement moyen établi par une série d'expériences avec différents germes aléatoires. On choisira alors le germe qui offre le résultat le plus proche du résultat moyen pour assurer la reproductibilité pour la mise en production.

Le moyen le plus utilisé pour se prémunir contre l'effet d'un germe aléatoire particulièrement bon ou mauvais est d'entraîner plusieurs modèles chacun avec un germe aléatoire différent et de combiner leurs prédictions selon le paradigme des méthodes ensemblistes d'apprentissage[335] [Brownlee, 2018a]. De la même manière, la validation croisée[336] est un moyen d'estimer la vraie performance d'un modèle en faisant la moyenne de la performance sur plusieurs sous-ensembles de données.

---

[333] En anglais: «randomness»
[334] En anglais: «stochastic gradient descent», «SGD»
[335] En anglais: «ensemble learning»
[336] En anglais: «cross-validation», «CV»

Cette stochasticité explique pourquoi les modèles d'apprentissage profond en production basent leurs résultats sur des ensembles de modèles pour donner des prédictions plus constantes.

Un problème n'a cessé de nous préoccuper, c'était celui que les données amplifiées devraient être fortement corrélées positivement. L'effet de ces corrélations est d'introduire davantage de variance [James et al, 2013], [WIKIPÉDIA, Multicollinearity], ce que nous avons observé expérimentalement.

Cela dit, en général la présence de données corrélées n'affectera pas les performances d'un modèle statistique, ni positivement, ni négativement. Par contre la présence de données corrélées va rallonger le temps de calcul. Elle peut également obscurcir le rôle des différents attributs et gêner la recherche de facteurs de causalité.

Tentons une courte démonstration que les données corrélées introduisent davantage de variance.

L'entraînement d'un réseau de neurones (une optimisation) donne une approximation d'une fonction non-linéaire de haute dimension. Appelons cette fonction $\hat{f}(X)$

Soit n variables aléatoires $X_i \, i = 1 \ldots n$, la variance de la moyenne de ces variables est en partant de la définition [Wikipédia, Variance]:

Variance, cas général - Formulation math. (fig. A11.1)

$$Var(\bar{X}) = Var\left(\frac{1}{n}\sum_{i=1}^{n} X_i\right) = \frac{1}{n^2}\left(\sum_{i=1}^{n}\sigma_i^2 + \sum_{i=1}^{n}\left(\sum_{j=1, j\neq i}^{n}\rho_{ij}\sigma_i\sigma_j\right)\right)$$

où σ_i^2 est la variance de X_i et ϱ_{ij} est la corrélation entre X_i et X_j lorsque les variables X_i et X_j sont non corrélées, l'équation ci-dessus devient simplement:

Variance, données non corrélées - Formulation math. (fig. A11.2)

$$Var(\bar{X}) = \frac{1}{n^2}\left(\sum_{i=1}^{n}\sigma_i^2\right)$$

Par contre, si les variables sont corrélées, la variance sera plus grande puisque le terme non-nul et positif (les données sont corrélées positivement) s'ajoute à la variance (ci-dessous):

Variance, terme ajouté pour les données corrélées - Formulation math. (fig. A11.3)
$\frac{1}{n^2}\left(\sum_{i=1}^{n}\left(\sum_{j=1,j\neq i}^{n}\rho_{ij}\sigma_i\sigma_j\right)\right)$

Maintenant si on applique la fonction approximée $\hat{f}(X)$ à des variables multidimensionnelles X_i ayant de fortes variances, on obtiendra des résultats de plus forte variance que si l'on appliquait la même fonction aléatoire $\hat{f}(X)$ à des variables aléatoires X_i de faibles variances.

La solution généralement recommandée serait soit de retirer les données trop fortement corrélées en se basant sur une analyse factorielle ou de de réduire la dimension[337] des données fortement corrélées avec un algorithme comme l'analyse en composante principale (ACP)[338].

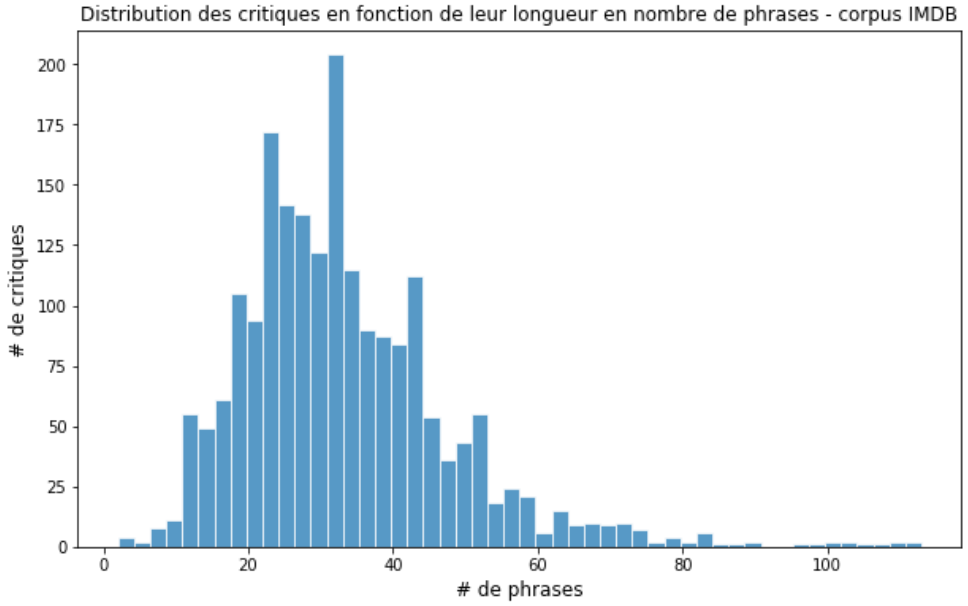
Cela dit, c'est un fait connu que les réseaux de neurones profonds se débrouillent mieux que certains algorithmes d'apprentissage classiques avec des données corrélées.

[337] En anglais: «dimension reduction», «dimensionality reduction». Le mot «dimensionnalité» ou en anglais «dimensionality» semble superflu puisque le mot dimension existe déjà. Normalement le mot «dimensionnalité» devrait désigner la propriété d'avoir une «dimension».
[338] En anglais: «principal component analysis», «PCA»

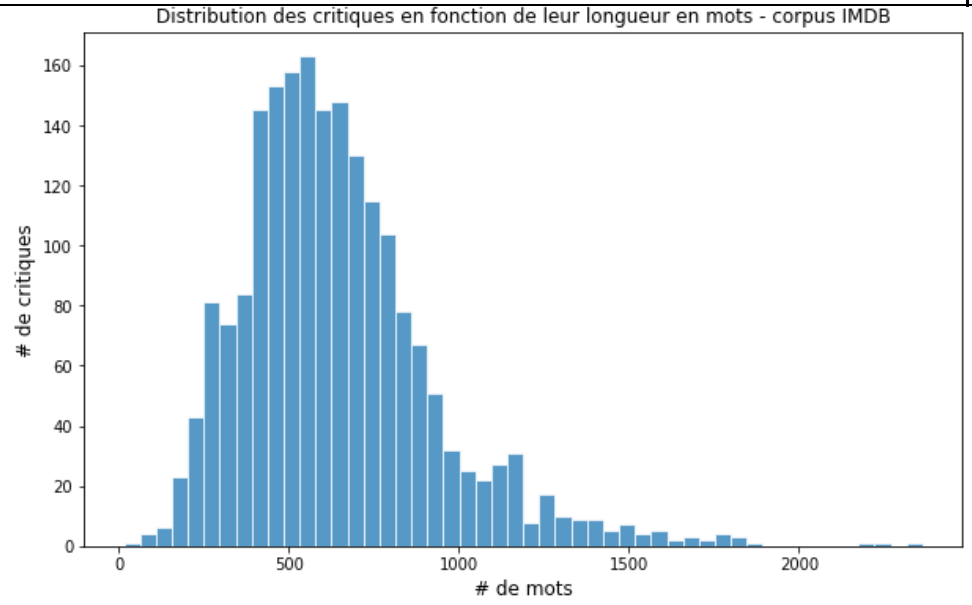
Annexe 12 - Expériences sur les données IMBD

A12.1 Exploration des données IMBD

Exploration des données - Longueur des critiques (fig. A12.1)

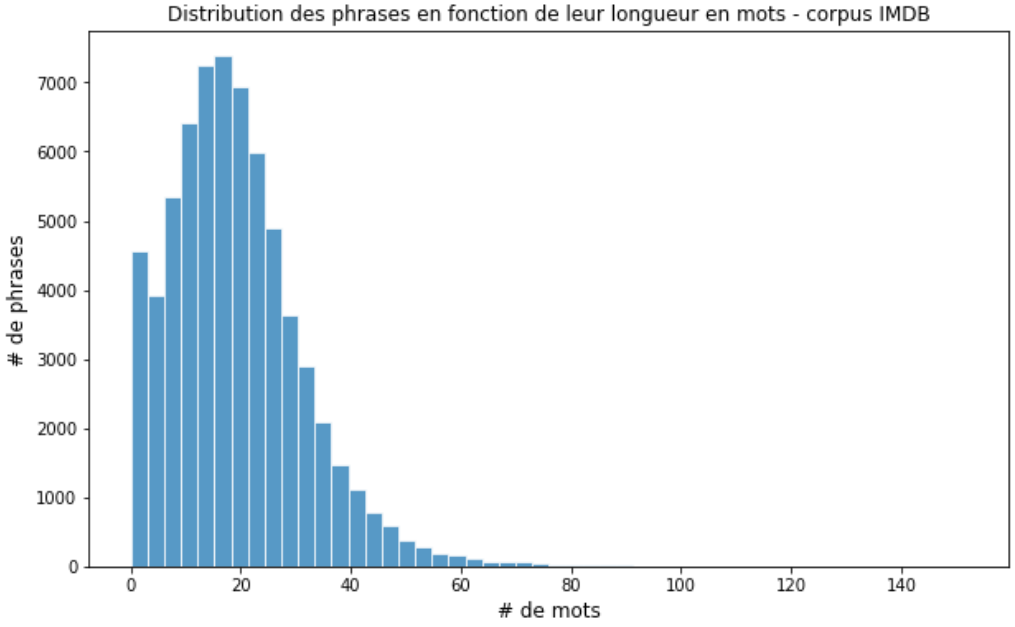


Longueur moyenne des critiques en phrases:	33
Longueur médiane des critiques en phrases:	31
L'écart-type de la longueur des critiques en phrases:	15



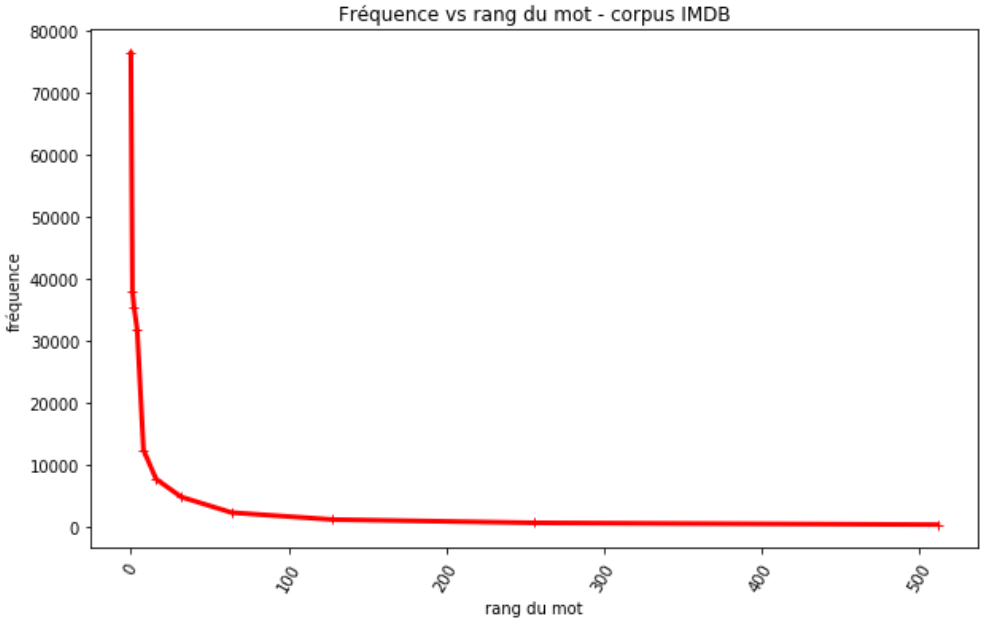
Longueur moyenne des critiques en mots:	644
Longueur médiane des critiques en mots:	602
L'écart-type de la longueur des critiques en mots:	285

Exploration des données - Longueur des phrases (fig. A12.2)



Longueur moyenne des phrases en mots:	19
Longueur médiane des phrases en mots:	18
L'écart-type de la longueur des phrases en mots:	12

Exploration des données - Fréquence des mots (fig. A12.3)



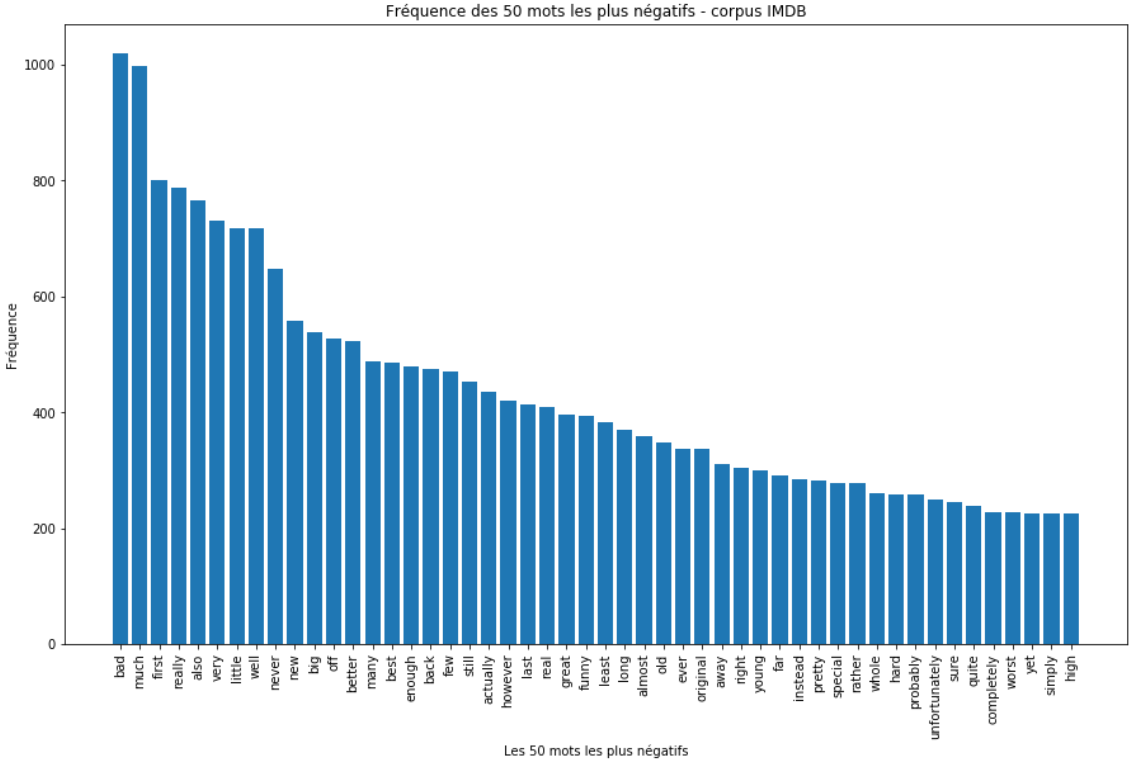
On voit que la distribution de la fréquence des mots dans le corpus IMBD suit une loi de Zipf. La loi de Zipf (découverte par le sténographe français Jean-Baptiste Estoup, puis popularisée par le linguiste américain George Kingsley Zipf) stipule que, étant donné un large échantillon de mots utilisés, la fréquence d'un mot est inversement proportionnelle à son rang dans le tableau des fréquences. Le mot n a donc une fréquence proportionnelle à 1/n.

Exploration des données - Mots positifs (fig. A12.4)

Exploration des données - Mots négatifs (fig. A12.5)

Nuage des mots négatifs - corpus IMDB

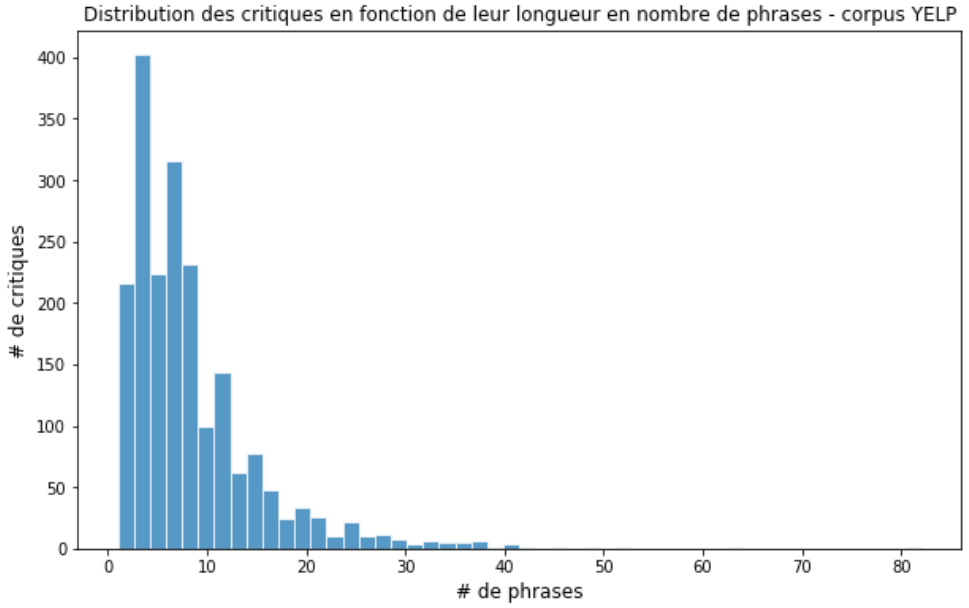
Fréquence des 50 mots les plus négatifs - corpus IMDB



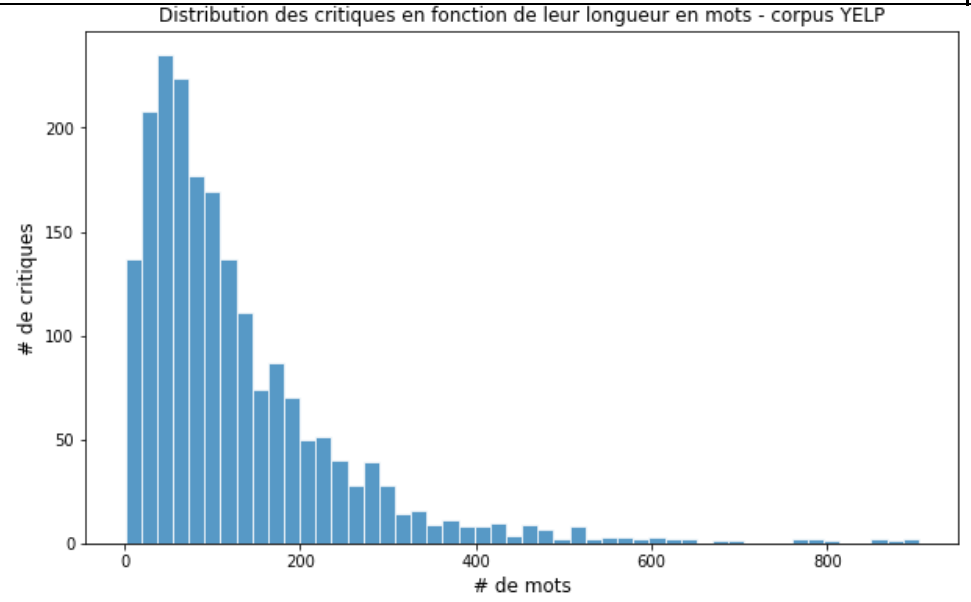
Annexe 13 - Expériences sur les données de YELP

A13.1 Exploration des données YELP

Exploration des données - Longueur des critiques (fig. A13.1)

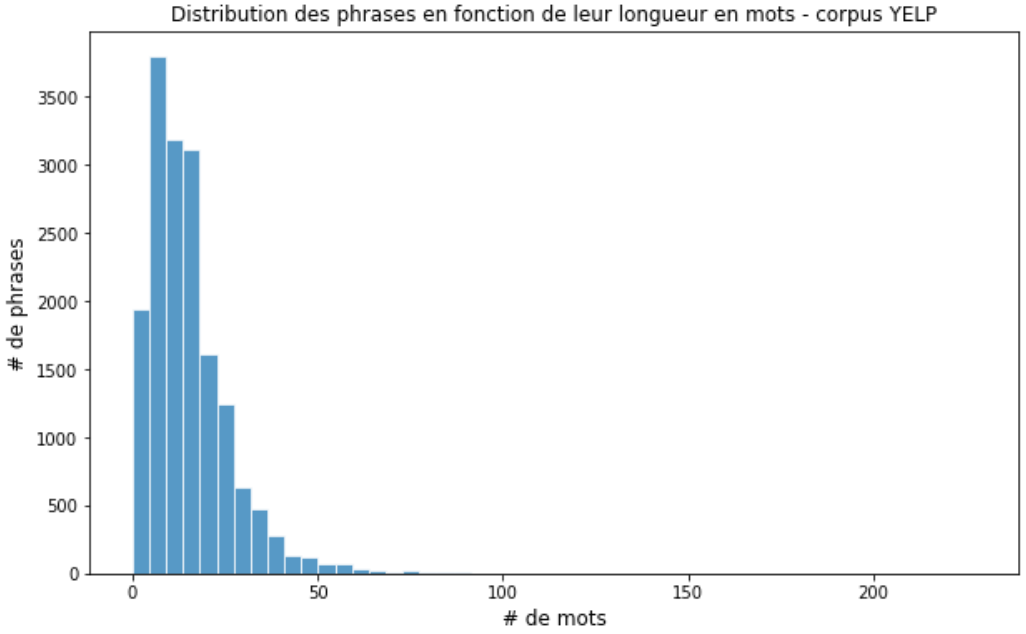


Longueur moyenne des critiques en phrases:	8
Longueur médiane des critiques en phrases:	6
L'écart-type de la longueur des critiques en phrases:	7



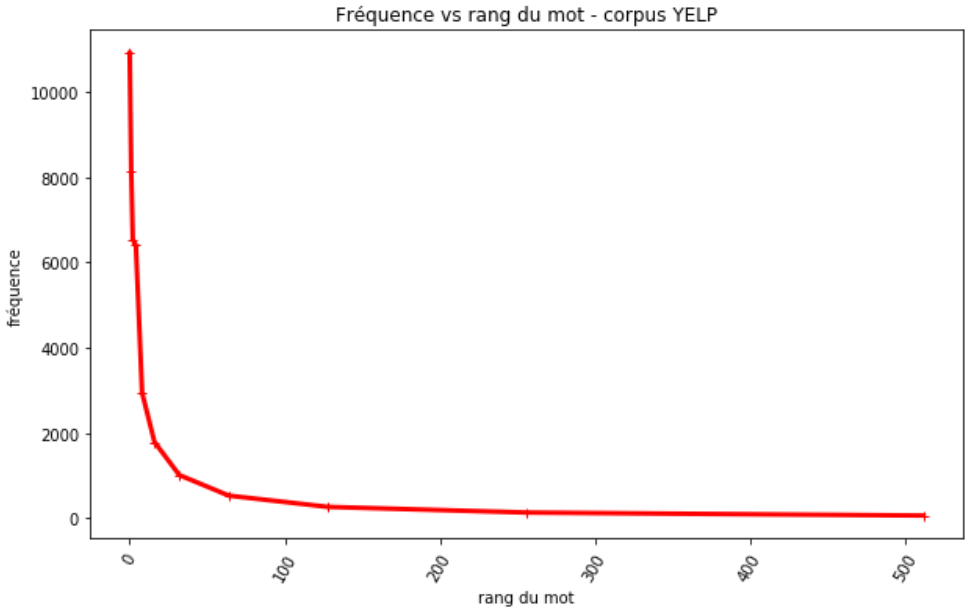
Longueur moyenne des critiques en mots:	128
Longueur médiane des critiques en mots:	94
L'écart-type de la longueur des critiques en mots:	119

Exploration des données - Longueur des phrases (fig. A13.2)



Longueur moyenne des phrases en mots:	15
Longueur médiane des phrases en mots:	13
L'écart-type de la longueur des phrases en mots:	12

Exploration des données - Fréquence des mots (fig. A13.3)

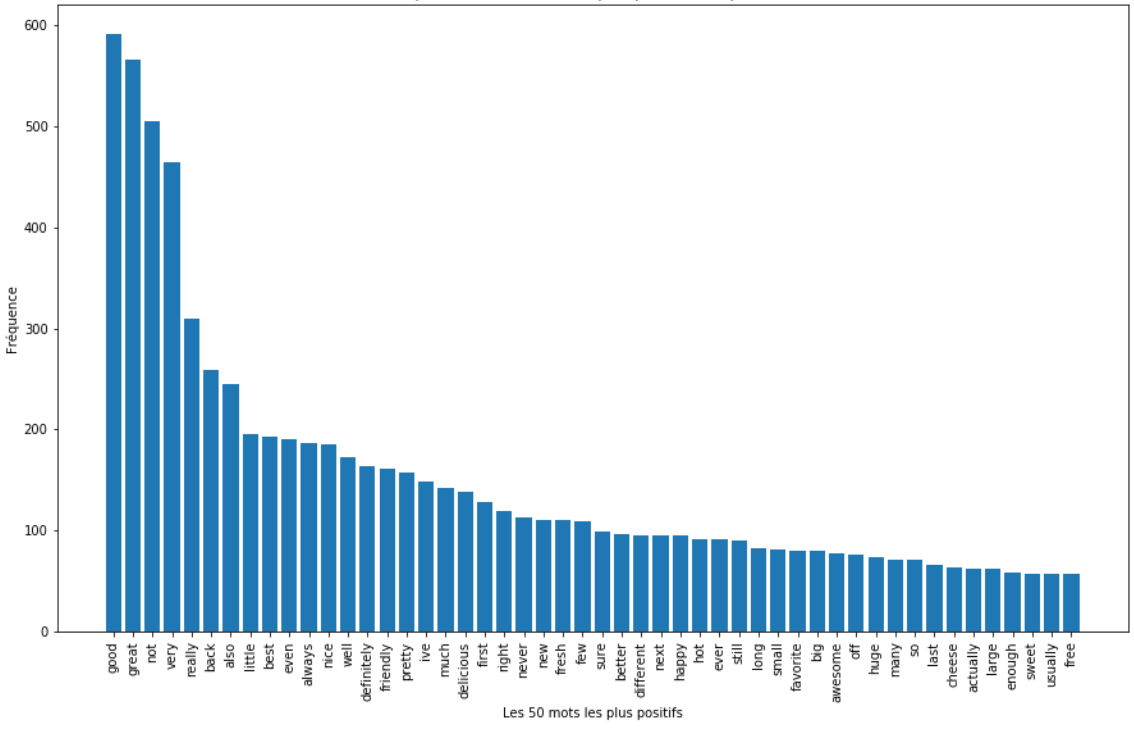


On voit que la distribution de la fréquence des mots dans le corpus IMBD suit une loi de Zipf. La loi de Zipf (découverte par le sténographe français Jean-Baptiste Estoup, puis popularisée par le linguiste américain George Kingsley Zipf) stipule que, étant donné un large échantillon de mots utilisés, la fréquence d'un mot est inversement proportionnelle à son rang dans le tableau des fréquences. Le mot n a donc une fréquence proportionnelle à 1/n.

Exploration des données - Mots positifs (fig. A13.4)

Nuage des mots positifs - corpus YELP

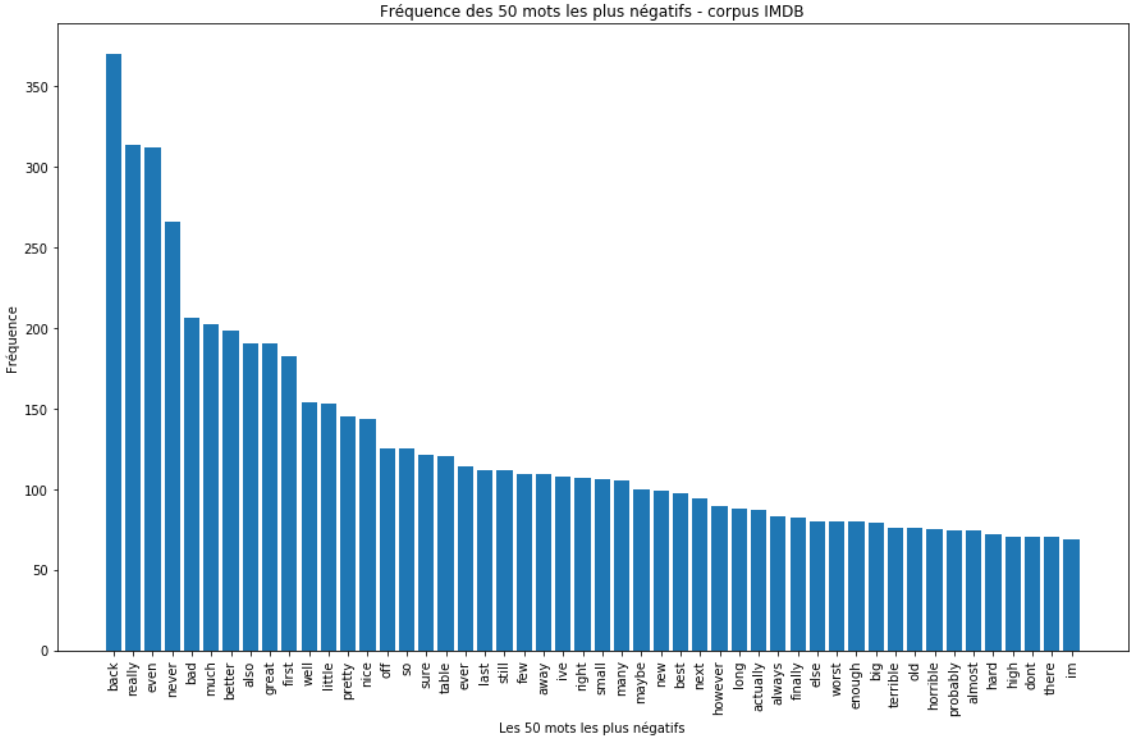
Fréquence des 50 mots les plus positifs - corpus YELP



Exploration des données - Mots négatifs (fig. A13.5)

Nuage des mots négatifs - corpus YELP

Fréquence des 50 mots les plus négatifs - corpus IMDB



A13.2 Résultats obtenus en amplifiant les données YELP

Exactitude de la prédiction de la polarité (fig. A13.6)					
Technique d'amplification textuelle	**Algo. classique**	**Architecture du réseau de neurones**			
	XGBoost	Perceptron multicouche	Réseau convolutif 1D	Réseau récurrent LMCT	Réseau récurrent biLMCT
0- Aucune base de référence	T:**82.50** E:92.00	T:**88.42±0.36** E: 96.96±0.12	T:**85.75±1.50** E: 94.94±0.49	T:**80.80±2.09** E: 93.26±1.31	T:**80.80±4.05** E: 91.45±5.12
1- Bruit textuel	T:**82.00** E:91.71	T:**89.62±0.50** E:99.99±0.00	T:**88.38±1.51** E:100.00±0.01	T:**82.90±1.02** E:99.96±0.04	T:**83.30±2.20** E: 99.89±0.08
2- Fautes d'orthographe	T:**84.00** E:91.70	T:**87.72±0.64** E:99.99±0.00	T:**87.30±1.52** E:99.99±0.00	T:**81.70±0.24** E: 99.94±0.03	T:**82.80±2.34** E: 99.91±0.07
3- Subst. lex. AdaGram[339]	N.D.	N.D.	N.D.	N.D.	N.D.
4- Subst. lex. WordNet	T:**80.00** E:80.71	T:**85.72±1.01** E:98.78±00.08	T:**85.75±1.71** E:98.89±0.16	T:**79.70±2.98** E: 95.53±1.46	T:**80.80±2.54** E: 93.91±1.10
5- Paraphrases exp. rég.	T:**82.50** E:92.77	T:**88.35±0.39** E:99.95±0.00	T:**88.35±1.23** E:100.00±0.00	T:**83.20±1.21** E: 99.97±0.02	T:**80.50±3.33** E: 99.86±0.13
6- Paraphrases arbres syntax.	T:**83.50** E:82.07	T:**85.05±1.09** E:97.95±0.18	T:**87.45±1.08** E:97.16±0.27	T:**82.40±1.07** E: 92.78±1.50	T:**83.30±0.51** E: 93.54±0.59
7- Rétro-traduction	T:**84.50** E:90.67	T:**88.92±0.78** E:99.88±0.01	T:**89.42±1.21** E:99.86±0.04	T:**84.90±1.53** E: 99.41±0.27	T:**84.08±2.76** E: 99.21±0.68
1 + 2	T:84.50 E:91.27	T:**88.65±0.59** E:100.00±0.00	T:**89.25±2.00** E:99.99±0.01	T:**82.30±3.14** E: 99.60±0.26	T:**80.25±2.63** E: 99.76±0.12
3 + 4	N.D.	N.D.	N.D.	N.D.	N.D.
1 + 2 + 5	T:**85.50** E:91.57	T:**88.85±0.45** E:99.99±0.00	T:**89.25±1.37** E:100.00±0.00	T:**84.20±1.96** E: 99.87±0.09	T:**81.97±1.65** E: 99.90±0.08
4 + 6 + 7	T:**83.50** E:82.19	T:**87.20±0.94** E:99.41±0.05	T:**81.83±2.97** E:98.46±0.59	T:**79.40±1.28** E: 95.04±1.03	N.D.
1+2+3+4+5+6+7	T:**84.00** E:87.31	T:**84.88±1.39** E: 99.76±0.03	T:**83.08±3.04** E:98.26±0.31	T:**80.20±1.63** E:97.49±0.34	N.D.

Le tableau des résultats d'expériences répétées 20 fois pour chacune des différentes techniques d'ADT (lignes) avec différentes architectures courantes de réseaux de neurones (colonnes). Avec T: données test, E: données d'entraînement, mesure d'exactitude avec l'écart-type. En vert les résultats statistiquement significatifs, en rouge les résultats non statistiquement significatifs.

[339] Note: Pour cette série d'expériences supplémentaires, la substitution lexicale par vecteurs-mots désambiguïsés avec l'algorithme AdaGram n'a pas été réalisée, d'où la mention N.D. (non disponible)

Annexe 14 - Temps de calcul

Temps de calcul d'une itération sur l'infrastructure partagée de Calcul Québec (fig. A14.1)			
Techniques d'ADT		**Modèle entraîné**	**Entraînement en heures**
1	bruit textuel	LMCT	7
2	fautes d'orthographe	LMCT	5
3	synonymes AdaGram	LMCT	4
4	synonymes WordNet	LMCT	4
5	paraphrases regex	LMCT	5
6	paraphrases grammaire	LMCT	5
7	paraphrases rétrotraduction	LMCT	5
1 + 2		LMCT	5
3 + 4		LMCT	5
1 + 2 + 5		LMCT	10
3 + 4 + 6 + 7		LMCT	9
1 + 2 + 3 + 4 + 5 + 6 + 7		LMCT	10
1	bruit textuel	biLMCT	9
2	fautes d'orthographe	biLMCT	8
3	synonymes AdaGram	biLMCT	6
4	synonymes WordNet	biLMCT	6
5	paraphrases regex	biLMCT	8
6	paraphrases grammaire	biLMCT	7
7	paraphrases rétrotraduction	biLMCT	8
1 + 2		biLMCT	10
3 + 4		biLMCT	10
1 + 2 + 5		biLMCT	15
3 + 4 + 6 + 7		biLMCT	15
1 + 2 + 3 + 4 + 5 + 6 + 7		biLMCT	16