Compositional Program Generation for Systematic Generalization

Tim Klinger*1, Luke Liu#2, Maxwell Crouse1, Soham Dan1, Parikshit Ram1 and Alexander Gray1

1IBM Research AI
2IBM Research AI

* 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 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 (l_TURN_LEFT, l_JUMP, l_TURN_LEFT, l_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 (l_TURN_LEFT, l_WALK, l_TURN_LEFT, l_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 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 ⊂ V* to a target language L’ is compositional if for every rule α ∈ Σ there is a function r(α) such that if α is defined, then μ(α(u1, u2, . . ., un)) = r(α)(μ(u1), μ(u2), . . ., μ(un)). 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(α)). 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
COGS, though we do not have final results to report.

Preliminary experiments show perfect generalization on the “add jump” (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

# Lark
https://github.com/lark-parser/lark

## 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


