EVIDENCE-BASED DEBUGGING WITH DRL-MONITOR

Giang Dao & Minwoo Lee
University of North Carolina at Charlotte
{gdao, minwoo.lee}@uncc.edu

Introduction Deep learning (DL) has suggested many effective solutions to complex problems. However, the complexity of model makes it extremely difficulty to debug when DL does not work or learn properly. Deep neural networks are black box models, which do not allow interpreting what they have learned. Therefore, researchers or developers need to spend a lot of time on running a lot of different experiments to fix an error. In deep reinforcement learning (DRL), we frequently observe that a well-working algorithm in one environment fails to learn in another similar environment (i.e., Atari results in Mnih et al. (2015); Van Hasselt et al. (2016)). Here, we define *debugging* in machine learning as efforts to gaining better performing models in a fuller sense of the term. Narrowing down to debugging DRL, we can focus on *identifying* the problem of erroneous learning progress to be able to further improve it as we use print statements for debugging in other programming.

In this context, understanding or interpreting how DL or DRL learns can help debugging. For instance, we can use interpretable DL for supervised learning problems (Yosinski et al., 2015; Olah et al., 2018; Li et al., 2017; Tulio Ribeiro et al., 2016; Montavon et al., 2017) can suggest good debugging options. In DRL problems, t-SNE and saliency maps (Zahavy et al., 2016; Greydanus et al., 2017) can provide visualization of how an agent learns. However, they only provide information about the converged model, which is not enough for debugging. DRL-Monitor (Dao et al., 2018) combines DRL with a statistical method to collect evidences from learning process of a DRL agent. It memorizes important moments during DRL training and provides Bayesian inferences for further analysis. Attached to any state-of-the-art DRL algorithms, the DRL-Monitor can help debugging at any stage of the training process.

Identifying Errors with DRL-Monitor Fig. 1 shows the overall structure of using DRL-Monitor to debug DRL algorithms. In reinforcement learning, an agent interacts with an environment to collect transition experiences (state, action, reward, next state). The agent learns from the experiences when applying DRL algorithms. DRL-Monitor observes the learning process, utilizes sparse Bayesian reinforcement learning framework proposed by Lee (2017) to re-estimate DRL with similar input and output of DRL. A quality check is performed to ensure the estimation of the monitor is very closed to DRL. If the monitor passes the quality check, the method will extract evidences from the monitor to perform analysis and identify errors (we call this an *evidence*). The detail of DRL-Monitor is described in Appendix A.

Each evidence includes a transition experience with a weight distribution to indicate how strong the evidence is. The weight distribution tells how good or bad a transition experience is. These evidences show the agent's behavior through learning process. If the agent has bad behaviors, DRL algorithm can be enforced to learn with the evidence to have better agent's behavior and perform better in the environment. A case study that utilizes DRL-Monitor as a debugger to identify bad behaviors and to analyze them is provided in Appendix B.

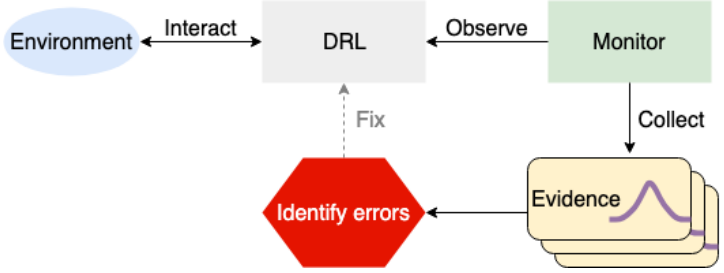

Figure 1: DRL-Monitor debugging cycle to improve learning performance. DRL agent interacts with environment to learn a policy while the monitor supervise the learning and collect evidences to identify errors and improve the learning performance of DRL algorithms.

REFERENCES

Giang Dao, Indrajeet Mishra, and Minwoo Lee. Deep Reinforcement Learning Monitor for Snapshot Recording. In *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pp. 591–598. IEEE, 2018.

Sam Greydanus, Anurag Koul, Jonathan Dodge, and Alan Fern. Visualizing and Understanding Atari Agents. *arXiv preprint arXiv:1711.00138*, 2017.

Minwoo Lee. *Sparse Bayesian Reinforcement Learning*. PhD thesis, Colorado State University, 2017.

Hao Li, Zheng Xu, Gavin Taylor, and Tom Goldstein. Visualizing the Loss Landscape of Neural Nets. *arXiv preprint arXiv:1712.09913*, 2017.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

Grégoire Montavon, Wojciech Samek, and Klaus-Robert Müller. Methods for Interpreting and Understanding Deep Neural Networks. *Digital Signal Processing*, 2017.

Chris Olah, Arvind Satyanarayan, Ian Johnson, Shan Carter, Ludwig Schubert, Katherine Ye, and Alexander Mordvintsev. The Building Blocks of Interpretability. *Distill*, 2018. doi: 10.23915/distill.00010. https://distill.pub/2018/building-blocks.

Michael E Tipping. Sparse Bayesian Learning and the Relevance Vector Machine. *Journal of Machine Learning Research*, 1:211–244, 2001. ISSN 15324435. doi: 10.1162/15324430152748236.

Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Why Should I Trust You?: Explaining the Predictions of Any Classifier. *arXiv preprint arXiv:1602.04938*, 2016.

Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

Jason Yosinski, Jeff Clune, Anh Mai Nguyen, Thomas J. Fuchs, and Hod Lipson. Understanding Neural Networks Through Deep Visualization. *CoRR*, abs/1506.06579, 2015. URL `http://arxiv.org/abs/1506.06579`.

Tom Zahavy, Nir Ben-Zrihem, and Shie Mannor. Graying the Black Box: Understanding DQNs. In *International Conference on Machine Learning*, pp. 1899–1908, 2016.

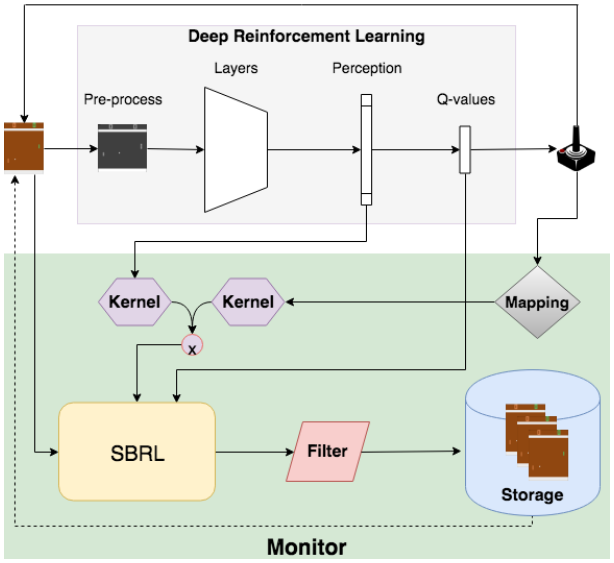

Figure 2: Deep reinforcement learning monitor framework (Dao et al., 2018)

A DRL-MONITOR

Fig. 2 shows the overall structure of Deep Reinforcement Learning Monitor. DRL-Monitor is consisted of two core modules: DRL module and monitor module. In DRL-Monitor, given a reinforcement learning task, the DRL module is the main workhorse for learning the task. The DRL algorithms take input and output actions with corresponding value such as state value (V) or state-action value (Q). The value is a numerical number represents how good of a given state or state-action input is. In order to reach state-of-the-art performance, DRL utilizes deep neural networks to approximate action and value given state input. An input is going through a series of layers from the neural networks to predict a value. The neural networks is updated by minimizing the following loss function:

$$\mathcal{L} = (r + \gamma \max_{a'} Q'(s', a') - Q(s, a))^2$$

To be able to systematically debug DRL with evidence, the monitor extracts a *perception*, an action, and an estimated output value for a given trajectory input. Perception is defined as what the neural networks interpret a state input and is a vector outputted from a neural network layer, which is before value output layer. The information extracted from DRL module is going through a sparse Bayesian reinforcement learning (SBRL) proposed by Lee (2017), which is built based on Sparse Bayesian Learning Tipping (2001). Two radial basis function kernels are applied for the perception and action to measure similarities between them:

$$\phi_i(\mathbf{p}, \mathbf{a}) = k_p(\mathbf{p}, \mathbf{p}_i) k_a(\mathbf{a}, \mathbf{a}_i).$$

The SBRL assumes the target $\mathbf{Q}$ is a weighted sum of the feature vectors $\hat{\mathbf{Q}} = \mathbf{\Phi w}$ with some noise ϵ such that:

$$\mathbf{Q} = \hat{\mathbf{Q}} + \epsilon$$

where ϵ is zero-mean Gaussian noise with variance σ^2. α is a set of hyper-parameters controlling the strength of the prior over the corresponding weights and is set to be infinity except for one starting as:

$$\alpha_i = \frac{||\phi_i||^2}{||\phi_i^\top \mathbf{Q}||^2 / ||\phi_i||^2 - \sigma^2}.$$

The posterior parameter distribution $p(w|\mathbf{Q}, \alpha, \sigma^2$ is the Gaussian distribution $\mathcal{N}(\mu, \Sigma)$ with $\mathbf{A} = \alpha \mathbf{I}$:

$$\Sigma = (\mathbf{A} + \sigma^{-2} \mathbf{\Phi}^\top \mathbf{\Phi}) \qquad \text{and} \qquad \mu = \sigma^{-2} \Sigma \mathbf{\Phi}^\top \mathbf{Q}.$$

The monitor re-estimates the DRL and remembers a sparse set of transition experiences (state, action, rewards, next state) as well as corresponding mean and variance information for further analysis. The monitor evaluates quality of the monitor estimation by comparing monitor predicted value with DRL predicted value with a heuristic method:

$$\sum(\hat{\mathbf{Q}}_i - \mathbf{Q}_i)^2 < \tau \sum(\bar{\mathbf{Q}} - \mathbf{Q}_i)^2 \quad \text{where } \tau \in [0, 1].$$

If the monitor produces a good quality of estimation, the sparse set of information will be move to a storage to be captured and considered as evidence. The sparse set of information is used to determine if there is a bug in the learning process.

B CASE STUDY: MS. PACMAN

We applied DRL-Monitor to observe a Double Deep Q-Network Van Hasselt et al. (2016) agent's behavior. The behaviors at 300K, 1.2M, and 2.4M trained through 2.4M simulation steps is presented in Fig. 3. At the beginning, the agent identifies that standing is not a good action indicated by a red circle. In fact, the agent should always move in MsPacman game environment to successfully survive. In the middle of the learning process (before the convergent) at 1.2M simulation steps, the agent realizes an important of food. Therefore, the agent develops a behavior that favors moving toward the food. Until this learning stage, the agent is developing a good behavior to interact with MsPacman game environment. However, at 2.4M simulation steps, the agent is still favor moving toward the food even though there is a ghost in the way. This bad behavior is caught with DRL-Monitor and telling the learning system that there is a bias behavior toward food collection of the agent that needs to be fixed.

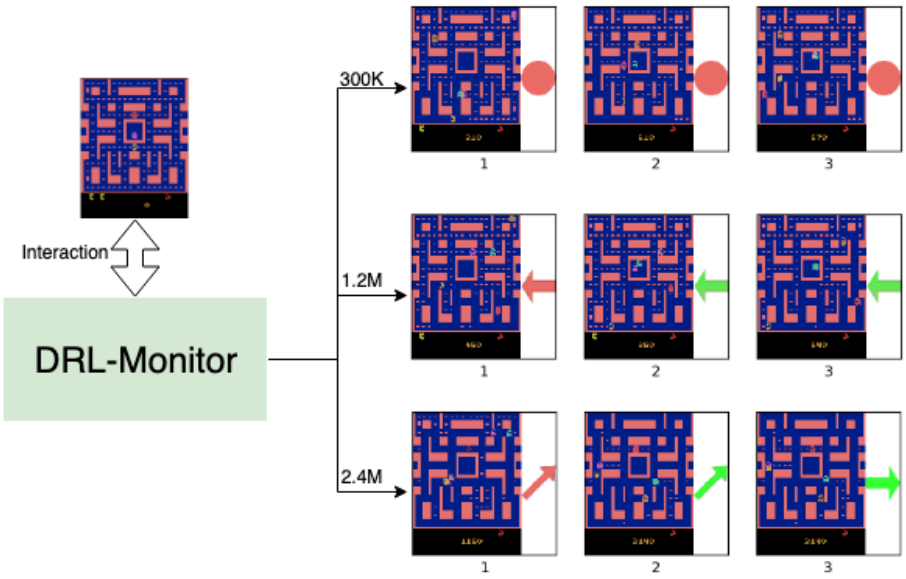

Figure 3: DRL-Monitor collects the best three most effective evidence of MsPacman behavior at 300K, 1.2M, and 2.4M simulation steps. The arrow shows action direction of MsPacman, the circle show agent choose standing action. Green and red correspondingly indicate positive and negative behavior of the agent.