Oscaro internship report : Reinforce recommendation system

Clément Bernard

September 13, 2020

Contents

1 Introduction

Oscaro is a french e-commerce company specialised in the sale of new automotive parts from manufacturers and wholesalers.
As common e-commerce websites, recommendations are available after each 'add-to-basket' to suggest new products for the client. The aim of these recommendations is to increase the final basket of the client and indeed the company's margin.
Therefore this 2 months internship aimed to create a recommendation system. Instead of using classic methods of recommendation (like collaborative filtering or content-based filtering), I used reinforcement learning approach to deal with this problem.
I implemented three different algorithms : *Q-learning*, *Double Deep Q-Network* (Double DQN) and *Deep Deterministic Policy Gradient* (DDPG). Nevertheless, I didn't manage to optimise the hyperparameters of the last two algorithms, which led to non optimal results.

2 Oscaro

Oscaro is a french company founded in 2003. It deals with the sale of automotive parts. The aim is to offer a large type of car products (that are coming from their manufacturers) to help either the specialists or the beginners for the purchase of automotive parts. The website is available in France, Belgium, Spain and Portugal.

The website is structured as follows : the main page offers an interface where the client can enter its license plate. Once the license plate is entered, the website will automatically update the products displayed to be adapted to the vehicle. Otherwise, the customer can choose not to enter its license plate, and therefore choose himself the car model he wants to explore. Then, the website displays a list of automotive parts (for example : car oil). For each category, there are sub lists that contain **generic article (genart)**.

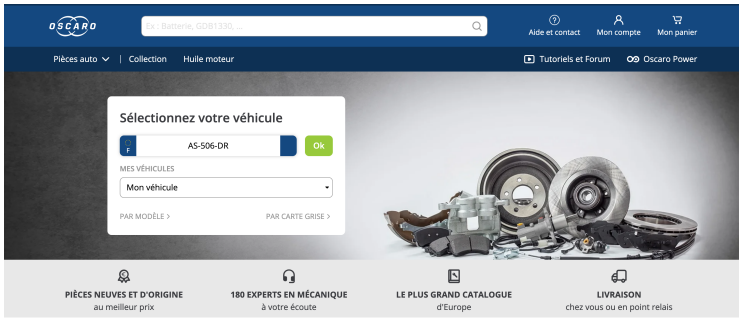


Figure 1: Home page of Oscaro.com

Cross-sell Once the client has entered in a genart page, he can choose a product among a list of same articles (that are coming from different manufac-

turers). Then, if he chooses to add to basket the product, a pop-up opens (the **cross-sell**) with recommendations.

Figure 2: Popin where the recommendations take place

This is all we need to understand the project, which mainly focus on the genarts and the cross-sell to create a new recommendation system.

3 Reinforcement learning

First of all, I've worked on classic reinforcement learning algorithms with OpenAI Gym libraries in order to have a first implementation of the algorithms.

Definition Reinforcement learning is a subclass of machine learning where it learns how to map situations to actions in order to maximise rewards. The learner is not told which action to do, and therefore will learn what actions will lead to high rewards through interactions with the environment.

Key elements A classic modelisation of a reinforcement learning problem uses several main elements : agent (the learner that takes actions), actions (decisions that will impact the environment) at time t (denoted A_t), environment (where or what is interacting with the agent), state (observation of the environment) at time t (denoted S_t) and reward (integer that gives feed-backs for choosing an action A_t given a state S_t) at time t (denoted R_{t+1}).

This is summarized in the following schema :

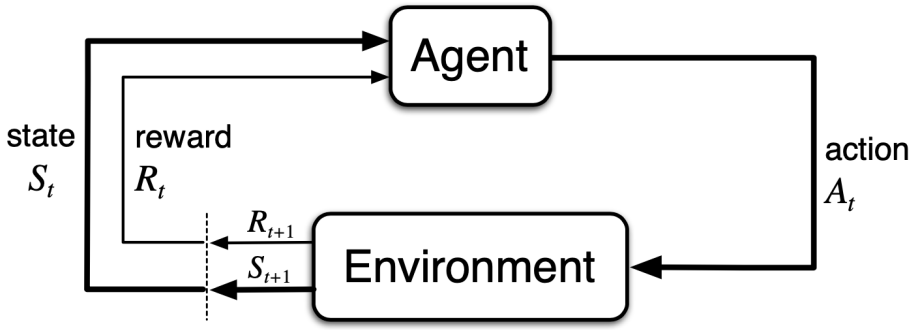


Figure 3: Reinforcement learning schema

3.1 Q-learning

The Q-learning algorithm is a tabular method that approximates the optimal (state,action) pair with the given recursive formula :

$$Q_{t+1}(S_t, A_t) = Q_t(S_t, A_t) + \alpha(R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q_t(S_t, A_t))$$

where α is the learning rate and γ the coefficient that deals with how important are the future rewards.

It leads to the Q learning algorithm :

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
    $S \leftarrow S'$
  until S is terminal

Figure 4: Q-learning algorithm

Limits Nevertheless, the limit of Q-learning is that it doesn't work well for continuous action or state space. Furthermore, for high action or state space, the time of convergence is very high as it requires to try every pair (state,action) to have an idea of the cumulative rewards.

3.2 Double Deep Q-Network (Double DQN)

To deal with continuous state space, a classic approach is to use function approximation : approximate the Q-values with a function (like neural network for instance). Therefore, we can feed the neural network with a new representation of the state, and then the neural network can output the Q-values for each action (the action space should be discrete).

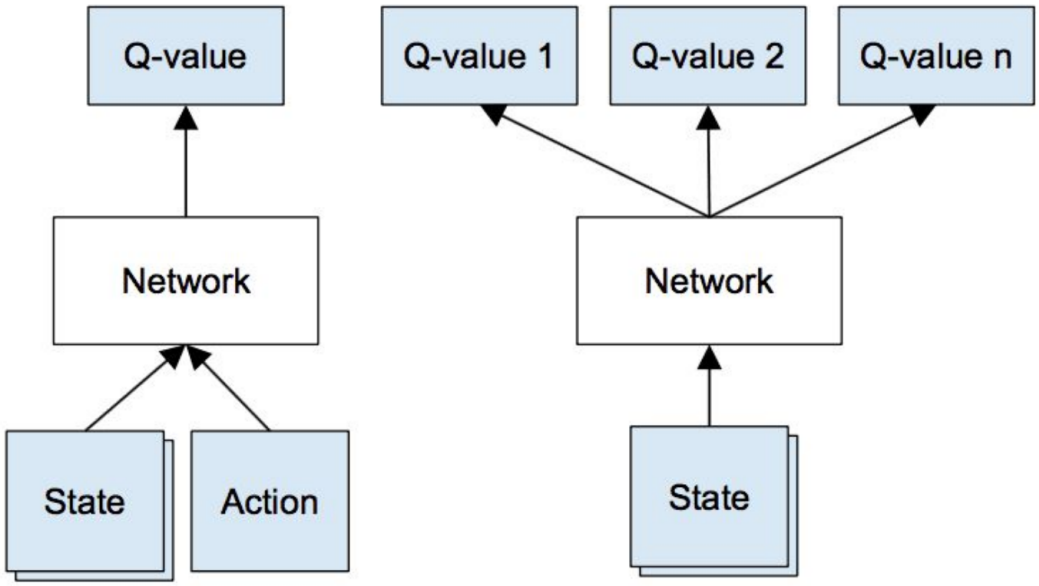


Figure 5: Deep Q-learning schema

As shown in Figure 3, there are two ways of approximation : we can either feed the neural network with a pair of (state,action) (and then the neural network will output the Q-value for this pair) or with the state (and the neural network will output the Q-values for all the different actions). I used the second representation.

Explanation The prediction, target and loss used in this method are the following :

$$prediction = Q(S_t, A_t)$$

$$target = R(S_t, A_t) + \gamma \max_a Q'(S_{t+1}, a, w)$$

where Q' is, for the double DQN algorithm, a copy of the neural network used for the prediction (which is updated every C iterations to the prediction neural network).

$$loss = (R(S_t, A_t) + \gamma \max_a Q'(S_{t+1}, a, w) - Q(S_t, A_t))^2$$

Limits This method of approximation of Q-values works well when the action space is not too large. Otherwise, it will be very time-consuming as it would require to predict Q-values for each action (which is quite large).

3.3 Deep Deterministic Policy Gradient (DDPG)

To avoid high computation for the Q-values, the DDPG algorithm separates the selection of the action from the evaluation of the expected rewards. It learns concurrently a policy (mapping from states to actions) and Q-values (expected rewards for a given pair (state, action)).
Here is the basic pseudo code for the DDPG algorithm :

Algorithm 1 DDPG algorithm

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ.
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer R
for episode = 1, M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state s_1
  for t = 1, T **do**
    Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
    Execute action a_t and observe reward r_t and observe new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in R
    Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
    Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
    Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
    Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}$$

    Update the target networks:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$
$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

  end for
end for

Figure 6: Deep Deterministic Policy Gradient pseudo code

3.4 Results

3.4.1 Environments

Cartpole The environment used for Q-learning and Double DQN algorithms is an environment from OpenAI gym : the *cartpole* (which is a car with a pole attached by an un-actuated joint). The aim of the agent is to learn how to stabilize the pole by moving the car either on the right or on the left. The

reward is set to +1 whenever the pole is not falling (more than 15 degrees from vertical).

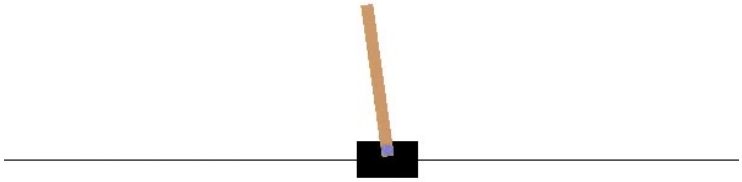


Figure 7: Cartpole environment

Q-learning The state space for this example is continuous (composed of four values for the car position, velocity, and two angles) and can't be applied directly to the Q-learning algorithm (as we represent a matrix for the Q-values for each pair (state, action)). To discrete the states, one way is to basically ignore the position and velocity of the car and to convert the angles to values between $[0; 6]$ and $[0; 12]$.

Double DQN As the state space is continuous and action space discrete, it was possible to apply this example to the Double DQN algorithm.

Pendulum The environment used for the DDPG environment is also one of the OpenAI gym library : the Pendulum. The aim is to stabilise the pendulum to the vertical by applying torque. The state representation is given by the pendulum speed and the angle, and the action is a number between [-2,2] that gives the torque given to the pendulum. Both action space and state space are continuous.

DDPG The state and action representations of this environment are continuous, therefore it can be directly enter to the DDPG algorithm.

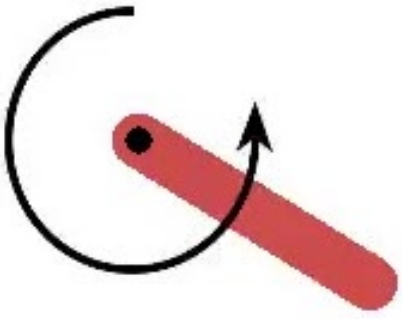


Figure 8: Pendulum environment

3.4.2 Rewards

For each algorithm, I stored the rewards per episode. It gives an idea of how good the decisions made by the agent were during this session.

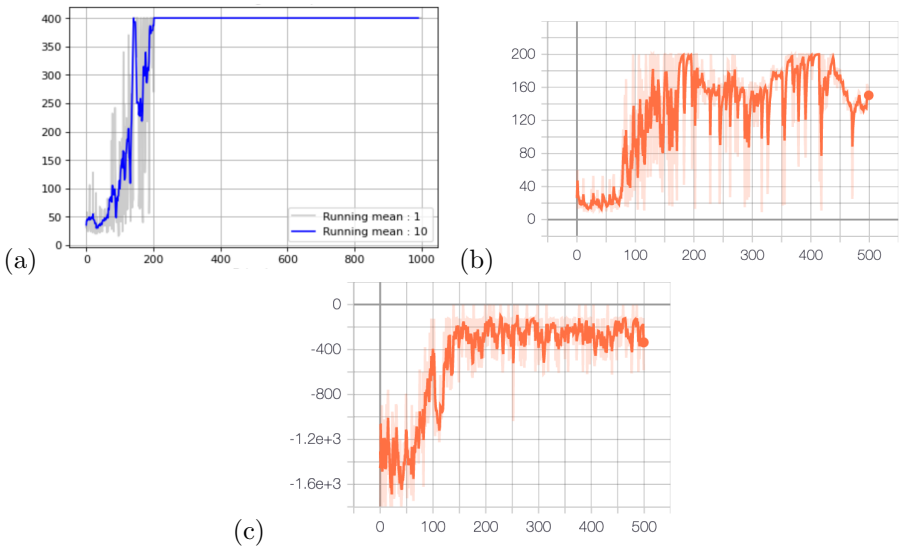


Figure 9: Rewards per session for (a) Q-learning with cartpole (b) Double DQN with cartpole (c) DDPG with pendulum

Figure 9 shows the results of the rewards : for each algorithm, the rewards

per session tend to increase, which means that the agent learns which actions to take in order to increase the rewards per session.

4 Oscaro modelisation

4.1 Reinforcement modelisation for recommendation

4.1.1 Markov Decision Processes

In order to understand well the concept of reinforcement learning for a recommendation, it requires to define the markov decision processes for recommendation.

Agent The agent is the algorithm that is going to take actions to maximise the rewards. Here the agent is our recommendation system.

Environment The environment is what interacts with the agent. Thus, the environment is the client who is consulting the oscaro website, and decides to accept or not the recommendation.

State A state in recommendation would be the content where the user is currently at. In our example, it would be the genart that is added to basket by the client.

Action An action is what leads to rewards. In our case, the action is 'suggesting a genart'. Another way would have been to suggest a list of genarts, but it would have increased significatly the size of the action space. In sake of simplicity, we consider that we only recommend one content.

Reward This is what we tend to maximise in our process. At the beginning, I have thought about a mix between :

1. Add-to-cart for the genart that is suggested
2. The final basket of the client

But after discussions with my manager, we have decided to only consider the final basket. Indeed, only considering the fact that the client adds a content to its basket would lead to maximising the length of the session, and not the total amount of the final basket. We aimed a system that leads to benefit for the company, and not a classic system of recommendation such as Spotify or Youtube (where the goal is to increase the duration of a session).

This is summarized in the Figure 10.

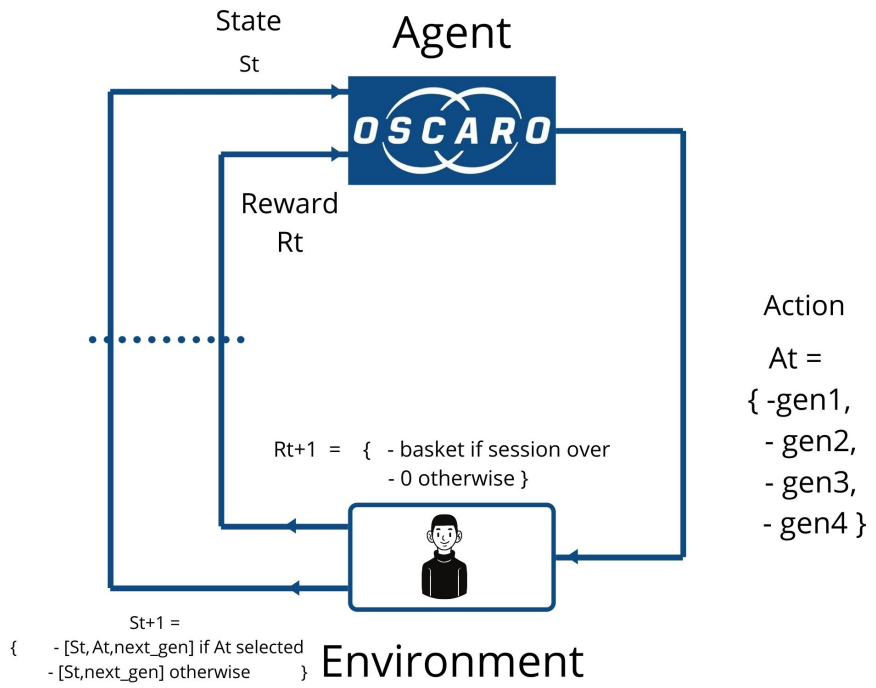


Figure 10: Oscaro recommendation modelisation for reinforcement learning

4.1.2 Session

For classic reinforcement learning algorithms (like the ones implemented above), the learning process is done online. Nonetheless, in our case, we can't afford the system to perform online as it will start badly and will cost too much for the company. To deal with it, we took the data stored from the recommendations that were done and trained the model offline.

In order to get the data required for the training process of the 3 algorithms, I worked on the translation of the data into tuples $(S_t, A_t, R_t, S_{t+1}, is\_done)$. To do so, we used the data from a google cloud table that stored the results of the past recommendations. It starts with the first state, which is the genart that will be added to the basket. Then, a popup of recommendation contents appear : this will be the actions (we create a pair (state,action) for each genart recommended, but with the same state). The reward is equal to 0 if the session isn't over and leads to another purchase, or the total amount of the basket if the session ends here. The next state is either the next genart that leads to create a popup, or nothing if the session ends here. Note that the reward is often 0 and equal to the final basket only at the end of the user session. Therefore, the algorithm shouldn't consider the immediate reward but the long-term reward, which means for our algorithm that the gamma value should be high (closer to 1 than 0).

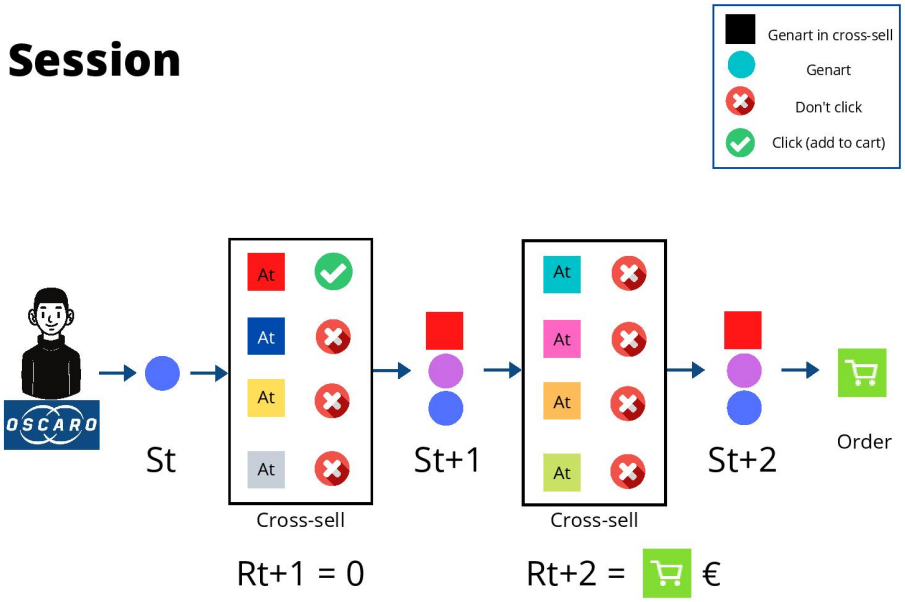


Figure 11: Translation of a user session into reinforcement learning modelisation

4.1.3 Genart vectorisation

For the Double DQN and the DDPG algorithms, the states need to be vectorized. To do so, I used the data available for the genarts. I plotted the distribution of each feature in order to one-hot-encode it. This is summarized in the figure 12 :

4.1.4 Client vectorisation

For the DDPG algorithm, we will use the vectorisation of the client in order to give information about the client himself. To do so, I used the data we got about each customer and I did the same process used for the state representation. This is summarized in the figure 13.

4.2 Algorithms adaptation

Q-learning It required no change to run it. The tuples $(S_t, A_t, R_t, S_{t+1}, is\_done)$ was enough.

Double DQN It only required to change the representation of the states, as explained above.

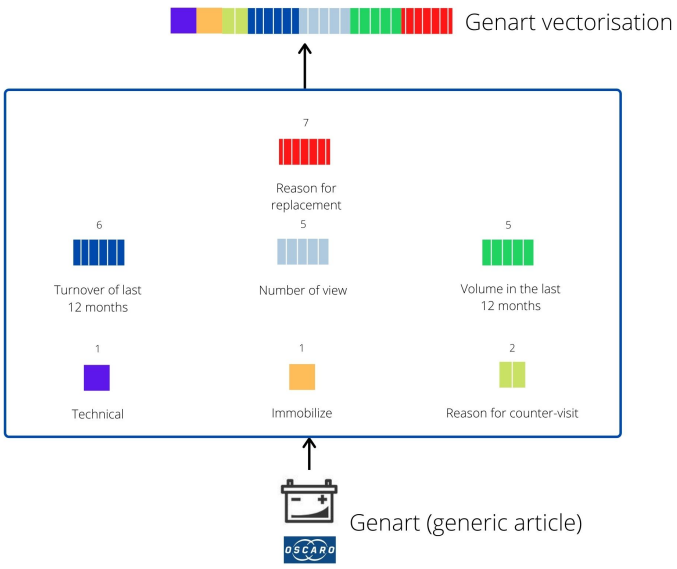


Figure 12: Genart vectorisation

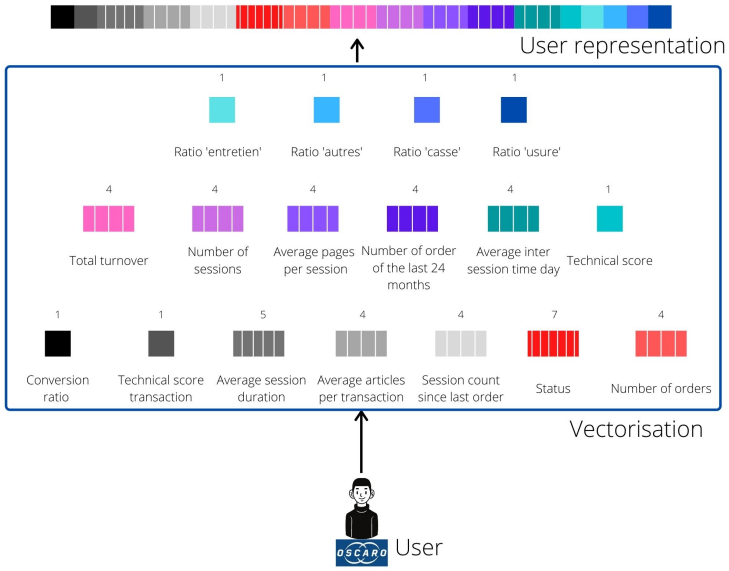


Figure 13: User vectorisation

DDPG This is the algorithm where I spent most of my time. I used the paper [1].

The aim was there to replace a simple state (genart) by information about the client, the current basket and interactions between genarts in the basket :

1. State representation : I created a vector with the user vectorisation, the genarts in the basket (each genart is also vectorised) and products of each genart in the basket within the other genarts in the basket.

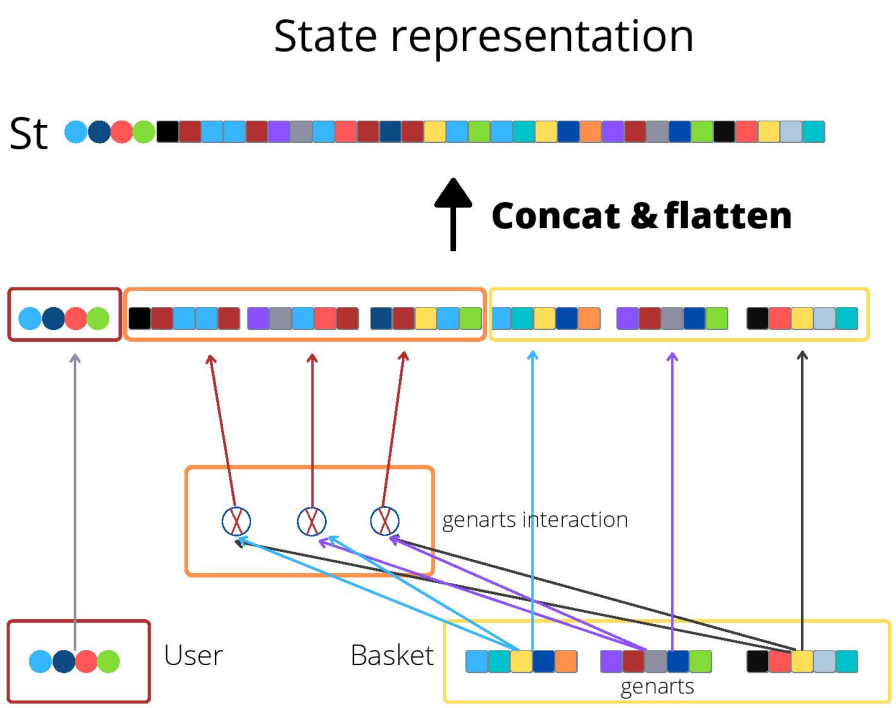


Figure 14: State representation for the DDPG algorithm

2. Actor network π_θ: The aim of this network is, given a state representation, to return the action what will maximise the cumulative rewards. To do so, it uses two relu networks and one tanh to then output a continuous representation of the action. In order to do the translation between the output of the tanh and the genart space, I computed the scalar product with each genart and the output of the tanh, and then the genart to recommend is the one that maximises the scalar product.

3. Critic network Q_w: The aim of this network is to predict the expected rewards for a given pair (state, action). It takes as input the relu output of the state representation concatenated with the output of the Actor network (continuous representation of an action). Then it passes through 2 relu networks and finally gives an estimation of the $Q(S_t, A_t)$ value. The critic network loss is the following :

$$\frac{1}{N}\sum_i (y_i - Q_w(S_i, A_i))^2$$

with

$$y_i = R_i + \gamma Q_{w'}(S_{i+1}, \pi_{\theta'}(S_{i+1}))$$

Note that the prime for the parameters means that this is the target network.

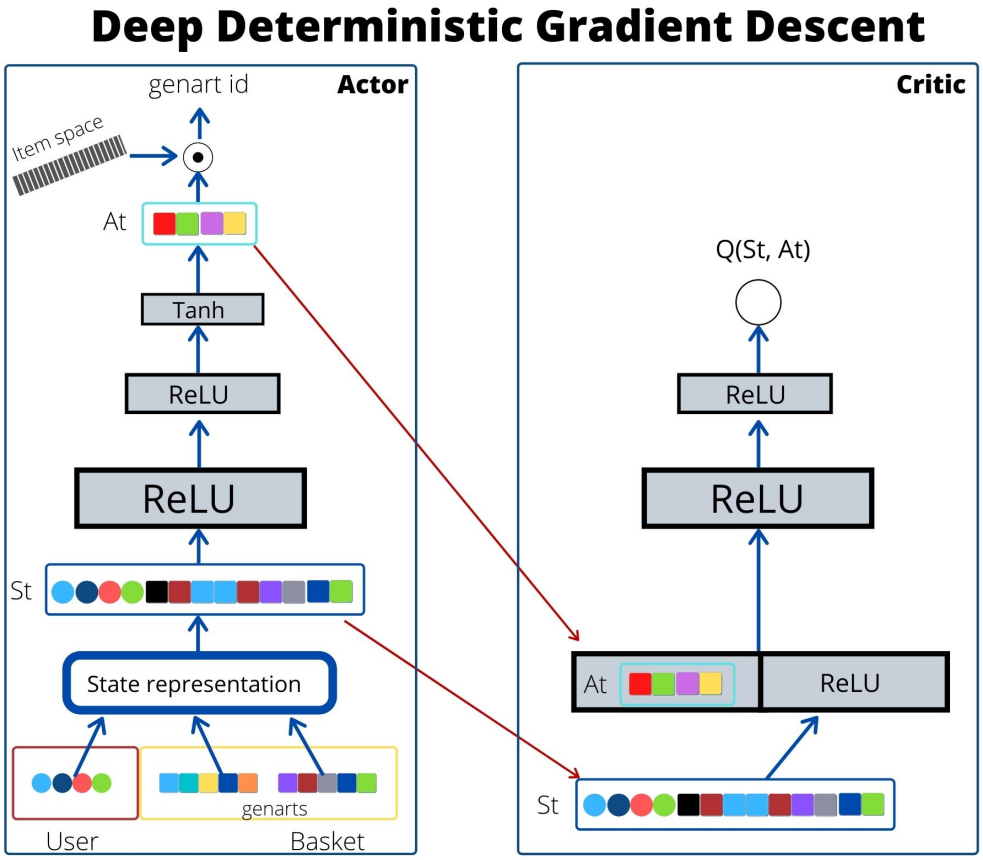


Figure 15: DDPG adaptation for oscaro

4.3 Metrics

As the learning process is done offline, it seemed hard to see the learning progress of the algorithms (ususally it can be visualised by looking at the rewards per epoch). Therefore, I only looked to the loss of the neural networks (for the Double DQN and the DDPG algorithms), and didn't focus on the Q-learning algorithm as it won't converge.

Score In order to know if the algorithm performs well on the validation set, one metric would consist in incrementing a counter whenever the recommendation of the algorithm is the same that was done in the past data (and the client has indeed converted this recommendation). This metric is the score I aimed to use, but can be used only when the user clicked in the recommendation. Furthermore, this metric is limited because it only compares to the previous

recommendation done in the website. It means we consider the recommendation system of the website as a reliable baseline. The real metric would be to run the algorithm online and look to the total amount of the basket (because we set the reward of the algorithm to be the total amount of the basket).

4.4 Results

4.4.1 Data

After spliting our data into tuples, I created a validation set with 50 000 tuples where the client has clicked in an article available (in order to use our custom score metric). The training set is composed of around 2 millions of tuples. I deleted all the outliers like the oscaro users, crawlers, robots, etc. Furthermore, as the size of the training set is quite huge, I decided to split the data loaded in python into batches of 50 000.

4.4.2 Learning process

Time The Q-learning algorithm was able to be trained in the 2 millions tuples in a reasonable time. Nevertheless, for the Double DQN and the DDPG, it required around 4 hours to be trained (with the GPU of google colab). I also crashed the GPU of google colab and had some pains with the use of google colab after the crashes. Futhermore, one bottleneck of my implementation is the batch used for loading the data from BigQuery. Indeed, the loading process in python to get the different batches of the training set was what required a lot of time. Finally, the update for the Double DQN and the DDPG algorithms was done every C new tuples (with mini-batches in the Memory replay) (C was set to be the size of the batch for the split of the training set divided by 100).

4.4.3 Loss

Double DQN In order to have an idea about the learning progress of the algorithm, one can think about the loss of the neural network used to approximate the Q-values. This is what is plotted in Figure 16.

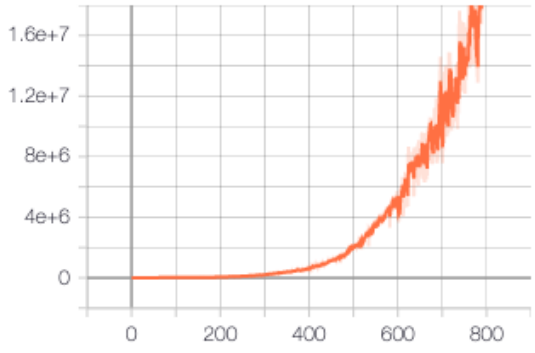
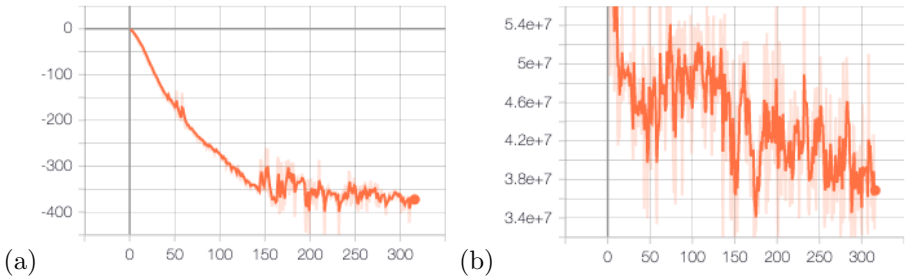


Figure 16: Double DQN loss of the neural network used for the prediction

We can clearly infer from the loss figure that the algorithm learned nothing. The hyperparameters weren't optimised enough. Therefore, we can't see if the

process isn't converging because of the Double DQN limits : the action space size, and also the fact that is doesn't take into account the basket and client information in the learning process (contrary to the DDPG implementation).

DDPG To have an idea of the learning process, one can focus on the losses of the actor and critic networks.



(a) (b)

Figure 17: Loss during the training process for the (a) Actor network and the (b) Critic network

On one hand, we see the actor loss that is decreasing until reaching a stable value, but the values move away from 0. Furthermore, the loss should be increasing for the actor network (the update is a gradient ascent and not a gradient descent). On the other hand, the loss for the critic network tends to decrease, which is the behaviour we except. Nonetheless, the values of loss are too high to be sure of the conclusion we should draw from it. Such as before, the hyperparameters have not been optimised.

4.4.4 Scores

The following table summarized the scores obtained by the algorithms in the validation set (with the time for training) :

Algorithms	Q-learning	Double DQN	DDPG
Score	3899/50000	57/50000	?
Time	$<$ 87min	87 min	250 min

Note that I didn't store the exact time of convergence of the Q-learning algorithm (as it is not suppose to be the best algorithm).
Furthermore, I don't have the score for the DDPG algorithm because the time required to compute it was too high (as I finished the implementation only a week before the end of the internship).

4.5 Limits

As my internship was a short one (2 months), my results are not finished. There are a lot of things I could have done better or differently. Here is a list of few limits of my work :

1. Memory replay : I added an element to the memory replay in a for loop, which is not efficient. I could have made the memory replay equal to the size of the training set, and then update the neural networks by just sampling in this memory replay.

2. Hyperparameters : I didn't have time to play with the hyperparamters for the double DQN and DDPG algorithms.

3. Genart representation : I did too much pre-processing in the genart representation. Indeed, at the end there are a lot of 0 and 1 in the representation (categorisation), which could lead to an inefficient product when we multiply multiple genarts between each other (in the basket for instance).

5 Conclusion

I tried through this project to go beyond the classic algorithms used in reinforcement learning. Indeed, these algorithms take the classic notations of the Markov Decision Processes such as tuples of states, actions, rewards and converge with classic environments. Nonetheless, the environment that we have is more complex as it corresponds to human-beings. The aim was then to take into account information about the client (like its current basket) and the articles that are sold.

The idea of using three algorithms was learn about classic reinforcement learning algorithms and its limits : starting from a tabular method (Q-learning) which can't be applied to continuous state space. Then, using neural networks to approximate the Q-table (DQN), but with a discrete action space. Finally, dividing the decision making with the Actor-Critic method to be able to work with continuous action space (DDPG).

Unfortunately, the results are not relevant for the different algorithms. It can be explained by optimisation issues : there are a lot of hyperparameters for these algorithms that are sensible. As I didn't have time to play with it, the behaviour of the losses of the neural networks infer that it doesn't learn well.

6 Acknowledgment

I would like to thank my manager Michaël Laroche who helped me during all my internship, and Maguerite Blanc for all the explanations of her past work on the cross-sell. Furthermore, I would like to thank all the data science team with whom I had great time this summer.

References

[1] Deep Reinforcement Learning based Recommendation with Explicit User-Item Interactions Modeling,
`https://arxiv.org/pdf/1810.12027.pdf`

[2] Richard S. Sutton and Andrew G. Barto, 2018. *Reinforcement Learning: An Introduction, second edition*

[3] Pytorch tutorials on reinforcement learning
https://pytorch.org/tutorials/intermediate/reinforcement_q_learning.html

[4] A Hands-On Introduction to Deep Q-Learning using OpenAI Gym in Python
https://www.analyticsvidhya.com/blog/2019/04/introduction-deep-q-learning-python/