[Paper Review] Neural Program Synthesis from Diverse Demonstration Videos

1. Introduction

- The main objective of the paper is to **interpret decision making logic in demonstration videos**. If successful, then machine will have ability to interpret reason behind behaviors, enabling them to collaborate with and mimic humans.
- The paper took declarative programs as representations of decision making logic, which reduced the problem of interpreting decision making logic from demonstration videos to **synthesizing underlying program from demonstration videos**.
- **Proposes a neural program synthesizer that can explicitly synthesize underlying program from behaviorally diverse and visually complicated demonstration videos.**

2. Problem Overview

- Program(**η**)
  - deterministic function that outputs an action a given a history of states H_t, at time t

$$a_t = \eta(H_t)$$
$$H_t = (s_1, s_2, ..., s_t)$$

  - single program used to infer action from history of states.
  - Given initial state s1 and state history H1, underlying program **η*** could generate demonstration **τ,** a sequence of state and action tuples.

$$a_1 = \eta(H_1)$$
$$T : s_2 \sim T(s_1, a_1)$$
$$...$$
$$\tau = ((s_1, a_1), (s_2, a_2), ..., (s_T, a_T))$$
$$D = \{\tau_1, \tau_2, ..., \tau_K\}$$

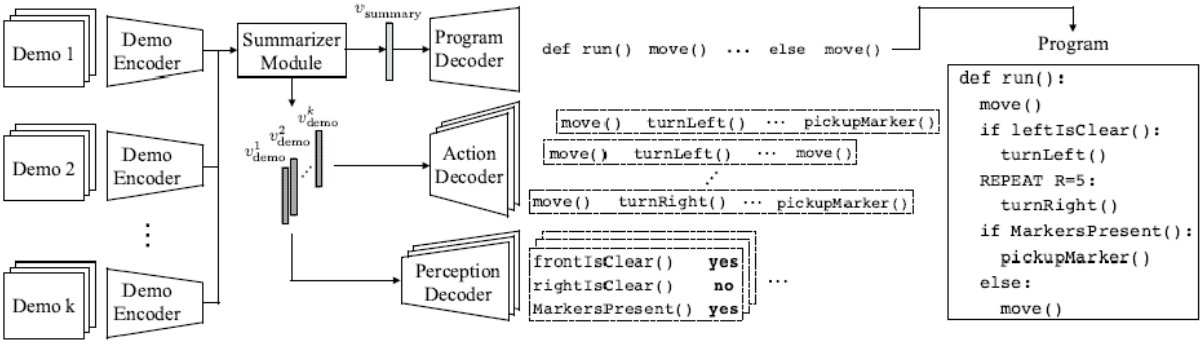
  - With different initial states, a set of demonstrations D could be generated.

- Problem Formulation
  - (While it is the paper's objective to infer the function eta from the demonstration set D(in order to reason decision making logic behind demonstration videos), instead, it predicts a sequence of codes C which is a more accessible representation.)
  - Sequence prediction problem where input is a set of demonstrations D and the output is a code sequence C.

$$D = \{\tau_1, \tau_2, ..., \tau_K\} \rightarrow C = \{w_1, w_2, ..., w_N\}$$

- Synthesized programs are defined in a domain specific language(DSL) with action primitives(ways agents interact with environments), perception primitives(ways agents percept environment), and control flows.

3. Model Architecture



- **Demonstration Encoder**: receives demonstration videos as input and produces an embedding(latent vector Z) that captures actions and perception of an agent.

  Encoder is constructed with strings of LSTM cells as it needs to handle demonstrations with variable number of frames. At each time step, input to the

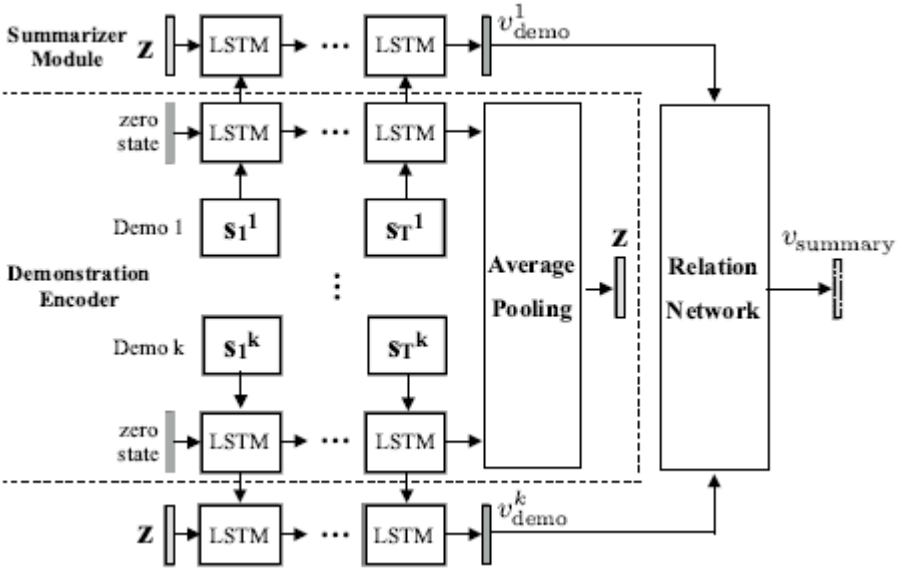
LSTM are cell state and hidden state from the previous time step(t-1), and state vector **v** of the time step(t), which is a embedding of state t. With these three inputs, LSTM produces cell state and hidden state of the time step t.

$$c_{enc}^{t}, h_{enc}^{t} = LSTM_{enc}(v_{state}^{t}, c_{enc}^{t-1}, h_{enc}^{t-1})$$

This goes on from $t \in \{1, T\}$ and at end, the final state tuple $(c_{enc}^{T}, h_{enc}^{T})$ is generated, which incorporates overall idea of the demonstration. These operations are applied to all K demonstrations to generate sequence of final state tuples.

$$(c_{enc}^{T,1}, h_{enc}^{T,1}), (c_{enc}^{T,2}, h_{enc}^{T,2}), ..., (c_{enc}^{T,K}, h_{enc}^{T,K})$$

These demonstration encoded outputs are average pooled to generate latent vector Z, which consists of $(c_{review}^{0}, h_{review}^{0})$. These are used as the initializer for the LSTM in the summarizer module.



- **Summarizer module**: summarizes where actions diverge between demonstrations and upon which branching conditions subsequent actions are taken.

  The summarizer module first re-encodes each demonstration with the context of latent vector Z(all encoded demonstrations) to infer branching conditions. Aside from cell state and hidden state from the previous step, hidden state from the encoder is given as input at each time step.

$$c_{review}^{t,k}, h_{review}^{t,k} = LSTM_{review}(h_{enc}^{t,k}, c_{review}^{t-1,k}, h_{review}^{t-1,k})$$

The last hidden state from reviewer LSTM becomes a demonstration vector: $v_{demo}^{k} = h_{review}^{T,k}$

The final summarization is performed across all K demonstration vectors and aggregates them into a single compact vector representation. This is done by relation network module.

g_theta is a MLP that generates relational embeddings for two demonstration vector i and j. It creates K^2 relational embeddings. v_summary is the average of aggregated k square relational embeddings of two (different/same) demonstration vectors.

$$v_{summary} = RN(v_{demo}^{1}, v_{demo}^{2}, ..., v_{demo}^{K}) = (1/K^2)\sum_{i,j}^{K} g_{\theta}(v_{demo}^{i}, v_{demo}^{j})$$

- **Program Decoder**: synthesize program codes from the summarized understanding of demonstrations.

  Initialized with summarized vector, decoder LSTM, at each time step, gets the previous token embedding as an input and outputs a probability of the following program tokens. By employing LSTM program decoder, the problem becomes sequence to sequence prediction.

  (During training previous ground truth token is fed as an input, and during inference, the predicted token in previous steps is fed.)

- **Learning**

  The model is trained to predict the following ground truth token w_i^*, given demonstration and previous code token w_{i-1}. Following is the objective of the model.

$$\mathcal{L}_{code} = -\frac{1}{NM}\sum_{m=1}^{M}\sum_{n=1}^{N} log\, p(w_{m,n}^{*}|W_{m,n-1}^{m}, D_m)$$

  Cross entropy loss should be minimized throughout training, which is calculated by taking expectation of log likelihood of ground truth token $w_{m,n}^{*}$ (token from m-th example and nth token), given that $W_{m,n-1}^{m}$(history of previous token inputs from 1 to n-1 time step) and D_m(m-th training demonstrations) exist.

- **Multi-task Objective**

  Predicting action sequences and predicting perception from demo vectors are given as auxiliary tasks, in case when the model couldn't be able to generate meaningful representations just from using encoder-summarizer-decoder, when environments increase in visual complexity.

  - Predicting action sequences

    Given a demo vector v_{demo}^k encoded by the summarizer module, an action decoder LSTM produces a sequence of actions. During training, sequential cross entropy loss is optimized.

$$\mathcal{L}_{action} = -\frac{1}{MKT}\sum_{m=1}^{M}\sum_{k=1}^{K}\sum_{t=1}^{T} log\, p(a_{m,t}^{k*}|A_{m,t-1}^k, v_{demo}^k)$$

    Loss is calculated by taking expectation of log likelihood of $a_{m,t}^{k*}$ the ground truth t-th action token in k-th demonstration of m-th training example, given there exists k-th demonstration vector and history of previous actions at time step t.

  - Predicting perceptions

    Perception vector $\Phi = \{\phi_1, \phi_2, ..., \phi_L\} \in \{0,1\}^L$ as an L dimensional binary vector is obtained by executing L perception primitives on a given state s. The perception loss is calculated by taking the expectation of log likelihood of $\phi_{m,t,l}^k$ the perception vector when history of encoded previous perception vectors and k-th demonstration vector exists.

$$\mathcal{L}_{perception} = -\frac{1}{MKTL}\sum_{m=1}^{M}\sum_{k=1}^{K}\sum_{t=1}^{T}\sum_{l=1}^{L} log\, p(\phi_{m,t,l}^{k*}|P_{m,t,l-1}^k, v_{demo}^k)$$

  By adding these two auxiliary tasks to the model, aggregated multi-task objective is as follows:

$$\mathcal{L} = \mathcal{L}_{code} + \alpha\mathcal{L}_{action} + \beta\mathcal{L}_{perception}, where\, \alpha = \beta = 1$$

4. Experiments

- Evaluation Metric
  - **Sequence Accuracy**: $Acc_{seq} = \frac{1}{M}\sum_{m=1}^{M} 1_{seq}(C_m^*, \hat{C}_m)$

    counts exact match of two code sequences. 1_seq is the indicator function of exact sequence match.
  - **Program Accuracy**: $Acc_{program} = \frac{1}{M}\sum_{m=1}^{M} 1_{prog}(C_m^*, \hat{C}_m)$

    checks if synthesized code is semantically identical with ground truth code. 1_prog returns 1 if any variations of synthesized code match with any variations of ground truth code
  - **Execution Accuracy**: $Acc_{execution} = \frac{1}{M}\sum_{m=1}^{M} 1_{execution}(D_m^*, \hat{D}_m)$

    counts exact match of execution result $\hat{D}$ of synthesized program code and the demonstrations D* generated by a ground truth program.
- Evaluation Setting
  - # of training programs: M_{train}
  - # of testing programs: M_{test}
  - Input set generation process: $C_m^* \rightarrow \eta_m^* \rightarrow D_m^*$

    Each program code is randomly sampled and compiled into an executable form. By running the program on different initial states, corresponding demonstrations are generated
  - Train set: $\Omega_{train} = \{(C_1^*, D_1^*), (C_2^*, D_2^*), ..., (C_{M_{train}}^*, D_{M_{train}}^*)\}$
  - Test set: $\Omega_{test} = \{(C_1^*, D_1^*), (C_2^*, D_2^*), ..., (C_{M_{test}}^*, D_{M_{test}}^*)\}$
- Baselines

  Two baselines are created to show the effectiveness of (1)explicitly modeling underlying programs and (2) using summarizer module and multi-task objective.
  - Induction baseline: bypasses synthesizing program and directly predicts action sequences.
  - Synthesis baseline: consists of demonstration encoder and program decoder
- Experiment Result(Karel)

  In Karel, agent navigates through a 8x8 gridworld with walls and interacts with markers based on the underlying program. It has 5 action and perception primitives.

Train : Validation : test = 25K : 5K : 5K

- Performance evaluation

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%**

Synthesis baseline outperforms induction baselines which shows the advantage of explicit modeling of underlying program. The summarizer module and multi-task objective introduce improvements in all evaluation metrics. Model often synthesizes programs that do not exactly match with the ground truth program but are semantically identical.

- Effect of summarizer module

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%

Table 2. Effect of the summarizer module. Employing the proposed summarizer module brings more improvement as the number of seen demonstration increases over *synthesis baseline* .

- Experiment Result(ViZDoom)

Doom is a 3D shooter game where player could move in continuous space to interact with monsters, items, and weapons. VizDoom is doom-based AI platform, where it has 7 action primitives including attack and 6 perception primitives checking the existence of different monsters and whether they are targeted. Train : Test = 80K : 8K

- Performance Evaluation

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%**

Synthesis baseline outperforms induction baseline in execution accuracy showing the strength of program synthesis for understanding diverse demonstrations. Summarizer module and multi-task objective brings in significant improvement in each accuracies. Synthesized program doesn't match the ground truth program in the code space, while it matches in the program space.

- Ability to infer underlying conditions

  To verify the importance of inferring underlying conditions, the evaluation is done only with programs containing single if-else statement with two branching consequences.

Methods	Execution	Program	Sequence
Induction baseline	26.5% (83.1%)	-	-
Synthesis baseline	59.9%	44.4%	36.1%
Ours-full	**89.4%**	**69.1%**	**58.8%**

  Induction baseline has difficulty inferring the underlying condition to match all the unseen demonstrations most of the time. In addition, the proposed model outperforms synthesis baseline in all metrics showing the effectiveness of summarizer module and multi-task objective in inferring underlying conditions.

5. Conclusion

The paper proposed the task of synthesizing a program from diverse demonstration videos. This was achieved by constructing a encoder-decoder model augmented with summarizer module to deal with branching conditions and a multi-task objective to induce meaning latent representation. It is evaluated on relatively simple environment Karel, and visually complex environment ViZDoom. The experiment results show that the model could reliably infer underlying program and achieve satisfactory performance.