Project Report - Model-Based Offline Reinforcement Learning

Lamha Goel (lg36694) and Kunjal Shah (kss3422)

I. Abstract

Using offline datasets for reinforcement learning can be valuable because online trial-and-error can be quite expensive and risky. While there are both model-free and model-based approaches for offline RL, model-free approaches tend to be overly conservative, and thus do not generalize well. In this project, we focus on two model-based offline RL approaches: MOPO [21] and COMBO [22]. MOPO was one of the pioneer approaches for model-based offline RL. COMBO was later proposed by a lot of the same folks that worked on MOPO and shows superior results than MOPO. We first implement these approaches, and then further experiment with various aspects of these to improve upon them while also gaining a better, deeper understanding. In particular, we focus on the penalty for MOPO, and use ideas from CQL [13] for enhancing COMBO. We are able to produce results similar to the original authors with our implementations, and are able to improve the results of MOPO with one of the proposed penalties. One of our proposed experiments for COMBO can be used to improve its speed. We also include some videos in the supplementary material to show some successes and some initial failures of our models.

II. Introduction

Utilizing offline datasets for reinforcement learning (RL) can be extremely beneficial since it can enable safe learning of policies reducing or even eliminating the risk of harm in the real-world. This is extremely useful for high-risk domains like autonomous driving or healthcare, where it is not feasible to allow an RL agent to explore the environment, thus limiting RL progress and application in such fields. It also opens up the possibility of learning from tons of already available data and thus brings a hope of accelerating the applications of RL. Within offline reinforcement learning paradigms, approaches can broadly be categorized as model-free and model-based approaches. Model-free algorithms directly learn the policy and learn only on the states in the offline datasets, which leads them to learn overly conservative algorithms. Model-based approaches, on the other hand, learn a model of the environment from the dataset and utilize this model along with the dataset to learn a policy which is generally much less conservative compared to model-free approaches and thus achieves better generalization.

In this project, we explore and experiment with model-based offline reinforcement learning approaches, particularly focusing on two state-of-the-art approaches: MOPO: Model-based Offline Policy Optimization [21] and COMBO: Conservative Offline Model-Based Policy Optimization [22].

Since our focus in this project is to gain a deep understanding of model-based reinforcement learning approaches, with a focus on MOPO and COMBO, we define the following goals:

1) Gain a theoretical understanding of MOPO and COMBO and the approaches used as building blocks for these
2) Implement MOPO and COMBO
3) Experiments with MOPO - experiments with the penalty function
4) Experiments with COMBO - experiment with the regularizer
5) Experiments with COMBO - experiment with learning β instead of tuning it as a hyperparameter

We'll test the implemented approaches on two environments: Hopper and Walker2D from OpenAI Gym using offline datasets provided in D4RL [5]. We attempt to be able to reproduce the results from the respective papers. With the various experiments, we expected to gain a deep understanding of the various components of these approaches and improve over their results. As our metric for all comparisons, we'll use the normalized scores, as is the standard defined in [5]. We'll additionally compare the time each approach takes to train to understand the performance and the time-efficiency trade-off.

III. Related Work

This project explores the intersection of two fields of reinforcement learning: offline RL, and model-based RL. There has been significant work in the field of offline reinforcement learning. Approaches are primarily categorized as model-free or model-based depending on whether they learn a dynamics model of the environment to support the policy learning process. Most of the work before MOPO [21] focused on model-free offline RL.

For model free offline RL setting, existing approaches build on two primary ideas. The first idea is to restrict the learned policy to be in proximity to the behavioral policy explicitly (as in [6],[12]) or implicitly by means of regularized variants of importance sampling based algorithms ([16],[19]), offline actor-critic methods ([15], [17]) applying uncertainty quantification to the predictions of the Q-values ([2],[10]) and learning conservative Q-values ([13],[18]). The other key idea is to quantify uncertainty, usually done using ensembles, to stabilize the Q-functions (as in [1], [12]). MOPO and COMBO do not explicitly restrict the policy to the dataset's distribution but they do use uncertainty estimation approaches to quantify

the risk of taking an out-of-distribution action (MOPO) or learn a conservative estimate for the Q function (COMBO).

In the field of model-based RL, most of the existing work had focused on the online RL setting. Models like local linear models [14], Gaussian processes [4] and neural network function approximators [7] have been proposed by earlier works for selecting actions using planning. MOPO and COMBO are built upon Model Based Policy Optimization (MBPO) [9] which learns an ensemble of probabilistic neural network models.
At the time MOPO was proposed, there had been little work in the intersection of these two fields: model-based offline RL. MOReL [11] was proposed in the same year - it constructs terminating states based on a hard threshold on uncertainty, which is more restrictive on the policy and effectively disallows it to take risky actions.
Besides these, there's three primary publications that act as building blocks for MOPO and COMBO and are thus, key to understanding them:

A. Soft Actor-Critic

Soft Actor-Critic (SAC) is one of the most popular approaches for model-free reinforcement learning, particularly because it has demonstrated good sample efficiency and it has a single hyperparameter (which can also be automatically tuned over the course of learning). It utilizes a maximum entropy formulation and thus, maximized entropy along with the reward - this incentivizes the agent to explore. It builds upon maximum entropy reinforcemnt learning and soft policy iteration approaches. In practical implementations of soft-actor critic, two soft Q-functions are used since that has consistently demonstrated improved stability and performance and speeds up training. The SAC model alternates between exploring (collecting new experience) and updating its policy approximators using off-policy data from the "replay pool". The same approach has been taken by MOPO and COMBO even though they are offline algorithms: in each epoch, they use the model to "generate new experience", and then learn using samples from the offline dataset and this generated experience.

B. Conservative Q-Learning (CQL)

In addition to the above ideas, COMBO is also inspired by Conservative Q-Learning (CQL), which is a model-free reinforcement learning approach which addresses the problem of overestimating values for unseen state-action pairs without explicitly needing the policy to stay away from out of distribution actions. The learned Q function ensures that the expected value of the policy lower bounds the true value. The approach modifies the Q learning update rule: it minimizes the expected Q value under a particular distribution of state-action pairs, maximizes the expected Q value under the distribution of the behavior policy used to generate the dataset and adds a regularizer (like KL divergence against a prior distribution). The key benefit of CQL is that it only changes the objective for the Q function - thus, it can be used in any actor-critic or Q learning variant simply by changing the update rule.

C. Model Based Policy Optimization (MBPO)

The key approach that MOPO and COMBO build upon is Model Based Policy Optimization (MBPO). MBPO is a model-based online RL approach that is built on the idea of short model-generated rollouts with a guarantee of monotonic improvement at each step. It uses a bootstrap ensemble of dynamics models, where each model is a probabilistic neural network whose outputs parameterize a Gaussian distribution with diagonal covariance. This formulation helps account for both aleatoric and epistemic uncertainty. A prediction is obtained from this ensemble of models by selecting a model uniformly at random . While compounding model errors make extended rollouts difficult in general, the authors find theoretically and empirically by approximating the model error on the distribution of the current policy that it is useful to include model generated data for small rollouts branched from real data as long as this error is sufficiently low. The approach uses SAC as a sub-component for policy optimization. The algorithm is shown in Figure 1.

IV. Data

We test our approaches on two of the MuJoCo based OpenAI Gym [3] environments: Hopper and Walker2D. Some sample captures are shown in Figure 2.

A. Hopper

The hopper environment has a 2D one-legged figure that consists of four body parts - the torso, the thigh, the leg, and a single foot on which the entire body rests. It is a stochastic environment in terms of the initial state, with a Gaussian noise added to a fixed initial state to add stochasticity. The goal of the robot is to hop in the forward direction by applying torques on the three joints connecting the four body parts.
Each action is a vector of size 3, with each value between -1 and 1. The values represent the torque applied on the three joint rotors: thigh joint, leg joint and foot joint. All three joints are hinge type joints.
Each observation is a vector of size 11 and contains both positional and velocity information. It includes information about the height of the hopper, angle of the top and of the three joints, and velocity of the top along with the angular velocity of the top and the hinge joints.
The reward has three components: healthy reward, forward reward, and control cost. Healthy reward is a constant reward given to the robot at each timestep that it is still healthy, i.e., its observations are within the allowed ranges. The forward reward is given to the robot to keep it moving in the forward direction: it is positive if it moves forward, and negative if it moves backward. Lastly, the control cost is to restrict the robot from taking too large a step since that would not be feasible/ideal in the real world.

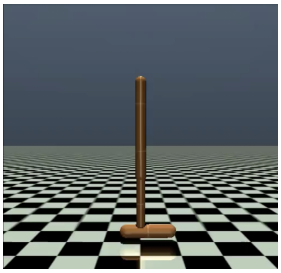
B. Walker2D

The Walker2D environment looks quite similar to Hopper. It has two legs instead of one making it possible for the robot to walk forward instead of hop. It is a 2d two-legged

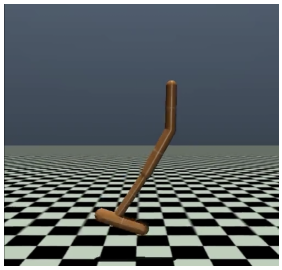
Algorithm 2 Model-Based Policy Optimization with Deep Reinforcement Learning

Initialize policy π_ϕ, predictive model p_θ, environment dataset $\mathcal{D}_{\text{env}}$, model dataset $\mathcal{D}_{\text{model}}$
for N epochs **do**
  Train model p_θ on $\mathcal{D}_{\text{env}}$ via maximum likelihood
  for E steps **do**
    Take action in environment according to π_ϕ; add to $\mathcal{D}_{\text{env}}$
    for M model rollouts **do**
      Sample s_t uniformly from $\mathcal{D}_{\text{env}}$
      Perform k-step model rollout starting from s_t using policy π_ϕ; add to $\mathcal{D}_{\text{model}}$
    for G gradient updates **do**
      Update policy parameters on model data: $\phi \leftarrow \phi - \lambda_\pi \hat{\nabla}_\phi J_\pi(\phi, \mathcal{D}_{\text{model}})$

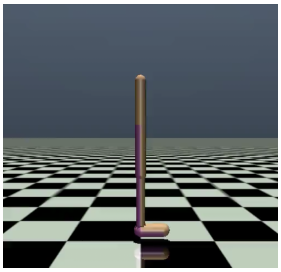
Fig. 1: Model-based Policy Optimization Algorithm, taken from [9]



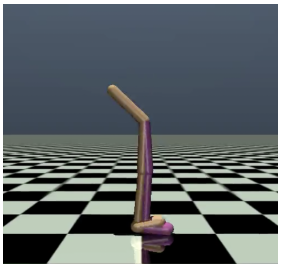
(a) Hopper: sample start state



(b) Hopper: in the middle of a hop



(c) Walker2D: sample start state



(d) Walker2D: in the middle of a walk

Fig. 2: Sample captures from the environments. Captures show both start states and intermediate states for clear understanding of the environment. For a better understanding, please see the videos provided in supplementary material.

figure with a single torso, two thighs, two legs and two feet. It is again a stochastic environment in terms of the initial state. The environment increases the number of independent state and control variables as compared to the classic control environments, making it a comparatively difficult task. The goal of the robot is to coordinate both sets of feet, legs, and thighs to move forward by applying torques on the six hinge joints connecting the six body parts.

Each action is a vector of size 6, with each value between -1 and 1. The values represent the torque applied on the six joint rotors: two each of thigh joints, leg joints and foot joints. All six joints are hinge type joints.

Each observation is a vector of size 17 and contains both positional and velocity information. It includes information about the height of the hopper, angle of the top and of the six joints, and velocity of the top along with the angular velocity of the top and the hinge joints.

The reward is the same as with Hopper. It has three components: healthy reward, forward reward, and control cost. Healthy reward is a constant reward given to the robot at each timestep that it is still healthy, i.e., its observations are within the allowed ranges. The forward reward is given to the robot to keep it moving in the forward direction: it is positive if it moves forward, and negative if it moves backward. Lastly, the control cost is to restrict the robot from taking too large a step since it would not be feasible/ideal in the real world.

C. *Offline data*

Since are focus is offline RL, we use offline datasets provided in D4RL [5] to train our models. In particular, for both Hopper and Walker2D, we use 3 different datasets (defined relative to an expert, i.e., a policy trained to completion using SAC):

- Medium: It includes one million samples from a policy trained to approximately 1/3 the performance of the expert.
- Medium-Replay: It uses the replay buffer of a policy trained up to the performance of the medium agent.
- Medium-Expert: It has a 50-50 split of medium and expert data (slightly less than two million samples in total).

The use of different datasets lets us analyze how the performance of the agent is impacted by the quality of data.

V. Methods

As part of the project, we start with implementation of MOPO and COMBO, which we describe below, followed by different analysis and experiments with each of these:

A. Model-based Offline Policy Optimization (MOPO)

MOPO builds on the concepts of MBPO (and similar to MBPO, uses SAC to optimize the policy). It effectively adapts MBPO for the offline RL setting. In particular, to account for the offline setting's distributional shift problem, it penalizes the reward using the uncertainty of the dynamics when generating a sample using the dynamics model. In general for a model based offline setting, two uncertainties need to be accounted for:

- Epistemic uncertainty: uncertainty of the model
- Aleatoric uncertainty: data's inherent randomness that cannot be explained away

Using an ensemble of models instead of a single dynamics model allows one to estimate both these uncertainties.
The "practical" algorithm is included in Figure 3. The key contribution is defined by Line 9 in the algorithm - penalization of the reward. Here, they model the uncertainty as the maximum standard deviation of the learned models.
For MOPO's implementation, instead of implementing from scratch, we utilize an open source GitHub repository [20]. Unfortunately, the official source code published by the authors of MOPO [21] has hard dependency on TensorFlow 1 which is unsupported in Google Colaboratory, our workspace environment, so we used this third-party repository instead.
After the vanilla implemntation of MOPO, we experimented with the penalty function in MOPO. In particular, besides the original penalty function of using maximum variance across the ensemble, we tried five different penalty functions based on the mean and variance of the models in the ensemble: max difference between means, max distance of means from their center, variance of the ensemble, variance of the individual model from ensemble and a combination of max variance and individual variance. These are discussed in detail in the experiments section.

B. Conservative Offline Model Based Policy Optimization

The key idea behind COMBO is that uncertainty estimation (as is done in MOPO) can be inaccurate in offline RL problems and thus they propose an approach which does not require uncertainty estimation, and instead learns a conservative estimate for Q value. It uses a similar objective as the one proposed for CQL, and adapts the Q value update rule to learn a conservative estimate so that they don't have to explicitly penalize the reward like in MOPO. The basic structure of COMBO is quite similar to MOPO (since they're both built on the ideas of MBPO), but instead of simply using vanilla SAC for policy optimization, they provide the update rule for the Q-value (similar to CQL, but in an offline setting). The approach for COMBO is included in Figure 4. The key idea here is combining CQL and MBPO appropriately for the offline setting.
We implemented COMBO ourselves in this project. The code is made available on GitHub [8]. After the initial implementation of COMBO, we further updated our implementation to accommodate various experiments. All these experiments are motivated by ideas from the paper on CQL [13] since that forms the guiding idea behind COMBO:

- Changing the regularizer for the Q value update equation
- Learn β automatically instead of via hyperparameter tuning
- (Fun experiment) Remove the term for maximizing Q value over the dataset in the Q value update equation

VI. Experiments

A. Experiments with MOPO

We started with the MOPO implementation from a third-party GitHub repository [20], and took some time to ensure we understand how it's working and the different components to it. Since the uncertainty estimation for reward penalization forms the key idea in MOPO, we experimented with different penalties. The authors propose to use maximum variance across all models in the ensemble as their penalty (Original), i.e.,

$$max_{i=1}^{N}||\Sigma_i(s_j, a_j)||_F$$

Based on similar ideas, and inspired by the idea of clustering, we also experimented with uncertainty estimation based on the means, this particularly focuses on the uncertainties associated with the model:

- Maximum difference between the means (shown in Figure 5a) (Difference of means):

$$max_{i,j}||\mu_i(s_k, a_k) - \mu_j(s_k, a_k)||_2$$

- Maximum distance of means from their "center" (shown in Figure 5b) (Means diff from center) :

$$\mu = mean_{i=1}^{N}(\mu_i(s_j, a_j))$$

$$max_{i=1}^{N}||\mu_i(s_j, a_j) - \mu||_2$$

We also tried using variance in different ways to estimate the uncertainty:

- Since the ensemble of models is basically a mixture of Gaussians, and we want to be conservative about our rewards but not "too conservative", we tried experimenting with the variance of the ensemble instead of maximum variance (Ensemble Variance):

$$\sum_i ||\Sigma_i(s_j, a_j)||_F + \sum_i ||\mu_i(s_j, a_j)||_2 - ||\sum_i \mu_i(s_j, a_j)||_2$$

- Based on the feedback in the poster session, we also tried to incorporate individual variance. Instead of using all models in the ensemble to estimate uncertainty, we only use the variance of the model we sampled the trajectory from as our penalty (Individual Var)
- Lastly, we tried combining the two successful variance based uncertainty estimation approaches: we take an equally weighted (0.5 weight for each) sum of maximum variance across ensemble and variance of the model we sample from as our penalty (Combined Var), i.e.:

$$0.5 * \text{Individual Var} + 0.5 * \text{Original}$$

Algorithm 2 MOPO instantiation with regularized probabilistic dynamics and ensemble uncertainty

Require: reward penalty coefficient λ rollout horizon h, rollout batch size b.

1: Train on batch data $\mathcal{D}_{\text{env}}$ an ensemble of N probabilistic dynamics $\{\widehat{T}^i(s', r \,|\, s, a) = \mathcal{N}(\mu^i(s,a), \Sigma^i(s,a))\}_{i=1}^N$.
2: Initialize policy π and empty replay buffer $\mathcal{D}_{\text{model}} \leftarrow \varnothing$.
3: **for** epoch $1, 2, \ldots$ **do** ▷ This for-loop is essentially one outer iteration of MBPO
4: **for** $1, 2, \ldots, b$ (in parallel) **do**
5: Sample state s_1 from $\mathcal{D}_{\text{env}}$ for the initialization of the rollout.
6: **for** $j = 1, 2, \ldots, h$ **do**
7: Sample an action $a_j \sim \pi(s_j)$.
8: Randomly pick dynamics $\widehat{T}$ from $\{\widehat{T}^i\}_{i=1}^N$ and sample $s_{j+1}, r_j \sim \widehat{T}(s_j, a_j)$.
9: Compute $\tilde{r}_j = r_j - \lambda \max_{i=1}^N \|\Sigma^i(s_j, a_j)\|_{\text{F}}$.
10: Add sample $(s_j, a_j, \tilde{r}_j, s_{j+1})$ to $\mathcal{D}_{\text{model}}$.
11: Drawing samples from $\mathcal{D}_{\text{env}} \cup \mathcal{D}_{\text{model}}$, use SAC to update π.

Fig. 3: Model-based Offline Policy Optimization Algorithm, taken from [21]

Algorithm 1 COMBO: Conservative Model Based Offline Policy Optimization

Require: Offline dataset $\mathcal{D}$, rollout distribution $\mu(\cdot|\mathbf{s})$, learned dynamics model $\widehat{T}_\theta$, initialized policy and critic π_ϕ and Q_ψ.

1: Train the probabilistic dynamics model $\widehat{T}_\theta(\mathbf{s}', r|\mathbf{s}, \mathbf{a}) = \mathcal{N}(\mu_\theta(\mathbf{s}, \mathbf{a}), \Sigma_\theta(\mathbf{s}, \mathbf{a}))$ on $\mathcal{D}$.
2: Initialize the replay buffer $\mathcal{D}_{\text{model}} \leftarrow \varnothing$.
3: **for** $i = 1, 2, 3, \cdots$, **do**
4: Collect model rollouts by sampling from μ and $\widehat{T}_\theta$ starting from states in $\mathcal{D}$. Add model rollouts to $\mathcal{D}_{\text{model}}$.
5: Conservatively evaluate π_ϕ^i by repeatedly solving eq. 2 to obtain $\hat{Q}_\psi^{\pi_\phi^i}$ using samples from $\mathcal{D} \cup \mathcal{D}_{\text{model}}$.
6: Improve policy under state marginal of d_f by solving eq. 3 to obtain π_ϕ^{i+1}.
7: **end for**

$$\hat{Q}^{k+1} \leftarrow \arg\min_Q \beta \left(\mathbb{E}_{\mathbf{s},\mathbf{a}\sim\rho(\mathbf{s},\mathbf{a})}[Q(\mathbf{s},\mathbf{a})] - \mathbb{E}_{\mathbf{s},\mathbf{a}\sim\mathcal{D}}[Q(\mathbf{s},\mathbf{a})]\right) + \frac{1}{2}\mathbb{E}_{\mathbf{s},\mathbf{a},\mathbf{s}'\sim d_f}\left[\left(Q(\mathbf{s},\mathbf{a}) - \widehat{\mathcal{B}}^\pi \hat{Q}^k(\mathbf{s},\mathbf{a})\right)^2\right]. \quad (2)$$

$$\pi' \leftarrow \arg\max_\pi \; \mathbb{E}_{\mathbf{s}\sim\rho,\mathbf{a}\sim\pi(\cdot|\mathbf{s})}\left[\hat{Q}^\pi(\mathbf{s},\mathbf{a})\right] \quad (3)$$

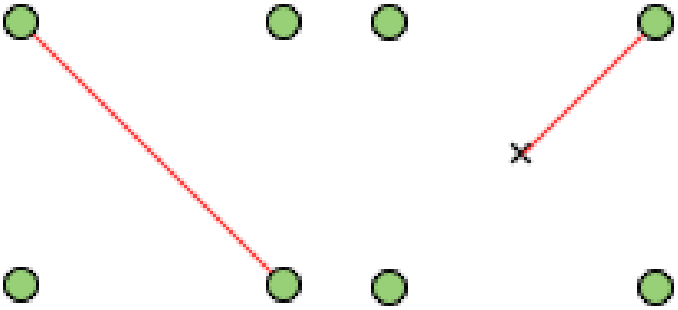
Fig. 4: Conservative Offline Model Based Policy Optimization Algorithm, taken from [22]

In all cases, we used dynamics models and actor-critic models with the same architecture as in the original publication. To choose hyperparameters, we used online rollouts as done in the original paper - the resulting best hyperparameter values were the same as the original publication.

Figure 6 shows how the rewards (unnormalized scores) evolved over the training period for MOPO when using original penalty (maximum variance). The curves for other penalties are quite similar, and hence omitted in favour of space. The sudden jumps in rewards seem to be coming from fortunate model rollouts. But the overall training seems to be jittery and unstable, which unfortunately seems to be a common problem with MOPO (based on the issues section of the official Github repository). Table I shows the normalized scores for the experiments discussed above. The implementation we used for MOPO produces similar results as published in the original paper. The two means-based penalties have quite contrasting results - based on our analysis, the scale of penalty for maximum difference of means is much larger than in maximum distance of means from center (as is expected). This leads to the learned policy becoming too conservative, which might be the cause of the worse performance of the former. While ensemble variance did not perform quite well, individual variance had similar performance as using the original penalty. The penalty using the combined variance outperformed the original penalty in all cases. Since this combines maximum variance and individual variance, which are more and less conservative respectively, their combination seems to be the perfect amount of conservative without impacting the generalizability of the policy.

	hopper-medium	hopper-medium -replay	hopper-medium -expert	walker2d-medium	walker-2d-medium-replay	walker-2d-medium-expert
Original	$\mathbf{28.0 \pm 12.4}$	$\mathbf{67.5 \pm 24.7}$	$\mathbf{23.7 \pm 6.0}$	$\mathbf{17.8 \pm 19.3}$	$\mathbf{39.0 \pm 9.6}$	$\mathbf{44.6 \pm 12.9}$
Original (Ours)	25.94 ± 13.48	61.96 ± 24.35	19.84 ± 9.24	14.39 ± 15.75	32.63 ± 10.49	40.32 ± 10.34
Difference of Means	10.34 ± 12.32	41.09 ± 20.46	6.24 ± 5.12	5.42 ± 3.25	11.34 ± 7.43	20.57 ± 10.46
Means Diff from Center	22.39 ± 12.45	64.55 ± 26.01	19.35 ± 11.65	15.96 ± 13.49	30.13 ± 9.25	38.29 ± 11.09
Ensemble Variance	18.25 ± 10.32	59.89 ± 30.96	14.43 ± 10.39	10.27 ± 10.20	18.35 ± 9.31	30.45 ± 15.21
Individual Variance	24.52 ± 14.22	65.33 ± 27.70	18.25 ± 8.94	18.29 ± 15.47	34.23 ± 13.89	39.58 ± 15.81
Combined Variance	$\mathbf{30.84 \pm 11.39}$	$\mathbf{88.49 \pm 15.30}$	$\mathbf{28.93 \pm 8.39}$	$\mathbf{22.48 \pm 11.86}$	$\mathbf{48.49 \pm 9.73}$	$\mathbf{42.79 \pm 10.82}$

TABLE I: Normalized scores on different datasets using MOPO with different penalties for the reward from the dynamics model. Expert score corresponds to 100, and score for the random policy corresponds to 0. Original demonstrates the results from the original publication, whereas Original (Ours) shows the results on our implementation of the approach from the paper. The other penalties are as described in the Experiments section. The top two results for each dataset are highlighted.



(a) Visualizing the maximum distance between means penalty: The green dots show the different means, and the red line marks the maximum distance between any pair of means. The length of the red line is used as the penalty.

(b) Visualizing the maximum distance from center of means penalty: The green dots show the different means, the cross marks their center (their mean) and the red line marks the maximum distance of any mean from the center (in this case, they are equidistant). The length of the red line is used as the penalty.

Fig. 5: Visualizing means based penalties for MOPO

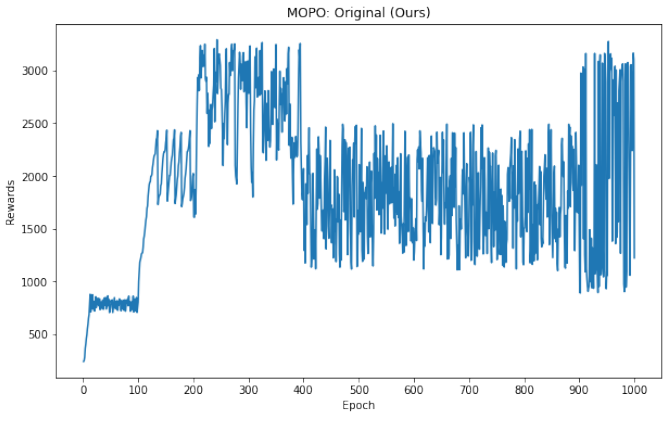


Fig. 6: Training curve for MOPO to demonstrating how rewards evolved over the training period.

B. Experiments with COMBO

We then implemented COMBO as well. Besides the vanilla implementation of COMBO, we tried the following experiments, insipired by ideas from CQL.

1) Changing the regularizer in COMBO: We tried experimenting with changing the regularizer for the Q value update equation, i.e., in the following general Conservative Q-Learning equation:

$$min_Q max_U \alpha(\mathbb{E}_{s\sim D, a\sim\mu(a|s)}[Q(s,a)] - \mathbb{E}_{s\sim D, a\sim\hat{\pi}_\beta(a|s)}[Q(s,a)]) + \frac{1}{2}\mathbb{E}_{s,a,s'\sim D}[(Q(s,a) - \hat{\beta}^{\pi_k}\hat{Q}^k(s,a))^2] + \mathbb{R}(\mu)$$

we try different values for $\mathbb{R}(\mu)$. The authors of COMBO use KL divergence against a prior distribution. For the prior distribution, they use a uniform distribution over the action space, i.e., they use random actions.
Instead, we propose to use KL divergence, but using the policy from the previous iteration for prior distribution.

2) Learn Beta: Similar to CQL where they're able to learn a hyperparameter α automatically instead of via hyperparameter tuning, we try to see if we can learn β automatically instead of hyperparameter tuning. The authors of COMBO tune β using the value of the following term to decide the optimal value:

$$\beta(\mathbb{E}_{s,a\sim\rho(s,a)}[Q(s,a)] - \mathbb{E}_{s,a\sim D}[Q(s,a)])$$

So, we propose to use the same term for gradient descent over β to automatically learn its value. We use a temperature parameter, similar to CQL to help with this learning.

3) Fun experiment: We tried this last experiment just for fun (mostly, just to verify our understanding). Since the authors particularly emphasize the importance of maximizing Q value over the dataset in the Q value update equation, we removed that term from the update equation to see the impact it has on the performance of the agent:

$$\hat{Q}^{k+1} \leftarrow argmin_Q \beta(\mathbb{E}_{s,a\sim\rho(s,a)}[Q(s,a)] - \mathbb{E}_{s,a\sim D}[Q(s,a)]) + \frac{1}{2}\mathbb{E}_{s,a,s'\sim d_f}[(Q(s,a) - \hat{\beta}^{\pi}\hat{Q}^k(s,a))^2]$$

Again, in all cases, we used dynamics models and actor-critic models with the same architecture as in the original publication (which were mostly the same as MOPO). To choose

	hopper-medium	hopper-medium -replay	hopper-medium -expert	walker2d-medium	walker-2d-medium-replay	walker-2d-medium-expert
Original	$\mathbf{97.2 \pm 2.2}$	$\mathbf{89.5 \pm 1.8}$	$\mathbf{111.1 \pm 2.9}$	$\mathbf{81.9 \pm 2.8}$	$\mathbf{56.0 \pm 8.6}$	$\mathbf{103.3 \pm 5.6}$
Original (Ours)	92.43 ± 10.82	$\mathbf{80.21 \pm 11.05}$	$\mathbf{98.25 \pm 8.52}$	$\mathbf{80.23 \pm 9.41}$	$\mathbf{58.23 \pm 12.34}$	$\mathbf{97.29 \pm 9.64}$
Learn beta	75.22 ± 9.83	60.09 ± 7.43	70.28 ± 8.15	65.29 ± 10.25	41.25 ± 6.49	79.26 ± 8.73
Previous Policy	$\mathbf{92.55 \pm 9.87}$	78.59 ± 10.62	96.14 ± 9.29	75.25 ± 8.28	54.28 ± 6.18	93.14 ± 7.25
Previous Policy, Learn beta	20.39 ± 6.32	21.01 ± 5.83	29.42 ± 7.93	19.25 ± 4.39	12.04 ± 5.28	16.94 ± 7.25
Remove maximization term	70.67 ± 11.07	58.46 ± 9.75	62.93 ± 6.10	60.24 ± 8.19	37.29 ± 6.26	72.18 ± 7.39

TABLE II: Normalized scores on different datasets using COMBO. Expert score corresponds to 100, and score for the random policy corresponds to 0. Original demonstrates the results from the original publication, whereas Original (Ours) shows the results on our implementation of the approach from the paper. The other rows correspond to the various enhancements discussed in the Experiments section. The top two results for each dataset are highlighted.

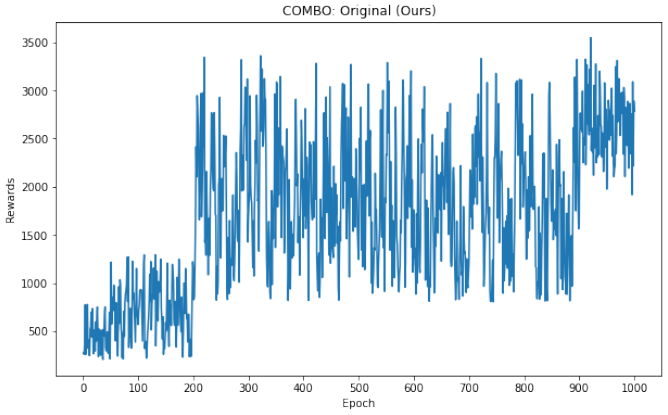


Fig. 7: Training curve for COMBO demonstrating how rewards evolved over the training period.

hyperparameters, we used the value of the regularization term, similar to the original paper - the resulting hyperparameter values were again the same as the original publication.

Figure 7 shows the evolution of rewards (unnormalized scores) over the training period for the vanilla implementation of COMBO. The curves for other enhancement didn't show any clear pattern either, so have been omitted. Similar to MOPO, the training is unstable. Table II shows the normalized scores for all the experiments discussed above for COMBO. Our implementation of COMBO produces similar results as published in the original paper. β learning seems to have been unsuccessful in both cases - while we tried this with various hyperparameter values, we are unsure whether it is not possible to learn β automatically in this case, or whether it needs more tuning - in either case though, it seems one is better off simply tuning β as a hyperparameter itself. As expected, when removing the maximization term, the performance degrades by about 20% in for all datasets. Lastly, using the previous policy as the regularizing policy for Q value update seems to have similar performance as using a uniform policy. This could be beneficial to make COMBO faster since we could reuse samples from previous iteration for regularization instead of having to sample from uniform policy each time.

C. Observations: MOPO vs COMBO trade-offs

In general, it seems COMBO has a superior performance than MOPO. But it comes with its own disadvantages. Particularly, while MOPO took about $30 - 40$ seconds for each epoch, COMBO took $3 - 4$ minutes for a single epoch, i.e., it was about 60x slower.

COMBO also seems to need much more real data than MOPO. MOPO uses only 5% real data in each batch, and the rest 95% can come from the dynamics model. When we used similar parameters for COMBO, it showed extremely inferior performance. We finally used 50% each of real and model-based data for each batch, as suggested in [22] to achieve the current performance.

VII. Conclusion

We find that COMBO has a much better performance than MOPO, but needs more time and data for training. It can be made a little faster using the previous policy for regularization instead of uniform policy. We also analyzed the importance of penalty in MOPO and found that combined variance is the "sweet spot" outperforming all other penalties. There are still a few things one can possibly explore in this area: there is still potential to auto-learn β in COMBO. Also, currently MOPO and COMBO are conservative about all rollouts from the model, but it may be possible to have a "confidence" score or something similar to only penalize the model rollouts when we truly are unsure about it. It may also be interesting to find a way to tune hyperparameters for MOPO without online rollouts, similar to COMBO. We also expected to see clear patterns in certain datasets being better for training offline RL approaches, but both MOPO and COMBO seem to have better performance on different kinds of datasets - it might be interesting to explore the reasons behind this. Lastly, since we worked with locomotion tasks, it may be interesting to see if these approaches generalize well to manipulation tasks as well.

VIII. Individual Contributions

Lamha - Code implementation and understanding (100%), enhancements and experiments (100%), debugging repository and MuJoCo setup issues, and 90% of the report.

Kunjal - Initial official MOPO repository setup and debugging, trying repositories for MOPO and COMBO, helping with experiments (running and monitoring Colab notebooks for training through multiple iterations and checkpoints over MOPO and COMBO) and 10% of the report.

References

[1] Rishabh Agarwal, Dale Schuurmans, and Mohammad Norouzi. Striving for simplicity in off-policy deep reinforcement learning. 2019.

[2] Rishabh Agarwal, Dale Schuurmans, and Mohammad Norouzi. An optimistic perspective on offline reinforcement learning. In *International Conference on Machine Learning*, pages 104–114. PMLR, 2020.

[3] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[4] Marc Deisenroth and Carl E Rasmussen. Pilco: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on machine learning (ICML-11)*, pages 465–472. Citeseer, 2011.

[5] Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4rl: Datasets for deep data-driven reinforcement learning. *arXiv preprint arXiv:2004.07219*, 2020.

[6] Scott Fujimoto, David Meger, and Doina Precup. Off-policy deep reinforcement learning without exploration. In *International conference on machine learning*, pages 2052–2062. PMLR, 2019.

[7] Yarin Gal, Rowan McAllister, and Carl Edward Rasmussen. Improving pilco with bayesian neural network dynamics models. In *Data-Efficient Machine Learning workshop, ICML*, volume 4, page 25, 2016.

[8] Lamha Goel. Github: Model based offline reinforcement learning. https://github.com/lamhagoel/mopo-1.

[9] Michael Janner, Justin Fu, Marvin Zhang, and Sergey Levine. When to trust your model: Model-based policy optimization. *Advances in Neural Information Processing Systems*, 32, 2019.

[10] Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, et al. Scalable deep reinforcement learning for vision-based robotic manipulation. In *Conference on Robot Learning*, pages 651–673. PMLR, 2018.

[11] Rahul Kidambi, Aravind Rajeswaran, Praneeth Netrapalli, and Thorsten Joachims. Morel: Model-based offline reinforcement learning. *Advances in neural information processing systems*, 33:21810–21823, 2020.

[12] Aviral Kumar, Justin Fu, Matthew Soh, George Tucker, and Sergey Levine. Stabilizing off-policy q-learning via bootstrapping error reduction. *Advances in Neural Information Processing Systems*, 32, 2019.

[13] Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative q-learning for offline reinforcement learning. *Advances in Neural Information Processing Systems*, 33:1179–1191, 2020.

[14] Sergey Levine and Vladlen Koltun. Guided policy search. In *International conference on machine learning*, pages 1–9. PMLR, 2013.

[15] Xue Bin Peng, Aviral Kumar, Grace Zhang, and Sergey Levine. Advantage-weighted regression: Simple and scalable off-policy reinforcement learning. *arXiv preprint arXiv:1910.00177*, 2019.

[16] Doina Precup, Richard S Sutton, and Sanjoy Dasgupta. Off-policy temporal-difference learning with function approximation. In *ICML*, pages 417–424, 2001.

[17] Noah Y Siegel, Jost Tobias Springenberg, Felix Berkenkamp, Abbas Abdolmaleki, Michael Neunert, Thomas Lampe, Roland Hafner, Nicolas Heess, and Martin Riedmiller. Keep doing what worked: Behavioral modelling priors for offline reinforcement learning. *arXiv preprint arXiv:2002.08396*, 2020.

[18] Samarth Sinha, Ajay Mandlekar, and Animesh Garg. S4rl: Surprisingly simple self-supervision for offline reinforcement learning in robotics. In *Conference on Robot Learning*, pages 907–917. PMLR, 2022.

[19] Richard S Sutton, A Rupam Mahmood, and Martha White. An emphatic approach to the problem of off-policy temporal-difference learning. *The Journal of Machine Learning Research*, 17(1):2603–2631, 2016.

[20] Junming Yang. Github: Model based offline policy optimization. https://github.com/junming-yang/mopo.

[21] Tianhe Yu, Garrett Thomas, Lantao Yu, Stefano Ermon, James Y Zou, Sergey Levine, Chelsea Finn, and Tengyu Ma. Mopo: Model-based offline policy optimization. *Advances in Neural Information Processing Systems*, 33: 14129–14142, 2020.

[22] Tianhe Yu, Aviral Kumar, Rafael Rafailov, Aravind Rajeswaran, Sergey Levine, and Chelsea Finn. Combo: Conservative offline model-based policy optimization. *Advances in neural information processing systems*, 34: 28954–28967, 2021.