Sapienza Università di Roma
Reinforcement Learning in
Artificial Intelligence

24/02/19

Soft Actor Critic in OpenAI-Gym BipedalWalker-v0 environment

Lorenzi
Flavio

Summary of the report

1.Soft Actor Critic

Soft Actor-Critic is an off-policy actor-critic algorithm based on the maximum entropy RL framework.
In this framework, the actor aims to simultaneously maximize expected return and entropy; that is, to succeed at the task while acting as randomly as possible (searching for a stochastic policy).
Entropy is a quantity which, roughly speaking, says how random a random variable is. <<...*If a coin is weighted so that it almost always comes up heads, it has low entropy; if it's evenly weighted and has a half chance of either outcome, it has high entropy…*>>.

Let x be a random variable with probability mass or density function P.
The entropy H of x is computed from its distribution P according to:

$$H(P) = \underset{x\sim P}{\mathrm{E}}[-\log P(x)]\,.$$

In entropy-regularized reinforcement learning, the agent gets a bonus reward at each time step proportional to the entropy of the policy at that time-step.
This changes RL problem into:

$$\pi^* = \arg\max_{\pi} \underset{\tau\sim\pi}{\mathrm{E}}\left[\sum_{t=0}^{\infty}\gamma^t\Big(R(s_t,a_t,s_{t+1}) + \alpha H\left(\pi(\cdot|s_t)\right)\Big)\right]$$

The maximum entropy objective to maximize is the Temperature parameter **α** that determines the relative importance of the entropy term against the reward. SAC makes use of four networks: a state value function V, a soft Q-function Q, a policy function π , a target value network.

The policy π(s) is implemented as an MLP network. Such parametrized policy it's used in off-policy algorithm in which a simple FIFO experience replay buffer (D) enables reuse of previously collected data for efficiency.

The off-policy algorithm used in this work is an Actor-Critic method which behaviour is the result of the 'Actor' component, determining the best action to take from current state (policy), and the 'Critic', that plays the role of evaluating how good is the selected action generating an output score given a state-action input.

The goal of this optimization problem is to generate an optimal policy which maximize the max entropy objective obtained after a training process on a given environment. $\pi^* = \arg\max_{\pi} J(\pi)$

2. Brief State of the Art

In 2017 the idea of Soft Actor Critic was set up in the offices of the Berkeley University in California. SAC has become a Model-Free Reinforcement Learning algorithm, which optimizes a stochastic policy in an off-policy way.

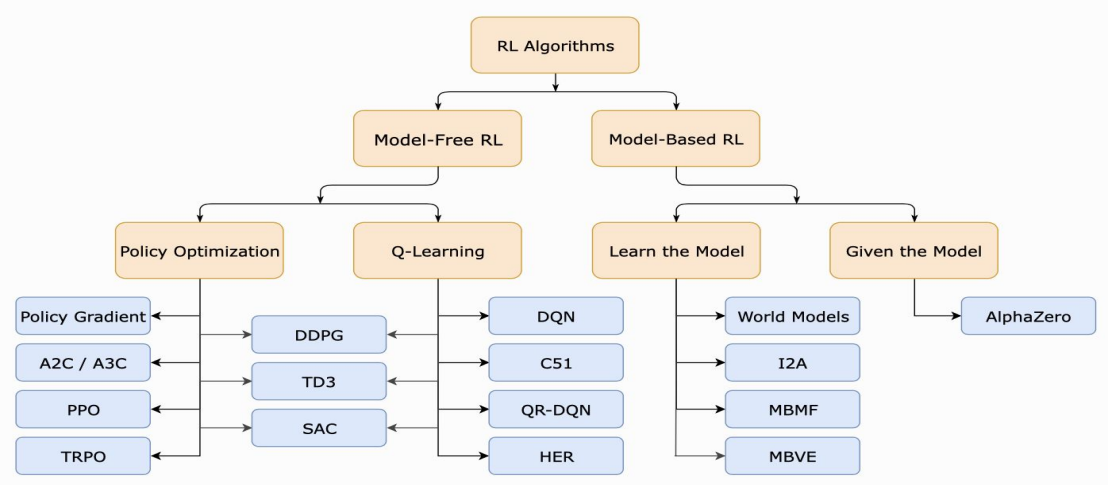


Image 1

It has been thought to be more efficient than traditional algorithms and to be robust to brittleness in convergence like is defined in the paper "*Soft Actor-Critic Demystified*", where it is shown a PyTorch implementation of SAC with a Minotaur Robot. Not only the Robot learns in a really short time duration but it also learns to generalize conditions that it hasn't seen during training. SAC thus brings us ever so close to using Reinforcement Learning in non-simulation environments for applications in robotics and other domains.

In the paper *"Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor"*, published on August 2018, it's shown that SAC achieves "state-of-the-art performance", outperforming prior on-policy and off-policy methods in sample-efficiency and asymptotic performance. Furthermore in contrast to other off-policy algorithms, it is very stable achieving similar performance across different random seeds. All this suggests us that SAC is a promising candidate for learning in real-world robotics tasks, as already mentioned above.

In this last document, in addition to explaining the quality of the algorithm and its theoretical part, there are important graphs about performance of SAC on the environments of OpenAI Gym,very similar to the one we will see in the next section.

Let's see it for example the trained HalfCheetah-v2 environment where the solid curves corresponds to the mean and the shaded region to the minimum and maximum returns over the five trials:

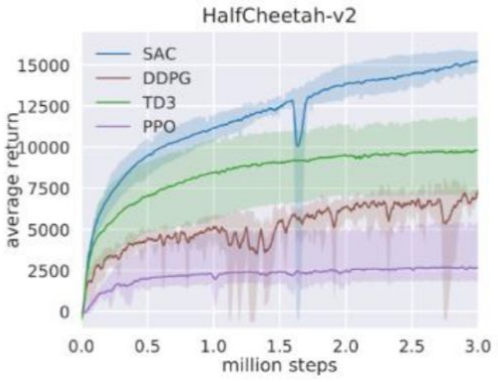


Image 2

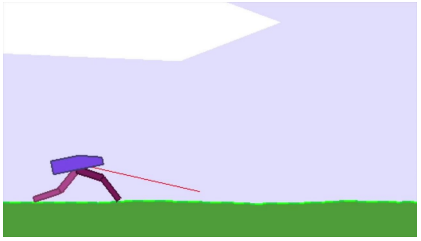
So it is shown a comparison between SAC and other classic RL algorithms, deep deterministic policy gradient (DDPG), twin delayed deep deterministic policy gradient (TD3), and proximal policy optimization (PPO). SAC achieves the best performance, and it performs well also in the worst case (key concept for real-world applications).

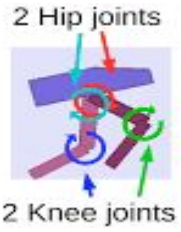
We can conclude that this was an innovative and revolutionary approach in a field which is constantly expanding like Reinforcement Learning, so it becomes very important for the future of the Artificial Intelligence.

3. Approaches to the problem

Now let's take a closer look at my personal approach to Reinforcement Learning Sac Algorithm carried out thanks to the library Spinning-up developed from Berkeley University that was born with the aim to help anyone want to learn deep reinforcement learning. My work is based on the OpenAI Gym environment BipedalWalker where we have the goal to train an humanoid agent to walk/run along a path. Reward is given for moving forward for 100 episodes, total 300+ points up to the far end. If the robot falls, it gets -100.

Image 3





My personal approach was developed as follows: first of all it played a key role the *code understanding* and the *hyperparameters tuning*. Than I tried *to find the best policy*, through graphs and agent rendering after each experiment to classify each singular performance.

3.1 Hyper-parameters

Our SAC algorithm is based on Adam, that is an optimization algorithm used instead of the classical stochastic gradient descent procedure to update network weights iterative based in training data.

```
pi_optimizer = tf.train.AdamOptimizer(learning_rate=lr)
```

Learning rate is a hyper-parameter used in training operation that controls how much we are adjusting the weights of our network with respect the loss gradient.
If it is reduced you will have a slower but also more accurate training, but too small it can be still inadequate, so it must be tested on the agent in question.
If it is high, then training may not converge or even diverge. Weight changes can be so big that the optimizer overshoots the minimum and makes the loss worse.

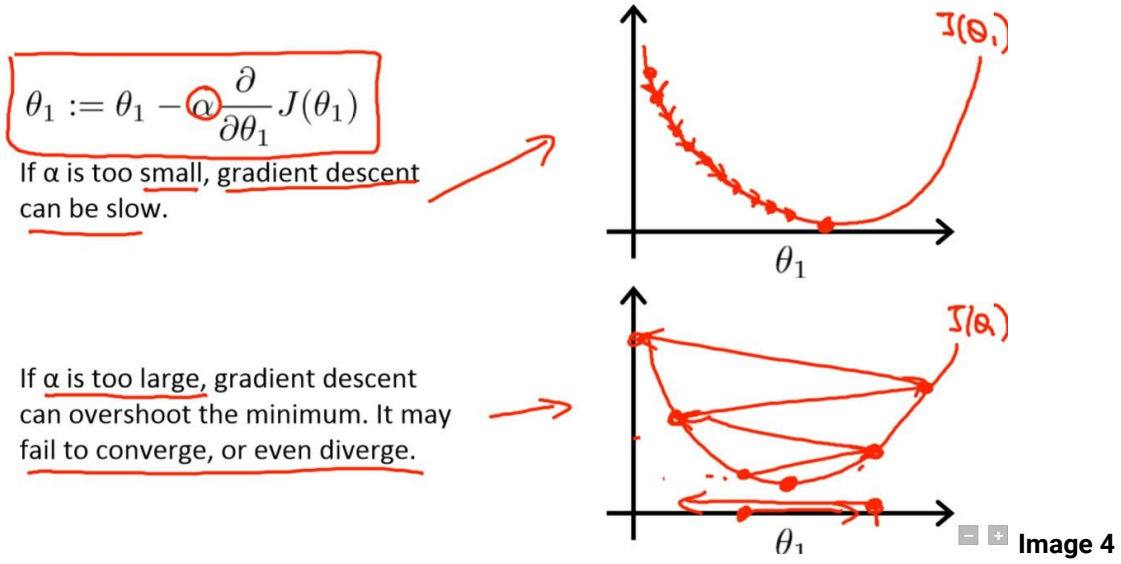


Image 4

During training, the agent must be tested on many epochs (gradient update) where steps per epoch indicates "*how big the cycle (training) is for each step*", that is the number of steps of batch interaction (state-action pairs) for the agent and the environment in each epoch.
It is the policy updating: at the end of the epoch we check the average cost and if it is improved we save a checkpoint.

```
total_steps = steps_per_epoch * epochs
```

The network structure takes inspiration from biological neurons and have a common architecture: an input layer, an output layer and one or more hidden layers. A hidden layer is made up of neurons connecting them to the layer before and after. These values have a big influence on the final output time it takes to train the network. In our case of study the size will be very extended enough to process an enormous amount of informations.
Finally we must talk about the *entropy regularization coefficient,* alpha: from the tuning process we understand its important role in the balancing process between exploration and exploitation; for higher alpha the agent is encouraged in exploring new action combination like a global search; on the other hand lower alphas lead to a more accurate exploitation of already know combination.

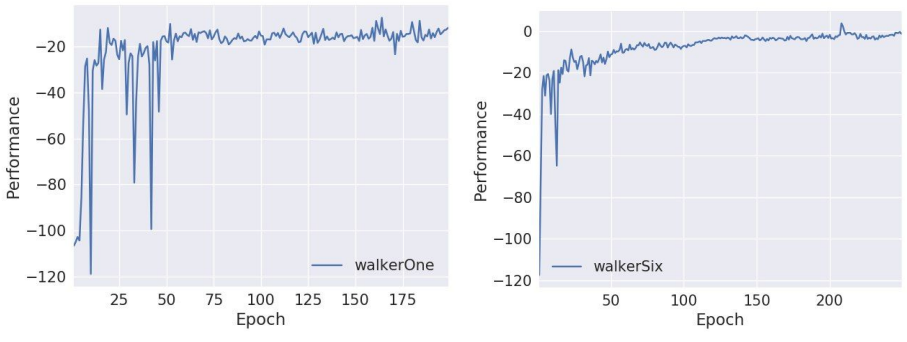
3.2. Finding the best performance: problem of the time

So, through setting hyperparameters it was possible with different tests to find the best possible performance for our agent.
Unfortunately, after some attempts, I immediately realized that the used network is not totally compatible with Bipedal env, because of the long duration of each episode (about 10 times more than its "brother" of Box2d-Gym LunarLender env), with a too large set of state-action combinations. So the training times were very long, even to get only small improvements in performance.

4.Results

In this section we will see my personal results with the goal to find the best policy for the agent against the incompatibilities, focusing attention on four main steps.

(1)I started my work by comparing performances between the default algorithm (walkerOne) and one of my first improvements (walkerSix).



Images 5

Despite my walkerSix is not ready yet to walk independently, we can see an improvement in performance. This was obtained by doubling the size of the hidden layers from [32,32] to [64,64], the number of steps for epoch (set to 10000 from 5000) and increasing the number of steps up to 250.

(2)From the graphs we note that it gets stuck in a local max point, so increasing only the steps per epoch or the episodes would improve the output, but very little.

As mentioned in the previous section, we must work on the size improvement of the hidden layers, but with the negative effect that the training time will be much longer (4 hours with size [512]; 10-15 hours at least with [1024]).

After a training of 5 hours in walkerSeven, without find success I put these new values into walkerOtto experiment:

steps_per_epochs= 15000; start_steps= 14000; epochs= 100; hidden_size= 1024*2

After a training of 15 hours the bipedal manages to stand up and slowly seems to go forward, but not enough. From this follows that In fact the performance of both are bad (image 6):

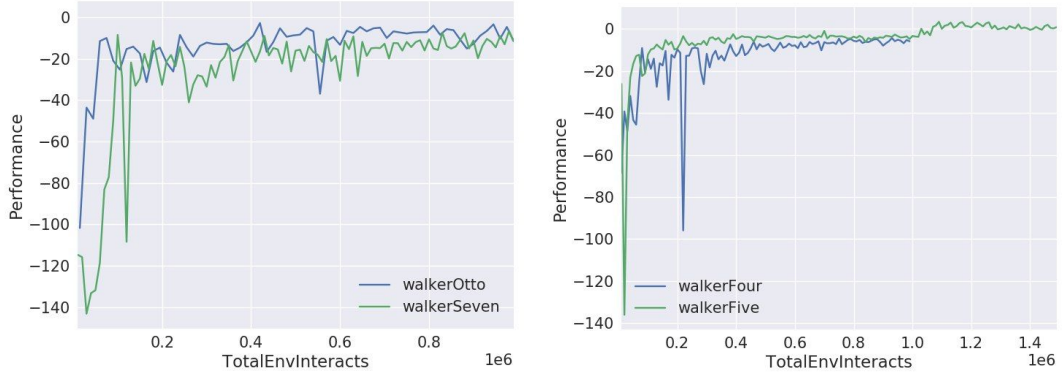


Image 6, 7

(3)From image 7 we can understand the learning-rate setting importance: both algorithm are launched with same steps_per_epochs and start_steps (10000), the same default values for alpha, with hidden_size= 512*2 (after the discovery that too large networks cause instability).

In walkerFour I put lr = 1*e^-3 while in walkerFive lr = 1*e^-4. It's evident that the agent with higher value oscillates more before finding stability. So I consider the second for the last exp. To conclude, moreover, a smaller lr is considered not efficient because after a training with lr = 1*e^-5, i found very bad results.

(4)So the best result I achieved (in according with graph performance) was putting into practice everything I understood from previous attempts, with the following hyperparameters:

```
seed=0,
steps_per_epoch=10000,
epochs=150,
replay_size=int(1e6),
gamma=0.99,
lr=1e-4,
alpha=0.2,
start_steps=10000,
hidden_size=[256,256]
```

Image 6

This is the output during the training:

```
|      AverageEpRet |         -42.9 |      #reward finale
|             EpLen |         1e+03 |
| TotalEnvInteracts |      1.49e+06 |      #somma max_ep_length
|     AverageQ1Vals |          48.3 |      #valori rete Q1
|     AverageQ2Vals |          48.3 |      #valori rete Q2
|      AverageVVals |          48.8 |      #rete V
|      AverageLogPi |         -2.71 |      #rete LogPi
|             LossPi |         -48.8 |          #perdita per le quattro reti
|             LossQ1 |        0.0024 |
|             LossQ2 |       0.00238 |
|              LossV |       0.00307 |
|               Time |      7.39e+03 |      #circa 148 min (2,5 ore)
-----------------------------------------------------------------------
```

In the end this is the output of the attached video, after first 5 episodes, about the learned policy:

```
(spinningup) MBP-di-Flavio:spinningup flaviolorenzi$ python -m spinup.run test_pol
2019-03-12 10:03:50.512601: I tensorflow/core/platform/cpu_feature_guard.cc:141] Y
Using default action op.
Logging data to /tmp/experiments/1552381435/progress.txt
Episode 0        EpRet -72.208   EpLen 1600
Episode 1        EpRet -66.732   EpLen 1600
Episode 2        EpRet -72.317   EpLen 1600
Episode 3        EpRet -65.732   EpLen 1600
Episode 4        EpRet -71.628   EpLen 1600 Episode 5    EpRet -67.289   EpLen 160
Traceback (most recent call last):
```

Image 7

So this is the best result I've found in terms of reward and performance.
However I found a similar result with the same rewards and with an agent who walks even better (called walkerT), starting from this solution (point 4) but increasing hidden size to 1024 and decreasing number of epochs to 10 (with more episode I

found worst results). The very problem is that the plot of the graphs cannot say us if this is better than previous ones (too few epochs to see performance graph).

From this I understand an important thing: concept described by image 4 is valid for the entire hyperparameters tuning, because if we go beyond a certain limit, instability immediately arises.

5.Conclusion

This report was aimed at the presentation, testing and evaluation of SAC, an off-policy maximum entropy deep reinforcement learning algorithm that provides an efficient learning keeping the benefits of entropy maximization and stability.
After many training and setting I can say that this SAC algorithm and the Bipedal env are not perfectly compatible, so at most we can found the best possible adaptation.
A successful learner will "solve" the reinforcement learning problem by achieving an average reward of 300 over 100 consecutive trials. The best result we obtained is -65.
So during my experience I observed that too high steps per epochs (more than 10000) can bring us to worst results (same with epochs), and increase the size of the hidden layers may not be good for the network (instability),unless we find the right combination.

Obviously during this time I thought about some possible important changes, which could increase the performance of the algorithm, but they are only ideas to be tested:

-We could think to other extensions about the Stochastic Gradient Descent and improve the already used (Adam and Polyak Averaging), like adaGrad: this strategy often improves convergence performance over standard stochastic gradient descent in settings where data is sparse and sparse parameters are more informative.
Otherwise we could improve the Adam with Adamax or Nadam.

-After my experience, an alternative solution that can bring the agent to a good result (at least to a positive reward) can be the following:
Starts from walkerT experiment and increase the hidden layers size to 2048.

This has been proven (test called walkerEl) with very few epochs , because of the too much training time, but it leads me to a result very similar to the walkerFive and walkerT, so I think with 20 episodes (and 10000 steps per epochs) we can reach very good results.
Obviously the estimated time of training is at least 24 hours.

6. References

- Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor; Haarnoja, Zhou, Abbeel, Levine
- Soft Actor-Critic Algorithms and Applications; Haarnoja, Zhou, Hartikainen, Tucker
- Actor-Critic Reinforcement Learning with Neural Networks in Continuous Games; Leuenberger, Wiering
- Off-Policy Actor-Critic; Degris, White, S. Sutton
- TensorFlow Deep Learning Projects; Grigorev, Shanmugamani Soft Actor-Critic Demystified, Kumar
- https://towardsdatascience.com/soft-actor-critic-demystified-b8427df61665
- https://github.com/openai/spinningup/blob/master/docs/algorithms/sac.rst
- https://github.com/openai/gym/blob/master/gym/envs/box2d/lunar_lander.py
- http://rail.eecs.berkeley.edu/deeprlcourse/static/homeworks/hw5b.pdf
- https://bair.berkeley.edu/blog/2018/12/14/sac/
- https://towardsdatascience.com/estimating-optimal-learning-rate-for-a-deep-neural-network-ce32f2556ce0