Neural Program Synthesis from Diverse Demonstration Videos

Presenter: Zhangheng Li

2021/11/11

Main Problem

Can robot understand human behaviors?

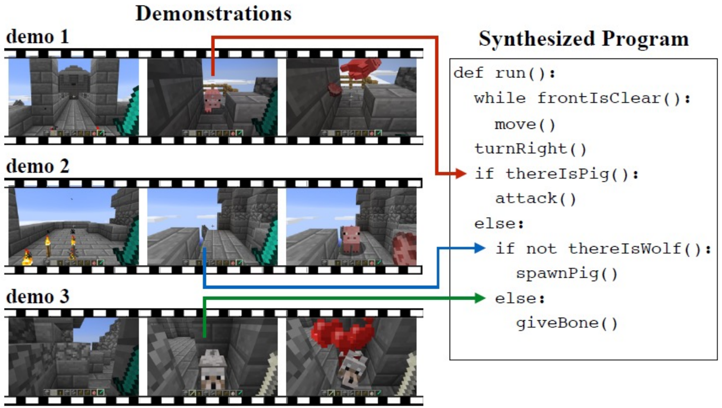
- ❖ Action recognition —— only know what, but how?
- ❖ Need to understand the underlying logic…

Why is it important for robot learning?

- ❖ Can better mimic human intelligence
- ❖ Can better interact with human

Bunel, Rudy R, Hausknecht, Matthew, et al. Leveraging grammar and reinforcement learning for neural program synthesis. ICLR 2018.
Xu, Danfei, Nair, Surajet al. Neural task programming: Learning to generalize across hierarchical tasks. ICRA 2018.
Devlin, Jacob, et al. Robustfill: Neural program learning under noisy i/o. ICML 2017.
Kaiser, Łukasz and Sutskever, Ilya. Neural gpus learn algorithms. ICLR 2016.

Neural Program Synthesis



Concept

- Interpretate behavior patterns as formal program languages
  - If else, while loop, recursion, etc.
  - Better generalization, interpretability

Existing works

- Learn latent programs (Kaiser et al. 2016, Xu et al. 2018)
  - Not good at interpretability
- Learn explicit programs with low-dim input (Devlin et al. 2017, Bunel et al. 2018)
  - Unable to scale to real-world input (e.g. image, video)

Neural Program Synthesis

Proposed Method

- Learn explicit programs from **demonstration videos**

Problem Statement

- Given a set of demonstration video trajectories $D = \{\tau_1, \tau_2, ..., \tau_K\}$
- Assume D is generated by an underlying program η^* (that generate actions)
- Assume η^* can be represented by a sequential code $C = (w_1, w_2, ..., w_N)$
- Obejective: Learn code C with diversity by sequential prediction

What we expect from the learned program

- Can interprete each trajectory
- Spot differences among trajectories and summarize conditions behind actions
- Describe the understanding in a written program language

Module Design

- **Demonstration Encoder**: encode each video trajectory as embeddings
- **Summarizer Module**: summarize embeddings and discover branching conditions
- **Program Decoder**: represent summarized understanding as program code sequence

Demonstration Encoder

Functionality

- Encode each of K video trajectories $\tau = ((s_1, a_1), (s_2, a_2), ..., (s_T, a_T))$

Composition

- A CNN to encode each video frame s_t

$$v^t_{\text{state}} = \text{CNN}_{\text{enc}}(s_t) \in \mathbb{R}^d$$

- An LSTM to summarize historical frames

$$c^t_{\text{enc}}, h^t_{\text{enc}} = \text{LSTM}_{\text{enc}}(v^t_{\text{state}}, c^{t-1}_{\text{enc}}, h^{t-1}_{\text{enc}})$$

Summarizer Module

Functionality

- Summarize embeddings and spot differences and conditions

Composition

- Re-encode (review) each trajectory with $\mathrm{LSTM}_{\mathrm{review}}$
  - Initialize $\mathrm{LSTM}_{\mathrm{review}}$ state as avg-pool of each trajectory embedding $c_{\mathrm{enc}}^{T,k}, h_{\mathrm{enc}}^{T,k}$
  - Produce each reviewed trajectory embedding $v_{\mathrm{demo}}^{k} = h_{\mathrm{review}}^{T,k}$
- Summarize relations of reviewed embeddings using **Relational Network** (Santoro et al. 2017)

$$v_{\mathrm{summary}} = \mathrm{RN}\left(v_{\mathrm{demo}}^{1}, \ldots, v_{\mathrm{demo}}^{K}\right) = \frac{1}{K^2} \sum_{i,j}^{K} g_{\theta}\left(v_{\mathrm{demo}}^{i}, v_{\mathrm{demo}}^{j}\right)$$

Program Decoder

- Initilized with v_{summary}, just predict the ground truth sequential program code using an **auto-regressive** LSTM
  - Given $C = \{w_1, w_2, ..., w_N\}$, when predicting w_n , feed w_{n-1} as input
  - During training: use ground truth code as w_{n-1}
  - During inference: use predicted w_{n-1} at last timestep

Obejective Function

- Given input demonstration set D and ground truth code $C = \{w_1, w_2, ..., w_N\}$

- $\mathcal{L}_{\text{code}}$: Use auto-regressive LSTM to learn C , use cross entropy loss to optimize
- Auxiliary Task:
  - Predict action sequences: $\mathcal{L}_{\text{action}} = -\frac{1}{MKT} \sum_{m=1}^{M} \sum_{k=1}^{K} \sum_{t=1}^{T} \log p(a_{m,t}^{k*} | A_{m,t-1}^{k}, v_{\text{demo}}^{k})$
  - Predict perceptions: $\mathcal{L}_{\text{perception}} = -\frac{1}{MKTL} \sum_{m=1}^{M} \sum_{k=1}^{K} \sum_{t=1}^{T} \sum_{l=1}^{L} \log p(\phi_{m,t,l}^{k*} | P_{m,t-1}^{k}, v_{\text{demo}}^{k})$
  - Refer to paper for more details…

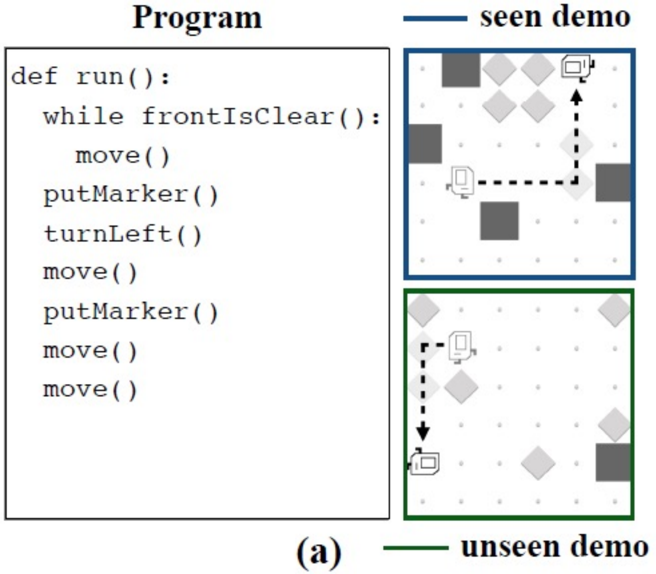
- Total loss: $\mathcal{L} = \mathcal{L}_{\text{code}} + \alpha \mathcal{L}_{\text{action}} + \beta \mathcal{L}_{\text{perception}}$

Experiment Setup

- Evaluation Criterion
  - Sequence acc: if learned ground truth code
  - Program acc: different sequence code lead to same program
  - Execution acc: generated actions compared to GT actions given same initial states
- Data
  - Generate multiple programs and corresponding demonstration set for training and testing
- Program primitives (Code elements)
  - Depend on tasks, such as IfElse(), While(), MoveForward(), TurnLeft(), DetectRight(), Repeat()

Pattis, Richard E. Karel the robot: a gentle introduction to the art of programming. John Wiley & Sons, Inc., 1981.

Karel

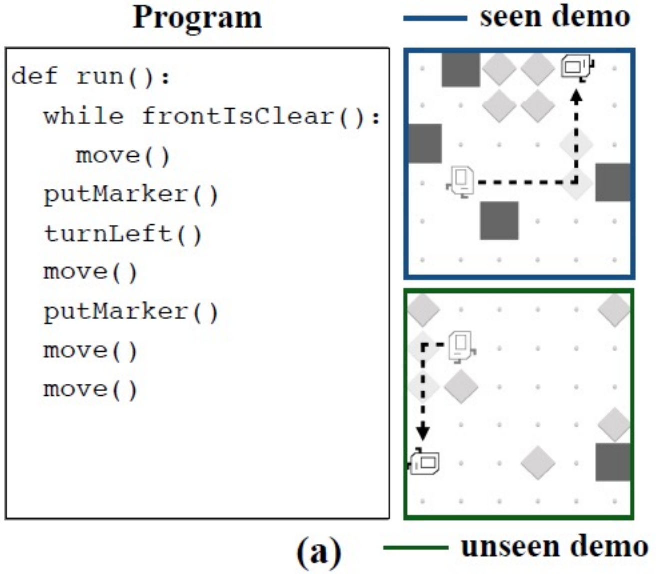


- A simple 2D goal finding game (Pattis et al. 1981)
  - 5 action primitives: for moving and interacting
  - 5 perception primitives: for detecting obstacles and markers
  - 25,000/5,000/5,000 train/val/test program samples
  - 10/5 seen/unseen (train/test) demonstrations in each corrsponding set

Methods	Execution	Program	Sequence
Induction baseline	62.8% (69.1%)	-	-
Synthesis baseline	64.1%	42.4%	35.7%
+ summarizer (ours)	68.6%	45.3%	38.3%
+ multi-task loss (ours-full)	**72.1%**	**48.9%**	**41.0%**

Pattis, Richard E. Karel the robot: a gentle introduction to the art of programming. John Wiley & Sons, Inc., 1981.

Karel



- A simple 2D goal finding game (Pattis et al. 1981)
    - 5 action primitives: for moving and interacting
    - 5 perception primitives: for detecting obstacles and markers
    - 25,000/5,000/5,000 train/val/test program samples
    - 10/5 seen/unseen (train/test) demonstrations in each corrsponding set

Methods	**k=3**	**k=5**	**k=10**
Synthesis baseline	58.5%	60.1%	64.1%
+ summarizer (ours)	**60.6%**	**63.1%**	**68.6%**
Improvement	2.1%	3.0%	4.5%

Ablation under variable **k**

Kempka, Michał, Wydmuch, Marek,et al. Vizdoom: A doom-based ai research platform for visual reinforcement learning. CIG 2106.

VizDoom (Doom-based)

- A 3D FPS game (Kempka et al. 2016)
  - 7/6 action/perception primitives
  - 80,000/8,000 train/test program samples

not inTarget Demon
→ moveRight()

inTarget Demon
→ attack()

Underlying Program

```
def run():
  if inTarget HellKnight:
    attack()
  if not inTarget Demon:
    moveRight()
  else:
    attack()
```

Synthesized Program

```
def run():
  if inTarget HellKnight:
    attack()
  if inTarget Demon:
    attack()
  else:
    moveRight()
```

Methods	Execution	Program	Sequence
Induction baseline	35.1% (60.6%)	-	-
Synthesis baseline	48.2%	39.9%	33.1%
Ours-full	**78.4%**	**62.5%**	**53.2%**

Summary and Limitation

- Very simple primitives (no function, recursion, computation, memory I/O etc.)
- Need (a lot) ground truth program codes for training —— unrealistic for real-world behaviors……
- No experimental comparison with inverse reinforcement learning

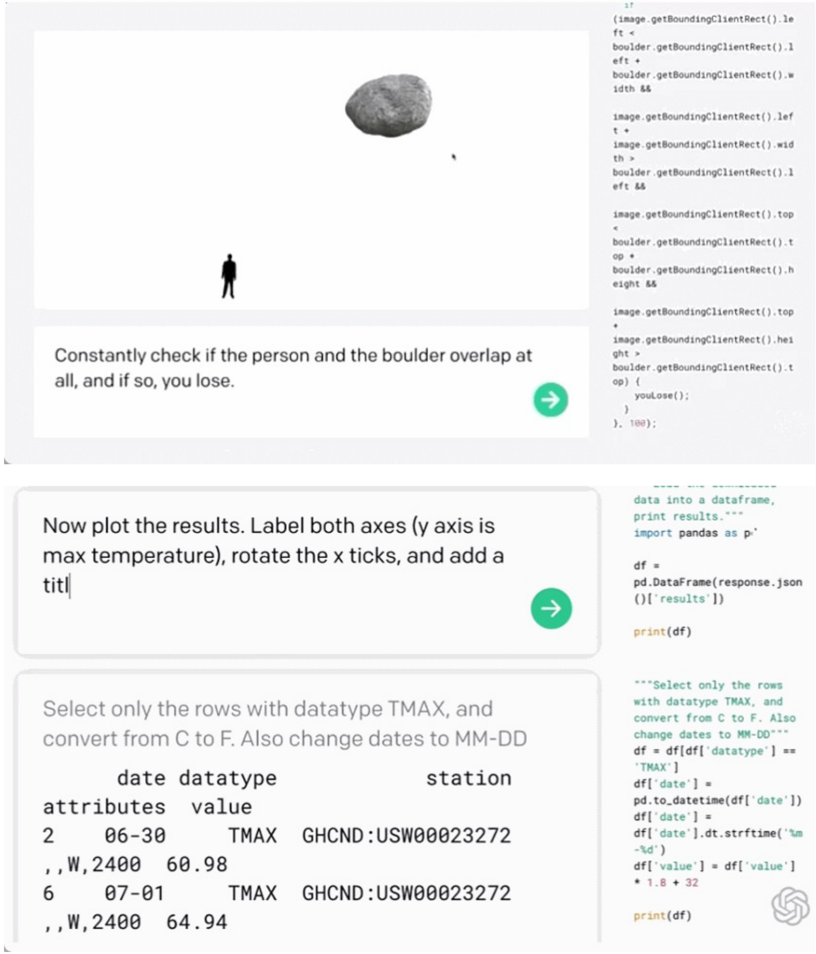
    - Inverse Reinforcement Learning: **label-free**, learn reward function

    - Neural Program Synthesis: label-required, learn execution program (Analogy: policy in more complex action space)

Extended Reading

OpenAI Codex

- Based on GPT-3
- Can generate programs from natural language
  - Various apps: games, plot charts…
  - Multiple languages: JavaScripts, python…



Thank you!

Questions?