Reinforcement Learning Lab 2 (Mandatory Part)

Eirini Stratigi 19940217-3844 stratigi@kth.se

Georgios Moschovis 19970325-7536 geomos@kth.se

Abstract

Q-learning is a simple yet quite powerful algorithm to create a cheat sheet for our Reinforcement Learning agent. It is a recursive equation, we can start with making arbitrary assumptions for all Q-values and with experience, it will converge to the optimal policy. However in some situations, it is not feasible to learn the Q-values for all states. Instead, one needs to generalize by learning about some small number of training states from experience and generalize that experience to new, similar situations. In that case, the Q-function can be approximated by a neural network, resulting in Deep Q-Learning (DQN). In 2^{nd} Lab we implemented a DQN algorithm and applied it to Lunar Lander environment with discrete actions for approximating the optimal policy.

1 Deep Q-network

As we mention above, in some situations, Q-Learning is not feasible to learn the Q-values for all states, when the dimensionality of the state and action spaces is extremely large. The idea of DQNs is simple: we replace the Q-Learning matrix with a Neural Network trying to approximate its values. It is usually referred to as the approximator or the approximating function, and denoted as $Q(s, a; \theta)$, where θ represents the trainable weights of the network. In Deep Reinforcement Learning, the training set is created as we go; we ask the agent to try and select the best action using the current network —and we record the state, action, reward and the next state it ended up at. We are also using an Experience Replay Buffer that we motivate in section 2.1.

2 Experience Replay Buffer and target network

One of the things that was really emphasized during the course is that Q-Learning with function approximation does not seem to work with Neural Networks approximation in its original form. We decide on a batch size $b \in \mathbb{R}$, and after b new records have been recorded in a memory buffer, we select b records at random from that buffer, and train the network. Moreover, apart from the main network (actor) that minimizes the cost function $J(\theta)$, we need a critic ϕ to estimate $Q^{\pi_\theta} \approx Q_\phi$. In the following subsections we motivate the need for both components.

$$
\begin{aligned}
J(\theta) &= \frac{1}{2}\mathbb{E}_{(s,a)\sim\mu_b}\left[\mathrm{BE}_\theta(s,a)^2\right] \\
&= \frac{1}{2}\mathbb{E}_{(s,a)\sim\mu_b}\left[\left(r(s,a) + \lambda \sum_{j=1}^{|\mathcal{S}|} P(j|s,a) \max_b Q_\theta(j,b) - Q_\theta(s,a)\right)^2\right]
\end{aligned}
$$

where we denote target $= r(s,a) + \lambda \sum_{j=1}^{|\mathcal{S}|} P(j|s,a) \max_b Q_\theta(j,b)$

2.1 Experience Replay Buffer

The main problem with Q-Learning with function approximation is that if we see the updates and follow a trajectory $\tau = \{s_i, a_i, r_i\}_{i=1}^{T}$, they are going to be correlated over time. For example, when being in a state $s_t \in \mathcal{S}$ the next state $s_{t+1} \in \mathcal{S}$ is close and the updates are going to be strongly correlated. We would like to have independent updates for the algorithm to converge. The solution to this problem is Experience Replay that uses a buffer $\mathcal{B}$, so that at time t we do not use a sample (s_t, a_t, r_t, s_{t+1}) as the experience, but rather experiences that are sampled from the buffer and are rather far from t and thus uncorrelated.

If the buffer size $|\mathcal{B}|$ is large enough the correlation between the updates in the batch are going to be insignificant because we assumed that, if the batch size b is small, then the positions of the samples (experiences) used in the update will be far enough from each other in time and thus the samples themselves will be uncorrelated.

2.2 Critic Network

One thing to make Q-learning with function approximation work in practice is to use two networks, thus two parameters, differentiate the two terms in Bellman Error that depend on θ as we may notice; the one that we do not derivate in the target and the one that we derivate as they are observed in $J(\theta)$ expression above. Then we observe that, if we do not fix the target, it is going to evolve as θ is going to evolve; so we are trying to track a target that is constantly moving (probably fast). An idea would be to fix the target for a number of successive steps, say $c \in \mathbb{R}$, so that we have convergence of $Q_\theta(s_t, a_t)$ to the fixed target and after that we can update the target.

For this purpose we introduce a second parameter ϕ that will represent the state action value function, therefore $Q_\phi(s, a)$ of the target, and then update the original parameter θ while keeping the target fixed. The latter parameter will be updated every c steps to align with the value of θ. By doing so we are essentially solving the problem introduced by the semi-gradient method (i.e. that we assumed that the terms we do not derivate are fixed although they actually depend on θ -meaning the one that we do not derivate in the target being dependent on θ).

3 Deep Q-network implementation

After incorporating all the above concepts into PyTorch code we ended up with a successful implementation of a Deep Q-network for the Lunar Lander environment with discrete actions. We trained several instances of this network, varying our set of hyperparameters as illustrated in table 1, where early stopping refers to checking the average reward of the past 50 episodes at the end of every episode k and if we cross a minimum threshold $\sum_{i=k-50}^{k} \frac{\mathcal{R}_i}{50} = 50$ we stop training, rather than the classic implementation of early stopping on a separate hold-out (validation) set and patience, while Result is *Pass* when we get an average total reward $\mathcal{R}$ that is at least 50 points on average over 50 episodes and *Fail* otherwise.

ID	Disc. factor γ	Num. of episodes T_E	Early stopping	Buffer size $\lvert\mathcal{B}\rvert$	Reward	Result
1	$\gamma = 0.99$	$T_E = 1000$	True	$\lvert\mathcal{B}\rvert = 10^5$	$\mathcal{R} = 204$	Pass
2a	$\gamma = 1.00$	$T_E = 1000$	True	$\lvert\mathcal{B}\rvert = 10^5$	$\mathcal{R} = 213$	Pass
$2a_{full}$	$\gamma = 1.00$	$T_E = 1000$	False	$\lvert\mathcal{B}\rvert = 10^5$	$\mathcal{R} = -129.7$	Fail
2b	$\gamma = 0.5$	$T_E = 1000$	True	$\lvert\mathcal{B}\rvert = 10^5$	$\mathcal{R} = 47.8$	Fail
$2b_{full}$	$\gamma = 0.5$	$T_E = 1000$	False	$\lvert\mathcal{B}\rvert = 10^5$	$\mathcal{R} = -12.4$	Fail
3a	$\gamma = 0.99$	$T_E = 200$	False	$\lvert\mathcal{B}\rvert = 10^5$	$\mathcal{R} = -65.6$	Fail
3b	$\gamma = 0.99$	$T_E = 1000$	False	$\lvert\mathcal{B}\rvert = 10^5$	$\mathcal{R} = 243$	Pass
3c	$\gamma = 0.99$	$T_E = 1000$	True	$\lvert\mathcal{B}\rvert = 5000$	$\mathcal{R} = 130$	Pass
3d	$\gamma = 0.99$	$T_E = 1000$	True	$\lvert\mathcal{B}\rvert = 30000$	$\mathcal{R} = 48.2$	Fail
$3d_{full}$	$\gamma = 0.99$	$T_E = 1000$	False	$\lvert\mathcal{B}\rvert = 30000$	$\mathcal{R} = 243$	Pass

Table 1: Details of DQN model variants. We experimented with different sets of hyperparameter values to investigate how buffer size $|\mathcal{B}|$ and discount factor γ affect convergence to the optimal policy with an exponentially decaying exploration-exploitation parameter ϵ.

4 Hyperparameter tuning

During the lab we experimented with different sets of hyperparameter values to investigate how buffer size $|\mathcal{B}|$ and the discount factor γ affect convergence to the optimal policy as it is illustrated in table 1. We notice that in general, the DQN model variants that have a *Pass* result have a large number of episodes $T_E = 1000$, a sufficiently large discount factor $\gamma = 0.99$ that allows for more far-sighted evaluation and that early stopping might affect the result, either positively or negatively, when allowing for a large discount factor $\gamma > 0.99$ or a large buffer size $|\mathcal{B}| = 3 \times 10^5$ respectively. Hereunder we will motivate our choices again and for the remaining hyperparameters that are constant for all models illustrated in table 1.

- **Discount factor** γ: a sufficiently large discount factor $\gamma = 0.99$ allows for more far-sighted evaluation that improves network convergence.
- **Replay buffer size** $|\mathcal{B}|$: a replay buffer size $|\mathcal{B}| = 10^5$ is proved enough to achieve a *Pass* result for our DQN. However, as we also motivate in section 6, increasing the buffer size to $|\mathcal{B}| = 3 \times 10^5$ improves approximation of the optimal policy.
- **Number of episodes T_E**: a rather large number of episodes $T_E = 1000$ is required for the network to converge; apart from cases where we are checking the average reward of the past 50 episodes at the end of every episode k and that we cross a minimum threshold $\sum_{i=k-50}^{k} \frac{\mathcal{R}_i}{50} = 50$. However, as we also motivate in section 6, not using that threshold improves approximation of the optimal policy.
- **Batch size N vs. learning rate**: the maximum batch size allowed $N = 128$ yields higher rewards. However, when we account for such high value of N, we need a higher learning rate $\alpha = 10^{-3}$ as well. Theory suggests that when multiplying the batch size by $k \in \mathbb{R}$, one should multiply the learning rate by $\sqrt{k}$ to keep the variance in the gradient expectation constant[1]; in practice we use the maximum allowed value for the learning rate. Moreover for ADAM optimizer we apply a learning rate $\alpha_{\text{ADAM}} = 10^{-4}$ and the default values for $\beta_{\text{ADAM}} = (0.9, 0.999)$ and $\epsilon_{\text{ADAM}} = 10^{-8}$.
- **Exploration-exploitation trade-off parameter** ϵ: since we account for a large number of episodes $T_E = 1000$ and a high value for the discount factor $\gamma = 0.99$; we need an exponentially decaying exploration-exploitation parameter ϵ to ensure convergence to a proper approximation of the optimal policy.

5 Quantitative analysis of the results

As we mention in section 3 we trained several instances of this network, varying our set of hyperparameters as illustrated in table 1, while reaching an overall conclusion that the DQN model variants that have a *Pass* result have a large number of episodes $T_E = 1000$, a sufficiently large discount factor $\gamma = 0.99$ that allows for more far-sighted evaluation. On the contrary, for a low value of the discount factor $\gamma_{\text{low}} = 0.5$ we see that the algorithm at episode $\frac{700}{1000}$ still does not get positive average rewards and around $\frac{80}{1000}$ episodes starts to receive positive average rewards. The algorithm finishes at 809 episode, whereas when opting for a high value of the discount factor $\gamma = 0.99$ that was done around 330 episodes. Experiments with a $\gamma_{\text{low}} = 0.5$ obviously end with a *Fail* reward and that is the case when training for a limited number of episodes $T_{E\text{low}} = 200$ as well. In section 6 we provide a qualitative view of our results for our *Pass* models as well.

As *one image speaks a thousand words*, **in Appendix we provide indicative plots** of the total rewards per episode and average number of steps per episode, for some of the model variants described in table 1 when performing hyperparameter tuning. More interestingly, in section 6 we provide a qualitative view of the learned policy of some among these models, also accompanied with plots of the state-action value function $Q(s, a), s \in \mathcal{S}, a \in \mathcal{A}$ and the greedy action $\max_{a \in \mathcal{A}} Q(s, a), s \in \mathcal{S}$ for the different policies.

[1] See page 5 of A. Krizhevsky, *One weird trick for parallelizing convolutional neural networks*, arxiv.org

6 Qualitative analysis of the results

To qualitatively analyze and compare the best among our DQN model variants, 1, 3b, $3d_{full}$, whose details we illustrate in table 1, for the respective Q-networks Q_{θ_1}, $Q_{\theta_{3b}}$, $Q_{\theta_{3d_{full}}}$, with parameters θ_1, θ_{3b}, $\theta_{3d_{full}}$ that solve the problem, we generate the following two types of plots. Considering a particular restriction of the state $s(y, \omega) = (0, y, 0, 0, \omega, 0, 0, 0)$, where $y \in [0, 1.5]$ is the height of the lander and $\omega \in [-\pi, \pi]$ is the angle of the lander we plot $\max_a Q_\theta(s(y, \omega))$ and $\arg\max_a Q_\theta(s(y, \omega))$. Precisely, we limit our interest in a subspace $\mathcal{S}_{\text{par}}$ of the original state space $\mathcal{S}$, parameterized by $y \in Y = \texttt{range(0, 1.5, 0.25)} \subset [0, 1.5]$ and $\omega \in \Omega = \texttt{range}(-\pi, \pi, 0.25\pi) \subset [-\pi, \pi]$ so that $\mathcal{S}_{\text{par}} \subset \mathcal{S}$. Figures 1, 2 refer to network Q_{θ_1}, figures 3, 4 refer to $Q_{\theta_{3b}}$ where we disable early stopping and therefore allow the network to train for more episodes, while figures 5, 6 refer to $Q_{\theta_{3d_{full}}}$ where we increase the size $|\mathcal{B}|$ of the replay buffer.

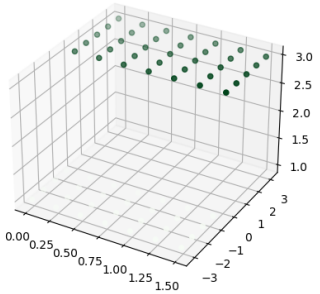

Figure 1: $\arg\max_a Q_{\theta_1}(s(y, \omega)), y \in Y, \omega \in \Omega$

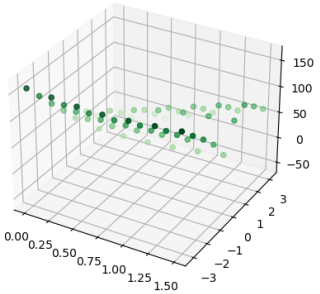

Figure 2: $\max_a Q_{\theta_1}(s(y, \omega)), y \in Y, \omega \in \Omega$

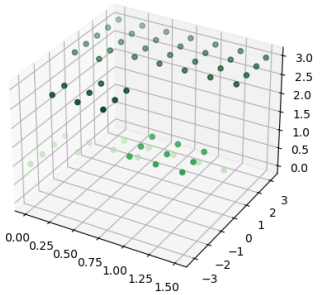

Figure 3: $\arg\max_a Q_{\theta_{3b}}(s(y, \omega)), y \in Y, \omega \in \Omega$

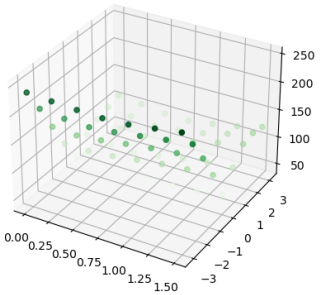

Figure 4: $\max_a Q_{\theta_{3b}}(s(y, \omega)), y \in Y, \omega \in \Omega$

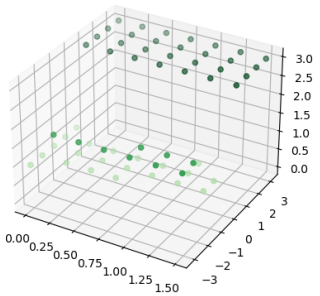

Figure 5: $\arg\max_a Q_{\theta_{3d}}(s(y, \omega)), y \in Y, \omega \in \Omega$

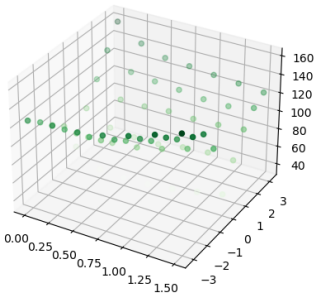

Figure 6: $\max_a Q_{\theta_{3d}}(s(y, \omega)), y \in Y, \omega \in \Omega$

From the above plots we may gain some useful insights. When using model Q_{θ_1}, which is trained for a limited number of episodes due to early stopping as described in section 3 and with a smaller sized replay buffer $|\mathcal{B}| = 10^5$, the agent learns to always perform one particular action as it is illustrated in Figure 1 and the range of the respective Q-values is rather limited as seen in Figure 2. However, when using model $Q_{\theta_{3b}}$, which allows the agent to explore for more episodes $T_E = 10^3$ we get more diverse actions as it is illustrated in Figure 3 and a wider range of Q-values as we may observe in Figure 4. Similarly, when we account for a larger size of the replay buffer $|\mathcal{B}| = 3 \times 10^5$ with model $Q_{\theta_{3d_{\text{full}}}}$, we observe a similar behavior as we may see in Figures 5 and 6. These observations are in line with our expectations as well.

Last but not least, we also compare the aforementioned configurations corresponding to DQN model variants, 1, 3b, $3d_{\text{full}}$, whose details we illustrate in table 1, for the respective Q-networks, with the random agent. As it is illustrated in Figure 7, all neural networks obtain higher rewards during inference than the random agent, which however comes at the cost of an increased number of steps. This is however again in line with our expectations as landing the agent in a safe manner requires more time than performing actions uniformly at random. Hereunder, we provide additional indicative plots of the total rewards per episode and average number of steps per episode, for some of the model variants described in table 1.

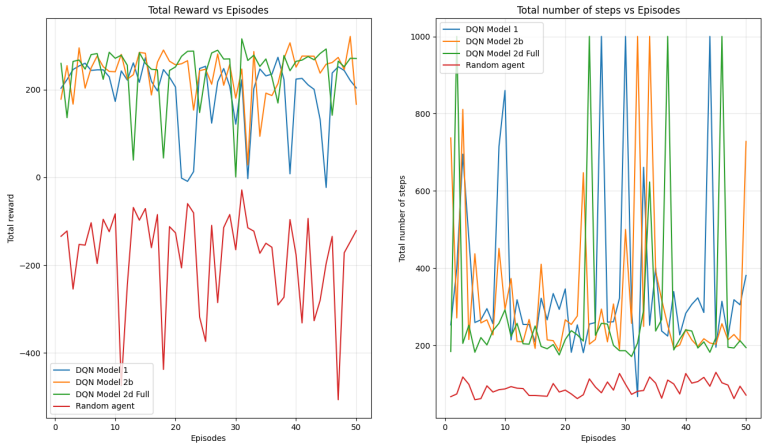

Figure 7: Comparison of several DQN variants with the random agent at **inference**

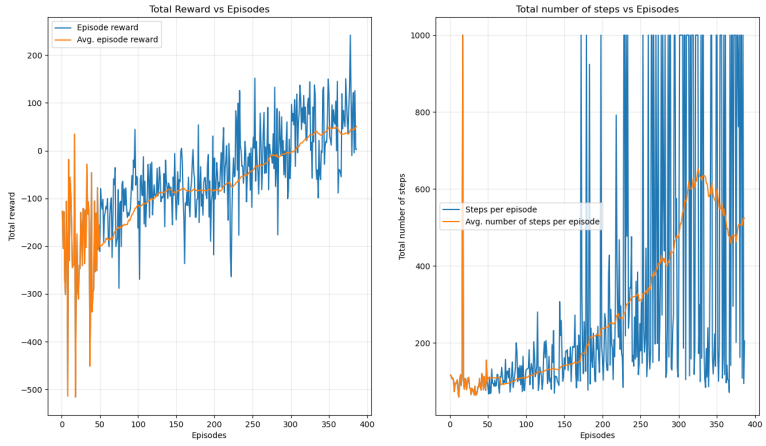

Figure 8: Training evolution of DQN model Q_{θ_1} and averages over 50 episodes

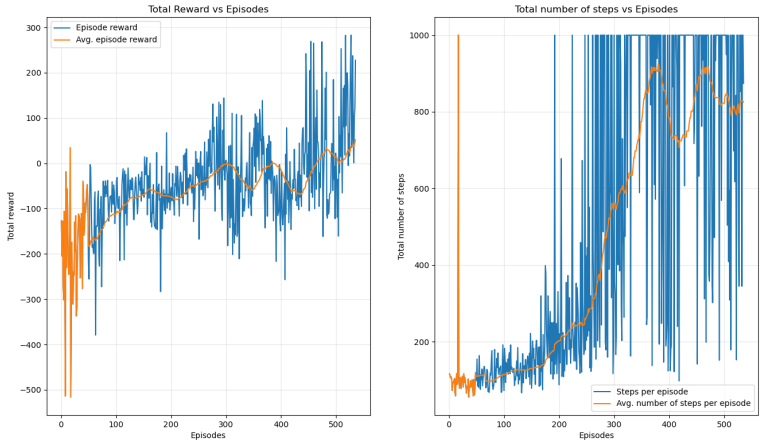

Figure 9: Training evolution of DQN model $Q_{\theta_{2a}}$ and averages over 50 episodes

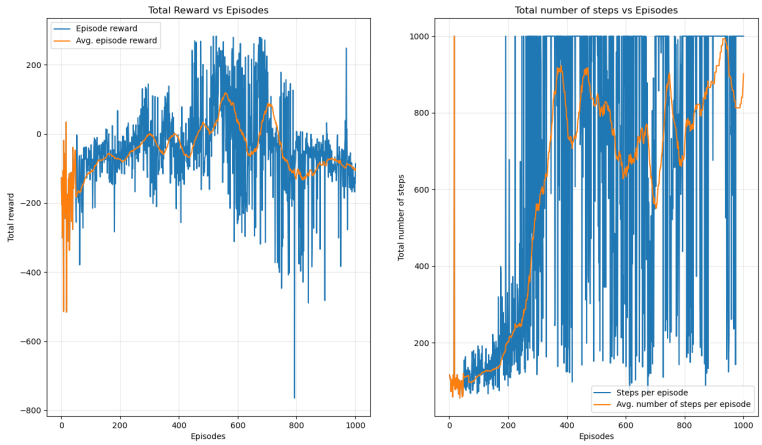

Figure 10: Training evolution of DQN model $Q_{\theta_{2a_{\text{full}}}}$ and averages over 50 episodes

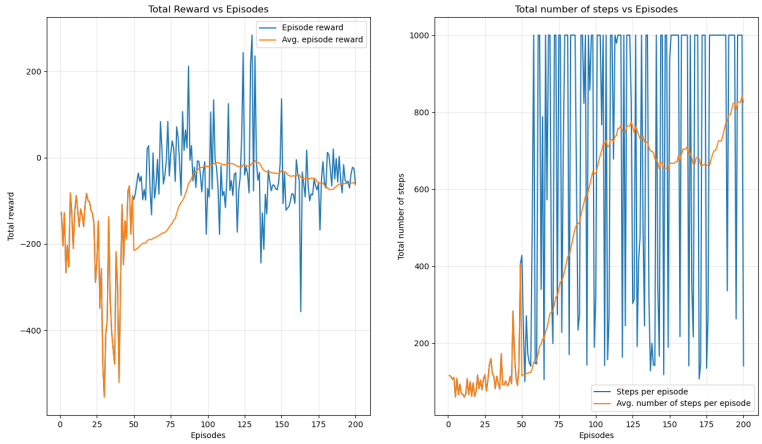

Figure 11: Training evolution of DQN model $Q_{\theta_{3a}}$ and averages over 50 episodes

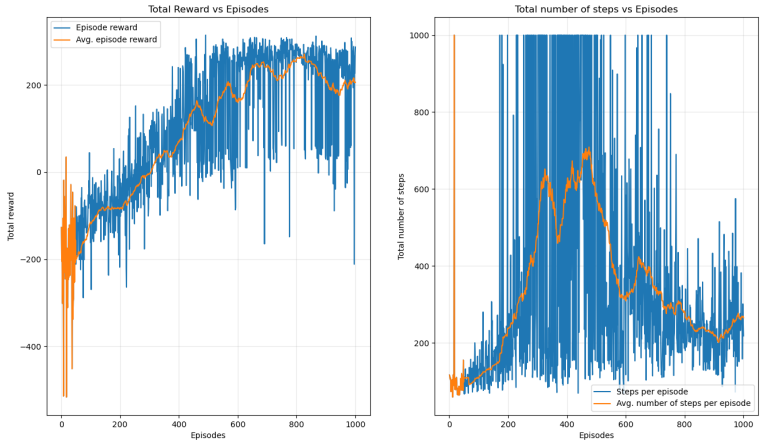

Figure 12: Training evolution of DQN model $Q_{\theta_{3b}}$ and averages over 50 episodes

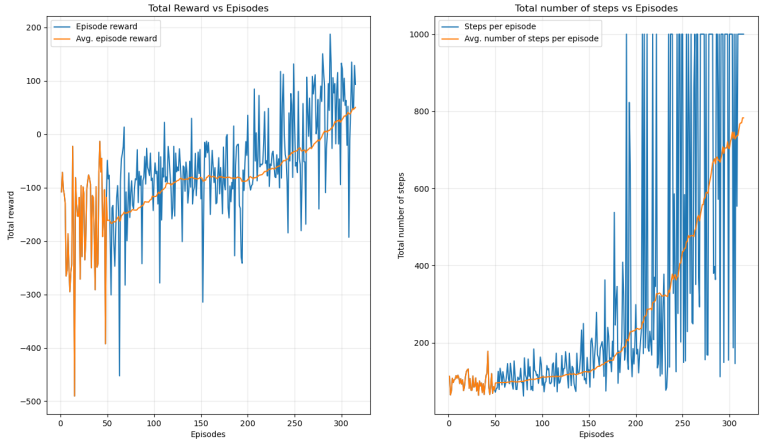

Figure 13: Training evolution of DQN model $Q_{\theta_{3d}}$ and averages over 50 episodes

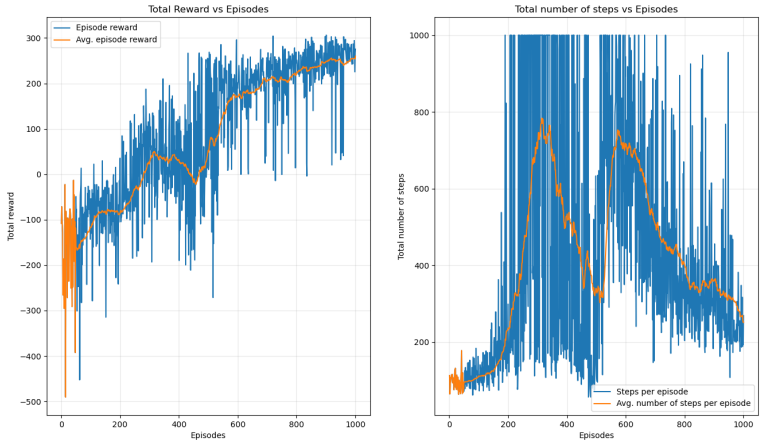

Figure 14: Training evolution of DQN model $Q_{\theta_{3d_{\text{full}}}}$ and averages over 50 episodes

Reinforcement Learning Bonus 2.1 (DDPG Algorithm)

Eirini Stratigi
19940217-3844
stratigi@kth.se

Georgios Moschovis
19970325-7536
geomos@kth.se

Abstract

Deep Deterministic Policy Gradient (DDPG)[1], is a model-free off-policy actor-critic algorithm, improving Deep *Q*-Networks (DQNs) that we address in the mandatory part. Precisely DQNs stabilize the learning of the so-called *Q*-function by using experience replay buffers and a separate target network. The original DQN works in discrete space, and DDPG extends it to continuous space with the actor-critic framework while learning a deterministic policy. In order to improve exploration, an exploration policy π' is constructed by adding noise $\mathcal{N}$ as $\pi'(s) = \pi_\theta(s) + \mathcal{N}$. Furthermore, DDPG implements a *conservative policy iteration* by performing soft updates on the parameters of both actor and critic, $\mathbf{w}' \leftarrow \tau\mathbf{w} + (1-\tau)\mathbf{w}'$. This way, the target network values are constrained to change slowly, different from the design in DQN that the target network stays frozen for some period of time. In 1^{st} Bonus we implemented DDPG and applied it to Lunar Lander environment with continuous actions for approximating the optimal policy.

1 DDPG Algorithm

As we mention above, in situations where we are dealing with a continuous action space, DQN is incapable of estimating *Q*-values corresponding to a good approximation of the optimal policy, since this would involve the discretization of a potentially high-dimensional action space and the number of degrees of freedom would increase exponentially. The idea of DDPG algorithm is to use a model-free, off-policy learning algorithm with deep function approximation, both actor and critic networks, as well as an Experience Replay Buffer. As in DQNs transitions are sampled from the environment according to the exploration policy π_θ and tuples (s_t, a_t, r_t, s_{t+1}) are stored in a replay buffer, then they are used for parameter updates and although convergence is not guaranteed due to function approximation, this network structure is suitable for approximation of the optimal policy in large and continuous action spaces [1].

2 Target networks and off-policy learning in DDPG

DDPG is a reinforcement learning technique that combines both *Q*-learning and Policy gradients and being an actor-critic technique consists of two models; the actor and critic. The actor is a policy network that takes the state as input and outputs the exact action (continuous), instead of a probability distribution over actions. The critic is a *Q*-value network that takes in state and action as input and outputs the *Q*-value. DDPG is an "off"-policy method where the actor computes the action directly instead of a probability distribution over actions and is used in a continuous action setting and is an improvement over the vanilla actor-critic.

[1] Lillicrap et al., *Continuous control with deep reinforcement learning*, arxiv.org

2.1 Target-actor and target-critic networks in DDPG

We have previously discussed in DQN that, given a state $s \in \mathcal{S}$, the optimal action $a \in \mathcal{A}$ is taken by taking $\arg\max_a Q(s, a)$. However, in DDPG, the actor is a policy network that outputs the action directly, which can be continuous. The policy is deterministic since it directly outputs the action. In order to promote exploration, Gaussian noise $w_t \sim \mathcal{N}(0, \sigma^2 I_m)$ is added to the action determined by the policy, typically using the Ornstein-Uhlenbeck process. To calculate the Q-value of a state, the actor output is fed into the critic Q-network to calculate the Q-value, before back-propagating the loss that uses a TD(0) objective.

Furthermore, both actor and critic Q-network, are trained with a target network; the importance of the target-critic Q-network is to give consistent targets during temporal difference backups. Precisely, the actor network may be denoted as $\pi_\theta(s)$, which specifies the current policy by deterministically mapping states to a specific action and it is updated by applying the chain rule to the expected return from the start distribution with respect to the actor parameters θ. The critic $Q_\phi(s, a)$, on the other hand, is learned using the Bellman equation as in Q-learning and to make it stable, since $Q_\phi(s, a)$ is being updated and calculates the target value $G_{t,\text{TD(0)}}$ simultaneously, we use target networks for **both the actor and critic** with "soft" target updates, with parameters θ and ϕ respectively, to ensure stability during training.

$$G_{t,\text{TD(0)}} = \mathcal{R}(s_t, a_t) + \gamma Q_\phi(s_{t+1}, \mu(s_{t+1}))$$

Policy learning in DDPG is fairly simple. Given a state $s \in \mathcal{S}$, we want to learn a deterministic policy $\pi_\theta(s)$, which gives the action $a \in \mathcal{A}$ that satisfies $\arg\max_a Q_\phi(s, a)$ but computing the maximum over actions in the target is a challenge in continuous action spaces. DDPG deals with this by using a target policy network $\pi_{\theta'}(s)$ to compute an action which approximately maximizes $Q_{\phi'}(s, a)$. When computing $\max_\theta \mathbb{E}_s \left[Q_\phi(s, \pi_\theta(s))\right]$ we perform SGA with respect to policy network parameters θ; assuming ϕ fixed (as constant). In the paper it is claimed that having both a target π', parameterized by θ' and Q', parameterized by ϕ' was required to have stable targets $G_{t,\text{TD(0)}}$ in order to consistently train the critic without divergence and although it slows learning, since the target network delays the propagation of value estimations, in practice it favours training stability; especially combined with "soft" target updates.

2.2 DDPG as an off-policy learning algorithm

As mentioned above, in DDPG, given a state $s \in \mathcal{S}$, we are aiming to learn a deterministic policy $\pi_\theta(s)$, which gives the action $a \in \mathcal{A}$ that satisfies $\arg\max_a Q_\phi(s, a)$ but computing the maximum over actions in the target can be a challenge in continuous action spaces. The Q-network is trained off-policy with samples from a replay buffer to minimize correlations between samples and a target Q-network to give consistent targets during temporal difference backups. To calculate the Q-value of a state, the actor output is fed into the Q-network during the calculation of TD-error. DDPG trains a deterministic policy in an off-policy way. Because the policy is deterministic, if the agent were to explore on-policy, in the beginning it would probably not try a wide enough variety of actions to find useful learning signals.

Apart from urging the actor network to explore a larger variety of actions to find useful signals, using an off-policy algorithm **reduces sample complexity** as well, since on-policy reinforcement learning algorithms have high sample complexity, and this is also the key inhibitor to translating the empirical performance of reinforcement learning algorithms from simulation to the real-world, although off-policy algorithms can be difficult to tune [2]. Apart from off-policy algorithms based on Q-function estimation attaining better sample complexity than direct policy optimization, important sampling can be used to correct for the data being off-policy, in practice however for high-dimensional observations, such as images, models of the environment can be difficult to fit and value-based methods can make importance sampling hard to use or even ill-conditioned, especially when dealing with continuous action spaces [3].

3 DDPG algorithm implementation

After incorporating all the above concepts into PyTorch code we ended up with a successful implementation of DDPG algorithm, including actor, critic, actor-target and critic-target networks, for the Lunar Lander environment with continuous actions. We trained several instances of this network, varying our set of hyperparameters as illustrated in Table 1, where Result is *Pass* when we get an average total reward $\mathcal{R}$ that is at least 50 points on average over 50 episodes and *Fail* otherwise. The number of episodes $T_E = 605$ was determined through trial and error using model with ID 1, after applying hyper-parameter tuning for $T_E \in [1, 1000] \cap \mathbb{N}$, measuring again the average total reward $\mathcal{R}$ over 50 episodes as a performance criterion.

For the majority of hyper-parameters we used the recommended parameter values, specifically we set $\gamma = 0.99$, $L = 30000$, $T_E = 300$ batch size $N = 64$, critic and target network update frequency $d = 2$, soft updates with $\tau = 10^{-3}$ and update rule $\mathbf{w}' \leftarrow \tau\mathbf{w} + (1-\tau)\mathbf{w}'$ for both networks $\mathbf{w} \in \theta, \phi$ and $\mathbf{w}' \in \theta', \phi'$ using Ornstein-Uhlenbeck process with noise parameters $\mu = 0.15, \sigma = 0.2$ and update rule $e_t = -\mu \times e_{t-1} + w_t$, starting from $e_0 = 0$ and using random samples $w_t \sim \mathcal{N}(0, \sigma^2 I_m)$. We only differentiated in the number of episodes $T_E = 605$, which was decided by trial and error as described above, while the respective plot of the average total reward $\mathcal{R}$ over 50 episodes can be seen in Figure 1 hereunder.

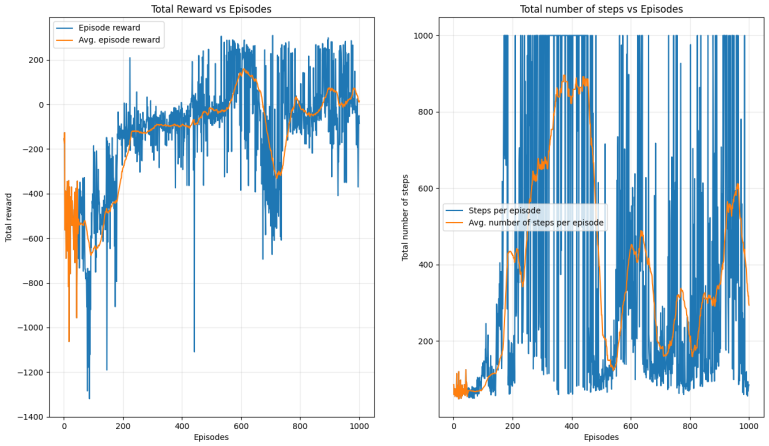

Figure 1: Training evolution of DDPG model Q_{θ_1} for $T_E = 1000$ and averages over 50 episodes

We notice that in general, the two DDPG model variants that have a *Pass* result have a large number of episodes $T_E = 605$, a sufficiently large discount factor $\gamma = 0.99$ that allows for more far-sighted evaluation and a large buffer size $|\mathcal{B}| \geq 3 \times 10^5$ respectively. Hereunder we will motivate our choices again and for the remaining hyperparameters that are constant for all models illustrated in Table 1. We also set the learning rate for the actor Adam optimizer to $\alpha_{\text{ADAM, Actor}} = 10^{-5}$ and for the critic Adam optimizer to $\alpha_{\text{ADAM, Critic}} = 10^{-4}$; thus $\alpha_{\text{ADAM, Actor}} < \alpha_{\text{ADAM, Critic}}$ and will also motivate this choice in section 3.2.

ID	Disc. factor γ	Num. of episodes T_E	Buffer size $\lvert\mathcal{B}\rvert$	Reward	Result
1	$\gamma = 0.99$	$T_E = 605$	$\lvert\mathcal{B}\rvert = 3 \times 10^5$	$\mathcal{R} = 183.7$	Pass
2a	$\gamma = 1.00$	$T_E = 605$	$\lvert\mathcal{B}\rvert = 3 \times 10^5$	$\mathcal{R} = -35.9$	Fail
2b	$\gamma = 0.5$	$T_E = 605$	$\lvert\mathcal{B}\rvert = 3 \times 10^5$	$\mathcal{R} = -200.9$	Fail
3a	$\gamma = 0.99$	$T_E = 605$	$\lvert\mathcal{B}\rvert = 10^5$	$\mathcal{R} = 70.4$	Fail
3b	$\gamma = 0.99$	$T_E = 605$	$\lvert\mathcal{B}\rvert = 9 \times 10^5$	$\mathcal{R} = 182$	Pass

Table 1: Details of DDPG model variants. We experimented with different sets of hyperparameter values to investigate how buffer size $|\mathcal{B}|$ and discount factor γ affect convergence to the optimal policy with the remaining parameters remaining constant.

3.1 Adaptive Moments (ADAM) optimizer

When performing either stochastic or minibatch Gradient Descent, and the loss $E(\mathbf{w})$ changes quickly at one direction and slowly at another, Gradient Descent will progress slowly along the shallow dimension and jitters along the steep one. To overcome this issue, we are using Adam optimizer, so that progress along steep directions is damped and meanwhile progress along flat directions is accelerated. Adam uses exponentially decaying average to discard history but also momentum as an estimate of the first-order gradient. It has bias corrections for first-order and second-order moments and converges rapidly after finding a local convex bowl. If t represents the current time step, Adam updates are given by the following formulas:

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \epsilon \frac{\mathbf{v}^{(t)}}{\delta + \sqrt{\mathbf{r}^{(t)}}}, \delta, \epsilon \in \mathbb{R}^+$$
$$\mathbf{v}^{(t+1)} = \rho_1 \mathbf{v}^{(t)} + (1 - \rho_1)\mathbf{g}^{(t)}, \rho_1 \in \mathbb{R}^+$$
$$\mathbf{r}^{(t+1)} = \rho_2 \mathbf{r}^{(t)} + (1 - \rho_2)\left(\mathbf{g}^{(t)}\right)^2, \rho_2 \in \mathbb{R}^+$$

3.2 Learning rates for the actor and critic networks

In policy gradient methods we need to evaluate Q_{π_θ} to guess $\nabla J(\theta)$ and to do so we include a policy evaluation method in the algorithm. Typically, we do function approximation so we consider the set of state-action value function parameterized by ϕ with using an actor-critic technique consists of two models; the actor and critic as we already motivated. In overall, we draw a set of experiences (s_t, a_t, r_t, s_{t+1}) that are sampled from the environment according to the exploration policy π_θ, we apply the policy evaluation method using the critic network that is doing the TD learning steps and update the actor weights as well.

Following the typical convention, in our implementation we set $\alpha_{\text{ADAM, Actor}} < \alpha_{\text{ADAM, Critic}}$, which means that we update θ **very slowly** compared to ϕ. This is because it is as I fix the policy π_θ for a long time to get a good critic. Remember that the critic goal is evaluating learnt policy π_θ, computing $Q_{\pi_\theta}(s, a), s \in \mathcal{S}, a \in \mathcal{A}$, so if π_θ is changing all the time the critic is not going to be very efficient. In other words, it is like when we fix the target in DQN, which we motivate in the mandatory part of the Lab. We fix the target because we do not want it to evolve too much so that the algorithm is more stable. In order to be efficient, we fix π_θ for a while and in that way we update ϕ, so that $Q_\phi(s, a)$ is a reasonable approximation of $Q_{\pi_\theta}(s, a)$.

On the other hand, if θ is changing very rapidly as illustrated in the red curve of Figure 2, Q_{π_θ} is also changing very rapidly because there is no way that ϕ is going to manage to follow. However, if we update θ very slowly compared to ϕ as we propose above and illustrate in the blue curve of Figure 2, it is no longer evolving in every step in a significant way; Q_{π_θ} is no longer difficult to track and the critic is not lost anymore.

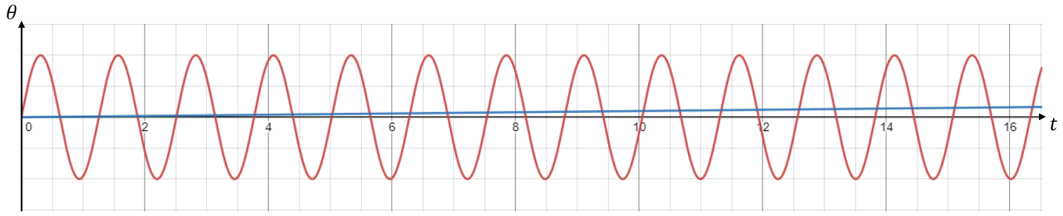

Figure 2: Evolution examples of θ over time t using different learning rates α_{Actor}.

3.3 General hyperparameter tuning

Finally, although the reasoning is quite identical as for DQN in the mandatory part, in the following we motivate our choices of other hyperparameters as well. One significant difference is that in DDPG we need a larger buffer size.

- **Discount factor** γ: a sufficiently large discount factor $\gamma = 0.99$ allows for more far-sighted evaluation that improves network convergence.
- **Replay buffer size** $|\mathcal{B}|$: a larger replay buffer size $|\mathcal{B}| \geq 3 \times 10^5$ is needed to achieve a *Pass* result for our DDPG and result to a decent approximation of the optimal policy. Recall that in DQN a buffer size $|\mathcal{B}| = 10^5$ was proved sufficient.
- **Number of episodes T_E**: a rather large number of episodes $T_E = 605$ is required for the network to converge that is larger than the proposed amount $T_E = 300$. As we already explained the number of episodes was determined through trial and error using model with ID 1, after applying hyper-parameter tuning for $T_E \in [1, 1000] \cap \mathbb{N}$, measuring again the average total reward $\mathcal{R}$ over 50 episodes as a performance criterion. The respective plot of the reward $\mathcal{R}$ over 50 episodes can be seen in Figure 1.
- **Batch size N vs. learning rate**: the proposed batch size $N = 64$ yields sufficient rewards. For this reason, we also use the proposed values for the learning rates $\alpha_{\text{ADAM, Actor}} = 10^{-5}$ and $\alpha_{\text{ADAM, Critic}} = 10^{-4}$, with $\alpha_{\text{ADAM, Actor}} < \alpha_{\text{ADAM, Critic}}$ for the reasons we motivated in section 3.2. Since we no longer account for a higher value of N, as we did in DQN, we no not need a higher learning rate. Moreover we keep the default values for $\beta_{\text{ADAM}} = (0.9, 0.999)$ and $\epsilon_{\text{ADAM}} = 10^{-8}$ for both networks' optimizers.

As *one image speaks a thousand words*, Figure 3 is a zoomed-in version of Figure 1 when training model Q_{θ_1} for $T_E = 605$ episodes and **in Appendix we provide additional indicative plots** of the total rewards per episode and average number of steps per episode, for model $Q_{\theta_{3b}}$ as well described in Table 1 when performing hyperparameter tuning. More interestingly, in section 4 we provide a qualitative view of the learned policy of some among these models, also accompanied with plots of the state-action value function $Q(s, a), s \in \mathcal{S}_{\text{par}}, a \in \mathcal{A}$ for the different policies; where $\mathcal{S}_{\text{par}} \subset \mathcal{S}$ a parameterized subspace of the original state space.

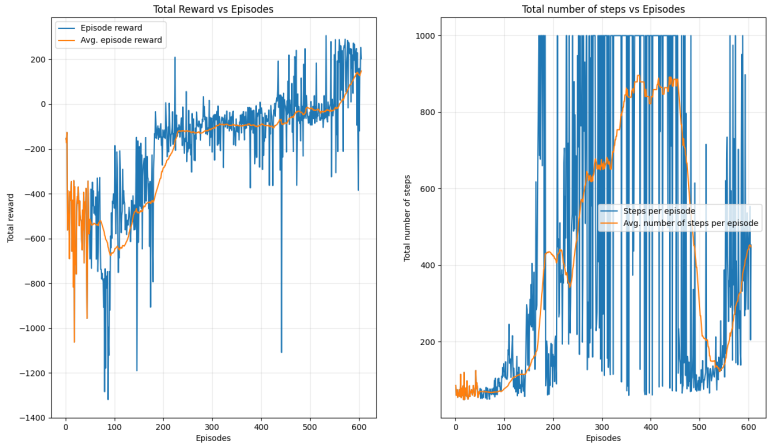

Figure 3: Training evolution of DDPG model Q_{θ_1} for $T_E = 605$ and averages over 50 episodes

4 Qualitative analysis of the results

To qualitatively analyze and compare the best among our DDPG model variants, 1 and 3b, whose details we illustrate in Table 1, for the respective networks Q_{θ_1}, $Q_{\theta_{3b}}$, with parameters θ_1, θ_{3b} respectively that solve the problem, we generate the following two types of plots. Considering a particular restriction of the state $s(y, \omega) = (0, y, 0, 0, \omega, 0, 0, 0)$, where $y \in [0, 1.5]$ is the height of the lander and $\omega \in [-\pi, \pi]$ is the angle of the lander we plot $Q_\omega\left(s(y, \omega), \pi_\theta(s(y, \omega))\right)$. Precisely, we limit our interest in a subspace $\mathcal{S}_{\text{par}}$ of the original state space $\mathcal{S}$, parameterized by $y \in Y =$ `range(0, 1.5, 0.25)` $\subset [0, 1.5]$ and $\omega \in \Omega =$ `range(`$-\pi$`, `π`, 0.25`π`)` $\subset [-\pi, \pi]$ so that

$\mathcal{S}_{\text{par}} \subset \mathcal{S}$. Figure 4 refers to network Q_{θ_1}, while Figure 5 refers to $Q_{\theta_{3b}}$ where we increase the size $|\mathcal{B}|$ of the replay buffer three times. Similarly to what we reported for DQN model variants, when using model Q_{θ_1}, which is trained with a smaller sized replay buffer $|\mathcal{B}| = 3 \times 10^5$, the range of the respective Q-values is more limited and their magnitude is larger as seen in Figure 4. Similarly, when we account for a larger size of the replay buffer $|\mathcal{B}| = 9 \times 10^5$ with model $Q_{\theta_{3b}}$, a wider range of Q-values with smaller magnitude as we may see in Figure 5. These observations are again in accordance with our expectations as well.

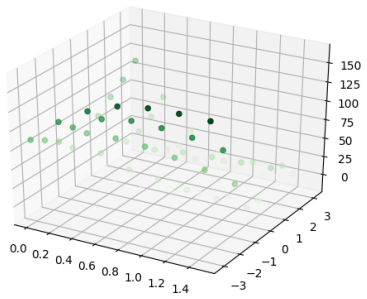

Figure 4: $Q_{\theta_1}\left(s(y,\omega), \pi_\theta(s(y,\omega))\right)$

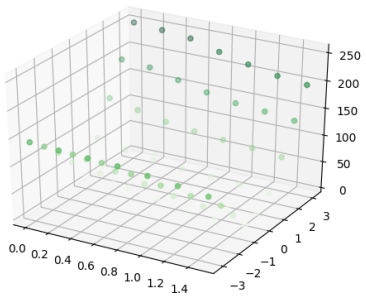

Figure 5: $Q_{\theta_{3b}}\left(s(y,\omega), \pi_\theta(s(y,\omega))\right)$

Last but not least, we also compare the aforementioned configurations corresponding to DDPG model variants, 1, 3b, whose details we illustrate in table 1, for the respective Q-networks, with the random agent. As it is illustrated in Figure 6, all neural networks obtain higher rewards during inference than the random agent, which however comes at the cost of an increased number of steps. This is however again in line with our expectations as landing the agent in a safe manner requires more time than performing actions uniformly at random. Hereunder, we provide additional indicative plots of the total rewards per episode and average number of steps per episode, for some of the model variants described in Table 1.

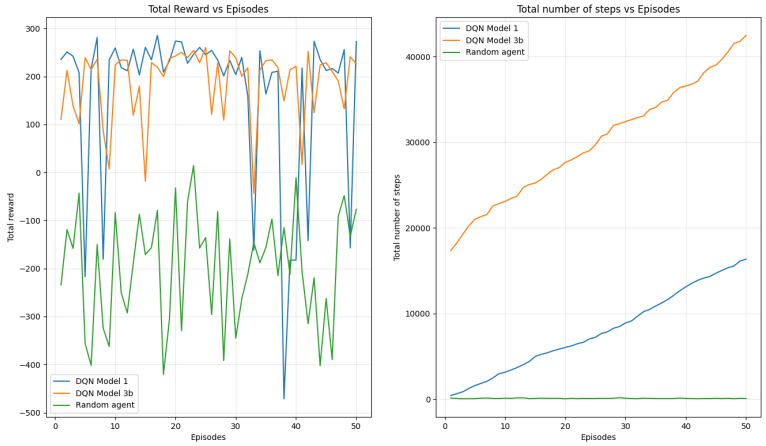

Figure 6: Comparison of several DQN variants with the random agent at **inference**

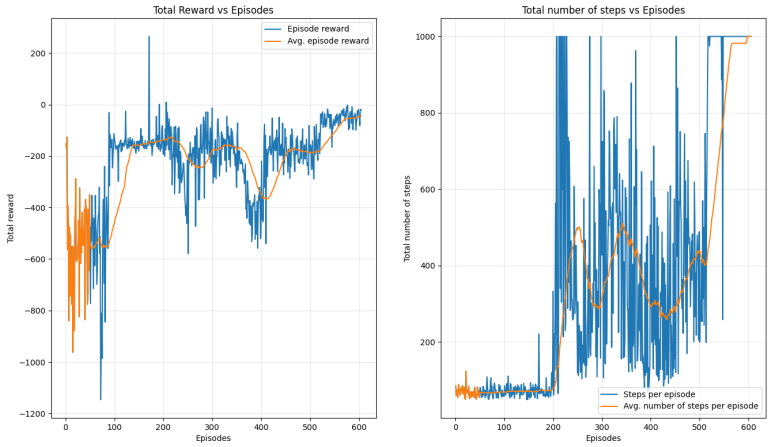

Figure 7: Training evolution of DDPG model $Q_{\theta_{2a}}$ for $T_E = 605$ and averages over 50 episodes

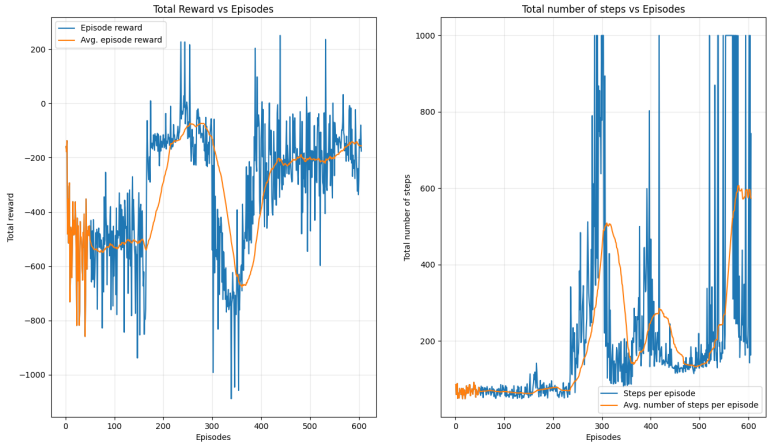

Figure 8: Training evolution of DDPG model $Q_{\theta_{2b}}$ for $T_E = 605$ and averages over 50 episodes

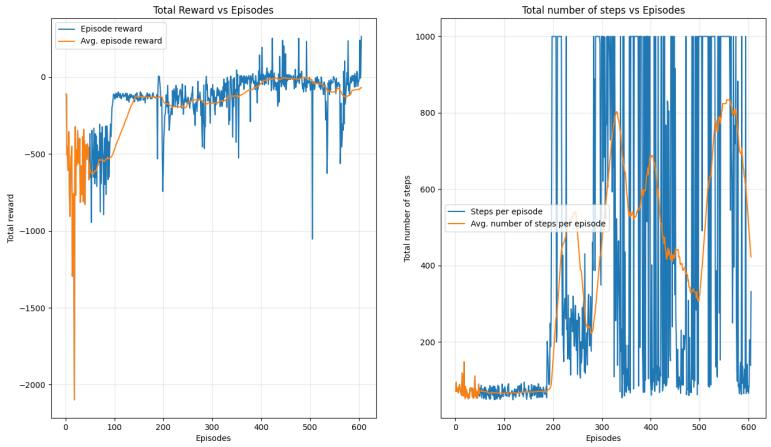

Figure 9: Training evolution of DDPG model $Q_{\theta_{3a}}$ for $T_E = 605$ and averages over 50 episodes

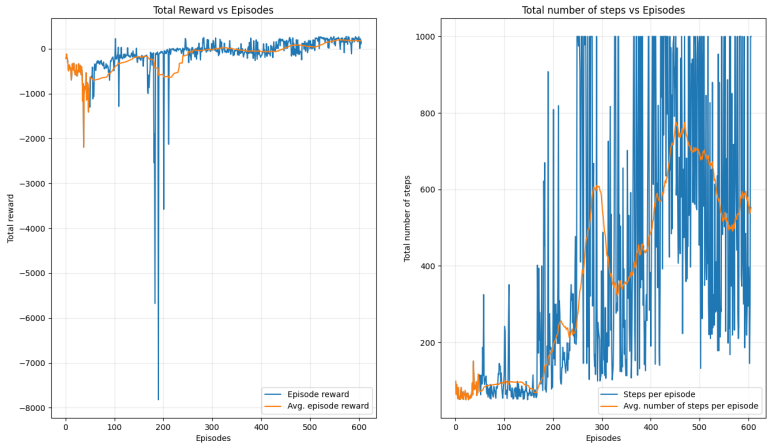

Figure 10: Training evolution of DDPG model $Q_{\theta_{3b}}$ for $T_E = 605$ and averages over 50 episodes

References

[1] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning.," in *ICLR* (Y. Bengio and Y. LeCun, eds.), 2016.

[2] R. Fakoor, P. Chaudhari, and A. J. Smola, "P3O: policy-on policy-off policy optimization," *CoRR*, vol. abs/1905.01756, 2019.

[3] A. Irpan, K. Rao, K. Bousmalis, C. Harris, J. Ibarz, and S. Levine, "Off-policy evaluation via off-policy classification," *CoRR*, vol. abs/1906.01624, 2019.