# Deep Reinforcement Learning for the Unity “Reacher” 20-Agent Environment

## 1. Introduction

The current report aims at solving the **Unity environment "Reacher"** as described in the accompanying README file.

This is a multi-agent environment with 20 agents. However, it is **not a multi-agent game** as each arm is trying to reach its own target, no agent affects the reward of any other agent, and no agent blocks, helps, or competes with another.

Here we have 20 agents in **the same environment instance**, each receiving its own observation and reward, all controlled by a **shared policy** and contributing to the gradients calculation. In spirit, it behaves more like **A2C-style parallelism,** although in the latter many environment instances are run in parallel.

It is a distributed training setup in which multiple agents **contribute experience to a shared learning process.**

The current solution implements the Deep Deterministic Policy Gradient (**DDPG**) agent on the basis that the actions (torques applied to the joints) are **continuous** and the Reacher environment has **stationary dynamics**, smooth reward structure, and **no adversarial agents**.

## 2. Methodology

### 1) DDPG agent

The agent trains a **shared deterministic actor** with a **share critic**. In this setup, the agents contribute independent experience to the shared replay buffer through the shared actor, while all learning ( actor, critic, and target updates ) is performed on a single shared set of networks.

### 2) Learning from the Environment Reward

In this environment, **20 identical agents** control double-jointed arms to reach moving target locations.
At each time step $t$, agent $i$receives a reward:

$$r_t^{(i)} = \begin{cases} +0.1 & \text{if the agent's hand is inside the target region} \\ 0 & \text{otherwise} \end{cases}$$

The objective is for the agents to maintain their hands within the target region for as many time steps as possible, thereby maximising cumulative reward.

**The learning process will use this reward accordingly.**

## 3) Replay buffer

For each agent $i$at time $t$, transitions are stored in the stored in the replay buffer.

$$(s_t^{(i)},\ a_t^{(i)},\ r_t^{(i)},\ s_{t+1}^{(i)},\ d_t^{(i)})$$

Where $s_t^{(i)}$is the state, $a_t^{(i)}$ is the action, $r_t^{(i)}$ is the reward, $s_{t+1}^{(i)}$ is the next state, and $d_t^{(i)}$ is the done status (terminal state or not).

$$\begin{aligned} s_t^{(i)} &\in \mathbb{R}^{33} \\ a_t^{(i)} &\in \mathbb{R}^{4} \\ r_t^{(i)} &\in \mathbb{R} \\ d_t^{(i)} &\in \{0, 1\} \end{aligned}$$

The replay buffer makes the training off-policy, decorrelated (random mini batches from the replay buffer almost make the training data **independent and identically distributed**), stable, and sample efficient. Without replay, the critic would chase its own moving target and explode.

## 4) Search for the best hyperparameters via grid search

The grid of parameters :

- Hidden layer's dimensions: (64, 64), (128,128), (256,256),
- optimizer:  Adam,
- learning rate : 1e-4, 3e-4, 5e-4,
- buffer_size: 50000,
- batch_size: 64, 128, 256, 500,
- exploration_noise_ratio: 0.05, 0.1, 0.2,
- n_warmup_batches: 5, 10,

- target_update_every: 2, 5, 10,
- tau: 0.001, 0.005, 0.01.

The experiment begins with a hyperparameter search conducted over 10 random trials. The trial achieving the highest performance in the final evaluation is selected as the best configuration.

These optimal hyperparameters are then used for multiple runs with 10 seeds (seeds 33,41,66,39,8,77,21,20,44,22).
Among these runs, the final reported model is the online actor network from the run that reaches the early stopping criterion in the smallest number of training episodes.
The objective is to achieve convergence in fewer than 163 training episodes.

## 5) Network architectures (actor + critic)

Note that the hidden dimensions in the below networks are determined by the search for the best parameters.

### *a) Shared actor FCDP*

It is the Fully Connected Deterministic Policy (FCDP).

$$\pi_\theta : \mathbb{R}^{33} \longrightarrow [-1,1]^4$$

Architecture:

- Fully-connected Multi Layer Perceptron (MLP) with 2 hidden layers (256,256).
- ReLU hidden activations.
- Tanh output to land in [−1,1]. Although the network output is already bounded to [-1,1], clipping is added as a safety.

$$z_1 = W_1 s + b_1, \quad h^{(1)} = \mathrm{ReLU}(z_1)$$
$$z_2 = W_2 h^{(1)} + b_2, \quad h^{(2)} = \mathrm{ReLU}(z_2)$$
$$z_3 = W_3 h^{(2)} + b_3, \quad a = \tanh(z_3)$$

So, per agent *I* and step *t*:

$$a_t^{(i)} = \pi_\theta(s_t^{(i)})$$

### *b) Shared critic FCQV*

For each agent, state s and action a, the critic estimates the **expected return**:

$$Q_\phi : (\mathbb{R}^{33},\ \mathbb{R}^{4}) \longrightarrow \mathbb{R}$$

Architecture:

- MLP with 2 hidden layers (256,256).
- State-action input ($s \in \mathbb{R}^{33}$, $a \in \mathbb{R}^{4}$).
- One fully connected layer applied to the state *s*. The encoded state is then concatenated with the action *a*, followed by additional fully connected layers with no output activation, allowing unbounded value estimates.

$$z_1 = W_s\, s + b_s, \qquad h^{(1)} = \mathrm{ReLU}(z_1)$$

$$x = \begin{bmatrix} h^{(1)} \\ a \end{bmatrix} \in \mathbb{R}^{h_1+4}$$

$$z_2 = W_{sa}\, x + b_{sa}, \qquad h^{(2)} = \mathrm{ReLU}(z_2)$$

$$z_3 = W_q\, h^{(2)} + b_q, \qquad Q_\phi(s,a) = z_3$$

- ReLU activation functions on all hidden layers.
- A final linear fully connected layer with 1 output, producing the action-value scalar.

### *c) Target networks initialization and update*

The Temporal Difference (TD) targets are slow-moving copies (parameters are not learned) of the shared actor and the central critic. If the same rapidly-changing networks were used to build the TD targets, the targets would move every gradient step and the critic would often diverge.

$$\text{target actor } \pi_{\theta^-}$$

$$\text{target critic } Q_{\phi^-}$$

At the start, a hard copy is initialised.

$$\theta^- \leftarrow \theta, \qquad \phi^- \leftarrow \phi$$

Every “**target_update_every**” environment steps (Hyperparameter, which best value was found to be 5 ), using a smoothing factor “**tau**” (Hyperparameter, which best value was found to be 0.01), a **soft update** (**Polyak** averaging / exponential moving average) is applied to the copies :

$$\theta^{-} \leftarrow (1-\tau)\theta^{-} + \tau\theta$$
$$\phi^{-} \leftarrow (1-\tau)\phi^{-} + \tau\phi$$

Only 1% of the new online parameters are blended into the target at each update; 99% of the old target is retained.

Note that because the actor and critic are shared between agents, the target actor and critic are also shared:

- one target actor $\pi_{\theta^-}$
- one target critic $Q_{\phi^-}$

## 6) Training process

### *a) Training actions (Gaussian noise injection)*

The exploration noise is used to generate the training data. The noisy transitions are then fed into the replay buffer. This avoids on-policy greedy trajectories and prevents the buffer from collapsing into local optima very quickly.

At every training step, Gaussian noise is added to the deterministic actor output to produce the executed noisy action:

$$\tilde{a}_t^{(i)} = \mathrm{clip}\big(\pi_\theta(s_t^{(i)}) + \varepsilon_t,\ [-1,1]^2\big)$$
$$\varepsilon_t \sim \mathcal{N}(0,\sigma^2)$$
$$\sigma = \mathrm{exploration\_noise\_ratio} \times a_{\max}$$

The exploration noise is a hyper parameter in the grid search. The best value was found to be 0.05. The environment bounds are $a_{max}$ = 1.

Clipping is required to keep the value within the environment boundaries.

$$\tilde{a}_t = \begin{cases} -1 & \text{if } a_t + \varepsilon_t < -1 \\ a_t + \varepsilon_t & \text{if } -1 \le a_t + \varepsilon_t \le 1 \\ 1 & \text{if } a_t + \varepsilon_t > 1 \end{cases}$$

The exploration noise is not used in the optimisation process to avoid polluting gradients.

From the per-step noisy actions, a step-level ratio "**ratio_noise_injected**" is calculated to measure how much exploration is actually happening relative to the agent's intended action. It is the relative size of the injected noise.

$$\text{ratio\_noise\_injected}(t) = \frac{1}{d}\sum_{j=1}^{d}\left|\frac{a_{t,j} - \tilde{a}_{t,j}}{h_j - \ell_j + \delta}\right|$$

*t* = Environment step at which the action is taken.
*d* : Dimension of the action vector (number of action components). In this environment, 4 per agent.

$a_t \in \mathbb{R}^d$: Greedy action produced by the current online actor.

$\tilde{a}_t \in \mathbb{R}^d$: Executed action after noise injection and clipping.

$\ell \in \mathbb{R}^d$ : Lower bound of the action space (-1)

$h \in \mathbb{R}^d$: Upper bound of the action space (1).

$\delta$: Small positive constant for numerical stability (1e-12).

In the log file, the ratio "**mean_exploration_ratio_over_100_episodes_from_training**" is an accumulation this per-step ratio over the episode :

$$\rho_{\text{episode}} = \frac{1}{T}\sum_{t=1}^{T}\text{ratio\_noise\_injected}(t)$$

*T* = number of environment steps in the episode.

*b) greedy actions*

They are the actions directly output by the actor.

Evaluation uses a greedy (no-noise) policy every episode.

$$a_t^{(i)} = \text{clip}(\pi_\theta(s_t^{(i)}), [-1, 1]^2)$$

Greedy actions are also directly used in the optimisation.

### c) *Online data collection*

Transitions are collected online from all agents  (with noisy actions) during training, into the replay buffer. As soon as any one of the 20 agents is done during environment interaction, the step loop breaks, ending the episode, and then the next episode starts with a fresh reset for all 20 agents :

$$\left(s_t,\ a_t,\ r_t,\ s_{t+1},\ d_t\right)$$

### d) *Optimisation*

Optimisation starts after the size of the replay buffer D has reached the below threshold (warm-up):

$$|\mathcal{D}| > (\text{batch\_size}) \times (\text{n\_warmup\_batches})$$
$$\mathcal{D} = \left\{\left(s_t,\ a_t,\ r_t,\ s_{t+1},\ d_t\right)\right\}$$

"**batch_size**" and "**n_warmup_batches**" are hyperparameters , which best values are found to be 500 and 5. So the warmup threshold is 2500 transitions.

Parameter optimisation is interleaved with environment interaction during online data collection, with one **actor–critic update** performed **at every environment step** using **minibatch** samples from the replay buffer.

Target networks are updated according to hyperparameter "**target_update_every**".

During optimisation, **at every environment step**, a minibatch is sampled randomly from the replay and provides the following array :

$$s \in \mathbb{R}^{B\times 33}$$
$$a \in \mathbb{R}^{B\times 4}$$
$$r \in \mathbb{R}^{B\times 1}$$
$$s' \in \mathbb{R}^{B\times 33}$$
$$d \in \mathbb{R}^{B\times 1}, \qquad d \in \{0,1\}$$
$$(s_t, a_t, r_t, s_{t+1}, d_t)\ \in\ \mathbb{R}^{33} \times \mathbb{R}^{4} \times \mathbb{R} \times \mathbb{R}^{33} \times \{0,1\}$$

This minibatch is used in the networks update described below. This optimisation is called **off-policy pattern**: **collect online, optimise offline** (minibatch from replay).

### e) *Online Critic update*

The next actions are computed using the target actor :

$$a' = \pi_{\theta^-}(s') \quad a' \in \mathbb{R}^{B\times 4}$$

If the online actor $\pi_\theta$ was used instead of the target one $\pi_\theta -$, then while training the critic, the policy would change, the value function would change, and the TD target would chase a moving object.

Next, the critic target is output:

$$Q_{\phi^-}(s', a') \in \mathbb{R}^{B\times 1}$$

Then TD target is computed:

$$y = r + \gamma\, Q_{\phi^-}(s', a')\,(1 - d)$$

Gamma is set to 0.99.

Next, the online critic prediction is calculated.

$$\hat{Q} = Q_\phi(s, a) \quad \hat{Q} \in \mathbb{R}^{B\times 1}$$

Then, the critic loss is output:

$$\mathcal{L}_{\text{critic}} = \frac{1}{2}\,\mathbb{E}_{(s,a,r,s',d)}\left[(\hat{Q} - y)^2\right]$$

The old gradients of the loss are cleared from memory before the new ones are computed with respect to the critic parameters via backpropagation. Note that the TD target is detached so that gradients from the loss do not propagate into the target network.

$$\nabla_\phi \mathcal{L}_{\text{critic}}(\phi)$$

Then, the parameters are updated via Adam update rule.

$$\phi \leftarrow \phi - \alpha\ \text{Adam}(\nabla_\phi\, \mathcal{L}_{\text{critic}}(\phi))$$

Alpha $\alpha$ is the learning rate. It is a hyperparameter. Its best value is found to be 0.0005.

### f) *Online Actor update*

It follows the critic update.

First, we compute the current action for each agent with the shared online actor.

$$a' = \pi_{\theta^-}(s') \quad a' \in \mathbb{R}^{B\times 4}$$

Then, we evaluate the online critic at those actions.

$$Q_\phi(s, a_\theta) \in \mathbb{R}^{B\times 1}$$

Note that no “stopgrad” is applied in the loss computation.

Then the actor loss is computed for the minibatch (using an average over the $B$samples):

$$\mathcal{L}_{\text{actor}}(\theta) = -\frac{1}{B}\sum_{i=1}^{B} Q_\phi(s_i, \pi_\theta(s_i))$$

The parameters are then updated using backpropagation on the loss and the Adam update rule.

### g) *Early stopping*

Training stops when any of these triggers:

- The goal performance is achieved when the 100-episode moving average (MA) of the max-over-agents episodic return score exceeds +30.
- The wall-clock elapsed time exceeds 180 minutes.
- The total number of episodes exceeds 200.

### h) *Evaluation*

#### i. Online evaluation during training

After *every* training episode, exactly 1 greedy episode (no noise) is run using the shared online actor. Each agent *i* accumulates its own undiscounted return:

$$R^{(i)} = \sum_{t=0}^{T-1} r_t^{(i)}$$

Then the episode score is the mean over agents and is stored.

$$S = \frac{1}{20}\sum_{i=1}^{20} R^{(i)}$$

Then a 100-episode MA of those results is computed.

**This MA is used for early stopping if it reaches or exceeds +30 and if there are at least 100 values in the MA calculation.**

This MA is used in the plot together with the episodic return score.

### ii. Final evaluation

After training finishes, the final model is evaluated over 100 episodes without noise.

For each episode, the same maximum score is computed.
The **mean and standard deviation** of these 100 episode scores are then calculated, providing a low-variance and reliable estimate of the policy's performance.

This mean performance is used as the comparison metric:

- first, to select the best hyperparameters across 10 trials, and
- once the optimal hyperparameters are fixed, to identify the best overall performance across 10 seeds.

#### *a) Training sequence*

Per training episode:

- Run one episode in Unity environment using noisy actions from the shared actor.
- Store each step into replay .
- Once warm-up is passed, do minibatch updates every step.
- Update targets every "target_update_every" steps.
- End episode when Unity environment signals done for anyone of the 20 agents.
- Evaluate immediately**:** run 1 greedy episode (no noise), calculate the episodic return, and record that evaluation score.
- Save an actor checkpoint online_policy_model.<episode>.tar.

After training ends:

- Run final evaluation of 100 greedy episodes with the final model, return mean ± std.

### *b) Network Usage and Reward Flow Across Training Phases*

The below table highlights that explorative actions are used for the data collection while greedy actions are used by the optimisation and evaluation phases.

| Phase | Shared Online Actor | Online Critic | Target Actor | Target Critic | Minibatch from replay |
|---|---|---|---|---|---|
| Data collection (explorative strategy) into replay buffer | X | | | | |
| Optimisation | X (greedy strategy) | X | X | X | X (explorative strategy) |
| Evaluation (greedy strategy) | X | | | | |

## 7) Bootstrap CI

The experiment uses a bootstrap 95% confidence interval to answer the question :

Given the final evaluation scores from multiple random seeds, is the mean performance very likely to be above the goal threshold (+0.5)?

The mean and the std are calculated over the list of final evaluation scores (10 seeds in our experiment).

Each bootstrap iteration creates a new list of 10 scores by randomly drawing from the original 10 scores allowing repeats. Then a mean is created for each resampled set. The script repeats that resampling 5000 times, ending up with a list of 5000 bootstrap means.

Then, the script takes quantiles:

- low = 2.5th percentile of bootstrap means.

- high = 97.5th percentile of bootstrap means.

That becomes the **bootstrap 95% confidence interval** for the **mean final score**.

The interpretation can be derived as follows:

- If the lower bound of the 95% CI is above the goal score of +0.5, then the script claims: “Significant (mean > goal @95%)? Yes”
- Otherwise: No (insufficient evidence that the mean is above the goal at 95% confidence)

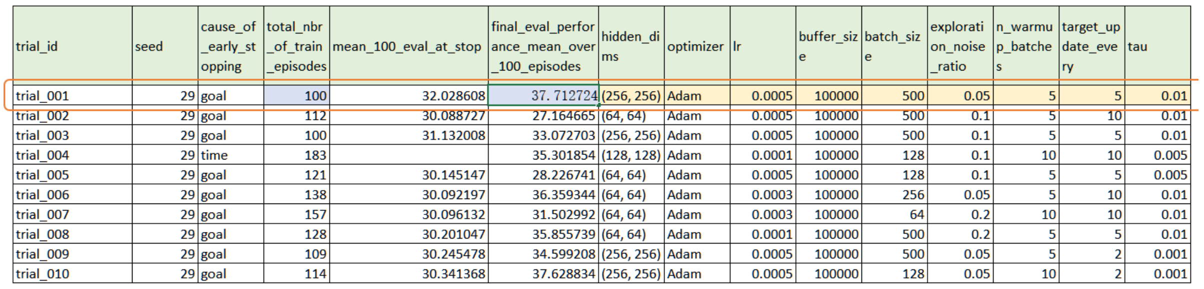
For meaningful inference, the bootstrap CI is only relevant when using multiple seeds.

## 3. Results

### 1) Best hyperparameters

The hyperparameter search found that trial 1 outperformed other trials with:

- 100 episodes required to exceed the goal score, with a 100-episode MA of evaluation score +32.0286 at stop during training
- 100-episode MA of the final evaluation performance (post training) of +**37.7127.**



| trial_id | seed | cause_of_early_stopping | total_nbr_of_train_episodes | mean_100_eval_at_stop | final_eval_performance_mean_over_100_episodes | hidden_dims | optimizer | lr | buffer_size | batch_size | exploration_noise_ratio | n_warmup_batches | target_update_every | tau |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| trial_001 | 29 | goal | 100 | 32.028608 | 37. 712724 | (256, 256) | Adam | 0.0005 | 100000 | 500 | 0.05 | 5 | 5 | 0.01 |
| trial_002 | 29 | goal | 112 | 30.088727 | 27.164665 | (64, 64) | Adam | 0.0005 | 100000 | 500 | 0.1 | 5 | 10 | 0.01 |
| trial_003 | 29 | goal | 100 | 31.132008 | 33.072703 | (256, 256) | Adam | 0.0005 | 100000 | 500 | 0.1 | 5 | 5 | 0.01 |
| trial_004 | 29 | time | 183 |  | 35.301854 | (128, 128) | Adam | 0.0001 | 100000 | 128 | 0.1 | 10 | 10 | 0.005 |
| trial_005 | 29 | goal | 121 | 30.145147 | 28.226741 | (64, 64) | Adam | 0.0005 | 100000 | 128 | 0.1 | 5 | 5 | 0.005 |
| trial_006 | 29 | goal | 138 | 30.092197 | 36.359344 | (64, 64) | Adam | 0.0003 | 100000 | 256 | 0.05 | 5 | 10 | 0.01 |
| trial_007 | 29 | goal | 157 | 30.096132 | 31.502992 | (64, 64) | Adam | 0.0003 | 100000 | 64 | 0.2 | 10 | 10 | 0.01 |
| trial_008 | 29 | goal | 128 | 30.201047 | 35.855739 | (64, 64) | Adam | 0.0001 | 100000 | 500 | 0.2 | 5 | 5 | 0.01 |
| trial_009 | 29 | goal | 109 | 30.245478 | 34.599208 | (256, 256) | Adam | 0.0005 | 100000 | 500 | 0.05 | 5 | 2 | 0.001 |
| trial_010 | 29 | goal | 114 | 30.341368 | 37.628834 | (256, 256) | Adam | 0.0005 | 100000 | 128 | 0.05 | 10 | 2 | 0.001 |

Best hyperparameters found:

- Hidden layer’s dimensions: (256, 256),
- optimizer:  Adam,
- learning rate : 5e-4,
- buffer_size: 100000,
- batch_size: 500,

- exploration_noise_ratio: 0.05,
- n_warmup_batches: 5,
- target_update_every: 5,
- tau: 0.01.

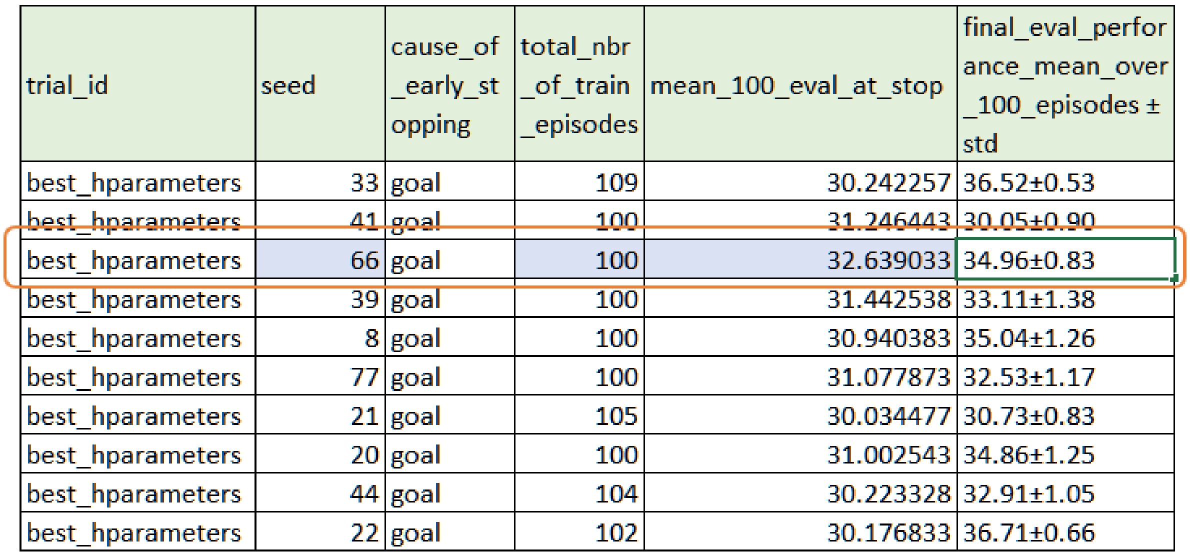
## 2) Solved environment

### a) *Best model over seeds*

Training across 10 seeds with the best hyperparameters found that seed 66 outperformed with the least episodes:

- **100** episodes required to exceed the goal score, with a 100-episode MA of evaluation score of +32.6390 at stop during training.
- 100-episode MA of the final evaluation performance (post training) of **34.96±0.83**

**It can be concluded that the current experiment surpasses Udacity's reference (100 < 163 episodes).**



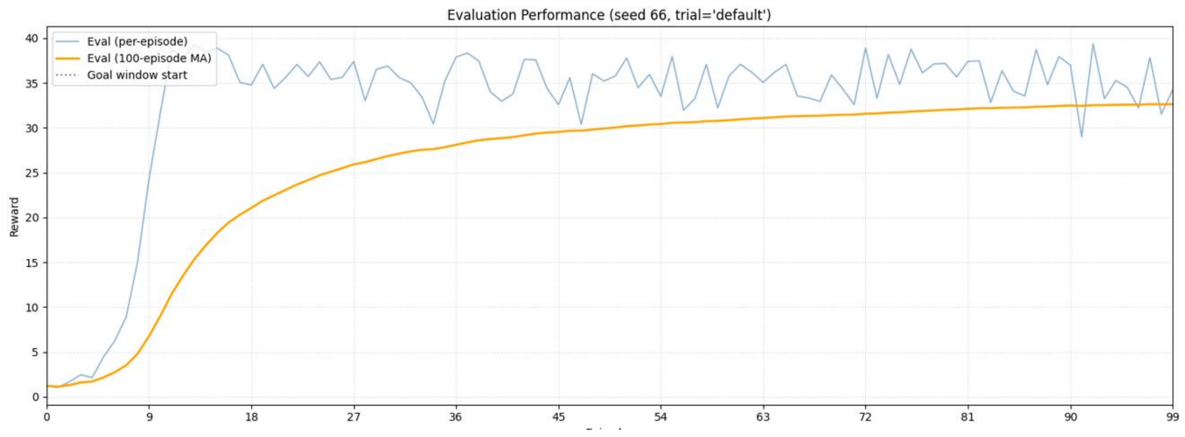
| trial_id | seed | cause_of_early_stopping | total_nbr_of_train_episodes | mean_100_eval_at_stop | final_eval_performance_mean_over_100_episodes ± std |
|---|---|---|---|---|---|
| best_hparameters | 33 | goal | 109 | 30.242257 | 36.52±0.53 |
| best_hparameters | 41 | goal | 100 | 31.246443 | 30.05±0.90 |
| best_hparameters | 66 | goal | 100 | 32.639033 | 34.96±0.83 |
| best_hparameters | 39 | goal | 100 | 31.442538 | 33.11±1.38 |
| best_hparameters | 8 | goal | 100 | 30.940383 | 35.04±1.26 |
| best_hparameters | 77 | goal | 100 | 31.077873 | 32.53±1.17 |
| best_hparameters | 21 | goal | 105 | 30.034477 | 30.73±0.83 |
| best_hparameters | 20 | goal | 100 | 31.002543 | 34.86±1.25 |
| best_hparameters | 44 | goal | 104 | 30.223328 | 32.91±1.05 |
| best_hparameters | 22 | goal | 102 | 30.176833 | 36.71±0.66 |

The weights of the best online policy model are then extracted using the script extract_weights_n_load_into_policy.py.

### b) *Plot of performance of during training*

The below plot shows that the 100-episode MA score (orange curve)  starts increasing very fast from episode 1 before reaching the goal in episode 100. During this period, a very mild variance is observed in the episodic score, reaching a high score around +38.

Moreover , the blue curve is always above the orange one, denoting the solid performance of the model.



The extracted weights and plot can be found in ./ best_model_and_weights_and_plot/ folder.

./ best_model_and_weights_and_plot/

online_policy_model.99.tar
online_policy_model.99.weights.npz
online_policy_model.99.weights.pt

### *c) Bootstrap CI result*

Regarding the statistical test with bootstrap 95% CI across all seeds , it was found that **the bootstrap mean is significantly above the goal score of +30 at 95%,** as shown below:

- list of 10 final seed-based eval scores[36.521, 30.046, 34.957, 33.106, 35.041, 32.525, 30.731, 34.857, 32.915, 36.707],
- bootstrap statistics : Mean=33.741, Std=2.266,
- bootstrap 95% CI for mean : [32.370, 35.062]

This means that **the models created with the best hyper-parameters found, were able to significantly outperform the threshold score of +30.**

## 4. Suggestions for future work

### 1) Replace simple Gaussian exploration with adaptive exploration

In the current experiment, the Gaussian noise is blind. Performance could be improved with an **adaptative exploration** to avoid premature exploration or over-exploration in sensitive regions. For example, we could make the noise increase slightly when the 100-episodes MA evaluation score plateaus, or decay it when the score improves.

### 2) Twin Delayed DDPG (TD3)

The learning core could be upgraded to **TD3** by using 2 critics $Q_{\emptyset 1}$, $Q_{\emptyset 2}$ and having the target use the minimum of both. This could reduce overestimation bias and massively improves stability.

$$y = r + \gamma \min_{i=1,2} Q_{\phi_i^-}(s', \pi_{\theta^-}(s'))$$