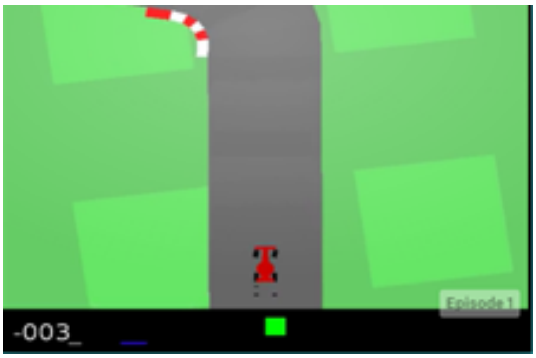
Final Assignment

Barak Hellman
305128977

Ava Bensoussan
324708775

1 Introduction

We chose the Car Racing Problem. A continuous control task to learn from pixels. A state consists of 96×96 pixels. There are six actions, we denote them *speed up*/*no speed up*, *brake*/*no brake* and *turn left*/*turn right*. The reward is -0.1 every frame and $+1000/N$ for every track tile visited, where N is the total number of tiles in track. The game is solved when the agent consistently gets 900+ points. The generated track is random every episode. An episode finishes when all tiles are visited.



We used the Soft Actor-Critic (SAC) algorithm [Haa+18] to solve the problem.

2 Background

2.1 Entropy-Regularized Reinforcement Learning

Entropy is a quantity which says how random a random variable is. 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

$$H(P) = \mathbb{E}_{x \sim P}[-logP(x)]$$

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

changes the RL problem to:

$$\pi^* = argmax_\pi \mathbb{E}_{\tau\sim\pi}[\sum_{t=0}^{\infty} \gamma^t (R(s_t, a_t, s_{t+1}) + \alpha H(\pi(\cdot|s_t)))]$$

where $\alpha > 0$ is the trade-off coefficient.
As a result, V^π and Q^π are changed to include the entropy bonuses from every timestep:

$$V^\pi(s) = \mathbb{E}_{\tau\sim\pi}[\sum_{t=0}^{\infty} \gamma^t (R(s_t, a_t, s_{t+1}) + \alpha H(\pi(\cdot|s_t)))|s_0 = s]$$

$$Q^\pi(s, a) = \mathbb{E}_{\tau\sim\pi}[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t, s_{t+1}) + \alpha \sum_{t=0}^{\infty} \gamma^t H(\pi(\cdot|s_t)))|s_0 = s, a_0 = a]$$

Maximum entropy reinforcement learning optimizes policies to maximize both the expected return and the expected entropy of the policy [Haa+18].

2.2 Soft Actor-Critic

The SAC algorithm incorporates three key ingredients [Haa+18]:

- Actor-critic architecture with separate policy and value function networks (like we saw in class)
- Off-policy formulation that enables reuse of previously collected data for efficiency
- Entropy maximization to encourage stability and exploration

The first version of SAC (2018) uses a fixed entropy parameter *alpha*, which turned out to be a very sensitive hyperparameter. Thus, the second version of SAC (2019) uses an update-able parameter *alpha*. In particular, *alpha* is updated by taking the gradient of the following objective function:

$$J_\alpha = \mathbb{E}[-\alpha log \pi_t(a_t|s_t; \alpha) - \alpha \cdot \overline{H}]$$

where $\overline{H}$ represents the desired minimum entropy, usually set to a zero vector.

3 Implementation

3.1 Actor-Critic Architecture

Note that all the actions of the car depend on its speed, the *speed up/no speed up* and *brake/no brake* actions affect the car's speed, and the *turn right* and *turn left* actions are affected by the car's speed (if the speed is too high, the car looses control when turning left or right). Therefore, we must know the car's speed every time the car performs an action. Since the observations of

the environment do not hold the car's speed, we must "remember" previous observations in order to compute the car's speed. Thus, we chose to use RNN networks, which have memory (their internal states) that allow us to compute the car's speed.
We created two RNN networks, one to approximate the policy (the actor) and one to approximate the Q-value function (the critic). The actor network consists of a preprocessing layer of size 64, a fully connected layer of size 20, an LSTM layer of size 40 and a fully connected output layer of size 32. The preprocessing layer processes the input image (reshapes, crops and converts it to grayscale). The critic network is similar to the actor network, but it has an additional fully connected layer of size 8, that comes after the preprocessing layer (for the actions). Both networks use the RELU activation function.
We initialized the actor learning rate, the critic learning rate and the alpha parameter (the trade-off coefficient) to 0.0005, and we used the Adam optimization algorithm for each of them.

3.2 Training

We trained the networks for approximately 100k iterations. In each iteration, we perform one step in the environment and then train the networks with three samples from the replay buffer.
We set the discount factor to 0.99.
In addition, In order to accelerate the training phase, we added a positive bonus reward to the *speed up* action and a negative bonus reward to the *brake* action.

4 Results

We present the average return of the computed policy vs the number of steps (figure 1), the loss of the actor's network (figure 2), the loss of the critic's network (figure 3) and the loss of the *alpha* parameter (figure 4).
The loss of the actor's network is computed as follows:

SL = RNN Sequence length

Sample a random minibatch of transitions

$$(s_1^i, a_1^i, r_1^i, \ldots, s_{SL}^i, a_{SL}^i, r_{SL}^i, s_{SL+1}^i) \sim D$$

for j = 1 to SL:

$$a'_j \sim \pi(\cdot|s_1^i, \ldots, s_j^i, \theta)$$
$$err_j^i = \alpha \cdot log[\pi(a'_j|s_1^i, \ldots, s_j^i, \theta)] - \min_{w \in \{w_1, w_2\}} Q(s_1^i, \ldots, s_j^i, a'_1, \ldots a'_j, w)$$
$$err^i = \sum_{j=1}^{SL} err_j^i$$

$$Loss(Actor) = mean(err^i)$$

The the loss of the critic's network is computed as follows:

SL = RNN Sequence length
Sample a random minibatch of transitions

$$(s_1^i, a_1^i, r_1^i, \ldots, s_{SL}^i, a_{SL}^i, r_{SL}^i, s_{SL+1}^i) \sim D$$

for j = 1 to SL:

$a'_{j+1} \sim \pi(\cdot|s_2^i, \ldots, s_{j+1}^i, \theta)$
$q_{j+1} = \min_{w^- \in \{w_1^-, w_2^-\}} Q(s_2^i, \ldots, s_{j+1}^i, a'_2, \ldots a'_{j+1}, w^-) - \alpha \cdot log[\pi(a'_{j+1}|s_2^i, \ldots, s_{j+1}^i, \theta)]$
$err_j^i = \frac{1}{2} \sum_{w \in \{w_1, w_2\}} (r_i + \gamma q_{j+1} - Q(s_1^i, \ldots, s_j^i, a_1, \ldots a_j, w))^2$
$err^i = \sum_{j=1}^{SL} err_j^i$
$Loss(Critic) = mean(err^i)$

The loss of the alpha parameter is computed as follows:
SL = RNN Sequence length
TH = Target-Entropy (set by default to: -DIM(action space) / 2)
Sample a random minibatch of transitions

$$(s_1^i, a_1^i, r_1^i, \ldots, s_{SL}^i, a_{SL}^i, r_{SL}^i, s_{SL+1}^i) \sim D$$

for j = 1 to SL:

$a'_j \sim \pi(\cdot|s_1^i, \ldots, s_j^i, \theta)$
$err_j^i = log(\alpha) \cdot [-log\pi(a'_j|s_1^i, \ldots, s_j^i, \theta) - TH]$
$err^i = \sum_{j=1}^{SL} err_j^i$
$Loss(Alpha) = mean(err^i)$

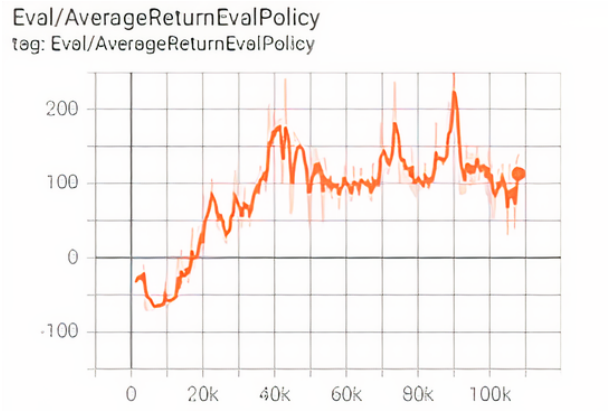


Figure 1: Average return

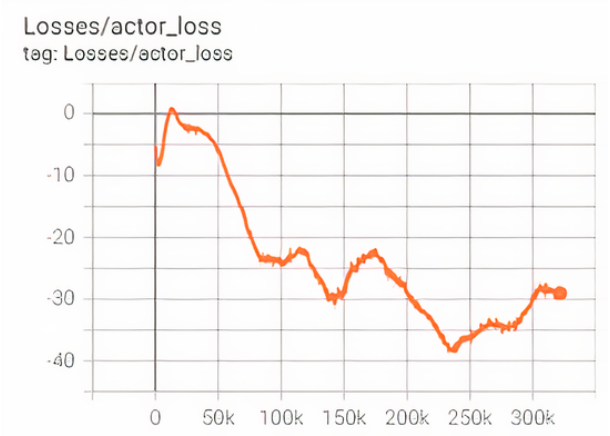


Figure 2: Actor network loss

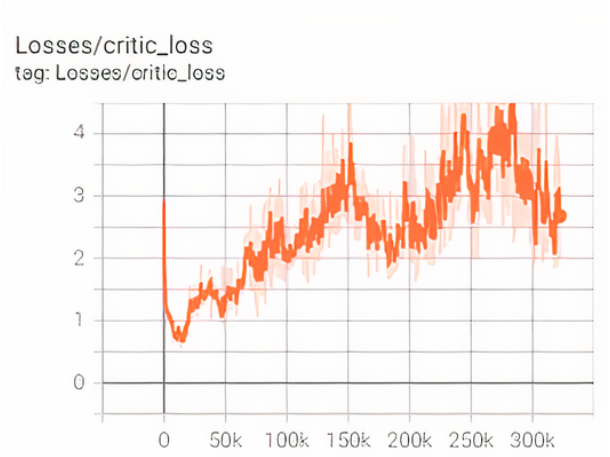


Figure 3: Critic network loss

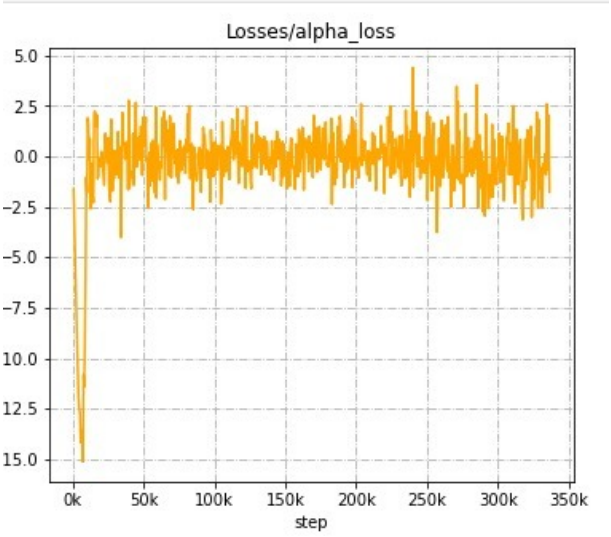


Figure 4: Alpha parameter loss

References

[Haa+18] Tuomas Haarnoja et al. "Soft actor-critic algorithms and applications". In: *arXiv preprint arXiv:1812.05905* (2018).