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
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 $D = \{\tau_1, \tau_2, \ldots, \tau_K\}$
- Assume $D$ is generated by an underlying program $\eta^*$ (that generate actions)
- Assume $\eta^*$ can be represented by a sequential code $C = (w_1, w_2, \ldots, w_N)$
- Objective: Learn code $C$ with diversity by sequential prediction
What we expect from the learned program

- Can interpret 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 $\tau = ((s_1, a_1), (s_2, a_2), \ldots, (s_T, a_T))$

Composition

- A CNN to encode each video frame $s_t$
  $$v^t_{state} = \text{CNN}_{enc}(s_t) \in \mathbb{R}^d$$
- An LSTM to summarize historical frames
  $$c_{enc}^t, h_{enc}^t = \text{LSTM}_{enc}(v^t_{state}, c_{enc}^{t-1}, h_{enc}^{t-1})$$
Summarizer Module

Functionality

- Summarize embeddings and spot differences and conditions

Composition

- Re-encode (review) each trajectory with $\text{LSTM}_{\text{review}}$
  - Initialize $\text{LSTM}_{\text{review}}$ state as avg-pool of each trajectory embedding $c_{\text{enc}}^T, h_{\text{enc}}^T$
  - Produce each reviewed trajectory embedding $v_{\text{demo}}^k = h_{\text{review}}^{T,k}$

- Summarize relations of reviewed embeddings using Relational Network (Santoro et al. 2017)

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

Program Decoder

- Initialized with $U_{\text{summary}}$, just predict the ground truth sequential program code using an auto-regressive LSTM
  - Given $C' = \{w_1, w_2, \ldots, w_N\}$, 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
Objective 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()
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 corresponding set

```
def run():
    while frontIsClear():
        move()
    putMarker()
    turnLeft()
    move()
    putMarker()
    move()
    move()
```

<table>
<thead>
<tr>
<th>Methods</th>
<th>Execution</th>
<th>Program</th>
<th>Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>Induction baseline</td>
<td>62.8% (69.1%)</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Synthesis baseline</td>
<td>64.1%</td>
<td>42.4%</td>
<td>35.7%</td>
</tr>
<tr>
<td>+ summarizer (ours)</td>
<td>68.6%</td>
<td>45.3%</td>
<td>38.3%</td>
</tr>
<tr>
<td>+ multi-task loss (ours-full)</td>
<td>72.1%</td>
<td>48.9%</td>
<td>41.0%</td>
</tr>
</tbody>
</table>
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 corresponding set

<table>
<thead>
<tr>
<th>Methods</th>
<th>k=3</th>
<th>k=5</th>
<th>k=10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synthesis baseline</td>
<td>58.5%</td>
<td>60.1%</td>
<td>64.1%</td>
</tr>
<tr>
<td>+ summarizer (ours)</td>
<td>60.6%</td>
<td>63.1%</td>
<td>68.6%</td>
</tr>
<tr>
<td>Improvement</td>
<td>2.1%</td>
<td>3.0%</td>
<td>4.5%</td>
</tr>
</tbody>
</table>

Ablation under variable k
VizDoom (Doom-based)

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

OPEN CODEX URL: https://openai.com/blog/openai-codex/
Thank you!

Questions?