RL-Course 2023: Final Project Report

LosNinos: Okan Coskun

February 25, 2024

1 Introduction

Reinforcement learning is becoming increasingly influential in multiple areas of our lives. As part of our final project for the Reinforcement Learning course at the University of Tuebingen, I have taken a deeper look into its application in the Laser Hockey challenge. Laser Hockey [2] is a custom environment, where two competing agents play a simulated hockey game against each other.
While there are different and various algorithms that can be considered for this task, I had a closer look to one specific solution. We don't just get to see on how the algorithm works, but we will also have a deep dive under its hood to see the mathematical formulas that form the algorithm. Furthermore, I will discuss and have a short look on its behaviour in regard to some changes.

2 Soft Actor-Critic

Soft Actor-Critic (SAC) is an off-policy algorithm. Unlike other off-policy algorithms (e.g. TD3), its exploration comes from its emphasis on maximizing the entropy, which represents the uncertainty in the agent's actions. By maximizing entropy, SAC encourages exploration and prevents the agent from prematurely converging to maybe suboptimal policies. In the following, the algorithm will be explained in more detail.

2.1 General Concept

To get a better understanding of the algorithm, I will quickly cover some terms. We have a stochastic policy π_ϕ (also called actor), which is parametertized by ϕ and responsible for returning the parameters of the action distribution. We usually have two Q-value critics $Q_{i_\theta}(s, a)$, which evaluate the state. The parameters are defined by θ. Last but not least, there are corresponding target Q-value critics $Q_{ti_\theta}(s, a)$. Usually, Q_θ is referred to the soft Q-value function. The term "soft" comes from the fact that the algorithm basically adds entropy to the objective function to ensure randomness.

2.2 Entropy regularization

The main goal is to learn an optimal policy π^* that maximizes both rewards and entropy, s.t.:

$$\pi^* = \arg\max_\pi \sum \mathbb{E}_{(s_t,a_t)\sim\rho_\pi}[r(s_t, a_t) + \alpha H(\pi(\cdot|s_t)) \tag{1}$$

whereas the entropy is defined as:

$$H(\pi(\cdot|s_t)) = \mathbb{E}_{a\sim\pi(\cdot|s)}[-\log(\pi(a|s))] \tag{2}$$

Entropy is usually used in the context of uncertainty or randomness. Having a distribution with higher entropy means that it is harder to predict which value will occur. In the context of reinfocement learning and taking actions, this basically means that a higher entropy favors exploration. It is a way to introduce and control stochasticy.

2.3 Policy Iteration

A general concept in reinforcment learning is the so called Policy Iteration, which can be divided into two parts, policy evaluation and policy iteration. Firstly, it starts with a random policy, which of course is not optimal, and it calculates the value for each visited state based on the reward it received for taking the actions. As we want to improve our policy, we introduce the second step. We change our policy by updating our behaviour based on the policy evaluation, as we get a brighter understanding of the environment. Combining these two steps, we get the policy iteration. This concept will now be explained in more detail w.r.t. SAC.

2.3.1 Soft Policy evaluation

When defining the value function, it is important to take the entropy into consideration.

$$\begin{aligned} V(s_t) &= \alpha H(\pi(\cdot|s_t)) + \mathbb{E}_{a\sim\pi(\cdot|s)}[Q_\theta(s_t,a)] \\ &= \mathbb{E}_{a\sim\pi(\cdot|s)}[-\log(\pi(a|s))] + \mathbb{E}_{a\sim\pi(\cdot|s)}[Q_\theta(s_t,a)] \\ &= \mathbb{E}_{a\sim\pi_\phi(\cdot|s_t)}[Q_\theta(s_t,a) - \alpha\log(\pi_\phi(a|s_t))] \end{aligned} \tag{3}$$

Breaking it down, this simply means that we take the average of the different actions given from the policy distribution π plus the entropy. In more detail, this means that "in the policy evaluation step of soft policy iteration, we wish to compute the value of a policy pi according to the maximum entropy objective. For a fixed policy, the soft Q-value can be computed iteratively, starting from any function Q : S × A → R and repeatedly applying a modified Bellman backup operator T pi".[1] Regardless of SAC, we know that we can apply the Bellman operator to achieve the following: [**empty citation**]

$$T^{\pi_\phi}[Q_\theta(s_t,a_t)] = r + \gamma V_\theta(s_{t+1}) \tag{4}$$

From the previous eqution, I can rewrite it as:

$$T^{\pi_\phi}[Q_\theta(s_t,a_t)] = r + \gamma\mathbb{E}_{a\sim\pi_\phi(\cdot|s_{t+1})}[Q_\theta(s_{t+1},a) - \alpha\log(\pi_\phi(a|s_{t+1}))] \tag{5}$$

where $V_\theta(s_{t+1})$ is defined as

$$V_\theta(s_{t+1}) = \mathbb{E}_{a\sim\pi_\phi(\cdot|s_{t+1})}[Q_\theta(s_{t+1},a) - \alpha\log(\pi_\phi(a|s_{t+1}))] \tag{6}$$

2.3.2 Policy improvement

Having covered the evaluation, we now need to also talk about the improvement step. The goal is to update the policy so that it is optimal w.r.t to the soft Q-value function. One wants tractable policies, therefore there is a restriction on the policies (commonly used ones are gaussians). For that restriction to work, we project the improved policy into the desired set of policies. It turned out that it is convenient to use the information projection defined in terms of the Kullback-Leibler divergence.[1] We will not go into heavy detail here, as it this would exceed the scope of this work. For the policy improvement step to work, we update the policy as per:

$$\pi_{new} = \arg\min_{\pi'\in\Pi} D_{KL}(\pi'_\phi(\cdot|s_t))||\frac{\exp(\alpha Q_\theta(s_t,\cdot))}{Z_\theta(s_t)}]. \tag{7}$$

As described in theorem 1 in Tuomas Haarnoj paper, repeating soft policy evaluation and soft policy imporovement from any $\pi \in \phi$ converges to a policy π^* s.t. $Q^{\pi^*}(s_t, a_t) \geq Q^{s_t, a_t}$ for all $\pi \in \phi$ and $(s_t, a_t \in SxA)$[1].

2.4 Lossfunctions and updates

Having talked about policy iteration, it is now time to have a look at updating the parameters. As mentioned earlier in the general concept chapter, our policy and critic are each parameterized by ϕ and θ.

2.4.1 Critic

The ciritc parameters are learned by minimizing the following term:

$$J_Q(\theta) = \mathbb{E}_{(s_t, a_t, s_{t+1}) \sim D}[(Q_\theta(s_t, a_t) - (r(s_t, a_t) + \gamma V_\theta(s_{t+1})))^2] \tag{8}$$

2.4.2 Actor

For the actor, we want it to become greedy w.r.t to the soft Q-value, hence why we want to update the policy towards the exponential of the soft Q-value.

$$J_\pi(\phi) = \mathbb{E}_{s_t \sim D}[D_{KL}(\pi_\phi(\cdot|s_t))||\frac{\exp(\alpha Q_\theta(s_t, \cdot))}{Z_\theta(s_t)}] \tag{9}$$

However, rather than directly minimizing this term, we use somehting similiar. Again, we won't go into heavy detail here, but what we basically do is that we get a term which has the same gradient w.r.t. θ.[3]

$$J_\pi(\phi) = \mathbb{E}_{s \sim D} \mathbb{E}_{a \sim \pi_\phi(\cdot|s)}[\alpha \log \pi_\phi(a|s) - Q_\theta(s, a)] \tag{10}$$

By minimizing this term, we can learn the policy parameters. [3]

2.4.3 Entropy adjustment

The α parameter is called temperatur. A higher value of α encourages higher entropy in the policy, making it more exploratory, while a lower value of α makes the policy more deterministic and focused on exploiting the current knowledge. The choice of α determines the balance between exploration and exploitation. One can either use a fixed value or α can be learned by minimizing

$$\text{loss}(\alpha) = \mathbb{E}_{s_t \sim D}[\mathbb{E}_{a \sim \pi_\phi(\cdot|s_t)}[-\alpha \log \pi_\phi(a|s_t) - \alpha \bar{H}]] \tag{11}$$

3 Project

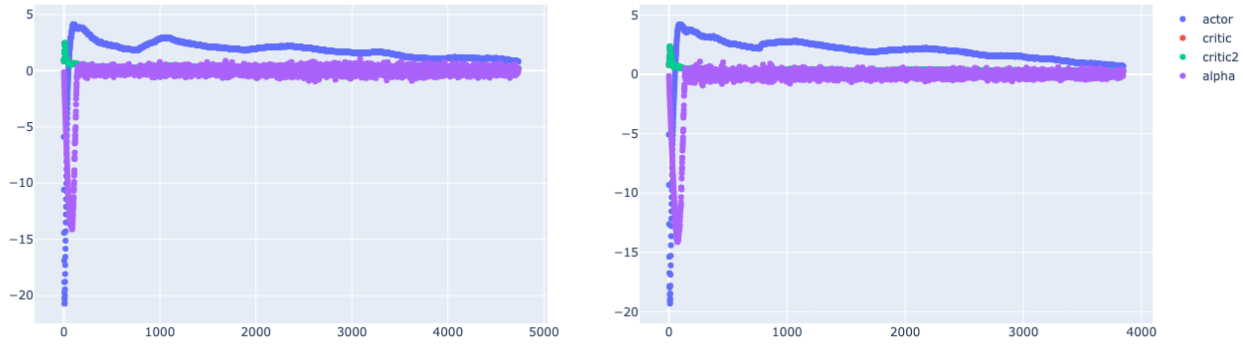
Having a deeper understanding of the SAC algorithm, we will have a closer look on the implementation and result I could observe while training and playing.

3.1 Training

The training is held simple: The agent starts (as suggested in the paper) playing against an opponent with mode "defending" for a sepcific amount of episodes. After that, the opponent changes its mode to either "shooting" or "normal" and we restart. The updates happen once we have enough samples to learn from in the buffer.

3.2 Losses

For the loss compuation, we used the Pytorch MSE-Loss (also referred to the squared L2 loss). Training for more episodes and even changing the reward parameters slightly still yielded the same shape of losses.



(a) agent1 (b) agent2

Figure 1: Loss behaviour of training

Note that the x-axis doesn't represent the episodes directly. I used a window function to calculate the mean for x (usually 400) amount of entries to save some space and to shorten the plot.

3.2.1 Interpretation of losses

- It is clear to see that especially at the beginning, we favor exploration, hence the peak in the entropy loss. However, our agent learns more about the environment and converges not exactly to zero but close, meaning that it becomes more and more deterministic. One could argue that it maybe returns to fast to zero, making the agent explore too less.
- The policy loss also decreases as we explore, but once the agent becomes better at solving the hockey game, it starts converging.
- The critics increase at the beginning, probably due to the fact that we don't have a stable reward income. Once the reward becomes stable, the value functions start to converge aswell.

3.3 Reward system

Generally, when working with an environment, you get the usual reward for taking an action in a specific state. It is neccessary to use this information. Thankfully, besides the usual reward, there were two more very interesting feedbacks given by the environment.

Reward	ClosenessToPuck	TouchPuck
Description	Reward based on how close to puck	Reward for touching the puck
Usage	Used to encourage the agent to approach the puck.	Reward agent when physically interacting with the puck.

PseudoCode	`reward += x * info[closeness]`
	`if closen.Old > closen.New:`
	`reward += y`
		`if not touched yet:`
		`reward -= punishment`
		`if touched:`
		`forget punishments,`
		`if touched happened under` t `steps:`
		`reward += extra_bit`

Table 1: Description and Usage of Reward Components

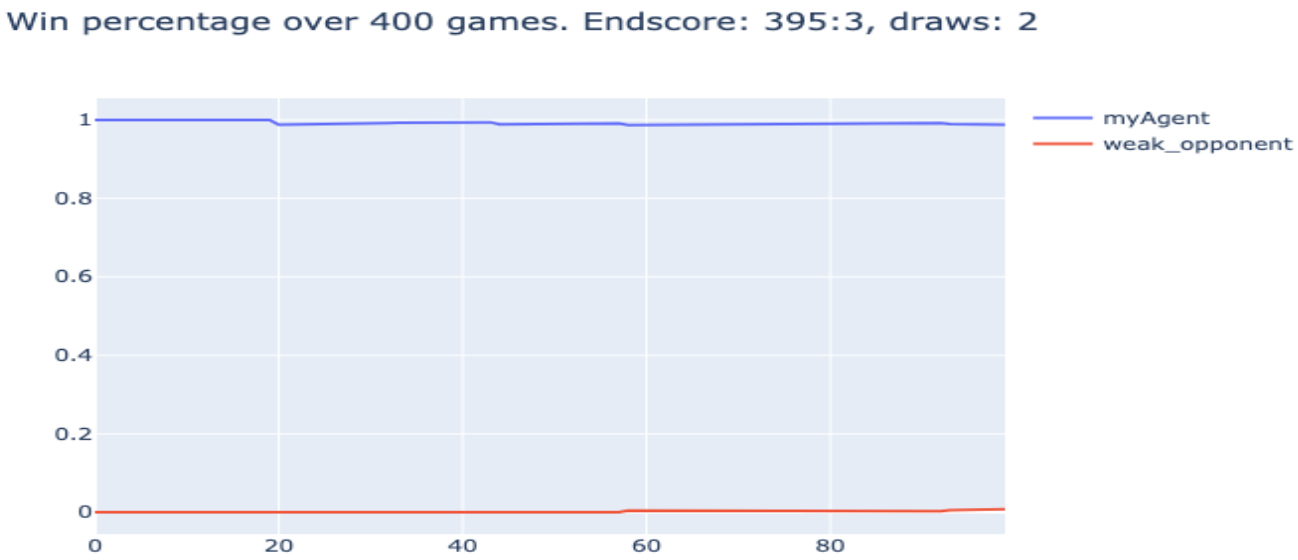


Figure 2: my agent vs. weak opponent in a tournament like setup

You can see that most of the times, the agent with the described reward system beats the weak opponent.

3.4 Conclusion

We have seen how the losses start to converge when we train the agent long enough and made reasonable statements on why they possibly behave like that. Furthermore, we observed that taking the extra information about reward into considerarion has a positive impact. Of course, it is important to test all hyperparameters, as some values will enhance or decrease the agents perfomance. However, even if those rewards were not given, it is worth to come up with own ideas on how to reward the agent regardless of the environments output. The task was to be able to beat the weak opponent of the environment, which was successfully done by the methods shown in this paper.

References

[1] Tuomas Haarnoja et al. “Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor”. In: *arXiv:1801.01290 [cs, stat]* (Aug. 8, 2018). arXiv: 1801.01290. URL: http://arxiv.org/abs/1801.01290 (visited on 08/10/2023).

[2] *martius-lab/laser-hockey-env: A simple laser-hockey gym environment for RL agents*. URL: https://github.com/martius-lab/laser-hockey-env (visited on 08/10/2023).

[3] Olivier Sigaud. “Soft Actor Critic”. In: (). URL: http://chronos.isir.upmc.fr/~sigaud/teach/sac.pdf.

[4] *Soft Actor-Critic — Spinning Up documentation*. URL: https://spinningup.openai.com/en/latest/algorithms/sac.html (visited on 08/10/2023).

[5] *Soft Actor-Critic (SAC) Agents - MATLAB & Simulink - MathWorks Deutschland*. URL: https://de.mathworks.com/help/reinforcement-learning/ug/sac-agents.html (visited on 08/10/2023).