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

```
9  : SORT ( a1 .. an n -- sorted )
10  1- DUP 0 DO >R R@ BUBBLE R> LOOP DROP
11  ;
12  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
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 $f: S \to S$
- **Program** = Transition Composition
- **Output** = Program(Input) $\to$ 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
<table>
<thead>
<tr>
<th></th>
<th>(D)</th>
<th>(R)</th>
<th>(c)</th>
<th>comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>([])</td>
<td>([])</td>
<td>11</td>
<td>execution start</td>
</tr>
<tr>
<td>2</td>
<td>[2 4 2 7 4]</td>
<td>[([)</td>
<td>8</td>
<td>pushing sequence to (D), calling SORT subroutine puts (A_{\text{SORT}}) to (R)</td>
</tr>
<tr>
<td>3</td>
<td>[2 4 2 7 3]</td>
<td>[(A_{\text{SORT}})]</td>
<td>9</td>
<td>1-</td>
</tr>
<tr>
<td>4</td>
<td>[2 4 2 7 3 3]</td>
<td>[(A_{\text{SORT}})]</td>
<td>9</td>
<td>DUP</td>
</tr>
<tr>
<td>6</td>
<td>[2 4 2 7 3 3 0]</td>
<td>[(A_{\text{SORT}})]</td>
<td>9</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>[2 4 2 7 3]</td>
<td>[(\text{Addr}_{\text{SORT}})]</td>
<td>9</td>
<td>DO</td>
</tr>
<tr>
<td>8</td>
<td>[2 4 2 7]</td>
<td>[(\text{Addr}_{\text{SORT}} 3)]</td>
<td>9</td>
<td>&gt;R</td>
</tr>
<tr>
<td>9</td>
<td>[2 4 2 7 3]</td>
<td>[(\text{Addr}_{\text{SORT}} 3)]</td>
<td>9</td>
<td>@R</td>
</tr>
</tbody>
</table>

Before the function call; Loop
Inside the Bubble Routine

10 | [2 4 2 7 3] | [ASORT 3 A BUBBLE] | 0 | calling BUBBLE subroutine puts A BUBBLE to R
11 | [2 4 2 7 3 3] | [ASORT 3 A BUBBLE] | 1 | DUP
12 | [2 4 2 7 3] | [ASORT 3 A BUBBLE] | 1 | IF
13 | [2 4 2 7] | [ASORT 3 A BUBBLE 3] | 1 | >R
14 | [2 4 2 7 2 7] | [ASORT 3 A BUBBLE 3] | 2 | OVER OVER
15 | [2 4 2 7 1] | [ASORT 3 A BUBBLE 3] | 2 | <
16 | [2 4 2 7] | [ASORT 3 A BUBBLE 3] | 2 | IF
17 | [2 4 7 2] | [ASORT 3 A BUBBLE 3] | 2 | SWAP
18 | [2 4 7 2 3] | [ASORT 3 A BUBBLE] | 3 | R >
19 | [2 4 7 3 2] | [ASORT 3 A BUBBLE] | 3 | SWAP
20 | [2 4 7 3] | [ASORT 3 A BUBBLE 2] | 3 | >R
21 | [2 4 7 2] | [ASORT 3 A BUBBLE 2] | 3 | 1-
22 | [2 4 7 2] | [ASORT 3 A BUBBLE 2] | 0 | ...BUBBLE
$$\text{read}_M(a) = a^T M$$

$$\text{write}_M(x, a): M \leftarrow M - (a1^T) \odot M + xa^T$$

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

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

Primitives - read, write, shift-increment, shift-decrement
$$\text{push}_M(x):\text{write}_M(x,p)$$  
(side-effect: $p \leftarrow \text{inc}(p)$)

$$\text{pop}_M() = \text{read}_M(p)$$  
(side-effect: $p \leftarrow \text{dec}(p)$)

Conditional jump $a$

$$\text{jump}(c,a): p = (\text{pop}_D() = \text{TRUE})$$
$$c \leftarrow pc + (1-p)a$$

**Composites** - push, pop
DUP
SWAP

OVER
DROP

+, -, *, /

IF..₁ELSE..₂THEN

\[
p = (\text{pop}_D() = 0) \\
p * ..₁ + (1 - p) * ..₂
\]

Composites - OVER, DUP, SWAP, IF.. ELSE
observe $e_1 \ldots e_m$

choose $w_1 \ldots w_m$

manipulate $e_1 \ldots e_m$

permute $e_1 \ldots e_m$

Sketches - Partial transition funcs, enc and dec specified
Execution - use program counter as attention vector

\[ S_{n+1} = \text{RNN}(S_n, P_\theta) = \sum_{i=1}^{\vert P \vert} c_i w_i(S_n) \]
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 on final stack state and stack pointer.
2. Includes a mask (to consider only elements < stack depth).

\[
\mathcal{L}(\theta) = \mathcal{H}(K_i \odot D_T(\theta, x_i), K_i \odot Y_i^D) \\
+ \mathcal{H}(K_i \odot d_T(\theta, x_i), K_i \odot y_i^d)
\]

\[
\mathcal{H}(x, y) = -x \log y
\]
## Sorting

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

<table>
<thead>
<tr>
<th>Train Length:</th>
<th>Test Length 8</th>
<th>Test Length: 64</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Seq2Seq</td>
<td>26.2</td>
<td>29.2</td>
</tr>
<tr>
<td>(\partial^4) Permute</td>
<td>100.0</td>
<td>100.0</td>
</tr>
<tr>
<td>(\partial^4) Compare</td>
<td>100.0</td>
<td>100.0</td>
</tr>
</tbody>
</table>
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
\{ \textbf{observe} \ R0 \ R-1 \ R-2 \ R-3 \rightarrow \textbf{permute} \ D0 \ D-1 \ D-2 \}  
\choose the first operation
\{ \textbf{observe} \ R0 \ R-1 \ R-2 \ R-3 \rightarrow \textbf{choose} \ + \ - \ * \ / \}  
\choose whether to swap intermediate result and the bottom number
\{ \textbf{observe} \ R0 \ R-1 \ R-2 \ R-3 \rightarrow \textbf{choose} \ SWAP \ NOP \}  
\choose the second operation
\{ \textbf{observe} \ R0 \ R-1 \ R-2 \ R-3 \rightarrow \textbf{choose} \ + \ - \ * \ / \}  

Key part of Word Problem Sketch
<table>
<thead>
<tr>
<th>Model</th>
<th>Accuracy (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Template Mapping</strong></td>
<td></td>
</tr>
<tr>
<td>Roy &amp; Roth (2015)</td>
<td>55.5</td>
</tr>
<tr>
<td>Seq2Seq* (Bouchard et al., 2016)</td>
<td>95.0</td>
</tr>
<tr>
<td>GeNeRe* (Bouchard et al., 2016)</td>
<td>98.5</td>
</tr>
<tr>
<td><strong>Fully End-to-End</strong></td>
<td></td>
</tr>
<tr>
<td>$\delta 4$</td>
<td>96.0</td>
</tr>
</tbody>
</table>

Results - Beats S2S Baseline
Sketch-based Models generalize well across lengths - Sorting
Sketch-based Models generalize well across lengths - Adding
Do the optimizations help?
How the PC was trained

(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.
\ address of the question on H
VARIBLE QUESTION
\ allotting H for representations and numbers
CREATE REPR_BUFFER 4 ALLOT
CREATE NUM_BUFFER 4 ALLOT
\ addresses of the first representation and number
VARIBLE REPR
VARIBLE 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