Constrained reinforcement learning by reward augmentation

Florian Dorner, Arjun Bhardwaj, Mirlan Karimov, Tanmay Goyal

1 Introduction

Reinforcement learning (RL) is concerned with learning a policy that maximizes the expected cumulative reward for some scalar reward signal and Richard Sutton's reward hypothesis[1] states "That all of what we mean by goals and purposes can be well thought of as maximization of the expected value of the cumulative sum of a received scalar signal (reward)". While they acknowledge the reward hypothesis, researchers from OpenAI have recently argued that explicit constraints should be used to ensure the safe behaviour of RL agents in potentially dangerous applications, as formulating constraints comes more natural to humans than reasoning about reward signals[9]. We present an augmentation procedure that modifies an RL task given a constraint and a set of parameters, such that maximizing reward for the modified task can yield good results in terms of the original reward and constraint violation for the right parameters. This is done by augmenting the state space with information about the constraint and penalizing constraint violations, both directly and by reducing future rewards after a constraint violation. The augmented task can then be solved using RL such that, unlike custom algorithms for RL with constraints, our approach can trivially leverage recent progress in deep RL. As constraints are often about unsafe behaviour, especially in real world applications, it can be crucial that they are respected during training as early as possible. This makes sample efficient learning very important. Our approach allows us to use modern off-policy algorithms like SAC [6, 7] and TD3 [4] which are a lot more sample efficient in the unconstrained setting [3] than PPO [11] and TRPO [10], modified version of which are currently used for constrained reinforcement learning [9, 1]. We show that the optimal policy to our modified problem is equal to the best deterministic constraint-respecting policy in the deterministic case with strictly positive rewards, as long as there are no unavoidable delayed costs and derive more general conditions on performance. We validate our approach on the PointGoal1 task from the SafetyGym benchmark [9]: With the right parameters, SAC trained on our modified problem learns to respect the constraint using half as many environment steps as the Lagrange-TRPO baseline while achieving around 70% of the final reward. However, our approach's performance deteriorates for longer training.

2 Models and Methods

In the RL framework [13], an agent interacts with its environment and receives positive or negative reinforcement. The environment is modelled as a Markov decision process (MDP), in which starting with the initial state $s_0 \sim p(s_0)$, the agent sees a state s_t at time t and picks an action a_t according to a policy $\pi(a_t|s_t)$ causing a state change according to the dynamics $p(s_{t+1}|s_t, a_t)$. The agent then receives a reward r_t that depends on s_t, a_t and s_{t+1}. For a given MDP with bounded rewards and a policy π, the value function is defined as the expected cumulative discounted reward when executing the policy starting in s, $V_\pi(s) = \mathbb{E}_{\pi, s_0=s}[\sum_t \gamma^t r_t]$ for a fixed discount factor $\gamma < 1$. Similarly, the Q-function is defined as $Q_\pi(s, a) = \mathbb{E}_{\pi, s_0=s, a_0=a}[\sum_t \gamma^t r_t]$. The goal of RL is to find a policy that maximizes $R_\pi = \mathbb{E}_{s_0 \sim p(s_0)}[V_\pi(s_0)]$ and a policy π is called better than another policy π' if $V_\pi(s) \geq V_{\pi'}(s)$ for all states s. A policy is called optimal if it is better than every other policy and there always exists a deterministic optimal policy for a given MDP. The value function follows the Bellman equation $V_\pi(s) = \mathbb{E}_\pi[r + \gamma V_\pi(s')]$, where s' is the next state after following π for one step and can be arrived at from arbitrary starting values using a

[1] http://incompleteideas.net/rlai.cs.ualberta.ca/RLAI/rewardhypothesis.html

fixed point iteration based on that equation. Similarly, $Q_\pi(s,a) = \mathbb{E}_\pi[r + \gamma Q_\pi(s', a')]$ where s' is the next state after taking action a and a' is the action taken by π in that state.
An important class of RL algorithms are Q-based methods like Q-learning which aim to directly learn the Q-function for the optimal policy by replacing a' in the Bellman equation with $\arg\max_a Q(s,a)$ for the current estimate for Q and act by choosing the action with the highest estimated $Q-$value. These methods are off-policy methods, meaning that they can update from data that was generated by a behavioural policy different from the learnt policy. Off-policy methods can achieve better sample efficiency by reusing sampled environment transitions and update on them multiple times by storing them in a so-called replay buffer and updating on batches from that buffer. In deep RL[2] with continuous actions, the argmax over actions is infeasible, such that methods like DDPG [8] and TD3[4] instead learn a deterministic parameterized policy π_θ to select actions with high Q-values. This policy is updated using stochastic gradient ascent on the deterministic policy gradient $\nabla R_{\pi_\theta} = \mathbb{E}[\nabla_a Q_\theta(s,a)_{|a=\pi_\theta(s)} \nabla_\theta \pi_\theta(s)]$ [12]. Then the Q-function is updated by minimizing the Bellman residual $\mathbb{E}[(Q_\theta(\mathbf{s_t}, \mathbf{a_t}) - r(\mathbf{s_t}, \mathbf{a_t}) - \gamma Q_\theta(\mathbf{s_{t+1}}, \pi_\theta(\mathbf{s_{t+1}})))^2]$ using stochastic gradient descent (SGD) for transitions sampled from the replay buffer. Various tricks are employed to stabilize these methods[3]. Meanwhile, policy-gradient methods are usually on-policy and gradually modify a behaviour policy. They use the policy gradient theorem to estimate the gradient of R_{π_θ} with respect to policy parameters θ as $\nabla_\theta R_{\pi_\theta} = \mathbb{E}[Q_\pi(s,a) \frac{\nabla_\theta \pi(a|s,\theta)}{\pi(a|s,\theta)}]$. Actor-critic methods try to learn the V-function and use $Q_\pi(s,a) = \mathbb{E}[r + \gamma V_\pi(s')]$ and subtract $V_\pi(s)$ to reduce variance. Trust region methods [10] aim to stabilize policy-gradient based on-policy learning by ensuring "small" updates, for example by constraining the KL divergence between the old and the updated policy. Soft actor critic (SAC) [6, 7] is an off-policy actor critic algorithm that aims to learn a policy that simultaneously maximizes expected return and entropy and thus performs strongly while acting as randomly as possible. SAC learns a so-called soft Q-function by minimizing the soft Bellman residual $\mathbb{E}\left[\left(Q_\theta(\mathbf{s_t}, \mathbf{a_t}) - r(\mathbf{s_t}, \mathbf{a_t}) - \gamma \mathbb{E}_{a' \sim \pi_\theta(\mathbf{s_{t+1}})}[Q_{\theta'}(\mathbf{s_{t+1}}, a') - \alpha \log(\pi_\theta(a'|\mathbf{s_{t+1}}))]\right)^2\right]$ for an entropy bonus α using SGD for samples from a replay buffer. The policy is then updated using SGD on $\mathbb{E}\left[\mathbb{E}_{\mathbf{a}_t \sim \pi_\phi}\left[\alpha \log(\pi_\phi(\mathbf{a}_t \mid \mathbf{s}_t)) - Q_\theta(\mathbf{s}_t, \mathbf{a}_t)\right]\right]$. As in TD3, the target network $Q_{\theta'}$ is updated gradually. The entropy bonus α can be adapted by SGD[7].

Constrained reinforcement learning [2] uses an MDP that also has a transition-dependent cost function c_t and a cost threshold C, called a constrained MDP (CMDP). The goal is to find a policy π that maximizes R_π given the cost-constraint $\mathbb{E}_\pi[\sum_t c_t] \leq C$. It is worth noting that the best policy in a CMDP does not have to be deterministic. Previous attempts at solving CMDPs with deep RL used trust region methods like TRPO[10] and PPO[11], combined with approximate guarantees about changes in the incurred cost at every policy update [1], or lagrangian optimization [9] where an adaptive linear combination of reward and cost is optimized. To apply sample efficient off-policy algorithms like SAC and TD3 to constrained RL, we transform a given CMDP CM into an MDP M designed to make optimal behaviour on M good on CM as well. We add the cumulative cost to the state space and multiply the reward by a small factor $(1-\alpha)$ once the constraint is violated. We also add a one-time penalty β for constraint violation and an additional penalty ζ for incurring costs after the constraint is violated. More precisely, we modify the reward to

$$r_t^M = (1 - \alpha \cdot \chi_{\{\sum_{s \leq t} c_s > C\}}) \cdot r_t^{CM} - \beta \cdot \chi_{\{\sum_{s \leq t} c_s > C\} \cap \{\sum_{s < t} c_s \leq C\}} - \zeta \cdot c_t \cdot \chi_{\{\sum_{s \leq t} c_s > C\}}$$

for $\alpha \in [0,1]$, $\beta \geq 0, \zeta \geq 0$ where χ_A denotes the characteristic function of a set A. This extends Geibel's Geibel [5] idea of a fixed penalty for constraint violations which has the

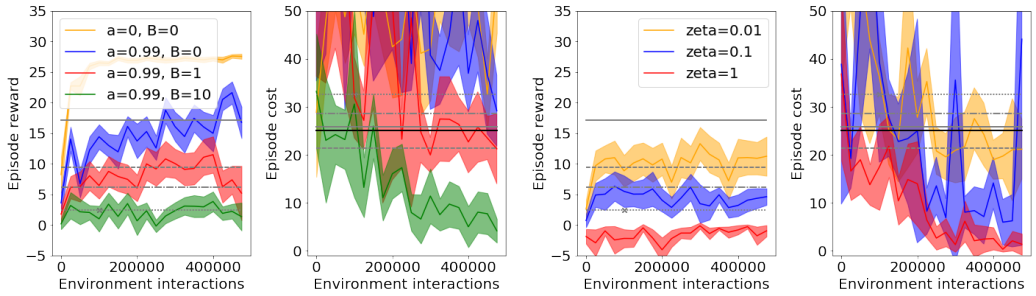
[2]RL that makes use of (deep) neural networks to represent the policy and value functions.

[3]DDPG uses a target network with parameter θ' lagging behind θ for the right term in the Bellman residual. Meanwhile, TD3 also learns a second Q-network and uses the minimum value of both networks on the right term in the Bellman residual to combat overestimation bias, updates the Q-function more often to make policy updates more accurate and adds noise to the target action $\pi_\theta(\mathbf{s_{t+1}})$ for regularization.

drawback that costs are not penalized after a constraint violation at all and that larger penalties, which are a problem for deep RL [14], can be necessary because future earned rewards are unaffected by the constraint violation. The transformed MDP M is implemented via a wrapper environment passing actions to the CMDP and storing accumulated costs. Costs are then clipped to equal at most $C + 1$ and appended to the state given to the agent, either as continuous dimension, or as a set of $C + 1$ indicator variables, with the n-th indicator active whenever the accumulated cost is at least n (we call this representation "buckets"). The bucket representation keeps observation magnitudes small, which is in line with the design philosophy behind the observation spaces in SafetyGym [9]. As the dynamics of the added dimension are known given the costs c_t, data augmentation can be used to further boost sample efficiency. We implement this by modifying the replay buffer and sampling a random integer between -10 and 10 and changing the cost for s and s' by said number while adapting the reward accordingly[4] whenever we sample a transition. To analyze our approach, we repeatedly relax and tighten our problem: First, a CMDP CM is augmented to include incurred cost in the state space. Then, the constraint on the expectation is replaced by a deterministic constraint. Lastly, we replace the constraint with our reward modification. If $\alpha = 1$ and β and ζ are positive for CM deterministic with positive rewards and a safe action with zero cost is available at every state, none of these modifications causes a reduction in the cumulative reward (when compared to deterministic policies in CM) or additional constraint violations for the optimal policy. We prove this and provide further results about the stochastic case with arbitrary bounded rewards in A.1.

3 Results

We focus our analysis on the PointGoal1 task from SafetyGym [9], a continuous control task in which a simple robot navigates a 2D-plane using two actuators, one for turning and one for moving backward and forward. Reward is gained for reaching randomly placed goals, while cost is incurred for spending time in randomly placed hazard-zones. The threshold C for the expected cost is 25 (plotted as a solid black line). We use the Lagrange-TRPO baseline[5] from [9]. We plot rewards and cost for Lagrange-TRPO after 100k (dotted), 500k (dotted and dashed), 1M (dashed) and 10M (solid) environment steps in grey.



(a) $\zeta = 0$ (b) $\alpha = 0.99, \beta = 1$

Figure 1: SAC on augmented PointGoal1 for different values of α and β. Costs are given to the agent as continuous dimension.

As expected, increasing α, β (figure 1(a)) and ζ (figure 1(b)) decreases both reward and incurred cost per episode[6] for SAC applied to the augmented task. We observe similar effects for some other tasks from SafetyGym, but the effect is not as clear for these[7]. The constraint is respected after 400k environment interactions for five out of seven parameter sets and the

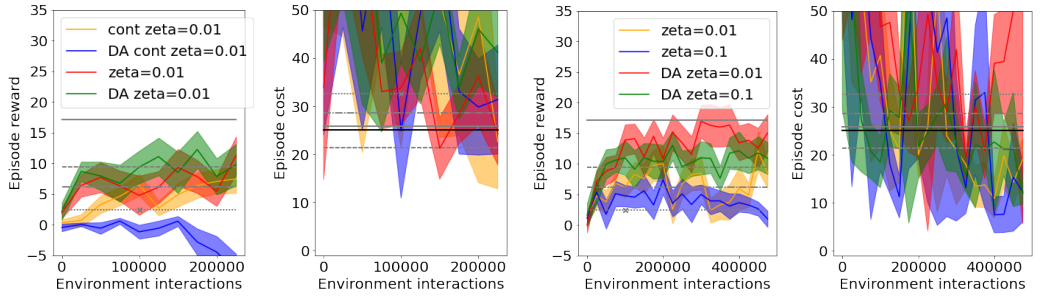
[4]Such that penalties are added when newly incurred and removed when no longer applicable.

[5]It outperforms Lagrange-PPO in terms of reward for most of the training on PointGoal1 while performing very similarly in terms of cost. The CPO-baseline does not manage to meet the constraint at all.

[6]The relationship between α, β and rewards/costs is less clear for TD3 and PPO (figures A.8, A.9).

[7]On CarGoal1 (see A.3 for task descriptions) we roughly observe the same relationship between the

best ($\alpha = 0.99, \beta = 1, \zeta = 0.01$) reaches 70% of the final performance of Lagrange TRPO (trained for 10M environment steps) in terms of reward. Next, we tested the proposed data augmentation as well as different representations for the cost (figures 2(a) and 2(b)).



(a) $\alpha = 0.99, \beta = 1$ (b) $\alpha = 0.99, \beta = 1$

Figure 2: SAC on augmented PointGoal1 with and without data augmentation for different values of ζ ($\beta = 1, \alpha = 0.99$) Costs are given to the agent as buckets unless stated differently ("cont").

Figure 2(a) shows that data augmentation only seems to work in combination with the bucket representation. Buckets also seems to slightly help early training performance in the experiments without data augmentation. In figure 2(b), we observe consistently better and more stable performance in terms of the reward when data augmentation and buckets are used, but performance in terms of costs looks worse with data augmentation. Still, the combination of data augmentation, buckets and $\zeta = 0.1$ learns to respect the constraint while also outperforming both agents without data augmentation in terms of reward. While this might be due to random fluctuations, we do see additional evidence for the efficacy of data augmentation in our experiments on other tasks from SafetyGym[8]. While the incurred cost looks lower for larger ζ with data augmentation, we do not observe a clear difference between $\zeta = 0.1$ and $\zeta = 0.01$ without data augmentation in this experiment as we did in 1(b). This indicates substantial variation in training performance for the same parameters. Lastly, we trained a few promising configurations for more steps (Figure 3).

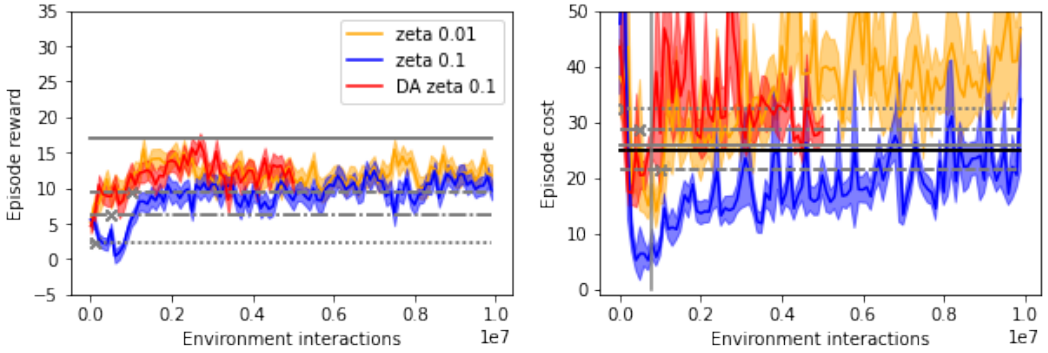


Figure 3: SAC on augmented PointGoal1 using the bucket-representation for cost; Different values of ζ and with and without data augmentation ($\beta = 1, \alpha = 0.99$). Costs are given to the agent as continuous dimension unless mentioned otherwise ("buckets").

parameters and cost/reward, albeit with more noise (figures A.23,A.13). On the more challenging PointGoal2 task, our agents are unable to achieve positive reward, but larger values of ζ and β seem to reduce costs (figures A.25,A.15). We don't see a strong effect of the parameters for PointPush1, where all agents seem to quickly learn to respect the cost constraint while failing to achieve any reward (figures A.24,A.14).

[8]The data augmented versions with buckets seem to consistently outperform continuous cost representations without data augmentation for CarGoal1, PointPush1 and PointGoal2 (figuresA.29, A.30 A.31) in terms of cost and reward (even the episode reward is still essentially zero for the latter two). The same is true for TD3 on all four environments (figures A.21,A.32,A.33,A.34), except for some slightly better performance by the version without data augmentation in terms of cost, at the cost of a lot of reward for Cargoal1.

While our approach requires fewer environment interactions (300k for $\zeta = 0.1$ with and without data augmentation, 400k for $\zeta = 0.1$ without data augmentation) to keep costs below the threshold than Lagrange-TRPO (800k), it either receives less reward during early training ($\zeta = 0.1$ without data augmentation and buckets) or eventually starts to consistently violate the constraint ($\zeta = 0.1$ with data augmentation and buckets, $\zeta = 0.01$ without). In all cases, the cost incurred by our approach clearly increases after around 1M steps and keeps increasing when data augmentation and buckets are not used, while it drops again (to a level above the constraint) with data augmentation. Furthermore, our approach's performance in terms of total episode reward (excluding periods in which we violate the constraints) caps at around 12, which is roughly 70% of the final performance achieved by Lagrange-TRPO (17). However, it is also worth noting that Lagrange-TRPO repeatedly violates the constraint for around 1M steps twice after first having learnt to respect the constraint, while our approach with $\zeta = 0.1$ only violates the constraint for much shorter periods of time after first having learnt to respect it.

4 Discussion

Our approach learns to respect the cost constraints using considerably less environment interactions than the Lagrange-TRPO baseline. This is important, especially in cases where avoiding costs during training is as important as avoiding them during deployment. For some parameters we are also able to improve on the reward more quickly than Lagrange-TRPO during early training. Furthermore, the stronger performance of SAC on our task compared to PPO shows that our approach can benefit from improvements in generic RL algorithms.

Interestingly, the benefits from data augmentation are rather small and inconsistent. Also our theoretical guarantees are rather weak, especially in highly stochastic environments, or if there are unavoidable delayed costs. Furthermore they only apply to the optimal policy, which is not necessarily found by deep RL algorithms. This is especially problematic in combination with the fact that our approach's performance is highly dependent on hyperparameters: In figure 3, we see that the cost incurred by the final policy is very different from the cost during early training, which makes it difficult to tune hyperparameters. Also, hyperparameter tuning requires additional environment interactions during which cost is incurred, which is a problem if constraint violations during training have adverse consequences. Luckily, α, β and ζ can easily be changed during training using a slightly modified version of our data augmentation procedure, such that future work identifying (heuristic) learning rules for these parameters, balancing costs and reward adaptively during training, seems promising.

5 Summary

We presented a novel approach to constrained reinforcement learning based on augmenting the state space with the previously incurred cost and penalizing constraint violations both directly (reward penalty for constraint violation and for costs incurred afterwards) and indirectly (reduced base reward after constraint violation). This approach can be combined with any reinforcement learning algorithm and is easily amenable to data augmentation along the state space dimension representing the cost when off-policy algorithms are used. Combined with SAC and the right parameters, we manage to respect the cost constraint in half as many environment samples as the Lagrange-TRPO baseline on PointGoal1 and achieve 70% of Lagrange-TRPO's final performance in terms of reward. However, while our approach performs strongly during early training, performance often deteriorates after some a while. Also, we observe a strong dependence on the parametrization of our penalties, which reduces the approach's efficacy as additional samples are needed for tuning the penalties.

References

[1] Joshua Achiam, David Held, Aviv Tamar, and Pieter Abbeel. Constrained policy optimization. *arXiv preprint arXiv:1705.10528*, 2017.

[2] Eitan Altman. *Constrained Markov decision processes*, volume 7. CRC Press, 1999.

[3] Anonymous. Measuring progress in deep reinforcement learning sample efficiency. In *Submitted to International Conference on Learning Representations*, 2021. URL `\url{https://openreview.net/forum?id=_QdvdkxOii6}`. under review.

[4] Scott Fujimoto, Herke Van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. *arXiv preprint arXiv:1802.09477*, 2018.

[5] Peter Geibel. Reinforcement learning for mdps with constraints. In *European Conference on Machine Learning*, pages 646–653. Springer, 2006.

[6] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*, 2018.

[7] 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.

[8] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[9] Alex Ray, Joshua Achiam, and Dario Amodei. Benchmarking safe exploration in deep reinforcement learning. *arXiv preprint arXiv:1910.01708*, 2019.

[10] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897, 2015.

[11] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[12] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. 2014.

[13] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[14] Hado P van Hasselt, Arthur Guez, Matteo Hessel, Volodymyr Mnih, and David Silver. Learning values across many orders of magnitude. In *Advances in Neural Information Processing Systems*, pages 4287–4295, 2016.

A Appendix

A.1 Theoretical results

We discuss several observations about the chain of modifications from CMDP to CMDP with lifted state space including costs, from that to deterministic constraints and from that to our augmented MDP with reward modification. The first two propositions are concerned with the first two approximations in order, while the third and fourth discuss different aspects of the last approximation.

For a given CMDP, CM a cost-conditioned policy maps state-cost pairs (s, c) where s_t is the original state at time t and c is the previously accumulated cost $\sum_{s<t} c_s$ to distributions over actions. As the distribution of c_t is a function of the state-action-state tuple $s_t, a_t, \sum_{s<t} c_s$ only depends on s_{t-1}, a_{t-1} and $\sum_{s<t-1} c_s$. Thus, a cost-conditioned policy can be seen as a policy on an augmented CMDP with states $(s_t, \sum_{s<t} c_s)$ and we can define the $V-$functions for cost-conditioned policies via the augmented CMDP. Because a cost-constrained policy can be constant in the cost c for all s, the set of cost-constrained policies includes all policies on CM and we have:

Proposition 1 *Let CM be a CMDP with the set of constraint-respecting policies Π nonempty. Then, the set of cost-conditioned constraint-respecting policies Π_c is nonempty and*

$$\sup_{\pi \in \Pi_c} R_\pi \geq \sup_{\pi \in \Pi} R_\pi.$$

Next, we have:

Proposition 2 *Let CM be a CMDP for which the set of cost-conditioned constraint-respecting policies Π_c is nonempty. Define Π_{cd} as the set of cost-conditioned policies π for which the probability that $\sum_t c_t > C$ at any t is zero. Then:*

$$\sup_{\pi \in \Pi_c} R_\pi \geq \sup_{\pi \in \Pi_{cd}} R_\pi = \sup_{\pi \in \Pi_{cd}} \mathbb{E}_{s_0}[V_\pi(s_0, 0)] \geq \sup_{\pi \in \Pi_c} \mathbb{E}_\pi[\sum_{k<t} \gamma^k r_k + \gamma^t \sup_{\pi \in \Pi_{cd}} V_\pi(s_t, \sum_{k<t} c_k)]$$

where $(t, s_t, \sum_{s<t} c_s)$ in the rightmost expectation is the first time, state, cumulative cost tuple for which the probability of a constraint violation would be positive after following π for another step, no matter what policy is followed afterward.

The first inequality stems from the fact that all policies with a zero probability to violate the constraint also won't do so in expectation. The second inequality can be seen by looking at the set of policies given by following $\pi \in \Pi_c$ until continuing to do so would entail a nonzero probability of violating the constraint and following a policy in Π_{cd} afterwards. If Π_{cd} is empty, the supremum equals $-\infty$ and the equation is trivially true. This tells us that moving from the constraint on the expectation to the hard constraint can be problematic in terms of our (generally non-sharp!) bound on returns if there are actions that perform well in terms of future reward but have a small chance of incurring costs[9] at a scale comparable to C, such that a) strong policies might use these actions and b) this incurs a positive probability to violate the constraint even before a lot of cost has been accumulated. Then, we have:

Proposition 3 *Let CM be a CMDP with a nonempty set of cost-conditioned policies π for which the probability that $\sum_t c_t > C$ at any t is zero, $\Pi_{cd} \neq \emptyset$. Also, assume $r_t \geq 0$.Let $M(\alpha, \beta, \zeta)$ be the corresponding reward-augmented MDP with optimal policy π^*_M. Then we have*

$$R^{CM}_{\pi^*_M} \geq \sup_{\pi \in \Pi_{cd}} R^{CM}_\pi$$

for the expected cumulative return R^{CM} in CM.

This is because a reward in M is always at most as large as the corresponding reward in CM and because the set of policies on M includes Π_{cd}, for policies in which the rewards are equal on M and CM. Note that we don't require rewards to be positive as long as β is sufficiently large: Returns are linear in rewards and the optimal policy is invariant under constant shifts of the reward such that we can always achieve $r \geq 0$ by shifting the rewards in CM: $\bar{r}^{CM} = r^{CM} + y$. In particular, the MDP $\bar{M}(\alpha, \beta, \zeta)$ with augmented reward $r^{\bar{M}}$

[9]This can either be immediate costs or a transition to a state in which future costs become unavoidable

for the version of CM with shifted reward $\bar{r}^{CM}$ induces the same returns as an augmented version of CM with the original rewards, $M(\alpha, \beta + \frac{\alpha y}{1-\gamma}, \zeta)$ up to a shift of y:

$$\begin{aligned} r_t^{\bar{M}} &= (1 - \alpha \cdot \chi_{\{\sum_{s \leq t} c_s > C\}}) \cdot (r_t^{CM} + y) - \beta \cdot \chi_{\{\sum_{s \leq t} c_s > C\} \cap \{\sum_{s < t} c_s \leq C\}} - \zeta \cdot c_t \cdot \chi_{\{\sum_{s \leq t} c_s > C\}} \\ &= [(1 - \alpha \cdot \chi_{\{\sum_{s \leq t} c_s > C\}}) \cdot (r_t^{CM}) - \alpha \cdot \chi_{\{\sum_{s \leq t} c_s > C\}} y - \beta \cdot \chi_{\{\sum_{s \leq t} c_s > C\} \cap \{\sum_{s < t} c_s \leq C\}} \\ &\quad - \zeta \cdot c_t \cdot \chi_{\{\sum_{s \leq t} c_s > C\}}] + y \end{aligned}$$

which up to a constant shift leads to the same returns as

$$(1 - \alpha \cdot \chi_{\{\sum_{s \leq t} c_s > C\}}) \cdot (r_t^{CM}) - (\beta + \frac{\alpha y}{1-\gamma}) \cdot \chi_{\{\sum_{s \leq t} c_s > C\} \cap \{\sum_{s < t} c_s \leq C\}} - \zeta \cdot c_t \cdot \chi_{\{\sum_{s \leq t} c_s > C\}},$$

because the β penalty gets applied only once at constraint violation, the α penalty is applied consistently after a violation, and as a single reduction in reward r at time t leads to the same discounted return as a persistent reduction in reward by $(1-\gamma)r$ starting at time t. Lastly, we identify a condition in which the constraint might be violated by strong policies in the augmented MDP:

Proposition 4 *Let CM be a CMDP and Π_{cd} the set of cost-conditioned policies π for which the probability that $\sum_t c_t > C$ at any t is zero. Let $M(\alpha, \beta, \zeta)$ be the corresponding reward-augmented MDP. Then π^*_M, the optimal policy in $M(\alpha, \beta, \zeta)$ will violate the constraint with positive probability, if and only if there exists a state-cost tuple (s, c) such that*

$$V_{\pi^*_M}(s, c) - \mathbb{E}_{\pi^*_M, (s,c)}[\beta \cdot \gamma^t + \sum_{k \geq t} \gamma^k (\alpha \cdot r_k + \zeta \cdot c_k)] \geq \sup_{\pi \in \Pi_{cd}} V_\pi(s, c)$$

where t in the expectation is the time of the first constraint violation.

This follows from comparing the value functions in $M(\alpha, \beta, \zeta)$: By linearity of the expectation, the left hand term is equal to the value of π^*_M in M. On the other hand, the value of any policy $\pi \in \Pi_{cd}$ is the same in both CM and M because the constraint is never violated. So unsurprisingly, increasing the penalty parameters makes constraint violations less likely, at least if rewards in CM are positive. As in proposition 2, small likelihoods of large costs are particularly problematic as they don't affect the expectation a lot but might still induce a positive probability of constrain violation. It is however worth noting that these approximation errors point in different directions: in proposition 2, reward was lost in order to avoid these unlikely costs, while the last proposition suggest that the risk of incurring the costs might be taken up in the augmented MDP, if a lot of reward is to gain. Delayed but unavoidable costs present another problem for our approximation.

Now consider CM deterministic with positive rewards, $\alpha = 1$ and $\beta, \zeta > 0$ for which a constraint-respecting policy exists starting in any state-cost pair (s, c) with $c \leq C$. Due to proposition 1, we neither lose reward from conditioning the policy on previously incurred costs, nor risk additional constraint violations. As long as only deterministic policies are considered, the expectation on the right hand side in proposition 2 becomes equal to $\sup_{\pi \in \Pi_c} R_\pi$, as policies in Π_c never get to a state where avoiding the constraint becomes impossible. According to proposition 3 we also don't loose reward moving from the hard constraint to M. Meanwhile, rewards are strictly negative in M whenever the constraint is violated and positive otherwise, such that the deterministic optimal policy will never choose a constraint-violating action over a constraint-respecting one. As our assumption of a constraint-respecting policy existing starting in every state rules out unavoidable delayed costs, our modification preserves the constraint-respecting property for the optimal policy.

A.2 More experimental results

First, we tested td3 on the unconstrained PointGoal1 task using different amounts of random exploration steps before the learnt policy is employed and different levels of exploration noise.

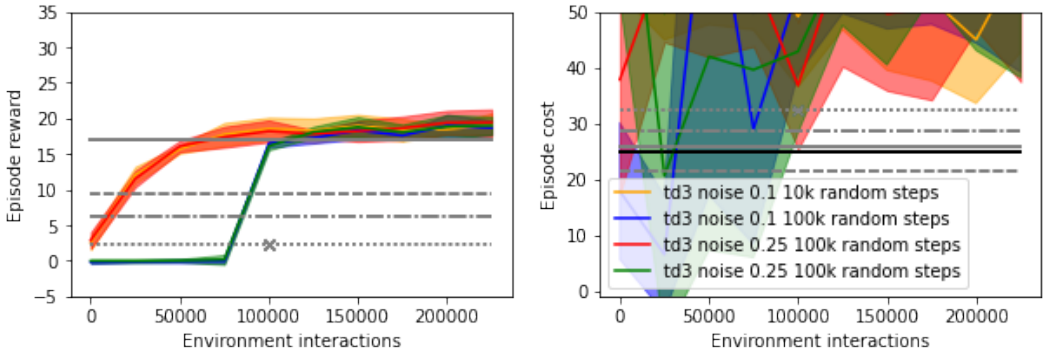


Figure A.1: TD3 on PointGoal1

Surprisingly, none of these parameters seemed to matter much for final performance. Also, final performance was lacking compared to the total reward of 25 reported for PPO and TRPO in [9] (after 10M training). As shown in the next figure, we tested the sensitivity of SAC on the unconstrained PointGoal1 task to the reward scale/entropy bonus parameter and the amount of random exploration steps before the learnt policy is employed.

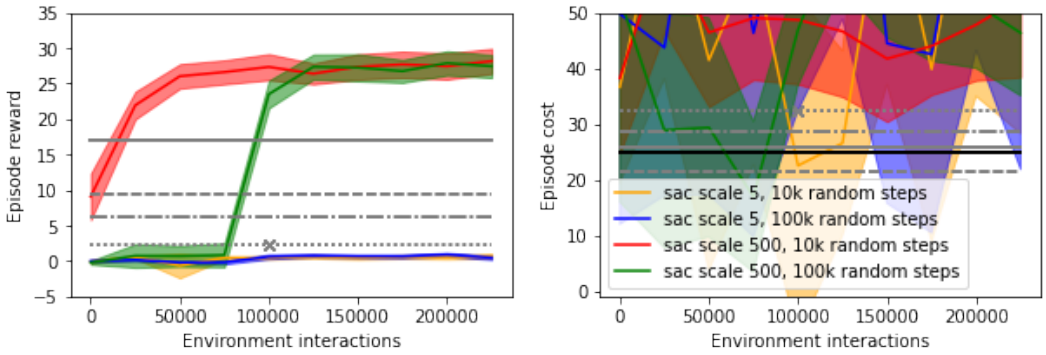


Figure A.2: SAC with fixed reward scale/entropy bonus on Pointgoal 1

SAC clearly outperforms TD3 (and seems to achieve slightly more reward than PPO and TRPO after 10M steps) with the right reward scale, but does not learn at all when this hyperparameter is chosen badly. Thus, we decided to use the adaptive learning rule for the reward scale described in [7]:

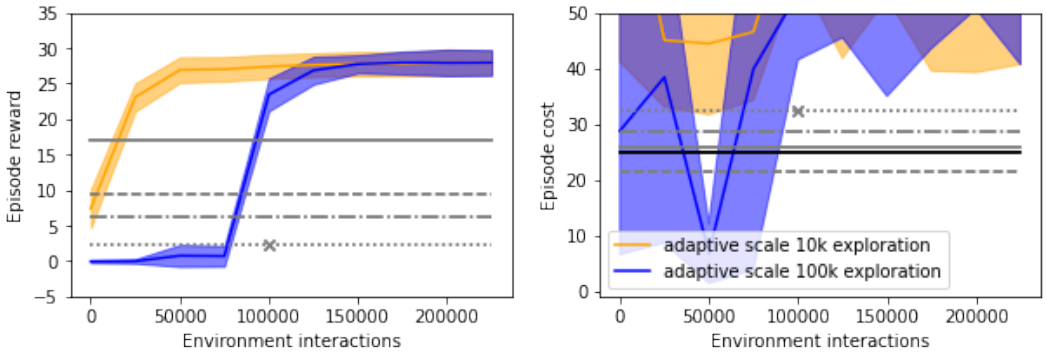


Figure A.3: SAC with learnt reward scale on Pointgoal 1

We observe almost no difference between SAC with the adaptive reward scale and the strong value for the reward scale from the last plot. Again, performance did not seem to depend on the amount of exploration steps. Next, we tested SAC on the CarGoal1,PointPush1 and PointGoal2 tasks.

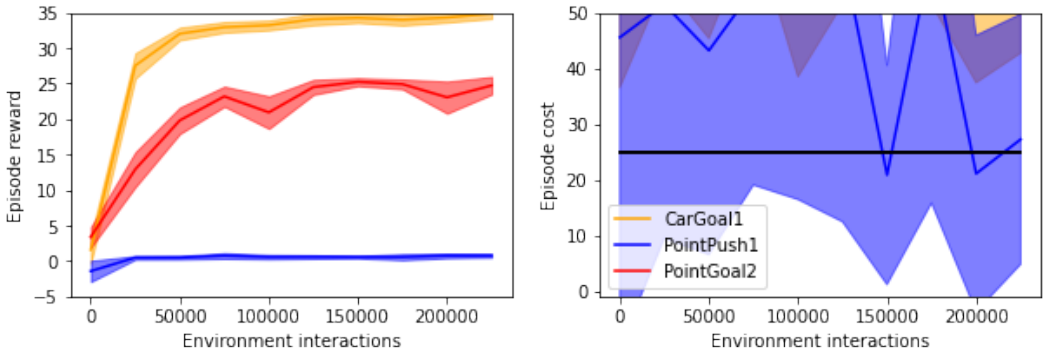


Figure A.4: SAC with learnt reward scale on other tasks

While SAC quickly outperforms the values reported for PPO and TRPO in [9] for both CarGoal (PPO: 25 TRPO: 31) and performs similarly on PointGoal2 (PPO and TRPO 22.4), it fails to learn on PointPush1 (PP0: 3 TRPO:8). Then, we tested different architectures and representations of the cost:

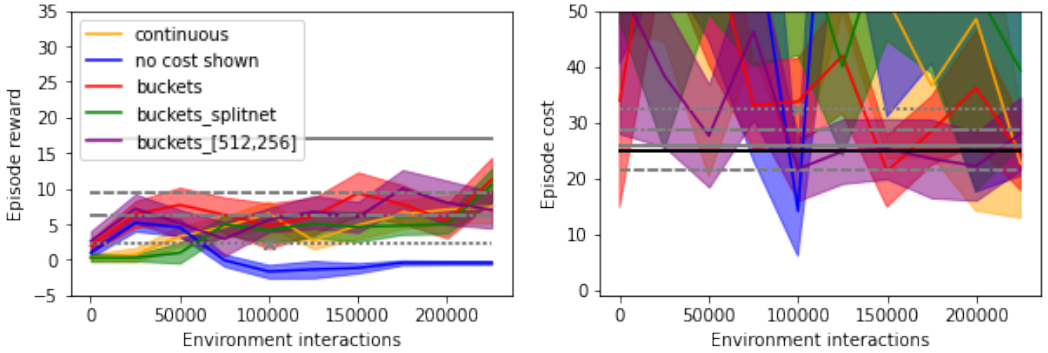


Figure A.5: SAC on augmented PointGoal1 for different representations of the cost and network architectures ($\zeta = 0.01, \beta = 1, \alpha = 0.99$).

We can see that the agent clearly performs worst if the policy is not conditioned on the previously incurred cost ("no cost shown"). Apart from that, we see weak evidence that encoding the cost in terms of indicator buckets leads to faster learning in terms of the reward, while increasing the size of the first network layer from 256 to 512 might slightly help with learning to respect the constraint. Meanwhile, using a separate network to learn a cost-avoiding policy once the cost constraint is violated[10] ("buckets_splitnet") does seem to hurt rather than help, which indicates that the difference in optimal behaviour in the augmented MDP between before and after the constraint is violated does not seem to be a major problem. We repeated this experiment for TD3:

[10]implemented as a switch that changes the used network if the last indicator bucket for the cost is equal to one

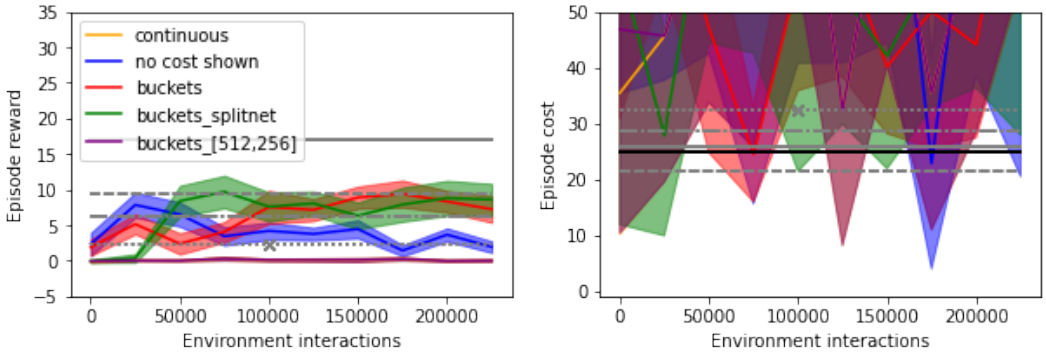


Figure A.6: TD3 on augmented PointGoal1 for different representations of the cost and network architectures ($\zeta = 0.01, \beta = 1, \alpha = 0.99$). Cost represented as continuous dimension in the state space.

Interestingly, the continuous representation and the larger network using the bucket representation performed worst this time. However it is worth noting that TD3 seems to sometimes randomly fail to learn because of bad initial exploration. Interestnigly, none of the runs came close to respecting the cost constraint. Next, the experiment was repeated for PPO:

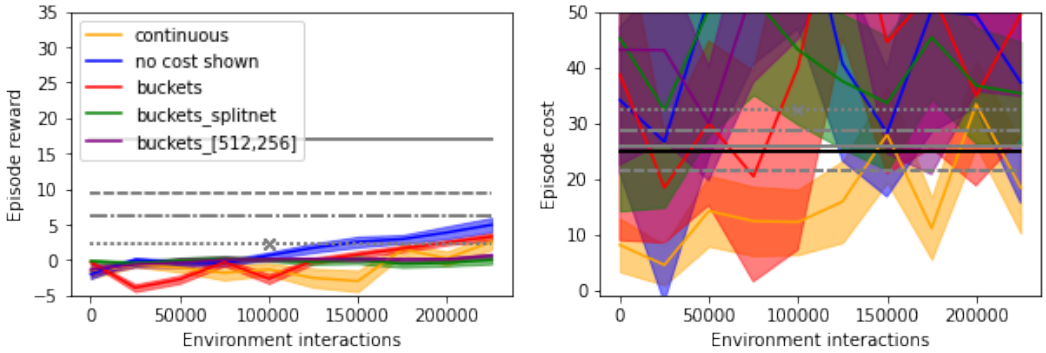


Figure A.7: PPO on augmented PointGoal1 for different representations of the cost and network architectures ($\zeta = 0.01, \beta = 1, \alpha = 0.99$). Cost represented as continuous dimension in the state space.

Here, the continuous representation is the only one respecting the cost constraint and rewards are generally pretty low. Interestingly, the agent that does not condition on the incurred cost seems to perform slightly better than the others in terms of reward, even if it still receives the penalties for violating the constraint. Next, we replicated figure 1(a) for td3 and PPO:

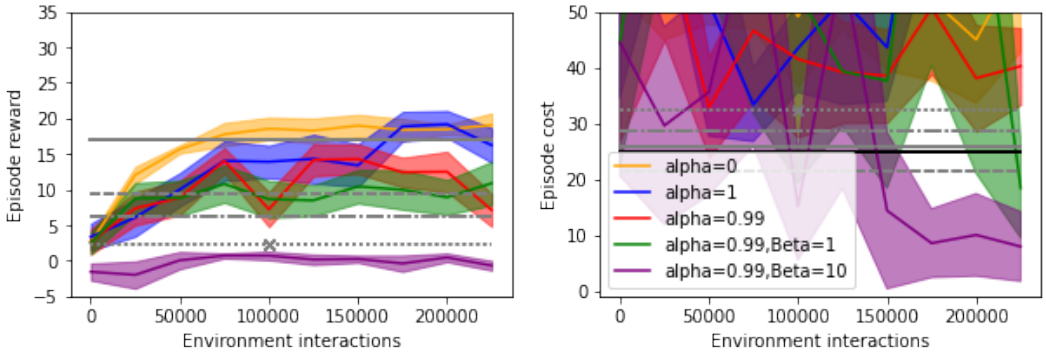


Figure A.8: TD3 on augmented PointGoal1 for different values of α and β. Cost represented as continuous dimension in the state space.

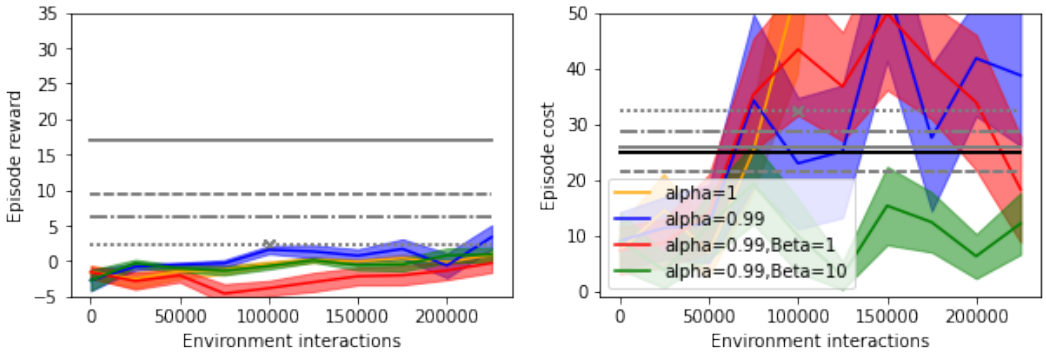


Figure A.9: PPO on augmented PointGoal1 for different values of α and β

We see $\beta = 10$ performing best in terms of cost in both cases while preventing the reward from improving. For Td3, larger alpha and beta roughly correspond to both less reward and (even more roughly) less cost. Reward is improving very slowly across the board for PPO and we don't see a clear relationship between cost and the parameters (except for $\beta = 10$).We also replicated figure 1(b) for td3 and PPO:

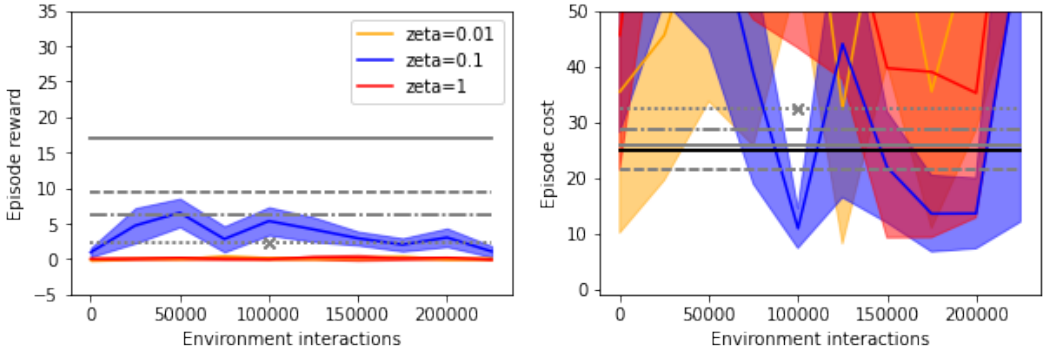


Figure A.10: TD3 on augmented PointGoal1 for different values of ζ ($\beta = 1, \alpha = 0.99$).

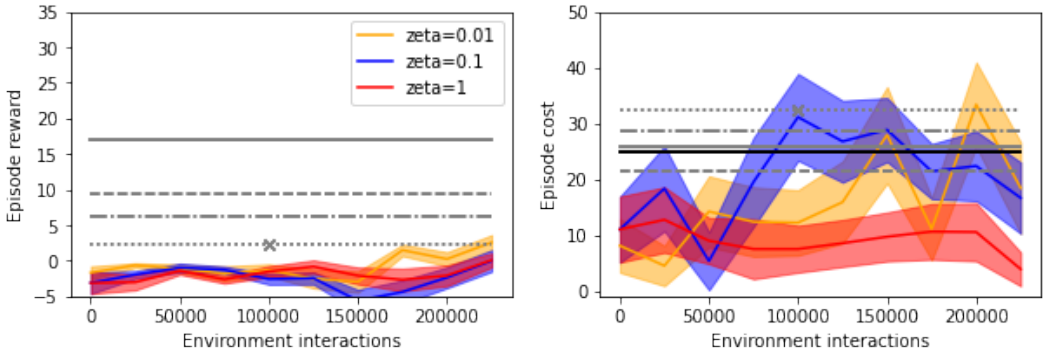


Figure A.11: PPO on augmented PointGoal1 for different values of ζ ($\beta = 1, \alpha = 0.99$). Cost represented as continuous dimension in the state space.

TD3 seems to only learn for $\zeta = 0.1$, probably due to insufficient exploration. Meanwhile PPO improves on reward only very slowly but stably has the lowest cost for the largest ζ.

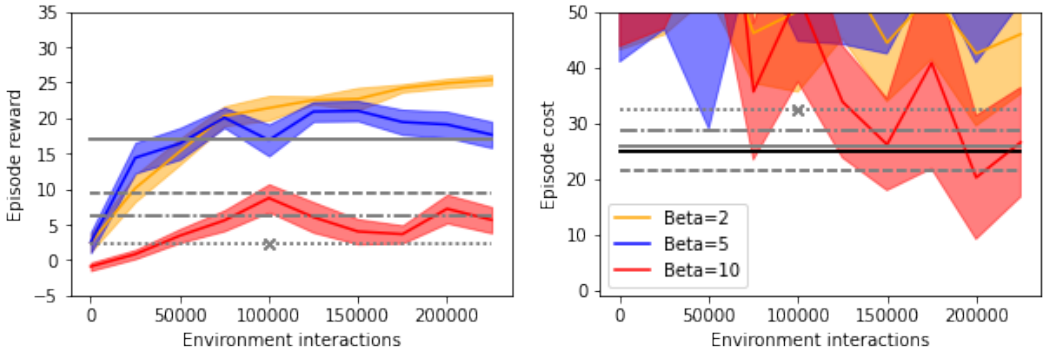


Figure A.12: SAC on augmented PointGoal1 without multiplicative reward reduction at constraint violation ($\zeta = 0.0, \alpha = 0$). Cost represented as continuous dimension in the state space.

Interestingly, $\beta = 10$ with the other parameters at zero, which is akin to Geibel's [5] idea of a fixed one-time penalty seems to perform comparably to our more general approach for the PointGoal1 environment.

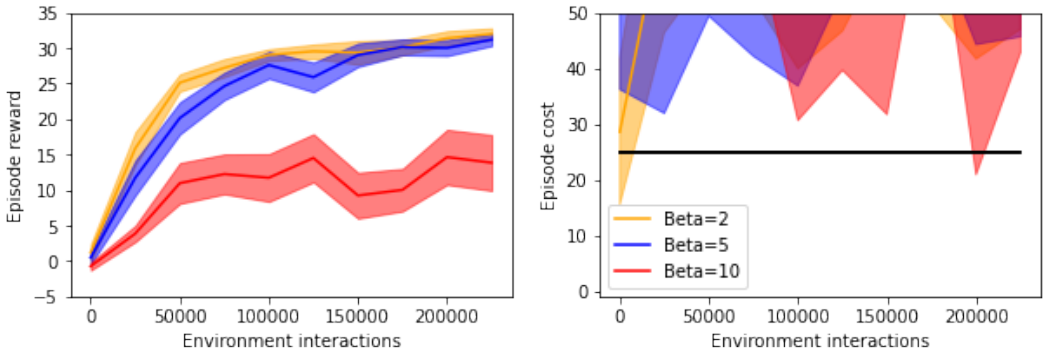


Figure A.13: SAC on augmented CarGoal1 with $\alpha = 0, \zeta = 0$ and different values of β. Cost representation continuous.

However, unlike for the parameter combination of $\alpha = 0.99, \beta = 1, \zeta = 0.1$, these strong parameters do not transfer to Cargoal1, where the constraint is far from being met with

these parameters.

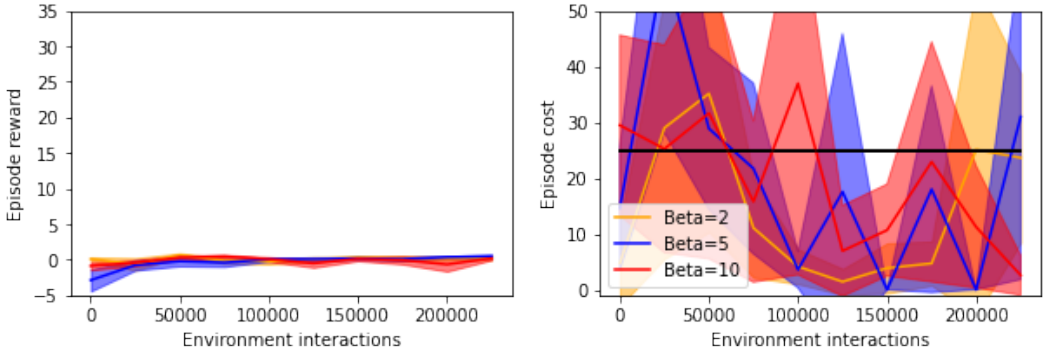


Figure A.14: SAC on augmented PointPush1 with $\alpha = 0, \zeta = 0$ and different values of β. Cost representation continuous.

On PointPush, where our agents are unable to achieve positive reward, all of the tested values for β are sufficient to achieve few constraint violations.

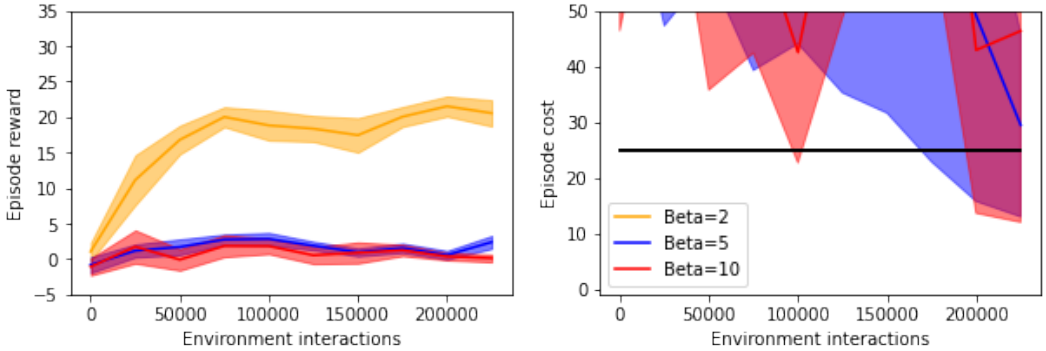


Figure A.15: SAC on augmented PointGoal2 with $\alpha = 0, \zeta = 0$ and different values of β. Cost representation continuous.

For PointGoal2, large penalty values do seem to reduce the cost but don't lead to constraint respecting policies within 250k environment interactions.

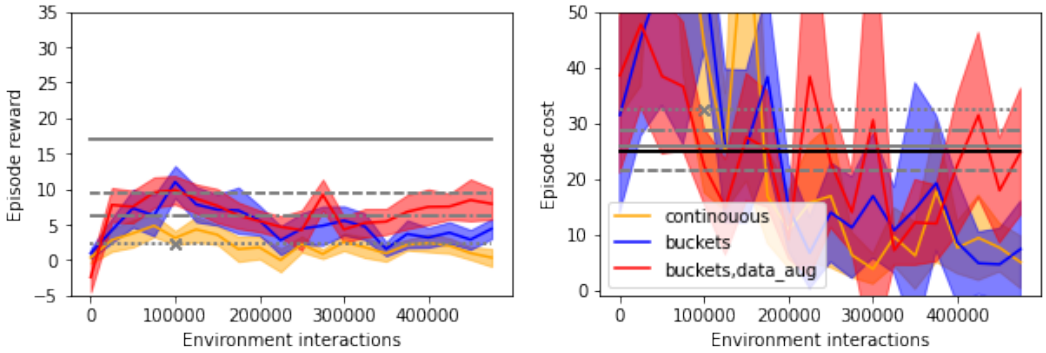


Figure A.16: SAC on augmented PointGoal1 for different representations of the cost and with and without data augmentation, without multiplicative reward reduction at constraint violation ($\zeta = 0.0, \beta = 10, \alpha = 0$).

The parametrization with large β and the other values at zero also seems to benefit from

using data augmentation and the bucket representation for the cost, but we can again observe slightly increased costs when using data augmentation. Next, we did some more experiments for different combinations of β and ζ:

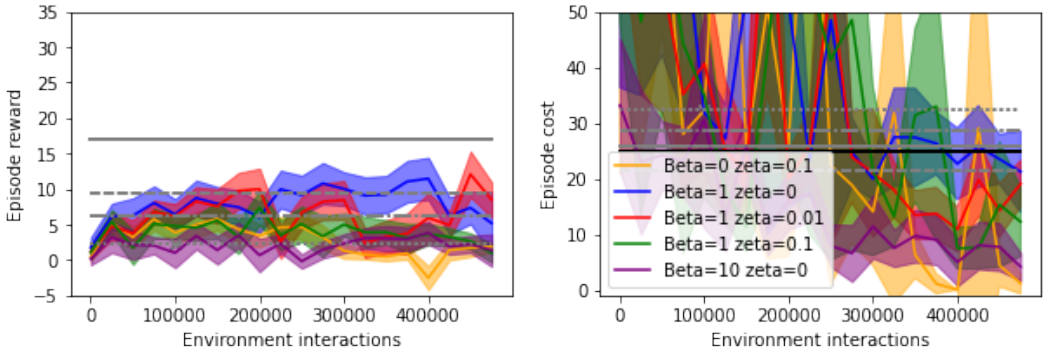


Figure A.17: SAC on augmented PointGoal1 for different values of β and ζ ($\alpha = 0.99$). Cost represented as continuous dimension in the state space.

While we can see a clear difference between $\beta = 1$ and $\beta = 10$, the effect of ζ is more noisy. Surprisingly, $\zeta = 0.1, \beta = 0$ receives less cost and reward than $\zeta = 0.1, \beta = 1$. Next, we tested whether increasing the batch size would help SAC, especially in combination with data augmentation:

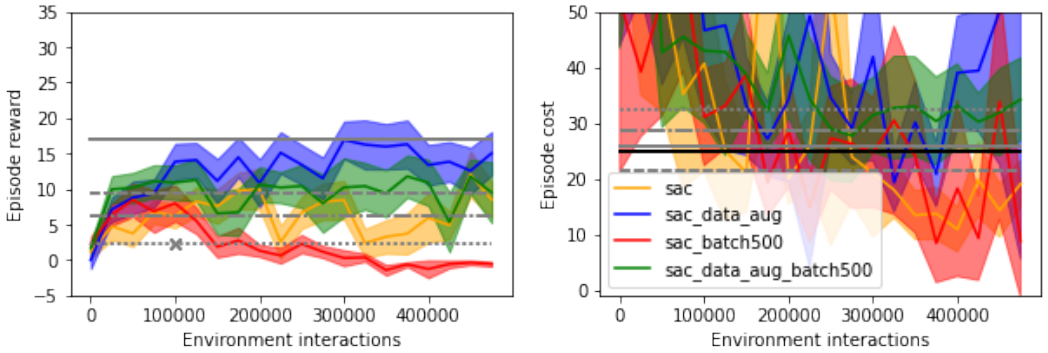


Figure A.18: SAC on augmented PointGoal1 with and without data augmentation and with different batch sizes (100 as in the other experiments and 500).$\alpha = 0.99, \beta = 1, \zeta = 0.01$. Cost represented in buckets when data augmentation is used, continuous otherwise.

While costs look a little more stable with a larger batch size of 500, they are not consistently lower than with the standard batch size of 100 and rewards are worse. This gap in rewards is even more pronounced without data augmentation, and in this case we also don't see evidence for increased stability with larger batches.

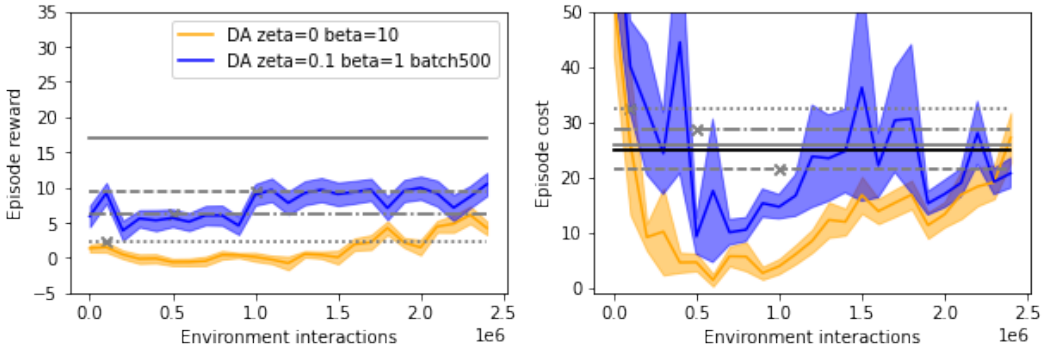


Figure A.19: SAC on augmented PointGoal1 with data augmentation and with different batch sizes (100 unless mentioned otherwise).$\alpha = 0.99$. Cost represented in buckets when data augmentation is used, continuous otherwise.

We still observe the increasing costs after longer training runs for larger batch sizes and with larger values of β rather than ζ. Next, we did a similar experiment for TD3:

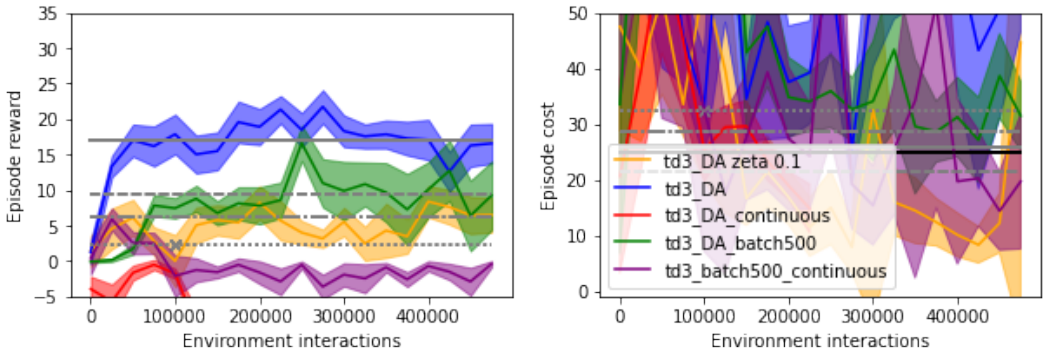


Figure A.20: Td3 on augmented PointGoal1 with and without data augmentation. $\alpha = 0.99, \beta = 1, \zeta = 0.01$ (unless stated otherwise). Cost represented in buckets unless stated otherwise ("continuous").

Again, we see potentially more stable costs when using larger batches. In the case of continuous cost representation, there seems to be a larger gap, but performance in terms of reward is abysmal either way. Also, we again see $\zeta = 0.1$ learning to respect the constraint while also achieving reasonable performance according to the reward. However, training with $\zeta = 0.1$ for longer did yield worse results (also during initial training).

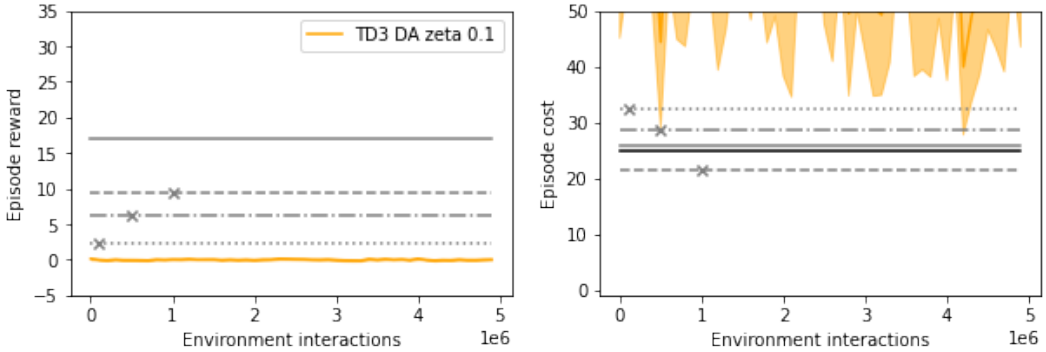


Figure A.21: Td3 on augmented PointGoal1 with data augmentation trained for a longer time. $\alpha = 0.99, \beta = 1, \zeta = 0.1$. Cost represented in buckets.

This is not the only time we've seen TD3 to fail learning for parameters for which it sometimes does learn and likely points to problems with initial exploration. Next, we had a look on further values for α and β for SAC with data augmentation:

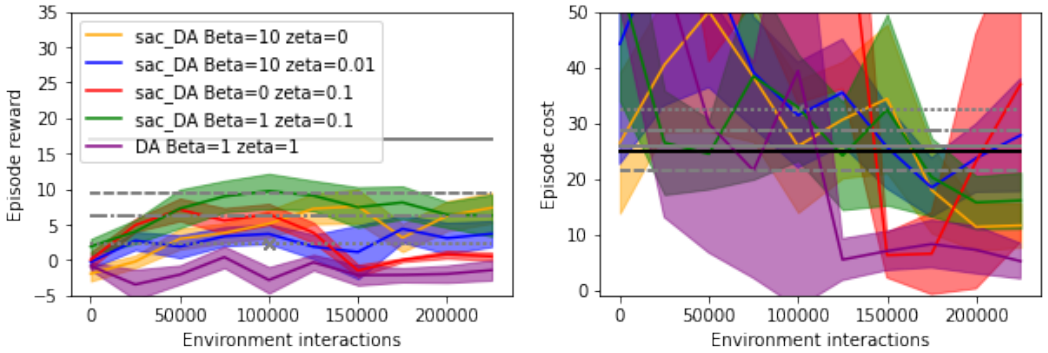


Figure A.22: SAC on augmented PointGoal1 with and without data augmentation and different environment parameters and $\alpha = 0.99, \beta = 1, \zeta = 0.01$ unless stated otherwise. Cost represented in buckets unless stated otherwise ("continuous").

While larger values of ζ consistently correspond to worse rewards, the effect is less clear for the costs (except for $\zeta = 1$). Interestingly, $\beta = 0, \zeta = 0.01$ performs better in terms than $\beta = 1, \zeta = 0.01$. Next, we have a closer look at the effect of β and ζ in the other three tasks:

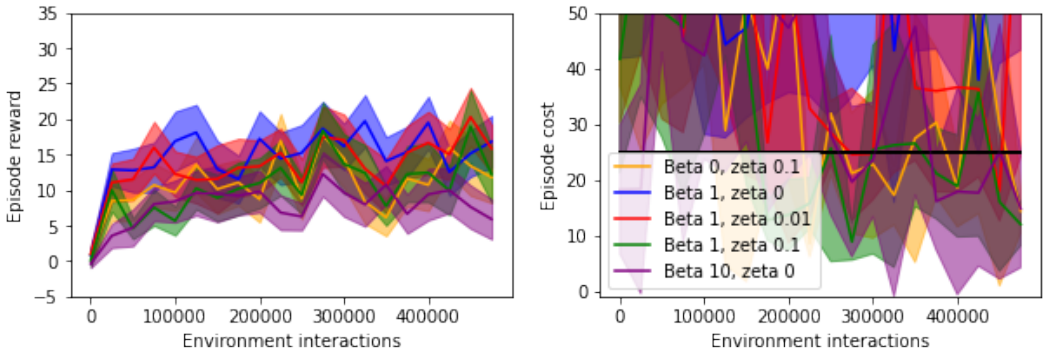


Figure A.23: SAC on augmented CarGoal1 with $\alpha = 0.99$ and different environment parameters. Cost representation continuous.

For CarGoal1, results look roughly similar to the results for PointGoal1, but are a lot more noisy, especially for the costs.

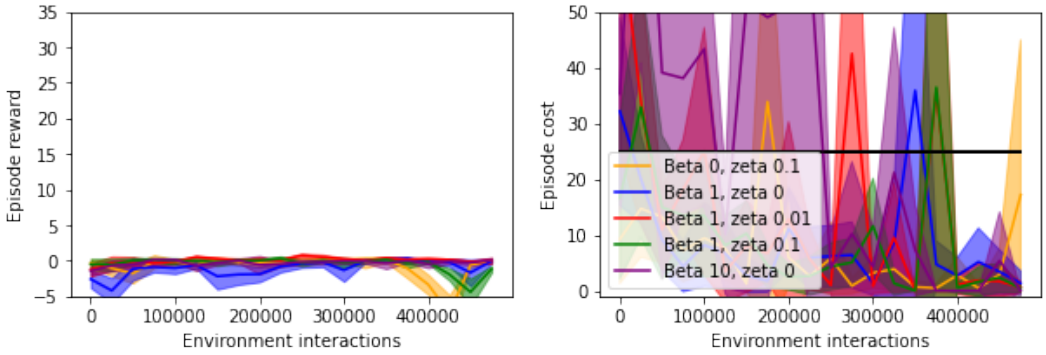


Figure A.24: SAC on augmented PointPush1 with $\alpha = 0.99$ and different environment parameters. Cost representation continuous.

For PointPush1, there is little to see, as no parametrization reaches positive reward and all seems to respect the constraint after some training.

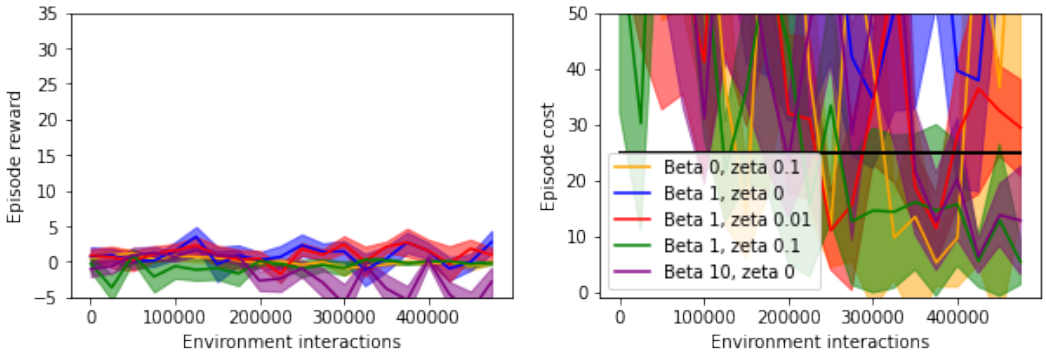


Figure A.25: SAC on augmented PointGoal2 with $\alpha = 0.99$ and different environment parameters. Cost representation continuous.

There is also little reward received across the board for PointGoal2, but this time only $\beta = 1, \zeta = 0.1$ and $\beta = 10, \zeta = 0$ learn to respect the constraint. We also compared different state representations and algorithms for the other three tasks:

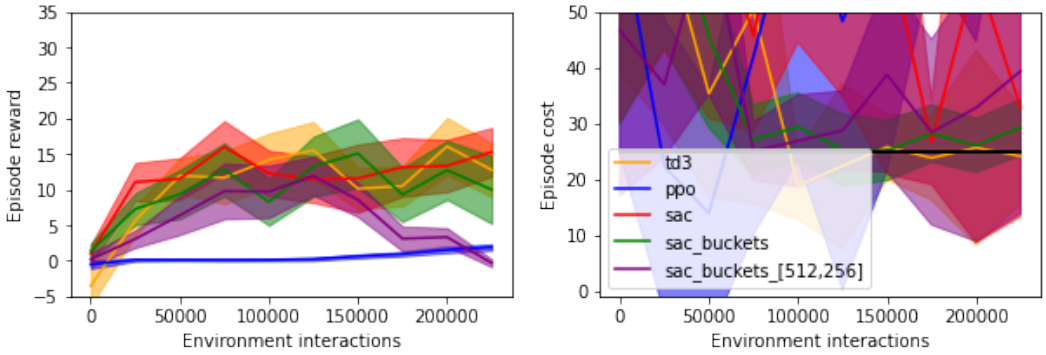


Figure A.26: Different algorithms, cost representations and network architectures on augmented CarGoal1 with $\alpha = 0.99, \beta = 1, \zeta = 0.01$. Cost representation continuous unless stated otherwise ("buckets").

For CarGoal1, SAC and TD3 clearly outperform PPO. SAC performs worse than TD3 both in terms of cost when costs are represented as a continuous dimension. SAC with buckets performs equally well in terms of rewards but similar to TD3 in terms of cost, while SAC with buckets and a larger first layer performs badly in terms of reward.

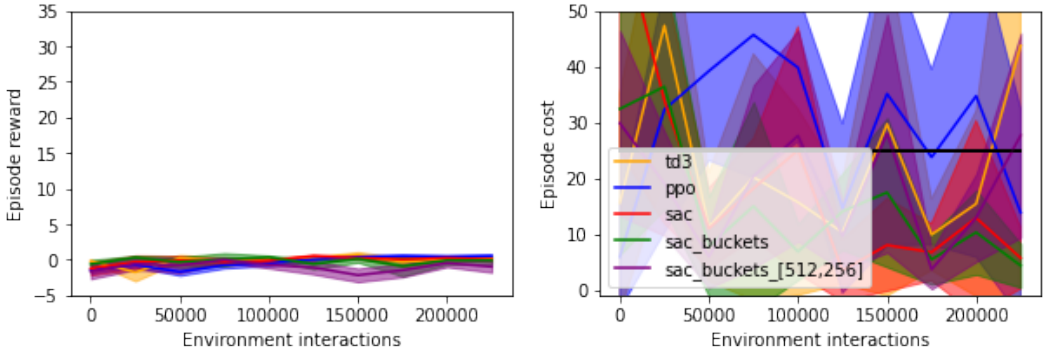


Figure A.27: Different algorithms, cost representations and network architectures on augmented PointPush1 with $\alpha = 0.99, \beta = 1, \zeta = 0.01$. Cost representation continuous unless stated otherwise ("buckets").

None of the algorithms achieves significant positive reward for PointPush1. PPO and TD3 seem to perform worst in terms of cost.

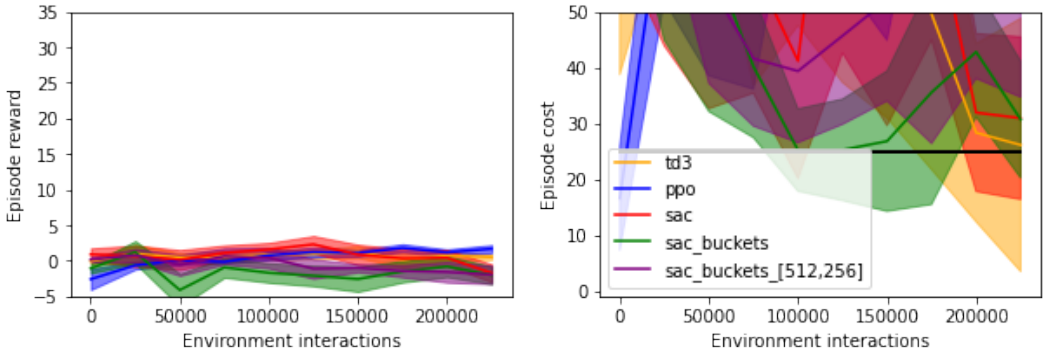


Figure A.28: Different algorithms, cost representations and network architectures on augmented PointGoal2 with $\alpha = 0.99, \beta = 1, \zeta = 0.01$. Cost representation continuous unless stated otherwise ("buckets").

The situation is similar for PointGoal2 in terms of rewards. This time, PPO and SAC using

the bucket representation and a larger first network layer clearly perform worst. Next, ee performed further experiments on data augmentation for the three other tasks:

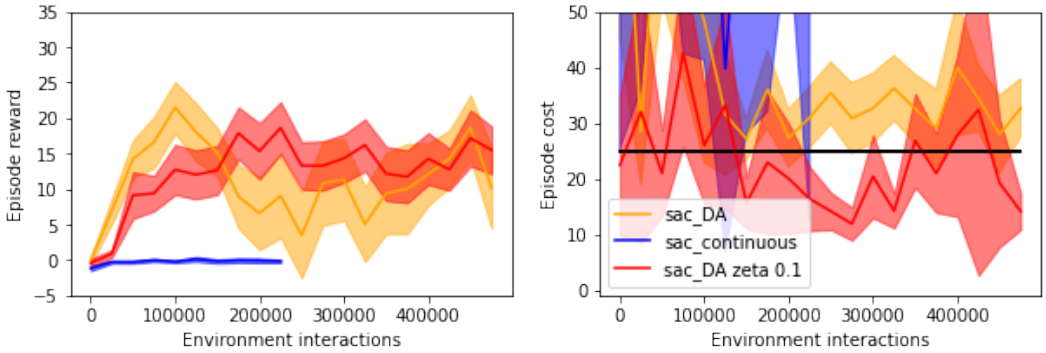


Figure A.29: SAC on augmented CarGoal1 with $\alpha = 0.99, \beta = 1$ with and without data augmentation. Cost representation as buckets unless stated otherwise ("continuous").

On CarGoal1, we can clearly see the combination of buckets and data augmentation outperform the continuous cost representation. $\zeta = 0.1$ seems to deal better with the constraint without leading to less reward than $\zeta = 0.01$.

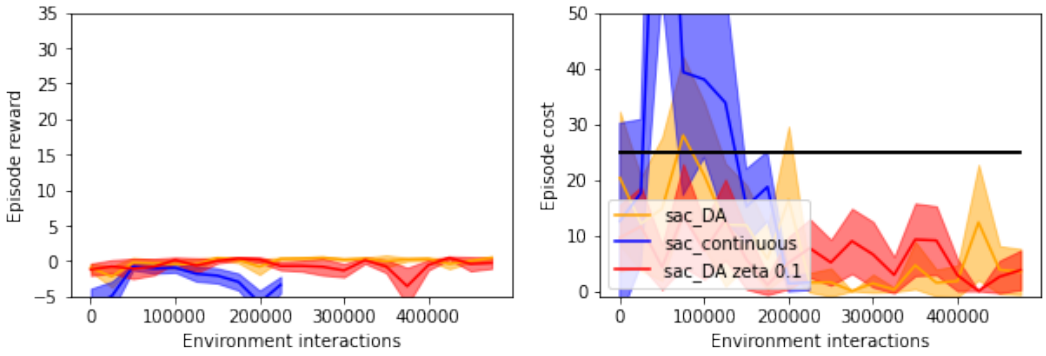


Figure A.30: SAC on augmented PointPush1 with $\alpha = 0.99, \beta = 1$ with and without data augmentation. Cost representation as buckets unless stated otherwise ("continuous").

For PointPush1, all rewards stay essentially zero, but they become negative more often for the continuous representation, which also takes longer to learn to respect the constraint.

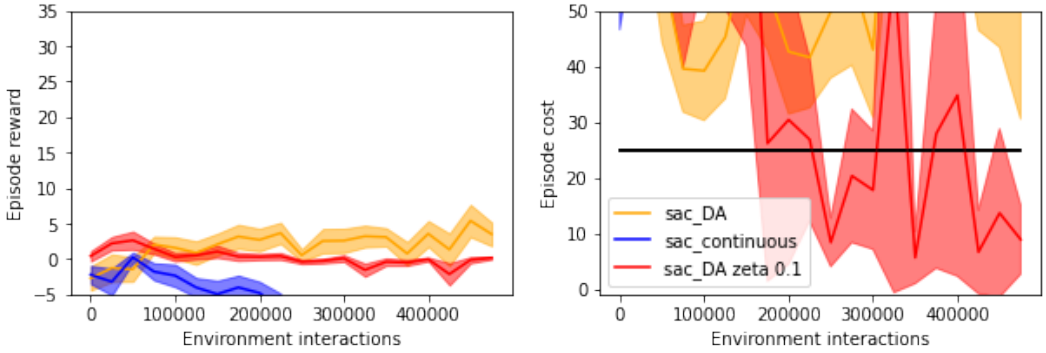


Figure A.31: SAC on augmented PointGoal2 with $\alpha = 0.99, \beta = 1$ with and without data augmentation. Cost representation as buckets unless stated otherwise ("continuous").

PointGoal2 looks similar for the comparison between continuous vs buckets/data augmentation. For $\zeta = 0.01$ rewards become positive but the constraint is clearly violated, while the constraint is mostly respected for $\zeta = 0.1$ at the cost of reaching zero reward. Lastly, we also tested data augmentation combined with buckets for TD3 on the other three tasks:

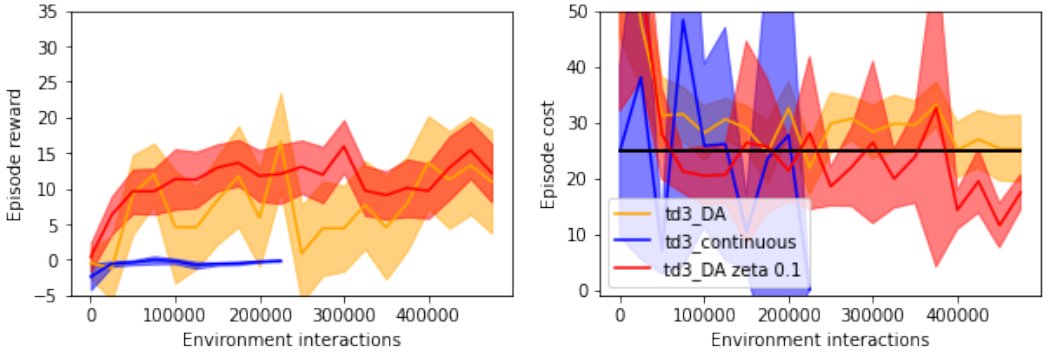


Figure A.32: TD3 on augmented CarGoal1 with $\alpha = 0.99, \beta = 1$ with and without data augmentation. Cost representation as buckets unless stated otherwise ("continuous").

Data augmentation and buckets clearly outperform the continuous representation in terms of reward and achieve comparable cost on CarGoal1. $\zeta = 0.1$ again looks slightly better in terms of cost while not missing out on reward.

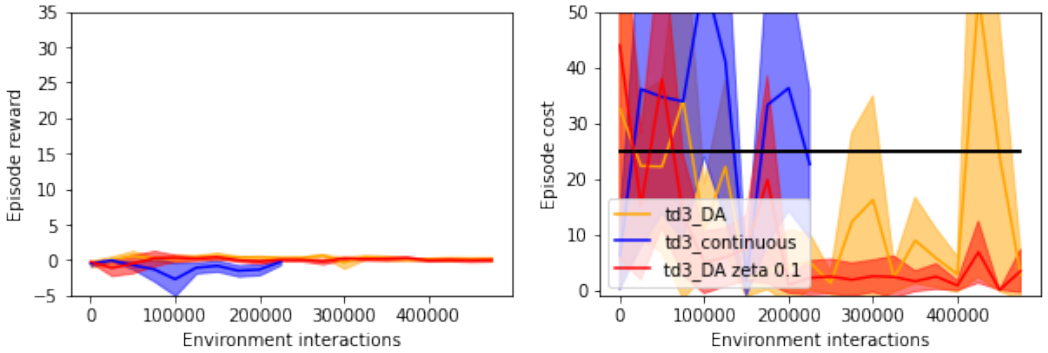


Figure A.33: TD3 on augmented PointPush1 with $\alpha = 0.99, \beta = 1$ with and without data augmentation. Cost representation as buckets unless stated otherwise ("continuous").

On PointPush1, data augmentation does not help TD3 to reach positive reward, but the incurred costs seem to be lower than for the continous representation.

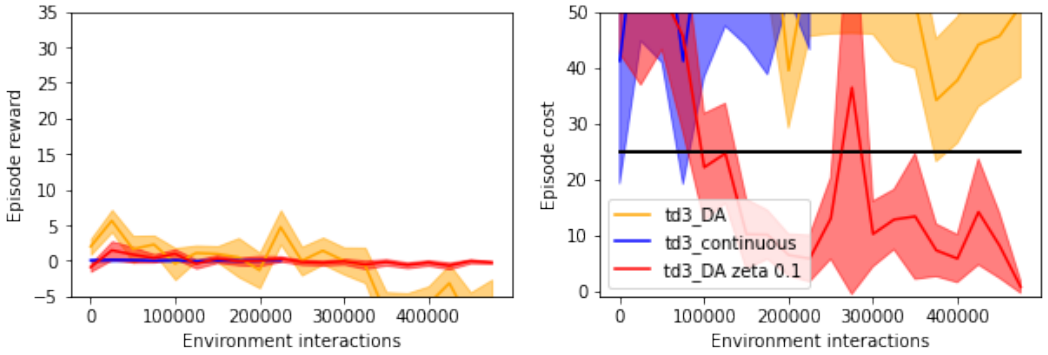


Figure A.34: TD3 on augmented PointGoal2 with $\alpha = 0.99, \beta = 1$ with and without data augmentation. Cost representation as buckets unless stated otherwise ("continuous").

On PointGoal2, reward for Td3 again stays close to zero for most of the training. Data augmentation with buckets and $\zeta = 0.1$ manages to learn to respect the constraint. Lastly, we include previous versions of figures 1(a) and 1(b) using shorter training runs:

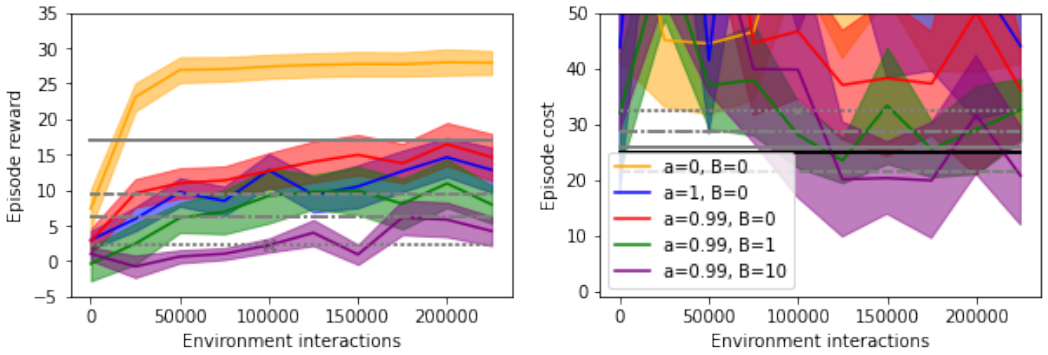


Figure A.35: SAC on augmented PointGoal1 with $\zeta = 0$. Cost representation continuous.

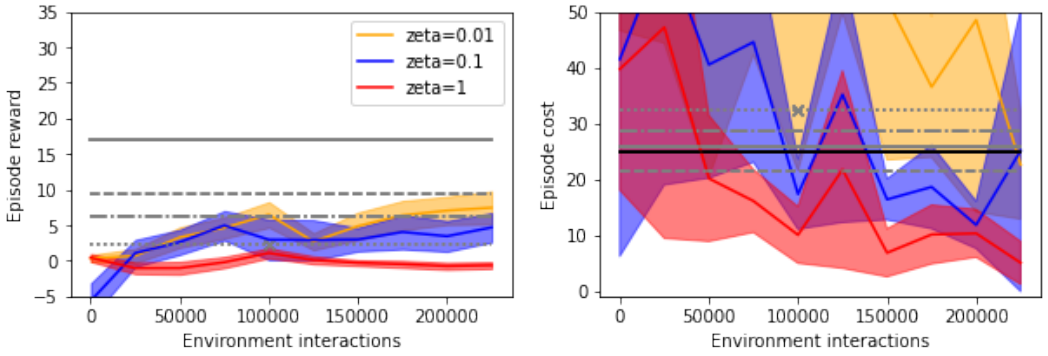


Figure A.36: SAC on augmented PointGoal1 with $\alpha = 0.99, \beta = 1$. Cost representation continuous.

A.3 Hyperparameters, plots and task descriptions

Unless stated otherwise, we used the default hyperparameters from Spinningup[11], expect that we used a fixed entropy constraint of -1 as described in [7] for SAC instead of a fixed

[11] https://github.com/openai/spinningup

reward scale/entropy bonus parameter[12].

The shaded region in our plots represent a 90% confidence interval assuming normally distributed errors for the reward/cost per episode averaged over 25 training episodes (100 training episodes for training on 5M or 10M environment steps.). We only report training performance, as we did not observe a benefits from removing stochasticity from the policies for testing.

For all tasks from SafetyGym, we directly apply our augmentation to the unmodified task. In particular, this means that the episode length is 1000 across the board. Appart from PointGoal1, we also look at the following tasks: CarGoal1 uses the same rewards and costs as PointGoal1 but the robot is replaced by a car that is more difficult to navigate as actions correspond to controlling a single front wheel rather than turning and moving forward/backward. TRPO-Lagrange achieves roughly 21 reward after 10M environment interactions on CarGoal1 and respects the constraint for the first 500k steps (at roughly zero reward) and for a longer period of multiple millions starting at around 1.5M steps. PointGoal2 is exactly like PointGoal1, but uses a bigger area and a larger number of hazards. TRPO-Lagrange achieves roughly 5 reward after 10M environment interactions on PointGoal2 and respects the constraint consistently after around 1M environment interactions. Lastly, PointPush1 uses the same point-shaped robot and costs as PointGoal1 but reward is obtained for pushing a box into a goal zone. TRPO-Lagrange achieves a reward of roughly 4.25 after 10M while staying below the cost constraint for the first 6M steps (at which point the reward is around 3) and slightly above the constraint afterwards.

[12]as implemented in our fork at https://github.com/flodorner/spinningup