Learning to Walk Stairs

Devesh Jeaven
School of Computer Science and Applied Mathematics
University of the Witwatersrand
GitHub Repository : https://github.com/deveshj48/Learning-to-Walk-Stairs

***Abstract*—Automated Quadrupedal robot control using reinforcement learning is extremely advantageous in solving real-world problems. These include, but are not limited to, military operations, space exploration and industrial applications. Over the past few years, there have been various algorithms presented for the purpose of training the agents to walk, in a way that does not require copious amounts of domain knowledge such as the model dynamics. Many of these approaches, however, train the robots on flat surfaces. We present a comparison of two popular methods, Soft Actor-Critic and Natural Evolution Strategies, in a simulated setting, for the task of learning to walk a flight of stairs. In particular, we compare the convergence rate, exploration of the agent when performing the task, and the ability to perform the task.**

I. Introduction

Legged robots consist of many joints and actuators that work together to operate in its environment. The process of designing controllers can be difficult and require vast amounts of domain knowledge to engineer the mechanics of a robot. Developments in deep reinforcement learning have simplified this process as no prior knowledge is required for a robot to accomplish a particular task [Haarnoja *et al.* 2018a].

Many reinforcement learning methods struggle with performing well in continuous spaces. Some approaches would discretize the action space, such as the use of tile coding, like Sherstov and Stone [2005] has done, which is a trade-off from the optimal solution, and can be expensive to perform when having multiple, very small discretizations [Lahire 2021]. Many approaches also require large amount of training training data and are very sensitive to hyperparmater tuning [Haarnoja *et al.* 2018a].

Soft-Actor Critic (SAC) and Natural Evolution Strategies (NES) are two methods that aim to solve these problems in a stochastic setting. In particular, these algorithms will be demonstrated by having a quadrupedal agent learn to walk a flight of stairs in a continuous, simulated setting.

Our key findings were as follows:

- The NES algorithms exhibited more exploration than the SAC algorithm.
- SAC received a higher return than NES for the same task.
- NES converged faster than SAC.

The report firstly provides preliminary information and related work, followed by the methodology which explains the algorithms used, environment setting, and hyperparameters. Finally, the experimentation section, detailing the results from applying both algorithms to the modified Ant environment.

II. Preliminaries

The goal of reinforcement learning is to find a policy that maximises the accumulated return.

A. Soft Actor-Critic

Many reinforcement algorithms struggle from high sample complexity and very sensitive convergence properties. The soft actor-critic (SAC) algorithm aims to solve these challenges by maximising both expected reward and entropy [Haarnoja *et al.* 2018b].

In actor-critic methods, the policy and value functions are independent, unlike traditional action-value methods. It consists of a Q-function that is trained (critic network), and a policy that is simultaneously optimised (actor network). The actor selects actions for the agent to take in its current environment and, as the name suggests, the critic *criticises* the actor's policy. As Figure 1 indicates, once an action is taken by the actor, the reward and next state is accepted by the critic. The critic evaluates these inputs using a temporal difference (TD) error, and the actor receives this feedback to improve [Sutton and Barto 2018].

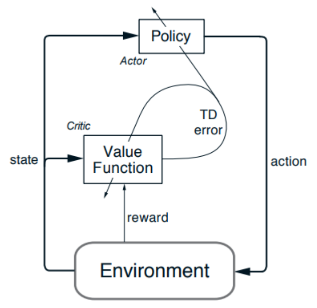


Fig. 1. Actor-Critic Architecture [Sutton and Barto 2018]

In practice, there is a third network, namely, a value network, which approximates the value of the states. This works together with the critic network to compute the TD error [Costa de Jesus *et al.* 2021]. The TD-error is computed as $\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$, where V is the value function, γ is the discount rate, s is the state at time t, and r is the reward.

Entropy regularisation is the main feature of SAC. Entropy is a measure of randomness, and SAC can control the randomness of the agent. This is related to the exploration-exploitation trade-off - the higher the entropy results, more exploration is done.

The computation of the value function of a policy (policy evaluation) and subsequently, the use of it to obtain a better policy (policy improvement), is known as policy iteration. For *soft* actor-critic in particular, the algorithm makes use of soft policy iteration. A soft return is when entropy is included in the accumulated return. Thus, through soft-policy iteration, we include entropy in our process of improving and evaluating the policy [Duan *et al.* 2021].

SAC is an off-policy algorithm. It is thus able to learn from past samples and replay buffers. Since it does not require new samples after each policy update like on-policy learning, it is highly sample-efficient.

SAC makes use of various efficient and innovative algorithms, such as target networks, double Q-trick and entropy regularisation

The target networks help us predict the next state actions and Q-values. This is only updated at every set interval, in order to stabilise the agent.

The double Q-trick consists of the algorithm containing two Q-functions and choosing the minimum value between the two approximators [Majid *et al.* 2023].

Putting together the key elements of SAC, we calculate the loss for each Q-value as follows:

$$L(\phi_i, \mathcal{D}) = \underset{(s,a,r,s',d)\sim\mathcal{D}}{E}[(\mathrm{Q}_{\phi_i}(s,a) - y(r,s',d)^2]$$

where the target y is:

$$y(r,s',d) = r + \gamma(1-d)(\underset{j=1,2}{min}\mathrm{Q}_{\phi_{targ,j}}(s',a') - \alpha log\pi_\theta(\tilde{a}'|s'))$$

In the target algorithm, $\mathcal{D}$ represents the replay buffer, $-\alpha log\pi_\theta(\tilde{a}'|s')$ represents the entropy and Q is a modified value function equation including entropy Lahire [2021].

B. Evolution Strategies

Evolution Strategies (ES) are a class of black box optimisation algorithms. This denotes that the algorithm does not take the specific problem into consideration, such as the agent and environment - it is purely an optimisation problem, and in particular, for continuous problems. This is an advantage as many real world problems are difficult to model directly [**?**].

ES is a subcategory of Evolutionary Algorithms, which mimic the the biological process of mutation, recombination and mutation. Other subcategories include Genetic Algorithms and Evolutionary Programming. [Ahrari and Shariat Panahi 2013].

We will be making use of OpenAI-ES, which is part of the category of Natural Evolution Strategies (NES). This means that is uses an estimated gradient on its parameters to update a search distribution [Wierstra *et al.* 2014]. In particular, the natural gradient is used, which improves on the slow convergence of gradient ascent. With gradient ascent, the Euclidean distance is used to measure the distance between distributions, develops a dependency to the parameterisation of a specific distribution. The natural gradient solves this by using a different, more 'natural' distance metric.

ES algorithms samples parameters of policies, unlike traditional RL algorithms, which samples actions from policies. Exploration is thus influenced by the parameters. [Salimans *et al.* 2017]

III. Related Work

Since many RL and ES methods surpassed human-level control, there have been various challenges and problems to solve. Due to this, there is an increasing amount of literature comparing these two strategies. Majid *et al.* [2023] compares the capabilities of various Deep RL (DRL) techniques to ES techniques, including parallelism, mulitagent learning, and non-Markovian learning. It was observed that DRL-based solutions seem to perform better in contexts that require scalable and adaptive behavior.

Milano and Nolfi [2022] found that RL methods perform better than ES methods when faced with environmental variation such as changing the initial position of the agent and changing the position of objects in an environment.

Both pieces of literature highlight relevant information for the purpose of our report, however, are both broad in the insight that it provides. We compare a specific use-case with different metrics to contribute to the related work.

IV. Methodology

A. Environment

In order to demonstrate the comparison of SAC and ES for the context of training a quadrupedal robot, we used the Ant-v4 OpenAI Gym environment. Aside from the agent's four legs, this environment was chosen as it consisted of a continuous action and observation space, which would effectively test the capabilities of both algorithms.

Stairs had to be placed into the environment, and there were various iterations of this. Firstly, a flight of five stairs was added onto the existing plane. This caused a problem, as the agent oftentimes abandoned the stairs and learned on the flat plane. Next, the plane was removed, leaving only the stairs. The agent thus learned that it is able to jump from the base of the stairs to the top of stairs. Since this was not the intended purpose of this experiment, we modified the flight of stairs from five to ten stairs. We also added a block at the top of the stairs which was used as a target destination for the agent. The final iteration of the environment is demonstrated in figure 2.

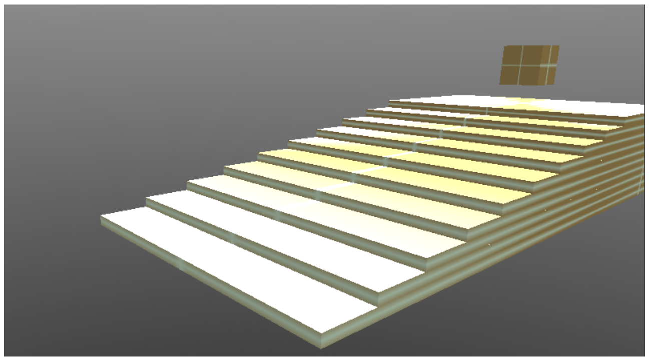


Fig. 2. Stairs Environment

Lastly, the reward function of the agent had to be modified in order for it to learn to reach the top of the stairs. The default reward was calculated as follows:

$$result = h\_reward + f\_reward - ctrl\_cost - contact\_cost$$

where the h_reward (healthy reward) and f_reward (forward reward) gives a positive reward if the agent is within its bounds and moves forward. The $ctrl\_cost$ and $contact\_cost$ produce negative rewards if the agent takes too large actions or the contact force is too large [OpenAI 2022][1]. We included a negative reward for the distance to the target. If the distance after the the agent takes an action is greater than the distance before the agent takes an action, then the agent gets rewarded negatively.

B. Soft Actor-Critic

Soft actor-critic (SAC) was used as it is highly effective in continuous spaces, and therefore, suitable for Ant-v4 to learn using this method. An implementation based on Haarnoja *et al.* [2018b] was used and adapted [Kumar 2019]. [2]

The structure of the networks consisted of 3 fully connected layers. The last layer of the policy network returned a mean and variance as the action, from a continuous action space, gets sampled from a Normal distribution. The action, however, is not directly sampled from the Normal distribution. It is calculated as follows:

$$action = \tanh(mean + std * z)$$

where the mean and std (standard deviation) is the output from the policy network, and $z \sim N(0, 1)$. This is the reparameterization trick, which removes the dependence on parameters [Haarnoja *et al.* 2018b].

The hyperparameters were as follows:

[1]https://github.com/openai/gym
[2]https://github.com/vaishak2future/sac

Hyperparameter	Value
Environment	CustomAnt-v0
Batch Size	128
Buffer Size	1000000
Learning Rate	0.0003
Hidden Layer Size	256
Discount Factor	0.99
Tau	0.01
Seed	42
Max Steps	1000000

TABLE I
HYPERPARAMETERS OF SAC

The maximum steps was set to 1000000 as there are multiple literatures, including Haarnoja *et al.* [2018a], whereby convergence occurs in this region. A large replay buffer also allows for more options for sampling actions.

Tau is also known as the smoothing factor, which is used when updating the target network weights to match the current value network weights.

The algorithm for SAC is as follows:

Algorithm 1 Soft Actor-Critic

```
Initialize parameter vectors $\psi$, $\bar{\psi}$, $\theta$, $\phi$.
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
        $\psi \leftarrow \psi - \lambda_V \hat{\nabla}_\psi J_V(\psi)$
        $\theta_i \leftarrow \theta_i - \lambda_Q \hat{\nabla}_{\theta_i} J_Q(\theta_i)$ for $i \in \{1, 2\}$
        $\phi \leftarrow \phi - \lambda_\pi \hat{\nabla}_\phi J_\pi(\phi)$
        $\bar{\psi} \leftarrow \tau\psi + (1 - \tau)\bar{\psi}$
    end for
end for
```

Fig. 3. Soft Actor Critic Algorithm [Haarnoja *et al.* 2018b]

C. Natural Evolution Strategies

Some advantages that ESs have over SAC, is that ESs do not require the problem to be modelled as a Markov Decision Process (MDP). Additionally, the objective function does not need to be differentiable [Majid *et al.* 2023]. There is thus no need for backpropagation. This is taken into consideration when comparing the algorithms in terms of convergence rate. It is worth noting that the absence of gradient calculations contributes to the algorithm being highly parallelizable. This is not explored in this paper.

The algorithm starts by choosing random parameters and tweaking them randomly. We move the guess towards whichever tweaks worked best. We therefore generate a population of parameter vectors, injected with Gaussian noise. The policy of each vector is run and we record the rewards. We update our parameters by taking a weighted sum of the population of parameters. This process is repeated until convergence [Salimans *et al.* 2017]. An implementation based on OpenAI-ES was used and adapted [Howuhh 2020].[3]

The hyperparameters were as follows:

Hyperparameter	Value
Environment	CustomAnt-v0
Number of Sessions	6500
Population Size	256
Steps per episode	1000
Standard deviation of noise	0.1
Hidden Layer Size	40

TABLE II
HYPERPARAMETERS OF ES

The algorithm was based on the following, from [Salimans *et al.* 2017]. F is the stochastic return provided by the environment and θ is the parameters of a stochastic policy, controlled by continuous actions.

Algorithm 1 Evolution Strategies

```
1: Input: Learning rate $\alpha$, noise standard deviation $\sigma$, initial policy parameters $\theta_0$
2: for $t = 0, 1, 2, \ldots$ do
3:     Sample $\epsilon_1, \ldots \epsilon_n \sim \mathcal{N}(0, I)$
4:     Compute returns $F_i = F(\theta_t + \sigma\epsilon_i)$ for $i = 1, \ldots, n$
5:     Set $\theta_{t+1} \leftarrow \theta_t + \alpha\frac{1}{n\sigma}\sum_{i=1}^{n} F_i\epsilon_i$
6: end for
```

Fig. 4. Evolution Strategies Algorithm

V. EXPERIMENTATION

When running experiments, we ran the algorithms on the original environment, as well as the custom stairs environment. This ensures that there is a baseline that we can compare to. It would also inform us if the reward function for the custom environment is not working for the agent, by generating graphs that are not similar in shape. The SAC environments were run for 1000000 steps, generating approximately 1400 episodes as shown by the x-axes of figure 5 and 6. It can be seen that the agent running on the stairs environment starts to converge from 600 episodes. Even though it converges, the agent does not successfully climb the stairs. This indicates that the reward function needs to be changed in order to better train the agent.

[3]https://github.com/Howuhh/evolution_strategies_openai

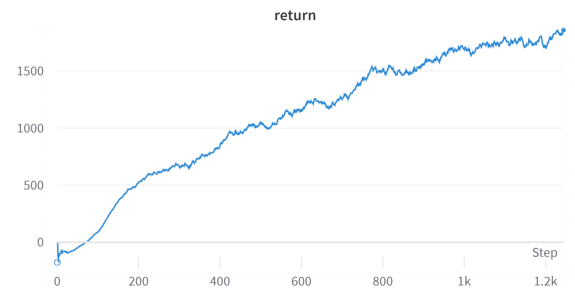


Fig. 5. Accumulated return when running SAC on the flat environment

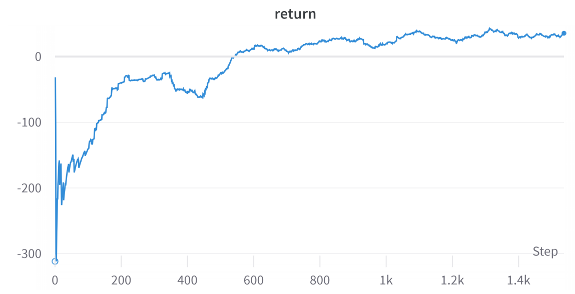
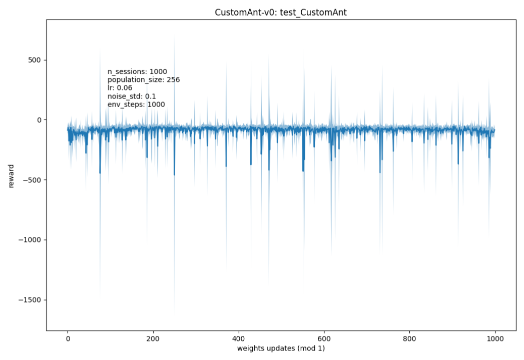


Fig. 6. Accumulated return when running SAC on the stairs environment

The ES algorithm ran for 1000 sessions. A session is one iteration of figure 4.



n_sessions: 1000
population_size: 256
lr: 0.06
noise_std: 0.1
env_steps: 1000

Fig. 7. Accumulated return when running NES on the flat environment

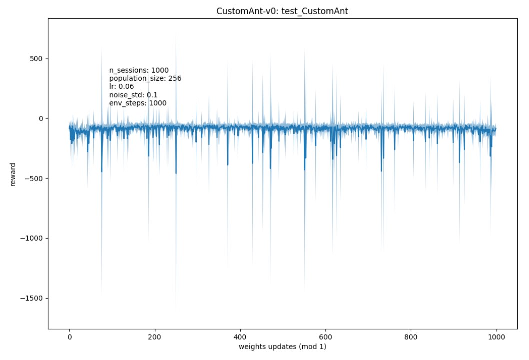


Fig. 8. Accumulated return when running NES on the stairs environment

We can see that the ES algorithm on both environments did not improve as we updated the weights, but instead converged to a value below 0. This could mean that more iterations are needed, or the hyperparameters need to be tuned. In particular, if we increase the standard deviation of noise, more noise would be injected into the parameters, and therefore, more exploration by the agent.

When comparing the animations of the Ant climbing the stairs, the ES algorithm climbed more stairs and it was evident that there was more exploration involved. It attempted more actions, and had more intentional movements.

VI. Conclusion

This paper has compared various aspects of NES and SAC in a continuous setting, including convergence, exploration, and its ability to complete the task. It can be seen that many factors need to taken into consideration, such as hyperparameters and modifications of the reward function, in order to more accurately learn the task. In future, reward shaping methods could be taken into consideration in order to better model the reward function. Additionally, learning hyperparameters could be more useful than modifying them manually.

References

[Ahrari and Shariat Panahi 2013] Ali Ahrari and Masoud Shariat Panahi. An improved evolution strategy with adaptive population size. *Optimization*, pages 1–20, 12 2013.

[Costa de Jesus *et al.* 2021] Junior Costa de Jesus, Victor Kich, Alisson Kolling, Ricardo Grando, Marco Cuadros, and Daniel Fernando Gamarra. Soft actor-critic for navigation of mobile robots. *Journal of Intelligent Robotic Systems*, 102, 06 2021.

[Duan *et al.* 2021] Jingliang Duan, Yang Guan, Shengbo Eben Li, Yangang Ren, Qi Sun, and Bo Cheng. Distributional soft actor-critic: Off-policy reinforcement learning for addressing value estimation errors. *IEEE transactions on neural networks and learning systems*, 33(11):6584–6598, 2021.

[Haarnoja *et al.* 2018a] Tuomas Haarnoja, Sehoon Ha, Aurick Zhou, Jie Tan, George Tucker, and Sergey Levine. Learning to walk via deep reinforcement learning. 2018.

[Haarnoja *et al.* 2018b] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. 2018.

[Howuhh 2020] Howuhh. evolution_strategies_openai. 2020.

[Kumar 2019] Vaishak Kumar. Soft actor critic demystified. 2019.

[Lahire 2021] Thibault Lahire. Actor loss of soft actor critic explained. 2021.

[Lazaric *et al.* 2007] Alessandro Lazaric, Marcello Restelli, and Andrea Bonarini. Reinforcement learning in continuous action spaces through sequential monte carlo methods. *Advances in neural information processing systems*, 20, 2007.

[Majid *et al.* 2023] Amjad Yousef Majid, Serge Saaybi, Vincent Francois-Lavet, R Venkatesha Prasad, and Chris Verhoeven. Deep reinforcement learning versus evolution strategies: a comparative survey. *IEEE Transactions on Neural Networks and Learning Systems*, 2023.

[Milano and Nolfi 2022] Nicola Milano and Stefano Nolfi. Qualitative differences between evolutionary strategies and reinforcement learning methods for control of autonomous agents. *Evolutionary Intelligence*, pages 1–11, 2022.

[OpenAI 2022] OpenAI. gym. 2022.

[Salimans *et al.* 2017] Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever. Evolution strategies as a scalable alternative to reinforcement learning. 2017.

[Sherstov and Stone 2005] Alexander A Sherstov and Peter Stone. Function approximation via tile coding: Automating parameter choice. In *International symposium on abstraction, reformulation, and approximation*, pages 194–205. Springer, 2005.

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

[Wierstra *et al.* 2014] Daan Wierstra, Tom Schaul, Tobias Glasmachers, Yi Sun, Jan Peters, and Jürgen Schmidhuber. Natural evolution strategies. *The Journal of Machine Learning Research*, 15(1):949–980, 2014.