Program-Guided Framework for Interpreting and Acquiring Complex Skills with Learning Robots

Shao-Hua Sun
Ph.D. candidate in Computer Science at the University of Southern California (USC)
Robotics

- Planning & Control
- Sensing & Perception
- Mobile Robot
- Robot Manipulation
- Human-robot Interaction
- Robot Locomotion
- Robot Manufacturing
- Robot Learning
- Modeling, Simulation & Diagnosis
- Adaptive Control
- Software Engineering
- Sensors
Robot Learning

Environment
- Structured
- Unstructured

Object
- Known
- Unseen

Task
- Pre-defined / Pre-programmed
- Diverse and Novel
Supervised Learning
Supervised Learning
Image Classification

Image

“Cat”
Machine Translation

English sentence: France is never cold in September

French sentence: la france est jamais froid en septembre
Automatic Speech Recognition

This is a supervised learning method.
Supervised Learning
Supervised Learning
Supervised Learning
Supervised Learning
Goal: maximize\[ \sum_{t=0}^{t=H} \gamma^t R_t(s_t, a_t) \]
Robot Learning via Reinforcement Learning

Goal: maximize $\sum_{t=0}^{t=H} \gamma^t R_t(s_t, a_t)$
Robot Learning via Reinforcement Learning

Goal: maximize $\sum_{t=0}^{t=H} \gamma^t R_t(s_t, a_t)$
Robot Learning via Reinforcement Learning

Agent / Policy

Environment

Goal: maximize \( \sum_{t=0}^{t=H} \gamma^t R_t(s_t, a_t) \)

Environment:
- \( R_t \): Reward at time step \( t \)
- \( S_{t+1} \): State at time step \( t+1 \)
- \( A_t \): Action at time step \( t \)

Agent / Policy:
- \( \pi(a_t|s_t) \): Policy at time step \( t \)

Robot Learning via Reinforcement Learning

Reward

-10

Action
Robot Learning via Reinforcement Learning

Goal: maximize \[ \sum_{t=0}^{t=H} \gamma^{t} R_t(s_t, a_t) \]
Robot Learning via Reinforcement Learning

Agent / Policy

Environment

Goal: maximize \( \sum_{t=0}^{t=H} \gamma^t R_t(s_t, a_t) \)
Robot Learning via Reinforcement Learning

Goal: maximize \( \sum_{t=0}^{t=H} \gamma^t R_t(s_t, a_t) \)
Robot Learning via Reinforcement Learning

Agent / Policy

Environment

Goal: maximize $\sum_{t=0}^{t=H} \gamma^t R_t(s_t, a_t)$
Robot Learning via Deep Reinforcement Learning

Goal: maximize \( \sum_{t=0}^{t=H} \gamma^t R_t(s_t, a_t) \)
Robot Learning via Deep Reinforcement Learning

Uninterpretable
Not generalizable
Limited to short-horizon tasks
No skill-reuse
Program-Guided Framework for Interpreting and Acquiring Complex Skills

Interpretable

Programmatic / Generalizable

Hierarchical

Modular
Stir-fry the onions until tender, and repeat this for garlic and carrots, if you have soy sauce, add some. Pour 2/3 cups the whisked eggs into the stir-fried and scramble.
Stir-fry the onions until tender, and repeat this for garlic and carrots, if you have soy sauce, add some. Pour 2/3 cups the whisked eggs into the stir-fried and scramble.
Neural Program Synthesis from Diverse Demonstration Videos

ICML 2018

Skill Specification → Program → High-level Plan → Low-level Execution

Program

```python
DEF run()
    while True:
        if isCurb():
            jump
        elseif isCeiling():
            crawl()
        else:
            walk()
```

Hyeonwoo Noh
Sriram Somasundaram
Joseph J. Lim
Imitation Learning

Demonstrations

Imitate

Neural Network Policy

Execution
Imitation Learning by Synthesizing Programs

Demonstrations → Synthesize → Program Policy → Execute

DEF run()
  if isFontClear():
    move
  else:
    turnLeft
    move
    turnLeft
    repeat(2):
      turnRight
      putMarker
Model Overview

Extract unique behaviors  Summarize  Decode

DEF run()
if isPointClear():
    move
else:
    turnLeft
    move
turnLeft
repeat(2):
    turnRight
    putMarker
Model Overview

DEF run()
if isPointClear():
    move
else:
    turnLeft
    move
    turnLeft
    repeat(2):
    turnRight
    putMarker

Program

Extract unique behaviors  Summarize  Decode
Model Overview

Extract unique behaviors  Summarize  Decode

Program

DEF run()
if isFontClear():
    move
else:
    turnLeft
    move
    turnLeft
    repeat(2):
    turnRight
    putMarker
Environments

Karel

```
DEF run()
    if isFontClear():
        move
    else:
        turnLeft
        move
        turnLeft
    repeat(2):
        turnRight
        putMarker
```

ViZDoom

```
DEF run()
    while isFontClear(HellKnight):
        attack
        moveForward
    if isThere(Demon):
        moveRight
    else:
        moveLeft
        moveBackward
```
Quantitative Results

Neural Network Policy

Demos → Execution

Program Policy

Demos → Program → Execution

### Neural Network Policy

<table>
<thead>
<tr>
<th>Karel</th>
<th>ViZDoom</th>
</tr>
</thead>
<tbody>
<tr>
<td>62.8%</td>
<td>78.4%</td>
</tr>
</tbody>
</table>

### Program Policy

DEF run()
    if isFontClear():
        move
else:
    turnLeft
        move
turnLeft
    repeat(2):
        turnRight
        putMarker
Takeaway

- Synthesize programs to imitate demonstrations

Demonstrations → Synthesize → Program Policy

DEF run()
  if isFontClear():
    move
ellse:
    turnLeft
    move
    turnLeft
    repeat(2):
    turnRight
    putMarker

→ Execute → Execution
Learning to Synthesize Programs as Interpretable and Generalizable Policies

NeurIPS 2021

Skill Specification → Program → High-level Plan → Low-level Execution

Program

```
DEF run()
    while True:
        if isCurb():
            jump
        elseif isCeiling():
            crawl()
        else:
            walk()
```

Reward

Dweep Trivedi  Jesse Zhang  Joseph J. Lim
Reinforcement Learning
Reinforcement Learning by Synthesizing Programs

```
DEF run()
    WHILE noMarkersPresent()
        IFE LELSE rightIsClear()
            turnRight
        ELSE
            WHILE frontIsClear()
                turnLeft
                move
    
```
Reinforcement Learning by Synthesizing Programs

- Grammar
- Environment Dynamics
- Desired Behavior

Model

Program Policy

```
DEF run()
    WHILE noMarkersPresent()
        IFELSE rightIsClear()
            turnRight
        ELSE
            WHILE frontIsClear()
                turnLeft
                move
```

Synthesize

Execute

Reward

Environment
LEAPS: Learning Embeddings for Latent Program Synthesis

**Stage 1**
Learn a program embedding space from randomly generated programs

- **Grammar**
- **Environment Dynamics**

**Stage 2**
Search for a task-solving program

- **Desired Behavior**
Stage 1: Learning a Program Embedding Space

Learnable mapping
Training Objective
Latent Program

Program $\rho$

Latent Program $z$

Reconstructed Program $\hat{\rho}$

Execute

Environment

$a_1, a_2, \ldots, a_t$

$\mathcal{L}^L$

$\mathcal{L}^R$

$\mathcal{L}^P$

$q_\phi$

$\pi(a|s,z)$

$a_1, a_2, \ldots, a_t$
Stage 2: Latent Program Search with Cross Entropy Method
Cross Entropy Method Trajectory Visualization

- Dataset Program
- GT Program
- CEM Population
- CEM Next Center

Iteration: 1
Avg. Reward: 0.1000
Next Candidate Program Reward: 0.1000
Karel Tasks

- StairClimber
- Maze
- FourCorners
- Harvester
- CleanHouse
- TopOff
Qualitative Results

StairClimber

DRL
LEAPS

FourCorners

DRL
LEAPS

Maze

DRL
LEAPS

TopOff

DRL
LEAPS
Quantitative Results

![Bar chart showing quantitative results for different tasks and algorithms. The x-axis represents tasks: Maze, StairClimber, TopOff, FourCorner, Harvester, and CleanHouse. The y-axis represents reward ranging from 0 to 1.2. The chart compares results for DRL, VIPER, Naive, and LEAPS algorithms.]
Zero-shot Generalization

Learning on 8x8 grids

StairClimber

```
DEF run()
  WHILE noMarkersPresent()
    turnRight
    move
    WHILE rightIsClear()
      turnLeft
```

Evaluation on 100x100 grids

Maze

```
DEF run()
  IF frontIsClear()
    turnLeft
    WHILE noMarkersPresent()
      turnRight
      move
```

Zero-shot Generalization

![Graph showing reward for Maze and StairClimber tasks for DRL, VIPER, and LEAPS.](image)
Zero-shot Generalization

The diagram shows the reward for different algorithms on two different environments: Maze and StairClimber. The algorithms compared are DRL, VIPER, and LEAPS. The x-axis represents the environments, and the y-axis represents the reward. The chart visualizes how well each algorithm performs in each environment, with DRL generally outperforming VIPER and LEAPS in the Maze environment.
Interpretability

Human Debugging Interface

Improved Performance

- **TopOff**
  - Original: 0.803200845304315
  - 3 Edits: 0.880000845304315
  - 5 Edits: 0.950000845304315

- **FourCorner**
  - Original: 0.600000845304315
  - 3 Edits: 0.680000845304315
  - 5 Edits: 0.750000845304315

- **Harvester**
  - Original: 0.400000845304315
  - 3 Edits: 0.480000845304315
  - 5 Edits: 0.550000845304315
Takeaway

- Synthesize **generalizable and interpretable programs** from rewards
Jump

Walk

Crawl

Video demonstration

Trajectory demonstration

Language instruction

Reward

Stir-fry the onions until tender, and repeat this for garlic and carrots. If you have soy sauce, add some. Pour 2/3 cups the whisked eggs into the stir-fried and scramble.

Jump

Crawl

Walk

Program

Observation

Low-level execution

DEF run()
while True:
if isCurb():
    jump
elseif isCeiling():
crawl()
else:
    walk()
Program Inference

Imitation learning from demonstrations

Reinforcement learning from rewards

Synthesize

Program

Execution

Demonstrations

execute

Program Synthesizer

LEAPS

Environment

Reward

Synthesize

Program Policy

Execute
Stir-fry the onions until tender, and repeat this for garlic and carrots, if you have soy sauce, add some. Pour 2/3 cups the whisked eggs into the stir-fried and scramble.
Program Guided Agent

ICLR 2020 (Spotlight)
Problem Formulation

Program (task)

```
def run():
    if is_there[River]:
        mine(Wood)
        build_bridge()
    if agent[Iron]<3:
        mine(Iron)
        place(Iron, 1, 1)
    else:
        goto(4, 2)
    while env[Gold]>0:
        mine(Gold)
```

Observation

```
x3  x1  x0
```

Plan (subtasks)

```
mine(Wood)
build_bridge()
mine(Iron)
place(Iron, 1, 1)
mine(Gold)
```

Execution

```
x3  x1  x1
```
Instructions

Programs

def run()
    if is_there[River]:
        mine(Wood)
        build_bridge()
        if agent[Iron] < 3:
            mine(Iron)
            place(Iron, 2, 3)
        else:
            goto(4, 2)
    while env[Gold] > 0:
        mine(Gold)

Natural Language Descriptions

If a river is in the environment, mine a wood and then use it to build a bridge. And then if agent has less than three iron, place an iron at (2,3). Otherwise if no river, goto location (4,2). Finally, whenever there’s still gold in the environment, mine a gold.

def run()
    while agent[Wood] <= 11:
        place(Wood, 2, 4)
        place(Iron, 1, 1)
        place(Iron, 8, 5)
        mine(Gold)
        mine(Gold)
        mine(Gold)
        mine(Gold)
        repeat(4):
            sell(Gold)
            sell(Iron)

While agent has no more than 11 wood, place wood at (2,4) and iron at (1,1), then place iron at (8,5) and mine gold twice, then mine gold. After the preceding procedure, sell gold and sell iron 4 times.
End-to-end Learning Baseline

```
def run():
    while env[Gold] > 0:
        mine(Gold)
    if is_there[River]:
        build_bridge()
        place(Wood, 2, 3)
```

Program

If an agent has more than 1 iron, place an iron on (2,3), and then if there are less than 3 gold in the environment, mine gold; otherwise, goto (4,2). While gold in the environment is larger than 2, keep mining gold.
def run():
    while env[Gold] > 0:
        mine(Gold)
        if is_there[River]:
            build_bridge()
            place(Wood, 2, 3)
Quantitative Results

Generalization

<table>
<thead>
<tr>
<th>Instruction Method</th>
<th>Natural language descriptions</th>
<th>Programs</th>
<th>Ours (concat)</th>
<th>Ours</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seq-LSTM</td>
<td>54.9±1.8%</td>
<td>56.7±1.9%</td>
<td>88.6±0.8%</td>
<td>94.0±0.5%</td>
</tr>
<tr>
<td>Transformer</td>
<td>52.5±2.6%</td>
<td>50.1±1.2%</td>
<td>94.0±0.8%</td>
<td>91.8±0.2%</td>
</tr>
<tr>
<td>Tree-RNN</td>
<td>32.4±4.9%</td>
<td>49.4±1.6%</td>
<td>40.9±1.5%</td>
<td></td>
</tr>
<tr>
<td>Transformer</td>
<td>38.2±1.2%</td>
<td>42.2±1.2%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ours (concat)</td>
<td></td>
<td>85.2±1.0%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ours</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Dataset: test, test-complex
- Generalization gap: 40.9%, 27.2%

(a) Instruction Length
(b) Instruction Complexity
Quantitative Results

Natural Languages < Programs

<table>
<thead>
<tr>
<th>Instruction Method</th>
<th>Natural language descriptions</th>
<th>Programs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Seq-LSTM</td>
<td>Transformer</td>
</tr>
<tr>
<td>Dataset</td>
<td>54.9±1.8%</td>
<td>52.5±1.2%</td>
</tr>
<tr>
<td>Generalization gap</td>
<td>40.9%</td>
<td>27.2%</td>
</tr>
<tr>
<td></td>
<td>32.4±4.9%</td>
<td>38.2±1.2%</td>
</tr>
<tr>
<td></td>
<td>38.8±1.2%</td>
<td>40.9±1.5%</td>
</tr>
<tr>
<td></td>
<td>42.2±1.2%</td>
<td>49.4±1.6%</td>
</tr>
<tr>
<td></td>
<td>50.1±1.1%</td>
<td>56.7±1.9%</td>
</tr>
<tr>
<td></td>
<td>85.2±1.0%</td>
<td>94.0±0.5%</td>
</tr>
<tr>
<td></td>
<td>88.6±0.8%</td>
<td>91.8±0.2%</td>
</tr>
</tbody>
</table>

(a) Instruction Length

(b) Instruction Complexity
## Quantitative Results

<table>
<thead>
<tr>
<th>Instruction Method</th>
<th>Natural language descriptions</th>
<th>Programs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Seq-LSTM</td>
<td>Tree-RNN</td>
</tr>
<tr>
<td>Dataset</td>
<td></td>
<td>Transformer</td>
</tr>
<tr>
<td>test</td>
<td>54.9±1.8%</td>
<td>52.5±2.6%</td>
</tr>
<tr>
<td>test-complex</td>
<td>32.4±4.9%</td>
<td>38.2±2.6%</td>
</tr>
<tr>
<td><strong>Generalization gap</strong></td>
<td>40.9%</td>
<td>27.2%</td>
</tr>
</tbody>
</table>

### Diagrams

- **(a) Instruction Length**
- **(b) Instruction Complexity**
Takeaway

- Specific tasks using programs

- Leverage the structure of programs with a modular framework
Composing Complex Skills by Learning Transition Policies

ICLR 2019

Skill Specification → Program → High-level Plan → Low-level Execution

Jump, Walk, Crawl
Learned Skills

\[ S \xrightarrow{\pi} a \]
Compose Complex Skills

High-level plan

Sequentially execute corresponding policies
Compose Complex Skills

High-level plan

Sequentially execute corresponding policies

An end state of a previous policy might not be a good initial state of the following policy
Compose Complex Skills

High-level plan

Sequentially execute corresponding policies

Jump

Walk

Crawl

Transition policies
Learning Transition Policies

- **Predictor** learns to judge if a state is good for executing the next policy.
- **Transition policy** learns from the predicted rewards.
Modular Framework
Qualitative Results

Locomotion

- Crawl
- Transition
- Walk
- Transition
- Walk Forward
- Transition
- Walk Backward

Manipulation

- Pick
- Transition
- Pick
- Toss
- Transition
- Hit
Quantitative Results

Sample Efficiency

(a) Repetitive picking up  (b) Repetitive catching  (c) Serve

(d) Patrol  (e) Hurdle  (f) Obstacle course
Quantitative Results

Locomotion

- Patrol
- Hurdle
- Obstacle course

Manipulation

- Repetitive picking up
- Repetitive catching
- Serve

- TRPO
- PPO
- Without TP
- TP-Task
- TP-Sparse
- TP-Dense
Takeaway

- Learning **transition policies** to smoothly compose learned skills
Jump
Walk
Crawl

Video demonstration
Trajectory demonstration
Language instruction
Reward

Stir-fry the onions until tender, and repeat this for garlic and carrots, if you have soy sauce, add some. Pour 2/3 cups the whisked eggs into the stir-fried and scramble.

Program

DEQ run()
   while True:
      if isCurb():
         jump
      elseif isCeiling():
         crawl()
      else:
         walk()
Stir-fry the onions until tender, and repeat this for garlic and carrots, if you have soy sauce, add some. Pour 2/3 cups the whisked eggs into the stir-fried and scramble.
Stir-fry the onions until tender, and repeat this for garlic and carrots, if you have soy sauce, add some. Pour 2/3 cups the whisked eggs into the stir-fried and scramble.
Primitive Skill Acquisition

- Goal: acquire a diverse set of primitive skills efficiently

Key directions:

- Meta-learning
- Meta-RL
- Learning from experts
Toward Multimodal Model-Agnostic Meta-Learning

Meta-learning Workshop at NeurIPS 2018

Skill Specification → Program → High-level Plan → Low-level Execution

Risto Vuorio | Hexiang Hu | Joseph J. Lim
Model-Agnostic Meta-Learning (MAML)

velocity = 1 m/s

velocity = 2 m/s

velocity = 3 m/s
Model-Agnostic Meta-Learning (MAML)

velocity = 1 m/s

velocity = 2 m/s

velocity = 3 m/s

velocity = 2.5 m/s

Fast adapt
Multimodal Task Distribution

Walk

Jump

Crawl
Multimodal Model-Agnostic Meta-Learning via Task-Aware Modulation

NeurIPS 2019 (Spotlight)
Multimodal Model-Agnostic Meta-Learning (MMAML)

![Diagram of MMAML](image)

- **Modulation Network**:
  - Input: \( \{X, y\} \times K \)
  - Task Encoder
  - Task Embedding
  - MLPs

- **Task Network**:
  - Input: \( X \)
  - Parameters: \( \theta_1, \tau_1, \theta_2, \tau_2, \ldots, \theta_n, \tau_n \)
  - Output: \( \hat{y} \)
Training Algorithm

Outer loop
- Task Encoder: produce the task embedding $\omega_g$
- MLPs: modulate the task network blocks $\omega_h$

Inner loop
- Task network: fast adapt through gradient updates $\theta$

---

**Algorithm 1 MMAML Meta-Training Procedure.**

1. **Input:** Task distribution $P(T)$, Hyper-parameters $\alpha$ and $\beta$
2. Randomly initialize $\theta$ and $\omega$.
3. **while** not DONE **do**
4. Sample batches of tasks $T_j \sim P(T)$
5. **for all** $j$ **do**
6. Infer $v = h([x, y; K; \omega_h])$ with $K$ samples from $D_{T_j}^{train}$.
7. Generate parameters $\tau = \{g_i(v; \omega_\theta) | i = 1, \ldots, N\}$ to modulate each block of the task network $f$.
8. Evaluate $\nabla_{\theta} \mathcal{L}_{T_j}(f(x; \theta, \tau); D_{T_j}^{train})$ w.r.t. the K samples
9. Compute adapted parameter with gradient descent:
   $\theta_{T_j} = \theta - \alpha \nabla_{\theta} \mathcal{L}_{T_j}(f(x; \theta, \tau); D_{T_j}^{train})$
10. **end for**
11. Update $\theta$ with $\beta \nabla_{\theta} \sum_{T_j \sim P(T)} \mathcal{L}_{T_j}(f(x; \theta', \tau); D_{T_j}^{val})$
12. Update $\omega_g$ with $\beta \nabla_{\omega_g} \sum_{T_j \sim P(T)} \mathcal{L}_{T_j}(f(x; \theta', \tau); D_{T_j}^{val})$
13. Update $\omega_h$ with $\beta \nabla_{\omega_h} \sum_{T_j \sim P(T)} \mathcal{L}_{T_j}(f(x; \theta', \tau); D_{T_j}^{val})$
14. **end while**
Regression

![Regression Diagram](image)

### Table: Comparison of Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>2 Modes</th>
<th>3 Modes</th>
<th>5 Modes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Post Modulation</td>
<td>Post Adaptation</td>
<td>Post Modulation</td>
</tr>
<tr>
<td>MAML [1]</td>
<td>-</td>
<td>1.085</td>
<td>-</td>
</tr>
<tr>
<td>Multi-MAML</td>
<td>-</td>
<td>0.433</td>
<td>-</td>
</tr>
<tr>
<td>LSTM Learner</td>
<td>0.362</td>
<td>-</td>
<td>0.548</td>
</tr>
<tr>
<td>Ours: MMAML (Softmax)</td>
<td>1.548</td>
<td>0.361</td>
<td>2.213</td>
</tr>
<tr>
<td>Ours: MMAML (FiLM)</td>
<td>2.421</td>
<td><strong>0.336</strong></td>
<td>1.923</td>
</tr>
</tbody>
</table>
Image Classification

(a) Omniglot  (b) Mini-ImageNet  (c) FC100  (d) CUB  (e) Aircraft

<table>
<thead>
<tr>
<th>Method &amp; Setup</th>
<th>2 Modes</th>
<th>3 Modes</th>
<th>5 Modes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>5-way</td>
<td>20-way</td>
<td>5-way</td>
</tr>
<tr>
<td></td>
<td>1-shot</td>
<td>1-shot</td>
<td>1-shot</td>
</tr>
<tr>
<td>MAML [1]</td>
<td>66.80%</td>
<td>44.69%</td>
<td>54.55%</td>
</tr>
<tr>
<td>Multi-MAML</td>
<td>66.85%</td>
<td>53.15%</td>
<td>55.90%</td>
</tr>
<tr>
<td>MMAML (ours)</td>
<td>69.93%</td>
<td>47.80%</td>
<td>57.47%</td>
</tr>
</tbody>
</table>
Learned Task Embedding (tSNE plot)

5-mode Classification

3-mode Regression

4-mode Reacher
Takeaway

- MAML struggles at learning from multimodal task distributions
- We propose multimodal MAML to alleviate the issue
Skill-based Meta-Reinforcement Learning

Deep RL workshop @ NeurIPS 2021
Meta-learning workshop @ NeurIPS 2021
submitted to ICLR 2022

Skill Specification → Program → High-level Plan → Low-level Execution

Taewook Nam, Karl Pertsch, Sung Ju Hwang, Joseph J. Lim
Meta-RL with Skills

Meta-RL

- Fast adapt to novel tasks
- Limited to short-horizon & dense-reward tasks

Skill-based RL

- Task-agnostic offline data
- Learning a novel task requires many samples

Ours
SiMPL: **Skill-based Meta Policy Learning**
Environments

(a) Maze Navigation

(b) Kitchen Manipulation

Meta-training Tasks

Target Tasks

Meta-training Tasks

Target Tasks

Bottom burner

Top burner

Slide cabinet

Hinge cabinet

Kettle

Light switch

Slide cabinet

Bottom burner

Microwave

Bottom burner

Light switch

Top burner

Kettle

Light switch

Slide cabinet

Hinge cabinet
Results

Maze Navigation

Episode 0
Episode 20
Episode 100

SiMPL (Ours)
SPIRL
MTRL
PEARL-ft
SAC

Completed Subtasks

Episode

Episode 0
Episode 20
Episode 100

PEARL-ft
MTRL

Meta-training Tasks
Target Task
Agent Trajectory

Episode

Episode 0
Episode 20
Episode 100

SiMPL (Ours)
SPIRL

Episode

Episode 0
Episode 20
Episode 100

PEARL-ft
MTRL
Generalizable Imitation Learning from Observation via Inferring Goal Proximity

NeurIPS 2021

Skill Specification → Program → High-level Plan → Low-level Execution

Youngwoon Lee
Andrew Szot
Joseph J. Lim
Learning from **Demonstration**

with expert's actions

Demo: \{s_1, a_1, s_2, a_2, s_3, a_3, \ldots\}

vs.

Learning from **Observation**

without expert's actions

Demo: \{s_1, s_2, s_3, \ldots\}
Learning from Observation via Inferring Goal Proximity

- **Expert Demonstrations**
  - Demo 1: 0.7, 0.8, 0.9, 1.0 (Goal)
  - Demo 2: 0.6, 0.7, 0.8, 0.9, 1.0 (Goal)
  - Demo N: 0.8, 0.9, 1.0 (Goal)

- **Learning Proximity Function**
  \[ f_{\phi}(s_{t+1}) = \text{Goal Proximity} \]

- **Joint Training**

- **Learning Policy**
  \[ \pi_{\theta}(a) = a \]

- **Agent Experience under Policy \( \pi_{\theta} \)**
  - Rollout 1: 0.4, 0.3, 0.2, 0.1 (Fail)
  - Rollout 2: 0.2, 0.3, 0.6, 1.0 (Goal)
  - Rollout M: 0.5, 0.8, 1.0 (Goal)

- **Proximity Reward**
  \[ f_{\phi}(s_{t+1}) - f_{\phi}(s_t) \]
Experiments

(a) Navigation 25% (b) Maze2D 50% (c) Ant Reach 0.05 noise (d) Fetch Pick 1.75x Noise (e) Fetch Push 1.75x Noise (f) Hand Rotate 0.35 Noise

(a) Navigation (b) Maze2D (c) Ant Reach (d) Fetch Pick (e) Fetch Push (f) Hand Rotate

(a) 100% Coverage (b) 75% Coverage (c) 50% Coverage (d) Proximity Heatmap
Program-Guided Framework for Interpreting and Acquiring Complex Skills

Interpretable
Programmatic / Generalizable
Hierarchical
Modular
Program-Guided Framework for Interpreting and Acquiring Complex Skills

Skill Specification -> Program -> High-level Plan -> Low-level Execution
def run():
    while frontIsClear():
        move()
        turnRight()
        if thereIsPig():
            attack()
        else:
            if not thereIsWolf():
                spawnPig()
            else:
                giveBone()
Learning to Synthesize Programs as Interpretable and Generalizable Policies

NeurIPS 2021
Learning to Synthesize Programs as Interpretable and Generalizable Policies

NeurIPS 2021

def run():
    while frontIsClear():
        move()
        turnRight()
        if thereIsPig():
            attack()
        else:
            if not thereIsWolf():
                spawnPig()
            else:
                giveBone()

Synthesized Program

def Task():
    if is_there[River]:
        mine(Wood)
        build_bridge()
    if agent[Iron] < 3:
        mine(Iron)
        place(Iron, 2, 3)
    else:
        goto(4, 2)
    while env[Gold] > 0:
        mine(Gold)

Program Guided Agent

ICLR 2020 (Spotlight)
Program Composing Complex Skills by Learning Transition Policies

Learning to Synthesize Programs as Interpretable and Generalizable Policies

NeurIPS 2021

Neural Program Synthesis from Diverse Demonstration Videos

Program

High-level Plan

Low-level Execution

ICLR 2020 (Spotlight)

ICLR 2019

Skill Specification

Program

Synthesized Program

def run()
while frontIsClear():
move()
turnRight()
if thereIsPig():
attack()
else:
if not thereIsWolf():
spawnPig()
else:
giveBone()

Synthesized Program

def run()
while frontIsClear():
move()
turnRight()
if thereIsPig():
attack()
else:
if not thereIsWolf():
spawnPig()
else:
giveBone()

Program Guided Agent

LEAPS Program Synthesizer

Synthesizer

Environment

Rewards

NeurIPS 2021

ICML 2018

NeurIPS 2021

Neural Program Synthesis from Diverse Demonstration Videos

Program

High-level Plan

Low-level Execution

ICLR 2020 (Spotlight)

ICLR 2019

Skill Specification

Program

Synthesized Program

def run()
while frontIsClear():
move()
turnRight()
if thereIsPig():
attack()
else:
if not thereIsWolf():
spawnPig()
else:
giveBone()
Learning to Synthesize Programs as Interpretable and Generalizable Policies

High-level Plan

Low-level Execution

Program Guided Agent

Program

def Task():
    if is_there[River]:
        mine(Wood)
        build_bridge()
    if agent[Iron] < 3:
        mine(Iron)
        place(Iron, 2, 3)
    else:
        goto(4, 2)
    while env[Gold] > 0:
        mine(Gold)
Toward Multimodal Model-Agnostic Meta-Learning

Multimodal Model-Agnostic Meta-Learning via Task-Aware Modulation

Skill-based Meta-Reinforcement Learning

Generalizable Imitation Learning from Observation via Inferring Goal Proximity
Thank You

Questions?