Supplementary Material: Learning State Representations from Random Deep Action-conditional Predictions

Zeyu Zheng
University of Michigan
zeyu@umich.edu

Vivek Veeriah
University of Michigan
vveeriah@umich.edu

Risto Vuorio
University of Oxford
risto.vuorio@cs.ox.ac.uk

Richard Lewis
University of Michigan
rickl@umich.edu

Satinder Singh
University of Michigan
baveja@umich.edu

A Potential Negative Societal Impact

While all AI advances can have potential negative impact on society through their misuse, this work advances our understanding of fundamental questions of interest to RL and at least at this point is far away from potential misuse.

B Source Code

We provide the source code for all experiments presented in the paper. The code can be found in the `./rgvfs_code` directory.

C Implementation Details

C.1 Experiments on the Empty Room Environment

Neural Network Architecture. The empty room environment is fully observable and so the state representation module is a feed-forward neural network that maps the current observation O_t to a state vector S_t. It is parameterized by a 3-layer multi-layer perceptron (MLP) with 64 units in the first two layers and 32 units in the third layer. The RL module has one hidden layer with 32 units and one output head representing the state value. (There is no policy head as the policy was given). The answer network module also has one hidden layer with 32 units and one output layer. ReLU activation is applied after every hidden layer. We applied a stop-gradient between the state representation module and the RL module.

Hyperparameters. Both the value function and the answer network were updated via TD. We used 8 parallel actors to generate data and updated the parameters every 8 steps. We used the Adam optimizer [6]. We searched the learning rate in $\{0.01, 0.001, 0.0001, 0.00001\}$ and selected 0.001 for all agents except the end-to-end agent which used 0.0001. The value function updates and the answer network updates used two separate optimizers with identical hyperparameters.

C.2 Atari Experiments

Neural Network Architecture. We used A2C [7] with a standard neural network architecture for Atari [8] as our base agent. Specifically, the state representation module consists of 3 convolutional

35th Conference on Neural Information Processing Systems (NeurIPS 2021).

layers. The first layer has 32 8×8 convolutional kernels with a stride of 4, the second layer has 64 4×4 kernels with stride 2, and the third layer has 64 3×3 kernels with stride 1. The RL module has one dense layer with 512 units and two output heads for the policy and the value function respectively. The answer network has one hidden dense layer with 512 units followed by the output layer. ReLU activation is applied after every hidden layer. We stopped the gradient from the RL module to the state representation module.

Hyperparameters. Following convention [8], we used a stack of the latest 4 frames as the input to the agent, i.e., the input to the state representation module at step t is $(O_{t-3}, O_{t-2}, O_{t-1}, O_t)$. We used 16 parallel actors to generate data and updated the agent's parameters every 20 steps. The entropy regularization was 0.01 and the discount factor for the A2C loss was 0.99. We used the RMSProp optimizer with learning rate 0.0007, decay 0.99, and $\epsilon = 0.00001$. The RL updates and the answer network updates used two separate optimizers with identical hyperparameters. We used two separate optimizers because the gradients from the RL loss and the gradients from the auxiliary loss may have different statistics. The gradient from the A2C loss was clipped by global norm to 0.5. The values for the above hyperparameters are taken from a well-tuned open-source implementation of A2C for Atari [3]. These values are used for all methods. When not stopping gradient from the RL loss, we mixed the RL updates and the answer network updates by scaling the learning rate for the answer network with a coefficient c. We searched c in $\{0.1, 0.2, 0.5, 1, 2\}$ on the 6 games in the main text. $c = 1$ worked the best for both rGVFs and baseline methods. The mixing coefficient was applied to the learning rate for the auxiliary loss because we used separate optimizers.

Baseline Methods. For MHVP, we used 10 value predictions following [4]. Each prediction has a unique discount factor, chosen to be uniform in terms of their effective horizons from 1 to 100 ($\{0, 1 - \frac{1}{10}, 1 - \frac{1}{20}, \ldots, 1 - \frac{1}{90}\}$). The architecture for MHVP is the same as rGVFs. We tried scaling MHVP up to 1024 predictions in the six Atari games but did not observe any significant performance improvement. Thus we decided to follow the original work. For PC, We followed the architecture design and hyperparameters used in the original work [5]. Specifically, we center-cropped the observation image to 80×80 and used 4×4 patches which resulted in 400 features. The discount factor was set to 0.9. The output of the state representation module is first mapped to a 2592-dimensional vector by a dense layer and then reshaped to a $32 \times 9 \times 9$ tensor. A deconvolutional layer then maps this 3D tensor to $|\mathcal{A}| \times 20 \times 20$ representing the action-values for each patch. Following [5], we used the dueling architecture [9]. For CURL, we followed the code accompanying the original paper [1]. We used random crop as the sole data augmentation. The $(84, 84, 4)$ stacked observation was first randomly cropped in to $(80, 80, 4)$ and then zero-padded to the original size. The output layer of the answer network has 128 units. The bilinear similarity was computed by a 128×128 matrix whose weights were learned together with the agent parameters. The exponential-moving-average target network used a step size of 0.001.

C.3 DeepMind Lab Experiments

Neural Network Architecture. For DeepMind Lab, we used the same RL module and answer network module as Atari but used a different state representation module to address the partial observability. Specifically, the convolutional layers in the state representation module were followed by a dense layer with 512 units and a GRU core [1, 2] with 512 units.

Hyperparameters. We used 32 parallel actors to generate data and updated the agent's parameters every 20 steps. The discount factor was 0.99. We searched the entropy regularization in $\{0.001, 0.003, 0.01, 0.03, 0.1\}$ for the A2C baseline on *explore_goal_locations_small*, *explore_object_locations_small*, and *lasertag_three_opponents_small* and selected 0.003. We used the RMSProp optimizer with decay 0.99 and $\epsilon = 10^{-8}$ without tuning. We searched the learning rate in $\{0.00003, 0.0001, 0.0003, 0.001\}$ for the A2C baseline and selected 0.0003. The gradient from the A2C loss was clipped by global norm to 0.5. The values for the above hyperparameters are used for all methods. The RL updates and the answer network updates used two separate optimizers with identical hyperparameters. We used two separate optimizers because the gradients from the RL loss and the gradients from the auxiliary loss may have different statistics. For end-to-end (not stop-gradient) agents, we searched the mixing coefficient c in $\{0.1, 0.2, 0.5, 1, 2\}$ on *explore_goal_locations_small*, *explore_object_locations_small*, and *lasertag_three_opponents_small*. $c = 1$ worked the best for

[1] `https://github.com/aravindsrinivas/curl_rainbow`

both rGVFs and baseline methods. The mixing coefficient was applied to the learning rate for the auxiliary loss because we used separate optimizers.

C.4 Hardware

Each agent was trained on a single NVIDIA GeForce RTX 2080 Ti in all of our experiments.

C.5 Computational Cost

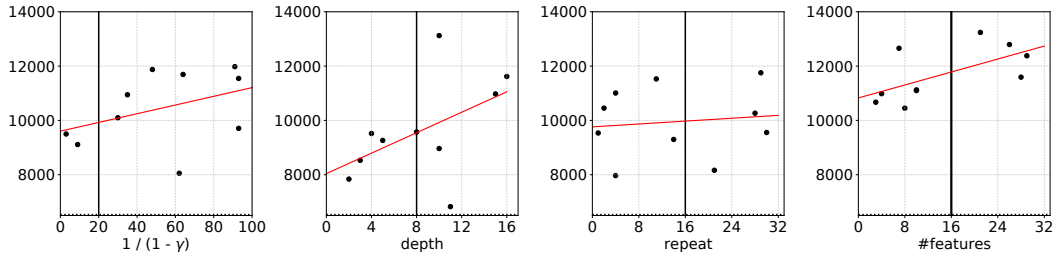
Like all auxiliary task methods, the additional computation of the GVF predictions introduces an overhead to the overall training pipeline. In our experiment, we found that the overhead was tiny. rGVFs was only 6% slower than the A2C baseline because the bottleneck was the agent-environment interaction rather than the parameter updates.

D Pseudocode for The Random Question Network Generator

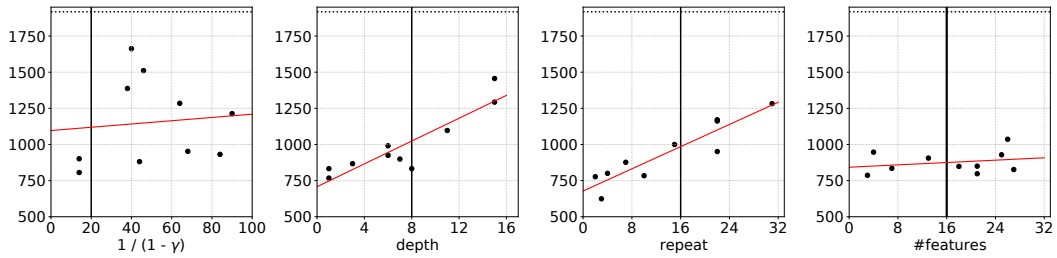
Algorithm 1 A Random Question Network Generator

Input: number of features n_p, discount factor γ, action set $\mathcal{A}$, depth D and repeat R
Output: a network G
$G \leftarrow$ an empty graph
$roots \leftarrow$ an empty set
$leaves \leftarrow$ an empty set
for $i = 1$ **to** n_p **do**
  create a new feature node f in G
  $roots \leftarrow roots \cup \{f\}$
  $leaves \leftarrow leaves \cup \{f\}$
  create a new prediction node v in G
  $leaves \leftarrow leaves \cup \{v\}$
  add edge $< v, f, 1 >$ to G
  add edge $< v, v, \gamma >$ to G
end for
for $d = 1$ **to** D **do**
  $expanded \leftarrow$ an empty set
  for $a \in A$ **do**
    $parent \leftarrow$ randomly select R nodes from $leaves$ without replacement
    for $p \in parent$ **do**
      create a new prediction node v in G
      mark v as conditioned on action a
      $expanded \leftarrow expanded \cup \{v\}$
      add edge $< v, p, 1 >$ to G
      $f \leftarrow$ randomly select a node from $roots$
      add edge $< v, f, 1 >$ to G
    end for
  end for
  $leaves \leftarrow expanded$
end for

E Additional Empirical Results



(a)



(b)

Figure 1: Scatter plots of scores in **(a)** BeamRider and **(b)** SpaceInvaders obtained by rGVFs with different hyperparameters. x-axis denotes the value of the hyperparameter. y-axis denotes the final game score after training for 200 million frames. The red line in each panel is the line of best fit. The dotted horizontal lines denote the performance of the end-to-end A2C baseline. The solid vertical lines denotes the values we used in our final experiments.

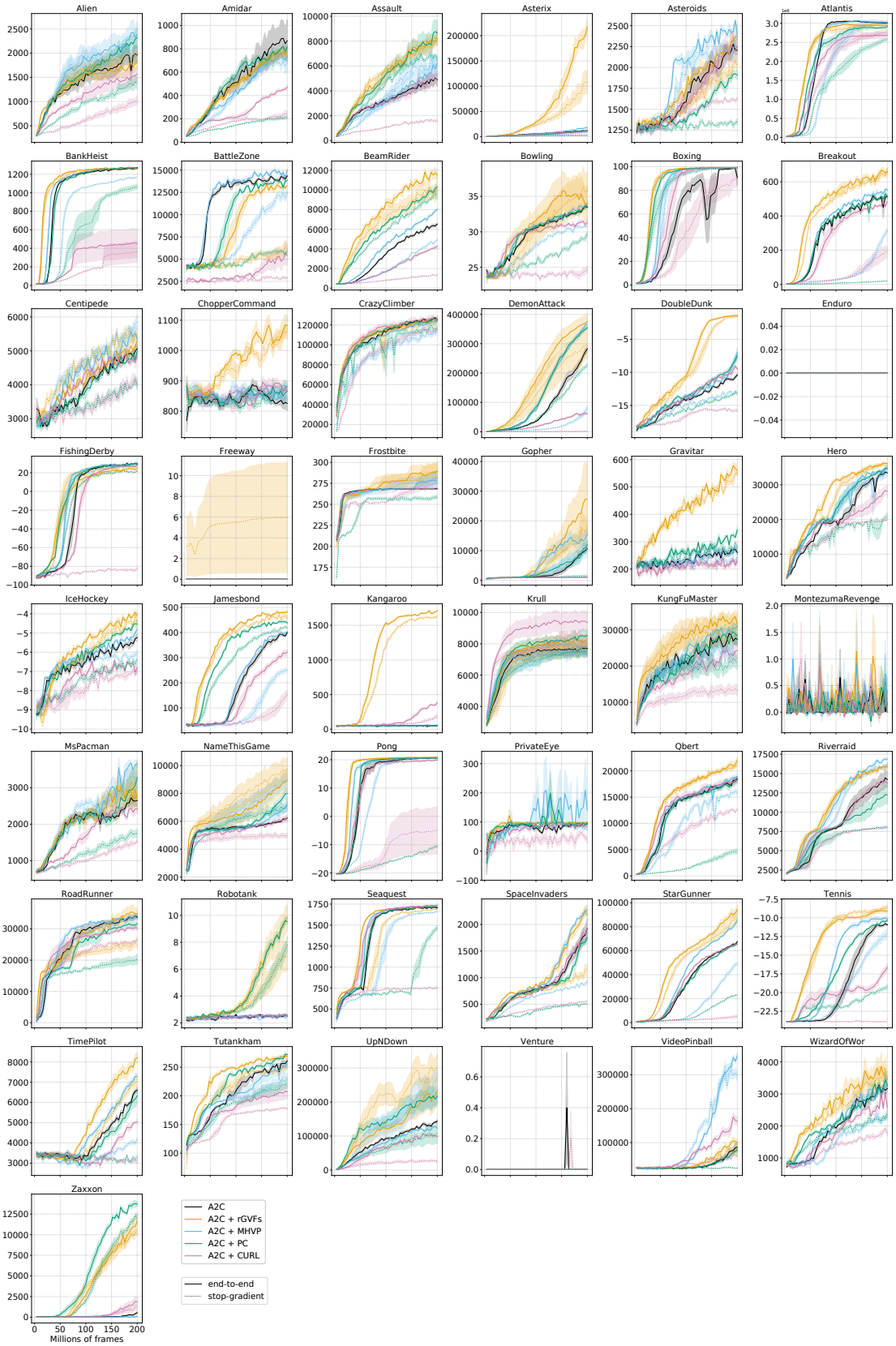


Figure 2: Learning curves in 49 Atari games. The x-axis denotes the number of frames. Each dark curve is averaged over 5 independent runs with different random seeds. The shaded area shows the standard error.

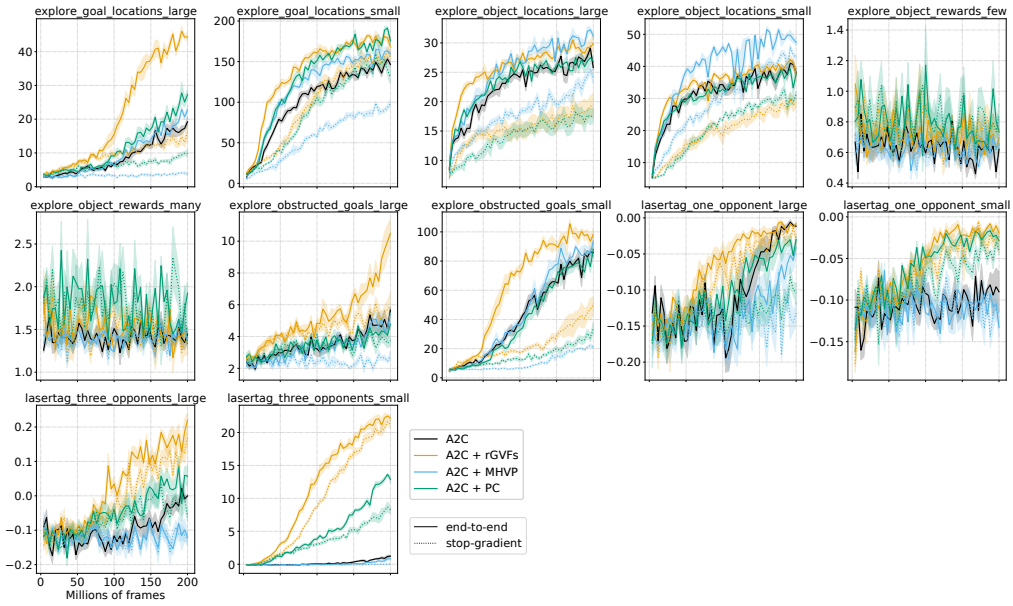


Figure 3: Learning curves in 12 DeepMind Lab environments. The x-axis denotes the number of frames. Each dark curve is averaged over 5 independent runs with different random seeds. The shaded area shows the standard error.

References

[1] Kyunghyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. In Dekai Wu, Marine Carpuat, Xavier Carreras, and Eva Maria Vecchi, editors, *Proceedings of SSST@EMNLP 2014, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation, Doha, Qatar, 25 October 2014*, pages 103–111. Association for Computational Linguistics, 2014.

[2] Junyoung Chung, Çaglar Gülçehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555, 2014.

[3] Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, Yuhuai Wu, and Peter Zhokhov. Openai baselines. `https://github.com/openai/baselines`, 2017.

[4] William Fedus, Carles Gelada, Yoshua Bengio, Marc G. Bellemare, and Hugo Larochelle. Hyperbolic discounting and learning over multiple horizons. *CoRR*, abs/1902.06865, 2019.

[5] Max Jaderberg, Volodymyr Mnih, Wojciech Marian Czarnecki, Tom Schaul, Joel Z. Leibo, David Silver, and Koray Kavukcuoglu. Reinforcement learning with unsupervised auxiliary tasks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

[6] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[7] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In Maria-Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 1928–1937. JMLR.org, 2016.

[8] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin A. Riedmiller, Andreas Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nat.*, 518(7540):529–533, 2015.

[9] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado van Hasselt, Marc Lanctot, and Nando de Freitas. Dueling network architectures for deep reinforcement learning. In Maria-Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 1995–2003. JMLR.org, 2016.