COUNT-BASED TEMPERATURE SCHEDULING FOR MAXIMUM ENTROPY REINFORCEMENT LEARNING

Dailin Hu[1,*], **Pieter Abbeel**[2], **Roy Fox**[1]

[1]Department of Computer Science, University of California, Irvine
[2]Department of Electrical Engineering and Computer Science,
University of California, Berkeley

ABSTRACT

Maximum Entropy Reinforcement Learning (MaxEnt RL) algorithms such as Soft Q-Learning (SQL) and Soft Actor–Critic trade off reward and policy entropy, which has the potential to improve training stability and robustness. Most MaxEnt RL methods, however, use a constant tradeoff coefficient (temperature), contrary to the intuition that the temperature should be high early in training to avoid overfitting to noisy value estimates and decrease later in training as we increasingly trust high value estimates to truly lead to good rewards. Moreover, our confidence in value estimates is state-dependent, increasing every time we use more evidence to update an estimate. In this paper, we present a simple state-based temperature scheduling approach, and instantiate it for SQL as Count-Based Soft Q-Learning (CBSQL). We evaluate our approach on a toy domain as well as in several Atari 2600 domains and show promising results.

1 INTRODUCTION

Deep Reinforcement Learning (RL) methods that use neural-network function approximators to learn control policies have shown great performance in domains including games (Mnih et al., 2015; Silver et al., 2017), robotic control (Haarnoja et al., 2017), and autonomous driving (Sallab et al., 2017). The training of such a function approximator, however, is often very sensitive to hyperparameter tuning in different environments (Henderson et al., 2018). Recent work in Maximum Entropy Reinforcement Learning (MaxEnt RL) (Fox et al., 2016; Haarnoja et al., 2018) trades off maximizing the policy values with its entropy, producing policies that are more robust to disturbances in the environment dynamics and reward function (Haarnoja et al., 2017; Eysenbach & Levine, 2021).

MaxEnt RL follows from the maximum entropy principle (Jaynes, 2003), which states that one should prefer maximally uninformed solutions, subject to the available evidence. MaxEnt RL therefore maximizes average policy entropy, subject to an attainable level of policy value. The Lagrangian of this constrained optimization problem trades off the traditional RL objective of maximum policy value with the expected policy entropy. The Lagrange multiplier is an *inverse-temperature* parameter β that determines the relative importance of the value and entropy terms, and corresponds to a level of value that we believe we can attain. Intuitively, in early stages of training, β should be lower to introduce higher stochasticity as we have low confidence that the current value estimates are attainable. During training, we become increasingly more confident in our value estimates, and we increase β such that the policy stochasticity decreases and eventually approaches zero. As $\beta \to \infty$, the conventional RL learning objective is recovered.

Most MaxEnt RL algorithms, such as Soft Q-Learning (SQL) (Haarnoja et al., 2017), use a constant temperature throughout training. The log-partition function in SQL's soft Bellman backup operator allows it to put higher weight on the policy entropy when the temperature is high, or approximate reward maximization arbitrarily well as the temperature decreases to 0. Selecting a constant inverse-temperature β may not reflect the changing confidence of the value estimates throughout learning. Even when there exists a constant β that works well, it is domain-dependent, which makes it hard to

*Correspondence to: dailinh@uci.edu

tune as a hyperparameter. Soft Actor–Critic (SAC) (Haarnoja et al., 2018) adjusts the temperature β^{-1} automatically with stochastic gradient descent, but often has poor performance in discrete action spaces (Christodoulou, 2019). Fox et al. (2016) proposes a linear inverse-temperature schedule, in which $\beta_i = \kappa i$ in training step i. This corresponds to the intuition that β should increase throughout learning, but the coefficient κ remains a domain-dependent hyperparameter to be tuned.

Importantly, our confidence in SQL's value estimates is very much state-dependent. Value estimates that have been updated more times, from more data, are more reliable. States that are more often experienced by early exploration will have their value estimates improve in accuracy, and their β should increase. Later in learning, some of these states may not be encountered as often, becoming less important to train, and their β should not increase as fast. When exploration reaches novel states, whose value estimates are less reliable, their β should start very low again, even late in training.

In this paper, we describe a simple state-dependent temperature scheduling method for MaxEnt RL based on a pseudo-count of state value updates derived from a CTS density model (Ostrovski et al., 2017). As a concrete instance of this method, adapted to the SQL algorithm, we present the Count-Based Soft Q-Learning (CBSQL) algorithm. Evaluating CBSQL on 6 popular games on the Atari 2600 platform suggests that our scheduling method improves over DQN and SQL with a fixed temperature, and display the potential to reach state-of-the-art performance with Rainbow integration (Hessel et al., 2018).

2 Preliminaries

We address MaxEnt RL in discrete action spaces. In this section, we will introduce the notation and framework of MaxEnt RL.

2.1 Reinforcement Learning

We consider environments modeled as a Markov Decision Process (MDP). We describe this process by a tuple $\langle \mathcal{S}, \mathcal{A}, p, r \rangle$, where $\mathcal{S}$ represents a state space and $\mathcal{A}$ represents a discrete action space. $p(s_{t+1}|s_t, a_t) : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ represents the probability distribution of the next state s_{t+1} given the current state s_t and action a_t. $r_t = r(s_t, a_t) : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ describes the reward for each transition t. The discounted return R, for a discount factor $0 \leq \gamma < 1$, is defined as $R = \sum_t \gamma^t r_t$.

A reinforcement learning agent learns a policy $\pi(a_t|s_t)$ for interacting with its environment. The agent and the environment jointly induce a distribution $p_\pi(\xi)$ over trajectories $\xi = s_0, a_0, s_1, a_1, \ldots$. It is also convenient to define a state distribution $p_\pi^\gamma(s) = (1-\gamma)\sum_t \gamma^t p_\pi(s_t = s)$, describing the distribution of s_t at a time t distributed geometrically with parameter $1-\gamma$. This satisfies $\mathbb{E}_{\xi \sim p_\pi}[R(\xi)] = \frac{1}{1-\gamma}\mathbb{E}_{s \sim p_\pi^\gamma}[\mathbb{E}_{(a|s)\sim\pi}[r(s,a)]]$.

2.2 Maximum Entropy Reinforcement Learning

The MaxEnt RL objective augments the standard reinforcement learning objective of maximizing expected discounted rewards by adding an entropy term

$$\pi^* = \arg\max_\pi \mathbb{E}_{s \sim p_\pi^\gamma}\left[\mathbb{E}_{(a|s)\sim\pi}[r(s,a)] + \tfrac{1}{\beta}\mathbb{H}[\pi(\cdot|s)]\right], \tag{1}$$

where β is an *inverse-temperature* parameter that controls the stochasticity of the optimal policy by determining the relative importance between the reward and policy entropy $\mathbb{H}[\pi]$. In early stages of training, β should intuitively be assigned a small value to induce stochasticity, and in later stages of training $\beta \to \infty$ to approach the deterministic behavior that optimizes the standard reinforcement learning objective.

2.3 Soft Q-Learning (SQL) and the Mellowmax Operator

RL methods often involve the state–action value function $Q(s,a) = \mathbb{E}[R|s_0 = s, a_0 = a]$ and the Bellman operator $\mathcal{B}[Q](s,a) = r(s,a) + \gamma\mathbb{E}_{(s'|s,a)\sim p}[\max_{a'} Q(s',a')]$. The MaxEnt RL objective

suggests a *soft* Bellman operator (Rubin et al., 2012)

$$\mathcal{B}[Q](s,a) = r(s,a) + \gamma \mathbb{E}_{(s'|s,a)\sim p}\left[\max_{\pi}\left(\mathbb{E}_{(a'|s')\sim\pi}[Q(s',a')] + \tfrac{1}{\beta}\mathbb{H}[\pi(\cdot|s')]\right)\right] \tag{2}$$

$$= r(s,a) + \gamma \mathbb{E}_{(s'|s,a)\sim p}\left[\tfrac{1}{\beta}\log\sum_{a'\in\mathcal{A}}\exp(\beta Q(s',a'))\right] \qquad \forall s\in\mathcal{S}, a\in\mathcal{A}, \tag{3}$$

where the optimizer in (2) is the softmax policy $\pi(a|s) \propto \exp(\beta Q(s,a))$, and the log-sum-exp expression in (3) is the log-partition function, also called the mellowmax operator (Asadi & Littman, 2017). Mellowmax is non-decreasing in β (Kim et al., 2019) and a non-expansion for a fixed β under the supremum norm (Fox et al., 2016), and as $\beta \to \infty$ it converges to pure maximization.

Soft Q-Learning (Fox et al., 2016; Haarnoja et al., 2017) uses a model-free empirical estimate of the soft Bellman operator as the target for learning a Q function. When the Q function has a tabular representation, the contraction property of the soft Bellman operator guarantees convergence to the operator's fixed point, the softmax of which is the optimal policy in (1) (Fox et al., 2016).

3 Count-Based Temperature Scheduling

3.1 Motivation

Empirical evidence from SQL suggests that a state-independent linear temperature scheduling, i.e. using $\beta_i = \kappa i$ in iteration i, can achieve good performance (Fox et al., 2016; Grau-Moya et al., 2018). Fox (2019) proposes a closed-form state-dependent expression for the temperature that completely eliminates bias in entropy-regularized value updates in two-action environments. If the gap between the action values $Q(s,a=0) - Q(s,a=1)$ has Gaussian uncertainty with mean μ_A and variance σ_A^2, the prescribed inverse-temperature is $\beta(s) = \frac{2\mu_A}{\sigma_A^2}$. The Gaussian assumption is motivated by the central limit theorem, which also suggests that the asymptotic behavior of β is to grow linearly in the number of data points used to estimate the value gap at each state.

We propose Count-Based Soft Q-Learning (CBSQL), an algorithm based on SQL that uses a state-dependent temperature schedule in which $\beta(s)$ grows linearly with the number of times that the algorithm updates $Q(s,a)$, for any action a. Formally, let $n(s,a)$ be the count of sampled data points of the form (s,a,r,s') used thus far in value updates. Then the inverse-temperature in CBSQL is $\beta(s) = \kappa\sum_a n(s,a)$, with $\kappa > 0$ a constant hyperparameter.

In addition to the aforementioned empirical and theoretical evidence supporting linear count-based scheduling, more insight can be gained by comparing two families of successful RL algorithms. The first family, consisting of such algorithms as G-Learning (Fox et al., 2016), SQL (Haarnoja et al., 2017), Path Consistency Learning (PCL) (Nachum et al., 2017a), and Soft Actor–Critic (SAC) (Haarnoja et al., 2018), aims to learn a policy that is the softmax of a value function $Q(s,a)$ with a low temperature. In iteration i, the policy target is

$$\pi_i(a|s) \propto \pi_0(a|s)\exp\beta_i(s)Q_i(s,a) \qquad \forall s\in\mathcal{S}, a\in\mathcal{A}, \tag{4}$$

with π_0 the uniform policy, and the policy is either updated toward the target (4), or set equal to it. Note that in their original formulations, most of these algorithms use a constant state-independent β. The second family of algorithms, consisting of Relative Entropy Policy Search (REPS) (Peters et al., 2010), Ψ-learning (Rawlik et al., 2010), Trust Region Policy Optimization (TRPO) (Schulman et al., 2015), and Trust-PCL (Nachum et al., 2017b), aims to use value estimates to gradually update the softmax policy. Instead of the policy entropy, which is the Kullback–Leibler (KL) divergence from a uniform prior policy, these algorithms consider a KL term with the current policy as the prior for the update. Moreover, instead of a low temperature β^{-1}, these algorithms place a large coefficient κ^{-1} on the KL term, to induce small updates in a "trust region". In iteration i, the policy update is therefore

$$\pi_i(a|s) \propto \pi_{i-1}(a|s)\exp\kappa_i(s)Q_i(s,a) \propto \pi_0(a|s)\exp\left(\sum_{j\le i}\kappa_j(s)Q_j(s,a)\right). \tag{5}$$

The two types of learning processes (4) and (5) can have different properties, because Q_i can change significantly between iterations. For the sake of our intuitive argument here, imagine that Q could be

approximated by a constant, and that the inverse-temperatures $\kappa_i(s)$ used in trust-region algorithms were also a constant κ. Then combining the above two equations, we have

$$\beta_i(s) \approx \sum_{j \leq i} \kappa_j(s) \approx \kappa i, \tag{6}$$

where κ is a small constant, and i is the number of times that the update equation (5) is applied in state s. In practice, updates are not applied to all actions in state s at the same time, leading to the heuristic definition $i = \sum_a n(s, a)$.

3.2 Pseudo-counts

Directly recording the number of times that the value update has been applied to state s is not useful in practical settings, where the state space is large, and most states will rarely be visited more than once. Moreover, a tabular mapping from s to i would fail to capture the similarity between different states that a Q function approximator does leverage, and would vastly underestimate the effective number of times that $Q(s, a)$ has been updated in states similar to s.

Instead, we use a pseudo-count method derived by Bellemare et al. (2016) from a simplified pixel-level CTS density model (Bellemare et al., 2014). Define ρ to be the CTS density model on the state space $\mathcal{S}$. Let $\rho_k(s)$ be the probability assigned by the model to $s \in \mathcal{S}$ after k updates of the model. Let $\rho'_k(s)$ be the probability that the model would assign to s if it were updated on s one more time. The pseudo-count can then be defined as

$$n_k(s) = \frac{\rho_k(s)(1 - \rho'_k(s))}{\rho'_k(s) - \rho_k(s)}. \tag{7}$$

Note that k is different from i in the last section and represents the total number of updates in all states. The effective number of times that state s has been updated is its pseudo-count $n_k(s)$, suggesting the state-dependent inverse-temperature

$$\beta(s) = \kappa \cdot n_k(s), \tag{8}$$

where κ is a small constant. Throughout our experiments, we set $\kappa = 0.01$.

In summary, we propose a model-free reinforcement learning algorithm that, on experience (s, a, r, s'), uses the SQL update rule

$$Q(s, a) \leftarrow r + \gamma \operatorname*{mellowmax}_{a'; \beta(s')} Q(s', a') = r + \tfrac{\gamma}{\beta(s')} \log \tfrac{1}{|\mathcal{A}|} \sum_{a' \in \mathcal{A}} \exp(\beta(s') Q(s', a')). \tag{9}$$

In tabular experiments, we also update

$$n(s) \leftarrow n(s) + 1, \tag{10}$$

whereas in deep-learning experiments, we update the density model with state s, and consider the pseudo-count (7). The inverse-temperature is then scheduled as $\beta(s) = \kappa \cdot n(s)$. We present the pseudocode in Alg 1.

4 Experiments

In this section, we evaluate our approach in tabular representation on a toy domain, and in deep learning on six popular Atari 2600 environments.

4.1 Tabular Representation

We compare our approach to DQN and SQL on a toy domain of noisy transitions in a chain-linked graph. This environment is small enough to directly find the optimal policy.

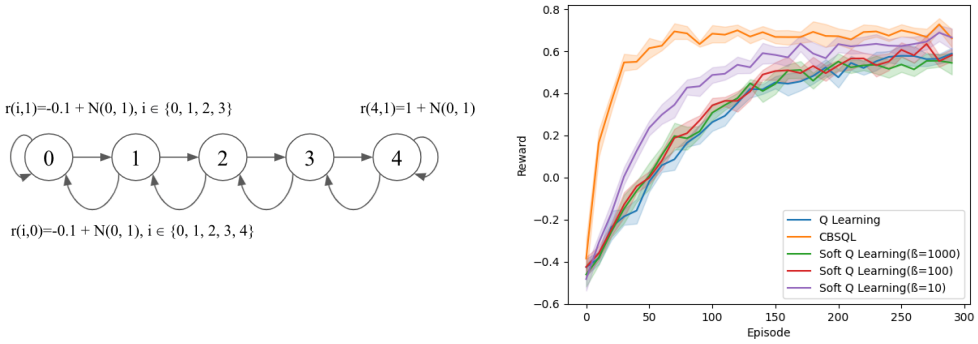
We define a noisy chain-walk problem with five states, each state with two possible actions (Figure 1a). The agent always starts at state 0 and each episode ends after 5 steps. The agent receives a reward of $+1$ when it takes action 1 at state 4, and a reward of -0.1 when it takes any action at any

Algorithm 1 Count-Based Soft Q Learning (SQL)

Initialize Q network parameters θ
Initialize target Q network $\bar{\theta} \leftarrow \theta$
Initialize an empty replay buffer $\mathcal{D} \leftarrow \emptyset$
Initialize a density model ρ
for each iteration **do**
  for each step t in the rollout **do**
    In state s_t, sample action a_t from the ϵ-greedy policy for $Q_\theta(s_t, \cdot)$
    Execute action a_t and observe reward r_t and new state s_{t+1}
    Store the transition (s_t, a_t, r_t, s_{t+1}) into the replay buffer $\mathcal{D}$
  end for
  for each gradient step **do**
    Sample random batch (s, a, r, s') from $\mathcal{D}$
    $\beta \leftarrow \kappa \cdot \frac{\rho(s')(1-\rho'(s'))}{\rho'(s')-\rho(s')}$
    $y = r + \gamma \, \text{mellowmax}_{a';\beta} \, Q_{\bar{\theta}}(s', a')$
    Perform gradient descent on $(y - Q_\theta(s, a))^2$
    Update the density model with state s
    Every `target_freq` steps, update $\bar{\theta} \leftarrow \theta$
  end for
end for

other states. The reward that agent receives is always corrupted with an additive Gaussian noise of $\mathcal{N}(0, 1)$.

The optimal expected reward is 0.6. Figure 1b shows the rewards averaged over 1000 runs comparing Q-learning, SQL with different constant inverse-temperature parameters $\beta \in \{10, 100, 1000\}$, and CBSQL with a tabular representation. These results suggest that CBSQL can converge significantly faster and obtain higher rewards than both SQL and Q learning within 300 episodes of training in this simple domain.



(a) Domain Demonstration

(b) Rewards averaged over 10000 runs

Figure 1: Noisy Chain-Walk Problem

4.2 Neural Network Representation

We evaluate our method on six Atari 2600 game from the arcade learning environment (Bellemare et al., 2013) and compare it with DQN and SQL under the same hyperparameter setting from Mnih et al. (2015). We train the agents on 3M frames and record the rewards. CBSQL appears to outperform DQN and fixed-temperature SQL in 4 of the 6 games (Table 1, Figure 2).

Game	DQN	SQL($\beta = 100$)	SQL($\beta = 1000$)	CBSQL
Breakout	5.9 ($\pm$5.9)	5.9 ($\pm$4.5)	5.1 ($\pm$4.7)	**8.2** ($\pm$6.1)
Freeway	21.0 ($\pm$1.5)	14.6 ($\pm$8.5)	22.56 ($\pm$4.7)	**25.82** ($\pm$4.9)
Pong	1.93 ($\pm$2.6)	**17.83** ($\pm$2.2)	16.31 ($\pm$2.7)	17.56 ($\pm$2.0)
Q*bert	568.4 ($\pm$1101.9)	828 ($\pm$1411.7)	564.5 ($\pm$1097.5)	**875.3** ($\pm$1254.6)
Seaquest	13.5 ($\pm$24.1)	4 ($\pm$60.2)	17.2 ($\pm$24.0)	**84.6** ($\pm$60.2)
Space Invaders	132.7 ($\pm$113.2)	**158.9** ($\pm$128.5)	132.25 ($\pm$118.4)	138.9 ($\pm$112.8)

Table 1: DQN, fixed-temperature SQL, and CBSQL average rewards ($\pm$ standard deviation). Raw scores are averaged over the last 100 testing episodes across 3 runs.

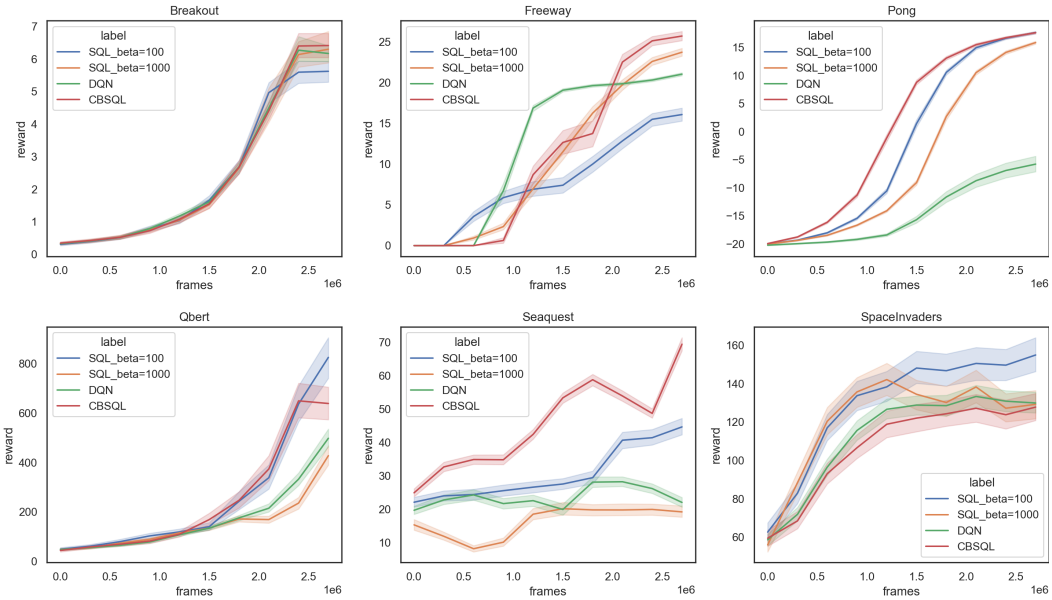


Figure 2: CBSQL results compared with DQN and fixed-temperature SQL, without Rainbow. Rewards are averaged over 3 runs.

DQN by itself is a powerful algorithm, but in recent years more extensions to it have been proposed that greatly improve its performance. Interestingly, SQL and CBSQL can be combined with many of these extensions, allowing us to compare these methods after integrating popular DQN extensions. We integrate CBSQL with Rainbow DQN (Hessel et al., 2018), a state-of-the-art reinforcement learning algorithm for memoryless agents, which includes multi-step targets (Sutton & Barto, 2018), double Q-learning (Hasselt, 2010), prioritized experience replay (Schaul et al., 2015), dueling networks (Wang et al., 2016), distributional RL (Bellemare et al., 2017), and noisy networks (Fortunato et al., 2017). All of these methods can be straightforwardly applied to soft Q-learning, with the exception of multi-step targets and distributional RL, which we discuss next.

Multi-step learning. Multi-step targets with a well-tuned number of steps n can lead to faster learning in on-policy RL algorithms (Sutton & Barto, 2018) by trading off the bias and variance of the return estimates (Kearns & Singh, 2000). The n-step truncated return at time t is $r_t^{(n)} = \sum_{k=0}^{n-1} \gamma^k r(s_{t+k}, a_{t+k})$. Hessel et al. (2018) defined a multi-step variant of DQN by minimizing the alternative loss $(r_t^{(n)} + \gamma^n \max_{a \in \mathcal{A}} Q_{\bar{\theta}}(s_{t+n}, a) - Q_\theta(s_t, a_t))^2$ and demonstrated through empirical experiments that n-step targets can outperform single-step targets in DQN, despite the off-policy experience providing biased estimates of the n-step return. In SQL with inverse-temperature β, the n-step truncated return is

$$\tilde{r}_t^{(n)} = r_t^{(n)} + \tfrac{1}{\beta} \sum_{k=1}^{n-1} \gamma^k \mathbb{H}[\pi(\cdot|s_{t+k})]. \tag{11}$$

Unfortunately, empirical policy entropy estimates are often very noisy. It is also unclear which β should be used in off-policy estimates of (11). These considerations call for further study, and in this work we simply use 1-step returns for SQL and CBSQL.

Distributional RL. Unlike conventional RL which estimates the expected return, distributional RL estimates the distribution of the return at time t over the stochasticity of the environment and the policy. The estimator uses a fixed N-dimensional vector $\boldsymbol{z}$ of values spaced evenly along the range $[v_{\min}, v_{\max}]$ of possible returns. The distribution of $Q(s_t, a_t)$ is then represented by a categorical distribution $\boldsymbol{p}_\theta(s_t, a_t)$ over the values of $\boldsymbol{z}$. Distributional DQN updates $\boldsymbol{p}_\theta(s, a)$ by minimizing its KL-divergence from a projected target distribution induced by the categorical distribution $\boldsymbol{p}_{\bar{\theta}}(s', a^*)$ over the values $r + \gamma \boldsymbol{z}$. Here the action a^* is chosen greedily with $a^* = \arg\max_{a'} \boldsymbol{z}^\intercal \boldsymbol{p}_{\bar{\theta}}(s', a')$. We adapt distributional RL to soft Q-learning and CBSQL by defining a policy distribution of

$$\pi(a'|s') = \frac{\exp \beta(s') \boldsymbol{z}^\intercal \boldsymbol{p}_{\bar{\theta}}(s', a')}{\sum_{\bar{a}'} \exp \beta(s') \boldsymbol{z}^\intercal \boldsymbol{p}_{\bar{\theta}}(s', \bar{a}')}$$

over the values $r + \gamma \left(\boldsymbol{z} - \frac{1}{\beta} \mathbb{D}[\pi(\cdot|s') \| \pi_0] \right)$.

We train the Rainbow-variations of SQL and CBSQL as described above over 500K frames each for the atari game Breakout and Q*bert against Rainbow-DQN(See Fig 3). Rainbow-CBSQL outperforms Rainbow DQN in both these environments.

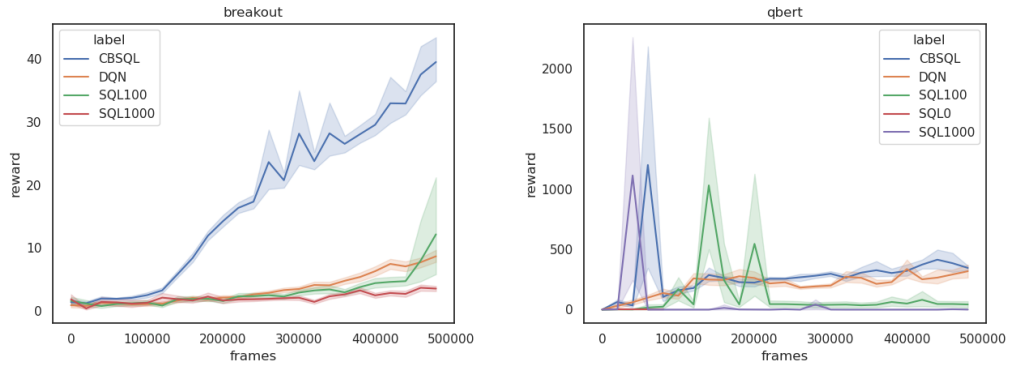


Figure 3: CBSQL results compared with DQN and fixed-temperature SQL, with Rainbow. Rewards are averaged over 5 runs.

Game	DQN	SQL($\beta = 100$)	SQL($\beta = 1000$)	CBSQL
Breakout	10.2 ($\pm$ 4.9)	5.5 ($\pm$ 3.6)	3.6 ($\pm$ 2.1)	**39.9** ($\pm$ 15.4)
Q*bert	356.0 ($\pm$ 202.8)	30.0 ($\pm$ 60.0)	1.0 ($\pm$ 4.9)	**370.0** ($\pm$ 43.0)

Table 2: CBSQL results compared with DQN and SQL with fixed temperatures, combined with Rainbow. Raw scores averaged over last 50 testing episodes across 5 runs.

5 Conclusion

In this paper, we presented a simple method for temperature scheduling in soft Q learning which could be potentially applied to other maximum entropy reinforcement learning algorithms. We showed through empirical experiments that our method can outperform DQN and SQL with or without the rainbow framework.

References

Kavosh Asadi and Michael L Littman. An alternative softmax operator for reinforcement learning. In *International Conference on Machine Learning*, pp. 243–252. PMLR, 2017.

Marc Bellemare, Joel Veness, and Erik Talvitie. Skip context tree switching. In *International conference on machine learning*, pp. 1458–1466. PMLR, 2014.

Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47: 253–279, 2013.

Marc G. Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Rémi Munos. Unifying count-based exploration and intrinsic motivation. *CoRR*, abs/1606.01868, 2016. URL `http://arxiv.org/abs/1606.01868`.

Marc G. Bellemare, Will Dabney, and Rémi Munos. A distributional perspective on reinforcement learning. *CoRR*, abs/1707.06887, 2017. URL `http://arxiv.org/abs/1707.06887`.

Petros Christodoulou. Soft actor-critic for discrete action settings. *arXiv preprint arXiv:1910.07207*, 2019.

Benjamin Eysenbach and Sergey Levine. Maximum entropy rl (provably) solves some robust rl problems. *arXiv preprint arXiv:2103.06257*, 2021.

Meire Fortunato, Mohammad Gheshlaghi Azar, Bilal Piot, Jacob Menick, Ian Osband, Alex Graves, Vlad Mnih, Remi Munos, Demis Hassabis, Olivier Pietquin, et al. Noisy networks for exploration. *arXiv preprint arXiv:1706.10295*, 2017.

Roy Fox. Toward provably unbiased temporal-difference value estimation. In *Optimization Foundations for Reinforcement Learning Workshop at NeurIPS*, 2019.

Roy Fox, Ari Pakman, and Naftali Tishby. Taming the noise in reinforcement learning via soft updates. *arXiv preprint arXiv:1512.08562*, 2016.

Jordi Grau-Moya, Felix Leibfried, and Peter Vrancx. Soft q-learning with mutual-information regularization. In *International conference on learning representations*, 2018.

Tuomas Haarnoja, Haoran Tang, Pieter Abbeel, and Sergey Levine. Reinforcement learning with deep energy-based policies. In *International Conference on Machine Learning*, pp. 1352–1361. PMLR, 2017.

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.

Hado Hasselt. Double q-learning. *Advances in neural information processing systems*, 23:2613–2621, 2010.

Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In *Thirty-second AAAI conference on artificial intelligence*, 2018.

Edwin T Jaynes. *Probability theory: The logic of science*. Cambridge university press, 2003.

Michael J Kearns and Satinder P Singh. Bias-variance error bounds for temporal difference updates. In *COLT*, pp. 142–147. Citeseer, 2000.

Seungchan Kim, Kavosh Asadi, Michael Littman, and George Konidaris. Deepmellow: removing the need for a target network in deep q-learning. In *Proceedings of the Twenty Eighth International Joint Conference on Artificial Intelligence*, 2019.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

Ofir Nachum, Mohammad Norouzi, Kelvin Xu, and Dale Schuurmans. Bridging the gap between value and policy based reinforcement learning. *arXiv preprint arXiv:1702.08892*, 2017a.

Ofir Nachum, Mohammad Norouzi, Kelvin Xu, and Dale Schuurmans. Trust-pcl: An off-policy trust region method for continuous control. *arXiv preprint arXiv:1707.01891*, 2017b.

Georg Ostrovski, Marc G Bellemare, Aäron Oord, and Rémi Munos. Count-based exploration with neural density models. In *International conference on machine learning*, pp. 2721–2730. PMLR, 2017.

Jan Peters, Katharina Mulling, and Yasemin Altun. Relative entropy policy search. In *Twenty-Fourth AAAI Conference on Artificial Intelligence*, 2010.

Konrad Rawlik, Marc Toussaint, and Sethu Vijayakumar. Approximate inference and stochastic optimal control. *arXiv preprint arXiv:1009.3958*, 2010.

Jonathan Rubin, Ohad Shamir, and Naftali Tishby. Trading value and information in mdps. In *Decision Making with Imperfect Decision Makers*, pp. 57–74. Springer, 2012.

Ahmad EL Sallab, Mohammed Abdou, Etienne Perot, and Senthil Yogamani. Deep reinforcement learning framework for autonomous driving. *Electronic Imaging*, 2017(19):70–76, 2017.

Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

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

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

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

Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Hasselt, Marc Lanctot, and Nando Freitas. Dueling network architectures for deep reinforcement learning. In *International conference on machine learning*, pp. 1995–2003. PMLR, 2016.

A HYPERPARAMETERS

Hyper-parameter	value
Discount factor γ	0.99
Exploration ϵ	0.01
Learning rate	1

Table 3: Hyper-parameters for tabular experiments.

Hyper-parameter	value
Grey-scaling	True
Observation down-sampling	(84, 84)
Frames stacked	4
Reward clipping	[-1, 1]
Discount factor γ	0.99
Initial exploration	1
Final exploration	0.1
Final exploration frame	1000000
Learning rate	0.00025
Replay buffer size	1000000
Minibatch size	32
Q network channels	[32, 64, 64]
Q network filter size	$8 \times 8, 4 \times 4, 3 \times 3$
Q network stride	4, 2, 1
Q network hidden units	512

Table 4: Hyper-parameters for DQN, SQL and CBSQL on Atari 2600. The values of all the hyper-parameters are based on the work from Mnih et al. (2015).

Hyper-parameter	value
Grey-scaling	True
Observation down-sampling	(84, 84)
Frames stacked	4
Reward clipping	[-1, 1]
Discount factor γ	0.99
Learning rate	0.0000625
Replay buffer size	1000000
Minibatch size	32
Q network channels	[32, 32, 64]
Q network filter size	$8 \times 8, 4 \times 4, 3 \times 3$
Q network stride	4, 2, 1
Q network hidden units	512
Noisy net σ_0	0.5
Multi-step returns n	4 for DQN, 1 for SQL and CBSQL
Distributional atoms	51
Distributional min/max values	[-10, 10]

Table 5: Hyper-parameters for DQN, SQL and CBSQL with Rainbow Integration on Atari 2600. The values of all the hyper-parameters are based on the work from Hessel et al. (2018).