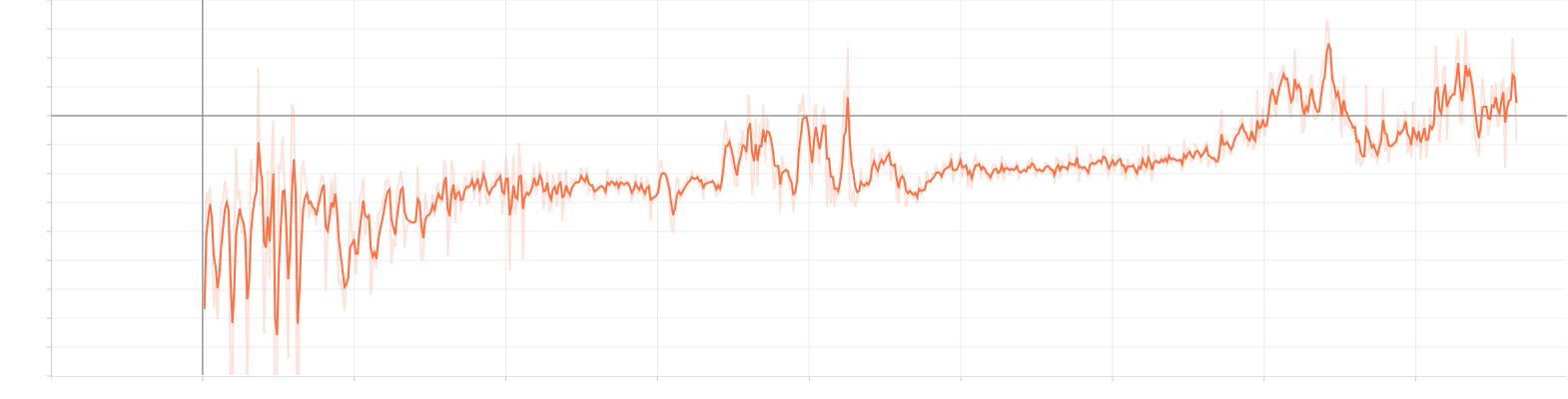
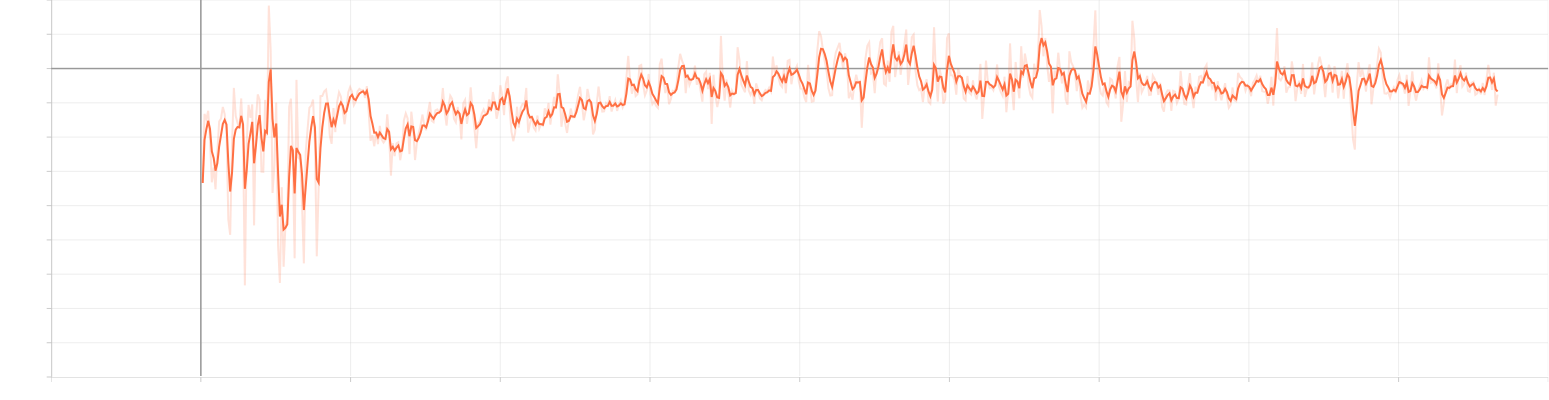
**DAM Damien & DURAND Enzo**

Tabular learning in the last project was only effective for environments with discrete state and action, but it’s no longer the case when one of the two is continuous with infinite possible values. In this report, we explore the simplest deep RL algorithms when the state space is continuous while action space stays discrete (for now), being Deep Q-Network (DQN) and its variation Double DQN (DDQN), as well as Deep Deterministic Policy Gradient (DDPG).

**TME 3 – DQN vs DDQN**

The DQN algorithm involves learning the Q-value of a state-action pair by using deep neural networks, instead of using MDP. However, the limitation of DQN is that if it happens to over-estimate the value of an action in a given state, the agent would tend to choose that action more often. In order to avoid that problem, we use DDQN which decouples action selection from value estimation.

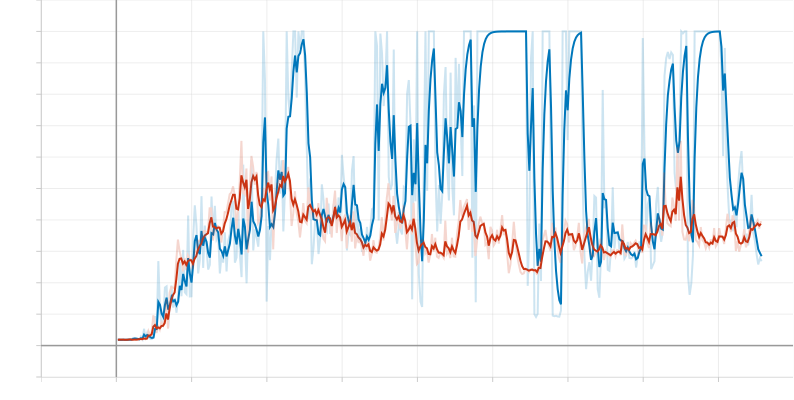
We tested both algorithms on the *CartPole-v1* and *LunarLander-v2* environments, and we observed that their performance was relatively similar in CartPole, whereas DDQN is more stable than DQN in LunarLander though they both struggle to obtain positive average reward. We plot the reward of both algorithms below.

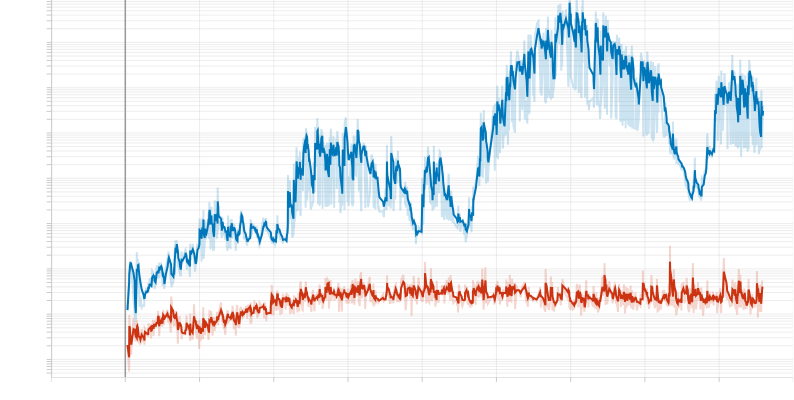
*Average reward of DQN (above) and DDQN (below) in LunarLander-v2*

We can observe that although DDQN does not help to increase reward, it is evidently more stable and less variant than DQN thanks to its separation property between action selection and value estimation. The advantage of DDQN is obviously clearer by looking at the loss value and reward plots below in CartPole, where DDQN’s loss value is kept at a very low value while DQN struggles to keep it in check. We should not, however, conclude prematurely that DQN is better just by looking at the reward plot. Although DQN was able to achieve the maximum reward of 500 in our experiment, the environment is one of the easiest ones in *gym*, and the algorithm is still very unstable and seemingly having a hard time staying at 500 reward. At the same time, DDQN is slower but much more stable and if given enough time, it will surely converge to the optimal policy for the environment.

Since the goal of this project is to compare DQN and DDQN, we did not try to find the optimal parameters to obtain the best reward for both environments.



*Reward obtained by DQN (blue) and DDQN (orange) in CartPole-v1*



*Loss value of DDQN (orange) and DQN (blue) on the log-scale in CartPole-v1*

The code for DDQN’s critic update function is as below

def compute\_ddqn\_loss(cfg, reward, must\_bootstrap, q\_values, target\_q\_values, action):

    max\_q = target\_q\_values.max(1)[0].detach()

target = reward[:-1] + cfg.algorithm.discount\_factor \* max\_q \* must\_bootstrap.int()

qvals = q\_values[0].gather(1, action[0].unsqueeze(-1)).squeeze(-1)

    # Compute critic loss

    td = target – qvals

    td\_error = td\*\*2

    critic\_loss = td\_error.mean()

    return critic\_loss

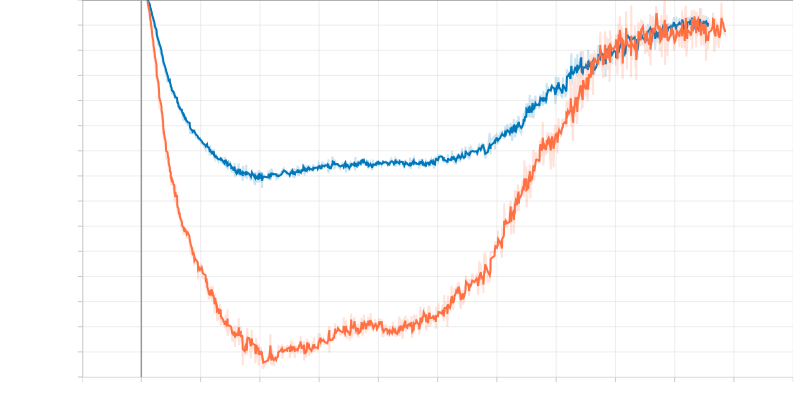
**TME 4 – TD3**

The DQN algorithm and its variation DDQN can only predict the value of each action (discrete) in a given state (continuous). In order to work with continuous action space, we started to learn DDPG and TD3, which consist of a critic and an actor network. The actor network outputs a vector of action given a state, and together this pair of state-action is passed through the critic network that outputs its Q-value.

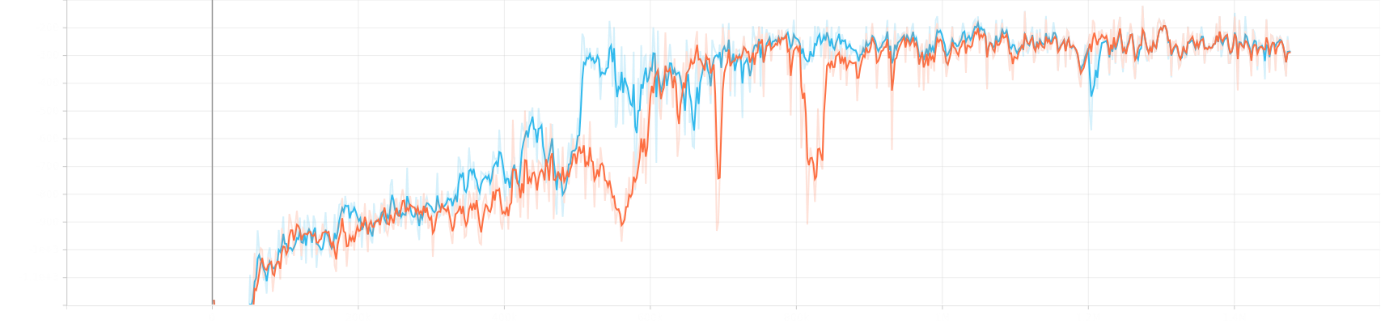
For this mini project, given the code of DDPG, we programmed the TD3 algorithm and compare the performance of both in the *Pendulum-v1* environment.

One of the properties of DDPG is that at the beginning of training when the algorithm has not obtained much information, the critic network tends to output the Q-value way too far from its true value. This is called an over-estimation and it slows down training and convergence. In order to mend this weakness, we train at the same time a second critic network whose job is the same as the first one, except that we choose the minimum Q-value output by both networks. This way, we reduce the over-estimation bias of the training and as a result, it converges faster.

Looking at the summed output Q-value of DDPG and TD3 below, we can see that at the beginning of training, both algorithms output low Q-values due to insufficient data, but TD3’s output is much closer to convergence since we choose a less over-estimated value every time. That is why we can observe earlier optimal reward with TD3 (negative 200).



*Q-value over-estimation by DDPG (orange) and TD3 (blue) in Pendulum-v1*



*Reward obtained by DDPG (orange) and TD3 (blue) in Pendulum-v1*