Neural Programmer-Interpreters

By Scott Reed & Nando de Freitas

Presenter: Zeqi Li

Motivation

Why do we learn and use machine learning?

Motivation

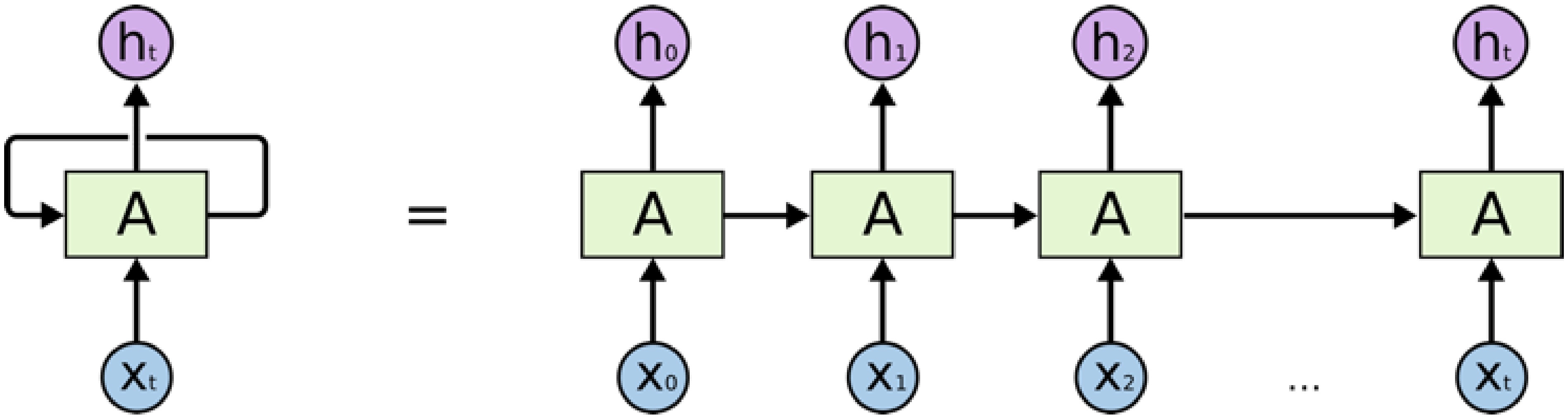
Consider the problem of teaching a machine to do some particular task automatically

Task could be as simple as adding numbers or as difficult as driving a car

Motivation

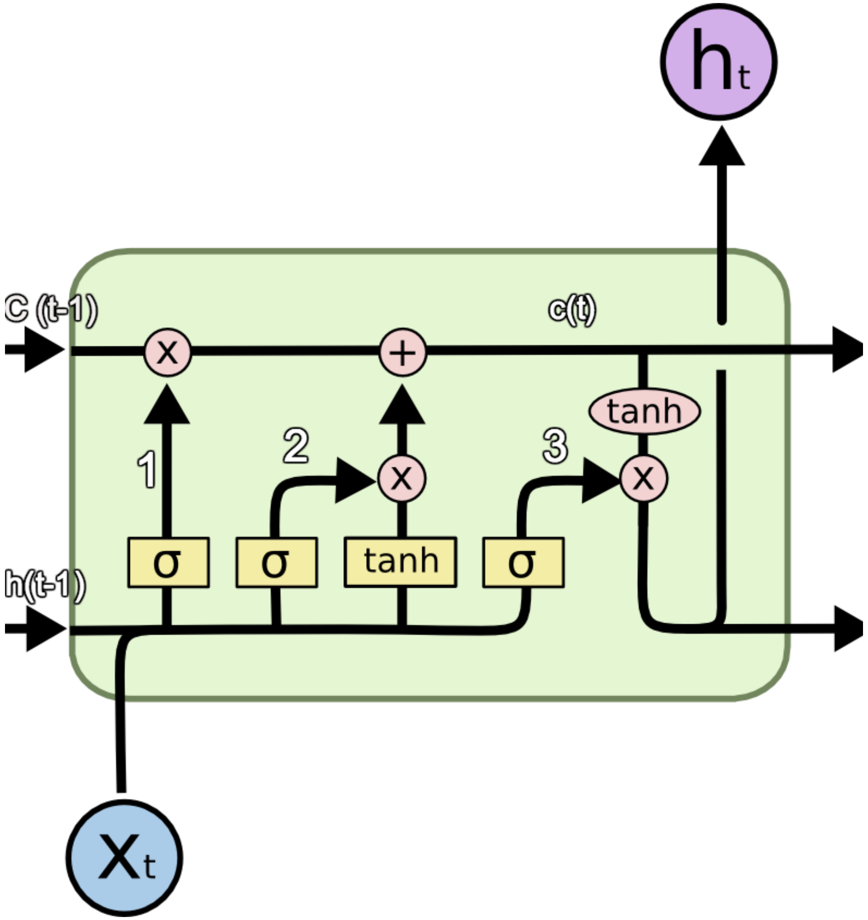
Neural Programmer-Interpreters (NPI) is an attempt to use **neural methods** to train machines to carry out simple tasks based on a **small amount of training data**.

Recurrent neural network (RNN)



- RNN is a neural network with feedback
- Hidden state is to capture history information and current state of the network

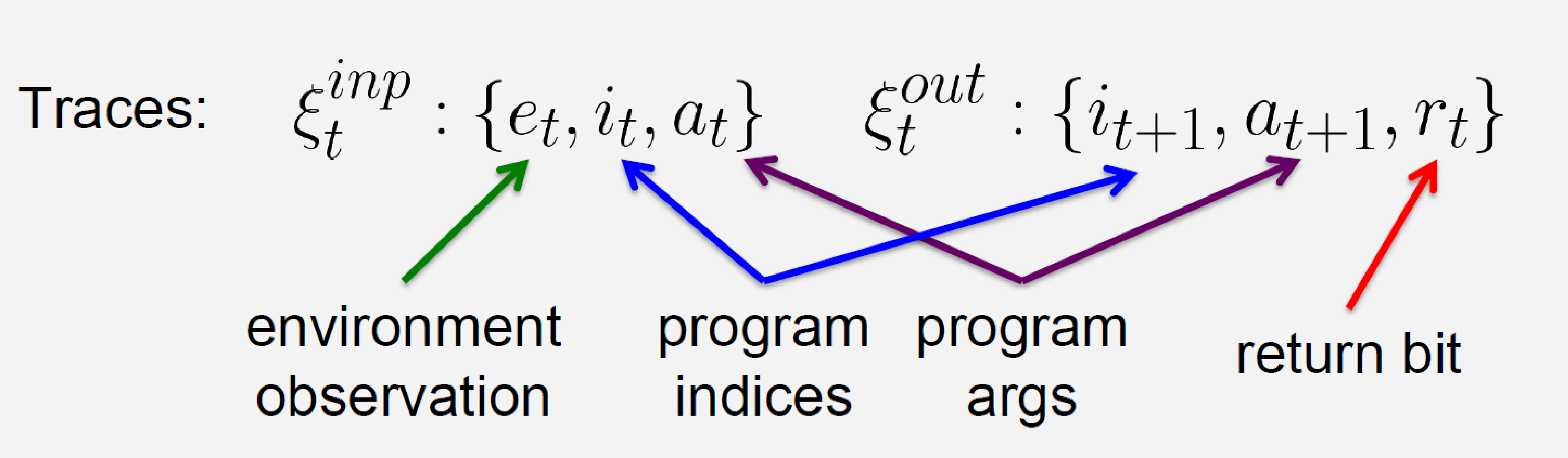
Long Short Term Memory (LSTM)



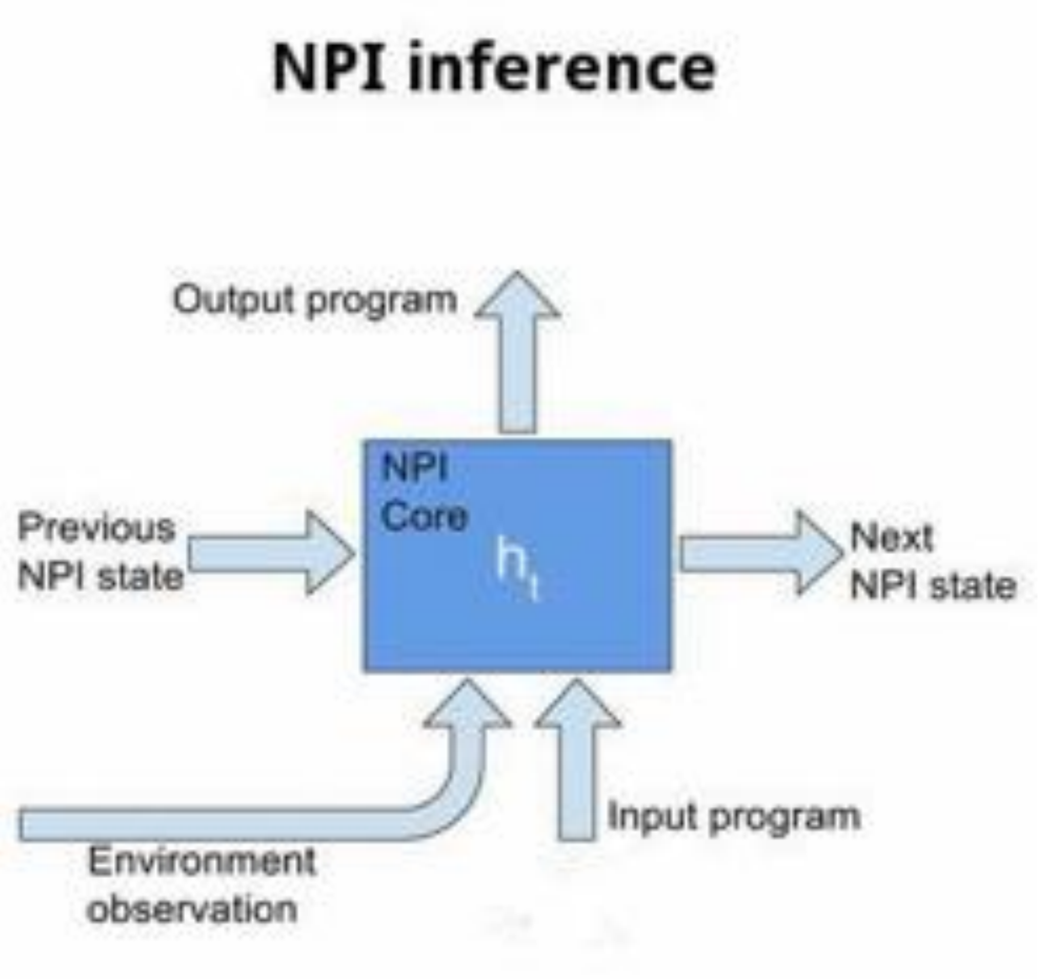
- LSTM is a special kind of RNN
- Gates are used to control information flow. Just like a valve

Model

- The NPI core is a LSTM network that learns to represent and execute programs given their execution traces



NPI core module



Algorithm - inference

Algorithm 1 Neural programming inference

Inputs: Environment observation e, program id i, arguments a, stop threshold α

function RUN(i, a)

  $h \leftarrow \mathbf{0}, r \leftarrow 0, p \leftarrow M^{\text{prog}}_{i,:}$ (1) ▷ Init LSTM and return probability.

  while $r < \alpha$ **do**

    $s \leftarrow f_{enc}(e, a)$ (2), $h \leftarrow f_{lstm}(s, p, h)$ ▷ Feed-forward NPI one step.

    $r \leftarrow f_{end}(h)$ (3), $k \leftarrow f_{prog}(h)$ (4), $a_2 \leftarrow f_{arg}(h)$ (5)

    $i_2 \leftarrow \arg\max_{j=1..N} (M^{\text{key}}_{j,:})^T k$ (6) ▷ Decide the next program to run.

    if i == ACT **then** $e \leftarrow f_{env}(e, p, a)$ (7) ▷ Update the environment based on ACT.

    else RUN(i_2, a_2) ▷ Run subprogram i_2 with arguments a_2

(1): M^{prog} and M^{key} are memory storing program embeddings and program keys

(2): f_{enc} is a domain-specific encoder (for different tasks, have different encoders)

(3): f_{end} is to calculate the probability of finishing the program

(4): f_{prog} is to retrieve the next program key from memory

(5): f_{arg} is to return the next program's arguments

(6): $(M^{key}_{j,:})^T k$ is to measure cosine similarity

(7): f_{env} is a domain-specific transition mapping

Algorithm - inference

Algorithm 1 Neural programming inference

1: **Inputs**: Environment observation e, program id i, arguments a, stop threshold α
2: **function** RUN(i, a)
3: $\quad h \leftarrow \mathbf{0}, r \leftarrow 0, p \leftarrow M^{\text{prog}}_{i,:}$ ▷ Init LSTM and return probability.
4: $\quad$ **while** $r < \alpha$ **do**
5: $\quad\quad s \leftarrow f_{enc}(e, a), h \leftarrow f_{lstm}(s, p, h)$ ▷ Feed-forward NPI one step.
6: $\quad\quad r \leftarrow f_{end}(h), k \leftarrow f_{prog}(h), a_2 \leftarrow f_{arg}(h)$
7: $\quad\quad i_2 \leftarrow \arg\max_{j=1..N} (M^{\text{key}}_{j,:})^T k$ ▷ Decide the next program to run.
8: $\quad\quad$ **if** $i ==$ ACT **then** $e \leftarrow f_{env}(e, p, a)$ ▷ Update the environment based on ACT.
9: $\quad\quad$ **else** RUN(i_2, a_2) ▷ Run subprogram i_2 with arguments a_2

Line 3: M^{prog} and M^{key} are memory banks to store program embeddings and program keys

Algorithm - inference

Algorithm 1 Neural programming inference

1: **Inputs**: Environment observation e, program id i, arguments a, stop threshold α
2: **function** RUN(i, a)
3: $\quad h \leftarrow \mathbf{0}, r \leftarrow 0, p \leftarrow M^{\text{prog}}_{i,:}$ $\triangleright$ Init LSTM and return probability.
4: $\quad$ **while** $r < \alpha$ **do**
5: $\quad\quad s \leftarrow f_{enc}(e, a), h \leftarrow f_{lstm}(s, p, h)$ $\triangleright$ Feed-forward NPI one step.
6: $\quad\quad r \leftarrow f_{end}(h), k \leftarrow f_{prog}(h), a_2 \leftarrow f_{arg}(h)$
7: $\quad\quad i_2 \leftarrow \arg\max_{j=1..N}(M^{\text{key}}_{j,:})^T k$ $\triangleright$ Decide the next program to run.
8: $\quad\quad$ **if** i == ACT **then** $e \leftarrow f_{env}(e, p, a)$ $\triangleright$ Update the environment based on ACT.
9: $\quad\quad$ **else** RUN(i_2, a_2) $\triangleright$ Run subprogram i_2 with arguments a_2

Line 7: $(M^{key}_{j,:})^T k$ is directly measurement for cosine similarity

Training

Directly maximize the probability of the correct execution trace output $\boldsymbol{\xi}^{out}$ conditioned on $\boldsymbol{\xi}^{inp}$:

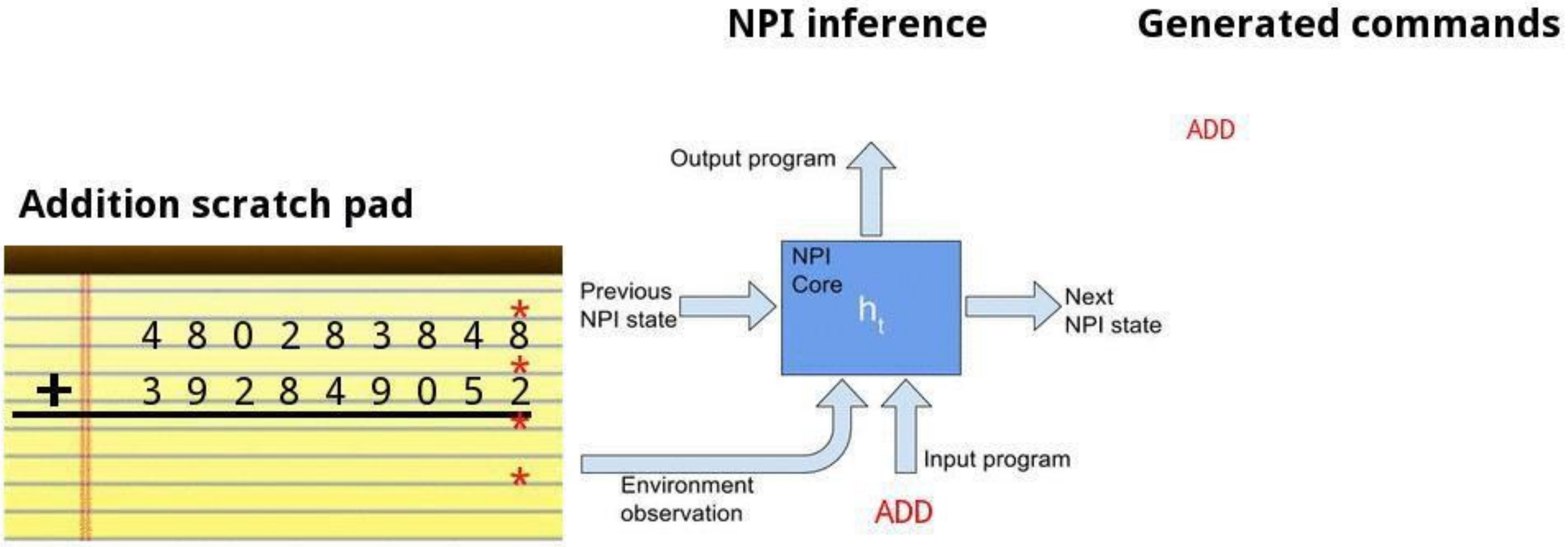
$$\theta^* = arg\max_{\theta} \sum_{(\boldsymbol{\xi}^{inp},\boldsymbol{\xi}^{out})} logP(\boldsymbol{\xi}^{out}|\boldsymbol{\xi}^{inp},\theta)$$

Then we can just run gradient ascent to optimize it

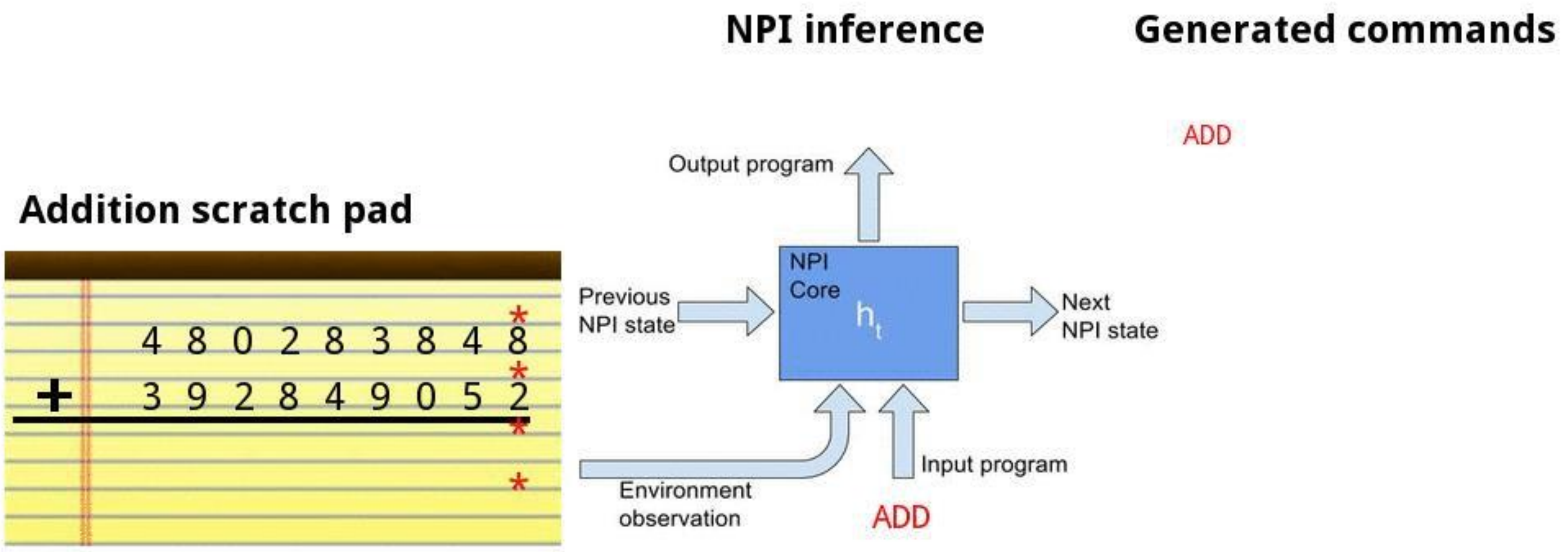
Tasks

- Addition
  - Teach the model the standard grade school algorithm of adding 2 base-10 numbers
- Sorting
  - Teach the model bubble sorting to sort an array of numbers in ascending order
- Canonicalizing 3D models
  - Teach the model to generate a trajectory of actions that delivers the camera to the target view, e.g, frontal pose at a 15° elevation

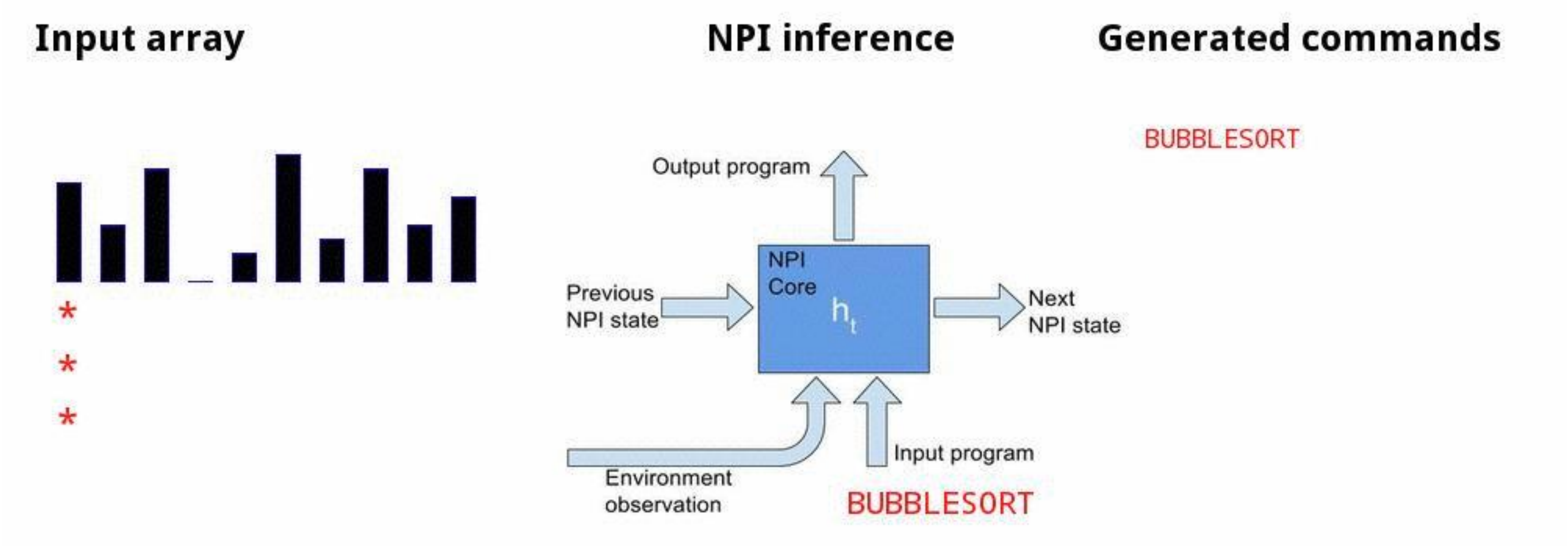
Adding numbers together



Addition demo



Bubble sort

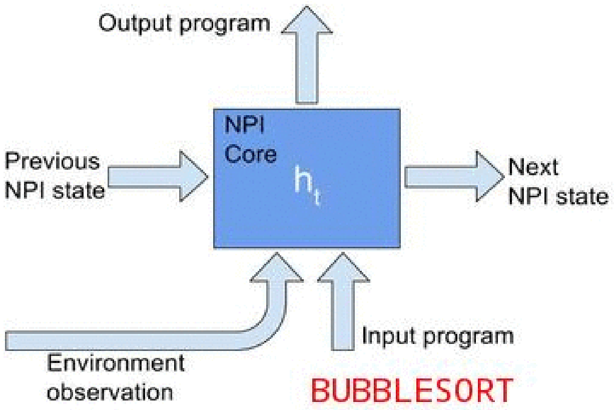


Input array
NPI inference
Generated commands
BUBBLESORT
Output program
NPI
Core
h_t
Previous
NPI state
Next
NPI state
Input program
Environment
observation
BUBBLESORT

Sorting demo

Input array

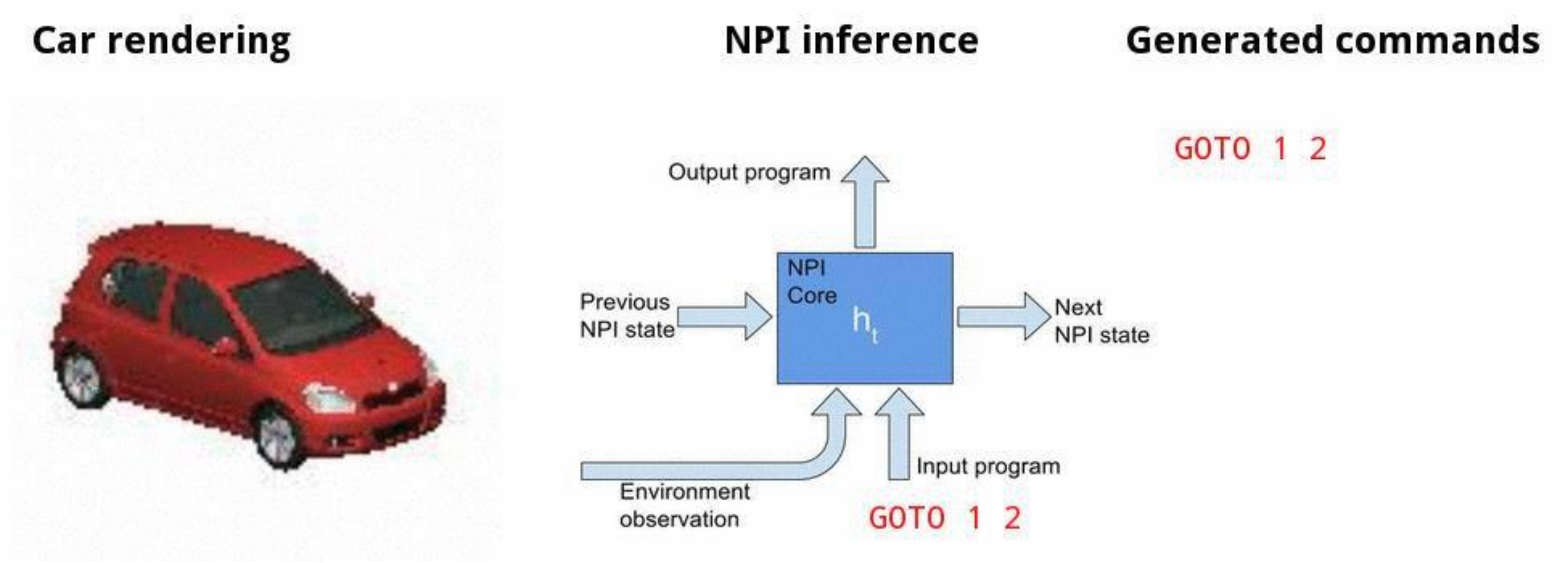
NPI inference



Generated commands

BUBBLESORT

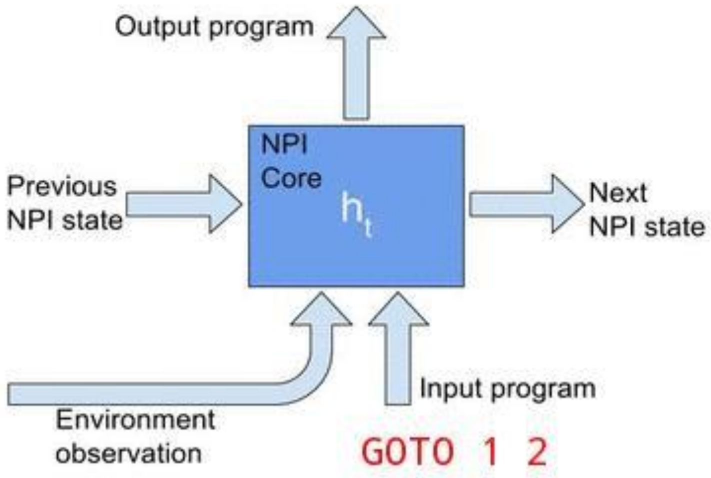
Canonicalizing 3D models



Canonicalizing demo

Car rendering

NPI inference



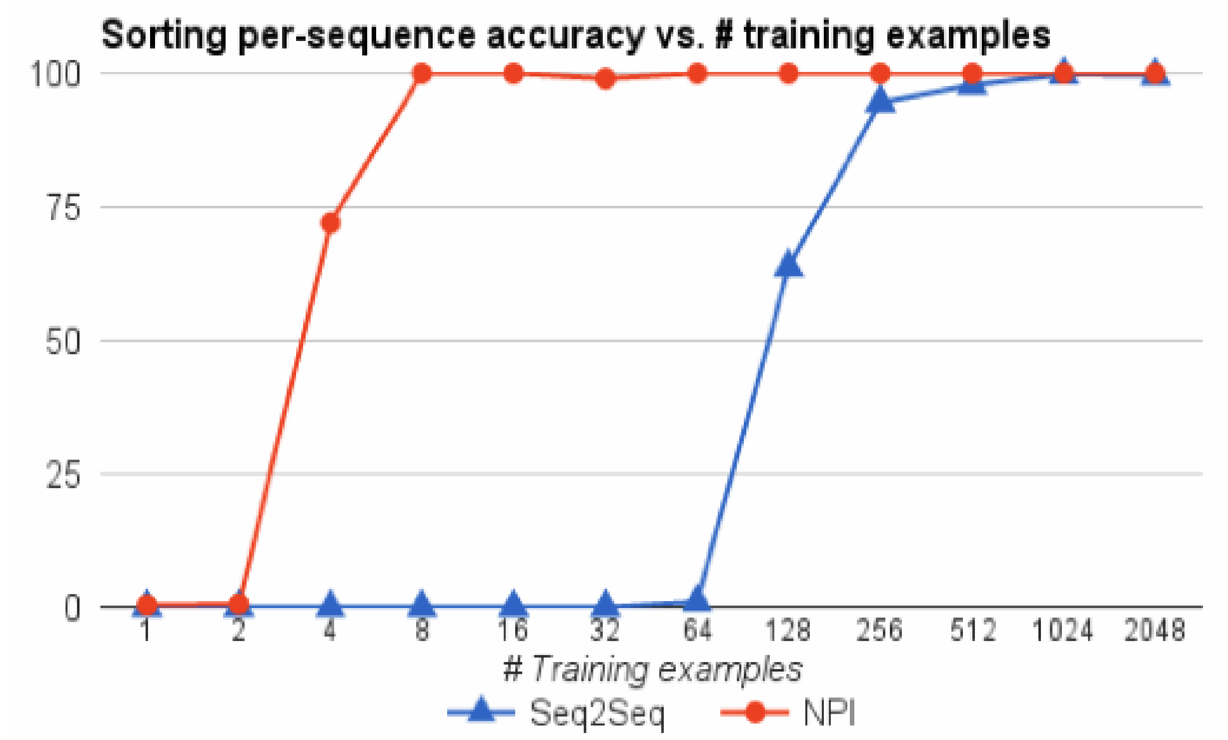
Generated commands

GOTO 1 2

Experiments

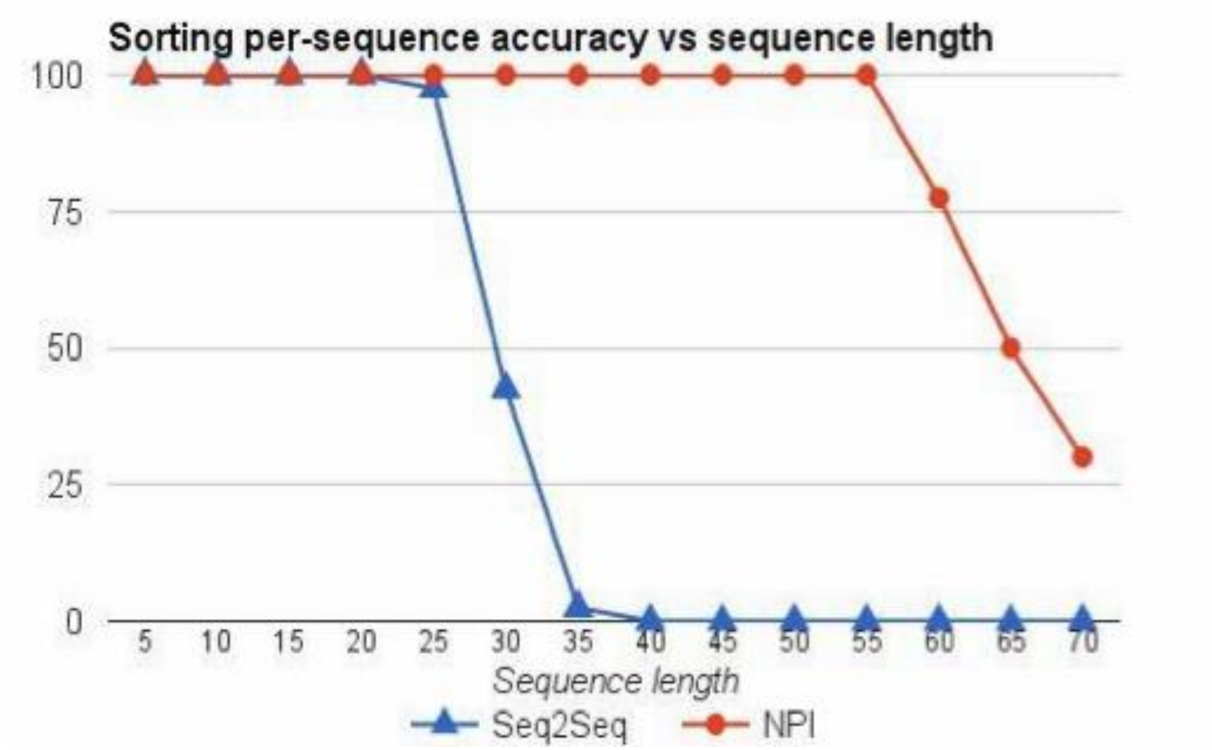
- Data Efficiency
- Generalization
- Learning new programs with a fixed NPI cores

Data Efficiency - Sorting



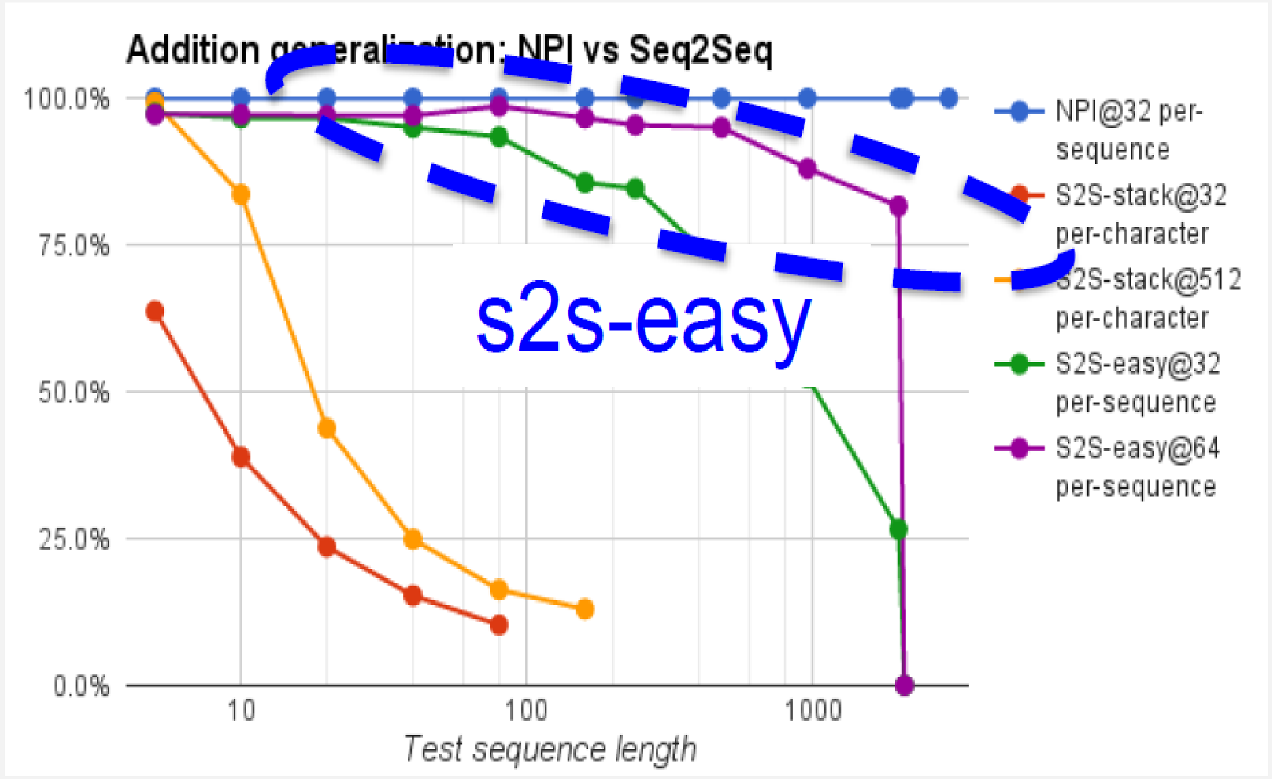
- Seq2Seq LSTM and NPI used the same number of layers and hidden units.
- Trained on length up to 20 arrays of single-digit numbers.
- NPI benefits from mining multiple subprogram examples per sorting instance, and additional parameters of the program memory.

Generalization - Sorting



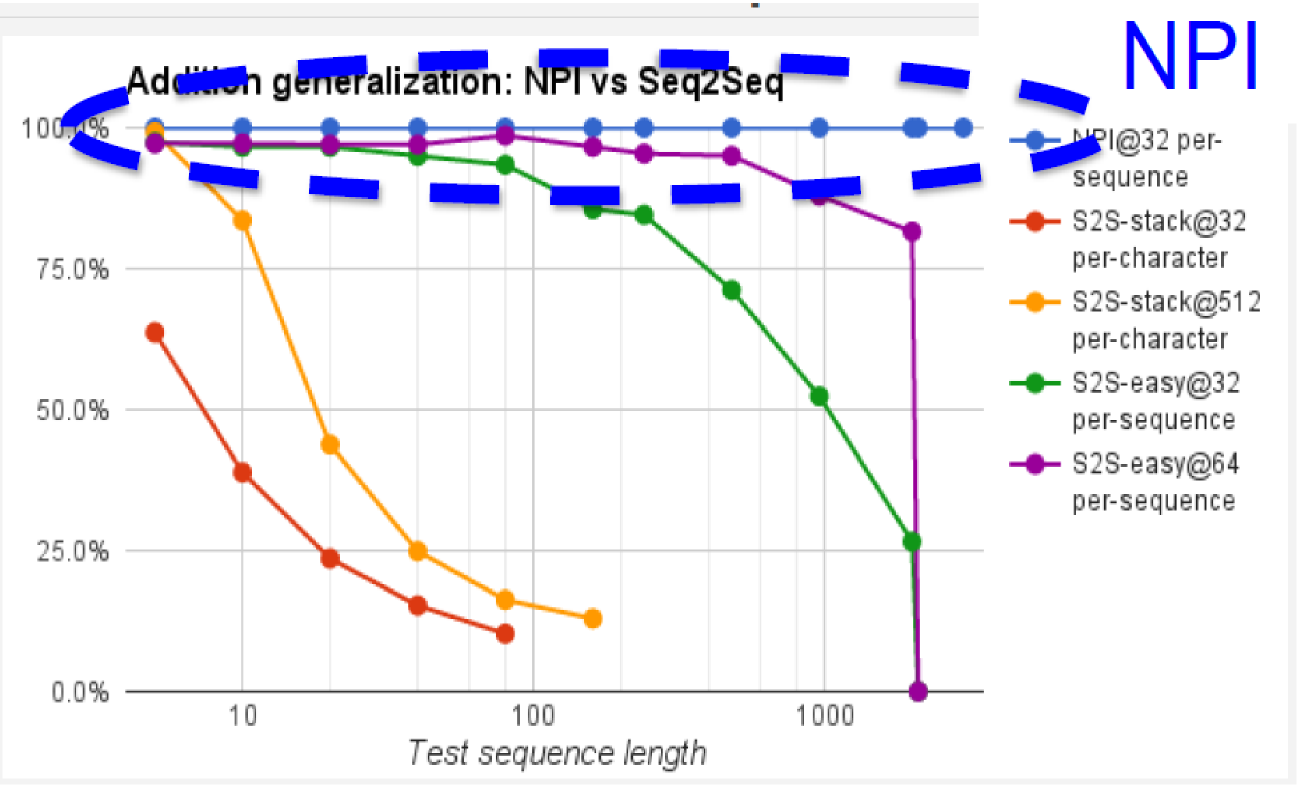
- For each length **up to 20**, we provided 64 example bubble sort traces, for a total of 1216 examples.
- Then, we evaluated whether the network can learn to sort arrays beyond length 20

Generalization - Adding



- NPI trained on **32 examples** for sequence **length up to 20**
- s2s-easy trained on **twice** as many examples as NPI (purple curve)
- s2s-stack trained on **16 times** more examples than NPI (orange curve)

Generalization - Adding

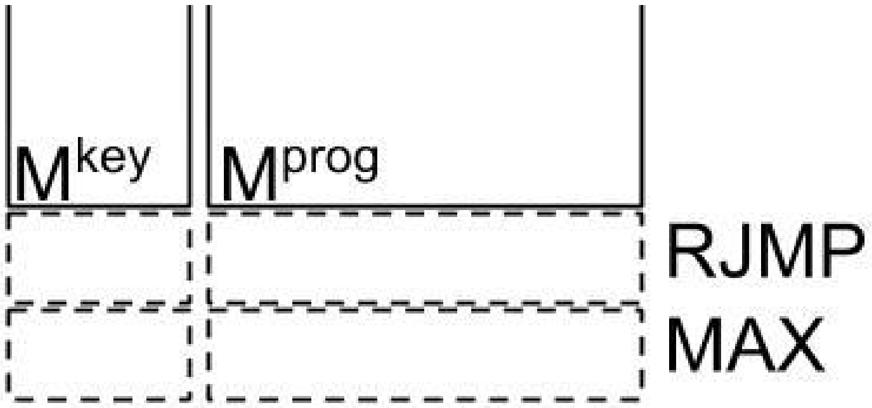


- NPI trained on **32 examples** for sequence **length up to 20**
- s2s-easy trained on **twice** as many examples as NPI (purple curve)
- s2s-stack trained on **16 times** more examples than NPI (orange curve)

Learning New Programs with a Fixed NPI Core

- Toy example: maximum-finding in an array
- Simple (not optimal) way: call BUBBLESORT and then take the right-most element of the array. Two new programs:
  - **RJMP**: Move all pointers to the rightmost position in the array by repeatedly calling RSHIFT program
  - **MAX**: Call BUBBLESORT and then RJMP
- Expand program memory by adding 2 slots. Then learn by backpropagation with the NPI core and all other parameters fixed.

Learning New Programs with a Fixed NPI Core



Only the memory slots of the new program are updated!
All other weights are fixed!

Protocol:

- Randomly initialize new program vectors in memory
- Freeze core and other program vectors
- Backpropagate gradients to new program vectors

Quantitative Results

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

- Numbers are per-sequence % accuracy
- + Max: indicates performance after addition of MAX program to memory
- “unseen” uses a test set with disjoint car models from the training set

Thanks!

Any questions and comments?

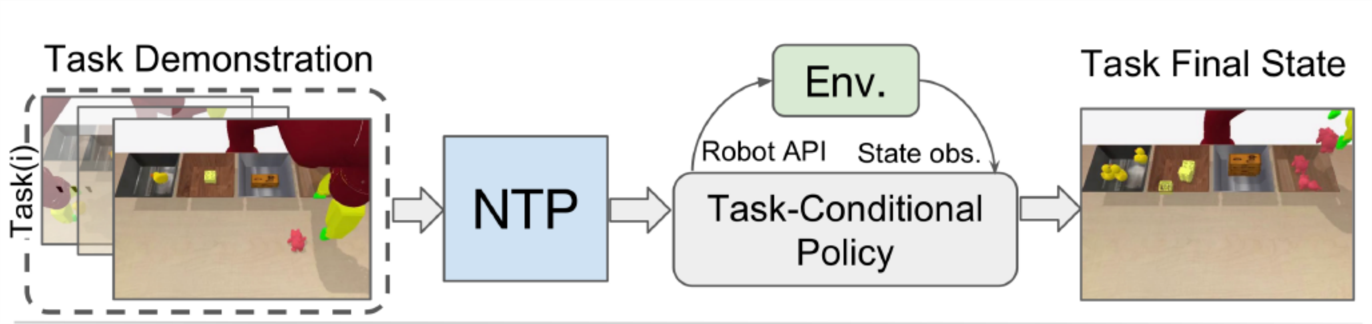
Neural Task Programming: Learning to Generalize Across Hierarchical Tasks

Danfei Xu, Suraj Nair, Yuke Zhu, Julian Gao, Animesh Garg, Li Fei-Fei, Silvio Savarese

Presented by Angran Li

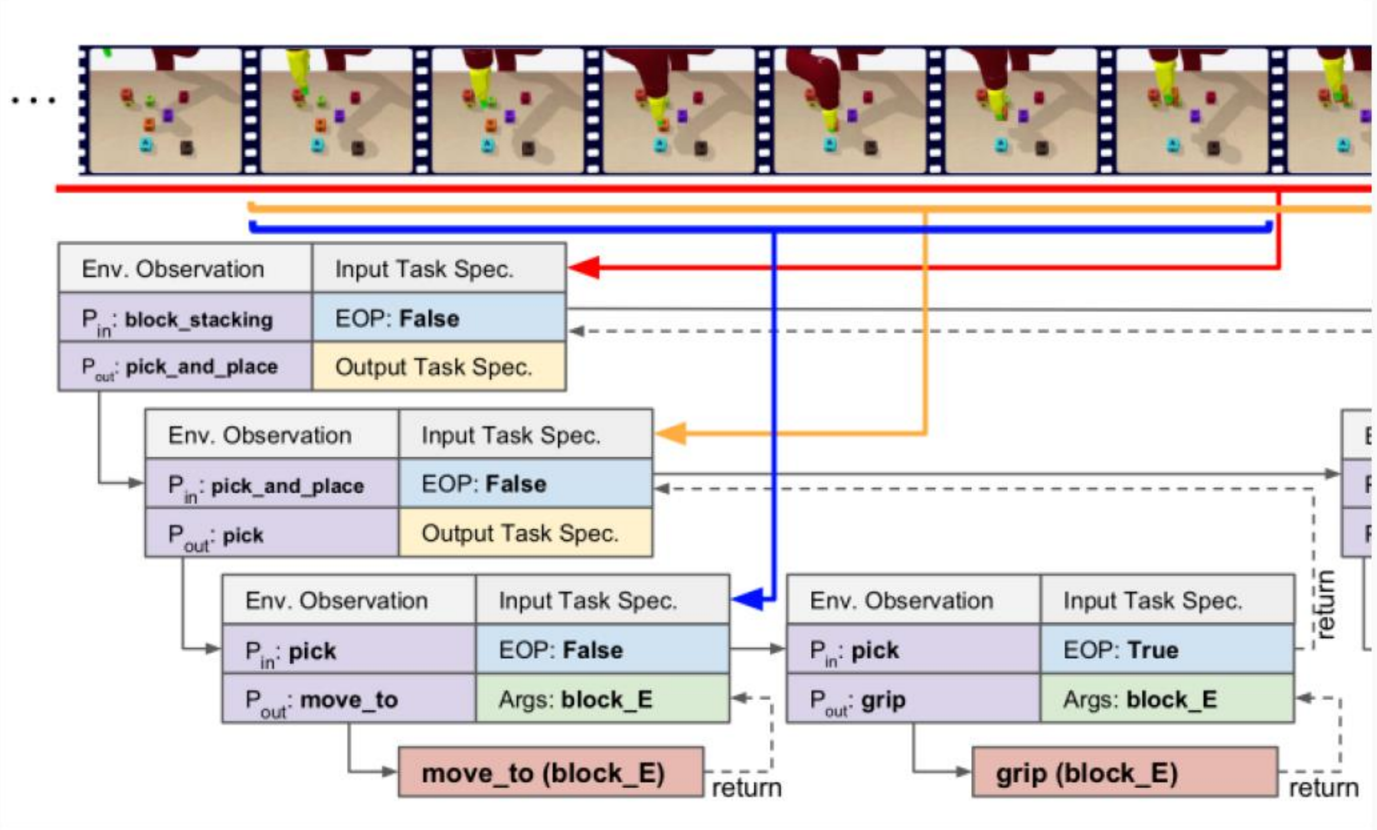
February 8, 2019

How the Algorithm works?



- Task Demonstration: state trajectory, first/third-person video demonstrations, or a list of language instructions.
- Task-Conditional Policy: a neural program.
- Using callable primitive actions to interact with the environment.

How the Algorithm works?



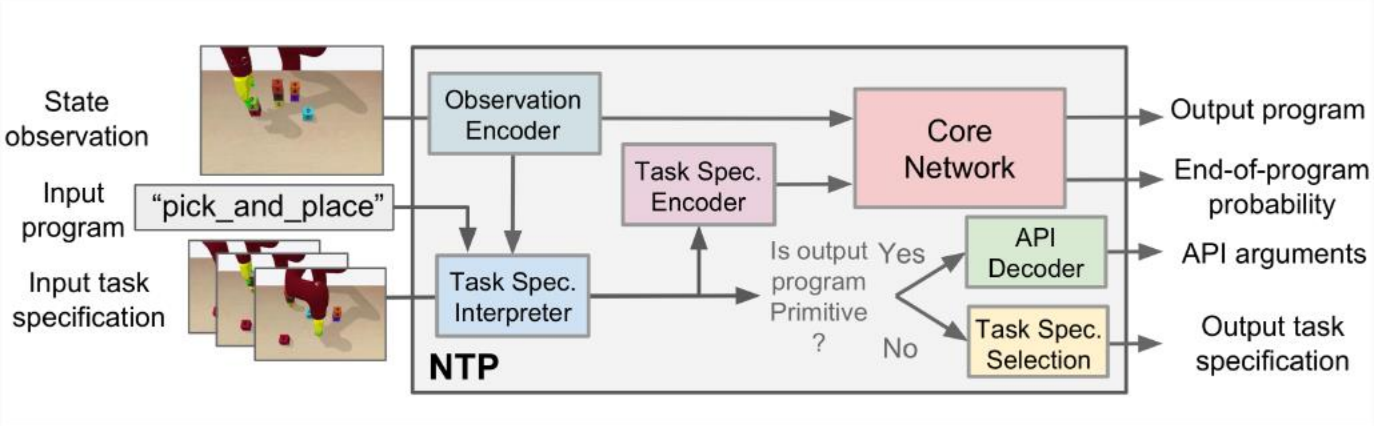
Top-level program `block_stacking` is recursively decomposed to bottom-level API `move_to` and `grip`.

Goals

Learning to Generalize Across Hierarchical Tasks

- Generalizing the learned polices to new task objectives
  - Task Length: more objects to transport.
  - Task Semantics: a different goal.
  - Task Topology: a different trajectory to the same goal.
- Hierarchical composition of primitive actions
  - Modularization and reusability.
  - Learn the latent structure in complex tasks, instead of fake dependencies.

Implementation: Neural Task Programming



- **Observation Encoder**: observation $o_i \Rightarrow$ state representation s_i
- **Task Spec. Interpreter**: $\Rightarrow$ API arguments a_i or task spec. ψ_{i+1}
- **Task Spec. Encoder**: task spec. $\psi_i \Rightarrow$ vector space ϕ_i
- **Core Network**: $s_i, p_i, \phi_i \Rightarrow p_{i+1}, r_i$

Implementation: Standing on the shoulder of NPI

Neural Task Programming combines the idea of **Few-Shot Learning from Demonstration** and **Neural Programmer-Interpreters**.

- Similarities when executing a program:
  - When the EOP probability exceeds a threshold α, control is returned to the caller program;
  - When the program is not primitive, a sub-program with its arguments is called;
  - When the program is primitive, a low-level basic action is performed.
- Two similar modules:
  - Domain-specific task encoders that map an observation to a state representation.
  - A key-value memory that stores and retrieves program embeddings.

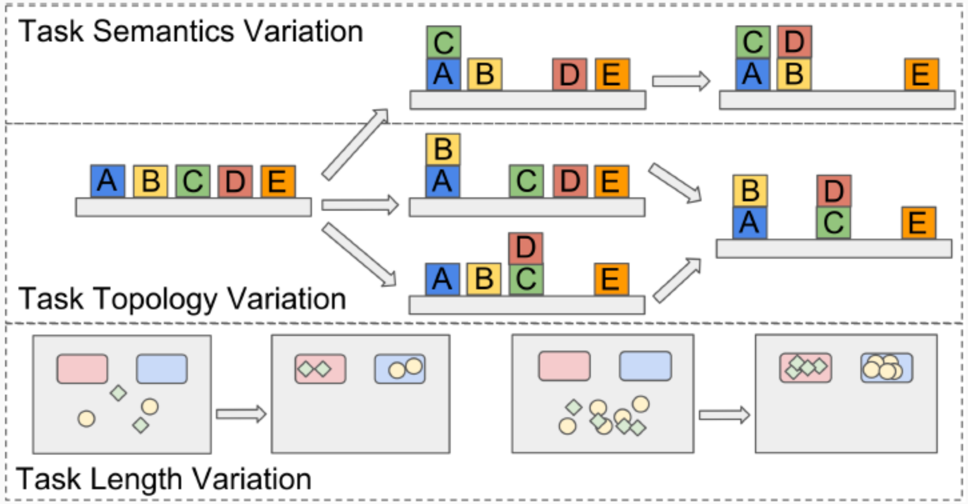
Implementation: NTP vs NPI

- **NPI:** one-shot generalization to tasks with longer lengths; can't generalizing to novel programs without training.
- **NTP:** generalizes to sub-task permutations (topology) and success conditions (semantics).
- Three main **diffenrences** of NTP than the original NPI:
  - NTP can interpret task specifications and perform hierarchical decomposition and thus can be considered as a **meta-policy**;
  - It uses robot **API**s as the primitive actions to scale up neural programs for complex tasks;
  - It uses a **reactive core network** instead of a **recurrent network**, making the model less history-dependent, enabling feedback control for recovery from failures.

Model Training

- The model is trained using rich supervision from program execution traces $\{\xi_t | \xi_t = (\Psi_t, p_t, s_t), t = 1 \ldots T\}$.
- The training objective is to maximize the probability of the correct executions over all the tasks in the dataset $D = \{(\xi_t, \xi_{t+1})\}$.
- For each task specification, the ground-truth hierarchical decomposition is provided by the expert policy, which is an agent with hard-coded rules.

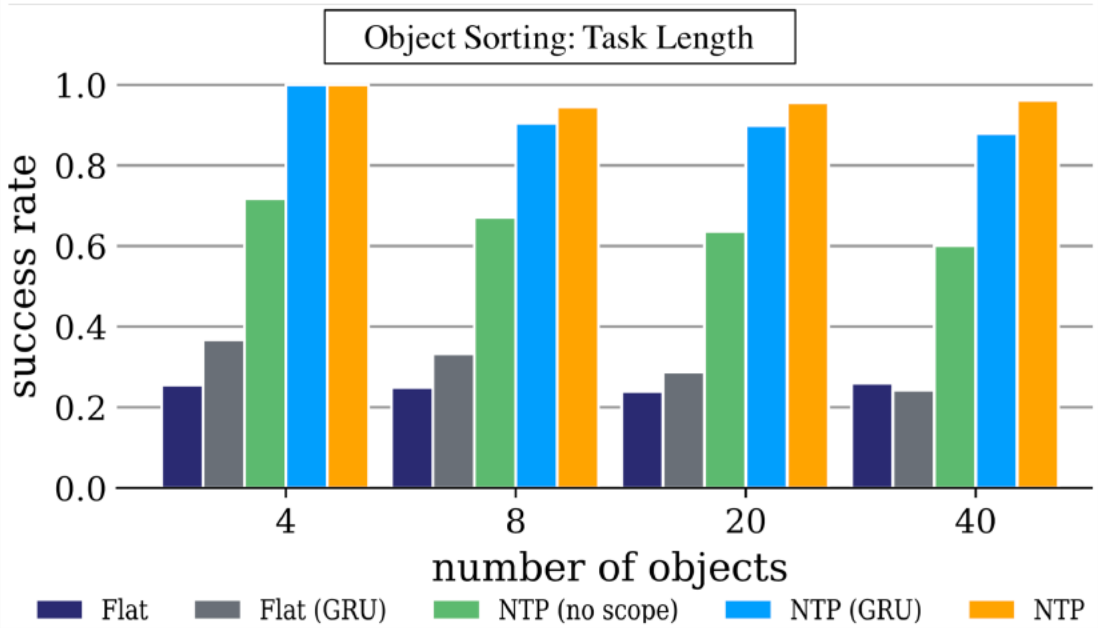
Experiments: Setup



- Generalization in 3 variations: semantics, topology, and length.
- Using image-based input without access to ground truth state.
- Working in real-world tasks combine these variations.

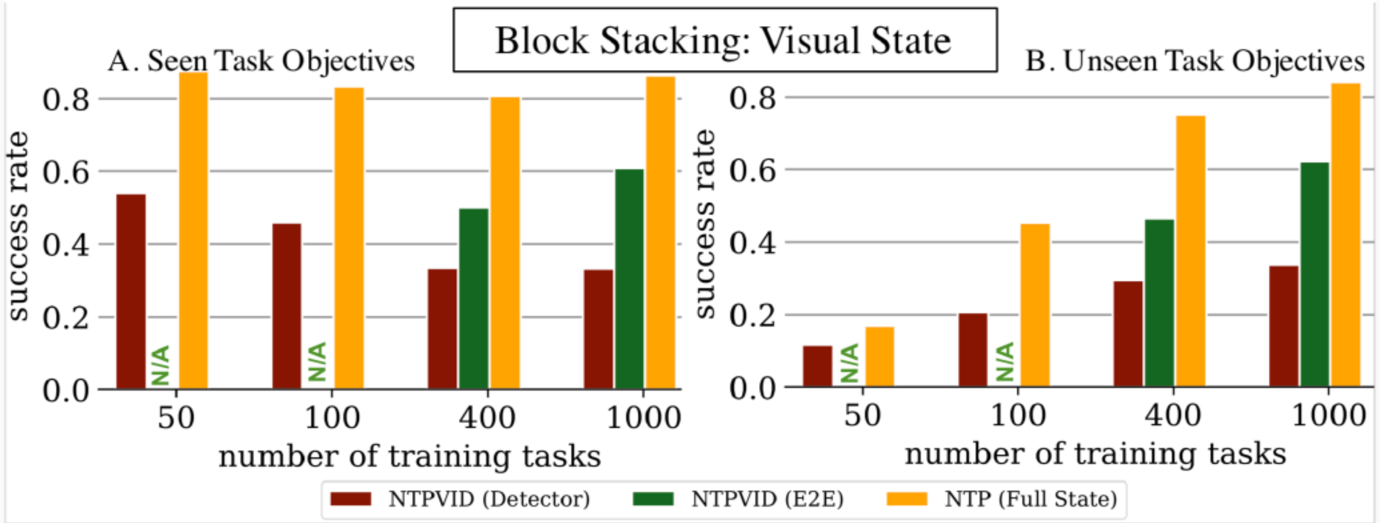
⇒ Three tasks: Object Sorting, Block Stacking, and Table Clean-up

Experiments: Object Sorting



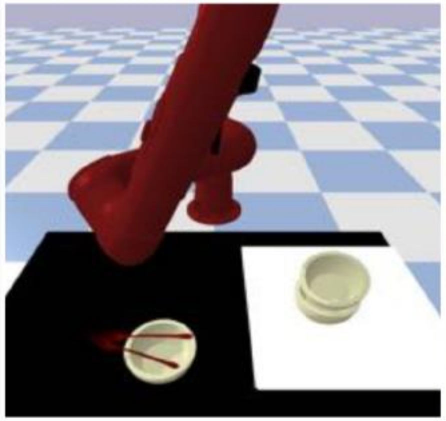
- **Flat**: non-hierarchical model, directly predicts the primitive APIs instead of calling hierarchical programs.
- **GRU**: Gated Recurrent Unit.

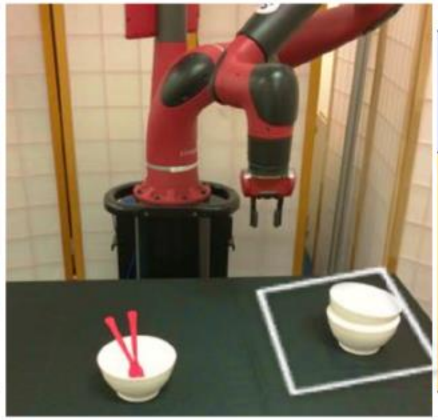
Experiments: Block Stacking



- **NTPVID(E2E)**: Trained with only visual information.
- **NTP(Full State)**: Trained with ground-truth hierarchical decomposition.

Experiments: Table Clean-up





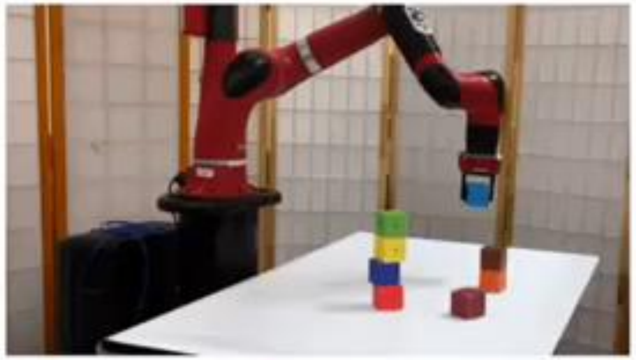
# Bowls, # Forks	Success
2 B, 1 F	1.00
2 B, 2 F	0.95
3 B, 2 F	0.75
3 B, 3 F	0.55

- Sort plastic bowls and forks into a stack, so they can be steadily carried away.
- Task variations:
    - Task length: number of bowls and forks varies.
    - Task topology: the ordering in which bowls are stacked varies.

Discussion & Future Work

- **Neural Task Programming:**
  - A meta-learning framework that learns modular and reusable neural programs for hierarchical tasks.
  - Generalizing well towards the variation of task length, semantics, and topology for complex tasks.
- **Future work:**
  - Improve the state encoder to extract more task-salient information such as object relationships;
  - Devise a richer set of APIs such as velocity and torque-based controllers;
  - Extend this framework to tackle more complex tasks on real robots.

Neural Task Programming: Learning to Generalize Across Hierarchical Tasks



Danfei Xu*, Suraj Nair*, Yuke Zhu, Julian Gao, Animesh Garg,
Li Fei-Fei, Silvio Savarese

Stanford Artificial Intelligence

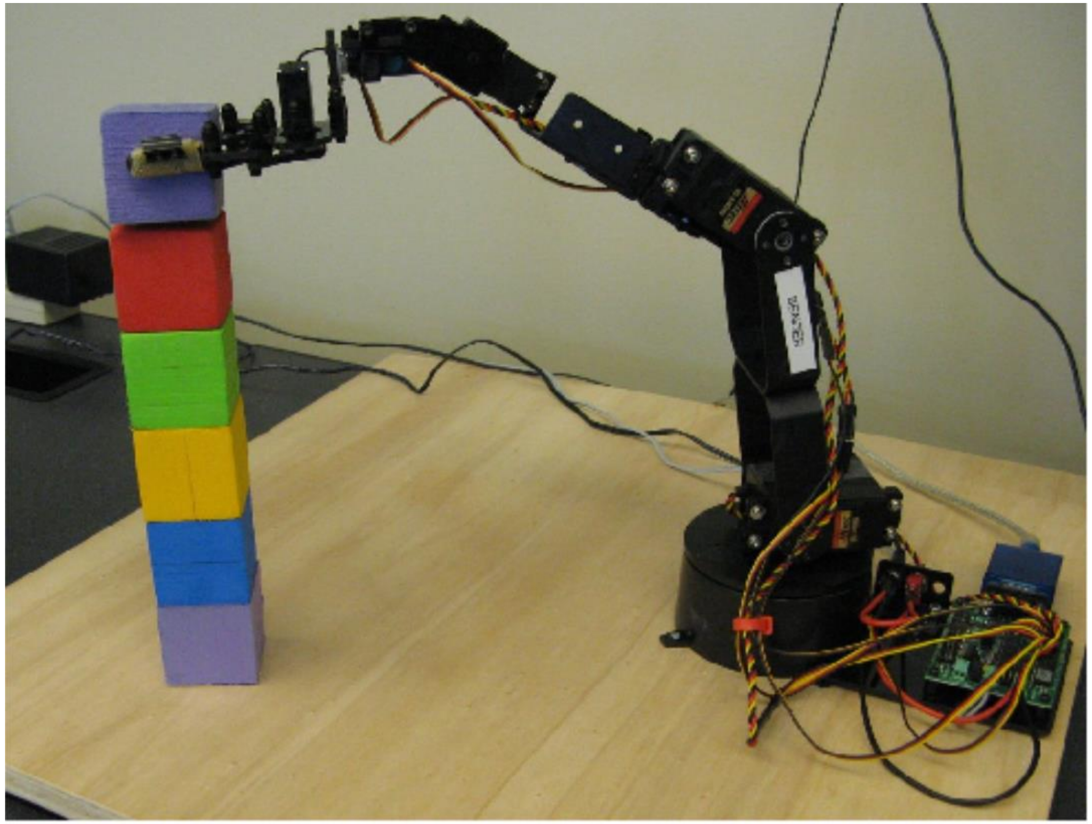
Questions?

TACO: Learning Task Decomposition via Temporal Alignment for Control

Kyriacos Shiarlis, Markus Wulfmeier, Sasha Salter, Shimon Whiteson, Ingmar Posner

Presented by: Zihang Fu

Motivation – Block Stacking Task



- Complex tasks can often be broken down into simpler sub-tasks
- Most Learning from Demonstration (LfD) algorithms can only learn a single policy for the whole task
- Resulting in more complex

Modular LfD

- Modelling the task as a composition of sub-tasks
- Reusable sub-policies (modules) are learned for each sub-task.
- Sub-policies are easier to learn and can be composed in different ways to execute new tasks.
- Enabling zero-shot imitation.

Key approach: provide the learner with additional information about the demonstration

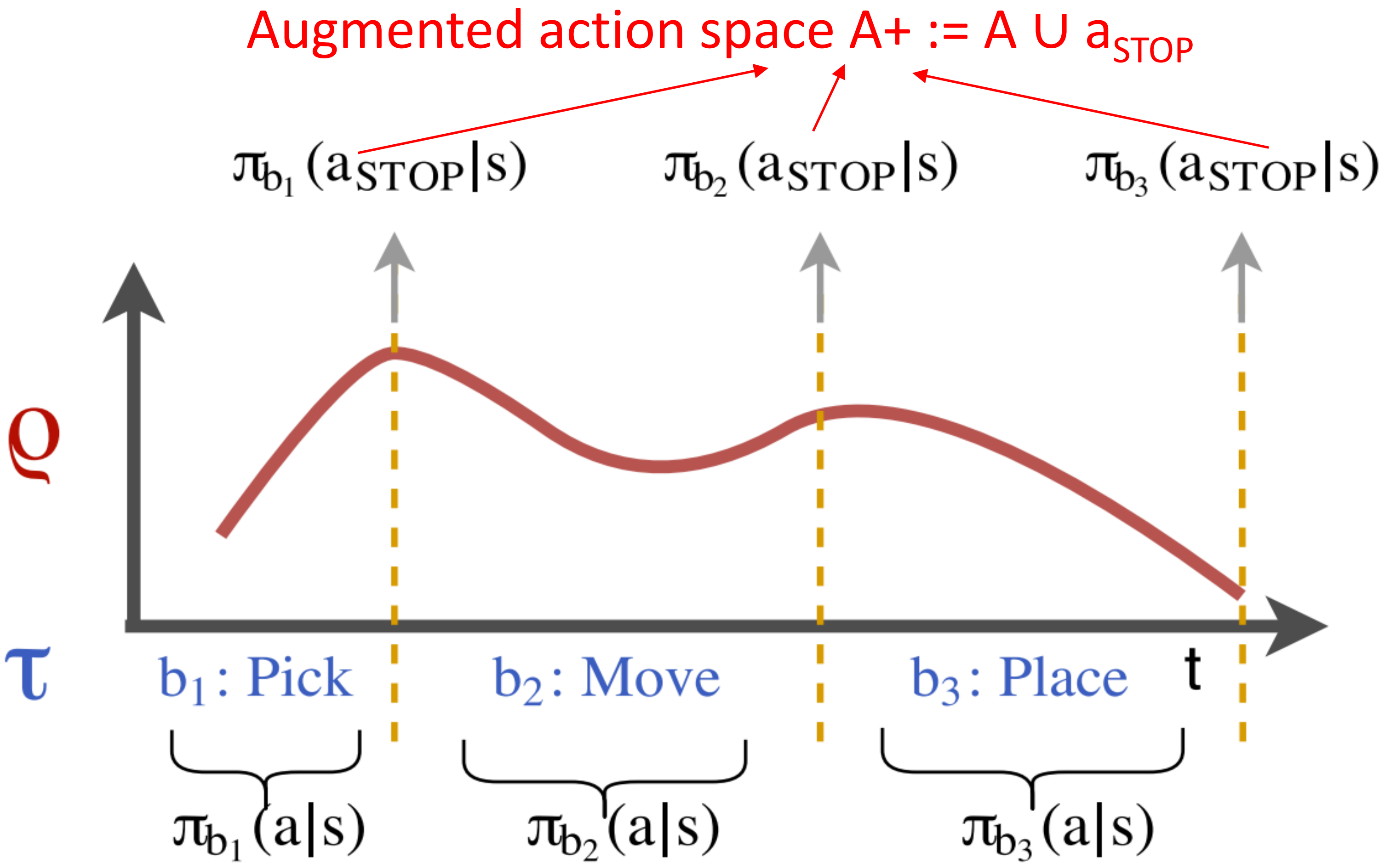
TACO: Temporal Alignment for Control

- Partly supervised
- Domain agnostic
- Demonstration is augmented by *task sketches - a sequence of sub-tasks that occur within the demonstration*

$$\boldsymbol{\tau} = (b1, b2, \ldots, bL),$$

- Simultaneously aligns the sketches with the observed demonstrations and learns the required sub-policies

Example: Block stacking task



Problem

How to align task-sketches with the demonstration?

Solution: Borrow temporal sequence alignment techniques from speech recognition!

TACO: Temporal Alignment for Control

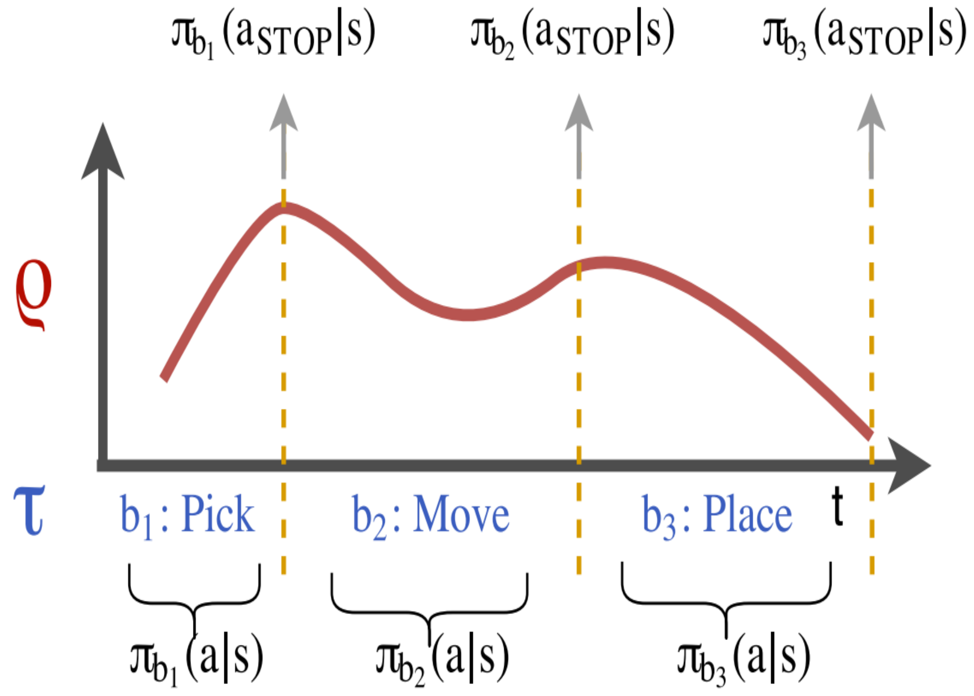
$\boldsymbol{\tau}$ = (b1, b2, . . . , bL),

Input sequence ρ with length T

A path $\zeta = (\zeta_1, \zeta_2, \ldots, \zeta_T)$ is a sequence of sub-tasks of the same length as the input sequence ρ, describing the active sub-task ζ_t at every time-step

$Z_{T,\boldsymbol{\tau}}$ is the set of all possible paths of length T for a task sketch $\boldsymbol{\tau}$

Eg. T = 6, $\boldsymbol{\tau}$ = (b1, b2, b3), ζ = (b1, b1, b2, b3, b3, b3)



TACO: Temporal Alignment for Control

Objective: Maximise the joint log likelihood of the task sequence and the actions conditioned on the states

$$p(\tau, \mathbf{a}_\rho | \mathbf{s}_\rho) = \sum_{\zeta \in \mathcal{Z}_{T,\tau}} p(\zeta | \mathbf{s}_\rho) \prod_{t=1}^{T} \pi_{\theta_{\zeta_t}}(a_t | s_t),$$

$p(\zeta/s_\rho)$ is the product of the stop, a_{STOP} , and nonstop, $\bar{a}_{STOP}$, probabilities associated with any given path.

Eg. T = 4, $s_\rho = (s_0, s_1, s_2, s_3)$, $\boldsymbol{\tau}$ = (b1, b2), ζ = (b1, b1, b2, b2)

$p(\zeta/s_\rho)$ = π_{b1}(non-stop) * π_{b1}(stop) * π_{b2}(non-stop) * π_{b2}(stop)

TACO: Temporal Alignment for Control

Problem: Impossible to compute all paths ζ in $Z_{T\ ,tau}$ for long sequence

Solution: Dynamic Programming

The (joint) likelihood of a being at sub-task l at time t can be formulated in terms of **forward variables**:

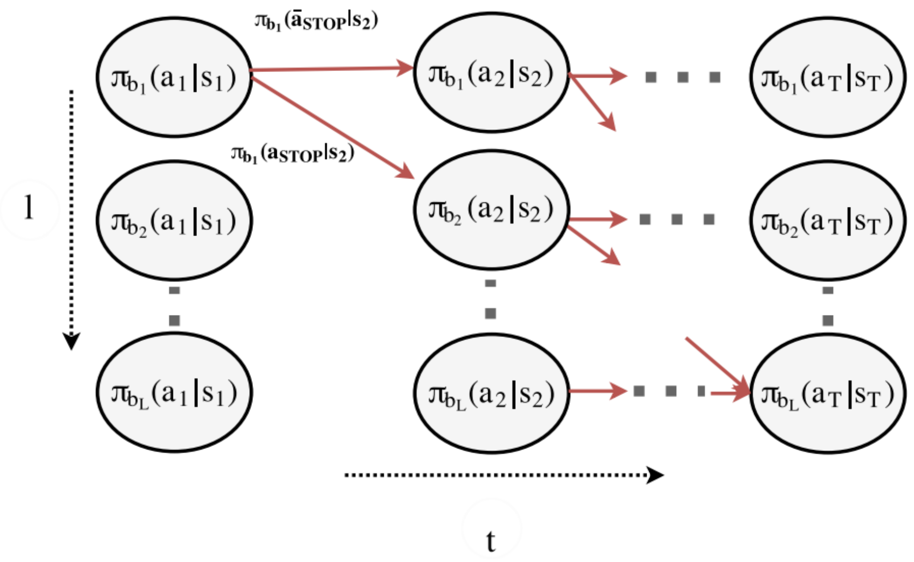
$$\alpha_t(l) := \sum_{\zeta_{1:t} \in \mathcal{Z}_{t,\tau_{1:l}}} p(\zeta|\mathbf{s}_\rho) \prod_{t'=1}^{t} \pi_{\theta_{\zeta_{t'}}}(a_{t'}|s_{t'}).$$

TACO: Temporal Alignment for Control

$$\alpha_1(l) = \begin{cases} \pi_{\theta b_1}(a_1|s_1), & \text{if } l = 1, \\ 0, & \text{otherwise.} \end{cases}$$

$$\alpha_t(l) = \pi_{\theta b_l}(a_t|s_t)\big[\alpha_{t-1}(l-1)\pi_{\theta b_{l-1}}(a_{STOP}|s_t) \\ +\alpha_{t-1}(l)(1-\pi_{\theta b_l}(a_{STOP}|s_t))\big].$$

$$\alpha_T(L) = p(\tau, \mathbf{a}_\rho|\mathbf{s}_\rho).$$



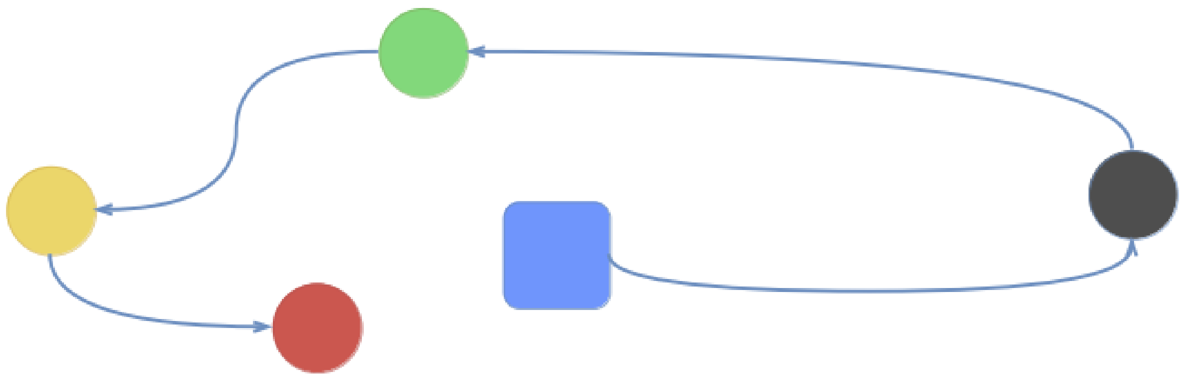
(b) TACO - Computation of forward variables $\alpha_t(l)$

Training: Maximize $\alpha_T(L)$ over θ

Experiments: Nav-World

Setup:

- The agent (blue) receives a route as a task sketch.
- $\boldsymbol{\tau}$ = (black, green, yellow, red)
- State space: (x, y) distance from each of the destination points
- Action space: (v_x, v_y) - the velocity



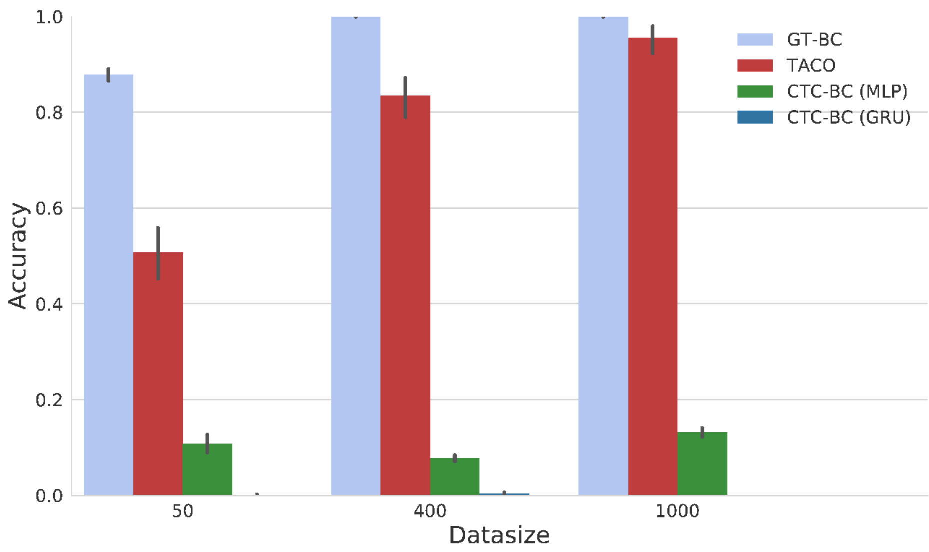
Training:

- Provided with state-action trajectories ρ and the task sketch.
- At the end of learning, the agent learns four sub-policies

Test:

- Given an unseen task sketch.
- Considered successful if the agent visits all destinations in the correct order

Experiments: Nav-World

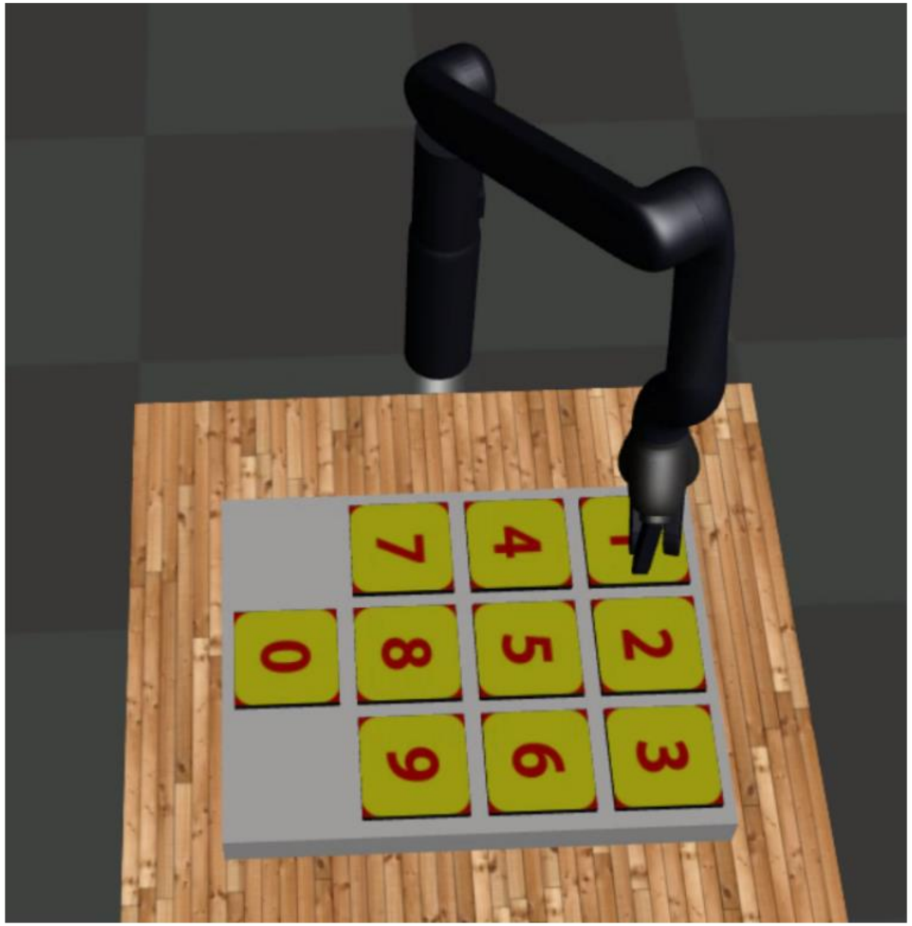


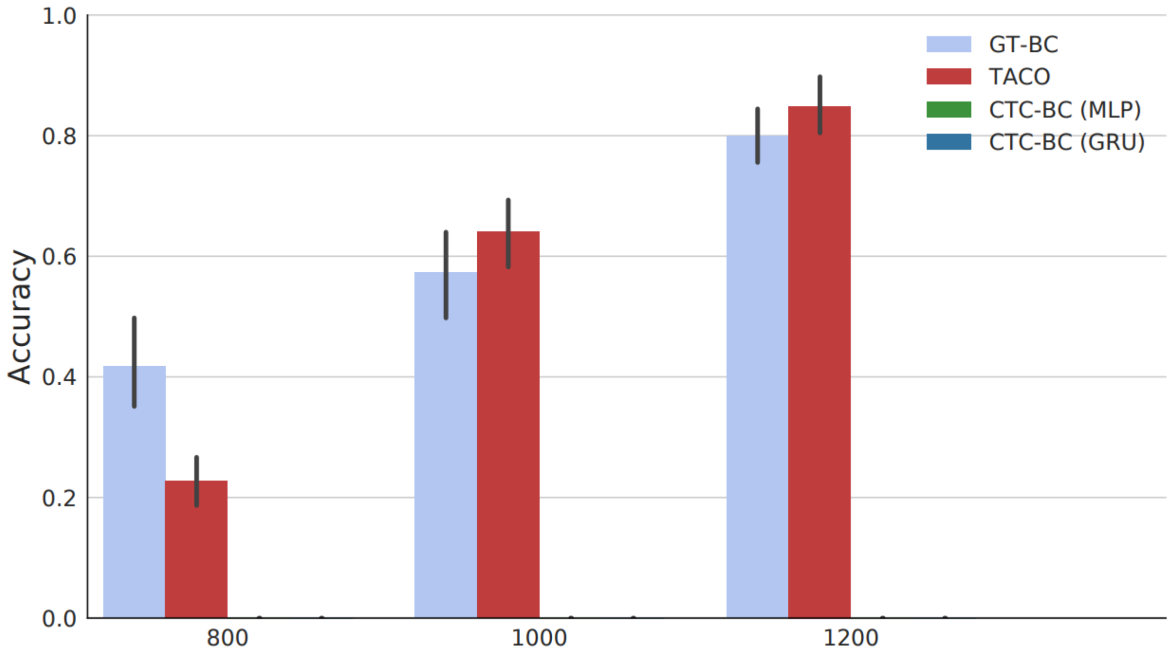
Success Rate

Algorithm	Nav-World
TACO	**95.3**
CTC-BC (MLP)	89.0
CTC-BC (GRU)	80.0
GT-BC (aligned w. TACO)	94.6

Alignment Accuracy

Experiments: Dial Domain





Summary: TACO - Temporal Alignment for Control

- Modular LfD
- Weak supervision - task sketch
- Optimising the sub-policies over a distribution of possible alignments

Future Work & Limitation

Limitation:

- Sub-tasks in the task sketch has to be placed in the correct order

Future work:

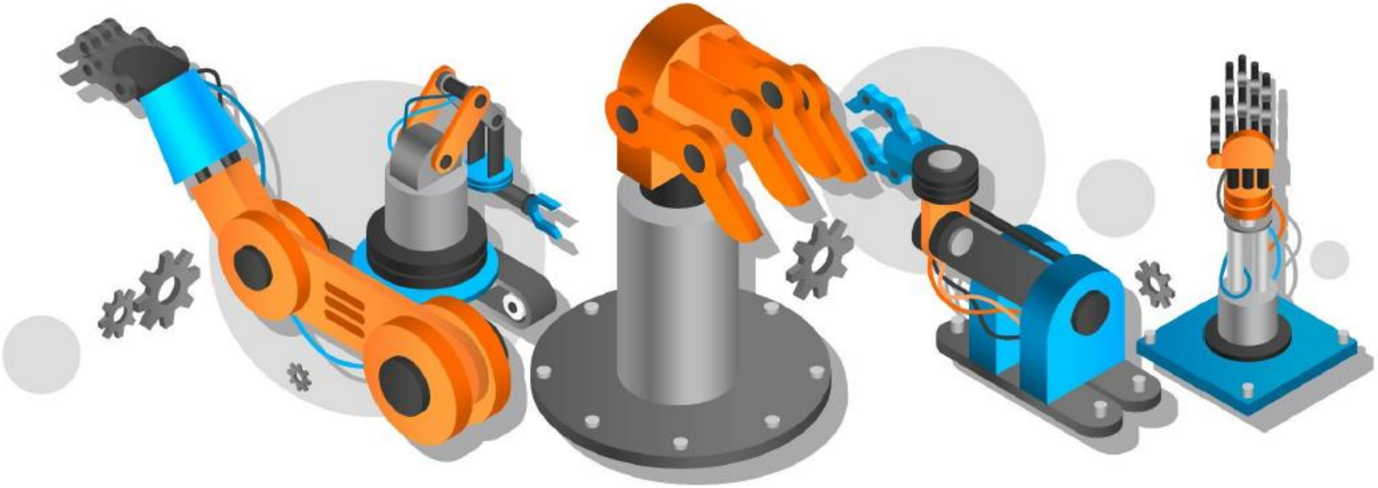
- Task sketches are dissimilar to natural human communication. Combination of TACO with architectures that can handle natural language.
- Hierarchical tasks decomposition.

Learning Movement Primitive Libraries through Probabilistic Segmentation

Rudolf Lioutikov, Gerhard Neumann, Guilherme Maeda, and Jan Peters

CSC2621: Imitation Learning for Robotics

February 8, 2019
Ilan Kogan



Agenda

- **Motivation for movement primitive libraries**
- Probabilistic segmentation (ProbS) algorithm derivation
- Experimental evaluation

If we can split complex tasks into movement primitives, we can generalize robot learning to completing new tasks

- Each **complex task** *(e.g., moving a box)* within a given domain *(e.g., manipulating boxes)* can be broken down into a possibly-duplicative set of **movement segments**
- Unique movement segments across the domain are termed **movement primitives** *(e.g., opening the claw)*

ADVANTAGES OF MOVEMENT PRIMITIVES

- The same primitives can be combined for **different tasks** within a domain
- The movement plan for a task can be adapted by **swapping** primitive order
- Transitions between primitives can be **optimized** for the entire domain at once

Task: Pick up at location A and deposit at location B

- Rotate 90° clockwise
- Open claw
- Lower arm
- Close claw
- Raise arm
- Rotate 90° counter-clockwise
- Lower arm
- Open claw
- Raise arm

movement segment   movement primitive

Efficiently acquiring these primitives without relying on human intervention is challenging

TRADITIONAL APPROACH

- Movement primitive acquisition is broken down into two problems: trajectory **segmentation** and **learning** of underlying primitive library
- Segmentation of observed trajectories
  - When did one movement primitive stop and the next begin?
  - Heuristics are commonly used *(e.g., when does the velocity of the arm become zero)*, but these are **task-dependent** and unclear when to apply one heuristic vs. another
  - Requires expert involvement to make time-invariant *(i.e., speed should not matter)*
- Learning of the underlying primitive library
  - Given the segments, how many primitives are present? What are they?

LIOUTIKOV ET AL.'S APPROACH

- Segment observed trajectories and learn the underlying library at once using an iterative **Expectation-Maximization (EM) algorithm**

Learn a **probabilistic representation of movement primitives** using the current segmentation

Improve segmentation by **down-weighting segmentations less likely** given current primitive library

Agenda

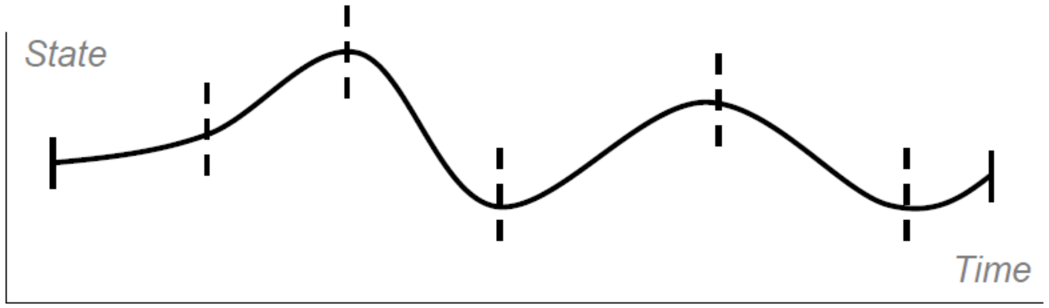
- Motivation for movement primitive libraries
- **Probabilistic segmentation (ProbS) algorithm derivation**
- Experimental evaluation

As the algorithm runs, false positive cuts are removed and the underlying library is learned from remaining segments

INPUT

An initial segmentation is selected based on an arbitrary heuristic that weakly **over-segments the task**

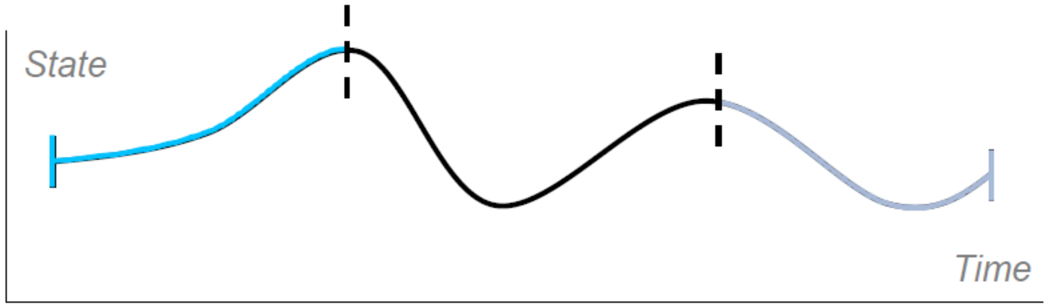
- Unfeasible to initialize with cut at every time step of the observation[1]



OUTPUT

The algorithm **eliminates false positive cuts** to determine the correct segmentation, and then **learns the underlying primitive library**

- Since cuts are only removed, the heuristic *must* weakly over-segment



1 No evidence for this claim is provided. Perhaps the authors are implicitly referring to the fact that a combinatorial explosion would render *their* EM approach futile, but one may be able to use other algorithms that would not face a similar fate *(e.g., by taking into account the non-independent nature of observations)*.

The Probabilistic Segmentation (ProbS) algorithm operates iteratively, converging to a locally-optimal solution

The initial set of **cutting points** from an arbitrary, over-segmenting heuristic

E-Step

1. Compute the probability α_s each segment is part of the **true segmentation** *(i.e., the probability the segment was generated by the primitives)*

M-Step

1. **Project each segment** into a lower-dimensional space using ridge regression, turning α_s into α_w
2. Implement the **Gaussian-means algorithm** to determine the number of primitives K and the initializing labels for each segment L
3. Run a **weighted-EM-GMM** with K clusters, initial labels L[1], and weights α_w to update mixture model

The underlying **mixture model** and the underlying **segmentation**

1 Since the Gaussian-means algorithm generates Gaussian distributions, the responsibility of each primitive for each possible segment can be calculated, or an overall average prior can be used as mixing weights. The authors do not specify exactly what “initial labelling L” means, but it is likely the latter.

Rather than working directly with segments, the authors use Probabilistic Movement Primitives (ProMPs)

- Probabilistic Movement Primitives (ProMPs) project trajectories (tasks) into a **lower dimensional weight space** with ridge regression
  - Working in a lower-dimensional space **makes clustering easier** by limiting the curse of dimensionality
  - Ridge regression provides a **distributional (probabilistic) interpretation** *(i.e., each observation of a primitive—a segment—is generated by a normal distribution),* providing a likelihood function to maximize
  - Basis functions can be chosen to provide **time invariance** *(i.e., normalizing by the length of the segment)*

$$\underbrace{w = (\Phi\Phi^T + \epsilon I)^{-1}\Phi s}$$

- Each segment *s* has a **matching projected segment *w***
  - The ridge regression penalty term drives the coordinates of *s* to zero, **inducing sparsity**
  - Still requires human intervention to choose a penalty term and basis functions![1]

1 The choice of basis function requires human intervention. While ProbS requires less human intervention than some other approaches discussed in the paper, substantial human intervention and parameter tuning is still required.

The distributional interpretation of ridge regression enables calculating the likelihood of a segment

- Without justification, each timestep from a segment is assumed to be **independently and identically distributed from a primitive distribution** *(but shouldn't they be highly correlated?)*
- The likelihood of a segment is therefore the **product of the probability density function values**

$$p(s|\theta_k) = \prod_{t=1}^{|s|} p(s_t|\theta_k)$$

$$= \prod_{t=1}^{|s|} \mathcal{N}(s_t|\Phi_t^T \mu_k, \Phi_t^T \Sigma_k \Phi_t)$$

- Without justification, each segment in a trajectory is assumed to be **independently selected** *(i.e., the likelihood of a trajectory is the product of the likelihood of each segment within it)*
- Since we do not know what the true segmentation is, we use the EM algorithm for optimization

The Expectation-Maximization algorithm can be used to segment trajectories

$$\alpha_w = \sum_{\mathcal{S} \in \mathcal{D}_s} p(\mathcal{S} | \tau, \Theta')$$

Main E-Step: Compute Segment Weighting[1]
Given the current model parameters, how likely is it that the identified projected segment belongs to the optimal segmentation *(i.e., sum over the probability of each segmentation S that includes segment s (D_s))?*

$$\begin{aligned}\Theta_{\mathcal{W}} &= argmax_{\Theta}\, Q_{\mathcal{W}}(\Theta, \Theta') \\ &= argmax_{\Theta} \sum_{w \in \mathcal{W}} \alpha_w \times log p(w|\Theta) \\ &= argmax_{\Theta} \sum_{w \in \mathcal{W}} \alpha_w \times log\left[\sum_{k=1}^{|\mathcal{M}|} \lambda_k\, p(w|\theta_k)\right]\end{aligned}$$

M-Step: Maximize Likelihood
Weighing by the probability of each previously-projected segment α_w, maximize the likelihood of seeing those segments across all segmentations and trajectories *W*

1 The authors reformulate the E-Step as a message-passing algorithm. Accordingly, they are able to reduce exponential complexity to quadratic complexity. This is done by, rather than calculating this term from scratch for each segment, storing information from preceding segments and leveraging the probabilistic relationship among them.

… but we have to apply it twice since we do not know the primitive that generated each segment

$$argmax_{\Theta} \sum_{w \in \mathcal{W}} \alpha_w \times \boxed{log[\sum_{k=1}^{|\mathcal{M}|} \lambda_k \, p(w|\theta_k)]}$$

Optimizing the sum inside of the log term is intractable; accordingly, we **use the EM algorithm again!**

$$Q_{\mathcal{W}}(\Theta, \Theta') = \sum_{k=1}^{|\mathcal{M}|} \sum_{w \in \mathcal{W}} \alpha_w \beta_{k,w} \, log \, \lambda_k p(w|\theta_k)$$

Sub E-Step: Compute Primitive Weighting
In the expectation step, we calculate the responsibilities: the probability $\beta_{k,w}$ each primitive k generated each projected segment w

Sub M-Step: Maximize Likelihood
In the maximization step, we optimize λ_k, μ_k, and Σ_k to maximize the weighted log likelihood of the observed data

Rather than assuming the number of components is known a priori, the Gaussian-means algorithm is used

- The **Gaussian-means algorithm** is a bisecting *k*-means algorithm *(i.e., a combination of hypothesis testing, k-means, and hierarchical clustering):*

  1. Start with one cluster
  2. Run *k*-means with k = 2
  3. Using the Anderson-Darling goodness-of-fit test, test the null hypothesis that the **observations within each created cluster are normally distributed**
  4. For all clusters that the null hypothesis is rejected, repeat steps two to four; else break

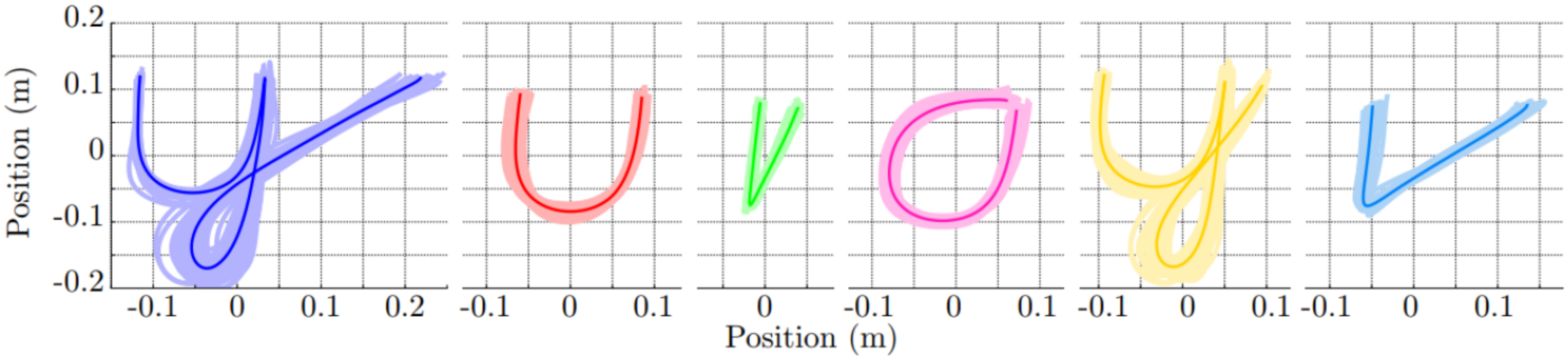
- Effectively, the **clusters continue to be divided** until we cannot reject the null hypothesis—as a result, the number of primitives is not assumed a priori
  - "A goodness-of-fit test is a measure of how much data you have." *– PJ Diggle*
  - Limitation: This implies **more observations will lead to more clusters!**

Agenda

- Motivation for movement primitive libraries
- Probabilistic segmentation (ProbS) algorithm derivation
- **Experimental evaluation**

ProbS excelled at writing letters in different orders …

- Using kinesthetic teaching, **27 combinations of the three letters “y”, “a”, and “u”** were shown to the robot and then velocities were processed using ProbS, EM-GMM, and BP-AR-HMM[1]
- ProbS performed better than the other two algorithms:
  - **More compact compression:** ProbS was able to represent the trajectories with fewer bits
  - **Higher-quality primitives:** The average log-likelihood of segments in a held-out trajectory (leave-one-out-cross-validation) was highest for ProbS

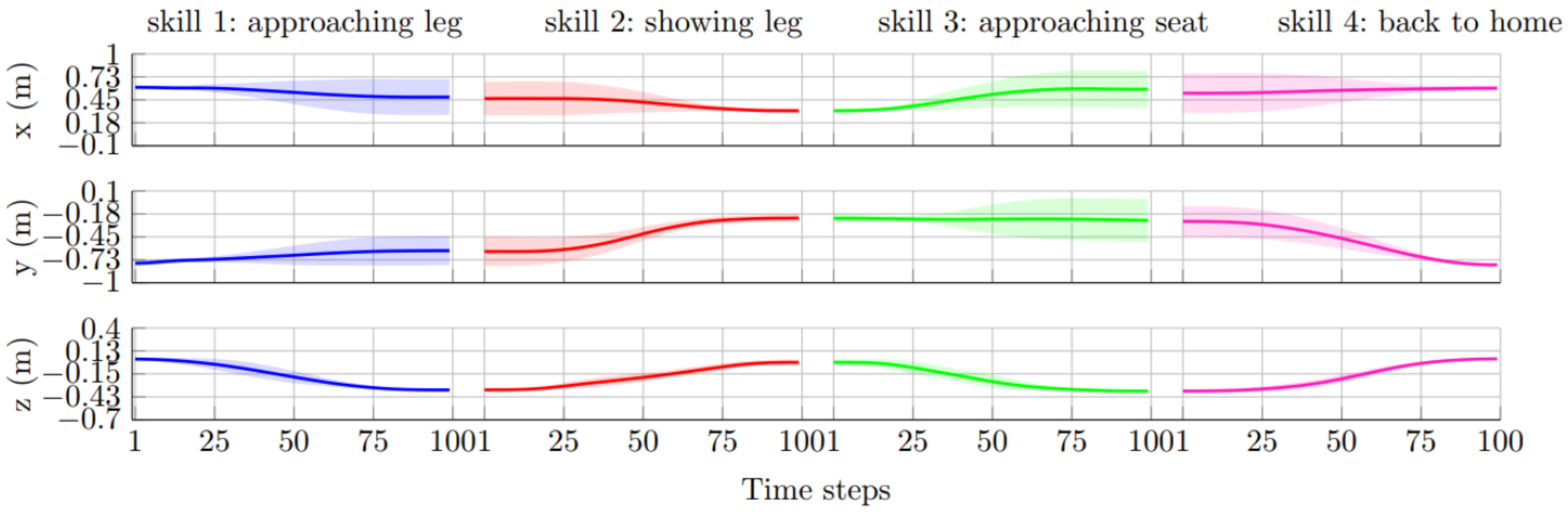


ProbS determined **six movement primitives** were present in the data set

1 The authors compared ProbS to a standard Expectation-Maximization Gaussian Mixture Model (EM-GMM) *(i.e., equivalent to ProbS but with a specified segmentation based on a heuristic)* and the state-of-the-art Beta Process Autoregressive Hidden Markov Model (BP-AR-HMM) *(Bayesian approach that uses an Indian buffet process prior)*

… putting together a chair differently than shown …

- The robot was shown how to put together a chair six times; **ProbS determined four movement primitives were present**



- These movement primitives were later combined to **assemble a chair in a new way**

… and identifying characteristic table tennis swings

- Multiple kinesthetic teaching demonstrations of table tennis swings were **segmented into a total of four movement primitives by ProbS:** forehand swing, backhand swing, and two waiting primitives

- The two waiting primitives resulted in a robot that barely moved, but **moved in opposite ways**

Notably, the authors do not demonstrate learning libraries **across tasks within a domain** and do not make clear how they **concatenated movement primitives** to perform new trajectories

Thank you!