Transfer Learning, Reinforcement Learning for Adaptive Control Optimization under Distribution Shift

Pankaj Rajak
San Deigo, CA, USA
rajapan@amazon.com

Wojciech Kowalinski
Seattle, WA, USA
kowalin.amazon.com

Fei Wang
Seattle, WA, USA
fiwan.amazon.com

Abstract

Many control systems rely on a pipeline of machine learning models and hand-coded rules to make decisions. However, due to changes in the operating environment, these rules require constant tuning to maintain optimal system performance. Reinforcement learning (RL) can automate the online optimization of rules based on incoming data. However, RL requires extensive training data and exploration, which limits its application to new rules or those with sparse data. Here, we propose a transfer learning approach called Learning from Behavior Prior (LBP) to enable fast, sample-efficient RL optimization by transferring knowledge from an expert controller. We demonstrate this approach by optimizing the rule thresholds in a simulated control pipeline across differing operating conditions. Our method converges 5x faster than vanilla RL, with greater robustness to distribution shift between the expert and target environments. LBP reduces negative impacts during live training, enabling automated optimization even for new controllers.

1 Introduction and Literature Review

A control system that integrates Machine Learning (ML) model scores and static rules can be conceptualized as a decision-making process that operates within a defined operational framework. This system leverages the predictive ML models to determine the outcomes and uses static rules to ensure decisions adhere to pre-defined guidelines or constraints. At its core, the system ingests information about the current state of the environment, which is processed by an ML model to yield predictive scores indicating the likelihood of future states or events. Concurrently, a set of static rules, embodying domain expertise and regulatory compliance, delineates the bounds of permissible actions, Figure 1. This type of control system is widely used in advertise ranking, content recommendation, bidding, stock trading and in linear dynamical systems Zhang et al. [2016] Yang et al. [2016] Ye et al. [2020] Lale et al. [2020] Jambor et al. [2012]. Traditionally, it is labor insensitive to manually change the bounds of actions defined by the static rules to adapt the fast-changing environment. Reinforcement Learning (RL) has emerged as a powerful framework for solving complex control problems where traditional methods struggle due to uncertainty, non-linearity, or the curse of dimensionality. With RL, one can automate this entire process of dynamic rule optimization by learning the action from feedback and environment through maximizing rewards. However, one challenge with RL is that it requires a large amount of training data as it needs to solve exploration-exploitation problem as part of its policy learning process. Thus, it is not possible to use RL in the cold start scenarios or domains of data sparsity. Further, for many real world problems such as health care or autonomous driving, RL agent training with risky action during its early stage of random exploration can leads to disastrous outcomes.

Transfer learning (TL) is one approach to solve the cold start and sparse data problems for RL based dynamic rule optimization. TL in RL Zhuangdi et al. [2020] utilizes external expertise from expert

Workshop on Distribution Shifts, 37th Conference on Neural Information Processing Systems (NeurIPS 2023).

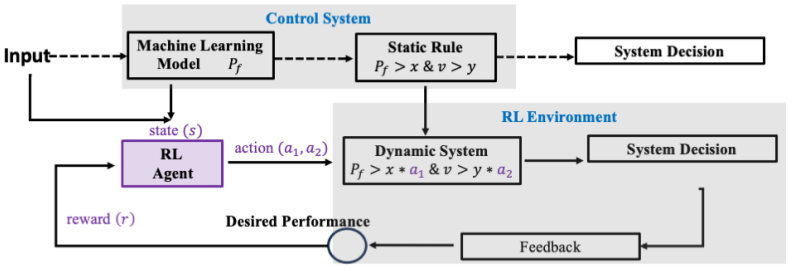

Figure 1: Schematic of the Optimal Control system consisting of ML models and static rules (Top), and reinforcement learning agent assisted dynamic rule for Control system optimization (Bottom.)

domains to benefit the learning process of another target task in a different target domain. Successful application of TL from an expert domain to a target domain can reduce the sample complexity in the target by encouraging it to only try safe actions which are most likely optimal and have no negative impact. However, TL in RL is harder as knowledge transfer needs to be done at the level of Markov decision process. Further, the algorithm needs to consider sub-optimality of the expert demonstration due to the data distribution shift between the expert and the target domain. Another challenge while applying TL in RL is that actions that are feasible in the expert domain may not be possible in the target domain and vice versa. To solve these challenges, many approaches have been proposed, that can be broadly categorized into following groups (a) representation learning, (b) reward shaping (RS) Ng et al. [1999] Wiewiora et al. [2003], (c) policy transfer Parisotto et al. [2016] and (d) learning from demonstration (LFD) for policy gradient $\pi(a|s)$ and Q-learning algorithms. An example of LFD in Q-Learning is Todd et al. [2017] proposed Deep Q-learning from Demonstration (DQfD). Here, it maintains two separate reply buffers, where one contains external demonstration and other agents' policy generated data from its own environment and during training examples are sampled from both buffers. Other examples of LFD in Q-Learning are Brys et al. [2015] Nair et al. [2018]. A representative work on LFD in policy gradient is Ho and Ermo [2016] paper on Generative Adversarial Imitation Learning (GAIL) and Kang et al. [2018] algorithm Policy Optimization from Demonstrations. Here, even though GAIL is similar to imitation learning, it can also be viewed as LFD as it tries to minimize the distribution divergence between the agent's policy and experts policy - provided via demonstration data - in its loss function.

In this work, we propose a new TL approach for RL training, called Learning from Behavior Priors (LBP) for automated dynamic rule optimization. First, a rule or a set of rules is chosen as an expert, whose policy data is used to create an expert demonstration in the form of expert variational auto-encoder prior. These expert VAE are later used for knowledge transfer during the RL training phase of another target rule. Thus, using LBP, we can solve the following problems: 1) cold start for rules with low volume, 2) reduce sample complexity, and 3) slow learning. We also compared our proposed method with other TL techniques in RL, like RS, LFD and representation learning, and show that our method converges faster in comparison to others and better at handling sub-optimal demonstration.

2 Methods

RL Agent Design for Adaptive Online Rule Optimization: A schematic of our RL agent for online rule optimization is shown in Figure 1. Here, the agent's input (state, s_t) is constructed using hourly incoming events to the rule. Specifically, s_t contains aggregated statistics of hourly incoming events such as total volume, event volume triggered using current rule expression and mean and median values of (i) numeric features of the events and (ii) the predictive score of events. Hence, the input to the agent is partially observed as it does not look at each event individually but at macro level using hourly aggregated statistics. Using (s_t), the agent first proposes an action (a_i) in a normalized space of $[-1, 1]$, which is then transformed within user defined range $[a_{min}, a_{max}]$. Here, the size of $|a_i|$ is equal to the number of tunable rule conditions. For example, in Figure 1, we have two such segments and actions (a_1, a_2), which are multiplied with the original threshold values of the rule to create a new condition $P_f > x * a_1 \;\&\; v > y * a_2$, Figure 1. This new dynamic rule makes the system decision on the incoming event. Later, the agent receives feedback from the environment, which is converted into a scalar reward r using equation 1. Along with r agent also receives s_{t+1}

computed using updated rule.

$$r = \frac{1}{|FPR_{observed} - FPR_{desired}| + \epsilon} \tag{1}$$

In equation 1, FPR is false positive rate, expressed as a proportion of sub-optimum actions. Here, $FPR_{desired}$ that user wants to achieve and $FPR_{observed}$ is computed using the feedback from the environment. ϵ is a small positive number added for numerical stability.

Expert Agent training using BCQ We use offline RL learning algorithm: Batch Constraint deep Q-learning (BCQ) Fujimoto and amd Doina Precup [2019] to train expert agent using historic dataset. We did not use online RL algorithms because they can take random exploratory actions that can have a negative impact under extreme condition. BCQ is an actor-critic based agent that uses a conditional variational auto-encoder (CVAE) to make sure the agent's state-action distribution matches with a provided offline training dataset (see supplementary material for pseudocode 2).

Transfer Learning (TL): Learning behavior prior (LBP) In our proposed LBR TL method, we do the knowledge sharing from the expert domain (rule) to the target domain (rule) through a set of expert CVAEs. First, RL agents using BCQ algorithm is trained in one or multiple expert domain. After training, we get CVAE (D^e) from expert that generates a distribution of optimal action for input state, $P(a_i|s_i)$. Later, we train the RL agent in the target domain using a modified version of BCQ 2 3, expert VAE $[D^e]$ and some historic dataset (B) in the target domain.

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha[r + \gamma \max_{a' \, s.t. (s',a') \in (B \cup [\mathbf{D^e(a'|s')]_{i=1}^n})} Q(s^{'}, a^{'})] \tag{2}$$

$$\phi \leftarrow \arg\max_{\phi} \sum_{(s,a); s \in B, \mathbf{a} \in \max_{\mathbf{Q(s,a)}} (\mathbf{B} \cup [\mathbf{D^e(a|s)]_{i=1}^n})} Q[s, a + \xi_\phi(s, a, \Phi)] \tag{3}$$

The difference between original BCQ and our proposed algorithm is the presence of D_e inside max in critic $(Q(s,a))$ and max operator itself in the actor $(\phi(a|s))$ (highlighted in bold). This helps 2 and 3 to handle the sub-optimal demonstration from expert as during such cases agent will ignore expert's recommendation and will use information from the dataset B. For example, $Q(s,a)$ training uses information from both D^e and B to find $a^{'}$ that has the highest Q to solve the bellman equation, 2. Similarly, for every $s \in B$, an actor ϕ looks at all the possible actions for s that is in B and proposed by D^e and takes the one action that has maximum Q as its policy $\phi(a|s)$. In 3, $\xi_\phi(s,a,\Phi)$ is the perturbation model so as to optimally perturb a within allowed range $[-\Phi, \Phi]$ near a. Finally, the target agent also maintains its own VAE that is trained on the fly using B and D^e,and thus learns to take the best action using multiple sources. This is done by training this VAE using target agent's reply buffer, which is filled during training with (s,a) with s sampled from B and a computed using equation 3. A pseudocode of our algorithm is in supplementary material at 1.

3 Results

For our experiment we use the following rule expressions as expert and target, where y_i is event feature, x_i is the ML predictive score and p_i is manually optimized static threshold.

$Expert\ Rule : (\mathbf{x2} > \mathbf{p1} * \mathbf{a_i})\ \&\ (y2 > p2)$

$Target\ Rule : (\mathbf{x2} > \mathbf{p3} * \mathbf{a_i})\ \&\ (y2 \leq p2)$

The above rules are designed such that expert and target rule have significant data distribution shift introduced by the condition clause $(y2 > p2)$ in the expert and $(y2 \leq p2)$ in the target rule. The expert and target conditions are mutually exclusive. The rule condition modified by the RL agent rule segment is highlighted in bold, where RL action is a_i and x_2 is a continuous predictive score ranging between 0-1. Ideally, we can choose any rule condition for RL optimization whose threshold value is highly sensitive to the rule performance.

During training, an expert agent is trained with $FPR_{desired} = 0.85$ and for target $FPR_{desired} = 0.75$ is used. Table 1 shows the performance of target agent using vanilla BCQ agent without TL,

Table 1: Mean FPR achieved by target rule over the course of training steps (1 episode = 1000 time steps). Here, desired FPR is 0.75, and FPR of static rule without RL agent is 0.915

TL Method	1 episode	5 episode	25 episode
RL (BCQ) but without TL	0.601	0.613	0.865
Reward Shaping	0.604	0.803	0.624
Representation Learning	0.611	0.913	0.971
Learning From Demonstration	0.868	0.788	0.766
Learning Behavior Prior	**0.757**	**0.754**	**0.753**

LBR and comparison of LBR w.r.t to other TL methods, including representation learning, reward shaping (RS) and learning from demonstration (LFD). Details of these methods are in supplementary materials in baseline TL models, Algorithm 2 and Figure 3. We can observe all algorithms move closer to its desired FPR of 0.75 as the training time increase. However, their overall performance variation can be explained by their internal learning mechanism. For example, (a) BCQ without TL and (b) BCQ with Reward Shaping, random offline samples $[(s_i, a_i, r_i, s_{i+1})]$ are provided as the initial reply buffer data for training, which may not contain any optimal actions. Since BCQ is an offline RL algorithm without exploration, its performance dependents highly upon the quality of the reply buffer data. This can cause lower performance if the buffer is not sufficiently large or mostly consists of sub-optimal actions. Similarly, RS tells which (s_i, a_i) pairs might have high reward, but it won't be helpful if the initial reply buffer data only contains suboptimal state-action pairs. For representation learning, features learned in the expert domain can become too specific to its domain and do not generalize well in the target domain, especially when there is a significant data distribution mismatch between expert and target. Only LFD and LBP are able to reach the desired performance. However, LBP converged faster to the desired performance in a single episode, whereas LFD took 25 episodes. We demonstrated this difference through the Q-value of state-action distribution over training steps, Figure 2. LBP agent policy converges when its Q-value becomes constant and does not change over time. LFD overestimate Q-value during initial training steps, which also resulted in its high FPR rate in single training episode. This is because of that expert's knowledge for LFD is provided via a reply buffer (s_i, a_i, r_i, s_{i+1}) that also contains expert's reward for every state-action. In the target domain, the agent may not receive the same reward. This data distribution mismatch between expert and target domain leads to inaccurate estimation of Q-value during the initial training steps, resulting in longer training time to correct sub-optimality introduced by incorrect reward from the expert. On the other hand, LBP provides expert knowledge through a state-action distribution and lets the target domain access the associated reward for those (s, a) pair in its environment. Thus, LBR does not inherit such estimation bias from the expert that leads to faster convergence. For additional experiment when expert and target rules are similar, see supplementary section additional experiments. Here also, LBP converges faster to the desired FPR.

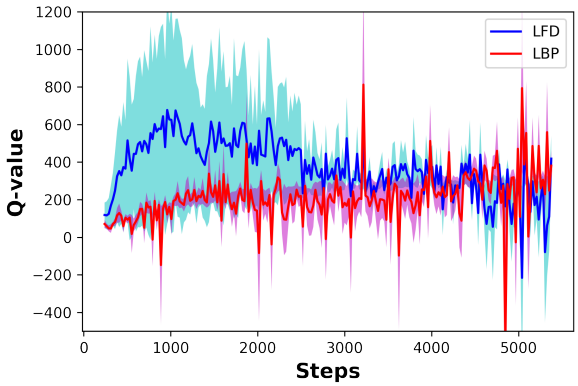

Figure 2: Q-Value of LFD and LBP as a function of training time (1 episode = 1000 step) in target

4 Conclusions

We propose a transfer learning model called Learning Behavior Prior (LBP) for fast and sample efficient reinforcement learning automated optimization of rules in an optimal control system. We also show that our proposed method LBR shows faster convergence to the optimal policy and desired performance in comparison to other TL methods when distribution mismatch exists between expert and target domains. Further, our proposed method is generic and can be applied to wide range of RL problems such as in robotics, healthcare and autonomous driving.

References

Weinan Zhang, Yifei Rong, Jun Wang, Tianchi Zhu, and Xiaofan Wang. Feedback control of real-time display advertising. *arXiv preprint arXiv:1603.01055*, 2016.

Xun Yang, Yasong Li, Hao Wang, Di Wu, Qing Tan, Jian Xu, and Kun Gai. Bid optimization by multivariable control in display advertising. *arXiv preprint arXiv:1905.10928*, 2016.

Yujian Ye, Dawei Qiu, Mingyang Sun, Dimitrios Papadaskalopoulos, and Goran Strbac. Deep reinforcement learning for strategic bidding in electricity markets. *IEEE Transactions on Smart Grid*, 2020. doi: 10.1109/TSG.2019.2936142.

Sahin Lale, Kamyar Azizzadenesheli, Babak Hassibi, and Anima Anandkumar. Reinforcement learning with fast stabilization in linear dynamical systems. *arXiv preprint arXiv:2007.12291*, 2020.

Tamas Jambor, Jun Wang, and Neal Lathia. Using control theory for stable and efficient recommender systems. *arXiv preprint arXiv:2007.12291*, 2012. doi: 10.1145/2187836.2187839. URL `https://doi.org/10.1145/2187836.2187839`.

Zhu Zhuangdi, Lin Kaixiang, K. Jain Anil, and Zhou Jiayu. Transfer learning in deep reinforcement learning: A survey. *arXiv preprint arXiv:2009.07888*, 2020.

Andrew Y. Ng, Daishi Harada, and Stuart J. Russell. Policy invariance under reward transformations: Theory and application to reward shaping. *ICML*, 1999.

E. Wiewiora, G. W. Cottrell, , and C. Elkan. Principled methods for advising reinforcement learning agent. *ICML*, 2003.

Emilio Parisotto, Jimmy Lei Ba, and Ruslan Salakhutdinov. Actor-mimic: Deep multitask and transfer reinforcement learning. *arXiv preprint arXiv:1511.06342*, 2016.

Hester Todd, Vecerik Matej, Olivierm Pietquin, Lanctot Marc, Schaul Tom, Piot Bilal, Horgan Dan, Quan John, Sendonaris Andrew, Dulac-Arnold Gabriel, Osband Ian, Leibo John, Agapiouand Joel, and Gruslys Audrunas. Deep q-learning from demonstrations. *arXiv preprint arXiv:1704.03732*, 2017.

T. Brys, A. Harutyunyan, H. B. Suay, M. E. Taylor S. Chernova, and A. Nowe. Reinforcement learning from demonstration through shaping. *International Joint Conference on Artificial Intelligence*, 2015.

A. Nair, B. McGrew, M. Andrychowicz, and and P. Abbee W. Zaremba. Overcoming exploration in reinforcement learning with demonstration. *ICRA*, 2018.

J. Ho and S. Ermo. Generative adversarial imitation learning. *NeurIPS*, 2016.

B. Kang, Z. Jie, and J. Fen. Policy optimization withdemonstrations. *ICML*, 2018.

Scott Fujimoto and David Meger amd Doina Precup. Off-policy deep reinforcement learning without exploration. *arXiv preprint arXiv:1812.02900*, 2019.

Supplementary Material

Algorithm 1: RL Agent- BCQ with Expert VAE prior (Learning Behavior Prior).

Input:
Historic Dataset in the Target Domain: B days, target rule expression: R_T, performance metric: M, time-interval between consecutive steps in RL environment: h hours, total time-step in env_t: $T = \frac{24B}{h}$, Expert VAE model: $[D^e]_{i=1}^n$, min-batch size: N, max perturbation: ϕ
Target Environment: env_t
Create env_t consisting of R_T and B sorted by time
RL agent initialization: BCQ_t
Q-networks : $Q_{\theta_1}, Q_{\theta_2}$
Actor (perturbation network): ξ_ϕ
VAE: $G_w = [E_{w_i}, D_{w_i}]$
Train BCQ_t**:**
Create empty buffer $B^{'}$ and add N examples using env_t and $[D^e]_{i=1}^n$
for *episode* $e = 0, \ldots, E$ **do**
  $s_0 = env_t.reset()$
  for *timestep* $t = 0, \ldots, T$ **do**
    Sample mini-batch of N transitions $(s, a, s^{'}, r)$ from B'
    `/* train target VAE:` G_w `*/`
    $\mu, \sigma = E_{w_i}(s, a); \hat{a} = D_{w_i}(s, z); z \sim \mathcal{N}(\mu, \sigma)$
    Sample n actions : $a_i \sim G_w(s^{'}) \cup [D^e(s^{'})]_{i=1}^n$
    Perturb each actions : $a_i = a_i + \xi_\phi(s^{'}, a, \phi))$
    `/* Calculate target y:` `*/`
    $y = r + \gamma \max_{a_i}[\lambda \min_{j=1,2} Q_{\theta_j}(s^{'}, a_i) + (1-\lambda)\max_{j=1,2} Q_{\theta_j}(s^{'}, a_i)]$
    $\theta \leftarrow \arg\min_\theta \sum (y - Q_\theta(s,a))^2$
    $\phi \leftarrow \arg\max_\phi \sum Q_\theta(s, a + \xi_\phi(s, a, \phi)), a \sim \max_{Q_\theta(s,a)}[G_w(s^{'}) \cup [D^e(s^{'})]_{i=1}^n]$
    Update Q network: $\theta_i^{'} \leftarrow \tau\theta + (1-\tau)\theta_i^{'}$
    Update Actor: $\phi_i^{'} \leftarrow \tau\phi + (1-\tau)\phi_i^{'}$;
    `/* Sample next timestep from` env_t `and add to` $B^{'}$ `*/`
    $a = BCQ_t.sample\_action(s_0)$
    $s_{next}, r = env_t.step(a)$
    $B^{'}.add(s_0, a, s_{next}, t)$
    $s_0 = s_{next}$

5 Baseline TL Methods

A schematic of different TL approaches in RL is shown in Figure 3, which are reward shaping (RS), representation learning, learning from demonstration (LFD) and our proposed method learning behavior prior (LBP). Implementation details for LBP is in Algorithm 1 and in appendix Algorithm 2 for others TL methods. From 3 and pseudocode, we can observe that different TL method uses different approach to transfer knowledge from expert to the target domain. LFD simply initializes the buffer of the target agent at the beginning of training with expert data that consists of tuples of (s_i, a_i, r_i, s_{i+1}) generated from expert policy. In the case of representation learning, expert RL agents actor and critic's network weight (except for the last layer) is used to initialize the target agents actor and critic's network. Finally, in RS, expert's critic network $Q^e(s, a)$ is used to construct an auxiliary function F as shown in 4 to modify the reward received by the target agent in the target domain 5. However, F is difficult to compute because during every interaction between RL agent and environment we get tuple like (s_i, a_i, r_i, s_{i+1}). Thus, we don't have the a_{i+1} for every s_{i+1} until the end of the episode, which can be very long. In our work, for fast calculation of F, we thus make a possible prediction of a_{i+1} for s_{i+1} using target agent as shown in the appendix in Algorithm 2.

$$F(S_i, S_{i+1}, a_i, a_{i+1}) = \gamma\phi(S_{i+1}, a_{i+1}) - \phi(S_i, a_i) \tag{4}$$

$$R^{'}(S_i, S_{i+1}, a_i) = r(S_i, S_{i+1}, a_i) + F(S_i, S_{i+1}, a_i) \tag{5}$$

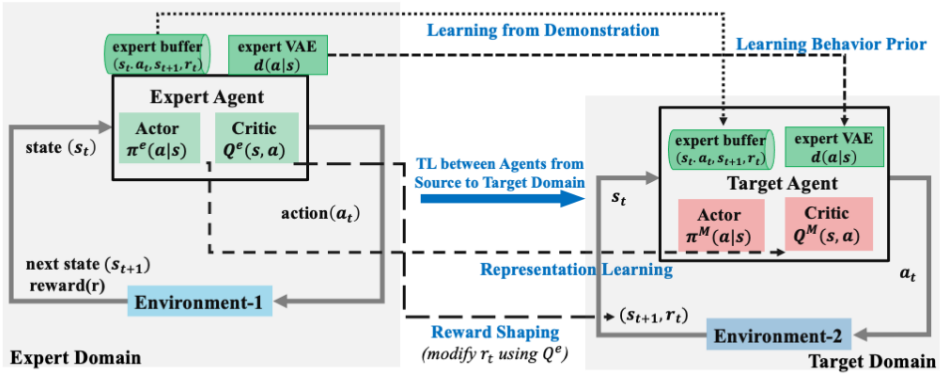

Figure 3: Schematic of TL approaches for knowledge transfer between RL agents from an expert domain (expert agent) to target domain (target agent.)

6 Additional Experiment

We show below results for another experiment on different type of rule condition. The expression of expert and target rule used in this experiment is shown below, where we can observe that expert and target rule are similar. In fact, a subset of input transaction fired by expert rule will also satisfy target rule.

$Expert\ Rule: (\mathbf{x1} * \mathbf{v1} > \mathbf{m1} * \mathbf{action_i}) \ || \ (x2 * v1 > m2) \ || \ (v2 > m3))$

$Target\ Rule: (\mathbf{x1} * \mathbf{v1} > \mathbf{m4} * \mathbf{action_i}) \ \& \ (v3 > m5) \ \& \ (v4 > m6)$

The results of this experiment is shown in Table 2. Here, again we observe that LBP converges faster to the desired performance. Representation Learning also converged closer to the desired performance within 1 episode. This has happened here because both expert and target rules are similar, and thus experts features are easily transferable in the target domain.

Algorithm 2: Baseline RL agent with/without TL

Input:

Historic Dataset: B days, Rule expression: R_T, performance metric: M, time-interval between consecutive steps in RL environment: h hours, total time-step in env_t: $T = \frac{24B}{h}$, min-batch size: N, max perturbation: ϕ

Target Environment: env_t

Create target environment env_t consisting of R_T and B sorted by time

RL agent initialization: BCQ

Q-networks : $Q_{\theta_1}, Q_{\theta_2}$

Actor (perturbation network): ξ_ϕ

VAE: $G_w = [E_{w_i}, D_{w_i}]$

if *Representation Learning* == *True* **then**

  $Q_{\theta_1}, Q_{\theta_2} \leftarrow \mathrm{Q}^e_\theta$

  $\xi_\phi \leftarrow \xi^e_\phi$

if *Learning from Demonstration* == *True* **then**

  Buffer B$^{'}$ $\leftarrow Expert\ demonstration$

else

  Initialize Buffer B$^{'}$ $\leftarrow Random\ Samples\ from$ B

BCQ Training:

for *episode* $e = 0, \ldots, E$ **do**

  $s_0 = env_t.reset()$

  for *timestep* $t = 0, \ldots, T$ **do**

    Sample mini-batch of N transitions $(s, a, r, s^{'})$ from B'

    `/* train VAE:` G_w `*/`

    $\mu, \sigma = E_{w_i}(s, a); \hat{a} = D_{w_i}(s, z); z \sim \mathcal{N}(\mu, \sigma)$

    Sample n actions : $a_i \sim G_w(s^{'})$

    Perturb each actions : $a_i = a_i + \xi_\phi(s^{'}, a, \phi))$

    `/* Calculate target y:` `*/`

    $y = r + \gamma \max_{a_i}[\lambda \min_{j=1,2} Q_{\theta_j}(s^{'}, a_i) + (1-\lambda)\max_{j=1,2} Q_{\theta_j}(s^{'}, a_i)]$

    $\theta \leftarrow \arg\min_\theta \sum (y - Q_\theta(s,a))^2$

    $\phi \leftarrow \arg\max_\phi \sum Q_\theta(s, a + \xi_\phi(s, a, \phi)), a \sim G_w(s)$

    Update Q network: $\theta^{'}_i \leftarrow \tau\theta + (1-\tau)\theta^{'}_i$

    Update Actor: $\phi^{'}_i \leftarrow \tau\phi + (1-\tau)\phi^{'}_i$;

    `/* Sample next timestep from` env_t `and add to` $B^{'}$ `*/`

    a = BCQ.sample$_action$(s$_0$)

    s_{next}, r = env$_t$.$step(a)$

    if *Reward Shaping* == *True* **then**

      a$_{next}$ $= BCQ.sample\_action(s_{next})$

      r=r+F(s,a,s$_{next}$, a_{next})

    $B^{'}.add(s_0, a, s_{next}, r)$

    $s_0 = s_{next}$

Table 2: Mean FPR achieved by target rule over the course of training steps (1 episode = 1000 time steps). Here, desired FPR is 0.30, and FPR of static rule without RL agent is 0.477

TL Method	1 episode	5 episode	25 episode
RL (BCQ) but without TL	0.729	0.730	0.409
Reward Shaping	0.644	0.362	0.298
Representation Learning	0.294	0.234	0.232
Learning From Demonstration	0.398	0.234	0.227
Learning Behavior Prior	**0.235**	**0.235**	**0.235**