CITY
UNIVERSITY OF LONDON
EST 1894

DEEP LEARNING: OPTIMIZATION INM707

By Elisa Chimenton & Hesam Rafiei

CITY UNIVERSITY OF LONDON
MSc Data Science

1 Domain & Task

The purpose of this project is to train an agent using reinforcement and deep learning techniques to solve basic and advance tasks. The initial (basic) task's purpose is to train an agent to find the most efficient route from Paris-Charles De Gaulle airport to Eiffel Tower in Paris. Moreover, the data were extracted randomly using the Paris underground map with different routes using trains and buses. The decision of agent is dependent on factors including the shortest duration and the most economical route combined. The figure below illustrates a specific map for different routes to the final destination with an indication of train stations which are the states. Moreover, the environment is deterministic and does not change over time (Static) and the current state will determine the available next states. The actions executed by the agent is stochastic.

The advanced task which is Cartpole V-1 using openAI gym environment will be discussed later in the report.

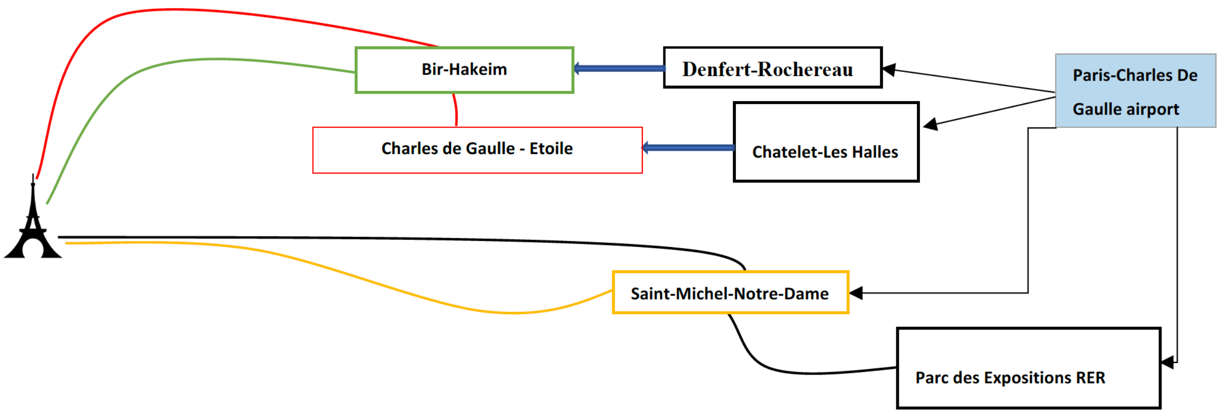


Figure 1. The navigation task map

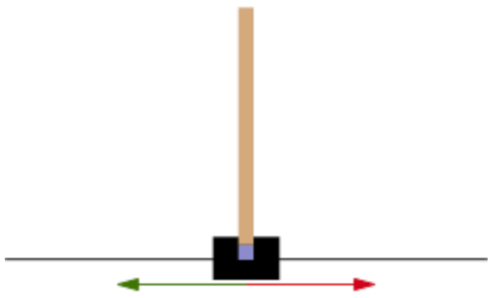


Figure 2. The advanced task

2 State transition & Reward function

In this specific task, as mentioned above the goal is to train the agent to find the optimal journey from the airport to Eiffel tower by earning the maximum reward.

2.1 State transition function

The state transition function defined as a function which predicts next state after the action is taken. With regards to the chosen problem and environment, the function is expressed as $\boldsymbol{s(t+1) = \delta(s(t), a(t)). s(t+1)}$ which indicates the next state is defined from the current action and state. Table 1 indicates an example of this task with its starting position (current states), actions (train/bus) and the next available states.

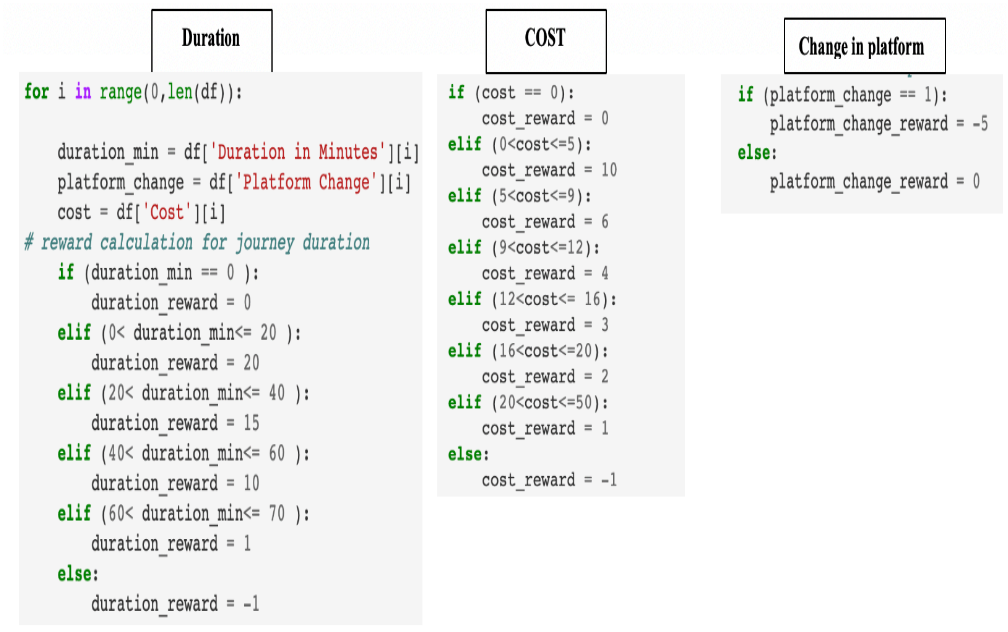
STATE (S)	**ACTION (A)**	**NEW STATE S(T+1)**
AIRPORT	Train to Denfert-Rochereau	Denfert-Rochereau
AIRPORT	Train to Chatelet - Les Halles	Chatelet - Les Halles
AIRPORT	Train to Saint-Michel-Notre-Dame	Saint-Michel-Notre-Dame

Table 1. Example of state transition function

2.2 Reward function with graphical representation

As we have implemented Q learning algorithm for this task; Appropriate rewards will be received during different time periods as the agent takes actions and move to the next state. The reward system defined equally by duration, cost and the changes in the platform. The figure below defines how the reward values were assigned. All of the statements were run by iteration.

After generating rewards for each state using the method above, additional rewards were added for the agent arriving at Eiffel tower (+200) and +20 rewards for the states which are one station away from the final destination. The final reward scores can be seen below.



Duration

```
for i in range(0,len(df)):

    duration_min = df['Duration in Minutes'][i]
    platform_change = df['Platform Change'][i]
    cost = df['Cost'][i]
# reward calculation for journey duration
    if (duration_min == 0 ):
        duration_reward = 0
    elif (0< duration_min<= 20 ):
        duration_reward = 20
    elif (20< duration_min<= 40 ):
        duration_reward = 15
    elif (40< duration_min<= 60 ):
        duration_reward = 10
    elif (60< duration_min<= 70 ):
        duration_reward = 1
    else:
        duration_reward = -1
```

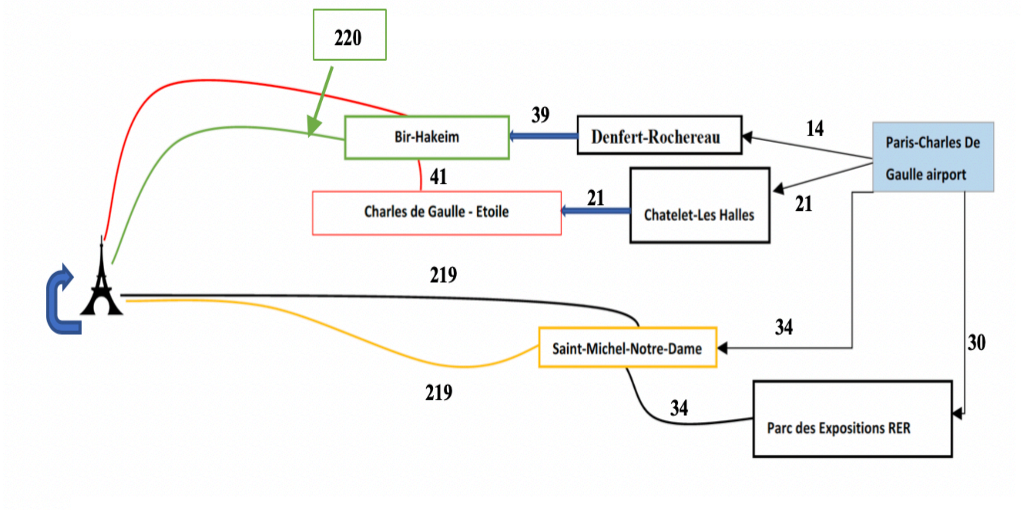
COST

```
if (cost == 0):
    cost_reward = 0
elif (0<cost<=5):
    cost_reward = 10
elif (5<cost<=9):
    cost_reward = 6
elif (9<cost<=12):
    cost_reward = 4
elif (12<cost<= 16):
    cost_reward = 3
elif (16<cost<=20):
    cost_reward = 2
elif (20<cost<=50):
    cost_reward = 1
else:
    cost_reward = -1
```

Change in platform

```
if (platform_change == 1):
    platform_change_reward = -5
else:
    platform_change_reward = 0
```

	CDG_airport	Denfert-Rochereau	Chatelet - Les Halles	Saint-Michel-Notre-Dame	Parc des Expositions RER	Bir-Hakeim	Charles de Gaulle - Etoile	Champ de Mars - Tour Eiffel
CDG_airport	0.0	14.0	21.0	34.0	30.0	0.0	0.0	0.0
Denfert-Rochereau	0.0	0.0	0.0	0.0	0.0	39.0	0.0	0.0
Chatelet - Les Halles	0.0	0.0	0.0	0.0	0.0	0.0	21.0	0.0
Saint-Michel-Notre-Dame	0.0	0.0	0.0	0.0	0.0	0.0	0.0	219.0
Parc des Expositions RER	0.0	0.0	0.0	34.0	0.0	0.0	0.0	0.0
Bir-Hakeim	0.0	0.0	0.0	0.0	0.0	0.0	0.0	220.0
Charles de Gaulle - Etoile	0.0	0.0	0.0	0.0	0.0	41.0	0.0	0.0
Champ de Mars - Tour Eiffel	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0



3 Choosing a policy

The policy is a function of the state (St) and action (At) which generates the probability of taking an action given the state to define the way the agent acts. In order to gain maximum rewards on each state, the agents need to discover the optimal policy. The epochs parameter has been set to 200 in which the agent will go through the full journey 200 times. Considering that the agent is in a model-free world with no previous knowledge of it, the policy will require some exploration instead of using a greedy policy(ε=0) which only take advantage of existing information to optimise immediate rewards (exploiting).Therefore, the ε-greedy approach was used and the value was initialised to be 1 to explore. Moreover, a number has been produced randomly within the range of 0 and 1 which defines when to explore or exploit. For instance, the agent will exploit by selecting the action with the largest reward when the random number is equal or greater than the ε value. The decay parameter has also been set to 0.8 which guides the agent to explore at first and then gradually exploit as it gains knowledge through exploration.

4 Q learning parameters

- **Learning rate (a) = 0.5**: which define how much the latest or new information overrides the earliest or old information. For instance, the agent will maintain only the previous knowledge and does not update the Q matrix with the new information from the current state, when the learning rate is zero. On the other hand, if the learning rate is 1 the agent will only focus on the latest information and generate a deterministic environment.
- **Discount rate(γ) = 0.5:** defines the current value of future which is in range of 0 and 1 When the discount rate is zero, the agent focuses on maximising the immediate return

by taking suitable action (myopic). As the discount rate increases, the agent focuses on long term rewards, in contrast to immediate rewards.

- **Learning policy (π) = epsilon-greedy:** This policy is based on initial exploration and it will gradually decrease into exploitation state after completion of each episode, by multiplying decay factor (0.8) to epsilon.

5 Update of Q matrix

Since we have implemented the Q learning algorithm; the below equation (Bellman) was used in order to update Q matrix.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_{a} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right]$$

- This equation generates values of expected long term reward given the current state and assuming to take the best possible actions.

5.1 Example of updating Q matrix for the current task

5.1.1 Variables (refer to reward matrix on figure 4)

- Initial position (state): CDG
- Possible actions form the initial state: A = (**Denfert-Rochereau, Chatelet-Les Halles, Saint-Michel-Notre-Dame, Parc des Expositions RER)**
- Choosing a random action (a) to move to the next state (e-greedy policy):
  a = train to Denfert-Rochereau
- Next state (s+1) by taking action **a**: → s+1 = Denfert-Rochereau

Possible states and action from the next state (S+1): → A+1 = **train to Bir-Hakeim**

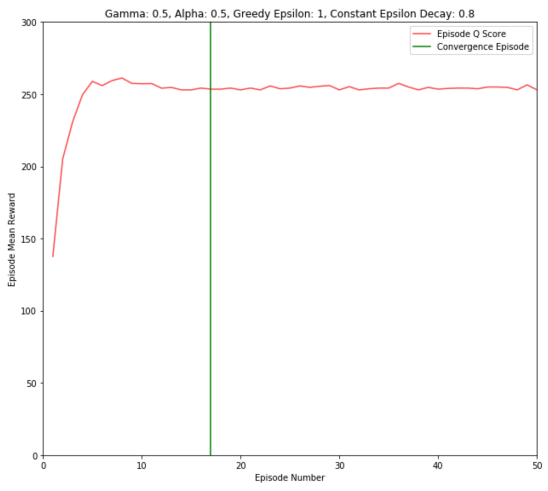
5.2 Updating Q matrix

- Q (CDG, Denfert-Rochereau) = 0 + 1 x [R+1 = 14 + 0.5 x max (0,0)] = 14

6 Analysis of result

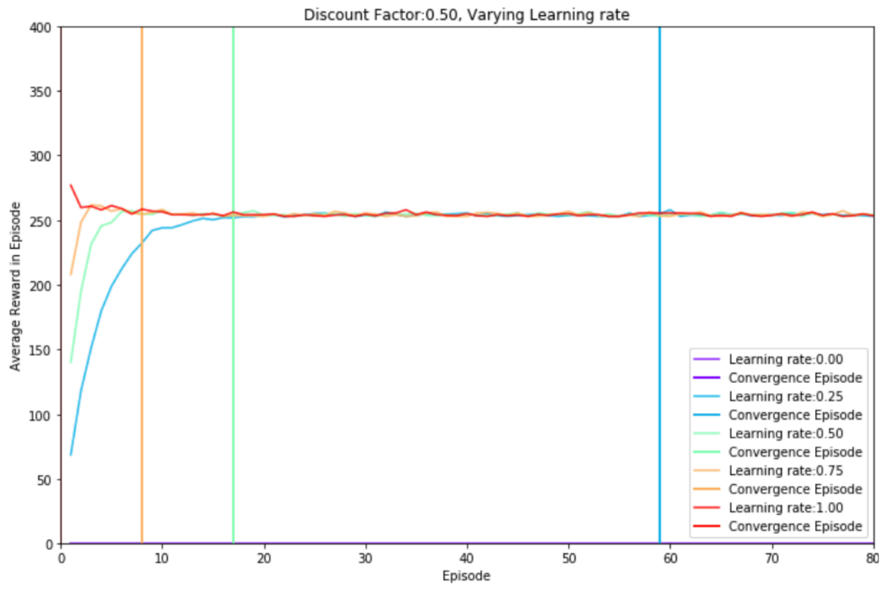
6.1 Evaluation metric and performance

The behaviour of the agent is controlled by the parameters provided to the Bellman equation and the mentioned policy. The grid search technique was used to find the optimum parameters including learning rate, discount rate, decay factor and epsilon by observing the impacts on the speed which the maximum mean episode Q score was achieved. Moreover, the convergence parameter which was set to 15 is the evaluation metric of the above analysis. This value indicates after 15 episodes if the Q values do not alter, the algorithm reached the convergence.



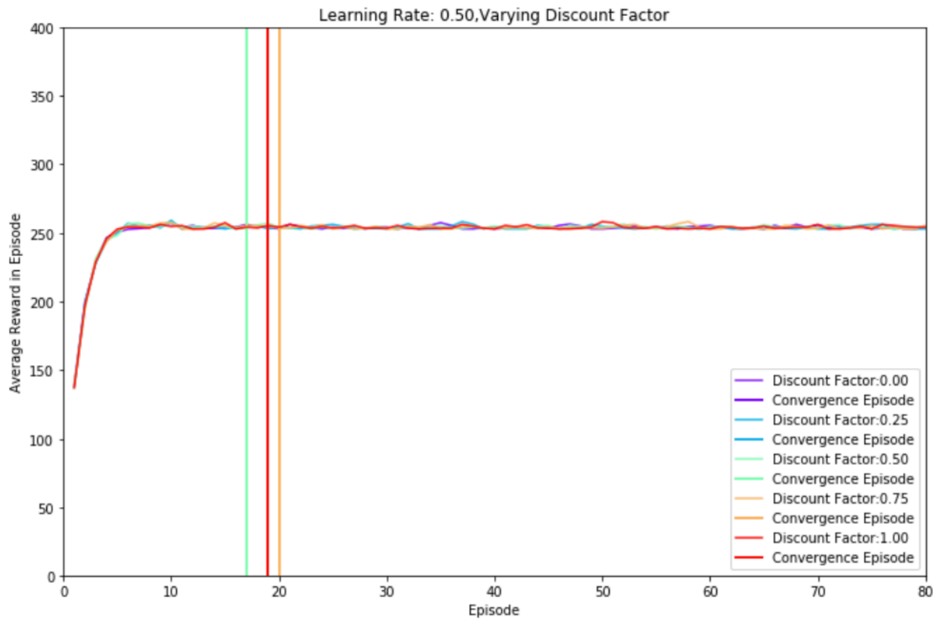
6.2 Learning rate variation with constant discount rate

As mentioned before, the grid search technique was used to observe the effects of different learning rates on the episode mean reward and the convergence of the algorithm over 200 episodes. The figure below indicates when the learning rate is zero, the Q matrix does not change(zeros), and the algorithm converges quickly. When greater learning rates were introduced, the average reward trend in the early episodes grew increasingly steeper which indicates a quicker updates of Q matrix values in contrast to lower learning rate which indicates a smoother curve.



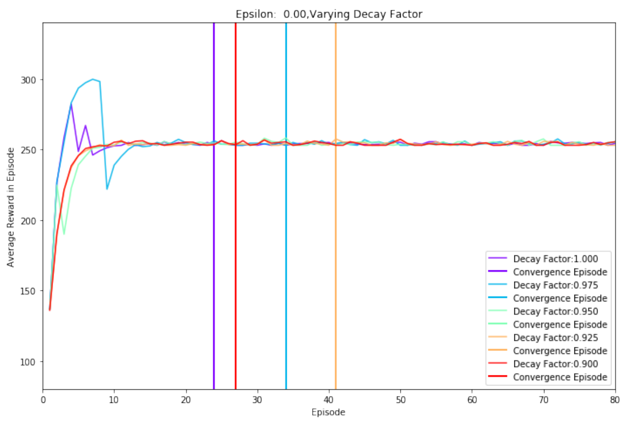
6.3 Discount rate variation with constant learning rate

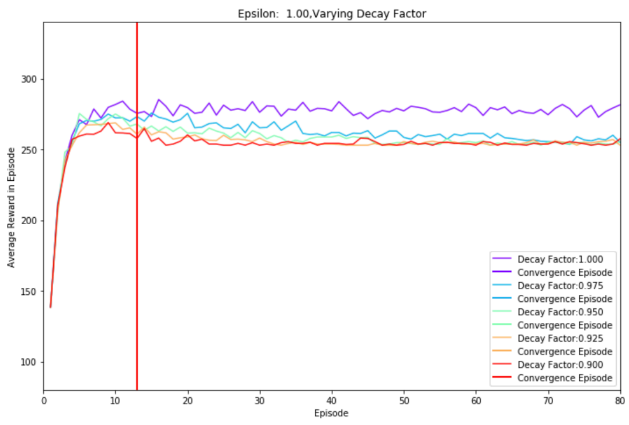
As mentioned, the discount rate values are within the range of 0 and 1. We set the values to increase incrementally by 0.25. However, the discount rate did not affect the mean reward curve and the algorithm convergence which can be influenced by the simplicity of the task.



6.4 Epsilon variations with constant decay factor

The figure below illustrates the effects of different policy parameters on the convergence speed and the average rewards on the episode. By having a quick glance, when the epsilon parameter is 1 and the decay factor is 1, the algorithm converges quicker compare to when epsilon is 0 which the decay factor is not effective. Furthermore, the steeper gradient on average reward curve can be seen when epsilon is 1 which indicates the only fast exploration state and no exploitation afterwards (converged after exploration).





7 Advanced Task

Cartpole- v1 from Open AI Gym environment was the task chosen to Advanced Task. In Cartpole, the environment is deterministic, the agent has to decide between two actions (moving the cart left or right (+1 or -1) with the objective that the pole attached to it stays upright. There are four states (cart position, cart velocity, pole angle and the velocity of the tip of the pole). As the agent observe the current state of the environment and chooses an action and get a reward of (+1) in consequence of this action and still getting it for every incremental timestep, the episodes end if the pole is 15 degrees from vertical, or cart 2.4 units from the centre.

In this part of coursework we will apply Deep Q-Learning algorithm (DQN) and Double Q learning, as we saw in the previous task we created a lookup table (matrix) and values of Q are updated after specific agent action as this example of shortest duration and the more economic price had only a few routes Q-learning worked well. However, this approach is slow and does not scale up in more complex environments, thus it is not feasible from computational and storage perspective.

The main goal of reinforcement learning is optimizing a cumulative future reward signal and it is also the goal of Q-learning function however studies observed that estimation values of Q-learning many times were overestimated, to overcome this problem DQN was created combining Q-learning with a flexible deep neural network because it provides flexible function approximation with the potential for more realistic estimation error. The predictions of DQN is used to create the target values (Mnih et al. 2015). Since DQN was proposed similar approaches were developed by the research community to supplement this idea of DQN. In 2015, another method called Double Q-learning was presented by Van Hasselt (2015) it proposes to be more generalizable to work with deep neural networks and have more accurate value estimates.

Double Q-Learning consists of two value functions, the first one works as an estimator which is trained and referred as the *Online Network* and it evaluates the greedy policy, then the second one *Target Network* is used to estimate the target values (van Hasselt et al., 2015). The weights of Online Network is transferred to Target Network often to keep it updated. Moreover, we used another methodology called experience replay (Lin et al., 1992) which stores the agent's experience in memory. These experiences are randomly sampled from memory and used to train the neural network. These types of learning consist of two steps – first gain experience and second update the model. At fixed intervals called batches, these experiences are sampled from the memory. The size of memory controls the number of experiences will be updated by the network. Using this approach agent can learn from his experiences and make better decisions.

The advantage of *Experience replay* in Reinforcement Learning is to reduce the quantity of exploration, however, this methodology uses more computation and more memory still being cheaper than agents' experience with the environment itself (Schaul et al., 2016). To improve even more this methodology, Schaul et al. (2016) developed the *Prioritized Experience Replay*, seeing that not all experience are important, the goal of this approach is to prioritise some transitions that are more relevant than others and it was made through sampling according to a probability distribution, then calculating agent's temporal difference (TD) error.

For implement this advanced task, we first selected as baseline algorithm a random search implementation that helped with evaluation of other implementations, it also improved our understanding on how to use Open AI Gym environment as this task is different from the first task that the environment was created by ourselves. Then, next step we combine Q-Learning with 'deep' neural network known such as Deep Q-Learning (DQN) (Mnih et al. 2015) and Double DQN (van Hasselt et al., 2015), Experience replay (Lin et al., 1992) and Prioritized Experience Replay (Schaul et al., 2016) and compared their performance.

We expect that the results of DQN and Double DQN Agent perform better than Q-Learning because Q-learning approximations using only one sample at a time is not effective, and we believe that using experience replay the model should be more efficient. We also expect improvements from DQN to Double DQN, as proposed by van Hasselt et al, (2015) DDQN would have better estimation values and also higher scores. We considered these approaches much more scalable as they also use continuous input, expanding the chance of better performance in realistic problems domain.

8 Analysis and Results

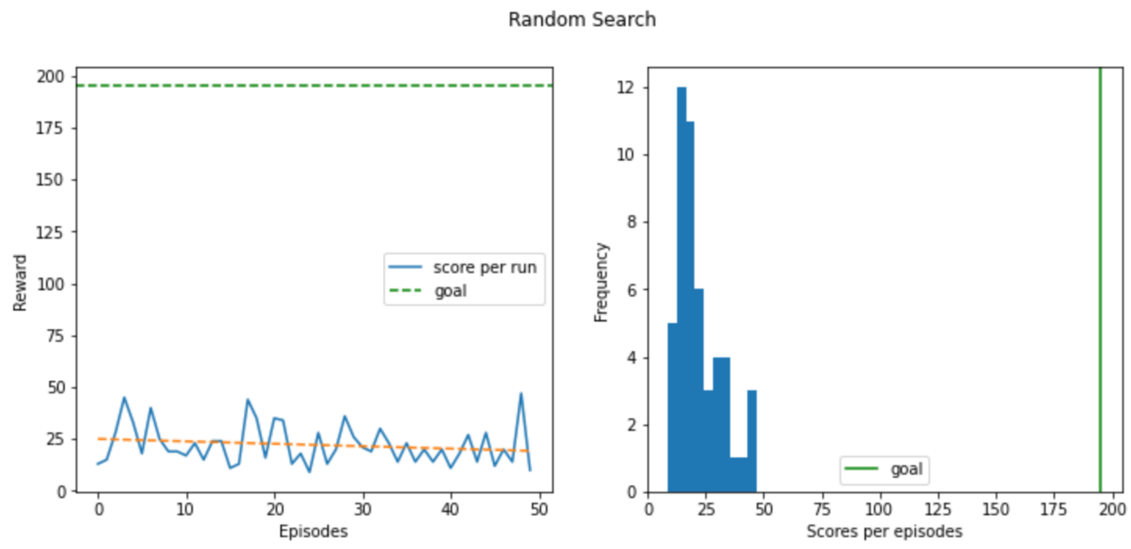
We trained the agent with following parameter:

- Env – represents the Open AI Gym environment for Cartpole-v1.
- Episodes – number of times it is going to play, Random Search was set for 50 episodes and the following algorithms (Q-learning, DQN, DDQN) for comparisons was set 150 episodes.
- Gamma – the discount factor was set 0.9. The idea is that the future rewards worth less than immediate rewards.
- Epsilon is the probability of choose an action at random otherwise choose greedy action.
- Epsilon decay was chosen according to DQN research (van Hasselt et al., 2015) and set 0.99.
- Learning rate for DQN was set 0.001 and 50 hidden nodes.

- DDQN – This model is very similar to DQN differing only in the last part of equation which we find the index for the highest Q-value and use it to acquire the action of Target Network.

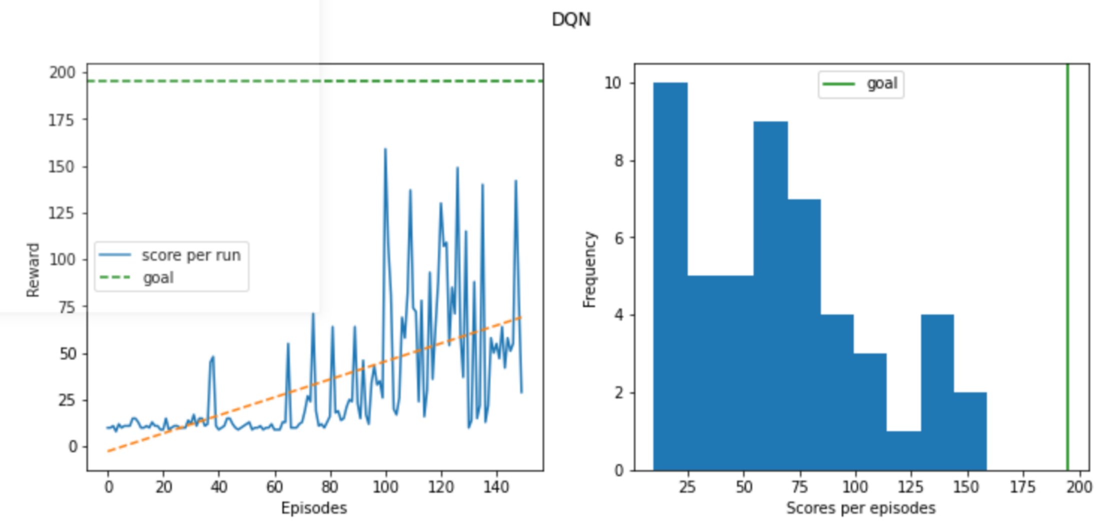
8.1 Random Search

As expected, this baseline shows that the agent is not learning from his experiences and their average performance is low, have not achieved the goal and getting reward of maximum 75.



8.2 DQN

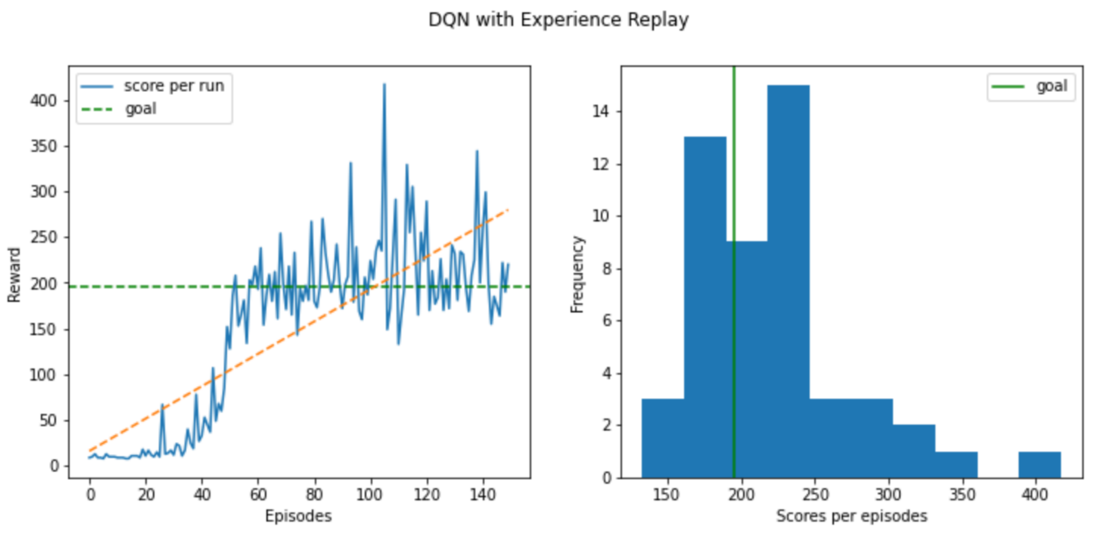
The DQN was implemented in a 2 layers network in Pytorch using Leaky Relu activation function, we also used mean squared error as loss function and Adam as our optimizer. Adam is first-order optimization algorithm that normally is used in problems with large amount of data or states. It was divided in 2 methods (predict, update) the neural network uses the agent's state as an input and predict the Q values from the actions and another method update the weights of the network after a training sample.

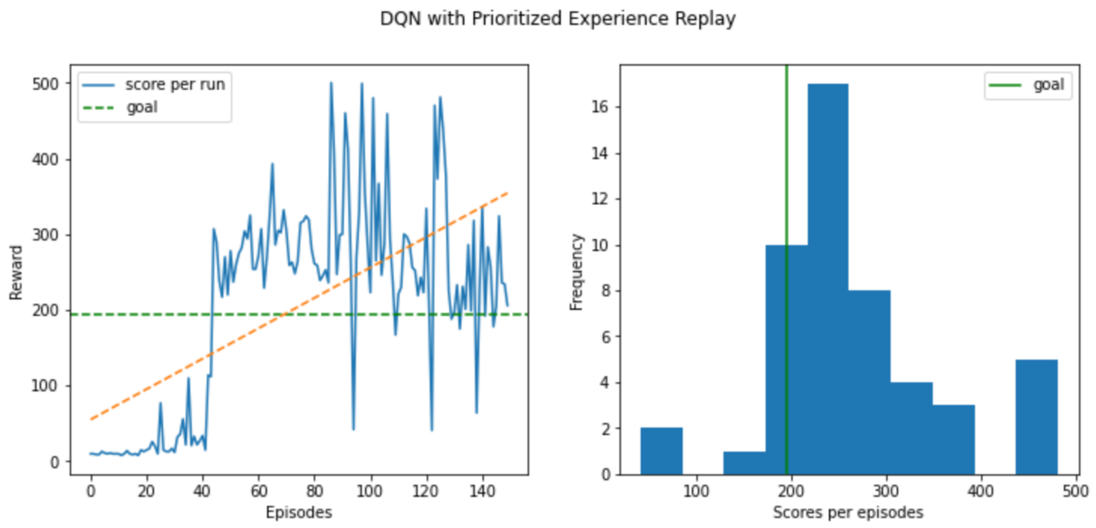


As expected, the performance of the agent (figure above) has improved. The reward was 175 much higher than random strategy also the trend line showed positive but the agent still without reach the goal after 150 epochs.

8.3 <u>DQN with experience replay and prioritized experience replay</u>

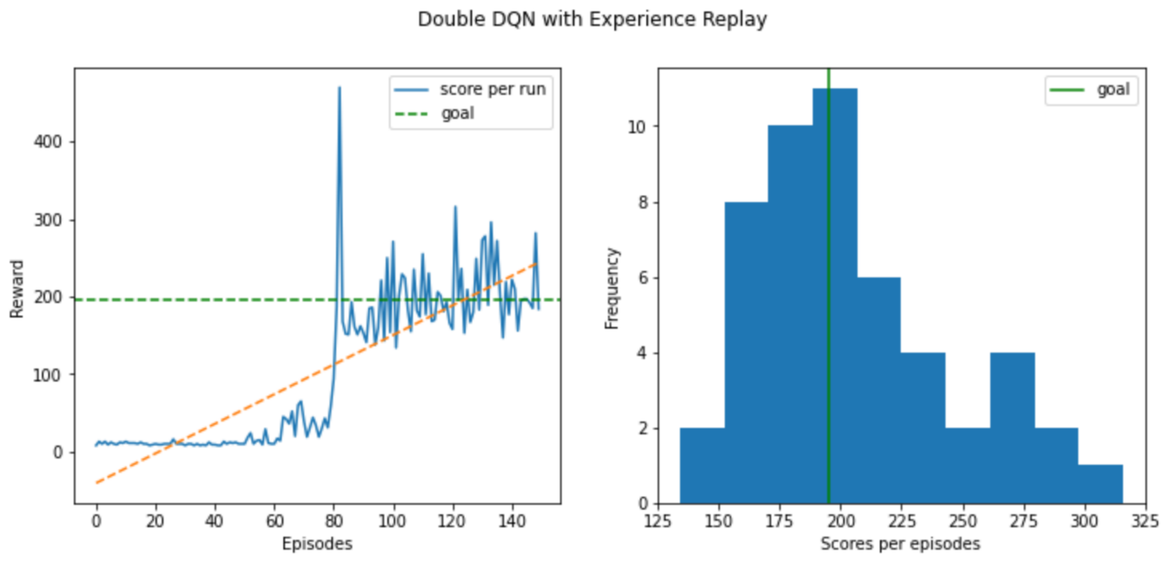
We trained a DQN-Agent with learning rate of 0.001, to run only DQN worked fine, however, to run DQN with experience replay MacBook Pro – Intel core i5 CPU was not enough, it stopped many times and the solution was run it in Google Collab. In Experience replay, batches of experiences are randomly sampled from memory and it is used to train the neural network. The experience is an array of (state, action, reward, next-state, done), instead of the agent learn from exploration, he picks his experience from the buffer.





The graphs above shows that our expectation was right, the neural network with experience replay performed much better and showed be a little more robust compared with Q-Learning and DQN only approach, after almost 60 episodes in first figure and 40 episodes in the second figure the agent reached the goal which was impossible for other approaches and it repeated many times, the rewards also was higher and it repeated many times, the trend line demonstrated this improvements.

8.4 DDQN with experience replay



The double DQL showed more robust compared with previous methodologies, this significant improvement seems to be related to separation of action and evaluation, it leads to more effectively reach the goal and remaining there longer, demonstrating be more stable and reliable learning.

9 Conclusion

This work demonstrated how deep learning could be applied to solve different optimization problems. We used two examples, the first one finds the optimal journey using a lookup table which for this case taking random actions was fine, however for Cartpole v-1 example random actions as we saw above do not get good results. Implementation of DQN that is an extension of Q-Learning seems to be a good choice, especially including experience replay, we could improve performance and reach easily the winning threshold. DDQN showed to be more robust than previous implementations. This example can generalize easily in other environments.

10 References

[1] Achiam, J., Knight, E. and Abbeel, P. (2019) 'Towards Characterizing Divergence in Deep Q-Learning', *arXiv:1903.08894 [cs]*. Available at: http://arxiv.org/abs/1903.08894 (Accessed: 26 May 2020).
[2] van Hasselt, H., Guez, A. and Silver, D. (2015) 'Deep Reinforcement Learning with Double Q-learning', *arXiv:1509.06461 [cs]*. Available at: http://arxiv.org/abs/1509.06461 (Accessed: 26 May 2020).
[3] Hessel, M., Modayil, J., van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M. and Silver, D. (2017) 'Rainbow: Combining Improvements in Deep Reinforcement Learning', *arXiv:1710.02298 [cs]*. Available at: http://arxiv.org/abs/1710.02298 (Accessed: 26 May 2020).
[4] Lin, L.-J. (1992) 'Self-improving reactive agents based on reinforcement learning, planning and teaching', p. 29.
[5] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S. and Hassabis, D. (2015) 'Human-level control through deep reinforcement learning', *Nature*, 518(7540), pp. 529–533. doi: 10.1038/nature14236.
[6] Paszke, A., (2019). Reinforcement Learning (DQN) tutorial. Retrieved from: https://pytorch.org/tutorials/intermediate/reinforcement_q_learning.html
[7] Schaul, T., Quan, J., Antonoglou, I. and Silver, D. (2016) 'Prioritized Experience Replay', *arXiv:1511.05952 [cs]*. Available at: http://arxiv.org/abs/1511.05952 (Accessed: 26 May 2020).