Actes de la 28e Conférence sur le Traitement Automatique des Langues Naturelles
Lille, France, 28 juin au 2 juillet 2021

Contribution d'informations syntaxiques aux capacités de généralisation compositionelle des modèles `seq2seq` convolutifs

Diana Nicoleta Popa[1] William N. Havard[1]
Maximin Coavoux[1] Laurent Besacier[2] Éric Gaussier[1]
(1) Univ. Grenoble Alpes, CNRS, Grenoble INP, LIG
(2) Naver Labs Europe
{diana.popa,william.havard,maximin.coavoux,eric.gaussier}
@univ-grenoble-alpes.fr
laurent.besacier@naverlabs.com

RÉSUMÉ

Les modèles neuronaux de type `seq2seq` manifestent d'étonnantes capacités de prédiction quand ils sont entraînés sur des données de taille suffisante. Cependant, ils échouent à généraliser de manière satisfaisante quand la tâche implique d'apprendre et de réutiliser des règles systématiques de composition et non d'apprendre simplement par imitation des exemples d'entraînement. Le jeu de données SCAN, constitué d'un ensemble de commandes en langage naturel associées à des séquences d'action, a été spécifiquement conçu pour évaluer les capacités des réseaux de neurones à apprendre ce type de généralisation compositionnelle. Dans cet article, nous nous proposons d'étudier la contribution d'informations syntaxiques sur les capacités de généralisation compositionnelle des réseaux de neurones `seq2seq` convolutifs.

ABSTRACT

Assessing the Contribution of Syntactic Information for Compositional Generalization of `seq2seq` Convolutional Networks

Classical sequence-to-sequence neural network architectures demonstrate astonishing prediction skills when they are trained on a sufficient amount of data. However, they fail to generalize when the task involves learning and reusing systematic rules rather than learning through imitation from examples. The SCAN dataset consists of a set of mapping between natural language commands and actions and was specifically introduced to assess the ability of neural networks to learn this type of compositional generalization. In this paper, we investigate to what extent the use of syntactic features help convolutional seq2seq models to better learn systematic compositionality.

MOTS-CLÉS : Compositionnalité, modèle convolutionnel seq2seq, jeux de données SCAN.

KEYWORDS: Compositionality, convolutional seq2seq model, SCAN dataset.

1 Introduction

Si les réseaux de neurones obtiennent des résultats remarquables sur différentes tâches de traitement automatique des langues (TAL), telles que la traduction automatique, les tâches de question-réponse ou la génération de texte, plusieurs études (par exemple Jia & Liang (2017) en lecture automatique,

Volume 1 : conférence principale, pages 134–141.

Input original	`jump opposite right twice and turn opposite right thrice`
ARBRE PRÉFIXE COMPLET	`(C (S (V (U jump) opposite right) twice) and (S (V turn opposite right) thrice))`
ARBRE POSTFIXE COMPLET	`((((jump U) opposite right V) twice S) and ((turn opposite right V) thrice S) C)`
ARBRE NON ÉTIQUETÉ COMPLET	`((((jump) opposite right) twice) and ((turn opposite right) thrice))`
ARBRE NON ÉTIQUETÉ PARTIEL	`(((jump opposite right) twice) and ((turn opposite right) thrice))`
CHUNKS-V	`(V jump opposite right) twice and (V turn opposite right) thrice`
Output	`RTURN RTURN JUMP RTURN RTURN JUMP RTURN RTURN RTURN RTURN RTURN RTURN`

TABLE 1: Input augmenté avec différents types d'informations syntaxiques.

ou Belinkov & Bisk (2018) en traduction automatique) montrent que leurs capacités de généralisation sont limitées. Plus récemment, il a été montré que les réseaux de neurones échouent à faire des généralisations compositionnelles simples, c'est-à-dire qu'ils ne parviennent pas à inférer le sens d'une expression complexe (*jump twice*) même en connaissant les sens respectifs de ses parties (*jump*, *twice*). Comme illustré par Dessi & Baroni (2019), les réseaux `seq2seq` convolutifs n'obtiennent qu'environ 70% d'exactitude sur le corpus de test de la tâche *jump* du jeu de données SCAN (Lake & Baroni, 2018), les GRU et LSTM étant encore plus faibles (respectivement 12.5% et 1.2%).

Ces résultats ne sont pas surprenants dans la mesure où Lake & Baroni (2018) ne donnent aucune information, à part les formes de surface, pour guider le modèle vers l'apprentissage de la compositionalité. Ces modèles apprennent essentiellement à imiter les exemples qu'ils voient lors de l'apprentissage. Ainsi, cela soulève la question de savoir si l'ajout d'informations, qui inciteraient le réseau de neurones à s'abstraire des formes de surface, amélioreraient les capacités compositionnelles du modèle. Nous proposons d'étudier cette question en analysant la contribution d'informations syntaxiques sur le jeu de données SCAN. Pour cela, un protocole expérimental est proposé avec différents types d'information syntaxiques ajoutées a priori sur les séquences d'entrée (voir Table 1) sans changer la nature de l'architecture étudiée (modèle `seq2seq` convolutif).

2 Travaux connexes

Lake & Baroni (2018) ont conçu le jeu de données SCAN pour évaluer les capacités de généralisation compositionnelle des modèles `seq2seq` classiques. Ce jeu de données comprend des commandes de navigation données en anglais simplifié et associées à des séquences d'actions. Nous donnons un exemple dans la Table 1 (lignes "*input original*" et "*output*"). Pour constituer le jeu de données, des commandes ont été générées à partir d'une grammaire hors contexte, tandis que les séquences d'actions ont été calculées à partir d'une fonction déterministe d'interprétation sémantique. Plusieurs divisions en corpus d'entraînement et d'évaluation ont été considérées sur ce jeu de données pour évaluer plusieurs types de généralisation. Dans cet article nous nous concentrons sur la tâche dite du *split* JUMP (Lake & Baroni, 2018). Dans le *split* JUMP, les seules occurrences de *jump* dans le corpus d'entraînement apparaissent de manière isolée (seules les autres actions pouvant être combinées). Dans le corpus d'évaluation, chaque exemple contient le token *jump* combiné avec d'autres tokens (*jump twice*, *jump opposite left*).

Dans l'article original de SCAN (Lake & Baroni, 2018), les réseaux récurrents (RNN, LSTM) échouent à réaliser ces généralisations. Dessi & Baroni (2019) ont évalué des modèles `seq2seq` convolutifs sur la même tâche et rapportent de bien meilleurs résultats que les RNN. Depuis ces articles fondateurs, les chercheur·es ont proposé des architectures et méthodes d'entraînement plus complexes. Par exemple, Lake (2019) propose une architecture dotée d'une mémoire (*memory-augmented*) et

entraînée par méta-apprentissage où chaque épisode d'apprentissage consiste à apprendre à combiner une nouvelle action primitive vue seulement de manière isolée.

Liu *et al.* (2020) proposent également une architecture dotée d'une mémoire et divisée en deux modules (*composer* et *solver*) . Récemment, deux modèles `seq2seq` ont obtenu de bons résultats sur le *split* JUMP. Le premier (Russin *et al.*, 2020b) présente un mécanisme d'attention basé sur deux vecteurs pour chaque mot : un vecteur contextualisé et un vecteur – dit "sémantique" – indépendant du contexte. Le second (Gordon *et al.*, 2020) présente un modèle `seq2seq` basé sur des représentations équivariantes pour certaines permutations de tokens.

Dans cet article, nous nous concentrons sur une architecture simple : un réseau `seq2seq` convolutif, et nous nous intéressons à la contribution des informations syntaxiques (données de différentes manières en input au modèle) aux capacités de généralisation compositionnelle du modèle.

3 Traits syntaxiques

Nous faisons l'hypothèse qu'une raison pour laquelle les modèles `seq2seq` échouent à apprendre la compositionalité systématique à partir de SCAN est que ces systèmes ne modélisent pas la structure syntaxique des énoncés. Pour tester cette hypothèse, nous proposons d'augmenter les exemples d'input de SCAN avec des informations syntaxiques explicites et d'évaluer si cela permet d'améliorer les capacités de compositionalité systématique des modèles. [1]

Pour cela, nous commençons par faire l'analyse syntaxique de chaque input à l'aide d'un analyseur CKY standard [2] et de la grammaire non ambiguë fournie en annexe de l'article SCAN (Lake & Baroni, 2018). Ensuite, nous augmentons les phrases d'input pour qu'elles aient l'une des formes suivantes :

i Arbre complet étiqueté en notation préfixe (ARBRE PRÉFIXE COMPLET) ;
ii Arbre complet étiqueté en notation postfixe (ARBRE POSTFIXE COMPLET) ;
iii Arbre complet non étiqueté (ARBRE NON ÉTIQUETÉ COMPLET) ;
iv Arbre partiel non étiqueté (ARBRE NON ÉTIQUETÉ PARTIEL), obtenu en retirant les parenthèses correspondant aux règles de grammaire unaire [3] ;
v Analyse syntaxique superficielle (syntagmes étiquetés V, CHUNKS-V).

Nous présentons la forme de l'entrée pour chaque cas dans la Table 1. Les deux premiers types d'input (i-ii) codent le maximum d'information syntaxique : parenthésage complet et étiquetage des syntagmes par une des étiquettes suivantes :

— U représente une action primitive (`walk, look, jump, run`) ;
— D représente un syntagme complexe comprenant un changement de direction (`turn, left, right`) ;
— V est utilisé pour des syntagmes ayant potentiellement un modifieur tel que `opposite` ou `around` ;
— S représente une proposition complète (potentiellement plusieurs actions primitives et leurs modifieurs) ;
— C représente des coordinations.

1. Les données augmentées sont librement disponibles : `https://github.com/mcoavoux/SCAN_syntax`.
2. Nous utilisons l'implémentation de la bibliothèque NLTK (Bird *et al.*, 2009).
3. Les tokens `walk`, `look`, `run` et `jump` ne peuvent être générés que par des règles unaires. Tous les non-terminaux (sauf l'axiome C) peuvent aussi être générés par des règles unaires.

Les trois derniers types d'input (ARBRE NON ÉTIQUETÉ COMPLET, ARBRE NON ÉTIQUETÉ PARTIEL et CHUNKS-V) permettent d'isoler les contributions des informations de structures et d'étiquetage : Le type d'input (iii) contient toute la structure syntaxique mais pas d'étiquette. Le type (iv) représente l'arbre non étiqueté et sans le parenthésage correspondant aux règles unaires. Par exemple, `jump` (Table 1) est au même niveau que `opposite` et `right` pour le protocole ARBRE NON ÉTIQUETÉ PARTIEL. Enfin, le type (v) représente un unique type de syntagme (V). Nous considérons les parenthèses ouvrantes et fermantes, ainsi que les étiquettes des syntagmes comme des tokens. Ces inputs enrichis permettent ainsi d'entraîner une même architecture en faisant varier la quantité d'information syntaxique fournie afin d'en étudier la pertinence et l'utilisation par un réseau `seq2seq` convolutif.

Le vocabulaire des versions enrichies en syntaxe que nous avons construites comprend 20 symboles au plus : les 13 symboles initiaux (setting ORIGINAL), ainsi que les parenthèses et éventuellement les non-terminaux.

4 Expériences

Protocole expérimental Comme Dessi & Baroni (2019), nous utilisons un modèle de type encodeur-décodeur convolutif (Gehring *et al.*, 2017), via l'implémentation de `fairseq` (Ott *et al.*, 2019) v.0.9.0, [4] et PyTorch 1.5.0. Ce choix est motivé par les conclusions de Dessi & Baroni (2019) qui montrent des résultats empiriques nettement superieurs par rapport aux LSTM. [5] À l'instar de travaux précédents (Loula *et al.*, 2018; Lake & Baroni, 2018; Dessi & Baroni, 2019) et pour assurer une comparabilité des résultats, nous dupliquons l'ensemble d'exemples d'entraînement original pour obtenir 100 000 exemples. Cependant, nous utilisons une méthode légèrement différente [6] : nous commençons par échantillonner aléatoirement 10% des données d'entraînement pour les utiliser comme ensemble de validation. Parmi les 90% restant, nous obtenons le jeu de données augmenté en dupliquant chaque commande pour qu'elle apparaîsse 7 fois. Pour le split JUMP, cela nous donne 92 421 exemples d'entraînement et 1 467 exemples pour la validation. Nous évaluons sur le même ensemble de test que les travaux précédents (7 706 exemples).

Pour calibrer les hyperparamètres, nous faisons varier le pas d'apprentissage (0.1, 0.01), la taille des batches (50, 100, 200, 500, 1000), le nombre de couches cachées (de 5 à 10), la taille des vecteurs de mots (128, 256, 512), la probabilité de *dropout* (0, 0.25, 0.5) et la taille du noyau de convolution (3, 4, 5). Ces plages de valeurs sont similaires à celles explorées dans des travaux précédents. Nous entraînons chaque configuration du modèle pendant une époque (tel que fait dans (Lake & Baroni, 2018)) et répliquons le procédé trois fois avec des graines aléatoires distinctes.

Effet des Informations Syntaxiques Nous présentons dans la Table 2 les résutats obtenus avec les meilleurs modèles en moyennant (ou non) sur 3 graines, c'est-à-dire en moyennant sur trois entrainements de la même architecture dont le seul paramètre de variation est l'initialisation des poids (*via* la graine aléatoire). Tandis que les scores non moyennés représentent une limite haute de scores qui peuvent effectivement être obtenus avec une architecture donnée, la moyenne des scores des 3 graines représente plus fidèlement un niveau moyen de performances d'un même architecture. Dans

4. https://github.com/pytorch/fairseq

5. Ils attribuent cette supériorité des CNN aux biais d'induction des *kernels* de convolutions qui traitent le texte par fenêtres de mots et favorisent l'apprentissage de patrons.

6. L'article original de SCAN (Lake & Baroni, 2018) échantillonne avec remplacement 100k exemples parmi les 14 670 exemples distincts du corpus d'entraînement.

	Exactitude tous modèles				Exactitude tous modèles, moyenné sur 3 graines aléatoires			
	Top 1	Top 5	Top 10	$\Delta(\%)$	Top 1	Top 5	Top 10	$\Delta(\%)$
ORIGINAL	84.88	81.89($\pm$2.01)	79.44($\pm$2.96)	-	68.39($\pm$4.3)	67.19($\pm$0.69)	66.04($\pm$1.33)	-
CHUNKS-V	73.05	71.10($\pm$1.21)	69.31($\pm$2.01)	-13.93	60.92($\pm$5.27)	59.29($\pm$0.87)	57.12($\pm$2.66)	-10.92
ARBRE NON ÉTIQUETÉ PARTIEL	82.98	77.93($\pm$3.53)	74.53($\pm$4.3)	-2.23	65.06($\pm$4.66)	63.92($\pm$1.05)	61.46($\pm$2.72)	-4.86
ARBRE NON ÉTIQUETÉ COMPLET	95.92	92.04($\pm$3.38)	88.37($\pm$4.41)	$+13.01$	80.62($\pm$10.86)	76.99($\pm$2.75)	73.47($\pm$4.04)	$+17.88$
ARBRE PRÉFIXE COMPLET	97.71	93.66($\pm$2.37)	91.54($\pm$2.78)	$+15.11$	**87.29**($\pm$4.66)	**84.56**($\pm$2.22)	**81.21**($\pm$3.45)	$+27.64$
ARBRE POSTFIXE COMPLET	**98.22**	**95.99**($\pm$1.75)	**93.05**($\pm$3.24)	$+15.72$	86.54($\pm$14.45)	81.27($\pm$2.65)	79.75($\pm$2.41)	$+26.54$
Gordon *et al.* (2020)	91.0 $\pm$ 27.4 (moyenne de 25 entraînements, médiane : 98.5)							
Russin *et al.* (2020b)	**99.1** $\pm$ 0.04 (moyenne de 5 entraînements, entraînement sur 200k itérations)							

TABLE 2: Exactitude ($\pm$ écart type) sur le corpus de test du split JUMP. ORIGINAL : données originales de Lake & Baroni (2018), voir la section 3 pour la description des autres configurations. *Top-1 tous modèles* : meilleure exactitude pour la meilleure graine aléatoire d'une configuration. *Top-1 tous modèles moyenné sur 3 graines aléatoires* : meilleure configuration où le score d'une configuration est calculée comme une moyenne des scores données par 3 graines aléatoires. Top *k* scores : score moyenné des *k* meilleurs modèles. La colonne Δ indique la variation relative entre le score *Top-1* d'un type d'input et le *Top-1* de l'input baseline.

Paramètres	ORIGINAL	CHUNKS-V	ARBRE NON ÉTIQUETÉ PARTIEL	ARBRE NON ÉTIQUETÉ COMPLET	ARBRE PRÉFIXE COMPLET	ARBRE POSTFIXE COMPLET
Tous modèles						
lr	0.01	0.01	0.01	0.01	0.01	0.01
bsz	50	100	100	50	50	50
layer	10	9	7	7	5	5
dim	512	512	512	512	512	512
kernel	3	3	5	3	5	3
dp	0.25	0.25	0.25	0.25	0.25	0.25
seed	2	3	3	3	2	1
Tous modèles, moyenné sur 3 graines aléatoires						
lr	0.1	0.1	0.01	0.01	0.01	0.01
bsz	500	500	100	50	50	50
layer	10	10	7	7	9	7
dim	256	128	512	512	512	512
kernel	5	5	3	3	3	3
dp	0.25	0.25	0.25	0.25	0.25	0.25

TABLE 3: Hyperparamètres des meilleures configurations sur le split JUMP. lr : pas d'apprentissage, bsz : taille des batches, layer : nombre de couches cachées, dim : dimension des vecteurs de mots, dp : probabilité de *dropout*.

les deux cas, nous présentons les scores pour les modèles qui sont dans le Top 1, Top 5 et Top 10 [7] ainsi que les écarts-types correspondants. Les différences de performances relatives par rapport aux modèles Top 1 entrainés sur ORIGINAL sont aussi présentées pour chacun des modèles entraînés avec différent niveaux d'annotations syntaxiques. Les hyperparamètres utilisés pour obtenir ces différents résultats sont présentés dans la Table 3.

Nous pouvons observer une amélioration de près de 16% sur les meilleurs scores et de près de 28% sur les scores moyennés lorsque nous utilisons des données syntaxiquement annotées. Que l'information soit présentée de manière préfixée ou suffixée (ARBRE PRÉFIXE COMPLET ou ARBRE POSTFIXE COMPLET) n'impacte pas significativement les résultats. On constate cependant une large différence dans les résultats obtenus avec ARBRE NON ÉTIQUETÉ COMPLET et ARBRE NON ÉTIQUETÉ PARTIEL, le premier étant meilleur que le second. Dans le premier cas, les règles unaires sont explicitement parenthésées alors que dans le second cas elles ne le sont pas. Parenthèser les règles unaires, qui sont utilisées pour générer l'ensemble des commandes primitives qui se comportent de la même manière (`run`, `jump`, `look`, `walk`) permet donc aux modèles de mieux généraliser.

7. 1 620 architectures en tout pour 4 860 modèles au total.

(a) Ts. les modèles FULL UNL. TREE

(b) Ts. les modèles (moy.) FULL UNL. TREE

(c) Ts. les modèles ARBRE POSTFIXE COMPLET

(d) Ts. les modèles (moy.) ARBRE POSTFIXE COMPLET

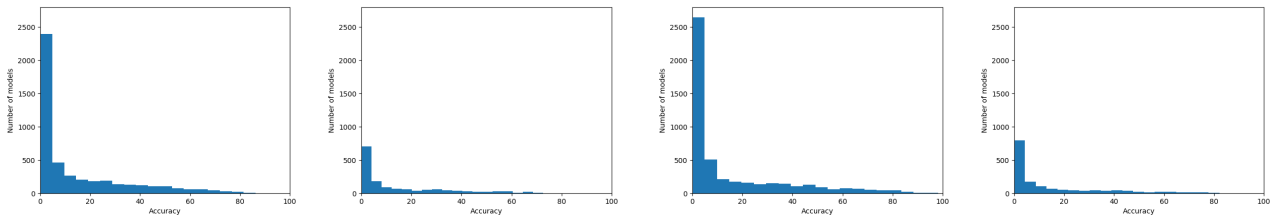

FIGURE 1: Histogrammes de l'exactitude sur les données de test de tous les modèles entrainées sur ARBRE NON ÉTIQUETÉ COMPLET sans faire la moyenne sur les différentes graines (a), en faisant la moyenne sur 3 graines (b) et pour ARBRE POSTFIXE COMPLET sans faire la moyenne sur les différentes graines (c), en faisant la moyenne sur 3 graines (d).

Ces résultats semblent indiquer que la tâche la plus ardue pour les modèles est d'isoler correctement les commandes primitives. En effet, les résultats chutent lorsque cette information n'est pas présente. En isolant un petit ensemble de commandes, le parenthésage rend l'analyse lexicale des tokens plus facile. Pour terminer, les résultats obtenus dans la configuration CHUNKS-V sont surprenamment inférieurs à ceux obtenus sur les données ORIGINAL, quand bien même une information sur la structure syntaxique, bien que minimale, soit présente. Cela peut s'expliquer par le fait que les modèles doivent traiter des séquences plus longues sans nécessairement avoir un gain proportionnel d'information pertinente en entrée.

Variation de graine. Nos résultats sont en accord avec les résultats de précédentes recherches. En effet, nous observons de fortes variations de performances lorsqu'une même architecture neuronale est entraînée avec différentes graines, comme le montrent les écarts types des meilleurs modèles (Top 1) de la Table 2. Nous présentons plus précisément comment cette variation se répartit dans différentes configurations syntaxiques dans la Figure 1. Cette figure montre la dispersion des résultats pour toutes les configurations d'hyperparamètres (en moyennant ou non sur 3 graines), pour ARBRE NON ÉTIQUETÉ COMPLET et ARBRE POSTFIXE COMPLET. Comme on peut le constater, la proportion de mauvais modèles (exactitude proche de 0) est très haute, aussi bien pour ARBRE NON ÉTIQUETÉ COMPLET que pour ARBRE POSTFIXE COMPLET. De plus, la proportion de modèles avec une exactitude supérieure à 50 décroît lorsque les résultats sont moyennés sur 3 graines. Cela explique la variance relativement haute que l'on peut observer dans la Table 2.

Perspectives architecturales. En analysant les configurations neuronales qui obtiennent les meilleurs scores, nous observons que celles-ci sont moins profondes lorsque de l'information syntaxique est présente (seulement besoin de 7 et 5 couches pour la meilleure graine et le meilleur modèle dans la configuration ARBRE POSTFIXE COMPLET), que lorsque cette information n'est pas présente (besoin de 9 et 10 couches pour obtenir les meilleurs performances pour ORIGINAL et pour les modèles entrainés sur des arbres syntatixques non-étiquettés). Ainsi, il semblerait que l'ajout d'informations syntaxiques permette d'entraîner des modèles moins profonds.

C'est d'autant plus remarquable que les séquences à traiter sont plus longues lorsqu'elles ont été annotées syntaxiquement (d'un facteur 2 pour CHUNKS-V et d'un facteur 4 pour les arbres complets) et que de longues séquences peuvent être fastidieuses à traiter pour des CNN avec un nombre limité de couches. Les meilleures performances que nous observons peuvent être dues au fait que les modèles

n'ont pas à inférer l'arbre syntaxique des phrases par eux-même et peuvent ainsi se concentrer sur l'apprentissage de la manière dont les différentes configurations syntaxiques affectent les sorties.

5 Conclusion

Nous avons étudié de quelle manière fournir des informations syntaxiques peut améliorer les capacités de généralisation compositionnelle de modèles séquence à séquence convolutifs. Pour ce faire, nous avons annoté les données du corpus SCAN avec différents niveaux d'informations syntaxiques, et avons évalué l'effet de telles annotations sur des architectures séquence à séquence convolutives standards. Nous avons observé que la structure brute de l'arbre syntaxique est l'information la plus importante pour permettre aux modèles neuronaux d'apprendre la compositionalité, l'étiquetage des syntagmes n'apportant que des gains de performances relatifs. De plus, le calibrage des hyper-paramètres trouve fréquement des modèles moins profonds lorsque ceux-ci traitent des données syntaxiquement annotées que lorsqu'ils traitent les données non-annotées. Cela suggère que les informations supplémentaires réduisent le besoin d'entrainer des modèles plus profonds, malgré l'augmentation de la longueur de la séquence d'entrée à traiter.

Dans de futurs travaux, nous souhaiterions travailler sur des partitions de données de SCAN plus difficiles comme la partition AROUND RIGHT (Loula *et al.*, 2018). Nous souhaiterions également étudier les capacités de généralisation d'autres architectures séquence à séquence, telles que celles proposées par Russin *et al.* (2020a,b) et Gordon *et al.* (2020), ainsi que sur des architectures basées sur Transformers (Vaswani *et al.*, 2017), telles que récemment étudiées sur un autre corpus par Hupkes *et al.* (2020).

6 Remerciements

Ce travail a été partiellement soutenu par l'Institut Multidisciplinaire en Intelligence Artificielle MIAI@Grenoble Alpes (ANR-19-P3IA-0003).

Références

BELINKOV Y. & BISK Y. (2018). Synthetic and natural noise both break neural machine translation. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*.

BIRD S., KLEIN E. & LOPER E. (2009). *Natural Language Processing with Python*. O'Reilly Media, Inc., 1st édition.

DESSI R. & BARONI M. (2019). CNNs found to jump around more skillfully than RNNs : Compositional generalization in seq2seq convolutional networks. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, p. 3919–3923.

GEHRING J., AULI M., GRANGIER D. & DAUPHIN Y. (2017). A convolutional encoder model for neural machine translation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1 : Long Papers)*, p. 123–135.

GORDON J., LOPEZ-PAZ D., BARONI M. & BOUCHACOURT D. (2020). Permutation equivariant models for compositional generalization in language. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*.

HUPKES D., DANKERS V., MUL M. & BRUNI E. (2020). Compositionality decomposed : How do neural networks generalise ? *Journal of Artificial Intelligence Research*, **67**, 757–795. DOI : 10.1613/jair.1.11674.

JIA R. & LIANG P. (2017). Adversarial examples for evaluating reading comprehension systems. In M. PALMER, R. HWA & S. RIEDEL, Éds., *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, EMNLP 2017, Copenhagen, Denmark, September 9-11, 2017*, p. 2021–2031 : Association for Computational Linguistics.

LAKE B. M. (2019). Compositional generalization through meta sequence-to-sequence learning. In H. WALLACH, H. LAROCHELLE, A. BEYGELZIMER, F. D' ALCHÉ-BUC, E. FOX & R. GARNETT, Éds., *Advances in Neural Information Processing Systems 32*, p. 9791–9801. Curran Associates, Inc.

LAKE B. M. & BARONI M. (2018). Generalization without systematicity : On the compositional skills of sequence-to-sequence recurrent networks. In J. G. DY & A. KRAUSE, Éds., *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 de *Proceedings of Machine Learning Research*, p. 2879–2888 : PMLR.

LIU Q., AN S., LOU J.-G., CHEN B., LIN Z., GAO Y., ZHOU B., ZHENG N. & ZHANG D. (2020). Compositional generalization by learning analytical expressions. In H. LAROCHELLE, M. RANZATO, R. HADSELL, M. F. BALCAN & H. LIN, Éds., *Advances in Neural Information Processing Systems*, volume 33, p. 11416–11427 : Curran Associates, Inc.

LOULA J., BARONI M. & LAKE B. (2018). Rearranging the familiar : Testing compositional generalization in recurrent networks. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP : Analyzing and Interpreting Neural Networks for NLP*, p. 108–114, Brussels, Belgium : Association for Computational Linguistics. DOI : 10.18653/v1/W18-5413.

OTT M., EDUNOV S., BAEVSKI A., FAN A., GROSS S., NG N., GRANGIER D. & AULI M. (2019). fairseq : A fast, extensible toolkit for sequence modeling. In *Proceedings of NAACL-HLT 2019 : Demonstrations*.

RUSSIN J., JO J., O'REILLY R. & BENGIO Y. (2020a). Compositional generalization by factorizing alignment and translation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics : Student Research Workshop*, p. 313–327, Online : Association for Computational Linguistics. DOI : 10.18653/v1/2020.acl-srw.42.

RUSSIN J., JO J., O'REILLY R. C. & BENGIO Y. (2020b). Systematicity in a recurrent neural network by factorizing syntax and semantics. In *Proceedings of the 42th Annual Meeting of the Cognitive Science Society, CogSci 2020, virtual, July 29 - August 1, 2020* : cognitivesciencesociety.org.

VASWANI A., SHAZEER N., PARMAR N., USZKOREIT J., JONES L., GOMEZ A. N., KAISER L. U. & POLOSUKHIN I. (2017). Attention is all you need. In I. GUYON, U. V. LUXBURG, S. BENGIO, H. WALLACH, R. FERGUS, S. VISHWANATHAN & R. GARNETT, Éds., *Advances in Neural Information Processing Systems 30*, p. 5998–6008 : Curran Associates, Inc.