Compositional Program Generation for Systematic Generalization

Tim Klinger*[1], **Luke Liu*[2]**, **Maxwell Crouse[1]**, **Soham Dan[1]**,
Parikshit Ram[1] and **Alexander Gray[1]**

[1]IBM Research AI
[2]New York University
* indicates equal contribution
{tklinger, Maxwell.Crouse, Soham.Dan, parikshit.ram, alexander.gray}@ibm.com, ql2078@nyu.edu

1 Introduction

Compositional generalization remains a difficult problem for neural models. There has been progress, but the hardest benchmark problems remain intractable without additional task-specific semantic information. In this abstract we describe a neuro-symbolic architecture Compositional Program Generator (CPG) which generalizes systematically and productively for sequence-to-sequence language tasks, given a context-free grammar of the input language and a dictionary mapping each input word to its interpretation in the output language. Our approach learns to generate type-specific symbolic semantic functions composed in an input-dependent way to produce the output sequence. In experiments with SCAN, CPG solves all splits and few-shot generalizes on the systematicity (”add jump”) split. It achieves perfect generalization as well on the COGS benchmark, after training for two epochs on sentences of length less than 13.

Although no satisfactory formal definition of compositional generalization has yet been given, it is usually taken to have two requirements: generalization to new (grammatical) combinations of parts of an input seen in training (systematicity); and generalization to longer sequences than seen in training (productivity). For example, having translated a command like “jump left twice” to (I_TURN_LEFT, I_JUMP, I_TURN_LEFT, I_JUMP) as in Figure 1; having seen the command “walk” in training; and knowing that “jump” and “walk” are both of the same type, the model should generalize systematically to translate “walk left twice” as (I_TURN_LEFT, I_WALK, I_TURN_LEFT, I_WALK) with no further training. This kind of generalization has proved challenging for standard neural models such as transformers [Vaswani *et al.*, 2017].

2 Approach

One strategy for achieving compositional generalization is to learn a compositional function. These are often described informally as “meaning” functions where “the meaning of the whole is a function of the meaning of the parts.” Usually attributed to Frege, this idea appears in some form as early as the sixth century BCE [Pagin and Westersthal, 2010]. In [Pagin and Westersthal, 2010] they give two formal definitions of compositional functions, one of which, the functional definition, we restate here with minor modifications. It assumes

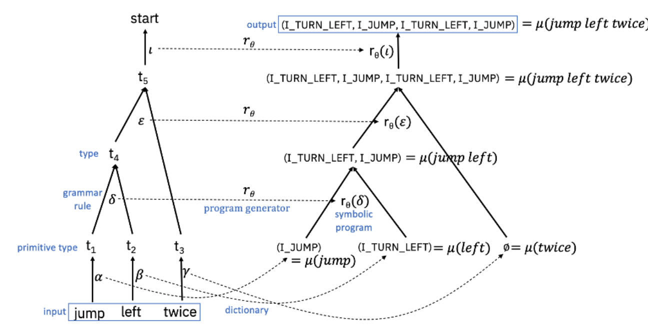

Figure 1: An example of compositional program generation on SCAN.

the input language L is defined over finite vocabulary V and is generated by a set of context-free grammar rules Σ (denoted α, β, γ, etc.) which map input types to an output type (or for primitive types, an input token to an output type).

A function μ from a source language $L \subset V^*$ to a target language L' is *compositional* if for every rule $\alpha \in \Sigma$ there is a function $r(\alpha)$ such that if α is defined, then $\mu(\alpha(u_1, u_2, \ldots, u_n)) = r(\alpha)(\mu(u_1), \mu(u_2), \ldots, \mu(u_n))$. Here V is a finite vocabulary and V^* denotes the set of all strings over that vocabulary. The function μ (“meaning”) is defined recursively over a structure (“parts”) determined by a parse of the input.

Compositional functions are a useful hypothesis class for learning models which compositionally generalize. First, they are defined recursively over an arbitrarily large input structure and hence productive by definition. Second, they offer a natural way to decompose the model by processing each rule α with a semantic module specific to that rule ($r(\alpha)$). Third, they allow the structure of the semantic computation to vary depending on the input. Finally, since r is not directly dependent on the input, a learner will generate the same semantic function for the same syntactic rule and the model will generate the same meaning for phrases with the same derivations in the grammar which is what is desired for systematic generalization.

We decompose the problem of learning a sequence-to-sequence model into sub-problems: specifying a context-free

$copy(x, x')$:
\# $|x| = |x'| = k$. α a grammar rule. θ trainable parameters
\# $P_\theta(\alpha)$ a probability distribution over $[0, \ldots, k-1]$
$s \leftarrow [j \sim P_\theta(\alpha)$ for i in $[0, \ldots, k-1]]$
\# x – source; x' – destination; s – indices to copy
$x'[i] \leftarrow x[s[i]]$ for i in s

(a) copy semantics

$subst(x, x')$:
\# $|x| = |x'| = k$. α a grammar rule. θ trainable parameters
\# $P'_\theta(\alpha)$ a probability distribution over $[0, \ldots, l]$
\# find target slots
$t \leftarrow$ find_slots(x')
$l \leftarrow |t|$
\# generated indices between 0 and the number of slots l.
\# index l indicates no substitution.
$s \leftarrow [j \sim P'_\theta$ for i in $[0, \ldots, l-1]]$
\# x – source; x' – destination; t – slot indices; s – source indices
$x'[t[i]] \leftarrow x[s[i]]$ if $s[i] \neq l$ else $x'[t[i]]$ for i in $[0, \ldots, l-1]$

(b) substitution semantics

Figure 2: (a) copy program (top); (b) substitution program (bottom)

grammar for the input language L; programmatically generating a parser (we use Lark [1]); learning or providing the dictionary that maps input tokens to their output representation (if any); and learning to generate a type-specific semantic function for each type in the grammar. We will focus mainly on the problem of generating type-specific functions and leave it to future work to learn the grammar. We are able to learn a dictionary jointly with the semantic functions for SCAN since it's quite simple; learning the dictionary for COGS is more involved and we don't try it here. With these assumptions we are able to solve SCAN in a productive and systematic way with interpretable semantic rules and with length-curricular training on sentences of commands of length ≤ 3 with few shot generalization on the "add jump" (systematicity) split. Using the same architecture and curricular training, preliminary experiments show perfect generalization on COGS, though we do not have final results to report.

Our approach to learning a compositional function is *compositional program generation* by which we mean both that the process of generating a (symbolic) program from an instance is compositional, generating each composition recursively, bottom-up over the structure of the parse, but also that the generated program is itself compositional with respect to that structure in the sense of [Pagin and Westersthal, 2010], obeying the requirement that the meaning of the whole be a function of the meaning of it's parts.

The CPG model learns to generate parameters for two kinds of programs, *copy* programs, which produce an output sequence by copying elements of the input sequence (with filtering and repeats allowed), and *substitution* programs which substitute objects from one list into slots in an expression (denoted y in our examples) for use in COGS. Figure 2 gives the semantics for these operations.

Using the Lark parser generator we generate a parser for the input grammar and run it to produce a linear sequence of parse steps which produce the parse tree for that input. Each parse step applies a context-free grammar rule α to a contiguous span of the input and replaces it with the type specified as the output of α. Grammar rules with arity l for non-primitive types (see Figure 1) are of the form: $t_0 t_1 \ldots t_{l-1} \rightarrow t$ for

[1] https://github.com/lark-parser/lark

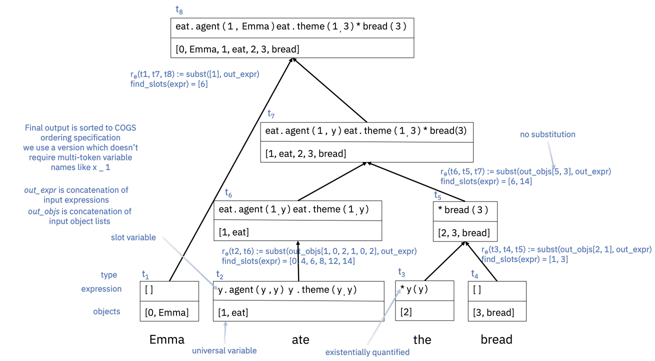

Figure 3: compositional generation on COGS

input grammar types t_i, $0 \leq i \leq l-1$, and output grammar type t. For primitive types the rules are of the form $x \rightarrow t$ for x an input token and t a primitive type. Disjunctive rules are of the form $t_0|t_1|\ldots|t_{l-1} \rightarrow t$. Figure 1 shows an example of a parse tree constructed by successively applying the parse sequence rules to the input.

The parse rule sequence is the input to a recursive model, the compositional learner, which learns the function r_θ shown in Figure 1. To initialize the process it maps the input tokens individually to their output language interpretation using the dictionary as shown in the bottom of Figure 1. If a token does not have a corresponding output token it is mapped to the empty list, denoted ϕ. For COGS, where the output is a logical form, variables are required and we map each input token to a pair of sequences as shown in Figure 3, the first holding an expression representing the meaning of the input word (which may contain slots denoted y, e.g. $y.agent(y, y)$); and a separate sequence for objects (universal variables and constants) (e.g. $[0, Emma]$). Substitution relies on a function $find\_slots(x)$ which returns the indices of the y variables (0-indexed). An example showing composition in COGS is shown in Figure 3. At each step the argument expression lists are concatenated to form the output expression and similarly with the object lists to form the output object list. After concatenation the generated substitution program is applied, to substitute objects from the concatenated object list for y variables in the output expression. Disjunctive rules are not associated with any substitution so the input object and expression lists are just concatenated into their output lists.

3 Experiments

All distributions for SCAN are implemented as linear layers (for COGS two-layer) followed by a Gumbel softmax estimator to sample. There are separate layers for each rule encouraging specialization. Training is curricular by input length and temperature is annealed for the Gumbel distributions in proportion to curriculum stage accuracy.

We've conducted experiments with both SCAN and COGS. For SCAN we modify the "add primitive" (systematicity) training data to have just a single example of the held-out word (one-shot). On SCAN the model generalizes perfectly on all splits after training on up to length 3 sentences, since all types have been seen in training by that point. COGS results also show perfect generalization for the subset of the training set containing sentences of length 13 or less after 2 epochs of training.

References

[Fodor and Pylyshyn, 1988] Jerry A Fodor and Zenon W Pylyshyn. Connectionism and cognitive architecture: A critical analysis. *Cognition*, 28(1-2):3–71, 1988.

[Herzig and Berant, 2021] Jonathan Herzig and Jonathan Berant. Span-based semantic parsing for compositional generalization. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 908–921, Online, August 2021. Association for Computational Linguistics.

[Hupkes *et al.*, 2020] Dieuwke Hupkes, Verna Dankers, Mathijs Mul, and Elia Bruni. Compositionality decomposed: how do neural networks generalise? *Journal of Artificial Intelligence Research*, 67:757–795, 2020.

[Keysers *et al.*,] Daniel Keysers, Nathanael Schärli, Nathan Scales, Hylke Buisman, Daniel Furrer, Sergii Kashubin, Nikola Momchev, Danila Sinopalnikov, Lukasz Stafiniak, Tibor Tihon, et al. Measuring compositional generalization: A comprehensive method on realistic data. In *International Conference on Learning Representations*.

[Kim and Linzen, 2020a] Najoung Kim and Tal Linzen. COGS: A compositional generalization challenge based on semantic interpretation. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 9087–9105, Online, November 2020. Association for Computational Linguistics.

[Kim and Linzen, 2020b] Najoung Kim and Tal Linzen. Cogs: A compositional generalization challenge based on semantic interpretation. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 9087–9105, 2020.

[Lake and Baroni, 2018] Brenden Lake and Marco Baroni. Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks. In *International Conference on Machine Learning*, pages 2873–2882. PMLR, 2018.

[Liu *et al.*, 2020] Qian Liu, Shengnan An, Jian-Guang Lou, Bei Chen, Zeqi Lin, Yan Gao, Bin Zhou, Nanning Zheng, and Dongmei Zhang. Compositional generalization by learning analytical expressions. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 11416–11427. Curran Associates, Inc., 2020.

[Liu *et al.*, 2021] Chenyao Liu, Shengnan An, Zeqi Lin, Qian Liu, Bei Chen, Jian-Guang Lou, Lijie Wen, Nanning Zheng, and Dongmei Zhang. Learning algebraic recombination for compositional generalization. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 1129–1144, 2021.

[Loula *et al.*, 2018] Joao Loula, Marco Baroni, and Brenden M Lake. Rearranging the familiar: Testing compositional generalization in recurrent networks. *arXiv preprint arXiv:1807.07545*, 2018.

[Pagin and Westersthal, 2010] Peter Pagin and Dag Westersthal. Compositionality i: Definitions and variants. *Philosophy Compass*, 5(3):250–264, 2010.

[Ruis *et al.*, 2020] Laura Ruis, Jacob Andreas, Marco Baroni, Diane Bouchacourt, and Brenden M Lake. A benchmark for systematic generalization in grounded language understanding. *Advances in Neural Information Processing Systems*, 33, 2020.

[Vaswani *et al.*, 2017] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.