Why Did the Robot Cross the Road? It was searching for the optimal policy!

Anonymous author

Abstract

Given the aim of solving the OpenAI Gym BipedalWalker-v3 environments in as few interactions as possible, this paper presents a method based on the Truncated Quantile Critics extension of the Soft Actor-Critic algorithm. The final implementation converges to a score over 300 in under 50 episodes for the easy environment, and is able to intermittently achieve similar scores on the hardcore environment – although it is unable to converge within the 1000 episode limit.

1 Methodology

We consider a (finite horizon) Markov decision process (MDP), defined by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, R, \gamma)$, where $\mathcal{S}$ and $\mathcal{A}$ represent the state and action spaces, $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, \infty)$ represents the unknown state transition density, R is a random variable reward function, and $\gamma \in [0, 1)$ is the discount factor. We propose a solution based on the off-policy Soft Actor-Critic (SAC) algorithm [7, 8, 9], extended with the overestimation bias reduction techniques from the Truncated Quantile Critics (TQC) approach [10].

The learning process of the agent – which involves the policy (π_ϕ), critic (Z_{ψ_n}), and target ($Z_{\bar{\psi}_n}$) networks, as well as the entropy temperature coefficient α – is governed by five key equations, which are computed in order:

1. **Entropy Temperature Loss ($\mathcal{L}_\alpha$):**

$$\mathcal{L}_\alpha = -\left(\log(\alpha) \cdot \left(\log \pi_\phi(a'_t \mid s_t) + \mathcal{H}_T\right)\right) \tag{1}$$

   We optimise the entropy temperature coefficient α to bring the stochastic estimate of the policy entropy $\log \pi_\phi(a'_t \mid s_t)$ closer to the target entropy $\mathcal{H}_T$, which is set heuristically to the negative dimensionality of the action space $\mathcal{A}$.

2. **Target Value ($y(s_t, a_t)$):**

$$y(s_t, a_t) = r_t + \gamma \left[z_{\bar{\psi}_n}(s_{t+1}, a'_{t+1}) - \alpha \log \pi_\phi(a'_{t+1} \mid s_{t+1})\right] \tag{2}$$

   TQC computes the temporal difference targets for the smallest kN elements of the quantile values of the next state and action, $z_{\bar{\psi}_n}(s_{t+1}, a'_{t+1})$, which is how the overestimation bias reduction is achieved.

3. **Quantile Critic Loss ($\mathcal{L}_{Z_{\psi_n}}$):**

$$\mathcal{L}_{Z_{\psi_n}} = \text{HuberQuantileLoss}(z_{\psi_n}(s_t, a_t), y(s_t, a_t)) \tag{3}$$

   The critic loss is defined similarly to standard methods, as the difference between the predicted and target values, but differing in the use of quantile values rather than mean estimates.

4. **Policy Loss ($\mathcal{L}_{\pi_\phi}$):**

$$\mathcal{L}_{\pi_\phi} = \alpha \log \pi_\phi(s_t \mid a'_t) - z_{\psi_n}(s_t, a'_t) \tag{4}$$

Similarly to the critic loss, the policy loss takes a distributional approach [3, 6], using quantile-based estimates rather than a single Q-value as in standard SAC.

5. **Target Network Update:**

$$\bar{\psi}_n = (1-\tau)\bar{\psi}_n + \tau\psi_n \tag{5}$$

The target network is updated using a Polyak update based on the hyperparameter τ.

The derivation process of the constituent terms of these equations is depicted in Figure 1 (authors' own). We refer the reader to the TQC paper for further details – whilst noting that, for the sake of clarity, our notation differs slightly from theirs.

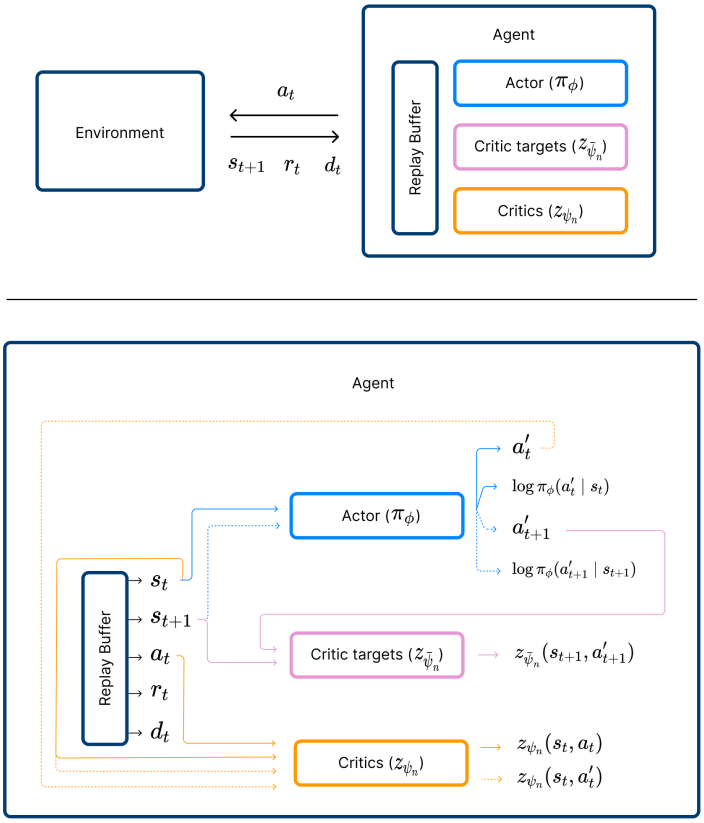

Figure 1: **Top:** A diagram representing the information exchanged between the agent and the environment during a single timestep. The agent makes action a_t given state s_{t-1}, the environment then returns the appropriate next state, reward, and 'done' information (s_{t+1}, r_t, d_t). **Bottom:** A diagram depicting the computations that occur within the agent during a single training step. These values are used to optimise the weights of the policy (π_ϕ), critic (Z_{ψ_n}), and target $(Z_{\bar{\psi}_n})$ networks via equations 1-5.

2 Convergence Results

Figure 2 presents the convergence results of our final implementation, referred to as TQC+, on both the easy and hardcore OpenAI Gym BipedalWalker-v3 environments [4]. In the spirit of demonstrating robustness, we use the same model configuration and hyperparameters for both environments.

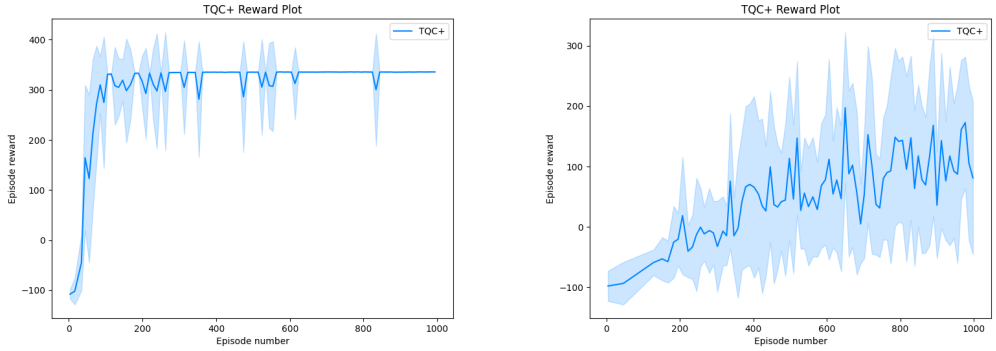

Figure 2: **Left:** Reward plot for the easy environment. The agent reaches a score of 300 in 49 episodes and a maximum reward of 336. **Right:** Reward plot for the hardcore environment. The agent reaches a maximum score of 313 but does not converge within 1000 episodes. In both plots, the line represents the average reward across the last 10 runs, whilst the shading indicates the standard deviation over the same period.

3 Experiments

Figure 3 presents the performance of our solution at various stages of its evolution. We began with a purely SAC approach, which we then extended into a TQC implementation. Our final implementation extends this slightly further and is a byproduct of extensive experimentation, as summarized in Table 1.

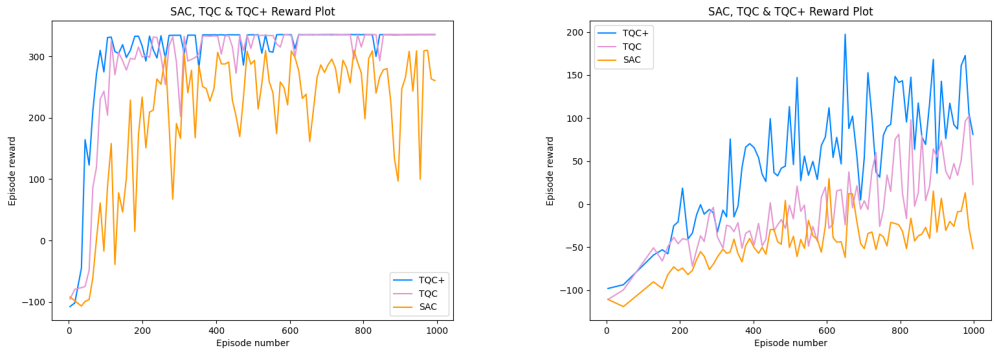

Figure 3: **Left:** Reward plot for the easy environment. **Right:** Reward plot for the hardcore environment. For both plots, the original SAC implementation is in yellow, TQC is in pink, and the final implementation is in blue. The plots reveal that each stage of evolution outperforms the last on both environments – given a metric of cumulative reward (AUC).

Technique	Description	Adopted?
Orthogonal Initialisation	Rather than Kaiming	×
Prioritized Experience Replay	[13]	×
Emphasizing Recent Experience	[14]	×
State Dependent Exploration	[10]	×
LR Scheduler	To expedite convergence	×
Layer Norm	To stabilize training [2]	×
Fixed Exploration Schedule	[8]	×
Sample Multiple Reuse	[12]	×
Increased Update-Data Ratio	[5]	✓
Variable Batch Size	$32 \rightarrow 256$ to balance robustness and convergence	✓
Uniform Sampling for n Steps	Diverse initial population in replay buffer	✓
Bayesian Hyperparameter Optimization	[1]	✓

Table 1: A summary of the additional techniques experimented with on top of the base TQC architecture, four of which are included within the final (TQC+) implementation.

4 Discussion

The code for our final implementation is adapted from the official TQC implementation [11]. This official implementation is not compatible with modern versions of PyTorch, so some (non-trivial) rearrangement of the loss computations was required (Section 1).

Beyond this change, and as summarized in Table 1, our implementation makes some subtle tweaks to the standard TQC algorithm for slightly improved performance (Section 3). Most of these are standard, hence, we refer the reader to the associated references for more information. One non-standard choice is to increase the batch size during training. We do this to address the lack of convergence of the original implementation, which only performs gradient updates after `batch_size` environment steps, which means that no updates are performed during the episodes where the agent dies before this threshold has been met. Given that large batch sizes are beneficial for training stability we increase the batch size through training to balance these convergence and robustness considerations.

5 Limitations

The performance of our implementation is highly sensitive to the choice of hyperparameters. Beyond being a fundamental limitation from a methodological perspective, this also has negative repercussions from an experimental perspective. Concretely, intuitively promising model enhancements would result in a decrease in model performance. It is plausible that these drops in performance are not caused by the lack of efficacy of the enhancements themselves, but rather the fact that they knock the model out of its local minima within the optimization landscape. Thus, lacking the computational resources to perform hyperparameter optimization for every model variant, a number of promising methods were discarded despite their potential to unlock better performance overall.

Another, somewhat related, limitation of our approach is that we present an architecture whose hyperparameters have been optimized for a single random seed, rather than an average of many. Consequently, we can make no guarantees about the general performance of the model. Nevertheless, we worked to mitigate this limitation by conducting only a light hyperparameter search with 20 trials, and used the same hyperparameters on both the easy and hardcore environments. Given this context, if one were to seek to achieve convergence on the hardcore environment within 1000 episodes, the most promising next step would be to perform a more thorough hyperparameter search, bespoke to the hardcore environment.

References

[1] Takuya Akiba et al. *Optuna: A Next-generation Hyperparameter Optimization Framework*. 2019. arXiv: 1907.10902 [cs.LG].

[2] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. *Layer Normalization*. 2016. arXiv: 1607.06450 [stat.ML].

[3] Marc G. Bellemare, Will Dabney, and Rémi Munos. *A Distributional Perspective on Reinforcement Learning*. 2017. arXiv: 1707.06887 [cs.LG].

[4] Greg Brockman et al. *OpenAI Gym*. 2016. arXiv: 1606.01540 [cs.LG].

[5] Xinyue Chen et al. *Randomized Ensembled Double Q-Learning: Learning Fast Without a Model*. 2021. arXiv: 2101.05982 [cs.LG].

[6] Will Dabney et al. *Implicit Quantile Networks for Distributional Reinforcement Learning*. 2018. arXiv: 1806.06923 [cs.LG].

[7] Tuomas Haarnoja et al. *Learning to Walk via Deep Reinforcement Learning*. 2019. arXiv: 1812.11103 [cs.LG].

[8] Tuomas Haarnoja et al. *Soft Actor-Critic Algorithms and Applications*. 2019. arXiv: 1812.05905 [cs.LG].

[9] Tuomas Haarnoja et al. *Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor*. 2018. arXiv: 1801.01290 [cs.LG].

[10] Arsenii Kuznetsov et al. *Controlling Overestimation Bias with Truncated Mixture of Continuous Distributional Quantile Critics*. 2020. arXiv: 2005.04269 [cs.LG].

[11] Arsenii Kuznetsov et al. *tqc_pytorch: Implementation of the TQC algorithm in PyTorch by Samsung Research*. https://github.com/SamsungLabs/tqc_pytorch. Accessed: 2024-03. 2020.

[12] Jiafei Lyu et al. *Off-Policy RL Algorithms Can be Sample-Efficient for Continuous Control via Sample Multiple Reuse*. 2023. arXiv: 2305.18443 [cs.LG].

[13] Tom Schaul et al. *Prioritized Experience Replay*. 2016. arXiv: 1511.05952 [cs.LG].

[14] Che Wang and Keith Ross. *Boosting Soft Actor-Critic: Emphasizing Recent Experience without Forgetting the Past*. 2019. arXiv: 1906.04009 [cs.LG].