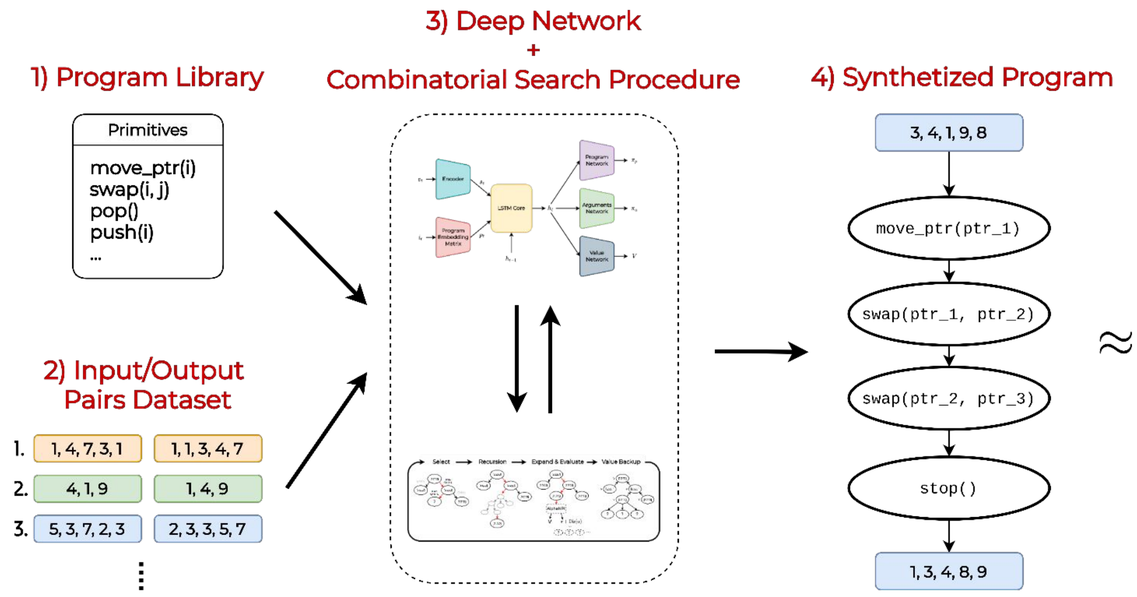
Learning compositional programs with arguments and sampling

Giovanni De Toni[1], Luca Erculiani[1], Andrea Passerini[1]

[1] Structured Machine Learning Group (SML), University of Trento, Italy

Motivation

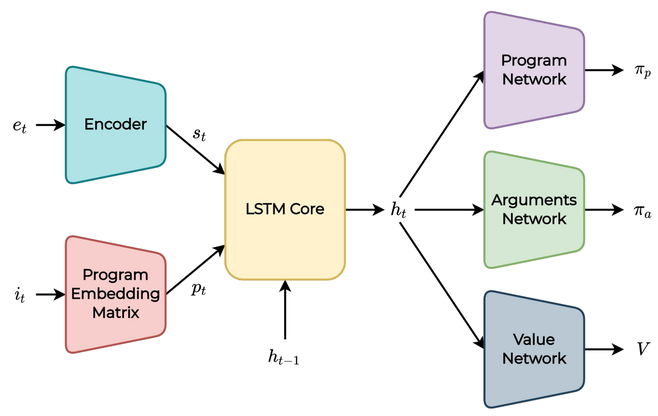


Code on GitHub

geektoni/learning_programs_with_arguments

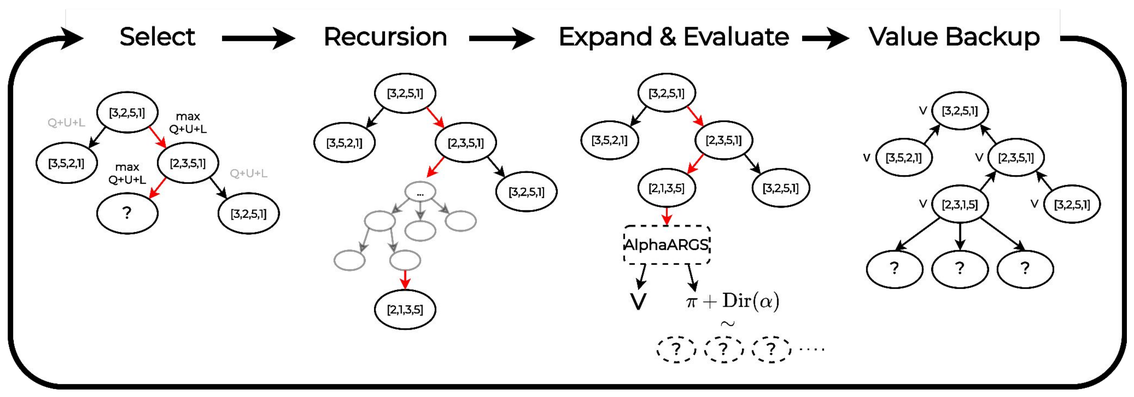
One of the most challenging goals in designing intelligent systems is to synthesize programs from data. Given requirements in the form of input/output pairs, we want to train a machine learning model to discover a compositional program to satisfy those requirements by merging combinatorial search procedures with deep learning. Previous works usually generated toy programs using a domain-specific language that did not provide high-level features, such as function arguments. We learn to generate functions that accept arguments, and we show the potential of our approach on the task of sorting lists of integers by learning the Quicksort algorithm.

Architecture



- We based it on **AlphaNPI** by Pierrot et al., 2019. **We adapted it to the novel setting by adding the Arguments Network module.**
- The modules concur in generating **two policies**: probability distributions over the **program space** and **arguments space**.
- We use those distributions to decide the next program/arguments pair to execute (e.g., `move_pointer_left(pointer_1)`).

(Approximate) Monte Carlo Tree Search



- The **Approximate Monte Carlo Tree Search (A-MCTS)** generates the execution traces needed to train the architecture.
- It runs several simulations of different programs by combining the atomic actions and already learned programs.
- Unlike the original MCTS, in our version, the expansion phase is **approximate.**

QuickSort Programs Hierarchy

1. partition_update()
2. partition()
3. quicksort_update()
4. quicksort()

```
def QuickSort(arr, low, high):
    if len(arr) == 1:
        return arr
    if low < high:
        pivot = partition(arr, low, high)
        QuickSort(arr, low, pivot-1)
        QuickSort(arr, pivot+1, high)
```

- We want to learn a **hierarchy of 4 programs** that can be combined to build the QuickSort algorithm.
- We learn these programs by exploiting a set of **atomic actions** that can accept arguments.

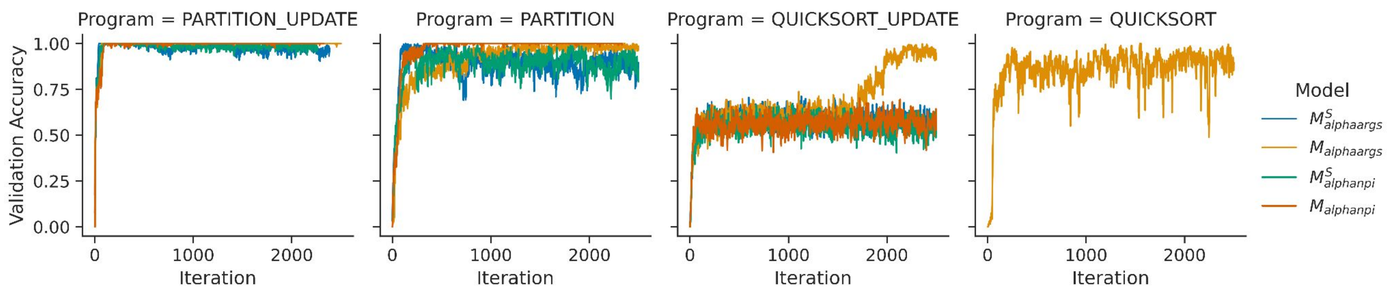
Training

$$l = \sum_{batch} - \underbrace{(\pi_p^{mcts})^T \log \pi_p}_{l_{policy}} - \underbrace{(\pi_a^{mcts})^T \log \pi_a}_{l_{arguments}} + \underbrace{(V - r)^2}_{l_{value}}$$

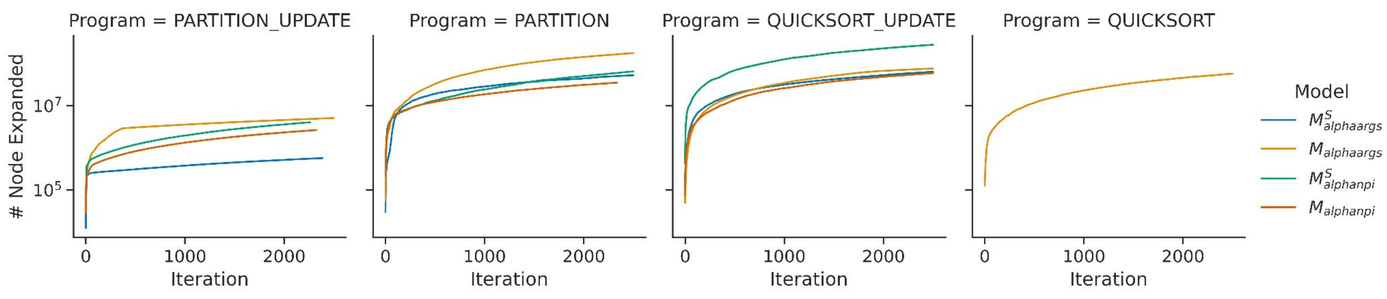
- We minimize the **cross-entropy between the A-MCTS and model policies** under a **reinforcement learning setting**.
- We employ **curriculum learning** to learn lower-level programs in the hierarchy first.
- We also do **retraining over failed environments** to improve robustness and generalization.

Experimental Evaluation

1) Training Accuracy (over all learned programs)



2) Total Nodes Expanded by A-MCTS and MCTS during training



3) Generalization Accuracy on lists of increasing length

Program	List Length	$M_{alphanpi}$	$M^S_{alphanpi}$	$M_{alphaargs}$	$M^S_{alphaargs}$
partition_update	5	**1.00**	**1.00**	**1.00**	0.96
	10	**1.00**	**1.00**	**1.00**	0.98
	20	**1.00**	0.98	**1.00**	0.96
	40	**1.00**	**1.00**	**1.00**	0.94
	60	**1.00**	**1.00**	**1.00**	0.96
partition	5	**1.00**	0.98	**1.00**	0.96
	10	**1.00**	0.78	0.98	0.78
	20	**1.00**	0.80	**1.00**	0.70
	40	**1.00**	0.80	**1.00**	0.82
	60	**1.00**	0.76	0.98	0.80
quicksort_update	5	0.72	0.58	**1.00**	0.60
	10	0.64	0.42	**0.96**	0.48
	20	0.56	0.34	**0.90**	0.56
	40	0.36	0.30	**0.98**	0.40
	60	0.38	0.26	**0.94**	0.44
quicksort	5	0.10	0.02	**1.00**	0.06
	10	0	0	**0.66**	0
	20	0	0	**0.42**	0
	40	0	0	**0.22**	0
	60	0	0	**0.02**	0

- We trained **four different models**. Two of them use the original AlphaNPI implementation. The other two uses our model. Moreover, we tested our Approximate MCTS against the original recursive MCTS.
- **Our model that supports the generation of functions that can accept arguments is the only one to learn QuickSort correctly** (red column). It also shows some generalization capabilities.
- In some cases, **Approximate MTCS shows to enable convergence by exploring fewer nodes in the search tree.**

License

Acknowledgements

We would like to thank Dr. Cristian Consonni that kindly shared with us this template poster.