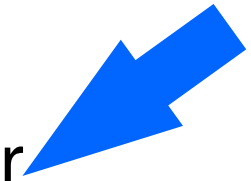
A Not that Comprehensive Introduction to Neural Programming

Lili Mou
doublepower.mou@gmail.com

Outline

- Introduction
- Symbolic execution
  - (Fully supervised) Neural programmer interpreter
  - (Weakly supervised) Neural symbolic machine
  - “Spurious programs” and inductive programming
    - Learning semantic parsers from denotations
    - DeepCoder
    - More thoughts on “spurious programs”
- Distributed execution
  - Learning to execute
  - Neural enquirer
  - Differentiable neural computer
- Hybrid execution
  - Neural programmer
  - Coupling approach

Outline



- Introduction
- Symbolic execution
  - (Fully supervised) Neural programmer interpreter
  - (Weakly supervised) Neural symbolic machine
  - “Spurious programs” and inductive programming
    - Learning semantic parsers from denotations
    - DeepCoder
    - More thoughts on “spurious programs”
- Distributed execution
  - Learning to execute
  - Neural enquirer
  - Differentiable neural computer
- Hybrid execution
  - Neural programmer
  - Coupling approach

NEURAL PROGRAMMER-INTERPRETERS

Scott Reed & **Nando de Freitas**
Google DeepMind
London, UK
scott.ellison.reed@gmail.com
nandodefreitas@google.com

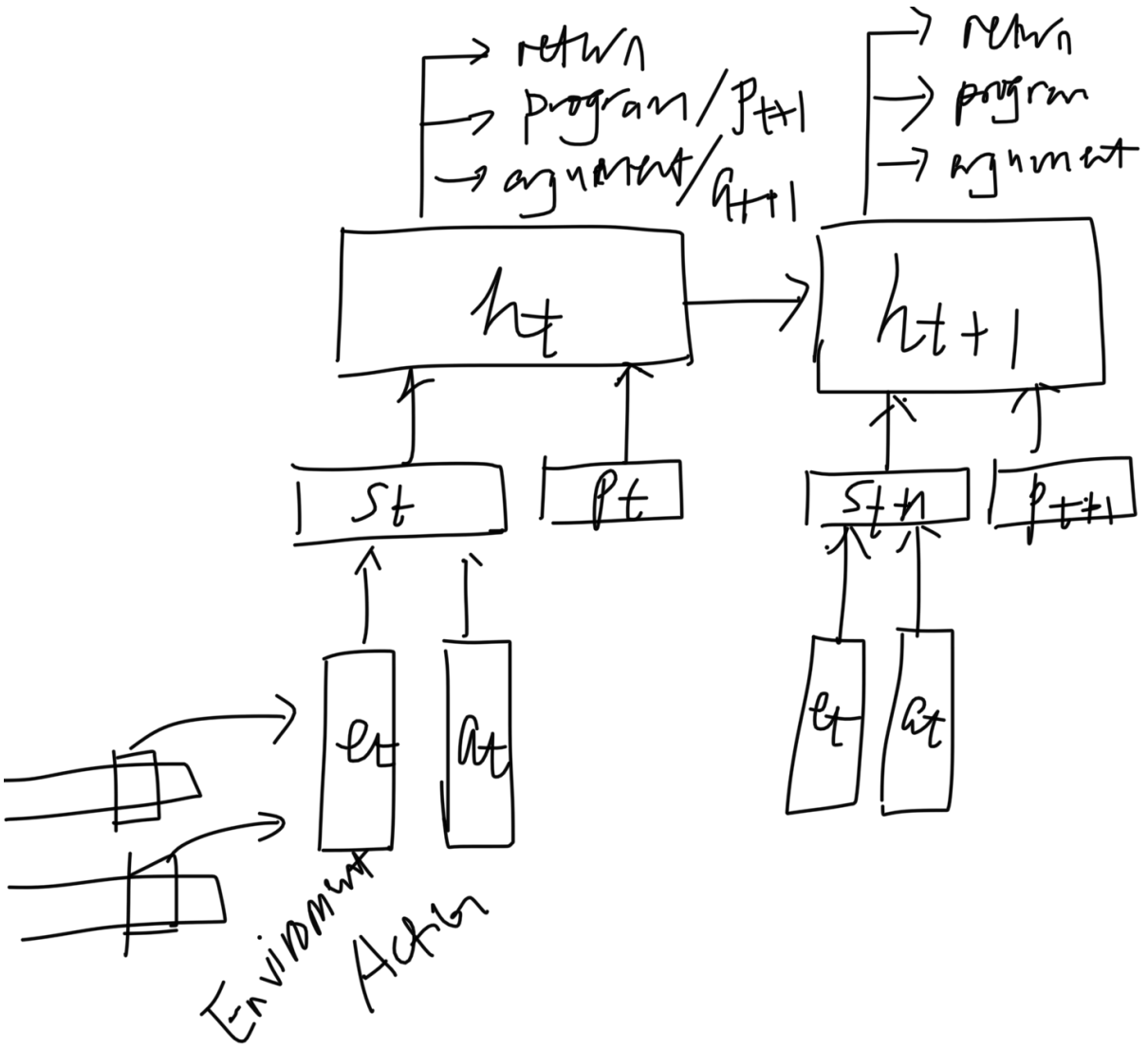
MAKING NEURAL PROGRAMMING ARCHITECTURES GENERALIZE VIA RECURSION

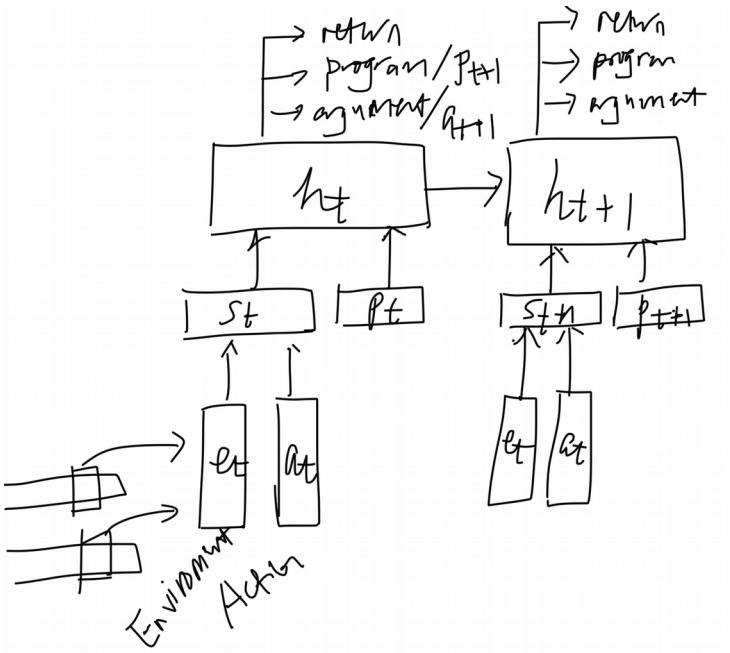
Jonathon Cai, Richard Shin, Dawn Song
Department of Computer Science
University of California, Berkeley
Berkeley, CA 94720, USA
{jonathon,ricshin,dawnsong}@cs.berkeley.edu

$$s_t = f_{enc}(e_t, a_t)$$

$$h_t = f_{lstm}(s_t, p_t, h_{t-1})$$

$$r_t = f_{end}(h_t), p_{t+1} = f_{prog}(h_t), a_{t+1} = f_{arg}(h_t)$$





When the program is called, the current context including the caller's memory state is stored on a stack; when the program returns, the stored context is popped off the stack to resume execution in the previous caller's context.

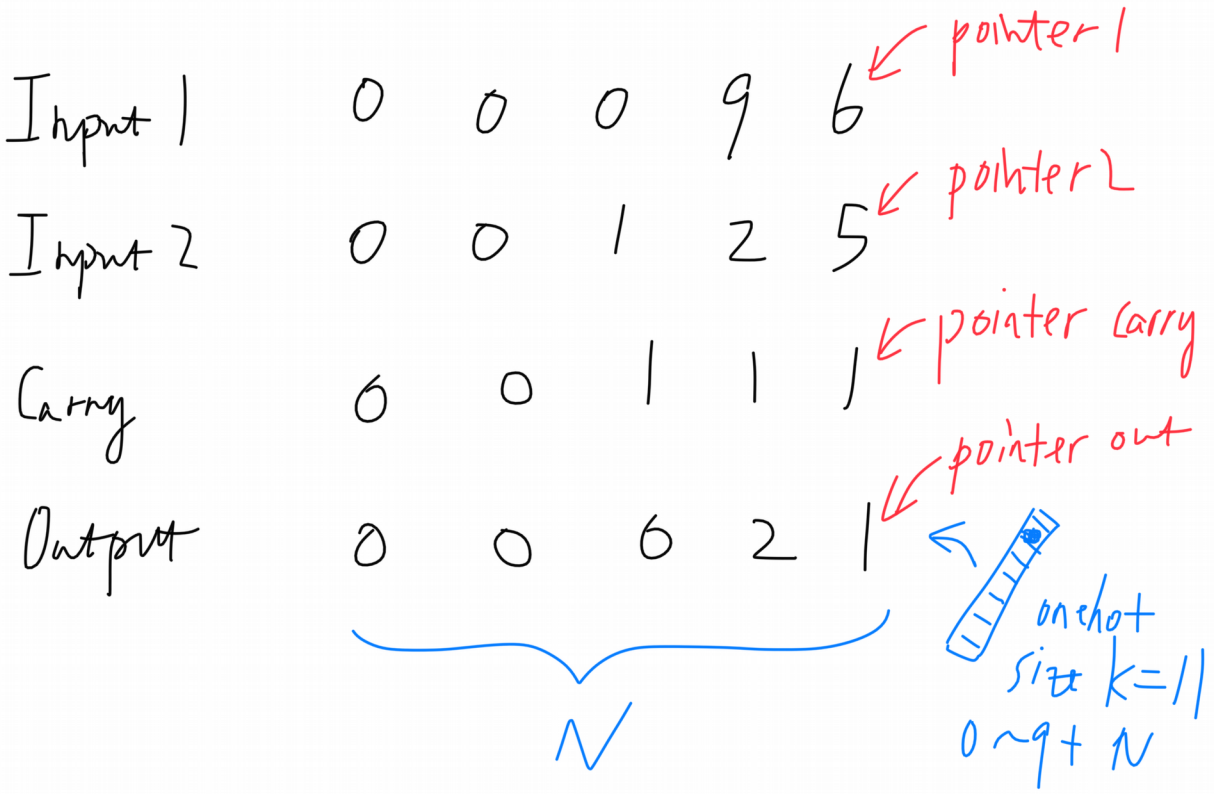
Algorithm 1 Neural programming inference

Inputs: Environment observation e, program p, arguments a, stop threshold α
function RUN(e, p, a)
    $h \leftarrow \mathbf{0}, r \leftarrow 0$
    while $r < \alpha$ **do**
        $s \leftarrow f_{enc}(e, a), h \leftarrow f_{lstm}(s, p, h)$
        $r \leftarrow f_{end}(h), p_2 \leftarrow f_{prog}(h), a_2 \leftarrow f_{arg}(h)$
        if p is a primitive function **then**
            $e \leftarrow f_{env}(e, p, a)$.
        else
            function RUN(e, p_2, a_2)

Example

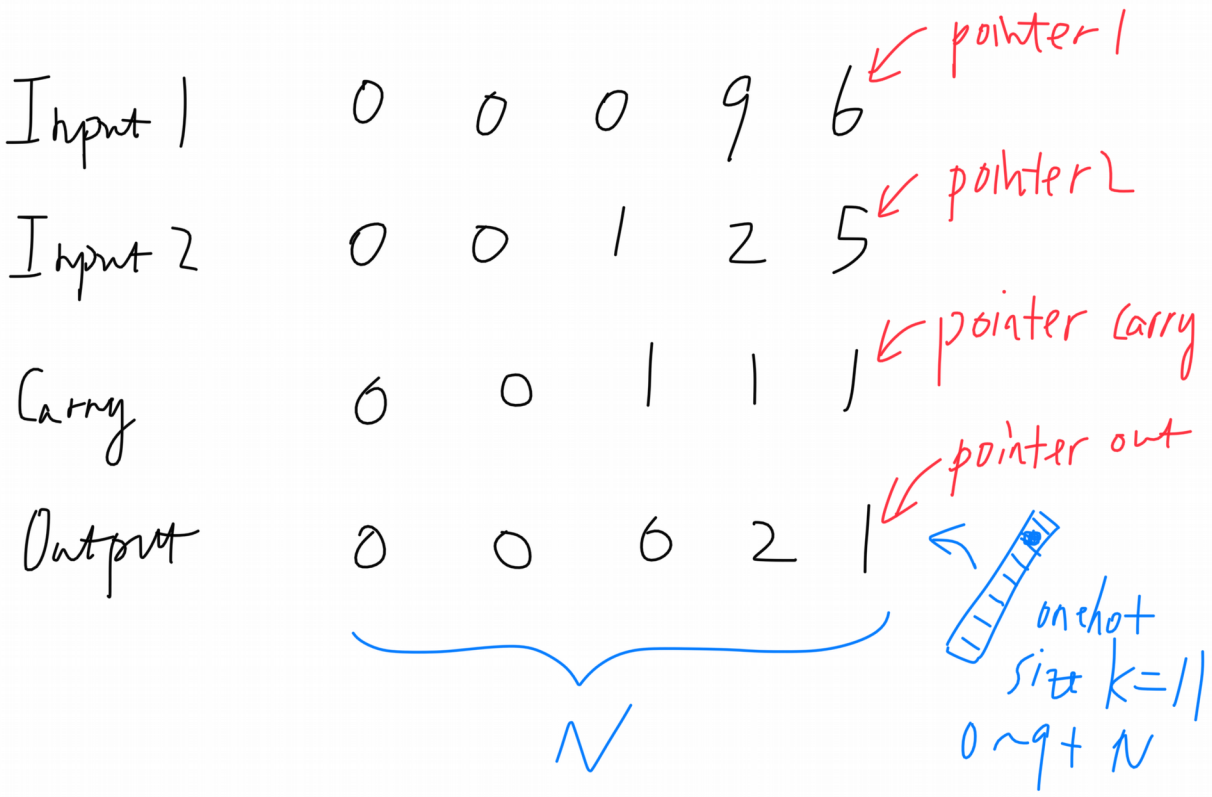
- Grade-school addition

$$f_{enc}(Q, i_1, i_2, i_3, i_4, a_t) = MLP([Q(1, i_1), Q(2, i_2), Q(3, i_3), Q(4, i_4), a_t(1), a_t(2), a_t(3)]),$$



Example

- Grade-school addition

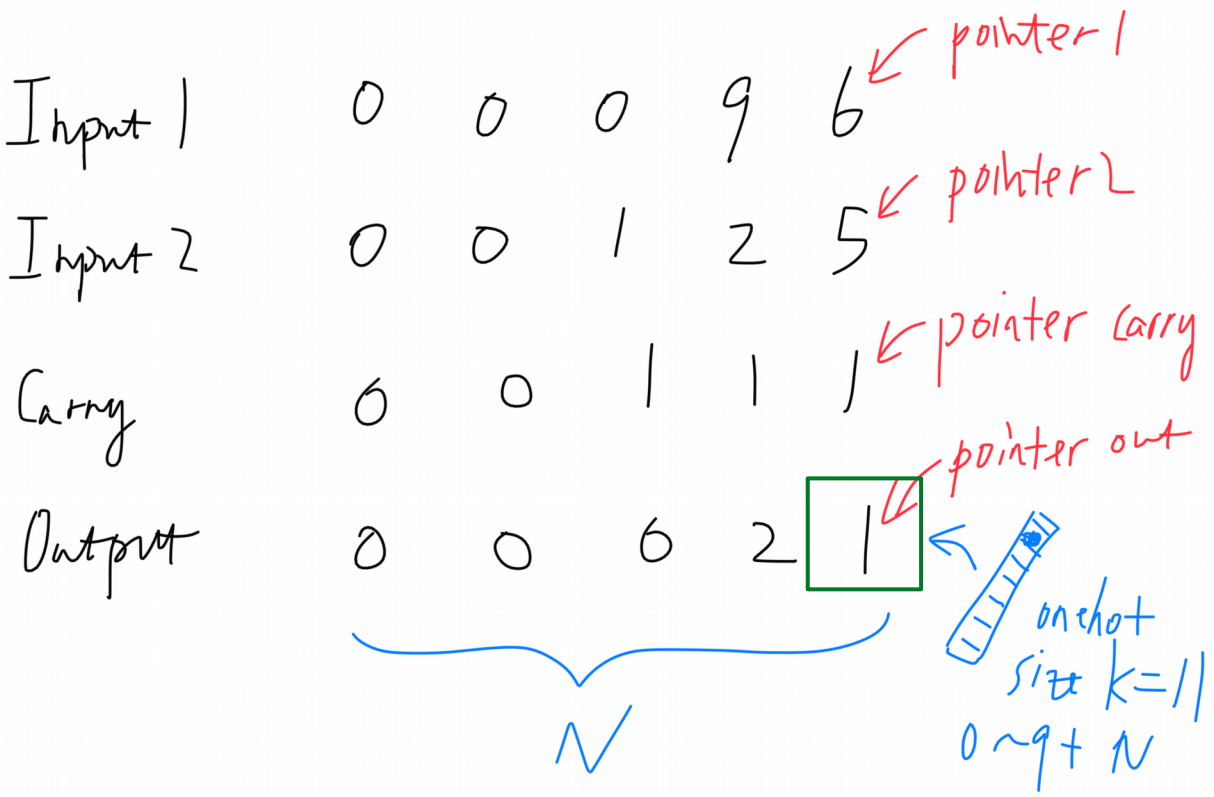



```
ADD Add for one digit (a wrapper)
ADD1
   WRITE OUT 1
   CARRY
      PTR CARRY LEFT
      WRITE CARRY 1
      PTR CARRY RIGHT
LSHIFT
   PTR INP1 LEFT
   PTR INP2 LEFT
   PTR CARRY LEFT
   PTR OUT LEFT
ADD1
   ...
```

Example

- Grade-school addition

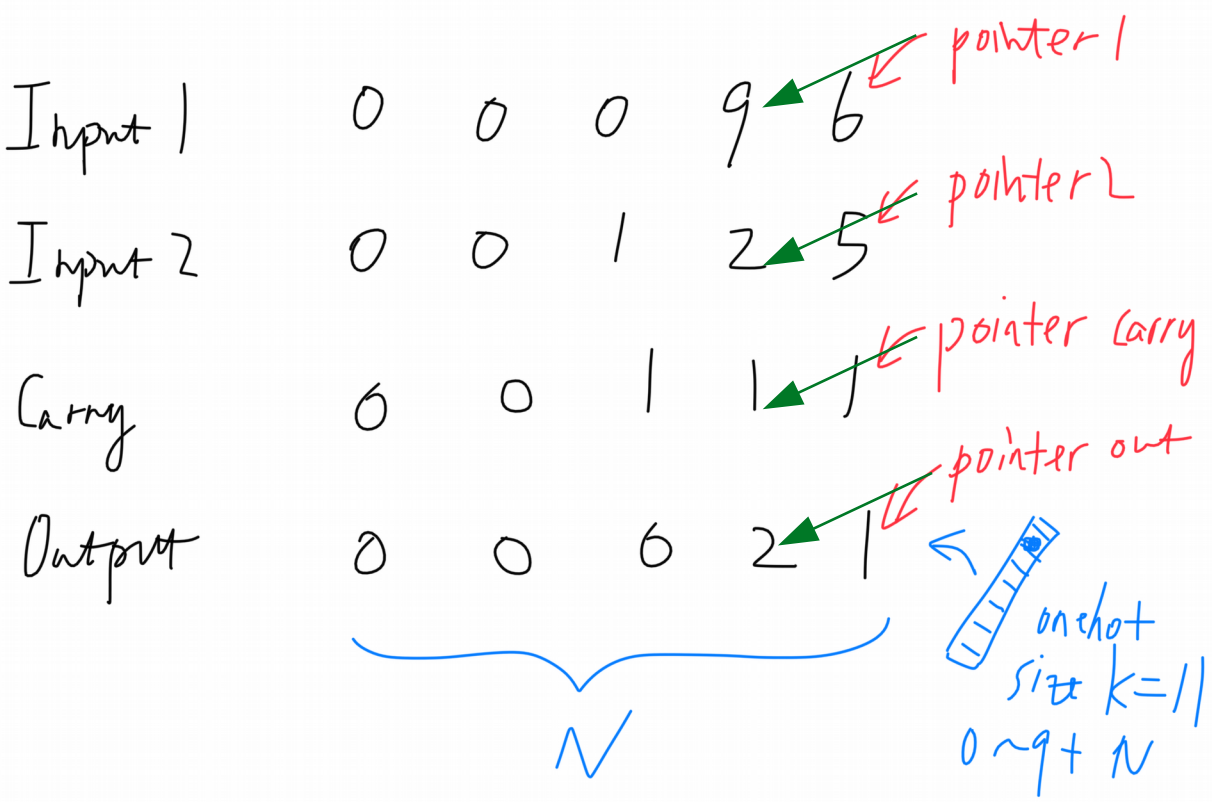
$$f_{enc}(Q, i_1, i_2, i_3, i_4, a_t) = MLP([Q(1, i_1), Q(2, i_2), Q(3, i_3), Q(4, i_4), a_t(1), a_t(2), a_t(3)])$$




```
ADD    Write "1" at the output slip
  ADD1
    WRITE OUT 1
    CARRY
      PTR CARRY LEFT
      WRITE CARRY 1
      PTR CARRY RIGHT
  LSHIFT
    PTR INP1 LEFT
    PTR INP2 LEFT
    PTR CARRY LEFT
    PTR OUT LEFT
  ADD1
    ...
```

Example

- Grade-school addition




```
ADD
  ADD1
    WRITE OUT 1
    CARRY
      PTR CARRY LEFT
      WRITE CARRY 1
      PTR CARRY RIGHT
LSHIFT
  PTR INP1 LEFT
  PTR INP2 LEFT
  PTR CARRY LEFT
  PTR OUT LEFT
ADD1      Move printers left
  . . .
```

Recursion?

Non-Recursive

```
1   ADD
2     ADD1
3       WRITE OUT 1
4       CARRY
5         PTR CARRY LEFT
6         WRITE CARRY 1
7         PTR CARRY RIGHT
8     LSHIFT
9       PTR INP1 LEFT
10      PTR INP2 LEFT
11      PTR CARRY LEFT
12      PTR OUT LEFT
13    ADD1
14      ...
```

Recursive

```
1   ADD
2     ADD1
3       WRITE OUT 1
4       CARRY
5         PTR CARRY LEFT
6         WRITE CARRY 1
7         PTR CARRY RIGHT
8     LSHIFT
9       PTR INP1 LEFT
10      PTR INP2 LEFT
11      PTR CARRY LEFT
12      PTR OUT LEFT
13    ADD
14      ...
```

The Magic Here

- No magic
- Fully supervised
  - What “ADD” sees is
    - ADD1, LSHIFT, ADD, RETURN
  - What “ADD1” sees is
    - WRITE (parameters: OUT, 1)
    - CARRY, RETURN
  - Etc.

Recursive

```
ADD
  ADD1
    WRITE OUT 1
    CARRY
      PTR CARRY LEFT
      WRITE CARRY 1
      PTR CARRY RIGHT
LSHIFT
    PTR INP1 LEFT
    PTR INP2 LEFT
    PTR CARRY LEFT
    PTR OUT LEFT
ADD
    . . .
```

Design Philosophy

- Define a “local” context, which the current step of execution fully depends on.
- Manually program in terms of the above “language.”
- Generate execution sequences (a.k.a. program traces) with some (not many) examples.

Outline

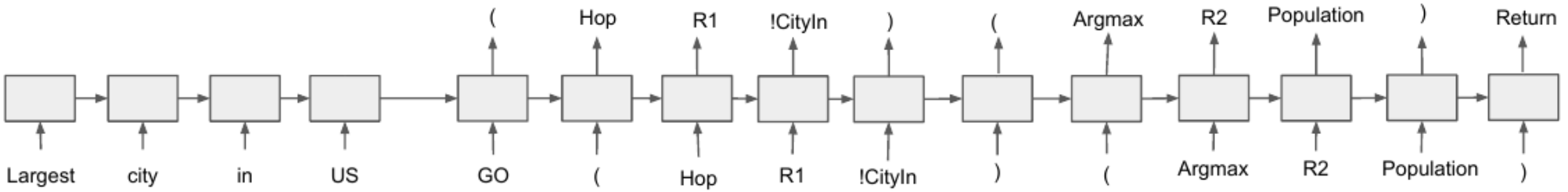
- Introduction
- Symbolic execution
  - (Fully supervised) Neural programmer interpreter
  - (Weakly supervised) Neural symbolic machine
  - “Spurious programs” and inductive programming
    - Learning semantic parsers from denotations
    - DeepCoder
    - More thoughts on “spurious programs”
- Distributed execution
  - Learning to execute
  - Neural enquirer
  - Differentiable neural computer
- Hybrid execution
  - Neural programmer
  - Coupling approach

Neural Symbolic Machines: Learning Semantic Parsers on Freebase with Weak Supervision

Chen Liang,* **Jonathan Berant,†** **Quoc Le, Kenneth D. Forbus, Ni Lao**

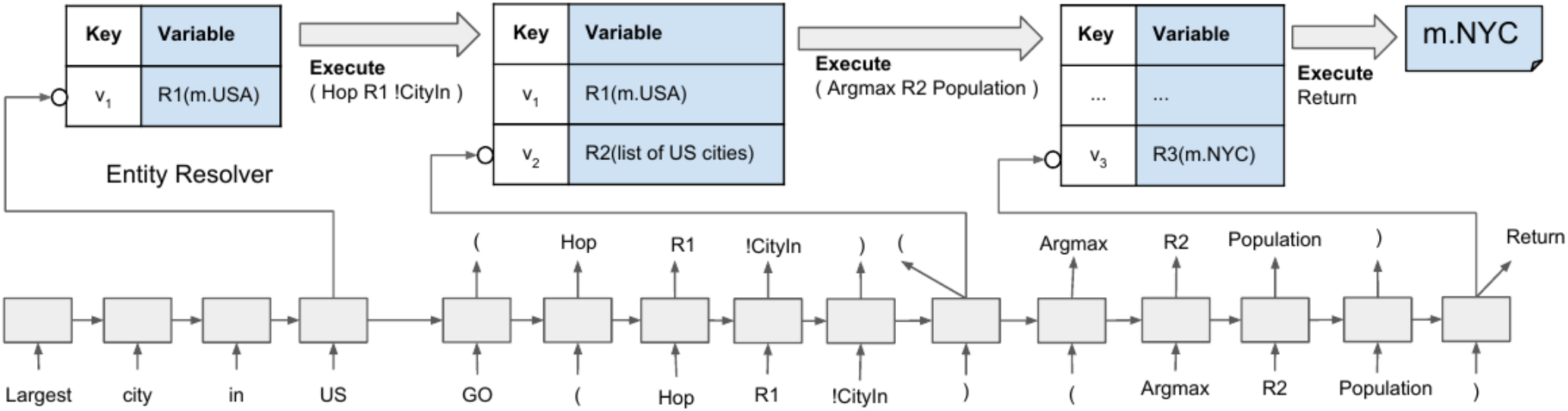
Northwestern University, Evanston, IL
{chenliang2013,forbus}@u.northwestern.edu
Tel-Aviv University, Tel Aviv-Yafo, Israel
joberant@cs.tau.ac.il
Google Inc., Mountain View, CA
{qvl,nlao}@google.com

Architecture



- Generally seq2seq

Architecture



- But with tricks
  - Entities marked with GRU output as embeddings
  - Searched with attention

Training

- Generally REINFORCE

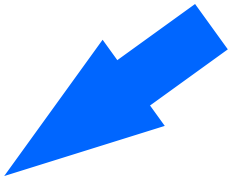
$$\nabla_\theta J^{RL}(\theta) = \sum_q \sum_{a_{0:T}} P(a_{0:T}|q,\theta)[R(q,a_{0:T}) - B(q)]\nabla_\theta \log P(a_{0:T}|q,\theta)$$

Training

- But with more tricks
  - Beam search instead of sampling
  - Keep the “best” obtained programs so far and perform supervised training in addition to REINFORCE

Outline

- Introduction
- Symbolic execution
  - (Fully supervised) Neural programmer interpreter
  - (Weakly supervised) Neural symbolic machine
  - “Spurious programs” and inductive programming
    - Learning semantic parsers from denotations
    - DeepCoder
    - More thoughts on “spurious programs”
- Distributed execution
  - Learning to execute
  - Neural enquirer
  - Differentiable neural computer
- Hybrid execution
  - Neural programmer
  - Coupling approach



Inferring Logical Forms From Denotations

Panupong Pasupat
Computer Science Department
Stanford University
ppasupat@cs.stanford.edu

Percy Liang
Computer Science Department
Stanford University
pliang@cs.stanford.edu

(ACL-16)

- Spurious programs
- Fictitious worlds

The Problem of "Spurious Programs"

Year	Venue	Position	Event	Time
2001	Hungary	2nd	400m	47.12
2003	Finland	1st	400m	46.69
2005	Germany	11th	400m	46.62
2007	Thailand	1st	relay	182.05
2008	China	7th	relay	180.32

x: "*Where did the last 1st place finish occur?*"

y: Thailand

Consistent

Correct

z_1: $\mathbf{R}[\texttt{Venue}].\texttt{argmax}(\texttt{Position.1st}, \texttt{Index})$

Among rows with Position = 1st, pick the one with maximum index, then return the Venue of that row.

z_2: $\mathbf{R}[\texttt{Venue}].\texttt{Index.max}(\mathbf{R}[\texttt{Index}].\texttt{Position.1st})$

Find the maximum index of rows with Position = 1st, then return the Venue of the row with that index.

z_3: $\mathbf{R}[\texttt{Venue}].\texttt{argmax}(\texttt{Position.Number.}\mathit{1}, \mathbf{R}[\lambda x.\mathbf{R}[\texttt{Date}].\mathbf{R}[\texttt{Year}].x])$

Among rows with Position number 1, pick one with latest date in the Year column and return the Venue.

Spurious

z_4: $\mathbf{R}[\texttt{Venue}].\texttt{argmax}(\texttt{Position.Number.}\mathit{1}, \mathbf{R}[\lambda x.\mathbf{R}[\texttt{Number}].\mathbf{R}[\texttt{Time}].x])$

Among rows with Position number 1, pick the one with maximum Time number. Return the Venue.

z_5: $\mathbf{R}[\texttt{Venue}].\texttt{Year.Number.}(\mathbf{R}[\texttt{Number}].\mathbf{R}[\texttt{Year}].\texttt{argmax}(\texttt{Type.Row}, \texttt{Index}) - \mathit{1})$

Subtract 1 from the Year in the last row, then return the Venue of the row with that Year.

Inconsistent

$\tilde{z}$: $\mathbf{R}[\texttt{Venue}].\texttt{argmin}(\texttt{Position.1st}, \texttt{Index})$

Among rows with Position = 1st, pick the one with minimum index, then return the Venue. (= Finland)

Solution: Generating fictitious data

Year	Venue	Position	Event	Time
2001	Hungary	2nd	400m	47.12
2003	Finland	1st	400m	46.69
2005	Germany	11th	400m	46.62
2007	Thailand	1st	relay	182.05
2008	China	7th	relay	180.32

x: "*Where did the last 1st place finish occur?*"

y: Thailand

Year	Venue	Position	Event	Time
2001	Finland	7th	relay	46.62
2003	Germany	1st	400m	180.32
2005	China	1st	relay	47.12
2007	Hungary	7th	relay	182.05

Figure 4: From the example in Figure 1, we generate a table for the fictitious world w_1.

	w	w_1	w_2	$\cdots$	
z_1	Thailand	China	Finland	$\cdots$	q_1
z_2	Thailand	China	Finland	$\cdots$	
z_3	Thailand	China	Finland	$\cdots$	
z_4	Thailand	Germany	China	$\cdots$	q_2
z_5	Thailand	China	China	$\cdots$	q_3
z_6	Thailand	China	China	$\cdots$	
$\vdots$	$\vdots$	$\vdots$	$\vdots$		

- Equivalent classes
  - q1={z1, z2, z3}
  - q2={z4}
  - q3={z5,z6}
- Choose worlds with most information gain

Outline

- Introduction
- Symbolic execution
  - (Fully supervised) Neural programmer interpreter
  - (Weakly supervised) Neural symbolic machine
  - “Spurious programs” and inductive programming
    - Learning semantic parsers from denotations
    - DeepCoder
    - More thoughts on “spurious programs”
- Distributed execution
  - Learning to execute
  - Neural enquirer
  - Differentiable neural computer
- Hybrid execution
  - Neural programmer
  - Coupling approach

DEEPCODER: LEARNING TO WRITE PROGRAMS

Matej Balog*
Department of Engineering
University of Cambridge

Alexander L. Gaunt, Marc Brockschmidt,
Sebastian Nowozin, Daniel Tarlow
Microsoft Research

```
a ← [int]
b ← FILTER (<0) a
c ← MAP (*4) b
d ← SORT c
e ← REVERSE d
```

An input-output example:

Input:

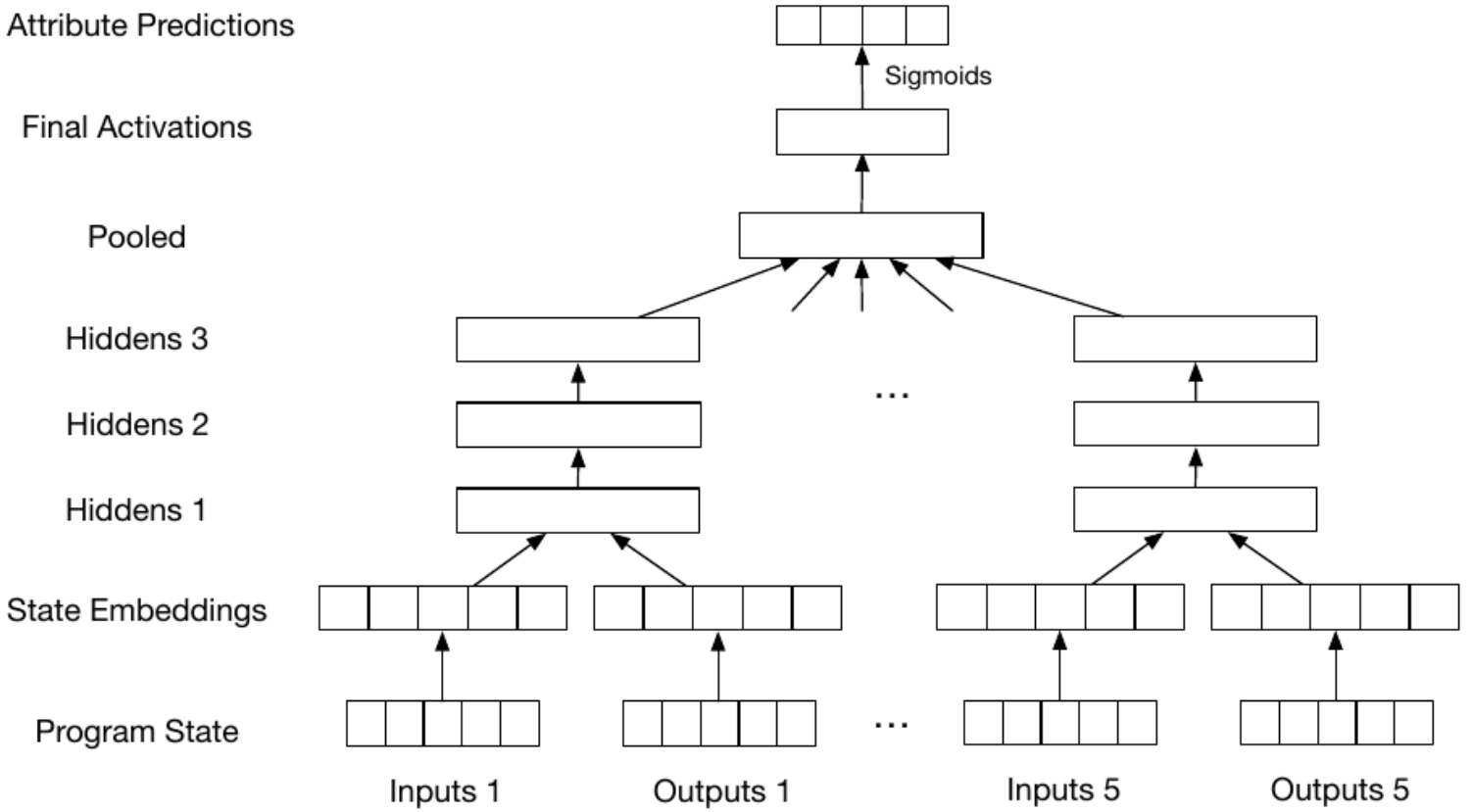
`[-17, -3, 4, 11, 0, -5, -9, 13, 6, 6, -8, 11]`

Output:

`[-12, -20, -32, -36, -68]`

Figure 1: An example program in our DSL that takes a single integer array as its input.

Figure 2: Neural network predicts the probability of each function appearing in the source code.



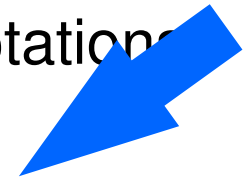
Search

- DFS
  - DFS can opt to consider the functions as ordered by their predicted probabilities from the neural network
- “Sort and add” enumeration
- Sketch

- Not really “structured” prediction

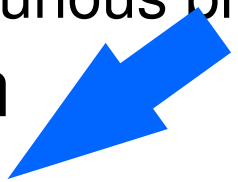
Outline

- Introduction
- Symbolic execution
  - (Fully supervised) Neural programmer interpreter
  - (Weakly supervised) Neural symbolic machine
  - “Spurious programs” and inductive programming
    - Learning semantic parsers from denotations
    - DeepCoder
    - More thoughts on “spurious programs”
- Distributed execution
  - Learning to execute
  - Neural enquirer
  - Differentiable neural computer
- Hybrid execution
  - Neural programmer
  - Coupling approach



Outline

- Introduction
- Symbolic execution
  - (Fully supervised) Neural programmer interpreter
  - (Weakly supervised) Neural symbolic machine
  - “Spurious programs” and inductive programming
    - Learning semantic parsers from denotations
    - DeepCoder
    - More thoughts on “spurious programs”
- Distributed execution
  - Learning to execute
  - Neural enquirer
  - Differentiable neural computer
- Hybrid execution
  - Neural programmer
  - Coupling approach



LEARNING TO EXECUTE

Wojciech Zaremba*
New York University
woj.zaremba@gmail.com

Ilya Sutskever
Google
ilyasu@google.com

Essentially a seq2seq model

Input:

```
j=8584
for x in range(8):
  j+=920
b=(1500+j)
print((b+7567))
```

Target: 25011.

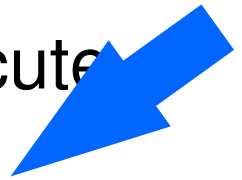
Input:

```
i=8827
c=(i-5347)
print((c+8704) if 2641<8500 else 5308)
```

Target: 12184.

Outline

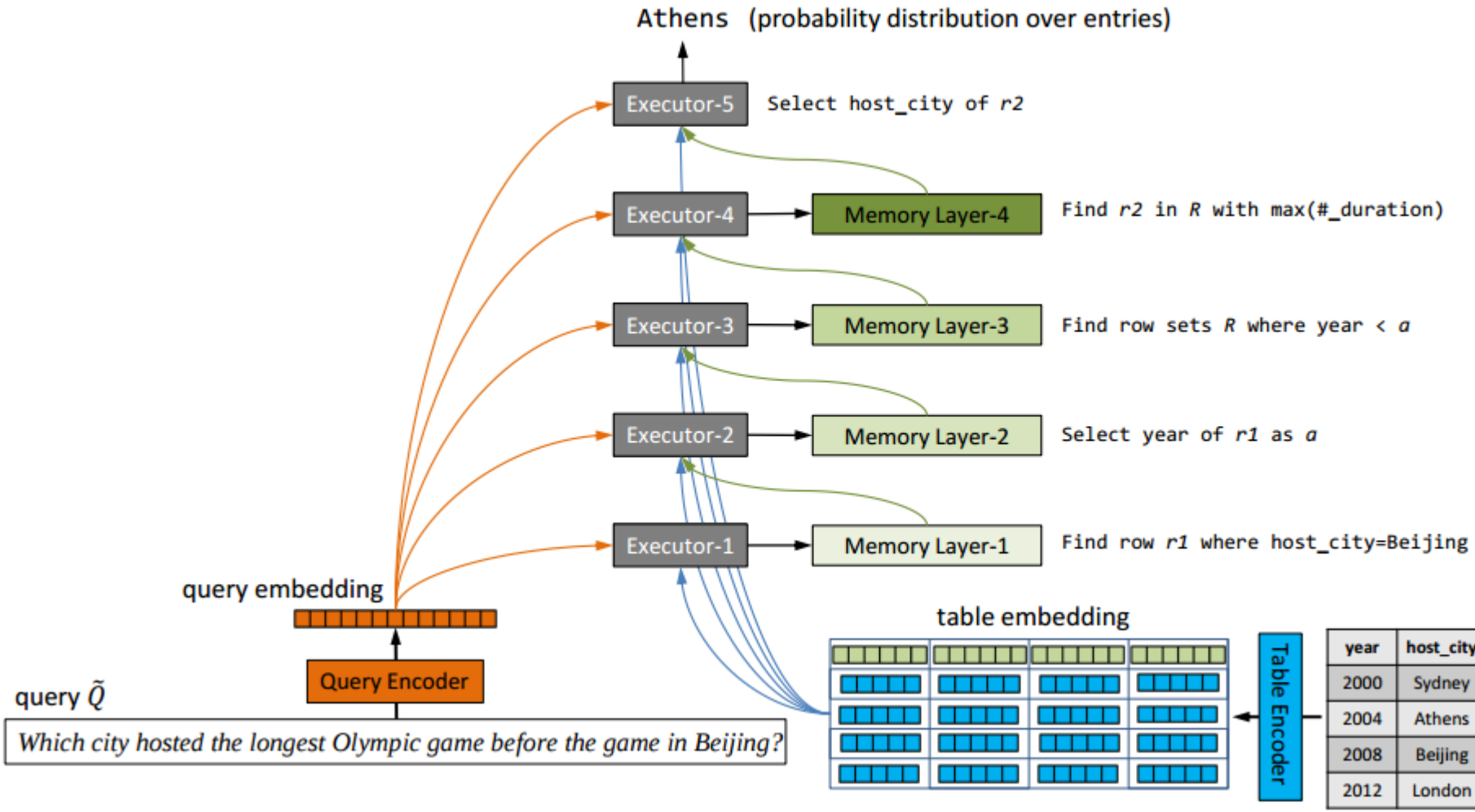
- Introduction
- Symbolic execution
  - (Fully supervised) Neural programmer interpreter
  - (Weakly supervised) Neural symbolic machine
  - “Spurious programs” and inductive programming
    - Learning semantic parsers from denotations
    - DeepCoder
    - More thoughts on “spurious programs”
- Distributed execution
  - Learning to execute
  - Neural enquirer
  - Differentiable neural computer
- Hybrid execution
  - Neural programmer
  - Coupling approach



Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence (IJCAI-16)

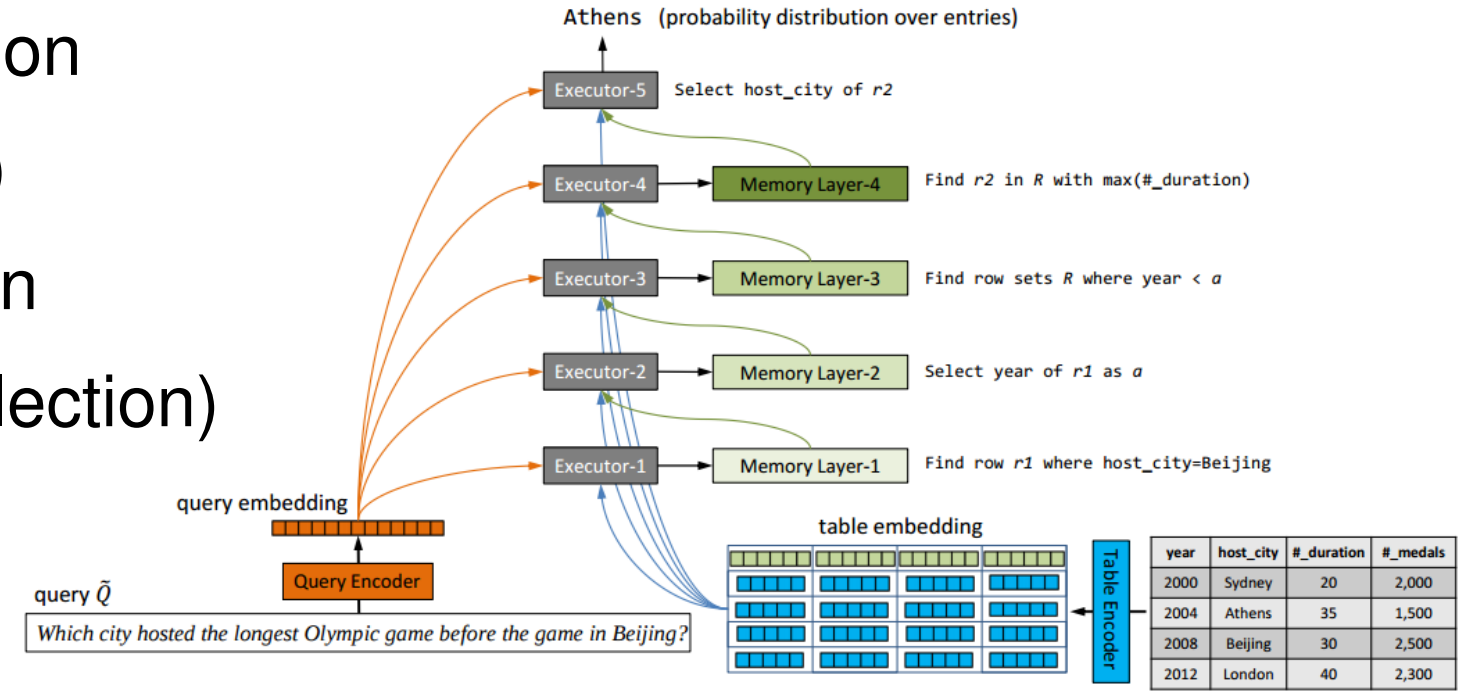
Neural Enquirer: Learning to Query Tables in Natural Language

Pengcheng Yin[1] **Zhengdong Lu**[2] **Hang Li**[2] **Ben Kao**[1]
[1] Department of Computer Science, The University of Hong Kong
[2] Noah's Ark Lab, Huawei Technologies
[1]{pcyin,kao}@cs.hku.hk [2]{Lu.Zhengdong,HangLi.HL}@huawei.com



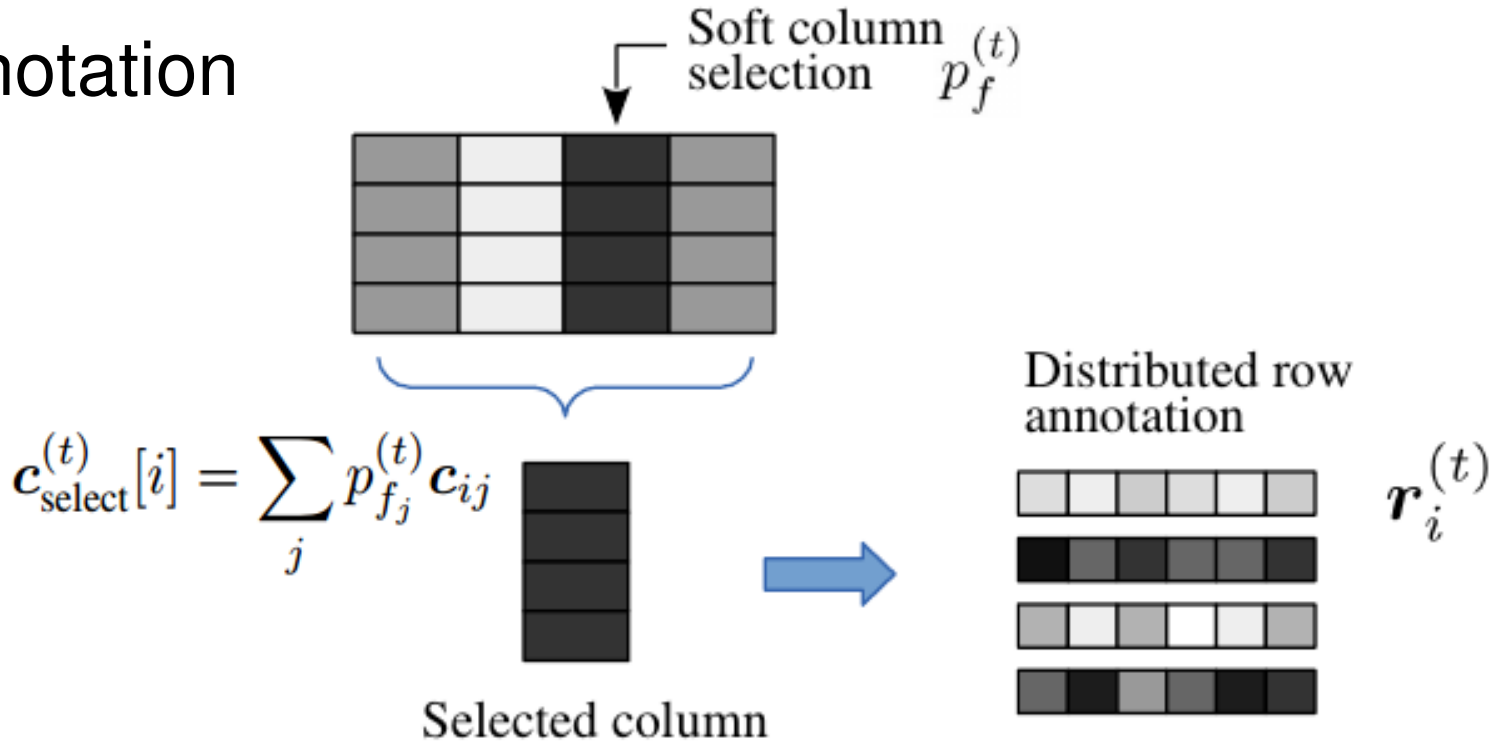
Three Modules

- Query encoder: BiRNN
- Table encoder
  - Concatenation of cell and field embeddings
  - Further processed by a multi-layer perceptron
- Executor
  - Column attention (soft selection)
  - Row annotation (distributed selection)



Executor

- The result of one-step execution softmax attention over columns and a distributed
  - Column attention
  - Row annotation



Details

• Let $\boldsymbol{r}_i^{(t-1)}$ be the previous step's row annotation results, where the subscript i indexes a particular row.

- Last step's execution information

$$\boldsymbol{g}^{(t-1)} = \mathrm{MaxPool}_i \left\{ \boldsymbol{r}_i^{(t-1)} \right\}$$

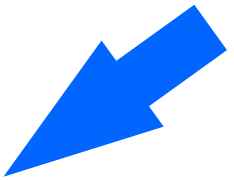
- Current step
  - Column attention $p_{f_j}^{(t)} = \mathrm{softmax}\left(\mathrm{MLP}\left([\boldsymbol{q}; \boldsymbol{f}_j; \boldsymbol{g}^{(t-1)}]\right)\right)$
  - Row annotation

$$\boldsymbol{c}_{\text{select}}^{(t)}[i] = \sum_j p_{f_j}^{(t)} \boldsymbol{c}_{ij}$$

$$\boldsymbol{r}_i^{(t)} = \mathrm{MLP}\left(\left[\boldsymbol{q}, \boldsymbol{g}^{(t-1)}, \boldsymbol{r}^{(t-1)}, \boldsymbol{c}_{\text{select}}^{(t)}[i]\right]\right)$$

Outline

- Introduction
- Symbolic execution
  - (Fully supervised) Neural programmer interpreter
  - (Weakly supervised) Neural symbolic machine
  - “Spurious programs” and inductive programming
    - Learning semantic parsers from denotations
    - DeepCoder
    - More thoughts on “spurious programs”
- Distributed execution
  - Learning to execute
  - Neural enquirer
  - Differentiable neural computer
- Hybrid execution
  - Neural programmer
  - Coupling approach

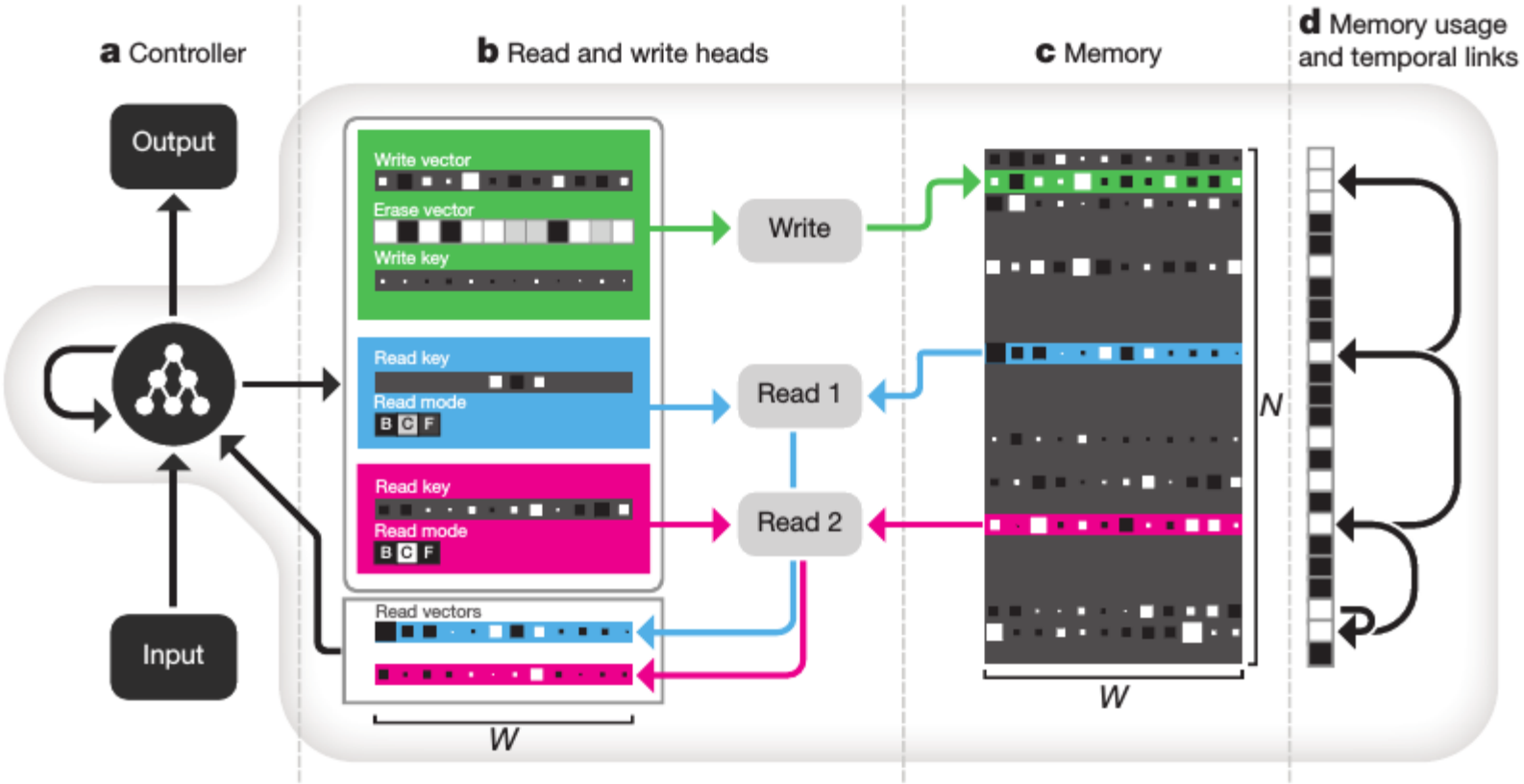


ARTICLE

doi:10.1038/nature20101

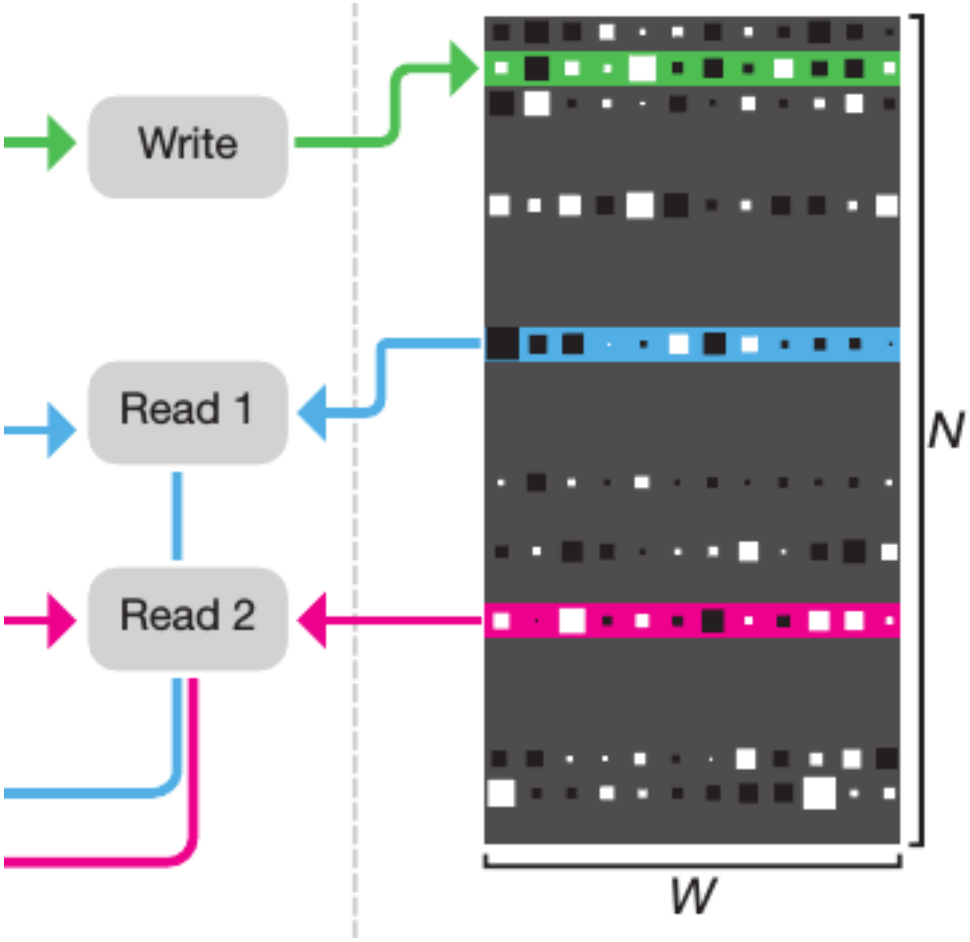
Hybrid computing using a neural network with dynamic external memory

Alex Graves[1]*, Greg Wayne[1]*, Malcolm Reynolds[1], Tim Harley[1], Ivo Danihelka[1], Agnieszka Grabska-Barwińska[1], Sergio Gómez Colmenarejo[1], Edward Grefenstette[1], Tiago Ramalho[1], John Agapiou[1], Adrià Puigdomènech Badia[1], Karl Moritz Hermann[1], Yori Zwols[1], Georg Ostrovski[1], Adam Cain[1], Helen King[1], Christopher Summerfield[1], Phil Blunsom[1], Koray Kavukcuoglu[1] & Demis Hassabis[1]



Memory and its Access

- Memory: $M \in \mathbb{R}^{N \times W}$
  - *N*: # of memory slots
  - *W*: Width/dim of a slot
- Reader:
  - *R* readers
  - Content addressing
    + Temporal linking addressing
- Writer:
  - Exact 1 writer
  - Content addressing + dynamic memory allocation



Architecture

- Input
  - Current signal
  - Previous step's memory read
- Output
  - Value

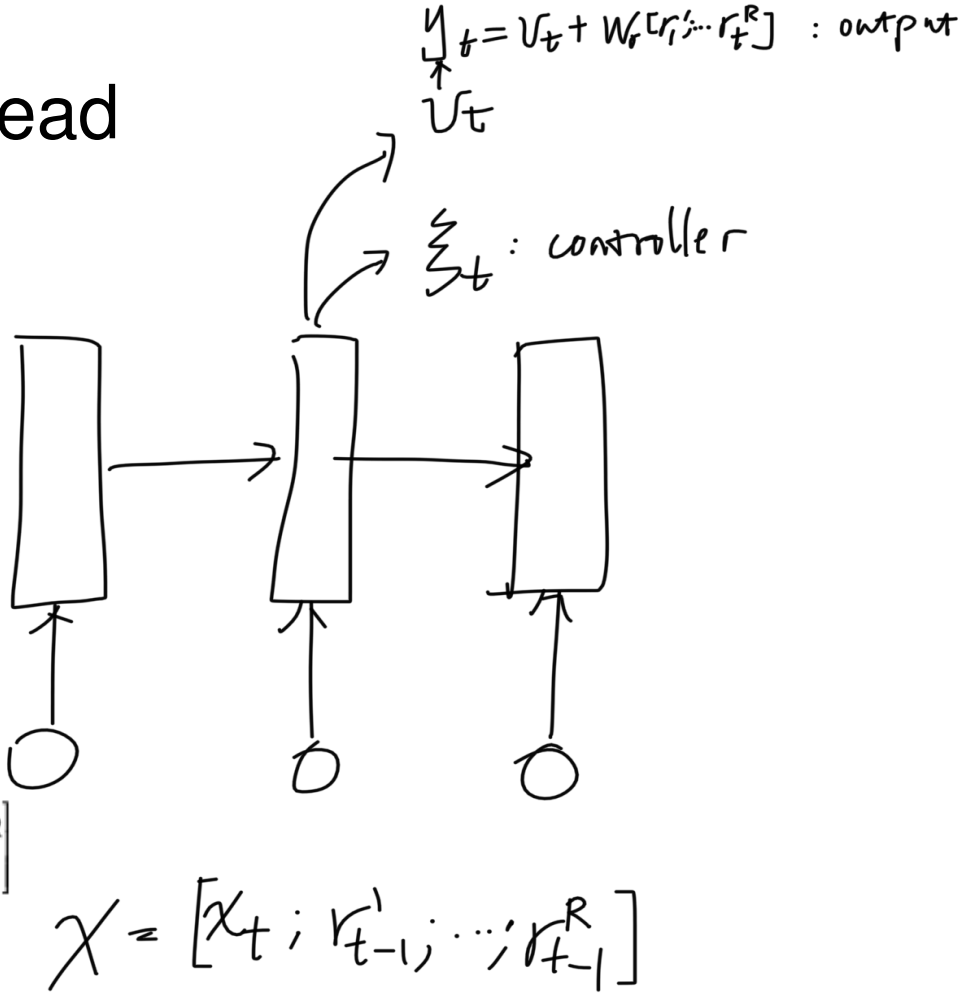
$$\boldsymbol{v}_t = W_v[\boldsymbol{h}_t^1; \ldots; \boldsymbol{h}_t^L]$$

$$\boldsymbol{y}_t = \boldsymbol{v}_t + W_r[\boldsymbol{r}_t^1; \ldots; \boldsymbol{r}_t^R]$$

(so that output is dependent on read memory)

  - Control $\boldsymbol{\xi}_t = W_\xi[\boldsymbol{h}_t^1; \ldots; \boldsymbol{h}_t^L]$

$$\boldsymbol{\xi}_t = \left[\boldsymbol{k}_t^{r,1}; \ldots; \boldsymbol{k}_t^{r,R}; \hat{\beta}_t^{r,1}; \ldots; \hat{\beta}_t^{r,R}; \boldsymbol{k}_t^w; \hat{\beta}_t^w; \hat{\boldsymbol{e}}_t; \boldsymbol{v}_t; \hat{f}_t^1; \ldots; \hat{f}_t^R; \hat{g}_t^a; \hat{g}_t^w; \hat{\pi}_t^1; \ldots; \hat{\pi}_t^R\right]$$



Controller: Preliminary

- To obtain a gate: sigmoid
- To obtain a distribution: softmax
- To constrain in [1, infinity)

$$\text{oneplus}(x) = 1 + \log(1 + e^x)$$

- A distribution

$$\mathcal{S}_N = \left\{ \boldsymbol{\alpha} \in \mathbb{R}^N : \boldsymbol{\alpha}_i \in [0,1], \ \sum_{i=1}^{N} \boldsymbol{\alpha}_i = 1 \right\}$$

- No more than a distribution (detracted by a default null prob.)

$$\Delta_N = \left\{ \boldsymbol{\alpha} \in \mathbb{R}^N : \boldsymbol{\alpha}_i \in [0,1], \sum_{i=1}^{N} \boldsymbol{\alpha}_i \leq 1 \right\}$$

A Glance at the Control Signals

- R read keys $\{\boldsymbol{k}_t^{\mathrm{r},i} \in \mathbb{R}^W; 1 \leq i \leq R\}$;
- R read strengths $\{\beta_t^{\mathrm{r},i} = \mathrm{oneplus}(\hat{\beta}_t^{\mathrm{r},i}) \in [1,\infty)\,; 1 \leq i \leq R\}$;
- the write key $\boldsymbol{k}_t^{\mathrm{w}} \in \mathbb{R}^W$;
- the write strength $\beta_t^{\mathrm{w}} = \mathrm{oneplus}(\hat{\beta}_t^{\mathrm{w}}) \in [1,\infty)$;
- the erase vector $\boldsymbol{e}_t = \sigma(\hat{\boldsymbol{e}}_t) \in [0,1]^W$;
- the write vector $\boldsymbol{v}_t \in \mathbb{R}^W$;
- R free gates $\{f_t^i = \sigma(\hat{f}_t^i) \in [0,1]; 1 \leq i \leq R\}$;
- the allocation gate $g_t^{\mathrm{a}} = \sigma(\hat{g}_t^{\mathrm{a}}) \in [0,1]$;
- the write gate $g_t^{\mathrm{w}} = \sigma(\hat{g}_t^{\mathrm{w}}) \in [0,1]$; and
- R read modes $\{\boldsymbol{\pi}_t^i = \mathrm{softmax}(\hat{\boldsymbol{\pi}}_t^i) \in \mathcal{S}_3\,; 1 \leq i \leq R\}$.

Memory Reading and Writing

- Reader
  - Reading weights $\mathbf{w}_t^{\mathrm{r},1}, \ldots, \mathbf{w}_t^{\mathrm{r},R} \in \Delta_N$
  - Read vectors $\boldsymbol{r}_t^i = M_t^\top \boldsymbol{w}_t^{\mathrm{r},i}$

    where *M*: <*N* x *W*>, each ***w***: <*N x 1*>

    (no more than attention)
- Writer
  - Writing weight regarding a memory slot $\mathbf{w}_t^{\mathrm{w}} \in \Delta_N$
  - Erasing vector regarding a memory dimension

    (given by the controller) $\boldsymbol{e}_t = \sigma(\hat{\boldsymbol{e}}_t) \in [0,1]^W$
  - Writing operation $M_t = M_{t-1} \circ (E - \boldsymbol{w}_t^{\mathrm{w}} \boldsymbol{e}_t^\top) + \boldsymbol{w}_t^{\mathrm{w}} \boldsymbol{v}_t^\top$

    E: **1**'s | w→1 & e → 1 ==> **M** is determined by **v**

Memory Addressing

- Content-based addressing
  - e.g., select **5**
- Dynamic memory allocation
  - Free(.), Malloc(.) and write to it
- Temporal memory linkage
  - Move to the next element in a linked list

- C.f. CopyNet, Latent Predictor Network

Content-Based Addressing

$$\mathcal{C}(M, \boldsymbol{k}, \beta)[i] = \frac{\exp\{\mathcal{D}(\boldsymbol{k}, M[i, \cdot])\beta\}}{\sum_j \exp\{\mathcal{D}(\boldsymbol{k}, M[j, \cdot])\beta\}}$$

Size: <*N*>

$$\mathcal{D}(\boldsymbol{u}, \boldsymbol{v}) = \frac{\boldsymbol{u} \cdot \boldsymbol{v}}{|\boldsymbol{u}||\boldsymbol{v}|}$$

k and beta given by controller

- R read keys $\{\boldsymbol{k}_t^{\mathrm{r},i} \in \mathbb{R}^W; 1 \le i \le R\}$;
- R read strengths $\{\beta_t^{\mathrm{r},i} = \mathrm{oneplus}(\hat{\beta}_t^{\mathrm{r},i}) \in [1,\infty); 1 \le i \le R\}$;
- the write key $\boldsymbol{k}_t^{\mathrm{w}} \in \mathbb{R}^W$;
- the write strength $\beta_t^{\mathrm{w}} = \mathrm{oneplus}(\hat{\beta}_t^{\mathrm{w}}) \in [1,\infty)$;

Dynamic Memory Allocation

- Memory usage vector $\boldsymbol{u}_t \in [0, 1]^N$

  1=used, 0=unused $\boldsymbol{u}_0 = \mathbf{0}$

- Memory retention vector $\boldsymbol{\psi}_t \in [0, 1]^N$

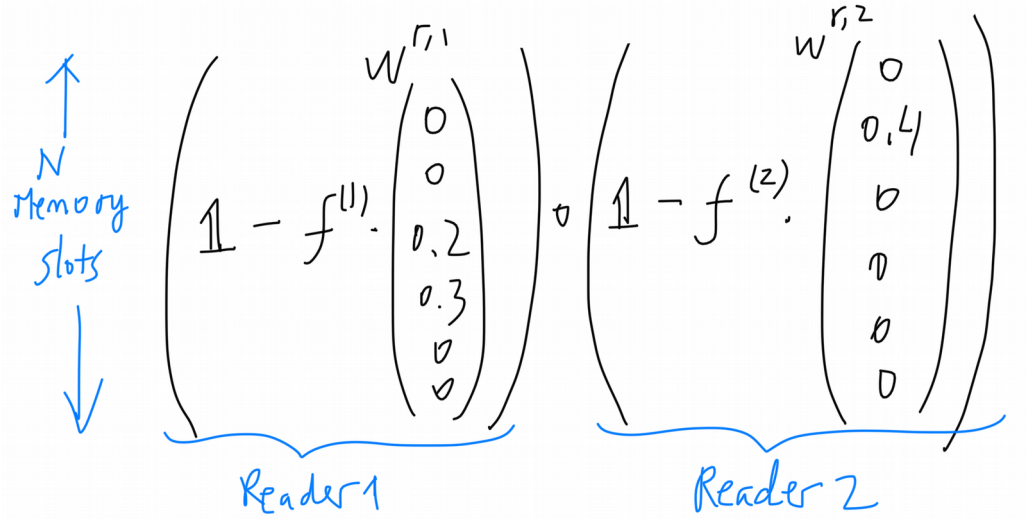
  how much each location will not be freed by the free gates

$$\boldsymbol{\psi}_t = \prod_{i=1}^{R} \left(\mathbf{1} - f_t^i \mathbf{w}_{t-1}^{\mathrm{r},i} \right)$$

where free gates are given by the controller

$$\{ f_t^i = \sigma(\hat{f}_t^i) \in [0,1]; 1 \le i \le R \}$$

Free is in the sense of "malloc and free," rather than "free-style."

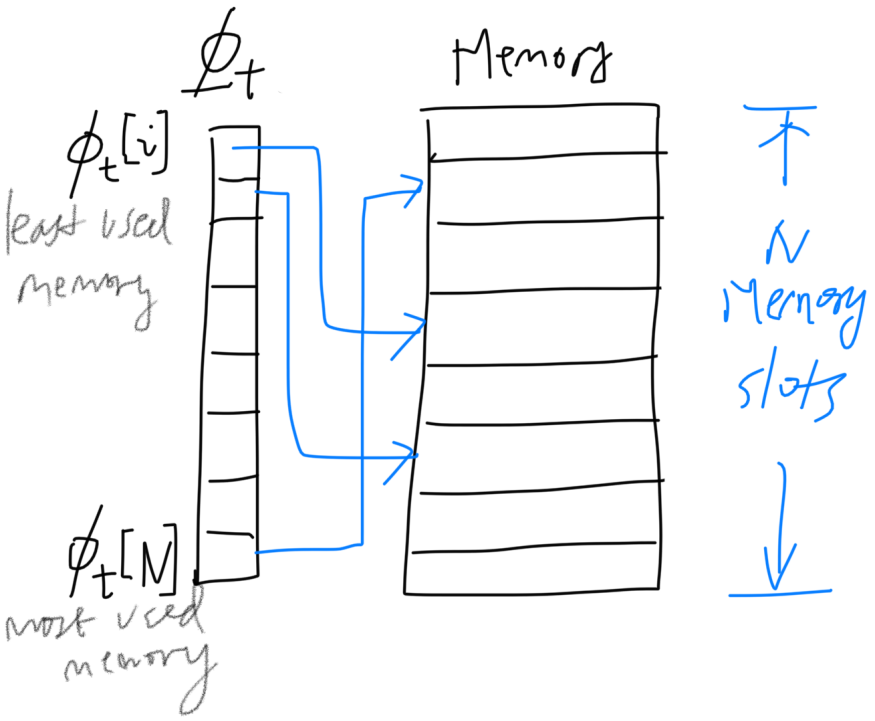


Read more=> retain less

f larger => retrain less

Dynamic Memory Allocation

- Memory usage vector recursion
  - 1=used, 0=unused $\boldsymbol{u}_0 = \mathbf{0}$
  - $\boldsymbol{u}_t = (\boldsymbol{u}_{t-1} + \boldsymbol{w}^{\mathrm{w}}_{t-1} - \boldsymbol{u}_{t-1} \circ \boldsymbol{w}^{\mathrm{w}}_{t-1}) \circ \boldsymbol{\psi}_t$
  - $\boldsymbol{u} = [\boldsymbol{u} * (\mathbf{1} - \mathbf{w}) + 1 * \mathbf{w}] * \mathbf{psi}$
  - $\boldsymbol{w}^{\mathrm{w}}_{t-1}$: last step's writing weight
  - write more => use more
- Free list $\boldsymbol{\phi}_t \in \mathbb{Z}^N$
  - Sorted list of memory usage
  - $\phi_t[1]$: least used memory index



Dynamic Memory Allocation

- Allocation weighting

$$\boldsymbol{a}_t[\phi_t[j]] = (1 - \boldsymbol{u}_t[\phi_t[j]]) \prod_{i=1}^{j-1} \boldsymbol{u}_t[\phi_t[i]]$$

  - **1 – *u***: used more => allocate less
  - $\phi_t \in \mathbb{Z}^N$: sorted list of memory usage
  - $\prod_{i=1}^{j-1} \boldsymbol{u}_t[\phi_t[i]]$ least used=> allocate more

Write Weighting

$$\boldsymbol{w}_t^{\mathrm{w}} = g_t^{\mathrm{w}}\left[g_t^{\mathrm{a}}\boldsymbol{a}_t + (1 - g_t^{\mathrm{a}})\boldsymbol{c}_t^{\mathrm{w}}\right]$$

- g^{w}: writing gate, deciding to write or not
- g^{a}: memory allocation gate for write weighting
- Both given by the controller

the allocation gate $g_t^{\mathrm{a}} = \sigma(\hat{g}_t^{\mathrm{a}}) \in [0,1]$;

the write gate $g_t^{\mathrm{w}} = \sigma(\hat{g}_t^{\mathrm{w}}) \in [0,1]$;

d Memory usage and temporal links

Read Weighting

$$\boldsymbol{w}_t^{\mathrm{r},i} = \boldsymbol{\pi}_t^i[1]\boldsymbol{b}_t^i + \boldsymbol{\pi}_t^i[2]\boldsymbol{c}_t^{\mathrm{r},i} + \boldsymbol{\pi}_t^i[3]\boldsymbol{f}_t^i$$

d Memory usage and temporal links

Temporal memory linkage

$$\boldsymbol{w}_t^{\mathrm{r},i} = \boldsymbol{\pi}_t^i[1]\boldsymbol{b}_t^i + \boldsymbol{\pi}_t^i[2]\boldsymbol{c}_t^{\mathrm{r},i} + \boldsymbol{\pi}_t^i[3]\boldsymbol{f}_t^i$$

- Precedence weighting $\boldsymbol{p}_t \in \Delta_N$

$$\boldsymbol{p}_0 = \mathbf{0}$$

$$\boldsymbol{p}_t = \left(1 - \sum_i \boldsymbol{w}_t^{\mathrm{w}}[i]\right)\boldsymbol{p}_{t-1} + \boldsymbol{w}_t^{\mathrm{w}}$$

  - Degree to which a slot is lastly written
  - Current writing weight plus $\boldsymbol{p}_{t-1}$
  - $\boldsymbol{p}_{t-1}$ discounted by summed writing weight

d Memory usage and temporal links

Temporal memory linkage

$$\boldsymbol{w}_t^{\mathrm{r},i} = \boldsymbol{\pi}_t^i[1]\boldsymbol{b}_t^i + \boldsymbol{\pi}_t^i[2]\boldsymbol{c}_t^{\mathrm{r},i} + \boldsymbol{\pi}_t^i[3]\boldsymbol{f}_t^i$$

- Linking matrix $L_t \in [0,1]^{N\times N}$

$$L_0[i,j]=0 \quad \forall\, i,j$$

$$L_t[i,i]=0 \quad \forall\, i$$

$$L_t[i,j] = (1 - \boldsymbol{w}_t^{\mathrm{w}}[i] - \boldsymbol{w}_t^{\mathrm{w}}[j])L_{t-1}[i,j] + \boldsymbol{w}_t^{\mathrm{w}}[i]\boldsymbol{p}_{t-1}[j]$$

  - Degree to which location i is written after j
  - slot i written at the current step
    slot j written at the last step
    ==> L[i,j] → 1
  - Slot i not written ==> L[i,j] tends to remain
  - Slot j written ==> L[i,j] → 0, i.e., refresh

d Memory usage and temporal links

Temporal memory linkage

$$\boldsymbol{w}_t^{\mathrm{r},i} = \boldsymbol{\pi}_t^i[1]\boldsymbol{b}_t^i + \boldsymbol{\pi}_t^i[2]\boldsymbol{c}_t^{\mathrm{r},i} + \boldsymbol{\pi}_t^i[3]\boldsymbol{f}_t^i$$

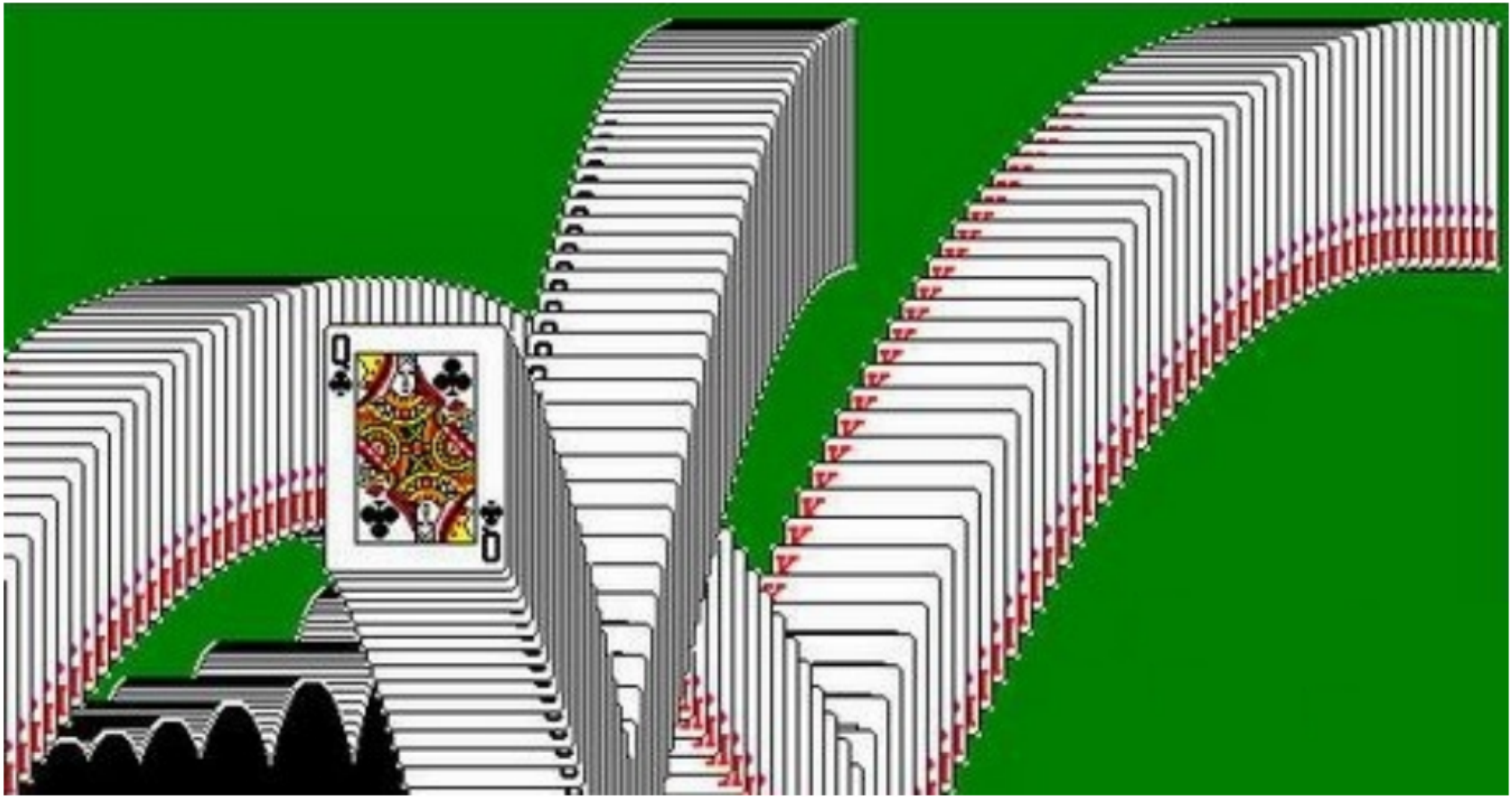
- Backward/forward weighting

$$\boldsymbol{b}_t^i \in \Delta_N \quad \boldsymbol{f}_t^i \in \Delta_N$$

$$\boldsymbol{f}_t^i = L_t \boldsymbol{w}_{t-1}^{\mathrm{r},i}$$

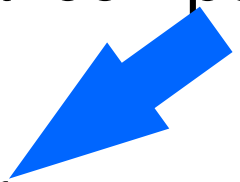
$$\boldsymbol{b}_t^i = L_t^\top \boldsymbol{w}_{t-1}^{\mathrm{r},i}$$

Linking weight * last step's reading weight



Outline

- Introduction
- Symbolic execution
  - (Fully supervised) Neural programmer interpreter
  - (Weakly supervised) Neural symbolic machine
  - “Spurious programs” and inductive programming
    - Learning semantic parsers from denotations
    - DeepCoder
    - More thoughts on “spurious programs”
- Distributed execution
  - Learning to execute
  - Neural enquirer
  - Differentiable neural computer
- Hybrid execution
  - Neural programmer
  - Coupling approach

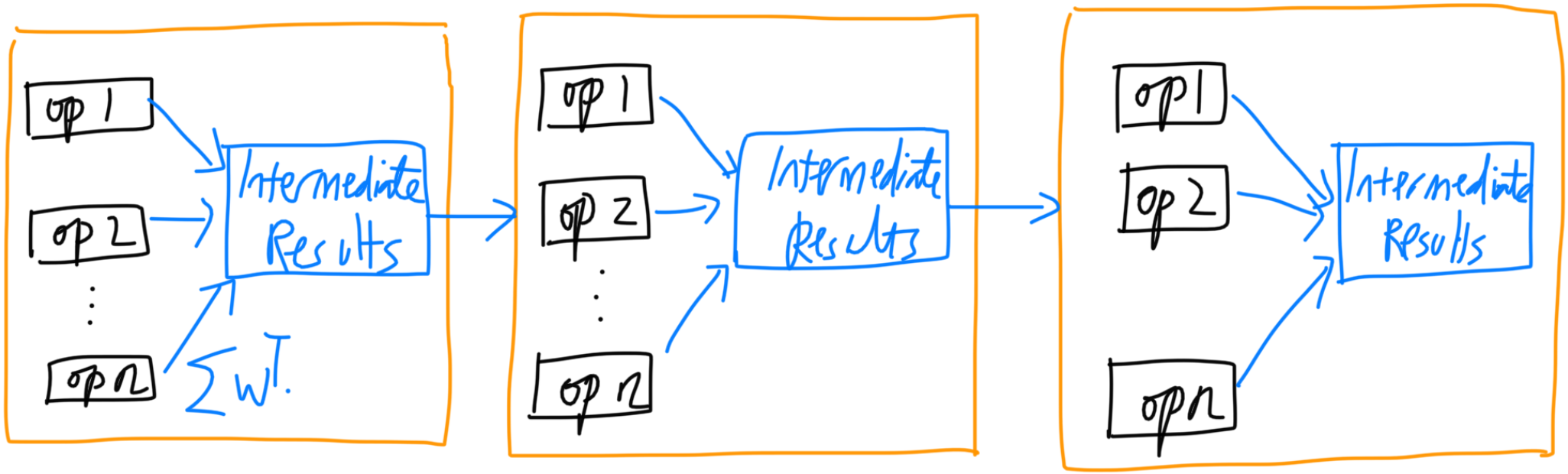


NEURAL PROGRAMMER: INDUCING LATENT PROGRAMS WITH GRADIENT DESCENT

Arvind Neelakantan*
University of Massachusetts Amherst
arvind@cs.umass.edu

Quoc V. Le
Google Brain
qvl@google.com

Ilya Sutskever
Google Brain
ilyasu@google.com

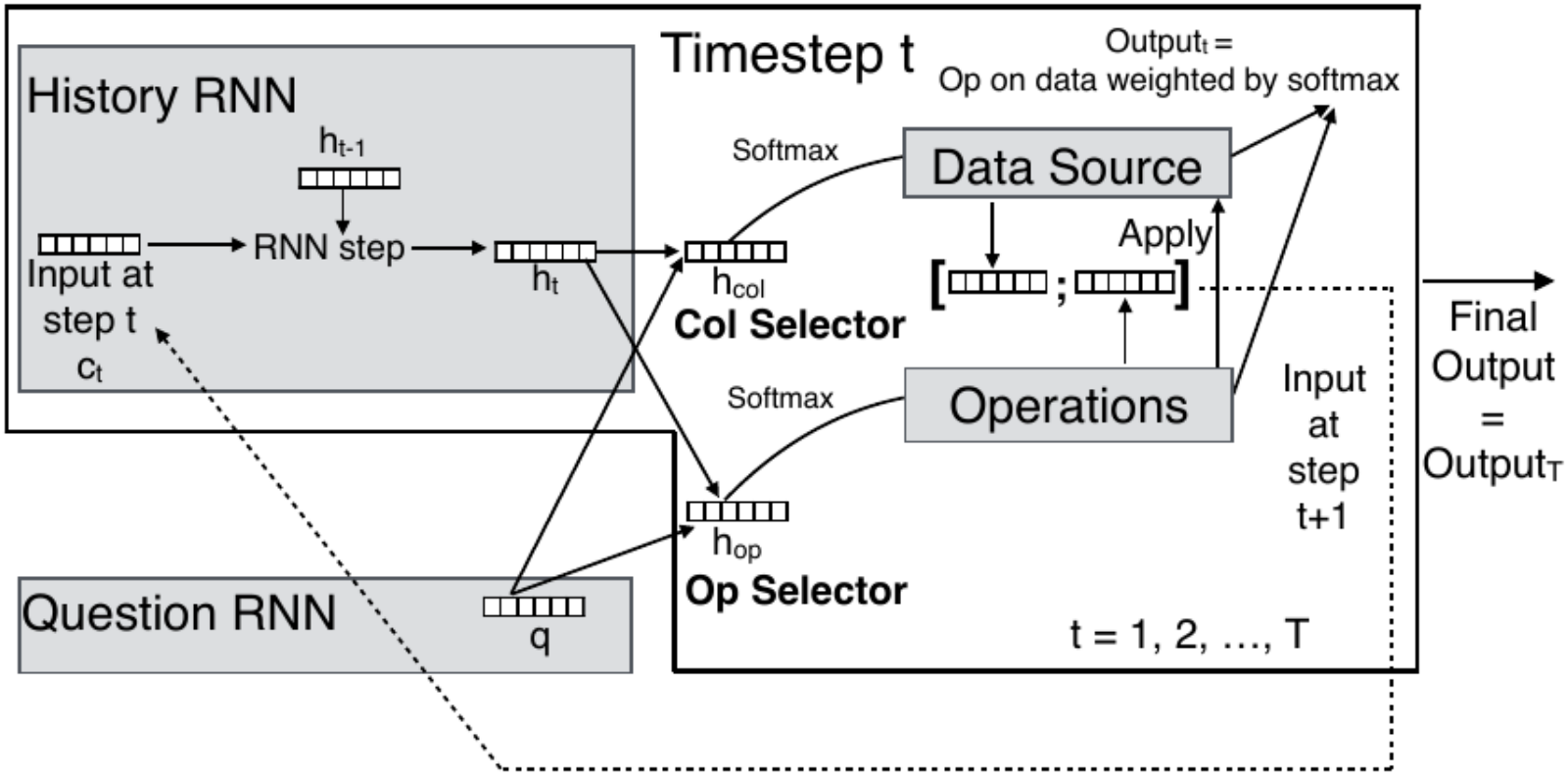


Idea

- Step-by-step fusion of different operators (and arguments)
- Trained by MSE
- Fully differentiable
- Drawbacks
  - Only numerical results (?)
  - Exponential # of combinatorial states

Arthitecture

- Question RNN
- Selector (Controller)
- Operators
- History RNN



Selector (Controller)

- Select an operator

$$\alpha_t^{op} = softmax(U \tanh(W^{op}[q; h_t]))$$

- Select a column for processing

$$\alpha_t^{col} = softmax(P \tanh(W^{col}[q; h_t]))$$

Operators

Type	Operation	Definition
Aggregate	Sum	$sum_t[j] = \sum_{i=1}^{M} row\_select_{t-1}[i] * table[i][j], \forall j = 1, 2, \ldots, C$
	Count	$count_t = \sum_{i=1}^{M} row\_select_{t-1}[i]$
Arithmetic	Difference	$diff_t = scalar\_output_{t-3} - scalar\_output_{t-1}$
Comparison	Greater	$g_t[i][j] = table[i][j] > pivot_g, \forall (i,j), i = 1, \ldots, M, j = 1, \ldots, C$
	Lesser	$l_t[i][j] = table[i][j] < pivot_l, \forall (i,j), i = 1, \ldots, M, j = 1, \ldots, C$
Logic	And	$and_t[i] = min(row\_select_{t-1}[i], row\_select_{t-2}[i]), \forall i = 1, 2, \ldots, M$
	Or	$or_t[i] = max(row\_select_{t-1}[i], row\_select_{t-2}[i]), \forall i = 1, 2, \ldots, M$
Assign Lookup	assign	$assign_t[i][j] = row\_select_{t-1}[i], \forall (i,j) i = 1, 2, \ldots, M, j = 1, 2, \ldots, C$
Reset	Reset	$reset_t[i] = 1, \forall i = 1, 2, \ldots, M$

$$\beta_{op} = softmax(ZU(op))$$

$$pivot_{op} = \sum_{i=1}^{N} \beta_{op}(i) qn_i$$

Type	Operation	Definition
Aggregate	Sum	$sum_t[j] = \sum_{i=1}^{M} row\_select_{t-1}[i] * table[i][j], \forall j = 1, 2, \ldots, C$
	Count	$count_t = \sum_{i=1}^{M} row\_select_{t-1}[i]$
Arithmetic	Difference	$diff_t = scalar\_output_{t-3} - scalar\_output_{t-1}$
Comparison	Greater	$g_t[i][j] = table[i][j] > pivot_g, \forall (i,j), i = 1, \ldots, M, j = 1, \ldots, C$
	Lesser	$l_t[i][j] = table[i][j] < pivot_l, \forall (i,j), i = 1, \ldots, M, j = 1, \ldots, C$
Logic	And	$and_t[i] = min(row\_select_{t-1}[i], row\_select_{t-2}[i]), \forall i = 1, 2, \ldots, M$
	Or	$or_t[i] = max(row\_select_{t-1}[i], row\_select_{t-2}[i]), \forall i = 1, 2, \ldots, M$
Assign Lookup	assign	$assign_t[i][j] = row\_select_{t-1}[i], \forall (i,j) i = 1, 2, \ldots, M, j = 1, 2, \ldots, C$
Reset	Reset	$reset_t[i] = 1, \forall i = 1, 2, \ldots, M$

Execution Output

More formally, the output variables are given by:

$$scalar\_answer_t = \alpha_t^{op}(\text{count})\, count_t + \alpha_t^{op}(\text{difference})\, diff_t + \sum_{j=1}^{C} \alpha_t^{col}(j)\alpha_t^{op}(\text{sum})\, sum_t[j],$$

$$lookup\_answer_t[i][j] = \alpha_t^{col}(j)\alpha_t^{op}(\text{assign})\, assign_t[i][j], \forall (i,j) i = 1, 2, \ldots, M, j = 1, 2, \ldots, C$$

The row selector variable is given by:

$$row\_select_t[i] = \alpha_t^{op}(\text{and})\, and_t[i] + \alpha_t^{op}(\text{or})\, or_t[i] + \alpha_t^{op}(\text{reset})\, reset_t[i] + \sum_{j=1}^{C} \alpha_t^{col}(j)(\alpha_t^{op}(\text{greater})\, g_t[i][j] + \alpha_t^{op}(\text{lesser})\, l_t[i][j]), \forall i = 1, \ldots, M$$

Text Matching

- Example: what is the sum of elements in column B whose field in column C is **word:1** and field in column A is **word:7**?
- Text matching (no textual operation or output)

$$row\_select_t[i] = \alpha_t^{op}(\text{and})and_t[i] + \alpha_t^{op}(\text{or})or_t[i] + \alpha_t^{op}(\text{reset})reset_t[i] +$$
$$\sum_{j=K+1}^{C} \alpha_t^{col}(j)(\alpha_t^{op}(\text{greater})g_t[i][j] + \alpha_t^{op}(\text{lesser})l_t[i][j]) +$$
$$\sum_{j=1}^{K} \alpha_t^{col}(j)(\alpha_t^{op}(\text{text\_match})text\_match_t[i][j], \forall i = 1, \ldots, M$$

Training Objective

- Scalar answer

$$L_{scalar}(scalar\_answer_T, y) = \begin{cases} \frac{1}{2}a^2, \text{if } a \leq \delta \\ \delta a - \frac{1}{2}\delta^2, \text{otherwise} \end{cases}$$

- List answer

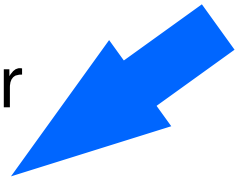
$$L_{lookup}(lookup\_answer_T, y) = -\frac{1}{MC}\sum_{i=1}^{M}\sum_{j=1}^{C}\Big(y[i,j]\log(lookup\_answer_T[i,j]) + (1-y[i,j])\log(1-lookup\_answer_T[i,j])\Big)$$

- Overall

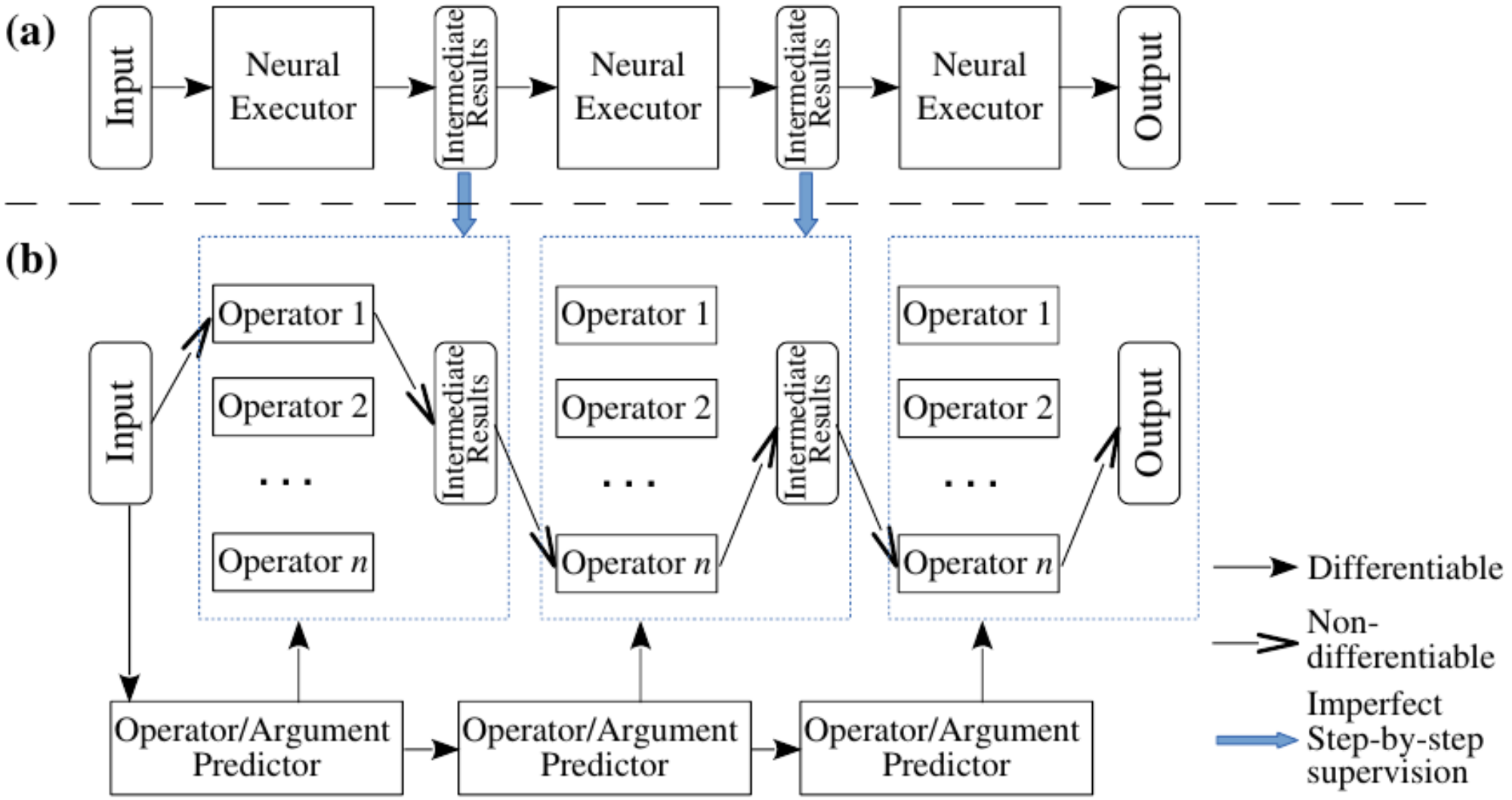
$$L = \frac{1}{N}\sum_{k=1}^{N}\Big([n_k == True]L_{scalar}^{(k)} + [n_k == False]\lambda L_{lookup}^{(k)}\Big)$$

Outline

- Introduction
- Symbolic execution
  - (Fully supervised) Neural programmer interpreter
  - (Weakly supervised) Neural symbolic machine
  - “Spurious programs” and inductive programming
    - Learning semantic parsers from denotations
    - DeepCoder
    - More thoughts on “spurious programs”
- Distributed execution
  - Learning to execute
  - Neural enquirer
  - Differentiable neural computer
- Hybrid execution
  - Neural programmer
  - Coupling approach



Overview


(a)
Input
Neural Executor
Intermediate Results
Neural Executor
Intermediate Results
Neural Executor
Output
(b)
Input
Operator 1
Operator 2
...
Operator n
Intermediate Results
Operator 1
Operator 2
...
Operator n
Intermediate Results
Operator 1
Operator 2
...
Operator n
Output
Operator/Argument Predictor
Operator/Argument Predictor
Operator/Argument Predictor
Differentiable
Non-differentiable
Imperfect Step-by-step supervision

Idea

- Fully neuralized model exhibits (imperfect) symbolic interpretation
- Use neural networks' intermediate results to learn initial policy of the symbolic executor
- Improve policy by REINFORCE or MML variants

Thank you for listening

Q & A?

References

1.Scott Reed, Nando de Freitas. “Neural programmer-interpreters.” In *ICLR*, 2016.

2.Jonathon Cai, Richard Shin, Dawn Song. “Making neural programming architectures generalize via recursion.” In *ICLR*, 2017.

3. Chen Liang, Jonathan Berant, Quoc Le, Kenneth D. Forbus, Ni Lao. “Neural symbolic machines: Learning semantic parsers on freebase with weak supervision.” *arXiv preprint arXiv:1611.00020*, 2016.

4. Arun Chaganty, Percy Liang. “Inferring logical forms from denotations.” In *ACL*, 2016.

5. Matej Balog, Alexander L. Gaunt, Marc Brockschmidt, Sebastian Nowozin, Daniel Tarlow. “DeepCoder: Learning to write programs.” In *ICLR*, 2017.

6. Wojciech Zaremba, Ilya Sutskever. “Learning to execute.” *arXiv preprint arXiv:1410.4615*, 2014.

7. Pengcheng Yin, Zhengdong Lu, Hang Li, Ben Kao. “Neural enquirer: Learning to query tables with natural language.” In *IJCAI*, 2016.

8. Alex Graves, et al. “Hybrid computing using a neural network with dynamic external memory.” *Nature*, 2016.

9. Arvind Neelakantan, Quoc Le, Ilya Sutskever. “Neural programmer: Inducing latent programs with gradient descent.” In *ICLR*, 2016.

10. Lili Mou, Zhengdong Lu, Hang Li, Zhi Jin. "Coupling distributed and symbolic execution for natural language queries." *arXiv preprint arXiv:1612.02741*, 2016.