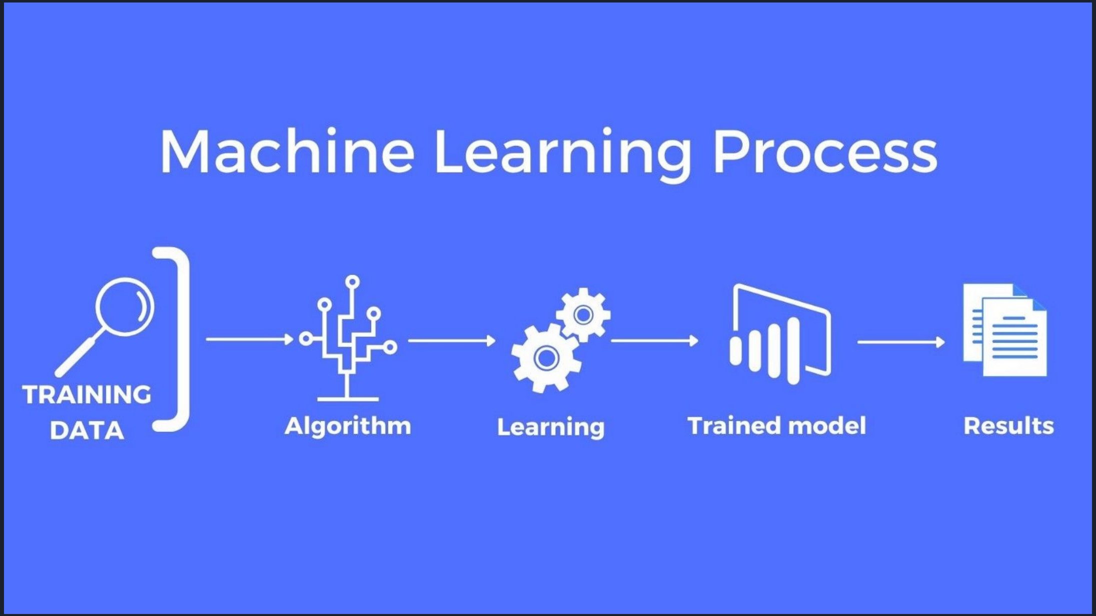
Neural Programmer-Interpreters

Scott Reed & Nando de Freitas

Presented by Benjamín Farías V.

Contents

Context - Machine Learning



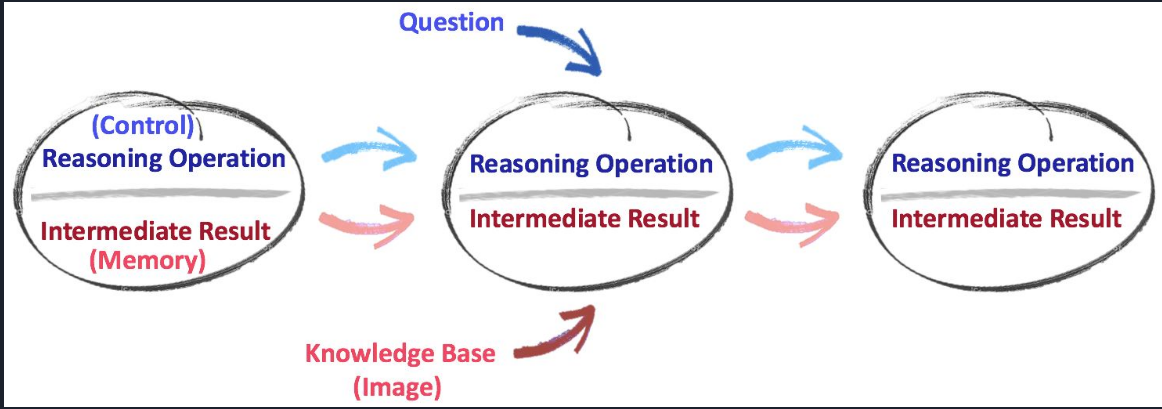
Pros:

- Good results
- Extensively studied

Cons:

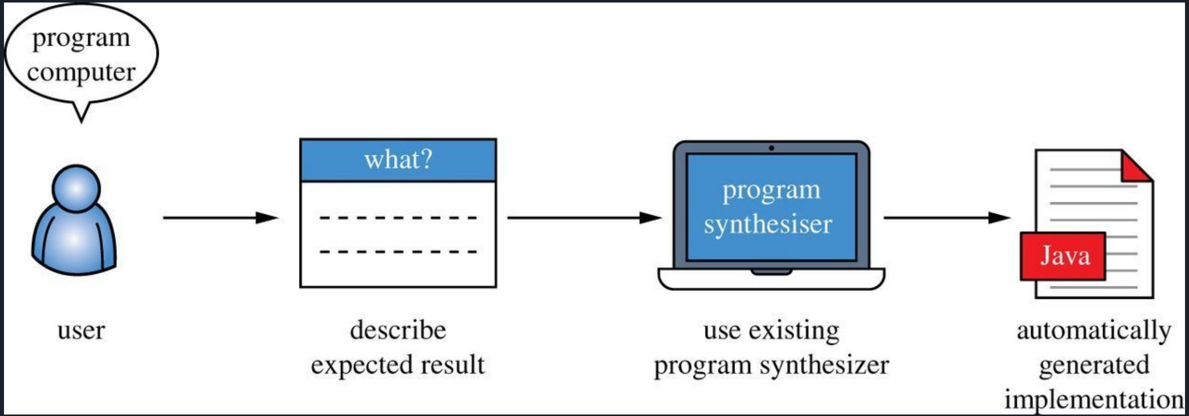
- Learns superficial patterns
- Requires a ton of data

Context - Machine Reasoning



- Learns logical rules from data
- Closer to human learning
- Requires less data

Context - Program Learning

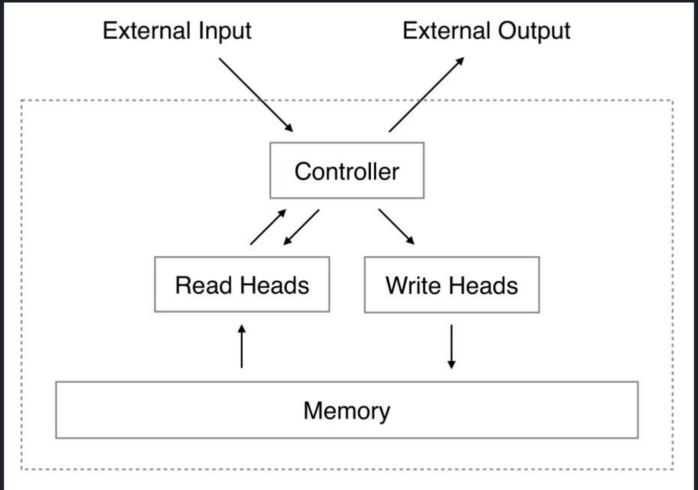


- Networks that can deduce and learn programs
- Machines could create their own programs!
- **Future:** Human Level AI?

Contents

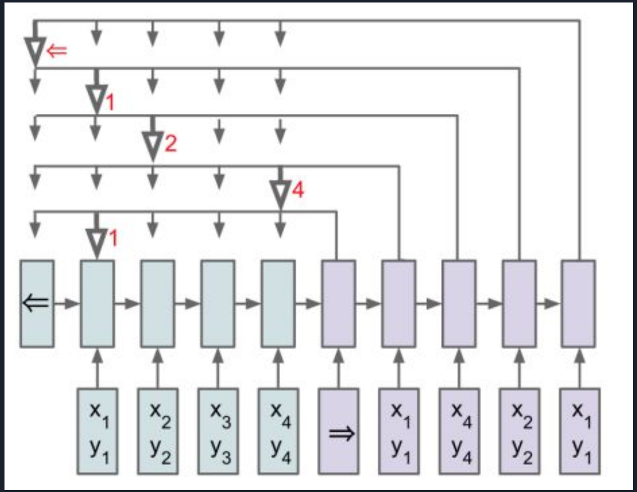
Related Work - RNNs

Neural Turing Machines



Learn & Execute Simple Programs

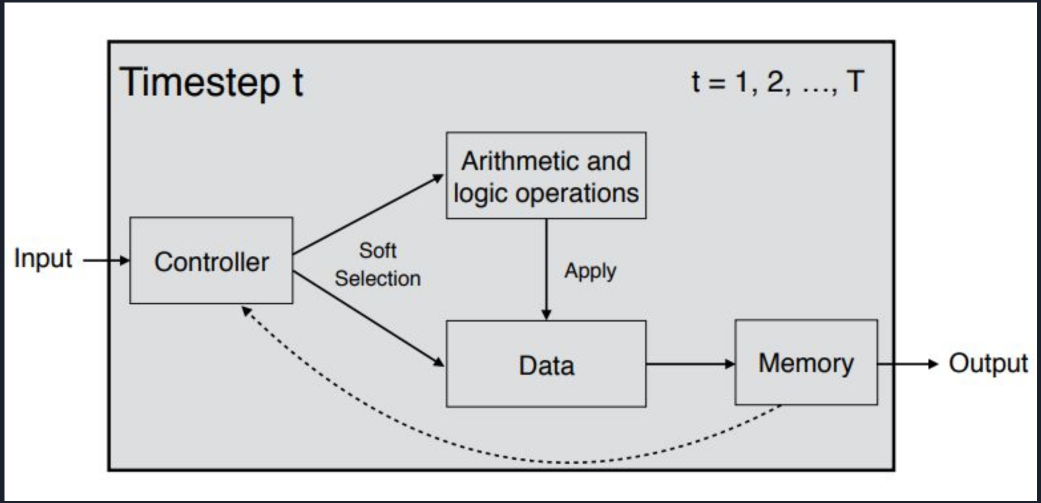
Pointer Networks



Output Space Depends on Input

Related Work - Program Induction

Neural Programmer



RNN + Controller + Operation + Memory

Related Work - Program Induction

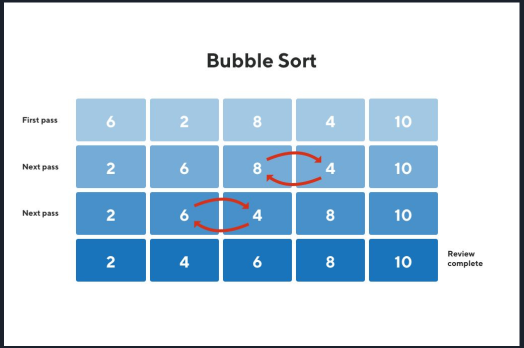
Curriculum Learning

Humans and animals learn much better when the examples are not randomly presented but organized in a meaningful order which illustrates gradually more concepts, and gradually more complex ones. . . . and call them "curriculum learning".

Bengio et al. (2009)

Contents

Model - Neural Programmer-Interpreter (NPI)

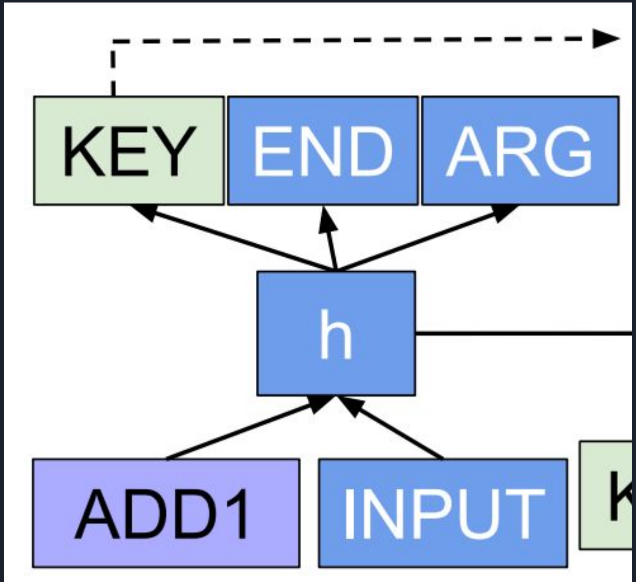


[0.32, 0.77, 0.67, ..., 0.42]

- Learn to **represent and interpret** programs

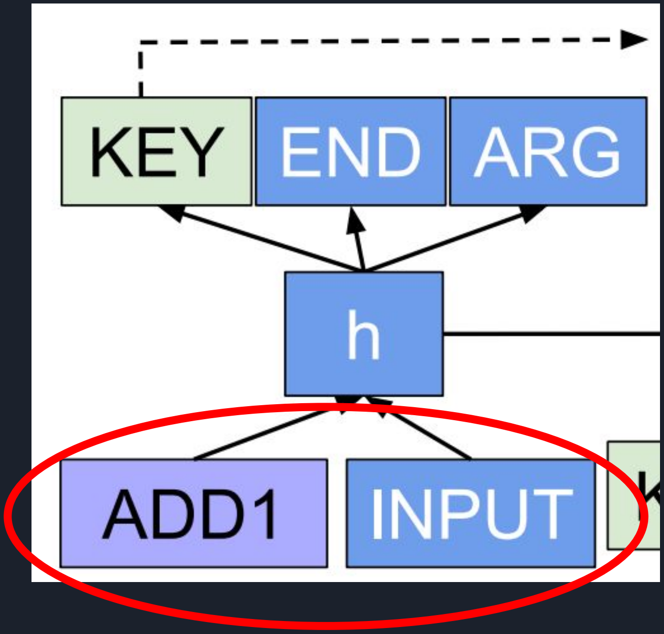
- **Programmer:** Learns new program representations
- **Interpreter:** Executes learned programs over more complex tasks

Model - NPI Core



- **Multi-Layer LSTM** network
- Acts as a program router
- Decides which program to call next

Model - NPI Core

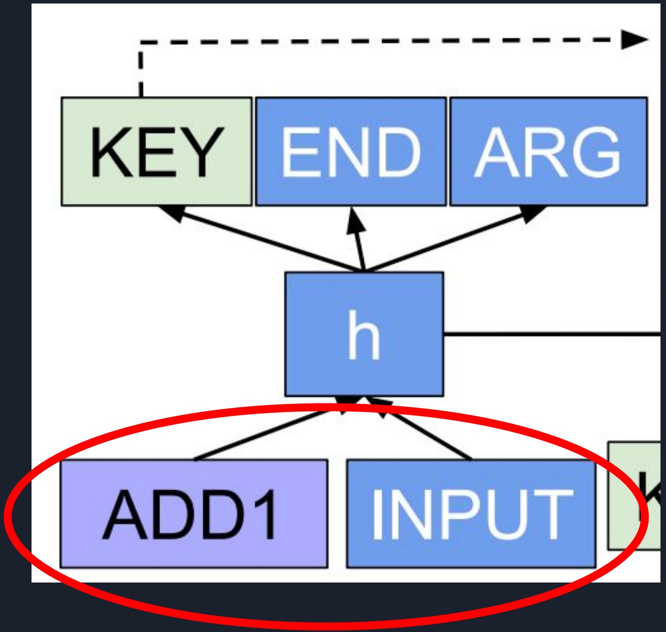


Program **State**

Input Components

- **State:** Environment observation + program arguments
- **Program:** Current program embedding

Model - NPI Core



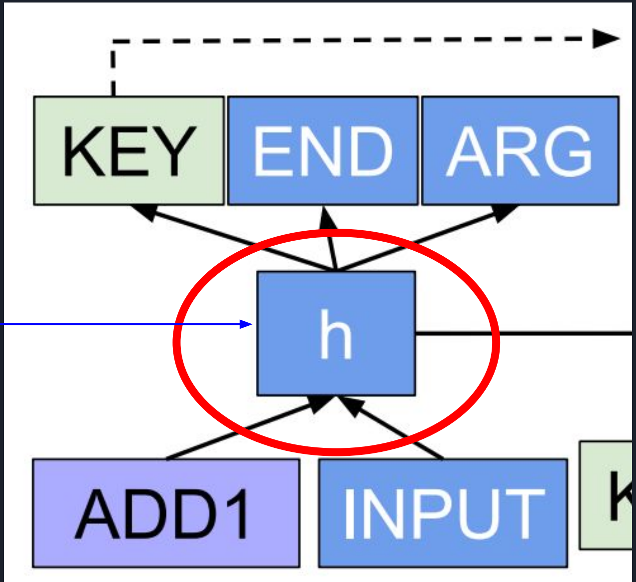
Program **State**

Input Components

- The **State** is obtained from a domain-specific encoder
- The **Program** is obtained from a memory module

Model - NPI Core

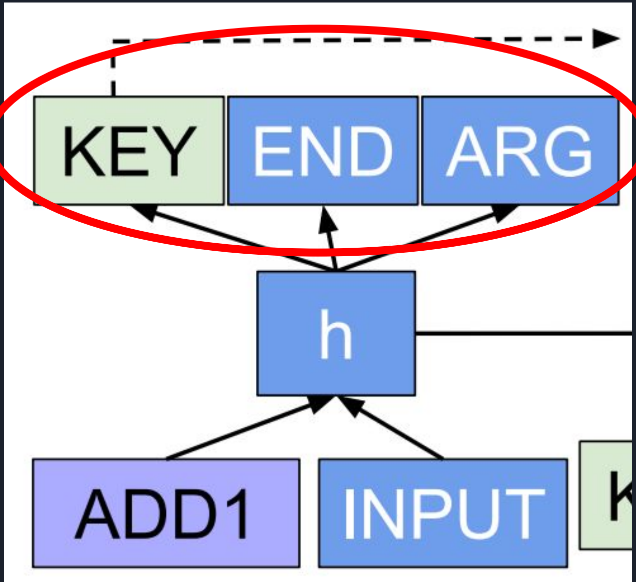
h-1



Hidden Component

- Receives the last hidden state **(h-1)**
- Computes the feed-forward step

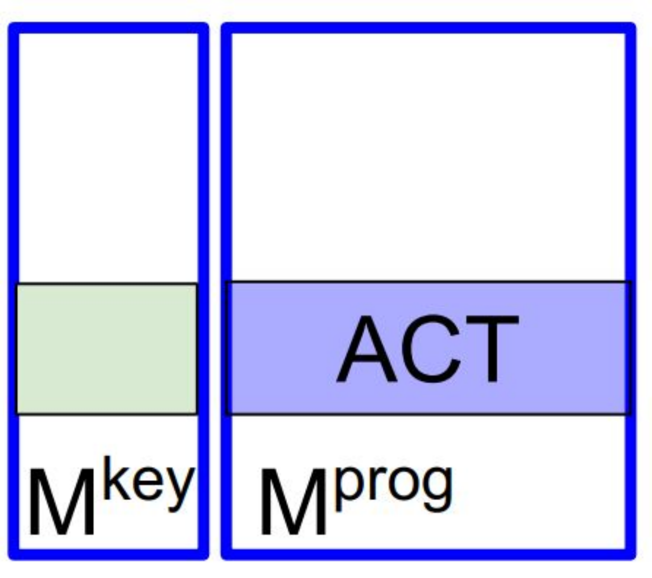
Model - NPI Core



Output Components

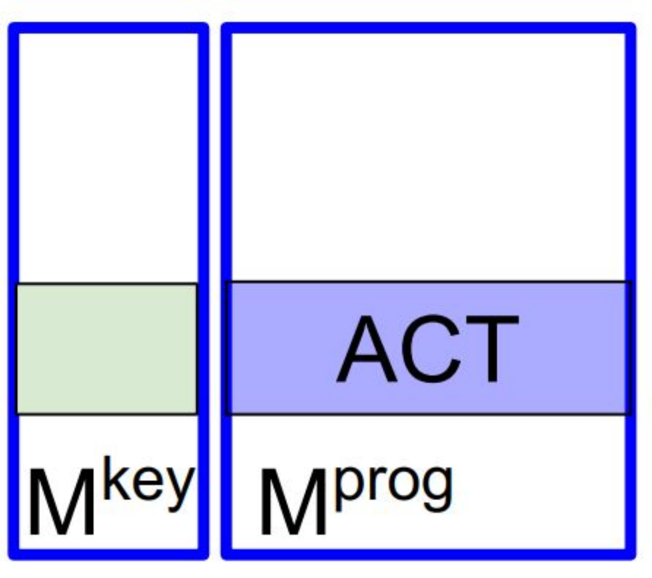
- **Key:** Lookup key embedding for next program
- **End:** Probability of returning
- **Arg:** Arguments for next program

Model - NPI Memory



- Global memory, has two components
- Each row in the **Key** component corresponds to the same row in the **Prog** component

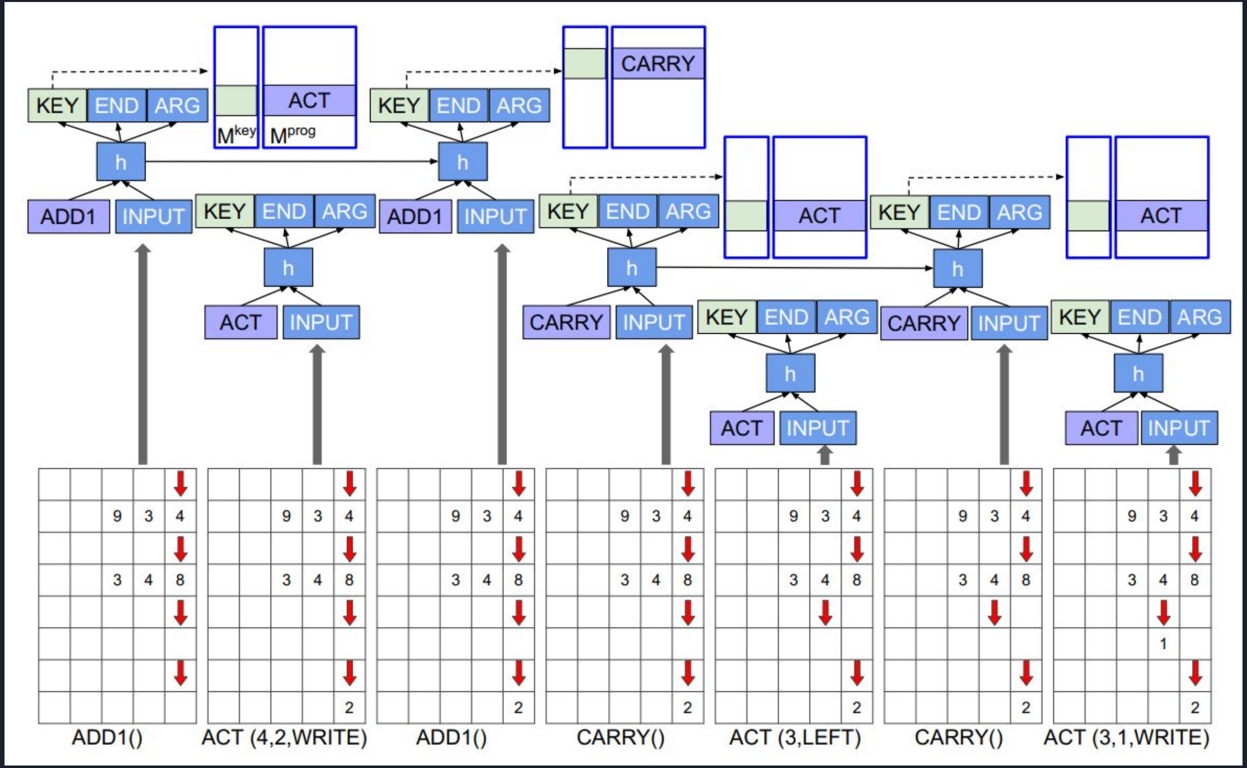
Model - NPI Memory



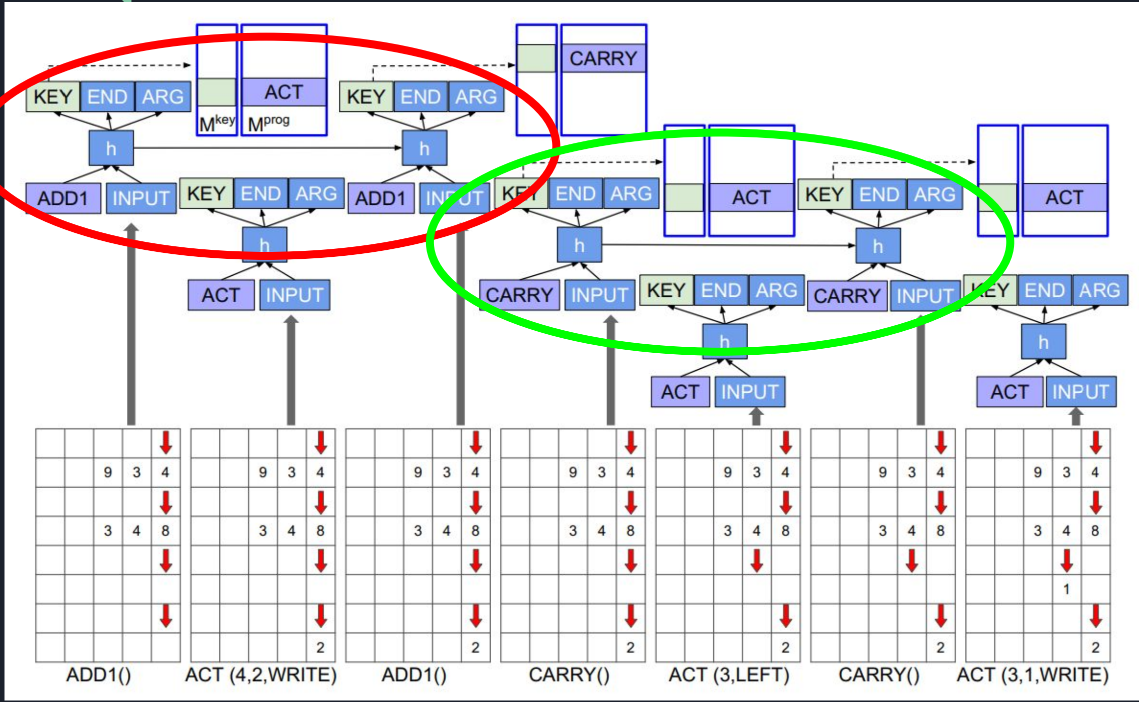
Memory Components

- **Key:** Stores all program keys
- **Prog:** Stores all program embeddings

Model - NPI Network (Addition)



Model - NPI Network (Addition)

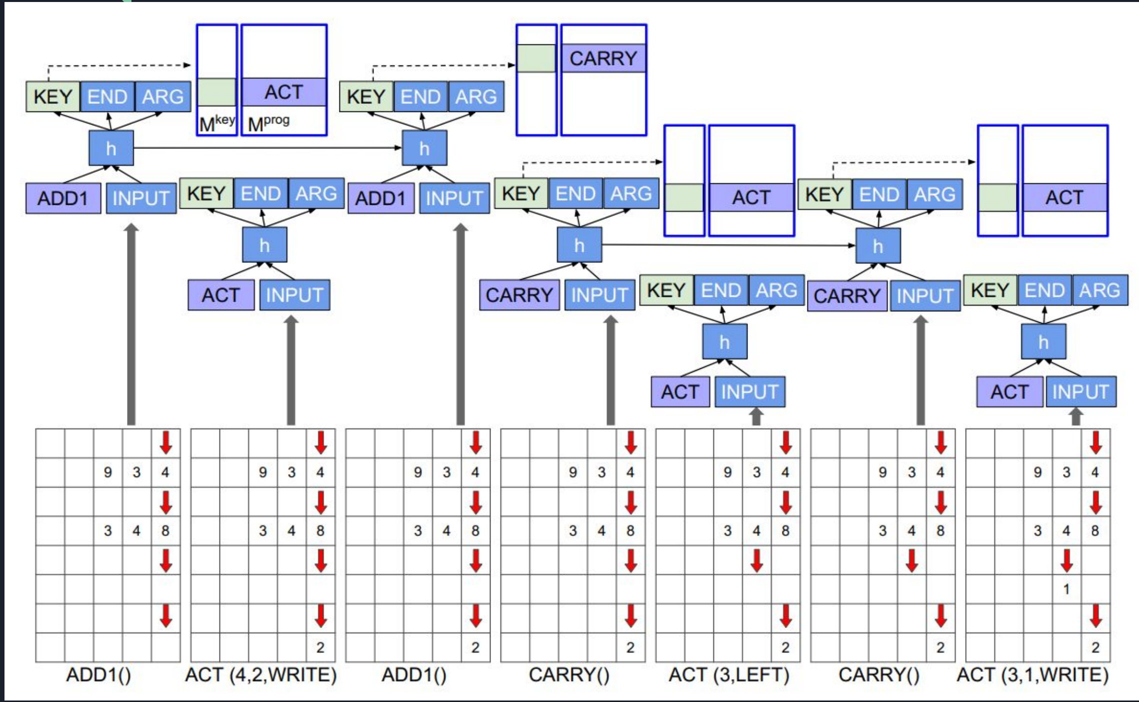


- Each program has one **NPI Core Network**
- **All NPI Cores** share the same weights
- Works like a **call stack**

ADD1( )

CARRY ( )

Model - NPI Network (Addition)



[ADD1]

[ADD1, ACT]

[ADD1]

[ADD1, CARRY]

[ADD1, CARRY, ACT]

[ADD1, CARRY]

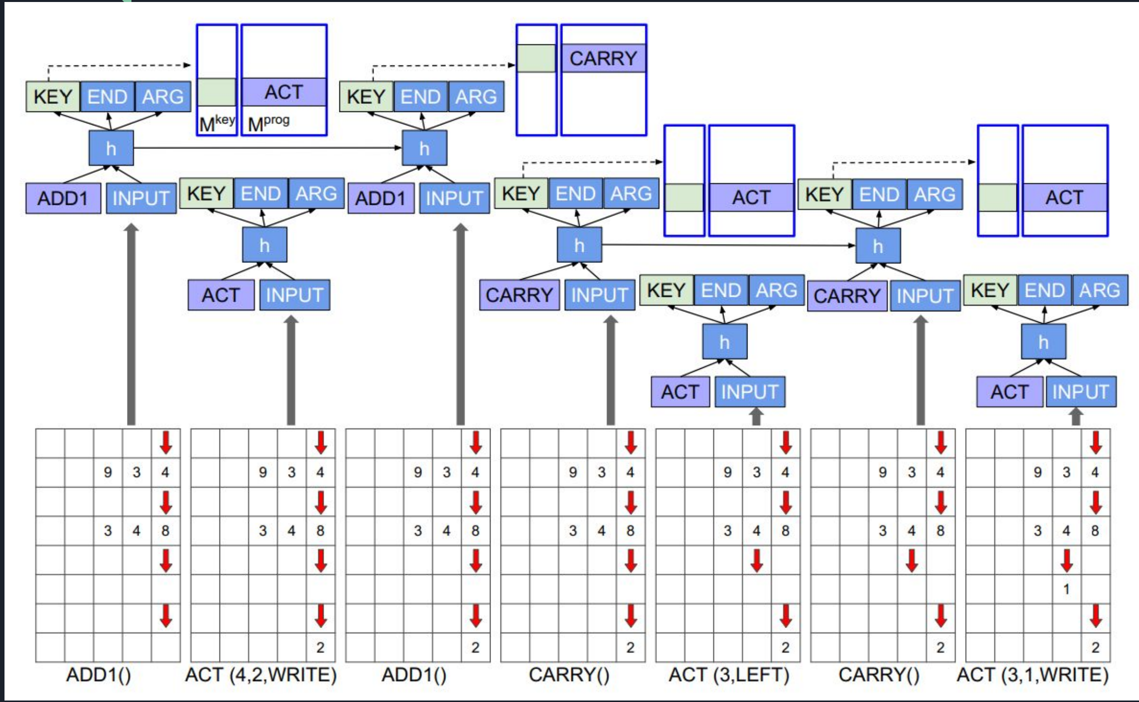
[ADD1, CARRY, ACT]

[ADD1, CARRY]

[ADD1]

[ ]

Model - NPI Network (Addition)



- The memory is shared
- The output is a sequence of **actions**

1. **ACT (4, 2, WRITE)**
2. **ACT (3, LEFT)**
3. **ACT (3, 1, WRITE)**

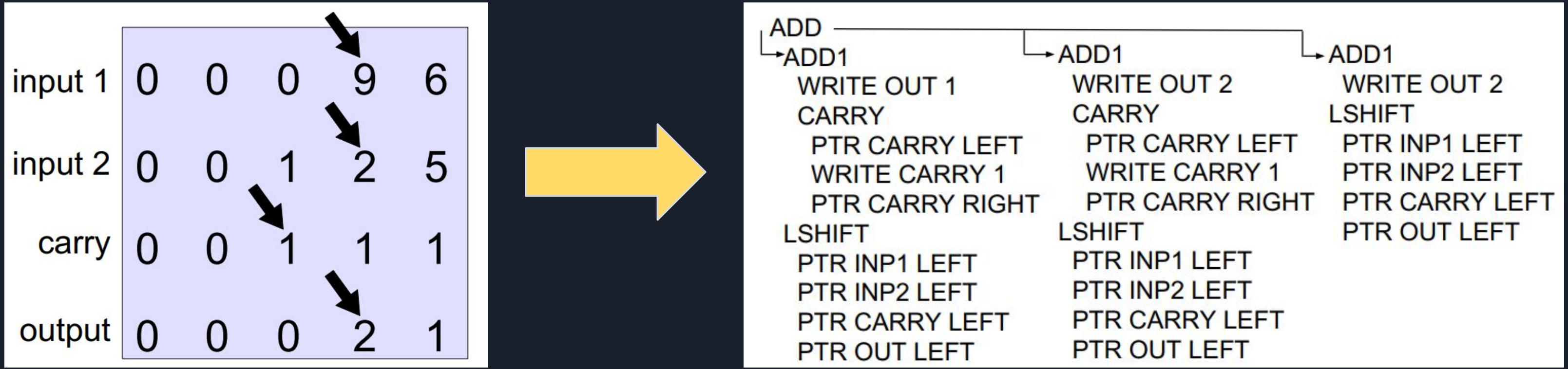
Model - NPI Training

```
ADD1
  WRITE OUT 2
LSHIFT
  PTR INP1 LEFT
  PTR INP2 LEFT
  PTR CARRY LEFT
  PTR OUT LEFT
```

- Use execution traces for real programs
- Predict the next program to be called
- Apply **curriculum learning** to focus on programs that the model is failing at

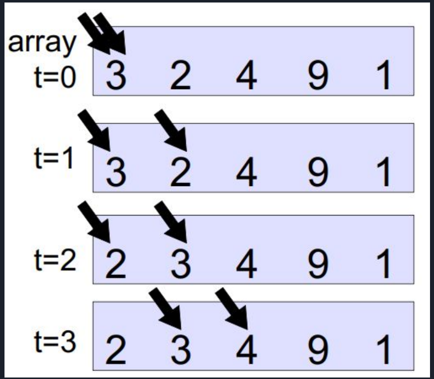
Contents

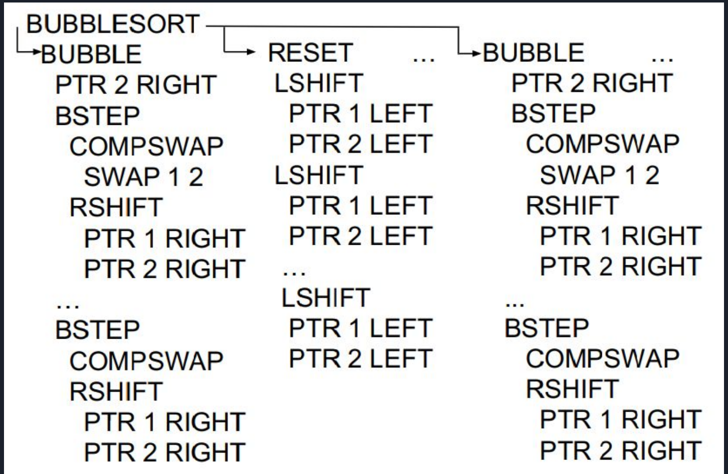
Experiments - Addition



- Addition of two base-10 numbers using a **scratch pad**
- The model can **move** the pointers and **write** numbers
- **Testing:** Addition for numbers with more digits

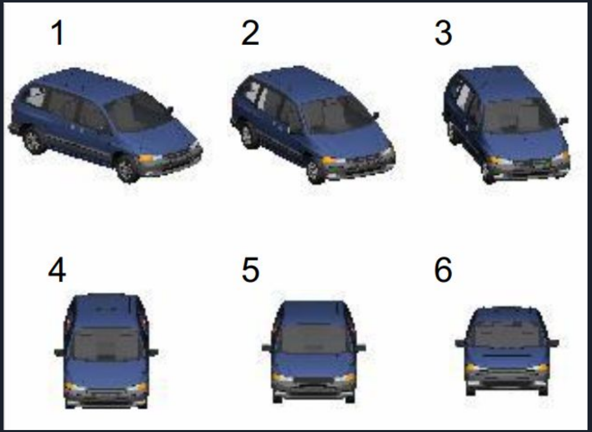
Experiments - Sorting





- Array sorting with **Bubble Sort** on a **scratch pad**
- The model can **move** the pointers and **swap** elements
- **Testing:** Sorting of longer arrays

Experiments - Canonicalize 3D Models

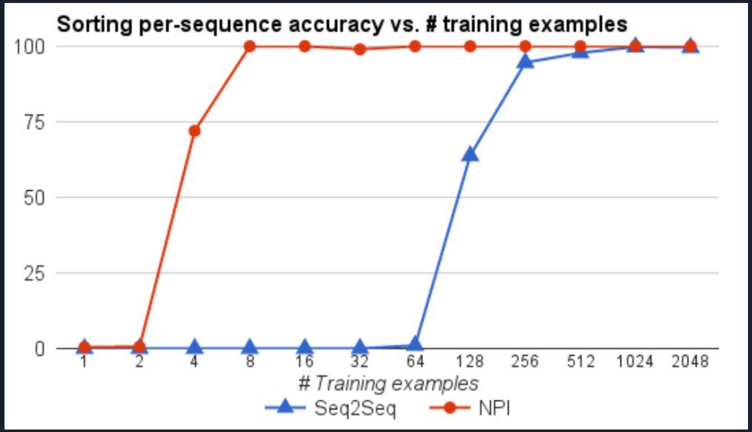



```
GOTO 1 2
  HGOTO
    RGOTO
      ACT(RIGHT)
      ACT(RIGHT)
      ACT(RIGHT)
  VGOTO
    DGOTO
      ACT(DOWN)
      ACT(DOWN)
```

- Move the **camera** to the target view by looking at the **image**
- The model can only see the **current rendering** of the car
- **Testing:** New car models and different positions

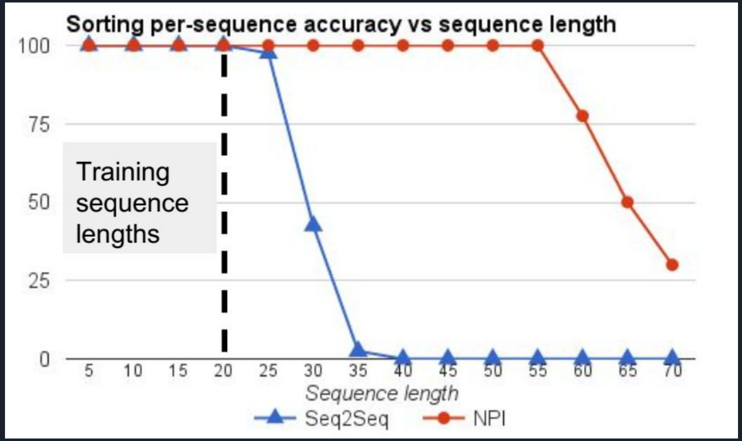
Experiments - Results (Sorting)

Accuracy VS # Training Examples



NPI learns at a way **faster rate** compared to a **Seq2Seq LSTM**

Accuracy VS Sequence Length



NPI generalizes to **longer sequences** compared to a **Seq2Seq LSTM**

Experiments - Results (Multitasking)

Task	Single	Multi	+ Max
Addition	100.0	97.0	97.0
Sorting	100.0	100.0	100.0
Canon. seen car	89.5	91.4	91.4
Canon. unseen	88.7	89.9	89.9
Maximum	-	-	100.0

- Single-Task models perform really well
- The **Multi-Task** model is comparable to **all** single-task models!
- **MAX** can be learned **without affecting performance** on previous tasks!

Contents

Conclusions

- The **NPI** can learn programs from very dissimilar environments
- **Strong generalization** in comparison to **Seq2Seq LSTMs**
- A trained **NPI** with a fixed core can continue to learn without forgetting

Contents

Personal Criticism

Good:

- Interesting approach to program learning by using composition
- Experiments show good generalization capabilities with little data

Bad:

- The model architecture is hard to understand from their explanation
- Training requires already having an implementation for each program

Bibliography

[1] Reed, S., De Freitas, N. (2016). *Neural Programmer-Interpreters.*

[2] Graves, A., Wayne, G., Danihelka, I. (2014). *Neural Turing Machines.*

[3] Vinyals, O., Jaitly, N., Fortunato, M. (2015). *Pointer Networks.*

[4] Bengio, Y., Louradour, J., Collobert, R., Weston, J. (2009). *Curriculum Learning.*

[5] Neelakantan, A., Le, Q., Sutskever, I. (2016). *Neural Programmer: Inducing Latent Programs With Gradient Descent.*