A Extended Experimental Results

A.1 Off-policy Setting: Risk-Averse performance

In this experiment, we intend to demonstrate the effectiveness of our algorithm as a risk-averse learner without introducing the extra layer of complexity of the offline setting.

A.1.1 Experimental Setup

As a toy example, we chose a 1-D car with state $s = (x, v)$, for position and velocity. The agent controls the car with an acceleration $a \in [-1, 1]$. The car dynamics with a time step $\Delta t = 0.1$ is

$$x_{t+1} = x_t + v_t \Delta t + 0.5 a_t (\Delta t)^2, \qquad v_{t+1} = v_t + a_t \Delta t.$$

The control objective is to move the car to $x_{\mathrm{g}} = 2.5$ as fast as possible, starting from rest. The episode terminates after 400 steps or when the agent reaches the goal.

To model the risk of crashing or of getting a speed fine, we introduce a penalization when the car exceeds a speed limit ($v > 1$). Hence, we use a random reward function given by

$$R_t(s, a) = -10 + 370 \mathbb{I}_{x_t = x_{\mathrm{g}}} - 25 \mathbb{I}_{v_t > 1} \cdot \mathcal{B}_{0.2},$$

where $\mathbb{I}$ is an indicator function and $\mathcal{B}_{0.2}$ is a Bernoulli Random Variable with probability $p = 0.2$. That is, $r_f = 370$ is a sparse reward that the agent gets at the goal and $r_d = -10$ is a negative reward that penalizes delays on reaching the goal. Finally, the agent receives a negative reward of $r_v = -25$ with probability 0.2 when it exceeds the $v > 1$ threshold. As the returns is a sum of bernoulli R.V. we know that it will be a Binomial distribution. For this particular case, we expect that if the number of steps is large enough, the Gaussianity assumption that WCPG does is good as Binomial distributions are asymptotically Gaussian (Vershynin, 2018). However, the episode terminates after at most thirteen risky steps and the approximation is not good.

Benchmarks We compare RAAC with a risk-neutral algorithm (D4PG algorithm by Barth-Maron et al. (2018) with IQN and 1-step returns) and WCPG by Tang et al. (2020), a competing risk-averse algorithm. Both RAAC and WCPG optimize the 0.1-CVaR of the returns.

A.1.2 Result Discussion

We ran the Car environment for RAAC, D4PG and WCPG using 5 independent random seeds. We evaluate final policies for 1000 interactions and report the averaged results with corresponding standard deviation in Table 2. As expected, D4PG ignores the low probability penalties and learns to accelerate the car with maximum power. Thus it has the largest expected return but the lowest CVaR. WCPG also fails to maximize the CVaR as it assumes that the return distribution is Gaussian. In turn, it under-estimates the variance of the return distribution of the maximum acceleration policy. Consequently, it over-estimates the CVaR of the returns and prefers the latter policy over the maximum-CVaR policy. In contrast, RAAC learns the full value distribution Z and computes the CVaR reliably. Thus, it learns to saturate the velocity below the $v = 1$ threshold and finds the highest CVaR policy.

Qualitative Evaluation In Figure 3 we show the different trajectories as an illustration of the resulting risk-averse behavior.

Table 2: Results of RAAC, WCPG, and D4PG in the car example. RAAC learns a policy that saturates the velocity before the risky region. WCPG and D4PG learn to accelerate as fast as possible, reaching the goal first with highest average returns but suffer from events with large penalty. We report mean (standard deviation) of each quantity.

Algorithm	$\mathrm{CVaR}_{0.1}$	Mean	Risky Steps	Total Steps
RAAC	**48.0 (8.3)**	48.0 (8.3)	**0 (0)**	33 (1)
WCPG	15.8 (3.3)	**79.8 (1.3)**	13 (0)	**24 (0)**
D4PG	15.6 (4.4)	**79.8 (2.0)**	13 (0)	**24 (0)**

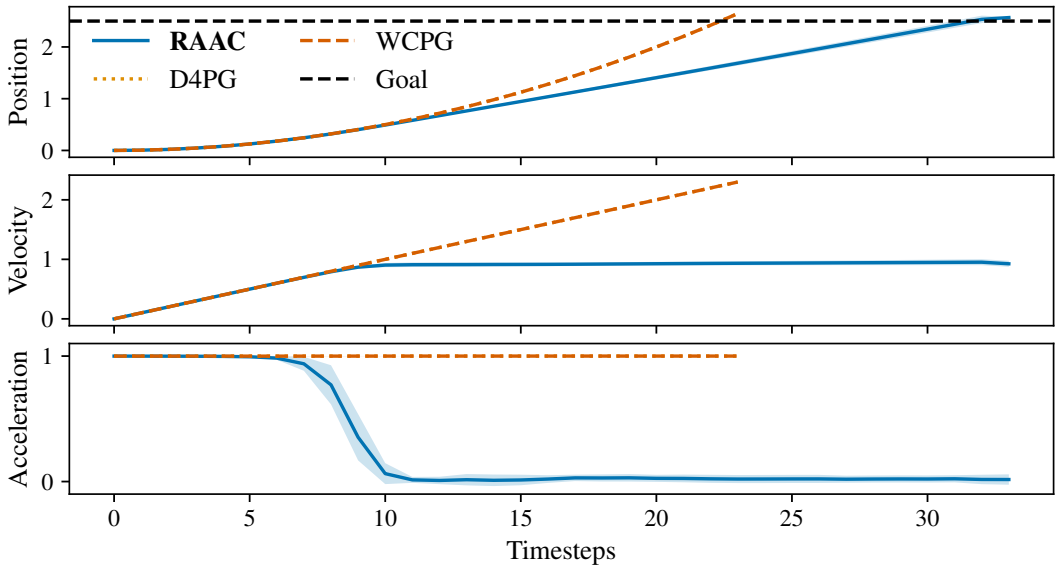

Figure 3: Evolution of car states and input control when following learned policies for RAAC, WCPG and D4PG (WCPG and D4PG are the same). We use policies from 5 independent seeds for each algorithm. RAAC learns to saturate the velocity below the speed limit.

A.2 MuJoCo Environments

We ran 5 independent random seeds and evaluate for 20 episodes the policy every 100 gradient steps for *HalfCheetah* and 500 gradient steps for *Hopper* and *Walker2d*. We plot the learning curves of the medium variants in Figure 4 and expert variant in Figure 5. To report the tests in Table 1, we early-stop the policy that outputs the best CVaR and evaluate on 100 episodes with 5 different random seeds.

Behavior policies For sake of reference, we evaluate the stochastic reward function on the state-action pairs in the behavior data set. Unfortunately, the data sets do not distinguish between episodes. Hence, to estimate the returns, we use the state-action distribution in the data set and split it into chunks of 200 time steps for the Half-Cheetah and 500 time steps for the Walker2D and the Hopper. We then compute the return of every chunk by sampling a realization from its stochastic reward function. Finally, we bootstrap the resulting chunks into 10 datasets by sampling uniformly at random with replacement and estimate the mean and $\text{CVaR}_{0.1}$ of the returns in each batch. We report the average of the bootstrap splits together with the standard deviation in Table 1.

A.3 Additional Experimental Details

A.3.1 Architectures

We use neural networks as function approximators for all the elements in the architecture.

Critic architecture: For the critic architecture, we build on the IQN network Dabney et al. (2018) but we extend it to the continuous action setting by adding an additional action input to the critic network, resulting in the function:

$$Z(s, a; \tau) = f(m_{sa\tau}(\left[m_{sa}([\psi_s(s), \psi_a(a)]), \psi_\tau(\tau)\right]), \tag{11}$$

where $\psi_s : \mathcal{X} \to \mathbb{R}^d$, $\psi_a : \mathcal{A} \to \mathbb{R}^d$, $m_{sa} : \mathbb{R}^{d \times d} \to \mathbb{R}^n$, $m_{sa\tau} : \mathbb{R}^n \to \mathbb{R}^n$, $\psi_\tau : \mathbb{R} \to \mathbb{R}^n$ and $f : \mathbb{R}^n \to \mathbb{R}$

For the embedding ψ_τ we use a linear function of n cosine basis functions of the form $\cos(\pi i \tau)$ $i = 1.., n$, with $n = 16$ as proposed in Dabney et al. (2018). For ψ_s, ψ_a we use a multi-layer perceptron (MLP) with a single hidden layer with $d = 64$ units for the Car experiment and with $d = 256$ units for all MuJoCo experiments. For the merging function m_{sa}, which takes as an input

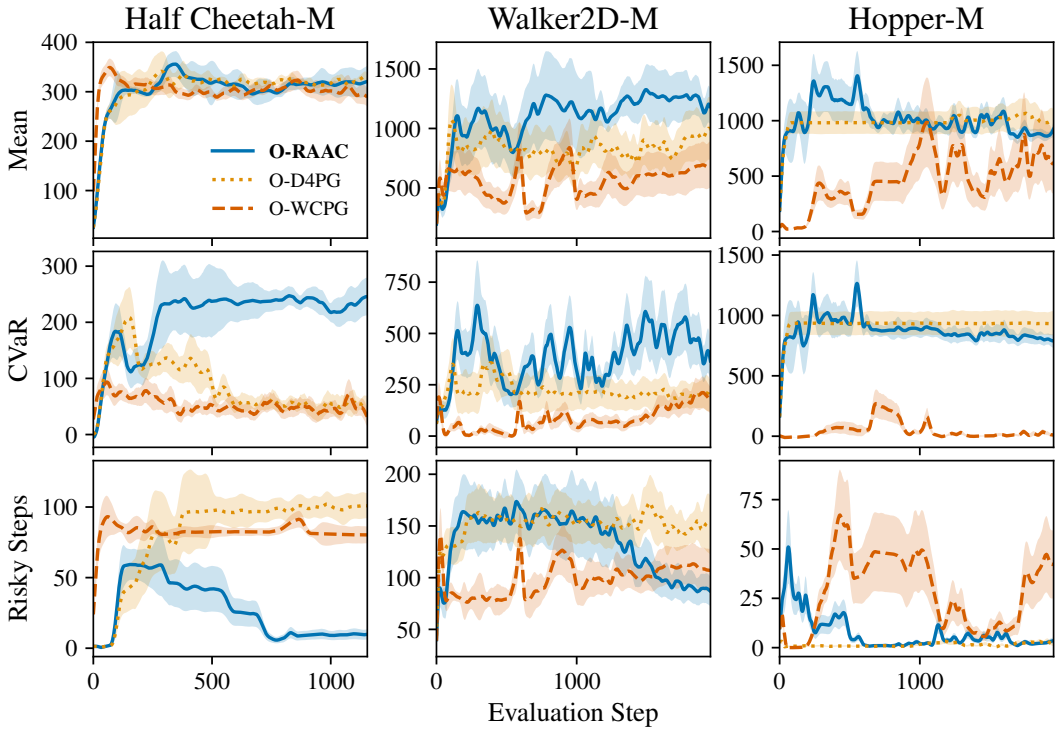

Figure 4: Experimental results across several Mujoco tasks for the Medium variant of each dataset.

the concatenation of $\psi_s(s)$ and $\psi_a(a)$, we use a single hidden layer with $n = 16$ units. For the merging function $m_{sa\tau}$, we force interaction between its two inputs via a multiplicative function $m_{sa\tau}(u_1, u_2) = u_1 \odot u_2$, where $u_1 = m_{sa}(\psi_s(s), \psi_a(a))$ and $u_2 = \psi_\tau(\tau)$ and $\odot$ denotes the element-wise product of two vectors. For f we use a MLP with a single hidden layer with 32 units We used ReLU non-linearities for all the layers.

Actor architecture: The architecture of the actor model is

$$\pi(a|s) = b + \lambda \xi_\theta(s, b) \tag{12}$$

where $\xi : \mathcal{A} \to \mathbb{R}^{\|\mathcal{A}\|}$ and b is the output of the imitation learning component. For the RAAC algorithm we remove b and set $\lambda = 1$.

For the Car experiments, we used a MLP with 2 hidden layers of size 64. For the MuJoCo experiments, based on Fujimoto et al. (2019), we used a MLP embedding with 3 hidden layers of sizes 400, 300 and 300. We used ReLU non-linearities for all the hidden layers and we saturate the output with a Tanh non-linearity.

VAE architecture: The architecture of the conditional VAE$_\phi$ is also based on Fujimoto et al. (2019). It is defined by two networks, an encoder $E_{\phi_1}(s, a)$ and decoder $D_{\phi_2}(s, z)$. Each network has two hidden layers of size 750 and it uses ReLU non-linearities.

A.3.2 Hyperparameters

All the network parameters are updated using Adam (Kingma & Ba, 2015) with learning rates $\eta = 0.001$ for the critic and the VAE, and $\eta = 0.0001$ for the actor model, as in Fujimoto et al. (2019). The target networks for the critic and the perturbation models are updated softly with $\mu = 0.005$.

For the critic loss (4) we use $N = N' = 32$ quantile samples, whereas to approximate the CVaR to compute the actor loss (5) (7) we use 8 samples from the uniform distribution between $[0, 0.1]$.

In Figure 6, we show an ablation on the effect of the hyper-parameter lambda. As we can see, a correct selection of lambda is of crucial performance as it trades-off pure imitation learning with

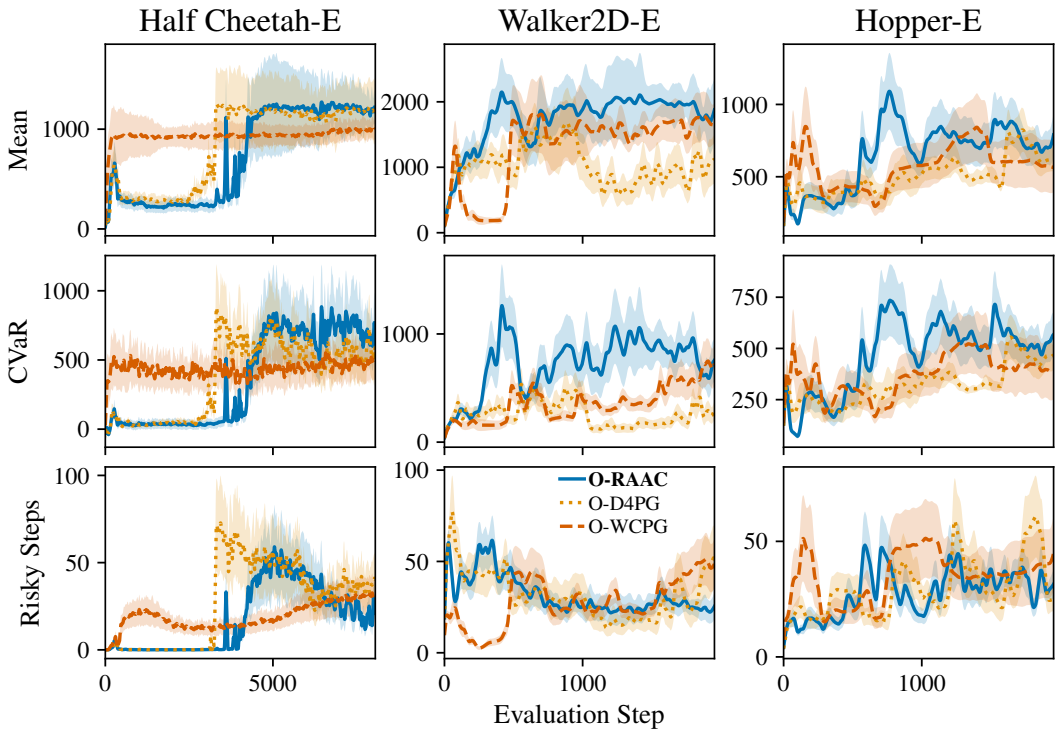

Figure 5: Experimental results across several Mujoco tasks for the Expert variant of each dataset.

pure reinforcement learning. As $\lambda \to 0$, the policy imitates the behavior policy has poor risk-averse performance. As $\lambda \to 1$, the policy suffers from the bootstrapping error and the performance is also low. We find values of $\lambda \in [0.05, 0.5]$ to be the best, although the specific λ is environment dependent. This observation coincides with those in Fujimoto et al. (2019, Appendix D.1).

For all MuJoCo experiments, the λ parameter which modulates the action perturbation level was experimentally set to 0.25, except for the HalfCheetah-medium experiment for which it was set to 0.5. As we can see from Figure 6, this is not the best value of λ, but rather a value that performs well across most environments.

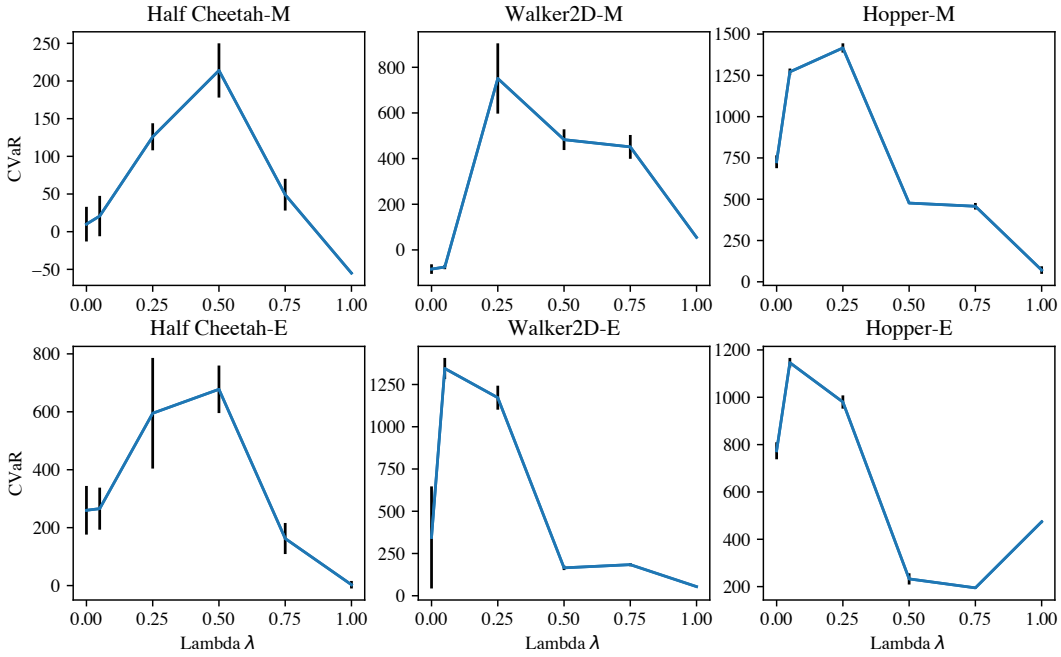

Figure 6: Effect of the hyperparameter λ on the CVaR of the returns for each of the MuJoCo environments. As $\lambda \to 0$, the policy imitates the behavior policy has poor risk-averse performance. As $\lambda \to 1$, the policy suffers from the bootstrapping error and the performance is also low. The best λ is environment dependent.