RAMBO-RL: Robust Adversarial Model-Based Offline Reinforcement Learning

Marc Rigter, Bruno Lacerda, Nick Hawes
Presented by: Anirud N
CE21B014

1 Introduction to Offline RL

Reinforcement Learning has been the SOTA for various real world applications. However, it is not always legally or safely possible for the agent to extensively explore and it can be prohibited as well. This brings the need for Offline RL - learning from a pre-existing dataset.
Traditional Online RL algorithms work poorly in offline RL due to the distributional shifts between the state action pairs in the dataset and those taken by the learned policy. 1 shows the overestimation in the unknown regions, i.e., those that are not present in the offline dataset but occur in the environment.

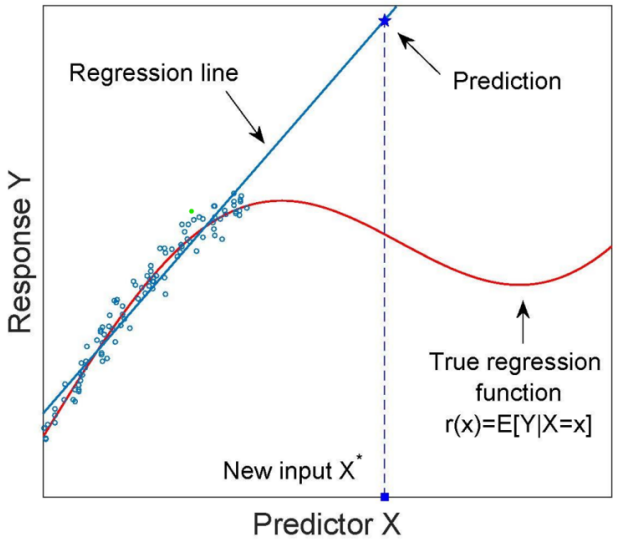


Figure 1: Overestimation in offline datasets

The literature mainly focusses on two different approaches to tackle the above issue.

- Model Free
- Model Based

Model-free algorithms train a policy from only the existing dataset and they either constrain the policy or modify the value function to incorporate conservatism.

Model-based algorithms use the dataset to learn a model for the environment. This model generates a synthetic dataset - so that we have more data to train on. The model aims to generalise beyond the states present in the dataset. To incorporate conservatism, it tries to estimate the uncertainty in the model. However, this could be unreliable for neural networks.

2 Intro to RARL and RAMBO

This paper avoids any "Uncertainty Estimation". It takes inspiration from the Robust Adversarial Reinforcement Learning Paper Fig.2. RARL algorithm basically has 2 policies to learned - one is

Algorithm 1 RARL (proposed algorithm)

Input: Environment $\mathcal{E}$; Stochastic policies μ and ν
Initialize: Learnable parameters θ_0^μ for μ and θ_0^ν for ν
for i=1,2,..N_{iter} **do**
  $\theta_i^\mu \leftarrow \theta_{i-1}^\mu$
  for j=1,2,..N_μ **do**
    $\{(s_t^i, a_t^{1i}, a_t^{2i}, r_t^{1i}, r_t^{2i})\} \leftarrow \text{roll}(\mathcal{E}, \mu_{\theta_i^\mu}, \nu_{\theta_{i-1}^\nu}, N_{\text{traj}})$
    $\theta_i^\mu \leftarrow \text{policyOptimizer}(\{(s_t^i, a_t^{1i}, r_t^{1i})\}, \mu, \theta_i^\mu)$
  end for
  $\theta_i^\nu \leftarrow \theta_{i-1}^\nu$
  for j=1,2,..N_ν **do**
    $\{(s_t^i, a_t^{1i}, a_t^{2i}, r_t^{1i}, r_t^{2i})\} \leftarrow \text{roll}(\mathcal{E}, \mu_{\theta_i^\mu}, \nu_{\theta_i^\nu}, N_{\text{traj}})$
    $\theta_i^\nu \leftarrow \text{policyOptimizer}(\{(s_t^i, a_t^{2i}, r_t^{2i})\}, \nu, \theta_i^\nu)$
  end for
end for
Return: $\theta_{N_{\text{iter}}}^\mu, \theta_{N_{\text{iter}}}^\nu$

Figure 2: RARL

the hero policy, and the second is the villain policy. The hero policy is the agent, which tries to maximize the rewards, whereas the villain policy aims for the hero to fail. It gets rewarded whenever the hero fails. Reward $r_t = r(s_t, a_t^1, a_t^2)$ is obtained from the environment. In our zero-sum game, the protagonist gets a reward $r_t^1 = r_t$ while the adversary gets a reward $r_t^2 = -r_t$. Hence each step of this MDP can be represented as $(s_t, a_t^1, a_t^2, r_t^1, r_t^2, s_{t+1})$.

The protagonist seeks to maximize the following reward function,

$$R^1 = \mathbb{E}_{s_0 \sim \rho, a_1 \sim \mu(s), a_2 \sim \nu(s)} \left(\sum_{t=0}^{T-1} r^1(s, a^1, a^2) \right).$$

Since the policies μ and ν are the only learnable components, $R^1 = R^1(\mu, \nu)$. Similarly, the adversary attempts to maximize its own reward: $R^2 = R^2(\mu, \nu) = -R^1(\mu, \nu)$.

$$R^{1*} = \min_\nu \max_\mu R^1(\mu, \nu) = \max_\mu \min_\nu R^1(\mu, \nu).$$

The difference in ROMBO is that, instead of having a villain policy, it has a villain Transition Dynamics. This model for transition dynamics is adversarially trained.

3 Problem Formulation

$$\pi = \arg\max_{\pi \in \Pi} \min_{\hat{T} \in \mathcal{M}_D} V_{\hat{T}}^\pi, \text{ where} \tag{1}$$

$$\mathcal{M}_D = \left\{ \hat{T} \,\middle|\, \mathbb{E}_D \left[\text{TV}(\hat{T}_{\text{MLE}}(\cdot \mid s, a), \hat{T}(\cdot \mid s, a))^2 \right] \leq \xi \right\} \tag{2}$$

$\hat{T}$ denotes both transition dynamics and rewards. $V_{\hat{T}}^\pi$ is the value function corresponding to policy π under transition dynamics $\hat{T}$. $\mathcal{M}_D$ refers to the set of models for the transition dynamics . Equation (2) ensures that the $\hat{T}$ that we choose to find the optimal policy on does not differ a lot from the actual environment transition dynamics.
Note that this is completely different, as MOPO and MoRel only modify the reward function by giving penalties with state action pairs that have high uncertainity.

4 Theoretical Motivation

Denote the true MDP transition function by T, and let $\mathcal{M}$ denote a hypothesis class of MDP models such that $T \in \mathcal{M}$. Let π denote the solution to Problem in the previous section for dataset $\mathcal{D}$. Then with probability $1-\delta$ for any policy, $\pi^* \in \Pi$, we have

$$V_T^{\pi^*} - V_T^{\pi} \leq (1-\gamma)^{-2} c_1 \sqrt{C_{\pi^*}} \sqrt{G_{\mathcal{M}_1} + G_{\mathcal{M}_2} + \xi_n^2 + \frac{\ln(c/\delta)}{|\mathcal{D}|}}, \text{ where}$$

$$C_{\pi^*} = \max_{T' \in \mathcal{M}} \frac{\mathbb{E}_{(s,a)\sim \tilde{d}_T^{\pi^*}} \left[\mathrm{TV}(T'(\cdot|s,a), T(\cdot|s,a))^2 \right]}{\mathbb{E}_{(s,a)\sim\rho} \left[\mathrm{TV}(T'(\cdot|s,a), T(\cdot|s,a))^2 \right]},$$

where ρ is the state-action distribution from which $\mathcal{D}$ was sampled, and c and c_1 are universal constants.

The quantity C_{π^*} is upper bounded by the maximum density ratio between the comparator policy, π^*, and the offline distribution, i.e. $C_{\pi^*} \leq \max_{(s,a)} \tilde{d}_T^{\pi^*}(s,a)/\rho(s,a)$. It represents the discrepancy between the distribution of data in the dataset compared to the visitation distribution of policy π^*. This shows that if we find a policy by solving Problem in the previous section, the performance gap of that policy is bounded with respect to any other policy π^* that has a state-action distribution which is covered by the dataset.

Furthermore, the value function under the worst-case model in the set defined by Problem **??** is a lower bound on the value function in the true environment.

Let T denote the true transition function for some MDP, and let $\mathcal{M}_{\mathcal{D}}$ be the set of MDP models defined in (2). Then for any policy π, with probability $1-\delta$ we have that

$$\min_{\widehat{T} \in \mathcal{M}_{\mathcal{D}}} V_{\widehat{T}}^{\pi} \leq V_T^{\pi}.$$

5 RAMBO - RL Algorithmic Framework

Algorithm 1 RAMBO-RL

Require: Normalised dataset, $\mathcal{D}$;

1: $\widehat{T}_\phi \leftarrow$ MLE dynamics model.
2: **for** $i = 1, 2, \ldots, n_{\text{iter}}$ **do**
3: Generate synthetic k-step rollouts. Add transition data to $\mathcal{D}_{\widehat{T}_\phi}$.
4: *Agent update*: Update π and Q_ϕ^π with an actor critic algorithm, using samples from $\mathcal{D} \cup \mathcal{D}_{\widehat{T}_\phi}$.
5: *Adversarial model update*: Update $\widehat{T}_\phi$ according to Eq. 9, using samples from $\mathcal{D}$ for the MLE component, and the current critic Q_ϕ^π and synthetic data sampled from π and $\widehat{T}_\phi$ for the adversarial component.

Figure 3: RAMBO Algorithm

3 shows the overall algorithm. Lets go over it one by one. Note that the dataset is normalised - ie each features in the state has been subtracted by the mean and divided by the standard deviation - for easier learning during training. Soft Actor Critic is used for agent training. We represent the dynamics model using an ensemble of neural networks. Each neural network produces a Gaussian distribution over the next state and reward:

$$\hat{T}_\phi(s', r|s, a) = \mathcal{N}(\mu_\phi(s,a), \Sigma_\phi(s,a)). \tag{3}$$

5.1 Model Gradient

The concept of model gradient is very similar to the policy gradient approach. As a result, even the equation is very similar. Let ϕ denote the parameters of a parametric MDP model $\hat{T}_\phi$, and let V_ϕ^π denote the value function for policy π in $\hat{T}_\phi$. Then:

$$\nabla_\phi V_\phi^\pi = \mathbb{E}_{s\sim d_\phi^\pi, a\sim\pi_\phi, (s',r)\sim\hat{T}_\phi}\left[(r + \gamma V_\phi^\pi(s')) \cdot \nabla_\phi \log \hat{T}_\phi(s', r|s, a)\right] \tag{4}$$

$$\nabla_\phi V_\phi^\pi = \mathbb{E}_{s\sim d_\phi^\pi, a\sim\pi_\phi, (s',r)\sim\hat{T}_\phi}\left[(r + \gamma V_\phi^\pi(s') - Q_\phi^\pi(s,a)) \cdot \nabla_\phi \log \hat{T}_\phi(s', r|s, a)\right] \tag{5}$$

Eq(5) is also derived from a similar approach, showing that adding a baseline that is independent of the transition dynamics, i.e., s' and r, helps in learning and it reduces variance. To estimate V_ϕ^π and Q_ϕ^π, we use the critic learnt by the actor-critic algorithm used for policy optimisation. Q_ϕ^π is the expected Value of taking a state s and following the policy thereafter. The advantage term here is normalised to avoid any issues in the model training specifically due to large variations in gradient.

5.2 Adversarial Model Training

This section talks about how do we use model gradient to solve the adversarial training problem. This can be formulated as :

$$\min_{\hat{T}_\phi} V_\phi^\pi, \quad \text{s.t.} \quad \mathbb{E}_\mathcal{D}\left[\mathrm{TV}(\hat{T}_{\mathrm{MLE}}(\cdot|s,a), \hat{T}_\phi(\cdot|s,a))^2\right] \leq \xi. \tag{6}$$

We convert this into an unconstrained optimisation problem using Lagrange multipliers.

$$\max_{\lambda\geq 0}\min_{\hat{T}_\phi} L(\hat{T}_\phi, \lambda) := V_\phi^\pi + \lambda\left(\mathbb{E}_\mathcal{D}\left[\mathrm{TV}(\hat{T}_{\mathrm{MLE}}(\cdot|s,a), \hat{T}_\phi(\cdot|s,a))^2\right] - \xi\right) \tag{7}$$

To facilitate for the easier tuning of the learning rate and for a simplified objective function, this was made into;

$$\mathcal{L}_\phi = \lambda V_\phi^\pi - \mathbb{E}_{(s,a,r,s')\sim\mathcal{D}}\left[\log \hat{T}_\phi(s', r|s, a)\right]. \tag{8}$$

Lesser the lamda - more importance is given to the $\hat{T}$ being close to $\hat{T_{MLE}}$ so that the model fits the dataset accurately, and less weight to the adversarial part. The aim of the adversarial training is to minimise $\mathcal{L}_\phi$

The gradient for this loss function is computed using a simple mini-batch gradient descent, and the gradient for the value term is calculated using Model Gradient as discussed in the previous section.

5.3 K synthetic rollouts

Rollouts are essentially used to generate synthetic data to better represent the environment for more effective training. But the paper: Model based Policy Optimisation discusses the issues of having very large synthetic data steps. Fig 4 shows that the variation increases as the number of steps increases, implying that the errors increase and get added progressively as the number of steps increases. So, it is necessary to set a threshold k for the number of steps while doing rollouts for the synthetic dataset. k is set as hyperparameter.

6 Experiments and Results

Fig 5 shows the results of RAMBO RL as compared to other algorthm for different datasets and different environments.

- RAMBO achieves the best total in MuJoCo
- RAMBO achieves the best overall score in Medium and Medium-replay –

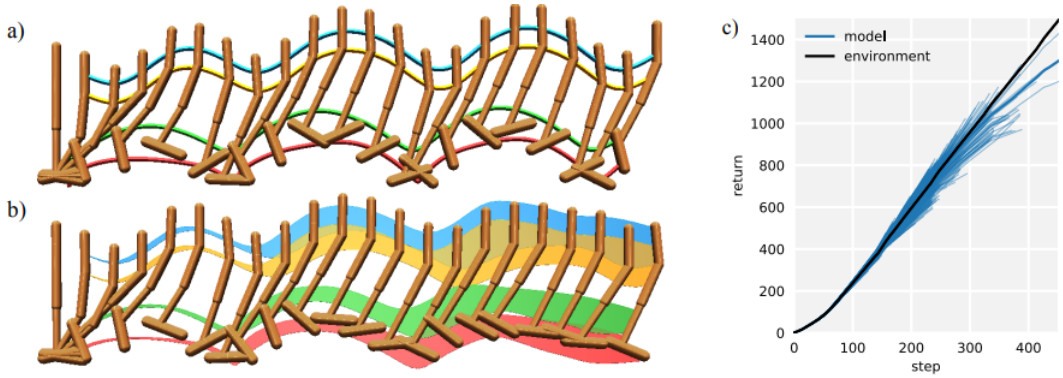


Figure 4: a) A 450-step hopping sequence performed in the real environment, with the trajectory of the body's joints traced through space. b) The same action sequence rolled out under the model 1000 times, with shaded regions corresponding to one standard deviation away from the mean prediction. The growing uncertainty and deterioration of a recognizable sinusoidal motion underscore accumulation of model errors. c) Cumulative returns of the same policy under the model and actual environment dynamics reveal that the policy is not exploiting the learned model. Thin blue lines reflect individual model rollouts and the thick blue line is their mean.

Figure 4: k step

		Ours		Model-based baselines				Model-free baselines			
		RAMBO	**RAMBO^{OFF}**	**RepB-SDE**	**COMBO**	**MOPO**	**MOReL**	**CQL**	**IQL**	**TD3+BC**	**BC**
Random	HalfCheetah	40.0 ± 2.3	33.5 ± 2.6	32.9	38.8	35.4	25.6	19.6	-	11.0	2.1
	Hopper	21.6 ± 8.0	15.5 ± 9.4	8.6	17.9	4.1	53.6	6.7	-	8.5	9.8
	Walker2D	11.5 ± 10.5	0.2 ± 0.6	21.1	7.0	4.2	37.3	2.4	-	1.6	1.6
Medium	HalfCheetah	77.6 ± 1.5	71.0 ± 3.0	49.1	54.2	69.5	42.1	49.0	47.4	48.3	36.1
	Hopper	92.8 ± 6.0	91.2 ± 16.3	34.0	94.9	48.0	95.4	66.6	66.3	59.3	29.0
	Walker2D	86.9 ± 2.7	89.1 ± 2.7	72.1	75.5	-0.2	77.8	83.8	78.3	83.7	6.6
Medium Replay	HalfCheetah	68.9 ± 2.3	67.0 ± 1.5	57.5	55.1	68.2	40.2	47.1	44.2	44.6	38.4
	Hopper	96.6 ± 7.0	97.6 ± 3.4	62.2	73.1	39.1	93.6	97.0	94.7	60.9	11.8
	Walker2D	85.0 ± 15.0	88.5 ± 4.0	49.8	56.0	69.4	49.8	88.2	73.9	81.8	11.3
Medium Expert	HalfCheetah	93.7 ± 10.5	79.3 ± 2.9	55.4	90.0	72.7	53.3	90.8	86.7	90.7	35.8
	Hopper	83.3 ± 9.1	89.5 ± 11.1	82.6	111.1	3.3	108.7	106.8	91.5	98.0	111.9
	Walker2D	68.3 ± 20.6	63.1 ± 31.3	88.8	96.1	-0.3	95.6	109.4	109.6	110.1	6.4
MuJoCo-v2 Total:		826.2 ± 33.8	785.5 ± 40.4	614.1	769.7	413.4	773.0	767.4	692.6*	698.5	300.8
AntMaze	Umaze	25.0 ± 12.0	23.8 ± 15.0	0.0	80.3	0.0	0.0	74.0	87.5	78.6	65.0
	Medium-Play	16.4 ± 17.9	5.6 ± 10.9	0.0	0.0	0.0	0.0	61.2	71.2	3.0	0.0
	Large-Play	0.0 ± 0.0	0.0 ± 0.0	0.0	0.0	0.0	0.0	15.8	39.6	0.0	0.0
	Umaze-Diverse	0.0 ± 0.0	0.0 ± 0.0	0.0	57.3	0.0	0.0	84.0	62.2	71.4	55.0
	Medium-Diverse	23.2 ± 14.2	8.4 ± 9.9	0.0	0.0	0.0	0.0	53.7	70.0	10.6	0.0
	Large-Diverse	2.4 ± 3.3	0.0 ± 0.0	0.0	0.0	0.0	0.0	14.9	47.5	0.2	0.0
AntMaze-v0 Total:		67.0 ± 14.9	37.8 ± 12.4	0.0	137.6	0.0	0.0	303.6	378.0	163.8	120.0

Table 2: Ablation of the adversarial updates for RAMBO. These results use the same rollout length for each dataset as RAMBO but with no adversarial updates (i.e. $\lambda = 0$). The scores are averaged over 5 seeds.

RAMBO (No Adversarial Training)	
MuJoCo-v2 Total: 694.4 ± 56.5	**AntMaze-v0 Total:** 45.8 ± 21.8

Table 3: Comparison between RAMBO and COMBO for the Single Transition Example. We use 20 seeds and ± captures the standard deviation over seeds. RAMBO outperforms COMBO ($p = 0.005$).

RAMBO: **1.49 ± 0.05**	**COMBO:** 1.41 ± 0.12

Figure 5: Results

- RAMBO performs well for suboptimal and noisy dataset
- Outperformed by most of the algorithms in Medium-Expert
- In Ant Maze, RAMBO performs better than Model, but it is outperformed by model-free algorithms (because model-based algorithms are aggressive and collide with the wall)
- Table 2 shows the importance of adversarial updates- without adversarial update ie $\lambda = 0$ we get lesser scores than with adversarial updates
- Table 3 shows the comparison of COMBO and RAMBO

6.1 Comparing RAMBO and COMBO

COMBO	ROMBO
• Penalizes the value function in state-action pairs that are unsupported by the offline dataset • Pessimism from the start	• Doesn't modify the value function, rather works with the distribution • Gradual pessimism

The two approaches are similar in terms that they don't use any sort of uncertainty estimation. But they are different as explained by the table above. A single transition environment was used to compare the two approaches, ie, take an action a from state s and reach state s' to receive a reward r(s') = s'. The MDP is formulated as below : The agent executes a single action, $a \in [-1, 1]$, from initial state s_0. After executing the action, the agent transitions to s' and receives a reward equal to the successor state, i.e. $r(s') = s'$, and the episode terminates.

The actions in the dataset are sampled uniformly from $a \in [-0.75, 0.7] \cup [-0.15, -0.1] \cup [0.1, 0.15] \cup [0.7, 0.75]$. In the MDP for this domain, the transition distribution for successor states is as follows:

- $s' \sim \mathcal{N}(\mu = 1,\ \sigma = 0.2)$, for $a \in [-0.8,\ -0.65]$.
- $s' \sim \mathcal{N}(\mu = 0.5,\ \sigma = 0.2)$, for $a \in [-0.2,\ -0.05]$.
- $s' \sim \mathcal{N}(\mu = 1.25,\ \sigma = 0.2)$, for $a \in [0.05,\ 0.2]$.
- $s' \sim \mathcal{N}(\mu = 1.5,\ \sigma = 0.2)$, for $a \in [0.65,\ 0.8]$.
- $s' = 0.5$, for all other actions.

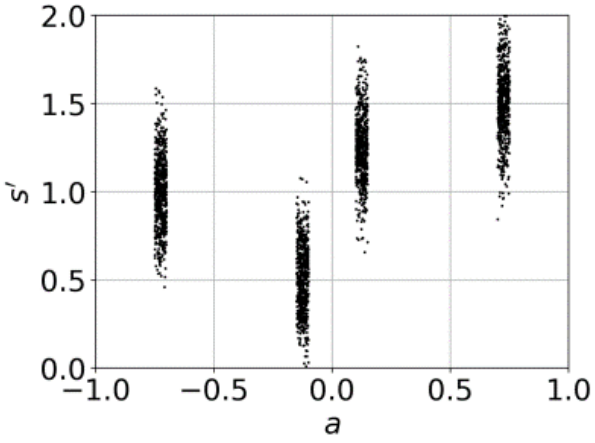
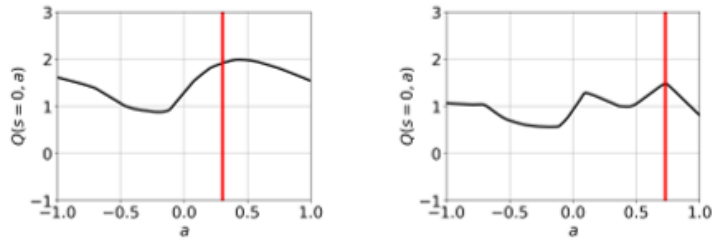
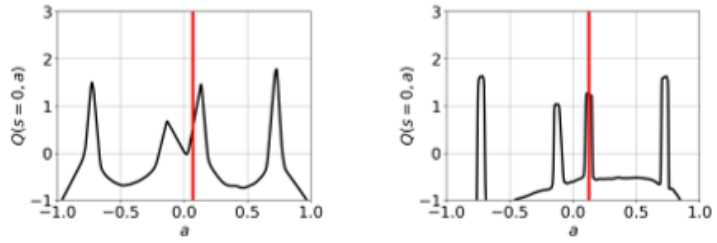


Figure 6: Transition data for the Single Transition Example after executing action a from s_0



(a) 5 iterations (b) 50 iterations

Figure 2: Q-values at the initial state during the training of RAMBO on the Single Transition Example. The red line indicates the mean action of the policy.



(a) 5 iterations (b) 50 iterations

Figure 3: Q-values at the initial state during the training of COMBO on the Single Transition Example ($\beta = 0.2$). The red line indicates the mean action of the policy.

Figure 7: Performance comparison

Fig 7 shows the compared performance between RAMBO and COMBO. (a) and (b) shows that for RAMBO - it goes gradually pessimistic. It is indeed optimistic initially as $\hat{T}$ starts from the MLE estimate and slowly deviates.The gradual introduction of pessimism avoids getting stuck at local maxima. In the second part for COMBO, Since it is very pessimistic right from the start, it poses the issue as the gradient update leads to getting stuck at local maxima. Pessimism from the start, value estimates penalize out-of-dataset actions and so the unexplored regions still have a very low value function estimates even after large number of iterations.
This experiment shows that the RAMBO is superior is better than COMBO.

7 Conclusions and Future Possibilities

- The failure in the Ant Maze problem was analyzed by combining RAMBO with reverse rollouts, as part of an Offline Reinforcement Learning approach with Reverse Model-based Imagination.
- It was observed that in a few environments within MoJoCo, the hyperparameter tuning resulted in $\lambda = 0$. This observation implies that certain problems might not require regularization for a successful policy to be trained offline with model-based reinforcement learning.
- A potential direction for future work could involve identifying problems that do not need regularization for effective offline policy training with model-based reinforcement learning.

Fig 8 shows that there are a few environments with $\lambda = 0$. This observation implies that certain problems might not require regularization for a successful policy to be trained offline with model-based reinforcement learning.

7.1 Reverse Rollouts

We could also use the concept of reverse rollouts to get synthetic data along with the normal rollouts. Reverse rollouts involve using offline data to improve policy learning by utilizing both forward and reverse dynamics models. The reverse dynamics model, denoted as $\hat{T}_r(s' \mid s, a)$, and the reward model, denoted as $\hat{r}(s, a)$, are estimated. This gives more diversity to the training data, leading to better generalization in various states of the environment. This could help achieve better performance in the AntMaze environment.

		$k = 2, \lambda = 3e-4$	$k = 5, \lambda = 3e-4$	$k = 5, \lambda = 0$
Random	HalfCheetah	34.5 ± 3.7 (**348.1**)	38.2 ± 3.9 (408.6)	40.0 ± 2.3 (377.6)
	Hopper	21.6 ± 8.0 (1179.7)	21.3 ± 9.2 (1293.2)	15.1 ± 9.4 (**188.9**)
	Walker2D	–	0.2 ± 0.5 (**1.2e8**)	11.5 ± 10.5 (2.5e9)
Medium	HalfCheetah	72.5 ± 4.4 (**671.1**)	77.6 ± 1.5 (717.4)	75.5 ± 6.0 (720.6)
	Hopper	–	92.8 ± 6.0 (**300.4**)	47.5 ± 36.0 (314.1)
	Walker2D	82.2 ± 11.4 (523.6)	76.8 ± 4.8 (2124.2)	86.9 ± 2.7 (**339.4**)
Medium Replay	HalfCheetah	65.4 ± 3.7 (623.5)	68.9 ± 2.3 (647.2)	64.7 ± 3.2 (**608.4**)
	Hopper	96.6 ± 7.0 (**262.4**)	93.5 ± 10.3 (309.1)	36.7 ± 9.5 (263.4)
	Walker2D	6.7 ± 2.5 (2.3e7)	62.1 ± 33.5 (4.3e6)	85.0 ± 15.0 (**259.0**)
Medium Expert	HalfCheetah	78.1 ± 6.6 (**987.1**)	93.7 ± 10.5 (1008.9)	92.9 ± 11.9 (1013.2)
	Hopper	36.4 ± 16.7 (344.1)	83.3 ± 9.1 (**342.8**)	77.6 ± 14.3 (344.4)
	Walker2D	68.3 ± 20.6 (**371.9**)	31.7 ± 49.8 (2.0e7)	50.8 ± 57.8 (6.9e9)
AntMaze	Umaze	8.8 ± 8.2 (−48.3)	25.0 ± 12.0 (**-54.2**)	3.8 ± 5.8 (−51.5)
	Medium-Play	5.8 ± 6.6 (1421)	8.4 ± 13.5 (**-70.77**)	16.4 ± 17.9 (−68.10)
	Large-Play	0.0 ± 0.0 (−77.9)	0.0 ± 0.0 (**-78.1**)	0.0 ± 0.0 (−77.9)
	Umaze-Diverse	0.0 ± 0.0 (790.8)	0.0 ± 0.0 (68.2)	0.0 ± 0.0 (**-65.4**)
	Medium-Diverse	3.8 ± 1.7 (**-72.4**)	1.6 ± 1.8 (−72.2)	23.2 ± 14.2 (−71.2)
	Large-Diverse	–	0.0 ± 0.0 (6.0e8)	2.4 ± 3.3 (**3.5e6**)

Figure 8: Hyperparameter tuning - significance of Lambda