7th ICML Workshop on Automated Machine Learning (2020)

Meta-SAC: Auto-tune the Entropy Temperature of Soft Actor-Critic via Metagradient

Yufei Wang[1*], **Tianwei Ni**[2*]
Carnegie Mellon University

Abstract

Exploration-exploitation dilemma has long been a crucial issue in reinforcement learning. In this paper, we propose a new approach to automatically balance between these two. Our method is built upon the Soft Actor-Critic (SAC) (Haarnoja et al., 2018a) algorithm, which uses an "entropy temperature" that balances the original task reward and the policy entropy, and hence controls the trade-off between exploitation and exploration. It is empirically shown that SAC is very sensitive to this *hyperparameter*, and the follow-up work (SAC-v2) (Haarnoja et al., 2018b), which uses constrained optimization for automatic adjustment, has some limitations. The core of our method, namely Meta-SAC, is to use metagradient along with a novel meta objective to automatically tune the entropy temperature in SAC. We show that Meta-SAC achieves promising performances on several of the Mujoco benchmarking tasks, and outperforms SAC-v2 over 10% in one of the most challenging tasks, `humanoid-v2`.

1. Introduction

Reinforcement learning algorithms need to find a good balance between exploration and exploitation. On the one hand, the agent needs to explore the environment to gather useful information. On the other hand, it needs to exploit the knowledge it already obtains to improve its policy. There have been numerous works that aimed to tackle the exploitation/exploration trade-off issue. One category of methods use intrinsic rewards / count-based bonuses to enhance the exploration (Ostrovski et al., 2017; Pathak et al., 2017). Another family of methods use an entropy term to guide the agent to explore and exploit in a balanced way (Mnih et al., 2016; Haarnoja et al., 2018a). Among those methods, Soft Actor-Critic (SAC) (Haarnoja et al., 2018a) achieves the state-of-the-art performance on Mujoco (Todorov et al., 2012), a set of RL benchmarking continuous control tasks, and also performs well on lots of other robotics tasks.

SAC augments the traditional RL objective (Sutton and Barto, 2018) with a policy entropy term:

$$J(\pi) := \sum_{t=0}^{T} \gamma^t \mathbb{E}_{\mathbf{s}_t, \mathbf{a}_t \sim \rho_\pi} \left[r(\mathbf{s}_t, \mathbf{a}_t) + \alpha \mathcal{H}(\pi(\cdot|\mathbf{s}_t)) \right] \tag{1}$$

This is also called the maximum entropy RL objective (Ziebart et al., 2008; Levine, 2018). It also defines a *soft* version of value functions and bellman operators, where a new policy update rule is derived. The entropy temperature α plays a key role in the maximum entropy

. [1] yufeiw2@andrew.cmu.edu, [2] tianwein@cs.cmu.edu, [*] indicates equal contribution.

RL objective as it is the *hyperparameter* that balances the expected original task reward and the expected policy entropy, thus balancing the exploitation versus exploration.

SAC (Haarnoja et al., 2018a) (referred to as SAC-v1 throughout) is known to be particularly sensitive to the entropy temperature. For large temperature, the policy is encouraged to become nearly uniform and thus fails to exploit the reward signal, which substantially degrading the performance; for small temperature, though the policy learns quickly at first, it then becomes nearly deterministic and gets stuck at poor local minimal due to lack of exploration. However, it is non-trivial to choose a proper value of the entropy temperature. The optimal value not only changes across different tasks, but also varies in the learning process as the policy improves. In SAC-v1, this problem is solved by treating α as a hyperparameter and determining its value by grid search. This brings significant computational costs and manual efforts, and needs to be done for each new task.

Recently, gradient-based methods have been developed for hyperparameter optimization for deep neural networks (Hutter et al., 2019; Feurer and Hutter, 2019). In a follow-up work of SAC-v1 (Haarnoja et al., 2018b) (referred to as SAC-v2 throughout), the authors derive a formula to adjust the value of α automatically in the learning process by adding an entropy constraint upon the original RL objective, turning it into a constrained optimization problem, and using dual gradient descent to solve it. However, the derived update rule has several drawbacks, among which the most severe one is that it introduced another hyperparameter "target entropy", which by itself needs to be tuned for each task. The authors give a heuristic formula for choosing this new hyperparameter, which performs empirically well on Mujoco tasks, however, it remains unknown whether it is the optimal choice for every task.

In this paper, we propose to leverage the metagradient method (Xu et al., 2018; Zheng et al., 2018) to automatically tune the value of α during the learning process. In contrast to constrained optimization in SAC-v2, metagradient method introduces no more adaptive hyperparameter.[1] It is also a different and principled way of adjusting α, where the value of α (thus the balance between exploration and exploitation) is optimized towards minimizing a *meta loss*. In the early metagradient works, the meta loss is mostly defined as the policy gradient loss after the current policy update. In this paper, we propose to use a different meta loss to be consistent with the evaluation metric (i.e., the classic RL objective). The new meta loss lies between the DDPG (Lillicrap et al., 2015) and SAC objective, and we empirically verify its effectiveness.

In summary, the main contributions of this paper are as follows: We propose and derive a new method to automatically tune the entropy temperature α in SAC, which is based on metagradient and a novel meta loss, and does not introduce any adaptive hyperparameters. We show that the proposed method achieves state-of-the-art performance on several of the Mujoco locomotion tasks (Todorov et al., 2012).

2. Preliminaries

2.1 Metagradient

Metagradient (Xu et al., 2018; Zheng et al., 2018) is a general method for adapting some of the hyperparameters in the learning algorithm online. We follow the notations in a recent

1. An adaptive hyperparameter is defined as the one that needs to be tuned for every task.

paper (Zahavy et al., 2020) to give a brief review on it. Let θ denotes all the learnable parameters of the algorithm, e.g., the weights for the policy and value network, and ζ denotes all the hyperparameters, e.g., the learning rate of the optimizer, the discount factor γ, and the entropy temperature α in SAC. Let η be a subset of the hyperparameters ζ that we want to adapt during the learning process. We call η the metaparameters. The update of the learnable parameters θ at step t is done by optimizing them w.r.t. a **learning loss** that depends both on θ and η:

$$\theta_{t+1}(\eta_t) \leftarrow \theta_t - \lambda_\theta \nabla_\theta L_{learn}(\theta_t, \eta_t) \tag{2}$$

The metagradient method adapts the value of η by optimizing them w.r.t. a **meta loss**:

$$\eta_{t+1} \leftarrow \eta_t - \lambda_\eta \nabla_\eta L_{meta}(\theta_{t+1}(\eta_t)) \tag{3}$$

where λ_θ and λ_η are the respective learning rates. Note that the learnable parameters θ and the metaparameter η are updated in an alternative fashion iteratively.

2.2 SAC

SAC (Haarnoja et al., 2018a) augments the standard RL objective with expected entropy of the policy by $J(\pi) := \sum_{t=0}^{T} \gamma^t \mathbb{E}_{\mathbf{s}_t, \mathbf{a}_t \sim \rho_\pi} [r(\mathbf{s}_t, \mathbf{a}_t) + \alpha \mathcal{H}(\pi(\cdot|\mathbf{s}_t))]$.

SAC is an off-policy algorithm, where it stores a collection of $\{(\mathbf{s}_t, \mathbf{a}_t, \mathbf{r}_t, \mathbf{s}_{t+1}\}_{i=1}^{N}$ transition tuples in a replay buffer $\mathcal{D}$. It uses a neural network with parameter ϕ for the policy as π_ϕ, and uses another neural network with paramter ω for the Q-value as Q_ω. For training ϕ and ω, it randomly samples a batch of transition tuples from the replay buffer, and performs stochastic gradient descent on minimizing the following loss objectives for ϕ and ω:

$$\begin{aligned} L_Q(\omega) &:= \mathbb{E}_{\mathbf{s}_t, \mathbf{a}_t \sim \mathcal{D}} \left[\frac{1}{2} (Q_\omega(\mathbf{s}_t, \mathbf{a}_t) - Q^{\text{tar}}(\mathbf{s}_t, \mathbf{a}_t))^2 \right] \\ \text{where } Q^{\text{tar}}(\mathbf{s}_t, \mathbf{a}_t) &:= r(\mathbf{s}_t, \mathbf{a}_t) + \gamma \mathbb{E}_{\substack{\mathbf{s}_{t+1} \sim p_e \\ \mathbf{a}_{t+1} \sim \pi_\phi}} [\hat{Q}(\mathbf{s}_{t+1}, \mathbf{a}_{t+1}) - \alpha \log(\pi_\phi(\mathbf{a}_{t+1}|\mathbf{s}_{t+1}))] \end{aligned} \tag{4}$$

$$L_\pi(\phi) := \mathbb{E}_{\mathbf{s}_t \sim \mathcal{D}, \mathbf{a}_t \sim \pi_\phi} [\alpha \log \pi_\phi(\mathbf{a}_t|\mathbf{s}_t) - Q_\omega(\mathbf{s}_t, \mathbf{a}_t)] \tag{5}$$

where $\hat{Q}$ is target Q function whose parameters are periodically copied from the learned Q_ω.

SAC-v2 (Haarnoja et al., 2018b) makes the first step towards automating the tuning of α online. It casts the policy entropy term into a constraint that requires the policy to have a minimal expected entropy:

$$\max_{\pi_{0:T}} \quad \mathbb{E}_{\mathbf{s}_t, \mathbf{a}_t \sim \rho_\pi} \left[\sum_{t=0}^{T} r(\mathbf{s}_t, \mathbf{a}_t) \right] \quad \text{s.t.} \quad \mathbb{E}_{\mathbf{s}_t, \mathbf{a}_t \sim \rho_\pi} [-\log(\pi_t(\mathbf{a}_t|\mathbf{s}_t))] \geq H \quad \forall t \tag{6}$$

α then becomes the dual variable in the dual problem. The optimal policy to this problem is time-varying. The authors derived an approximated update formula w.r.t. α using dual gradient descent on $L(\alpha) := \mathbb{E}_{\mathbf{s}_t \sim \mathcal{D}, \mathbf{a}_t \sim \rho_\pi} [-\alpha \log \pi(\mathbf{a}_t|\mathbf{s}_t) - \alpha H]$ and dropping the time dependencies. This update formula is empirically proved to work well on Mujoco tasks.

However, there are some issues with it. First, it has several key assumptions that is generally not true in real applications. For example, the derivation and convergence of the

rule requires convexity, which does not hold for neural networks. The time dependency for the optimal solution is dropped for approximation. Second, for updating α, this formula introduces another hyperparameter H, the minimal expected entropy. The authors give an empirical formula for the value of $H = -\dim(\mathbf{a})$, i.e. the negative dimension of the action space, but it remains unknown how this is derived and whether it would work for each task. Besides, it seems contradictory to introduce an extra adaptive hyperparameter in the way of trying to free the tuning of the original one.

In the next section, we will introduce our proposed method of using metagradient to automate the turning of α, which has minimal assumptions and introduces no more adaptive hyperparameters.

3. Meta-SAC

We now demonstrate how the learning loss and meta loss is instantiated when applying metagradient to adapt the entropy temperature α in SAC. Using the notations in the subsection 2.1, the learnable parameters are $\theta = \{\phi, \omega\}$, and the metaparameter is $\eta = \{\alpha\}$. Therefore, the learning loss for θ are $L_Q(\omega)$ and $L_\pi(\phi)$, and Eq. 2 becomes:

$$\begin{aligned} \phi_{t+1}(\alpha_t) &\leftarrow \phi_t - \lambda_\phi \nabla_\phi L_\pi(\phi_t, \alpha_t) \\ \omega_{t+1}(\alpha_t) &\leftarrow \omega_t - \lambda_\omega \nabla_\omega L_Q(\omega_t, \alpha_t) \end{aligned} \tag{7}$$

The choice of meta loss is critical for updating the metaparameter η. Most of the previous works (Zheng et al., 2018; Xu et al., 2018; Zahavy et al., 2020) use policy gradient loss as the meta loss. However, in our initial experiments, we find that policy gradient loss performs poorly. Instead, we propose to use the following meta loss:

$$L_{meta}(\alpha_t) := \mathbb{E}_{\mathbf{s}_0 \sim \mathcal{D}_0} \left[-Q_{\omega_t}(\mathbf{s}, \pi^{\text{det}}_{\phi_{t+1}(\alpha_t)}(\mathbf{s})) \right] \tag{8}$$

In this meta loss, $\pi^{\text{det}}_{\phi_{t+1}(\alpha_t)}(s)$ denotes the deterministic version of the updated policy (e.g., when the policy is parameterized as a Gaussian, the deterministic version always chooses the mean of the Gaussian), and $\mathcal{D}_0$ denotes a special replay buffer that stores the initial states of the environments.

The main intuition is to make the meta loss *consistent* with the **evaluation metric** $M(\pi)$, i.e. the standard RL objective that only considers the task reward, as this is the quantity of concern to us:

$$M(\pi) := \mathbb{E}_{\mathbf{s}_t, \mathbf{a}_t \sim \rho_\pi} \left[\sum_{t=0}^{T} \gamma^t r(\mathbf{s}_t, \mathbf{a}_t) \right] = \mathbb{E}_{\mathbf{s}_0 \sim p_e, \mathbf{a}_0 \sim \pi} \left[Q^\pi(\mathbf{s}_0, \mathbf{a}_0) \right] \tag{9}$$

where Q^π is the classic Q-value function for policy π without considering the entropy term, and the initial state $\mathbf{s}_0$ is sampled from the environment. We now elaborate on how the evaluation metric leads to the design of our meta loss:

1. The evaluation metric $M(\pi)$ does not consider the policy entropy, thus we drop the entropy term in the SAC loss (Eq. 5). Moreover, normally during evaluation we cast the stochastic policy into a deterministic one, and this leads to the design of using $\pi^{\text{det}}_{\phi_{t+1}(\alpha_t)}(\mathbf{s})$.

2. As shown in Eq. 9, the evaluation metric is expectation over initial state distribution. Therefore, we collect and store some initial states into a special buffer $\mathcal{D}_0$, and sample a batch of initial states $\mathbf{s}_0$ from $\mathcal{D}_0$ to train meta loss. This method performs much better than sampling from *arbitrary* states, as shown by our ablation studies in appendix D.1.
3. The evaluation metric is based on the classic Q-value that does not consider the entropy term. However, in our meta loss we still use the soft Q value as it helps exploration, and our experiments show that using the classic Q-value performs worse in appendix D.3. This is the key discrepancy between our meta loss and the DDPG loss (Lillicrap et al., 2015). Our meta loss actually lies between the DDPG loss (we use soft-Q instead of classic Q) and the SAC loss (we drop the entropy term).
4. We use the old soft Q function Q_{ω_t} instead of the updated one $Q_{\omega_{t+1}(\alpha_t)}$. The main reason is that taking derivative of $Q_{\omega_{t+1}(\alpha_t)}(\mathbf{s}, \pi^{\text{det}}_{\phi_{t+1}(\alpha_t)}(\mathbf{s}))$ w.r.t. α is very numerically unstable in our early experiments.

The full algorithm of Meta-SAC is provided in appendix A.

4. Experiments

We compare Meta-SAC with three state-of-the-art off-policy RL algorithms, including SAC-v1, SAC-v2, and TD3 (Fujimoto et al., 2018) which made several improvements upon DDPG (Lillicrap et al., 2015). The main competitor is SAC-v2, as it is the only existing method that automatically tunes the value of α.

Our experiments aim to answer the following questions: (1) How does Meta-SAC perform across a range of different environments? (2) How does the entropy temperature α evolve during the learning process? (3) How effective is the proposed new meta objective?[2]

Our source code is available online[3]. All the hyperparameters are listed in appendix C.

4.1 Mujoco Learning Curves

To answer the first question, we select four representative environments from the Mujoco benchmarking continuous control tasks (Todorov et al., 2012), namely `Ant-v2, Hopper-v2, Humanoid-v2, Walker2d-v2`. The environment details can be found in appendix B. Among them, `Humanoid-v2` is one of the most challenging tasks that can be solved by current RL algorithms (Salimans et al., 2017), with a very high-dimensional state space (376).

The first row of Figure 1 shows the average return of evaluation rollouts through training process for Meta-SAC (blue), SAC-v1 (red), SAC-v2 (black) and TD3 (green). We follow the evaluation standard of SAC-v2: we run five different instances of each algorithm with different random seeds, do 10 evaluation rollouts every 10000 training environment steps, and report the mean of the return of these 10 rollouts. As shown in the figure, Meta-SAC achieves comparable performance with SAC-v2 on the easy tasks (`Ant-v2,Hopper-v2,Walker2d-2`), and performs slightly worse than SAC-v1 which uses grid search for the value of α for each task. In the most difficult task `Humanoid-v2`, Meta-SAC performs significantly better than all the other methods. It not only achieves a final return that is 10% higher than the others,

2. Due to space limit, the third question is answered in appendix D as ablation studies.
3. `https://github.com/twni2016/Meta-SAC`

but also demonstrates faster convergence. These results show that Meta-SAC can be a strong alternative to SAC-v2, especially for complex tasks.

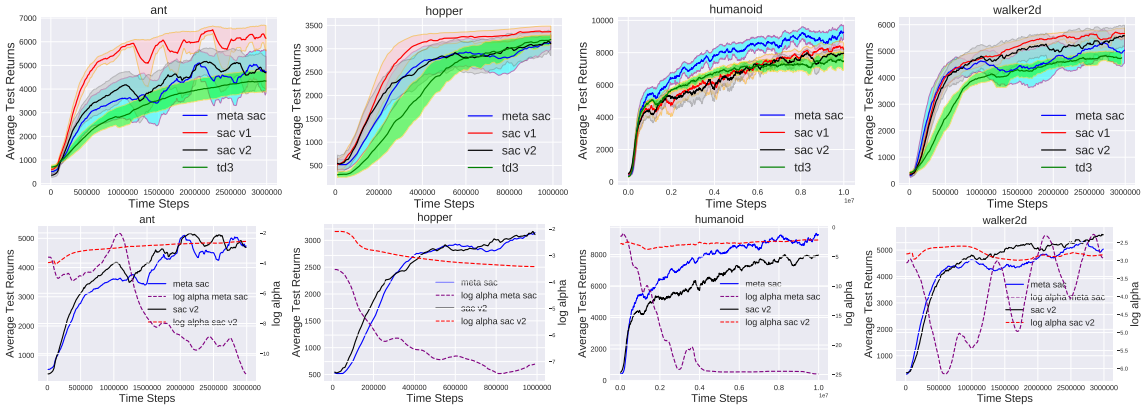


Figure 1: The first row shows the average test returns during the training process on the 4 Mujoco tasks of all compared algorithms. The curves show the mean and the shaded regions show half of the std over 5 random seeds. The curves are smoothed over the last 20 evaluations for better visualization. The second row shows the average test returns (solid) and the corresponding $\log \alpha$ curves (dashed) for the 4 tasks during the training process in SAC-v2 and Meta-SAC.

4.2 How Entropy Temperature Changes during Learning?

The second row of Figure 1 shows how differently $\log \alpha$ changes during the training process between SAC-v2 and Meta-SAC. In SAC-v2, the change of $\log \alpha$ is *mild*, which reaches a plateau after just a few learning steps. On the contrast, $\log \alpha$ changes dramatically in Meta-SAC along with a much larger scale. Generally in the later learning stages, α almost converges to zero, which means almost no entropy bonus is given and the SAC objective becomes almost *equivalent* to the DDPG objective. This might explain why Meta-SAC performs much better in the `Humanoid-v2`, as large entropy could indeed help exploration in the early stage of the training, but in the later stages when the policy has been trained well, a smaller exploration bonus would help the policy exploit better what it already learned. We verify this by decaying the temperature in SAC-v1 shown in appendix E. This phenomenon also matches how DQN-style method adjusts the value of ϵ in ϵ-greedy strategy, where it gradually decays to zero during the training.

5. Conclusion

In this paper, we present an auto-tuning method on the entropy temperature of the SAC algorithm using metagradient, with a novel meta loss aimed to be consistent with the evaluation metric. We verify its effectiveness in Mujoco benchmarking tasks, where it achieves state-of-the-art performance on one of the most difficult tasks, `Humanoid-v2`.

Acknowledgments

The authors thank Shicong Cen and Jiaqiang Ruan for the discussion in the early experiments. We also thank CMU 10-715 class lecturer Nihar B. Shah and the classmates for their suggestions in the early version. We thank the anonymous reviewers whose comments helped improve and clarify this paper.

References

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

Matthias Feurer and Frank Hutter. Hyperparameter optimization. In Hutter et al. (2019), pages 3–38.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1126–1135. JMLR. org, 2017.

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

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*, 2018a.

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*, 2018b.

Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. Neural networks for machine learning lecture 6a overview of mini-batch gradient descent.

Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren, editors. *Automatic Machine Learning: Methods, Systems, Challenges*. Springer, 2019.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Sergey Levine. Reinforcement learning and control as probabilistic inference: Tutorial and review. *arXiv preprint arXiv:1805.00909*, 2018.

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.

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937, 2016.

Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

Georg Ostrovski, Marc G Bellemare, Aäron van den Oord, and Rémi Munos. Count-based exploration with neural density models. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2721–2730. JMLR. org, 2017.

Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 16–17, 2017.

Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever. Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864*, 2017.

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.

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

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012.

Zhongwen Xu, Hado P van Hasselt, and David Silver. Meta-gradient reinforcement learning. In *Advances in neural information processing systems*, pages 2396–2407, 2018.

Tom Zahavy, Zhongwen Xu, Vivek Veeriah, Matteo Hessel, Junhyuk Oh, Hado van Hasselt, David Silver, and Satinder Singh. Self-tuning deep reinforcement learning. *arXiv preprint arXiv:2002.12928*, 2020.

Zeyu Zheng, Junhyuk Oh, and Satinder Singh. On learning intrinsic rewards for policy gradient methods. In *Advances in Neural Information Processing Systems*, pages 4644–4654, 2018.

Brian D Ziebart, Andrew L Maas, J Andrew Bagnell, and Anind K Dey. Maximum entropy inverse reinforcement learning. 2008.

Appendix A. Meta SAC Algorithm

Algorithm 1: Meta-SAC

Initialize Q network parameters ω_0, policy network parameters ϕ_0, and alpha α_0.
Empty replay buffer $\mathcal{D}$, learning rates $\lambda_\omega, \lambda_\phi, \lambda_\alpha$, batch size B, start step T_S
Collect a buffer $\mathcal{D}_0$ of initial states $\mathbf{s}_0$ from environment resets
for *each training timestep* t **do**
  $\mathbf{a}_t \sim \pi_\phi(\mathbf{a}_t|\mathbf{s}_t)$
  $\mathbf{s}_{t+1} \sim p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$
  $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathbf{s}_t, \mathbf{a}_t, r(\mathbf{s}_t, \mathbf{a}_t), \mathbf{s}_{t+1})\}$
  if $t > T_S$ **then**
    Sample a batch of transitions $\mathcal{B} = \{(\mathbf{s}, \mathbf{a}, r(\mathbf{s}, \mathbf{a}), \mathbf{s}')\}_{i=1}^{B}$ from $\mathcal{D}$.
    $\omega_{t+1} \leftarrow \omega_t - \lambda_\omega \hat{\nabla}_\omega L_Q(\omega, \alpha_t)$ using $\mathcal{B}$ and Eq. 4
    $\phi_{t+1} \leftarrow \phi_t - \lambda_\phi \hat{\nabla}_\phi L_\pi(\phi_t, \alpha_t)$ using $\mathcal{B}$ and Eq. 5
    $\alpha_{t+1} \leftarrow \alpha_t - \lambda_\alpha L_{meta}(\alpha_t)$ using $\mathcal{D}_0$ and Eq. 8
  end
end

In practice, we do resampling on a new minibatch $\mathcal{B}'$ from replay buffer $\mathcal{D}$ for updating Q value Q_ω and policy π_ϕ to avoid overfitting on the minibatch $\mathcal{B}$ used for metaparameter α updates. This trick is also applied in (Finn et al., 2017; Zheng et al., 2018) for online cross-validation. The ablation study on resampling is in appendix D.2.

Appendix B. Environment Details

We carry out experiments on the locomotion tasks of simulated robots in Mujoco environments (Todorov et al., 2012) wrapped in OpenAI Gym (Brockman et al., 2016). The states are the robots' generalized positions and velocities, and the actions are joint torques. The reward is defined to be proportional to the speed regularized with the acceleration in order to make locomotion more smooth. Early termination is added when the robot falls over, determined by thresholds on the height and torso angles. The locomotion tasks are challenging due to the high-dimensional state and action space, non-smooth dynamics, and underactuated systems (Schulman et al., 2015; Lillicrap et al., 2015).

Figure 2 shows the rendering of two of the locomotion tasks, `Ant-v2` and `Humanoid-v2` at one frame (figure sources[4]).

Table 1 shows the dimensionality on state ($\dim(\mathbf{s})$) and action ($\dim(\mathbf{a})$), along with a brief description of the tasks (Brockman et al., 2016) . The state space is unbounded, and the action space is in general bounded within $(-1, 1)$ in each dimension, except for `Humanoid-v2` whose action is bounded within $(-0.4, 0.4)$.

4. `https://gym.openai.com/envs/Ant-v2/` and `https://gym.openai.com/envs/Humanoid-v2/`

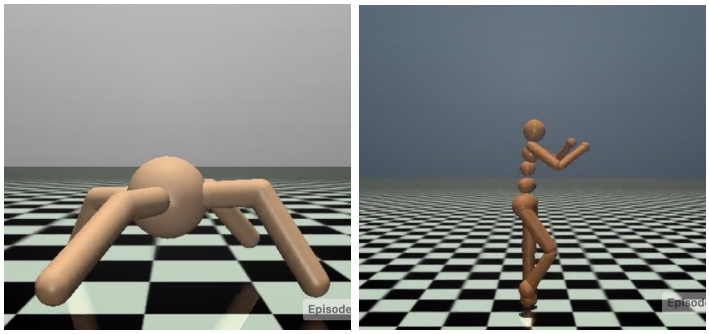


Figure 2: Rendering of simulated environments in `Ant-v2` (left) and `Humanoid-v2` (right).

Task name	dim(s)	dim(a)	Description
`Ant-v2`	111	8	Make a 3D four-legged robot walk forward as fast as possible.
`Hopper-v2`	11	3	Make a 2D one-legged robot hop forward as fast as possible.
`Humanoid-v2`	376	17	Make a 3D bipedal robot walk forward as fast as possible.
`Walker2d-v2`	17	6	Make a 2D bipedal robot walk forward as fast as possible.

Table 1: Brief information of the locomotion tasks.

Appendix C. Hyperparameters

C.1 SAC Hyperparameters

For SAC-v1 and SAC-v2, we directly use their original hyperparameters (refer to Section D in the appendix of SAC-v2 (Haarnoja et al., 2018b)). Specifically, we parameterize the policy network θ and the Q network ϕ both with a MLP of two hidden layers, with 256 neurons and the ReLU (Nair and Hinton, 2010) activation for each layer. The action output of the policy network is parameterized by a squashed Gaussian (Haarnoja et al., 2018a) with the same output range as the action space, where the mean and the diagonal covariance matrix are learnable parameters.

We use a batch size of 256 and the Adam (Kingma and Ba, 2014) optimizer with a learning rate of $3 \cdot 10^{-4}$. The discount factor γ is 0.99, the target smoothing coefficient τ is 0.05, and the replay buffer size is 10^6. The start step T_S is 10000 for each task.

Table 2 shows the hyperparameters that are different for each task in the baseline SAC algorithms.

	`Ant-v2`	`Hopper-v2`	`Humanoid-v2`	`Walker2d-v2`
temperature $\boldsymbol{\alpha}$ for SAC-v1	0.2	0.2	0.05	0.2
entropy target $\boldsymbol{H}$ for SAC-v2	-8	-3	-17	-6
training timesteps (in *Million*)	3	1	10	3

Table 2: Hyperparameters that are different for each task in SAC.

C.2 Meta-SAC Hyperparameters

The hyperparameters that are different from SAC to Meta-SAC are listed as follows:

- We change the policy optimizer to RMSProp (Hinton et al.) with $\epsilon = 10^{-12}$ for better performance, and keep the same learning rate in Meta-SAC. Note that SAC performs similarly under both Adam and RMSProp. The main reason that we use RMSProp is that our implementation requires backpropagating through the update process of the optimizer, and the update rule of RMSProp is more numerically stable during backpropagation compared with Adam.
- The metaparameter, the entropy temperature α is parameterized in the form of $\log \alpha$, and we clip its value to be below zero. This ensures $0 < \alpha \leq 1$.
- The learning rate for $\log \alpha$ is $3 \cdot 10^{-4}$, which is kept to be the same as the learning rate of the policy or the Q function. Its gradient norm is clipped below 0.05 to stabilize training.
- The size of replay buffer $\mathcal{D}_0$ is same as batch size 256.

It should be emphasized that though we introduce several extra hyperparameters, we keep all of them the same value across different tasks, whereas SAC-v1 and SAC-v2 have to choose different values of the entropy temperature α or the entropy target H in each task.

Appendix D. Ablation Studies on Meta Objective

In this section, we aim to answer the third question that we proposed in the main paper: How effective is the proposed new meta objective? We do the ablative analysis under the same experiment setting as Meta-SAC except for the meta objective.

D.1 Ablation Study on Using Initial States

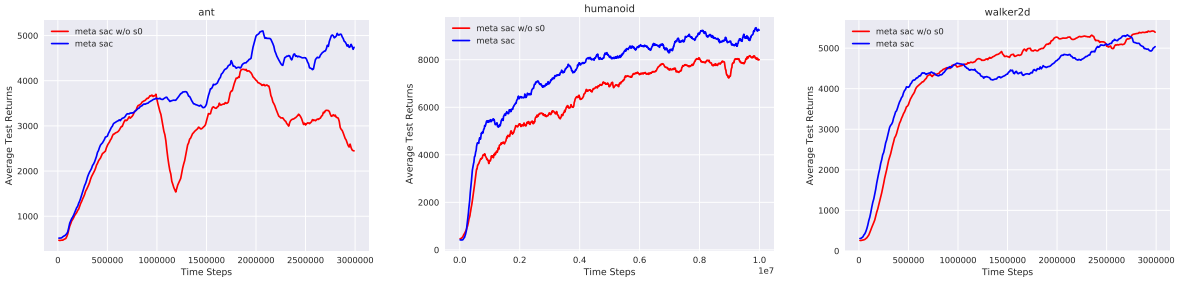
First we test the effectiveness of using the environment initial states instead of arbitrary states in the meta loss. The result is shown in the first row of Figure 3. As can be seen, using the environment initial states significantly boost the performance of Meta-SAC, especially in `Ant-v2` and `Humanoid-v2`.

D.2 Ablation Study on Resampling

Then we test how the Meta-SAC performs if we change the soft-Q value used in Eq. 8 to the classic Q-value. The result is shown in the second row of Figure 3. As can be seen, when using the classic Q-function in the meta-loss, the performances degrades obviously in `Ant-v2` and `Walker2d-v2`. We suspect the reason is that such a meta loss discourages exploration and thus hurts the performance.

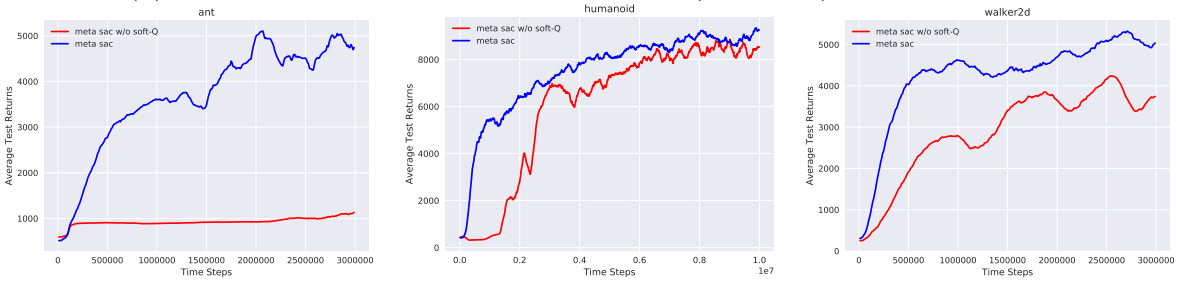
D.3 Ablation Study on Using Soft-Q Value

Finally, we test how the Meta-SAC performs if we change the soft-Q value used in Eq. 8 to the classic Q-value. The result is shown in the third row of Figure 3. As can be seen, when using the classic Q-function in the meta-loss, the performances degrades obviously in `Ant-v2` and `Walker2d-v2`. We suspect the reason is that such a meta loss discourages exploration and thus hurts the performance.



(a) Ablation study on using arbitrary states (red curve) in the meta loss.

(b) Ablation study on not using resampling (red curve) in the meta loss.



(c) Ablation study on using classic Q (red curve) in the meta loss.

Figure 3: Ablation studies on meta objective.

Appendix E. Ablation Study on Small Alpha in SAC-v1 on Humanoid

Figure 4 shows the result of SAC-v1 with different *fixed* alpha varying from $e^{-3} \approx 0.05$ to $e^{-7} \approx 0.001$ by grid search on log scale on `Humanoid-v2` task. We can see that simply decaying alpha will only worsen the final performance. Therefore, the early stage of meta-SAC where it maintains a high value of alpha to help exploration does contribute to its good performance.

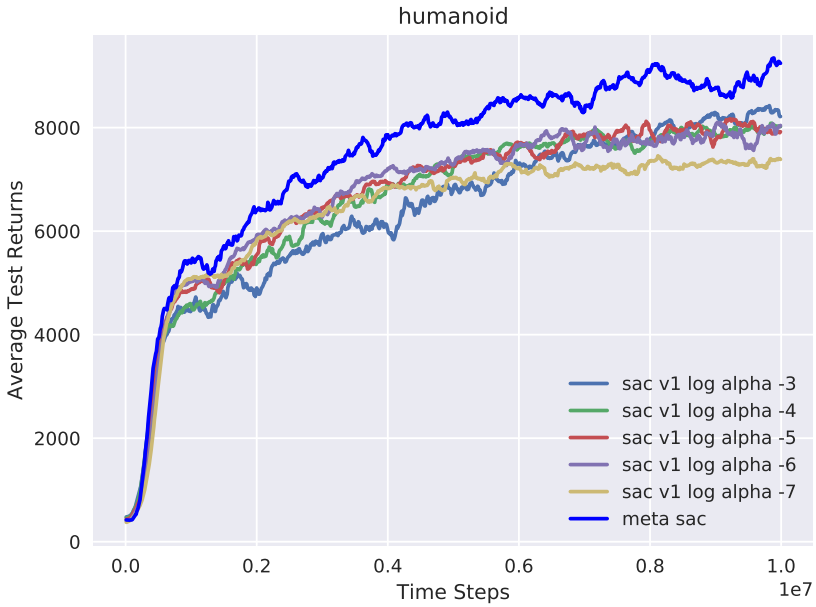


Figure 4: Ablation study on small alpha in SAC-v1 in Humanoid-v2.