Appendix

A. SUNRISE: Soft actor-critic

Background. SAC (Haarnoja et al., 2018) is a state-of-the-art off-policy algorithm for continuous control problems. SAC learns a policy, $\pi_\phi(a|s)$, and a critic, $Q_\theta(s,a)$, and aims to maximize a weighted objective of the reward and the policy entropy, $\mathbb{E}_{s_t,a_t\sim\pi}\left[\sum_t \gamma^{t-1} r_t + \alpha\mathcal{H}(\pi_\phi(\cdot|s_t))\right]$. To update the parameters, SAC alternates between a soft policy evaluation and a soft policy improvement. At the soft policy evaluation step, a soft Q-function, which is modeled as a neural network with parameters θ, is updated by minimizing the following soft Bellman residual:

$$\mathcal{L}^{\texttt{SAC}}_{\texttt{critic}}(\theta) = \mathbb{E}_{\tau_t\sim\mathcal{B}}[\mathcal{L}_Q(\tau_t,\theta)],$$
$$\mathcal{L}_Q(\tau_t,\theta) = \left(Q_\theta(s_t,a_t) - r_t - \gamma\mathbb{E}_{a_{t+1}\sim\pi_\phi}\left[Q_{\bar{\theta}}(s_{t+1},a_{t+1}) - \alpha\log\pi_\phi(a_{t+1}|s_{t+1})\right]\right)^2,$$

where $\tau_t = (s_t, a_t, r_t, s_{t+1})$ is a transition, $\mathcal{B}$ is a replay buffer, $\bar{\theta}$ are the delayed parameters, and α is a temperature parameter. At the soft policy improvement step, the policy π with its parameter ϕ is updated by minimizing the following objective:

$$\mathcal{L}^{\texttt{SAC}}_{\texttt{actor}}(\phi) = \mathbb{E}_{s_t\sim\mathcal{B}}\left[\mathcal{L}_\pi(s_t,\phi)\right], \text{ where } \mathcal{L}_\pi(s_t,\phi) = \mathbb{E}_{a_t\sim\pi_\phi}\left[\alpha\log\pi_\phi(a_t|s_t) - Q_\theta(s_t,a_t)\right].$$

We remark that this corresponds to minimizing the Kullback-Leibler divergence between the policy and a Boltzmann distribution induced by the current soft Q-function.

SUNRISE without UCB exploration. For SUNRISE without UCB exploration, we use random inference proposed in Bootstrapped DQN (Osband et al., 2016a), which randomly selects an index of policy uniformly at random and generates the action from the selected actor for the duration of that episode (see Line 3 in Algorithm 2).

Algorithm 2 SUNRISE: SAC version (random inference)

1: **for** each iteration **do**
2: // RANDOM INFERENCE
3: Select an index of policy using $\widehat{i} \sim$ Uniform$\{1, \cdots, N\}$
4: **for** each timestep t **do**
5: Get the action from selected policy: $a_t \sim \pi_{\phi_{\widehat{i}}}(a|s_t)$
6: Collect state s_{t+1} and reward r_t from the environment by taking action a_t
7: Sample bootstrap masks $M_t = \{m_{t,i} \sim \text{Bernoulli}\,(\beta) \text{ — } i \in \{1,\ldots,N\}\}$
8: Store transitions $\tau_t = (s_t, a_t, s_{t+1}, r_t)$ and masks in replay buffer $\mathcal{B} \leftarrow \mathcal{B} \cup \{(\tau_t, M_t)\}$
9: **end for**
10: // UPDATE AGENTS VIA BOOTSTRAP AND WEIGHTED BELLMAN BACKUP
11: **for** each gradient step **do**
12: Sample random minibatch $\{(\tau_j, M_j)\}_{j=1}^B \sim \mathcal{B}$
13: **for** each agent i **do**
14: Update the Q-function by minimizing $\frac{1}{B}\sum_{j=1}^B m_{j,i}\mathcal{L}_{WQ}\left(\tau_j,\theta_i\right)$
15: Update the policy by minimizing $\frac{1}{B}\sum_{j=1}^B m_{j,i}\mathcal{L}_\pi(s_j,\phi_i)$
16: **end for**
17: **end for**
18: **end for**

B. Extension to Rainbow DQN

B.1. Preliminaries: Rainbow DQN

Background. DQN algorithm (Mnih et al., 2015) learns a Q-function, which is modeled as a neural network with parameters θ, by minimizing the following Bellman residual:

$$\mathcal{L}^{\texttt{DQN}}(\theta) = \mathbb{E}_{\tau_t\sim\mathcal{B}}\left[\left(Q_\theta(s_t,a_t) - r_t - \gamma\max_a Q_{\bar{\theta}}(s_{t+1},a)\right)^2\right], \tag{8}$$

where $\tau_t = (s_t, a_t, r_t, s_{t+1})$ is a transition, $\mathcal{B}$ is a replay buffer, and $\bar{\theta}$ are the delayed parameters. Even though Rainbow DQN integrates several techniques, such as double Q-learning (Van Hasselt et al., 2016) and distributional DQN (Bellemare et al., 2017), applying SUNRISE to Rainbow DQN can be described based on the standard DQN algorithm. For exposition, we refer the reader to Hessel et al. (2018) for more detailed explanations of Rainbow DQN.

Algorithm 3 SUNRISE: Rainbow version

for each iteration **do**
  for each timestep t **do**
    // UCB EXPLORATION
    Choose the action that maximizes UCB: $a_t = \arg\max_{a_{t,i} \in \mathcal{A}} Q_{\texttt{mean}}(s_t, a_{t,i}) + \lambda Q_{\texttt{std}}(s_t, a_{t,i})$
    Collect state s_{t+1} and reward r_t from the environment by taking action a_t
    Sample bootstrap masks $M_t = \{m_{t,i} \sim \text{Bernoulli}\,(\beta) \text{ — } i \in \{1, \ldots, N\}\}$
    Store transitions $\tau_t = (s_t, a_t, s_{t+1}, r_t)$ and masks in replay buffer $\mathcal{B} \leftarrow \mathcal{B} \cup \{(\tau_t, M_t)\}$
  end for
  // UPDATE Q-FUNCTIONS VIA BOOTSTRAP AND WEIGHTED BELLMAN BACKUP
  for each gradient step **do**
    Sample random minibatch $\{(\tau_j, M_j)\}_{j=1}^{B} \sim \mathcal{B}$
    for each agent i **do**
      Update the Q-function by minimizing $\frac{1}{B} \sum_{j=1}^{B} m_{j,i} \mathcal{L}_{WQ}^{\texttt{DQN}}(\tau_j, \theta_i)$
    end for
  end for
end for

B.2. SUNRISE: Rainbow DQN

Bootstrap with random initialization. Formally, we consider an ensemble of N Q-functions, i.e., $\{Q_{\theta_i}\}_{i=1}^{N}$, where θ_i denotes the parameters of the i-th Q-function.[3] To train the ensemble of Q-functions, we use the bootstrap with random initialization (Efron, 1982; Osband et al., 2016a), which enforces the diversity between Q-functions through two simple ideas: First, we initialize the model parameters of all Q-functions with random parameter values for inducing an initial diversity in the models. Second, we apply different samples to train each Q-function. Specifically, for each Q-function i in each timestep t, we draw the binary masks $m_{t,i}$ from the Bernoulli distribution with parameter $\beta \in (0, 1]$, and store them in the replay buffer. Then, when updating the model parameters of Q-functions, we multiply the bootstrap mask to each objective function.

Weighted Bellman backup. Since conventional Q-learning is based on the Bellman backup in (8), it can be affected by error propagation. I.e., error in the target Q-function $Q_{\bar{\theta}}(s_{t+1}, a_{t+1})$ gets propagated into the Q-function $Q_\theta(s_t, a_t)$ at the current state. Recently, Kumar et al. (2020) showed that this error propagation can cause inconsistency and unstable convergence. To mitigate this issue, for each Q-function i, we consider a weighted Bellman backup as follows:

$$\mathcal{L}_{WQ}^{\texttt{DQN}}(\tau_t, \theta_i) = w(s_{t+1}) \left(Q_{\theta_i}(s_t, a_t) - r_t - \gamma \max_a Q_{\bar{\theta}_i}(s_{t+1}, a) \right)^2,$$

where $\tau_t = (s_t, a_t, r_t, s_{t+1})$ is a transition, and $w(s)$ is a confidence weight based on ensemble of target Q-functions:

$$w(s) = \sigma\left(-\bar{Q}_{\texttt{std}}(s) * T\right) + 0.5, \tag{9}$$

where $T > 0$ is a temperature, σ is the sigmoid function, and $\bar{Q}_{\texttt{std}}(s)$ is the empirical standard deviation of all target Q-functions $\{\max_a Q_{\bar{\theta}_i}(s, a)\}_{i=1}^{N}$. Note that the confidence weight is bounded in $[0.5, 1.0]$ because standard deviation is always positive.[4] The proposed objective $\mathcal{L}_{WQ}^{\texttt{DQN}}$ down-weights the sample transitions with high variance across target Q-functions, resulting in a loss function for the Q-updates that has a better signal-to-noise ratio. Note that we combine the proposed weighted Bellman backup with prioritized replay (Schaul et al., 2016) by multiplying both weights to Bellman backups.

[3] Here, we remark that each Q-function has a unique target Q-function.

[4] We find that it is empirically stable to set minimum value of weight $w(s, a)$ as 0.5.

UCB exploration. The ensemble can also be leveraged for efficient exploration (Chen et al., 2017; Osband et al., 2016a) because it can express higher uncertainty on unseen samples. Motivated by this, by following the idea of Chen et al. (2017), we consider an optimism-based exploration that chooses the action that maximizes

$$a_t = \max_a \{Q_{\texttt{mean}}(s_t, a) + \lambda Q_{\texttt{std}}(s_t, a)\}, \tag{10}$$

where $Q_{\texttt{mean}}(s, a)$ and $Q_{\texttt{std}}(s, a)$ are the empirical mean and standard deviation of all Q-functions $\{Q_{\theta_i}\}_{i=1}^N$, and the $\lambda > 0$ is a hyperparameter. This inference method can encourage exploration by adding an exploration bonus (i.e., standard deviation $Q_{\texttt{std}}$) for visiting unseen state-action pairs similar to the UCB algorithm (Auer et al., 2002). This inference method was originally proposed in Chen et al. (2017) for efficient exploration in DQN, but we further extend it to Rainbow DQN. For evaluation, we approximate the maximum a posterior action by choosing the action maximizes the mean of Q-functions, i.e., $a_t = \max_a \{Q_{\texttt{mean}}(s_t, a)\}$. The full procedure is summarized in Algorithm 3.

C. Implementation details for toy regression tasks

We evaluate the quality of uncertainty estimates from an ensemble of neural networks on a toy regression task. To this end, we generate twenty training samples drawn as $y = x^3 + \epsilon$, where $\epsilon \sim \mathcal{N}(0, 3^2)$, and train ten ensembles of regression networks using bootstrap with random initialization. The regression network is as fully-connected neural networks with 2 hidden layers and 50 rectified linear units in each layer. For bootstrap, we draw the binary masks from the Bernoulli distribution with mean $\beta = 0.3$. As uncertainty estimates, we measure the empirical variance of the networks' predictions. As shown in Figure 1(b), the ensemble can produce well-calibrated uncertainty estimates (i.e., variance) on unseen samples.

D. Experimental setups and results: OpenAI Gym

Environments. We evaluate the performance of SUNRISE on four complex environments based on the standard benchmarking environments[5] from OpenAI Gym (Brockman et al., 2016). Note that we do not use a modified Cheetah environments from PETS (Chua et al., 2018) (dented as Cheetah in POPLIN (Wang & Ba, 2020)) because it includes additional information in observations.

Training details. We consider a combination of SAC and SUNRISE using the publicly released implementation repository (`https://github.com/vitchyr/rlkit`) without any modifications on hyperparameters and architectures. For our method, the temperature for weighted Bellman backups is chosen from $T \in \{10, 20, 50\}$, the mean of the Bernoulli distribution is chosen from $\beta \in \{0.5, 1.0\}$, the penalty parameter is chosen from $\lambda \in \{1, 5, 10\}$, and we train five ensemble agents. The optimal parameters are chosen to achieve the best performance on training environments. Here, we remark that training ensemble agents using same training samples but with different initialization (i.e., $\beta = 1$) usually achieves the best performance in most cases similar to Osband et al. (2016a) and Chen et al. (2017). We expect that this is because splitting samples can reduce the sample-efficiency. Also, initial diversity from random initialization can be enough because each Q-function has a unique target Q-function, i.e., target value is also different according to initialization.

Learning curves. Figure 4 shows the learning curves on all environments. One can note that SUNRISE consistently improves the performance of SAC by a large margin.

Effects of ensembles. Figure 5 shows the learning curves of SUNRISE with varying values of ensemble size on all environments. The performance can be improved by increasing the ensemble size, but the improvement is saturated around $N = 5$.

E. Experimental setups and results: noisy reward

DisCor. DisCor (Kumar et al., 2020) was proposed to prevent the error propagation issue in Q-learning. In addition to a standard Q-learning, DisCor trains an error model $\Delta_\psi(s, a)$, which approximates the cumulative sum of discounted Bellman errors over the past iterations of training. Then, using the error model, DisCor reweights the Bellman backups based on a confidence weight defined as follows:

$$w(s, a) \propto \exp\left(-\frac{\gamma \Delta_\psi(s, a)}{T}\right),$$

[5]We used the reference implementation at `https://github.com/WilsonWangTHU/mbbl` (Wang et al., 2019).

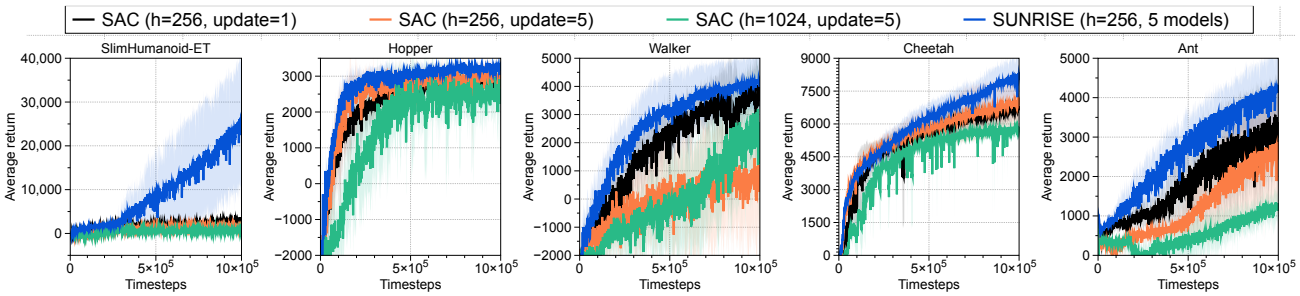


Figure 4. Learning curves of SUNRISE and single agent with h hidden units and five gradient updates per each timestep on OpenAI Gym. The solid line and shaded regions represent the mean and standard deviation, respectively, across four runs.

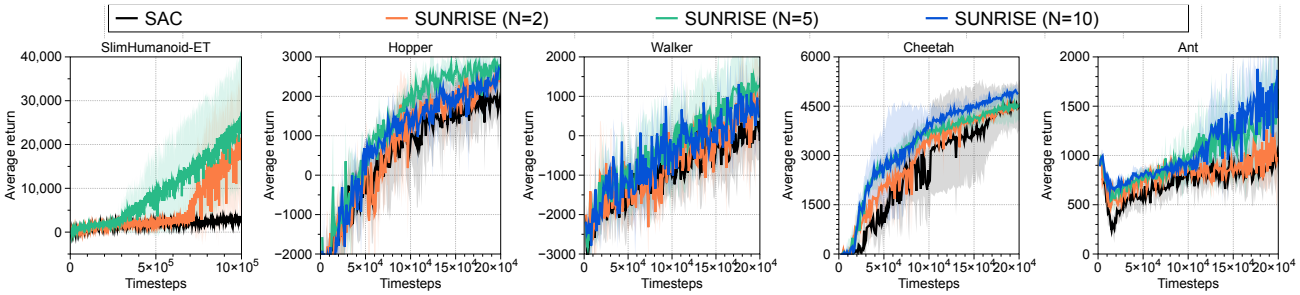


Figure 5. Learning curves of SUNRISE with varying values of ensemble size N. The solid line and shaded regions represent the mean and standard deviation, respectively, across four runs.

where γ is a discount factor and T is a temperature. By following the setups in Kumar et al. (2020), we take a network with 1 extra hidden layer than the corresponding Q-network as an error model, and chose $T = 10$ for all experiments. We update the temperature via a moving average and use the learning rate of 0.0003. We use the SAC algorithm as the RL objective coupled with DisCor and build on top of the publicly released implementation repository (`https://github.com/vitchyr/rlkit`).

Hyperparameter	**Value**	**Hyperparameter**	**Value**
Random crop	True	Initial temperature	0.1
Observation rendering	$(100, 100)$	Learning rate $(f_\theta, \pi_\psi, Q_\phi)$	$2e-4$ cheetah, run
Observation downsampling	$(84, 84)$		$1e-3$ otherwise
Replay buffer size	100000	Learning rate (α)	$1e-4$
Initial steps	1000	Batch Size	512 (cheetah), 256 (rest)
Stacked frames	3	Q function EMA τ	0.01
Action repeat	2 finger, spin; walker, walk	Critic target update freq	2
	8 cartpole, swingup	Convolutional layers	4
	4 otherwise	Number of filters	32
Hidden units (MLP)	1024	Non-linearity	ReLU
Evaluation episodes	10	Encoder EMA τ	0.05
Optimizer	Adam	Latent dimension	50
$(\beta_1, \beta_2) \rightarrow (f_\theta, \pi_\psi, Q_\phi)$	$(.9, .999)$	Discount γ	.99
$(\beta_1, \beta_2) \rightarrow (\alpha)$	$(.5, .999)$		

Table 4. Hyperparameters used for DeepMind Control Suite experiments. Most hyperparameters values are unchanged across environments with the exception for action repeat, learning rate, and batch size.

F. Experimental setups and results: DeepMind Control Suite

Training details. We consider a combination of RAD and SUNRISE using the publicly released implementation repository (https://github.com/MishaLaskin/rad) with a full list of hyperparameters in Table 4. Similar to Laskin et al. (2020), we use the same encoder architecture as in (Yarats et al., 2019), and the actor and critic share the same encoder to embed image observations.[6] For our method, the temperature for weighted Bellman backups is chosen from $T \in \{10, 100\}$, the mean of the Bernoulli distribution is chosen from $\beta \in \{0.5, 1.0\}$, the penalty parameter is chosen from $\lambda \in \{1, 5, 10\}$, and we train five ensemble agents. The optimal parameters are chosen to achieve the best performance on training environments. Here, we remark that training ensemble agents using same training samples but with different initialization (i.e., $\beta = 1$) usually achieves the best performance in most cases similar to Osband et al. (2016a) and Chen et al. (2017). We expect that this is because training samples can reduce the sample-efficiency. Also, initial diversity from random initialization can be enough because each Q-function has a unique target Q-function, i.e., target value is also different according to initialization.

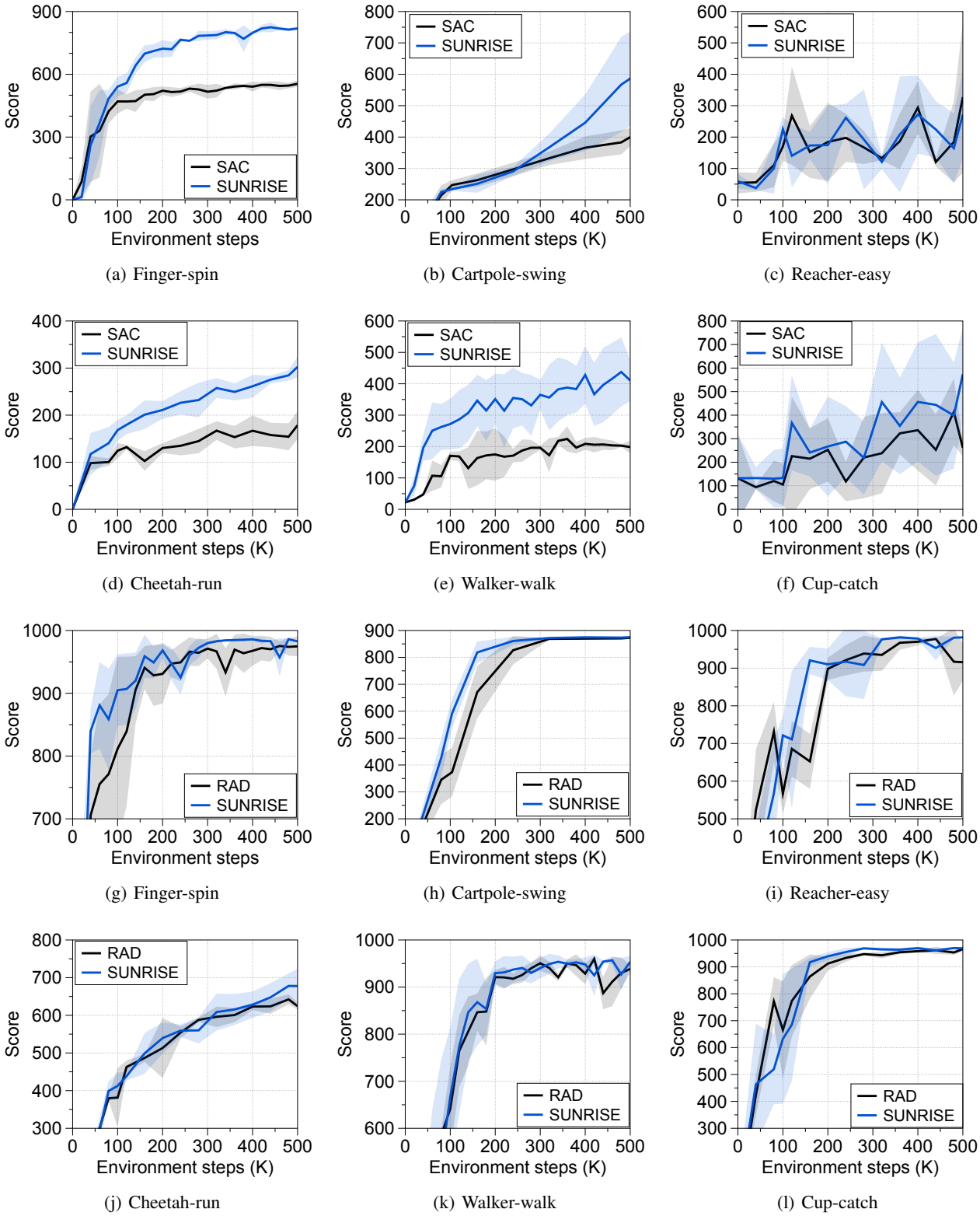
Learning curves. Figure 6(g), 6(h), 6(i), 6(j), 6(k), and 6(l) show the learning curves on all environments. Since RAD already achieves the near optimal performances and the room for improvement is small, we can see a small but consistent gains from SUNRISE. To verify the effectiveness of SUNRISE more clearly, we consider a combination of SAC and SUNRISE in Figure 6(a), 6(b), 6(c), 6(d), 6(e), and 6(f), where the gain from SUNRISE is more significant.

G. Experimental setups and results: Atari games

Training details. We consider a combination of sample-efficient versions of Rainbow DQN and SUNRISE using the publicly released implementation repository (https://github.com/Kaixhin/Rainbow) without any modifications on hyperparameters and architectures. For our method, the temperature for weighted Bellman backups is chosen from $T \in \{10, 40\}$, the mean of the Bernoulli distribution is chosen from $\beta \in \{0.5, 1.0\}$, the penalty parameter is chosen from $\lambda \in \{1, 10\}$, and we train five ensemble agents. The optimal parameters are chosen to achieve the best performance on training environments. Here, we remark that training ensemble agents using same training samples but with different initialization (i.e., $\beta = 1$) usually achieves the best performance in most cases similar to Osband et al. (2016a) and Chen et al. (2017). We expect that this is because splitting samples can reduce the sample-efficiency. Also, initial diversity from random initialization can be enough because each Q-function has a unique target Q-function, i.e., target value is also different according to initialization.

Learning curves. Figure 7, Figure 8 and Figure 9 show the learning curves on all environments.

[6] However, we remark that each agent does not share the encoders unlike Bootstrapped DQN (Osband et al., 2016a).



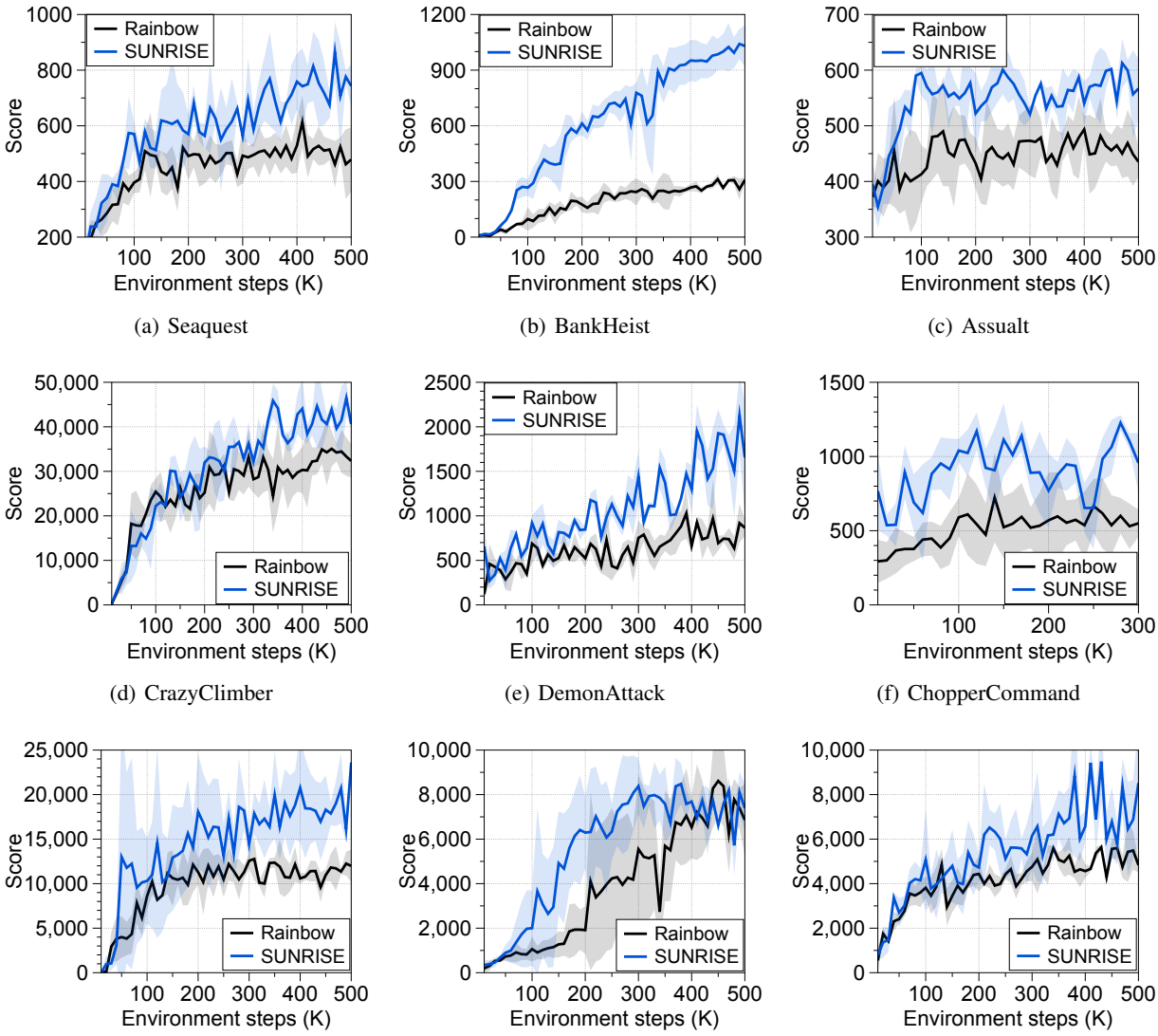
(a) Finger-spin (b) Cartpole-swing (c) Reacher-easy

(d) Cheetah-run (e) Walker-walk (f) Cup-catch

(g) Finger-spin (h) Cartpole-swing (i) Reacher-easy

(j) Cheetah-run (k) Walker-walk (l) Cup-catch

Figure 6. Learning curves of (a-f) SAC and (g-I) RAD on DeepMind Control Suite. The solid line and shaded regions represent the mean and standard deviation, respectively, across five runs.

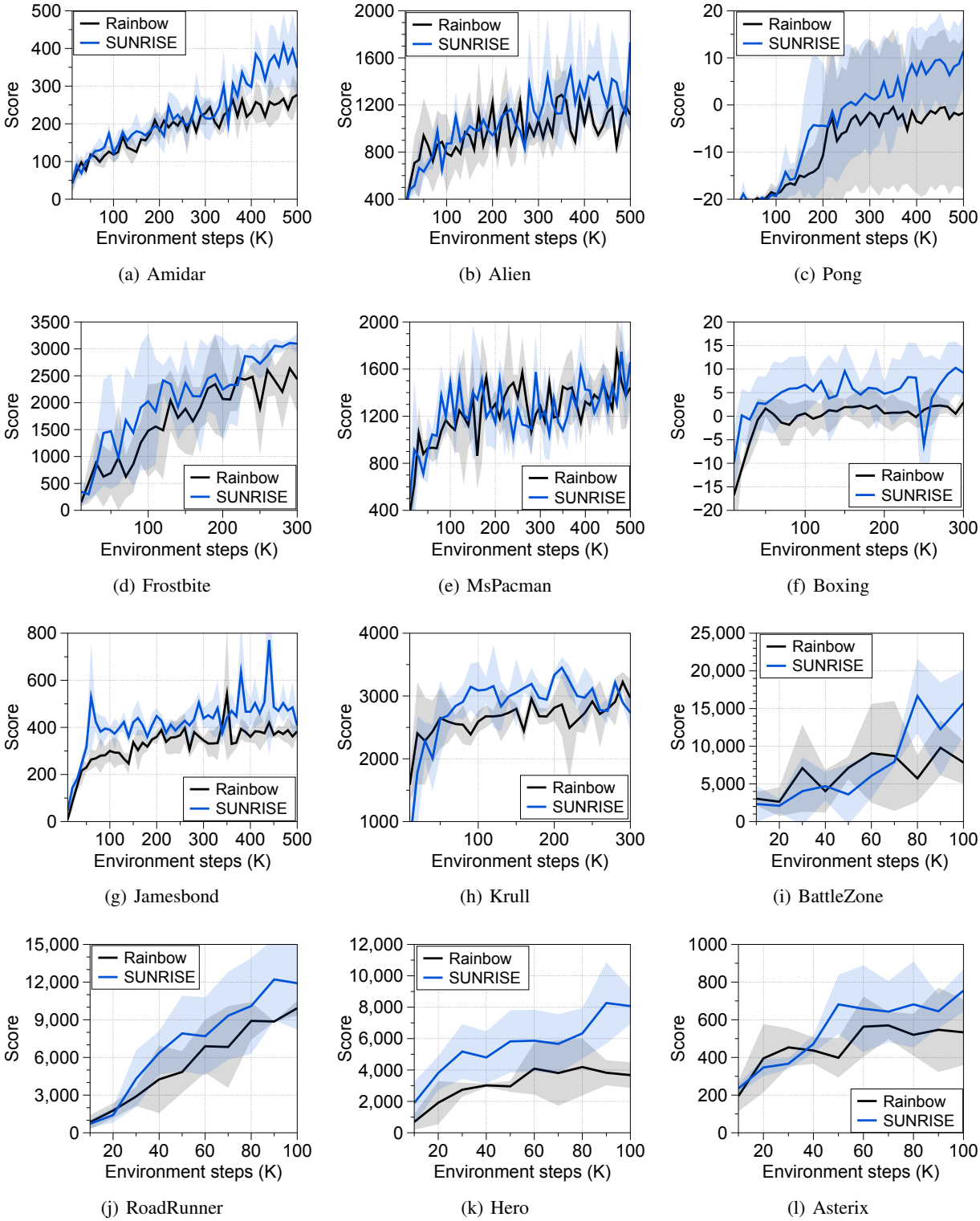


(a) Seaquest (b) BankHeist (c) Assualt

(d) CrazyClimber (e) DemonAttack (f) ChopperCommand

(g) KungFuMaster (h) Kangaroo (i) UpNDown

Figure 7. Learning curves on Atari games. The solid line and shaded regions represent the mean and standard deviation, respectively, across three runs.



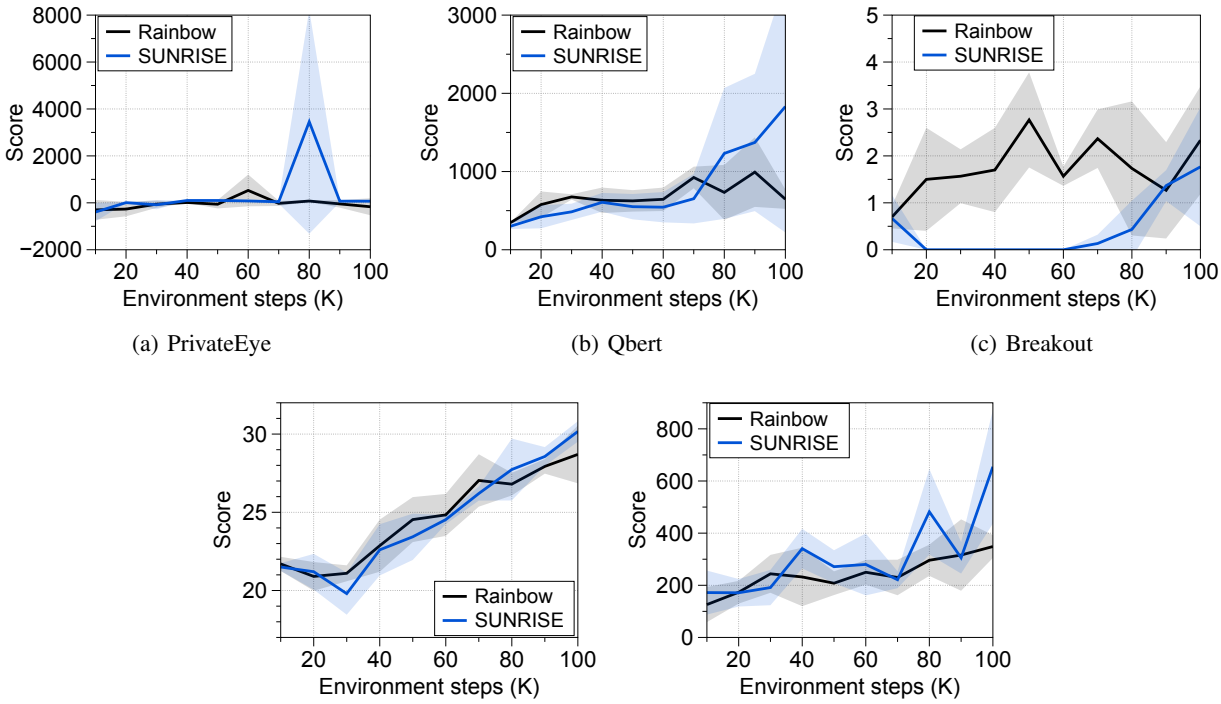
(a) Amidar (b) Alien (c) Pong

(d) Frostbite (e) MsPacman (f) Boxing

(g) Jamesbond (h) Krull (i) BattleZone

(j) RoadRunner (k) Hero (l) Asterix

Figure 8. Learning curves on Atari games. The solid line and shaded regions represent the mean and standard deviation, respectively, across three runs.



(a) PrivateEye (b) Qbert (c) Breakout

(d) Freeway (e) Gopher

Figure 9. Learning curves on Atari games. The solid line and shaded regions represent the mean and standard deviation, respectively, across three runs.