Soft Actor-Critic: Deep Reinforcement Learning for Robotics

Finn Rietz

***Abstract*— Deep reinforcement learning has produced astonishing results in the past, however, most of these stem from purely virtual domains. Many challenges are faced when these methods are applied in real-world robotic scenarios, the most fundamental being the high sample-inefficiency, which is typically associated with deep reinforcement learning methods. Thus, this paper presents an extensive review of those challenges and explores the capabilities of the recent soft actor-critic algorithm, which indicates a new state of the art for end-to-end learning of policies. We deeply explore the algorithm first from a theoretical, then from a practical perspective and examine how the algorithm relates to the identified challenges in the joint field of robotics and deep reinforcement learning. Finally, we analyze the results attained by the algorithm and compare these to other state of the art methods.**

I. Introduction

Reinforcement learning holds the potential to let an agent autonomously learn optimal behavior for a specific task, without the need of a human engineer designing the desired behavior patterns. Naturally, this appears promising for the field of robotics, since the agent can be a robot, operating in a real-world setting. Significant advances in the joint field of robotics and reinforcement learning have been made with the recent breakthroughs in deep learning, which allow the learning of high-level features from raw sensory input [8]. These advances in the field of deep learning make the high-level goal of end-to-end learning of optimal policies, directly from high-dimensional, raw, sensory input more attainable, producing astonishing results: RL in combination with methods from deep learning, referred to as deep reinforcement learning, has shown astonishing results at various tasks, including superhuman performance on atari games [11], continuous control on simulated physics tasks [3], precise, dexterous manipulation through a robot hand [25] and general robotic control (partly end-to-end) [15, 9].

While deep reinforcement learning has proven strong capabilities in purely simulated or entirely virtual settings, a wide array of challenges arises on real-world robotic tasks. However, the recently introduced off-policy, model-free actor-critic deep RL algorithm *Soft Actor-Critic* (SAC) by Haarnoja et al. successfully tackles many of these problems and exhibits promising results for the deep RL with robotics domain [21, 20].

II. Basics

In reinforcement learning, the key idea is to have an agent autonomously learn desired behavior patterns, through an trial-and-error based interaction with an environment. More technically, the agent tries to accumulate as much *reward* R as possible and through this maximization learns which actions, in which state of the environment, produce the highest reward, thus learning behavior. The reward is a scalar value and received on every time step of the environment, however, often the agent receives zero reward and only a positive value on completion of the task. To formalize, in reinforcement learning scenarios, we always have a *state function* $s \in S$, modeling the observations the agent makes and containing all relevant information for the current state of the environment [7]. Further, reinforcement learning problems typically involve an *action function* $a \in A$, modeling the actions the agent can execute. Both functions can either be discrete or continuous, while not all algorithms are capable of dealing with both cases.

For reinforcement learning with robotics, the action function usually contains the motor torque to apply to each joint. The state function is highly dependent on the scenario, but for end-to-end reinforcement learning problems, the state function could be the output of the robot's vision system.

Formally speaking, the behavior the agent is to learn is denoted as the policy π and can be considered as a mapping function from states to actions. The policy can either be deterministic or probabilistic [7]. In the deterministic case, the same actions are always executed in the same state, so that $\pi(s) = a$. In probabilistic policies, action is for a state is sampled from a distribution, so that $a \sim \pi(s, a) = P(a|s)$

Further, we must introduce the notion of a *value function*. Value functions can express the total amount of reward the agent can expect, given it starts in state s and follows policy π, thus functioning as a goodness measure of the states for a specific policy [23]. Formally, the value functions for a state s is defined as

$$v_\pi(s) = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s \right], \text{ for all } s \in S \quad (1)$$

where γ functions as a discounting factor, putting more weight on reward that is expected in nearer future.

Similarly, the *action-value function* or *Q-function* measures the expected reward, given we are in state s, take action a and follow policy π thereafter [23]. Formally, the equation for the action-value function $q_\pi(s, a)$ is almost identical as for the value function $v_\pi(s)$, but we also

consider the selected action:

$$q_\pi(s,) = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s, A_t = a \right] \quad (2)$$

A. On-policy versus off-policy learning

In general, reinforcement learning algorithms can be categorized into *on-policy* and *off-policy* methods. In on-policy methods, we employ only one policy, meaning that the same policy that is used to generate exploratory behavior will be updated to become the optimal, final desired behavior policy [23]. On-policy methods, however, are known to be affected by the exploration versus exploitation dilemma [23]. To explore the dynamics of the environment and discover the optimal behavior policy, the algorithm must select actions in a non-greedy fashion, since we can never know whether our current policy is optimal, or whether there is a better one that can still be explored. We say a policy is exploring when a non-optimal action is selected (meaning a different action in the same space has higher expected reward). The policy is exploiting, when it greedily selects the action where the highest reward is expected. The difference between on- and off-policy methods is that off-policy methods maintain multiple policies, usually one for generating exploratory behavior and a second, target policy that will be shaped into the optimal, final policy [23]. Thus, off-policy methods do not face the exploration versus exploitation dilemma, since the non-optimal, exploratory part must not be integrated into the final policy, thus allowing the final policy to greedily maximize the expected reward. Off-policy methods have another advantage over on-policy methods: They make it possible to reuse old training data. In on-policy methods, training samples are generated for and by the very current policy only. Since off-policy learning allows us to update a policy other than the one that was used to generate the behavior, we can store past training experiences in a replay buffer, and reuse the already collected experience to update the target policy. This enables learning multiple times from just one experience [8]. This makes off-policy methods much more sample efficient than on-policy methods, which is a vital property for real-world robot reinforcement learning settings [24, 26, 20, 21, 8, 15, 22].

B. Model-free versus model-based methods

Similar to on-policy and off-policy methods, reinforcement learning methods can be categorized to be either *model-based* or *model-free*. In model-based algorithms, we employ a learned model of the environment, allowing the algorithm to predict how the environment will respond to the actions selected in any given state [23]. This model can store probabilities of observing follow-up states (and the reward associated with them), which allows for *planning*, where we consider future states without having observed them in the current interaction [23]. Thus, appropriate actions for a state are chosen by planning on the learned model of the environment, with the underlying idea that given enough sampled data, the learned model will be close enough to the real environment [5, 22]. Naturally, this is a statistically efficient way to reuse past experiences sampled from the environment, as each piece of experience is stored in the model of the environment and thus used many more times for predicting the best option to take [5]. However, while model-based methods can work well on simple tasks with low-dimensional state and actions spaced, they tend to fail on complex, continuous state and action space control tasks, due to *model bias* [22]. Model bias describes the problem of model-based methods wherein the policy optimization step, regions of the environment are exploited where insufficient data has been collected, leading to catastrophic failure [22]. As the name implies, in model-free reinforcement learning methods, such a model of the environment is not learned [23]. Thus, in model-free methods, planning for the future is not possible. Instead, we directly learn either the action-value function or a policy, which can produce the same optimal behavior as in planned, model-based methods [5]. Recall the value functions from Section II, which describe the value of a state (or state and action) for a given policy. These are the functions we learn in model-free reinforcement learning, for example in Q-learning [23, 8]. Model-free methods are less statistically efficient than model-based methods and even simple tasks can require millions of interaction steps sampled from the environment [21, 20, 22, 5]. That is because it can be very hard to learn complex tasks, based on a random trial and error search without a model of the environment. Think about traveling from your workplace to your home without a model of the environment (a map), by randomly turning at every possible location. To summarize: In practice, learning a model of the environment and planning actions would be the more data-efficient approach. However, especially in real-world robotic setting, this model is often simply to complex to learn (think about humans interacting with the environment as well, which could hardly be modeled for), which is why model-based methods are mostly used in purely simulated settings and in real-world tasks, we have to rely on model-free methods.

C. Value-based versus policy-based methods

A final feature that we can use to categorize or describe reinforcement learning methods is by whether they try to learn a value function or a parameterized policy directly. In *value-based* reinforcement learning methods, we try to learn the optimal *value-function* (the Q-function) and select actions according to the learned value of each actions in each state. An example of this would again be Q-learning [8, 23]. The policies used in value-based methods heavily rely on that learned value function (for example always taking the actions that produce the highest expected reward), to such an extent, that the

policies would not exist without the estimated action-value function. However, learning the value-function is not the only possible approach. We can also learn a parameterized policy directly, that can select actions without relying on an action-value function [23]. The central idea in *policy-based* methods is to learn preferences for taking an action in a state, that solely depend on the reward that has been obtained using that action, where a higher preference for an action means that the action will be taken more often [23]. Note, that a value function might still be used to learn the policy parameters [23]. Thus, the key idea for policy-based methods is that the parameters of the policy are enough to select which action to take in which state. Thus, policy-based methods also use a different notation for the policy: $\pi(a|s, \boldsymbol{\theta}) = \Pr\{A_t = a|S_t = s, \boldsymbol{\theta}_t = \boldsymbol{\theta}\}$. One advantage policy-based methods have over value-based methods is that the policy might simply be a much simpler function to approximate than the value function [23]. A disadvantage, however, is, that policy-based methods (at least policy gradient methods) are updated using the total reward accumulated during one interaction episode, thus only at the end of each transition. This is problematic when we might never reach a terminal state, due to very complex behavior.

D. Actor-critic architectures

In the previous section, we learned about value-based and policy-based reinforcement learning methods. Now, since the main algorithm in this paper is the Soft actor-critic algorithm by Harrnoja et al [20], we will introduce the actor-critic architecture here. As the name of the algorithm implies, Haarnoja et al. use a *soft* version of the actor-critic architecture, where soft relates to a stochastic actor, instead of deterministic one as in the deep deterministic policy gradient (DDPG) algorithm by Lilicrpa et al [10]. The actor-critic architecture provides the bases for most modern reinforcement learning algorithms and tries to combine value-based and policy-based methods, trying to combine the advantages of both. Thus, actor-critic methods learn a policy, dubbed the *actor* and a value-function, called the *critic* [23]. The critic learns to approximate the value function, that is, at each time step we observe a reward from the environment, and update the critic to mimic that behavior, which is exactly what the normal value function would do as well. Notice how this was not possible for purely policy-based methods, that could only be updated once the total reward for one interaction episode had been observed. The actor is progressively updated to maximize the approximated value function from the critic. This repetitive improvement of the critic (value function) and the actor function (policy) using that value function is a form of the generalized policy iteration (GPI) algorithm [23]. In practice, we often find neural networks as function approximators for both the actor and the critic. This is, however, not limited to actor-critic methods, but rather the general trend and idea behind deep reinforcement learning.

III. Related Work

Here, we review major challenges currently faced in the joint field of deep reinforcement learning and robotics. We prioritized this over a review of other state of the art algorithms for multiple reasons: First, few papers go into depth regarding the challenges in applying deep reinforcement learning in real-world robotics and only mention these briefly. Second, this review is of more value for the attendees of the class and third, Haarnoja et al. already provide a comparison of SAC to other state of the art methods, thus we redirect the interested reader to said comparison in the paper by Haarnoja et al. [20].

Deep reinforcement learning methods are infamous for being highly data inefficient. Even simple tasks can require millions of exploratory steps until a near-optimal policy is found. In simulated or purely virtual settings, this is might not be a big problem, but for real-world robotic applications, data generation in the real-world, on expensive robots, through random interaction, is dangerous [24]. If we were to apply classical reinforcement learning methods, which achieve promising results in virtual settings, to a robot, training would take impractical amounts of time, and due to random exploration, the robot would likely suffer critical damage in the process. Thus, the poor sample efficiency of deep reinforcement learning methods is often listed as one, if not the most severe limiting factor for applying these methods directly in real-world robotic scenarios [20, 21, 15, 25].

Reinforcement learning (specifically model-free reinforcement learning) is often described as a trial-and-error search for optimal behavior. That is, we sample from (interact with) the environment and receive some measure of how good the actions we selected in each situation were. Again, in a simulated setting, this is not problematic. However, when the sampling process corresponds to controlling the joints of an expensive robot, we can only randomly try actions for a certain amount of time until our robot is bound to take damage (or just breaks down from wear and tear). Thus, in a real-world robotic setting, we must ensure *safe exploration* and try to minimize training/exploration time. This makes minimal and safe exploration key aspects of applied deep reinforcement learning in robotic settings [15, 20, 7]. One option is to set the maximum allowed velocity for each joint of the robot to limit damage during training, to assure safe exploration, which has been done by Gu et al. [15].

It is apparent, that real-world robotic scenarios emit many properties which make reinforcement learning harder in the physical domain harder, compared to a purely virtual setting like atari games. Kober et al. list many challenging aspects of reinforcement learning in real-world robotics, including changing dynamics of the robot due to wear and tear or hidden external factors influencing sensory data, for example, light affecting the vision system and thus the state representation [7].

Since training the real-world is clearly problematic, a

valid approach would be to simply train reinforcement learning algorithms in a simulated model of the real-world environment, and transfer the final behavior policy on the robot. Theoretically, with perfect simulations, this would be possible. In practice, however, creating models that accurately capture all behavioral dynamics of the real-world environment is challenging and not always realistic. Small imperfections in the simulated model can cause the robot to learn a policy, which can not directly be applied to the robot, and requires further modification [7]. This mismatch between the simulated model and real-world environment is known as the *reality gap* or *sim-to-real* problem [17]. The sim-to-real problem poses a research field on its own, and significant advances have been made in recent years [17, 25]. The key point is that simulating accurate real-world scenarios is a non-trivial problem and policies learned in a simulation can, without additional measures, not be applied on a real-robot.

Another obstacle often encountered in robotic deep reinforcement learning is that of *sparse rewards* or *reward engineering* [7, 26, 14]. This describes the problem, that in classical reinforcement learning, the reward is often simply a binary measure. The environment emits *one reward* upon the successful completion of the task at hand. However, since many robot manipulation, or control, tasks are complex and require many steps before completion, it is likely that the agent never reaches the goal state by random exploration and thus never observe positive reward. In such a case, the agent never observes any variance in the reward function, which renders learning impossible [7]. This gives rise to yet another reinforcement learning subfield, dubbed reward engineering. However, manually engineering a specific reward function is a non-trivial task (requiring high-level domain knowledge) and counter-acts one of the reinforcement learning fundamental principals: Learning optimal behavior through a trial and error search [26]. This often causes learning to be entirely dependent on the manually-designed reward function, which often requires more complex state representations (e.g. additional sensors for the robot), which in turn makes learning the state space more challenging (curse of dimensionality) [7]. Further, reinforcement learning algorithms are notorious for exploiting the reward function in a non-anticipated manner, producing unintended behavior [7, 14]. This effect is amplified by manually designing a complex reward function, that tries to guide the agent through specific regions of the state space.

To conclude this section, robotic deep reinforcement learning for real-world interaction scenarios has many challenging properties, rendering this a hard, yet promising, topic. The challenges reported in this paper include high sample-inefficiency, causing impractical training times, the need for safety measures for the robot to prevent critical damage to the hardware and environment during exploration, external factors affecting the state, the reality gap making data collection in a simulation of the environment challenging and finally sparse rewards, describing the need for sophisticatedly designed reward functions.

IV. Soft Actor-Critic

In early 2018, Haarnoja et al. published the first version of the soft actor-critic algorithm [21]. However, in late 2018, Harrnoja et al. published a refined version of the algorithm in a second paper, improving upon the original version of the algorithm [20]. Both papers are similarly structured, and we adopt the structure for the description of the main algorithm, as the separation chosen by Haarnoja et al. appears very logical. in the first of the two following sections, we will provide a theoretical approach to SAC and in the second section, a more practical, hands-on approach to the algorithm is provided, as in both reference papers by Haarnoja et al. [21, 20].

A. SAC in theory

The soft actor-critic algorithm is described by Haarnoja et al. as an off-policy, model-free (actor-critic) algorithm, that is applicable to real-world robotic learning [20]. The algorithm is based on the maximum entropy reinforcement learning framework by Ziebart [6], which enables learning of a *stochastic* actor. Stochasticity of the actor is achieved by including an entropy regularization term into the classic reinforcement learning objective function. Thus, we no longer maximize the reward, but also the entropy associated with each state under a policy. This causes the actor policy to maximize the reward while acting as random as possible. This soft actor-critic, off-policy, model-free architecture has three main advantages over reinforcement learning methods that utilize the classical reinforcement objective without the entropy regularization term: Firstly, the actor is motivated to explore more widely, while giving up on unpromising areas in the state-function [20]. Secondly, the modified objective allows the policy to learn multiple, optimal behavior patterns, and thirdly, it considerably speeds up learning [20, 21, 19].

To formalize this idea, first consider the classical reinforcement learning objective $\sum_t \mathbb{E}_{(\mathbf{s}_t,\mathbf{a}_t)\sim\rho_\pi}[r(\mathbf{s}_t,\mathbf{a}_t)]$ (the sum of rewards), where we try to maximize that sum. In the scope of this paper, $\sim$ denotes that we sample from a distribution, thus, for the sum of rewards this simply means that we select actions based on the policy distribution for each state. Since SAC is based on the maximum entropy framework, the entropy of each visited state is maximized alongside the reward, producing the following objective for the optimal policy $\pi*$:

$$\pi^* = \underset{\pi}{\operatorname{argmax}} \sum_t \mathbb{E}_{(\mathbf{s}_t,\mathbf{a}_t)\sim\rho_\pi}[r(\mathbf{s}_t,\mathbf{a}_t) + \alpha\mathcal{H}(\pi(\cdot|\mathbf{s}_t))] \quad (3)$$

Here, $\mathcal{H}(P) = \mathbb{E}_{x\sim P} - logP(x)$ is the entropy $\mathcal{H}$ of the random variable x, having the probability density or mass function P and α is the temperature parameter,

Algorithm 1 Pratical soft actor-critic (adapted from [20])

Input: θ_1, θ_2, ϕ ▷ Initial parameters
  $\bar{\theta_1} \leftarrow \theta_1, \bar{\theta_2} \leftarrow \theta_2$ ▷ Initialize target network weights
  $D \leftarrow \emptyset$ ▷ Initialize empty replay buffer
  for each iteration **do**
    for each environment step **do**
      $\mathbf{a}_t \sim \pi_\phi(\mathbf{a}_t|\mathbf{s}_t)$ ▷ Sample action from the policy
      $\mathbf{s}_{t+1} \sim p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$ ▷ Sample transition from the environment
      $D \leftarrow D \cup \{(\mathbf{s}_t, \mathbf{a}_t, r(\mathbf{s}_t, \mathbf{a}_t), \mathbf{s}_{t+1})\}$ ▷ Store transition in the replay buffer
    end for
    for each gradient step **do**
      $\theta_i \leftarrow \theta_i - \lambda_Q \hat{\nabla}_{Q_i} J_Q(\theta_i)$ for $i \in \{1, 2\}$ ▷ Update the Q-function parameters
      $\phi \leftarrow \phi - \lambda_\pi \hat{\nabla}_\phi J_\pi(\phi)$ ▷ Update policy weights
      $\alpha \leftarrow \alpha - \lambda \hat{\nabla}_\alpha J(\alpha)$ ▷ Adjsut temperature
      $\bar{\theta_i} \leftarrow \tau\theta_i + (1-\tau)\bar{\theta_i}$ for $i \in \{1, 2\}$ ▷ Update target network weights by fraction τ
    end for
  end for
Output: θ_1, θ_2, ϕ ▷ Optimized parameters

weighting the entropy regularization term versus the reward and controlling the degree of stochasticity of the policy [19, 20, 21]. This is the older version of the objective function, as used in the initial version of the algorithm [21]. Here, α is treated as a hyperparameter, thus allowing us to only use learn policies with a fixed degree of stochasticity. However, Haarnoja et al. report that correctly adapting the α hyperparameter for every task was a non-trivial problem and that choosing non-ideal values for the hyperparameter could drastically decrease the performance of the SAC algorithm [20]. To address this problem, Haarnoja et al. found a way to automatically adjust the α hyperparameter during training, by treating the entropy term as a constraint. Thus, the new goal becomes to find a stochastic policy with maximum return, that satisfies a minimum amount of expected entropy:

$$\max_{\pi_{0:T}} \mathbb{E}_{\rho_\pi}\left[\sum_{t=0}^{T} r(\mathbf{s}_t, \mathbf{a}_t)\right] \text{ subject to:} \\ \mathbb{E}_{(\mathbf{s}_t, \mathbf{a}_t)} \sim \rho_\pi[-\log(\pi_t(\mathbf{a}_t|\mathbf{s}_t))] \geq \mathcal{H} \ \forall t \tag{4}$$

Here, $\mathcal{H}$ is the minimum amount of expected entropy to be satisfied by the policy. Haarnoja et al. provide an extensive derivation of an adjusted version of the GPI algorithm (which is used to learn the actor and critic functions in actor-critic methods), dubbed *soft policy iteration*, to learn value function and policies in the maximum entropy framework. Further, Haarnoja et al. provide a formal proof for convergence of the soft policy iteration algorithm. However, we chose not to repeat this in this seminar paper, as it does not provide significantly more insight into the idea behind SAC. Instead, we now move on to explore the practical version of the SAC algorithm.

B. SAC in practise

Even though Haarnoja et al. provide methods that allow to recursively solve for the optimal policy that satisfies the constraint in Equation IV-A, the practical algorithm relies on neural networks as function approximators for the Q-function and the policy [20, 21]. The well-known gradient descent algorithm is used to optimize the function approximators [20, 21]. This is necessary, because the soft-policy iteration algorithm, which would be used to find the optimal policy satisfying Equation IV-A, only works for discrete domains [21]. However, real-world robotic settings can hardly be discretized, requiring continuous actions and environments. Thus, Haarnoja et al. provide the *practical* version of the algorithm, using neural networks as function approximators for the soft Q-function and the policy. We will now explore how to compute the gradients for these neural network function approximators. For that, we must begin by introducing the soft state value function, which in turn allows us to compute soft Q-values for the soft policies [20]. The soft state value function allows us to determine the soft Q-values, according to the maximum entropy objective and is given by [20]:

$$V(\mathbf{s}_t) = \mathbb{E}_{\mathbf{a}_t \sim \pi}[Q(\mathbf{s}_t, \mathbf{a}_t) - \alpha \log \pi(\mathbf{a}_t|\mathbf{s}_t)], \tag{5}$$

The soft Q-function is parameterized by θ. The soft Q-function parameters are trained to minimize the following Bellman residual [20]:

$$J_Q(\theta) = \mathbb{E}_{(\mathbf{s}_t, \mathbf{a}_t) \sim D} \\ \left[\frac{1}{2}(Q_\theta(\mathbf{s}_t, \mathbf{a}_t) - (r(\mathbf{s}_t, \mathbf{a}_t) + \gamma \mathbb{E}_{\mathbf{s}_{t+1} \sim \rho}[V_{\bar{\theta}}(\mathbf{s}_{t+1})]))^2\right] \tag{6}$$

Here, $\sim D$ means that we sample from a replay buffer, which is typical for off-policy methods as this allows us to reuses old experiences. We find that the soft value function is implicitly parameterized by $\bar{\theta}$,

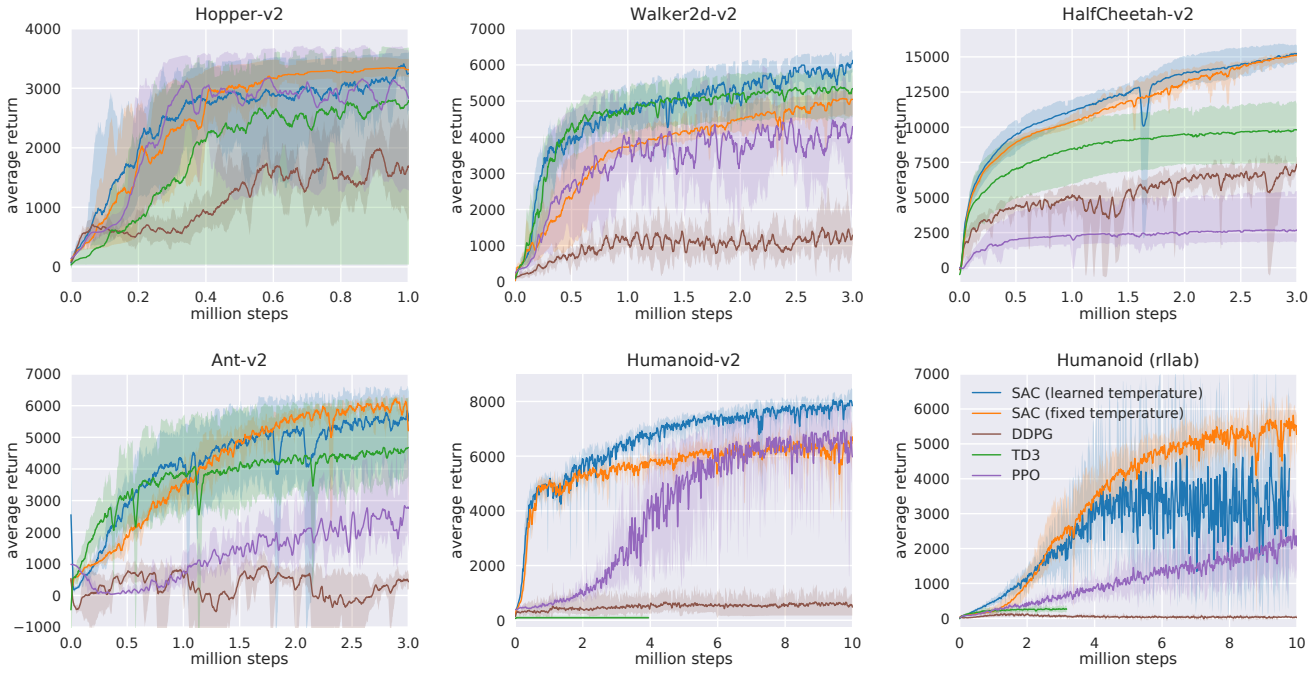

Fig. 1: Results of the simulated benchmark experience. Figure taken from [20]

which represents the use of target networks [20]. Target networks are always updated by a fraction of the weights of their non-target, up-to-date counterpart and have been shown to stabilize training [11]. We can observe that the Bellman residual measures the difference between the two sides of the equation. Thus, the Bellman residual will be small, when both sides are similar. Baired reports that as long as the Bellman residual is non-zero, the policy is no yet optimal [2]. As suggested by Baired, Haarnoja et al. train the soft Q-function to minimize the Bellman residual. For this, the stochastic gradient of Equation IV-B is used. Further, Haarnoja et al. train two independent soft Q-functions with parameters θ_i and use the minimal one of the two to compute the stochastic gradient [20]. This has been proposed by Fujimoto et al., who showed that the *double clipped Q-learning* trick can mitigate overestimation bias in the value function [18]. Additionally, Haarnoja et al. report that this significantly speeds up learning on challenging tasks [20].

The soft policy function approximator is parameterized by the vector ϕ. The parameters of the soft policy learned by minimizing the following objective [21, 20]:

$$J_\pi(\phi) = \mathbb{E}_{\mathbf{s}_t \sim D}\left[\mathrm{D}_{\mathrm{KL}}\left(\pi_\phi(\cdot|\mathbf{s}_t)\middle\|\frac{\exp(\frac{1}{\alpha}Q_\theta^{\pi_{\text{old}}}(\mathbf{s}_t,\cdot))}{Z_\theta^{\pi_{\text{old}}}(\mathbf{s}_t)}\right)\right] \tag{7}$$

Here, Haarnoja et al. exploit the Kullback-Leibler divergence. We can think about the Kullback-Leibler divergence as a measure of the difference between two probability distributions [1]. Thus, for the case of SAC, we minimize the distance between the current policy and the exponential of the soft Q-function, normalized by some intractable function Z. By approximating the gradient of Equation IV-B, we can again make use of stochastic gradients and gradient descent to update the parameters of the soft policy, as for the parameters of the soft Q-function.

Finally, without going fully into depth, the automatic update of the temperature parameter α (that controls the degree of stochasticity associated with the soft policy) is updated by computing the dual gradients for the following objective:

$$J(\alpha) = \mathbb{E}_{\mathbf{a}_t \sim \pi_t}[-\alpha \log \pi_t(\mathbf{a}_t|\mathbf{s}_t) - \alpha\bar{\mathcal{H}}] \tag{8}$$

Dual gradient descent is a popular method for optimizing dual lagrangian problems, thus, to optimize an objective subject to a specific constraint [4]. Thus, Haarnoja et al. apply the dual gradient method to update the temperature value, which subjects the policy to satisfy a minimum amount of entropy.

This covers all the relevant updates for the practical implementation of the soft actor-critic algorithm. The full algorithm is presented in Algorithm 1.

V. Results and Discussion

Haarnoja et al. conducted three distinct experiments in their paper, introducing the adjusted version of the SAC algorithm featuring automatic learning of the temperature parameter α [20]. The first experiment poses a simulated benchmark scenario, providing a comparison to other, on-policy and off-policy methods. The results for the benchmark experiment mostly used OpenAI gym suite [12], with the exception of complex, 21-dimensional Humanoid (rrlab) environment [13]. The results are shown in Figure 1. In the benchmarking experiment, the DDPG algorithm by Lilicrpa et al. [10] serves as a

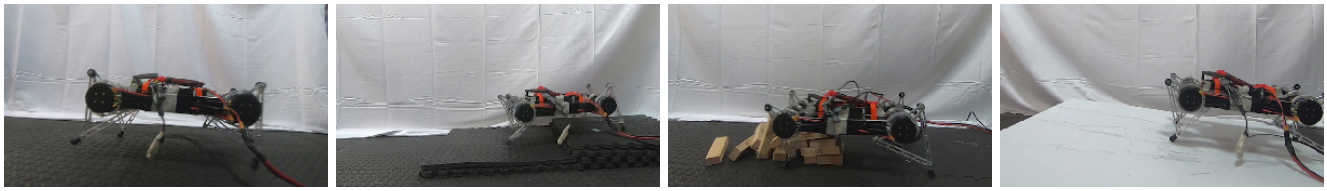

Fig. 2: Quadropedal walking experiment. From left to right: Training scenario, stairs setting, bricks setting, hill setting. The policy was only trained on the training scenario. Images taken from [20].

comparison to a different, off-policy algorithm. Further, DDPG is considered to be one of the more data-efficient off-policy methods, thus being closely related to the SAC algorithm. In fact, SAC can be considered as a stochastic version of the DDPG algorithm, because the main difference between the two is that SAC learns stochastic policies (due to the entropy objective), while DDPG learns deterministic policies. The TD3 by Fujimoto et al. [18] can be considered an extension to the DDPG algorithm, where the double Q learning has been introduced and applied to the DDPG algorithm. Thus the TD3 algorithm is even more similar to SAC than DDPG, since SAC also applied the double Q learning trick, again with the difference that SAC learns stochastic policies, while TD3 learns deterministic ones. Finally, the PPO algorithm by Schulman et al. [16] serves as an efficient and stable representative for the on-policy methods family. The key findings to take away from this benchmark are the following: SAC performs comparably well than the other state of the art methods on the easier tasks but significantly outperforms them on the more challenging tasks (Humandoid-v2 and Humanoid rllab). Note, that both the DDPG and TD3 algorithm fail to make any progress on the complex environments. This is not surprising since the DDPG algorithm (and its variations) tend to fail at tasks featuring high-dimensional spaces [20]. Still, even on most simpler scenarios, SAC tends to show better sample efficiency than the other two off-policy methods. Further, we can observe that the PPO algorithm (on-policy) had not yet reached convergence, specifically on the complex rllab humanoid scenario. This behavior is as antisipated since PPO is an on-policy method, which typically requires more data to converge. It is possible, that the algorithm would reach similarly strong results (possibly outperforming SAC), but as stated in the Related Work section of this paper, sample efficiency is a crucial criterion for real-world robotic reinforcement learning. The results attained by SAC on the simulated benchmark experiment indicate that SAC exceeds the current state of the art both in sample efficiency and final performance [20].

Further, two real-world robotic experiments have been conducted. We will only briefly report on these here. The first of the two real-world tasks applies the AC algorithm to a small, quadrupedal minitaur robot, with the goal of learning walking gaits [20]. SAC successfully learns to walk within 2 hours of real-world training, over roughly 400 episodes of maximal 500 steps [20]. Further, the learned policy generalized well to unseen territory. The policy has been trained on a flat surface, but could, without further training, walk up and down a slope, burst through a small wooden brick wall and climb small stairs [20]. While these results are promising, it must be noted that Haarnoja et al. conducted some reward engineering. Specifically, the value functions penalize large pitch angles and the extension of the front legs under the robot [20]. Haarnoja et al. do not provide results for a version of the algorithm that did not use this value function. Thus, we can not know to which extent the manually designed value function impacted the exciting results reported by Haarnoja et al. Nevertheless, this experiment is likely to be the first example for a deep reinforcement learning method that learned an underactuated quadrupedal locomotion task directly in the real-world, without using any simulated pretraining [20]. See Figure IV-B for images from the experiment [1].

Finally, the second real-world tasks involved a 3-finger dexterous robotic hand, that had to learn to rotate a valve-like object into the correct position, directly from the RGB images perceived by a stationary camera [20]. This is extraordinarily challenging tasks, due to the challenging, end-to-end perception system of the state and the physical difficulty of rotating the valve with a complex, 9 degrees of freedom hand [20]. However, the robot learned correct finger gaits to rotate the valve from a random starting position to the desired target position, within 20 hours of real-world training (including reset and neural network training times), over 300k environment interaction steps [20]. Thus, SAC is capable of learning complex manipulation tasks, without manually enhanced reward functions, directly in an end-to-end manner. Further, when the image perception system is replaced and the valve position is directly fed into the neural network function approximator, learning only took 3 hours, which is substantially faster than previous results on the same task (PPO, 7.4 hours) [20, 27].

Concluding the results section, the SAC algorithm by Haarnoja et al. outperforms the existing state of the art methods on high-dimensional, continuous domains, which are typical for real-world robotic scenarios. Further, SAC can learn directly from raw sensory input, without the need for simulated pretraining, which has

[1]The project website features videos of training and testing: https://sites.google.com/view/sac-and-applications/

been demonstrated in two distinct settings. These exciting results are obtained by including the entropy into the reward function, motivating the agent to explore more widely and allow to learn multiples modes of near-optimal behavior, which produces robust policies that generalize well to unseen variations of the environment.

VI. Conclusion

The SAC algorithm by Haarnoja et al., specifically the newer version of the algorithm with automatic temperature adjustment, indicates a new state of the art on high-dimensional continuous problems. This makes the algorithm attractive for real-world robotic scenarios, which often emit such properties. Mainly, this is achieved by significantly speeding up the learning process by exploiting the entropy associated with specific states. The problem of safe exploration, or how this might be affected by the entropy object, is not addressed by Haarnoja et al. Further, Haarnoja et al. used slightly modified reward functions for the quadrupedal locomotion task. Sadly, a baseline condition using the unaltered reward function is not provided. Thus, it is impossible to know to what extent the manipulated reward function contributed to the performance of the algorithm, which is critique-worthy. Apart from this, the paper presents exciting results, which contributed heavily towards the goal of learning policies for complex tasks in an end-to-end manner, directly in the real world.

References

[1] Solomon Kullback and Richard A Leibler. "On information and sufficiency". In: *The annals of mathematical statistics* 22.1 (1951), pp. 79–86.

[2] Leemon Baird. "Residual algorithms: Reinforcement learning with function approximation". In: *Machine Learning Proceedings 1995*. Elsevier, 1995, pp. 30–37.

[3] W. D. Smart and L. Pack Kaelbling. "Effective reinforcement learning for mobile robots". In: *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No.02CH37292)*. Vol. 4. May 2002, 3404–3410 vol.4. DOI: 10.1109/ROBOT.2002.1014237.

[4] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[5] Peter Dayan and Yael Niv. "Reinforcement learning: the good, the bad and the ugly". In: *Current opinion in neurobiology* 18.2 (2008), pp. 185–196.

[6] Brian D Ziebart. "Modeling purposeful adaptive behavior with the principle of maximum causal entropy". PhD thesis. figshare, 2010.

[7] Jens Kober, J. Andrew Bagnell, and Jan Peters. "Reinforcement learning in robotics: A survey". In: *The International Journal of Robotics Research* 32.11 (2013), pp. 1238–1274. DOI: 10.1177/0278364913495721. eprint: https://doi.org/10.1177/0278364913495721. URL: https://doi.org/10.1177/0278364913495721.

[8] Volodymyr Mnih et al. "Playing Atari with Deep Reinforcement Learning". In: *CoRR* abs/1312.5602 (2013). arXiv: 1312.5602. URL: http://arxiv.org/abs/1312.5602.

[9] Sergey Levine et al. *End-to-End Training of Deep Visuomotor Policies*. 2015. arXiv: 1504.00702 [cs.LG].

[10] Timothy P Lillicrap et al. "Continuous control with deep reinforcement learning". In: *arXiv preprint arXiv:1509.02971* (2015).

[11] Volodymyr Mnih et al. "Human-level control through deep reinforcement learning". In: *Nature* 518.7540 (Feb. 2015), pp. 529–533. ISSN: 00280836. URL: http://dx.doi.org/10.1038/nature14236.

[12] Greg Brockman et al. "Openai gym". In: *arXiv preprint arXiv:1606.01540* (2016).

[13] Yan Duan et al. "Benchmarking deep reinforcement learning for continuous control". In: *International Conference on Machine Learning*. 2016, pp. 1329–1338.

[14] Justin Fu, Katie Luo, and Sergey Levine. "Learning robust rewards with adversarial inverse reinforcement learning". In: *arXiv preprint arXiv:1710.11248* (2017).

[15] S. Gu et al. "Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates". In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. May 2017, pp. 3389–3396. DOI: 10.1109/ICRA.2017.7989385.

[16] John Schulman et al. "Proximal policy optimization algorithms". In: *arXiv preprint arXiv:1707.06347* (2017).

[17] J. Tobin et al. "Domain randomization for transferring deep neural networks from simulation to the real world". In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Sept. 2017, pp. 23–30. DOI: 10.1109/IROS.2017.8202133.

[18] Scott Fujimoto, Herke van Hoof, and David Meger. "Addressing function approximation error in actor-critic methods". In: *arXiv preprint arXiv:1802.09477* (2018).

[19] Tuomas Haarnoja et al. "Composable deep reinforcement learning for robotic manipulation". In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2018, pp. 6244–6251.

[20] Tuomas Haarnoja et al. "Soft Actor-Critic Algorithms and Applications". In: *CoRR* abs/1812.05905 (2018). arXiv: 1812.05905. URL: http://arxiv.org/abs/1812.05905.

[21] Tuomas Haarnoja et al. "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor". In: *CoRR* abs/1801.01290 (2018). arXiv: 1801.01290. URL: http://arxiv.org/abs/1801.01290.

[22] Thanard Kurutach et al. "Model-ensemble trust-region policy optimization". In: *arXiv preprint arXiv:1802.10592* (2018).

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

[24] T. Johannink et al. "Residual Reinforcement Learning for Robot Control". In: *2019 International Conference on Robotics and Automation (ICRA)*. May 2019, pp. 6023–6029. DOI: 10.1109/ICRA.2019.8794127.

[25] OpenAI et al. *Solving Rubik's Cube with a Robot Hand*. 2019. arXiv: 1910.07113 [cs.LG].

[26] Avi Singh et al. *End-to-End Robotic Reinforcement Learning without Reward Engineering*. 2019. arXiv: 1904.07854 [cs.LG].

[27] Henry Zhu et al. "Dexterous manipulation with deep reinforcement learning: Efficient, general, and low-cost". In: *2019 International Conference on Robotics and Automation (ICRA)*. IEEE. 2019, pp. 3651–3657.

Eidesstattliche Erklärung

Hiermit versichere ich, Finn Rietz, an Eides statt, dass ich die vorliegende Seminararbeit mit dem Titel *Soft Actor-Critic: Deep Reinforcement Learning for Robotics*, sowie die Präsentationsfolien zu dem dazugehörigen mündlichen Vortrag ohne fremde Hilfe angefertigt und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Alle Teile, die wörtlich oder sinngemäß einer Veröffentlichung entstammen sind als solche kenntlich gemacht.

Die Arbeit wurde in dieser oder ähnlicher Form noch nicht veröffentlicht, einer anderen Prüfungsbehörde vorgelegt oder als Studien- oder Prüfungsleistung eingereicht.

Declaration of an Oath

Hereby I, Finn Rietz, declare that I have authored this thesis, titled *Soft Actor-Critic: Deep Reinforcement Learning for Robotics*, and the presentation slides for the associated oral presentation independently and unaided. Furthermore, I confirm that I have not used other than the declared sources / resources.

I have explicitly marked all material which has been quoted either literally or by content from the used sources.

This thesis, in same or similar form, has not been published, presented to an examination board or submitted as an exam or course achievement.

Hamburg, October 22, 2020

Finn Rietz