WCSAC: Worst-Case Soft Actor Critic for Safety-Constrained Reinforcement Learning

Yang, Q.; Simão, T. D.; Tindemans, S.H.; Spaan, M.T.J.

Publication date
2021

Document Version
Accepted author manuscript

Published in
Proceedings of the 35th AAAI Conference on Artificial Intelligence (AAAI-21)

Citation (APA)

Important note
To cite this publication, please use the final published version (if applicable). Please check the document version above.
Hyperparameters

We list the hyperparameters used in the Empirical Analysis, which are summarized in Table 1. Unless an algorithm is specified, the setting is applicable for all methods, i.e., SAC, SAC-Lagrangian, and WCSAC. All runs in the experiment use separate feedforward Multilayer Perceptron (MLP) actor and critic networks. The size of all neural networks (all actors and critics of the algorithms) is $(32, 32)$. We use a replay buffer of size $10^6$ for each algorithm to store the experience. In each gradient step, we use batches of size 32 to update all function parameters. The discount factor is set to be $\gamma = 0.99$, and the target smoothing coefficient is set to be $\tau = 0.005$ to update the target networks. The learning rates of all parameters are set to be $0.001$. We set $d = 15$, $H_0 = -1$ for the safety constraint and entropy constraint respectively. The rest of the hyperparameters are explained in the Empirical Analysis part of the paper.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size of networks</td>
<td>$(32, 32)$</td>
<td></td>
</tr>
<tr>
<td>Size of replay buffer</td>
<td>$10^6$</td>
<td>$</td>
</tr>
<tr>
<td>Batch size</td>
<td>32</td>
<td></td>
</tr>
<tr>
<td>Discount factor</td>
<td>0.99</td>
<td>$\gamma$</td>
</tr>
<tr>
<td>Target smoothing coefficient</td>
<td>0.005</td>
<td>$\tau$</td>
</tr>
<tr>
<td>Learning rate</td>
<td>0.001</td>
<td></td>
</tr>
<tr>
<td>Epoch length</td>
<td>30000</td>
<td></td>
</tr>
<tr>
<td>Episodic length</td>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>Number of epochs</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>Safety constraint</td>
<td>15</td>
<td>$d$</td>
</tr>
<tr>
<td>Entropy constraint</td>
<td>$-1$</td>
<td>$H_0$</td>
</tr>
<tr>
<td>Risk level</td>
<td>$[0.1, 0.5, 0.9]$</td>
<td>$\alpha$ of WCSAC</td>
</tr>
<tr>
<td>Number of CPUs</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Evaluation episodes after training</td>
<td>100</td>
<td>With deterministic agent</td>
</tr>
</tbody>
</table>

Table 1: Summary of hyperparameters.
Environment builder

We use the Safety Gym engine (safety_gym.envs.engine.Engine) to create the point navigation environment (Ray, Achiam, and Amodei 2019). A configuration dict is used to specify the size of the map, the type of the robot, the task to finish, the size and location of the goal, the signals the agent can receive, and the size and location of the hazard. So we can create the environment through:

```python
from safety_gym.envs.engine import Engine

cfg = {
    'placements_extents': [-1.5, -1.5, 1.5, 1.5],
    'robot_base': 'xmls/point.xml',
    'task': 'goal',
    'goal_size': 0.3,
    'goal_keepout': 0.305,
    'goal_locations': [(1.1, 1.1)],
    'observe_goal_lidar': True,
    'observe_hazards': True,
    'constrain_hazards': True,
    'lidar_max_dist': 3,
    'lidar_num_bins': 16,
    'hazards_num': 1,
    'hazards_size': 0.7,
    'hazards_keepout': 0.705,
    'hazards_locations': [(0, 0)]
}

e = Engine(config)
```

As to more details about creating a custom environment, we refer the reader to Ray, Achiam, and Amodei (2019). The code is available at https://github.com/AlgTUDelft/WCSAC, which is adapted from Safety Starter Agents (Ray, Achiam, and Amodei 2019) and Spinning Up (Achiam 2018).
Adaptive weights

We present the change of adaptive safety weights and entropy weights during training in Figure 1, where a small weight (close to 0) means that the constraint is approximately satisfied. SAC method does not consider safety, so we do not have a plot for SAC in Subfigure (a). Compared to SAC (in the entropy weights) and SAC-Lagrangian, we can see that WCSAC methods (WCSAC-0.1, WCSAC-0.5, and WCSAC-0.9) need relatively more interactions with the environment to get constraint-satisfying policies. But the difference between the WCSAC methods is not obvious, e.g., WCSAC-0.1 does not need more interactions than WCSAC-0.9 to get constraint-satisfying policies, even though we have higher safety requirements for WCSAC-0.1.

![Figure 1: Change of safety weights and entropy weights during training.](image)

References
