Supplementary Material of Neural Program Synthesis from Diverse Demonstration Videos

Shao-Hua Sun [*1] **Hyeonwoo Noh** [*2] **Sriram Somasundaram** [1] **Joseph J. Lim** [1]

A. Detailed Network Architectures

The details descriptions of our proposed architecture are as follows. To alleviate the difficulty of reproducing our results, we make the code available at https://github.com/shaohua0116/demo2program/.

A.1. Demonstration Encoder

The demonstration encoder consists of a stack of convolutional layers and an LSTM. The stack of convolutional layers consists of three layers for Karel environment, which can be represented as: $C_{\{3,2,16\}} \rightarrow C_{\{3,2,32\}} \rightarrow C_{\{3,2,48\}}$, where $C_{k,s,n}$ denotes a convolutional layer with a kernel size k, stride s, and a number of channel n. For ViZDoom, to handle input frames with increased visual complexity, we stack two more convolutional layers $C_{\{3,2,48\}}$ afterwards. All the convolutional layers have LeakyReLu (Maas et al., 2013) as a non-linear activation, followed by a batch normalization layer (Ioffe & Szegedy, 2015). Then, the feature maps are flatten and passed to an LSTM. We experiment with RNN, GRU, and LSTM and found that LSTM works the best. A single layer LSTM with 512 hidden dimensions is used for all the experiments.

A.2. Summarizer Module

For the relation network of the summarizer module, we employ two fully-connected layers with 512 hidden dimensions as a function $g_\theta(v^i_{\text{demo}}, v^j_{\text{demo}})$, and LeakyReLu is used as a non-linear activation. For the first summarization, we experiment with RNN, GRU, and LSTM for summarizer module and found that LSTM works the best. A single layer LSTM with 512 hidden dimensions is used for the experiment.

[*]Equal contribution [1]Department of Computer Science, University of Southern California, California, USA [2]Department of Computer Science and Engineering, POSTECH, Pohang, Korea. Correspondence to: Shao-Hua Sun <shaohuas@usc.edu>.

Proceedings of the 35^{th} International Conference on Machine Learning, Stockholm, Sweden, PMLR 80, 2018.

A.3. Program Decoder

To build a token embedding function that is used for producing embedding vectors of program tokens, we create an embedding lookup with a hidden size of 128. An LSTM with a hidden size of 512 is utilized to decode program tokens. We experiment decoding program tokens while attending to all demonstrations or encoding the demonstration vector with attention mechanisms proposed in (Luong et al., 2015; Xu et al., 2015), but they do not show improvement. We believe it is partially because our summarizer module learns how to effectively summarize all demonstrations as a single compact vector.

B. Training Details

We implement the proposed model and its submodules in TensorFlow (Abadi et al., 2016) and trained it using a fixed learning rate of 10^{-3} with Adam optimizer (Kingma & Ba, 2015). The batch size that is used for training models is 128 for the Karel environment and 32 for ViZDoom.

C. Program Induction Baseline

To evaluate the effectiveness of explicitly modeling underlying programs, we implement a *program induction baseline* based on the One-shot imitation learning model proposed in (Duan et al., 2017). Since the model proposed in the original paper is not able to:

1. Incorporate multiple seen specification demonstration sequences
2. Handle a varying-length number of demonstrations
3. Deal with visual input

We make modifications as follows:

1. Augment the demonstration encoder with a stack of convolutional layers to process visual input
2. Remove temporal dropout, temporal convolution, and neighborhood attention
3. Add an LSTM with an attention mechanism (Luong et al., 2015). We also experimented with the monotonic attention mechanism (Raffel et al., 2017) and empirically found (Luong et al., 2015) works better.

4. Replace the context network with an average pooling layer to handle a varying-length number of demonstrations

5. Change the manipulation network to an LSTM decoder, which optimizes the predictions of one-hot action vectors at each time step

To make sure this baseline is well-trained, we extensively searched proper network architectures as well as ran a hyperparameter sweep on learning rate, learning rate schedule, etc., to select the *induction baseline* model that performs the best on each environment.

D. Program Accuracy

The *program accuracy* requires comparison of the synthesized program and the ground truth program in the program space, which is too huge to enumerate and make comparisons. An alternative to the program space is to make comparisons in the code space while identifying all program aliases to check whether a ground truth code is an alias of the synthesized code. However, this approach is intractable as well because the enumeration of all aliased programs itself is an intractable problem in general.

While intractable in general, the *program accuracy* is computable when the DSL is relative simple and some assumption is made, because the identification of all aliased programs become possible. For example, we assume the maximum iteration of the while statement, which could be altered to a finite number of repetitive if statements. We exploit the syntax and relation of the perception and action; for example, "if markersPresent" has "if not noMarkersPresent" as an aliased program. Based on these assumptions, we design enumeration rules to identify all program aliases and make comparisons.

E. Dataset Details

Both Karel and Vizdoom environment share the same control flow which is defined as follows:

$$
\begin{aligned}
\text{Program } m &\coloneqq \text{def run}() : s \\
\text{Statement } s &\coloneqq \text{while}(b) : (s) \mid s_1; s_2 \mid a \mid \text{repeat}(r) : (s) \\
&\quad \mid \text{if}(b) : (s) \mid \text{ifelse}(b) : (s_1) \text{ else} : (s_2) \\
\text{Repetition } r &\coloneqq \text{Number of repetitions} \\
\text{Condition } b &\coloneqq \text{percept} \mid \text{not } b \\
\text{Perception } p &\coloneqq \text{Domain dependent perception primitives} \\
\text{Action } a &\coloneqq \text{Domain dependent action primitives}
\end{aligned}
$$

Here, the only difference in the control flow is the number of repetitions used for the repeat statement, where the Karel environment uses the repetition in the range of $[0, 19]$ and the ViZDoom environment uses the repetition in the range of $[2, 4]$.

E.1. KAREL

We use 5 action primitives and 5 perception primitives for Karel, which is formally defined as follows:

$$
\begin{aligned}
\text{action} &\coloneqq \text{move} \mid \text{turnRight} \mid \text{turnLeft} \mid \text{pickMarker} \\
&\quad \mid \text{putMarker} \\
\text{perception} &\coloneqq \text{frontIsClear} \mid \text{leftIsClear} \mid \text{rightIsClear} \\
&\quad \mid \text{markersPresent} \mid \text{noMarkersPresent}
\end{aligned}
$$

For Karel environment, we use $8 \times 8 \times 16$ state representation, where each channel of the state representation has its own meaning.

$$
\begin{aligned}
&0 : \text{agent facing north} \mid 1 : \text{agent facing south} \| \\
&2 : \text{agent facing west} \mid 3 : \text{agent facing east} \mid \\
&4 : \text{wall or empty} \mid 5 \sim 15 : 0 \sim 10 \text{markers}
\end{aligned}
$$

E.2. VIZDOOM

ViZDoom model contains 7 action primitives and 6 perception primitives, which is formally defined as follows:

$$
\begin{aligned}
\text{action} &\coloneqq \text{moveBackward} \mid \text{moveForward} \mid \text{moveLeft} \\
&\quad \mid \text{moveRight} \mid \text{turnLeft} \mid \text{turnRight} \mid \text{attack} \\
\text{perception} &\coloneqq \text{isThere } m \mid \text{inTarget } m \\
\text{monster } m &\coloneqq \text{demon} \mid \text{hellKnight} \mid \text{revenant}
\end{aligned}
$$

ViZDoom environment has $120 \times 160 \times 3$ images as a state representations. We resize them to $80 \times 80 \times 3$ to feed to our model as an input.

To generate meaningful programs and collecting diverse behavior we use heuristics to sample codes and demonstrations. Given each state we sequentially increase the program length by adding more statements. At the same time action is instantly taken in the environment and the state transition is performed. Whenever the statement with condition is sampled for the program, we give higher sampling probability to perception that makes current state more diverse.

References

Abadi, Martín, Barham, Paul, Chen, Jianmin, Chen, Zhifeng, Davis, Andy, Dean, Jeffrey, Devin, Matthieu, Ghemawat, Sanjay, Irving, Geoffrey, Isard, Michael, et al. Tensorflow: A system for large-scale machine learning. In *OSDI*, 2016.

Duan, Yan, Andrychowicz, Marcin, Stadie, Bradly, Ho, OpenAI Jonathan, Schneider, Jonas, Sutskever, Ilya, Abbeel, Pieter, and Zaremba, Wojciech. One-shot imitation learning. In *NIPS*, 2017.

Ioffe, Sergey and Szegedy, Christian. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015.

Kingma, Diederik P and Ba, Jimmy. Adam: A method for stochastic optimization. In *ICLR*, 2015.

Luong, Thang, Pham, Hieu, and Manning, Christopher D. Effective approaches to attention-based neural machine translation. In *EMNLP*, 2015.

Maas, Andrew L, Hannun, Awni Y, and Ng, Andrew Y. Rectifier nonlinearities improve neural network acoustic models. In *ICML*, 2013.

Raffel, Colin, Luong, Minh-Thang, Liu, Peter J., Weiss, Ron J., and Eck, Douglas. Online and linear-time attention by enforcing monotonic alignments. In *ICML*, 2017.

Xu, Kelvin, Ba, Jimmy, Kiros, Ryan, Cho, Kyunghyun, Courville, Aaron, Salakhudinov, Ruslan, Zemel, Rich, and Bengio, Yoshua. Show, attend and tell: Neural image caption generation with visual attention. In *ICML*, 2015.