travail et rapport très bien mené. Quelques imprécisions, notamment sur le niveau mot/car et sur les métriques. Qu'est-ce qui ne fonctionne pas ?

Rapport du TP1: textes à trous

Kristof Boucher Charbonneau
Présenté à monsieur Philippe Langlais

18 octobre 2018

Résumé

Ce travail présente une architecture neuronale combinant une représentation vectorielle des caractères et des mots et des afin de prédire le ou les mots manquants d'une séquence. À l'aide d'un réseau de convolutions (CNN) de nouvelles informations sont extraites des caractères, puis un « Long Short-Term Memory » (LSTM) permet d'identifier les informations importantes des séquences temporelles. Notre analyse suggère que d'entraîner un réseau en simulant les conditions réelles améliore la robustesse des prédictions.

1. Introduction

La génération de texte est une tâche qui peut être employée afin de prédire les mots manquants d'une phrase, une sous-tâche de la modélisation du langage. Cette modélisation implique de maintenir une distribution de probabilité sur l'ensemble des mots. Traditionnellement, la génération de texte utilise les modèles « n-gram » ainsi que quelques techniques de lissage afin de trouver le meilleur mot subséquent possible (Brown et al., 1992). L'état de l'art actuel en génération de texte utilise maintenant les réseaux de neurones.

Plus particulièrement, les réseaux de neurones conditionnés sur les caractères plutôt que les mots ont démontré de meilleurs résultats (Kim et al., 2016) en termes de précision et de rapidité. Ces résultats sont expliqués par le fait que ces réseaux doivent simplement maintenir une table des caractères possibles du langage plutôt que de l'ensemble du vocabulaire, ce qui contribue à une optimisation considérable de la mémoire, à réduire les paramètres et l'espace de décision.

Ce projet du cours de traitement automatique des langues naturelles explore la possibilité de modéliser la langue anglaise à l'aide d'un réseau de neurones conditionné sur les caractères en s'inspirant d'une architecture proposée en 2016 (Kim et al., 2016) et en y apportant des modifications afin d'obtenir de meilleures performances. Les résultats des expériences démontrent une bonne compréhension du texte par le réseau et une « accuracy » de 84,16%. Quatre sections sont présentées dans ce rapport décrivant le corpus de données utilisées, l'architecture, l'entraînement du réseau et une discussion sur les résultats obtenus.

2. Corpus

Les données utilisées pour l'entraînement et l'évaluation sont des phrases qui ont été extraites de la septième version du corpus « Europarl » (Koehn, 2005). Cet ensemble contient les textes parlementaires de l'Europe en 21 langues différentes dont le but initial était la traduction automatique. Un traitement préliminaire a donc été réalisé pour adapter ce corpus à la génération de texte en substituant certains mots de la phrase par un jeton « <unk> ». Le travail est effectué sur la langue anglaise. L'ensemble d'entraînement comporte 1M de phrases alors que l'ensemble de tests en comporte 1000 avec une quantité variable de jetons inconnus.

2.1. Prétraitement

La distribution des données utilisées est similaire à une distribution normale : beaucoup d'éléments apparaissent peu de fois alors que peu d'éléments apparaissent plusieurs fois. Le vocabulaire a été réduit afin de conserver seulement les éléments qui apparaissent plus de 50 fois dans le corpus, ce qui représente une réduction de 71,45% du vocabulaire initiale. Cette décision a été prise notamment par souci de mémoire du matériel disponible, mais également liée à l'hypothèse que la probabilité d'obtenir un élément est proportionnelle à son nombre d'occurrences.

Les éléments constitués de caractères ne faisant pas partie de l'anglais ont été supprimés. Ainsi, une ta-

Mise en page provenant du gabarit de la conférence ICML 2017: https://icml.cc/Conferences/2017/StyleAuthorInstructions

ble de 94 caractères disponibles a été utilisée, celle-ci comporte différents signes de ponctuation, des chiffres et des lettres de case différente. La case des mots a été respectée puisque les résultats expérimentaux démontrent très peu de différence. Finalement, le jeton inconnu a été remplacé par un caractère n'apparaissant pas dans le corpus : « | » afin de considérer seulement un caractère plutôt que cinq lors du décodage.

3. Architecture

S'inspirant du travail de Kim et al. (2016); Boom et al. (2018), l'architecture présentée dans cette section décrit en détail les différentes couches du réseau, mais également les modifications qui ont été apportées par rapport à l'implémentation originale. Un vecteur de caractères formant un mot est d'abord transformé dans un nouvel espace vectoriel puis un réseau convolutif (CNN) utilise cet espace pour extraire de nouvelles primitives (« features »). Ce résultat est ensuite concaténé à un espace vectoriel de mots. Finalement, un réseau récurrent (RNN) prédit le prochain mot. Cette architecture est illustrée par 1.

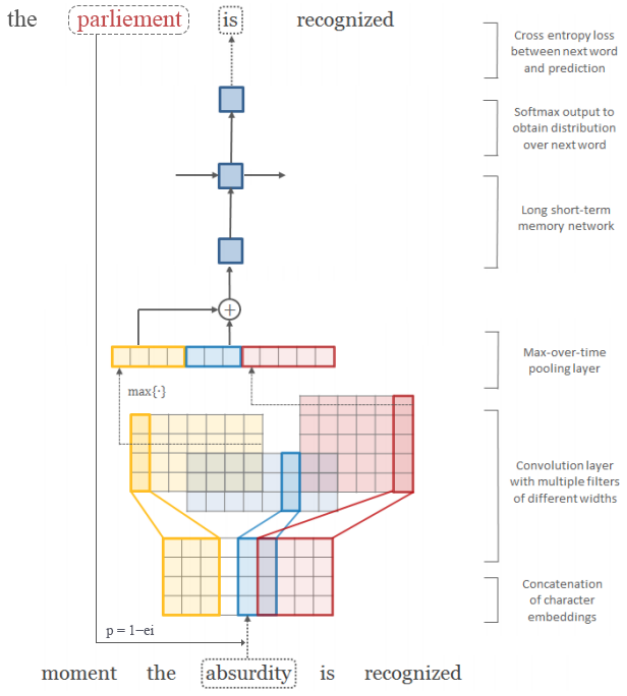

Figure 1. Architecture du réseau. En rouge, une mauvaise prédiction du réseau utilisé au temps actuel et en gris, une bonne prédiction. Inspiré de Kim et al. (2016)

3.1. Espace vectoriel des caractères et des mots

Deux ensembles de données sont utilisés comme valeurs de départ dans le réseau de neurones : une matrice de caractères et une matrice de mots qui sont projetés dans un espace vectoriel de dimension fixe. Ces espaces vectoriels permettent de représenter de façon discrète des mots et des caractères dans un espace de dimension non variable. Une caractéristique importante de ces espaces est la relation entre la distance de deux vecteurs et leur corrélation qui nous informe de leur similarité.

Nous rapportons l'entraînement d'une variante du réseau utilisant soit un espace vectoriel pour les caractères ou soit un espace vectoriel pour les mots (1): la combinaison de ces deux espaces permet d'obtenir de meilleurs résultats. Notre hypothèse est que l'utilisation de la représentation vectorielle des caractères induit une contrainte sur les mots utilisant seulement les caractères disponibles alors que l'utilisation de la représentation vectorielle des mots limite le vocabulaire disponible.

3.2. Réseau convolutif

L'utilisation de plusieurs couches de convolution permet d'exploiter la structure temporelle de la représentation des caractères afin d'extraire les données les plus importantes. Ces couches transforment les vecteurs d'entrés en tableaux de caractéristiques (« feature maps »). Une couche de « pooling » permet ensuite d'extraire la valeur la plus grande des relations trouvées par la couche précédente.

Le réseau utilise quatre filtres de taille différente afin de simuler quatre modèles probabiliste « n-gram» traditionnels où « n » est la taille d'un filtre. Cette variation dans la taille d'un filtre peut être interprétée comme l'utilisation de plusieurs modèles « n-gram » avec un historique variable.

Plus formellement, les couches convolutives utilisent conjointement les données avec des noyaux et des poids afin d'extraire les caractéristiques. Chaque neurone est connecté à ses voisins dans la couche précédente à l'aide de poids qui sont appris par le réseau.

$$Y_i = \sigma(W_i * x) \tag{1}$$

La couche « max pooling » sélectionne les valeurs les plus grandes en fonction de la taille de la fenêtre d'extraction. Intuitivement, pour notre problème, les valeurs extraites sont les prochains mots les plus probables.

3.3. Réseau récurrent

La prédiction de texte est la tâche de prédire le prochain mot y_t étant donné un historique des mots précédents $y_{y-1}, y_{t-2}...$ soit $_{y_t} = p(y_t|y_{t-n})$. Où n est le nombre de mots à considérer dans l'historique. Les réseaux de neurones récurrents (RNNs) sont une approche intéressante afin de résoudre les problèmes où une dépendance temporelle est présente. Cependant, les RNN ont beaucoup de difficulté à capturer ces dépendances puisqu'ils sont particulièrement susceptibles au problème de disparition du gradient (« gradient vanishing problem »). Plusieurs alternatives ont été créées afin de résoudre ce problème, comme le « Long Short-Term Memory » (LSTM) (Hochreiter & Schmidhuber, 1997).

L'architecture d'un LSTM compose trois « portes » qui contrôlent l'information enregistrée dans la mémoire interne ainsi que l'information retournée par le réseau.

$$i_t = \sigma(W_i h_{t-1} + U_i x_t + b_i) \quad (2)$$
$$f_t = \sigma(W_f h_{t-1} + U_f x_t + b_f) \quad (3)$$
$$\tilde{c_t} = \tanh(W_c h_{t-1} + U_c x_t + b_c) \quad (4)$$
$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c_t} \quad (5)$$
$$o_t = \sigma(W_o h_{t-1} + U_o x_t + b_o) \quad (6)$$
$$h_t = o_t \odot t\tanh(c_t) \quad (7)$$

Corollairement: i est la porte d'entrée qui permet de choisir l'information à mettre à jour, c est un vecteur de nouvelles valeurs à enregistrer, f est la porte qui permet de supprimer l'information qui n'est plus importante, o est la porte qui décide quelle information retourner et finalement h permet de changer le domaine de l'information entre -1 et +1. Les poids W_i, W_f, W_c, W_o contrôlent l'état caché interne alors que les poids U_i, U_f, U_c, U_o sont les poids des entrées x_t et b_i, b_f, b_c, b_o sont les biais pour chacune des portes.

4. Entraînement

Afin d'entraîner le modèle, nous utilisons l'entropie croisée entre une séquence de vrais mots y_i et une séquence des mots qui ont été générés p_i par la dernière couche totalement connectée: $\mathcal{L} = -\sum_i y_i \log(p_i)$. Ce modèle est optimisé par une technique adaptative du gradient appelé « Adam » (Kingma & Ba, 2014).

Toutes les expériences ont été réalisées sur une machine disposant de deux cartes graphiques Nvidia GTX 1080 TI ayant 11 Gb de VRAM. L'entraînement du meilleur réseau prend environ 47 minutes pour 500 « epochs ».

4.1. Regularization

Comme le réseau est relativement profond et la stabilité de ses composantes dépend des données d'entrées, trois formes de régularisation ont été utilisées. La régularisation d'un réseau de neurones permet d'empêcher le sur-apprentissage, le sous-apprentissage ou de rendre celui-ci plus « stable ».

La première forme de régularisation utilisée est le « dropout » (Srivastava et al., 2014) et est appliquée sur la couche récurrente du modèle. Cette technique permet de désactiver des neurones de façon aléatoire afin d'empêcher le sur-apprentissage du réseau. En rendant certains neurones non fiables, il est possible d'empêcher la création de liens entre les neurones qui pourrait lier à une perte de généralisation sur de nouvelles données.

La seconde régularisation est réalisée lors de l'optimisation et est également liée à la couche récurrente. La propagation d'un tel réseau dépend de la taille de la séquence temporelle. Nos expériences démontrent que le gradient du LSTM a tendance à devenir trop grand et empêche l'apprentissage. Nous limitons la valeur du gradient à une borne (« gradient clipping ») (Pascanu et al., 2012) pour contrer cet effet.

Finalement, nous avons découvert que le réseau a du mal à converger vers un optimum. Pour pallier à ce problème, nous avons configuré le coefficient d'apprentissage selon une fonction exponentielle conditionnée sur le nombre d'itérations afin de décroitre le coefficient lorsque le réseau n'apprend plus. Nous notons un gain de performance d'environ 8%.

4.2. « Teacher Forcing »

La prédiction d'un mot manquant au sein d'une séquence de mots est une tâche relativement simple. Cependant, il peut arriver que plus d'un mot consécutif soit manquant: le réseau doit donc apprendre à utiliser le mot généré précédemment pour prédire le second. Pour simuler cette contrainte, il existe une technique appelée « Teacher Forcing » (Goodfellow et al., 2016) qui utilise la prédiction de l'itération précédente y^{t-1} à l'itération actuelle t.

[Figure Teacher Forcing]

Cependant, si la dimension de l'axe temporelle est importante, cette technique n'est pas idéale puisque la propagation de l'erreur de la prédiction est exponentielle. Pour régler ce problème, nous avons utilisé une technique appelée « Schedule Sampling » (Bengio et al., 2015). Plutôt que de toujours utiliser les pré-

dictions de l'itération précédente, cet algorithme remplace aléatoirement certaines données de l'itération actuelle par celles générées précédemment à un rythme dicté par une fonction « sigmoid ». Soit un nombre de données généré au temps précédent à utiliser dénoté par $1 - \epsilon_i$, i l'itération actuel du réseau et k la rapidité à d'échantillonnage: $1 - \epsilon_i = k/(k + \exp i/k)$. La différence entre un entraînement avec et sans échantillonnage est rapportée dans 1. En forçant l'utilisation d'un mot généré de façon aléatoire nous simulons les conditions réelles et nous contribuons à la robustesse de celui-ci.

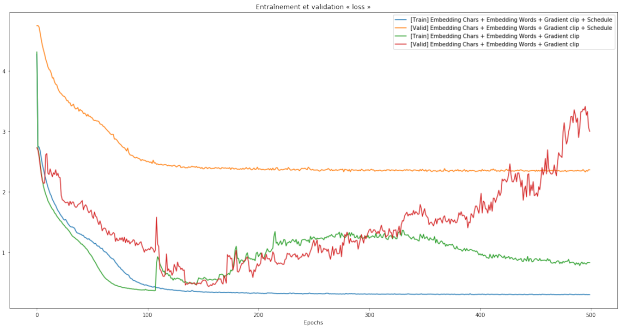

Figure 2. « Loss » de l'entraînement et de la validation. Les pics sont causés par l'échantillonnage.

5. Résultats et discussion

Dans cette section nous rapportons les résultats des expériences qui ont été réalisées sur le corpus présenté en 2. Un « baseline » très simple a été réalisé afin de pouvoir comparer la performance du modèle présenté. Celui-ci correspond au modèle « LSTM+Dense » dans (1).

2 rapporte la combinaison des différents hyperparamètres qui forme le meilleur modèle. Notons qu'à la fin de l'entraînement, pratiquement tous les mots générés sont considérés comme les mots futurs par « Schedule Sampling ». Une analyse sans ce mécanisme a également réalisé. Bien que le gain de « accuracy » ne soit pas considérable, la différence de « precision » indique une fréquence plus élevée du bon mot correctement prédit par rapport aux mauvaises prédictions et que la différence de « recall » indique une fréquence plus élevée de bonne prédiction. Ces résultats dénotent que le modèle entraîné avec échantillonnage est plus robuste au bruit et est en mesure de corriger ses erreurs de prédictions 3. De plus, nous croyons que le nombre de filtres et la dimension utilisée permettent également d'améliorer ces résultats puisque la variabilité de la taille de l'historique induite par les tailles différente des filtres permettent de considérer une fenêtre de mots plus ou moins grande.

Les hyperparamètre finaux qui ont été sélectionnés sont une combinaison du travail de Kim et al. (2016); Low (2016). Contrairement à ces deux travaux, notre implémentation n'utilise aucune couche « highway ». Notre hypothèse est que la séquence temporelle utilisée par notre modèle n'est pas aussi longue que celle recommandée par les auteurs, de même que la quantité des mini-lots (« mini-batch ») même si notre vocabulaire semble plus long.

Nous notons également que le réseau est sensible à l'initialisation des paramètres. Il arrivait qu'un optimum local soit trouvé très tôt et que même avec l'aide de diverses techniques de régularisation il était impossible de sortir de l'optimum.

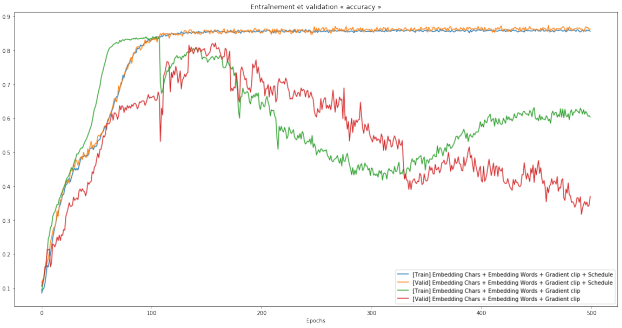

Figure 3. « Accuracy » de l'entraînement et de la validation. Les pics sont causés par l'échantillonnage.

6. Conclusion

Ce travail a permis d'évaluer la performance d'un réseau de neurones prédisant les mots manquants d'une séquence à l'aide de couches différentes et d'un espace vectoriel pour les caractères et les mots. Un modèle existant à été implémenté avec Tensorflow et une exploration d'hyperparamètres différents à été réalisée en plus d'avoir apporté des modifications au modèle original. Nous avons démontré qu'ajouter un échantillonnage aléatoire lors de l'entraînement permet au réseau d'apprendre à corriger ses erreurs et à être plus robuste lorsqu'il faut générer des prédictions dans un environnement réel.

Même si les résultats démontrent de bonnes performances, il aurait été intéressant d'explorer l'effet d'une couche récurrente bidirectionnelle qui aurait permis de connaître l'information du « futur » afin de générer le mot manquant. Nous examinerons cette nouvelle architecture lors d'une prochaine itération.

Tableau 1. Comparaison des résultats obtenus avec les différentes combinaisons.

Combinaison	*accuracy*
LSTM + Dense (baseline)	68.74
Embedding Chars + Embedding Words + SGD + Gradient clip	24.30
Embedding Words + Gradient clip + Schedule	39.51
Embedding Chars + Gradient clip + Schedule	70.02
Embedding Chars + Embedding Words + Gradient clip	82.33
Embedding Chars + Embedding Words + Gradient clip + Schedule	**84.16**

Tableau 2. Hyperparamètres des différentes couches du réseau de neurones.

Couche	Hyperparamètre	Valeur
Embedding Chars	Dimension	15
CNN	Filtres	50, 100, 150, 200, 200, 200
	Noyau	1 x 15, 2 x 15, 3 x 15, 4 x 15, 5 x 15, 6 x 15
	Pas	1
Max Pooling	Dimensions	14 x 1, 13 x 1, 12 x 1, 11 x 1, 10 x 1, 9 x1
Embedding Words	Dimension	600
Batch Normalization	Statistiques	0.99, 0.001
LSTM	Neurones	600, 600
	Dropout	0.5
Dense	Neurones	dim. sortie.

References

Bengio, Samy, Vinyals, Oriol, Jaitly, Navdeep, and Shazeer, Noam. Scheduled sampling for sequence prediction with recurrent neural networks. *CoRR*, abs/1506.03099, 2015. URL http://arxiv.org/abs/1506.03099.

Boom, Cedric De, Demeester, Thomas, and Dhoedt, Bart. Character-level recurrent neural networks in practice: Comparing training and sampling schemes. *CoRR*, abs/1801.00632, 2018. URL http://arxiv.org/abs/1801.00632.

Brown, Peter F, DeSouza, Peter V, Mercer, Robert L, Della Pietra, Vincent J, and Lai, Jenifer C. Class-Based n-gram Models of Natural Language. *Association for Computational Linguistics*, 1992. ISSN 08912017. doi: 10.1097/00004583-200107000-00010.

Goodfellow, Ian, Bengio, Yoshua, and Courville, Aaron. *Deep Learning*. MIT Press, 2016.

Hochreiter, Sepp and Schmidhuber, Jürgen. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735. URL http://dx.doi.org/10.1162/neco.1997.9.8.1735.

Kim, Yoon, Jernite, Yacine, Sontag, David, and Rush, Alexander M. Character-aware neural language models. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI'16, pp. 2741–2749. AAAI Press, 2016. URL http://dl.acm.org/citation.cfm?id=3016100.3016285.

Kingma, Diederik P. and Ba, Jimmy. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. URL http://arxiv.org/abs/1412.6980.

Koehn, Philipp. Europarl: A Parallel Corpus for Statistical Machine Translation. In *Proceedings of MT Summit X*, 2005. ISBN 9747431262. doi: 10.3115/1626355.1626380.

Low, Melvin. Character-level recurrent text prediction. 2016.

Pascanu, Razvan, Mikolov, Tomas, and Bengio, Yoshua. Understanding the exploding gradient problem. *CoRR*, abs/1211.5063, 2012. URL http://arxiv.org/abs/1211.5063.

Srivastava, Nitish, Hinton, Geoffrey, Krizhevsky, Alex, Sutskever, Ilya, and Salakhutdinov, Ruslan. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Dropout: A Simple Way to Prevent Neural Networks from Overfitting*, 2014. ISSN 15337928. doi: 10.1214/12-AOS1000.

Tableau 3. Meilleur modèle: « Recall », « Precision » et « F1 » lié à « Schedule Sampling »

Modèle	Recall	Precision	F1
... avec Schedule Sampling	84.98	86.37	85.67
... sans Schedule sampling	79.37	83.69	80.96