Discovering the compositional structure of vector representations with Role Learning Networks

Paul Soulos,[1] R. Thomas McCoy,[1] Tal Linzen,[1], & Paul Smolensky[2,1]
[1]Department of Cognitive Science, Johns Hopkins University
[2]Microsoft Research AI, Redmond, WA USA
{psoulos1,tom.mccoy,tal.linzen,smolensky}@jhu.edu

Abstract

Neural networks are able to perform tasks that rely on compositional structure even though they lack obvious mechanisms for representing this structure. To analyze the internal representations that enable such success, we propose ROLE, a technique that detects whether these representations implicitly encode symbolic structure. ROLE learns to approximate the representations of a target encoder $\mathcal{E}$ by learning a symbolic constituent structure and an embedding of that structure into $\mathcal{E}$'s representational vector space. The constituents of the approximating symbol structure are defined by structural positions — roles — that can be filled by symbols. We show that when $\mathcal{E}$ is constructed to explicitly embed a particular type of structure (string or tree), ROLE successfully extracts the ground-truth roles defining that structure. We then analyze a GRU seq2seq network trained to perform a more complex compositional task (SCAN), where there is no ground truth role scheme available. For this model, ROLE successfully discovers an interpretable symbolic structure that the model implicitly uses to perform the SCAN task, providing a comprehensive account of the representations that drive the behavior of a frequently-used but hard-to-interpret type of model. We verify the causal importance of the discovered symbolic structure by showing that, when we systematically manipulate hidden embeddings based on this symbolic structure, the model's resulting output is changed in the way predicted by our analysis.

1 Overview

Certain AI tasks consist in computing a function φ that is governed by strict rules: e.g., if φ is the function mapping a mathematical expression to its value (e.g., mapping '$19-2*7$' to 5), then φ obeys the rule that $\varphi(x+y) = \texttt{sum}(\varphi(x), \varphi(y))$ for any expressions x and y. This rule is **compositional**: the output of a structure (here, $x+y$) is a function of the outputs of the structure's constituents (here, x and y). For a **fully-compositional** task, completely determined by compositional rules, an AI system that can assign appropriate symbolic structures to inputs and apply appropriate compositional rules to these structures will correctly process arbitrary novel combinations of familiar constituents. This is a core capability of symbolic AI systems. Other tasks, including most natural language tasks such as machine translation, are only partially characterizable by compositional rules because natural language is only partially compositional in nature. On these "**partially-compositional**" AI tasks, this strategy of compositional analysis has demonstrated considerable, but limited, generalization.

Deep learning research has shown that Neural Networks (NNs) often surpass symbolic AI systems for partially-compositional tasks [21], and exhibit good generalization (although generally falling short of symbolic AI systems) on fully-compositional tasks [11, 12]. Given that standard NNs have no obvious mechanisms for representing symbolic structures, parsing inputs into such structures, nor applying compositional symbolic rules to them, this success raises the question we address in this

33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada.

paper: *How do NNs achieve such strong generalization on partially-compositional tasks, and good performance on fully-compositional tasks?*

An important step towards answering this question was reported in McCoy et al. [12], which showed that when trained on highly compositional tasks, standard NNs learned representations that were well approximated by symbolic structures (Sec. 2). We refer to the networks to be analyzed as **target NNs**, because we will propose a new type of NN (in Sec. 3) — the **Role Learner (ROLE)** — which is used to *analyze* the target network. In contrast with the analysis model of McCoy et al. [12], which relies on a hand-specified hypothesis about the underlying structure, ROLE *automatically* learns a symbolic structure that best approximates the internal representation of the target network. Automating the discovery of structural hypotheses provides two advantages. First, ROLE achieves success at analyzing networks for which it is not clear what the underlying structure is. We show this in Sec. 5, where ROLE successfully uncovers the symbolic structures learned by a seq2seq RNN trained on the SCAN task [11]. Second, removing the need for hand-specified hypotheses allows the data to speak for itself, which simplifies the burden on the user, who only needs to provide input sequences and associated embeddings.

2 NN embedding of symbol structures

We build on McCoy et al. [12], which introduced the analysis task **DISCOVER (DISsecting COmpositionality in VEctor Representations)**: take a NN and, to the extent possible, find an explicitly-compositional approximation to its internal distributed representations. [12] showed that, in GRU [4] encoder-decoder networks performing simple, fully-compositional string manipulations, the medial encoding (between encoder and decoder) could be extremely well approximated, up to a linear transformation, by **Tensor Product Representations (TPRs)** [18], which are explicitly-compositional vector embeddings of symbolic structures. To represent a string of symbols as a TPR, the symbols in the string 337 might be parsed into three constituents $\{3:\mathrm{pos}1, 7:\mathrm{pos}3, 3:\mathrm{pos}2\}$, where $\mathrm{pos}n$ is the role of n^{th} position from the left edge of the string; other role schemes are also possible, such as roles denoting right-to-left position. The embedding of a constituent $7:\mathrm{pos}3$ is $\mathbf{e}(7:\mathrm{pos}3) = \mathbf{e}_{\mathrm{F}}(7) \otimes \mathbf{e}_{\mathrm{R}}(\mathrm{pos}3)$, where $\mathbf{e}_{\mathrm{R}}, \mathbf{e}_{\mathrm{F}}$ are respectively a vector embedding of the roles and a vector embedding of the **fillers** of those roles: the digits. The embedding of the whole string is the sum of the embeddings of its constituents. In general, for a symbol structure $\mathtt{S}$ with roles $\{r_k\}$ that are respectively filled by the symbols $\{\mathtt{f}_k\}$, $\mathbf{e}_{\mathrm{TPR}}(\mathtt{S}) = \sum_k \mathbf{e}_{\mathrm{F}}(\mathtt{f}_k) \otimes \mathbf{e}_{\mathrm{R}}(r_k)$.

The main technical contribution of the present paper is the Role Learner (ROLE) model, a RNN network that learns its own role scheme to optimize the fit of a TPR approximation to a given set of internal representations in a pre-trained target NN. This makes the DISCOVER framework more general by removing the need for human-generated hypotheses as to the role schemes the network might be implementing. Learned role schemes, we will see in Sec. 5.1, can enable good TPR approximation of networks for which human-generated role schemes fail.

This work falls within the larger paradigm of using analysis techniques to interpret neural networks (see [3] for a recent survey), often including a focus on compositional structure [9, 8, 11, 6]. Most closely related are the approaches of [2], [5], and [1], who also propose methods for uncovering symbolic structure in vector representations, and [14] and [20], which also seek to understand neural networks by extracting more interpretable symbolic models that approximate their behavior.

3 The Role Learner (ROLE) Model

ROLE[1] produces an embedding of an input string $\mathtt{S}$ by producing a TPR $\mathbf{T}(\mathtt{S})$ and then applying a linear transformation $\mathbb{W}$. ROLE is trained to approximate a pre-trained target string-encoder $\mathcal{E}$. Given a set of N training strings $\{\mathtt{S}^{(1)}, \ldots, \mathtt{S}^{(N)}\}$, ROLE minimizes the total mean-squared error (MSE) between its output $\mathbb{W}\,\mathbf{T}(\mathtt{S}^{(i)})$ and $\mathcal{E}$'s corresponding output, $\mathcal{E}(\mathtt{S}^{(i)})$.

ROLE is an extension of the Tensor-Product Encoder (TPE) introduced in McCoy et al. [12] , which produces a linearly-transformed TPR given a string of symbols and pre-assigned role labels for each symbol (see Appendix A.1 for details). Crucially, ROLE is not *given* role labels for the input symbols, but *learns to compute* them. More precisely, it learns a dictionary of n_{R} d_{R}-dimensional

[1] Code available at https://github.com/psoulos/role-decomposition

role-embedding vectors, $\boldsymbol{R} \in \mathbb{R}^{d_{\mathrm{R}} \times n_{\mathrm{R}}}$, and, for each input symbol $\mathtt{s}_t$, computes a soft-attention vector $\boldsymbol{a}_t$ over these role vectors: the role vector assigned to $\mathtt{s}_t$ is then the attention-weighted linear combination of role vectors, $\boldsymbol{r}_t = \boldsymbol{R}\,\boldsymbol{a}_t$.

ROLE uses an LSTM [7] to compute the role-assigning attention-vectors $\boldsymbol{a}_t$ from its learned embedding $\boldsymbol{F}$ of the input symbols $\mathtt{s}_t$: at each t, the hidden state of the LSTM passes through a linear layer and then a softmax to produce $\boldsymbol{a}_t$ (depicted in Appendix A.2). Since a TPR for a discrete symbol structure deploys a discrete set of roles specifying discrete structural positions, ideally a single role would be selected for each $\mathtt{s}_t$: $\boldsymbol{a}_t$ would be one-hot. We add regularization similar to that in [15] to encourage one-hot vecotrs (Appendix A.3).

4 A simple fully-compositional task

We first apply ROLE to two target Tensor Product Encoder (TPE) models which are fully compositional by design. Since we know what role scheme each target model deploys, we can test how well ROLE learns these ground-truth roles. The TPEs are trained on the fully compositional task of autoencoding sequences of digits. We use two types of TPEs: one that uses a simple left-to-right role scheme and one that uses a complex tree position role scheme (see [12] for explanations of the various role schemes). We analyze the performance of ROLE in two ways. **Substitution Accuracy** is the probability that the decoder produces the correct output string when it is fed the ROLE approximation. The **V-Measure** [17] assesses the extent to which the clustering of the role vectors assigned by ROLE matches the ground truth role assignments.

The ROLE approximation of the left-to-right TPE attained perfect performance, with a substitution accuracy of 100% and a V-Measure of 1.0, indicating that the role scheme it learned perfectly matched the ground truth. On the significantly more complex case of tree position roles, ROLE achieves essentially the same accuracy as the target encoder $\mathcal{E}$ and has considerable success at recovering the ground truth roles for the vectors it was analyzing (target accuracy = 98.62%, substitution accuracy = 98.61%, V-Measure = 0.815). These results show that, when a target model has a known fully compositional structure, ROLE can successfully find that structure.

5 The SCAN task

We have established that ROLE can uncover the compositional structure used by a model that is compositional by design. But how can models *without* explicit compositional structure still be as successful at fully compositional tasks as fully compositional models? Our hypothesis is that, though these models have no *constraint* forcing them to be compositional, they still have the *ability* to implicitly learn compositional structure. To test this hypothesis, we apply ROLE to a standard RNN-based seq2seq [19] model trained on a fully compositional task. Because the RNN has no constraint forcing it to use TPRs, we do not know *a priori* whether there exists any solution.

We consider the SCAN task [11], which was designed to test compositional generalization and systematicity. SCAN is a synthetic sequence-to-sequence mapping task, with an input sequence describing an action plan, e.g., `jump opposite left`, being mapped to a sequence of primitive actions, e.g., `TL TL JUMP` (see Sec. 5.2 for a complex example). We use `TL` to abbreviate `TURN_LEFT`, sometimes written `LTURN`; similarly, we use `TR` for `TURN_RIGHT`. The SCAN mapping is defined by a complete set of compositional rules [11, Supplementary Fig. 7].

5.1 The compositional structure of SCAN encoder representations

For our target SCAN encoder $\mathcal{E}$, we trained a standard GRU (see Appendix A.4 for details). $\mathcal{E}$ achieves 98.47% (full-string) accuracy on the test set. We provide ROLE with 50 roles to use as it wants (additional training information is in Appendix A.5). We evaluate the substitution accuracy that this learned role scheme provides in three ways. The **continuous** method tests ROLE in the same way as it was trained, with input symbol s_t assigned role vector $\boldsymbol{r}_t = \boldsymbol{R}\,\boldsymbol{a}_t$. In the **snapped** method, $\boldsymbol{a}_t$ is replaced at evaluation time by the one-hot vector $\boldsymbol{m}_t$ singling out role $m_t = \arg\max(\boldsymbol{a}_t)$: $\boldsymbol{r}_t = \boldsymbol{R}\,\boldsymbol{m}_t$. Our final evaluation method, the **discrete** method, uses discrete roles without having such a train/test discrepancy; in this method, we use the one-hot vector $\boldsymbol{m}_t$ to output roles for every symbol in the dataset and then train a TPE which uses the one-hot vector $\boldsymbol{m}_t$ as input during training.

Table 1: Mean substitution accuracy for learned (bold) and pre-defined role schemes on SCAN across three random initializations. Standard deviation was below 1% for all schemes except for snapped.

Continuous	**Snapped**	**Discrete**	LTR	RTL	Bi	Tree	Wickel	BOW
94.83%	81.71% ±7.28	92.44%	6.68%	6.96%	10.72%	4.31%	44.00%	4.52%

The mean substitution accuracy for various learned and predefined role schemes is shown in Table 1. All of the predefined role schemes provide poor approximations, none surpassing 44.00% accuracy. The role scheme learned by ROLE does significantly better than any of the predefined role schemes. The success of ROLE shows that the target model's compositional behavior relies on compositional internal representations: it was by no means guaranteed to be the case that ROLE would be successful here, so the fact that it is successful tells us that the encoder has learned compositional representations.

5.2 Precision constituent-surgery on internal representations to produce desired outputs

The substitution-accuracy results above show that if the *entire* learned representation is replaced by ROLE's approximation, the output remains correct. But do the *individual words* in this TPR have the appropriate causal consequences when processed by the decoder? To address this causal question [16], we actively intervene on the constituent structure of the internal representations by replacing one constituent with another syntactically equivalent one, and see whether this produces the expected change in the output of the decoder. We take the encoding generated by the RNN encoder $\mathcal{E}$ for an input such as `jump opposite left`, subtract the vector embedding of the `opposite` constituent, add the embedding of the `around` constituent, and see whether this causes the output to change from the correct output for `jump opposite left (TL TL JUMP)` to the correct output for `jump around left (TL JUMP TL JUMP TL JUMP TL JUMP)`. The roles in these constituents are determined by the algorithm described in Appendix A.6. If changing a word leads other roles in the sequence to change (according to the algorithm), we update the encoding with those new roles as well. Such surgery can be viewed as based in a more general extension of the analogy approach used by Mikolov et al. [13] for analysis of word embeddings. An example of applying a sequence of two such constituent surgeries to a sequence are shown in Figure 1 (left).

The proportion of cases for which a random sequence of k such successive surgeries produced the correct output at each step is shown in Figure 1 (right). The baseline of 0 substitutions shows the encoder's accuracy of 98.5%. The accuracy stays above 83% for any number of successive surgeries.

6 Conclusion

We have introduced ROLE, a neural network that learns to approximate the representations of an existing target neural network $\mathcal{E}$ using an explicit symbolic structure. ROLE successfully discovers symbolic structure both in models that explicitly define this structure and in an RNN without explicit structure trained on the fully-compositional SCAN task. Uncovering the latent symbolic structure of NN representations on fully-compositional tasks is a significant step towards explaining how they can achieve the level of compositional generalization that they do, and in future work we plan to use ROLE to impart a bias for compositionality in models trained on partially compositional tasks.

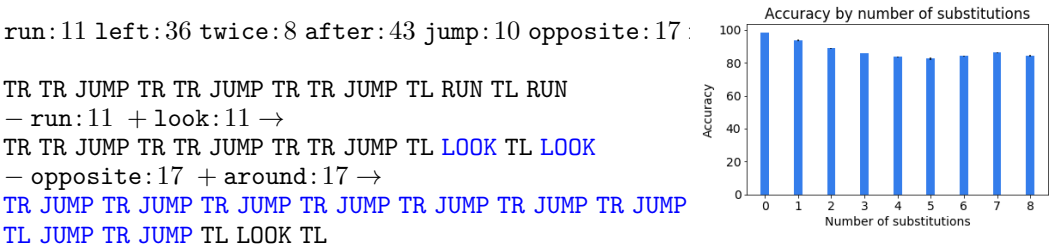

Figure 1: Left: Example of two successive constituent surgeries. Altered output symbols are in blue. Right: Mean constituent-surgery accuracy across three runs.

7 Acknowledgments

This material is based upon work supported by the National Science Foundation Graduate Research Fellowship Program under Grant No. 1746891, and work partially supported by NSF grant BCS-1344269. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

For helpful comments we are grateful to the members of the Johns Hopkins Neurosymbolic Computation group and the Microsoft Research AI Deep Learning Group. Any errors remain our own.

References

[1] Samira Abnar, Lisa Beinborn, Rochelle Choenni, and Willem Zuidema. Blackbox meets blackbox: Representational similarity & stability analysis of neural language models and brains. In *Proceedings of the 2019 ACL Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 191–203, Florence, Italy, August 2019. Association for Computational Linguistics. doi: 10.18653/v1/W19-4820. URL https://www.aclweb.org/anthology/W19-4820.

[2] Jacob Andreas. Measuring compositionality in representation learning. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=HJz05o0qK7.

[3] Yonatan Belinkov and James Glass. Analysis methods in neural language processing: A survey. *Transactions of the Association for Computational Linguistics*, 7:49–72, 2019.

[4] Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar, October 2014. Association for Computational Linguistics. doi: 10.3115/v1/D14-1179. URL https://www.aclweb.org/anthology/D14-1179.

[5] Grzegorz Chrupała and Afra Alishahi. Correlating neural and symbolic representations of language. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2952–2962, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1283. URL https://www.aclweb.org/anthology/P19-1283.

[6] John Hewitt and Christopher D Manning. A structural probe for finding syntax in word representations. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4129–4138, 2019.

[7] Sepp Hochreiter and Jürgen Schmidhuber. 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.

[8] Dieuwke Hupkes, Sara Veldhoen, and Willem Zuidema. Visualisation and'diagnostic classifiers' reveal how recurrent and recursive neural networks process hierarchical structure. *Journal of Artificial Intelligence Research*, 61:907–926, 2018.

[9] Dieuwke Hupkes, Verna Dankers, Mathijs Mul, and Elia Bruni. The compositionality of neural networks: integrating symbolism and connectionism. *arXiv preprint arXiv:1908.08351*, 2019.

[10] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference for Learning Representations*, 2015.

[11] Brenden M. Lake and Marco Baroni. Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks. In *ICML*, 2018. arXiv 1711.00350v3.

[12] R. Thomas McCoy, Tal Linzen, Ewan Dunbar, and Paul Smolensky. RNNs implicitly implement tensor-product representations. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=BJx0sjC5FX.

[13] Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig. Linguistic regularities in continuous space word representations. In *Proceedings of NAACL-HLT*, pages 746–751, 2013.

[14] Christian W Omlin and C Lee Giles. Extraction of rules from discrete-time recurrent neural networks. *Neural networks*, 9(1):41–52, 1996.

[15] Hamid Palangi, Paul Smolensky, Xiaodong He, and Li Deng. Question-answering with grammatically-interpretable representations. In *AAAI*, 2017.

[16] Judea Pearl. *Causality*. MIT Press, Cambridge, MA, 2000.

[17] Andrew Rosenberg and Julia Hirschberg. V-measure: A conditional entropy-based external cluster evaluation measure. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning(EMNLP-CoNLL)*, pages 410–420, 2007.

[18] P. Smolensky. Tensor product variable binding and the representation of symbolic structures in connectionist systems. *Artif. Intell.*, 46(1-2):159–216, November 1990. ISSN 0004-3702. doi: 10.1016/0004-3702(90)90007-M. URL `http://dx.doi.org/10.1016/0004-3702(90)90007-M`.

[19] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems*, pages 3104–3112, 2014.

[20] Gail Weiss, Yoav Goldberg, and Eran Yahav. Extracting automata from recurrent neural networks using queries and counterexamples. In *ICML*, pages 5244–5253, 2018.

[21] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Lukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. Google's neural machine translation system: Bridging the gap between human and machine translation. *CoRR*, abs/1609.08144, 2016. URL `http://arxiv.org/abs/1609.08144`.

A Appendix

A.1 Tensor Product Encoder (TPE) Architecture

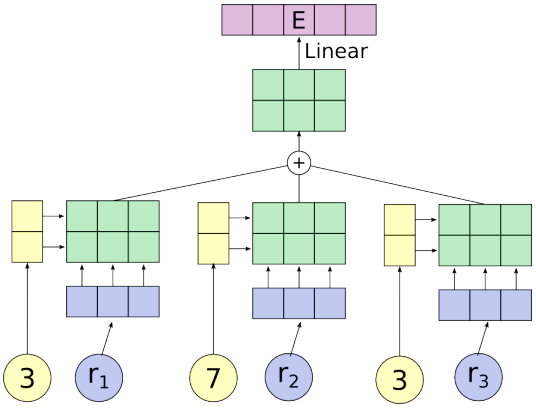

Figure 2: The Tensor Product Encoder architecture. The yellow circle is an embedding layer for the fillers, and the blue circle is an embedding layer for the roles. These two vector embeddings are combined by an outer product to produce the green matrix representing the TPR of the constituent. All of the constituents are summed together to produce the TPR of the sequence, and then a linear transformation is applied to resize the TPR to the target encoders dimensionality. ROLE replaces the role embedding layer and directly produces the blue role vector.

A.2 The Role Learner (ROLE) Architecture

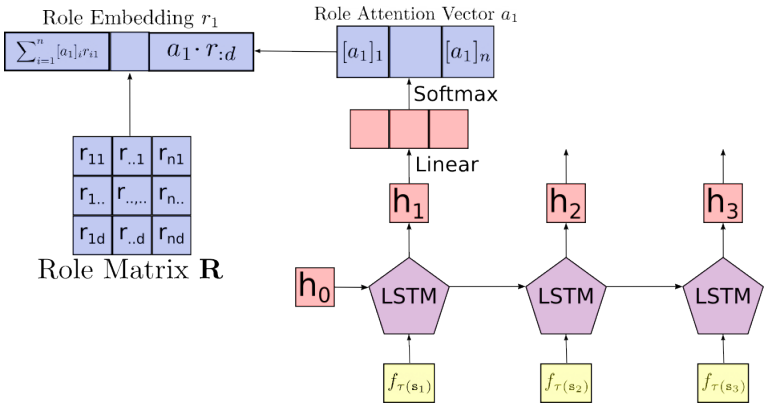

Figure 3: The role learning module. The role attention vector a_t is encouraged to be one-hot through regularization; if a_t were one-hot, the produced role embedding r_t would correspond directly to one of the roles defined in the role matrix $\boldsymbol{R}$. The LSTM can be unidirectional or bidirectional.

A.3 ROLE regularization

Letting $\boldsymbol{A} = \{\boldsymbol{a}_t\}_{t=1}^{T}$, the regularization term applied during ROLE training is $R = \lambda(R_1 + R_2 + R_3)$, where λ is a regularization hyperparameter and:

$$R_1(\boldsymbol{A}) = \sum_{t=1}^{T}\sum_{\rho=1}^{n_\mathrm{R}}[\boldsymbol{a}_t]_\rho(1-[\boldsymbol{a}_t]_\rho);\;\; R_2(\boldsymbol{A}) = -\sum_{t=1}^{T}\sum_{\rho=1}^{n_\mathrm{R}}[\boldsymbol{a}_t]_\rho^2;\;\; R_3(\boldsymbol{A}) = \sum_{\rho=1}^{n_\mathrm{R}}\left([\boldsymbol{s}_{\boldsymbol{A}}]_\rho(1-[\boldsymbol{s}_{\boldsymbol{A}}]_\rho)\right)^2$$

Since each $\boldsymbol{a}_t$ results from a softmax, its elements are positive and sum to 1. Thus the factors in $R_1(\boldsymbol{A})$ are all non-negative, so R_1 assumes its minimal value of 0 when each $\boldsymbol{a}_t$ has binary elements; since these elements must sum to 1, such an $\boldsymbol{a}_t$ must be one-hot. $R_2(\boldsymbol{A})$ is also minimized when each

$\boldsymbol{a}_t$ is one-hot because when a vector's L^1 norm is 1, its L^2 norm is maximized when it is one-hot. Although each of these terms individually favor one-hot vectors, empirically we find that using both terms helps the training process. In a discrete symbolic structure, each position can hold at most one symbol, and the final term R_3 in ROLE's regularizer R is designed to encourage this. In the vector $\boldsymbol{s_A} = \sum_{t=1}^{T} \boldsymbol{a}_t$, the ρ^{th} element is the total attention weight, over all symbols in the string, assigned to the ρ^{th} role: in the discrete case, this must be 0 (if no symbol is assigned this role) or 1 (if a single symbol is assigned this role). Thus R_3 is minimized when all elements of $\boldsymbol{s}$ are 0 or 1 (R_3 is similar to R_1, but with squared terms since we are no longer assured each element is at most 1). It is important to normalize each role embedding in the role matrix $\mathbf{R}$ so that small attention weights have correspondingly small impacts on the weighted-sum role embedding.

A.4 RNN trained on SCAN

To train the standard RNN on SCAN, we ran a limited hyperparameter search similar to the procedure in Lake and Baroni [11]. Since our goal was to produce a single embedding that captured the entire input sequence, we fixed the architecture to GRU with a single hidden layer. We did not train models with attention, since we wanted to investigate whether a standard RNN could capture compositionality. The remaining hyperparameters were hidden dimension and dropout. We ran a search over the hidden dimension sizes of 50, 100, 200, and 400 as well as dropout with a value of 0, .1, and .5 applied to the word embeddings and recurrent layer. Each network was trained with the ADAM optimizer [10] and a learning rate of .001 for 100,000 steps with a batch-size of 1. The best performing network had a hidden dimension or 100 and dropout of .1.

A.5 ROLE trained on SCAN

For the ROLE models trained to approximate the GRU encoder trained on SCAN, we used a filler dimension of 100, a role dimension of 50 with 50 roles available. For training, we used the ADAM [10] optimizer with a learning rate of .001, batch size 32, and an early stopping patience of 10. The role assignment module used a bidirectional 2-layer LSTM [7]. We performed a hyperparameter search over the regularization coefficient λ using the values in the set [.1, .02, .01]. The best performing value was .02, and we used this model in our analysis.

A.6 SCAN Role Analysis

The algorithm below characterizes our post-hoc interpretation of which roles the Role Learner will assign to elements of the input to the SCAN model. This algorithm was created by hand based on an analysis of the Role Learner's outputs for the elements of the SCAN training set. The algorithm works equally well on examples in the training set and the test set; on both datasets, it exactly matches the roles chosen by the Role Learner for 98.7% of sequences (20,642 out of 20,910).[2]

A.6.1 A role-assignment algorithm implicitly learned by the SCAN seq2seq encoder

The input sequences have three basic types that are relevant to determining the role assignment: sequences that contain *and* (e.g., *jump around left and walk thrice*), sequences that contain *after* (e.g., *jump around left after walk thrice*), and sequences without *and* or *after* (e.g., *turn opposite right thrice*). Within commands containing *and* or *after*, it is convenient to break the command down into the command before the connecting word and the command after it; for example, in the command *jump around left after walk thrice*, these two components would be *jump around left* and *walk thrice*.

- Sequence with *and*:
  - Elements of the command before *and*:
    - Last word: 28
    - First word (if not also last word): 46
    - *opposite* if the command ends with *thrice*: 22
    - Direction word between *opposite* and *thrice*: 2

[2]This figure of 98.7% is so constant across datasets presumably because the synthetic nature of the SCAN dataset means that any reasonably-sized sample from it will be similarly representative of the entire dataset.

        - *opposite* if the command does not end with *thrice*: 2
        - Direction word after *opposite* but not before *thrice*: 4
        - *around*: 22
        - Direction word after *around*: 2
        - Direction word between an action word and *twice* or *thrice*: 2
    - Elements of the command before *and*:
        - First word: 11
        - Last word (if not also the first word): 36
        - Second-to-last word (if not also the first word): 3
        - Second of four words: 24
    - *and*: 30
- Sequence with *after*:
    - Elements of the command before *after*:
        - Last word: 8
        - Second-to-last word: 36
        - First word (if not the last or second-to-last word): 11
        - Second word (if not the last or second-to-last word): 3
    - Elements of the command after *after*:
        - Last word: 46
        - Second-to-last word: 4
        - First word if the command ends with *around right*: 4
        - First word if the command ends with *thrice* and contains a rotation: 10
        - First word if the command does not end with *around right* and does not contain both *thrice* and a rotation: 17
        - Second word if the command ends with *thrice*: 17
        - Second word if the command does not end with *thrice*: 10
    - *after*: 17 if no other word has role 17 or if the command after *after* ends with *around left*; 43 otherwise
- Sequence without *and* or *after*:
    - Action word directly before a cardinality: 4
    - Action word before, but not directly before, a cardinality: 34
    - *thrice* directly after an action word: 2
    - *twice* directly after an action word: 2
    - *opposite* in a sequence ending with *twice*: 8
    - *opposite* in a sequence ending with *thrice*: 34
    - *around* in a sequence ending with a cardinality: 22
    - Direction word directly before a cardinality: 2
    - Action word in a sequence without a cardinality: 46
    - *opposite* in a sequence without a cardinality: 2
    - Direction after *opposite* in a sequence without a cardinality: 26
    - *around* in a sequence without a cardinality: 3
    - Direction after *around* in a sequence without a cardinality: 22
    - Direction directly after an action in a sequence without a cardinality: 22

To show how this works with an example, consider the input *jump around left after walk thrice*. The command before *after* is *jump around left*. *left*, as the last word, is given role 8. *around*, as the second-to-last word, gets role 36. *jump*, as a first word that is not also the last or second-to-last word gets role 11. The command after *after* is *walk thrice*. *thrice*, as the last word, gets role 46. *walk*, as the second-to-last word, gets role 4. Finally, *after* gets role 17 because no other elements have been assigned role 17 yet. These predicted outputs match those given by the Role Learner.

A.6.2 Discussion of the algorithm

We offer several observations about this algorithm.

1. This algorithm may seem convoluted, but a few observations can illuminate how the roles assigned by such an algorithm support success on the SCAN task. First, a sequence will contain role 30 if and only if it contains *and*, and it will contain role 17 if and only if it contains *after*. Thus, by implicitly checking for the presence of these two roles (regardless of the fillers bound to them), the decoder can tell whether the output involves one or two basic commands, where the presence of *and* or *after* leads to two basic commands and the absence of both leads to one basic command. Moreover, if there are two basic commands, whether it is role 17 or role 30 that is present can tell the decoder whether the input order of these commands also corresponds to their output order (when it is *and* in play, i.e., role 30), or if the input order is reversed (when it is *after* in play, i.e., role 17).

   With these basic structural facts established, the decoder can begin to decode the specific commands. For example, if the input is a sequence with *after*, it can begin with the command after *after*, which it can decode by checking which fillers are bound to the relevant roles for that type of command.

   It may seem odd that so many of the roles are based on position (e.g., “first word" and “second-to-last word"), rather than more functionally-relevant categories such as “direction word." However, this approach may actually be more efficient: Each command consists of a single mandatory element (namely, an action word such as *walk* or *jump*) followed by several optional modifiers (namely, rotation words, direction words, and cardinalities). Because most of the word categories are optional, it might be inefficient to check for the presence of, e.g., a cardinality, since many sequences will not have one. By contrast, every sequence will have a last word, and checking the identity of the last word provides much functionally-relevant information: if that word is not a cardinality, then the decoder knows that there is no cardinality present in the command (because if there were, it would be the last word); and if it is a cardinality, then that is important to know, because the presence of *twice* or *thrice* can dramatically affect the shape of the output sequence. In this light, it is unsurprising that the SCAN encoder has implicitly learned several different roles that essentially mean the last element of a particular subcommand.

2. The algorithm does not constitute a simple, transparent role scheme. But its job is to describe the representations that the original network produces, and we have no a priori expectation about how complex that process may be. The role-assignment algorithm implicitly learned by ROLE is interpretable locally (each line is readily expressible in simple English), but not intuitively transparent globally. We see this as a positive result, in two respects.

   First, it shows why ROLE is crucial: no human-generated role scheme would provide a good approximation to this algorithm. Such an algorithm can only be identified because ROLE is able to use gradient descent to find role schemes far more complex than any we would hypothesize intuitively. This enables us to analyze networks far more complex than we could analyze previously, being necessarily limited to hand-designed role schemes based on human intuitions about how to perform the task.

   Second, when future work illuminates the computation in the original SCAN GRU seq2seq decoder, the baroqueness of the role-assignment algorithm that ROLE has shown to be implicit in the seq2seq encoder can potentially explain certain limitations in the original model, which is known to suffer from severe failures of systematic generalization outside the training distribution (Lake and Baroni, 2018). It is reasonable to hypothesize that systematic generalization requires that the encoder learn an implicit role scheme that is relatively simple and highly compositional. Future proposals for improving the systematic generalization of models on SCAN can be examined using ROLE to test the hypothesis that greater systematicity requires greater compositional simplicity in the role scheme implicitly learned by the encoder.

3. While the role-assignment algorithm of A.8.1 may not be simple, from a certain perspective, it is quite surprising that it is not far more complex. Although ROLE is provided 50 roles to learn to deploy as it likes, it only chooses to use 16 of them (only 16 are ever selected as the $\arg\max(\boldsymbol{a}_t)$; see Sec. 6.1). Furthermore, the SCAN grammar generates 20,910 input sequences, containing a total of 151,688 words (an average of 7.25 words per input). This

means that, if one were to generate a series of conditional statements to determine which role is assigned to each word in every context, this could in theory require up to 151,688 conditionals (e.g., "if the filler is 'jump' in the context 'walk thrice after ____ opposite left', then assign role 17"). However, our algorithm involves just 47 conditionals. This reduction helps explain how the model performs so well on the test set: If it used many more of the 151,688 possible conditional rules, it would completely overfit the training examples in a way that would be unlikely to generalize. The 47-conditional algorithm we found is more likely to generalize by abstracting over many details of the context.

4. Were it not for ROLE's ability to characterize the representations generated by the original encoder in terms of implicit roles, providing an equally complete and accurate interpretation of those representations would necessarily require identifying the conditions determining the activation level of each of the 100 neurons hosting those representations. It seems to us grossly overly optimistic to estimate that each neuron's activation level in the representation of a given input could be characterized by a property of the input statable in, say, two lines of roughly 20 words/symbols; yet even then, the algorithm would require 200 lines, whereas the algorithm in A.8.1 requires 47 lines of that scale. Thus, by even such a crude estimate of the degree of complexity expected for an algorithm describing the representations in terms of neuron activities, the algorithm we find, stated over roles, is 4 times simpler.