CNNs found to jump around more skillfully than RNNs: Compositional generalization in seq2seq convolutional networks

Roberto Dessì
CIMeC, University of Trento
roberto.dessi@studenti.unitn.it

Marco Baroni
ICREA
Facebook AI Research
mbaroni@fb.com

Abstract

Lake and Baroni (2018) introduced the SCAN dataset probing the ability of seq2seq models to capture compositional generalizations, such as inferring the meaning of *"jump around"* 0-shot from the component words. Recurrent networks (RNNs) were found to completely fail the most challenging generalization cases. We test here a convolutional network (CNN) on these tasks, reporting hugely improved performance with respect to RNNs. Despite the big improvement, the CNN has however not induced systematic rules, suggesting that the difference between compositional and non-compositional behaviour is not clear-cut.

1 Introduction

Recent deep neural network successes rekindled classic debates on their natural language processing abilities (e.g., Kirov and Cotterell, 2018; McCoy et al., 2018; Pater, 2018). Lake and Baroni (2018) and Loula et al. (2018) proposed the SCAN challenge to directly assess the ability of sequence-to-sequence networks to perform systematic, compositional generalization of linguistic rules. Their results, and those of Bastings et al. (2018), have shown that modern recurrent networks (gated RNNs, such as LSTMs and GRUs) generalize well to new sequences that resemble those encountered in training, but achieve very low performance when generalization must be supported by a systematic compositional rule, such as "to X twice you X and X" (e.g., to jump twice, you jump and jump again).

Non-recurrent models, such as convolutional neural networks (CNNs, Kalchbrenner et al., 2016; Gehring et al., 2016, 2017) and self-attentive models (Vaswani et al., 2017; Chen et al., 2018) have recently reached comparable or better performance than RNNs on machine translation and other benchmarks. Their linguistic properties are however still generally poorly understood. Tang et al. (2018) have shown that RNNs and self-attentive models are better than CNNs at capturing long-distance agreement, while self-attentive networks excel at word sense disambiguation. In an extensive comparison, Bai et al. (2018) showed that CNNs generally outperform RNNs, although the differences were typically not huge. We evaluate here an out-of-the-box CNN on the most challenging SCAN tasks, and we uncover the surprising fact that *CNNs are dramatically better than RNNs at compositional generalization*. As they are more cumbersome to train, we leave testing of self-attentive networks to future work.

2 SCAN

SCAN studies compositionality in a simple command execution environment framed as a supervised sequence-to-sequence task. The neural network receives word sequences as input, and has to produce the correspondence action sequence. Examples are given in Table 1. Lake and Baroni (2018) originally introduced 4 train/test splits, of which we consider 2.[1] In the *random* split, the training set includes 80% of randomly selected distinct SCAN commands, with the remaining 20% in the test set. This requires generalization, as no test command is encountered in training, but there is no systematic difference between the commands in the two sets. In the *jump* split,

[1]We also tested our CNNs on SCAN's *length* split, where test commands require systematically longer actions than the training ones. Accuracy was near 0%, as the learned positional embeddings of our CNN architecture do not generalize beyond training lengths. We leave the investigation of more flexible positional encodings (as in, e.g., Vaswani et al., 2017) to future work. We also experimented with SCAN's *turn left* split, obtaining near-perfect generalization. As RNNs were already performing very well in this split, we focus in the paper on the more challenging *jump* case.

Split	Train Command	Test Command
random	*walk opposite left*; *turn left twice and look*	*walk and jump right twice*; *run and run thrice*
jump	*jump*; *turn left twice after look*	*turn left twice after jump*; *run twice and jump*
around-right	*jump around left*; *turn opposite right twice*	*walk around right*; *look around right and jump left*

Table 1: Training and test examples for the three splits used in our experiments.

the *jump* command is only seen in isolation during training, and the test set consists of all composite commands with *jump*. A system able to extract compositional rules (such as "*X twice* means to X and X") should have no problem generalizing them to a new verb, as in this split. Loula et al. (2018) proposed a set of new SCAN splits, the most challenging one being the *around-right* split. The training partition contains examples of *around* and *right*, but never in combination. The test set contains all possible *around right* commands. Loula and colleagues want to test "second-order modification", as models must learn how to compositionally apply the *around* function to *right*, which is in turn a first-order function modifying simple action verbs.

3 Experimental setup

Model We use the fully convolutional encoder-decoder model of Gehring et al. (2017) out of the box, using version 0.6.0 of the fairseq toolkit.[2] The model uses convolutional filters and Gated Linear Units (Dauphin et al., 2016) along with an attention mechanism that connects the encoder and the decoder. Attention is computed separately for each encoder layer, and produces weighted sums over encoder input embeddings and encoder outputs. See the original paper for details.

Training The shift in distribution between training and test splits makes SCAN unsuitable for validation-set tuning. Instead, following Lake and Baroni (2018) and Loula et al. (2018), we train on 100k random samples with replacement from the training command set. We explore different batch sizes (in terms of number of tokens per batch: 25, 50, 100, 200, 500, 1000), learning rates (0.1, 0.01, 0.001), layer dimensionalities (128, 256, 512), layer numbers (6 to 10), convolutional kernel width (3, 4, 5) and amount of dropout used (0, 0.25, 0.5). For all other hyperparameters, we accept recommended/default fairseq values. Each configuration is run with 5 seeds, and we report means and standard deviations.

[2] `https://github.com/pytorch/fairseq`

	random	jump	around-right
LSTM	99.8	1.2	2.5±2.7
GRU	**100.0**±0.0	12.5±6.6	–
CNN	**100.0**±0.0	**69.2**±8.2	**56.7**±10.2

Table 2: Test accuracy (%) on SCAN splits (means across 5 seeds, with standard deviation if available). Top LSTM results from Lake and Baroni (2018)/Loula et al. (2018), GRU from Bastings et al. (2018).

4 Results

Our main results are in Table 2. CNNs, like RNNs, succeed in the *random* split, and achieve much higher accuracy (albeit still far from being perfect) in the challenging *jump* and *around-right* splits.

The SCAN tasks should be easy for a system that learned the right composition rules. Perhaps, CNNs do not achieve 100% accuracy because they only learned a subset of the necessary rules. For example, they might correctly interpret the new expression *jump twice* because they induced a *X twice* rule at training time, but fail *jump thrice* because they missed the corresponding *X thrice* rule. Since SCAN semantic composition rules are associated with single words in input commands, we can check this hypothesis by looking at error distribution across input words. It turns out (Fig. 1) that errors are not associated to specific input commands. Error proportion is instead relatively stable across command words. Direct inspection reveals no traces of systematicity: errors cut across composition rules. Indeed, we often find minimal pairs in which changing one action verb with another (distributionally equivalent in SCAN) turns a correctly executed command into a failed one. For example, in the *jump* split, the CNN correctly executes *jump left after walk*, but fails *jump left after run* (jumping is forgotten). Analogously, in the *around-right* split, *run around right* is correctly executed, but "*walk around right*" is not (the network stops too early).

Robustness Fig. 2 shows a big difference in stability between *random* and the other splits across top hyperparameter configurations. The *random*

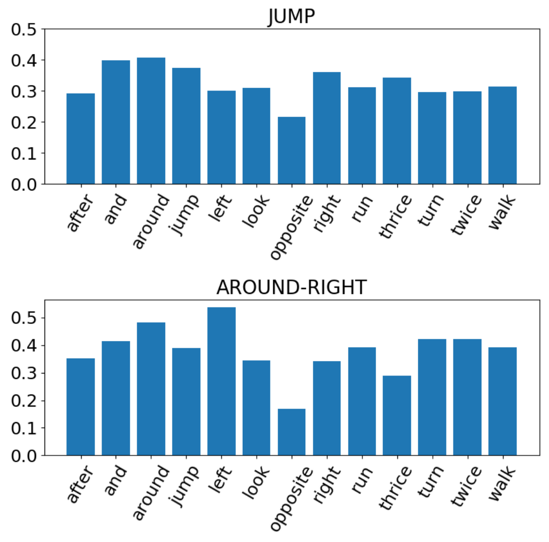

Figure 1: Proportion of commands with a certain command word (over total commands with that word) wrongly executed by best CNNs.

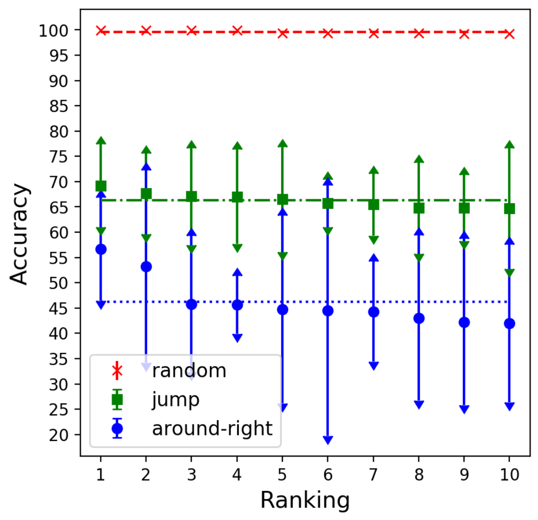

Figure 2: Accuracies (%) of top-10 models on *random*, *jump* and *around-right*. Arrows denote standard deviations, dashed lines average accuracy across top-10.

results are very stable. *Jump* accuracy is relatively stable across hyperparameters, but has large variance across initialization seeds. For the most challenging *around-right* split, we observe instability both across seeds and hyperparameters (although even the lowest end of the reported accuracies is well above best RNN performance in the corresponding experiments). Another question is whether the best configurations are shared, or each split requires an *ad-hoc* hyperparameter choice. We find that there are configurations that achieve good performance across the splits. In particular, the *best overall configuration*, found by minimizing ranks across splits, has 0.01 learning rate, 25-tokens batch size, 0.25 dropout, 6 layers, 512 layer dimensionality, and kernels of width 5. Such model was 13th best (of about 2.5K explored) on the *random* split (with mean cross-seed accuracy of 99.92%, off by 0.05% from top configuration), 32th on the *jump* split (60.67% mean accuracy, off by 8.62%), and 2nd in the *around-right* split (mean 53.25% accuracy, off by 3.45%).

Kernel width One important difference between recurrent and convolutional architectures is that CNN kernel width imposes a strong prior on the window of elements to be processed together. We conjecture that relatively wide encoder and decoder widths, by pushing the network to keep wider contexts into account, might favour the acquisition of template-based generalizations, and hence better compositionality. To investigate this, we varied encoder and decoder widths of the best-overall model between 1 and 5.[3]

Fig. 3 shows that the *random* split confirms our expectations, as both wider encoder and decoder windows improve performance. The *jump* results follow the same trend, although in a less clear-cut way. Still, the narrowest encoder-decoder combination has the worst performance, and the widest one the top one. For the *around-right* split, it is also better to use the widest encoder, but top performance is achieved with the *narrowest* decoder (width=1). Indeed, with the narrow decoder we obtain *around-right* accuracies that are even above the absolute-best *jump*-split performance. Since the novel output templates in the *around-right* split are by construction long (they involve executing an *around* command that requires repeating an action 4 times), we would have rather expected models keeping track of a larger decoding window to fare better, particularly in this case. We tried to gain some insight on the attested behaviour by looking at performance distribution in function of input and output length, failing to detect different patterns in the wide-decoder *jump* model vs. the narrow-decoder *around-right* model (analysis not reported here for space reasons). Looking qualitatively at the errors, we note that, for both splits, the narrower decoder tends to skip trajectory sub-chunks (e.g., executing "*jump around right*" with 3 instead of 4 right turns followed by

[3]At least on the encoder side, larger widths seem excessive, as the longest commands are 9-word-long.

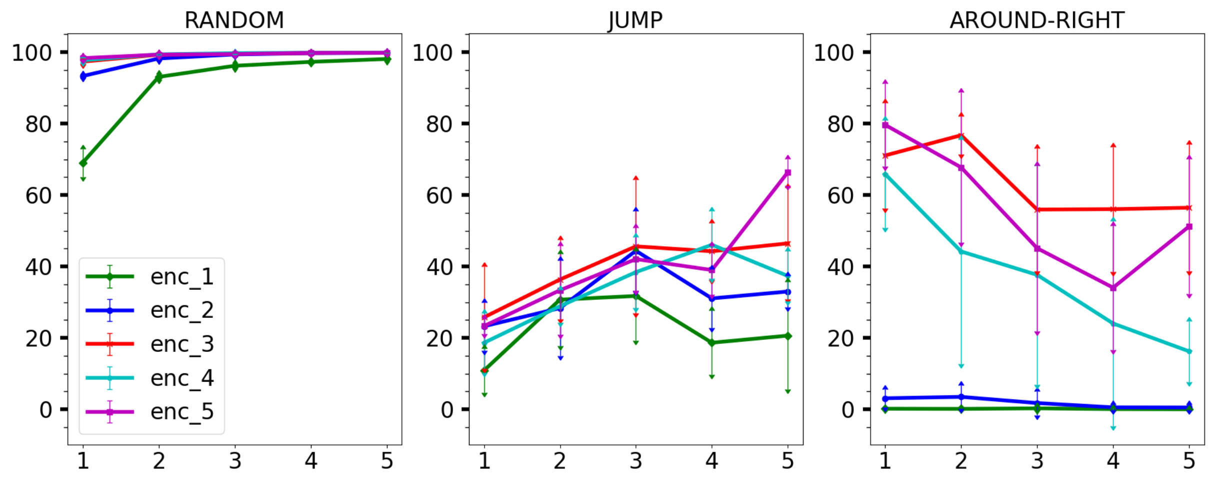

Figure 3: Mean accuracies (%) across 5 seeds, in function of decoder (x axis) and encoder (colors) kernel widths. Arrows denote standard deviations. Best viewed in color.

jumps), whereas the wider kernel is more likely to substitute actions (e.g., turning left instead of right) than undershooting the length. This impressionistic observation is supported by the fact that, for both splits, the narrow-kernel errors have considerably larger variance than the wide-kernel errors with respect to ground-truth length, indicating that, with narrow decoder kernel, the model is less stable in terms of output sequence length. This, however, only confirms our conjecture that a wider decoder kernel helps length management. We still have no insight on why the narrower kernel should be better on the *around-right split*.

Multi-layer attention The fairseq CNN has attention from all layers of the decoder. Is the possibility to focus on different aspects of the input while decoding from different layers crucial to its better generalization skills? Fig. 4 reports accuracies when applying attention from a subset of the 6 layers only. The *random* split differences are minimal, but ablating attentions greatly affects performance on the compositional splits (although, in both cases, there is a single ablated configuration that is as good as the full setup).

5 Conclusion

Compared to the RNNs previously tested in the literature, the out-of-the-box fairseq CNN architecture reaches dramatically better performance on the SCAN compositional generalization tasks. The CNN is however not learning rule-like compositional generalizations, as its mistakes are non-systematic and they are evenly spread across different commands. Thus, the CNN achieved a considerable degree of generalization, even on an explicitly compositional benchmark, without something akin to rule-based reasoning. Fully understanding generalization of deep seq2seq models might require a less clear-cut view of the divide between statistical pattern matching and symbolic composition. In future work, we would like to further our insights on the CNN aspects that are crucial for the task, our preliminary analyses of kernel width and attention.

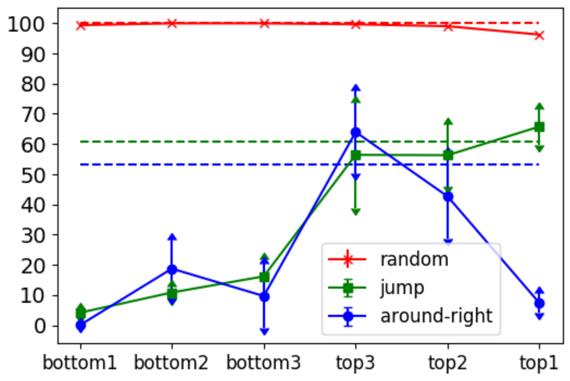

Figure 4: Accuracy (%) of overall-best model with attention only from first layer (*bottom1*), first two layers (*bottom2*), . . . , last two layers (*top2*), top layer only (*top1*). Means and standard deviations across 5 seeds. Dashed lines show full multi-layer attention results.

Concerning the comparison with RNNs, the best LSTM architecture of Lake and Baroni has two 200-dimensional layers, and it is consequently more parsimonious than our best CNN (1/4 of parameters). In informal experiments, we found shallow CNNs incapable to handle even the simplest *random* split. On the other hand, it is hard to train very deep LSTMs, and it is not clear that the latter models need the same depth CNNs require to "view" long sequences. We leave a proper formulation of a tighter comparison to future work.

Acknowledgements

We thank Brenden Lake, Michael Auli, Myle Ott, João Loula, Joost Bastings and the reviewers for comments and advice.

References

Shaojie Bai, Zico Kolter, and Vladlen Koltun. 2018. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. https://arxiv.org/abs/1803.01271.

Joost Bastings, Marco Baroni, Jason Weston, Kyunghyun Cho, and Douwe Kiela. 2018. Jump to better conclusions: SCAN both left and right. In *Proceedings of the EMNLP BlackboxNLP Workshop*, pages 47–55, Brussels, Belgium.

Mia Xu Chen, Orhan Firat, Ankur Bapna, Melvin Johnson, Wolfgang Macherey, George Foster, Llion Jones, Mike Schuster, Noam Shazeer, Niki Parmar, Ashish Vaswani, Jakob Uszkoreit, Lukasz Kaiser, Zhifeng Chen, Yonghui Wu, and Macduff Hughes. 2018. The best of both worlds: Combining recent advances in neural machine translation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 76–86. Association for Computational Linguistics.

Yann N. Dauphin, Angela Fan, Michael Auli, and David Grangier. 2016. Language modeling with gated convolutional networks. *CoRR*, abs/1612.08083.

Jonas Gehring, Michael Auli, David Grangier, and Yann N. Dauphin. 2016. A convolutional encoder model for neural machine translation. *CoRR*, abs/1611.02344.

Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann Dauphin. 2017. Convolutional sequence to sequence learning. In *Proceedings of ICML*, pages 1243–1252, Sydney, Australia.

Nal Kalchbrenner, Lasse Espeholt, Karen Simonyan, Aäron van den Oord, Alex Graves, and Koray Kavukcuoglu. 2016. Neural machine translation in linear time. *CoRR*, abs/1610.10099.

Christo Kirov and Ryan Cotterell. 2018. Recurrent neural networks in linguistic theory: Revisiting Pinker and Prince (1988) and the past tense debate. *Transactions of the Association for Computational Linguistics*. In press.

Brenden Lake and Marco Baroni. 2018. Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks. In *Proceedings of ICML*, pages 2879–2888, Stockholm, Sweden.

Joao Loula, Marco Baroni, and Brenden Lake. 2018. Rearranging the familiar: Testing compositional generalization in recurrent networks. In *Proceedings of the EMNLP BlackboxNLP Workshop*, pages 108–114, Brussels, Belgium.

Thomas McCoy, Robert Frank, and Tal Linzen. 2018. Revisiting the poverty of the stimulus: Hierarchical generalization without a hierarchical bias in recurrent neural networks. In *Proceedings of CogSci*, pages 2093–2098, Madison, WI.

Joe Pater. 2018. Generative linguistics and neural networks at 60: Foundation, friction, and fusion. *Language*. In press.

Gongbo Tang, Mathias Müller, Annette Rios, and Rico Sennrich. 2018. Why self-attention? A targeted evaluation of neural machine translation architectures. In *Proceedings of EMNLP*, pages 4263–4272, Brussels, Belgium.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008.