Table des matières

Liste des tableaux

Liste des figures

1. Introduction

La majorité des systèmes de help desks proposés aux PME offrent généralement des outils d'analyse qui permettent de déterminer la qualité de prestation d'agent(e)s de support sur base de la quantité de tickets traités et du temps de réponse. En revanche, peu d'outils semblent réellement offrir la possibilité de recenser automatiquement la qualité d'une prestation du point de vue de l'utilisateur. En effet, souvent, la possibilité de recueillir des statistiques sur la qualité du service requiert l'envoi d'enquêtes de satisfaction dont les réponses, selon les régions et les cultures peuvent être plus ou moins sévères et donc ne pas représenter la réalité de l'interaction. De plus, les émotions ne sont pas non plus toujours fidèlement retranscrites sachant que les clients disposent soudainement d'une opportunité pour donner leur avis en dehors des faits, laissant potentiellement échapper les détails de l'interaction.

C'est pour cette raison que dans le cadre de mon travail de Bachelor je m'intéresse à la classification de texte et à l'analyse de sentiments. Il sera intéressant d'analyser leur potentiel impact positif dans une entreprise. En effet, cette analyse permettrait de déterminer la satisfaction client en se basant uniquement sur l'échange réel à l'instant de même où l'interaction a lieu.

Dans ce document, je vais d'abord procéder à une analyse portant sur ce qui constitue le traitement automatique de langage naturel et l'analyse de sentiments. À la suite de cela, je vais effectuer une étude approfondie des réseaux de neurones en passant par une présentation de leurs origines et les principaux modèles qui auront mené aux architectures d'apprentissage profond. Ensuite je me focaliserai sur une architecture particulière, le réseau de neurones convolutif, qui, malgré sa mise en œuvre régulière dans le cadre du traitement d'image, se prête aussi bien aux tâches de classification de texte et d'analyse de sentiments. Je détaillerai aussi les outils courants que j'utiliserai dans le cadre de la partie pratique de ce travail. Enfin, j'effectuerai une mise en œuvre pratique de classification de texte pour démontrer et critiquer les possibilités présentées par les réseaux de neurones convolutifs dans le cadre d'une analyse de sentiments destinés aux services de support client.

2. Le traitement automatique de langage naturel

2.1 Aspects généraux

2.1.1 Définition

Dans sa définition la plus pure, le traitement automatique du langage naturel (TALN) est une discipline qui a pour but d'étudier l'ensemble des techniques qui permettent à une machine d'analyser et synthétiser le langage humain[1]. Le TALN repose sur plusieurs disciplines : l'informatique, la linguistique et l'intelligence artificielle. Il a pour but de concevoir divers outils (procédés et algorithmes) capables de traiter des données de langage naturel. Ceux-ci servent entre-autre à manipuler, puiser, et produire de l'information qui pourra ensuite être utilisée pour générer de la valeur.

2.1.2 Principes

Le plus grand défi posé par le TALN est le fait de pouvoir correctement interpréter et extraire le sens d'une séquence de termes, ce qui ne serait pas forcement possible si ces termes étaient pris individuellement. L'être humain a un grand talent pour exprimer, percevoir et interpréter les sens complexes et nuancés du langage tout en étant relativement médiocre lorsqu'il s'agit de formellement comprendre et décrire les règles qui le définissent.

Le langage est par définition intrinsèquement symbolique et discret dans le sens où il ne se limite pas à une simple séquence de caractères mais bien à de termes ou mots sensés décrire des objets, concepts, actions, événements ou idées contextuelles. Des mots comme « pomme » ou « banane » évoquent par exemple le concept du fruit ou plus généralement celui de la nourriture mais ils sont aussi symboliquement très distincts. De plus, la nature compositionnelle du langage, c'est-à-dire le nombre quasiment illimité de combinaisons de mots possibles qui peuvent être utilisés pour produire des phrases au sens particulier ajoute un degré de complexité et de défi non-négligeable pour les machines. Le sens d'une phrase peut en effet souvent être entièrement perdu si les mots ne sont pris en compte et interprétés qu'en tant qu'éléments individuels.

Un autre problème est que le langage naturel peut être extrêmement ambigu et varié au niveau de sa structure, de son sens et de son interprétation et il peut aussi régulièrement changer ou évoluer. Par exemple, une phrase peut être interprétée différemment selon le contexte dans lequel elle a été produite, et selon les mots employés. Similairement

[1] LEXICO, [sans date]. natural language processing | Definition of natural language processing by Lexico. In : *Lexico Dictionaries* | *English* [en ligne]. [Consulté le 24 août 2019]. Disponible à l'adresse : https://www.lexico.com/en/definition/natural_language_processing.

aux différents styles d'écriture d'une personne à une autre, il est facilement possible de tirer le même sens de deux phrases même si les mots employés ne sont pas les mêmes.

2.1.3 Domaines de recherche et applications

Les tâches les plus courantes liées au TALN peuvent être regroupées de manière non-exclusive dans les domaines suivants[2] :

- **L'analyse de la syntaxe**, qui regroupe tous ce qui relève des aspects théoriques et structurels du langage. Ceci comprend entre autres la lemmatisation (l'extraction de la forme neutre, dite canonique d'un groupe de mots semblables), l'étiquetage morpho-syntaxique (la catégorisation et codification de mots en fonction de leur catégorie grammaticale qui peut aussi être spécifique au contexte d'utilisation) et l'analyse syntaxique (l'extraction de liens existant entre les mots d'une phrase) ;

- **L'analyse de la sémantique**, qui correspond à l'étude du sens que les mots et les phrases peuvent avoir dans le langage (CNRTL[6]). Des exemples d'applications sémantiques de TALN sont la traduction automatique (l'exemple le plus performant est Google Translate avec son système qui utilise la traduction basée sur la statistique), le résumé automatique de texte (le principe d'extraire l'essentiel de l'information d'un document par le biais de de techniques d'abstraction, d'extraction ou de compression de phrase) et la correction orthographique (l'analyse en temps réel de mots ou groupes de mots dans le but de proposer des améliorations syntaxiques sur base de sémantique) ;

- **Le traitement de la parole**, qui rassemble toutes les tâches qui utilisent la parole pour reconnaître des aspects linguistiques. Ceci inclut, entre autres, la reconnaîssance de l'écriture manuscrite (l'utilisation de réseaux de neurones pour déterminer et traduire en charactères UTF-8 des lettres fournies en entrée pour ensuite les rassembler en mots et phrases), la reconnaîssance vocale (la segmentation d'un signal audio en éléments qui pourront être comparés à des phonèmes. Les phonèmes seront comparés entre eux. Ils peuvent être confrontés à des librairies de mots et de phrases. Ce procédé peut mener a la détection de langues et de dialectes ;

[2] WIKIPEDIA., 2019. Traitement automatique du langage naturel [en ligne]. S.l. : s.n. [Consulté le 24 août 2019]. Disponible à l'adresse : https://fr.wikipedia.org/w/index.php?title=Traitement_automatique_du_langage_naturel&oldid=159047154.

- **L'extraction d'informations**, soit l'ensemble des tâches qui consistent à extraire des informations structurées à partir d'un corpus de documents déstructurés. Due à sa difficulté, cette discipline est restreinte à quelques domaines spécifiques tels que les fusions d'entreprises et la forensique numérique. Parmi les tâches liées à l'extraction d'informations, nous comptons notamment l'alimentation de bases de connaissances (le fait d'extraire des faits spécifiques d'un corpus documentaire), la reconnaîssance d'entités nommées (l'extraction d'informations d'objets textuels catégorisables dans des classes telles que des noms de personnes, d'organisations, d'entreprises, de lieux, etc. d'un lot de documents) et l'analyse de sentiments (le fait d'extraire des informations permettant de déterminer, en se basant sur les mots d'un texte, l'état émotionnel du contenu de manière générale).

2.2 L'analyse de sentiments[3, 4, 5]

2.2.1 Définition

Apparu au début des années 2000, l'analyse de sentiments, aussi appelé « opinion mining » en anglais, est une technique d'analyse de données liée au TALN et qui consiste à analyser des corps de textes dans le but d'en extraire des opinions ou sentiments et un taux de confiance lié à ceux-ci.

L'analyse de sentiments est capable de déterminer la polarité d'une séquence de mots, c'est-à-dire identifier une tendance positive ou négative, ainsi que le sarcasme ou l'ironie de cette séquence.

Cette technique est surtout utilisée pour analyser le contenu en ligne comme les avis clients de sites de commerce, les discussions dans les forums, les textes des blogs, et surtout les échanges sur les réseaux sociaux. Leur analyse permet de transformer des informations peu structurées en données concrètes et interprétables sur les opinons des internautes par rapport à des produits, marques, faits divers, expériences ou tout autre sujet qui permet d'exprimer une opinion. L'analyse de ces types de contenus peut en effet aider à déterminer des tendances sociales ou commerciales, informer les

[3] WIKIPEDIA, 2019. Opinion mining [en ligne]. S.l. : s.n. [Consulté le 11 septembre 2019]. Disponible à l'adresse : https://fr.wikipedia.org/w/index.php?title=Opinion_mining&oldid=159547346.

[4] LI, Nan et WU, Desheng Dash, 2010. Using text mining and sentiment analysis for online forums hotspot detection and forecast. In : *Decision Support Systems*. janvier 2010. Vol. 48, n° 2, p. 354-368. DOI 10.1016/j.dss.2009.09.003.

[5] UNIGE, 2019. Analyse de sentiments en text mining [en ligne]. S.l. : s.n. [Consulté le 11 septembre 2019]. Disponible à l'adresse : http://edutechwiki.unige.ch/fr/Analyse_de_sentiments_en_text_mining

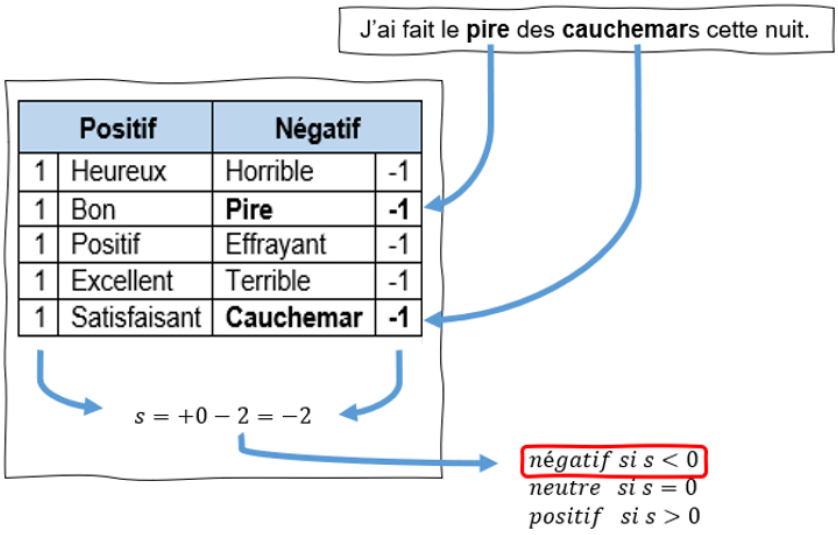
entreprises pour faciliter leurs décisions stratégiques, optimiser leur rendement et augmenter leur revenu.

2.2.2 Méthodes existantes

2.2.2.1 L'analyse des sentiments basée sur des règles

L'exploitation d'un système basé sur des règles précises, claires et surtout manuellement définies permet d'identifier la polarité et la subjectivité d'une séquence de mots. C'est par le biais d'un lexique – un dictionnaire de mots positifs et négatifs désignant des opinions et des expressions – que l'analyse d'un fragment de texte permet de produire un sentiment. Un exemple simple de ce type de système consiste à dresser deux listes de mots et expressions, l'une négative, associée à une valeur négative et l'autre positive, associée à une valeur positive. Ensuite, chaque mot du fragment à analyser est décorrélé du reste du texte, ce qui en apprentissage machine est la tokenization, pour ensuite être comparé aux listes afin d'être classifié comme positif ou négatif. Finalement, il suffit de compter le nombre d'occurrences des mots dans ces listes et d'en faire la somme pour obtenir un score s qui si $s < 0$, est négatif, si $s > 0$, est positif et si $s = 0$, est neutre.

Figure 1 : Analyse des sentiments basée sur des règles



(Thery Ehrlich, 2019)

L'intérêt de cette approche est qu'elle nécessite aucun entraînement et est relativement facile à debugger mais elle peut s'avérer être imprécise.

2.2.2.2 L'analyse de sentiments basée sur l'apprentissage machine[6]

Il est aussi possible d'utiliser un système qui ne dépend pas de règles mais qui va plutôt utiliser un modèle de classification dans lequel un algorithme reçoit des données sous la forme de fragments de texte. Ce modèle supervisé sera entraîné pour déterminer quel sentiment et/ou opinion est avancé suivant le label d'entraînement fourni avec l'échantillon de texte.

L'avantage d'utiliser ce type de système est qu'il est facilement évolutif et très précis mais il nécessite en revanche de grandes quantités de données d'entraînement pour fonctionner correctement. Les modèles les plus appréciés en matière d'analyse de sentiments sont :

- Les algorithmes Naïve Bayes ;
- Les machine à vecteurs de support ;
- Les réseaux de neurones.

Chaque approche ci-dessus présente des avantages et des inconvénients. Les algorithmes Naïve Bayes et les machines à vecteurs de support sont capables d'obtenir de bons résultats avec peu de données d'entraînement. Les machines à vecteurs de support sont cependant plus performantes mais elles ont de plus grands besoins en ressources pour atteindre ce niveau de performance. Malgré les avantages de ces deux modèles, leur performance atteint une limite qui ne peut pas être dépassée même si le jeu de données d'entraînement est plus conséquent. Les réseaux de neurones cependant, malgré le fait qu'ils doivent disposer de jeux de données d'entraînement plus conséquents, ne sont pas limités de la même manière que les modèles précédents. En effet, plus le jeu de données d'entraînement est grand, plus la performance du modèle croitra. C'est pour cette raison que nous allons privilégier cette architecture. Les réseaux de neurones sont effectivement très extensibles et ceci est un facteur extrêmement important aujourd'hui sachant que l'influx de données au sein des sociétés augmente considérablement d'année en année.

[6] MONKEYLEARN., 13:15. Text Classification: A Comprehensive Guide to Classifying Text with Machine Learning. In : *MonkeyLearn* [en ligne]. 13:15. [Consulté le 3 octobre 2019]. Disponible à l'adresse : https://monkeylearn.com/text-classification.

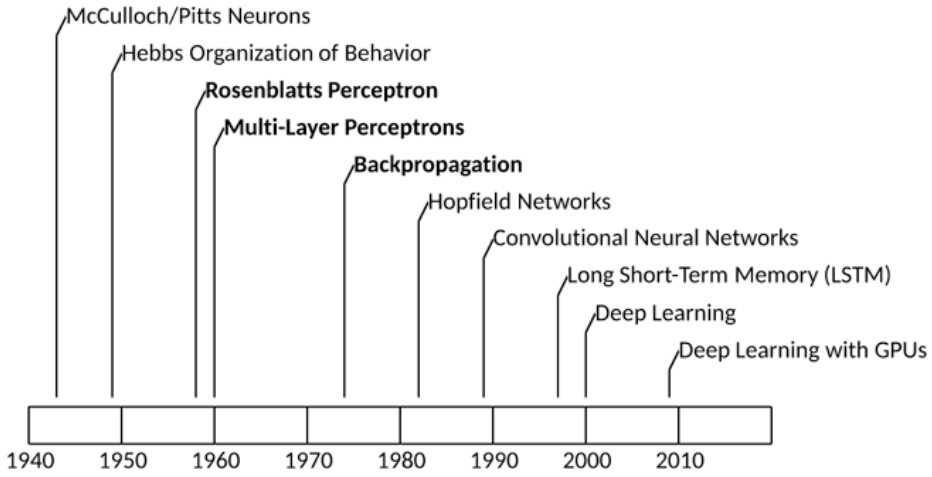
3. Les réseaux de neurones

3.1 Historique

- **1943** : le neurophysiologiste Warren McCulloch et le logicien Walter Pitts créent le premier modèle mathématique visant à représenter fidèlement un neurone biologique. Ils créeront même un modèle physique à base de circuits électriques pour appuyer leurs théories ;

- **1949** : le psychologue Donald Hebb écrit dans son livre *Organisation of Behavior: A Neuropsychological Theory* avoir observé d'un point de vue biologique qu'une voie neurale peut davantage se renforcer si elle est régulièrement activée ;

Figure 2 : Chronologie des réseaux de neurones



(IBM Developer, 2017 – https://developer.ibm.com/articles/cc-cognitive-neural-networks-deep-dive/)

- **1958** : le psychologue Frank Rosenblatt crée le perceptron, un modèle neuronal simple, utilisé pour classer des données en deux ensembles distincts. Le modèle a cependant un défaut de taille, dans le sens où il est impossible de correctement classifier le « ou » exclusif (XOR). La solution à cela sera d'employer deux couches de perceptrons mais c'est en s'appuyant notamment sur ce défaut que les adeptes des méthodes traditionnelles feront pression pour faire fortement réduire les subventions accordées aux projets de recherche sur les réseaux de neurones, et ce durant plus d'une décennie ;
- **1975** : le Dr. Paul Werbos crée l'algorithme de rétropropagation du gradient qui permettra d'entraîner avec succès des perceptrons multicouches, offrant ainsi des nouvelles possibilités d'applications pour les réseaux de neurones multicouches et la reprise des projets de recherches en la matière.

C'est d'ailleurs à la suite de l'introduction de la rétropropagation du gradient que les réseaux de neurones sont devenus davantage une référence en matière

d'apprentissage automatique. Plus tard, ce sera via l'utilisation de processeurs graphiques aussi appelés GPGPUs (General Practice Graphic Processing Unit) capables d'effectuer des traitements matriciels très rapides que l'entraînement de larges réseaux aura été rendu possible, déléguant ainsi l'entraînement et l'exécution de ceux-ci à des groupes d'accélérateurs d'IA. Ces nouvelles technologies auront à leur tour donné naissance à de nouvelles architectures d'apprentissage profond comme les réseaux de neurones convolutifs et les réseaux de neurones récurrents qui auront grandement élargi le champ d'applications possibles que proposent les réseaux de neurones et qui seront abordés ultérieurement dans le présent document.

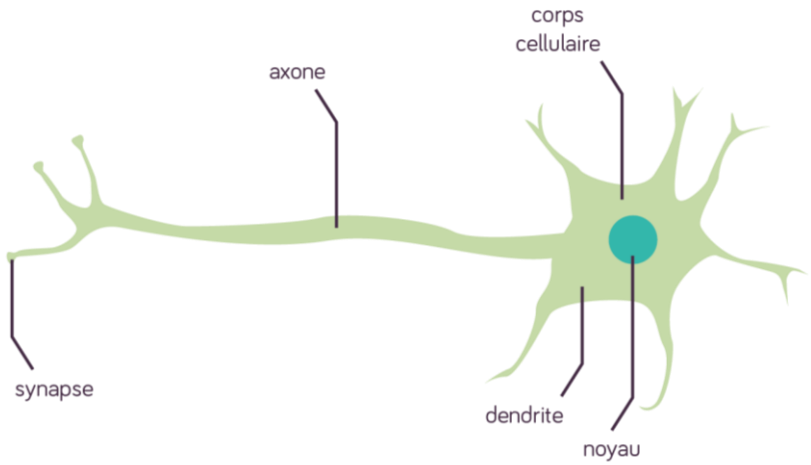
3.2 Le neurone artificiel

Imaginé par Frank Rosenblatt, le perceptron est une représentation mathématique de neurone biologique (Rosenblatt, 1958) inspiré par le neurone formel de Warren McCulloch et Walter Pitts ainsi que par la théorie sur le renforcement des voies neurales de Donald Hebb. Avant d'approfondir le sujet du perceptron, il est judicieux de parler du neurone biologique et du neurone formel. De plus, cette partie du document cherchera avant tout à modéliser le fonctionnement d'un unique perceptron, de l'entrée d'information à la sortie de la réponse en passant par son traitement.

3.2.1 Le neurone biologique

Aussi appelé cellule nerveuse, le neurone est une cellule appartenant au système nerveux et dont le plus grand nombre est présent dans le cortex cérébral. Son rôle est de recevoir, analyser puis transmettre des informations sous la forme de signaux bioélectriques appelés influx nerveux et ce sans distinction du rôle du message envoyé qui peut être perceptif, moteur, émotif ou cognitif.

Figure 3 : Neurone biologique



(Schoolmouv.fr – https://www.schoolmouv.fr/definitions/neurones/definition)

3.2.1.1 Composition

- **Le soma** : il s'agit du corps principal du neurone. Il est le centre de traitement d'information de la cellule ;
- **Les dendrites** : rattaché au soma, les dendrites sont des branches qui servent de points d'entrée aux influx nerveux provenant d'autres neurones ;
- **L'axone** : aussi relié au soma, il est l'exclusif chemin de sortie de l'information traitée par le neurone ;
- **Les synapses** : se trouvant à l'extrémité de l'axone, ils ont pour rôle est de transmettre l'information traitée vers d'autres neurones.

3.2.1.2 Fonctionnement[7]

1) Les dendrites reçoivent des neurotransmetteurs qu'ils convertissent en influx nerveux qui sont acheminés jusqu'au soma ;
2) Le soma récupère les influx nerveux entrants, les module et les traite. C'est ici qu'entre en jeu la notion de seuil où, si l'intensité du stimulus apporté par les entrées provoque une dépolarisation chimique suffisamment forte, le déclenchement du potentiel d'action qui génère un influx nerveux qui est envoyé à l'axone se fait. En contrepartie, si la dépolarisation est trop faible pour passer le seuil, le neurone ne génère pas d'influx nerveux de sortie ;
3) L'axone transporte ensuite l'influx jusqu'aux synapses qui le transforment en messages chimiques appelés neurotransmetteurs et qui font le pont entre les synapses et les dendrites des neurones voisins ;
4) Une fois les neurotransmetteurs envoyés vers les prochains neurones, ils sont à nouveau transformés en influx nerveux afin d'être traités par les prochains neurones.

Selon les derniers calculs théoriques, le cerveau humain moyen peut contenir entre 86 et 100 milliards de neurones, chacun connectés à des milliers d'autres neurones. Aussi, la puissance de calcul de ce réseau de cellules nerveuse est estimée à environ un

[7] UNIVERSITE DE GENEVE, 2019. La transmission de l'influx nerveux [en ligne]. S.l. : s.n. [Consulté le 29 août 2019]. Disponible à l'adresse : https://edu.ge.ch/decandolle/sites/localhost.decandolle/files/sn5-influx_nerveux2e_version.pdf

zettaflop, soit mille milliards de milliards (10^{21}) d'opérations par seconde[8]. En comparaison, Summit, le super ordinateur le plus puissant du moment ne dispose que d'une puissance de calcul maximale de 200 petaflops, soit 0.0002 zettaflops[9]. De plus, il est communément accepté que les neurones sont arrangés de manière hiérarchique[10], où chacun à un rôle et une responsabilité spécifique. En effet, chaque neurone est théoriquement responsable d'une caractéristique particulière comme une maison peut par exemple être reconnue par sa porte, ses fenêtres et son toit.

Un autre aspect extrêmement important qui démontre davantage la complexité étonnante du cerveau est sa neuroplasticité. En effet, le cerveau est capable de reconfigurer ses circuits au fur et à mesure que ses neurones meurent, redistribuant leurs responsabilités à ceux qui sont encore fonctionnels.

C'est en se basant sur ce concept de neuroplasticité qu'un groupe de chercheurs et chercheuses du MIT, menés par la professeure Anna Roe, ont cherché à observer la manière dont réagirait une zone du cerveau si sa fonction de base devait être altérée par la réception de stimuli non-originaux[11]. C'est dans le cadre de ce projet de recherche que les liens entre le cortex auditif et les nerfs auditifs chez des furets ont été rompus et remplacés par des nerfs optiques.

Lorsque des projections visuelles furent ensuite envoyées au cortex auditif, l'équipe de professeure Roe observa que le cortex auditif s'adaptait à la nouvelle source d'information et commençait à traiter les images comme s'il savait le faire depuis toujours.

Ces observations inspirèrent l'ingénieur Jeff Hawkins à explorer la capacité des colonnes du néocortex à traiter toute information sans distinction de leur origine[12]. Ceci suggérerait

---

8 INDIANA UNIVERSITY, [sans date]. Understand measures of supercomputer performance and storage system capacity. In : [en ligne]. [Consulté le 27 septembre 2019]. Disponible à l'adresse : https://kb.iu.edu/d/apeq.

9 OAK RIDGE LEADERSHIP COMPUTING FACILITY, [sans date]. Summit. In : Oak Ridge Leadership Computing Facility [en ligne]. [Consulté le 27 septembre 2019]. Disponible à l'adresse : https://www.olcf.ornl.gov/olcf-resources/compute-systems/summit/.

10 DATABASE, Institute of Medicine (US) Committee on a National Neural Circuitry, PECHURA, Constance M. et MARTIN, Joseph B., 1991. *Overview of Neuroscience Research: A Closer Look at the Neural Hierarchy* [en ligne]. S.l. : National Academies Press (US). [Consulté le 3 octobre 2019]. Disponible à l'adresse : https://www.ncbi.nlm.nih.gov/books/NBK234389/.

11 ROE, Aw, PALLAS, Sl, KWON, Yh et SUR, M, 1992. Visual projections routed to the auditory pathway in ferrets: receptive fields of visual neurons in primary auditory cortex. In : *The Journal of Neuroscience*. 1 septembre 1992. Vol. 12, n° 9, p. 3651-3664. DOI 10.1523/JNEUROSCI.12-09-03651.1992.

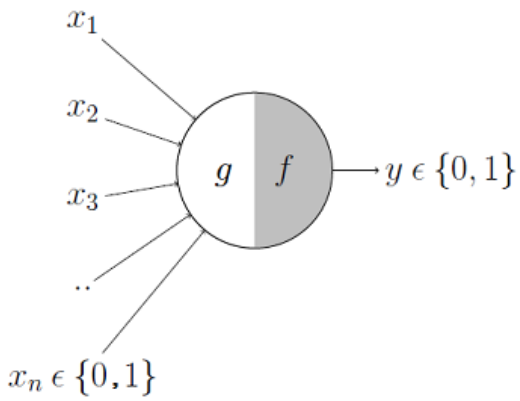
12 HAWKINS, Jeff, AHMAD, Subutai et CUI, Yuwei, 2017. A Theory of How Columns in the Neocortex Enable Learning the Structure of the World. In : *Frontiers in Neural Circuits* [en ligne]. 2017. Vol. 11. [Consulté le 28 août 2019]. DOI 10.3389/fncir.2017.00081. Disponible à l'adresse :

donc que tous les neurones du cerveau utilisent, d'une certaine manière, le même algorithme pour traiter les informations qu'elles reçoivent et qu'il suffirait de trouver cet algorithme pour pouvoir réellement créer des réseaux de neurones adaptables à tout type de circonstances.

3.2.2 Le neurone formel[13]

Imaginé par Warren S. McCulloch et Walter H. Pitts en 1943[14], le neurone formel est la toute première tentative de représentation mathématique d'un neurone biologique. Il s'agit d'une fonction algébrique dont la valeur dépend des paramètres appelés coefficients ou poids[15].

Figure 4 : Neurone formel (MCP neuron)



(Akshay Chandra Lagandula, 2018 – https://tinyurl.com/y4qa4kbq)

3.2.2.1 Composition

Le neurone formel peut être divisé en deux parties :

- g : la partie qui joue le rôle des dendrites en recevant des entrées x_1 à x_n qui ont chacune une valeur de soit 1, soit 0 et qui sont par la suite agrégées en une valeur unique ;

https://www.frontiersin.org/articles/10.3389/fncir.2017.00081/full#targetText=The%20input%20layer%20of%20each,neurons%20arranged%20in%20mini%2Dcolumns.&targetText=The%20feedforward%20input%20of%20cells,patterns%20(Jones%2C%202000).

[13] LAGANDULA, Akshay Chandra, 2018. McCulloch-Pitts Neuron — Mankind's First Mathematical Model Of A Biological Neuron. In : *Medium* [en ligne]. 7 novembre 2018. [Consulté le 29 août 2019]. Disponible à l'adresse : https://towardsdatascience.com/mcculloch-pitts-model-5fdf65ac5dd1.

[14] PITTS, MCCULLOCH, 1943. A Logical Calculus of the Ideas Immanent in Nervous Activity [en ligne]. S.l. : s.n. [Consulté le 29 août 2019]. Disponible à l'adresse : http://www.cse.chalmers.se/~coquand/AUTOMATA/mcp.pdf

[15] UNIVERSALIS, Encyclopædia, [sans date]. RÉSEAUX DE NEURONES FORMELS. In : Encyclopædia Universalis [en ligne]. [Consulté le 27 juillet 2019]. Disponible à l'adresse : http://www.universalis.fr/encyclopedie/reseaux-de-neurones-formels/

- f : la partie qui fait office de fonction de décision ou déclenchement. Si la somme des entrées ne dépasse pas le seuil prévu par f, le neurone ne sera pas activé et ne générera donc pas de sortie.

3.2.2.2 Fonctionnement

Tenant compte des informations précédemment citées, l'algorithme du neurone formel pour un vecteur d'entrées X est la suivante :

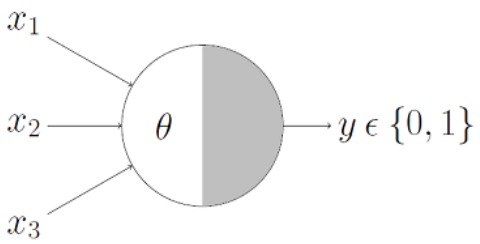
$$g(X) = \sum_{i=1}^{n} x_i$$

$$y = f\big(g(X)\big) = \begin{cases} 1 \; si \; g(X) \geq \theta \\ 0 \; si \; g(X) < \theta \end{cases}$$

où θ est le paramètre de seuil qui, si atteint, permet d'activer le neurone.

Sachant que le neurone formel reçoit et renvoie des valeurs booléennes, un cas d'utilisation typique serait de tester des problèmes booléens avec le neurone. Par ailleurs, une représentation plus concise du neurone serait la suivante, sachant que le neurone ne s'active que si la somme des entrées est plus grande ou égale à θ :

Figure 5 : Représentation concise du neurone formel

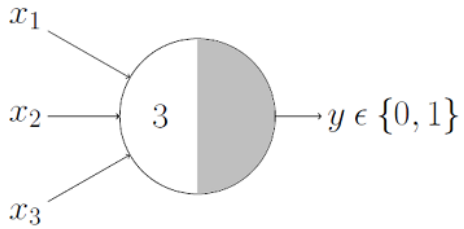


(Akshay Chandra Lagandula, 2018 – https://tinyurl.com/y4qa4kbq)

Voici quelques exemples de problèmes booléens qui peuvent être appliqués au neurone formel :

Fonction ET (AND) :

Figure 6 : Représentation de la fonction ET

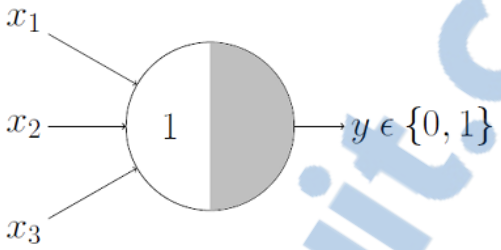


(Akshay Chandra Lagandula, 2018 – https://tinyurl.com/y4qa4kbq)

Dans ce cas de figure, il faut que toutes les entrées soient égales à 1 pour que le seuil soit atteint et que le neurone puisse s'activer.

Fonction OU (OR) :

Figure 7 : Représentation de la fonction OU



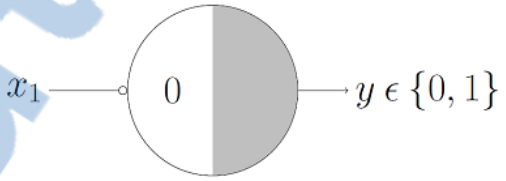
(Akshay Chandra Lagandula, 2018 – https://tinyurl.com/y4qa4kbq)

Dans le cas du OU, il faudrait que le vecteur X contenant les entrées x_1 à x_n vaille :

$$X = \begin{bmatrix}1\\0\\0\end{bmatrix} \text{ ou } X = \begin{bmatrix}0\\1\\0\end{bmatrix} \text{ ou } X = \begin{bmatrix}0\\0\\1\end{bmatrix} \text{ ou } X = \begin{bmatrix}1\\1\\0\end{bmatrix} \text{ ou } X = \begin{bmatrix}1\\0\\1\end{bmatrix} \text{ ou } X = \begin{bmatrix}0\\1\\1\end{bmatrix} \text{ ou } X = \begin{bmatrix}1\\1\\1\end{bmatrix}$$

Fonction PAS (NOT) :

Figure 8 : Représentation de la fonction PAS



(Akshay Chandra Lagandula, 2018 – https://tinyurl.com/y4qa4kbq)

Ici x_1 est une entrée inhibitoire, ce qui implique que si la valeur en entrée vaut 1, alors celle-ci sera interprétée comme étant égale à -1. Ceci sous-entendrait donc que si x_1 ne vaut pas 0, alors le seuil ne sera pas dépassé et le déclenchement échouerait.

Nous voyons donc que si nous comparons le neurone formel au neurone biologique, nous pouvons observer certaines équivalences.

Table 1 : Equivalences entre un neurone biologique et formel

Neurone biologique	Neurone formel
Dendrites / entrées	$g(X)$
Soma	$f(\ldots)$ modulé par θ
Axone / sortie	y

(Thery Ehrlich, 2019)

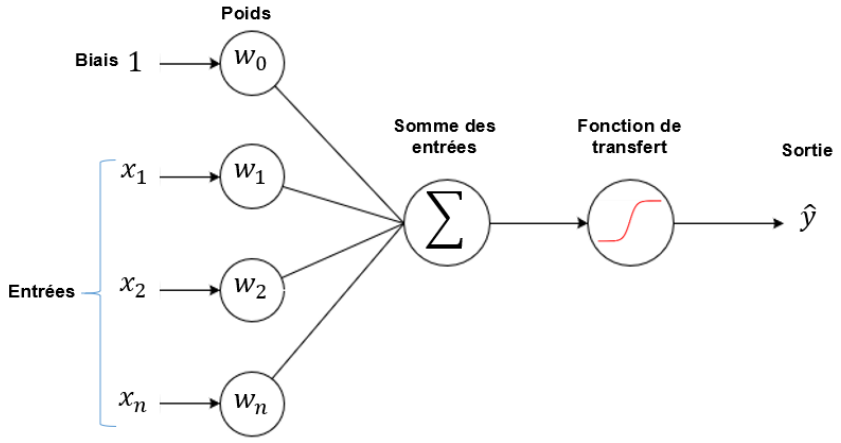
Néanmoins, ce modèle présente des limitations. Il n'accepte que des valeurs booléennes en entrée, le seuil doit être défini manuellement et il n'y a aucune mesure d'importance entre les entrées.

C'est à cause de cela que ce modèle n'est pas employé aujourd'hui et que nous avons plutôt opté à utiliser un modèle plus généraliste : le perceptron.

3.2.3 Le perceptron[16,17,18]

C'est dans le 6ème numéro du volume 65 du Psychogical Review de 1958 que le psychologue Frank Rosenblatt décrit le perceptron, un modèle probabiliste de stockage et d'organisation d'information dans le cerveau. Son modèle reprend les caractéristiques établies avec le neurone formel et y ajoute les observations liées au renforcement des voies neurales de Donald Hebb, ainsi que d'autres améliorations destinées à davantage généraliser le neurone artificiel.

Figure 9 : Perceptron



(Thery Ehrlich, 2019)

3.2.3.1 Améliorations apportées par ce modèle

- L'utilisation possible de nombres réels en entrée ;
- L'introduction d'un seuil arbitraire (capable d'apprendre) comme entrée ;

---

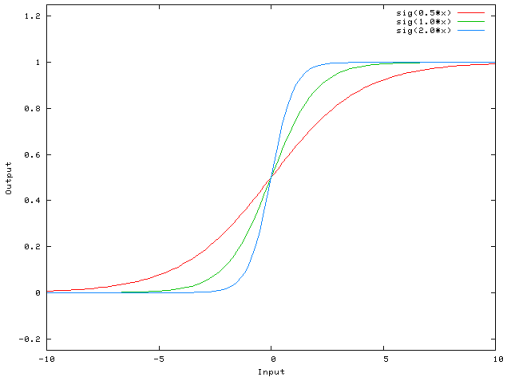
16 SHARMA, SAGAR, 2019. What the Hell is Perceptron? In : *Medium* [en ligne]. 5 mars 2019. [Consulté le 25 août 2019]. Disponible à l'adresse : https://towardsdatascience.com/what-the-hell-is-perceptron-626217814f53.

17 ROSENBLATT, F., 1958. The perceptron: A probabilistic model for information storage and organization in the brain. In : *Psychological Review*. 1958. Vol. 65, n° 6, p. 386-408. DOI 10.1037/h0042519.

18 SIMPLILEARN., [sans date]. What is Perceptron | Simplilearn. In : *Simplilearn.com* [en ligne]. [Consulté le 1 septembre 2019]. Disponible à l'adresse : https://www.simplilearn.com/what-is-perceptron-tutorial.

- La prise en compte du degré d'importance de chaque entrée grâce à un poids aussi appelé coefficient ou poids synaptique (w_0 … w_n) dont l'ajustement aura pour effet de modifier la pente de la tangente de la fonction de transfert ;

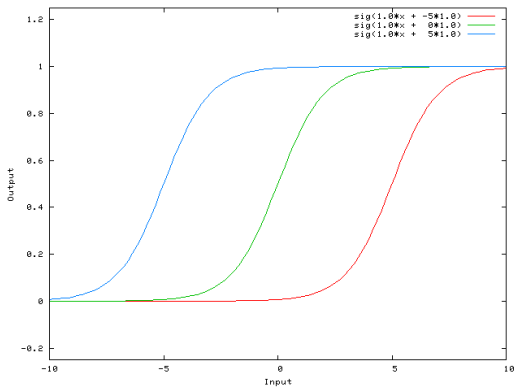
Figure 10 : Impact d'une modification du poids des entrées



(Nate Kohl, 2010 – https://stackoverflow.com/questions/2480650/role-of-bias-in-neural-networks)

- L'introduction d'une constante (généralement de valeur 1) à la position x_0 appelé « biais » qui peut être modulé avec le poids w_0 pour adapter le modèle par rapport aux entrées. Ceci se fait en déplaçant la tangente de la courbe de la fonction de transfert vers le haut ou vers le bas. Ce déplacement a pour effet de déplacer toute la courbe vers la droite ou la gauche et permet de mieux séparer les classes à prédire (ex : voitures vs motos, fruits vs légumes, morts vs vivants). Par analogie, l'ajustement du poids du biais peut être assimilé au déplacement de l'ordonnée à l'origine d'une fonction linéaire.[19]

Figure 11 : Impact d'une modification du poids du biais



(Nate Kohl, 2010 – https://stackoverflow.com/questions/2480650/role-of-bias-in-neural-networks)

[19] KOHL NATE, 2010. artificial intelligence - Role of Bias in Neural Networks. In : *Stack Overflow* [en ligne]. [Consulté le 1 septembre 2019]. Disponible à l'adresse : https://stackoverflow.com/questions/2480650/role-of-bias-in-neural-networks.

3.2.3.2 Fonctionnement et apprentissage[20,21]

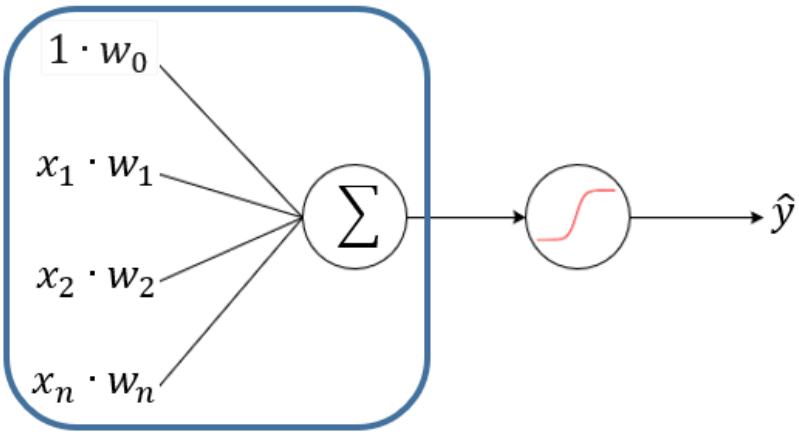
3.2.3.2.1 Initialisation :

Le perceptron est initialisé avec des poids w aléatoires qui sont ajustés au fur et à mesure que le modèle est entraîné. Les entrées reçoivent ensuite chacune une donnée x à traiter sous la forme d'une valeur réelle. Les données initiales proviennent d'un jeu de données d'entraînement préalablement mélangées qui auront été extraites d'un plus grand jeu dont environ un tiers servira de jeu de validation et de jeu de test.

3.2.3.2.2 Propagation avant :

L'étape suivante consiste à calculer le produit de chaque entrée avec son poids respectif en incluant le biais et d'ensuite en faire la somme.

Figure 12 : Calcul de la somme des produits des entrées et poids



(Thery Ehrlich, 2019)

Autrement dit, avec un vecteur X à n dimensions et un vecteur W à n dimensions :

$$g(W,X) = \sum_{i=0}^{n} w_i x_i = w_0 + w_1 x_1 + w_2 x_2 + \ldots + w_n x_n$$

Le résultat obtenu est ensuite passé à travers une fonction de transfert. Cette fonction peut être linéaire ou non-linéaire en fonction du résultat recherché et des données fournies. En revanche, sans le cadre de réseaux multicouches, il est important d'employer des fonctions non-linéaires, sinon les différentes couches seraient comme

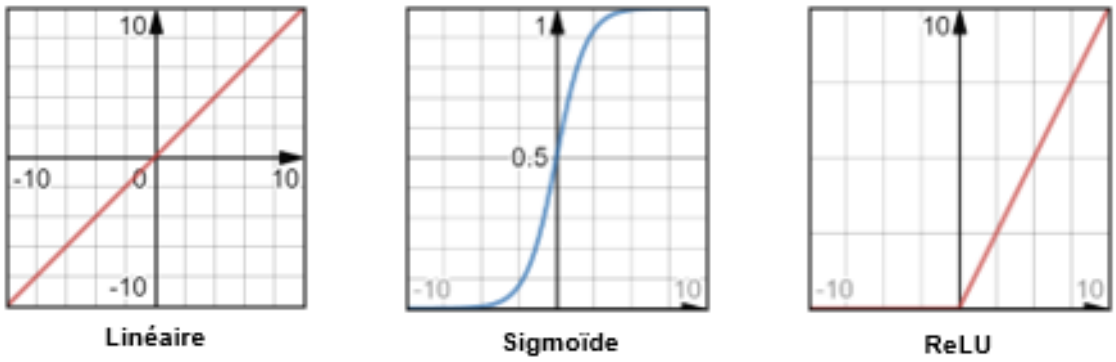
[20] COURSERA, [sans date]. Machine Learning - Home. In : *Coursera* [en ligne]. [Consulté le 7 septembre 2019]. Disponible à l'adresse : https://www.coursera.org/learn/machine-learning/home/welcome.

[21] ML CHEATSHEET, [sans date]. Logistic Regression — ML Cheatsheet documentation. In : [en ligne]. [Consulté le 7 septembre 2019]. Disponible à l'adresse : https://ml-cheatsheet.readthedocs.io/en/latest/logistic_regression.html#cost-function.

un réseau à couche unique, étant donné que la composition de la fonction linéaire reste invariablement linéaire.

.

Figure 13 : Exemples de fonctions de transfert



(Thery Ehrlich, 2019)

La fonction de transfert sert essentiellement à normaliser les résultats obtenus en bornant les résultats entre 0 et 1 ou -1 et 1 suivant la fonction choisie. Historiquement, la fonction logistique ou sigmoïde est la plus utilisée mais ce sont les fonction tanh et ReLU (Rectified Linear Unit) qui, aujourd'hui, sont les plus régulièrement employées.

$$Sigmoid(X,W) = \frac{1}{1+e^{-x_i w_i}}$$

Voici comment la sortie du perceptron est calculée par rapport au neurone formel si le seuil est fixé à θ, sachant qu'il reçoit en entrée les vecteur X et W à n dimensions :

$$g(W,X) = \sum_{i=0}^{n} w_i\, x_i$$

$$\hat{y} = f(g(X)) = \begin{cases} 1\ si\ g(W,X) \geq \theta \\ 0\ si\ g(W,X) < \theta \end{cases}$$

3.2.3.2.3 Application de la fonction de coût pour chaque prédiction :

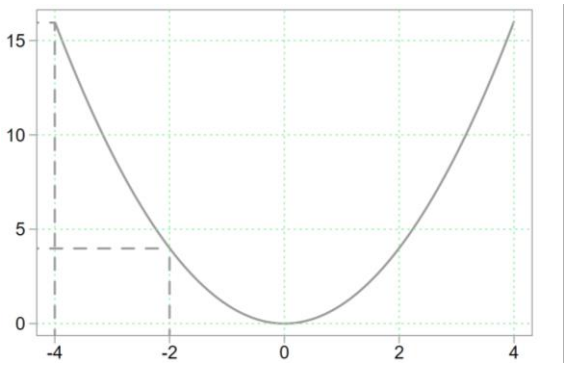
Ensuite vient l'étape qui permet de déterminer le degré de précision des prédictions. Chaque résultat $\hat{y}$ obtenu est comparé à sa valeur d'entraînement y, équivalente et préalablement fournie en guise de référence, afin de déterminer sa précision. C'est durant cette étape que l'apprentissage commence réellement, sachant qu'ici, la fonction de coût d'entropie croisée est appliquée afin de connaître le l'écart entre chaque prévision $\hat{y}$ et donnée d'entraînement y. Si la différence est trop grande, les poids des

entrées ainsi que le biais, sont ajustés dans le but de compenser pour l'erreur occasionnée.

Dans le contexte d'un cas de régression linéaire où les entrées et résultats attendus sont continus, l'utilisation d'une fonction de coût comme l'erreur quadratique moyenne serait le choix idéal, sa formule et représentation visuelle étant les suivantes :

$$J(W) = \frac{1}{2n}[\sum_{i=1}^{n}(\hat{y}_i - y_i)^2]$$

Figure 14 : Représentation graphique du coût d'erreur quadratique

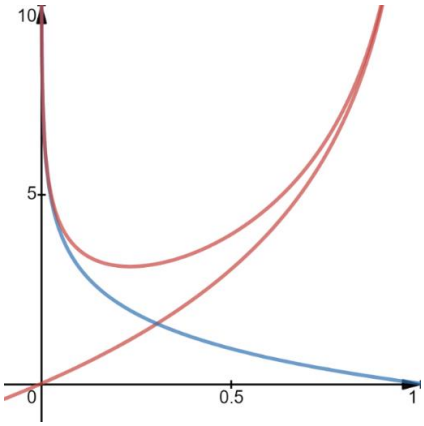


(Eran Raviv, 2017 – https://eranraviv.com/outliers-and-loss-functions/)

En revanche, dans les cas de classification comme la régression logistique et certains réseaux de neurones où les données fournies et prédites sont discrètes, il s'agira d'introduire des fonctions de coût adaptés à la classification. L'une des fonctions de coût les plus populaires pour ce type de cas est l'entropie croisée dont la formule et représentation sont les suivantes :

$$\mathrm{J(W)} = -\frac{1}{n}\sum_{i=1}^{n} y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)$$

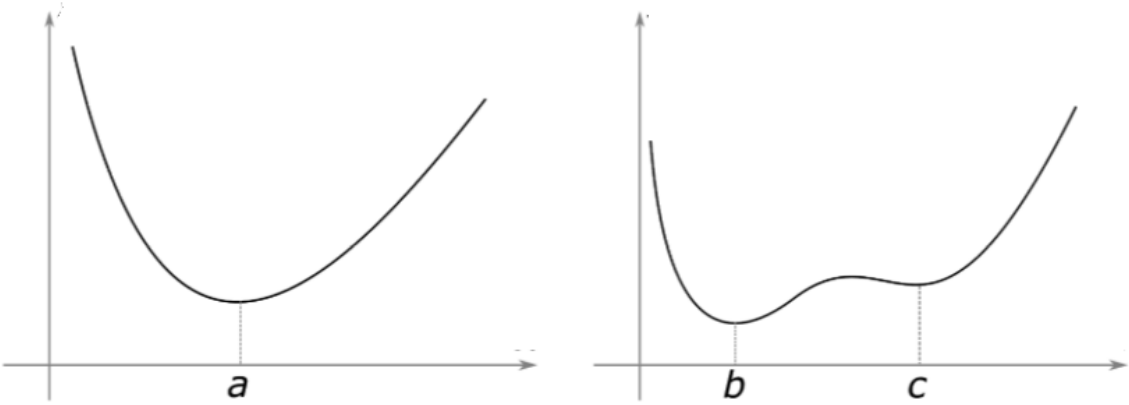
Figure 15 : Représentation graphique de l'entropie croisée



(Thery Ehrlich, 2019)

Lorsque la fonction de coût comprend une fonction non linéaire, la courbe que devra « naviguer » l'algorithme de descente du gradient peut devenir une fonction non-convexe et donc introduire des minimas multiples.

Figure 16 : Fonction convexe (gauche) et non-convexe (droite)



(Asteroide Santana, 2018 – https://tinyurl.com/y44owqya)

Le fait d'avoir des minimas multiples introduit par conséquent un nouveau concept dans le sens où il peut exister un minima qui est meilleur que les autres dans le sens où il se rapproche le plus de 0 : le minima global. Ceci pose donc un problème qui suggère la possibilité de rester piégé dans un minima local, si le poids initial est proche d'un minima local et ceci a donc pour effet de potentiellement impacter la performance du modèle. Pour pallier ceci, des fonctions d'optimisation plus complexes comme Adam[22] ont été introduites pour prendre en compte certains facteurs comme l'élan (la moyenne des gradients précédents) et adapter le taux d'apprentissage dans le but de le faire surpasser les minimas locaux afin d'atteindre le minima global.

Néanmoins les opinions concernant l'utilité de pourchasser le minima global ne sont pas unanimes comme peut en attester le document publié en 2014 par la professeure Anna Choromanska et ses collègues[23]. En effet ils estiment que plus la taille d'un réseau de neurones grandit, plus l'importance d'atteindre le minima global diminue, et que le fait de le rechercher augmente le risque de surentraînement du modèle utilisé.

[22] KINGMA, Diederik P. et BA, Jimmy, 2014. Adam: A Method for Stochastic Optimization. In : *arXiv:1412.6980 [cs]* [en ligne]. 22 décembre 2014. [Consulté le 2 octobre 2019]. Disponible à l'adresse : http://arxiv.org/abs/1412.6980.

[23] CHOROMANSKA, Anna, HENAFF, Mikael, MATHIEU, Michael, AROUS, Gérard Ben et LECUN, Yann, 2014. The Loss Surfaces of Multilayer Networks. In : *arXiv:1412.0233 [cs]* [en ligne]. 30 novembre 2014. [Consulté le 29 septembre 2019]. Disponible à l'adresse : http://arxiv.org/abs/1412.0233.

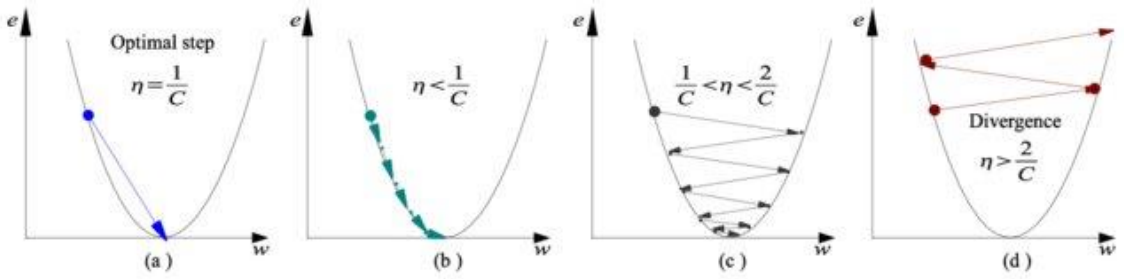
3.2.3.2.4 Minimisation du coût avec l'algorithme de descente de gradient :

Une fois la fonction de coût déterminée, l'étape suivante est de minimiser le coût afin d'augmenter la performance du modèle. Pour ce faire, il est nécessaire de calculer la descente du gradient dont la formule est la suivante :

$$w_i^{t+1} = w_i^t - \alpha \frac{\partial}{\partial w_i^t} J(W)$$

La valeur α, appelé le taux d'apprentissage, permet ici de définir la taille d'un pas à prendre à chaque itération pour éventuellement atteindre le minima global. Comme on peut l'observer dans la figure 17, il est possible d'atteindre le minima d'un coup, pourvu qu'α soit exactement fixé à la bonne valeur. Un α trop faible, en revanche, provoquera un temps de calcul conséquent (b) tandis qu'une valeur trop élevée pourrait augmenter le temps nécessaire pour converger vers (c) ou même complètement manquer le minima et diverger de celui-ci (d), augmentant le coût au lieu de le faire diminuer.

Figure 17 : Les effets de α sur la descente de gradient



(Tom Ducket, 2008 – https://tinyurl.com/y4kmu83e)

La dérivée partielle de $J(W)$ permet d'obtenir le gradient, ou la pente de la tangente de la fonction d'entropie croisée, ce qui permet de savoir quand le minima est atteint. Sachant que le minima a une pente de 0, la multiplication de α avec la dérivée permet de stopper la progression de la recherche du minima et d'obtenir donc les poids minimisés par l'algorithme de descente du gradient. Par conséquent, tant que la dérivée ne vaut pas 0, l'algorithme continue à itérer.

Voici la forme dérivée de $J(W)$:

$$\frac{\partial}{\partial w_j} J(W) = \frac{1}{n} \sum_{i=1}^{n} (\hat{y}_i - y_i) x_i$$

Et la forme développée de l'algorithme du gradient incluant la dérivée de l'entropie croisée :

$$w_j^{t+1} = w_j^t - \alpha \frac{1}{n} \sum_{i=1}^{n} (\hat{y}_i - y_i) x_i$$

3.2.3.2.5 Itération jusqu'à convergence avec mise à jour des poids et du biais

Tout au long de ce processus de minimisation du coût, les poids et le biais sont constamment ajustés. Ceci est répété autant de fois que nécessaire jusqu'à obtention d'un coût estimé comme étant suffisamment proche de zéro pour satisfaire les attentes fixées pour que le modèle soit estimé comme étant entraîné. Le coût minimal est par conséquent atteint. Et dès que le taux d'erreur et la taille du gradient stagnent un certain temps, il est admissible de supposer que l'algorithme de descente du gradient a atteint le minima. Il est aussi possible d'utiliser le principe de l'*early-stopping* pour d'interrompre le processus dès que le taux d'erreur du jeu de validation arrête de décroître.

Une fois ces étapes effectuées, le modèle est enfin mis à l'épreuve avec des données de test, jusqu'ici non-utilisées, et la performance de l'algorithme est désormais déterminée. Le perceptron est désormais prêt à être employé dans des cas réels.

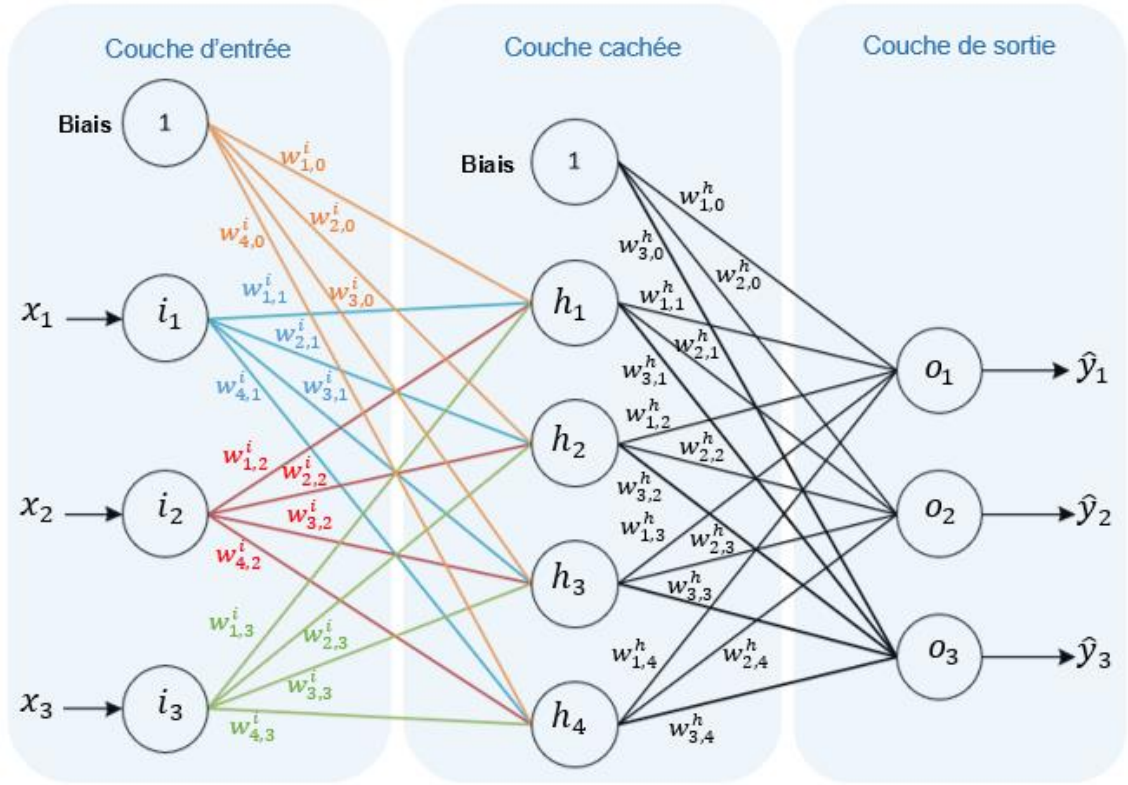
3.3 Le perceptron multicouche

3.3.1 Nouveautés[24]

Le fait d'ajouter des couches de neurones supplémentaires permet de pouvoir soumettre au modèle des problèmes de plus en plus complexes, qu'il s'agisse de classifier une image, convertir un signal audio en texte ou extraire des informations d'un texte dans le but d'en résumer son contenu. Le modèle de réseau le plus basique est constitué de trois couches de neurones : une pour les entrées, une pour la ou les sorties et une couche intermédiaire appelée la couche cachée.

[24] JONES, 2017. A neural networks deep dive. In : *IBM Developer* [en ligne]. 24 juillet 2017. [Consulté le 25 août 2019]. Disponible à l'adresse : https://developer.ibm.com/articles/cc-cognitive-neural-networks-deep-dive/.

Figure 18 : Réseau de neurones à trois couches



(Thery Ehrlich, 2019)

La figure 18 permet d'observer un certain nombre de changements par rapport au schéma du perceptron unique en figure 9 :

- Chaque nœud (i pour « input », h pour « hidden » et o pour « output ») est un neurone à part entière, comprenant une fonction de sommation pondérée ainsi qu'une fonction de transfert ;
- Chaque nœud transmet sa sortie vers tous les nœuds de la couche suivante dans le but d'exécuter de nouveaux traitements qui sont distincts des précédents ;
- Chaque couche, sauf la couche de sortie, a un biais fixé à 1, qui ne reçoit aucune valeur en entrée ;
- Chaque sortie est pondérée ($w_{j,k}^{couche}$).

Au niveau de son fonctionnement, le perceptron multicouche est presque identique au perceptron unique. Les étapes sont :

1) L'initialisation des entrées et poids ;
2) La propagation avant ;

3) L'application de la fonction de coût ;

4) La rétropropagation ;

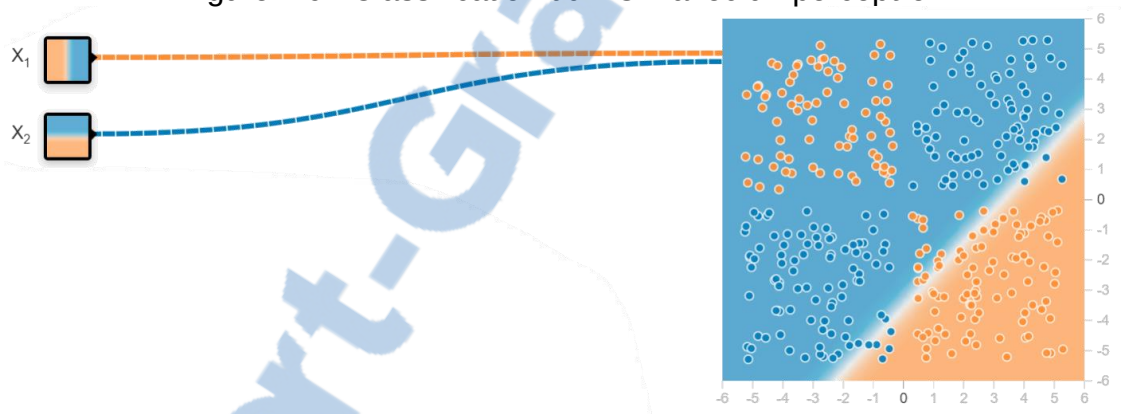
5) La mise à jour des poids ;

6) L'itération jusqu'à convergence.

La seule différence notoire d'un point de vue algorithmique est l'ajout de l'étape de rétropropagation qui complémente l'algorithme du gradient afin de pouvoir propager la mise à jour des poids vers tous les neurones du réseau.

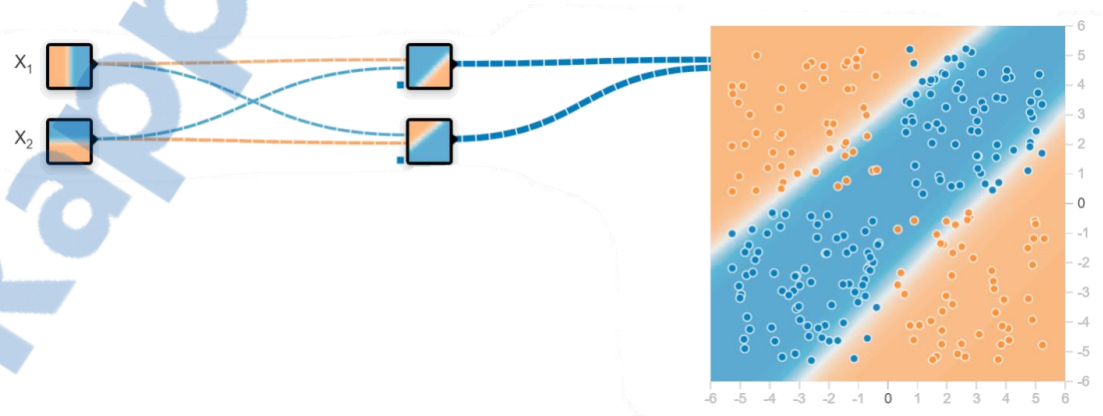
Néanmoins, d'un point de vue d'expressivité, l'utilisation d'un réseau de neurones permet de représenter des problèmes non-linéairement séparables, comme par exemple le « ou » exclusif (XOR), ce qui n'est pas le cas pour le perceptron.

Figure 19 : Classification du XOR avec un perceptron



(Thery Ehrlich, 2019 – https://playground.tensorflow.org)

Figure 20 : Classification du XOR via un réseau de neurones



(Thery Ehrlich, 2019 – https://playground.tensorflow.org)3

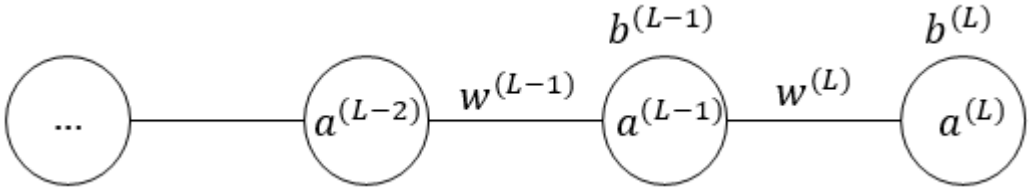
3.3.2 La rétropropagation[25,26,27]

Inventé en 1975 par le Dr. Paul Werbos, l'algorithme de rétropropagation du gradient est une des techniques d'entraînement les plus populaires. Son but principal est de calculer le gradient dans les cas complexes à couches multiples et de redistribuer les ajustements d'erreurs de prévision d'un modèle vers tous les poids et biais concernés, et ce, indépendamment de leur emplacement sur le réseau. L'algorithme permet d'améliorer la précision du modèle d'époque en époque, c'est-à-dire après avoir effectué une passe sur l'ensemble des données d'entraînement. Il s'agit donc d'une amélioration qui permet de calculer précisément le gradient pour toutes les couches en se référant à la chaîne d'événements qui influencent la fiabilité du modèle.

La rétropropagation repose sur le principe de la règle de dérivation en chaîne qui dit que la variation du taux d'un composant a un impact sur le taux de tous les composants qui le suivent.

Sachant que le but de l'algorithme est d'augmenter ou diminuer la valeur du biais et des poids d'un neurone afin de diminuer l'écart entre la prédiction et le résultat attendu, naturellement, si un autre neurone, qui lui aussi dispose d'un biais et de poids, est la cause de la sous-performance du modèle, il n'est que naturel de vouloir l'ajuster. Cet ajustement peut être facilement illustré avec le plus simple des réseaux de neurones :

Figure 21 : Réseau de neurones simplifié



(Thery Ehrlich, 2019)

[25] GEVA, [sans date]. Backpropagation in Neural Networks: Process, Example & Code. In : *MissingLink.ai* [en ligne]. [Consulté le 8 septembre 2019]. Disponible à l'adresse : https://missinglink.ai/guides/neural-network-concepts/backpropagation-neural-networks-process-examples-code-minus-math/.

[26] NIELSEN, Michael, 2015. Neural Networks and Deep Learning. In : [en ligne]. 2015. [Consulté le 9 septembre 2019]. Disponible à l'adresse : http://neuralnetworksanddeeplearning.com.

[27] WERBOS, Paul et JOHN, Paul, 1974. (13) (PDF) Beyond regression : new tools for prediction and analysis in the behavioral sciences /. In : *ResearchGate* [en ligne]. [Consulté le 9 septembre 2019]. Disponible à l'adresse : https://www.researchgate.net/publication/35657389_Beyond_regression_new_tools_for_prediction_and_analysis_in_the_behavioral_sciences.

Sachant que le coût C comprend l'ensemble des poids w et biais b du réseau ou chaque neurone est représenté par a et où L est la dernière couche du réseau, le coût s'écrit :

$$C(b_1, w_1, b_2, w_2, b_3, w_3) = (a - y)^2$$

Sachant cela et si nous nous limitons à un seul exemple C_0, nous savons que :

$$C_0(\dots) = \left(a^{(L)} - y\right)^2$$

où

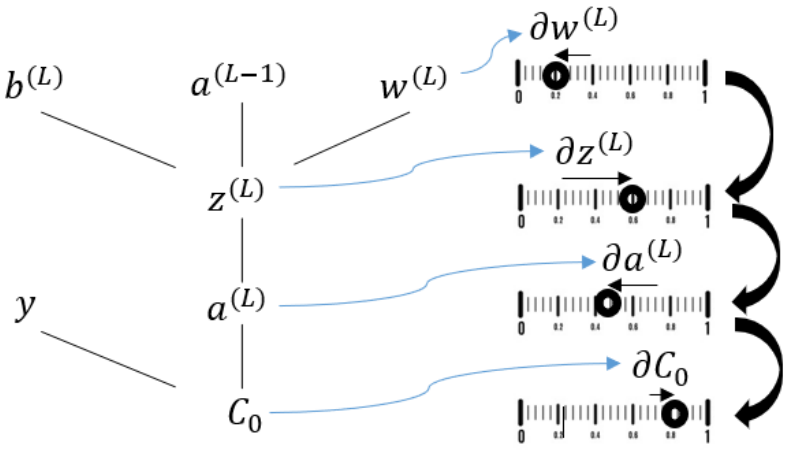
$$a^{(L)} = \sigma(z^{(L)})$$

et

$$z^{(L)} = b^{(L)} + w^{(L)} a^{(L-1)}$$

Il est aussi possible d'assimiler la représentation ci-dessus à la figure 22 qui permet de visualiser le comportement d'une modification du poids $w^{(L)}$sur $z^{(L)}$, $a^{(L)}$ et C_0. En effet, nous voyons que le biais, l'action précédente et le poids permettent de calculer $z^{(L)}$ qui correspond à la fonction de sommation des coefficients. $z^{(L)}$permet aussi, à son tour, de calculer $a^{(L)}$ qui correspond au traitement de la fonction de transfert. La différence entre $a^{(L)}$ et y donne la fonction de coût C_0.

Figure 22 : Conceptualisation de la règle de dérivation en chaîne



(Thery Ehrlich, 2019)

Cette illustration permet donc de voir qu'une variation de $\partial w^{(L)}$ provoque une modification de $\partial z^{(L)}$ qui modifie $\partial a^{(L)}$ qui altère finalement ∂C_0. Cette influence distribuée s'appelle « la règle de dérivation en chaîne » et elle se note :

$$\frac{\partial C_0}{\partial w^{(L)}} = \frac{\partial z^{(L)}}{\partial w^{(L)}} \cdot \frac{\partial a^{(L)}}{\partial z^{(L)}} \cdot \frac{\partial C_0}{\partial a^{(L)}}$$

où

$$\frac{\partial C_0}{\partial a^{(L)}} = 2\left(a^{(L)} - y\right), \qquad \frac{\partial a^{(L)}}{\partial z^{(L)}} = \sigma'\left(z^{(L)}\right), \qquad \frac{\partial z^{(L)}}{\partial w^{(L)}} = a^{(L-1)}$$

ce qui donne

$$\frac{\partial C_0}{\partial w^{(L)}} = \frac{\partial z^{(L)}}{\partial w^{(L)}} \cdot \frac{\partial a^{(L)}}{\partial z^{(L)}} \cdot \frac{\partial C_0}{\partial a^{(L)}} = 2(a^{(L)} - y)\, \sigma'(z^{(L)})\, a^{(L-1)}$$

Bien que le résultat ci-dessus ne répond pas à tous les cas possibles, la majorité du travail est ici accomplie. En effet, sa forme pour le biais $b^{(L)}$ ne nécessiterait que les changements suivants :

$$\frac{\partial C_0}{\partial b^{(L)}} = \frac{\partial z^{(L)}}{\partial b^{(L)}} \cdot \frac{\partial a^{(L)}}{\partial z^{(L)}} \cdot \frac{\partial C_0}{\partial a^{(L)}} = \cancel{2(a^{(L)} - y)}\sigma'(z^{(L)})\, a^{(L-1)}$$

Et pour les cas de $a^{(L-1)}$:

$$\frac{\partial C_0}{\partial a^{(L-1)}} = \frac{\partial z^{(L)}}{\partial a^{(L-1)}} \cdot \frac{\partial a^{(L)}}{\partial z^{(L)}} \cdot \frac{\partial C_0}{\partial a^{(L)}} = w^{(L)}\sigma'(z^{(L)})\, a^{(L-1)}$$

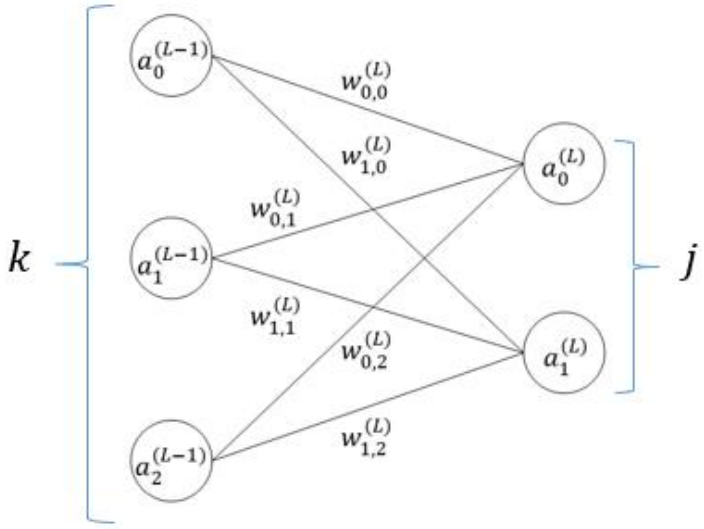
De ce fait, on peut constater qu'il suffit d'itérer la même séquence pour les couches précédentes :

$$\frac{\partial C_0}{\partial a^{(L-2)}} = \frac{\partial z^{(L-1)}}{\partial a^{(L-2)}} \cdot \frac{\partial a^{(L-1)}}{\partial z^{(L-1)}} \cdot \frac{\partial z^{(L)}}{\partial a^{(L-1)}} \cdot \frac{\partial a^{(L)}}{\partial z^{(L)}} \cdot \frac{\partial C_0}{\partial a^{(L)}}$$

$$\frac{\partial C_0}{\partial a^{(L-2)}} = w^{(L-1)}\sigma'(z^{(L-1)})\, a^{(L-2)} w^{(L)}\sigma'(z^{(L)})\, a^{(L-1)}$$

Néanmoins, si des nœuds supplémentaires s'ajoutent aux couches, il faut prendre en compte les nouvelles connexions entres eux. Il suffit d'ajouter un indice, en plus de l'exposant, qui détermine la couche dans laquelle se trouvent le nœuds, le poids ou le biais.

Figure 23 : Réseau à nœuds multiples pour deux couches



(Thery Ehrlich, 2019)

Par conséquent, si nous voulons calculer l'influence d'une variation de taux des $k^{\text{ème}}$ nœuds de la couche $L-1$ sur les nœuds d'indice j de la couche L dans la figure 21, la fonction de coût prendrait la forme suivante :

$$C_0 = \sum_{j=0}^{n_L-1} \left(a_j^{(L)} - y_i\right)^2$$

où

$$a_j^{(L)} = \sigma(z_j^{(L)})$$

et

$$z_j^{(L)} = b_j^{(L)} + w_{j,0}^{(L)} a_0^{(L-1)} + w_{j,1}^{(L)} a_1^{(L-1)} + w_{j,2}^{(L)} a_2^{(L-1)}$$

Ce qui nous donne, pour n'importe quel $k^{\text{ème}}$ nœud

$$\frac{\partial C_0}{\partial a_k^{(L-1)}} = \sum_{j=0}^{n_L-1} \frac{\partial z_j^{(L)}}{\partial a_k^{(L-1)}} \cdot \frac{\partial a_j^{(L)}}{\partial z_j^{(L)}} \cdot \frac{\partial C_0}{\partial a_j^{(L)}}$$

Nous voyons ici que les nœuds de la couche $L-1$ influencent les nœuds de la couche L par trois chemins différents et ceux-ci doivent être additionnés pour obtenir l'influence des nœuds de la couche $L-1$ sur le coût. Il suffit ensuite d'appliquer le même procédé pour tous les poids et biais.

4. Architectures utilisées avec le TALN

4.1 Les réseaux de neurones convolutifs[28, 29, 30, 31, 32 ,33]

D'abord imaginé par David H. Hubel et Torsten N. Wiesel entre 1950 et 1960, ce type de réseau de neurones était conçu pour reproduire le fonctionnement du cortex visuel où les neurones capturent chacun une section précise de l'ensemble du champ visuel auquel ils sont exposés afin de le cartographier. Ils ont aussi émis la théorie que la perception de l'environnement chez les animaux se faisait par le biais d'une architecture neuronale hiérarchique en couches et ils ont présumé que cette hiérarchie disposait de groupes de cellules dites simples, capables de détecter les bords d'objets et qui fournissaient cette information à des groupes de cellules dites complexes qui en faisaient la synthèse et allaient peu à peu augmenter le degré d'abstraction lié à l'objet perçu. Ceci permet de construire peu à peu la forme et l'identité de l'image.

En 1998, Yan LeCun s'est inspiré des théories de Hubel et Wiesel pour développer l'architecture LeNet-5[34], un réseau de neurones convolutifs à 7 couches capable de classifier des nombres écrits à la main de manière efficace. LeCun avait repris notamment les concepts de regroupement de cellules spécialisés, de hiérarchie de responsabilités et d'invariance spatiale qui définissaient la possibilité de reconnaître un objet indépendamment de sa position dans l'espace.

---

[28] BRITZ, Denny, 2015. Understanding Convolutional Neural Networks for NLP. In : *WildML* [en ligne]. 7 novembre 2015. [Consulté le 11 septembre 2019]. Disponible à l'adresse : http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/.

[29] IBM, 2018. Convolutional neural networks. In : *IBM Developer* [en ligne]. 2 mai 2018. [Consulté le 28 août 2019]. Disponible à l'adresse : https://developer.ibm.com/articles/cc-convolutional-neural-network-vision-recognition/.

[30] BINEY, Kingsley, 2019. Sentiment Analysis using 1D Convolutional Neural Networks in Keras. In : *Medium* [en ligne]. 1 mai 2019. [Consulté le 11 septembre 2019]. Disponible à l'adresse : https://medium.com/@romannempyre/sentiment-analysis-using-1d-convolutional-neural-networks-part-1-f8b6316489a2.

[31] WIKIPEDIA., 2019. *Convolutional neural network* [en ligne]. S.l. : s.n. [Consulté le 12 septembre 2019]. Disponible à l'adresse : https://en.wikipedia.org/w/index.php?title=Convolutional_neural_network&oldid=914010072.

[32] MISHRA, Mayank, 2019. Convolutional Neural Networks, Explained. In : [en ligne]. [Consulté le 13 septembre 2019]. Disponible à l'adresse : https://www.datascience.com/blog/convolutional-neural-network.

[33] DESHPANDE, Adit, [sans date]. A Beginner's Guide To Understanding Convolutional Neural Networks. In : [en ligne]. [Consulté le 13 septembre 2019]. Disponible à l'adresse : https://adeshpande3.github.io/adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/.

[34] LECUN, Yann et al., 1998. Gradient-Based Learning Applied to Document Recognition. In : [en ligne]. [Consulté le 29 septembre 2019]. Disponible à l'adresse : http://yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf

Depuis LeNet-5 un nombre important d'architectures ont été développées, dont ResNet, DenseNet et Inception.[35]

Aujourd'hui, ces réseaux de neurones sont l'outil de référence pour tout ce qui a trait au traitement d'image. Ceci comprend l'analyse, la classification et la génération d'information émanant de données du domaine visuel. Les champs d'applications les plus connus sont le Deepfake qui est le nom donné aux réseaux capables de remplacer le visage d'une personne avec celle d'un autre tout en préservant les expressions du sujet receveur, la reconnaîssance d'objets pour les véhicules autonomes ou l'apprentissage par renforcement qui permet entre autres d'entraîner une machine à jouer à un jeu vidéo ou contrôler un robot.

4.1.1 Fonctionnement

Les réseaux de neurones convolutifs sont composés de deux parties :

- Une partie dédiée à l'extraction des caractéristiques de l'image fournie en entrée. C'est durant cette phase que la construction de la forme la plus abstraite possible d'une image est effectuée. Elle est composée de trois opérations qui sont répétées un certain nombre de fois (généralement trois fois au minimum) :
    - L'opération de convolution ;
    - L'opération d'activation ;
    - L'opération de *max pooling*.

  Cette partie permet, au fur et à mesure des itérations, de détecter des caractéristiques de plus en plus complexes dans une image.

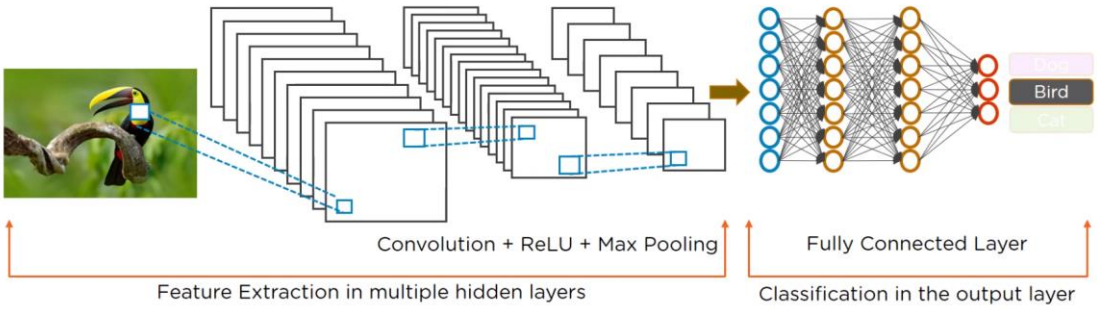
Figure 24 : Evolution de la détection de caractéristiques

(SINGH, Aditya, 2017 – https://tinyurl.com/h3a2bft)

[35] SLAZEBNI, Lana, [sans date]. Convolutional Neural Network Architectures: from LeNet to ResNet. In : [en ligne]. [Consulté le 29 septembre 2019]. Disponible à l'adresse : http://slazebni.cs.illinois.edu/spring17/lec01_cnn_architectures.pdf

- Une partie responsable de la classification qui reçoit et regroupe les résultats de toutes les passes de convolution faites par la première partie. Cette partie n'est rien d'autre qu'un perceptron multicouche chargé de classifier les données prétraitées qui lui sont fournies en entrée.

Figure 25 : Exemple de réseau convolutif



(Anonyme, sans date – https://www.quora.com/What-is-the-difference-between-CNN-and-RNN)

Étant donné que ce type d'architecture a été fondamentalement conçu dans le but de reproduire le fonctionnement du cortex visuel, le type d'entrée le plus utilisé est l'image. Par conséquent, sa structure de données peut-être représentée par une matrice tridimensionnelle $\mathbb{R}^{n\times m\times k}$ où chaque cellule correspond à un pixel contenant une valeur entre 0 et 255. Les dimensions de cette matrice correspondent aux caractéristiques suivantes :

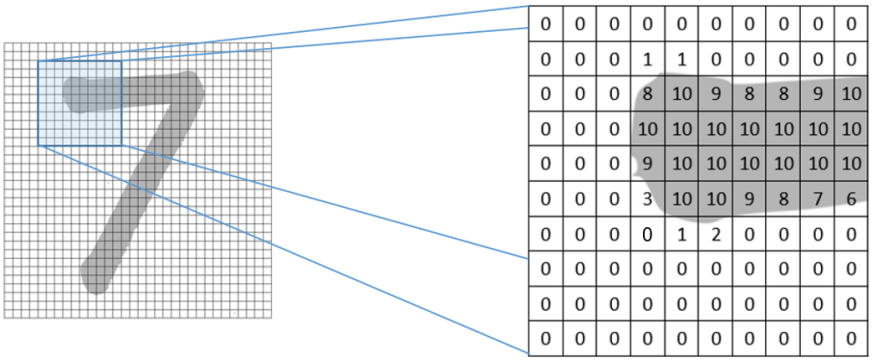
- Une dimension pour définir la hauteur de l'image ;
- Une dimension qui définit la largeur de l'image ;
- Une dimension de profondeur utilisée pour définir les trois couleurs (RGB) de l'image. Si l'image est en noir et blanc il n'y aurait qu'une seule couche de profondeur.

Sachant cela, le traitement d'image pose un problème majeur pour les architectures traditionnelles. Effectivement, si une image de 1000x1000 pixels avec ses 3 couches RGB est fournie en entrée, 3 millions de neurones sont nécessaires rien que pour capturer et traiter l'ensemble de ces valeurs. Ceci a pour effet, si la couche suivante contient par exemple 1000 neurones pour le traitement suivant, de nécessiter une matrice de poids $W^{(i)}(1000{,}3000000)$. Ces trois milliards de poids ont pour conséquence de nécessiter un jeu de données de test immense pour éviter le surapprentissage du modèle et ils ont aussi pour conséquence d'exercer un trop grand impact en ressources machine (processeur et mémoire) pour entraîner le modèle. Les

réseaux de neurones convolutifs apportent donc une solution à cela en proposant de dissocier le prétraitement en fils distincts qui, une fois rendus suffisamment abstraits, sont à nouveau passés à travers un réseau interconnecté afin d'être classifiés.

Dans le cas présent, nous utiliserons seulement une image en noir et blanc comme référence afin de simplifier la description du fonctionnement du procédé de convolution. Ceci nous donnera donc une matrice à seulement deux dimensions (hauteur x largeur).

Figure 26 : Représentation d'une image sous forme de matrice



(Thery Ehrlich, 2019)

4.1.1.1 L'opération de convolution

L'opération de convolution consiste à appliquer un filtre par balaiement itératif sur toutes les zones d'une image qui est fournie en entrée. Ce filtre, tout comme l'image, est une matrice de valeurs (ou poids) dont le rôle en première couche est de détecter des caractéristiques simples, comme par exemple les bords (verticaux, horizontaux et diagonaux), qui sont présents dans l'image. Par convention, le filtre est carré ou cubique, ses hauteur et largeur sont d'habitude impaires et son nombre de canaux est identique à l'image source (5x5 ou 5x5x3 par exemple). De plus, il existe toute une sélection de filtres qui permettent de détecter les bords dans une image. Dans le cas présent, un filtre de détection de type Sobel est utilisé.

Figure 27 : Filtre 5x5 de type Sobel

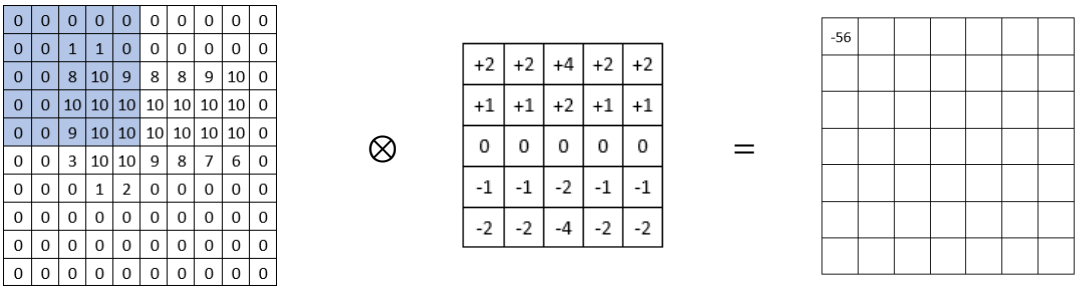
+2	+2	+4	+2	+2
+1	+1	+2	+1	+1
0	0	0	0	0
-1	-1	-2	-1	-1
-2	-2	-4	-2	-2

(Thery Ehrlich, 2019)

Le processus de balayage consiste à placer le filtre au coin supérieur gauche de l'image pour capturer une première zone de l'image – le champ réceptif. Il s'agit ensuite

d'appliquer un produit scalaire entre ce champ et le filtre puis d'en faire la somme pour obtenir une première valeur convolée. Ce résultat est ensuite placé dans une nouvelle matrice de qui sera plus compressée que celle d'origine.

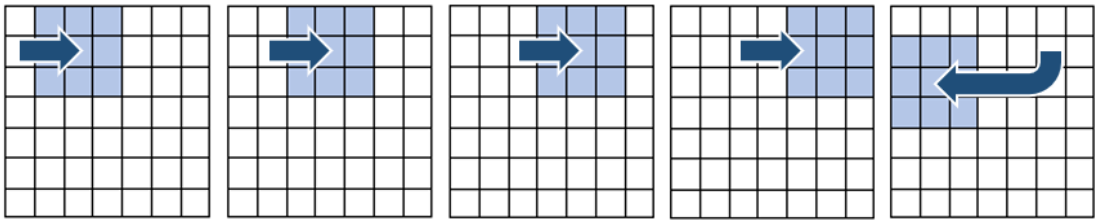
Figure 28 : Itération de convolution



(Thery Ehrlich, 2019)

Une fois ce processus terminé, le filtre se décale d'un ou plusieurs pas (s'il s'agit d'une grande image par exemple) sur la droite pour effectuer le même traitement sur le nouveau champ.

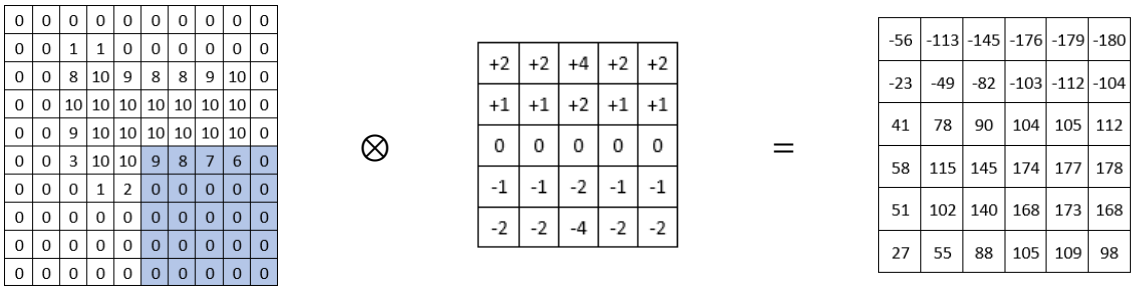
Figure 29 : Balayage convolutif



(Thery Ehrlich, 2019)

Ceci est itéré jusqu'à ce que toute l'image soit convolée. La matrice de valeurs convolée est ensuite prête à être utilisée pour l'opération suivante.

Figure 30 : Convolution terminée

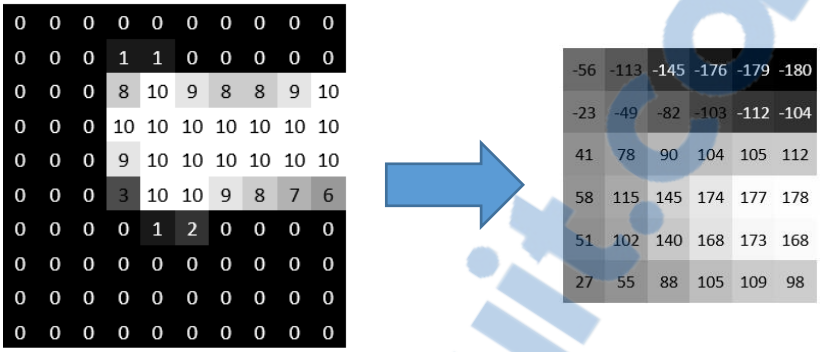


(Thery Ehrlich, 2019)

Ici, pour la détection des bords horizontaux, la représentation de l'image est plus ou moins préservée sachant que le champ effectif représente une portion de l'image qui

correspond à un trait horizontal. Le fait d'appliquer d'autres filtres avec des angles différents peut en contrepartie ne pas préserver la représentation de l'image.

Figure 31 : Comparaison entre l'entrée et la caractéristique convolée



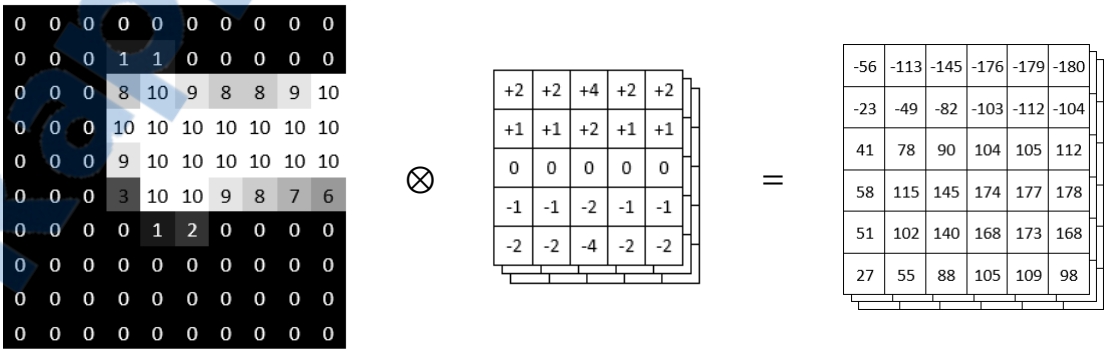
Matrice originale Matrice convolée

(Thery Ehrlich, 2019)

Il est important de noter que le filtre utilisé ici a été manuellement défini. D'habitude, le modèle apprend automatiquement quels filtres il doit utiliser sachant qu'au départ, chaque filtre est initialisé avec des valeurs aléatoires puis entraîné. Ceci permet de généraliser les filtres pour qu'ils puissent être utilisés pour un maximum de cas de figure. Ce processus de mise à jour et d'optimisation des filtres se fait d'ailleurs par rétropropagation.

De plus, il est possible de détecter différents types de bords dans une image en une seule passe en ajoutant une troisième dimension au filtre pour ajouter autant de filtres que nécessaire. Ceci a aussi pour effet de produire une matrice de valeurs convolées à trois dimensions au lieu de deux.

Figure 32 : Application de filtres multiples sur une seule passe de convolution



(Thery Ehrlich, 2019)

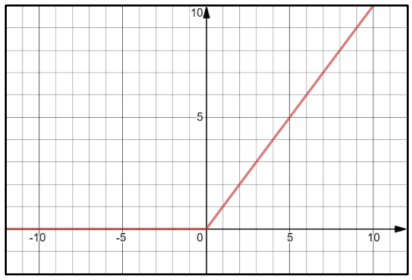
D'un point de vu algébrique, l'opération de convolution est représentée de la manière suivante[36] :

$$(f * g)(i) = \sum_{j=1}^{m} g(j) \cdot f(i - j + m/2)$$

4.1.1.2 L'opération d'activation

En regardant la matrice convolée de la figure 28, Il est possible de constater que le motif dessiné par les plus grandes valeurs collectées reproduit grossièrement la matrice originale. Mais il y a aussi des valeurs négatives aberrantes en haut et en bas de l'image qui représentent l'autre côté du trait. Il s'agit alors de passer la matrice convolée à travers la fonction d'activation qui le cas échéant est une fonction ReLU (Rectified Linear Unit).

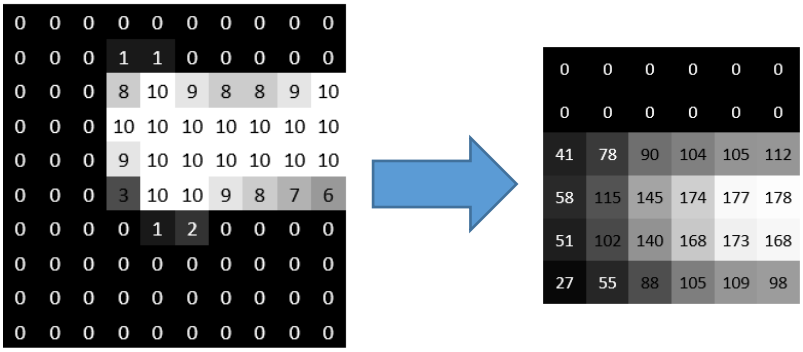
Figure 33 : Fonction ReLU



(Thery Ehrlich, 2019)

Cette fonction a pour rôle de réduire à zéro toute valeur négative et de préserver les valeurs positives telles qu'elles ont été calculées.

Figure 34 : Résultat de l'opération ReLU



Matrice originale Matrice convolée rectifiée

(Thery Ehrlich, 2019)

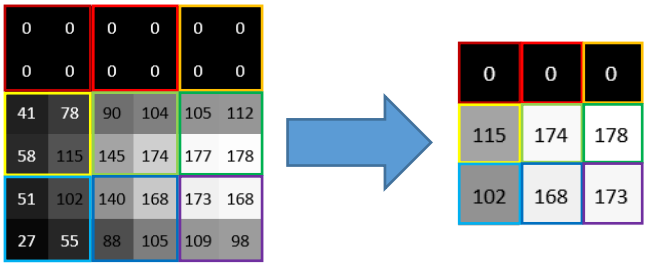
[36] ANON., 2013. CS1114 Section 6 : Convolution : [en ligne]. [Consulté le 27 septembre 2019]. Disponible à l'adresse : https://www.cs.cornell.edu/courses/cs1114/2013sp/sections/S06_convolution.pdf

La comparaison directe entre la matrice fournie en entrée et le résultat convolé et rectifié en figure 29 permet de constater que la forme initiale est bien préservée malgré le fait qu'elle est plus grossière. Cependant, il est aussi possible de confirmer que le filtre a bel et bien détecté qu'une portion de l'image contient un bord horizontal comme le démontre la zone avec les valeurs les plus élevées.

4.1.1.3 L'opération de *max pooling*

L'opération de *max pooling* est la dernière opération effectuée dans le contexte d'une itération de détection caractéristique. Elle consiste à effectuer un sous-échantillonnage de la matrice convolée dans le but de préserver uniquement les valeurs maximales de chaque champ traité et de retenir l'essentiel des caractéristiques de l'image.

Figure 35 : *Max pooling*



(Thery Ehrlich, 2019)

Une fois l'opération de *max pooling* effectuée, Il sera soit possible de relancer une itération de détection de caractéristique, soit lancer l'étape de classification pleinement connectée.

Ayant fait un tour d'horizon du traitement d'une portion de l'image initiale, si l'on reprend celle-ci en admettant qu'il s'agit aussi d'une image couleur, la démarche complète théorique serait la suivante :

1) L'image 32x32x3 est fournie en entrée ;
2) 6 différents filtres aux dimensions 5x5x3 (f = 5 dans la figure X) balaient par pas de 1 (p = 1) toute l'image ;
3) Une matrice convolée 28x28x6 est générée ;
4) La fonction d'activation ReLU est appliquée dans le but de rectifier la matrice et d'initialiser toutes les valeurs négatives à zéro ;
5) L'opération de *max pooling* est effectuée en balayant un filtre aux dimensions 2x2x6 par pas de 2 sur toute la matrice rectifiée ;

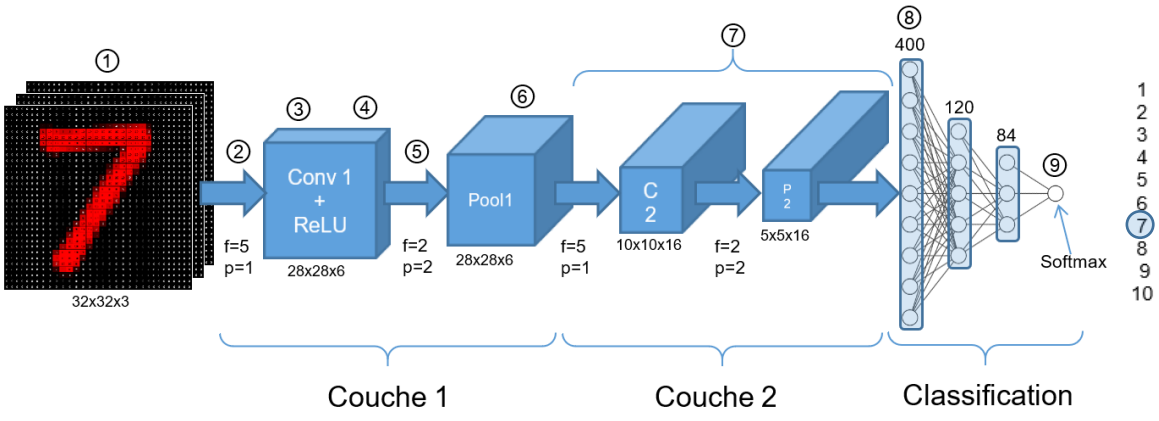
6) Une matrice sous-échantillonnée aux dimensions 14x14x6 est générée, contenant toutes les valeurs maximales des champs effectifs filtrés de la matrice ;

7) Réitérer une fois encore avec 16 filtres pour la nouvelle itération (choix qui dépend du cas d'utilisation) ;

8) Une fois l'étape de détection des caractéristiques terminées, en admettant que la seconde matrice de pooling à comme dimensions 5x5x16, celle-ci est aplatie en un vecteur de 400 dimensions (400x1). Ces 400 nœuds sont ensuite fournis dans une nouvelle couche pleinement connectée disposant de 120 nœuds et donc d'une matrice de poids de 120x400 valeurs plus 120 biais, soit 48'120 paramètres. Le tout est ensuite réduit davantage dans une couche finale de 84 nœuds dont la matrice des poids de 84x120 plus 84 biais compte 10164 paramètres ;

9) Finalement, les 84 unités restantes sont fournies dans un seul nœud qui déterminera parmi 10 sorties possibles la bonne valeur à prédire. Ce dernier nœud aura aussi pour rôle d'appliquer une fonction softmax qui somme l'ensemble des classes à 1 et permet d'exprimer la prédiction en une probabilité.

Figure 36 : Fonctionnement d'un réseau de neurones convolutif



(Thery Ehrlich, 2019)

4.1.2 Fonctionnement dans le cadre de l'analyse de sentiments[37,]

Sachant que la reconnaîssance de caractéristiques comme des objets ou émotions dans un image par le biais des concepts mis en avant par les réseaux de neurones convolutifs sont efficaces, l'idée de pouvoir adapter ce type d'architecture pour analyser du texte est tout aussi sensé. Mais en réalité, certains aspects appliqués à l'analyse d'image ne

[37] BRITZ, Denny, 2015. Understanding Convolutional Neural Networks for NLP. In : *WildML* [en ligne]. 7 novembre 2015. [Consulté le 17 septembre 2019]. Disponible à l'adresse : http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/.

peuvent être employés avec l'analyse de sentiments. Par exemple, la notion d'invariance spatiale n'a pas de sens dans le cas de l'analyse de texte car au contraire, la position des mots et lettres peut avoir un lourd impact sur la compréhension des mots et des phrases ainsi que de leurs subtilités. Aussi, dans le contexte d'une image, construire son « identité » en faisant évoluer les caractéristiques détectées durant chaque passe de convolution de caractéristiques de bas niveau (bords) à des caractéristiques de plus en plus complexes (formes et objets) a du sens. En revanche, dans le cas d'un fragment de texte, une lettre est une lettre, un mot est un mot, etc. et essayer de faire de l'abstraction de ce type de données n'est pas possible car le sens et le contexte d'une séquence de mots reste invariable. Sachant cela, l'utilisation d'un réseau de neurones convolutif ne semble pas être un très bon candidat pour faire du TALN. En effet, les réseaux de neurones récurrents semblent mieux correspondre à ce type d'activité, sachant qu'ils se rapprochent intuitivement de la façon dont l'être humain construit et analyse du texte, c'est-à-dire de manière séquentielle, en se référant toujours à ce qui précède le terme courant. Cependant, les réseaux de neurones convolutifs sont tout de même capables de correctement accomplir une tâche d'analyse de texte et le réel avantage de cette architecture réside avec le fait qu'elle est très rapide en termes de vitesse de traitement.

4.1.2.1 Word Embeddings

Dans son séminaire publié en 2017 pour les cours en ligne de l'université de Stanford[38], le professeur Christopher Manning explique qu'au départ le fait de construire des modèles où les mots sont des éléments atomiques pose rapidement un problème dans le sens où leur représentation se fait via ce que l'on appelle de l'encodage *one-hot* où le lexique de mots est un long vecteur de 0 et le mot qui nous intéresse est le seul 1 du vecteur. Ceci pose donc un problème, surtout sachant que certains lexiques vocabulaires utilisés en apprentissage profond peuvent compter plusieurs millions de mots (plus de 13 millions de mots pour le Web 1T 5-gram[39] corpus de Google par exemple). De plus, la nature atomique de ces représentations fait entièrement abstraction des relations entre les mots (par exemple les synonymes ou les équivalences de genre). Sachant cela, une solution possible consisterait à représenter les mots en tant que vecteurs dont les valeurs représentent d'autres mots qui ont un lien avec ceux-ci et qui sont eux aussi des vecteurs formatés de manière similaire (homme et femme

[38] STANFORD UNIVERSITY SCHOOL OF ENGINEERING, [sans date]. (10) Lecture 2 | Word Vector Representations: word2vec - YouTube. In : [en ligne]. [Consulté le 30 septembre 2019]. Disponible à l'adresse : https://www.youtube.com/watch?v=ERibwqs9p38.

[39] LDC, [sans date]. Web 1T 5-gram Version 1 - Linguistic Data Consortium. In : [en ligne]. [Consulté le 30 septembre 2019]. Disponible à l'adresse : https://catalog.ldc.upenn.edu/LDC2006T13.

pour reine et roi par exemple). C'est en se basant sur ce principe que l'on peut, via des bibliothèques de *word embeddings* comme word2vec[40] ou GloVe[41], construire de phrases composées de vecteurs contextuellement associables et utilisables dans les réseaux de neurones.

4.1.2.2 Réseau de neurones convolutif à une dimension, utilisant des *word embeddings*[42]

Sachant que le principe de base des réseaux de neurones convolutifs est de partir du plus simple (traits verticaux ou horizontaux) pour aller vers le plus abstrait (objets complexes), pour retranscrire ce principe dans un cas textuel, l'image est remplacée par une phrase ou un paragraphe disposant d'un sens subtilement défini par la combinaison et l'ordre des mots qui en font partie. Ceci étant dit, les mots peuvent donc être vus comme les blocs les plus simples de cette structure.

Le second aspect à s'appliquer aux réseaux de neurones est le fait que ceux-ci doivent impérativement recevoir des valeurs numériques en entrée. Il existe plusieurs approches pour passer du texte vers des valeurs numériques. La technique appelée « bag of words » est, par exemple, une méthode utilisée pour les modèles de classification de texte qui n'utilisent pas les réseaux de neurones et elle consiste à comptabiliser le nombre de fois que chaque mot apparaît dans le fragment de texte à classifier. L'objectif final est de transformer le texte en un vecteur dont la longueur correspond au nombre de mots présents dans la séquence. Le problème est que l'association des mots est ici ignorée et ceci réduit considérablement l'efficacité des modèles qui utilisent cette technique. C'est ici que les *word embeddings* prennent tout leur sens, sachant qu'ils permettent de préserver les relations entre les mots.

C'est en 2014 que Yoon Kim publie un document[43] inspiré par les travaux de Ronan Collobert et Jason Weston[44], décrivant plusieurs approches possibles pour effectuer de

---

40 MIKOLOV, Tomas, CHEN, Kai, CORRADO, Greg et DEAN, Jeffrey, 2013. Efficient Estimation of Word Representations in Vector Space. In : *arXiv:1301.3781 [cs]* [en ligne]. 16 janvier 2013. [Consulté le 30 septembre 2019]. Disponible à l'adresse : http://arxiv.org/abs/1301.3781.

41 ANON., [sans date]. GloVe: Global Vectors for Word Representation. In : [en ligne]. [Consulté le 1 octobre 2019]. Disponible à l'adresse : https://nlp.stanford.edu/projects/glove/.

42 SOCHER, Richard, [sans date]. (32) Lecture 13: Convolutional Neural Networks - YouTube. In : [en ligne]. [Consulté le 10 octobre 2019]. Disponible à l'adresse : https://www.youtube.com/watch?v=Lg6MZw_OOLI&list=PL3FW7Lu3i5Jsnh1rnUwq_TcylNr7EkRe6&index=14.

43 KIM, Yoon, 2014. Convolutional Neural Networks for Sentence Classification. In : *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)* [en ligne]. Doha, Qatar : Association for Computational Linguistics. 2014. p. 1746-1751. [Consulté le 10 octobre 2019]. Disponible à l'adresse : http://aclweb.org/anthology/D14-1181

44 COLLOBERT, Ronan, WESTON, Jason, BOTTOU, Leon, KARLEN, Michael, KAVUKCUOGLU, Koray et KUKSA, Pavel, [sans date]. Natural Language Processing (Almost) from Scratch. In : NATURAL

l'analyse de la classification de texte en combinant l'utilisation de *word embeddings* avec un réseau de neurones convolutif. Cette nouvelle approche consiste donc à prétraiter le jeu de données en transformant les mots en *word embeddings*. Les vecteurs résultants sont ensuite disposés dans une matrice où chaque rang correspond à un mot.

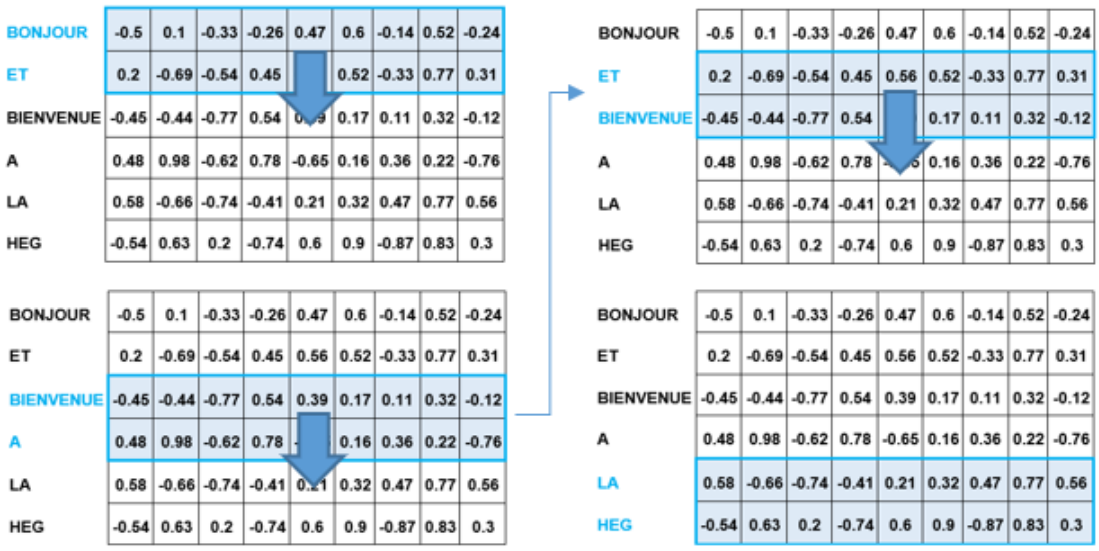
Figure 37 : Exemple fictif de *word embeddings* dans une matrice

BONJOUR	-0.5	0.1	-0.33	-0.26	0.47	0.6	-0.14	0.52	-0.24
ET	0.2	-0.69	-0.54	0.45	0.56	0.52	-0.33	0.77	0.31
BIENVENUE	-0.45	-0.44	-0.77	0.54	0.39	0.17	0.11	0.32	-0.12
A	0.48	0.98	-0.62	0.78	-0.65	0.16	0.36	0.22	-0.76
LA	0.58	-0.66	-0.74	-0.41	0.21	0.32	0.47	0.77	0.56
HEG	-0.54	0.63	0.2	-0.74	0.6	0.9	-0.87	0.83	0.3

(Thery Ehrlich, 2019)

Une fois ce prétraitement effectué, l'opération de convolution peut prendre place mais certains changements seront cependant aussi appliqués à cette opération. En effet, là où une image est convolée sur deux dimensions, via un mouvement de va et vient descendant, dans le cadre d'une classification de texte, la structure horizontale de la matrice ne doit en aucun cas être tronquée durant l'opération sachant que la caractéristique serait incomplète et la sémantique ne serait donc pas préservée.

Figure 38 : Exemple de convolution 1D

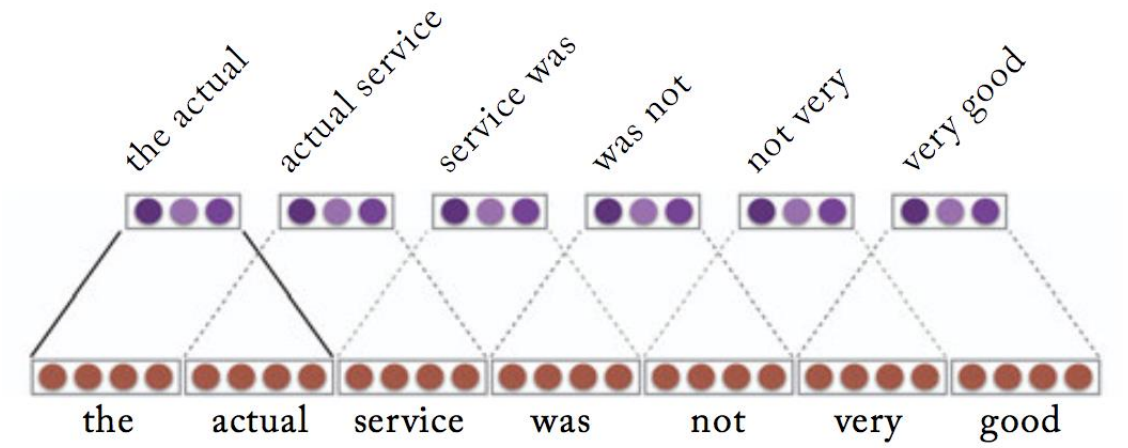


(Thery Ehrlich, 2019)

LANGUAGE PROCESSING. p. 45. [Consulté le 10 octobre 2019]. Disponible à l'adresse : http://www.jmlr.org/papers/volume12/collobert11a/collobert11a.pdf

Il s'agit donc balayer de haut en bas (la dimension temporelle) l'ensemble de la matrice avec divers filtres dont la largeur est identique à celle de la matrice mais la hauteur est variable. Cette hauteur de filtre définit le nombre de mots qui sont pris en compte dans une opération de convolution et cette combinaison de n mots porte le nom de n-gram où un uni-gram correspond à un mot ou le degré de granularité le plus bas. Par conséquent, un filtre à deux dimensions « temporelles » balaiera, par exemple, un groupe de 2-grams de la matrice. L'avantage de ce balaiement sur plusieurs dimensions temporelles est qu'elle permet de préserver le contexte sémantique qui est utilisé dans la classification de texte. Sachant cela, un grand nombre de filtres à dimensions variables seront utilisés pour prendre en compte des structures en 3-grams, 4-grams, etc. ce qui permet de récupérer d'avantage le contexte sémantique.

Figure 39 : visualisation de 2-grams



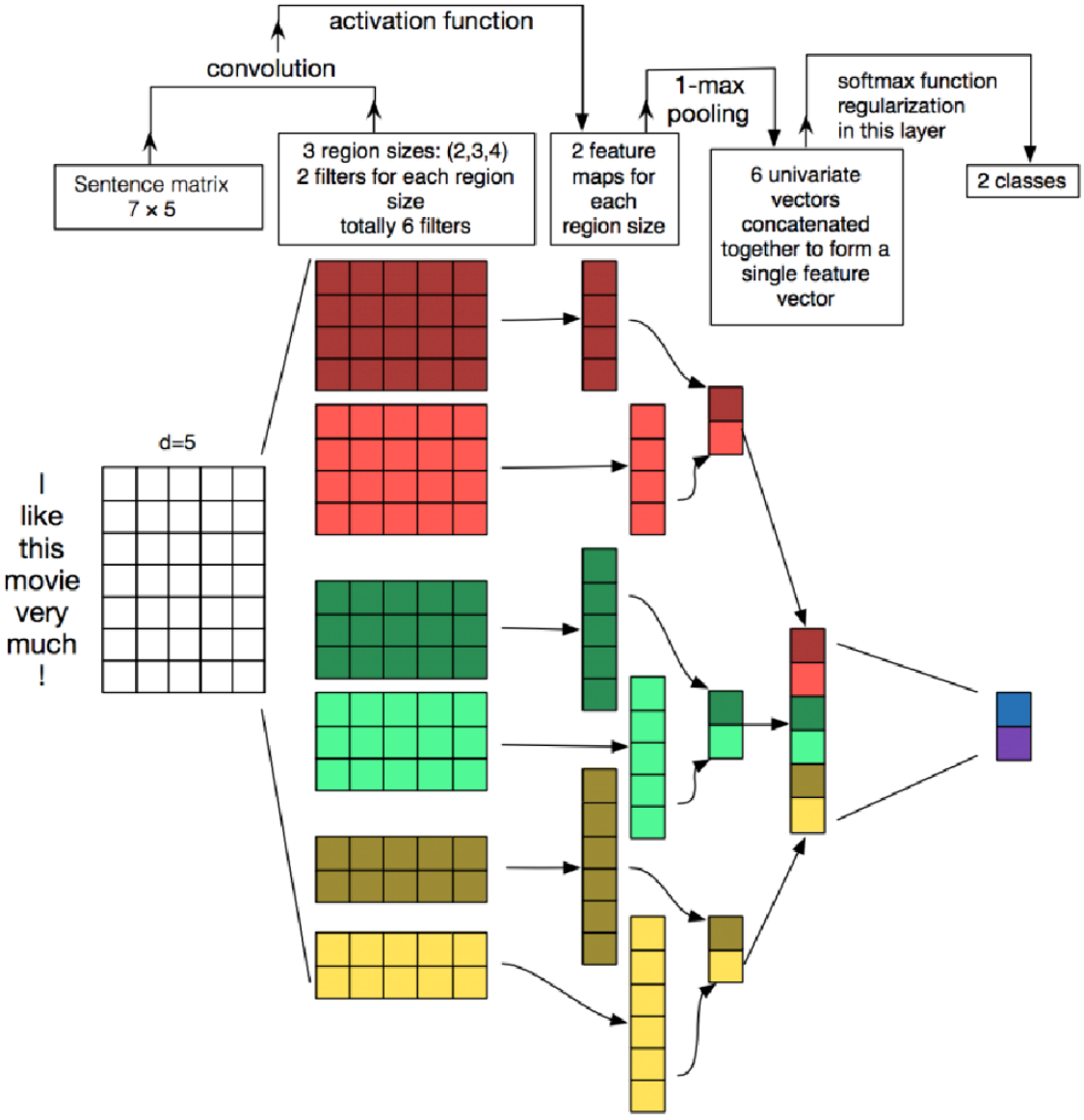
(Yoav Goldberg, 2017 – Neural Network Methods for NLP)

De plus ces filtres seront aussi des *word embeddings* d'abord initialisés aléatoirement (à moins d'utiliser des librairies d'*embeddings* pré-entraînés) qui s'entraîneront avec le modèle et finiront par peu à peu correspondre à des types de mots dont le contexte sémantique peut correspondre au mots fournis en entrée. Si un filtre finit par correspondre à un vecteur désignant des mots positifs, l'espoir est qu'il permette de capturer des valeurs de la matrice qui lui sont similaires et qui, via leur produit scalaire, retourneront des valeurs importantes.

Donc, une fois l'opération de convolution terminée, une fonction d'activation ReLU est appliquée au résultat convolé afin de mettre les valeur négatives à zéro. Ensuite, une opération de *max pooling* est effectuée sur le vecteur résultant et, comme pour le cas de classification d'image, cette opération consistera à retenir la valeur la plus grande produite par le produit scalaire entre le n-gram et le filtre.

Finalement, le résultat est passé dans une couche dense, pleinement connectée, comme pour le cas de classification d'image pour effectuer la tâche de classification.

Figure 40 : Réseau de neurones convolutif 1D avec *word embeddings*



(Zhang et Wallace, 2015 – https://arxiv.org/pdf/1510.03820.pdf)

4.1.2.3 Réseau de neurones convolutif à une dimension, utilisant des *character embeddings*[45]

Alternativement au modèle précédent, Xiang Zhang, Junbo Zhao et Yann Le Cun ont publié en 2015 un document décrivant une approche qui consiste à utiliser une représentation vectorielle à l'échelle des caractères au lieu des mots. Ceci se fait via

[45] ZHANG, Xiang, ZHAO, Junbo et LECUN, Yann, 2015. Character-level Convolutional Networks for Text Classification. In : *arXiv:1509.01626 [cs]* [en ligne]. 4 septembre 2015. [Consulté le 19 septembre 2019]. Disponible à l'adresse : http://arxiv.org/abs/1509.01626.

l'utilisation d'un encodage *one-hot* de caractères où un alphabet complet doté aussi de caractères spéciaux est utilisé pour quantifier une phrase fournie en entrée.

Figure 41 : Encodage *one-hot* de mots (quantification)

	F	A	C	E	B	O	O	…
A	0	1	0	0	0	0	0	…
B	0	0	0	0	1	0	0	…
C	0	0	1	0	0	0	0	…
D	0	0	0	0	0	0	0	…
E	0	0	0	1	0	0	0	…
F	1	0	0	0	0	0	0	…
…	…	…	…	…	…	…	…	…
.	0	0	0	0	0	0	0	…
,	0	0	0	0	0	0	0	…
-	0	0	0	0	0	0	0	…

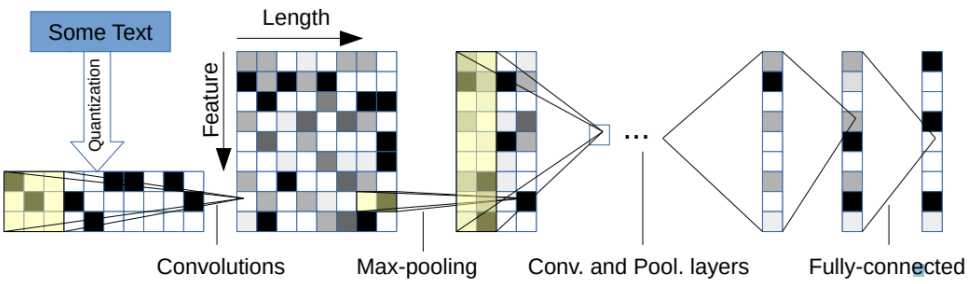
(Thery Ehrlich, 2019)

Comme pour le modèle précédent, sachant que l'opération de convolution doit pourvoir être effectuée sur un une matrice aux dimensions fixes, il est nécessaire de définir une taille limite pour la donnée fournie en entrée. Ceci aura pour effet de soit mettre à zéro les cellules restantes si la phrase fournie est trop petite, soit tronquer le texte si celui-ci s'avère être trop grand.

Une fois ce prétraitement effectué, des opérations de convolution sont effectuées avec des filtres multiples comme pour le modèle précédent. Ceci génère des matrices convolées sur lesquelles des opérations de *max pooling* sont appliquées dans le but de graduellement réduire le nombre de caractéristiques (sous-échantillonnage).

Une fois les opérations de convolution terminées, les matrices sont désormais des vecteurs convolés. Ces vecteurs sont ensuite concaténés les uns derrière les autres dans le but de finir avec un unique vecteur « aplati » qui est fourni en entrée à la couche dense afin d'effectuer la tâche de classification.

Figure 42 : Réseau de neurones convolutif 1D avec *character embeddings*



(Zhang, Zhao et Le Cun, 2015 – https://arxiv.org/pdf/1510.03820.pdf)

5. Mise en pratique

Dans le cadre de la mise en pratique d'un réseau de neurones convolutif capable de classifier du texte, nous allons procéder à une analyse de sentiments basée sur un jeu de données préexistant. Il s'agit du jeu de données de critiques de films de l'université de Stanford qui est déjà labellisé. Le choix d'utiliser un jeu de données non liées à la problématique du présent travail vient du fait qu'aucun jeu de données de tickets de support labellisé n'existe à ce jour et le temps nécessaire à sa création dans le cadre de ce projet ne serait pas réalisable dans les délais impartis.

En contrepartie, les grandes lignes d'une solution orientée service de support sera détaillé à la fin du document.

5.1 Objectif et démarche utilisée

Il s'agira de créer un réseau de neurones convolutif à une dimension qui se servira de *word embeddings* et qui aura pour responsabilité de classifier les critiques de films soit en positif, soit en négatif.

La démarche utilisée est la suivante :

1) Ajouter une séquence de remplissage composée de zéros à chaque fin de séquence dans le but de normaliser la longueur de chaque critique ;
2) Transformer les mots en *word embeddings*, c'est-à-dire en vecteurs denses représentant chaque mot et les mots qui lui sont associés ;
3) Faire passer les termes vectorisés plusieurs fois à travers une couche de convolution unidimensionnelle, une fonction ReLU et une opération de *max pooling* ;
4) Passer le vecteur résultant dans une couche pleinement connectée pour entamer la procédure de classification ;
5) Appliquer un *drop out* pour empêcher le modèle d'apprendre les données par cœur ;
6) Injecter les résultats de la couche précédente dans une couche avec un neurone unique doté d'une fonction d'activation sigmoïde pour effectuer la classification.

5.2 Outils

5.2.1 TensorFlow[46, 47]

Initialement développée par Google Brain en 2011 sous le nom de DistBelief, TensorFlow est disponible au grand public depuis 2017. Il s'agit d'une librairie d'apprentissage machine libre qui fournit un ensemble d'outils nécessaires à la création de modèles d'apprentissage machine robustes et faciles à déployer.

Utilisable avec les langages populaires Python, C++, Java et Go, TensorFlow propose de quoi facilement modéliser des solutions par le biais d'APIs de haut niveau comme Keras.

5.2.2 Keras[48]

Keras est une API de haut niveau développée pour le langage de programmation Python et qui facilite la production rapide de modèles d'apprentissage profonds. Il repose entre par-dessus les librairies TensorFlow de Google, CNTK de Microsoft et Theano. Il permet de prototyper des réseaux de neurones convolutifs, récurrents, ainsi que des modèles qui combinent les deux architectures. Tout comme TensorFlow, Keras est conçu pour utiliser le CPU et GPU pour accélérer le temps de traitement.

Les atouts principaux de Keras sont sa facilité d'utilisation, sa capacité à être modulable, et son extensibilité.

5.3 Mise en œuvre

5.3.1 Préparation des données[49]

Le jeu de données utilisé dans le cadre de cette application pratique est un jeu de 25'000 critiques de films tirées du site IMDB et divisé en 12'500 critiques positives et négatives. Chaque critique est une séquence de nombres entiers où chaque nombre correspond à l'indice de popularité d'un mot particulier. Chaque indice correspond donc à la fréquence d'utilisation d'un mot, l'indice 5 correspondant, par exemple, au 5ème mot le plus utilisé dans la donnée étudiée. Ceci permet de pouvoir considérablement réduire le nombre de mots à analyser en nous focalisant uniquement sur les mots le plus communs que nous souhaitons inclure dans notre analyse. Dans notre cas de figure, nous ne prenons en

[46] TENSORFLOW, [sans date]. TensorFlow. In : [en ligne]. [Consulté le 6 octobre 2019]. Disponible à l'adresse : https://www.tensorflow.org/.

[47] WIKIPEDIA, [sans date]. TensorFlow. In : [en ligne]. [Consulté le 6 octobre 2019]. Disponible à l'adresse : https://www.tensorflow.org/.

[48] KERAS, [sans date]. Home - Keras Documentation. In : [en ligne]. [Consulté le 6 octobre 2019]. Disponible à l'adresse : https://keras.io/.

[49] KERAS, [sans date]. Datasets - Keras Documentation. In : [en ligne]. [Consulté le 8 octobre 2019]. Disponible à l'adresse : https://keras.io/datasets/#imdb-movie-reviews-sentiment-classification.

compte que les 5000 mots les plus utilisés. Chaque critique est aussi accompagnée d'un label désignant le sentiment de celle-ci qui peut être soit positive, soit négative.

Figure 43 : Création des jeux d'entraînement, validation et test

```
# Training, Validation & Test set creation

max_words = 5000
maxlen = 400

(train_data, train_labels), (val_data, val_labels) = imdb.load_data(
    num_words=max_words,
    skip_top=0,
    seed=1753,
    start_char=1,
    oov_char=2,
    index_from=3)

val_data, test_data = np.split(val_data, 2)
val_labels, test_labels = np.split(val_labels, 2)
```

(Thery Ehrlich, 2019)

Pour notre modèle, nous divisons notre jeu de données en trois parties :

- Le jeu d'entraînement : 12'500 échantillons et labels ;
- Le jeu de validation : 6'250 échantillons et labels ;
- Le jeu de test : 6'250 échantillons et labels.

Le jeu de données est divisé ainsi afin de pouvoir entraîner puis contrôler la performance du modèle avec deux jeux de données distincts. Ceci permet de s'assurer que le modèle n'a pas appris par cœur les données d'entraînement.

5.3.2 Remplissage de séquence

Une fois les jeux d'entraînement, validation et test crées, il s'agit de normaliser la structure de chaque critique pour que l'ensemble des données aient la même structure.

Figure 44 : Remplissage et troncage des séquences

```
# Pad word sequences to same length

train_data = sequence.pad_sequences(train_data, maxlen=maxlen)
val_data = sequence.pad_sequences(val_data, maxlen=maxlen)
test_data = sequence.pad_sequences(test_data, maxlen=maxlen)
```

(Thery Ehrlich, 2019)

Dans notre cas de figure, nous avons opté pour une longueur maximale de 400 mots par critique. Si une critique s'avère être trop courte, un remplissage de séquence est appliqué où l'espace vide est rempli avec des zéros pour compenser les mots

manquants. Si au contraire le nombre de mots d'une critique dépasse la limite de 400 mots, la séquence est tronquée.

5.3.3 Définition du modèle

L'étape suivante consiste à définir la structure du modèle utilisé pour entraîner puis prédire le sentiment de nos données de test.

Figure 45 : Définition du modèle

```
# Model definition

model = Sequential()

model.add(Embedding(max_words, embedding_dims, input_length=maxlen))
model.add(Dropout(dropout))

model.add(Conv1D(filters, filter_size, activation='relu'))
model.add(MaxPooling1D())
model.add(Conv1D(filters, filter_size, activation='relu'))
model.add(MaxPooling1D())
model.add(Conv1D(filters, filter_size, activation='relu'))
model.add(GlobalMaxPooling1D())

model.add(Dense(hidden_dims, activation='relu'))
model.add(Dropout(dropout))
model.add(Dense(1, activation='sigmoid'))
```

(Thery Ehrlich, 2019)

Une couche d'*embedding*[50] en début de modèle permet de transformer les indices de mots en des vecteurs denses (*word embeddings*) de taille fixe. Dans notre cas, la couche d'*embedding* prend comme paramètres notre vocabulaire, soit le nombre de mots à considérer (max_words), le nombre de dimensions pour chaque vecteur correspondant à un mot (embedding_dims) et la longueur fixe des séquences (maxlen) qui sont passées dans le modèle.

Ensuite les *word embeddings* sont passés dans une couche de convolution à une dimension dotée d'une fonction d'activation ReLU. Une opération de *max pooling* est ensuite appliquée pour sous-échantillonner le résultat convolé. Ceci est répété à trois reprises sauf pour l'opération de *max pooling* qui est remplacé par une opération finale de *max pooling* global qui applique en même temps le *max pooling* et aplatit le résultat obtenu en un long vecteur prêt à être classifié.

Le vecteur est enfin passé à travers une couche dense pleinement connectée (un réseau de neurones classique). Une dernière couche dense est finalement utilisée pour

[50] KERAS, [sans date]. Embedding Layers - Keras Documentation. In : [en ligne]. [Consulté le 8 octobre 2019]. Disponible à l'adresse : https://keras.io/layers/embeddings/.

classifier le résultat via une fonction sigmoïde sachant qu'il s'agit d'un cas de classification binaire. Si l'analyse de sentiments portait sur plus de deux classes, la fonction sigmoïde serait remplacée par une fonction softmax.

Notons qu'après la couche d'*embedding* et la première couche dense, un dropout[51] est appliqué aux données pour aléatoirement en rendre certaines nulles. Ceci permet d'appliquer de la régularisation et éviter que le modèle soit surentraîné.

5.3.4 Compilation

C'est au moment de la compilation que nous allons définir le type d'algorithme d'optimisation et la fonction de coût à utiliser.

Figure 46 : Compilation du modèle définit

```
model.compile(optimizer=optimizers.adam(lr=lr), loss='binary_crossentropy',
              metrics=['accuracy'])
```

(Thery Ehrlich, 2019)

Dans le cadre de cette mise en pratique nous avons opté pour l'algorithme d'optimisation Adam, que nous avons survolé plus tôt dans le document, et une fonction de coût d'entropie croisée binaire, sachant que les prédictions de ce modèle ne pourront être que positives ou négatives.

5.3.5 Pré-entraînement

Avant d'entraîner le modèle il est nécessaire de déterminer les bons hyperparamètres à utiliser afin de maximiser les chances d'avoir un modèle final aussi performant que possible.

Figure 47 : Entraînement avec des hyperparamètres aléatoires

```
for i in range (20):

        # Random Hyper-parameter Testing

        batch_size = random.choice([8, 16, 32, 64, 128])
        embedding_dims = random.randint(10, 100)
        filters = random.choice([25, 50, 75, 100, 125, 150, 175, 200, 250, 300, 350, 400, 450, 500])
        filter_size = 3
        hidden_dims = random.randint(50, 500)
        dropout = random.choice([0.1, 0.2, 0.3, 0.4, 0.5])
        lr = random.choice([0.001, 0.0001])
```

(Thery Ehrlich, 2019)

[51] KERAS, [sans date]. Core Layers - Keras Documentation. In : [en ligne]. [Consulté le 8 octobre 2019]. Disponible à l'adresse : https://keras.io/layers/core/.

La première étape consiste à entraîner le modèle avec des hyperparamètres aléatoires. Dans le cadre de ce travail nous avons effectué un total de 20 passes de pré-entraînement. Les résultats obtenus ont permis de d'établir des valeurs d'hyperparamètres qui laissent présager un scénario idéal.

Figure 48 : Entraînement avec des lots d'hyperparamètres plus précis

```
# Hyper-parameter Fine-Tuning

batch_sizes_lst = [250, 500, 1000]
embedding_dims_lst = range(22, 30, 1)
filters_lst = range(350, 450, 25)
filter_size = 3
hidden_dims_lst = range(300, 400, 25)
dropout_list = np.arange(0.1, 0.4, 0.1)
lr_lst = [0.001, 0.0001]

for batch_size in batch_sizes_lst:
    for embedding_dims in embedding_dims_lst:
        for filters in filters_lst:
            for hidden_dims in hidden_dims_lst:
                for dropout in dropout_list:
                    for lr in lr_lst:
```

(Thery Ehrlich, 2019)

Suite à cela, une nouvelle série de passes, cette fois-ci avec des hyperparamètres mieux définis, est effectuée et les résultats sont ensuite divisés en lots pour comparer puis extraire le meilleur résultat possible. Une fois le résultat le plus performant déterminé, il est appliqué au modèle puis affiné davantage pour produire les meilleures prédictions possibles.

Figure 49 : Hyperparamètres finaux

```
# Hyper-parameter settings

batch_size = 8
embedding_dims = 22
filters = 425
filter_size = 3
hidden_dims = 325
dropout = 0.4
lr = 0.0001
```

(Thery Ehrlich, 2019)

Durant le processus ci-dessus, des tailles de lots de données (batch_size) plus grands ont été utilisés pour accélérer le processus. Une fois le préapprentissage terminé, les tailles de lots ont été réduits à 8 échantillons par lot pour augmenter la performance du modèle.

5.3.6 Entraînement

Une fois les hyperparamètres finaux déterminés, le modèle est ré-entraîné 10 fois pendant 50 époques afin de définir les poids qui permettront d'obtenir la meilleure performance possible.

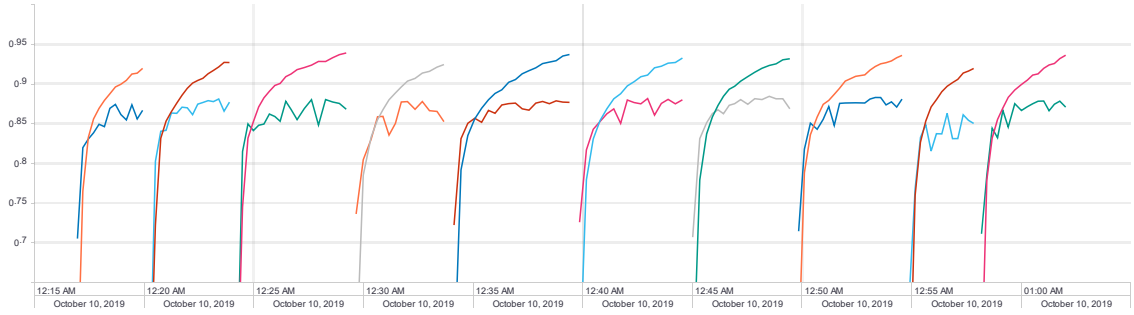
Figure 50 : Fonction d'entraînement du modèle

```
model.fit(train_data, train_labels, epochs=50, batch_size=batch_size,
          validation_data=(val_data, val_labels), callbacks=callback_list)
```

(Thery Ehrlich, 2019)

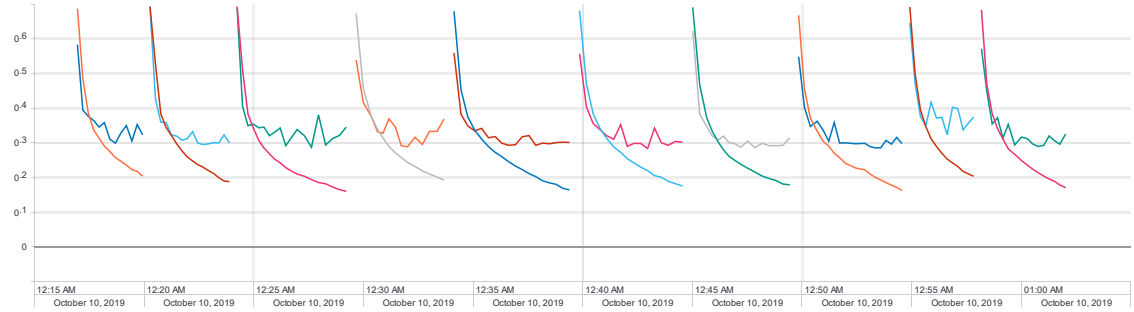
Grâce à TensorFlow, nous utilisons aussi de l'*early-stopping* pour interrompre l'entraînement du modèle dès que sa performance stagne durant un certain nombre d'époques. Pour notre cas, l'*early-stopping* interrompt l'entraînement entre la 12ème et la 16ème époque.

Figure 51 : Performance par rapport aux jeux d'entraînement et validation



(Thery Ehrlich, 2019)

Figure 52 : Erreur par rapport aux jeux d'entraînement et validation

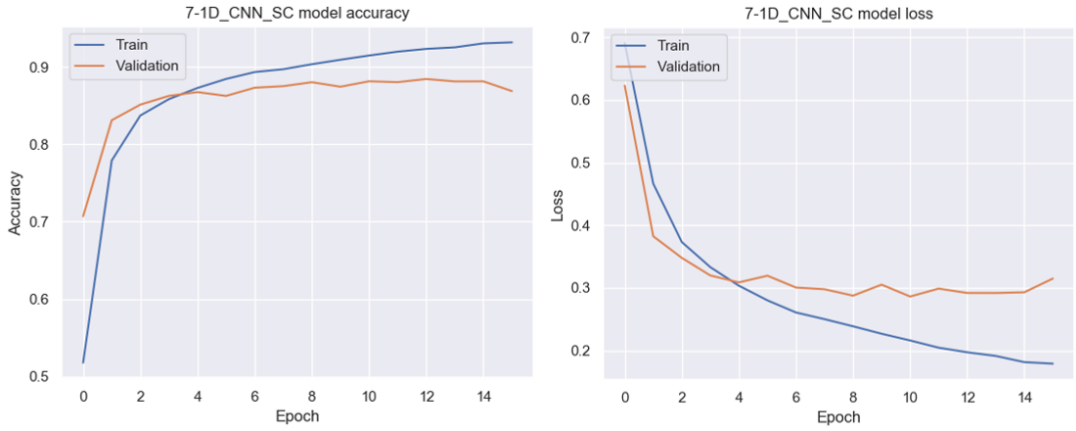


(Thery Ehrlich, 2019)

À la suite de ces multiples passes d'entraînement, il est possible d'observer que le modèle 7-1D_CNN_SC est celui qui arrive à prédire le mieux les données de validation avec une performance maximale se situant à 0.8846 contre 0.2861 pour l'erreur. Le

choix d'avoir effectué plusieurs passes d'entraînement aura donc permis de récupérer le meilleur modèle possible.

Figure 53 : Performance et erreur du meilleur modèle



(Thery Ehrlich, 2019)

5.4 Prédiction et résultats avec des données inconnues

Le modèle étant désormais entraîné, il peut être utilisé pour prédire le sentiment d'une critique de film via la fonction *predict*.

La fonction permet de sortir un pourcentage de performance sur un échantillon aléatoire de données pris du jeu de test, jusqu'ici non- utilisé. Il est aussi possible de faire imprimer les échantillons, leurs scores et leurs labels afin de visualiser la prédiction pour chaque donnée.

Figure 54 : Fonction *predict*

```
# Predict a random sample review from the test_data dataset

results = []
accuracy = 0

for i in range(100):
    random_sample = random.randint(0, len(test_data) - 1)

    test_prediction = model.predict(test_data)
    test_review_text = ' '.join(id_to_word[id] for id in np.trim_zeros(test_data[random_sample]))

    print('X = {}'.format(test_review_text))
    print('Predicted = {}'.format(test_prediction[random_sample]))
    print('Label = {}'.format(test_labels[random_sample]))

    if (test_prediction[random_sample] > 0.5 and test_labels[random_sample] == 1) or \
            (test_prediction[random_sample] < 0.5 and test_labels[random_sample] == 0):
        results.append(1)
    else:
        results.append(0)

accuracy = sum(results)
print('Accuracy: {}%'.format(accuracy))
```

(Thery Ehrlich, 2019)

Concernant les résultats obtenus, nous pouvons observer une performance moyenne de 88.4% sur 10 échantillons aléatoires de 100 critiques chacun.

Figure 55 : Résultats après 10 passes

```
Accuracy: 85%
Accuracy: 91%
Accuracy: 90%
Accuracy: 88%
Accuracy: 97%
Accuracy: 88%
Accuracy: 82%
Accuracy: 89%
Accuracy: 85%
Accuracy: 89%
```

(Thery Ehrlich, 2019)

Les résultats obtenus permettent donc de voir que le modèle prédit efficacement le sentiment des critiques sachant que le jeu compte 50% de critiques négatives contre 50% de critiques positives. Le fait que la performance du modèle se situe au-dessus de cette valeur permet de conclure avec confiance que le modèle est efficace.

Figure 56 : Prédiction d'une critique positive

```
X = <START> the movie is not as funny as the director's <UNK> and only
other movie <UNK> <UNK> <UNK> did have its moments but it did not
satisfy me why it needed to be so <UNK> i don't know but i give <UNK>
Predicted = [0.36903933]
Label = 0
```

(Thery Ehrlich, 2019)

Figure 57 : Prédiction d'une critique négative.

```
X = <START> if you're going to spoof james bond it's a brilliant idea
to find a leading man who resembles both sean <UNK> and leonard so
step forward jean who captures perfectly the <UNK> <UNK> that <UNK> as
a smile plus the self <UNK> of thinking he is <UNK> plays it like a
<UNK> who can hold his own at <UNK> the plot has him looking for our
man in who has disappeared <UNK> and includes such <UNK> as a nazi
cell <UNK> inside a <UNK> if there is a <UNK> note it is the leading
lady who has all the <UNK> of a <UNK> <UNK> and is <UNK> but only just
by it's good for one viewing but that's about it<END>
Predicted = [0.7412836]
Label = 1
```

(Thery Ehrlich, 2019)

5.5 Difficultés rencontrées

Les plus grandes difficultés rencontrées dans le cadre de cette mise en pratique étaient, en premier lieu, le réglage des hyperparamètres qui demandaient beaucoup de temps

pour arriver à des résultats satisfaisants. En effet, au départ, il est important d'utiliser des valeurs aléatoires pour établir une base depuis laquelle il est ensuite possible d'affiner le modèle. Ceci n'était pas une évidence au départ et il a été nécessaire d'effectuer des expérimentations supplémentaires. En second lieu, les résultats obtenus au départ, en termes de performance et d'erreur, semblaient indiquer que le modèle était surentraîné et que, par conséquent, les données d'entrainement étaient apprises par cœur par le réseau. Ce n'était que partiellement vrai sachant que le principal défaut d'un réseau de neurones est qu'il nécessite un jeu de données relativement conséquent pour s'entraîner efficacement. La petite taille du jeu de données utilisé aura fait que le modèle n'arrivait pas à dépasser les 90% d'efficacité en termes de performance. C'est donc uniquement la taille du jeu de données qui contribuait à cette limitation.

5.6 Solution pour un service de support

Si un outil permettant de faire de l'analyse de sentiments pour déterminer la qualité d'un service de support devait être réalisé, les concepts d'analyse de sentiments détaillés dans ce travail resteraient fondamentalement les mêmes. Il serait en revanche nécessaire d'ajouter un ensemble de règles et traitements capables d'aussi prendre en compte les facteurs temporels et quantitatifs de l'échange entre le client et l'agent de support que l'analyse de sentiments seule ne pourrait pas fournir. Ces facteurs pourraient entre-autre inclure :

- Le délai de réponse ;
- Le délai de résolution ;
- Le nombre d'échanges nécessaires à la résolution du ticket ;
- Le sentiment du premier message ;
- Le sentiment des derniers messages ;
- Le la moyenne de messages positifs par rapport au messages négatifs.

Une ébauche de tâches à appliquer pour ce type de scénario pourrait être :

1) L'isolation des messages clients ;
2) L'analyse de sentiments de chaque message ;
3) La quantification des messages positifs par rapport aux messages négatifs où si les messages sont majoritairement négatifs un poids négatif est automatiquement appliqué ;

4) La prise en compte du sentiment du premier message pour établir si l'aspect négatif établit au point 3 provient d'une frustration initiale chez le client. Si ceci est le cas mais que la fin de l'échange se termine sur une note satisfaisante pour le client, un poids positif est alors appliqué ;

5) La prise en compte ensuite des trois derniers messages afin d'établir si le ticket se termine sur un point positif ou non. Par exemple, si le sentiment des trois derniers messages est négatif, le ticket est considéré comme étant négatif ;

6) L'ajout d'un taux plus ou moins négatif, se basant sur le temps d'attente entre la dernière réponse client et la réponse de l'agent ;

7) L'ajout d'un autre taux au résultat qui lui dépend du temps de résolution total.

Ce procédé permettrait d'obtenir un résultat positif et négatif suivant l'expérience client. Une valeur négative indiquerait une expérience insatisfaisante tandis qu'un résultat positif indiquerait une expérience client de qualité.

D'autres paramètres pourraient certainement être pris en compte à l'avenir pour améliorer l'algorithme de ce système d'évaluation client mais les points ci-dessus permettraient déjà d'établir une base sur laquelle se reposer. Aussi, ce modèle pourrait être adapté pour mesurer la performance des agents de support en ajustant les taux en fonction des attentes de la hiérarchie.

6. Conclusion

Malgré le regret de ne pas avoir eu le temps de découvrir d'autres modèles comme les LTSM récurrents, je pense que le fait d'avoir utilisé un réseau de neurones convolutif au lieu d'un réseau de neurones récurrent était ultimement le bon choix. En effet, bien que le réseau de neurones récurrent soit un choix populaire (étant donné qu'il utilise intuitivement une forme d'interprétation de texte similaire à celle de l'humain) et qu'il n'utilise aucune forme de rembourrage pour les données (étant donné qu'il peut accepter n'importe quelle longueur d'entrée, ce que le réseau de neurones convolutifs n'est pas capable de faire), le contexte d'utilisation au sein d'un service de support demande néanmoins une rapidité de traitement que le réseau de neurones convolutif seul peut efficacement fournir.

Il serait intéressant de se pencher un jour sur l'opportunité de labelliser un jeu de données de tickets de support et de chercher à mettre en place l'idée de solution qui m'avait donné envie de me lancer dans ce travail initialement.

L'expérience acquise suite à la rédaction de ce travail m'a permis de réellement apprécier la complexité et le haut potentiel que cette forme d'apprentissage machine présente ainsi que la place qu'elle occupe dans notre quotidien.

Bibliographie

1 LEXICO, [sans date]. natural language processing | Definition of natural language processing by Lexico. In : Lexico Dictionaries | English [en ligne]. [Consulté le 24 août 2019]. Disponible à l'adresse : https://www.lexico.com/en/definition/natural_language_processing

2 WIKIPEDIA., 2019. Traitement automatique du langage naturel [en ligne]. S.l. : s.n. [Consulté le 24 août 2019]. Disponible à l'adresse : https://fr.wikipedia.org/w/index.php?title=Traitement_automatique_du_langage_naturel&oldid=159047154

3 WIKIPEDIA, 2019. Opinion mining [en ligne]. S.l. : s.n. [Consulté le 11 septembre 2019]. Disponible à l'adresse : https://fr.wikipedia.org/w/index.php?title=Opinion_mining&oldid=159547346

4 LI, Nan et WU, Desheng Dash, 2010. Using text mining and sentiment analysis for online forums hotspot detection and forecast. In : Decision Support Systems. janvier 2010. Vol. 48, n° 2, p. 354-368. DOI 10.1016/j.dss.2009.09.003

5 UNIGE, 2019. Analyse de sentiments en text mining [en ligne]. S.l. : s.n. [Consulté le 11 septembre 2019]. Disponible à l'adresse : http://edutechwiki.unige.ch/fr/Analyse_de_sentiments_en_text_mining

6 MONKEYLEARN., 13:15. Text Classification: A Comprehensive Guide to Classifying Text with Machine Learning. In : MonkeyLearn [en ligne]. 13:15. [Consulté le 3 octobre 2019]. Disponible à l'adresse : https://monkeylearn.com/text-classification

7 UNIVERSITE DE GENEVE, 2019. La transmission de l'influx nerveux [en ligne]. S.l. : s.n. [Consulté le 29 août 2019]. Disponible à l'adresse : https://edu.ge.ch/decandolle/sites/localhost.decandolle/files/sn5-influx_nerveux2e_version.pdf

8 INDIANA UNIVERSITY, [sans date]. Understand measures of supercomputer performance and storage system capacity. In : [en ligne]. [Consulté le 27 septembre 2019]. Disponible à l'adresse : https://kb.iu.edu/d/apeq

9 OAK RIDGE LEADERSHIP COMPUTING FACILITY, [sans date]. Summit. In : Oak Ridge Leadership Computing Facility [en ligne]. [Consulté le 27 septembre 2019]. Disponible à l'adresse : https://www.olcf.ornl.gov/olcf-resources/compute-systems/summit/

10 DATABASE, Institute of Medicine (US) Committee on a National Neural Circuitry, PECHURA, Constance M. et MARTIN, Joseph B., 1991. Overview of Neuroscience Research: A Closer Look at the Neural Hierarchy [en ligne]. S.l. : National Academies Press (US). [Consulté le 3 octobre 2019]. Disponible à l'adresse : https://www.ncbi.nlm.nih.gov/books/NBK234389/

11 ROE, Aw, PALLAS, Sl, KWON, Yh et SUR, M, 1992. Visual projections routed to the auditory pathway in ferrets: receptive fields of visual neurons in primary auditory cortex. In : The Journal of Neuroscience. 1 septembre 1992. Vol. 12, n° 9, p. 3651-3664. DOI 10.1523/JNEUROSCI.12-09-03651.1992

12 HAWKINS, Jeff, AHMAD, Subutai et CUI, Yuwei, 2017. A Theory of How Columns in the Neocortex Enable Learning the Structure of the World. In : Frontiers in Neural Circuits [en ligne]. 2017. Vol. 11. [Consulté le 28 août 2019]. DOI 10.3389/fncir.2017.00081. Disponible à l'adresse : https://www.frontiersin.org/articles/10.3389/fncir.2017.00081/full#targetText=The%20input%20layer%20of%20each,neurons%20arranged%20in%20mini%2Dcolumns.&targetText=The%20feedforward%20input%20of%20cells,patterns%20(Jones%2C%202000)

13 LAGANDULA, Akshay Chandra, 2018. McCulloch-Pitts Neuron — Mankind's First Mathematical Model Of A Biological Neuron. In : Medium [en ligne]. 7 novembre 2018. [Consulté le 29 août 2019]. Disponible à l'adresse : https://towardsdatascience.com/mcculloch-pitts-model-5fdf65ac5dd1

14 PITTS, MCCULLOCH, 1943. A Logical Calculus of the Ideas Immanent in Nervous Activity [en ligne]. S.l. : s.n. [Consulté le 29 août 2019]. Disponible à l'adresse : http://www.cse.chalmers.se/~coquand/AUTOMATA/mcp.pdf

15 UNIVERSALIS, Encyclopædia, [sans date]. RÉSEAUX DE NEURONES FORMELS. In : Encyclopædia Universalis [en ligne]. [Consulté le 27 juillet 2019]. Disponible à l'adresse : http://www.universalis.fr/encyclopedie/reseaux-de-neurones-formels/

16 SHARMA, SAGAR, 2019. What the Hell is Perceptron? In : Medium [en ligne]. 5 mars 2019. [Consulté le 25 août 2019]. Disponible à l'adresse : https://towardsdatascience.com/what-the-hell-is-perceptron-626217814f53

17 ROSENBLATT, F., 1958. The perceptron: A probabilistic model for information storage and organization in the brain. In : Psychological Review. 1958. Vol. 65, n° 6, p. 386-408. DOI 10.1037/h0042519

18 SIMPLILEARN., [sans date]. What is Perceptron | Simplilearn. In : Simplilearn.com [en ligne]. [Consulté le 1 septembre 2019]. Disponible à l'adresse : https://www.simplilearn.com/what-is-perceptron-tutorial

19 KOHL NATE, 2010. artificial intelligence - Role of Bias in Neural Networks. In : Stack Overflow [en ligne]. [Consulté le 1 septembre 2019]. Disponible à l'adresse : https://stackoverflow.com/questions/2480650/role-of-bias-in-neural-networks

20 COURSERA, [sans date]. Machine Learning - Home. In : Coursera [en ligne]. [Consulté le 7 septembre 2019]. Disponible à l'adresse : https://www.coursera.org/learn/machine-learning/home/welcome

21 ML CHEATSHEET, [sans date]. Logistic Regression — ML Cheatsheet documentation. In : [en ligne]. [Consulté le 7 septembre 2019]. Disponible à l'adresse : https://ml-cheatsheet.readthedocs.io/en/latest/logistic_regression.html#cost-function

22 KINGMA, Diederik P. et BA, Jimmy, 2014. Adam: A Method for Stochastic Optimization. In : arXiv:1412.6980 [cs] [en ligne]. 22 décembre 2014. [Consulté le 2 octobre 2019]. Disponible à l'adresse : http://arxiv.org/abs/1412.6980

23 CHOROMANSKA, Anna, HENAFF, Mikael, MATHIEU, Michael, AROUS, Gérard Ben et LECUN, Yann, 2014. The Loss Surfaces of Multilayer Networks. In : arXiv:1412.0233 [cs] [en ligne]. 30 novembre 2014. [Consulté le 29 septembre 2019]. Disponible à l'adresse : http://arxiv.org/abs/1412.0233

24 JONES, 2017. A neural networks deep dive. In : IBM Developer [en ligne]. 24 juillet 2017. [Consulté le 25 août 2019]. Disponible à l'adresse : https://developer.ibm.com/articles/cc-cognitive-neural-networks-deep-dive/

25 GEVA, [sans date]. Backpropagation in Neural Networks: Process, Example & Code. In : MissingLink.ai [en ligne]. [Consulté le 8 septembre 2019]. Disponible à l'adresse : https://missinglink.ai/guides/neural-network-concepts/backpropagation-neural-networks-process-examples-code-minus-math/

26 NIELSEN, Michael, 2015. Neural Networks and Deep Learning. In : [en ligne]. 2015. [Consulté le 9 septembre 2019]. Disponible à l'adresse : http://neuralnetworksanddeeplearning.com

27 WERBOS, Paul et JOHN, Paul, 1974. (13) (PDF) Beyond regression : new tools for prediction and analysis in the behavioral sciences /. In : ResearchGate [en ligne]. [Consulté le 9 septembre 2019]. Disponible à l'adresse : https://www.researchgate.net/publication/35657389_Beyond_regression_new_tools_for_prediction_and_analysis_in_the_behavioral_sciences

28 BRITZ, Denny, 2015. Understanding Convolutional Neural Networks for NLP. In : WildML [en ligne]. 7 novembre 2015. [Consulté le 11 septembre 2019]. Disponible à l'adresse : http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/

29 IBM, 2018. Convolutional neural networks. In : IBM Developer [en ligne]. 2 mai 2018. [Consulté le 28 août 2019]. Disponible à l'adresse : https://developer.ibm.com/articles/cc-convolutional-neural-network-vision-recognition/

30 BINEY, Kingsley, 2019. Sentiment Analysis using 1D Convolutional Neural Networks in Keras. In : Medium [en ligne]. 1 mai 2019. [Consulté le 11 septembre 2019]. Disponible à l'adresse : https://medium.com/@romannempyre/sentiment-analysis-using-1d-convolutional-neural-networks-part-1-f8b6316489a2

31 WIKIPEDIA., 2019. Convolutional neural network [en ligne]. S.l. : s.n. [Consulté le 12 septembre 2019]. Disponible à l'adresse : https://en.wikipedia.org/w/index.php?title=Convolutional_neural_network&oldid=914010072

32 MISHRA, Mayank, 2019. Convolutional Neural Networks, Explained. In : [en ligne]. [Consulté le 13 septembre 2019]. Disponible à l'adresse : https://www.datascience.com/blog/convolutional-neural-network

33 DESHPANDE, Adit, [sans date]. A Beginner's Guide To Understanding Convolutional Neural Networks. In : [en ligne]. [Consulté le 13 septembre 2019]. Disponible à l'adresse : https://adeshpande3.github.io/adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/

34 LECUN, Yann et al., 1998. Gradient-Based Learning Applied to Document Recognition. In : [en ligne]. [Consulté le 29 septembre 2019]. Disponible à l'adresse : http://yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf

35 SLAZEBNI, Lana, [sans date]. Convolutional Neural Network Architectures: from LeNet to ResNet. In : [en ligne]. [Consulté le 29 septembre 2019]. Disponible à l'adresse : http://slazebni.cs.illinois.edu/spring17/lec01_cnn_architectures.pdf

36 ANON., 2013. CS1114 Section 6 : Convolution : [en ligne]. [Consulté le 27 septembre 2019]. Disponible à l'adresse : https://www.cs.cornell.edu/courses/cs1114/2013sp/sections/S06_convolution.pdf

37 BRITZ, Denny, 2015. Understanding Convolutional Neural Networks for NLP. In : WildML [en ligne]. 7 novembre 2015. [Consulté le 17 septembre 2019]. Disponible à l'adresse : http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/

38 STANFORD UNIVERSITY SCHOOL OF ENGINEERING, [sans date]. (10) Lecture 2 | Word Vector Representations: word2vec - YouTube. In : [en ligne]. [Consulté le 30 septembre 2019]. Disponible à l'adresse : https://www.youtube.com/watch?v=ERibwqs9p38

39 LDC., [sans date]. Web 1T 5-gram Version 1 - Linguistic Data Consortium. In : [en ligne]. [Consulté le 30 septembre 2019]. Disponible à l'adresse : https://catalog.ldc.upenn.edu/LDC2006T13

40 MIKOLOV, Tomas, CHEN, Kai, CORRADO, Greg et DEAN, Jeffrey, 2013. Efficient Estimation of Word Representations in Vector Space. In : arXiv:1301.3781 [cs] [en ligne]. 16 janvier 2013. [Consulté le 30 septembre 2019]. Disponible à l'adresse : http://arxiv.org/abs/1301.3781

41 ANON., [sans date]. GloVe: Global Vectors for Word Representation. In : [en ligne]. [Consulté le 1 octobre 2019]. Disponible à l'adresse : https://nlp.stanford.edu/projects/glove/

42 SOCHER, Richard, [sans date]. (32) Lecture 13: Convolutional Neural Networks - YouTube. In : [en ligne]. [Consulté le 10 octobre 2019]. Disponible à l'adresse : https://www.youtube.com/watch?v=Lg6MZw_OOLI&list=PL3FW7Lu3i5Jsnh1rnUwq_TcylNr7EkRe6&index=14

43 KIM, Yoon, 2014. Convolutional Neural Networks for Sentence Classification. In : arXiv:1408.5882 [cs] [en ligne]. 25 août 2014. [Consulté le 30 septembre 2019]. Disponible à l'adresse : http://arxiv.org/abs/1408.5882

44 COLLOBERT, Ronan, WESTON, Jason, BOTTOU, Leon, KARLEN, Michael, KAVUKCUOGLU, Koray et KUKSA, Pavel, [sans date]. Natural Language Processing (Almost) from Scratch. In : NATURAL LANGUAGE PROCESSING. p. 45. [Consulté le 10 octobre 2019]. Disponible à l'adresse : http://www.jmlr.org/papers/volume12/collobert11a/collobert11a.pdf

45 ZHANG, Xiang, ZHAO, Junbo et LECUN, Yann, 2015. Character-level Convolutional Networks for Text Classification. In : arXiv:1509.01626 [cs] [en ligne]. 4 septembre 2015. [Consulté le 19 septembre 2019]. Disponible à l'adresse : http://arxiv.org/abs/1509.01626

46 TENSORFLOW, [sans date]. TensorFlow. In : [en ligne]. [Consulté le 6 octobre 2019]. Disponible à l'adresse : https://www.tensorflow.org/

47 WIKIPEDIA, [sans date]. TensorFlow. In : [en ligne]. [Consulté le 6 octobre 2019]. Disponible à l'adresse : https://www.tensorflow.org/

48 KERAS, [sans date]. Home - Keras Documentation. In : [en ligne]. [Consulté le 6 octobre 2019]. Disponible à l'adresse : https://keras.io/.

49 KERAS, [sans date]. Datasets - Keras Documentation. In : [en ligne]. [Consulté le 8 octobre 2019]. Disponible à l'adresse : https://keras.io/datasets/#imdb-movie-reviews-sentiment-classification

50 KERAS, [sans date]. Embedding Layers - Keras Documentation. In : [en ligne]. [Consulté le 8 octobre 2019]. Disponible à l'adresse : https://keras.io/layers/embeddings/

51 KERAS, [sans date]. Core Layers - Keras Documentation. In : [en ligne]. [Consulté le 8 octobre 2019]. Disponible à l'adresse : https://keras.io/layers/core/

Annexe 1 : Dépôt du code de mise en pratique

https://github.com/ehrlicht/1D_CNN_SC