Soft-Actor Reinforcement Learning Training Tests and Guides with TensorFlow

Burak Toy

November 5, 2025

Contents

1 Soft Actor Critic Agent in TensorFlow

This section will be filled later.

1.0.1 Actor Loss

Actor loss is defined as in the Equation 1.

$$J_\pi(\phi) = E_{s_t \sim D}\left[E_{a_t \sim \pi_\theta}\left[\alpha \log(\pi_\phi(a_t|s_t)) - Q_\theta(s_t, a_t)\right]\right] \tag{1}$$

2 Mountain Car Continuous Environment

MountainCarContinuous-v0 is an environment, provided by OpenAI Gym. The environment is fixed, where the agent tries to reach to the flag while giving acceleration command. The environment can be seen in Figure 1.

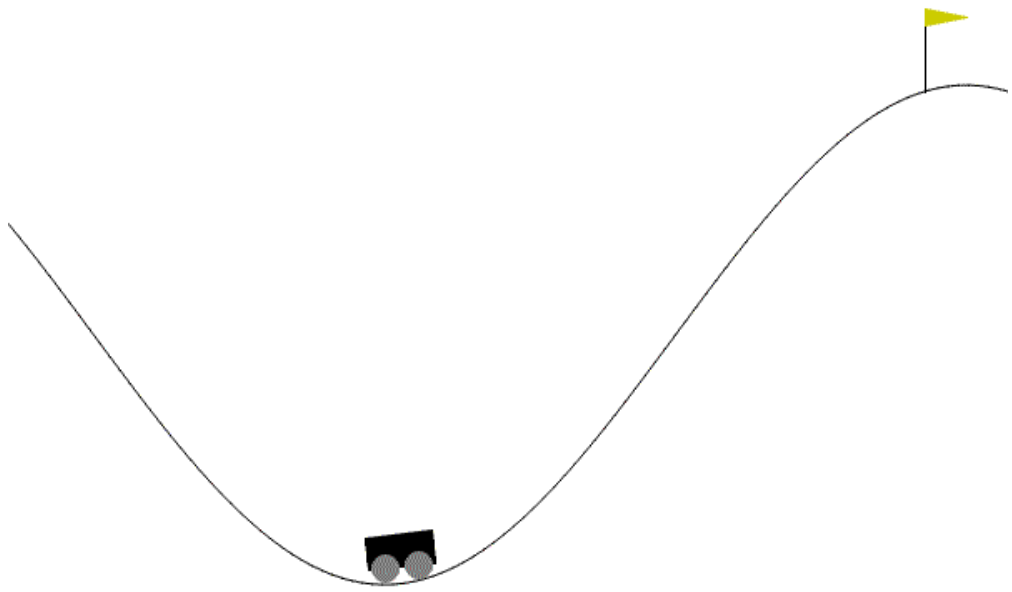

Figure 1: Mountain Car V0 Environment by Gymnasium

Agent is always initialized in the local minima point in the environment, with no velocity. State is composed of position of the agent in x-axis, and velocity of the agent. Thus, state space size is 2, whereas action space size is 1.

Reward function is defined $-0.1 * (action)^2$ in each step, and agent gets a reward of 100 if it hits target, which is the only termination condition.

Collisions are defined inelastic and non-terminating, where velocity of the agent gets damped to zero upon collision. Velocity is clipped at ± 0.07.

2.1 Training with High Minimum Buffer Size and Consecutive Actions Trick

In this training, some important parameters were as in the below.

- Minimum Buffer Size = 4000
- Number of Consecutive Actions = 4
- Initial Temperature (α) = 1.0
- Batch Size = 512
- Learning Rate = 0.0003
- $\tau = 0.01$
- $\gamma = 0.99$

2.1.1 Consecutive Actions Trick

This trick is designed to increase the odds of getting a hit reward while filling the replay buffer, so that value functions can easily back-propagate from the high reward state. Success of this trick will be shown later in this report.

Note that it is only applicable while filling up the replay buffer to minimum buffer size. After buffer is filled enough and training is initiated, this trick gets cancelled.

2.1.2 Loss Results

Average actor loss, average critic loss (sum of critic loss 1 and 2), and average temperature loss can be seen in Figures 2, 3 and 4, respectively. All loss metrices show good convergence and stable behavior. Note that negative loss with high absolute value is something desired.

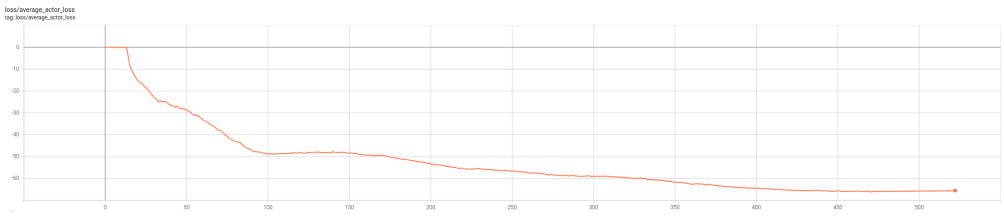

Figure 2: Average Actor Loss

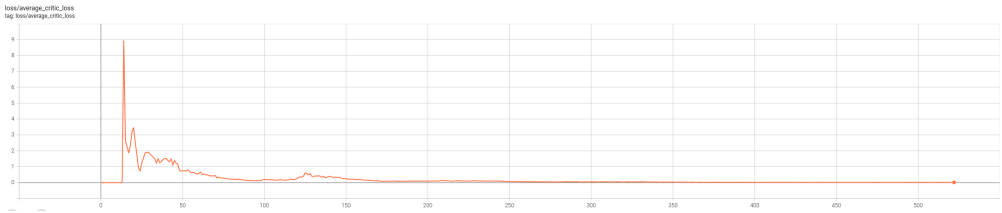

Figure 3: Average Critic Loss

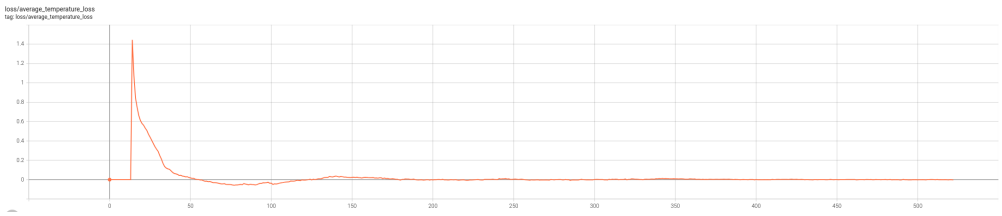

Figure 4: Average Temperature Loss

2.1.3 Reward Profile

The average reward of all episodes are given in Figure 5. It can be seen that after batching to replay buffer is completed, the agent shows an increase in average reward for the episodes. As training progresses, agent showed minor improvements in rewards, and achieving solution for the task. Note that training began on the episode that is marked with black arrow. It is clear that as the time goes, agent becomes more precise and accurate.

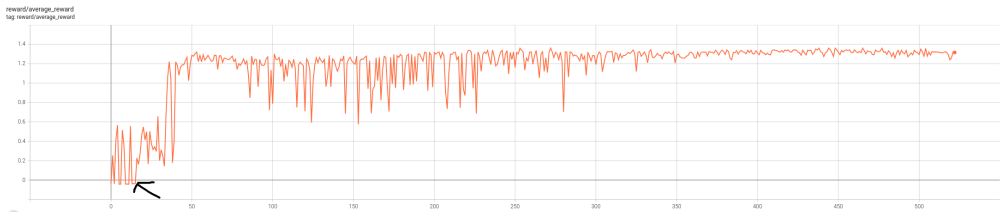

Figure 5: Average Reward

2.1.4 Temperature Profile

The average temperature values are given in Figure 6. It is clear that agent started with high temperature to force exploration, and as time goes, it becomes more greedy. There are indeed oscillations, which in fact helped agent in getting out of local minimas and finding more optimal solutions.

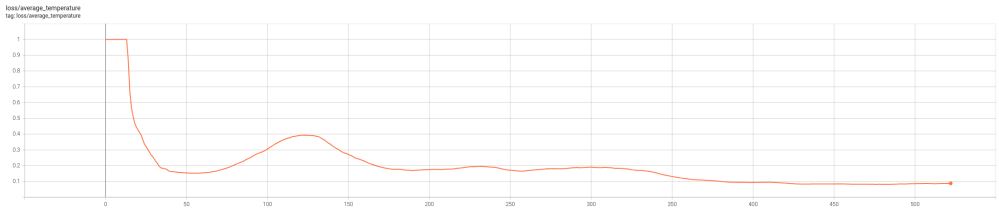

Figure 6: Average Temperature

2.1.5 Final Comments

Since agent were able to get replay data from buffer about actually reaching to the reward state, it was able to backpropagate value function, and ended up with learning the environment. Agent executed almost perfect actions in consecutive tries, and showed that it learned the almost optimal policy.

2.2 Training with High Minimum Buffer Size and No Consecutive Actions Trick

In this training, some important parameters were as in the below.

- Minimum Buffer Size = 4000
- Number of Consecutive Actions = 0
- Initial Temperature (α) = 1.0
- Batch Size = 512
- Learning Rate = 0.0003
- $\tau = 0.01$
- $\gamma = 0.99$

2.2.1 Loss Results

Average actor loss, average critic loss (sum of critic loss 1 and 2), and average temperature loss can be seen in Figures 7, 8 and 9, respectively. All loss metrices show good convergence and stable behavior.

However, average actor loss was expected to be a high negative number, but it converges to around zero. To understand why this is an issue, reader should refer to Section 1.0.1. In summary, loss value represents the negative Q value of the state and action tuple, and since $Q(s, a)$ is needed to be high (this is because the target reward 100 is also a high number), the loss is expected to be negativly high. In this case, loss is a small negative number, which means that agent did not learn the fact that it can reach to a target reward of 100.

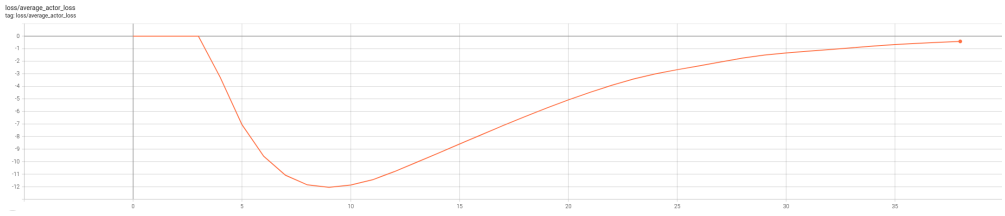

Figure 7: Average Actor Loss

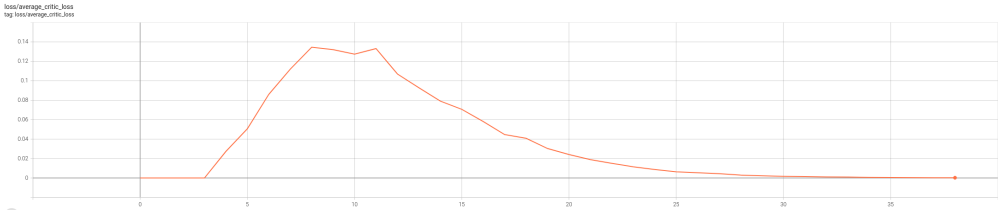

Figure 8: Average Critic Loss

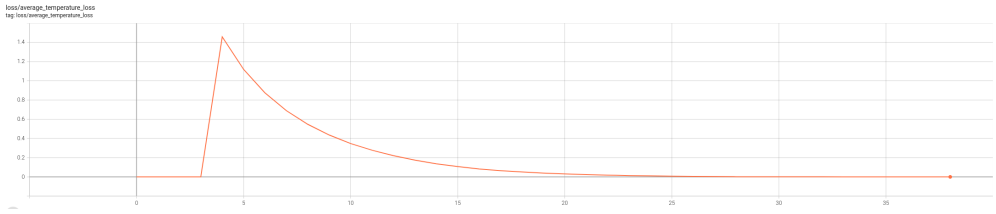

Figure 9: Average Temperature Loss

2.2.2 Reward Profile

The average reward of all episodes are given in Figure 10. It can be seen that after batching to replay buffer is completed, the agent shows an increase in average reward for the episodes. As training progresses, agent showed minor improvements in rewards. However, the average rewards converged to zero, instead of around 95. This was because agent found a local minima, and decided to stay at the initial spawn position. Recall that reward function was $-0.1 * (action)^2$, and agent figured out that it could dodge high penalties by simply providing a very low action output.

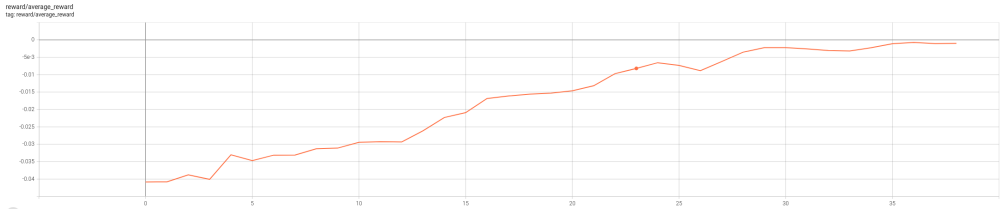

Figure 10: Average Reward

2.2.3 Temperature Profile

The average temperature values are given in Figure 11. It is clear that agent started with high temperature to force exploration, but temperature fastly decayed. This proves that agent become greedy without performing any needed explorations.

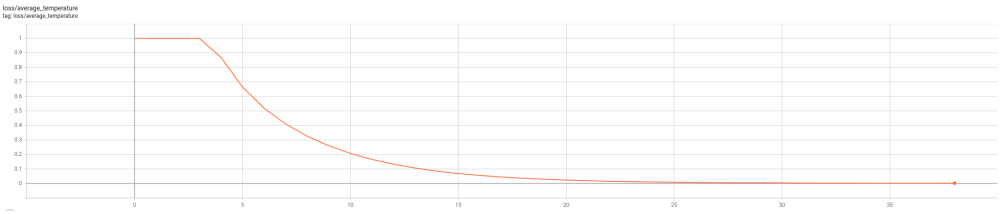

Figure 11: Average Temperature

2.2.4 Final Comments

Agent did not have enough data in its replay buffer that ended with a target hit reward, thus were not able to actually learn to reach to the target. At the end, it decided that best action was not doing anything, and it always stayed in the initial spawn point, minimizing the penalty without getting and reward. Note that agent were not able to understand that there was a reward.

3 3DOF Point Mass

In this training, a 3D environment is created, with x-y dimensions 1000 m to 1000 m, and maximum allowed height was also 1000 m. Agent was allowed to hit to the edges without episode termination, but its velocity was damped to 0 in case of bounce. Maximum allowed velocity was 10 meters/second. Agent mass was selected as 1 kg, to make calculations easier.

Both target and agent gets initialized in random positions, and the state vector is composed of agent position, velocity, and target position.

3.1 Reward Analysis

For the 3DOF point mass task in plain environment, where agent and target were initialized at random points, reward function were tailored as given below.

```
def giveReward(
  action: floatMatrix,
  targetHit: bool,
  missilePosition: floatMatrix,
  missileVelocity: floatMatrix,
  targetPosition: floatMatrix,
  distanceBefore: float,
  isBounced: bool,
```

```
) -> float:

    # --- Distance and direction ---
    direction_to_target = targetPosition - missilePosition
    distance = np.linalg.norm(direction_to_target) + 1e-9
    direction_unit = direction_to_target / distance

    # --- Normalize action to get acceleration direction ---
    action_norm = np.linalg.norm(action) / (np.sqrt(3) * 1.0)  # Normalize by max possib

    # --- Distance-based reward component ---
    distance_delta: float = (distanceBefore - distance) / np.sqrt( (environment.xDim **2

    # --- Velocity alignment (cosine similarity) ---
    vel_norm = np.linalg.norm(missileVelocity) + 1e-9
    velocity_alignment = float(np.dot(missileVelocity.T, direction_unit) / vel_norm)  #

    # --- Reward calculation ---
    reward: float = -PENALTY_PER_STEP  # Base time step penalty
    reward = -REWARD_ACTION_PENALTY_COEFF * ((action_norm * 2) ** 2)  # Small step penal
    reward += REWARD_DISTANCE_DIFF_COEFF * distance_delta  # Reward for reducing distanc
    reward += REWARD_VELOCITY_PROJ_COEFF * (velocity_alignment * (vel_norm / 10.0))  # R

    if isBounced:
        reward -= PENALTY_BOUNCE_COEFF  # Penalty for bouncing off boundaries

    if targetHit:
        reward += REWARD_HIT_BONUS  # Reward for hitting the target

    return reward
```

Different coefficients for reward scalers were tested, and it was clear that the learning ability & process of the agent were highly dependent on the small changes in reward functions. This shows that creating the perfect reward function is needed, in order to train an agent that performs complicated tasks in complex environments, such as 6DOF terrain.

The success rate for the trains with different params are given in the Figure 12. Note that an episode is considered to be successfull, if agent hits the target without bouncing from the edges, and under 1000 iterations with $\Delta t = 1$.

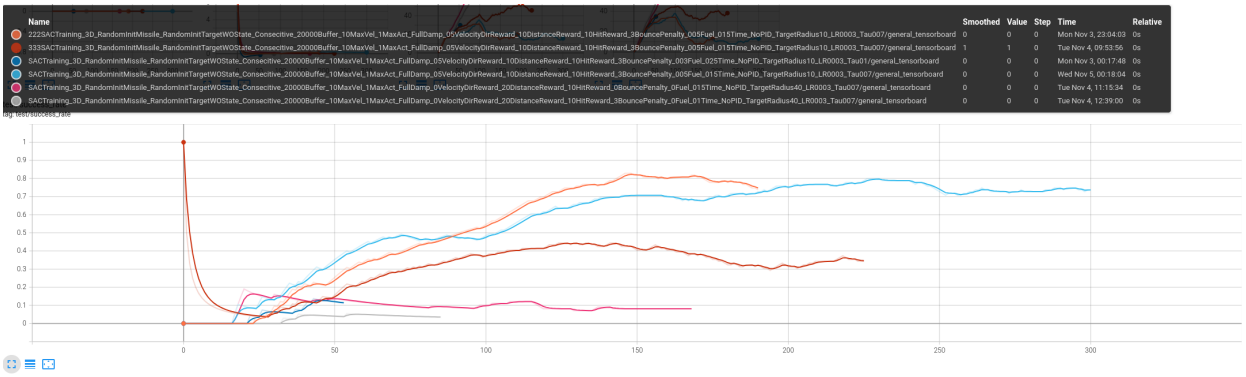

Figure 12: Success Rates For Different Reward Coefficents

It can be seen that very small changes in coefficients can cause significant changes in the success of the training process. The best results is achieved with the parameters given below.

```
PENALTY_PER_STEP: float = 0.15
REWARD_ACTION_PENALTY_COEFF: float = 0.05
REWARD_DISTANCE_DIFF_COEFF: float = 10.0
PENALTY_BOUNCE_COEFF: float = 3.0
REWARD_VELOCITY_PROJ_COEFF: float = 0.5
REWARD_HIT_BONUS: float = 10.0
```

3.2 Behavior Cloning Trick

As discussed in Section 2, it is highly important for agent to receive a target reward in early buffering stage, so that it can aim for that target in exploration phase. However, in a big environment such that agent experienced in this section, it is very hard to hit that target, since environment is very big. Thus, behavior cloning method is proposed and applied.

In this method, the agent policy network (not the critic networks) is pre-trained with PID outputs, so that policy can have an insight on what is happening in environment. The training principle of behavior cloning method is given in Figure 13.

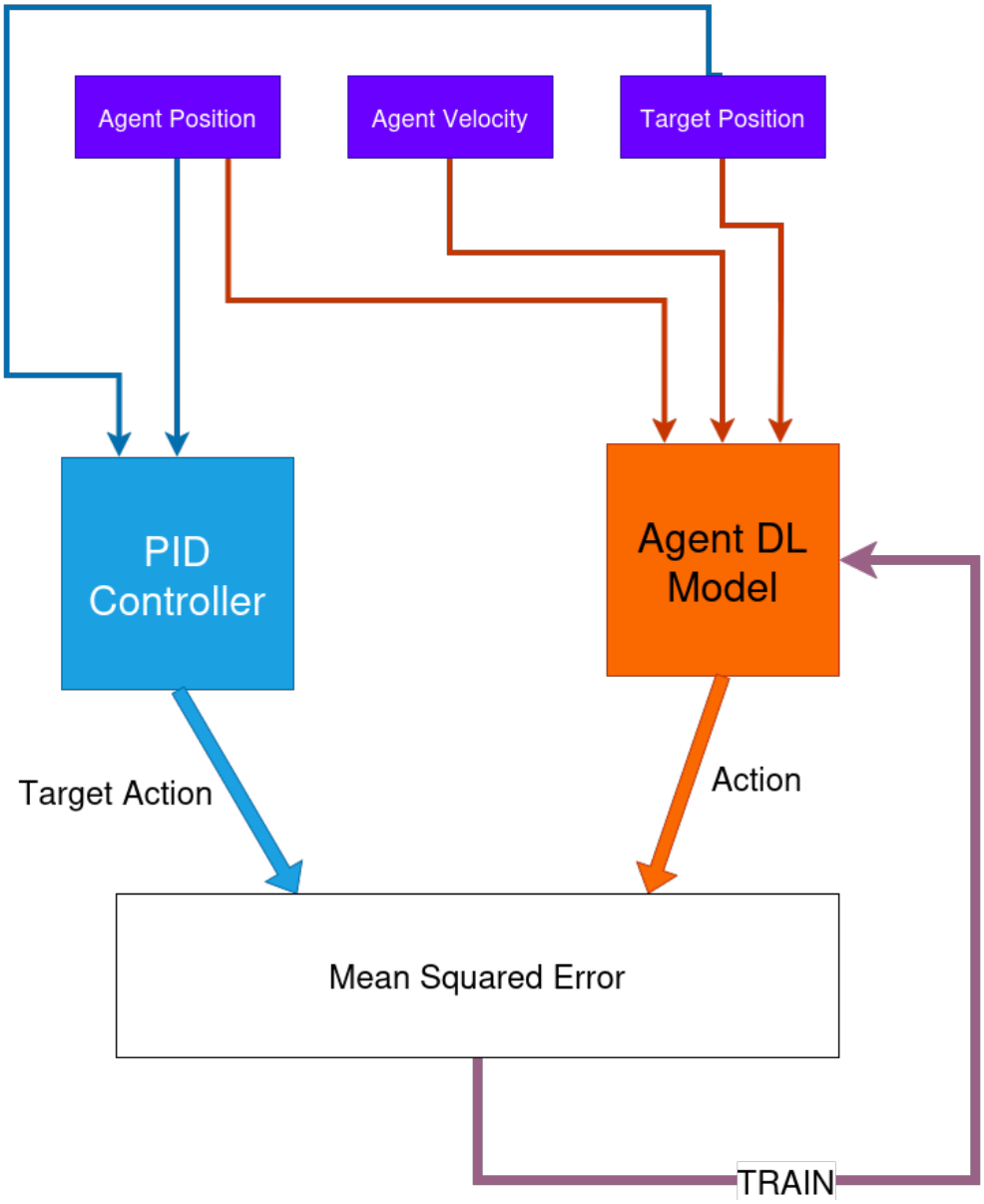

Figure 13: Behavior Cloning Training Chart

This method is applied with the optimal reward coefficent given above, and a clear improvement in the performance was seen. The success rates for the trains with behavior cloning on and off can be seen in Figure 14. The green results are for behavior cloning for 30k steps, the dark blue for 10k steps, and the light blue is for no behavior cloning. It is clear that behavior cloning increases the success of the agent.

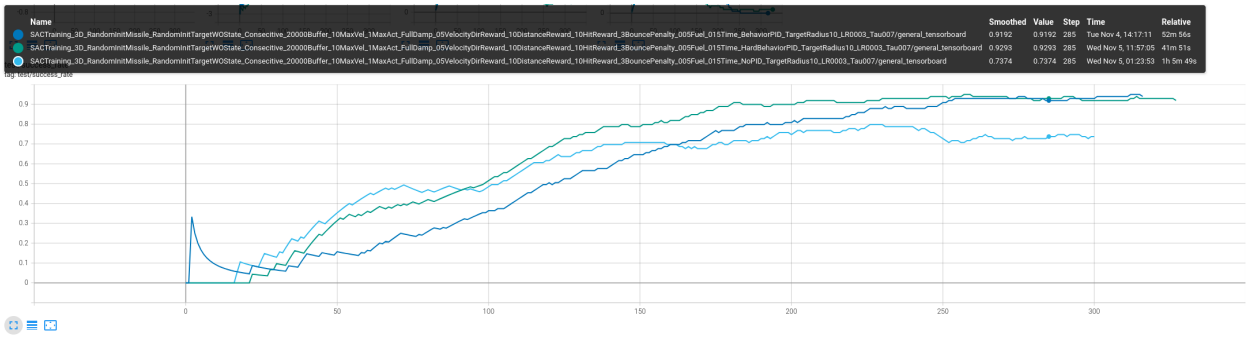
Name
Smoothed
Value
Step
Time
Relative
SACTraining_3D_RandomInitMissile_RandomInitTargetWOState_Consecitive_20000Buffer_10MaxVel_1MaxAct_FullDamp_05VelocityDirReward_10DistanceReward_10HitReward_3BouncePenalty_005Fuel_015Time_BehaviorPID_TargetRadius10_LR0003_Tau007/general_tensorboard
0.9192
0.9192
285
Tue Nov 4, 14:17:11
52m 56s
SACTraining_3D_RandomInitMissile_RandomInitTargetWOState_Consecitive_20000Buffer_10MaxVel_1MaxAct_FullDamp_05VelocityDirReward_10DistanceReward_10HitReward_3BouncePenalty_005Fuel_015Time_HardBehaviorPID_TargetRadius10_LR0003_Tau007/general_tensorboard
0.9293
0.9293
285
Wed Nov 5, 11:57:05
41m 51s
SACTraining_3D_RandomInitMissile_RandomInitTargetWOState_Consecitive_20000Buffer_10MaxVel_1MaxAct_FullDamp_05VelocityDirReward_10DistanceReward_10HitReward_3BouncePenalty_005Fuel_015Time_NoPID_TargetRadius10_LR0003_Tau007/general_tensorboard
0.7374
0.7374
285
Wed Nov 5, 01:23:53
1h 5m 49s
tag: test/success_rate
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
50
100
150
200
250
300

Figure 14