CS 252R

DeepCoder

Learning To Write Programs

Eric Zhang — October 15th, 2020

Inductive Program Synthesis

"Inductive" = Inference

Input-output example:
Input:
`2, [3 5 4 7 5]`
Output:
`[7]`
	Description:
	A new shop near you is selling n paintings. You have $k < n$ friends and you would like to buy each of your friends a painting from the shop. Return the minimal amount of money you will need to spend.

Input-output example:
Input:
`[6 2 4 7 9],`
`[5 3 6 1 0]`
Output:
`27`
	Description:
	In soccer leagues, match winners are awarded 3 points, losers 0 points, and both teams get 1 point in the case of a tie. Compute the number of points awarded to the winner of a league given two arrays w, t of the same length, where $w[i]$ (resp. $t[i]$) is the number of times team i won (resp. tied).

Input-output example:
Input:
`[6 2 4 7 9],`
`[5 3 2 1 0]`
Output:
`4`
	Description:
	Alice and Bob are comparing their results in a recent exam. Given their marks per question as two arrays a and b, count on how many questions Alice got more points than Bob.

Neural Networks + Synthesis

End-to-end vs. integrated

- **End-to-end**
  - Neuro-Symbolic Program Synthesis (R3NN)
  - Machine translation with some knowledge of the context-free grammar
- **Integrated**
  - Using deep learning to give *hints* to existing domain-specific search techniques
  - Train neural network on a dataset

Learning-Augmented Algorithms

Background topic

- **Idea:** take existing algorithms, add machine learning as a black-box, and improve the performance assuming competence of the learned model.
- The best learning-augmented algorithms don't even decrease in worst-case performance when the ML model outputs bad predictions!
- **Example:** learning-augmented binary search.
  - Binary search takes $O(\log n)$ time, where the size of the array is n.
  - We can start with an *initial guess* from a neural network for the true index, and if this is within t from the actual index, then we can get $O(\log t)$ time!

Augmented Alg.

Learned Bloom filter

- Particularly nice because Bloom filters can get false positives anyway, so we're already assuming some error.
- Even simple neural networks can greatly reduce the storage costs.
- "The Case for Learned Index Structures" (Google, 2018).

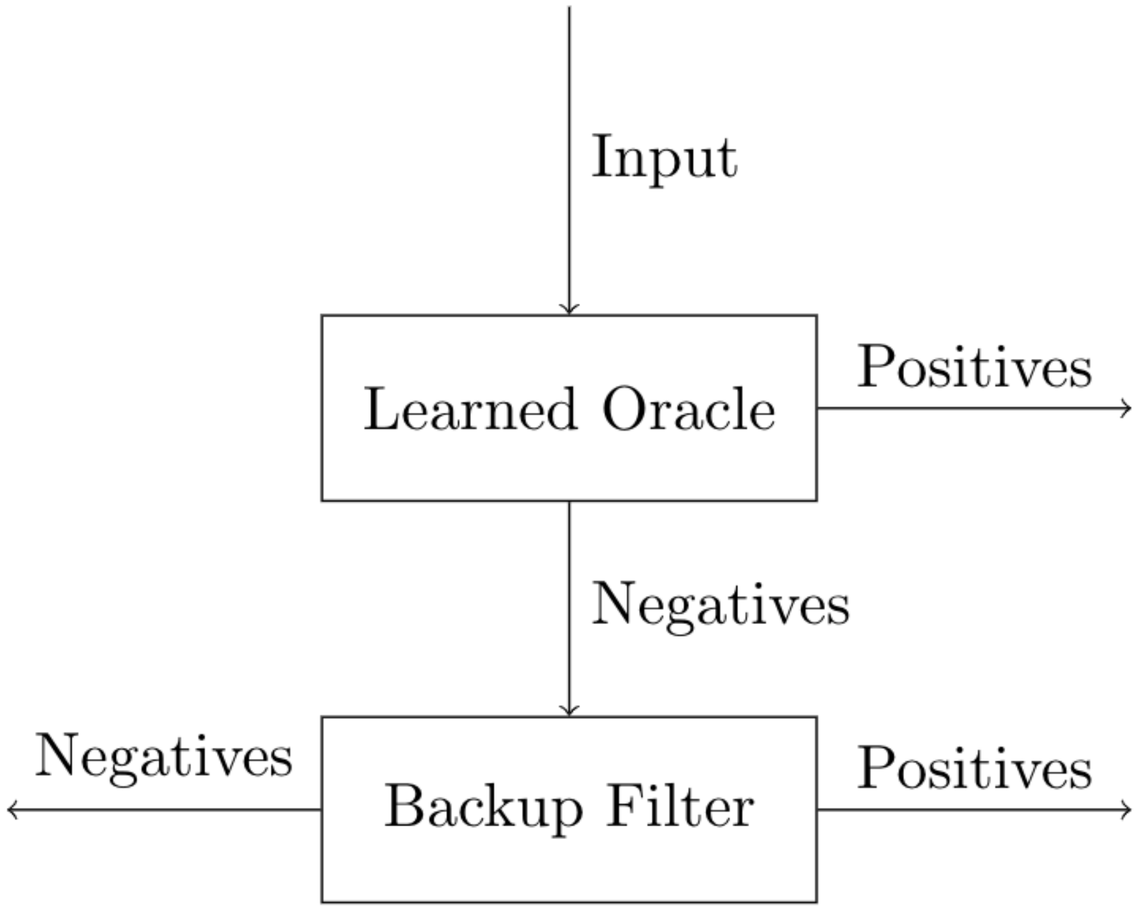
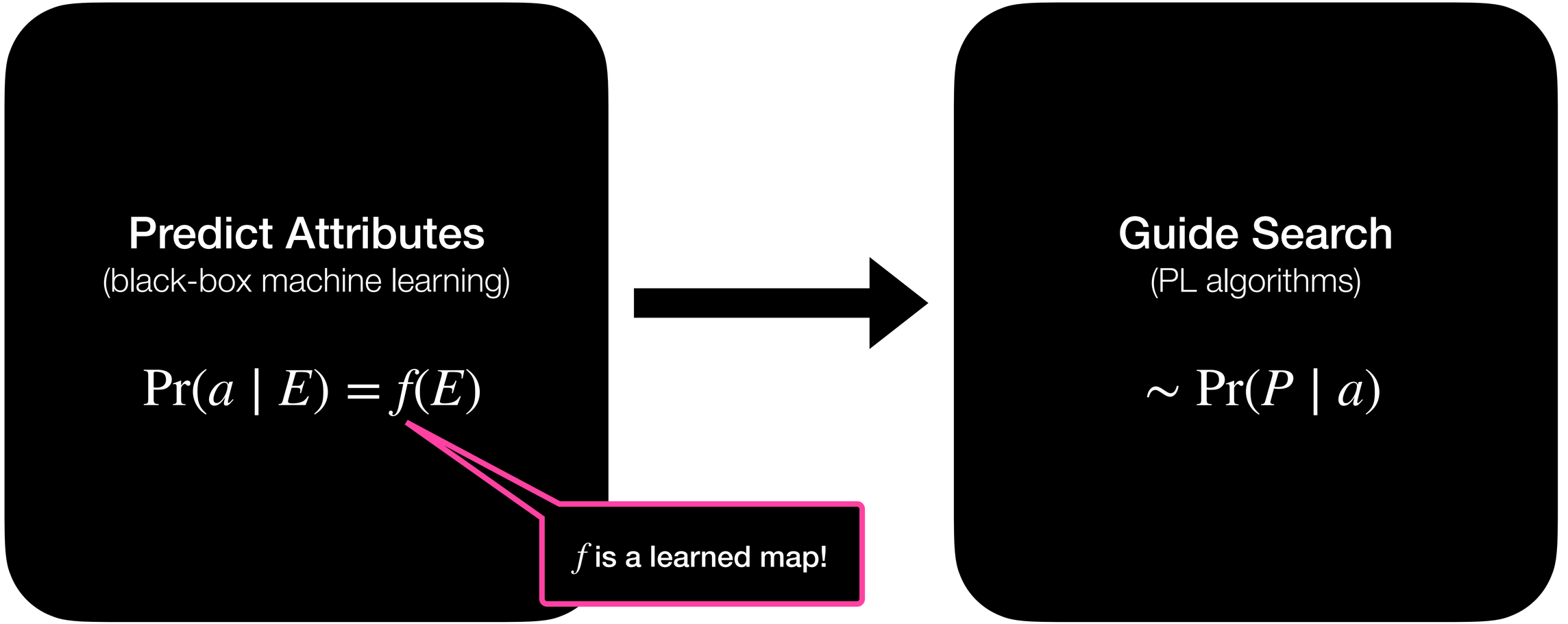


Figure 1: A learned Bloom filter.

LIPS

“Learned Inductive Program Synthesis”

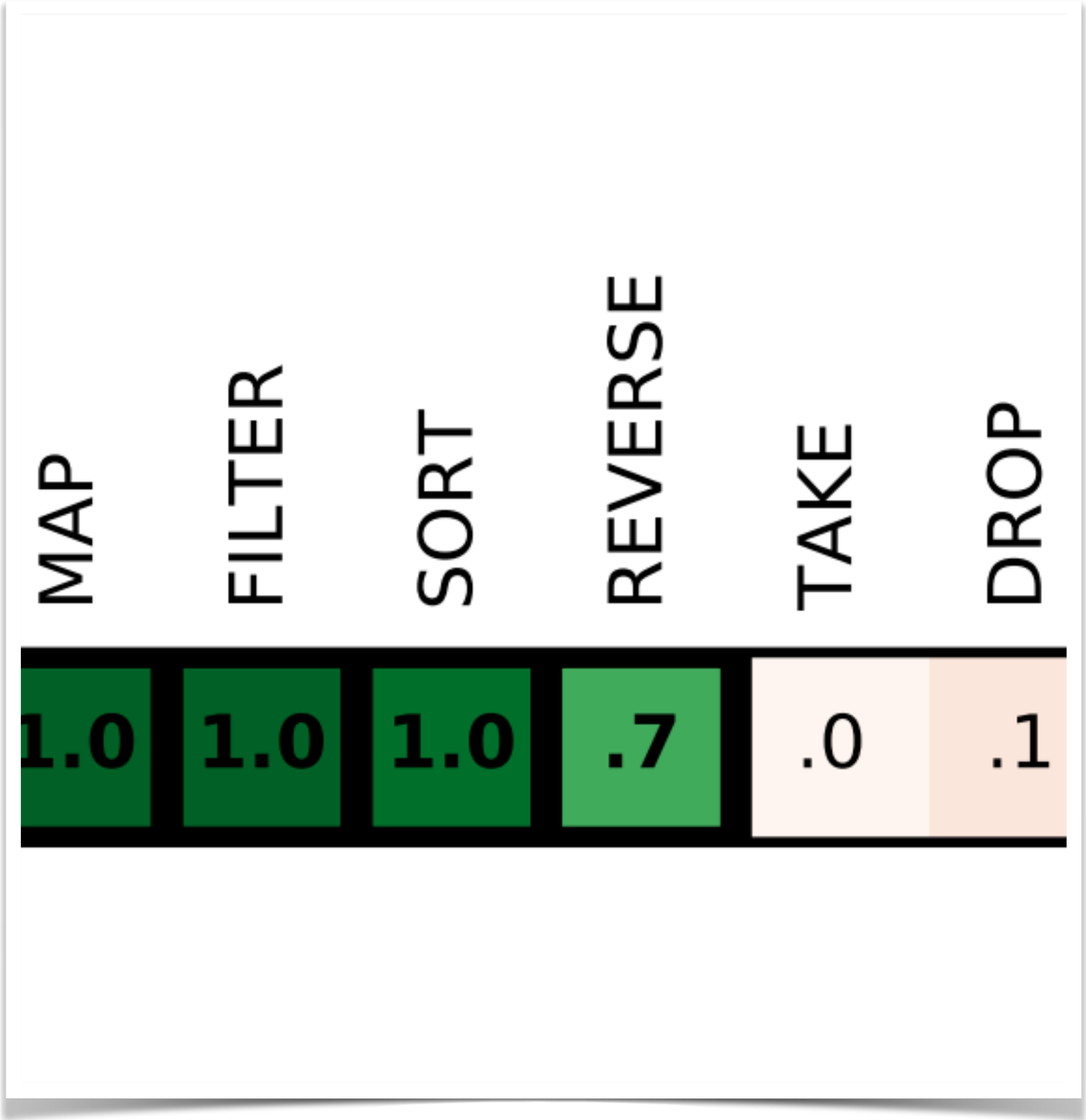


Part I: Machine Learning

Attributes

Summarizing P

- DSL has $C = 34$ functions.
- Program P is represented by an attribute vector $a \in \mathbb{R}^C$.
- $\mathscr{A} : \mathscr{P} \to \mathbb{R}^C$ attribute function simply gives a 0—1 vector detecting *presence* for each function in the program.



High-Level Attributes

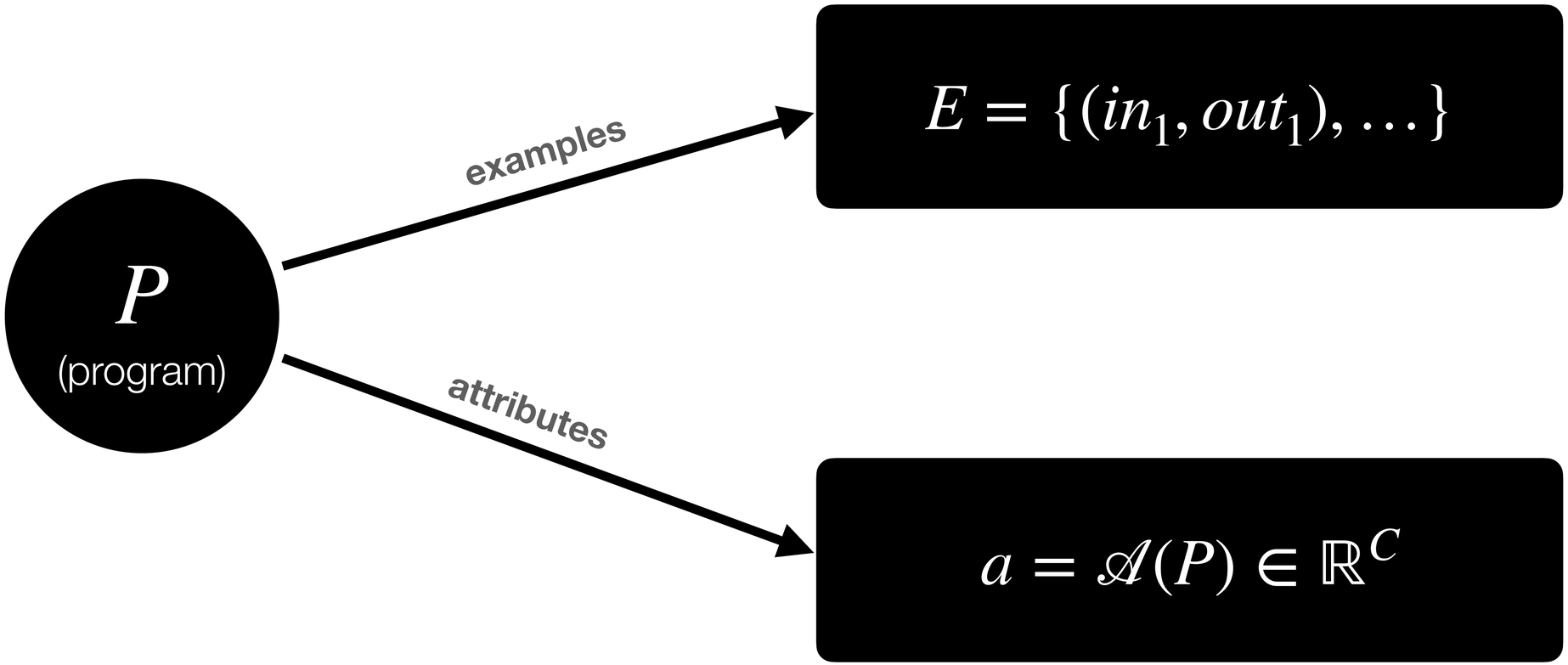
…may be reflected in the input data!

An input-output example:
Input:
`[-17, -3, 4, 11, 0, -5, -9, 13, 6, 6, -8, 11]`
Output:
`[-12, -20, -32, -36, -68]`

- All the outputs are even => might have the `map`, `filter`, `(*n)` functions.
- Outputs are sorted => might have the `sort`, `reverse` functions.
- **Goal:** model this human intuition.

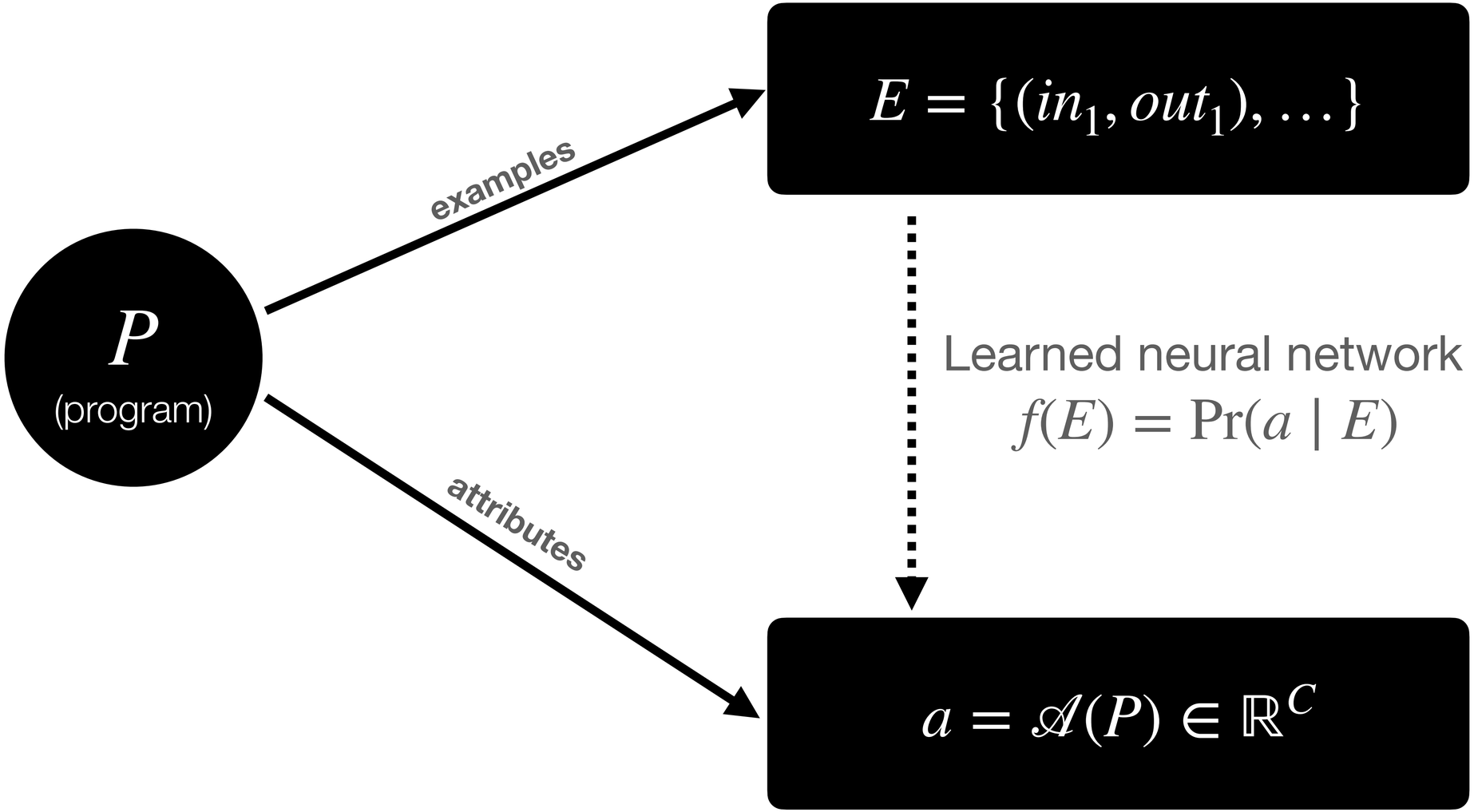
Generating Examples

I/O examples are represented by the set E

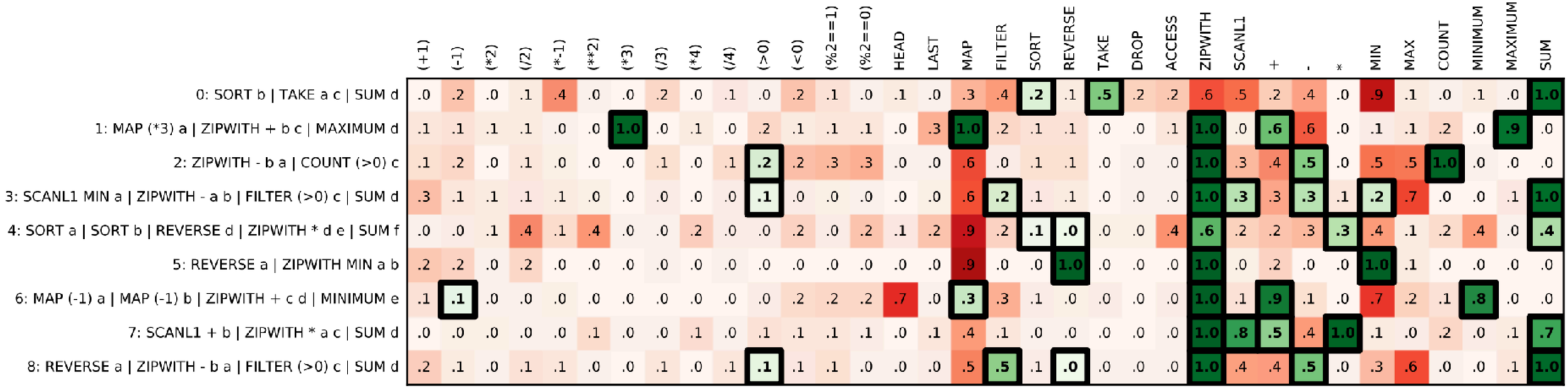


Training the Neural Network

Deep Learning = Black-Box Function Estimator



```
[
  {
    "program": "LIST|LIST|COUNT,>0,0|ZIPWITH,+,1,0|ACCESS,2,3",
    "examples": [
      {
        "inputs": [
          [-46, -23, -78, 10],
          [125, 105, -69]
        ],
        "output": 82
      },
      {
        "inputs": [
          [90, 103, -57, 13, -45, 28, -30, 68, -113, 60, -71,
            48, -117, 79, -42, -43, 37, -96],
          [13, -52, 48, 6, -8, -55, 35, 75]
        ],
        "output": null
      },
      // ...
    ],
    "attribute": [
      0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0,
      0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0
    ]
  },
  // ...
]
```


(+1) (-1) (*2) (/2) (*-1) (**2) (*3) (/3) (*4) (/4) (>0) (<0) (%2==1) (%2==0) HEAD LAST MAP FILTER SORT REVERSE TAKE DROP ACCESS ZIPWITH SCANL1 + - * MIN MAX COUNT MINIMUM MAXIMUM SUM
0: SORT b | TAKE a c | SUM d .0 .2 .0 .1 .4 .0 .0 .2 .0 .1 .0 .2 .1 .0 .1 .0 .3 .4 .2 .1 .5 .2 .2 .6 .5 .2 .4 .0 .9 .1 .0 .1 .0 1.0
1: MAP (*3) a | ZIPWITH + b c | MAXIMUM d .1 .1 .1 .1 .0 .0 1.0 .0 .1 .0 .2 .1 .1 .1 .0 .3 1.0 .2 .1 .1 .0 .0 .1 1.0 .0 .6 .6 .0 .1 .1 .2 .0 .9 .0
2: ZIPWITH - b a | COUNT (>0) c .1 .2 .0 .1 .0 .0 .0 .1 .0 .1 .2 .2 .3 .3 .0 .0 .6 .0 .1 .1 .0 .0 .0 1.0 .3 .4 .5 .0 .5 .5 1.0 .0 .0 .0
3: SCANL1 MIN a | ZIPWITH - a b | FILTER (>0) c | SUM d .3 .1 .1 .1 .1 .0 .0 .0 .0 .0 .1 .0 .0 .0 .0 .0 .6 .2 .1 .1 .0 .0 .0 1.0 .3 .3 .3 .1 .2 .7 .0 .0 .1 1.0
4: SORT a | SORT b | REVERSE d | ZIPWITH * d e | SUM f .0 .0 .1 .4 .1 .4 .0 .0 .2 .0 .0 .2 .0 .2 .1 .2 .9 .2 .1 .0 .0 .0 .4 .6 .2 .2 .3 .3 .4 .1 .2 .4 .0 .4
5: REVERSE a | ZIPWITH MIN a b .2 .2 .0 .2 .0 .0 .0 .0 .0 .0 .0 .0 .0 .0 .0 .0 .9 .0 .0 1.0 .0 .0 .0 1.0 .0 .2 .0 .0 1.0 .1 .0 .0 .0 .0
6: MAP (-1) a | MAP (-1) b | ZIPWITH + c d | MINIMUM e .1 .1 .0 .0 .0 .0 .0 .0 .0 .0 .0 .2 .2 .2 .7 .0 .3 .3 .1 .0 .0 .0 .0 1.0 .1 .9 .1 .0 .7 .2 .1 .8 .0 .0
7: SCANL1 + b | ZIPWITH * a c | SUM d .0 .0 .0 .0 .0 .1 .0 .0 .1 .0 .1 .1 .1 .1 .0 .1 .4 .1 .0 .0 .0 .0 .0 1.0 .8 .5 .4 1.0 .1 .0 .2 .0 .1 .7
8: REVERSE a | ZIPWITH - b a | FILTER (>0) c | SUM d .2 .1 .0 .1 .1 .0 .0 .1 .0 .1 .1 .1 .1 .0 .0 .0 .5 .5 .1 .0 .0 .0 .0 1.0 .4 .4 .5 .0 .3 .6 .0 .0 .1 1.0

“The key in the LIPS formulation is to ensure that it is feasible to generate a large dataset (ideally millions of programs).”

https://arxiv.org/pdf/1611.01989.pdf, p.3

Menon et al., 2013

A Machine Learning Framework for Programming By Example

(a) Sample of clues used. `LIST` denotes a list-, `E` a string-nonterminal in the grammar.

Feature	**Suggested rule(s)**
Substring s appears in output but not input?	`E` → "`s`", `LIST` → {`E`}
Duplicates in input but not output?	`LIST` → `dedup`(`LIST`)
Numbers on each input line but not output line?	`LIST` → `count`(`LIST`)

Dataset Generation

How to pick reasonable input-output pairs

- “We enforce a constraint on the output value bounding integers to some predetermined range, and then propagate these constraints backward through the program to obtain a range of valid values for each input.”
- Issues: maybe a couple random samples could be unrepresentative of the program as a whole?

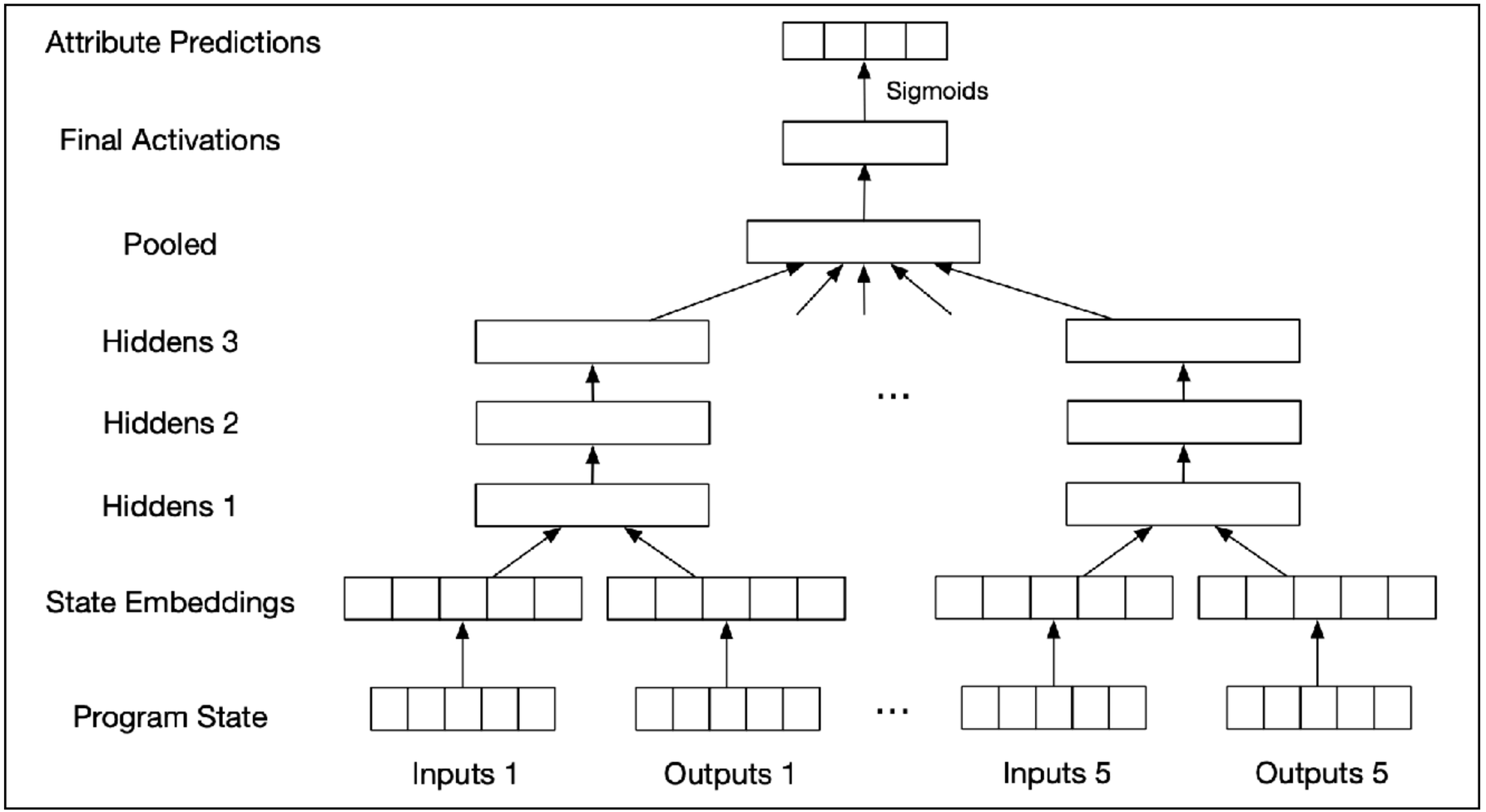
Architecture

Details of the neural network

- We want to learn a representative map $f : E \mapsto a$.
- Loss function: **mean binary cross-entropy.**
- Network architecture: **fully connected, 3-layer.**
  (Aside: Why not recurrent?)
- Embed each word in the input/output into a vector, combination of one-hot encoding for types, with learned embeddings for integers like “2”, “5”, “42”, up until some maximum representable value.
- Each I/O pair is mapped to an attribute vector, and these are averaged.

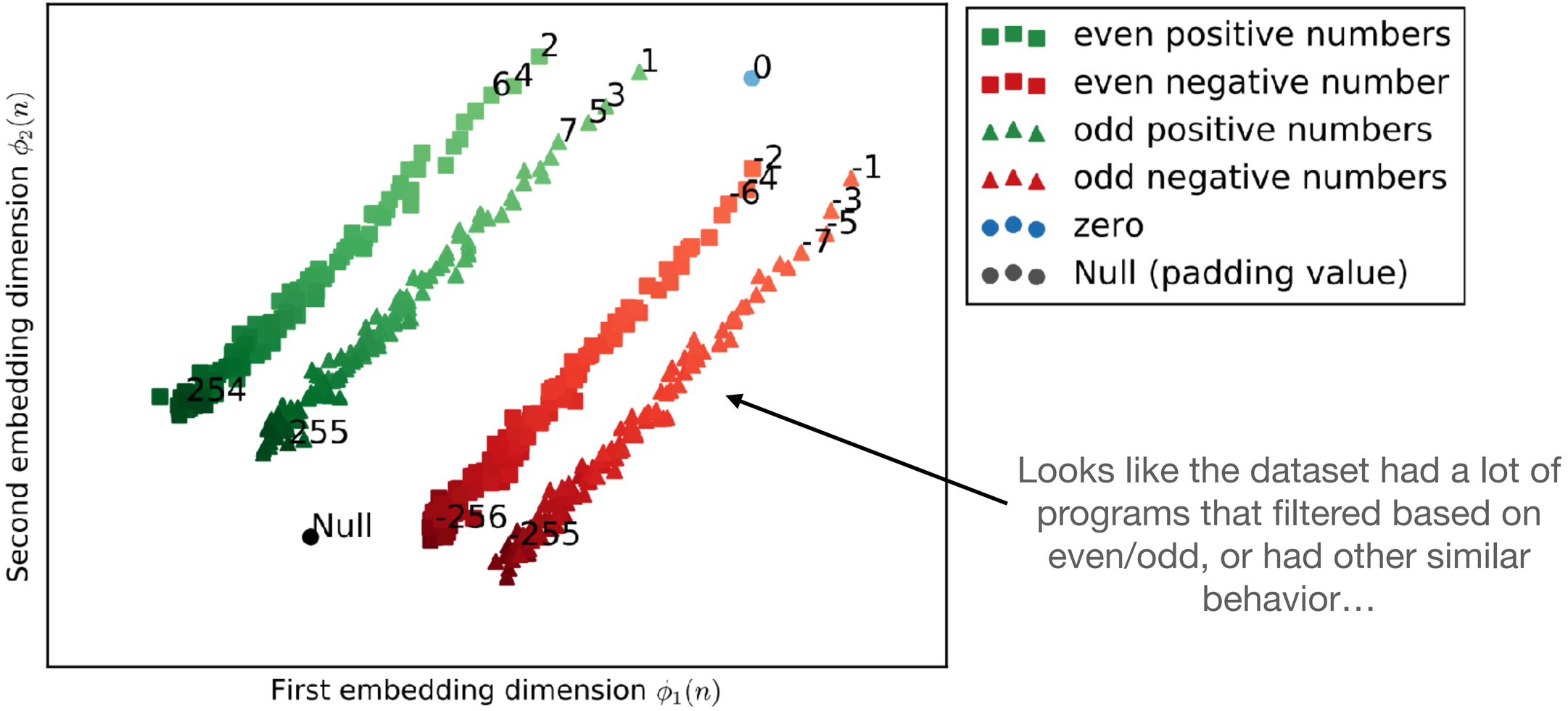
Architecture Schematic

Time-Distributed Feed-Forward Network + Average Pooling



Learned Embedding for [-256, 255]

Dense embedding in a 20-dimensional space



Looks like the dataset had a lot of programs that filtered based on even/odd, or had other similar behavior…

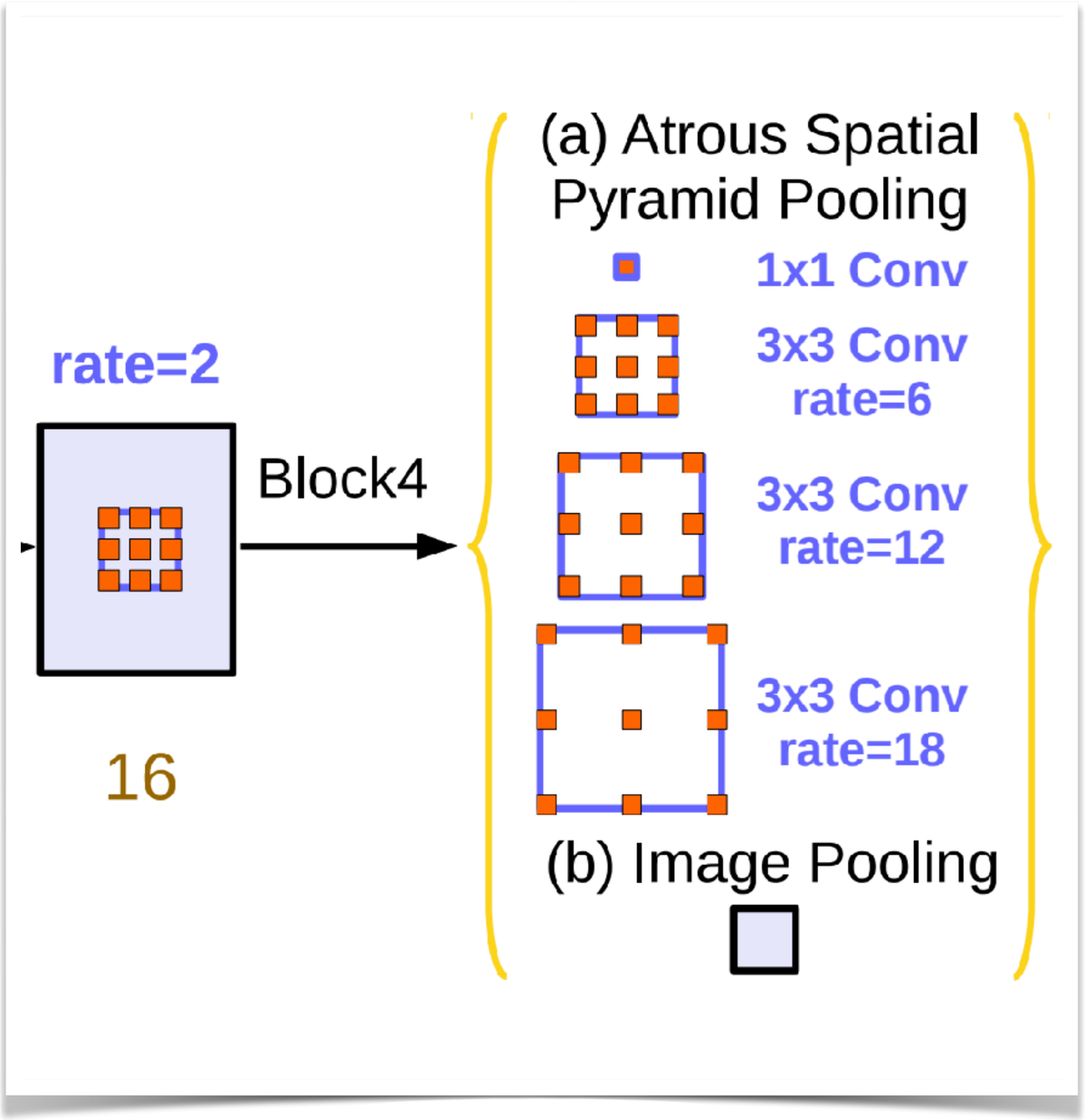
Thoughts: Extensions?

Slightly less minimal LIPS

- Encoding I/O elements
  - How would you encode other data types? Embeddings.
  - One-hot encoding of arrays versus RNN/Transformer encoding.
    - *(Note: authors tried using a GRU but couldn't get it to work well)*
- More specific attribute vectors
  - Context-sensitive: probability of each token with k-token lookbehind
  - Advantages? Disadvantages?

Other Encoder Architectures

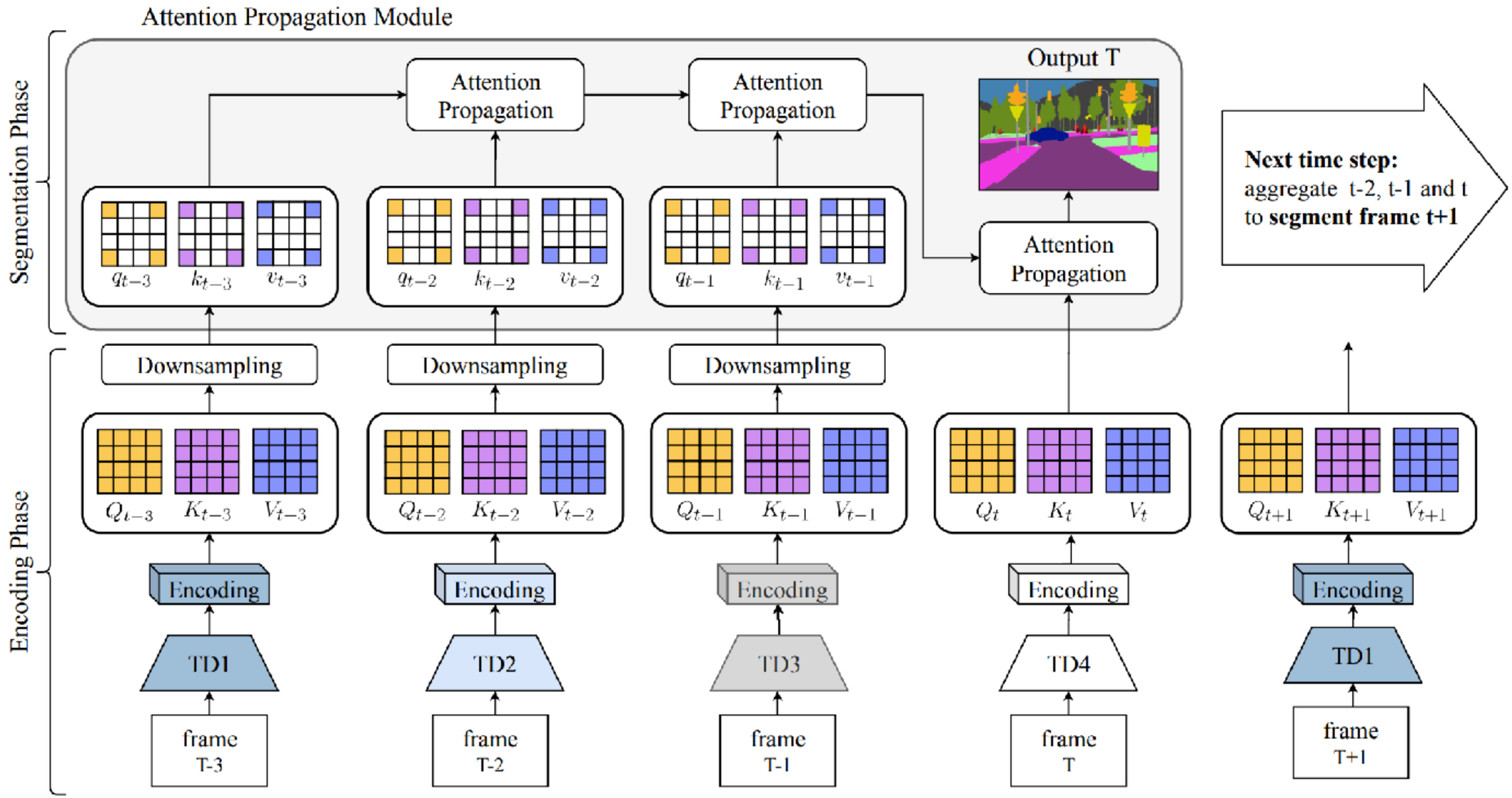
From the DL community



https://arxiv.org/pdf/1706.05587.pdf

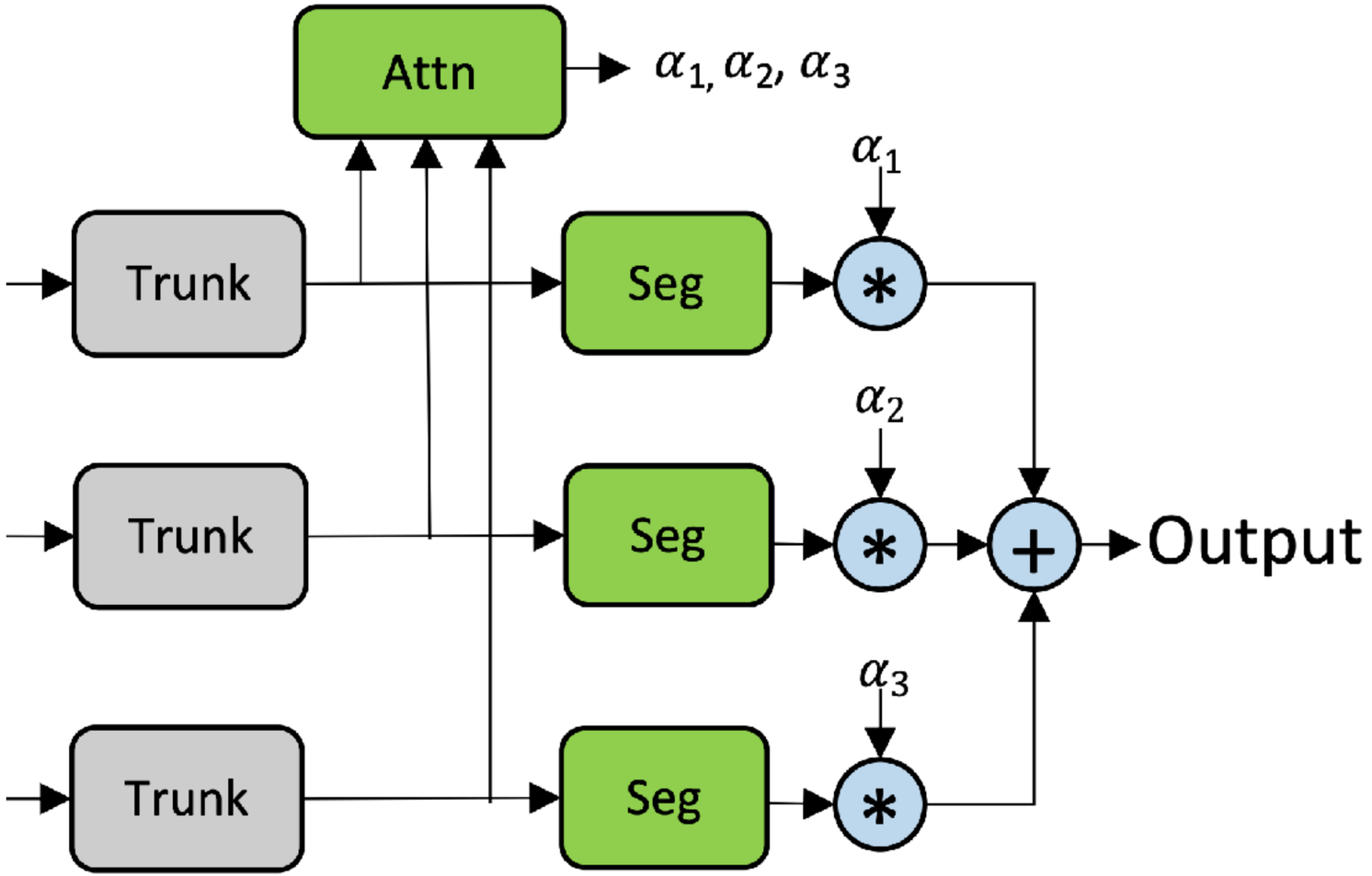
Attention

https://arxiv.org/pdf/2004.01800v2.pdf



Attention

https://arxiv.org/pdf/2005.10821.pdf



Specificity

Where's the right learning interface?

- Divide between "perceptual" and "symbolic".
- **Perceptual:** view E (the set of examples), generate easily-interpretable attribute vector a. More specific attributes are harder to interpret.
  - Extreme case is when a is a complex latent vector with no easy interpretation, and you just pass this into a neural translation algorithm.
  - Where have we seen this before?
- **Symbolic:** plug in a into the search algorithm, which tends to be easy even if the search algorithm wasn't *designed* necessarily for LIPS.

Discussion

Part I: Machine Learning

What are the advantages, disadvantages, and alternatives of:

1. Choice of attribute vectors.
2. Neural network (encoder) architecture.
3. Augmented algorithm versus E2E machine learning.

Part II: Search Algorithms

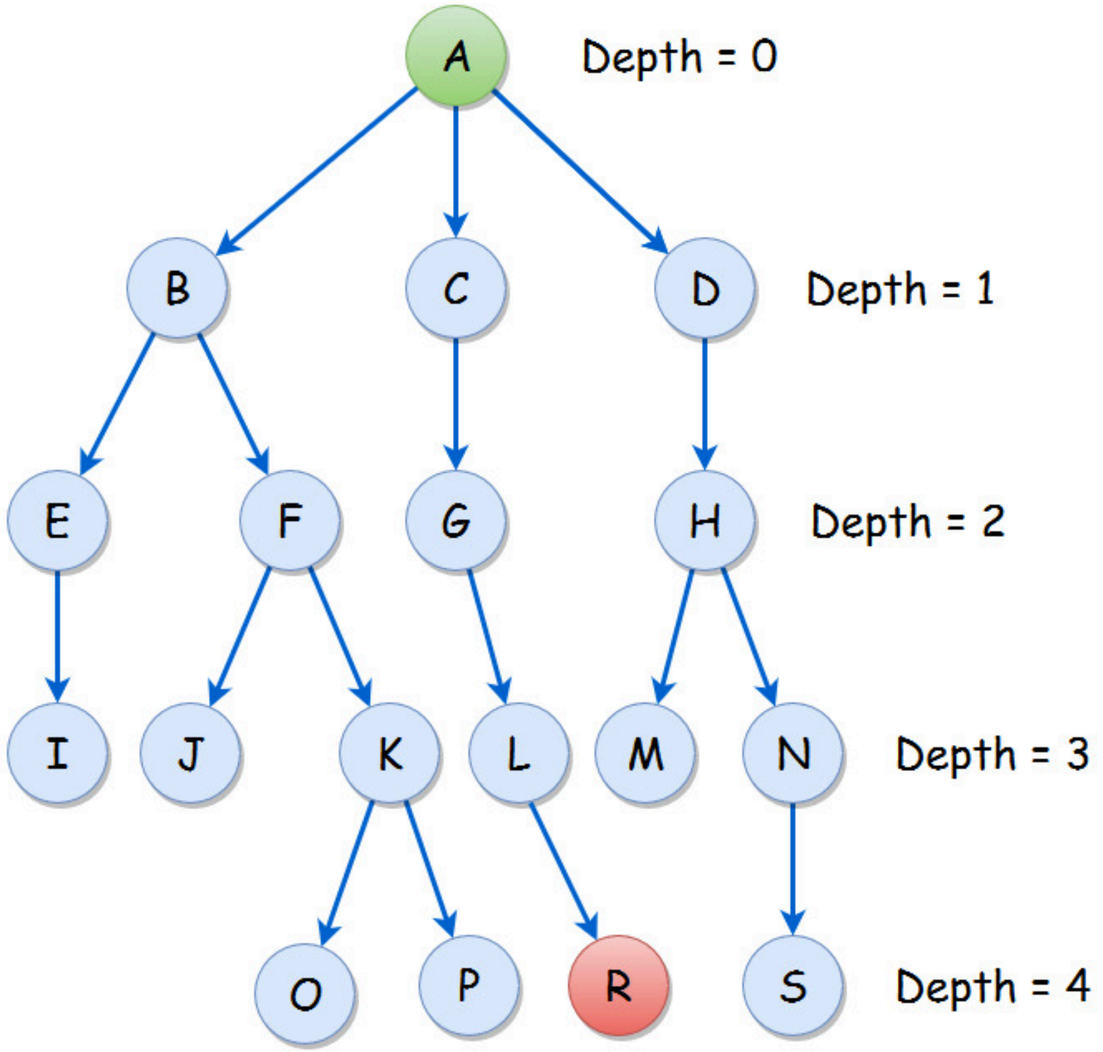
“We use the neural network’s predictions to ***augment*** search techniques from the programming languages community…”

https://arxiv.org/pdf/1611.01989.pdf, p.1 (emphasis mine)

Depth-First Search

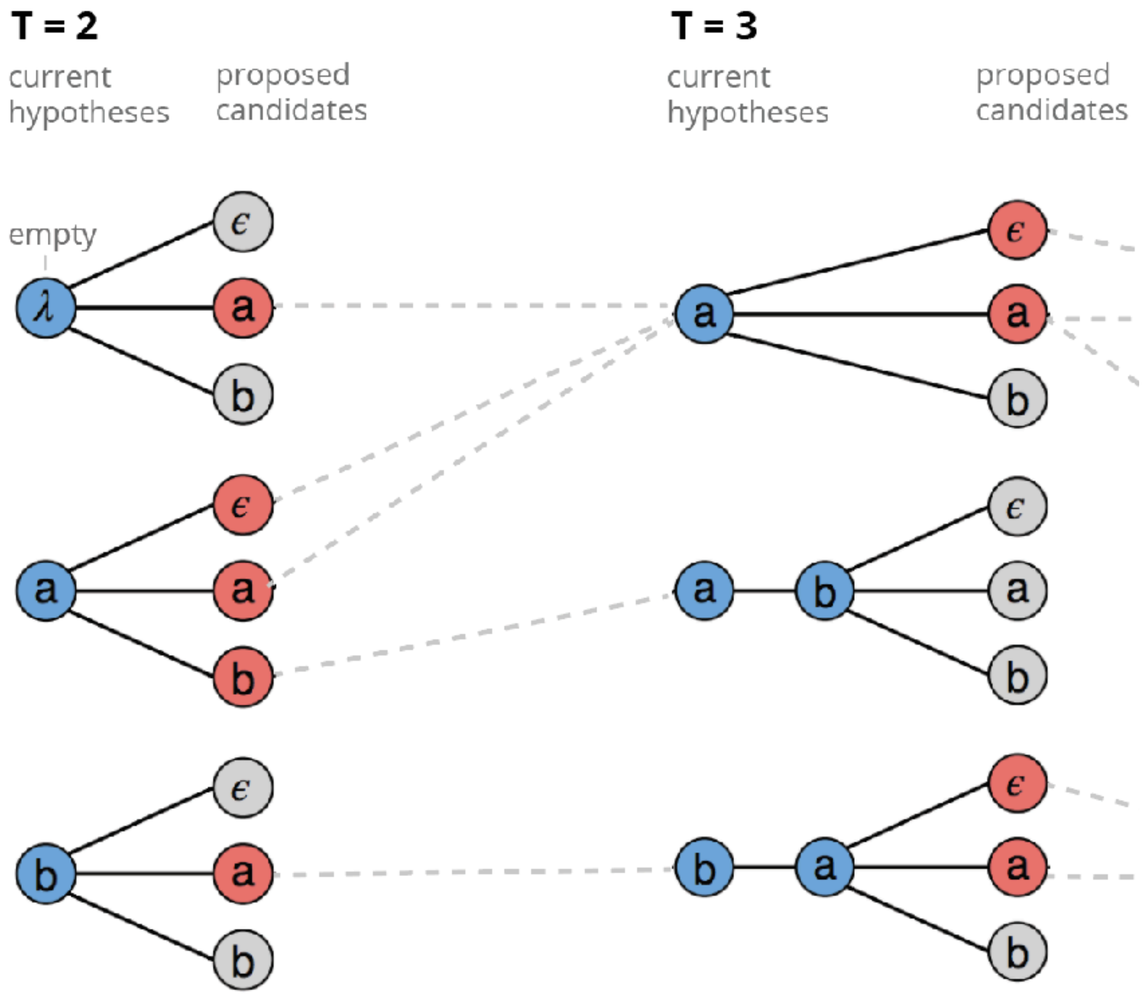
With Iterative Deepening

- Search through all programs of length $\leq T$.
- Considered a solution if we execute it on $M = 5$ examples and all are ✅.
- C++ implementation runs at 3×10^6 programs / second.



“Sort and Add”

Active function set reduces memory requirements



CTC beam search algorithm with an output alphabet {ε,a,b} and a beam size of three.

Mult
in to

Sketch

SMT-based synthesis tool, fills in “holes” in code

Synthesis | Published: 02 August 2012

Program sketching

Armando Solar-Lezama

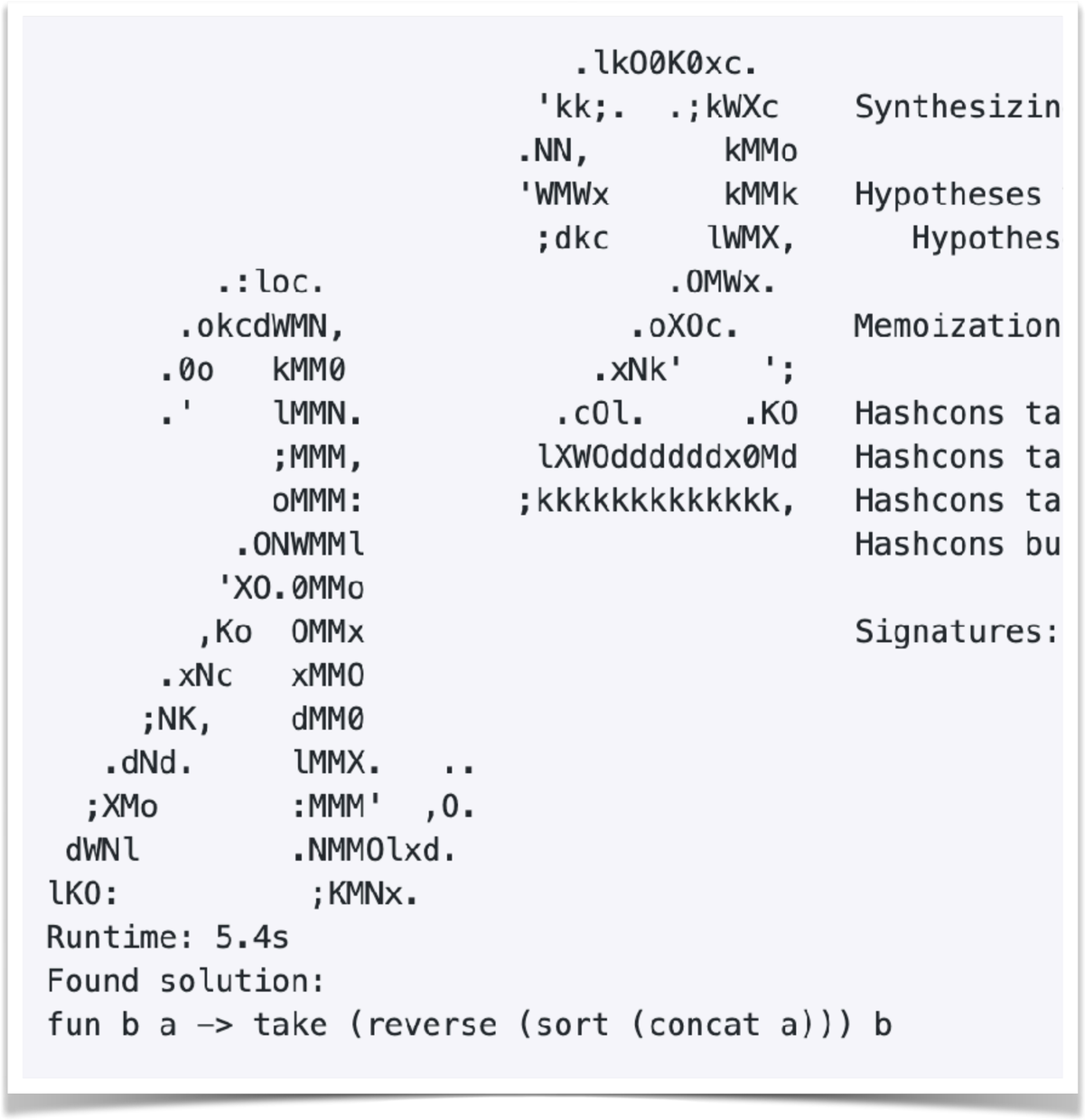
International Journal on Software Tools for Technology Transfer **15**, 475–495(2013) | Cite this article

Abstract

Sketching is a synthesis methodology that aims to bridge the gap between a programmer’s high-level insights about a problem and the computer’s ability to manage low-level details. In sketching, the programmer uses a partial program, a sketch, to describe the desired implementation strategy, and leaves the low-level details of the implementation to an automated synthesis procedure. In order to generate an implementation from the programmer provided sketch, the synthesizer uses counterexample-guided inductive synthesis (CEGIS). Inductive synthesis refers to the process of generating candidate implementations from concrete examples of correct or incorrect behavior. CEGIS combines a SAT-based inductive synthesizer with an automated validation procedure, a bounded model-checker, that checks whether the candidate implementation produced by inductive synthesis is indeed correct and to produce new counterexamples. The result is a synthesis procedure that is able to handle complex problems from a variety of domains including ciphers, scientific programs, and even concurrent data-structures.

λ^2

Enumerative search and deduction with a small library of functions



Alternative Decoders

Or “what didn’t work for them”

- RNN decoder predicts things token by token + beam search
  - “We combined this… by initializing the RNN using the *pooled final layer of the encoder*.”
  - (This might be a straw man; it’s not really “search” at this point.)
  - R3NN paper has a similar, but more reasonable comparison.
- **Q:** How far did this RNN sequence model get? (length T)

$$T \leq 2$$

RNN decoder: attributes → program

Discussion

Part II: Search Algorithms

1. What other search algorithms would you like to see, augmented with machine learning predictions?

Part III: Experiments

“These programs have been inspired by simple tasks appearing on ***real programming competition websites***.”

https://arxiv.org/pdf/1611.01989.pdf, p.12 (emphasis mine)

F. Realistic Gameplay

(Source: Codeforces 1430F)

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

Recently you've discovered a new shooter. They say it has realistic game mechanics.

Your character has a gun with magazine size equal to k and should exterminate n waves of monsters. The i-th wave consists of a_i monsters and happens from the l_i-th moment of time up to the r_i-th moments of time. All a_i monsters spawn at moment l_i and you have to exterminate all of them before the moment r_i ends (you can kill monsters right at moment r_i). For every two consecutive waves, the second wave starts not earlier than the first wave ends (though the second wave can start at the same moment when the first wave ends) — formally, the condition $r_i \le l_{i+1}$ holds. Take a look at the notes for the examples to understand the process better.

You are confident in yours and your character's skills so you can assume that aiming and shooting are instant and you need exactly one bullet to kill one monster. But reloading takes exactly 1 unit of time.

One of the realistic mechanics is a mechanic of reloading: when you reload you throw away the old magazine with all remaining bullets in it. That's why constant reloads may cost you excessive amounts of spent bullets.

You've taken a liking to this mechanic so now you are wondering: what is the minimum possible number of bullets you need to spend (both used and thrown) to exterminate all waves.

Note that you don't throw the remaining bullets away after eradicating all monsters, and you start with a full magazine.

Input

The first line contains two integers n and k ($1 \le n \le 2000$; $1 \le k \le 10^9$) — the number of waves and magazine size.

Examples

input

```
2 3
2 3 6
3 4 3
```

output

```
9
```

input

```
2 5
3 7 11
10 12 15
```

output

```
30
```

input

```
5 42
42 42 42
42 43 42
43 44 42
44 45 42
45 45 1
```

output

```
-1
```

Example Program

Sum of min k elements

Program 0:

```
k ← int
b ← [int]
c ← SORT b
d ← TAKE k c
e ← SUM d
```

Input-output example:

Input:

```
2, [3 5 4 7 5]
```

Output:

```
[7]
```

Example Program

Given 3-point wins and 1-point ties, find the top-scoring team

Program 1:

```
w ← [int]
t ← [int]
c ← MAP (*3) w
d ← ZIPWITH (+) c t
e ← MAXIMUM d
```

Input-output example:

Input:

```
[6 2 4 7 9],
[5 3 6 1 0]
```

Output:

```
27
```

Example Program

Given widths and heights, make rectangles of minimum total area

Program 4:

```
x ← [int]
y ← [int]
c ← SORT x
d ← SORT y
e ← REVERSE d
f ← ZIPWITH (*) d e
g ← SUM f
```

Input-output example:

Input:

```
[7 3 8 2 5],
[2 8 9 1 3]
```

Output:

```
79
```

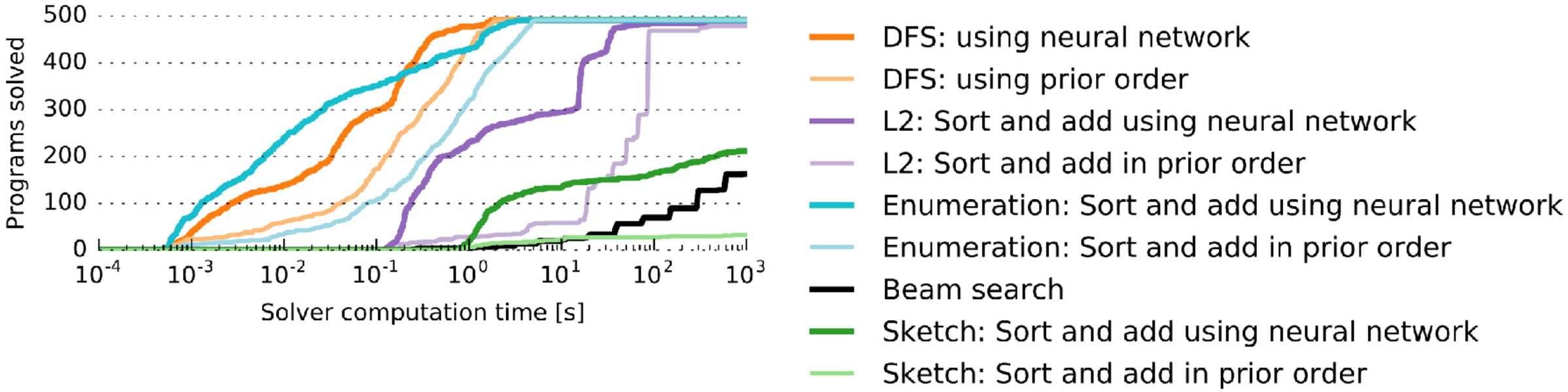
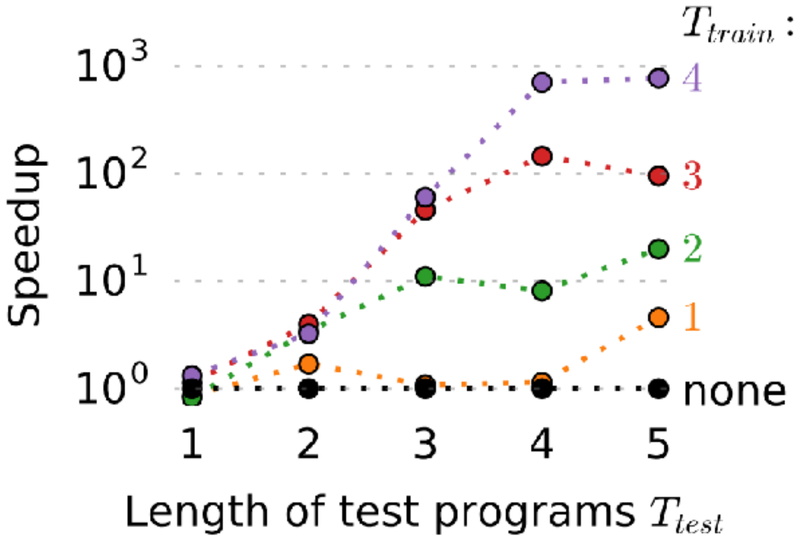


Figure 5: Number of test problems solved versus computation time.

Timeout needed	DFS			Enumeration			λ^2
to solve	20%	40%	60%	20%	40%	60%	20%
Baseline	$163s$	$2887s$	$6832s$	$8181s$	$>10^4s$	$>10^4s$	$463s$
DeepCoder	$24s$	$514s$	$2654s$	$9s$	$264s$	$4640s$	$48s$
Speedup	**6.8×**	**5.6×**	**2.6×**	**907×**	**>37×**	**>2×**	**9.6×**

(a)



(b)

Figure 3: Search speedups on programs of length $T = 5$ and influence of length of training programs.

Discussion

Part III: Experiments

1. What other experiments might you have wanted to see?
2. What was most surprising in the results?
3. What could be some practical applications for learned program synthesis, given these results?

https://arxiv.org/abs/1611.01989