POLYPUBLIE
Polytechnique Montréal

POLYTECHNIQUE MONTRÉAL
UNIVERSITÉ D'INGÉNIERIE

Titre: Title:	Prédiction des scores de pertinence dans le cadre de l'appariement article-chercheur pour l'organisation de conférences scientifiques
Auteur: Author:	Etienne Six
Date:	2018
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Six, E. (2018). Prédiction des scores de pertinence dans le cadre de l'appariement article-chercheur pour l'organisation de conférences scientifiques [Master's thesis, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/3785/

Document en libre accès dans PolyPublie
Open Access document in PolyPublie

URL de PolyPublie: PolyPublie URL:	https://publications.polymtl.ca/3785/
Directeurs de recherche: Advisors:	Michel C. Desmarais
Programme: Program:	Génie informatique

Ce fichier a été téléchargé à partir de PolyPublie, le dépôt institutionnel de Polytechnique Montréal
This file has been downloaded from PolyPublie, the institutional repository of Polytechnique Montréal

https://publications.polymtl.ca

UNIVERSITÉ DE MONTRÉAL

PRÉDICTION DES SCORES DE PERTINENCE DANS LE CADRE DE L'APPARIEMENT ARTICLE-CHERCHEUR POUR L'ORGANISATION DE CONFÉRENCES SCIENTIFIQUES

ETIENNE SIX
DÉPARTEMENT DE GÉNIE INFORMATIQUE ET GÉNIE LOGICIEL
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION
DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES
(GÉNIE INFORMATIQUE)
DÉCEMBRE 2018

© Etienne Six, 2018.

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé :

PRÉDICTION DES SCORES DE PERTINENCE DANS LE CADRE DE L'APPARIEMENT ARTICLE-CHERCHEUR POUR L'ORGANISATION DE CONFÉRENCES SCIENTIFIQUES

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

M. GAGNON Michel, Ph. D., président
M. DESMARAIS Michel, Ph. D., membre et directeur de recherche
M. CHARLIN Laurent, Ph. D., membre

DÉDICACE

À tous ceux et celles qui m'ont aidé et encouragé,
Merci infiniment !

REMERCIEMENTS

J'aimerais remercier mon directeur de recherche Michel Desmarais pour son soutien à la fois technique et humain tout au long de ma maîtrise. Je voudrais aussi remercier tous mes collègues de laboratoire pour leurs nombreux conseils et remarques qui m'ont été très utiles. Finalement, j'aimerais remercier ma famille et mes amis, dont le soutien tout au long de ma maîtrise m'a beaucoup aidé.

RÉSUMÉ

La prédiction des scores de pertinences article-chercheur est un des sujets centraux de l'organisation de conférences scientifiques par le biais de méthodes d'apprentissage automatique. Cependant, les données disponibles pour effectuer ces prédictions sont très variées et les méthodes pour exploiter chaque type de données restent nombreuses.

Ce mémoire décrit nos expériences pour prédire les scores de pertinences article-chercheur par apprentissage supervisé en utilisant des résumés d'articles. Les données utilisées sont un ensemble de votes de pertinences pour des articles de NIPS 2006. Pour chaque vote, le résumé de l'article soumis ainsi que des résumés d'articles ayant été écrits par le votant sont utilisés pour prédire ledit vote.

L'originalité de cette recherche réside entre autres dans l'utilisation de réseaux de neurones pour effectuer cette prédiction, et ce malgré la faible taille du jeu de données disponible. Les résultats obtenus sont encourageants : notre modèle à base de réseaux de neurones convolutifs a des performances meilleures que les modèles que nous avons pris pour référence ; notre modèle à base de réseaux de neurones récurrents a quant à lui des performances similaires au meilleur modèle de référence.

ABSTRACT

Predicting paper-reviewer suitability scores is paramount to the organization of scientific conferences by using machine learning techniques. However, the available data to predict such scores is very varied and so is the exploitation of each type of data.

In this thesis, we describe our attempts at predicting the paper-reviewer suitability scores using supervised learning on articles abstracts. The data used consists of suitability votes for articles from NIPS 2006. For each vote, the abstract of the submitted article as well as some abstracts of articles from the voter are used to predict the said vote.

Part of what makes the research original is the fact that we managed to use neural networks to do the prediction, even though the dataset was rather small. The results are encouraging: our convolutional neural network model performs better than the models we took as references. As for the recurrent neural network model, it has similar performances to our best model of reference.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES SIGLES ET ABRÉVIATIONS

NIPS	Conference on Neural Information Processing Systems
RNN	Réseaux de neurones récurrents
CNN	Réseaux de neurones convolutionnels
PLNE	Problème linéaire avec des nombreux entiers
MSE	Erreur quadratique moyenne
ReLU	Unité de Rectification Linéaire
NLP	Traitement automatique du langage naturel
LDA	Allocation de Dirichlet latente

CHAPITRE 1 INTRODUCTION

Une étape fondamentale de toute conférence scientifique est l'évaluation des articles soumis à un panel d'experts pour décider s'ils sont pertinents. Les organisateurs de la conférence sont chargés d'effectuer l'attribution des articles aux évaluateurs membres du panel, de sorte que chaque article soit évalué par quelques scientifiques compétents. Depuis plusieurs années déjà, il est clair qu'organiser une conférence sans l'aide d'un outil informatique n'est plus possible. En effet, un tel outil permet d'organiser les articles, de centraliser les commentaires des évaluateurs et de transmettre les retours aux personnes ayant soumis leur article. Désormais, compte-tenu de la taille de certaines conférences ainsi que du nombre d'articles soumis, il semble en outre pertinent que l'outil aide les organisateurs dans la phase d'attribution des articles, qui peut se voir comme un appariement entre les articles soumis et les membres du panel.

1.1 Définitions et concepts de base

1.1.1 Score de pertinence

La notion de *score de pertinence* est un des principes de base utilisé pour effectuer l'appariement de manière automatisée. Chaque score de pertinence est relatif à un article et à un évaluateur. Ce score est censé exprimer la pertinence de l'expertise de l'évaluateur vis-à-vis du contenu de l'article. Un score de pertinence parfait implique que l'évaluateur maîtrise parfaitement toutes les notions nécessaires à la compréhension de l'article. A l'inverse, un score de pertinence faible indique que l'évaluateur a des lacunes concernant le contenu de l'article. Un tel score n'est évidemment pas connu a priori, mais à supposer qu'on arrive à l'obtenir, ou bien à l'estimer précisément, on obtient une manière simple d'attribuer les articles. On peut en effet définir un problème linéaire avec des nombres entiers (PLNE) :

$$J(x) = \sum_{e} \sum_{a} s_{ea} x_{ea} \tag{1.1}$$

Où $x_{ea} \in \{0, 1\}$ dénote l'attribution de l'article a à l'évaluateur e et s_{ea} est le score de pertinence liant l'article a à l'évaluateur e. La maximisation de J garantit le plus haut score et donc la meilleure répartition possible.
A l'évidence, ne sont pas pris en compte à ce stade des problèmes comme le nombre désiré d'évaluateurs par articles ou bien encore les contraintes pratiques sur le nombre d'articles

minimum et maximum qu'un évaluateur doit examiner. Ces contraintes peuvent s'exprimer sous forme d'équations pour compléter le PLNE :

$$\forall a \ : \ \sum_{e} x_{ea} = N_{évaluateurs} \tag{1.2}$$

$$\forall e \ : \ \sum_{a} x_{ea} > C_{min} \ et \ \sum_{a} x_{ea} < C_{max} \tag{1.3}$$

Où $N_{évaluateurs}$ indique le nombre d'évaluateurs imposé pour chaque article et C_{min} et C_{max} correspondent aux charges minimale et maximale de chaque évaluateur en termes de nombre d'articles.
Si la résolution de ce PLNE n'est directement traitée dans ce mémoire, son introduction permet de souligner le rôle fondamental du score de pertinence dans l'appariement des articles et des évaluateurs.

1.1.2 Réseaux de neurones

L'utilisation des *réseaux de neurones* est désormais extrêmement répandue dans de nombreuses tâches d'apprentissage automatique et leur utilisation s'accroit encore rapidement. Cette branche de l'apprentissage machine permet l'exploitation de données très complexes grâce à une capacité d'apprentissage arbitrairement grande, dépendante de l'architecture du réseau et de sa taille. Leur popularité et le développement de méthodes adaptées à l'analyse de textes en faisaient des candidats parfaits pour essayer de prédire les scores de pertinences grâce aux résumés des articles.

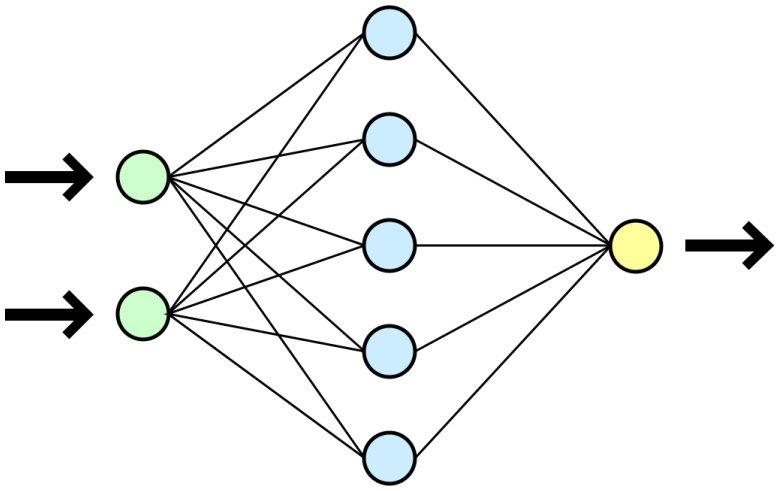

Figure 1.1 Exemple de réseau de neurones simple

Le Figure 1.1 illustre l'architecture basique d'un réseau de neurones. Un réseau de neurones possède une entrée (verte), une ou bien des couches cachées (bleue) et une sortie (jaune). Les couches d'entrée et de sortie sont définies par le type de données traitées ainsi que par le type de données attendues en sortie. Si l'objectif est de classifier des vecteurs de dimension 50, l'entrée sera de dimension correspondante. De même, la dimension de la couche de sortie dépendra de la dimension de la prédiction qui doit être réalisée. Cependant les couches cachées présentent une grande flexibilité et sont à l'origine de la grande capacité des réseaux de neurones. En effet, leur nombre est arbitraire, tout comme leur taille, ainsi que les connexions qu'elles ont entre elles et avec les couches d'entrée et sortie.
La propagation de l'information se fait par multiplication matricielle puis par application d'une non-linéarité entre chaque couche du réseau, aussi appelée fonction d'activation. Ces non-linéarités permettent aux réseaux de neurones de modéliser des fonctions arbitrairement complexes sans trop de difficulté, ce qui fait leur force. L'apprentissage des poids se fait par rétroprogation du gradient : cela permet, à partir de l'erreur sur la prédiction finale, de calculer le gradient de l'erreur pour chaque neurone et de le modifier en conséquence.
Il existe de nombreux types de réseaux de neurones qui dépendent de l'architecture des couches cachées. Ceux qui ont été utilisés dans le cadre de ce travail sont les RNN et les CNN. Leur rôle dans notre recherche ainsi que leur fonctionnement respectifs seront élaborés par la suite.

1.2 Éléments de la problématique

Comme nous l'avons évoqué précédemment, la prédiction du score de pertinence article-chercheur est un enjeu majeur dans l'organisation de conférence scientifique. Pour effectuer cette prédiction, l'utilisation de l'apprentissage automatique est un passage obligatoire. En effet, il est inenvisageable que les organisateurs tentent eux-mêmes de prédire ces scores, cela prendrait beaucoup trop de temps et irait donc à l'encontre de la raison même d'être de ces scores. Par ailleurs, il est possible que les évaluateurs puissent être consultés pour prédire ces scores. Malgré cela, il est en pratique difficile d'obtenir les scores pour tous les couples article-chercheur : l'apprentissage automatique est donc nécessaire.
Comme dans tout problème d'apprentissage machine, se posent plusieurs questions :

1. Quelles données sont disponibles pour ce problème ?
2. Quelle est la complexité des données (dimension) ?
3. Quel prix - en terme de temps - est-on prêt à payer pour augmenter la précision des prédictions ?

Les données disponibles ne sont pas très nombreuses. En effet, pour effectuer un apprentissage supervisé - ce qui nous intéresse ici - il faut avoir accès à des données potentiellement sensibles : des votes d'évaluateurs pour des articles de conférences. Ces votes représentent théoriquement une expertise réclamée de l'évaluateur, mais peuvent être utilisés comme des votes de préférence. C'est pourquoi, si certaines bases existent et sont étudiées dans la communauté scientifique, elles font parfois l'objet d'accords restrictifs et ne peuvent pas être ré-utilisées par tous. Par ailleurs, faire évaluer les scores de pertinence par des personnes extérieures est possible mais uniquement si ceux-ci sont des experts du domaine. Cela n'est donc pas possible à grande échelle en pratique. Cependant, il existe certaines bases ouvertes sur le sujet que nous pourrons utiliser.
Par ailleurs, la complexité des bases de données est importante : les votes à prédire sont de simples entiers mais les données disponibles pour le faire sont complexes. En plus des données de liens de citations, il s'agit en effet de données textuelles : les résumés des articles, desquels on peut inférer l'expertise des évaluateurs et donc prédire les votes. Ce type de données est parmi les plus complexes du fait de la richesse de la langue, de sa diversité et du fort potentiel d'ambiguïté. Néanmoins, il a récemment été montré que les réseaux de neurones permettent de traiter efficacement ce type de données et des progrès sont fait régulièrement dans le domaine.
Pour finir, nous supposons ici que la précision des prédictions est fondamentale. Il n'est pas souhaitable de choisir un modèle plus simple car il est plus rapide à s'exécuter : on cherche le modèle qui permet les meilleures prédictions en temps raisonnable. C'est pourquoi la durée d'entraînement potentiellement longue d'un réseau de neurones n'est pas problématique.
Nous posons donc l'hypothèse suivante :

Il est possible d'obtenir de bons résultats dans la prédiction de scores de pertinence article-chercheur, cela en utilisant un apprentissage machine basé sur les réseaux de neurones.

1.3 Objectifs de recherche

Nous avons défini une série d'objectifs à accomplir pour confirmer ou infirmer la validité de notre hypothèse de recherche.

1. Mettre au point un modèle à base de réseaux de neurones capable de répondre au problème ;
2. Trouver une base de données utilisable par ce modèle ;
3. Evaluer le modèle en utilisant le jeu de données ainsi que des métriques claires ;
4. Comparer les résultats à ceux obtenus grâce à des méthodes de références.

1.4 Plan du mémoire

Ce mémoire est divisé en cinq chapitres. Dans le premier chapitre, les concepts de base ainsi que nos objectifs de recherche sont présentés. Le deuxième chapitre présente une revue de littérature sur les domaines abordés. Dans le troisième chapitre est expliquée la méthodologie utilisée pour évaluer notre modèle. Le quatrième chapitre présente notre modèle à base de réseaux de neurones et ses particularités. Le cinquième et dernier chapitre expose la recherche des hyperparamètres de notre modèle, ainsi que l'ensemble des résultats obtenus lors de l'évaluation.

CHAPITRE 2 REVUE DE LITTÉRATURE

Notre revue de littérature sera centrée principalement sur deux points : la modélisation de l'expertise des évaluateurs et l'utilisation des réseaux de neurones. Dans un premier temps, nous allons donc voir les bases de l'appariement des évaluateurs et des articles. Ce champ de recherche existe depuis un certain nombre d'années mais l'application de l'apprentissage supervisé n'y est pratiqué à notre connaissance que depuis le début des années 1990 et encore plus depuis les années 2000. Les concepts clés concernant l'apprentissage supervisé dans ce domaine seront présentés ici. Cela inclut le type des données utilisées ainsi que les différentes techniques mises en oeuvre pour exploiter ces données. Par la suite, nous nous intéresserons à l'apprentissage supervisé grâce à des réseaux de neurones appliqués aux données textuelles. Ces techniques sont très récentes, les réseaux de neurones ayant d'abord explosé grâce à leur utilisation pour la classification d'images. C'est plus récemment que tout un panel de techniques adaptées au traitement de la langue naturelle ont été mises en place : les techniques spécifiques qui nous ont été utiles durant le projet seront exposées ici.

2.1 Modélisation de l'expertise et prédiction des scores de pertinences

Bien que notre utilisation des réseaux de neurones est à notre connaissance une technique encore peu explorée dans le champ de la modélisation de l'expertise et de l'appariement article-évaluateur, il nous a été utile de savoir quelles autres techniques ont été et sont utilisées actuellement. Avant cela, il est utile de préciser le type de bases de données sur lequel se fondent la plupart de ces articles. La partie la plus importante est sans aucun doute les scores de pertinences article-évaluateurs : pour un certain nombre de couples article-évaluateur, ceux-ci sont disponibles. Ceux-ci peuvent être issus directement des votes d'utilisateurs ou bien ils peuvent être une évaluation d'autres experts de la pertinence (Mimno and McCallum, 2007). Cela permet d'aborder le problème sous l'angle de l'apprentissage supervisé : l'objectif est de prédire ces scores de pertinences de la manière la plus précise possible. Nous revenons par la suite sur les méthodes exactes d'évaluation de la précision. Pour effectuer ces prédictions, il est possible d'utiliser les résumés des articles évalués, ainsi que des résumés d'articles écrits par les évaluateurs. Un couple "résumé de l'article évalué"-"corpus de résumés de l'évaluateur" est étiqueté par le score de pertinence connu. L'objectif se résume donc à trouver un modèle f tel que :

$$s_{ij} = f(a_i, c_j) + w_{ij}, \forall i, j \in I, J \tag{2.1}$$

où I et J sont les ensembles d'indices représentant respectivement les articles et les évaluateurs, s_{ij} est le score pour l'article i par l'évaluateur j, a_i représente le résumé de l'article i, c_j représente le corpus de l'évaluateur j et w_{ij} est un bruit centré (bruit blanc gaussien).

2.1.1 Extraction d'information et traitement du langage naturel

L'extraction automatique d'information des textes est donc un des points fondamentaux pour la modélisation de l'expertise. En effet, sans le traitement automatique du langage naturel (NLP) il ne serait pas possible de se baser sur le texte des articles, qui représente la plus grande source d'information pour notre problème. L'exploitation de liens de citations, que nous n'avons pas auparavant évoquée, est par ailleurs possible (Rodriguez et al., 2006) mais les meilleurs résultats actuels ont été, à notre connaissance, obtenus grâce au NLP. Il faut pour cela représenter la distribution sur les mots des articles et des évaluateurs. Cela peut se faire de manière traditionnelle avec une analyse fréquentielle des termes dans les articles évalués et dans les corpus des évaluateurs grâce à la métrique fréquence du terme, fréquence inverse de document (TF-IDF). Il est aussi possible d'élaborer un modèle de langage, par exemple grâce à l'utilisation de l'allocation de Dirichlet latente (LDA) (Wei and Croft, 2006; Mimno and McCallum, 2007; Blei et al., 2003). Il s'agit, dans tous les cas, d'obtenir une représentation vectorielle des articles et des auteurs, facilement interprétable et susceptible d'être exploitée pour calculer des métriques comme les scores de pertinence. Certains modèles sont plus complexes et bien qu'utilisant les mêmes méthodes, subdivisent les représentations des auteurs (en personnes) et des articles (en sujets) (Mimno and McCallum, 2007).

2.1.2 Autres méthodes de prédiction

Par ailleurs, il existe d'autres méthodes pour extraire de l'information des bases disponibles, directement depuis la distribution des votes et non plus grâce au texte. Il est possible d'effectuer une régression linéaire où chaque évaluateur se voit attribuer un jeu de paramètres (Charlin et al., 2011). Dans les cas où la base de données est suffisamment importante cette méthode est très efficace. Cependant, chaque évaluateur ayant une représentation indépendante, cela implique que chacun doit avoir un nombre de points de référence (votes) suffisamment important pour que cette représentation ait un bon potentiel de généralisation. Par ailleurs, dans ce cas, il peut aussi être pertinent d'utiliser du filtrage collaboratif (Salakhutdinov and Mnih, 2008; Charlin et al., 2011). On procède alors à la factorisation de la matrice des scores de pertinences, par exemple grâce à la factorisation probabilistique de matrice, ce qui permet d'obtenir des facteurs latents de très faibles dimensions. Il existe une version bayésienne de cette factorisation probabilistique, qui améliore encore les résultats, mais doit

calculée grâce à modèle de Monte-Carlo par chaînes de Markov. Les variables latentes doivent être inférées par échantillonage

2.1.3 Évaluation des prédictions

Il existe plusieurs manières d'évaluer les prédictions réalisées grâce aux méthodes mentionnées précédemment. L'évaluation de la pertinence d'un classement top-k en est une (Mimno and McCallum, 2007). On classe les évaluateurs par ordre de pertinence pour chaque article, c'est-à-dire par score de pertinence prédit décroissant. On se débarasse ensuite des doublons : certains modèles ont de multiples représentations des évaluateurs et le même évaluateur peut donc se retrouver plusieurs fois dans le classement. On peut ensuite calculer la proportion d'entre eux donc l'expertise est réellement pertinente, c'est-à-dire dont le score réel est supérieur à un seuil arbitrairement fixé.

$$S = \frac{1}{dim(I)} \sum_{i \in I} \left(\frac{1}{dim(top_i)} \sum_{j \in top_i} s_{ij}^b \right) \tag{2.2}$$

où I est l'ensemble des indices des articles évalués, top_i est l'ensemble des indices des k premiers évaluateurs recommandés pour l'article i et s_{ij}^b est le score de pertinence binaire pour l'article i et l'évaluateur j. Le score binaire est de 1 si le couple article-évaluateur est pertinent et de 0 sinon.
Cela donne un pourcentage de suggestion pertinente et donne une idée de l'efficacité de la méthode. Cette méthode a l'avantage d'être facilement interprétable car elle correspond exactement à la conception humaine du système : un bon modèle suggérera beaucoup d'évaluateurs pertinents. L'objectif est donc de maximiser ce score.
Par ailleurs, il est aussi possible d'utiliser la mesure de l'erreur quadratique moyenne (MSE) pour donner une idée de l'erreur par rapport à la réalité. Cette méthode est particulièrement utile en apprentissage supervisé car c'est une des erreurs "type".

$$J = \frac{1}{dim(S^{obs})} \sum_{s_{ea} \in S^{obs}} (\hat{s}_{ea} - s_{ea})^2 \tag{2.3}$$

où S^{obs} est l'ensemble des votes observés, $\hat{s}_{ea}$ est la prédiction pour l'article a et l'évaluateur e et s_{ea} est le score réel pour l'article a et l'évaluateur e.
L'objectif est bien entendu de minimiser ce score, qui correspond à une erreur. Cette métrique est moins claire à interpréter dans l'absolu : plusieurs modèles doivent donc être confrontés pour avoir une idée de leur performance relative. Il est à noter qu'un modèle entraîné avec pour objectif la minimisation du MSE peut être évalué d'après la première métrique.

2.2 Traitement du langage naturel grâce aux réseaux de neurones

L'introduction et le développement des réseaux de neurones a été une étape majeure dans le domaine de l'apprentissage automatique. Nous nous intéressons uniquement ici à leur application aux données textuelles, domaine où d'importantes avancées ont été faites ces dernières années. Nous nous intéresserons tout d'abord au rôle fondamental des représentations vectorielles des mots. Celles-ci permettent l'application des différentes techniques décrites par la suite en fournissant une représentation dense et nuancée des mots. Nous verrons ensuite plus en détail deux types de modèles de réseaux de neurones adaptés au NLP : les RNN et les CNN. Les premiers ont été conçus avec l'idée sous-jacente qu'ils pouvaient être appliqués aux données textuelles, les deuxièmes ont subi des modifications pour s'y adapter.

2.2.1 Représentation vectorielle des mots

Le domaine du traitement automatique de la langue naturelle a été un des domaines de recherche les plus complexes de la fin du siècle dernier en informatique. Beaucoup de recherches portaient, et portent toujours, sur la création d'ontologies. Celles-ci permettent de structurer l'information de manière compréhensible par un ordinateur. Les représentations de mots word2vec et GloVe ont aidé à révolutionner le domaine du NLP (Mikolov et al., 2013; Pennington et al., 2014). Ces représentations sont calculées de manière non supervisée, ce qui évite d'avoir à étiqueter des données manuellement. Leur calcul s'effectue sur d'énormes bases de données contenant du texte, telles que Wikipedia ou Twitter. De telles bases de données présentent l'avantage d'être disponibles facilement. L'idée générale est de prédire un mot en fonction de son contexte ou bien l'inverse. Le contexte est défini comme les mots qui entourent un mot donné, comme on peut le voir sur la Figure 2.1 : le contexte est en blanc encadré.

On essaye de prédire la probabilité de trouver un mot compte tenu du contexte. Formellement, l'objectif est de maximiser la fonction objectif :

$$J = \frac{1}{T} \sum_{t=1}^{T} \sum_{-c<j<c, j\neq 0} log(p(\omega_{t+j}|\omega_t)) \tag{2.4}$$

où c est la taille de la fenêtre de contexte, T correspond au nombre de mots et ω_t correspond au mot t.
Il s'agit de maximiser la log-probabilité de tout mot du contexte étant donné le mot central. Cette méthode s'oppose notamment à l'encodage one-hot et présente plusieurs avantages. Il permet l'obtention de vecteurs de taille arbitraire, généralement de dimension comprise entre

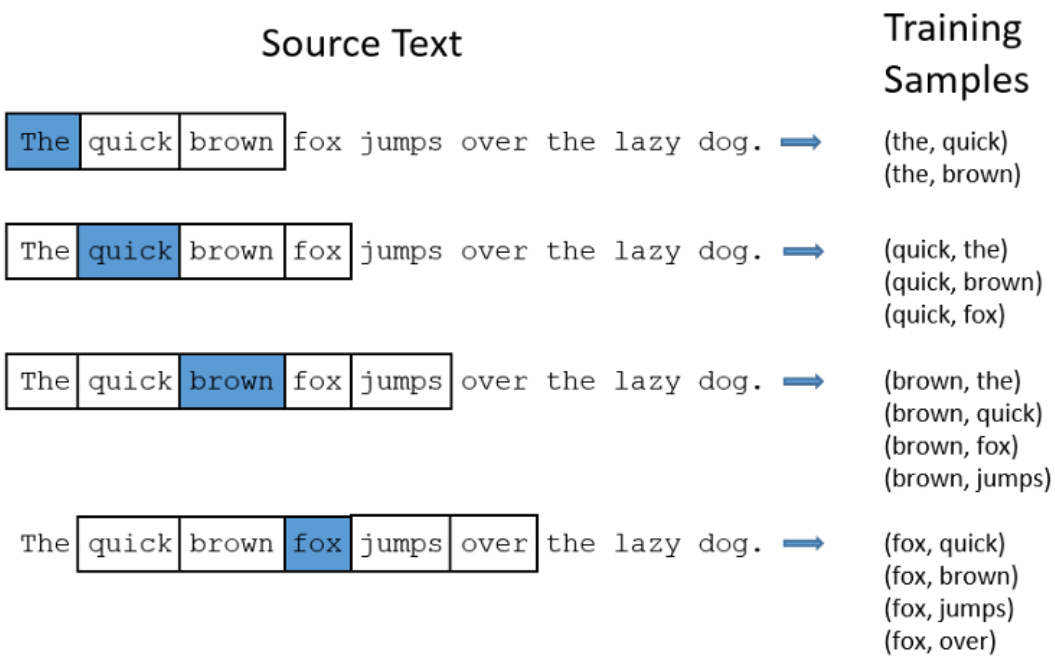

Figure 2.1 Relation linéaire entre les encodages appris

50 et 1000. Ces vecteurs sont très nuancés car il sont composés de nombres réels (compris entre 0 et 1 de par l'utilisation d'un softmax) et ne sont pas binaires. L'incorporation de nouveaux mots est facile puisqu'il suffit de réentrainer le modèle sur des textes contenant les mots en question. Il permet aussi en pratique l'utilisation d'un vocabulaire très large tant qu'une base de données suffisamment grosse est disponible pour l'entrainement et puisque la taille des vecteurs est indépendante de la taille du vocabulaire. Finalement, la similarité entre mots est présente dans la représentation même du mot et correspond à une proximité spatiale. Il existe des relations linéaires entre les différents mots appris grâce à word2vec.

La première partie de la Figure 2.2 illustre la relation existante entre les représentations des mots *man*, *woman*, *king* et *queen*. On notera leur représentation avec des crochets. Intuitivement, il est clair qu'il existe quasiment la même relation entre les mots *man* et *woman* ou *king* et *queen*. C'est le cas aussi avec leur représentation. On obtient :

$$[king] - [man] + [woman] \approx [queen] \tag{2.5}$$

Une relation similaire existe avec le temps d'un verbe :

$$[swimming] - [walking] + [walked] \approx [swam] \tag{2.6}$$

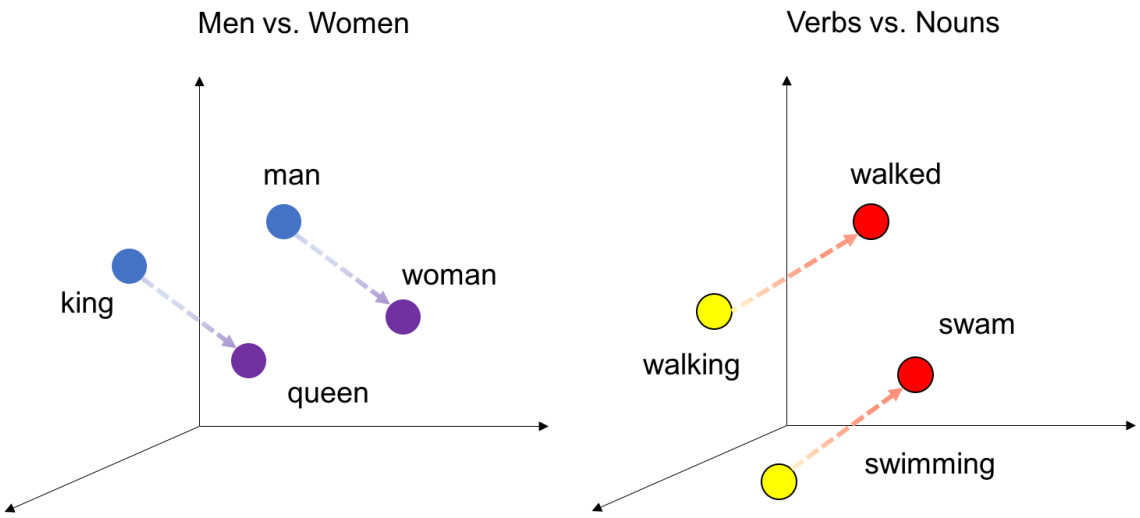

Figure 2.2 Relation linéaire entre les encodages appris

Un nombre important de relations linéaires similaires ont été identifiées comme entre les comparatifs et les superlatifs ou entre les pays et leur capitale.

En ce qui concerne GloVe (Pennington et al., 2014), le calcul est différent puisqu'il s'agit d'effectuer une réduction de dimension d'une matrice de co-occurence. L'intention et la construction derrière cette autre représentation diffèrent donc celle de word2vec. En pratique, les deux représentations ont cependant des performances très similaires.
Pour des projets tels que celui que nous avons mené, où la base de données est de taille modeste, ces représentations apportent une forte valeur ajoutée. En effet, une fois calculée sur un corpus important et généraliste, il est possible de réutiliser ces représentations vectorielles directement. Elles permettent en effet d'apporter des informations sur les mots qui n'auraient pas forcément pu être extraites dans ce corpus précis. Cependant, cela revient aussi à abandonner un peu de la spécificité du corpus. Le choix est donc moins évidemment lorsqu'on travaille avec un corpus plus important : la possibilité d'entrainer de nouvelles représentations devient alors pertinente.

2.2.2 Les réseaux de neurones récurrents

Maintenant que nous avons vu comment sont calculées les représentations vectorielles de mots, voyons quelles architectures nous permettent de les exploiter au mieux. Les RNN sont une architecture spécialisée de réseaux de neurones permettant de traiter des informations séquentielles. Un RNN applique un calcul de manière récursive à chaque étape d'une séquence d'entrée, conditionné par le précédent résultat calculé, appelé l'état (Elman, 1990). La Figure

2.3 en donne une représentation simplifiée.

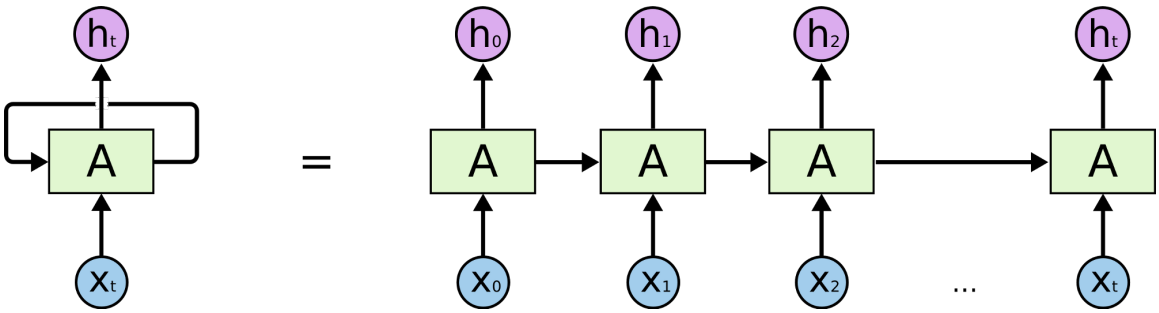

Figure 2.3 Réseau de neurones récurrent basique

La formule décrivant ce processus est (Goodfellow et al., 2016) :

$$\boldsymbol{h}^{(t)} = f(\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)}; \boldsymbol{\theta}) \tag{2.7}$$

où θ correpond aux poids du réseau. Plus précisément, on a :

$$\begin{aligned} \boldsymbol{a}^{(t)} &= \boldsymbol{b} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)} \\ \boldsymbol{h}^{(t)} &= tanh(\boldsymbol{a}^{(t)}) \\ \boldsymbol{o}^{(t)} &= \boldsymbol{c} + \boldsymbol{V}\boldsymbol{h}^{(t)} \\ \hat{\boldsymbol{y}}^{(t)} &= softmax(\boldsymbol{o}^{(t)}) \end{aligned} \tag{2.8}$$

où chaque $\hat{\boldsymbol{y}}^{(t)}$ est une sortie à l'étape t. La Figure 2.4 illustre ce fonctionnement. La fonction de coût pour une séquence $\boldsymbol{x}$ couplée à une séquence $\boldsymbol{y}$ serait donc la somme des erreurs par chaque étape temporelle. Par exemple pour le MSE.

$$L(\boldsymbol{x}) = \sum_t (\hat{\boldsymbol{y}}^{(t)} - \boldsymbol{y}^{(t)})^2 \tag{2.9}$$

Cependant, il est aussi possible de considérer seulement la sortie finale du réseau pour les tâches de classification ou bien de régression. Une séquence $\boldsymbol{x}$ est alors couplée avec un unique $\boldsymbol{y}^{final}$. C'est les cas pour l'analyse de sentiments par exemple, où le résultat attendu est binaire et concerne la séquence entière. Le calcul des gradients (et donc l'apprentissage) se fait par rétropropagation à travers le temps. Pour faire simple, on "déroule" le réseau de neurones comme dans la Figure 2.3 puis on applique l'algorithme classique de rétropropagation, mais avec des poids partagés entre chaque étape temporelle "déroulée".
La principale force d'un RNN est d'être capable d'enregistrer le résultat des précédents calculs et de l'utiliser pour le calcul actuel. Cela le rend idéal pour l'analyse de données séquentielles

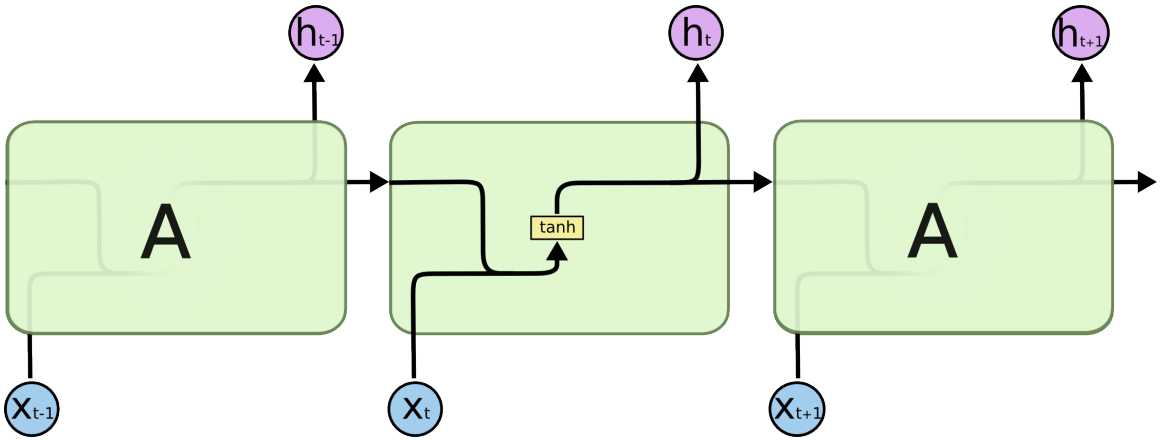

Figure 2.4 Réseau de neurones récurrent basique (2)

comme le texte, où chaque phrase est un ensemble ordonné de mots avec une forte dépendance. Les représentations vectorielles des mots évoquées précédemment sont d'ailleurs très utiles à ce moment-là : elles sont utilisées comme entrée à chaque étape de la séquence. Cependant, il existe des limitations techniques à la modélisation d'une longue séquence : la disparition de gradient. L'information cesse de se propager quand la profondeur du réseau devient trop grande : c'est le cas pour le RNN. Deux architectures de RNN ont été développées spécifiquement pour répondre à ce problème : les réseaux de neurones récurrents à mémoire court terme et long terme (LSTM) (Hochreiter and Schmidhuber, 1997) puis plus récemment les réseaux récurrents à portes (GRU) (Cho et al., 2014).

Les réseaux de neurones récurrents à mémoire court terme et long terme

Les LSTMs règlent le problème de la disparition du gradient en introduisant deux types de "mémoires", une mémoire à court terme et une mémoire à long terme qui n'est que partiellement modifiée à chaque étape de la séquence.

Les réseaux récurrents à portes

Les GRUs fonctionnent sur un principe similaire mais leur architecture a été simplifiée et optimisée. Cela leur permet d'apprendre plus vite et ils ont moins tendance à surapprendre que les LSTMs.

2.2.3 Les réseaux de neurones convolutionnels

A la structure séquentielle des RNN, s'oppose la structure hiérarchique des CNN. Ce type de réseaux de neurones ont eux aussi des performances très intéressantes sur les données textuelles, en particulier pour extraire des informations clés. (Kalchbrenner et al., 2014; Kim, 2014; Lopez and Kalita, 2017). Les CNN sont en fait plusieurs couches de convolutions successives reliées entre elles par des fonctions d'activation non-linéaires. Contrairement à un réseau complètement connecté traditionnelement, les liaisons sont locales : chaque région en entrée est connectée à un neurone de la couche suivante. Chaque couche applique différents filtres convolutionnels et combine leurs résultats.
Les poids des filtres convolutionnels sont appris par rétropropagation du gradient dans le réseau. Cette architecture à base de filtres de convolution permet aux CNN d'apprendre des caractéristiques de plus en plus haut niveau. Pour se représenter ce qui cela veut dire, il est plus facile d'envisager leur application à des images : la première couche détecte des côtés à partir des pixels bruts, puis la deuxième détecte des formes à partir des côtés et ainsi de suite. Un processus similaire s'effectue pour du texte mais est moins facile à visualiser. Au lieu d'une image, on utilise une matrice représentant le texte : chaque ligne de la matrice correspond à un mot, dans l'ordre de la phrase. Cette ligne est la représentation vectorielle évoquée précédemment. Pour une phrase de 20 mots avec une taille d'encodage de chaque mot de 50, le résultat est une matrice de taille 20x50. Contrairement à une image, le filtre englobe presque tout le temps les lignes entièrement, la convolution s'effectue donc seulement selon une direction. La taille de la fenêtre dans l'autre sens (le nombre de mots consécutifs considérés) est de l'ordre de 3 à 5.
En fonction du nombre de filtres à chaque couche du réseau, la dimension des données est susceptible d'augmenter. C'est pourquoi la technique du pooling a été introduite : il s'agit de réduire la taille de l'espace de redescription (*feature map*) tout en préservant l'information qu'elles contiennent. Le max-pooling global ne retient que la plus grande valeur de chaque carte de caractéristique par exemple.
Finalement, on utilise en général à la fin du processus un ou des réseaux complètement connectés pour obtenir le résultat final.

Dans la Figure 2.5, on voit un phrase analysée par un réseau convolutif. Il y a 4 filtres de largeurs variables. Chaque filtre est ensuite réduit à une valeur unique par max-pooling et finalement un couche complètement connectée finit le réseau.

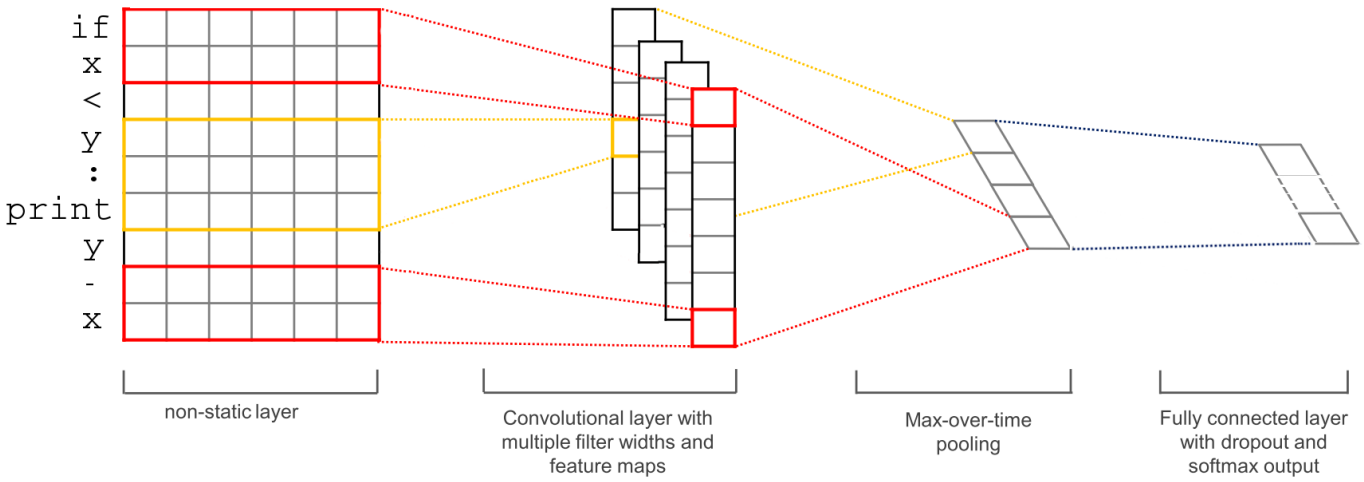

Figure 2.5 Réseau de neurones convolutif appliqué à du texte

2.2.4 Les techniques d'attention

L'utilisation de techniques d'attention permet de simplifier un problème sans pour autant perdre d'informations le concernant. Cette technique a d'abord été utilisée pour la traduction avec des réseaux de neurones récurrents (Bahdanau et al., 2014) : elle a ensuite été transposée dans de nombreux autres domaines, comme le traitement d'images. L'idée est que parfois, un réseau de neurones doit se fonder sur plusieurs informations pour prédire quelque chose : différents mots dans une phrase, différents fragments d'images, différents textes dans un corpus. Cependant, ces fragments d'informations apportent rarement tous la même quantité d'information. Par exemple, pour identifier un sport sur une photo, il est nettement plus utile d'avoir un fragment qui contient un surfeur plutôt qu'un bout d'image montrant la plage.

Figure 2.6 Mécanisme d'attention appliqué à une image

C'est pourquoi il est possible d'entraîner une partie du réseau pour déterminer la pertinence de chaque morceau d'informations à notre disposition. La Figure 2.6 illustre le concept : les pixels utilisés pour la prédiction du mot "surfboard sont clairs, les autres sont foncés.
Bien sûr, en pratique l'attention n'est pas un mécanisme binaire : on peut l'implémenter comme une pondération. Si on essaye de prédire le sentiment positif ou négatif d'un article à partir de plusieurs de ses paragraphes, on peut par exemple calculer une telle prédiction pour chaque paragraphe individuellement et calculer en parallèle le poids que chaque paragraphe aura dans la prédiction finale. Celle-ci correspondra donc à la somme des prédictions individuelles, pondérées par l'attention que l'on porte à chacune.

CHAPITRE 3 BASE DE DONNÉES ET MÉTHODOLOGIE D'ÉVALUATION

Nous allons examiner dans ce chapitre la base de données qui a été utilisée pour effectuer notre recherche. Après l'avoir présentée et avoir montré quelque unes de ses propriétés, nous allons nous intéresser à l'augmentation de données à laquelle nous avons procédé pour permettre à notre modèle de s'entraîner sur cette base de taille modeste. Finalement, notre méthodologie d'évaluation sera présentée, dont les métriques et les méthodes de prédiction de référence.

3.1 Base de données utilisée

La base de données que nous avons utilisée pour créer, entraîner et évaluer notre modèle est intitulée 'Data for Reviewer Matching, version 1.0'. Ces données ont été mises en forme intialement pour l'article *Expertise Modeling for Matching Papers with Reviewers* (Mimno and McCallum, 2007).
Cette base contient :

- Le titre et le résumés de 148 articles de NIPS 2006
- La liste des évaluateurs de NIPS 2006
- Une archive d'articles pour chaque évaluateur (15000 articles au total)
- 393 "scores" des évaluateurs pour les 148 articles sus-mentionnés

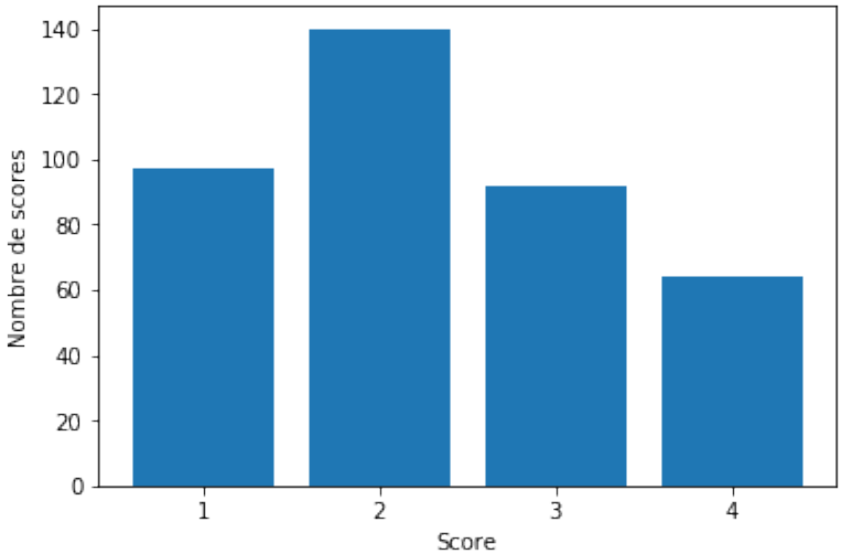

Figure 3.1 Distribution des scores

Ces 393 scores de pertinence article-évaluateur sont au fondement de notre recherche. En effet, notre but a été de les prédire de la meilleure manière possible. C'est pourquoi, nous avons procédé à quelques analyses statistiques de scores en question. En particulier, la Figure 3.1 indique la répartition des scores. On constate que ceux-ci sont relativement bien répartis avec malgré tout une préférence assez marquée pour le score de 2, correspondant à la capacité d'évaluer l'article sans pour autant être un expert. Le score le moins commun est logiquement 4, qui indique une expertise pour tous les domaines abordés par l'article.

Par ailleurs, il nous a été utile de nous intéresser aux résumés des articles et notamment leur nombre de mots. Nous avons analysé le nombre de mots des 148 articles évalués pour lesquels des scores de pertinence sont disponibles, mais aussi les quelque 15000 articles écrits par les évaluateurs.
La Figure 3.2 indique la répartition du nombre de mots pour les articles évalués. Le nombre de mots moyen est de 144 mots et la déviation standard est de 85 mots. Cela semble relativement normal puisque seuls les résumés des articles sont disponibles.

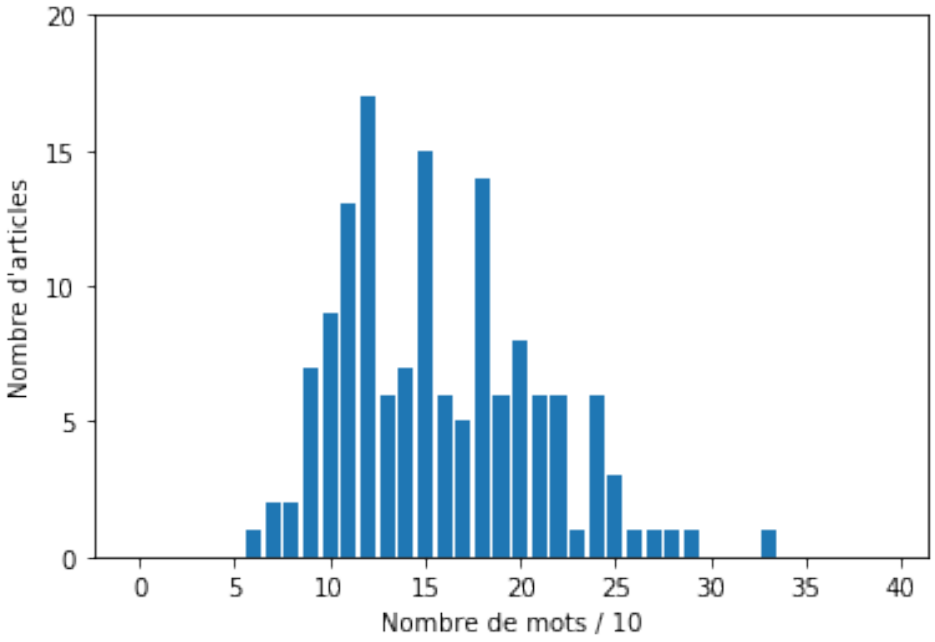

Figure 3.2 Répartition du nombre de mots pour les articles évalués

La répartition des mots des 15.000 articles écrits par les évaluateurs se trouve à la Figure 3.3. Le nombre moyen de mots est 88 et la déviation standard est de 143 mots. On comprend vite qu'il y a un problème en regardant la répartition. En réalité, pour un grand nombre d'article, seul le titre est disponible et pas le résumé en lui-même : cela explique le grand nombre d'articles dont le nombre de mots disponibles est inférieur à 50. La Figure 3.4 représente la

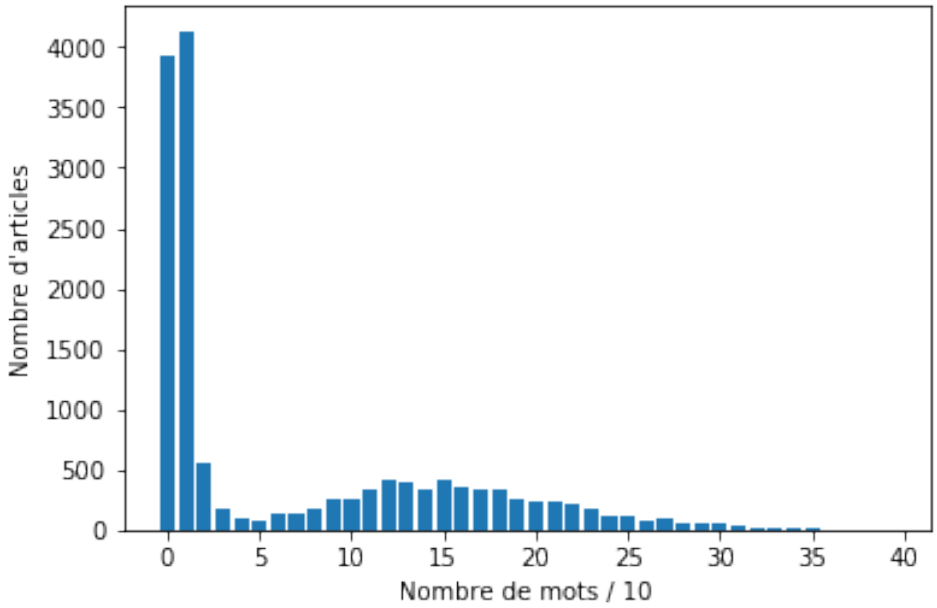

Figure 3.3 Répartition du nombre de mots pour les articles des évaluateurs (1)

répartition du nombre de mots lorsque l'on omet les articles en question : on constate une répartition proche de celle des articles évalués avec une moyenne de 174 mots mais un écart type plus important de 200 mots.

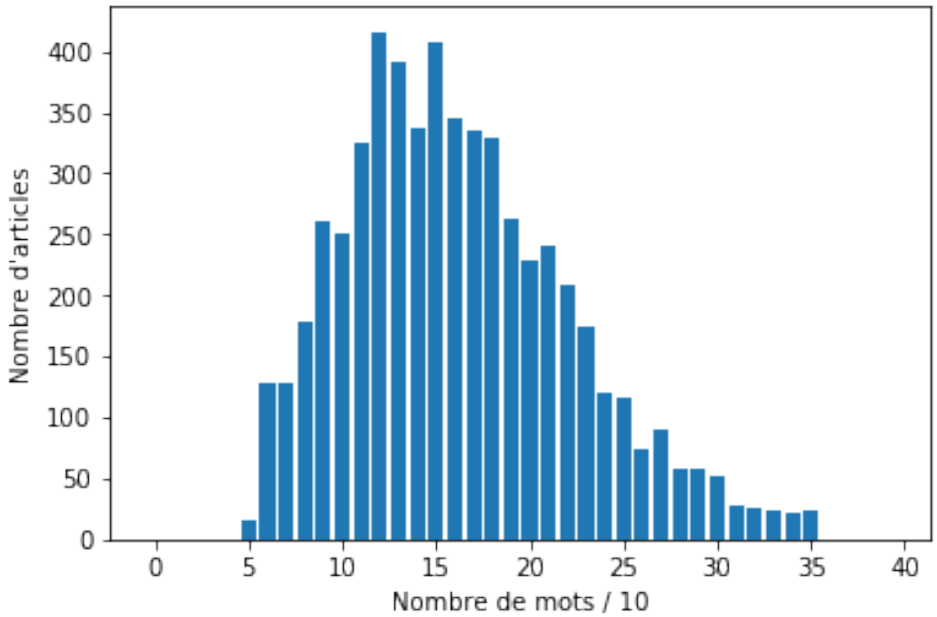

Figure 3.4 Répartition du nombre de mots pour les articles des évaluateurs (2)

3.2 Augmentation des données

3.2.1 Problème de surapprentissage

Initialement, nous avons essayé d'entraîner un modèle avec les 393 points de données disponibles. Pour cela, nous avons utilisé un modèle similaire au modèle retenu finalement (cf. Chapitre 4). Cependant, il a été nécessaire de réduire sa capacité, notamment en réduisant la dimension de l'encodage des articles. Cela a permis de lutter contre le surapprentissage, mais a grandement réduit les performances du modèle. Le meilleur modèle entraîné en terme d'erreur quadratique moyenne sur le jeu de validation avait un MSE de 0,921 sur le jeu de données de test, à peine meilleur que le score moyen. Même une augmentation marginale de la capacité du modèle, pour essayer d'améliorer l'apprentissage, engendrait un surapprentissage : dès les premières itérations le MSE de validation se remettait à augmenter, sans passer par un minimum intéressant.

3.2.2 Solution apportée

Comme nous venons de le voir, pour entraîner notre modèle, les 393 scores de pertinences n'étaient pas suffisants, c'est pourquoi nous avons procédé à une augmentation des données. Cette procédure est très répandue pour les images : on peut déformer légèrement une image, la tronquer, lui faire effectuer une translation ou lui ajouter un bruit de fond sans pour autant modifier l'étiquette qui lui est associée. Cela permet d'augmenter la taille de la base de données et donc d'éviter les problèmes de surapprentissage. Cela diminue par contre la qualité des données. Cependant pour des données textuelles une telle procédure est moins évidente car toutes les méthodes citées ne s'appliquent pas. C'est pourquoi nous avons dû faire des hypothèses pour continuer à travailler avec cette base de données. Nous avons supposé que :

- L'expertise d'un évaluateur peut être inférée à partir de seulement 3 de ses articles
- Des groupes d'articles choisis aléatoirement d'un même évaluateur peuvent être considérés comme indépendants
- Même les articles dont seuls le titre est connu peuvent nous être utiles

Ces hypothèses ont été retenues puisqu'en pratique cela nous a permis d'obtenir de bons scores comme nous le verrons dans le Chapitre 5.

Pour chaque score connu, associé à un évaluateur et un article évalué, nous avons créé de nouveaux points de données sur lesquels baser notre modèle. Les articles de chaque évaluateur ont été groupés par trois et pour chaque groupe, nous avons créé un point de données correspondant. Les nouvelles données n'associent pas un évaluateur et un article évalué à

un score de pertinence, mais un groupe de trois articles d'un évaluateur et un article évalué à un score de pertinence. Comme pour beaucoup d'évaluateurs, nous avions de nombreux articles, chacun des 393 points de données a été démultiplié. Cette augmentation de données est illustrée par les Figures 3.5 et 3.6 : on obtient 3 points de données à partir d'un seul évaluateur dont on a 9 articles. Chaque carré représente un article de l'évaluateur, le carré rouge est l'article évalué.

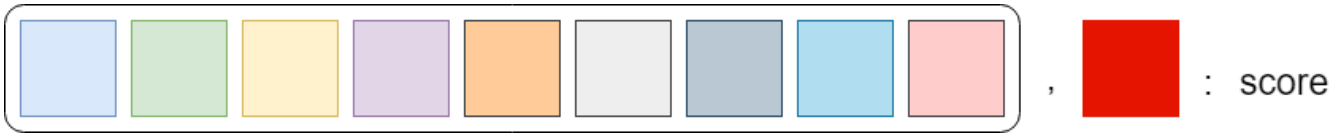

Figure 3.5 Point de données initial

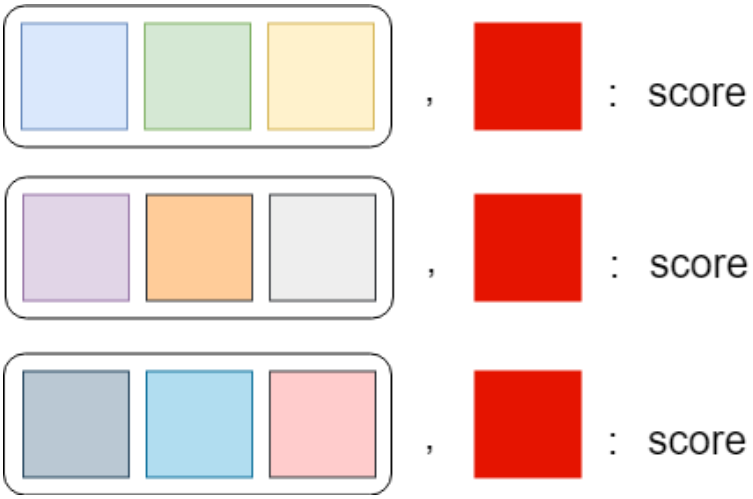

Figure 3.6 Nouveaux points de données après augmentation

La Figure 3.7 montre à quel point la répartition des scores est proche de la répartition initiale, ce qui montre que le jeu de données n'a pas été dénaturé dans l'opération. La tableau 3.1, qui compare la moyenne et l'écart-type des deux jeux de données, permet de confirmer cette impression. L'écart-type a diminué mais de manière très marginale et la moyenne est sensiblement la même.

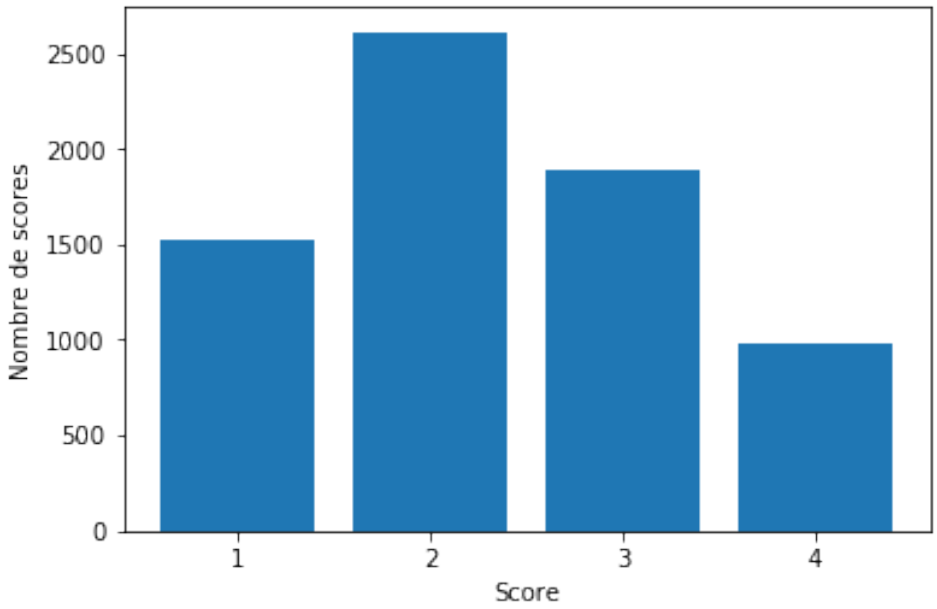

Figure 3.7 Répartition des scores après modification des données

Tableau 3.1 Comparaison entre les données initiales et augmentées

Données	Moyenne des scores	Ecart-type des scores
Initiales	2.31	1.02
Augmentées	2.33	0.97

3.3 Points de comparaison de notre modèle

Nous devons maintenant définir les algorithmes qui vont servir de points de comparaison pour évaluer la performance de notre modèle. Nous nous inspirons évidemment de ce qui est fait dans la littérature, mais notre augmentation de données nous empêche de comparer directement le résultat avec des résultats existants. Nous évaluerons nos algorithmes de référence sur le jeu de données augmentées et la métrique que nous utiliserons est le MSE. Nous en avons choisi 3 qui nous ont semblé pertinents.

3.3.1 Séparation du jeu de données

Le jeu de données a été séparé aléatoirement en un jeu de données d'entrainement, un jeu de données de validation et un jeu de données de test. Les jeux de données de validation et de test représentent chacun environ 15% des données totales. Le jeu de données de test est utilisé uniquement pour évaluer le modèle après que les hyperparamètres aient été choisis grâce au jeu de données de validation.

3.3.2 Prédiction du score moyen

Le premier point de comparaison est extrêmement simple puisqu'il s'agit de toujours prévoir le score moyen du jeu de données entier.

$$\begin{aligned} \hat{s} &= \frac{1}{dim(I)} \sum_{i \in I} s_i \\ MSE &= \frac{1}{dim(I)} \sum_{i \in I} (s_i - \hat{s})^2 \end{aligned} \tag{3.1}$$

où I est la liste des indices des points du jeu de données et $\boldsymbol{s_i}$ est le score pour le point $i \in I$.

Le MSE correspond donc à la variance de notre jeu de données : 0.936. Cela peut sembler trop simple mais en réalité, cela donne déjà une borne supérieure à l'erreur attendue et un ordre de grandeur de ce que l'on peut espérer obtenir. En effet, si le score d'un algorithme de prédiction est moins bon que cette référence, il ne présente aucun intérêt et n'apprend pas correctement sur le jeu de données. Par ailleurs, pour les données de "scores" comme c'est le cas ici, il existe en général une variabilité importante et donc il est rarement réaliste d'obtenir une erreur inférieure à la moitié de la variance, et encore.

3.3.3 Régression linéaire

Pour cette méthode et la suivante, il nous a fallu obtenir une représentation vectorielle des articles différente de notre encodage. Nous avons utilisé pour cela une représentation vectorielle de dimension 1000 obtenue grâce au TF-IDF et calculée grâce à la bibliothèque scikit-learn sous Python.

A notre connaissance, l'utilisation d'une régression linéaire se fait en général en attribuant un jeu de paramètres θ_e par évaluateur e, qui est appris, et à partir de la représentation ω_a d'un article a pour prédire le score $s_{ea} = \theta_e \cdot \omega_a$. C'est-à-dire que pour chaque évalutateur, on effectue une régression linéaire différente. Cependant, cette manière de faire n'est pas pratique dans notre cas, puisque seuls quelques scores sont disponibles pour chaque évalua-

teur : il serait donc impossible d'entraîner et d'évaluer ces modèles efficacement. De plus, chaque évaluateur se résume à une unique représentation θ_e donc le principe même de notre augmentation de données ne fonctionne pas. C'est pourquoi nous avons fait le contraire : chaque article évalué est représenté par un vecteur $\theta_{article}$, qui est appris, et les $\omega_{archive}$ sont les représentations de chaque archive de 3 résumés de la base de données augmentée. Les 3 résumés sont simplement concaténés pour calculer leur représentation. C'est-à-dire qu'on entraîne un modèle de régression linéaire différent pour chaque article évalué. Ce modèle est représenté par les poids $\theta_{article}$. On obtient donc :

$$\hat{s}_{article-archive} = \theta_{article} \cdot \omega_{archive} \tag{3.2}$$

On entraîne ce modèle avec pour objectif la minimisation du MSE.

$$MSE = \frac{1}{n} \sum_{article,archive} (s_{article-archive} - \hat{s}_{article-archive})^2 \tag{3.3}$$

où n est le nombre de points de données disponibles.
En pratique, une régularisation L2 a aussi été utilisée pour éviter le surapprentissage. Cette méthode améliore notablement notre point de comparaison : on obtient de MSE de 0.793.

3.3.4 Réseau de neurones avec mécanisme d'attention mais sans encodeur

On utilise cette dernière méthode de référence pour évaluer la valeur ajoutée qu'a notre encodeur. En effet, on utilise ici une architecture identique à celle qui est utilisée à la fin de notre modèle complet. Les représentations des articles évalués et de chaque article de l'archive (on les considère cette fois individuellement) sont concaténées pour donner un vecteur de taille 2000 et deux réseaux de neurones prédisent respectivement le score de pertinence et la pondération attribuée à ce score. Cela correspond exactement à ce qui est décrit au chapitre suivant et nous n'élaborerons donc pas plus ici. Le MSE optimal est de 0.836, ce qui est plus élevé qu'avec la régression linéaire : ce n'est pas étonnant vu que la taille des données est grande (dimension 1000) par rapport à leur quantité (environ 7000 échantillons). Les algorithmes plus simples sont en général plus adaptés donc ce cas. Cela justifie de vouloir trouver un moyen d'encoder les articles en plus petite dimension.

3.3.5 Méthode non supervisée

Nous avons essayé d'utiliser une méthode non supervisée fondée sur la similarité cosinus entre les articles. L'idée était de voir si la similarité cosinus entre les vecteurs TF-IDF représentant

l'article évalué et l'archive de l'évaluateur pouvait être utilisée pour prédire le score. Cependant, l'erreur quadratique moyenne (1,21) s'est avérée beaucoup trop élévée, même comparée à la méthode utilisant le score moyen : cette méthode n'a pas été retenue.

3.3.6 Résumé des méthodes de prédiction de référence

En résumé, nos points de comparaison sont donc :

Tableau 3.2 Comparaison de performances de nos points de comparaison

Méthode	MSE
Score moyen	0,936
Régression linéaire	0,793
Réseau de neurones	0,836

CHAPITRE 4 MODÈLE PROPOSÉ

4.1 Point de départ et objectif du modèle

Les données à notre disposition et que nous avons présentées au chapitre précédent sont constituées de :

- Une archive de 3 résumés pour chaque évaluateur
- Le résumé de l'article soumis
- Le score de pertinence pour cette paire article-archive

Notre objectif lors de la conception de ce modèle était la création d'un modèle prédictif supervisé *de bout en bout*. Tout le modèle est donc entraîné avec pour objectif la minimisation du MSE sur les scores de pertinences avec en entrée pour chaque score l'archive de l'évaluateur et le résumé de l'article considéré.

Il est important de souligner que les résumés correspondent tous au même format de données, mais ils auront des rôles différents dans le modèle. Nous nous attacherons donc à toujours préciser si un résumé est celui d'un article évalué ou bien fait partie d'une archive.

Par ailleurs, les hyperparamètres du modèle ne sont pas discutés ici mais dans le dernier chapitre : ceux mentionnés ici correspondent aux meilleurs pour chaque type de modèle CNN et RNN.

4.2 Encodage de chaque résumé

4.2.1 Description du problème

La première étape de notre modèle est la mise en place d'un encodeur pour transformer chaque résumé fait de texte brut en une représentation vectorielle exploitable par un réseau de neurones. De plus, nous cherchons également à diminuer la taille de cet encodage au maximum pour augmenter autant que possible la performance du modèle et son potentiel de généralisation.

Cette étape fait partie du modèle global est a donc été entraînée comme telle, avec pour objectif la minimisation du MSE sur les scores de pertinences. Son importance est cruciale car la taille des données est réduite de manière importante et seul un apprentissage efficace à cette étape a permis d'obtenir de bons résultats.

Deux méthodes ont été explorées pour effectuer cet encodage : un encodage à base de CNN et un à base de RNN.

4.2.2 Encodage à base de CNN

Nous nous sommes étendus sur le sujet dans le chapitre précédent, mais il est important de rappeler d'où viennent les dimensions des données de la Figure 4.1. Chaque résumé est :

— s'il fait plus de 300 mots, tronqué pour le ramener à 300 mots

— s'il fait moins de 300 mots, remplis par des mots 'vides'

Un vocabulaire est ensuite créé avec tous les mots des résumés : celui-ci est de taille 25038. Chaque mot est remplacé par un encodage one-hot. Cela nous donne la forme initiale des résumés : 300x25038.

La première étape de l'encodage est l'utilisation d'un encodage word2vec pré-entraîné sur 4 milliards de mots de Wikipédia, où chaque mot est représenté par un vecteur de dimension 50. Cela permet d'obtenir un encodage de taille 300x50 de chaque article. Cette première étape d'encodage est la seule qui n'est pas entraînée sur notre jeu de données. Le nombre de poids que cela représente est en effet trop important et les tentatives d'entraîner ces poids ne se sont pas montrées concluantes.

Ensuite, une convolution à 2 dimensions est utilisée pour ramener chaque filtre à une seule dimension. En effet, la taille du noyau de convolution est de 3x50 et il englobe donc chaque représentation vectorielle issue de word2vec complètement : la convolution se fait dans une seule direction (les mots) mais sur des données en deux dimensions. Comme nous l'avons vu dans la revue de littérature, c'est une condition indispensable à l'utilisation de CNN sur des données textuelles.

Après une étape de max-pooling et une nouvelle convolution les filtres ont une dimension de 144x1. Une dernière couche complètement connectée (dense) permet d'obtenir l'encodage final : un vecteur de taille 200.

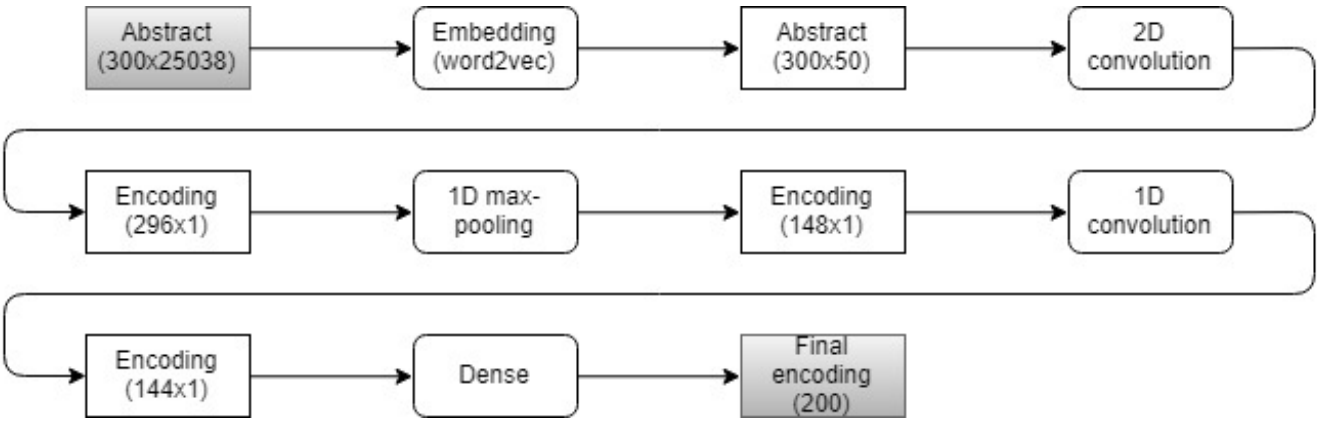

Figure 4.1 Encodage des résumés par CNN

4.2.3 Encodage à base de RNN

L'encodage grâce à un RNN fonctionne de manière similaire : après avoir utilisé word2vec, comme l'illustre la Figure 4.2, on considère chaque article comme une séquence de ses mots. Cette séquence est ensuite traitée par un RNN de type LSTM.

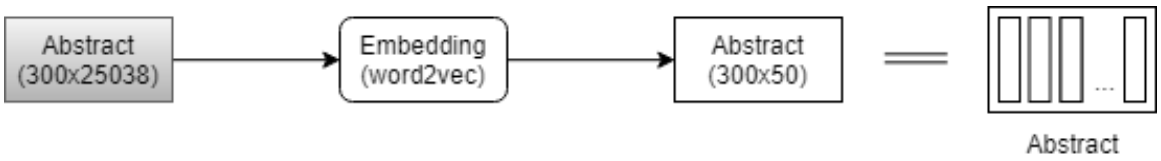

Figure 4.2 Encodage des résumés par RNN (1)

Les vecteurs de sorties du LSTM sont de taille 200, seul le dernier est récupéré afin de représenter l'encodage attendu. Cela est illustré dans la Figure 4.3 :

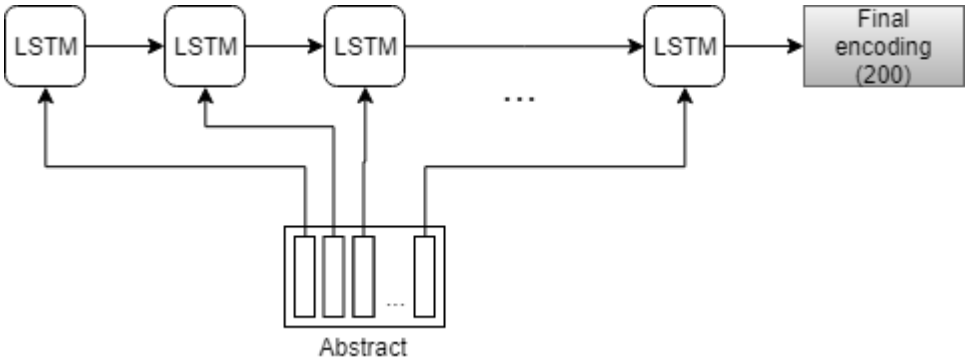

Figure 4.3 Encodage des résumés par RNN (2)

4.2.4 Utilisation de cette couche d'encodage dans le modèle

La première étape du modèle est d'utiliser cet encodeur sur tous les articles considérés.
Il est important de souligner que bien que chaque archive soit constituée de 3 articles, les poids de l'encodeur sont toujours partagés. Les différents articles de l'archive de l'évaluateur sont donc tous encodés de la même manière. L'article évalué est lui aussi encodé de la même façon. Cet aspect nous semble important bien qu'il ne soit pas à proprement parlé le principal résultat de notre recherche. En effet, cela montre que tous les articles ont un rôle équivalent dans notre architecture : l'encodage est donc plus général qu'un encodage pour article à évaluer ou bien pour article d'évaluateur. Cela ouvre la possibilité d'entraîner cet encodeur sur notre problème et de le réutiliser dans des contextes moins spécifiques, comme l'extraction

d'un vecteur d'expertise.
Dans la Figure 4.4, on peut visualiser cette étape. L'encodeur correspond soit à un CNN, soit à un RNN.

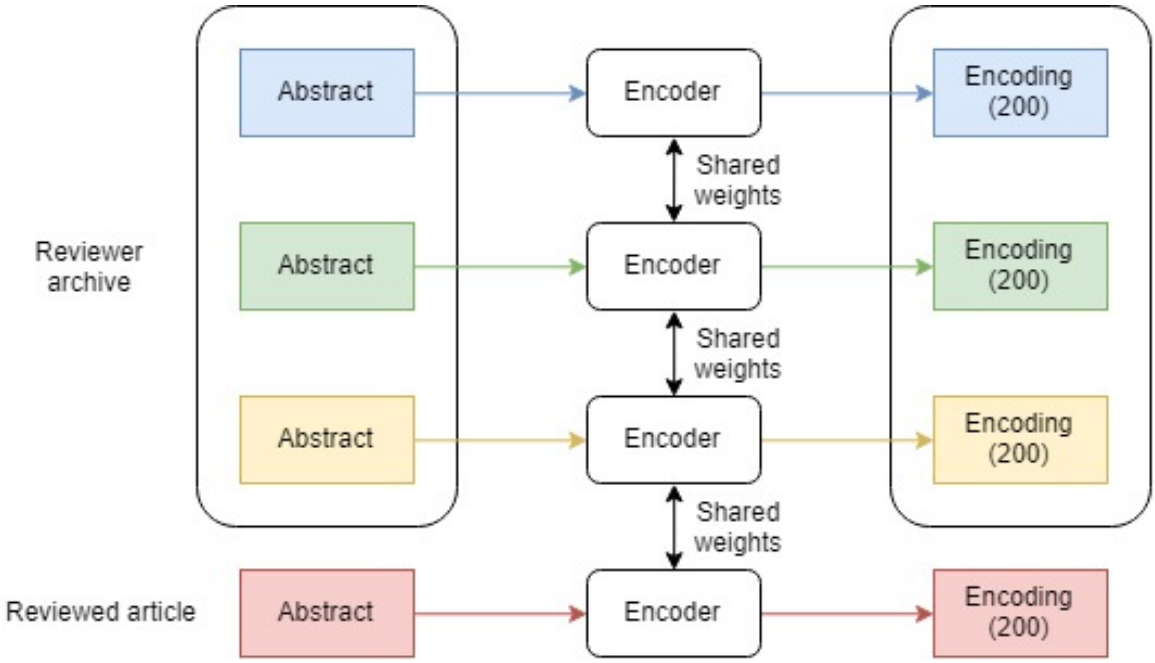

Figure 4.4 Encodage des articles d'une archive et d'un article soumis

Ces encodages peuvent désormais être utilisés pour effectuer la prédiction du score. Pour cela, chaque vecteur d'encodage de l'archive est concaténé avec le vecteur encodant l'article évalué, donnant un vecteur de taille 400 comme le montre le début de la Figure 4.5.

4.2.5 Mécanisme d'attention

Vient ensuite la partie prédictive à proprement parler de notre modèle et un problème se pose : comment définir l'importance de chaque article de la prédiction ? En effet, l'expertise des chercheurs est rarement strictement unithématique. Considérer tous les articles de manière équivalente est donc biaisé : un article peut être très pertinent pour évaluer le score de pertinence alors qu'un autre, portant sur une autre expertise du chercheur, ne ferait qu'ajouter du bruit à l'information existante. C'est particulièrement vrai avec l'augmentation de données : les articles ayant été choisis aléatoirement, certains peuvent ne pas être pertinents. Cela exclut donc la possibilité de simplement prédire le score pour chaque article et faire un moyenne. Une autre méthode commune consisterait à prendre le score maximal prédit mais de nouveau cette méthode s'avère d'une performance moindre à celle retenue : l'utilisation d'un mécanisme d'attention.

Ce mécanisme nous permet de régler notre problème. En plus de chaque prédiction de score va s'ajouter une prédiction de la pertinence, ou bien d'attention. Plus l'attention sera importante, plus la prédiction de score de pertinence de l'article aura du poids dans la prédiction finale.
La prédiction intermédiaire du score se fait par le biais d'une couche de neurones entièrement connectée. Celle-ci prend en entrée la concaténation des deux encodages et donne en sortie une valeur unique : la prédiction. La fonction d'activation de cette couche est une unité de rectification linéaire (ReLU). Le calcul de l'attention se fait exactement de la même manière, mais les poids utilisés pour le calcul de la prédiction et ceux utilisés pour le calcul de l'attention sont complètement indépendants.

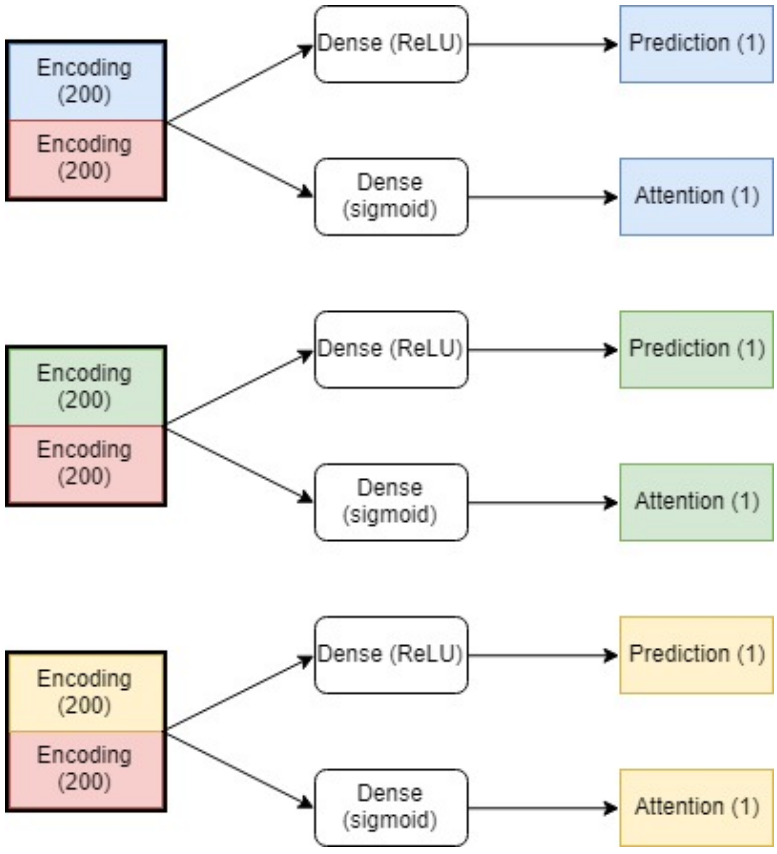

Figure 4.5 Calcul des prédictions individuelles et de l'attention

On normalise ensuite les attentions à une somme unitaire, grâce à une fonction softmax. Chaque score d'attention peut alors s'interpréter comme une pondération de l'importance de chaque prédiction. Un simple produit scalaire permet d'obtenir notre résultat final.

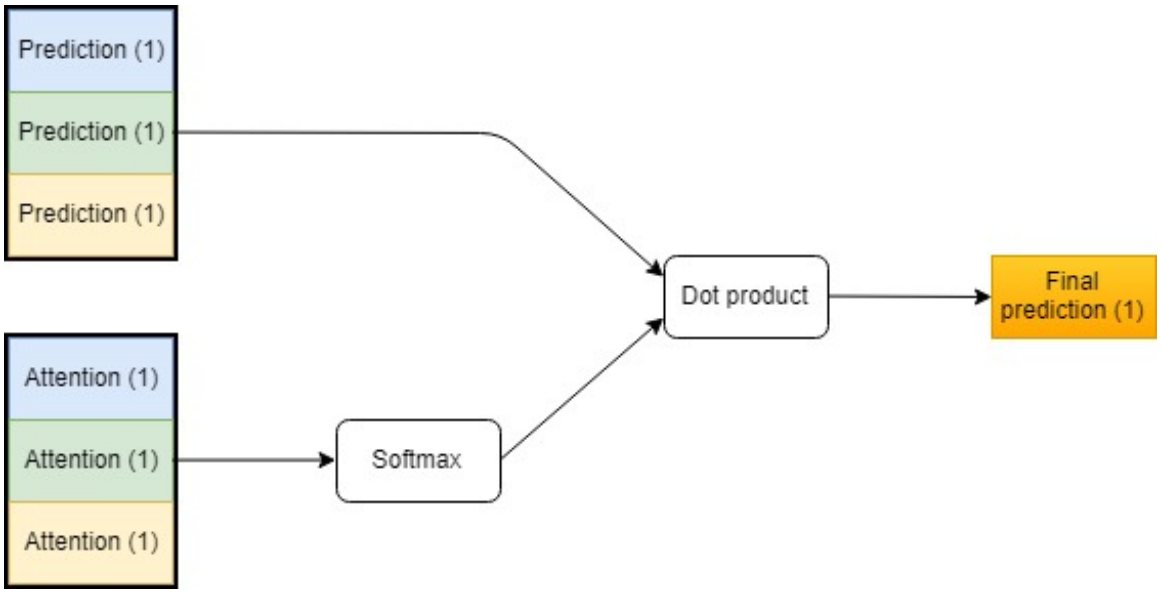

Figure 4.6 Calcul de la prédiction finale grâce à l'attention

4.3 Résumé du modèle utilisé

Notre modèle peut donc être résumé par les équations suivantes. Une archive et un article donnés sont indicés par $i \in I$.

$$\begin{aligned} \boldsymbol{E}^{(i)} &= f(\boldsymbol{article\_evalué}, \boldsymbol{W_1}) \\ \boldsymbol{E_1}^{(i)} &= f(\boldsymbol{article\_archive\_1}, \boldsymbol{W_1}) \\ \boldsymbol{E_2}^{(i)} &= f(\boldsymbol{article\_archive\_2}, \boldsymbol{W_1}) \\ \boldsymbol{E_3}^{(i)} &= f(\boldsymbol{article\_archive\_3}, \boldsymbol{W_1}) \end{aligned} \tag{4.1}$$

où $\boldsymbol{E}^{(i)}$, $\boldsymbol{E_1}^{(i)}$, $\boldsymbol{E_2}^{(i)}$ et $\boldsymbol{E_3}^{(i)}$ sont les encodages vectoriels respectivement de l'article évalué et de chacun des trois articles de l'archive, f représente l'architecture de l'encodeur choisi et $\boldsymbol{W_1}$ représente les poids appris de l'encodeur.
Ensuite, les prédictions et le mécanisme d'attention peuvent être décrits par l'équation qui suit.

$$
\begin{aligned}
\hat{\boldsymbol{s}}_{\mathbf{1}}^{(i)} &= g(\boldsymbol{E}^{(i)}, \boldsymbol{E}_{\mathbf{1}}^{(i)}, \boldsymbol{W}_{\mathbf{2}}) \\
\hat{\boldsymbol{s}}_{\mathbf{2}}^{(i)} &= g(\boldsymbol{E}^{(i)}, \boldsymbol{E}_{\mathbf{2}}^{(i)}, \boldsymbol{W}_{\mathbf{2}}) \\
\hat{\boldsymbol{s}}_{\mathbf{3}}^{(i)} &= g(\boldsymbol{E}^{(i)}, \boldsymbol{E}_{\mathbf{3}}^{(i)}, \boldsymbol{W}_{\mathbf{2}}) \\
\boldsymbol{a}_{\mathbf{1}}^{(i)} &= h(\boldsymbol{E}^{(i)}, \boldsymbol{E}_{\mathbf{1}}^{(i)}, \boldsymbol{W}_{\mathbf{3}}) \\
\boldsymbol{a}_{\mathbf{2}}^{(i)} &= h(\boldsymbol{E}^{(i)}, \boldsymbol{E}_{\mathbf{2}}^{(i)}, \boldsymbol{W}_{\mathbf{3}}) \\
\boldsymbol{a}_{\mathbf{3}}^{(i)} &= h(\boldsymbol{E}^{(i)}, \boldsymbol{E}_{\mathbf{3}}^{(i)}, \boldsymbol{W}_{\mathbf{3}})
\end{aligned}
\tag{4.2}
$$

où $\hat{\boldsymbol{s}}_{\mathbf{1}}^{(i)}$, $\hat{\boldsymbol{s}}_{\mathbf{2}}^{(i)}$ et $\hat{\boldsymbol{s}}_{\mathbf{3}}^{(i)}$ sont les scores individuellement prédits, $\boldsymbol{a}_{\mathbf{1}}^{(i)}$, $\boldsymbol{a}_{\mathbf{2}}^{(i)}$ et $\boldsymbol{a}_{\mathbf{3}}^{(i)}$ sont les pondérations pour chacun de ces scores, g et h représentent les réseaux de neurones utilisés pour prédire respectivement les scores et la pondération et $\boldsymbol{W}_{\mathbf{2}}$ et $\boldsymbol{W}_{\mathbf{3}}$ représentent les poids appris de ces réseaux de neurones.
La prédiction finale est :

$$
\hat{\boldsymbol{s}}^{(i)} = \hat{\boldsymbol{s}}_{\mathbf{1}}^{(i)} * \boldsymbol{a}_{\mathbf{1}}^{(i)} + \hat{\boldsymbol{s}}_{\mathbf{2}}^{(i)} * \boldsymbol{a}_{\mathbf{2}}^{(i)} + \hat{\boldsymbol{s}}_{\mathbf{3}}^{(i)} * \boldsymbol{a}_{\mathbf{3}}^{(i)} \tag{4.3}
$$

L'objectif que l'on cherche à minimiser et qui sert à entraîner l'ensemble des réseaux de neurones utilisés est :

$$
\boldsymbol{J} = \frac{1}{n} \sum_{i \in I} (\hat{\boldsymbol{s}}^{(i)} - \boldsymbol{s}^{(i)})^2 \tag{4.4}
$$

où $n = dim(I)$.

CHAPITRE 5 RÉSULTATS DE L'ÉVALUATION ET INTERPRÉTATION

Dans ce chapitre, nous allons présenter les résultats obtenus grâce à notre modèle. En particulier, nous allons nous intéresser à l'étude de nombreux hyperparamètres du modèle et leur influence sur sa capacé à apprendre, la capacité du modèle à généraliser ou encore la vitesse d'apprentissage. Sauf précision, c'est le modèle à base de CNN qui est utilisé. Les termes *loss* et *val_loss* utilisés dans les graphiques correspondent respectivement à l'erreur sur le jeu de données d'entraînement et à l'erreur sur le jeu de données de validation.

5.1 Taille de l'encodage

La performance du réseau de neurones avec mécanisme d'attention mais sans encodeur, utilisé comme modèle de référence et présenté au Chapitre 3, nous a donné des signes de l'importance qu'aurait la taille de l'encodage dans la performance finale du modèle. En effet, il est clair que plus l'encodage est de dimension importante, plus l'information que l'on garde sur les articles encodés est pertinente et précise. Cependant, cette dimension est à double tranchant : plus elle augmente, plus elle rend l'apprentissage de la dernière partie du réseau, entièrement connectée, difficile et sensible au sur-apprentissage. Par ailleurs, si elle est trop petite, elle ne permet pas un apprentissage d'aussi bonne qualité.
Pour illuster l'impact de la taille de l'encodage, voici quelques figures qui montrent les courbes d'apprentissage avec un nombre croissant de dimensions pour l'encodage.

L'erreur d'entraînement, en bleu, est parfois plus élevée que l'erreur de validation, en orange, à cause de l'utilisation de dropout pour l'entraînement. Un certain nombre de poids du réseau sont aléatoirement ignorés : cela favorise la généralisation, mais diminue évidemment la performance perçue du réseau. Le dropout n'est plus appliqué pour le calcul de l'erreur de validation, d'où un perfomance supérieure initialement.
Lors de l'optimisation des hyperparamètres, il est ressorti que les meilleures performances étaient atteintes avec un encodage de dimension 200, d'où la présence de la Figure 5.2 à titre de référence. Sur la Figure 5.1, on constate qu'avec un nombre si faible de dimensions, un plateau de performance est rapidement atteint (la courbe est ici tronquée mais a atteint son minimum) : la capacité du modèle est trop faible. Sur la Figure 5.3, on constate qu'avec 1000 dimensions, les performances sur le jeu de données d'entraînement deviennent rapidement excellentes. Cependant, la courbe de pertes de validation commence rapidement à augmenter de nouveau : cela indique qu'il y a surapprentissage et qu'un tel modèle aurait une très mauvaise capacité de généralisation.

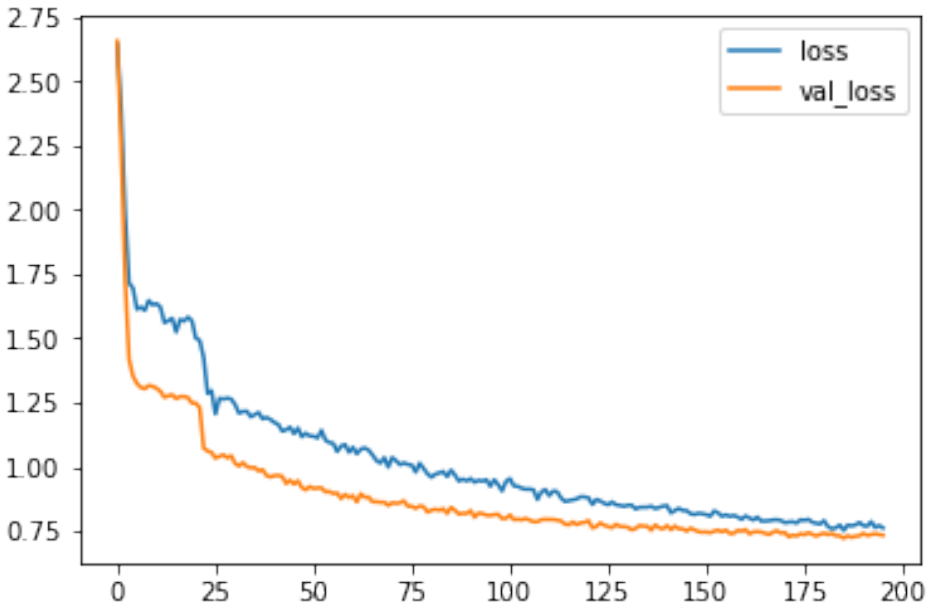

Figure 5.1 Encodage de dimension 10

Cette recherche d'équilibre entre capacité suffisante et risque de surapprentissage est une des clés de ce travail.

5.2 Utilisation du dropout

Comment nous l'avons évoqué précédemment, l'utilisation du dropout permet d'éviter le surapprentissage. Cependant un dropout approprié doit être trouvé pour éviter de perdre trop d'information. Dans notre architecture, nous utilisons du dropout deux fois : avant la couche dense de l'encodeur (CNN) et avant le calcul des prédictions et de l'attention.
Dans le cas où celui-ci est très élevé (90%) comme dans la Figure 5.4, nous pouvons constater que l'apprentissage n'est pas très bon : cela revient à baisser la capacité du modèle.
Dans le cas contraire, où nous le supprimons carrément, le surapprentissage est évident comme le montre la Figure 5.5.
Le dropout que nous avons donc choisi est de 50% : c'est celui qui donnait les meilleurs résultats. Un dropout plus faible, de l'ordre de 25%, n'avait quasiment aucun impact sur l'apprentissage. D'un autre côté, augmenter le dropout au-delà de 50% créait un phénomène de sous-apprentissage.

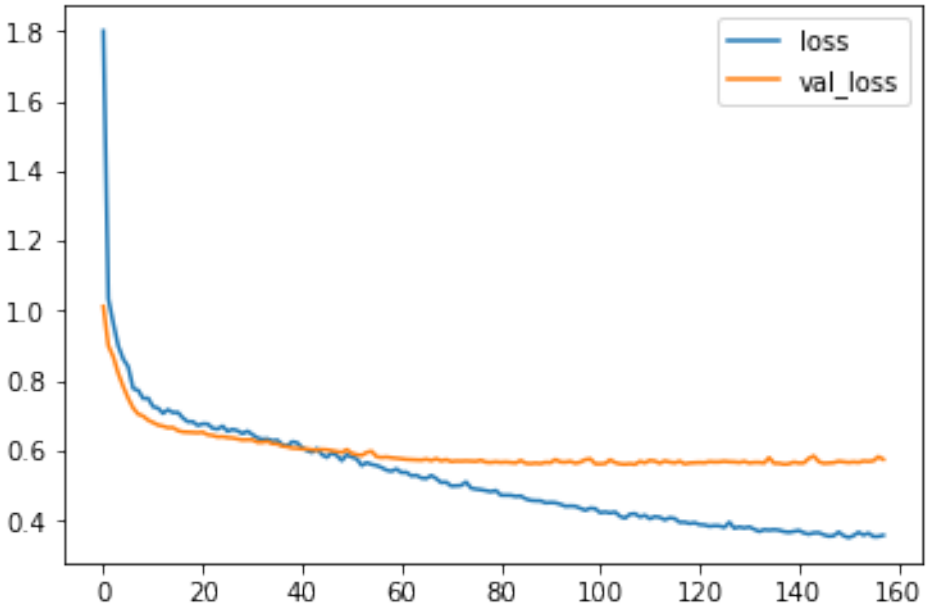

Figure 5.2 Encodage de dimension 200

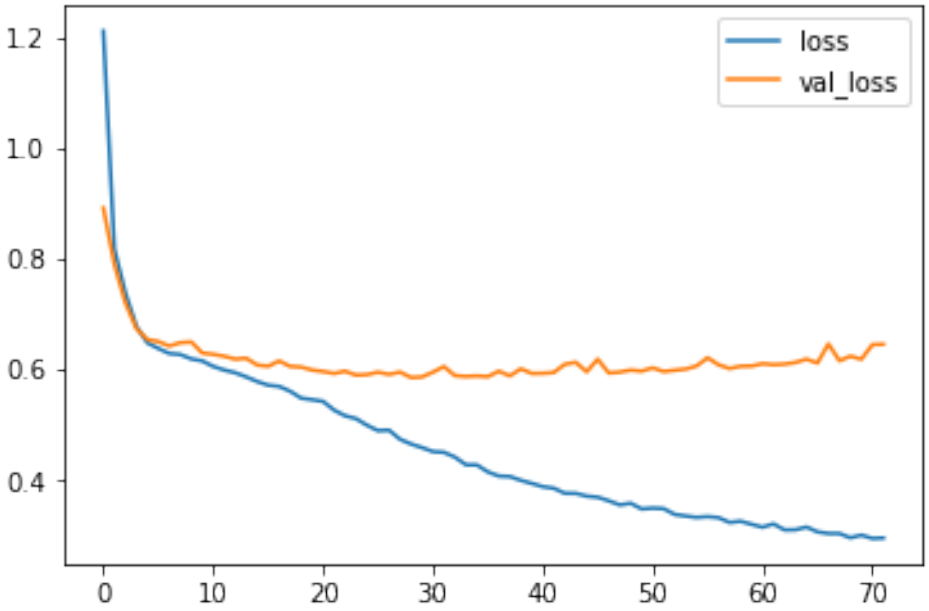

Figure 5.3 Encodage de dimension 1000

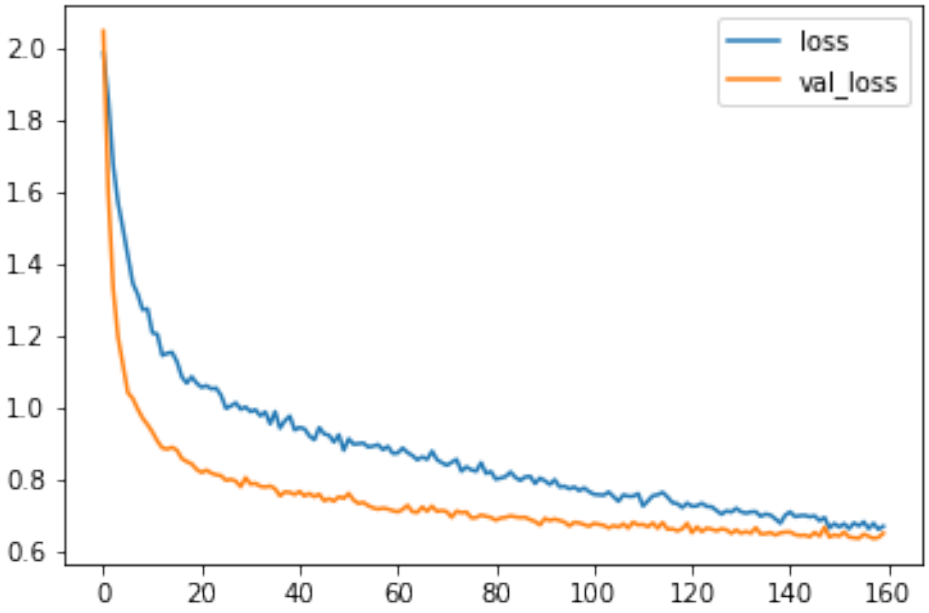

Figure 5.4 Dropout de 90%

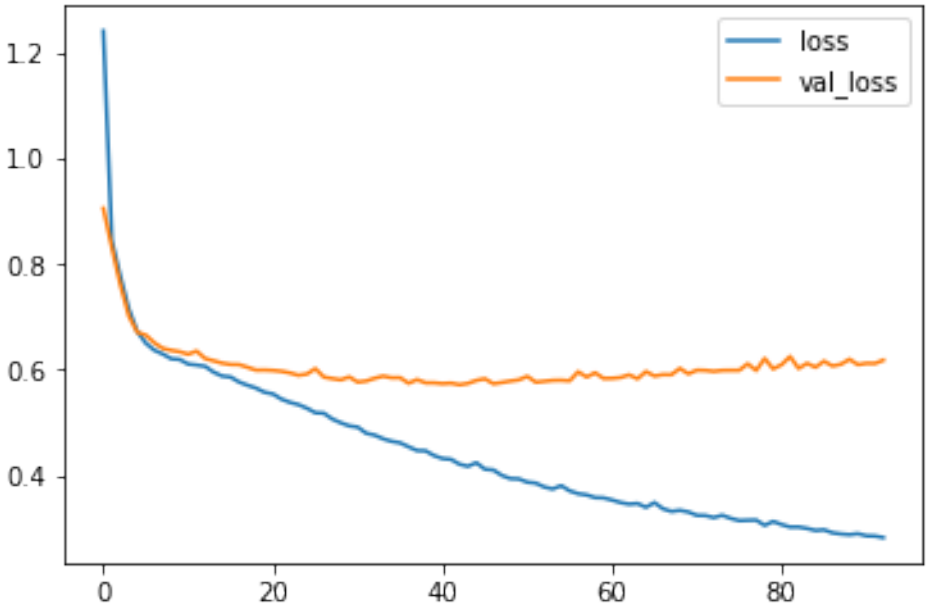

Figure 5.5 Pas de dropout

5.3 Embedding GloVe

Les embeddings GloVe que nous avons utilisés ont été entraîné sur un corpus de 6 milliards d'entités lexicales et le vocabulaire qu'ils couvrent est de plus de 400000 expressions. Une des premières questions que nous nous sommes posée est de savoir si ces embeddings entraînés sur un corpus non spécialisé seraient de suffisamment bonne qualité pour notre problème. En particulier, est-ce que tous les mots spécifiques au domaine existent bien dans le vocabulaire. Si ce n'était pas le cas, on perdrait des informations très importantes. Ces mots spécifiques sont justement ceux qui sont susceptibles de faire de distinction subtile entre les évaluateurs. Après étude de ce problème, nous en avons déduit que le vocabulaire était suffisamment riche pour couvrir tous les mots de notre base de données. En effet, les mots non reconnus par l'embedding étaient pratiquement tous des mots avec des fautes d'orthographe. Comme nous avons voulu modifier aussi peu que possible la base de données, nous n'avons pas procédé à une correction orthographique et les embeddings de ces mots ont été fixés au vecteur nul : cela signifie en pratique que ces mots sont totalement ignorés.

5.3.1 Taille de l'embedding utilisé

Les emdeddings étaient disponibles en plusieurs dimensions (50, 100, 200 et 300), la taille du vocabulaire et le corpus d'origine étant bien sûr les mêmes. Voici les résultats obtenus pour les meilleures architecture de CNN pour chaque taille.

Tableau 5.1 Meilleurs résultats obtenus en fonction de la taille de l'embedding

Taille de l'embedding GloVe	MSE validation
50	0.598
100	0.619
200	0.634
300	0.631

Evidemment pour chaque taille d'embedding, il a fallu optimiser le reste du réseau de neurones pour s'adapter. Le MSE a été testé sur le jeu de données de validation. Comme le montre le Tableau 5.1, les embeddings de dimension 50 ont donné le meilleur résultat. Les autres donnent un MSE similaire mais malgré tout légèrement plus élevé.

5.3.2 Entraînement de l'embedding

Par ailleurs, en plus du choix de la taille des embeddings GloVe, il est possible d'entraîner les embeddings à partir d'un certain temps. L'idée est que les embeddings peuvent être utilisés

tels quels au début pour permettre d'entraîner le reste du réseau de neurones. Après quelques itérations, il est possible d'entraîner à leur tour les embeddings pour les améliorer pour notre tâche. Cette technique semblait prometteuse et ne posait pas de problème d'implémentation mais malheureusement il y existait un autre problème de taille. Le nombre de poids que représente la couche d'embeddings est énorme : dès qu'on leur permet d'être modifiés, on constate un décrochage dans la courbe d'erreur. L'erreur sur le jeu de données d'entraînement chute immédiatement, alors que celle sur le jeu de validation augmente légèrement. Encore une fois, le surapprentissage est notre limitation première.

5.4 Taux d'apprentissage

Un autre hyperparamètre important est le taux d'apprentissage du modèle. Le taux d'apprentissage correspond à la vitesse à laquelle les poids du modèle sont mis à jour lors de la rétropropagation. Un taux d'apprentissage élevé permet donc au modèle d'apprendre plus rapidement. Cependant, comme le montre la Figure 5.6 un taux d'apprentissage élevé crée une instabilité dans l'apprentissage : le minimum local ou global est "raté" car les paramètres sont trop modifiés.

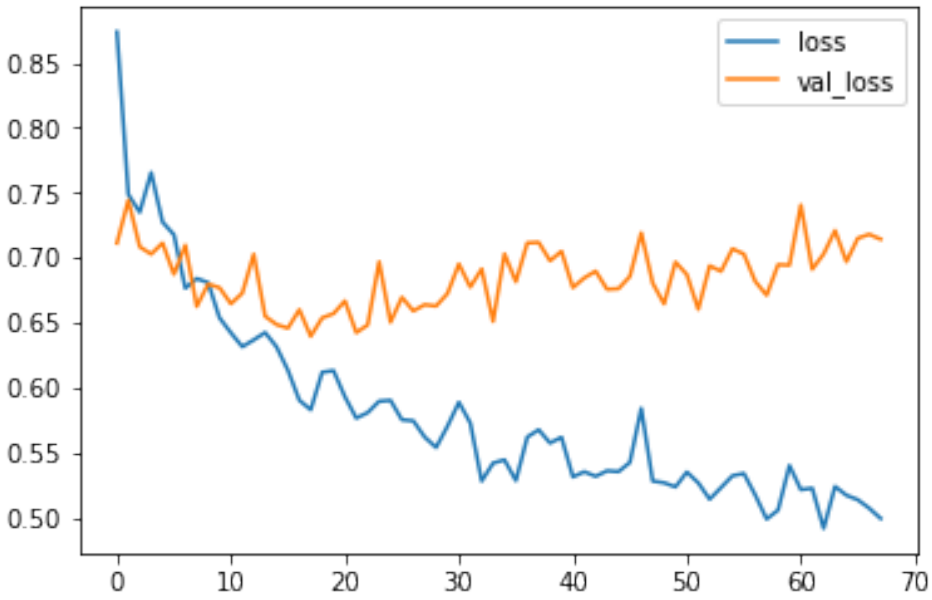

Figure 5.6 Taux d'apprentissage élevé (0.01)

D'un autre côté, un taux d'apprentissage plus faible garantit des courbes plus lisses et stables, comme illustré dans la Figure 5.7 mais sera beaucoup plus lent. Pire, si le modèle se retrouve dans un mauvais minimum local, il est possible qu'il se retrouve bloqué.

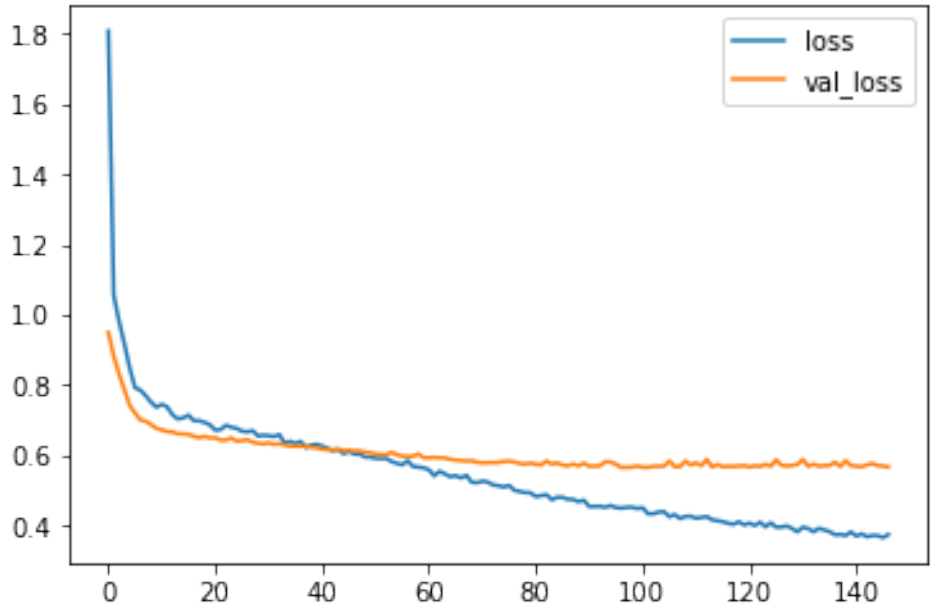

Figure 5.7 Taux d'apprentissage faible (0.0001)

Il n'y a donc pas de "bon" taux d'apprentissage dans l'absolu. Nous avons donc utilisé pour le modèle final une réduction automatique du taux d'apprentissage quand l'erreur de la validation cesse de diminuer. Plus exactement, lorsque l'erreur sur le jeu de validation n'a pas diminuée depuis 10 époques, on divise le taux d'apprentissage par 10. Ces deux dernières valeurs appelées respectivement "patience" et "facteur de réduction", sont eux-mêmes des hyperparamètres. Sur la Figure 5.8, on voit que le taux d'apprentissage élevé au début fait que l'apprentissage est instable, puis il est diminué (quatre fois) pour arriver à un taux tellement faible que l'apprentissage n'a plus lieu. Les variations de la courbe de perte sur le jeu d'entraînement sont dues au dropout.

5.5 Autres paramètres

Parmi les autres hyperparamètres de notre modèle, on trouve :

- Les fonctions d'activation
- La taille de échantillons d'entrainement
- La patience du mécanisme d'arrêt précoce de l'entrainement

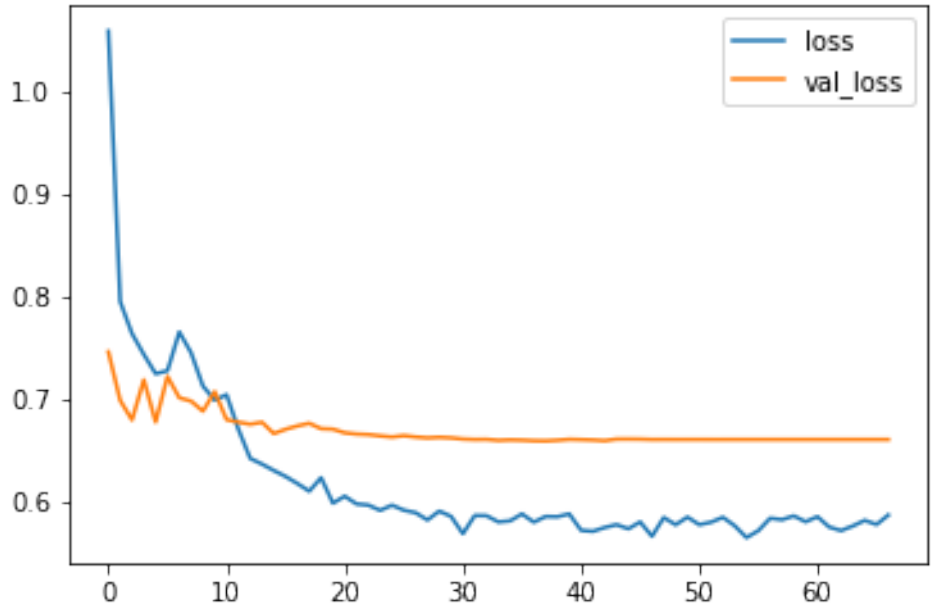

Figure 5.8 Réduction automatique du taux d'apprentissage

5.5.1 Fonctions d'activation

Le choix des fonctions d'activation s'est fait selon ce qui se pratique le plus dans la littérature scientifique à l'heure actuelle. En l'occurrence, la fonction Unité de Rectification Linéaire (ReLU) nous a servi la plupart du temps.

5.5.2 Taille des échantillons d'entraînement

La taille des échantillons d'entraînement était aussi importante. En effet, plus la taille de chaque échantillon d'entraînement est importante, plus le gradient calculé sera pertinent. Cependant, une taille plus petite permet une mise à jour plus fréquente du gradient et donc un apprentissage plus rapide. Par ailleurs, cela permet d'utiliser une quantité de mémoire plus faible et est donc plus rapide. Après quelques tentatives, nous avons décidé d'utiliser une taille d'échantillon d'entraînement de 64. Cela signifie que 64 échantillons du jeu de données sont montrés au modèle entre chaque mise à jour des poids.

5.5.3 Arrêt précoce de l'entraînement

Finalement, nous avons utilisé un mécanisme d'arrêt précoce de l'entraînement. Après un certain nombre de passages à travers le jeu de données au complet (époque) et alors que l'erreur sur le jeu de validation ne s'améliore pas, l'entrainement est arrêté. Cela gagne du

temps et permet d'effectuer plus d'expériences. Nous avons choisi une patience de 50. Ce paramètre est plus important qu'il en a l'air. En effet, un réseau peut s'entraîner en quelques époques ou en quelques centaines. Le laisser s'entraîner par défaut très longtemps est donc un choix inefficace. Cependant, il arrive aussi, comme on peut le voir sur la Figure 5.9 qu'un plateau semble être atteint (époque 10 à 25) avant que le modèle se débloque et continue d'apprendre. Choisir une patience trop faible peut empêcher ce phénomène en interrompant l'entraînement trop tôt.

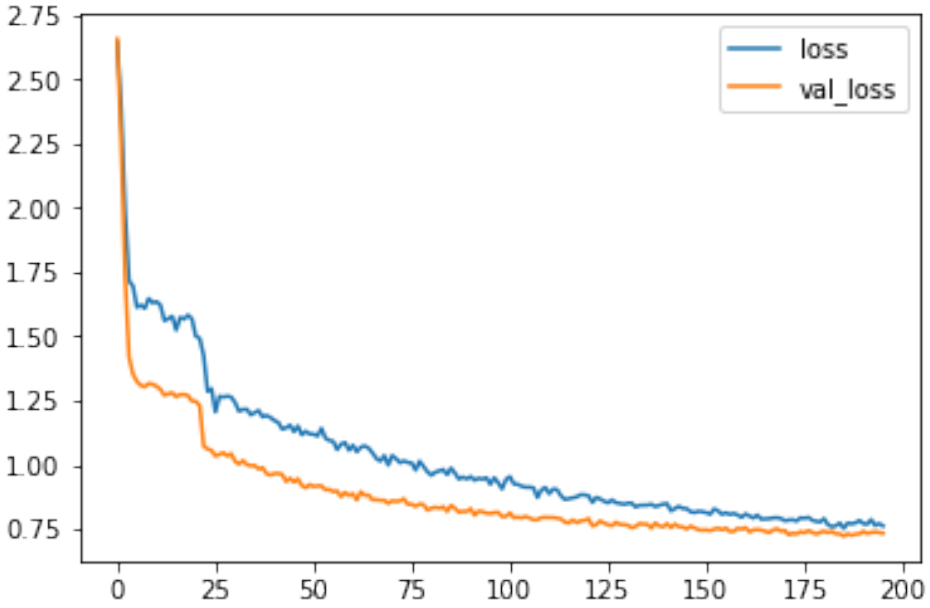

Figure 5.9 Petit plateau au début

5.6 Choix des hyperparamètres

Nous avons essayé de montrer l'importance que peuvent avoir tous ces hyperparamètres sur notre modèle. Cependant, les choix des hyperparamètres ne peuvent pas se faire indépendamment les uns des autres. En effet, si l'on procède ainsi on trouvera très probablement une solution sous-optimale même en itérant le processus plusieurs fois. Cela s'explique par le fait que tous ces paramètres agissent les uns sur les autres et sont donc interdépendants.
Pour les réseaux de neurones, compte-tenu de la longue durée de l'entraînement il est en général possible de procéder par combinaison aléatoire d'hyperparamètres. En effet, il est trop long de tous les tester et le hasard finit par trouver une solution acceptable.
Dans notre cas, notre modèle à base de CNN ne prend que quelques minutes à entraîner, nous avons donc pu effectuer une recherche par quadrillage des paramètres.

Les hyperparamètres considérés étaient :

— La taille de l'embedding initial : 50, 100, 200 ou 300
— Le niveau de dropout : par incrément de 10% de 10% jusqu'à 90%
— La taille de l'encodage produit par nos encodeurs : 10, 50, 100, 200, 300, 500 ou 1000
— La largeur du noyau de convolution : 3, 5, 10
— Le taux d'apprentissage : 0.0001, 0.0003, 0.001, 0.003 ou 0.01

Le reste des paramètres incluant le nombre de filtres, le nombre de couches a été optimisé à la main de façon indépendante, car chaque nouveau paramètre ajouté dans une recherche par quadrillage augmente considérablement le temps d'exécution. La combinaison optimale de paramètres sus-mentionnés est présentée par le Tableau 5.2.

Tableau 5.2 Meilleurs hyperparamètres pour le modèle CNN

Hyperparamètre	Valeur choisie
Taille de l'embedding initial	50
Dropout	50%
Taille de l'encodage	200
largeur du noyau de convolution	3
taux d'apprentissage	0.001

Au contraire, notre modèle à base de RNN était plus long à entrainer et il ne nous a pas été possible de faire une recherche exhaustive des hyperparamètres. Les paramètres ont donc été choisis à la main. Leur combinaison n'est selon toute vraisemblance pas tout à fait optimale. C'est probablement pour cela que sa performance finale était décevante.

5.7 Performance finale des deux modèles

Voici les résultats finaux de notre recherche. Pour les modèles CNN et RNN, seules les meilleures architectures sont retenues et nous avons évalué un intervalle de confiance à 95% de l'erreur en entrainant 10 fois le modèle de manières indépendantes.

Tableau 5.3 Comparaison de perfomances des modèles et des points de comparaison

Méthode	MSE	Intervalle de confiance
Meilleur CNN	0,643	± 0,054
Meilleur RNN	0,732	± 0,096
Vote moyen	0,936	
Régression linéaire	0,793	
Réseau de neurones	0,836	

On constate que pour les deux modèles l'intervalle de confiance est assez large car la variance était élevée sur les 10 tests effectués. Par ailleurs, tous les modèles testés avaient des performances similaires sur le jeu de données de validation.
Cependant, on peut conclure que les performances du CNN sont très bonnes : il bat tous les points de comparaison. Il n'est pas possible de dire de même pour le RNN : les performances de la régression linéaire sont dans l'intervalle de confiance de l'erreur attendue.

CHAPITRE 6 CONCLUSION

Ce mémoire présente un modèle de réseaux de neurones pour la prédiction du score de pertinence article-chercheur dans le cadre de l'organisation d'une conférence scientifique et rapporte les résultats de son évaluation.

6.1 Synthèse des travaux

Le meilleur modèle que nous avons développé et que nous avons sélectionné se fonde sur l'utilisation de CNN pour l'encodage des résumés d'articles sous forme vectorielle. On constate que cet encodage est pertinent : il préserve l'information puisque l'apprentissage reste possible et réduit beaucoup la taille des données, notamment en comparaison avec les méthodes classiques comme le TF-IDF. Les résultats obtenus sont convaincants et notre modèle final a une performance meilleure que les références que nous avons choisies à titre de comparaison. De plus, les temps de calcul restent raisonnables : la méthode a donc un certain potentiel pour s'appliquer concrètement à l'organisation de conférences scientifiques.
Compte-tenu de la faible taille du jeu de données à notre disposition, il a été utile de procéder à une augmentation artificielle de la taille de la base de données. A priori cette augmentation des données représentait une perte d'information importante, puisque chaque archive d'évaluateur s'est vue limitée au maximum à 3 articles. Cela restreint donc les données connues pour prédire chaque vote. Cependant, l'augmentation de la taille du jeu de données a eu un impact très positif : la possibilité d'augmenter la capacité de notre modèle sans tomber dans le surappprentissage. L'amélioration des résultats a été flagrante : cela est logique au vu de l'augmentation significative du nombre de points sur lesquels baser notre apprentissage supervisé. Cette technique a aussi été un des enseignements majeurs de notre recherche.

6.2 Limitations de la solution proposée

Les limitations de la solution proposée sont principalement dues à l'absence de point de comparaison disponible dans la littérature scientifique. En effet, la base qui a été utilisée ici a été transformée pour pouvoir être utilisée efficacement par les réseaux de neurones. Il en résulte l'impossibilité de comparer nos résultats à des résultats publiés sur la même base de données. Pour pallier à cela, nous avons établi des points de comparaisons qui nous semblaient pertinents en implémentant différents modèles pour comparer leur performance au nôtre.

6.3 Améliorations futures

Une amélioration future de notre recherche est de l'appliquer à des nouvelles bases auxquelles nous n'avons malheureusement pas pu avoir accès jusqu'à présent. Avec un nombre de votes plus importants, il ne sera pas nécessaire d'effectuer d'augmentation du jeu de données et une comparaison directe avec les méthodes disponibles dans la littérature pourra être faite. Cela augmentera sans aucun doute la crédibilité et la portée de notre recherche.
Par ailleurs, il serait intéressant de s'intéresser non seulement à l'expertise des chercheurs, mais aussi à leur degré d'expertise. En effet, sur un sujet donné, des articles très poussés ou bien au contraire assez basiques peuvent être écrits. Notre modèle ne fait pas état d'un degré de complexité - en tout cas pas explicitement. Le modèle pourrait donc s'enrichir si l'on essayait à un certain point d'évaluer la complexité et de l'inclure dans le calcul final de pertinence.

RÉFÉRENCES

D. Bahdanau, K. Cho, et Y. Bengio, “Neural machine translation by jointly learning to align and translate”, 2014.

D. M. Blei, A. Y. Ng, et M. I. Jordan, “Latent dirichlet allocation”, *J. Mach. Learn. Res.*, vol. 3, pp. 993–1022, Mars 2003. En ligne : `http://dl.acm.org/citation.cfm?id=944919.944937`

L. Charlin, R. Zemel, et C. Boutilier, “A framework for optimizing paper matching”, dans *Proceedings of the Twenty-Seventh Conference on Uncertainty in Artificial Intelligence*, série UAI’11. Arlington, Virginia, United States : AUAI Press, 2011, pp. 86–95. En ligne : `http://dl.acm.org/citation.cfm?id=3020548.3020560`

K. Cho, B. van Merrienboer, Ç. Gülçehre, F. Bougares, H. Schwenk, et Y. Bengio, “Learning phrase representations using RNN encoder-decoder for statistical machine translation”, *CoRR*, vol. abs/1406.1078, 2014.

J. L. Elman, “Finding structure in time”, *COGNITIVE SCIENCE*, vol. 14, no. 2, pp. 179–211, 1990.

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

S. Hochreiter et J. Schmidhuber, “Long short-term memory”, *Neural Comput.*, vol. 9, no. 8, Nov. 1997.

N. Kalchbrenner, E. Grefenstette, et P. Blunsom, “A convolutional neural network for modelling sentences”, *CoRR*, vol. abs/1404.2188, 2014.

Y. Kim, “Convolutional neural networks for sentence classification”, *CoRR*, vol. abs/1408.5882, 2014.

M. M. Lopez et J. Kalita, “Deep learning applied to NLP”, *CoRR*, vol. abs/1703.03091, 2017.

T. Mikolov, K. Chen, G. Corrado, et J. Dean, “Efficient estimation of word representations in vector space”, *CoRR*, vol. abs/1301.3781, 2013.

D. Mimno et A. McCallum, “Expertise modeling for matching papers with reviewers”, dans *KDD*, 2007.

J. Pennington, R. Socher, et C. D. Manning, “Glove : Global vectors for word representation.” dans *EMNLP*, vol. 14, 2014, pp. 1532–1543.

M. A. Rodriguez, J. Bollen, et H. V. de Sompel, “The convergence of digital libraries and the peer-review process”, *Journal of Information Science*, vol. 32, no. 2, pp. 149–159, 2006.

R. Salakhutdinov et A. Mnih, “Bayesian probabilistic matrix factorization using markov chain monte carlo”, dans *Proceedings of the 25th International Conference on Machine Learning*, série ICML ’08. New York, NY, USA : ACM, 2008, pp. 880–887. DOI : 10.1145/1390156.1390267. En ligne : http://doi.acm.org/10.1145/1390156.1390267

X. Wei et W. B. Croft, “Lda-based document models for ad-hoc retrieval”, dans *SIGIR*, 2006.