Programming With A Differentiable Forth Interpreter

Varun Gangal, CMU
Based on the work of
Matko Bosnjak et al

What's Forth?

- Kind of like a **cross between Python and Assembly**
- **High-level** imperative programming language **BUT**
- Can manipulate **registers**, **stack exposed**, load-stores
- **It's nice!** because it is **close to natural language** (even Python is), but **without assuming many layers** of abstraction or compiling below (exposes stack etc)
- **It's dangerous**! No type-checking, no scope, no data-code separation, no mem.management

Reverse Polish Notation

- **Postfix** as opposed to **infix** notation
- **Simple** notion of **precedence**, no **lookahead**
- 3 4 + ; not 3+4; 234*+ not 2+3*4
- No arguments or return values, no stack management
- One stack for all functions to operate on.
- Stack operations: **SWAP, DROP, DUP**
- Advantages: Super-fast execution, compilation

Example Code in Forth

```
: SORT ( a1 .. an n -- sorted )
  1- DUP 0 DO >R R@ BUBBLE R> LOOP DROP
;
2 4 2 7 4 SORT \ Example call
```

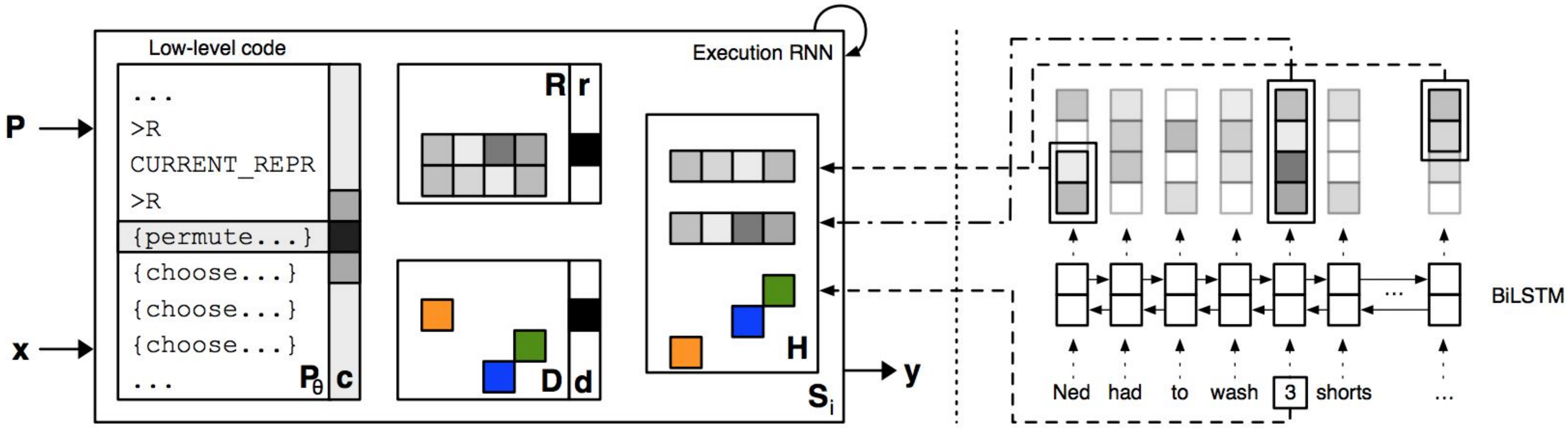
- Literals pushed to DSTACK
- Call SORT, PC pushed to RSTACK
- TOS = Top of Stack, NOS = End of Stack
- 1- deducts TOS by 1. DUP duplicates TOS etc etc

Quotable Quotes

- “If C gives you enough rope to hang yourself with, FORTH is a flamethrower crawling with cobras”

Program State in Forth

1. DStack **D** : All operations,
2. RStack **R** : Return address, Buffer stack
3. Heap **H**
4. Program counter **c**: Next statement to be executed



P
x
Low-level code
...
>R
CURRENT_REPR
>R
{permute...}
{choose...}
{choose...}
{choose...}
...
Pθ
c
R
r
D
d
Execution RNN
H
Si
y
BiLSTM
Ned
had
to
wash
3
shorts
...

Partial Procedural Knowledge

- How to visit a sequence
- How to traverse a tree
- **Sketch** : An incompletely specified code fragment.
- Provide a procedural prior
- Recollect rule templates from last time - kind of like that

What our model includes

1. Does the job of the **compiler** (**maintain** and **update program state**)
2. Takes in **inputs** (also inits program state with them)
3. Takes in **partially specified programs** a.k.a **sketches**
4. Learns learnable part of the programs
5. Trained on **input-output pairs**
6. Point 1 grants us end-to-end differentiability
7. It also makes our reads, writes, PC soft (uncertain)

What are we trying to do here?

- Program statement = Transition function f: S -> S
- Program = Transition Composition
- Output = Program(Input) -> Program encodes prior
- Sketches (more in detail later) : Incompletely specified statements/functions - sort of like rule templates from the logic stuff last time
- In this paper, all the transition functions are differentiable. The NN model is the compiler.

Let’s kind of walkthrough a Forth program - Bubble Sort

```
: BUBBLE ( a1 ... an n-1 -- one pass )
    DUP IF >R
        OVER OVER < IF SWAP THEN
        R> SWAP >R 1- BUBBLE R>
        { observe D0 D-1 -> permute D-1 D0 R0}
        1- BUBBLE R>
        { observe D0 D-1 -> choose NOP SWAP }
        R> SWAP >R 1- BUBBLE R>
    ELSE
        DROP
    THEN
;
: SORT ( a1 .. an n -- sorted  )
  1- DUP 0 DO >R R@ BUBBLE R> LOOP DROP
;
2 4 2 7 4 SORT \ Example call
```

Just focus on the green lines for now! - Other 2 are sketches

	D	R	c	comment
1	[]	[]	11	execution start
2	[2 4 2 7 4]	[]	8	pushing sequence to D, calling SORT subroutine puts A_{SORT} to R
3	[2 4 2 7 3]	[A_{SORT}]	9	1-
4	[2 4 2 7 3 3]	[A_{SORT}]	9	DUP
6	[2 4 2 7 3 3 0]	[A_{SORT}]	9	0
7	[2 4 2 7 3]	[$Addr_{SORT}$]	9	DO
8	[2 4 2 7]	[$Addr_{SORT}$ 3]	9	>R
9	[2 4 2 7 3]	[$Addr_{SORT}$ 3]	9	@R

Before the function call; Loop

10	[2 4 2 7 3]	[A_{SORT} 3 A_{BUBBLE}]	0	calling BUBBLE subroutine puts A_{BUBBLE} to R
11	[2 4 2 7 3 3]	[A_{SORT} 3 A_{BUBBLE}]	1	DUP
12	[2 4 2 7 3]	[A_{SORT} 3 A_{BUBBLE}]	1	IF
13	[2 4 2 7]	[A_{SORT} 3 A_{BUBBLE} 3]	1	>R

14	[2 4 2 7 2 7]	[A_{SORT} 3 A_{BUBBLE} 3]	2	OVER OVER
15	[2 4 2 7 1]	[A_{SORT} 3 A_{BUBBLE} 3]	2	<
16	[2 4 2 7]	[A_{SORT} 3 A_{BUBBLE} 3]	2	IF
17	[2 4 7 2]	[A_{SORT} 3 A_{BUBBLE} 3]	2	SWAP

18	[2 4 7 2 3]	[A_{SORT} 3 A_{BUBBLE}]	3	R >
19	[2 4 7 3 2]	[A_{SORT} 3 A_{BUBBLE}]	3	SWAP
20	[2 4 7 3]	[A_{SORT} 3 A_{BUBBLE} 2]	3	>R
21	[2 4 7 2]	[A_{SORT} 3 A_{BUBBLE} 2]	3	1-
22	[2 4 7 2]	[A_{SORT} 3 A_{BUBBLE} 2]	0	...BUBBLE

Inside the Bubble Routine

$$\text{read}_{\mathbf{M}}(\mathbf{a}) = \mathbf{a}^T \mathbf{M}$$

$$\text{write}_{\mathbf{M}}(\mathbf{x}, \mathbf{a}) : \mathbf{M} \leftarrow \mathbf{M} - (\mathbf{a}\mathbf{1}^T) \odot \mathbf{M} + \mathbf{x}\mathbf{a}^T$$

$$\text{inc}(\mathbf{p}) = \mathbf{p}^T \mathbf{R}^{\mathbf{1}+}$$

$$\text{dec}(\mathbf{p}) = \mathbf{p}^T \mathbf{R}^{-}$$

Primitives - read, write, shift-increment, shift-decrement

$$\text{push}_{\mathbf{M}}(\mathbf{x}) : \text{write}_{\mathbf{M}}(\mathbf{x},\mathbf{p}) \qquad \text{(side-effect: } \mathbf{p} \leftarrow \text{inc}(\mathbf{p}))$$

$$\text{pop}_{\mathbf{M}}(\,) = \text{read}_{\mathbf{M}}(\mathbf{p}) \qquad \text{(side-effect: } \mathbf{p} \leftarrow \text{dec}(\mathbf{p}))$$

Conditional jump $\mathbf{a}$

$$jump(\mathbf{c},\mathbf{a}) : p = (pop_{\mathbf{D}}() = TRUE)$$
$$\mathbf{c} \leftarrow p\mathbf{c} + (1-p)a$$

Composites -push, pop

DUP	$push_{\mathbf{D}}(read_{\mathbf{D}}(\mathbf{d}))$
SWAP	$x = read_{\mathbf{D}}(\mathbf{d}), y = read_{\mathbf{D}}(\mathbf{d}^{-1})$
	$: write_{\mathbf{D}}(\mathbf{d}, y), write_{\mathbf{D}}(\mathbf{d}^{-1}, x)$
OVER	$push_{\mathbf{D}}(read_{\mathbf{D}}(\mathbf{d}))$
DROP	$pop_{\mathbf{D}}()$
+, -, *, /	$write_{\mathbf{D}}(\{op\}(read_{\mathbf{D}}(\mathbf{d}^{-1}), read_{\mathbf{D}}(\mathbf{d})), \mathbf{d})$
IF $.._1$ELSE $.._2$THEN	$p = (pop_{\mathbf{D}}() = \mathbf{0})$
	$p * .._1 + (1-p) * .._2$

Composites - OVER, DUP, SWAP, IF.. ELSE

observe $e_1 ... e_m$

choose $w_1 ... w_m$

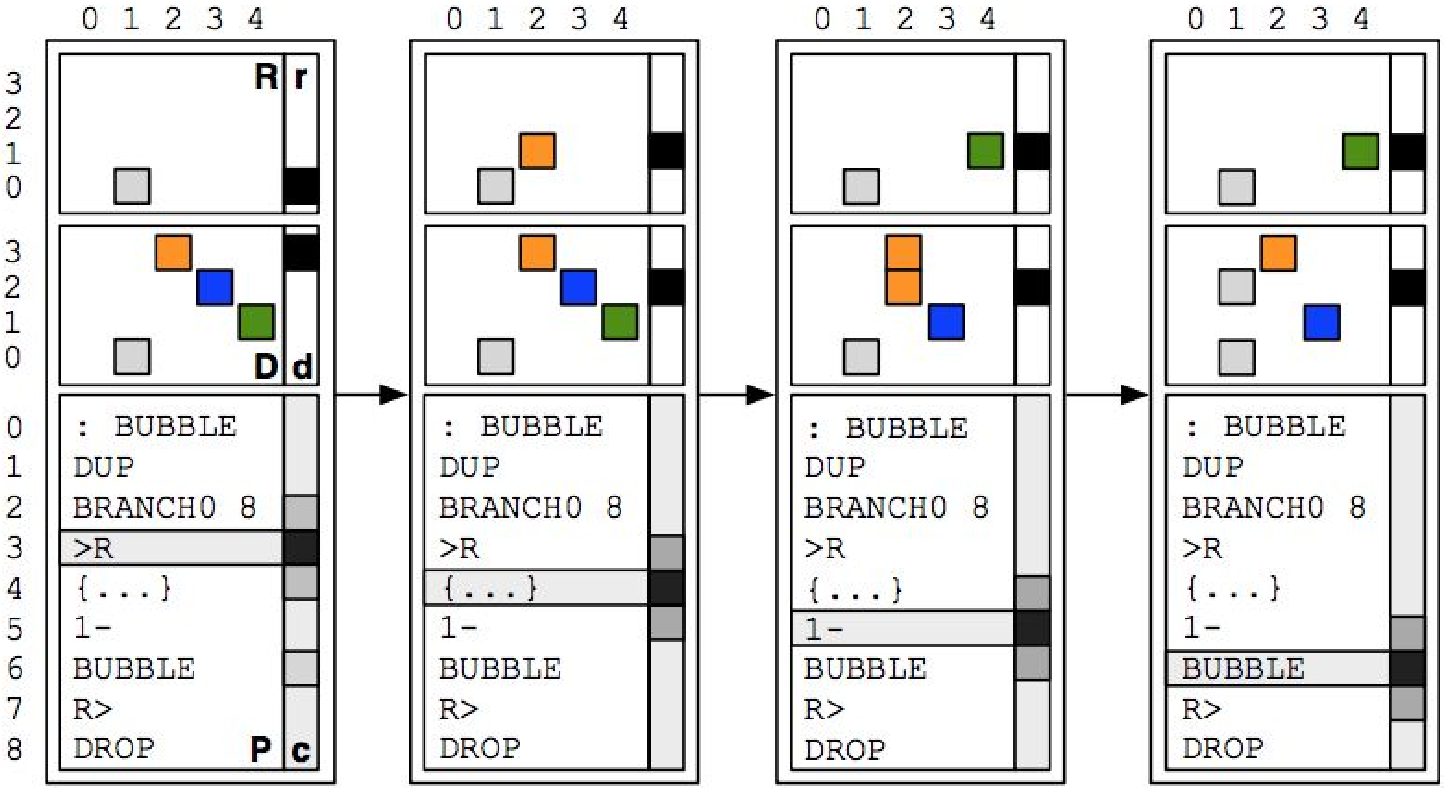
manipulate $e_1 ... e_m$

permute $e_1 ... e_m$

Sketches - Partial transition funcs, enc and dec specified

$$\mathbf{S}_{n+1} = \mathrm{RNN}(\mathbf{S}_n, \mathbf{P}_\theta) = \sum_{i=1}^{|P|} \mathbf{c}_i \mathbf{w}_i(\mathbf{S}_n)$$

Execution - use program counter as attention vector



Traces - Discrete Init, later everything's soft

Optimizations - For shorter gradient paths, faster training

- When no entry-exit, get composite transition function (symbolically)

Training

1. Training is based based on final stack state and stack pointer.
2. Includes a mask (to consider only elements <stack depth).

$$\mathcal{L}(\theta) = \mathcal{H}(\mathbf{K}_i \odot \mathbf{D}_T(\theta, \mathbf{x}_i), \mathbf{K}_i \odot \mathbf{Y}_i^D) \\ + \mathcal{H}(\mathbf{K}_i \odot \mathbf{d}_T(\theta, \mathbf{x}_i), \mathbf{K}_i \odot \mathbf{y}_i^d)$$

$$\mathcal{H}(\mathbf{x}, \mathbf{y}) = -\mathbf{x} \log \mathbf{y}$$

Sorting

Table 1: Accuracy (Hamming distance) of Permute and Compare sketches in comparison to a Seq2Seq baseline on the sorting problem.

	Test Length 8			Test Length: 64		
Train Length:	2	3	4	2	3	4
Seq2Seq	26.2	29.2	39.1	13.3	13.6	15.9
∂4 Permute	100.0	100.0	19.82	100.0	100.0	7.81
∂4 Compare	100.0	100.0	49.22	100.0	100.0	20.65

Word Problems Dataset - Examples

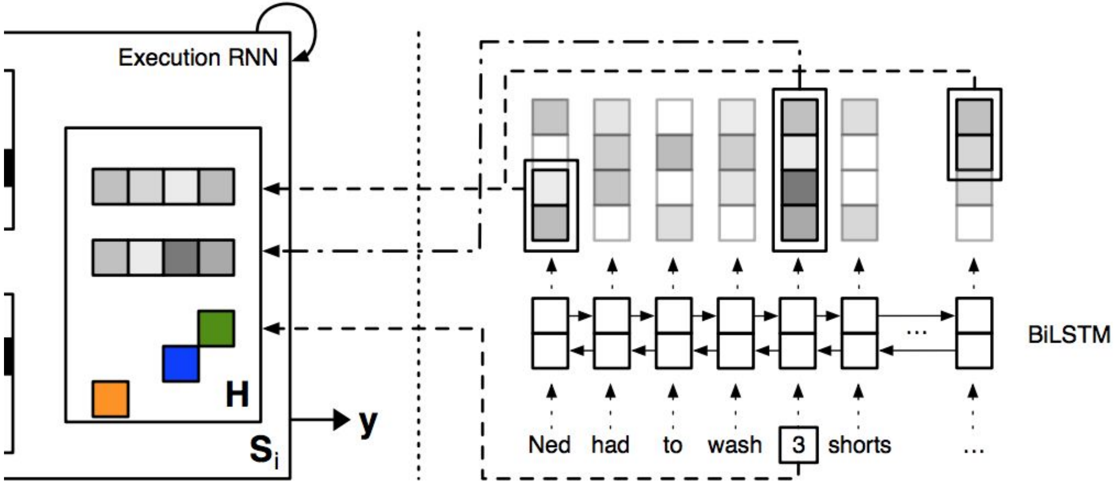
A florist had 50 roses. If she sold 15 of them and then later picked 21 more, how many roses would she have?

Ryan has 72 marbles and 17 blocks. If he shares the marbles among 9 friends, how many marbles does each friend get?

- Roy & Roth '15. *CC*. 4 basic operators, upto 3 operands
- Prior approaches map to expressions *e.g* **(50-15)+21**
- This one solves directly
- About 150 each for train, dev, test

Encoding the question

- BiLSTM to encode the question
- What's used: States corresponding to numbers, and the final state, also numbers themselves

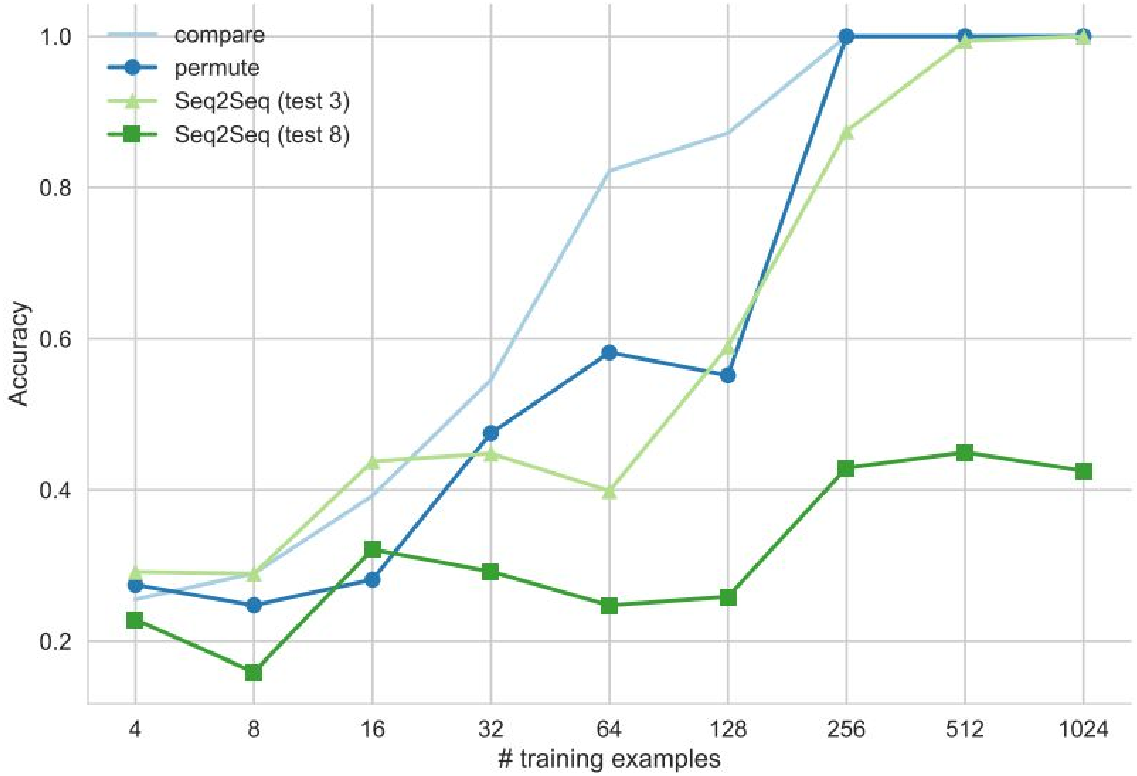


```
\ permute stack elements, based on the question and number representations
{ observe R0 R-1 R-2 R-3 -> permute D0 D-1 D-2 }
\ choose the first operation
{ observe R0 R-1 R-2 R-3 -> choose + - * / }
\ choose whether to swap intermediate result and the bottom number
{ observe R0 R-1 R-2 R-3 -> choose SWAP NOP }
\ choose the second operation
{ observe R0 R-1 R-2 R-3 -> choose + - * / }
```

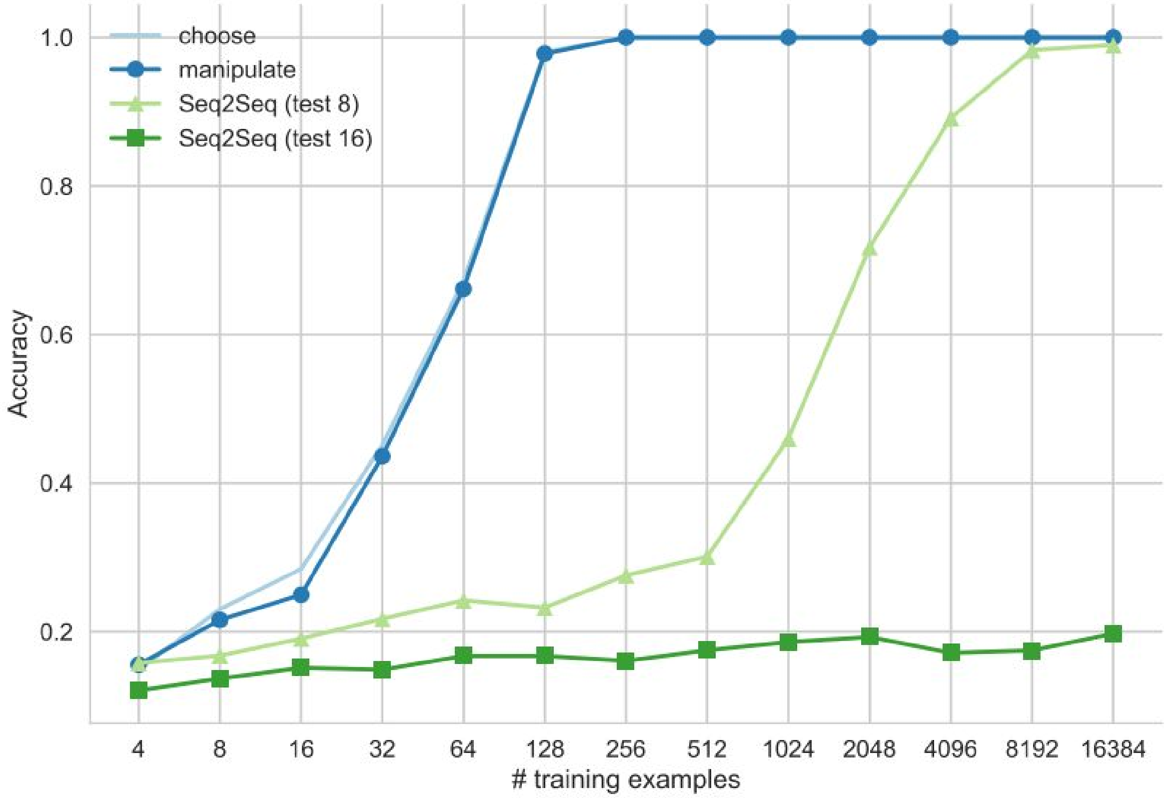
Key part of Word Problem Sketch

Model	Accuracy (%)
Template Mapping	
Roy & Roth (2015)	55.5
Seq2Seq* (Bouchard et al., 2016)	95.0
GeNeRe* (Bouchard et al., 2016)	98.5
Fully End-to-End	
$\partial 4$	96.0

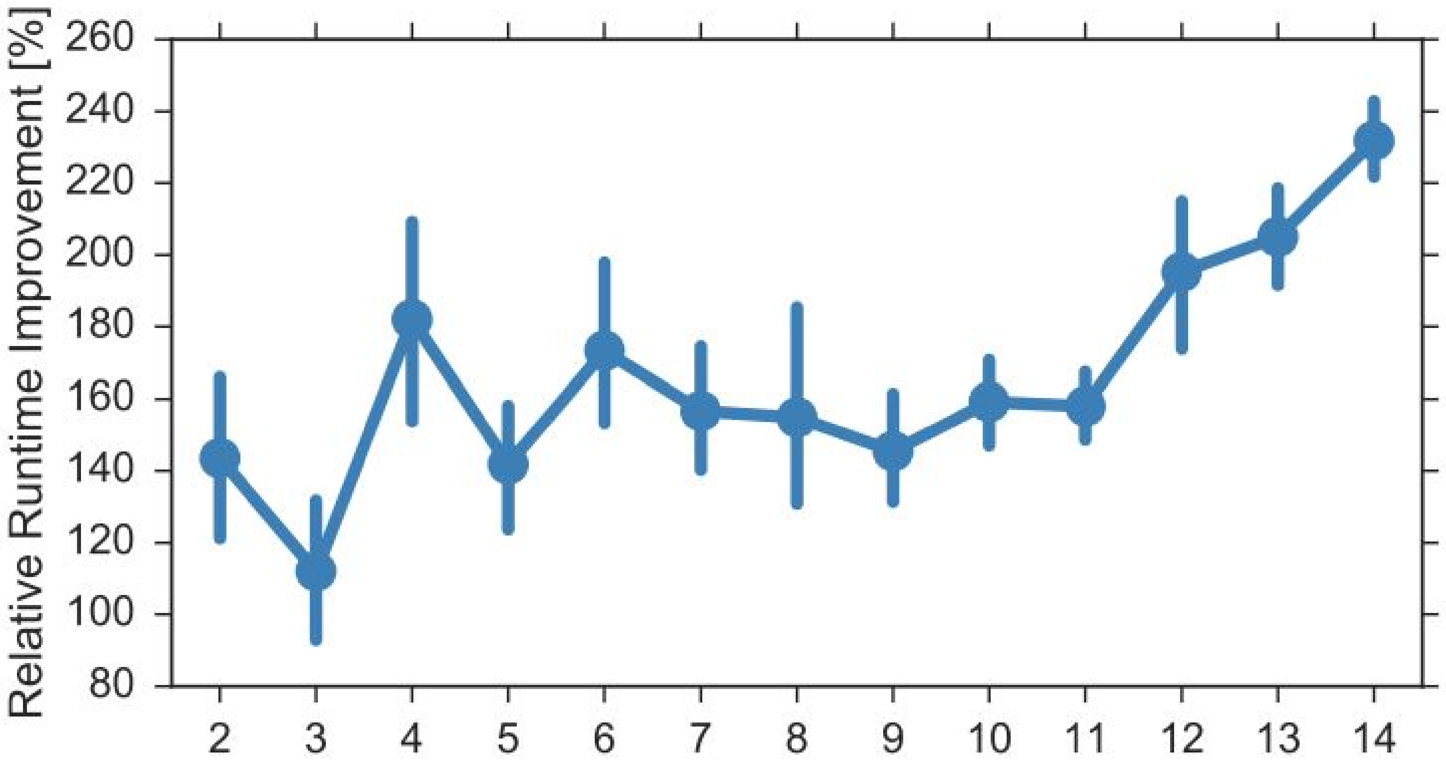
Results - Beats S2S Baseline



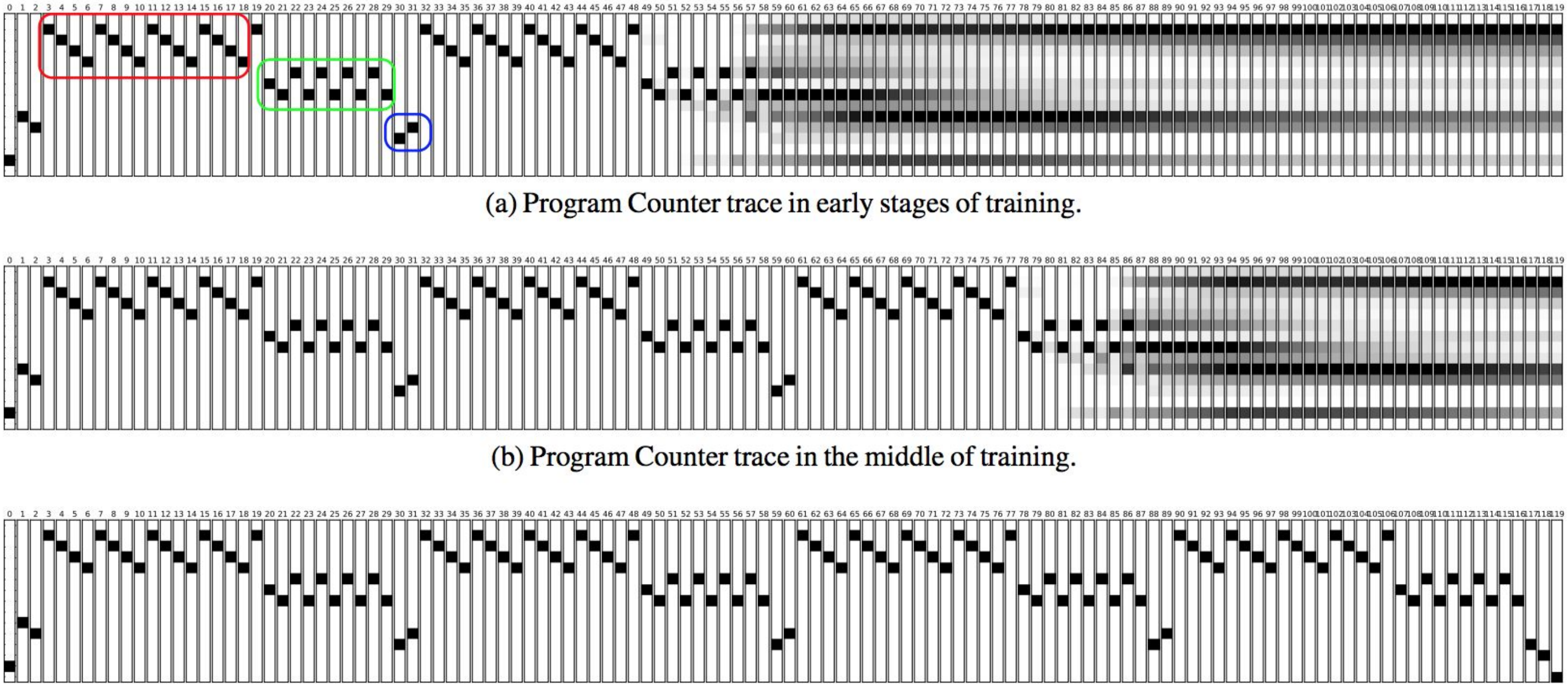
Sketch-based Models generalize well across lengths - Sorting



Sketch-based Models generalize well across lengths - Adding



Do the optimizations help?



(a) Program Counter trace in early stages of training.

(b) Program Counter trace in the middle of training.

(c) Program Counter trace at the end of training.

How the PC was trained

```
\ address of the question on H
VARIABLE QUESTION
\ allotting H for representations and numbers
CREATE REPR_BUFFER 4 ALLOT
CREATE NUM_BUFFER 4 ALLOT
\ addresses of the first representation and number
VARIABLE REPR
VARIABLE NUM

REPR_BUFFER REPR !
NUM_BUFFER NUM !

\ macro function for incrementing the pointer to numbers in H
MACRO: STEP_NUM
  NUM @ 1+ NUM !
;

\ macro function for incrementing the pointer to representations in H
MACRO: STEP_REPR
  REPR @ 1+ REPR !
;

\ macro functions for fetching current numbers and representations
MACRO: CURRENT_NUM NUM @ @ ;
MACRO: CURRENT_REPR REPR @ @ ;

\ copy numbers to D
CURRENT_NUM
STEP_NUM
CURRENT_NUM
STEP_NUM
CURRENT_NUM

\ copy question vector, and representations of numbers to R
QUESTION @ >R
CURRENT_REPR >R
STEP_REPR
CURRENT_REPR >R
STEP_REPR
CURRENT_REPR >R
```