Applying Soft Actor-Critic to OpenAI Ant

Zubin Bhuyan
Student id: 01744486
zubin_bhuyan@student.uml.edu

***Abstract*—In recent years, deep reinforcement learning has been successfully employed to solve a plethora of challenging problems. These RL algorithms are not just limited to simulations, but can be used in robots in real life. One such model-free approach is the Soft Actor Critic algorithm which allows for efficient learning from samples and better stability. This report discusses how SAC can be used for training to adapt with abruptly introduced limitations in robot capability and the robot can make improvements over time. Simulation of SAC was done on the Ant envionment in Open AI Gym framework.**

***Index Terms*—reinforcement learning, soft actor critic, openai**

I. Introduction

Reinforcement Learning is a form of machine learning where an agent "learns" from its experiences and strives to maximize its rewards by choosing the most appropriate action. It has been the area of interest for researchers from a vast number of domains, especially because the reward-driven approach is suitable for a many problems requiring an exploratory approach to a solution.

Advances in the last decade has enabled reinforcement learning approaches to take advantage of deep neural architectures (Fig 2[1]). RL approaches have been successfully applied not only to simulations [1], but to real world tasks [2] as well.

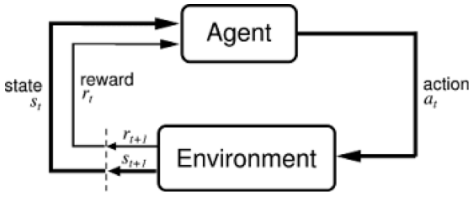

Fig. 1. A simple RL schema.

Traditional reinforcement learning paradigm comprise of an *environment* and an *agent*. The agent can perform a set of *actions* which takes it from one *state* to another, and with each transition is associated a *reward*. Fig 1 is a simple representation of the RL schema. The goal is to maximize the rewards collected over time. Policy is the term used to describe the rule based on which an agent takes actions. Policies in deep RL are parameterized, and these parameters are tweaked to achieve the most optimal policy possible.

Two approaches of model-free RL are *policy optimization* (learning an approximator value function) and *Q-learning*, which requires learning an approximate $Q_\theta(s, a)$. A more detailed explanation of RL concepts and ideas can be found in [3].

While there can be many variations in the low-level implementation, this is the high-level idea of reinforcement learning usually remains the same. Fig 3 is a non-exhaustive RL algorithm classification [2].

The focus of this report will be on Soft Actor Critic (SAC) algorithm [4] which has been shown to be able to applicable to a number of real life situations [5].

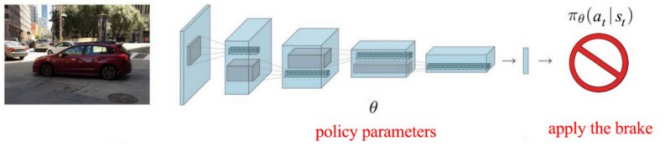

Fig. 2. A general representation of how deep NN architecture is used for RL.

The subsequent section discusses some previous work done in this area. Section III describes the methodology of SAC. This is followed by section IV which describes the experimental setup and the results. Section V is the conclusion.

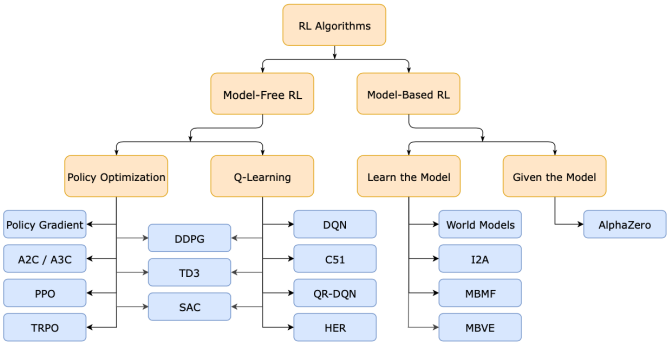

Fig. 3. Classification of RL algorithms.

II. Related Work

Model-free policy optimization Advantage Actor Critic algorithms perform gradient ascent directly to maximize the reward. In such methods the *critic* estimates a value function (either of Q or V), and the policy is updated by the *actor*. Both the functions of *actor* and *critic* are parameterized with deep neural networks, which are updated in every step of training.

Asynchronous Advantage Actor Critic (A3C) [6] is an actor-critic method where multiple workers perform parallel

[1]Image source: http://bit.ly/2WcqoIn

[2]Image source: https://bit.ly/30b3zb3

training and independeltly update a global value function. The independent nature of the workers allow for a more effective exploration. The A3C architecture is illustrated in fig 4.

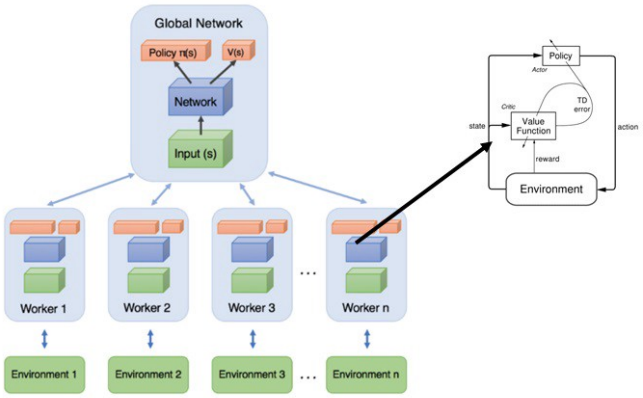

Fig. 4. Architecture of A3C.

The idea of maximizing entropy for learning adaptive behaviour was explored in [7] and [8]. It has been shown that policies maximizing entropy led to more exhaustive exploration and better action sequences [9].

III. Methodology

Soft-Actor critic (SAC) [4] is a model-free, off-policy RL method which is relatively more sample efficient than other algorithms. Another advantage of SAC is that it is "robust to brittleness in convergence" or adaptive to a changing environment. The main idea of SAC is *entropy regularization*, i.e. it seeks to maximize the policy entropy so that it encourages more exploration. This results in a higher likelihood of actions with similar Qvalues getting same probability. The following SAC objective function comprises of an additional entropy term along with the usual reward term.

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

A Q-network is trained with examples with the goal of minimizing the following error

$$J_Q(\theta) = \mathbb{E}_{(\mathbf{s}_t, \mathbf{a}_t) \sim \mathcal{D}} \left[\frac{1}{2} \left(Q_\theta(\mathbf{s}_t, \mathbf{a}_t) - \hat{Q}(\mathbf{s}_t, \mathbf{a}_t) \right)^2 \right]$$

A *policy network* is also optimized for parameters. The algorithm is given below:

Algorithm 1 Soft Actor-Critic

Input: θ_1, θ_2, ϕ
$\bar{\theta}_1 \leftarrow \theta_1, \bar{\theta}_2 \leftarrow \theta_2$
$\mathcal{D} \leftarrow \emptyset$
for each iteration **do**
  for each environment step **do**
    $\mathbf{a}_t \sim \pi_\phi(\mathbf{a}_t|\mathbf{s}_t)$
    $\mathbf{s}_{t+1} \sim p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$
    $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathbf{s}_t, \mathbf{a}_t, r(\mathbf{s}_t, \mathbf{a}_t), \mathbf{s}_{t+1})\}$
  end for
  for each gradient step **do**
    $\theta_i \leftarrow \theta_i - \lambda_Q \hat{\nabla}_{\theta_i} J_Q(\theta_i)$ for $i \in \{1, 2\}$
    $\phi \leftarrow \phi - \lambda_\pi \hat{\nabla}_\phi J_\pi(\phi)$
    $\alpha \leftarrow \alpha - \lambda \hat{\nabla}_\alpha J(\alpha)$
    $\bar{\theta}_i \leftarrow \tau\theta_i + (1-\tau)\bar{\theta}_i$ for $i \in \{1, 2\}$
  end for
end for
Output: θ_1, θ_2, ϕ

This algorithm has been tested on the Minitaur robot (for a walking task on unknown terrain), and the Dynamixel Claw (valve rotation task).

In this project we use the SAC algorithm to train the Ant agent of OpenAI Gym to walk in regular condition and compare the results when one of its legs is disabled. The experiments are described in details in the next section.

IV. Experiments and Results

A. Experimental setup

The primary goal of the experiments were to evaluate if the four-legged Ant [10], learning to walk with the soft-actor algorithm, could cope with the adversity of losing one of its limbs. The OpenAI Gym [11] toolkit was used along with the MuJoCo [12] Ant environment. *rlkit*[3], a Python repository of reinforcement learning algorithms, was used for initial testing. Two other repositories[4] [5] were also used for reference.

All simulations were performed on Virtual Machines on Google Cloud platform. These VMs each had 8 vCPUs, 30 GB of memory, 80 GB of SSD and one NVIDIA $K80$ graphcs card.

B. Simulation parameters and hyperparameters

The simulations for which the graphs are given below, a *learning rate* of 0.003, *temperature* value of 0.1, a *replay buffer* of $100,000$, and a *discount factor* of 0.99. Batch size for training was 256, optimizer used was *Adam*, activation function was ReLU, and policy was evaluated every 10 episodes.

The VMs were accessable through a console interface; the results of the simulations were stored as Tensorboard logs and were viewed later on a laptop.

[3]https://github.com/vitchyr/rlkit
[4]https://github.com/haarnoja/sac
[5]https://github.com/vitchyr/rlkit

C. Results

1) Training Ant with 4 legs: The reward achieved while testing and training were logged with the Tensorboard module. Fig 5 and 6 show how the reward values. for training and testing gradually increase with increasing episodes. This training was done for 1350 episodes, which took about than 9 hours to complete.

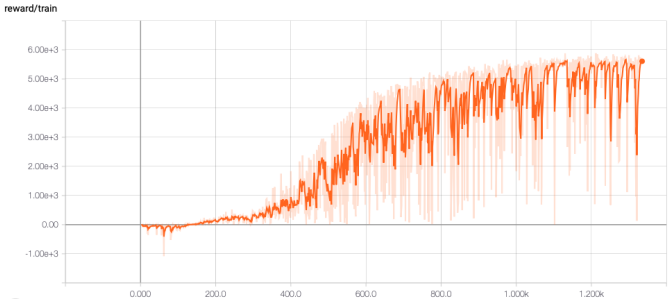

Fig. 5. Reward curve for training the Ant with 4 legs.

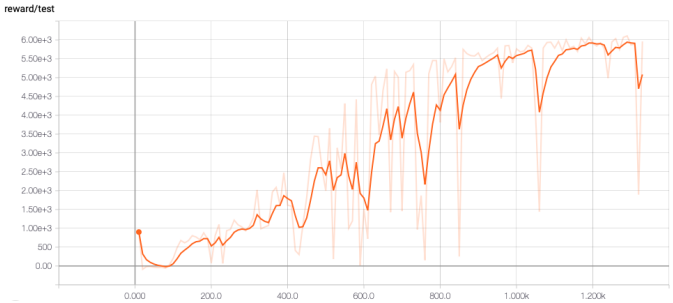

Fig. 6. Reward curve while testing the Ant environment with 4 legs.

2) Disabling one leg after 700 episodes of training: For the second part of the experiment, the agent was required to walk with 3 legs. So, after initial training of 700 episodes with 4 legs, one leg was disabled. The initial training was given so that the agent would at least get a change for some free exploration. Fig 6 and 8 show how the rewards change over time as the training goes on. We can observe that there is a steep fall in the reward after a leg is disabled. However, the value quickly goes up again, indicating that the agent was somehow able to move even with three legs. It should also be noted here that the rewards are not as high as it was with 4 legs.

A simulation was also done where one leg is disabled and agent gets no prior training. The reward values obtained in testing this case for 625 episodes of training is shown in fig 10. The graph indicates that the agent struggles to perform the task, where as in the other cases the agent's reward was increasing at 625 episode.

V. Conclusion

This report presents a study of application of soft actor critic algorithm to the four-legged Ant agent in OpenAI Gym. Simulations were performed for different cases where one of

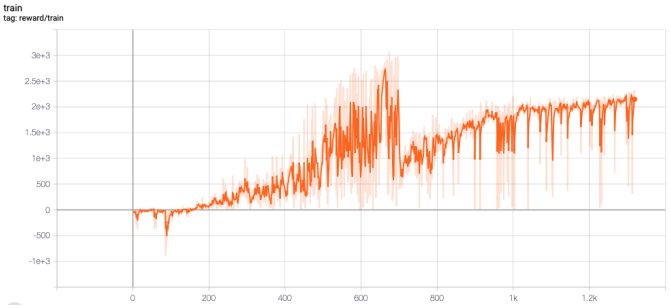

Fig. 7. Training the Ant agent with 3 legs (with initial 4-legged training of 700 episodes).

Fig. 8. Testing the Ant agent with 3 legs after every 10 epochs (with initial 4-legged training of 700 episodes).

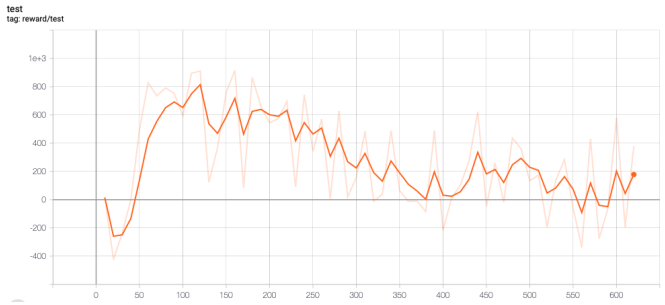

Fig. 9. Reward graph for agent with 1 leg disabled (no prior training).

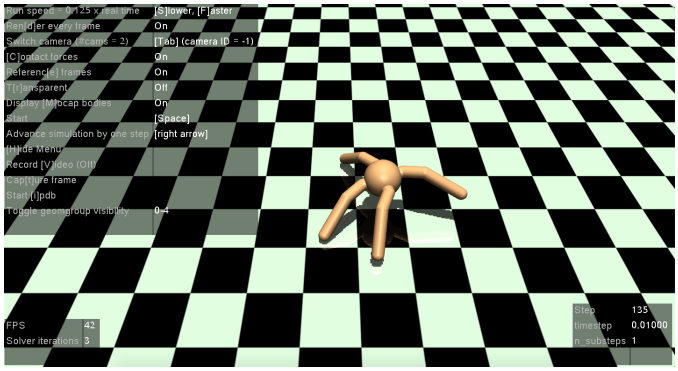

Fig. 10. Visualizing trained Ant agent with 4 legs.

the legs would be arbitrarily disabled. It was observed that if the agent had some training with 4 legs before one of its legs was turned off, it could adapt faster than than the case where it had no prior training.

This work can be further extended to analyze how much of prior training can be considered to be sufficient in order for an agent to complete a task with limited capabilities or in adversarial conditions. There is also scope for improvement in exploration during training for situations where initial training is not available.

REFERENCES

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, Feb. 2015.

[2] F. Ebert, C. Finn, S. Dasari, A. Xie, A. X. Lee, and S. Levine, "Visual foresight: Model-based deep reinforcement learning for vision-based robotic control," *CoRR*, vol. abs/1812.00568, 2018.

[3] R. S. Sutton and A. G. Barto, *Introduction to Reinforcement Learning*. Cambridge, MA, USA: MIT Press, 1st ed., 1998.

[4] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," *CoRR*, vol. abs/1801.01290, 2018.

[5] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, and S. Levine, "Soft actor-critic algorithms and applications," *CoRR*, vol. abs/1812.05905, 2018.

[6] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," *CoRR*, vol. abs/1602.01783, 2016.

[7] PhD thesis.

[8] B. D. Ziebart, A. Maas, J. A. Bagnell, and A. K. Dey, "Maximum entropy inverse reinforcement learning," in *Proc. AAAI*, pp. 1433–1438, 2008.

[9] T. Haarnoja, H. Tang, P. Abbeel, and S. Levine, "Reinforcement learning with deep energy-based policies," *CoRR*, vol. abs/1702.08165, 2017.

[10] J. Schulman, P. Moritz, S. Levine, M. I. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," *CoRR*, vol. abs/1506.02438, 2016.

[11] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," 2016.

[12] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033, IEEE, 2012.