Deep Learning Lab 2022 SS

Worst-Case Soft Actor-Critic for Safety-Constrainted RL: A PyTorch Toolkit

Luca Pfrang, Akshay L Chandra, Sri Harsha Koribille
{pfrangl, lagandua, koribils}@cs.uni-freiburg.de

Supervior: Baohe Zhang

Abstract

In safety-critical domains where free exploration is expensive or infeasible, we turn to safety-constrained reinforcement learning (Safe-RL) methods that incentivize safe exploration. In this setting, the agent gets additional safety signals a.k.a costs. With separate reward and cost signals, Safe-RL methods are often modelled as a constrained optimization problem where policies are learned by estimating both expected long-term rewards as well as costs. In this project, we explore, implement and slightly extend a recent work [11] that learns optimal policies by estimating long-term cost distribution (i.e., mean and variance). We also extend the Worst-Case Soft Actor Critic (WCSAC) by smoothing the policy through smoothness-inducing regularization. Our code, results and this report are available at https://github.com/acl21/wcsac/.

1. Introduction

In traditional RL problems, agents are typically allowed to explore environments without having to worry about their safety. However, unsafe exploration in safety-critical problems can lead to either severe damage of the environment or the agent itself. That is why research interest in RL methods with safety as priority is growing ever more but it is still an open problem. We refer the readers to [1, 9–11] for a thorough review of recent efforts in Safe-RL. In this work, we focus our attention on the Worst-Case Soft Actor-Critic (WCSAC) algorithm proposed by Yang et al. [11] that uses a separate safety critic to estimate the distribution of long-term accumulated cost to achieve risk control. Section 2 includes necessary background, Section 3 contains the main algorithm we re-implement in PyTorch and Section 4 explains our contributions, findings, and our attempt to extend [11]. Section 5 has all the experimental details, hyperparameters, architectural choices, etc. Section 6 has our experimental findings and discussion.

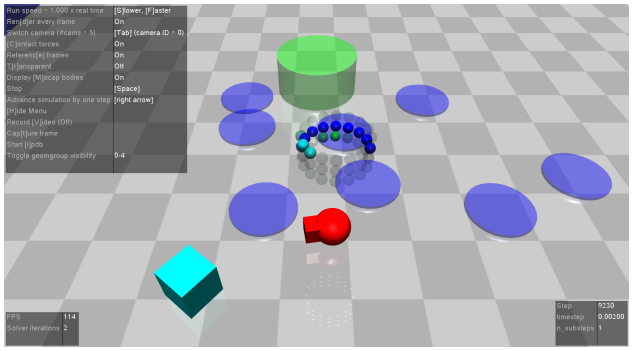


Figure 1. Screenshot of the `Safexp-PointGoal1-v0` environment in Safety-Gym [6]. Agent in red is expected to reach the green cylindrical goals, avoiding circular hazards in purple on the way. The cube in cyan is a vase, the agent is not penalized for hitting it.

2. Background

In Safe-RL the agent maximizes reward without causing harm to itself or the environment, also referred to as the safe exploration problem. In [6] the authors propose Constrained Reinforcement Learning as a general framework for solving the safe exploration problem. In this framework, safety is formulated as a constraint on cost the agent has to adhere. The WCSAC algorithm offers a way of solving the safe exploration problem by learning long term costs associated with agent behaviour. More specifically, it is build upon the Soft Actor Critic (SAC) algorithm [3] and the SAC-Lagrangian algorithm [1] as introduced below.

2.1. Soft Actor Critic

In Maximum Entropy Reinforcement Learning the standard reward objective is extended to include an entropy term

on the policy, forcing the actor to output stochastic actions:

$$J(\pi) = \sum_{t=0}^{T} \mathop{\mathbb{E}}_{(s_t,a_t)\sim\rho_\pi} [r(s_t,a_t) - \beta\mathcal{H}(\pi(a_t|s_t))].$$

Most prominently, the Soft Actor Critic algorithm achieves stable performance on various continous control tasks, while being less brittle to hyperparameters than comparable algorithms [3].

The SAC algorithm is easiest derived from its tabular representative: Soft Policy Iteration. In Soft Policy Iteration the policy update equates to minimizing the KL-Divergence between the current policy and the exponentiated state-action values, as shown below.

$$\pi_{new} = \arg\min_{\pi'\in\Pi} D_{KL}\left(\pi'(\cdot,s_t)\middle\|\frac{exp(\frac{1}{\beta}Q^{\pi_{old}}(s_t,\cdot))}{Z^{\pi_{old}}(s_t,\cdot)}\right) \tag{1}$$

Similarly, for its deep RL and off-policy variant, we take the expectation over Equation 1. For a more detailed description of the algorithm, we refer the reader to [3].

2.2. SAC-Lagrangian

SAC-Lagrangian is an extension of the SAC algorithm to Safe-RL [1]. Besides training a reward and entropy critic, SAC-Lagrangian additionally learns a cost critic, subsequently referred to as safety critic. In line with the Constrained RL framework, adhering to a certain cost level is formulated as a constraint on the reward objective, as shown in Equation 2:

$$\begin{aligned} &\max_{\pi} \mathop{\mathbb{E}}_{(s_t,a_t)\sim\rho_\pi}\left[\sum_t \gamma^t r(s_t,a_t)\right] \\ &\text{s.t.} \mathop{\mathbb{E}}_{(s_t,a_t)\sim\rho_\pi}\left[\sum_t \gamma^t c(s_t,a_t)\right] \leq d. \end{aligned} \tag{2}$$

Intuitively, the expected long term cost of the agents current trajectory is informing the actor about how well it is adhering to the cost constraint. Constrained optimization problems can be solved with the Lagrangian-multiplier method by solving the Lagrangian as described below:

$$\begin{aligned} &\min_{\kappa\geq 0}\max_{\pi} \mathcal{L}(\pi,\kappa) \doteq f(\pi) - \kappa g(\pi), \\ &\text{where } f(\pi) = \mathop{\mathbb{E}}_{(s_t,a_t)\sim\rho_\pi}\left[\sum_t \gamma^t r(s_t,a_t)\right] \\ &\text{and } g(\pi) = \mathop{\mathbb{E}}_{(s_t,a_t)\sim\rho_\pi}\left[\sum_t \gamma^t c(s_t,a_t)\right] - d, \end{aligned} \tag{3}$$

where κ denotes the Lagrangian-multiplier for the safety constraint. Yang et. al. [11] build on the approach by Haarnoja et. al. [2] to adaptively learn the safety temperature κ, in addition to the entropy temperature β. The intuition behind this is that we do not want to enforce safety and stochasticity where it is not needed. We rather want the policy decide where it needs entropy and safety, while satisfying the respective constraints on average. Consequently, the actor loss is formulated as:

$$\begin{aligned} J_\pi(\theta) = \mathop{\mathbb{E}}_{s_t\sim D_t, a_t\sim\pi_\theta} [&\beta\log\pi_\theta(a_t|s_t) - Q^r_{\pi_\theta}(s_t,a_t) \\ &+\kappa Q^c_{\pi_\theta}(s_t,a_t)] \end{aligned} \tag{4}$$

where D is the replay buffer and θ indicates parameters of the policy π. Consequently, κ is learnt by minimizing the loss:

$$J_\kappa(\theta) = \mathop{\mathbb{E}}_{s_t\sim D_t, a_t\sim\pi_\theta} [\kappa(d - Q^c_{\pi_\theta}(s_t,a_t))], \tag{5}$$

increase when safety is violated and decrease whenever safety is met. Similarly, we learn β by minimizing:

$$J_\beta(\theta) = \mathop{\mathbb{E}}_{s_t\sim D_t, a_t\sim\pi_\theta} [\beta(-\bar{\mathcal{H}} - \log(\pi_\theta(s_t,a_t))].$$

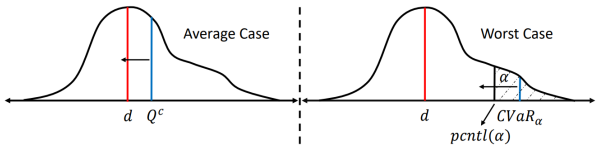


Figure 2. Comparison of risk avoidance strategy in SAC-Lagrangian (left) and WCSAC (right). SAC-Lagrangian pushes the mean of the distribution to satisfy the safety threshold d, resulting in *average case* safety. In contrast, WCSAC forces the right tail of the distribution to be beneath the safety threshold, resulting in *worst case* safety. Image Source: [11]

3. Worst-Case Soft Actor Critic

SAC-Lagrangian models the long-term cost of following a policy as its expected value. However, given uncertainty inherent to the environment and the policy, the expected long-term cost can consistently underestimate the potential risk of the agents trajectory. Worse, this behavior can go undetected by the algorithm, if the expected cost is satisfying the constraint, while the true observed cost violates the constraint. For this reason, the authors of [11] follow [10] and learn the distributional safety critic. Consequently, [11] propose focusing on the right tail of the cost distribution for worst case awareness.

Similar to [10], [11] approximate the distributional safety critic $C_\pi(s,a)$ as a Gaussian, i.e.,

$$C_\pi(s,a) \sim \mathcal{N}(Q^c_\pi(s,a), V^c_\pi(s,a))$$

where the variance $V_\pi^c(s,a) = \mathbb{E}_{p^\pi}[C^2|s,a] - (Q_\pi^c(s,a))^2$. To estimate Q_π^c, the standard Bellman function is used:

$$Q_\pi^c(s,a) = c(s,a) + \gamma \sum_{s' \in S} p(s'|s,a) \sum_{a' \in A} \pi(a'|s') Q_\pi^c(s',a'). \quad (6)$$

where $s' \sim p(\cdot|s,a)$, $a' \sim \pi(\cdot|s')$. The projection equation for estimating $V_\pi^c(s,a)$ is:

$$\begin{aligned} V_\pi^c(s,a) = c(s,a)^2 - Q_\pi^c(s,a)^2 \\ + 2\gamma c(s,a) \sum_{s' \in S} p(s'|s,a) \sum_{a' \in A} \pi(a'|s') Q_\pi^c(s',a') \\ + \gamma^2 \sum_{s' \in S} p(s'|s,a) \sum_{a' \in A} \pi(a'|s') V_\pi^c(s',a') \\ + \gamma^2 \sum_{s' \in S} p(s'|s,a) \sum_{a' \in A} \pi(a'|s') Q_\pi^c(s',a') \end{aligned} \quad (7)$$

We refer the readers to [10] for the proof of this equation. Consequently, the authors train two neural networks to estimate the safety critic, i.e., Q_π^c and V_π^c.

Figure 2 depicts the cost distribution p^π given a state and action pair. The SAC-Lagrangian optimizes the policy by changing the shape of the distribution until Q_π^c (blue line) is shifted to the left of the boundary (red line). However for risk averse learning, it is desired to minimize the expected worst case performance instead of the average case. The authors accomplished this by using Conditional Value-at-Risk (CVaR) [7] instead of expected cost. CVaR represents the expected cost if we were to experience the top α percentile of the possible outcomes

$$CVaR_\alpha \doteq \mathbb{E}_{p^\pi}[C|C \geq pcntl(\alpha)] \quad (8)$$

where $pcntl(\alpha)$ is the α percentile of p^π. Subsequently, α is also referred to as the risk-level. As $\alpha \to 0$, the policy becomes more pessimistic and risk-averse and with increasing α, the policy becomes more risk-neutral. By varying the risk-level, the CVaR allows us to control the strength of risk aversion we want to enforce on the policy.

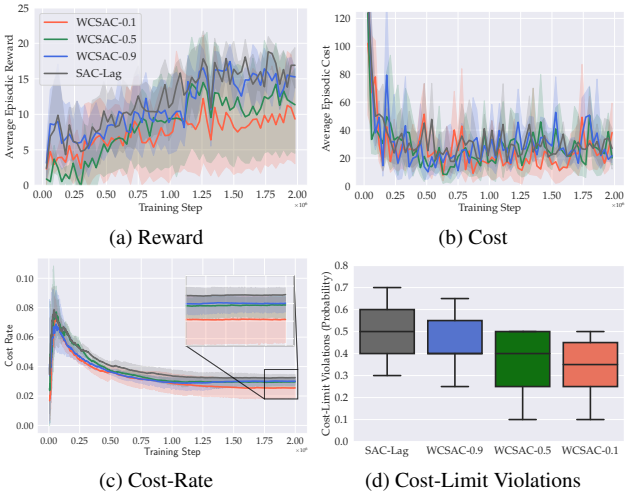
Given the worst case estimation of the expected cost of our current policy the actor loss in Equation 4 is modified to incorporate the $CVaR_\alpha$:

$$J_\pi(\theta) = \mathop{\mathbb{E}}_{s_t \sim D_t, a_t \sim \pi_\theta} [\beta \log \pi_\theta(a_t|s_t) - Q_{\pi_\theta}^r(s_t,a_t) + \kappa \Gamma_\pi(s_t,a_t,\alpha)], \quad (9)$$

with $CVaR_\alpha = \Gamma_\pi(s,a,\alpha)$. As a consequence, Equation 5 changes to:

$$J_\kappa(\theta) = \mathop{\mathbb{E}}_{s_t \sim D_t, a_t \sim \pi_\theta} [\kappa(d - \Gamma_\pi(s_t,a_t,\alpha))]. \quad (10)$$

For a complete description of the entire algorithm, we refer to the original paper [11].



(a) Reward (b) Cost

(c) Cost-Rate (d) Cost-Limit Violations

Figure 3. Our WCSAC benchmarks on Safety-Gym's `Safexp-PointGoal1-v0` environment. The lines are the average of all runs, and the shaded area is the standard deviation.

4. Our Contributions

4.1. A PyTorch Toolkit for WCSAC and Safe-RL

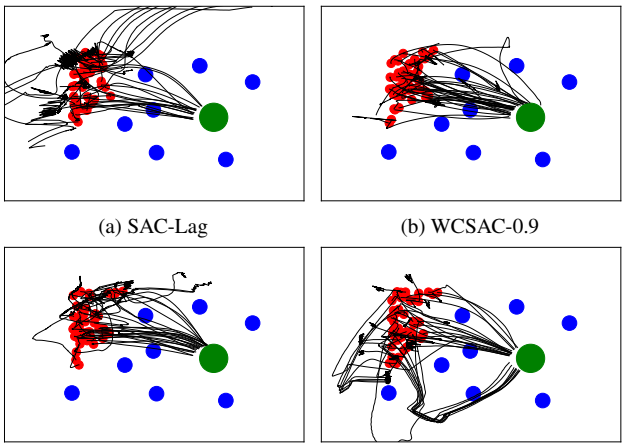
We re-implemented and benchmarked the original TensorFlow implementation of WCSAC[1] in PyTorch, the most popular deep learning framework as of August 2022[2]. Compared to the original implementation, our toolkit is more modular, easier to extend, uses a better configuration management framework, i.e., Hydra, and can act as a base code repository for any future Safe-RL projects. Built around OpenAI's Safety-Gym [6], our benchmarks on more complex safety environments are in parallel to the official results reported in [11]. However, our toolkit doesn't support multi-GPU training at the moment and we hope to change this in the near future.

4.2. WCSAC+: WCSAC with Policy Smoothing

In theory, learning worst case estimates of the long term expected cost is a principled way to solve the safe exploration problem. Additionally, our experiments show that we can replicate the performance as shown in the WCSAC paper [11]. We even generalize to a more complex environment.

In practice, we observe that increased safety also reliably comes at the cost of reduced reward. As much as this outcome is expected, we also observe some unwanted agent behaviours. When we looked closely at the safest policies in action, we noticed a consistent yet undesirable agent behaviour - whenever the agent was in immediate danger (see Figures 4(c) and 4(d)). The agent essentially faces a con-

[1] https://github.com/AlgTUDelft/WCSAC/
[2] https://paperswithcode.com/trends



(a) SAC-Lag (b) WCSAC-0.9

(c) WCSAC-0.5 (d) WCSAC-0.1

Figure 4. Trajectory Analysis. Green circle shows the goal, blue circles are hazards and the red circles are random agent starting points at different evaluation episodes.

flict between going through a hazardous area and avoiding it, resulting in it being paralysed or stuck in some situations. Roughly speaking, the agent is *scared* to commit to a move when in some states near hazards. Therefore, despite the principled approach we hypothesize that the WCSAC algorithm suffers from various practical limitations, possibly arising from long understood RL optimization artifacts.

Motivated to avoid the above-mentioned behaviour, we propose a two-fold solution. Firstly, we improve the actor's ability to generalize with weight decay [4], to that end we simply optimize the actor with AdamW. Secondly, we attempt to obtain smoother policies by optimizing actor weights with smoothness regularization, inspired by [8]. Smoothness doesn't come for free especially when dealing with overparameterized deep neural networks in deep RL. Famously, the reparametriziation trick is employed when dealing with continuous action space and we want to take one step further in this direction and smoothen the policy even more. Our smoothness-inducing regularization encourages the output of the policy to not change much when injecting small perturbation to the input of the policy (observed state). Our intuition is that, if the agent is forced to have smoother trajectories, the policy could potentially learn to plan better paths instead of twitchy sharp tuning ones.

To that end, slightly modifying the proposed method in [8], we define the smoothness regularization term as follows: for a fixed state $s \in S$ and a policy π, we encourage the output $\pi(s)$ and $\pi(\tilde{s})$ to be similar, where state $\tilde{s}$ is obtained by injection of a small perturbation to state s. We assume the perturbation set $\mathcal{B}(s, \epsilon) = \{\tilde{s} : ||s - \tilde{s}||_2 \leq \epsilon\}$ is an ϵ-radius ball around s in Euclidean space. To obtain a smooth policy , we encourage smoothness at each state

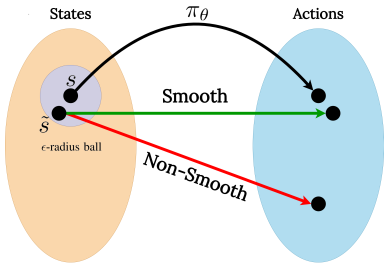


Figure 5. Simplified illustration of policy smoothing.

of the entire trajectory. We achieve this by minimizing the following smoothing loss:

$$J_{\pi_2}(\theta) = \mathop{\mathbb{E}}_{s_t \sim D_t, \tilde{s}_t \sim \mathcal{B}(s,\epsilon)} [\lambda_s(D_J(\pi_\theta(\cdot|s), \pi_\theta(\cdot|\tilde{s})))]$$

where $D_J(\cdot, \cdot)$ is Jeffery's Divergence and λ_s is the smoothness regularization coefficient. The Jeffrey's divergence for two distributions P, Q is defined by:

$$D_J(P, Q) = \frac{D_{KL}(P||Q) + D_{KL}(Q||P)}{2}$$

where D_{KL} is Kullback–Leibler divergence.

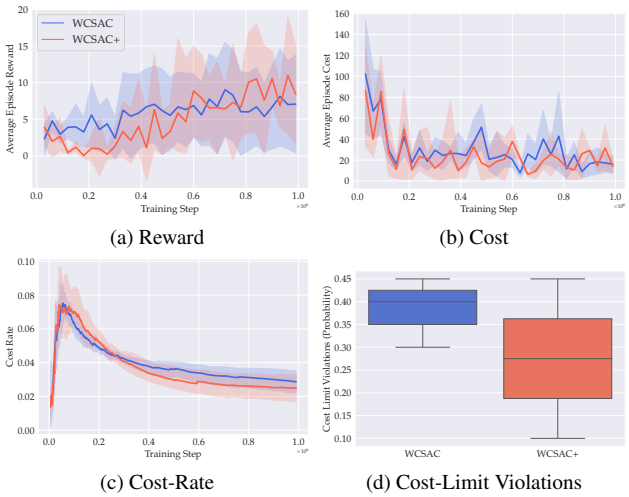
Figure 5 shows a simplied illustration of policy smoothing. We argue that our proposed variant could potentially side-step some of the practical limitations, as explained above. Specifically, we hope that smoothing the actor prevents the agent from paralyzing, since regularization improves the actor's ability to incorporate constraints. Note that we fully understand the risk of smoothing the policy as the agent might have to make sharp turns in some situations to avoid hazards. However, as our environment uses static hazards (within episodes) we argue that our choice is reasonable for this setting. We emphasize that a deeper investigation of the described limitations is necessary, but out of scope for our project.

5. Experimental Setup

We benchmark WCSAC and evaluate our variant, WCSAC+, against the former on the Safety-Gym[3] benchmark [6]. We specifically chose a more complex safety environment than used in [11] for novel contribution reasons. As shown in Figure 1, a point agent navigates in a 2D environment to reach the green cylindrical goals while trying to avoid the hazardous areas in blue. All our experiments were evaluated on `Safexp-PointGoal1-v0`, in this, one vase is randomly placed in the scene, but the agent is not penalized for hitting it.

To the best of our abilities, we set the hyperparameters exactly similar to the original implementation, except the

[3] https://github.com/openai/safety-gym



(a) Reward (b) Cost

(c) Cost-Rate (d) Cost-Limit Violations

Figure 6. WCSAC+ vs WCSAC ($\alpha = 0.1$)

learning rate for optimizing κ, i.e., Equation (5). The learning rate set in the original implementation, 0.05, made κ explode in our runs after a few iterations. The following are the hyperparameters used in our benchmark experiments: $lr = 0.001$ with Adam optimizers for all learners (actor, double-Q critic, safety-critic, also β and κ learners), discount $\gamma = 0.99$, target network smoothing temperature $\tau = 0.005$, cost-limit $d = 15$, batch size 256. For all learners, we used an MLP with two intermediate layers, 256 neurons each. A full list of these hyperparameters are available at `config/agent/wcsac.yaml` in our repository. All the agents are trained for 2 million episodes, where the maximal length of each episode is 1000 environment interaction steps. The agent is evaluated every 30000 steps on 20 episodes. All benchmarks were averaged over 5 random seeds. Overall, we compare SAC-Lagrangian (SAC-Lag) and WCSAC at two different risk levels in safety, i.e., WCSAC-0.9 ($\alpha = 0.9$: almost risk-neutral) and WCSAC-0.1 ($\alpha = 0.1$: highly risk-averse).

WCSAC+ experiments have similar hyperparameters to WCSAC except the actor optimizer being AdamW (weight decay). Due to computational and time restrictions, we were only able to average WCSAC+ results over 3 random seeds, with training done for only one million steps. We set risk-level $\alpha = 0.1$ and choose $\epsilon = 0.001$ and $\lambda_s = 1$ based on a quick grid-search (evaluated on 150,000 training steps). For baseline, we re-use the 5 random seed WCSAC-0.1 runs from benchmark experiments.

To analyse agent trajectories in the environment, we take each algorithm's final training step's saved policy from all 5 seeds and evaluate it on 10 episodes (maximum 1000 steps). For consistency, we re-use a our environment with fixed hazards and a fixed goal over the course of the evaluation episodes.

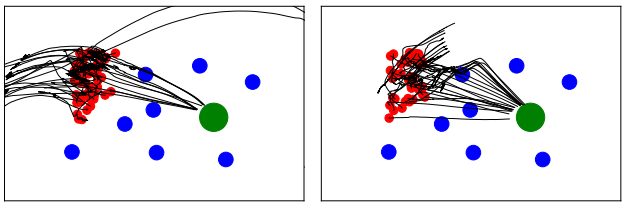
6. Results and Discussion

WCSAC Benchmarks: Our WCSAC benchmarks report the following metrics: average episodic return, average episodic cost, cost-rate at each environment interaction step (cumulative costs divided by number of steps) and cost-limit violation statistics. Cost-limit violations show how many times out of 20 episodic evaluations the agent, post training, broke the set cost-limit. Figure 3 shows all the benchmark results. We can observe in Figures 3(a) and 3(b) all algorithms learn to reach the goal (fairly high episodic rewards). However, the risk-neutral algorithms, SAC-Lag and WCSAC-0.9, do not satisfy safety constraint the most (higher episodic costs as well), which is expected of them. We can also see in Figure 3(c), that throughout the training steps, risk-averse WCSAC-0.1 has progressively lower cost-rate on average than the other algorithms. As expected, WCSAC-0.5 slots right between WCSAC-0.9 and WCSAC-0.1 in performance. This is also supported by the evidence in Figure 3(d). Final WCSAC-0.1 policies are least likely to violate the cost-limit while SAC-Lag and WCSAC-0.9 policies are more likely to do that than WCSAC-0.5 policies, which again are more likely to violated the limit than WCSAC-0.1 policies. All these results are consistent with the reported results in the WCSAC paper [11].

Agent Trajectory Analysis: Figure 4 compares agent trajectories of all five final polices of each algorithm. A pattern emerges here: SAC-Lag and WCSAC-0.9 agents, both considered on the risk-neutral side of the safety spectrum, behave as if they do not care about the hazards, see Figures 4(a) and 4(b). In contrast, in Figures 4(c) and 4(d), we see that WCSAC-0.5 and WCSAC-0.1 agents appear to be more keen on avoiding the hazards. Figure 4(d) particularly highlights this where the WCSAC-0.1 agent is taking a longer route to avoid hazards.

However, noticeably, in many of the runs the risk-averse agents are simply too safe i.e., they are safe at the cost of not reaching the goal. This harmful artifact of risk-averse agents on complex dynamic environments is not particularly addressed or discussed but can be clearly noticed from Figure 4(a) of the WCSAC paper [11]. There, WCSAC-0.1 achieves the maximum reward of only less than 10, even after training for three million steps. This clearly suggests that the agent does not always move towards the goal.

Even though the risk-level allows us to calibrate safety on a spectrum of risk neutrality to risk aversion, our observations indicate that we fail to trade off reward and cost reasonably for low risk-levels. Albeit this phenomenon could be probably be avoided by carefully tuning the risk-level to a reasonable perception of risk for the agent. Alternatively, we stress that despite the theoretically reasoned idea



(a) WCSAC (b) WCSAC+

Figure 7. Trajectory Analysis: WCSAC+ vs WCSAC ($\alpha = 0.1$)

of WCSAC, a more elaborate approach for trading off risk and reward might hold the key to more knowledgeable agent behaviour in the face of adversary.

Damping Trick for Better Convergence: In the official WCSAC implementation, we noticed that the authors employed the Modified Differential Method of Multipliers (MDMM) [5] by introducing a damping factor to prevent κ from converging to 0 prematurely. The MDMM modifies Equation 10 to the following:

$$J_\kappa(\theta) = \mathop{\mathbb{E}}_{s_t \sim D_t, a_t \sim \pi_\theta} [(\kappa - \delta)(d - \Gamma_\pi(s_t, a_t, \alpha))],$$

where δ is the damping strength. MDMM can generally be used when optimizing an additive objective, such as our Lagrangian. In theory, for non-convex Pareto fronts, as is generally the case for function approximation, MDMM avoids the Lagrange-multipliers to oscillate once a constraint is reached [5]. Note that this only controls the rate of converge but not the solution. In our preliminary runs, we did not use damping and immediately noticed a problem with κ optimization in some runs (not all): when the cost-limit (d) is high, κ tends to converge to 0 immediately after the learning starts. When d is low, the κ value tends to rise and then falls to converge to 0. And when the κ value is 0, the objective function essentially reduces to that of SAC's, making the cost signal useless while training. With damping (δ=10), we avoided this problem. To our surprise, this damping trick was not mentioned or discussed in the entire WCSAC paper. We want to emphasize that this damping trick is crucial for improving WCSAC's convergence rate.

WCSAC Hyperparameter Sensitivity: Overall, we found WCSAC algorithm to be quite sensitive to hyperparameter as well as the task dependent parameter d (cost-limit). Before exactly matching all the hyperparameters to that of the official implementation, we had a hard time with convergence as either β or κ exploded in value or the agent simply never learned anything meaningful. To be clear, all ablation were run for only one million steps without damping.

On `DynamicEnv-v0` Environment: In the WCSAC paper, the algorithm's performance is evaluated on a custom designed dynamic environment, called `DynamicEnv-v0`. During our ablation runs on `DynamicEnv-v0`, we noticed an obvious pattern emerge - despite being dynamic, the environment places the hazards either far apart from each other or bunches them together making it arguably easier for the agent to navigate. Even when trained for only 100000 steps, the agent was able to easily avoid the hazards simply because it came across one fewer times while trying to reach the goal. Note that `DynamicEnv-v0` only places three hazards randomly in the scene, compared to eight hazards in `Safexp-PointGoal1-v0` for the exact same placement area. Figure 8 shows nine randomly generated examples plotted to scale i.e., the edges represent true environment boundaries. To that end, we noticed the problem in the official code[4]. One of the parameters used to generate the environment, `hazard_keepout` controls the minimum distance between any two hazards. In `DynamicEnv-v0`, `hazard_keepout` is set to 0.305, compared to 0.18 of `Safexp-PointGoal1-v0`. This allows the hazards to be roughly two times as far apart as in the case of `Safexp-PointGoal1-v0`. Even in cases where the hazards are randomly bunched together, with `goal_keepout`, a parameter to control minimum distance between the goal and a hazard or the next goal, being set to 0.305, the environment is forced to place the goals away from the hazard bunch. We suspect this explains the differences between the paper's cost graphs and ours.

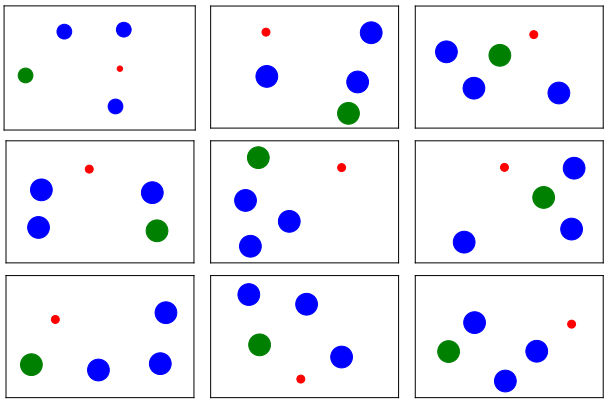


Figure 8. `DynamicEnv-v0` Environment. Not cherry picked.

WCSAC+: Figures 6 and 7 show the comparisons between our proposed variant WCSAC+ and WCSAC. After training for one million steps, both WCSAC+ and WCSAC agents are not very different in performance. Despite some encouraging signs in Figures 6(c) and 6(d), i.e., WCSAC+ has lower cost-rate and lower probability of cost-limit violations - we believe this is through pure random chance.

[4] See line 55 in https://github.com/AlgTUDelft/WCSAC/blob/main/wc_sac/sac/wcsac.py

Note that WCSAC results were averaged over 5 random seed runs, while WCSAC+ results were averaged over only 3 random seed runs. Agent trajectory analysis in Figure 7 confirms this. WCSAC+ and WCSAC do not show noticeable differences in agent trajectories. Hence, we are inclined to conclude that policy smoothing did not contribute towards better agent trajectories.

WCSAC+ Evaluations Need Better Hyperparameter Tuning: We performed a grid search to tune ϵ and λ_s, by training agents for only 150000 steps. Also, we used reward and cost-rate to choose the best combination and did not compare agent trajectories. Perhaps with better hyperparameter tuning and/or by evaluating after training for longer than one million steps and/or by making λ_s learnable, i.e., yet another Lagrange multiplier, we feel we can have a better understanding of the role of policy smoothing in Safe-RL. We leave these investigations for future work.

7. Conclusion

In this project, we successfully benchmark WCSAC algorithm that solves safety-constrained RL problems. In addition, we developed an open-source, modular PyTorch toolkit that can act as a base code-repository to any Safe-RL problem. Our benchmark results are consistent with the results reported in the WCSAC paper. We also attempt to extend WCSAC with policy smoothing - WCSAC+ is regularized to have smoother policies for better steering of the agent in the environment. However, our experiments show no convincing evidence of improved agent trajectories. We feel a more thorough investigation and hyperparameter tuning is necessary to deeply understand how, if at all, policy smoothing contributes positively to Safe-RL. Furthermore, we reason that although the WCSAC approach is principled, we observe limitations in practical performance. While the agent significantly reduces the cost rate at low risk-level, it disproportionately loses sight of its reward objective. For low risk-levels, WCSAC fails at subtly differentiating between a safe (although risky) and a unsafe policy. Therefore, we conclude that while managing to avoid costs, future work has to focus on making more subtle trade-offs between cost and reward.

8. Acknowledgements

Firstly, we thank our project supervisor Baohe Zhang for guiding us with valuable suggestions and insights.

The division of work within the team is as follows: Initially, Koribille provided code for SAC and SAC Lagrangian (our baseline). Using this implementation, Pfrang and Chandra wrote, debugged the WCSAC code. Pfrang formulated, proposed and wrote code for policy smoothing. Koribille handled all priliminary ablation runs and hyperparameter tuning - WCSAC benchmarks parameters and policy smoothing parameters (ϵ and λ_s). Chandra handled all the server side of things - from setting up the Safety-Gym environment to running and keep track of experiments, creating figures, etc. All three of us contributed equally to report writing and the poster.

References

[1] Sehoon Ha, P Xu, Zhenyu Tan, Sergey Levine, and Jie Tan. Learning to walk in the real world with minimal human effort. In *CoRL*, 2020. 1, 2

[2] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018. 2

[3] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, and Sergey Levine. Soft actor-critic algorithms and applications. *CoRR*, abs/1812.05905, 2018. 1, 2

[4] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *ArXiv*, abs/1711.05101, 2017. 4

[5] John C. Platt and Alan H. Barr. Constrained differential optimization. In *NIPS*, 1987. 6

[6] Alex Ray, Joshua Achiam, and Dario Amodei. Benchmarking Safe Exploration in Deep Reinforcement Learning. 2019. 1, 3, 4

[7] R Tyrrell Rockafellar, Stanislav Uryasev, et al. Optimization of conditional value-at-risk. *Journal of risk*, 2:21–42, 2000. 3

[8] Qianli Shen, Y. Li, Haoming Jiang, Zhaoran Wang, and Tuo Zhao. Deep reinforcement learning with smooth policy. *ArXiv*, abs/2003.09534, 2020. 4

[9] Krishna Parasuram Srinivasan, Benjamin Eysenbach, Sehoon Ha, Jie Tan, and Chelsea Finn. Learning to be safe: Deep rl with a safety critic. *ArXiv*, abs/2010.14603, 2020. 1

[10] Yichuan Tang, Jian Zhang, and Ruslan Salakhutdinov. Worst cases policy gradients. In *CoRL*, 2019. 1, 2, 3

[11] Qisong Yang, Thiago D. Simão, Simon H Tindemans, and Matthijs T. J. Spaan. Wcsac: Worst-case soft actor critic for safety-constrained reinforcement learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35, 2021. 1, 2, 3, 4, 5