Reinforcement Learning for Formula 1 Race Strategy

Understand how Deep Q-Networks learn to decide pit stops like AlphaGo learns to play Go.

Ashref Maiza Jun 24 · 15 min read ★

This article presents my experiment prototyping a deep reinforcement learning agent to help Formula 1 teams optimize their strategy decisions in real-time during the race. A neural network is designed as a mapping function from observed lap data to control actions and instantaneous reward signals. The conceptual framework introduced here is analyzed in the context of Monaco GP, but the same approach can be improved and generalized to all Grand Prix events.

Why does it matter?

Formula 1 is one kind of war without bullets and the main weapon in this war is innovation. Deciding a race strategy is the dark art in this sport. Fans usually enjoy the high speed maneuvers performed by brilliant drivers they get to see on TV. What they do not see is how much preparation is required beforehand to achieve better results, involving the use of mathematical models and probability theory. By racing strategy we mainly refer to these two crucial decisions:

- When the driver should go to pit stop for tyre change?
- Which tyre compound the crew should fit on the car?

Photo by gustavo Campos on Unsplash

Tyres degrade during the race which may lead to slower lap times. This process is inevitable and can be quicker or smoother depending on the compound (soft, medium, hard, etc.) and the track that holds the race.
If the lap time is slow to a level that the car loses more time over a number of laps than a pit stop would cost, then probably the tyres need to be changed.

Nowadays, Formula 1 teams use Monte Carlo simulations which they run for many hours before the race in order to decide a strategy for each of their two drivers. The way these simulations work is by sampling the race events lap by lap thousands of times in different configurations (pit stop lap, tyre compounds, positions, etc.) for the driver in question but also for all other opponents. They can only run a limited number of simulations due to the large space of possible scenarios and they pick the strategies that, among those simulations, revealed to lead to the best average outcomes.

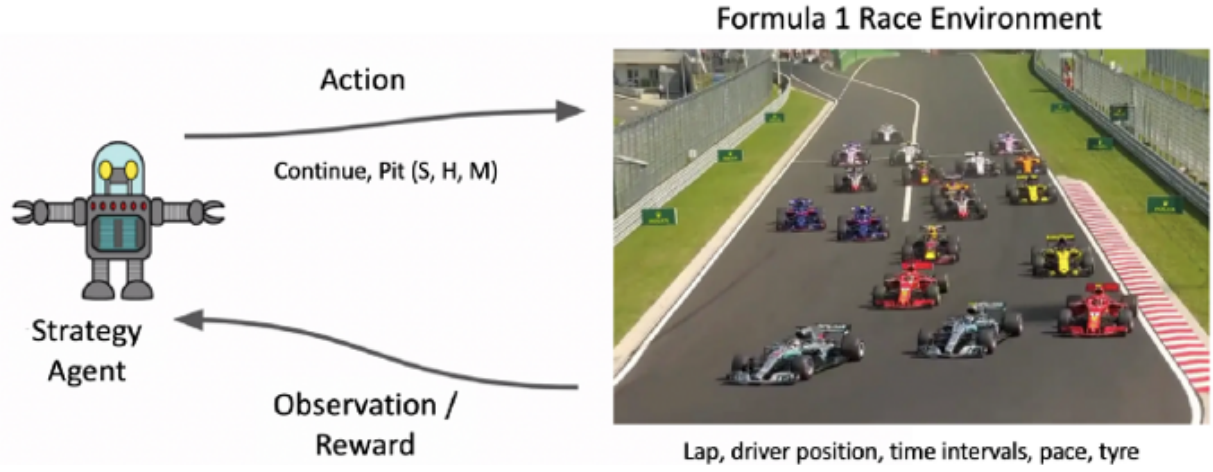
FIA 2019 & 2020 dry tyre compounds (Soft, Medium, Hard)

During the race event, many new scenarios could happen and race strategists sometimes find themselves obliged to adapt their decisions based on human judgements. Sometimes these decisions work well and lead to scoring more championship points. In other cases, it leads to frustration - losing important positions to other competitors who performed better decisions.

This situation motivated me to think of how we could teach an AI system to play the game of Race Strategy. The goal is to help a Formula 1 driver cross the finish line winning places (or not losing any) against other competitive team strategies.

RL Problem Formulation

Reinforcement Learning (RL) is an advanced machine learning (ML) technique which takes a very different approach to training models than other machine learning methods. Its super power is that it learns very complex behaviors without requiring any labeled training data, and can make short term decisions while optimizing for a longer term goal.



RL in the context of Formula 1 racing

In RL, an agent learns the optimal behavior to perform a certain task by interacting directly with the environment and maximizing the total reward it gets. In Formula 1 racing, the total reward (return) could be thought of as the number of places gained or lost by a specific driver at the end of the race. So, the agent can decide to pit at a particular lap and lose some positions if it thinks that the driver would achieve a better position by the end of the race.

For simplification, we will consider the set of possible actions at each lap as follows:

- Action 0: Pit stop for the Soft compound
- Action 1: Pit stop for the Medium compound
- Action 2: Pit stop for the Hard compound
- Action 3: Continue (No pit stop)

Prediction vs Control

The first interesting idea to introduce by applying RL for Formula 1 race strategy is the concept of “Control”.

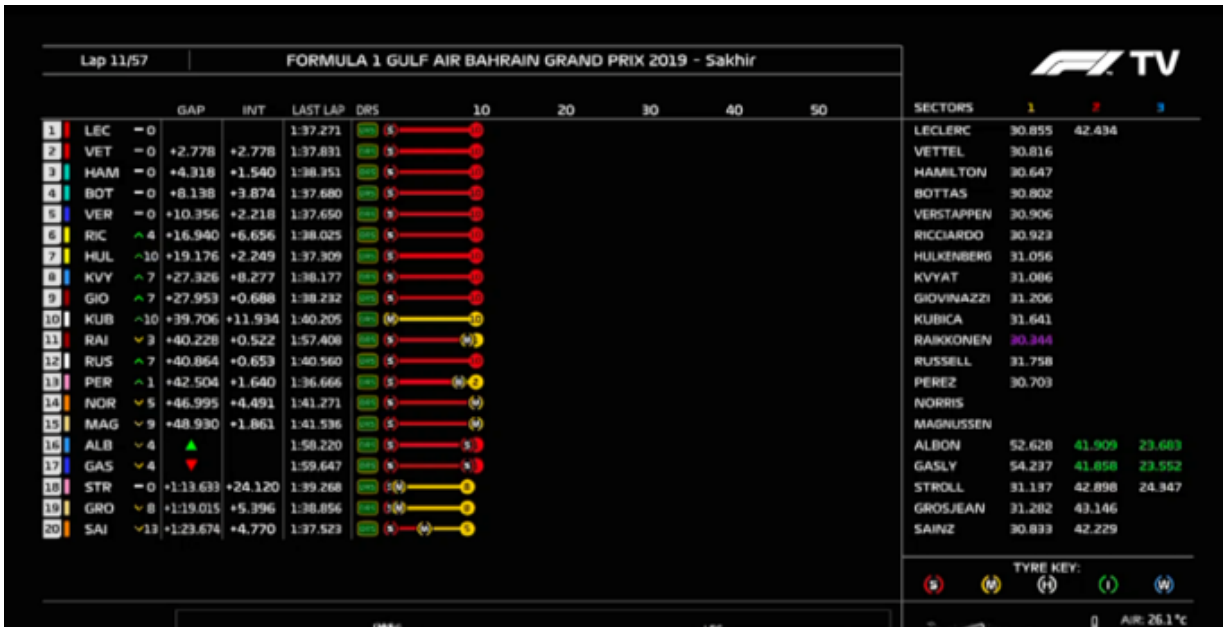
A prediction task in Reinforcement Learning is where a policy is being given, and the goal is to measure how well it performs at any given state. This is somehow similar to what the

simulations run by F1 teams try to achieve. Before the race, they want to predict how good a certain strategy would be if it was applied starting from the first lap.

But, things get really interesting in RL when we start performing control! A control task is when the policy is not fixed, and the goal is to find the optimal behavior. That is to find the optimal policy that given any state, always provides the best decision that maximizes the expected total reward. In Formula 1, this would be very interesting as it offers a way to learn how to exploit very complex patterns and come up with strategies that adapt in real time to each lap and that hopefully beats the best strategy experts from the competition.

MDP & Model-Free Control

Let's consider that one state in our Formula 1 racing environment corresponds to the information we get on the following TV screen.



F1 TV Data Channel

A state is represented mainly by the lap number, drivers ranking, time interval between drivers and their pace (last lap time) as well as their tyre compounds (Soft, Medium or Hard) and age (in number of laps). A safety car flag could also be considered. Besides, there is a rule that imposes at least two different dry compounds during the race for each car. So, we can imagine a flag for each driver indicating wether or not a second dry compound was used.

A Formula 1 race can then be formulated as a Markov Decision Process (MDP) where the probability of transitioning from a state to another relies only on the last observed state. At lap 11, all we need to decide for the next lap is the situation we observe at lap 11 (lap 1 to 10 become much less important).

> "The future is independent of the past given the present."

Planning a strategy for the next laps in the race may require a perfect model of the environment where we know exactly what would be the reward from taking an action a when in state s and the probability of transitioning from state s to state s' under

action a. But in reality, that perfect model of the environment is hard to obtain and many events happening during the race may be stochastic. Think of how the reward depends on the behavior of other cars, the continuous change in lap times and driver positions. So, we could not use a methodology like dynamic programming. By definition, a state value V(s) is the cumulative reward estimated for being in state s and following the policy moving forward. Dynamic programming is an iterative approach that relies on full-width backups of these state values. It is too expensive when the number of possible states is too large like in the case of Formula 1 racing.

For these reasons, we will rely on model-free reinforcement learning where the main idea is to sample particular trajectories from a state to evaluate actions in a trial-and-error setup by interacting with the environment.

Designing a Formula 1 race emulator

State of the art reinforcement learning has been usually demonstrated on classic games like Atari, Chess or Go. These are perfectly observable environments and we like studying them because they can be formulated using simple rules. They also allow to easily benchmark AI performance against human-level performance.

Formula 1 racing, though, is a real-world issue. It is an imperfect information game because partially observable. The

main challenge consists in building an emulator that would respect the logic and complex rules of the game.

We previously discussed the set of possible actions (pit stop and tyre compound) and the environment state (information we see on TV). This information could be provided to all teams by the F1 broadcast center and platforms like SBG Sports Software. We will use a Pandas dataframe to represent every state in the environment. Further information is included such as the potential pace of the car [potential_pace] and a flag indicating if a second dry compound has already been used [second_dry].

tla	lap	position	potential_pace	pace	compound	age	interval	gap	second_dry
HAM	3	1	77.185	77.185	0.0	4.0	nan	nan	0
BOT	3	2	77.271	77.271	0.0	4.0	0.781	0.781	0
VER	3	3	77.66	77.66	0.0	4.0	0.128	0.909	0
VET	3	4	77.966	77.966	0.0	4.0	1.253	2.162	0
MAG	3	5	78.128	78.128	0.0	4.0	1.605	3.766	0
RIC	3	6	78.237	78.237	0.0	4.0	0.1	3.867	0
KVY	3	7	78.29	78.29	0.0	4.0	1.256	5.122	0
GAS	3	8	78.06	78.29	0.0	4.0	0.214	5.337	0
SAI	3	9	78.436	78.436	0.0	4.0	0.63	5.967	0
ALB	3	10	78.672	78.672	0.0	4.0	1.115	7.082	0
HUL	3	11	79.249	79.249	1.0	4.0	0.47	7.552	0
NOR	3	12	79.303	79.303	1.0	4.0	1.844	9.396	0
GRO	3	13	79.046	79.303	0.0	4.0	0.522	9.918	0
RAI	3	14	79.134	79.303	0.0	4.0	1.063	10.981	0
LEC	3	15	78.526	78.526	1.0	4.0	3.285	14.266	0
PER	3	16	79.802	79.802	1.0	4.0	0.914	15.18	0
STR	3	17	80.425	80.425	1.0	4.0	0.406	15.586	0
GIO	3	18	79.204	79.204	0.0	4.0	1.52	17.106	0
RUS	3	19	81.056	81.056	1.0	4.0	1.581	18.687	0
KUB	3	20	81.33	81.33	1.0	4.0	1.163	19.85	0

Example of state in the Formula 1 race environment

The potential pace is the estimation of car pace in free air when not blocked by other cars. It is computed using a specific function taking into account fuel mass, tyre compound and their age.

Each lap is a new step in the environment which introduces changes in the observable measures (pace, tyre age, interval, etc.) and can lead to new drivers ranking. The agent decides a strategy to only one car at a time. After each step in the environment, a reward is calculated as the number of places gained or lost by that particular car.

Some of the important tools necessary to compute a step is the overtake model. This model provides a probability of overtake for each driver. It takes into account the interval to driver in front, potential pace of driver in front, potential pace of driver in question and a parameter representing the difficulty of overtake which is specific to the race track. Another important tool is the time spent in pit stop, this can be learned from past races or estimated during the weekend and has major impact on where a car will end up after pit stop.

Open AI Gym is an open source framework that provides significant help in structuring and implementing custom environments. Building a reinforcement learning environment that emulates well the dynamics of a Formula 1 Race is very important and challenging. This requires a deep understanding of the sport and many efforts in coding and testing the implementation. In order to develop and evaluate the approach, we decided to parametrize the emulator as for Monaco GP, a track where overtaking is known to be difficult. We used the qualifying results of Monaco 2019 to initialize the starting grid and train the system for that particular race.

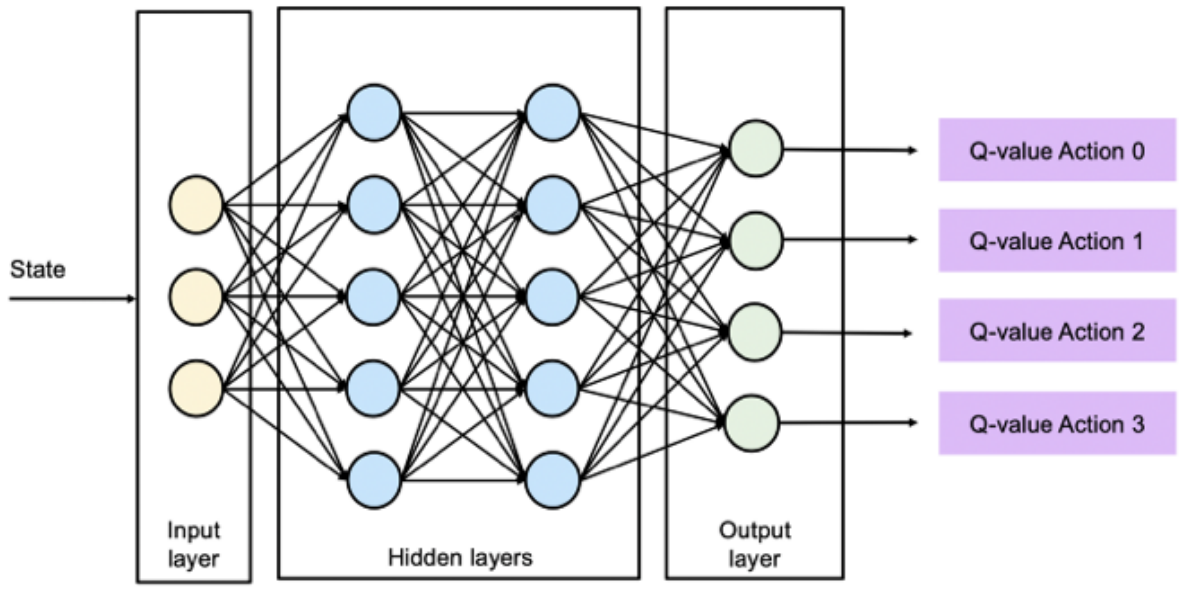
Designing the agent (Deep Q-Network)

After implementing the environment, we need to design the agent responsible of recommending a pit stop decision at each lap. The agent, when associated to a specific car has one ultimate goal which is maximizing the total reward that could be obtained by that car. Remember that total reward is defined as the number of places gained or lost across an entire race episode.

Q-learning is one of the techniques used in reinforcement learning to find the optimal policy according to which the agent should adapt its behavior. For each state, it is possible to estimate the total reward that would be obtained by taking a specific action and continuously following the policy. This total reward obtained from a (state, action) pair is called the Q-value. If we can estimate the Q-value for each (state, action) pair, the agent will behave optimally at each state by deciding the action that has the largest estimated Q-value maximizing the total reward.

Because the space of possible states in a Formula1 race is infinite, we cannot store all states in memory and compute a Q-value for each (state, action) combination. We need a neural network to approximate the Q-value function. Usually, it is called a Deep Q-Network (DQN) and the idea was first used by DeepMind to build an artificially intelligent system capable of playing Atari Games better than the best human experts. Without this approach, it becomes very hard to maintain

computation and memory efficiency especially in cases like Formula 1 racing which comes with continuous and high cardinality feature spaces.



Deep Q-Network

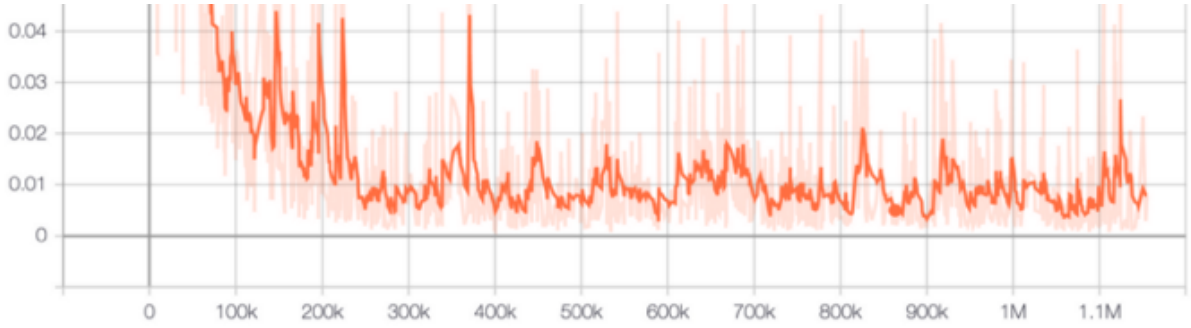
We implement a dense neural network in TensorFlow. It takes as input a vectorized representation of the environment state and outputs the estimated Q-value for each action. The next action is then determined by the maximum output from this network.

There are two main ideas that make this approach stable compared to naive Q-learning: experience replay and fixed Q-targets .

Experience replay: We store the agents experience (state, action, reward, next_state) in a replay memory so that we can

randomly sample batches of transitions and use them as training data. The major advantage of applying this technique is that it stabilizes the Q-learning method as it decorrelates the trajectories and uses efficient updates of the neural network. Usually in the literature, it is recommended to use a replay memory of size ~1M transitions. Nevertheless, the best results on this use case were obtained by using a reduced size for the replay memory ~15000 transitions and a batch size of 32 transitions. This forces the network to learn from each transition frequently and faster for a short time period until it definitely disappears from the replay memory and gets replaced by new unseen transitions that are sampled by following an improved policy.

Fixed Q-targets: The target values for training this model are obtained by using a target network which is a past version of the main behavior network. We keep this target network frozen for a certain number of steps in the mini-batch learning process and teach the behavior network to estimate Q-values towards those frozen targets. By choosing the right number of steps after which we update the target network ~760 steps, we could achieve a stabilized training.



Q-estimator MSE loss

The objective of the optimization is to minimize the MSE loss between the target values generated using the target network and the estimated values generated using the behavior network.

$$\mathcal{L}_i(w_i) = \mathbb{E}_{s,a,r,s' \sim \mathcal{D}_i}\left[\left(r + \gamma \max_{a'} Q(s', a'; w_i^-) - Q(s, a; w_i)\right)^2\right]$$

Target Estimation

We may think that the network would be learning its own predictions, but because the reward r is the unbiased true reward, the network will update its weights using backpropagation to finally converge to the optimal Q-values.

In the following example, we can observe the behavior of the Q-network applied to Pierre Gasly's car (GAS). It estimates the total reward from each combination of a given state (at lap 3) and all possible actions (action 0: pit for soft, action 1: pit for medium, action 2: pit for hard, action 3: continue). The agent

estimates a total reward of +0.69 places gained by the end of the race if action 3 (continue) is applied. Because this Q-value is the highest outcome, it will then recommend action 3 as a decision for that particular state. If Gasly was to pit for the soft compound at that lap, this would put him at the risk of losing 3 positions by the end of the race according to the list of Q-values estimated below.

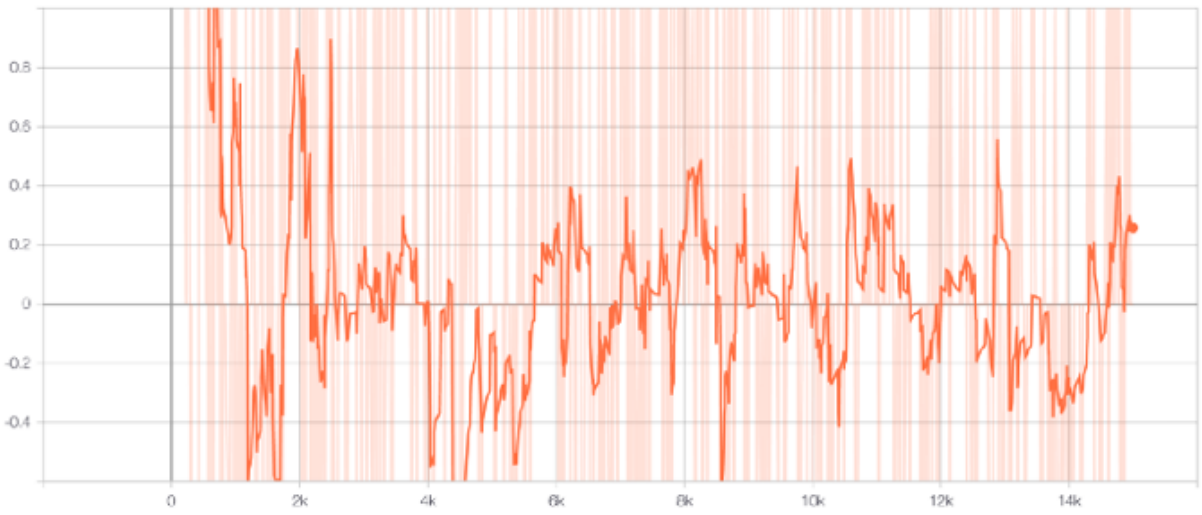
tla	lap	position	potential_pace	pace	compound	age	interval	gap	second_dry
HAM	3	1	77.185	77.185	0.0	4.0	nan	nan	0
BOT	3	2	77.271	77.271	0.0	4.0	1.031	1.031	0
VER	3	3	77.66	77.66	0.0	4.0	1.02	2.051	0
VET	3	4	77.966	77.966	0.0	4.0	2.041	4.092	0
MAG	3	5	78.128	78.128	0.0	4.0	0.266	4.358	0
RIC	3	6	78.237	78.237	0.0	4.0	0.728	5.086	0
KVY	3	7	78.29	78.29	0.0	4.0	2.562	7.649	0
GAS	3	8	78.06	78.06	0.0	4.0	2.816	10.464	0
SAI	3	9	78.436	78.436	0.0	4.0	0.317	10.781	0
ALB	3	10	78.672	78.672	0.0	4.0	0.035	10.816	0
HUL	3	11	79.249	79.249	1.0	4.0	0.158	10.974	0
NOR	3	12	79.303	79.303	1.0	4.0	0.739	11.713	0
GRO	3	13	79.046	79.303	0.0	4.0	0.073	11.786	0
RAI	3	14	79.134	79.134	0.0	4.0	1.623	13.409	0
LEC	3	15	78.526	79.134	1.0	4.0	0.002	13.411	0
PER	3	16	79.802	79.802	1.0	4.0	0.357	13.769	0
STR	3	17	80.425	80.425	1.0	4.0	0.12	13.889	0
GIO	3	18	79.204	80.425	0.0	4.0	0.018	13.907	0
RUS	3	19	81.056	81.056	1.0	4.0	0.053	13.96	0
KUB	3	20	81.33	81.33	1.0	4.0	0.13	14.089	0

```
Q-values: [-3.0144467  -2.2404468  -2.3527777   0.69367933]
Action: 3, Reward:0
```

Training in a multi-agent setup

In 2017, DeepMind introduced AlphaGo Zero, a reinforcement learning algorithm that learned to master the game of Go without seeing any previous human play. The system starts off without knowing any technique and learns to improve its ability only by playing itself. The essence of this idea is that AI systems, in order to beat human-level performance, should not be constrained by the limits of human knowledge.

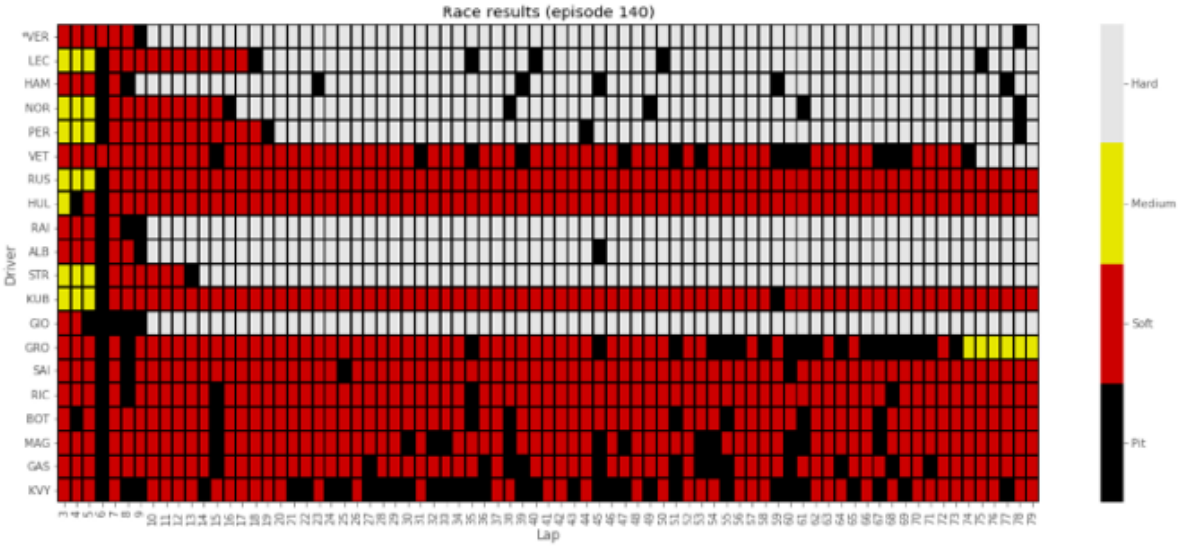
By analogy, we can think of a Formula 1 race as a multi-player game involving 20 cars where each car is trying to maximize its ranking by the end of the race and beat other opponent strategies. If all other players fix their policies, then the best response is the optimal policy against those policies. Thus, it is interesting to optimize the system towards the Nash Equilibrium which is a joint policy for all players such that every player's policy is a best response. The best response is then a solution to a single-agent RL problem where other players become part of the environment. We populate the replay memory by generating RL experience where the agent is playing an older version of itself. In addition to the behavior network and the target network, we decided to use a third network responsible of generating decisions for other cars when optimizing for a specific one. Just like the target network, this opponent network is a past version of the behavior network but updated 10 times less frequently than the target network (every ~7600 steps).



Reward for 15000 episodes (Nash equilibrium)

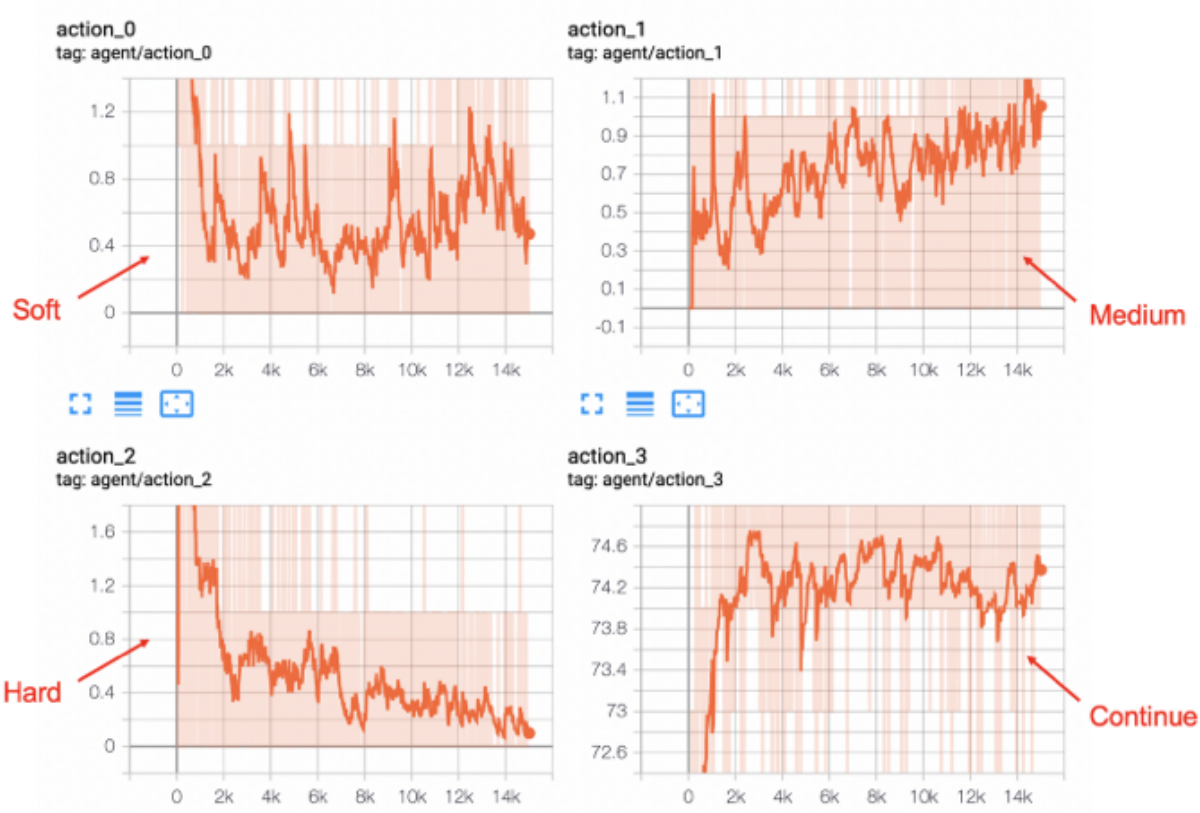
While training the system to play 15000 races against itself, we could notice the convergence towards the Nash Equilibrium as the total reward obtained at each episode oscillated around zero for different cars and race trajectories. During that process, the system was learning to design a race strategy and improve its ability only by self-play!

This plot shows the decisions made by the system for each car at each lap in the early stages of the learning process. We can see that it was pitting too often which is not common sense in Formula 1 as so much time was wasted.

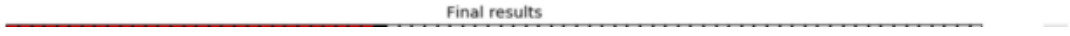


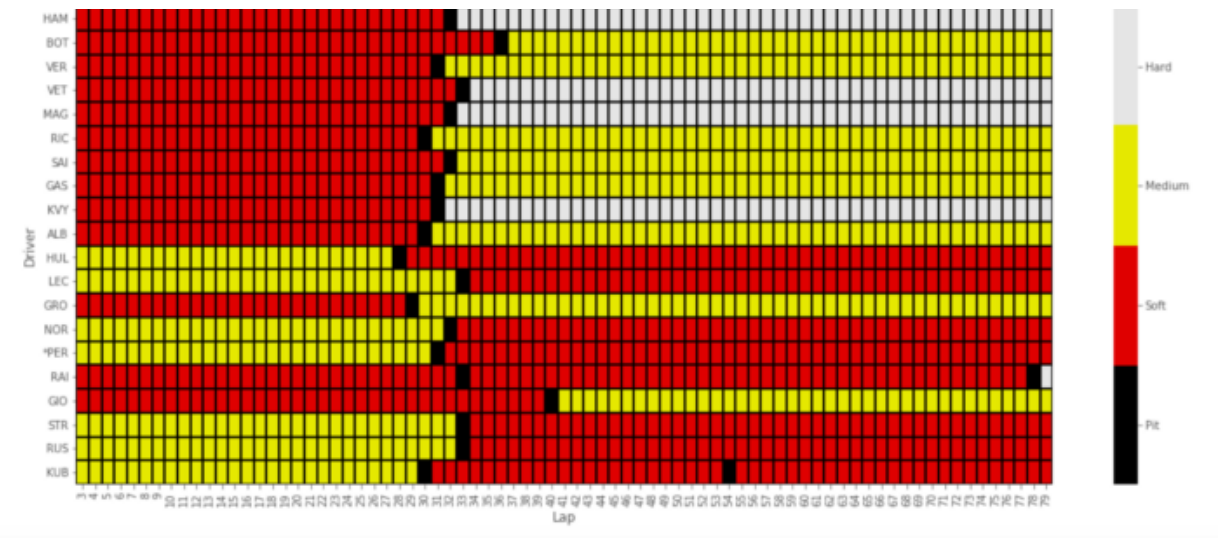
However, The following graphs show exactly how the system learned it should continue more often and optimize the use of tyres. For each episode, we count the number of laps where each action has been taken except for the first two laps (used for environment initialization). Notice how the system is learning to increase the frequency of the continue decision

(action 3) and make less use of the hard compound (action 2) as compared to other faster compounds (medium and soft).



A nice way to evaluate the final system outside the multi-agent training process is by using it for a specific car and make it compete against an older version of itself applied to other cars. We will use the agent trained for 15000 episodes as the control agent for Perez (PER) and use same agent trained only for 12000 episodes (80%) to decide opponent strategies for all other cars. Knowing that Perez started Monaco 2019 race in the 16th position, it is interesting to see how he could gain +1 position by the end of the race following the latest agent's behavior policy.





Race simulation using optimal agent for PER

In this case, we are evaluating the agents performance against an older version of itself that also appears to be a strong strategist. Notice how the opponent strategy system learned to decide a one-stop strategy for many drivers with no prior knowledge. It decided the pit stop lap and tyre choice depending on the specific situation of each car it controls and managed to maintain the driver's position as part of the Nash Equilibrium.

For Raikkonen (RAI), it decided two sprints on the fastest soft compound, then pit stop at the very end of the race for a hard compound. The reason of this last pit stop is to respect the rule of using two different dry compounds in the same race. It clearly demonstrates the ability of the agent to optimize its decisions while autonomously learning the rules of the game.

What would be more interesting is to compare the performance of the agent against predefined strategies decided by Formula 1

teams while interacting with a real race event.

Summary

In this work, we introduced a deep reinforcement learning approach that offers the advantage of optimizing the Formula 1 race strategy live during the race. This solution differs from classic simulation approaches in some notable ways:

- Monte Carlo simulations sampled before the race do not adapt to the reality of occurring events during the race and they are not guaranteed to find the optimal policy. The introduced approach converges to an optimal policy and can be applied at each particular lap in the race to decide the best possible action.
- With neural networks and function approximation, we considerably reduce computation and memory complexity required to estimate the value of an extremely large number of possible (state, action) pairs in the environment. Unlike traditional Monte Carlo methods that work only on small and finite MDPs, this methodology has the advantage of scalability.
- The agent consisting of a neural network is trained to cross many features representing the state of the environment with no prior assumptions. This may help learn very complex patterns about the race dynamics which could be difficult to identify by humans.

- With one single neural network, we can define an optimal strategy for any driver on the grid. This offers the possibility to control both drivers of the same team and evaluate all actions decided by the competition.

What is exciting about RL is that the system doesn't mainly rely on history data to improve decision making. It learns directly by interacting with the environment. Surprisingly, for many real world applications, interacting with an environment may be possible. We define a set of actions, the space of observations and focus the effort on engineering the most convenient reward function. This AI framework for Formula 1 race strategy can be enhanced by taking into account the probability of safety car and the weather conditions that may change the course of a race.