Reinforcement Learning- Final Course Project

Elad Prager (ID. 200865780),Shira Moscovitch (ID. 300093150)

Submitted as final project report for the RL course, IDC, 2022

1 Introduction

Autonomous vehicles are one of the most exciting, up-and-coming applications of deep reinforcement learning. The motivation behind this filed is enormous and includes safety, better commuting experience, environmental benefits and more.In this project we tried to solve several variations of the highway-env [2], which is a collection of environments for autonomous driving and tactical decision-making tasks.Our goal was to help the user car overtake the bot cars on its own in a roadway environment. We trained the user car using deep reinforcement learning models while exploring different models parameters and configurations.

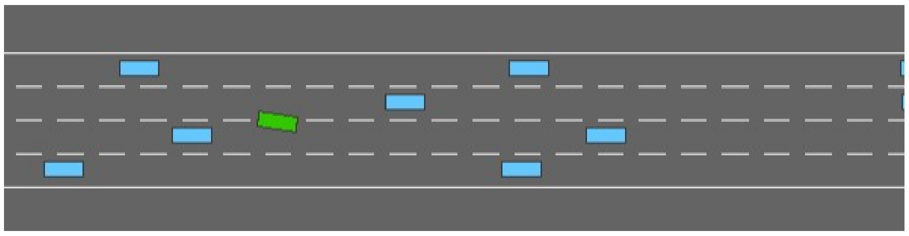

Figure 1: Highway env

In this challenge we tried to solve three types of environments, Highway, Merge and Roundabout. We solved each environment separately and also tried to implement a super agent that can solve any on these three environments.The target of any environment is slightly different though the main target of all the environments is to drive through the lanes or merge lane or runabout safely. The model reward functions penalized the user car every time it slows down, every time it crashes into bot car and if there are any bot cars in front of it. The state space was the raw pixels (Observations: 4 X 128 X 128 image), therefore we trained CNN Neural Networks. We will show our approach in solving the Highway-env environments with DQN based algorithms and explore how new innovations in the field can help us get better results. We will also experiment with different hyper parameters to improve results and speed convergence.

1.1 Related Works

1.1.1 Deep Q Learning Network (DQN)

In traditional Q learning, a table is constructed to learn state-action Q-values by exploring and exploiting the environment with experiences. Given a current state, the goal is to learn a policy that helps the agent taking the action that will maximize the total reward. It is an off-policy algorithm meaning the function learns from random actions that are outside the current policy. The Q-values are learnt by playing with the environment and incrementally updated by the Bellman equation:
$Q(s,a) \leftarrow Q(s,a) + \alpha[R + \gamma max_{a'} Q(s',a') - Q(s,a)]$
In this equation α is the learning rate and γ is the discount-factor of the training network.

This can however be expensive in terms of memory and computation.To handle this issues, Mnih (2013) [3] came up with a variant of Q-learning that uses deep neural networks. The key idea behind DQN is to train a network that learns the Q values for state-action pairs. There are few components in the DQN:
1. Replay buffer - A large enough memory that saves previous experiments by tuples of transitions {state, action, next-state, reward, extra -info}
2. A training network (Q) - We feed into the network a sampled batch from the Replay buffer. the aim of the network is to predict the Q-values.The Q-values corresponding to the actions from the batch together with the rewards, will be used to compute the model loss.
3. A target network (Q^T)- The target network is similar to the training network and is used to "construct labelled data". The target network takes the next states and predicts the best Q value out of all actions (Q-values target). For stability, the target network will be updated once in few episodes, this is also a hyper parameter of the model.
The Loss function is based on the TD-error:
$Loss = \frac{1}{2}(R + \gamma max_{a'} Q^T(s',a') - Q(s,a))^2$
Where the first expression is the target given by the target network and the second expression is the prediction given by the training network.

1.1.2 Double Q-learning

In some stochastic environments Q-learning is showing overestimation that introduces a maximization bias in learning and since Q-learning involves bootstrapping (learning estimates from estimates) such overestimation can be problematic. Traditional DQN tends to significantly overestimate action-values (Q-values) that is leading to unstable training and low quality policy. Hasselt(2010) [5] proposed the double Q-learning method to solve this. Double Q-Learning uses two different action-value networks, Q and $\hat{Q}$, as estimators. $\hat{Q}$ is our target model like in DQN (Q^T), but here for target evaluation we use action selected by maximizing Q-values calculated using the Q network. Here the target is given by:

$R + \gamma max_{a'} Q^T(s', a')$ while $max_{a'}$ is given by the Q-network

1.1.3 Dueling DQN (DDQN)

In 2016 Wang [6] presented the novel dueling architecture which explicitly separates the representation of state values and state-dependent action advantages via two separate streams. The key motivation behind this architecture is that for some games, it is unnecessary to know the value of each action at every time-step. In the Highway-env, for example, it is not necessary to know which action to take at all times but only in certain situation like, collision is coming or when we need to move a lane. By explicitly separating two estimators, the dueling architecture can learn which states are (or are not) valuable, without having to learn the effect of each action for each state.

Our dueling network represents two separate estimators, one for the state value function and one for the state-dependent action advantage function:
$A^\pi(s, a) = Q^\pi(s, a) - V^\pi(s)$
The Advantage quantity is obtained by subtracting the Q-value, by the V-value.The Advantage value shows how advantageous selecting an action is relative to the others at the given state.
The architecture is similar to the DQN, we have convolution layers to process game-play frames. From there, we split the network into two separate streams, one for estimating the state-value and the other for estimating state-dependent action advantages. After the two streams, the last module of the network combines the state-value and advantage outputs.

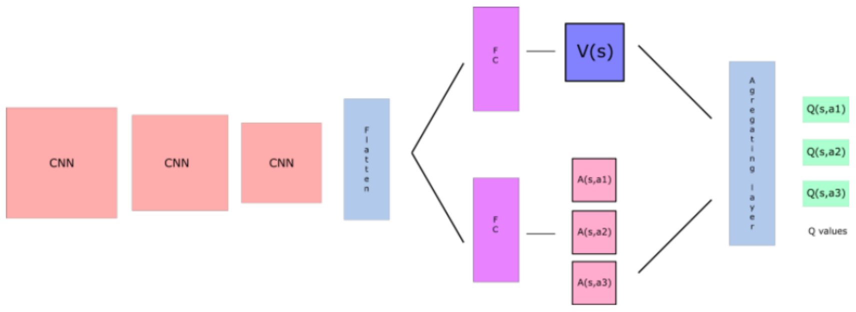

Figure 2: Dueling DQN architecture.Reference: here.

1.1.4 prioritized replay buffer

The Prioritized replay buffer was introduced by Schaul (2015) [4] with the idea that some experiences may be more important for learning than others, but might occur less frequently. When batches of experiences are randomly selected from the buffer, rare experiences have small chance to be selected.The priority buffer tries to change the sampling distribution by using a criterion to define the

priority of each tuple of experience. A greedy TD-error prioritization will select always the transitions with highest TD error, however to avoid over-fitting that such approach can introduce, the paper suggest an interpolation between pure greedy prioritization and uniform random sampling.
The probability of sampling transition is defined by: $P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}$
Where $p_i = |\delta_i| + \epsilon > 0$ is the priority of transition i.
The probability of being sampled is monotonic in a transition's priority, while guaranteeing a non-zero probability even for the lowest-priority transition.
The estimation of the expected value with stochastic updates relies on those updates corresponding to the same distribution as its expectation.

Prioritized replay introduces bias because it changes distribution of the samples, and therefore changes the solution that the estimates will converge to. This bias can be corrected with by reducing the weights of the often seen sample (importance-sampling (IS) weights).

1.1.5 Categorical DQN (C51)

C51 is a Q-learning algorithm based on DQN by bellemare (2017) [1]. Like DQN, it can be used on any environment with a discrete action space. The main difference between C51 and DQN is that rather than simply predicting the Q-value for each state-action pair, C51 predicts a histogram model for the probability distribution of the Q-value. By learning the distribution rather than simply the expected value, the algorithm is able to stay more stable during training, leading to improved final performance. In Categorical DQN (C51) the possible returns are limited to a discrete set of fixed values (51), and the probability of each value is learned through interacting with environments.

2 Solution

2.1 General approach

We experimented with few algorithms DQN, Double-DQN, Dueling DQN, with random replay buffer, prioritized replay buffer and with C51 as they are described in previous section. We chose those algorithms since they are suitable for discreet environment and a relatively simple for implementation but with many improvement to test and optimize.

2.2 Design

For the DQN algorithm and variations, we used a CNN with 3 convolution layers (of size 32, 64 and 64) and one fully connected layer of size 256. We used the Huber loss function, which is not very sensitive to outliers in the data and Adam optimizer.The entire optimization was done on a stochastic environment with probability of 85% to take the suggested action.

3 Experimental results

Our strategy was first to converged the the Highway-env (easy) on classic DQN by training 5000 episodes and with suitable parameters, this was our baseline model. Then we checked the the selected algorithm with different hyper parameters.The relatively small number of episodes allowed us to iterate fast while seeing the trends of the convergence.
The tested algorithms with the train results of mean rewards in last 100 episodes, are showed in this table

Model	eprewmean	eplenmean
DQN	31.49	39.85
Double DQN	30.32	38.01
DQN with prioritized replay	35.59	45.3
Dueling DQN	39.63	49.31
DQN + C51	33.5	42.1
DDQN with prioritized replay and C51*	44.44	58.45

DQN Models comparison

*model with optimized hyper parameters.

Double DQN did not improve the model, we do see a big improvement though with priority replay and with Dueling DQN. We also see some improvement with C51. The priority buffer reduced the loss noise significantly as can be seen in the image below. DDQN improves the learning and gave significantly higher rewards and convergence.

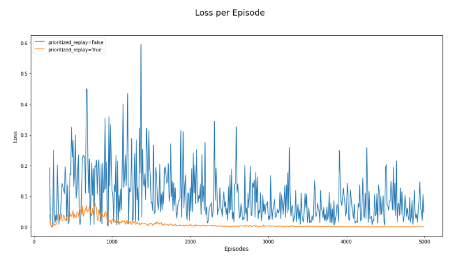

Figure 3: DQN loss with and without Priority replay

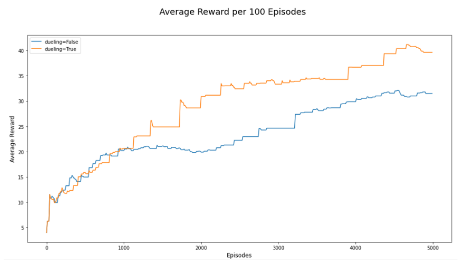

Figure 4: DQN VS. DDQN

From the results we've seen the selected algorithm was DDQN with priority replay and C51. We also tested few hyper-parameters against the baseline, the parameters we looked at were buffer size, exploration fraction (how many steps are used for exploration), final epsilon (for greedy policy), target network update period, discount-factor and learning rate.
The Baseline model was a Vanilla DQN with 32 batch size, buffer of size 15000, exploration fraction 0.2 and final epsilon 0.05, target network update period was

50, discount-factor was 0.8 and 5e-4 learning rate .
The selected model was DDQN with priority replay and C51, buffer size was 10000, batches of 32 ,exploration fraction 0.2 and final epsilon 0.01, target network update period 100, discount-factor was 0.9 and learning rate was kept 5e-4. Over all the optimized model gave 40% percent improvement in mean rewards.

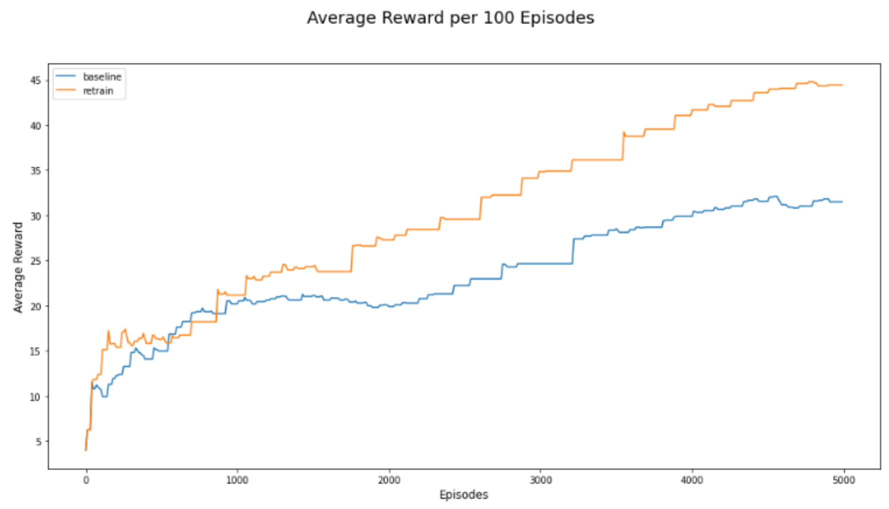

Figure 5: Base-line model VS final model

3.0.1 EX1 - Highway-Env (Easy)

Next, we've tested our best model configuration, over the easiest environment configuration, consists of 3 lanes, a low vehicle count and density, with a non-aggressive behavior. We've trained the model over 20k iterations, resulted with a great initial proof of concept. Our agent managed to solve the environment with a high reward of 364 points, surviving the whole 500 steps.

3.0.2 EX2 - Highway-Env (Medium)

The next environment configuration, consists of 6 lanes, a higher vehicle count and density, with a more aggressive behavior. Using the previous best model configurations, our agent managed to solve the environment with a high reward of 375 points, bypassing over dozens of neighbouring vehicles, and surviving the whole 500 steps.

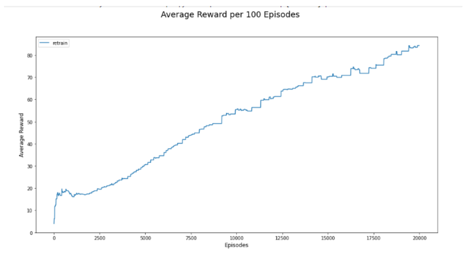

Figure 6: Training rewards on Highway-Env Easy

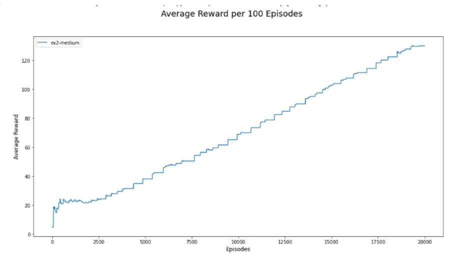

Figure 7: Training rewards on Highway-Env Medium

3.0.3 EX3 - Super Highway Agent

In this section we were asked to train a ”Super” agent who knows how to handle three environments: the highway, merge and roundabout environments. We've started experimenting these three environments individually.

The highway environment consists of 3 lanes with more vehicle count and density than EX1, solved easily with a high reward of 371 points, surviving the whole 500 steps. Next, over the merge environment, our agent starts on a main highway but soon approaches a road junction with incoming vehicles on the access ramp. In this scenario, our agent managed to maintain a high speed while making room and bypassing the vehicles over the road junction, so that they can safely merge in the traffic.

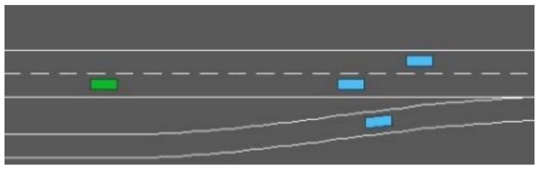

Figure 8: Merge environment

Finally, over the roundabout environment, our agent approached a roundabout with flowing traffic, and handle lane changes and longitudinal control to pass the roundabout as fast as possible while avoiding collisions. This scenario is more challenging, however, our agent managed to cope it nicely, and pass the roundabout safely.

It is worth to note that the main idea regarding the merge and the roundabout environments is passing the junction and the roundabout safely. Therefore, the duration over these environments is shorter than the previous highway environments, and consequently the reward which stands on 26 and 85 points (respectively) is lower.

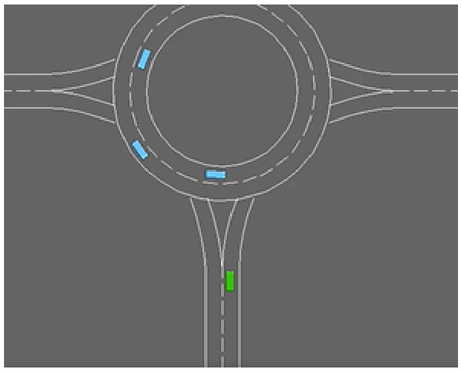

Figure 9: Roundabout environment

After we proved that each environment can be solved separately, we've moved to the challenging part; training a "Super" agent who knows to handle all three of the environments. In this section we've conducted three experiments.

Over the first experiment, our strategy was to train the network on the first environment for a short period, then start a new training session using the previous network weights on the next environment, and keep doing so over all three environments in a loop. Unfortunately, this strategy worked poorly, it seems that starting a new training session for each environment is overwriting most of the previous behaviour, this is probably has to do with high exploration when moving between environments.The naive solution didn't transfer the parameters between models.For future work it worth checking this strategy with transforming parameters such as epsilon between environments.

Next, in order to solve the previous issues, we've decided to train one long training session using all three environments. In order to do so, we've updated our buffer to keep collecting observations from all three environments. we've let the buffer to collect 500 observations from each environment before switching to the next environment, when the transition between one environment to the next is done randomly. Now, our buffer will consist with tens of thousands of observations from all three environments, and by using prioritized replay our model will replay the transitions with the higher expected learning progress. Additionally, since we are using one training session, that strategy can be easily measured. As a result, the agent managed to solve the highway and merge environment quite nicely, surviving all 500 steps over the highway environment, and bypassing the vehicles over the road junction. Unfortunately, the agent had difficulties to solve the roundabout environment.

Finally, on our third experiment, we've tried to fill the buffer with roundabout environment's observations more frequently. Now, the transition between one environment to the next won't be totally random, but rather be chosen by a the following probability: 60 percent out of the roundabout environment's observation, and 20 for each of the other environment's observations. Also, since we're trying to train three environment over one training session, we've decided to extended our time steps to 30K iterations. Although the a dedicated trained model showed good results on the highway environment, the agent had difficulties to cope with both the roundabout and the merge environment. That might suggest that the buffer modification might lead to a lower performance over the merge environment, probably following the lack of merge environment's observations.

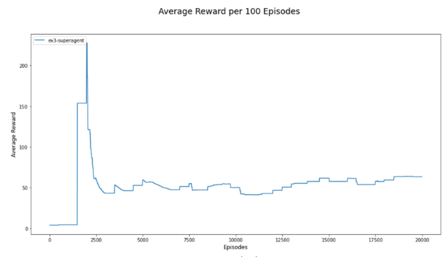

Figure 10: Training Supper agent with one buffer (second trial)

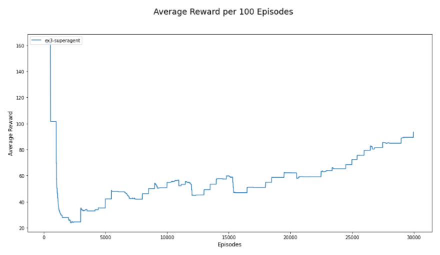

Figure 11: Training Supper agent with prioritized roundabout (third trial)

To summarise, our agent managed to successfully solve all three environments separately. Regarding the 'super' agent, we can conclude that the second experiment lead to the best results, solving both the highway and the merge environments nicely. Finally, although we were resulted with a high reward over the roundabout environment, we still were not pleased with the agent behaviour, and it might be that the roundabout environment's reward system lead the agent to mostly choose an idle behavior.

4 Discussion

In this paper we've conducted a comprehensive study regarding the highway environment. We've started with a short explanation about the environment, and briefly present the theoretical background regarding the different DQN extensions. Then we've continued with some technical experiments, we've conducted a large number of experiments to tune our hyper-parameters, and to find which algorithmic extension improve our model. By doing so, we've significantly improve our baseline model. Next, we've tested the best model over a variety of environment configurations. Fortunately, our agent managed to cope the tasks nicely, reaching high speed while avoiding collisions with neighbouring vehicles.

Next, in a future work we would examine advanced reinforcement algorithms for the comparison part (such as A2C, PPO, etc.) and test how the DQN algorithm is copping with the parking, intersection and the racetrack environments.

5 Code

Link to colab notebook can be found here.

References

[1] Marc G Bellemare, Will Dabney, and Rémi Munos. A distributional perspective on reinforcement learning. In *International Conference on Machine Learning*, pages 449–458. PMLR, 2017.

[2] Edouard Leurent. An environment for autonomous driving decision-making. `https://github.com/eleurent/highway-env`, 2018.

[3] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[4] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

[5] Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning, 2015.

[6] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Hasselt, Marc Lanctot, and Nando Freitas. Dueling network architectures for deep reinforcement learning. In *International conference on machine learning*, pages 1995–2003. PMLR, 2016.

https://www.udemy.com/course/artificial-intelligence-reinforcement-learning-in-python

https://www.udemy.com/course/deep-reinforcement-learning-in-python

https://www.udemy.com/course/cutting-edge-artificial-intelligence

https://www.udemy.com/course/practical-ai-with-python-and-reinforcement-learning

https://github.com/Officium/RL-Experiments/tree/a82c9be9187fe08fd8108ec23956e5bb8d5a6efc
https://medium.com/mlearning-ai/deep-q-learning-with-pytorch-and-openai-gym-the-taxi-cab-puzzle-e7a3028f732

https://pradeepgopal1997.medium.com/mini-project-2-2cafa300895c

https://towardsdatascience.com/double-deep-q-networks-905dd8325412

https://towardsdatascience.com/dueling-deep-q-networks-81ffab672751