Learning Compositional Rules via Neural Program Synthesis

Maxwell Nye* MIT
Armando Solar-Lezama MIT
Joshua Tenenbaum MIT
Brenden Lake NYU and Facebook AI

Abstract

Humans can learn systematic, compositional rules from very little data, and learn to quickly understand entirely novel rule-based systems. However, previous neural network translation approaches are not able to consistently learn this compositional behavior. In this work, we present a neuro-symbolic model which learns entire rule systems from a small set of examples. Instead of training a model to predict the correct output given a novel input, we train our model via meta-learning to induce the explicit system of rules governing the behavior of all previously seen examples. We draw on existing work in the neural program synthesis literature, building a rule-synthesis method which exceeds human performance in an artificial instruction learning domain, and outperforms neural meta-learning techniques.

1 Introduction

Humans have the remarkable ability to learn systematic, compositional rules from very little data. For example, a person can learn a novel verb, “to dax," from a few examples, and immediately understand what it means to “dax twice" or “dax around the room quietly." When learning their native language, children face the more challenging task of learning many interrelated concepts simultaneously, including the meaning of both verbs and modifiers (“twice", “quietly”, etc.), and how they combine to form complex meanings. This quintessentially human ability is an important skill, allowing humans to bootstrap complex systems from very little data.

People can also learn to quickly interpret entirely novel rule systems. In Lake et al. (2019), human subjects were shown to be able to learn a novel artificial language (see Fig 1). Given only 14 examples, participants were able to induce a set of general rules and primitives for how to translate a sequence of nonce words into a sequence of colored circles.

An important goal of AI is to build systems which possess this sort of rule-learning ability. Previous neural-network translation approaches—which rely on very large training sets and map input sequences directly to output sequences—are not able to consistently learn systematic, compositional behavior from such few examples (Lake & Baroni, 2017; Lake, 2019; Loula et al., 2018).

In this work, we present a model which learns entire rule systems from examples. Our key idea is to frame the problem as explicit rule-learning; instead of training a model to predict the correct output given a novel input, we train our model to induce the explicit system of rules governing the behavior of all previously seen examples. Once inferred, this rule system can be used to predict the behavior of any new example.

This explicit rule-based approach confers several advantages compared to a pure input-output based approach. Instead of learning a black-box input-output mapping, and applying it to each new query item for which we’d like to predict an output, we instead search for an explicit *program* which we can check against previous examples (the). This allows us to propose and check candidate programs,

*Corresponding Author. mnye@mit.edu

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

only terminating search when the proposed solution is consistent with prior data. This framing also allows immediate and automatic generalization: once the correct rule system is learned, it can be correctly applied in novel scenarios which are a) arbitrarily complex and b) outside the distribution of previously seen examples. To build our rule-learning system, we draw on existing work in the neural program synthesis literature (Ellis et al., 2019; Yang & Deng, 2019), allowing us to solve complex rule-learning problems which are difficult for both neural and traditional symbolic methods.

Our model is trained via meta-learning. Assuming a distribution of rule systems, or a "meta-grammar," we train our model by sampling grammar-learning problems and training on these sampled problems. We demonstrate that our rule-synthesis method, when trained on a general meta-grammar of rule-systems as in Lake (2019), can match or exceed human performance, and outperform neural meta-learning techniques.

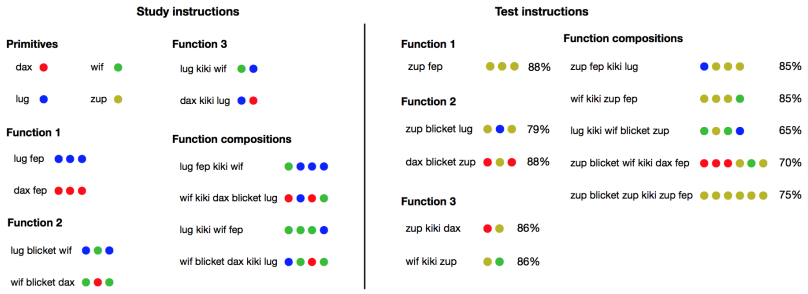

Figure 1: An example of few-shot learning of instructions. In Lake et al. (2019), participants learned to execute instructions in a novel language of pseudowords by producing sequences of colored circles. Generalization performance is shown next to each test instruction, as the percent correct across participants. When conditioned on the study instructions, our model is able to predict the correct output sequences on the held out test instructions by synthesizing the grammar in Fig. 2. Figure reproduced from Lake et al. (2019)

2 Our Approach

In this section we outline our general approach to solving rule-learning tasks.

Overview: Given a small support set of input-output examples, our goal is to produce the outputs corresponding to a query set of inputs. To do this, we build a neural program induction model which accepts the given examples and synthesizes a symbolic program, which we can execute on query inputs to predict the desired query outputs. Our symbolic program consists of an "interpretation grammar," which is a sequence of *rewrite rules*, each of which represents a transformation of token sequences. The interpretation grammar is discussed below. At test time, we employ our neural program induction model to drive a simple search process. This search process proposes candidate symbolic programs by sampling from the neural program induction model and symbolically checks whether candidate programs satisfy the support examples by executing them on the support inputs. The search process builds the program iteratively, by sampling one rewrite rule at a time, and checking this rule for consistency with the support examples. Our model is trained via a meta-learning approach. During a training episode, our model is given support examples, and the goal is to infer an underlying program to explain the support and held-out query examples.

Model: Figure 2 illustrates our model. The neural model is relatively simple and consists of two components: an encoder, which encodes each support example and each previous rule into a vector, and a decoder RNN, which decodes the next rule while attending to the examples and the previous rules. For each support example, the input sequence and output sequence are each encoded into a vector by an input RNN encoder and an output RNN encoder, respectively, and then combined via a linear layer to produce one vector per support example. Likewise, each previous rule is encoded via a separate rule RNN encoder. The decoder RNN is initialized with the sum of all of the support example vectors combined with the sum of the rule vectors, and decodes the output by attending to the support examples and the previous rules at each decoding step. The decoder attends to both the

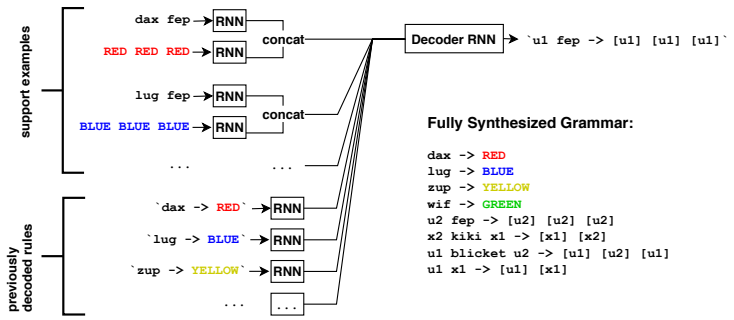

Figure 2: Illustration of our synthesis-based rule learner. Support examples and previously decoded rules are encoded via RNNs. The decoder RNN attends over the resulting vectors and decodes the next rule. Bottom right: an example of a fully synthesized grammar which solves the task in Fig. 1.

support examples and the previous rules by performing sequential double attention, first attending to the rules, and then to the support examples, as in Devlin et al. (2017).

Interpretation Grammar: The interpretation grammar used in this work consists of an ordered list of rules. Each rule consists of a left hand side (LHS) and a right hand side (RHS). The left hand side consists of the input words, string variables `x1`, `x2` and primitive variables `u1`, `u2`. Evaluation proceeds as follows: An input sequence is checked against the rules, in order of the rule priority, and if the rule LHS matches the input sequence, it is replaced with the RHS. If the RHS has variables, they are evaluated recursively through the same process.

Iterative Grammar Construction: Our model iteratively constructs an interpretation grammar by producing one rule at a time. After each new rule is produced, it is checked against the support examples. If any of the support examples is now "solved"—ie, the new partial grammar transforms the input sequence into the correct output sequence—then that example is removed from the support set. The new rule is added to the list of "previous rules." To infer the next rule, the neural network is conditioned on the set of previous rules and the remaining support examples. Inference proceeds until all support examples have been solved, at which point the iteratively constructed grammar can be applied to the held-out query set.

Meta-training: We train our model in a similar manner to Lake (2019). During each training episode, we randomly sample an interpretation grammar from a distribution over interpretation grammars, or "meta-grammar." We then randomly sample a set of input sequences consistent with the sampled interpretation grammar, and apply the interpretation grammar to each input sequence to produce the corresponding output sequence. This gives us a support set of input-output examples. We train our network via supervised learning to iteratively output the interpretation grammar when conditioned on the support set of input-output examples. In the example in Figure 1, we would take the "study instructions" as our support set, and the "test instructions" would comprise the query set.

3 Experiments

Our experimental domain is the paradigm introduced in Lake et al. (2019). The goal of this domain is to learn compositional, language-like rules from a very limited number of examples.

Training details: To perform these rule-learning tasks, we trained our model on a series of meta-training episodes. During each episode, a grammar was sampled from the meta-grammar distribution, and our model was trained to recover this grammar given a support set of example sequences. In our experiments, the meta-grammar consisted of all grammars with 4 *primitive* rules and 2-4 *higher-order* rules. Primitive rules map a word to a color (e.g. `dax -> RED`), and higher order rules encode variable transformations given by a word (e.g. `x1 kiki x2 -> [x2] [x1]`). (In a higher-order rule, the LHS can be one or two variables and a word, and the RHS can be any sequence of those variables.) For each grammar, we trained with a support set of 10-20 randomly sampled examples.

We implemented our models in PyTorch, and used RNN embedding and hidden sizes of 200. We trained our models for 12 hours on a Titan-x GPU, using a batch size of 64 and a learning rate of 0.001 using the Adam optimizer (Kingma & Ba, 2014).

Alternate Models: We compare our model against two alternate models. The first alternate model is a non-iterative version of our synthesis model, dubbed the **Non-iterative** baseline. This model is trained to decode all rules at once, seperated by a special `NEW_RULE` character. Therefore, this model does not encode previous rules, or modify the support set during inference. The second alternate model is the **meta seq2seq** model introduced in Lake (2019). This model is also trained on episodes of randomly sampled grammars. However, instead of synthesizing a grammar, the meta seq2seq model conditions on support examples and attempts to translate query inputs directly to query outputs in a sequence to sequence manner.

Test details: Our synthesis methods were tested by sampling from the neural network until a candidate grammar was found which was consistent with all of the support examples. We used a sampling timeout of 30 sec., and sampled with a batch size of 64. For the meta seq2seq model, we greedily decoded the most likely output sequence for each query input, as in Lake (2019). For each of our experiments, we tested on 50 held-out test grammars, each containing 10 query examples.

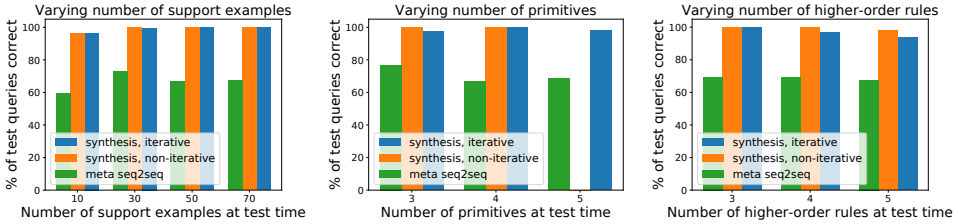

Figure 3: Main experimental results. We train on random grammars with 4 primitives, 2-4 higher order rules, and 10-20 support examples. At test time, we vary the number of support examples (right), primitive rules (center), and higher-order rules (left). The synthesis-based approaches achieve near-perfect accuracy for most test conditions.

Results: To evaluate our rule-learning model and alternate models, we test the trained models on a battery of evaluation schemes. In general, we observe that the synthesis methods are much more accurate than than the meta seq2seq method, and the iterative synthesis method is less brittle than the non-iterative synthesis method. Our results are displayed in Figure 3. We observed that, when the support set was too small, there were often not enough examples to disambiguate between several grammars which all satisfied the support set, but may not satisfy the query set. Thus, in our first experiment, we varied the number of support examples during test time and evaluated the accuracy of each of our models. We observed that, when we increased the number of support elements to 50 or more, the probability of failing any of the query elements fell to nearly zero for the synthesis-based methods. When the test grammars come from the same distribution as the training grammars, the synthesis-based models can almost always recover the correct grammar. However, we wanted to determine how well these models could generalize outside the training distribution. For our second and third experiments, we varied the number of primitives in the test grammars, and the number of higher-order functions in the test grammars, respectively. We observe that the non-iterative synthesis model is quite brittle: if it is only trained on 4 primitives, it cannot generalize to 5 primitives. Only the iterative model is able to generalize to 5 primitives when trained on 4 primitives. Both synthesis models are able to correctly translate query items when grammars have 5 higher order rules, even though they were trained only on 2-4 higher order rules.

Furthermore, in instances where the synthesis based methods have perfect accuracy because they recover exactly the generating grammar (or some equivalent grammar), they would also be able to trivially generalize to query examples of any size or complexity, as long as these examples followed the same generating grammar. On the other hand, as reported in Lake (2019), approaches which attempt to neurally translate directly from inputs to outputs cannot accurately translate query sequences which are much longer than the training examples or test-time support examples. This is a clear conceptual advantage of the synthesis based approach; symbolic rules, if accurately inferred, necessarily allow correct translation in every circumstance.

Discussion: We present a neuro-symbolic program induction model which can learn rule-based systems via meta-learning. We demonstrate that our model can learn to infer unseen rule systems in few-shot artificial language-learning domain previously tested on humans. Future work will include testing on larger rule-learning problems, such as the SCAN dataset (Lake & Baroni, 2017).

References

Jacob Devlin, Jonathan Uesato, Surya Bhupatiraju, Rishabh Singh, Abdel-rahman Mohamed, and Pushmeet Kohli. Robustfill: Neural program learning under noisy i/o. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 990–998. JMLR. org, 2017.

Kevin Ellis, Maxwell Nye, Yewen Pu, Felix Sosa, Josh Tenenbaum, and Armando Solar-Lezama. Write, execute, assess: Program synthesis with a repl. *arXiv preprint arXiv:1906.04604*, 2019.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Brenden M Lake. Compositional generalization through meta sequence-to-sequence learning. *arXiv preprint arXiv:1906.05381*, 2019.

Brenden M Lake and Marco Baroni. Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks. *arXiv preprint arXiv:1711.00350*, 2017.

Brenden M Lake, Tal Linzen, and Marco Baroni. Human few-shot learning of compositional instructions. *arXiv preprint arXiv:1901.04587*, 2019.

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

Kaiyu Yang and Jia Deng. Learning to prove theorems via interacting with proof assistants. *arXiv preprint arXiv:1905.09381*, 2019.