A Feature Analysis Tool for Batch RL Datasets

Ruiyang Xu*
Northeastern University
USA
ruiyang@ccs.neu.edu

Zhengxing Chen*
Facebook
USA
czxttkl@fb.com

Abstract

Batch RL is concerned about learning a decision policy from a given dataset without interacting with the environment. Although research is actively going on learning-related issues (convergence speed, stability, and safety), empirical challenges should be emphasized even before learning but are largely ignored. That is, whether the underlying Markov Decision Process (MDP) are valid and meaningful. This study proposes a model-based method to check whether given data has a well-formed MDP through a heuristic-based feature analysis. We tested our method on a constructed environment as well as a real-world environment. Our results show that our approach can identify potential problems of the data. As far as we know, performing validity analysis on batch RL data is a novel direction, and we envision that our tool serves as a motivational example to open the door to this area.

CCS Concepts: • **Computing methodologies** → **Neural networks**; **Causal reasoning and diagnostics**.

Keywords: datasets, neural networks, feature analysis, causal inference, model-based RL

ACM Reference Format:
Ruiyang Xu and Zhengxing Chen. 2018. A Feature Analysis Tool for Batch RL Datasets. In *Proceedings of Workshop on Deep Reinforcement Learning for Knowledge Discovery (DRL4KD '21)*. ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/1122445.1122456

1 Introduction

conventional reinforcement learning algorithms, which based on an online learning paradigm (typically with the latest learned policy, and using that experience to interact with the environment to get real-time feedback, and then using those feedback to improve the policy [19]. However, such an algorithm usually poses difficulty in solving real-world problems where online interaction is impractical since data collection is expansive. As a result, researchers start paying attention to data-driven reinforcement learning (also known as Batch RL) approaches that utilize only previously collected offline data, without any other online interaction. It is interesting to see whether the data-driven approach can also be applied with RL problems once deep neural networks are incorporated [5, 9]. However, data that came from real-world environments can be quite noisy for an offline RL algorithm. Since a large number of state features might be defined, and meanwhile, only a few of them are relevant to the prediction of rewards and transitions, the RL algorithm could become unstable once being fed with those irrelevant features. Such an issue causes a challenge to both the learning algorithm and feature engineering. The latter even needs to be redesigned if the learned model exhibits excessively noisy rewards or transitions. An inappropriately designed feature set could formulate the target problem as a Partially Observable Markov Decision Processes (POMDPs) instead of Markov Decision Processes (MDPs), and POMDP usually causes problems to most offline RL algorithms [7]. In this paper, we propose a method of feature analysis for a given logged dataset. Our feature analysis borrows the idea from causal inference, and model-based RL [1, 11, 14] and tries to figure out potential causal relations among the input action, state features, and target prediction on rewards and state transitions.

The core idea of causal inference [1] is to make interventions. However, it is challenging to apply interventions on the given offline dataset because the underlying structural causal model (SCM [13]) is unknown. We learn a world model to tackle this problem, essentially an MDP estimator from the given dataset. Once we have the world model, we apply interventions to the state features and actions and see how those interventions affect the predicted rewards and state transitions. We then measure each feature's sensitivity to the intervention and use the results to indicate whether the given dataset is suitable for offline RL.

2 Related Work

Model-based RL (MBRL) [20] dates back to Sutton's earlier works on the Dyna algorithm [16–18]. In the Dyna algorithm, learning iterates between two phases: 1. gathering data from interaction with the environment and learning

*Both authors contributed equally to this research.

DRL4KD '21, April, 2021, Ljubljana, Slovenia

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/1122445.1122456

a world model. And 2. policy improvement with data generated by the learned model. The Dyna algorithm allows a policy to be learned with model-free algorithms without interaction with the real environment. Recent improvements of the Dyna algorithm based on learning with an ensemble of models and combining with more advanced policy learning algorithms like TRPO [10] and Meta-Learning [4].

Nevertheless, model-based RL has only been explored lightly for batch RL. An earlier work of Ross et al. [15] showed that performing MBRL directly to batch RL can have arbitrarily large sub-optimality. Latter, Kidambi et al. [8] proposed a new approach using pessimistic MDP and showed that their algorithm "matches or exceeds state-of-the-art results in widely studied offline RL benchmarks". However, it is challenging to evaluate a Model-based batch RL algorithm, given that one only has access to a limited dataset. Liu et al. [11] gives a counterfactual inference based MBRL and shows that it can vastly reduce the discrepancy between the learned and real environment dynamics, yet a concrete metric to evaluate the model performance still cannot be derived from their method.

Another line of this research is related to causal inference [1]. Buesing et al. [3] proposed a way to perform offline policy evaluation through casting MDPs into structural causal models (SCM [13]) for causal inference, where trajectories generated by the behavioral policy are viewed as observations and actions generated by target policy are viewed as interventions. Through a presumed SCM, which works as an inductive bias, the author shows that their method improves policy evaluation and search results on a non-trivial grid-world task.

3 Preliminary

3.1 Markov Decision Process

Reinforcement learning focuses on solving decision problems, usually formed as Markov decision processes (MDPs). An MDP contains the following components:

- State Space $\mathcal{S}$, which contains all possible states $s \in \mathcal{S}$ in a decision problem. In terms of the game, it contains all possible legal game state.
- Action Space $\mathcal{A}$, which contains all possible actions $a \in \mathcal{A}$ in a decision problem. In terms of the game, it contains all possible legal moves.
- Transition probabilities $\mathcal{T}(s'|s,a)$, which defines the dynamic from one state to another, namely, taking action a at state s has a probability $\mathcal{T}(s'|s,a)$ to arrive at state s'.
- Rewards $\mathcal{R}(s,a,s')$, which defines the expected reward after taking action a in state s and moving to state s'.
- Reward discount factor $\gamma \in (0,1]$, which weighs the importance of future rewards. Typically, the farther the distance of a reward from the current state, the less effective the reward brings to the current decision.

Decisions making on a specific MDP can be abstracted as a policy $\pi(a|s)$, which defines the probability distribution of taking some action a given some state s. That means, starting from a state s_t and given a policy π, an agent can move to next state by either sampling an action from π or taking a greedy action which has the highest probability. This process continues until the agent comes into a terminal state. Solving an MDP means to find an optimal policy that maximizes the state value function:

$$V^{\pi}(s) = \mathbb{E}_{\pi}[\, G_t | s_t = s]$$

where T is the episode's length, and G_t is the cumulative discounted reward:

$$G_t = \sum_{k=0}^{T} \gamma^i R(s_{t+k}, a_{t+k}, s_{t+k+1})$$

3.2 World Model

Our method is based on the world model [6]. To be specific, we only use the Mixture Density Recursive Neural Network (MDN-RNN network) to supervised learning the training data. The MDN-RNN takes in the current state and action, predicting the next state and reward. The next state is represented as a mixture Gaussian density, and the network learns the mean and standard deviation for each Gaussian [2]. The world model is a flexible probabilistic model that can deal with even stochastic reward and transitions because the underlying RNN model could output parameters for a mixture of Gaussian distributions.

In this research, we only care about the reward, and the next state predicted; therefore, we use the following loss function to train the RNN:

$$L = \underbrace{(r - \hat{r})^2}_{\text{reward loss}} \underbrace{-log \sum_{k}^{K} \alpha_k(s,a)\Phi(\hat{s}, \mu(s,a), \sigma(s,a))}_{\text{transition loss}}$$

where $\hat{r}$ and $\hat{s}$ are target values for reward and next state. α_k (the weight for the kth Gaussian in the mixture), μ and σ are output heads from the RNN. And the given number K is the total number of Gaussians in this mixture density model. The intuition behind this loss function is that we want to minimize the MSE on reward predicted, meanwhile maximize the log-likelihood for the next state predicted, using the given number of mixture Gaussians.

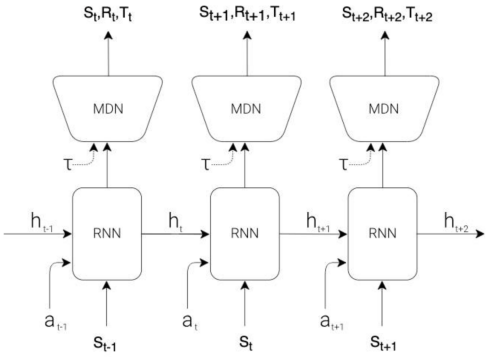


Figure 1. The MDN-RNN network for supervised learning the dynamics of the target environment.

4 Methodology

We first train a world model on the given dataset, which is sampled through an unknown behavior policy (in our experiment, we use the random policy as the behavior policy). After the training finished, we apply feature analysis on an evaluation batch, based on the following metrics:

- Reward importance metric (importance): a feature f_i's Reward importance (importance) is measured by the change of reward prediction when replacing all the appearance of f_i with its mean value. A higher reward sensitivity means that the feature interferes more with the reward signal and hence more importance to the reward prediction, under the Markov assumption.
- State sensitive metric (sensitivity): a feature f_i's state sensitive (sensitivity) is measured by the change of state prediction when randomly shuffling the input actions. To be specific, for a given feature f_i and for each state s and any possible action a, sensitivity measures the change of f_i on the predicted next state s'. A high reward importance but low state sensitivity indicates that the feature does interfere with the reward signal. However, the agent cannot control it, which indicates a hidden factor or POMDP.

We run the feature analysis process on an ensemble of models to perform a statistic test and compute a confidence interval for the experimental results. To perform a statistic test on reward importance and state sensitivity, we collect those values from each model in the ensemble and then build a normal distribution based on the collected values. We then perform a t-test on the given distribution with the null hypothesis (the null case in (a)). We take a confidence interval of 95%. Given the ensemble of models trained independently, we first test the existence of sensitive reward features. If there is no such feature, then the given dataset might not be suitable for RL. One might want to try Bandit algorithms or other approaches instead. Otherwise, if the feature is of reward importance, we would like to see whether it is controllable or not (Fig. 2). By shuffling the input actions, we measure the same feature's change on the predicted next state. An uncontrollable feature might introduce noises to the training data and make the state vector complicated. However, we suggest treating those features carefully because they are not always redundant. Some of them might indicate POMDP, which means using a more sophisticated model (e.g., DQN with multi-step inputs) might be helpful.

5 Experiment

5.1 Environments constructed

We constructed environments to simulate different situations in batch RL. All environments share the same state vector form, namely $s^t = [f_0^t, f_1^t, ..., f_N^t]$ (and $N = 10$ in our experiment), and there are only two actions, 0 or 1; yet they have different transition and reward functions. Based on the causal relation between states, actions, rewards, and possible hidden factors, we created and run feature analysis on the following environments:

1. Null relation: in this case, none of the state, action, and reward are dependent on each other. The state transition can be defined as: for any s^{t+1}, $P(f_i^{t+1} = f_i^t + 1) = 1 - \frac{i}{N}$. While the reward is a random signal from either 0 or 1 with probability 0.5. Such an environment works as a baseline, which will help us understand how sensitive metrics look like when everything is non-sensitive.
2. Action to reward causal relation: in this case, only actions have causal relations with the rewards; anything else keeps independent. That means that the state transition is the same as the case (1), but the reward signal will change with the probability $P(r = 1|a = 1) = 1$.
3. Action to state causal relation: in this case, only actions have causal relations with the states, anything else keeps independent. That means that the reward is the same as case (1), but the state will change with the probability
$$P(f_i^{t+1} = f_i^t + 1|a = 1) = 1 - \frac{i}{N}$$
and
$$P(f_i^{t+1} = f_i^t + 1|a = 0) = \frac{i}{N}$$
4. State to reward causal relation: in this case, only states have causal relations with the rewards; anything else keeps independent. That means that the state transition is the same as the case (1), but the reward signal will change with the probability $P(r = 1|f_k > \theta)$. Notice that in this case, we only focus on one specific feature, and the reward will change only when that feature increased larger than a certain threshold. In our experiment, we use $k = 0$ and $\theta = 50$.
5. Action to the state to reward causal relation: in this case, we have a full causal relation, which means that

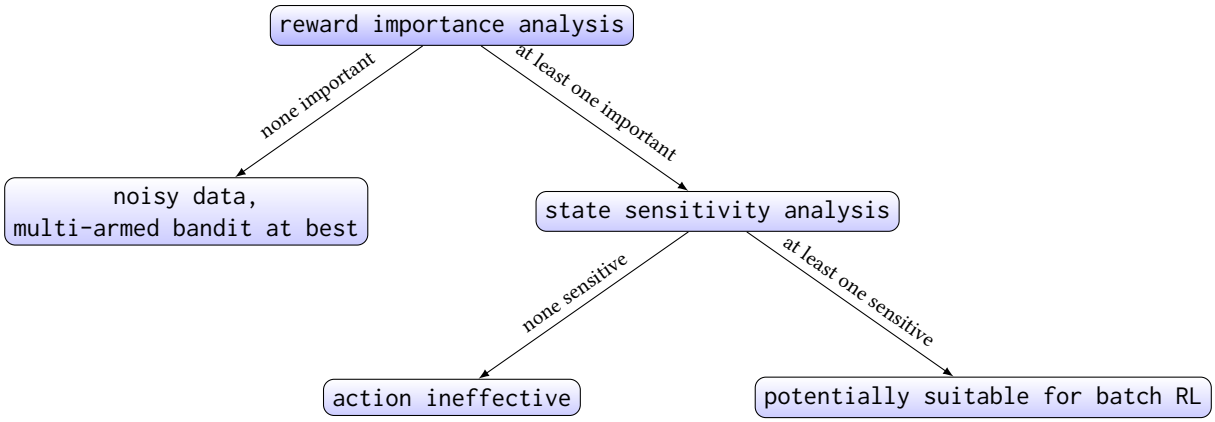


Figure 2. The process of feature analysis

the state transition behaves like (3), while the reward acts like (4).

6. Hidden factor as a confounder to state and reward. In this case, we have conditionally independent states and rewards, given hidden factors h's, which take value either 0 or 1. The state will change with the probability

$$P(f_i^{t+1} = f_i^t + 1 | h = 1) = 1 - \frac{i}{N}$$

and

$$P(f_i^{t+1} = f_i^t + 1 | h = 0) = 0.5(1 - \frac{i}{N})$$

Meanwhile the reward changes with the probability

$$P(r = 1 | h = 1) = 1$$

7. Action to state with Hidden factor as a confounder to state and reward. In this case, like in (6), hidden factors affect states and rewards, but the state is also affected by the actions chosen. As a result, the state changes with the following probabilities:

$$P(f_i^{t+1} = f_i^t + 1 | h = 1, a = 1) = 1 - \frac{i}{N}$$

$$P(f_i^{t+1} = f_i^t + 1 | h = 1, a = 0) = \frac{i}{N}$$

$$P(f_i^{t+1} = f_i^t + 1 | h = 0, a = 1) = 0.5(1 - \frac{i}{N})$$

$$P(f_i^{t+1} = f_i^t + 1 | h = 0, a = 0) = 0.5(\frac{i}{N})$$

The training and evaluation batches are sample from those environments with a given episode length $T = 100$, that means a terminal state is reached once $t = T = 100$.

5.2 Real-world environments

Besides the constructed environment, we also test our method on two real-world problems:

1. Load balancer [12]: jobs arrive over time, and a load balancing agent sends them to one of N (and $N = 10$ in our experiment) servers. The jobs arrive according to a Poisson process, and the job sizes follow a Pareto distribution. The N servers process jobs from their queues at different rates (the rates are set to $[1, 0, 1, 0, ..., 1, 0]$ in our experiment). On each job arrival, the agent observes a state $s = (q_1, q_2, ..., q_N, M)$, where q_i denoting the queue length at the server i and M is the size of the incoming job. It then takes action from $\{1, 2, ..., N\}$, and sends the job to one of the servers. The goal of the load balancer is to minimize the average job completion time. The reward corresponding to this goal is $R_t = -\tau \times j$, where τ is the time elapsed since the last action, and j is the total number of jobs in the queues.
2. Bipedal-walker-v2: this is the same environment from OpenAI Gym project [1]. The state has 24 features: hull angle, hull angular velocity, horizontal velocity, vertical velocity, hip joint 1 angle, hip joint 1 speed, knee joint 1 angle, knee joint 1 speed, leg 1 ground contact flag, hip joint 2 angle, hip joint 2 speed, knee joint 2 angle, knee joint 2 speed, leg 2 ground contact flag, and 10 Lidar readings. And the action is a 4-D vector with continuous values: Hip 1 Torque, Knee 1 Torque, Hip 2 Torque, Knee 2 Torque. Reward is given for moving forward, total 300+ points up to the far end. If the robot falls, it gets -100. Also, applying motor torque costs a small amount of points.

5.3 Experimental results

We first show our results for those constructed environments in Fig. 5, where each row contains two measurement metrics for the same environment. It can be seen from those figures that our method measures the causal relations among action, state, and reward adequately:

[1] See https://github.com/openai/gym/wiki/BipedalWalker-v2 for details.

1. Null relation: it can be seen from (b) and (c) that both of the reward sensitivity and state sensitivity are extremely low (less than 0.1) since this case reflects no causal relations (which means that none of the actions or rewards affects the prediction of the reward), the value here can be regarded as a baseline and used as a null hypothesis in a statistic test for the following cases. It should be noticed that there is a natural decreasing trend in (c), that is because of a natural decreasing of variance for predicted state transitions. Due to our environment's construction, features with higher feature numbers are much more stable than those with lower feature number. That is why we have a decreasing trend of variance for those features.
2. Action to reward causal relation: it can be seen from (e) that the predicated rewards are highly sensitive to the first actions. Moreover, from (f), we see that state sensitivity is insignificant, which means no apparent causal effect on state transitions. One can also notice the decreasing trend of variance, and the reason is the same as in (c).
3. Action to state causal relation: it can be seen from (h) that the reward sensitivity is extremely low, which means that there is no apparent causal effect on the reward. And from (i), we see that state sensitivity forms a parabolic curve due to the way the environment was constructed. To be specific, since the sensitivity is correlated with the probability of changing of a the predicted state feature after shuffling the action, we can calculate this probability to verify our experiment result. Given a state feature f_i, after shuffling, it either sees the same action as before the shuffling or a different action. And the predicted feature might change only if our model sees a different action, which means

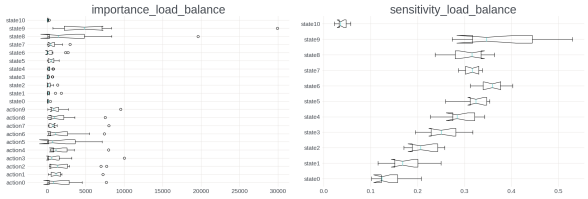
$$
\begin{aligned}
&P(\hat{f}_i^{t+1} \neq f_i^{t+1}) \\
&= P(a=0)P(f_i^{t+1} = f_i^t | a=0) \\
&P(\hat{a}=1)P(\hat{f}_i^{t+1} = f_i^t + 1 | \hat{a}=1) \\
&+ P(a=1)P(f_i^{t+1} = f_i^t | a=1)P(\hat{a}=0) \\
&P(\hat{f}_i^{t+1} = f_i^t + 1 | \hat{a}=0) \\
&+ P(a=0)P(f_i^{t+1} = f_i^t + 1 | a=0) \\
&P(\hat{a}=1)P(\hat{f}_i^{t+1} = f_i^t | \hat{a}=1) \\
&+ P(a=1)P(f_i^{t+1} = f_i^t + 1 | a=1) \\
&P(\hat{a}=0)P(\hat{f}_i^{t+1} = f_i^t | \hat{a}=0) \\
&= (\frac{i}{N})^2 - \frac{i}{N} + \frac{1}{2}
\end{aligned}
$$

   where $\hat{f}_i^{t+1}$ is the predicted state feature after the shuffling, while f_i^{t+1} is the predicted state feature before the shuffling. $\hat{a}$ is the shuffled action and a is the original action.
4. State to reward causal relation: it can be seen from (k) that only state feature 0 is a sensitive feature, which is consistent with our environment design. Furthermore, from (i), there is no causal effect on state transitions.
5. Action to the state to reward causal relation: since state features 0 intercepts the causal information flow from action to reward, it can be seen from (n) that state feature 0 has been weighted more than other state features and actions. On the other hand, in (o), we have similar results as (i).
6. Hidden factor as a confounder to state and reward. It can be seen from (q) that the learned model cannot tell it apart from a direct state to reward causal relation, even though the reward and state are conditionally independent with each other given the confounder h. In other words, without seeing h, one cannot claim that the state and reward are independent. Meanwhile, state transitions are non-sensitive to the action shuffling (r).
7. Action to state with Hidden factor as a confounder to state and reward. Like in (5), It is to be noted that it is challenging for our model to distinguish between case (m) and (s) because the learning is based on the correlation in the data; it cannot distinguish between causality and correlation. Therefore, for case (7), since the hidden factor is unobservable to the model, it will somehow learn correlations between state and reward, even though they are conditionally independent. However, one should notice that in (s), the action is disconnected with reward; that is why we see the actions are flat (t). On the other hand, for state transition sensitivity (n), we have similar results as in (i) and (o).

We also show the experimental results for the two real-world problems as following:

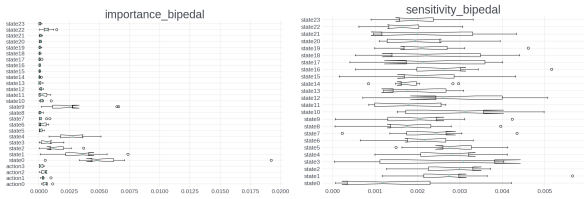
1. Load balancer: since this problem can be treated as a well-defined MDP, our method can efficiently apply feature analysis. It can be seen in Fig. 3 (a) that state features are less relevant to the rewards than action features, which is because that the reward is defined only based on the current queue size from each server, while the current action determines the change of queue size for each state. On the other hand, when it comes to the causal effects on the state transitions, one can see that the importance and sensitivity of state feature increases while the serving rate decreases, which is because those servers with lower serving rate pose to accumulate jobs in the queue, hence creating more variance to the state feature than those with a higher serving rate.
2. Bipedal-walker-v2: it is to be noted that this problem is POMDP because the agent cannot fully observe the environment, except the information from scanning the landscape every few seconds. Moreover, due to the environment's complexity, the dataset sampled

from a random policy rarely tells us anything. As a result, we see that neither rewards nor state transitions are sensitive to the action or state features (Fig. 4). Nevertheless, we do see that in (a), some features seem to be relatively more sensitive than others, and we guess that our method can still retrieve certain causal information from this complex environment. However, we should point out that for a complex POMDP, the world model cannot thoroughly learn it merely based on a randomly sampled dataset.



(a) (b)

Figure 3. Feature importance (a) and sensitivity (b) measured for the load balancer environment. Notice that state 0 to state 9 are queue lengths and state 10 is the size of incoming job.



(a) (b)

Figure 4. Feature importance (a) and sensitivity (b) measured for the bipedal-walker-v2 environment.

6 Conclusion

We proposed a feature analysis method for offline RL training data that helps RL practitioners evaluate feature sensitivities and examine reward/transition functions. Our approach can be used to accelerate feature engineering iterations and potentially improve training performance. Offline RL algorithms are suitable for problems were: (1) taking actions to lead to state transitions, (2) rewards are predictable by states or actions. If either condition is unsatisfied, it raises a flag for a more in-depth understanding of data. At the core of our method is the world model, with the inputs as state-action pairs and the outputs as the reward's predictions, next state. In other words, the world model attempts to learn the underlying MDPs from the logged data. We perform experiments both on sets of constructed ad-hoc environments and real-world environments. We show that the results are expected and consistent with the environment. We also notice that, for a complex environment (usually POMDP), the learned world model is highly biased by the behavior policy on which the training dataset is sampled. Nevertheless, our method can still be regarded as a reference for environment design and feature engineering in offline RL.

References

[1] James Bannon, Brad Windsor, Wenbo Song, and Tao Li. 2020. Causality and Batch Reinforcement Learning: Complementary Approaches To Planning In Unknown Domains. arXiv:2006.02579 [cs.LG]

[2] Christopher M. Bishop. 1994. *Mixture density networks.* Technical Report.

[3] Lars Buesing, Theophane Weber, Yori Zwols, Sebastien Racaniere, Arthur Guez, Jean-Baptiste Lespiau, and Nicolas Heess. 2018. Woulda, Coulda, Shoulda: Counterfactually-Guided Policy Search. arXiv:1811.06272 [cs.LG]

[4] Ignasi Clavera, Jonas Rothfuss, John Schulman, Yasuhiro Fujita, Tamim Asfour, and Pieter Abbeel. 2018. Model-Based Reinforcement Learning via Meta-Policy Optimization. arXiv:1809.05214 [cs.LG]

[5] Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. 2020. D4RL: Datasets for Deep Data-Driven Reinforcement Learning. arXiv:2004.07219 [cs.LG]

[6] David Ha and Jürgen Schmidhuber. 2018. Recurrent World Models Facilitate Policy Evolution. arXiv:1809.01999 [cs.LG]

[7] Matthew Hausknecht and Peter Stone. 2017. Deep Recurrent Q-Learning for Partially Observable MDPs. arXiv:1507.06527 [cs.LG]

[8] Rahul Kidambi, Aravind Rajeswaran, Praneeth Netrapalli, and Thorsten Joachims. 2020. MOReL : Model-Based Offline Reinforcement Learning. arXiv:2005.05951 [cs.LG]

[9] Aviral Kumar, Justin Fu, George Tucker, and Sergey Levine. 2019. Stabilizing Off-Policy Q-Learning via Bootstrapping Error Reduction. arXiv:1906.00949 [cs.LG]

[10] Thanard Kurutach, Ignasi Clavera, Yan Duan, Aviv Tamar, and Pieter Abbeel. 2018. Model-Ensemble Trust-Region Policy Optimization. arXiv:1802.10592 [cs.LG]

[11] Yao Liu, Omer Gottesman, Aniruddh Raghu, Matthieu Komorowski, Aldo Faisal, Finale Doshi-Velez, and Emma Brunskill. 2019. Representation Balancing MDPs for Off-Policy Policy Evaluation. arXiv:1805.09044 [cs.LG]

[12] Hongzi Mao, Shaileshh Bojja Venkatakrishnan, Malte Schwarzkopf, and Mohammad Alizadeh. 2019. Variance Reduction for Reinforcement Learning in Input-Driven Environments. arXiv:1807.02264 [cs.LG]

[13] Judea Pearl. 1995. Causal Diagrams for Empirical Research. *Biometrika* 82, 4 (1995), 669–688. http://www.jstor.org/stable/2337329

[14] Danilo J. Rezende, Ivo Danihelka, George Papamakarios, Nan Rosemary Ke, Ray Jiang, Theophane Weber, Karol Gregor, Hamza Merzic, Fabio Viola, Jane Wang, Jovana Mitrovic, Frederic Besse, Ioannis Antonoglou, and Lars Buesing. 2020. Causally Correct Partial Models for Reinforcement Learning. arXiv:2002.02836 [cs.LG]

[15] Stephane Ross and J. Andrew Bagnell. 2012. Agnostic System Identification for Model-Based Reinforcement Learning. arXiv:1203.1007 [cs.LG]

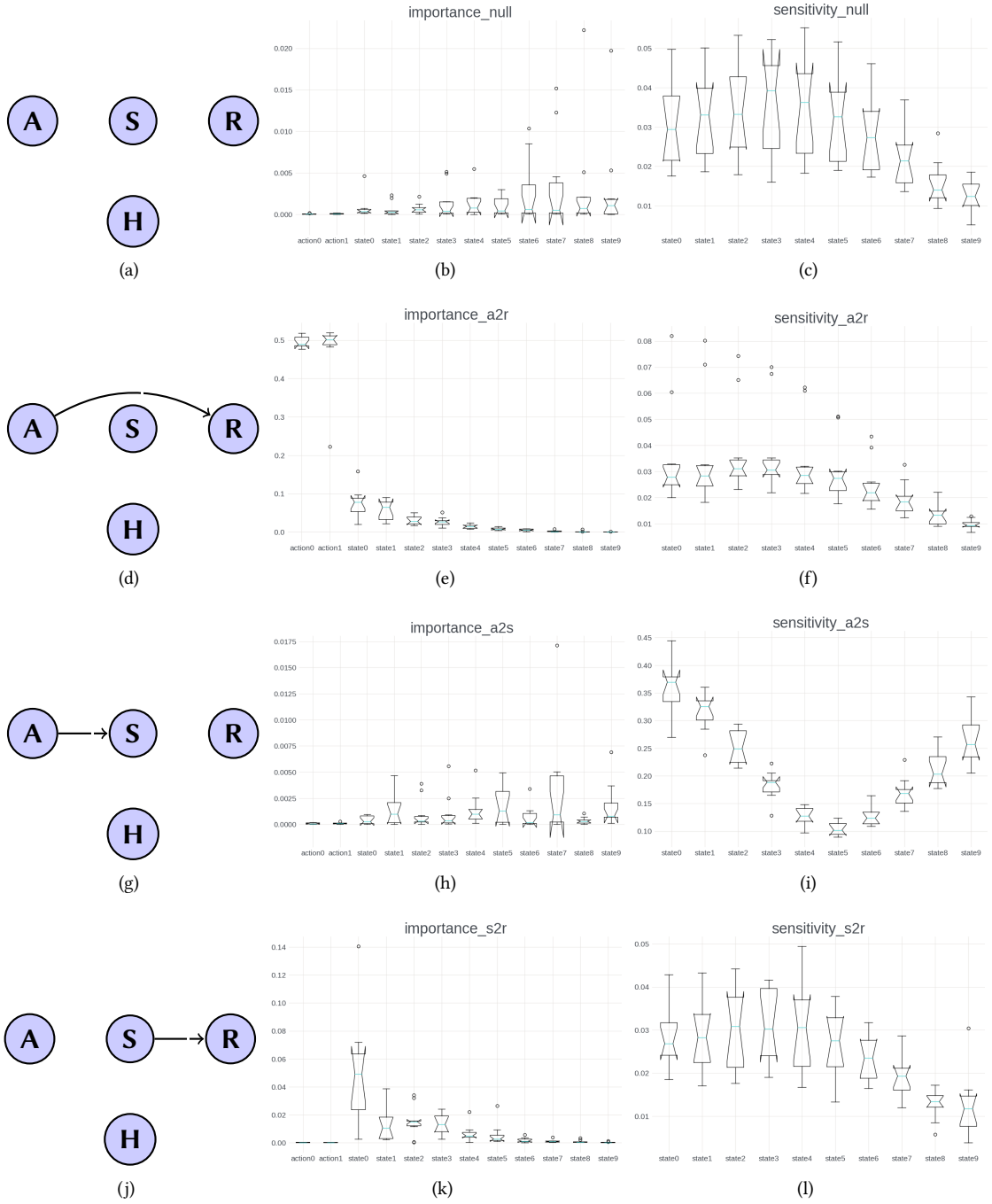
[16] Richard S. Sutton. 1990. Integrated Architectures for Learning, Planning, and Reacting Based on Approximating Dynamic Programming. In *In Proceedings of the Seventh International Conference on Machine Learning*. Morgan Kaufmann, 216–224.

[17] Richard S. Sutton. 1991. Dyna, an Integrated Architecture for Learning, Planning, and Reacting. *SIGART Bull.* 2, 4 (July 1991), 160–163. https://doi.org/10.1145/122344.122377

[18] Richard S. Sutton. 1991. Planning by Incremental Dynamic Programming. In *In Proceedings of the Eighth International Workshop on Machine Learning*. Morgan Kaufmann, 353–357.

[19] Richard S. Sutton and Andrew G. Barto. 1998. *Introduction to Reinforcement Learning* (1st ed.). MIT Press, Cambridge, MA, USA.

[20] Tingwu Wang, Xuchan Bao, Ignasi Clavera, Jerrick Hoang, Yeming Wen, Eric Langlois, Shunshi Zhang, Guodong Zhang, Pieter Abbeel, and Jimmy Ba. 2019. Benchmarking Model-Based Reinforcement Learning. arXiv:1907.02057 [cs.LG]



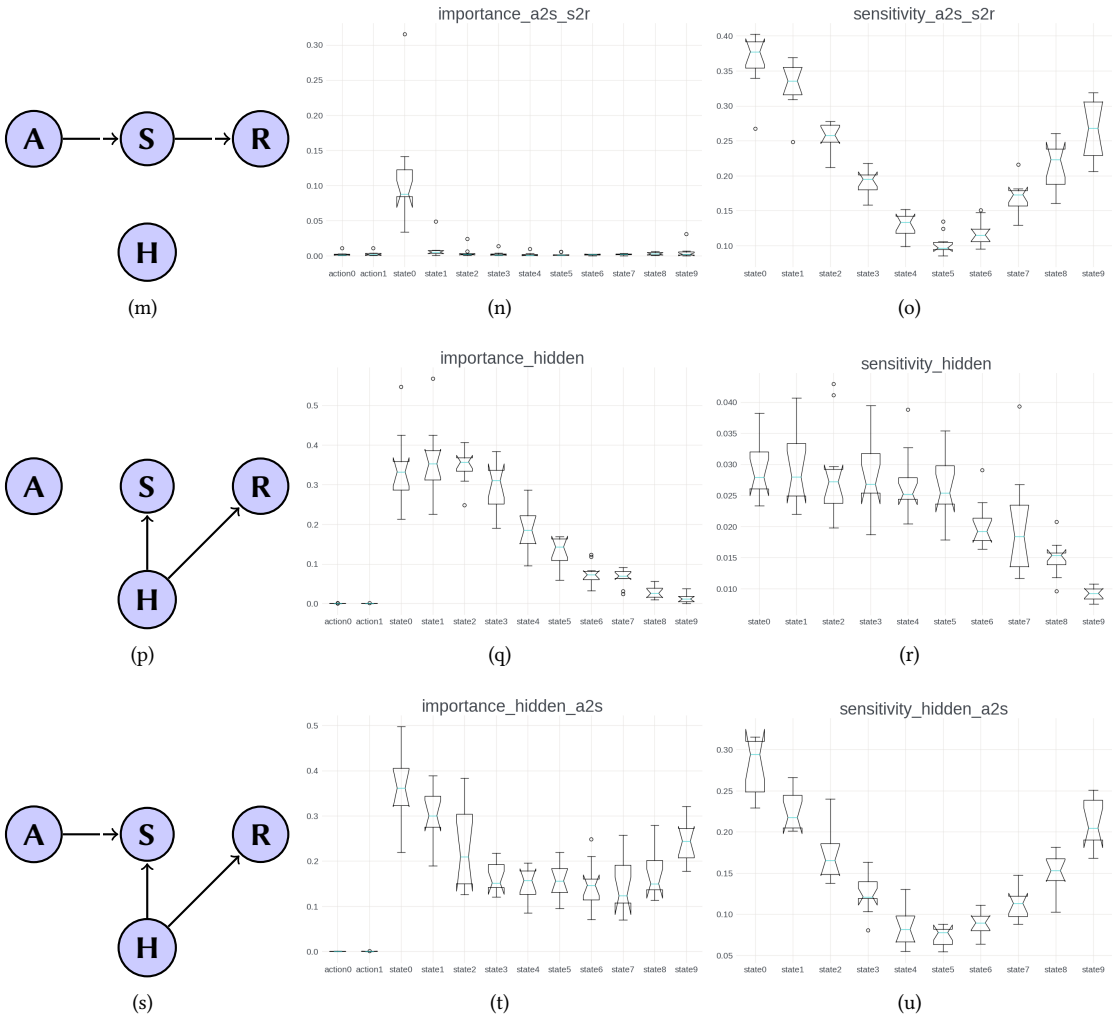


Figure 5. Feature importance and sensitivity measured for different environments listed in section 5.1.