Explaining a Reinforcement Learning Agent via Prototyping - Supplementary Material

Ronilo J. Ragodos
Department of Business Analytics
University of Iowa
Iowa City, IA 52242
ronilo-ragodos@uiowa.edu

Tong Wang *
Department of Business Analytics
University of Iowa
Iowa City, IA 52242
tong-wang@uiowa.edu

Qihang Lin
Department of Business Analytics
University of Iowa
Iowa City, IA 52242
qihang-lin@uiowa.edu

Xun Zhou
Department of Business Analytics
University of Iowa
Iowa City, IA 52242
xun-zhou@uiowa.edu

A Appendix

A.1 Encoder Pre-training

To begin the pre-training routine, we construct a VAE whose encoder-decoder structures are based on the ResNet-18 architecture[1]. We do not use ResNet-18 models that were pre-trained themselves on ImageNet[2]. The variational encoder f_θ and decoder g_ϕ are both based on the ResNet-18 architecture. The implementation of our VAE comes from `https://github.com/julianstastny/VAE-ResNet18-PyTorch`. In order to facilitate encoding stacks of 4 greyscale images (instead of RGB images), we change the encoder's first convolutional unit to accept 4 input channels and remove the last fully connected layer. Likewise, the decoder outputs 4 channel states. Having created a VAE in this manner, we proceed with Algorithm 1. We perform $N = 100$ epochs of training for each experiment. The VAE is always trained on the same training data as is used in the downstream training task. See Section A.2 for details.

ProtoX Pre-Training Hyperparameters All experiments use a learning rate of $\eta = 1e-4$ and the Adam optimizer[3]. All experiments set $\delta = 15$, $\mid B \mid = 128$, and $m_1 = 2$, $m_2 = 2.5$.

Recall the training objective of a β-VAE:

$$\mathcal{L}_{VAE}(\theta, \phi, \beta; x, z) = \mathbb{E}_{g_\phi(z|x)}[log f_\theta(x \mid z)] - \beta D_{kl}(g_\phi(z \mid x) \| p(z)) \tag{1}$$

and the quadruplet loss:

$$\begin{aligned}\mathcal{L}_{\text{quadruplet}}(s_t, s_t^+, s_t^-, s_t^{--}) = \quad & \max(\left\| f_\theta(s_t) - f_\theta(s_t^+) \right\|_2^2 - \left\| f_\theta(s_t) - f_\theta(s_t^-) \right\|_2^2 + m_1, 0) \\ & + \max(\left\| f_\theta(s_t) - f_\theta(s_t^+) \right\|_2^2 - \left\| f_\theta(s_t^-) - f_\theta(s_t^{--}) \right\|_2^2 + m_2, 0)\end{aligned} \tag{2}$$

In the siamese VAE objective, the coefficient of the quadruplet loss term is set to 1000 and the coefficient of the KL-Divergence term is 1.5 (technically making the model a β-VAE).

*corresponding author

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

Algorithm 1 Pre-training phase

Require: Training data $\mathcal{D}_{tr}$, validation data $\mathcal{D}_{val}$, VAE with CNN encoder f_θ and decoder g_ϕ, window size $\delta > 0$, batch size $\mid B \mid$, margins m_1 and m_2, learning rate $\eta_e > 0$, and number of training epochs N.

1: **for** Epoch $e = 0, 1, ..., N$ **do**
2: Sample batch B from $\mathcal{D}_{tr}$
3: Let
$$\mathcal{T} = \{(a, p, n_1, n_2) \in B^4 | (a, p, n_1, n_2) \text{ is a valid quadruplet.}\}$$
4: Compute gradients of Eq. 1 w.r.t θ and ϕ over $\mathcal{T}$
5: Compute gradients of Eq. 2 w.r.t θ over B
6: Update θ and ϕ according to their gradients
7: **end for**

A.2 Behavior Cloning Training

A.2.1 ProtoX/ResNet-BC

All experiments use 30,000 samples from the expert for training. These 30,000 samples come from one set of trajectories. 80% of this is used for training and the remaining 20% is used for validation/early stopping. Another *separate* set of trajectories is used to gather 10,000 more samples for fidelity evaluation. The data used to evaluate sensitivity are generated the same way. In the Pong and Super Mario games, the states are cropped to remove the score information. In particular, rectangles of height 15 are removed from the states.

To begin our downstream behavior cloning training routine, we construct a ProtoX model whose encoder f_θ is pre-trained using Algorithm 1. The isometry layer A is initialized to the identity matrix. Although the identity is a trivial minimizer of the isometry penalty, we have noticed empirically that trained ProtoX models do not have A set to be the identity.

ProtoX Downstream Hyperparameters We allow a maximum of 1000 training epochs of training and set an early stopping threshold of 50. We use the Adam optimizer. The learning rate in each experiment is set to $5e-6$. Recall the training objective of ProtoX:

$$\mathcal{L}(\{p_k^a\}_{k=1,...,K}^{a\in\mathcal{A}}, A, W) = \frac{1}{n}\sum_{i=1}^{n} \mathcal{L}_{CE}(y_i, (e^a(x_i, W))_{a\in\mathcal{A}}) + \lambda_1 Sep + \lambda_2 Clst + \lambda_3 Rep + \lambda_4 Iso, \quad (3)$$

In all experiments, we set $\lambda_1 = 1e-4$, $\lambda_2 = 4e-4$, $\lambda_3 = 1e-5$, and $\lambda_4 = 1e-8$. When training ResNet-BC, everything is the same except 1) the prototype-projection step is removed and 2) only Cross Entropy is minimized.

A.2.2 VIPER

VIPER uses the DAGGER algorithm [4] to train decision tree classifiers. Because decision trees cannot process pixels as ProtoX can, we must manually extract meaningful features from the game states supplied by the OpenAI Gym Atari emulator. By inspecting the emulator's RAM at each time step, we may extract meaningful features without using the pixel-based states. At each time step of Pong games played by the PPO expert, we extract the positions of the player paddle, enemy paddle, and ball. By aggregating 4 time steps, we obtain the velocity, acceleration, and jerk of the paddles and ball. By supplying the VIPER agent with all of this information, we assume the Pong environment is appropriately modeled as fully-observed MDP.

We perform a similar processing routine for Seaquest. From the RAM states, we supply VIPER with 1) the position, velocity, acceleration, and jerk of the player submarine, 2) player direction, 3) player missile position and direction, 4) player oxygen level, 5) number of divers collected by the diver, and finally 6) the positions and directions of divers and enemy missiles.

We were able to mine this information from the RAM states using RAM annotations given here: `https://github.com/mila-iqia/atari-representation-learning/blob/master/atariari/benchmark/ram_annotations.py`.

A.2.3 GAIfO

The actor and critic in our implementation are given as a twin-headed network. They share an encoder given as a ResNet-18. The policy head and value head are both single linear layers. The discriminator is a 3-layer MLP with ReLU activations. Each network uses the Adam Optimizer. We substitute GAIfO's TRPO update step with PPO. We use a PPO implementation from `https://raw.githubusercontent.com/vpj/rl_samples/master/ppo.py` with minimal changes made to suit GAIfO. In particular, the reward function was changed as per the pseudocode provided in [5]. The code was also changed to accommodate Pong, Seaquest, and Super Mario Bros. The original implementation was meant to only handle structured data.

Our GAIfO implementation employs early stopping with a threshold of 20 epochs. The metric to evaluate model improvement is fidelity. To obtain a fidelity measurement, we roll out the GAIfO policy and check the percentage of time steps in which it agreed with the expert.

Algorithm 2 ProtoX Training Routine

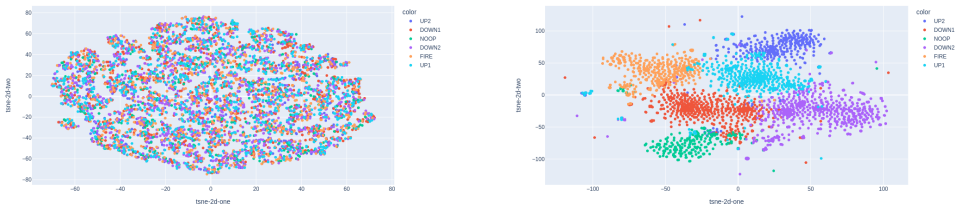
Require: Expert dataset $\mathcal{D} = \{(x_i, a_i)\}_{i=1}^{n}$, ProtoX $\mathcal{P}$, number of epochs N
  for Epoch $e = 0, 1, ..., N$ **do**
    Fit $\mathcal{P}$ on $\mathcal{D}$ minimizing $\mathcal{L}$
    if $e \ mod \ 25 = 0$ **then**
      for $k = 0, ..., K-1$ **do**
        $p_k^a \leftarrow \arg\min_{(x,a') \in \mathcal{D} | a'=a} \|Af_\theta(x) - p_k^a\|_2$
      end for
      Freeze prototypes and Isometry layer
      for $l = 0, ..., 19$ **do**
        Fit $\mathcal{P}$ on $\mathcal{D}$ minimizing $\mathcal{L}$
      end for
      Unfreeze prototypes and Isometry layer
    end if
  end for
  Return the best $\mathcal{P}$ on validation.

A.3 Similar States and Effect of Isometry Layer

In this section, we provide more examples of what states our model thinks are similar. To do this, we first sample a set of training data states x_t and show the top 10 states y_t that maximize $sim(f_\theta(x_t), f_\theta(y_t))$. To show how the isometry layer changes encodings, we also overlay the top 10 states y'_t that maximize $sim(Af_\theta(x_t), Af_\theta(y'_t))$. See Figures 2 and 3 an example from Super Mario Bros. 1-1. We note that both f_θ and Af_θ generally see states within a small temporal window as similar. These results show that the isometry layer is effective in allowing some evolution of the encoding space without throwing away what was learned by f_θ.

We also visualize the effect of the isometry layer using t-SNE plots. For each game, we compare t-SNE plots of the encodings of f_θ versus those of Af_θ. Generally speaking, we see that Af_θ partitions the encodings by their corresponding actions. There are large groups for each action that are further partitioned into smaller groups. The pretrained encoders f_θ seem to find many small groups sharing the same action. However, these small groups are not part of larger groups with the same associated action as with Af_θ. Rather, they are mixed. This phenomenon presumably occurs because, after all, the objective of ProtoX should encourage Af_θ to separate encodings by their corresponding action.

A.3.1 Pong



(a) f_θ (b) Af_θ

Figure 1: t-SNE plots of encodings for Pong

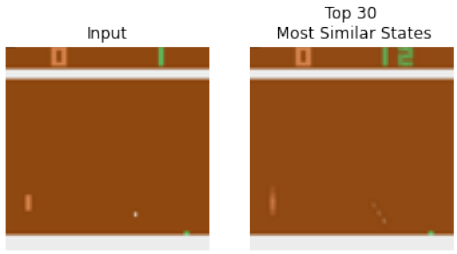


Figure 2: The top 30 most similar states, as viewed by f_θ, include those where the player score varies and the ball and paddles vary within a small temporal window.

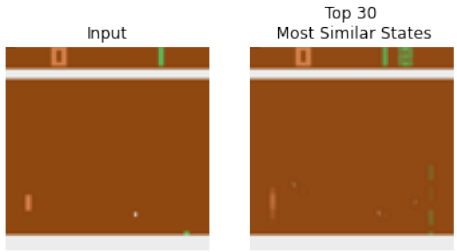
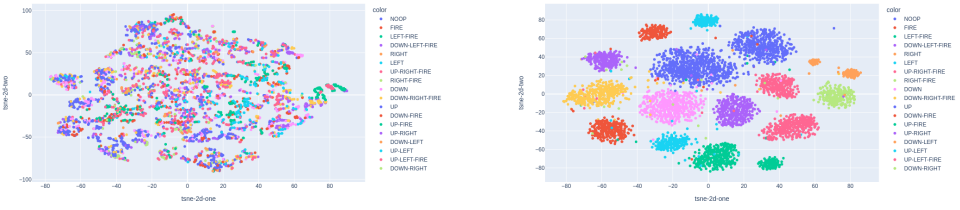


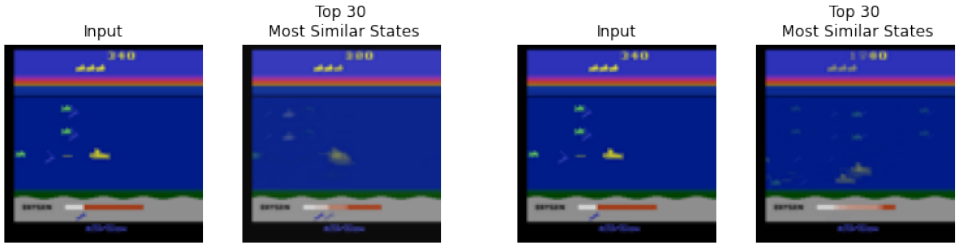
Figure 3: The top 30 most similar states, as viewed by Af_θ, include those where the player score varies, and the ball and paddle positions vary significantly (but along a trajectory in which the ball bounces off the bottom of the screen and the player paddle moves to intercept it).

A.3.2 Seaquest



(a) f_θ (b) Af_θ

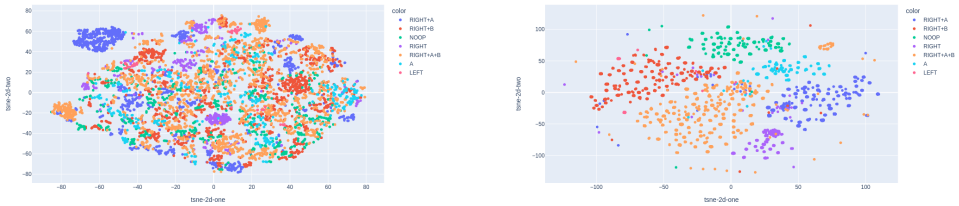
Figure 4: t-SNE plots of encodings for Seaquest



(a) f_θ (b) Af_θ

Figure 5: For Seaquest, f_θ sees the top 30 most similar states just as those that lie in a small temporal window around the input, while Af_θ captures a few different circumstances where the player submarine is shooting at an enemy to the right of the screen.

A.3.3 Super Mario 1-1



(a) f_θ (b) Af_θ

Figure 6: t-SNE plots of encodings for Super Mario 1-1

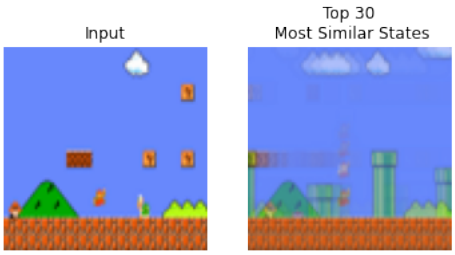


Figure 7: The top 30 most similar states, as viewed by f_θ, capture variation in Mario's vertical position along with instances where he is jumping over pipes.

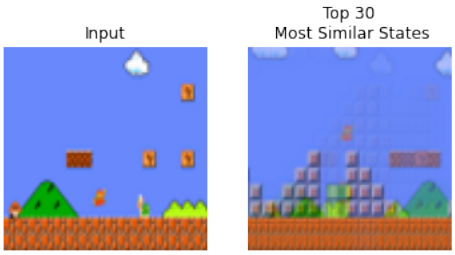


Figure 8: The top 30 most similar states, as viewed by Af_θ, capture Mario jumping over a pipe, over a barrier, over a Goomba, and up a stairway.

We also provide gridded versions of Figures 4 and 5 in the main paper, to more clearly show the image contents.

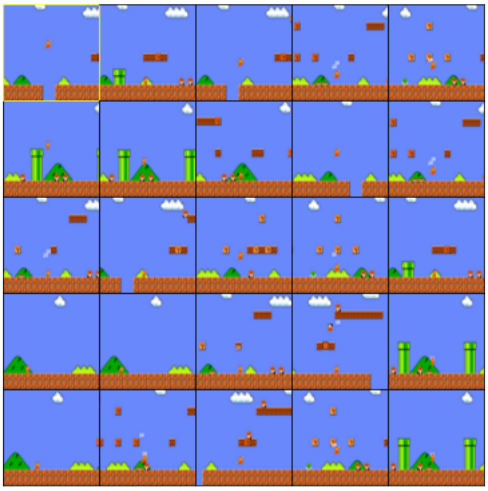
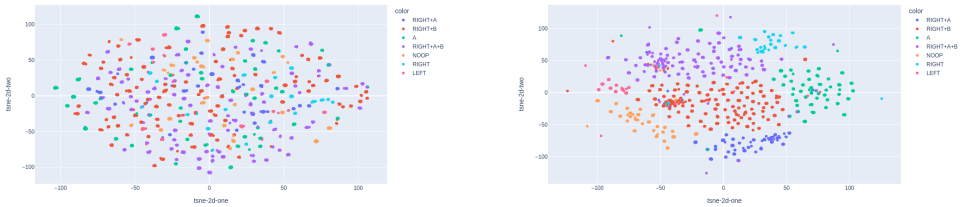


Figure 9: The top 24 most similar states, as viewed by f_θ, for the same input used in Figure 4 of the main paper. The yellow bordered image is the input.

A.3.4 Super Mario 8-3



(a) f_θ (b) Af_θ

Figure 10: t-SNE plots of encodings for Super Mario 8-3

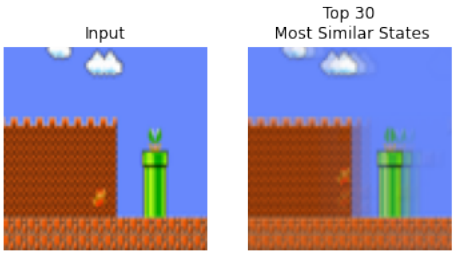


Figure 11: The top 30 most similar states, as viewed by f_θ, just include Mario's trajectory jumping over the piranha plant in the pipe.

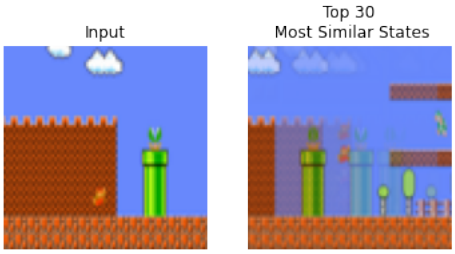


Figure 12: The top 30 most similar states, as viewed by Af_θ, include game frames both before and after Mario has completed his jump over the piranha plant in the pipe.

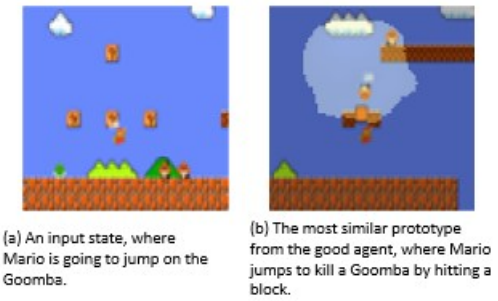


Figure 13: This is further justification of the claim in Figure 8 in the main paper that the good agent was jumping to kill the Goomba. In this example, a different scenario where the agent tries to kill a Goomba (left) is justified using the same prototype from Figure 8c (right) in the main paper. In the prototype, the agent focuses on a region encompassing where Mario will kill the Goomba.

A.4 Sensitivity Analyses

In this section, we perform sensitivity analyses for Pong, Seaquest, and Super Mario 8-3 for each baseline in order to compare with those of ProtoX. See Table 1 below for the results. We find that ProtoX is competitive with its black-box analogue ResNet-BC and far superior to VIPER and GAiFO in terms of sensitivity to flip-points.

Model	Pong	Seaquest	Super Mario 1-1	Super Mario 8-3
ProtoX	84%	87%	94%	89%
ResNet-BC	80%	99%	97%	98%
VIPER	40%	21%	–	–
GAiFO	22%	6%	3%	3%

Table 1: Comparison of sensitivities across models and games

A.5 Importance Maps

The importance maps we generate are meant to show which areas of ProtoX's learned prototypes are important to the agent for a given input state s. The input state s is of size $84 \times 84 \times 4$; s is composed of 4 consecutive greyscale game frames of size 84×84. The prototypes p are latent state representations though, and so need to be mapped back into the game state space. Denote the game state representative of p as x. We create the importance maps by repeatedly masking patches of x, and checking how the similarity with s changes. More concretely, we construct a tensor x' by masking a contiguous $7 \times 7 \times 4$ region of x. We then compute $\Delta = \mid sim(Af_\theta(s), x) - sim(Af_\theta(s), x') \mid$. This process is similar to an ablation study that seeks to test if ProtoX still views prototypes as similar to inputs after masking various patches of the prototypes. A large Δ indicates that the masked region is important. The intensity of the importance map at a given pixel is the average of all Δ values computed for patches containing the pixel. We iteratively mask patches over the entire image, moving the mask horizontally and vertically with a step of 1 pixel each time. Then every pixel (except the

ones on the edges) will be evaluated in 49 such evaluations. We average the resulting 49 Δs to obtain the importance of that pixel. Then, to highlight the most important area, we keep only the top 95% of pixels with the highest values. The rest are set to 0. See the algorithm block for a detailed description. To generate the importance maps, we use $H = W = 84$, and $m = 14$. See Figure 13 for another example of an importance map.

Algorithm 3 Importance Map Generation

Require: Input state s, prototype p, prototype state representative s_p, state height H, state width W, heat map h initialized to $0_{H \times W}$, mask size m

$e \leftarrow f(s)$
$s \leftarrow sim(e, p)$
$r \leftarrow \lfloor m/2 \rfloor$
for $i = -r, , ..., H - r$ **do**
  for $j = -r, ..., W - r$ **do**
    $sp' \leftarrow sp$
    $x_{min} = max(0, i)$
    $x_{max} \leftarrow min(i + m, 84)$
    $y_{min} \leftarrow max(0, j)$
    $y_{max} \leftarrow min(j + m, 84)$
    for $c = 1, 2, 3, 4$ **do**
      $sp'[c][xmin : xmax, ymin : ymax] \leftarrow 0$
    end for
    $f' \leftarrow f$
    $e' \leftarrow f(sp')$
    $s' \leftarrow sim(e', p)$
    $h[xmin : xmax, ymin : ymax] = h[xmin : xmax, ymin : ymax] + \mid s - s' \mid$
  end for
end for
$h \leftarrow h/m^2k$
return h

A.6 Links to Relevant Resources

The emulator for the Super Mario Bros. games can be found at:
`https://github.com/Kautenja/gym-super-mario-bros`

The (PyTorch) PPO experts for the Super Mario Bros. levels can be found at:
`https://github.com/uvipen/Super-mario-bros-PPO-pytorch`

References

[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[2] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

[3] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014.

[4] Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 627–635. JMLR Workshop and Conference Proceedings, 2011.

[5] Faraz Torabi, Garrett Warnell, and Peter Stone. Generative adversarial imitation from observation, 2018.