Racing with Reinforcement Learning

"*Outracing champion Gran Turismo drivers with deep reinforcement learning*"

Krishna Agaram
QuML@Aalto University

July 4, 2023

The authors

- The (long) list of authors, from all three HQs of Sony AI:

Peter R. Wurman[1]✉, Samuel Barrett[1], Kenta Kawamoto[2], James MacGlashan[1], Kaushik Subramanian[3], Thomas J. Walsh[1], Roberto Capobianco[3], Alisa Devlic[3], Franziska Eckert[3], Florian Fuchs[3], Leilani Gilpin[1], Piyush Khandelwal[1], Varun Kompella[1], HaoChih Lin[3], Patrick MacAlpine[1], Declan Oller[1], Takuma Seno[2], Craig Sherstan[1], Michael D. Thomure[1], Houmehr Aghabozorgi[1], Leon Barrett[1], Rory Douglas[1], Dion Whitehead[1], Peter Dürr[3], Peter Stone[1], Michael Spranger[2] & Hiroaki Kitano[2]

1. Sony AI, New York, NY, USA.
2. Sony AI, Tokyo, Japan.
3. Sony AI, Zürich, Switzerland.

- Here's a link to the paper.

The problem statement

- Crucial for upcoming AI: *real-time* decisions in physical environments interacting with humans.
- **Automobile racing** is an *extreme* example. Try to build agent for racing.
- Sim for training: PlayStation game *Gran Turismo*. Faithfully reproduces complex multi-agent interactions.

Previous work

- Common Approach: *pre-compute trajectories* ("the policy") and model-based *predictive control* to execute them.

- Problems:
  1. Small modelling inaccuracy can be catastrophic, because of the very small margin for error.
  2. Won't work if there are other racers around - you cannot guarantee to be on that trajectory, and importantly, you are missing out on player-to-player interactions.

- Workarounds: supervised learning to model vehicle dynamics + imitation learning, evolutionary (gradient-free) approaches or reinforcement learning to learn driving policies.

- Some success in solo driving. None have tackled racing at the highest levels.

This paper's contribution

- Racers must become highly skilled in four areas:
  1. control - how to drive
  2. tactics - how to overtake, defend
  3. etiquette - playing fair
  4. strategy - when to make a move

- Authors use model-free off-policy deep RL to build agent *GT Sophy*. Tackles the first three requirements but not yet strategic.

- Demonstrate by competing 4v4 against top human drivers on three car and track combinations.

Building up to the algorithm: Reinforcement Learning

- Goal: get agent to take actions that maximize

$$J = \sum_{t=0}^{T-1} \gamma^t \mathbb{E}_{(s_t,a_t)\sim\rho}\left[r_t \equiv R(s_t,a_t)\right] = \mathbb{E}_{(s_0,\ldots a_{T-1})\sim\pi,\rho}\left[G^\pi\right]$$

where the return $G^\pi \equiv G_0^\pi$ is defined by $G_t^\pi = \sum_{i=0}^{T-t-1} \gamma^i r_{t+i}$.

- If $\mathcal{S}, \mathcal{A}$ discrete, one way: Q-learning. Iterate the below:

$$q(s,a) \leftarrow q(s,a) + \alpha_t(\mathcal{Q} - q(s,a))$$

with target $\mathcal{Q} \equiv r + \gamma \max_{a'} q(s',a')$ while exploring the environment (the s, a, r, s' come from here) via an ϵ-greedy policy to choose actions. Provably converges to the optimal q-values and policy.

Building up to the algorithm: DRL, DQN

- Extension of Q-learning when $\mathcal{S}$ is cont. - DQN: The q-function update is now gradient descent on function approximator with loss:

$$\mathcal{L}(\theta) \equiv \mathbb{E}_{\rho}\left[(\mathcal{Q} - q_{\theta}(s,a))^2\right]$$

with the rest as before. Estimate the expectation over many *experiences* (s, a, r, s'), which are drawn from a replay buffer at each step.

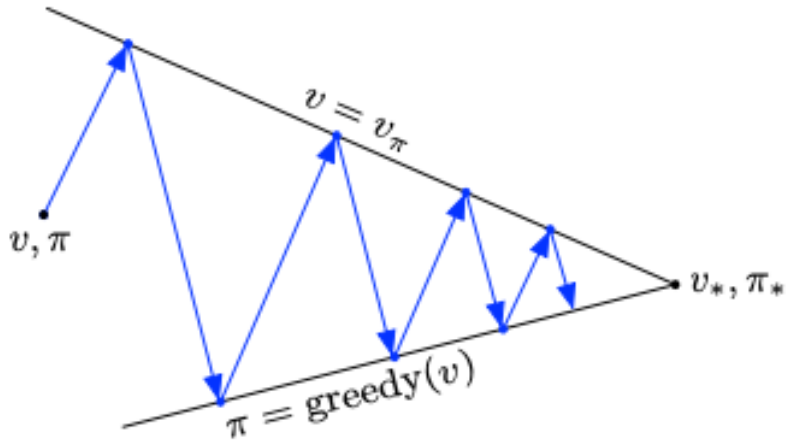
Building up to the algorithm: Policy-gradient

- What we need are policies. *Policy-gradient* methods - parameterize a (stochastic) policy and adjust it to maximize expected returns. So objective:

$$J(\phi) = \mathbb{E}\left[G^{\pi_\phi}\right] = \mathbb{E}_{s_0 \sim p_0}\left[v^{\pi_\phi}(s_0)\right] = \mathbb{E}_{s_0 \sim p_0, \hat{a} \sim \pi(\cdot|s_0)}\left[q^{\pi_\phi}(s_0, \hat{a})\right]$$

- Typically also maintain an estimate to either v^{π_ϕ} or q^{π_ϕ}.

- It is a type of policy-iteration procedure - repeatedly evaluate a policy (estimate its value) and then use the estimate to improve the policy (via the policy gradient).



Building up to the algorithm: MaxEnt RL

- The Max-Entropy framework for RL encourages maximum returns with as *diverse* a policy as possible.

$$J(\pi) = \sum_{t=0}^{T-1} \gamma^t \mathbb{E}_{(s_t,a_t)\sim\rho} \left[r_t + \alpha \mathcal{H}(\pi(\cdot|s_t)) \right] = \mathbb{E}_{\pi,p,r} \left[G^\pi \right]$$

- Define *Soft* q-function-

$$q(s,a) = r + \gamma \mathbb{E}_{s'\sim p(\cdot|s,a)} \left[v(s') \right]$$

where $v(s) \equiv \mathbb{E}_{a\sim\pi(\cdot|s)} \left[q(s,a) - \alpha \log \pi(a|s) \right]$, motivating the soft-$q$ loss function

$$\mathcal{L}(\theta) = \mathbb{E}_{s,a,r,s'\sim\mathcal{D},a'\sim\pi(\cdot|s')} \left[(q_\theta(s,a) - (r + \gamma[q(s',a') - \alpha \log \pi(a'|s')]))^2 \right]$$

$\mathcal{D}$ is the replay buffer.

Building up to the algorithm: SAC

- Policy-gradient method (*soft actor-critic*) for MaxEnt Reinforcement learning.

- Overestimating a value is bad - we may repeatedly choose suboptimal action and get stuck. Partial fix: two approximators are used for the q-function, min is taken.

- Policy objective similar to before -

$$J(\phi) \equiv \mathbb{E}_{s\sim p_0}\left[v^{\pi_\phi}(s)\right] = \mathbb{E}_{s\sim p_0,\hat{a}\sim\pi(\cdot|s_0)}\left[\min_n q(s,\hat{a};\theta^n) - \alpha\log\pi(\hat{a}|s)\right]$$

- Value loss: $n \in \{1,2\}$:

$$\mathcal{L}(\theta^n) = \mathbb{E}_{s,a,r,s'\sim\mathcal{D},a'\sim\pi(\cdot|s')}\left[(q(s,a;\theta^n) - \mathcal{SAC}^{\text{target}})^2\right]$$

$$\mathcal{SAC}^{\text{target}} \equiv (r + \gamma\min_n q(s',a';\theta^n) - \alpha\log\pi(a'|s'))$$

- α auto-tuned to move the entropy towards target entropy $\mathcal{H}$ - $J(\alpha) = \alpha\mathbb{E}_{s\sim\mathcal{D}}\left[\mathcal{H} - \mathcal{H}(\pi(\cdot|s))\right] = \alpha\mathbb{E}_{s\sim\mathcal{D},\hat{a}\sim\pi(\cdot|s)}\left[\mathcal{H} + \log\pi(\hat{a}|s)\right]$

Almost there! Quantile Regression & distributional RL

- Define $\rho_\tau(m) \equiv m(\tau - \mathbb{1}_{m<0})$. Then $\arg\min_u \mathbb{E}_{Y\sim p}\left[\rho_\tau(Y-u)\right]$ is (by calculus) the τ-th quantile $F_Y^{-1}(\tau)$ of p where F is the CDF.
- Thus, one can apply stochastic gradient descent to the loss $\mathcal{L}_\tau(\theta) \equiv \mathbb{E}_Y\left[\rho_\tau(Y - y_\theta)\right]$ to obtain parameter θ such that y_θ is the τ-th quantile of the distribution of Y - called *quantile regression*.
- *Distributional RL*: try to approximate the distribution of the returns (in particular, various quantiles) rather than just the expectation. More informative, shown to do better on some problems. An important use is in risk-sensitive RL scenarios.

The algorithm: QR-SAC

- Essentially D(istributional)-SAC. Fix a set of quantiles $\tau_0 = 0, \tau_1, \ldots, \tau_N = 1$. Maintain a network ($\theta$) taking as input the pair (s, a) and outputting $Z_{\tau_i}(s, a; \theta)$ for each i.
- Z_τ is an estimate for the τ-th quantile of the distribution $[G_t^\pi | s_t = s, a_t = a]$. The network is trained with the sum of the quantile regression losses at each τ_i - for each $k \in \{1, 2\}$,

$$\mathcal{L}(\theta^k) = \sum_{i=0}^{N-1} \left(\sum_{j=0}^{N-1} (\tau_{j+1} - \tau_j) \rho_{\tau_j}(y_j - Z_{\tau_i}(s, a; \theta^k)) \right)$$

where (let $k^* = \arg\min_k Q(s, \hat{a}; \theta^k)$)

$$y_i = \sum_{k=0}^{n-1} \gamma^k r_{t+k} + \gamma Z_{\tau_i}(s_{t+n}, \hat{a}; \bar{\theta}^{,k^*}) - \alpha \log \pi(\hat{a} | s_{t+n})$$

is the n-step target τ_i-th quantile.

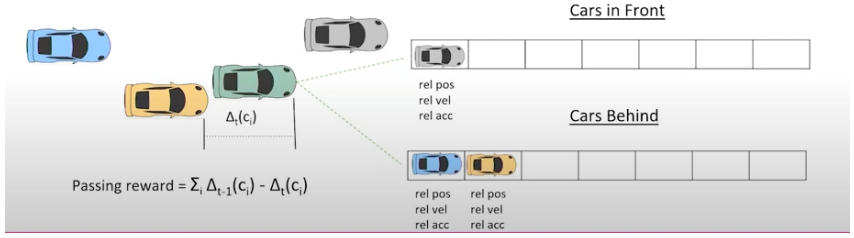
- Policy loss: $J(\phi) = \mathbb{E}_{s \sim \mathcal{D}, \hat{a} \sim \pi_\phi(\cdot|s)} \left[Q(s, \hat{a}; \theta^{k^*}) - \alpha \log \pi_\phi(\hat{a} | s) \right]$

Deploying the algorithm: Setup

- GT runs only on PlayStations. Agent ran on a separate computing device communicating asynchronously. Each GT Sophy instance controlled up to 20 cars on its PS. $10 - 20$ PSs were typically used with a GPU updating the (common) neural network parameters.

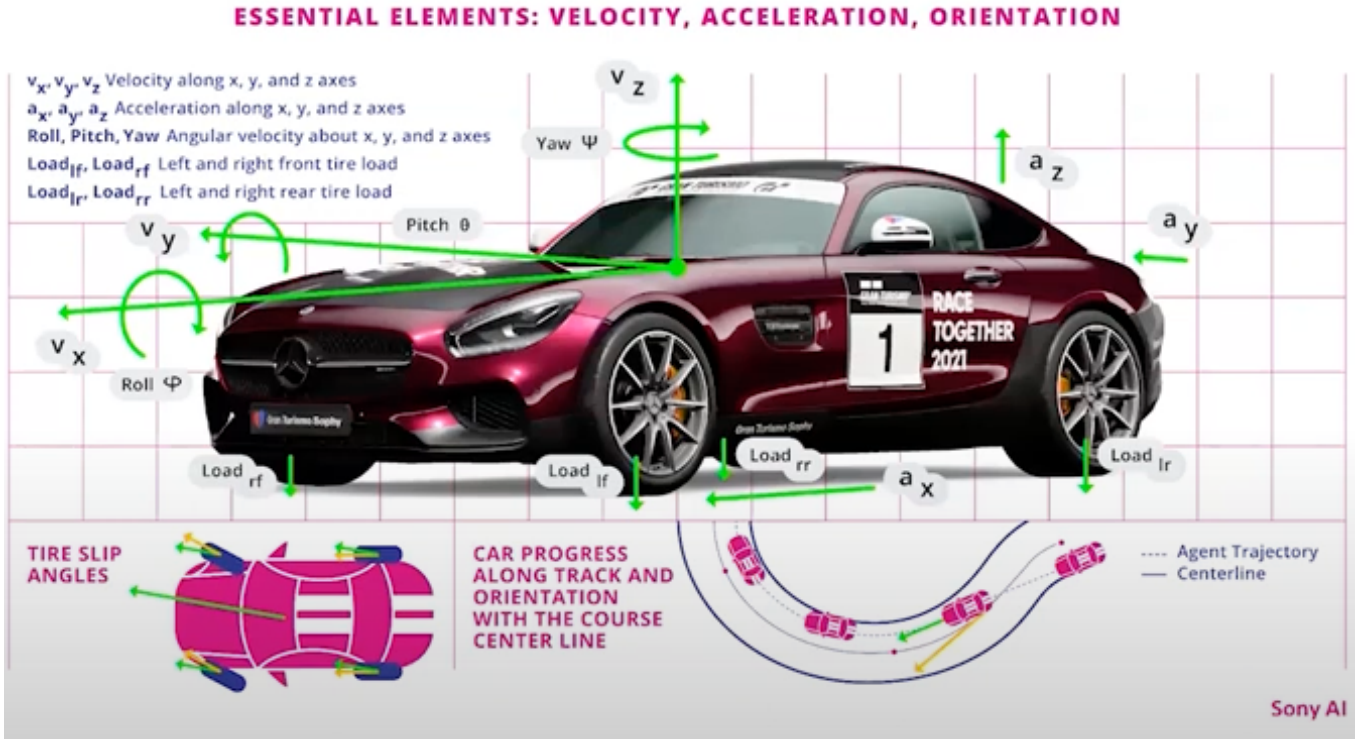
- The agent is given a static map defining the left and right edges and the centre line of the track. Approaching course segment encoded as 60 equally spaced 3D points along each line fed to the network.

- Through an API, agent observes positions, velocities, etc about all opponents, maintains two lists of state features: for cars in front of the agent and for cars behind - also passed to network.



Deploying the algorithm: State and action features

- A *lot* of state features.



- 2 action features: the rate of acceleration normalized to $[-1, 1]$, and the steering (L to R), also normalized. Automatic transmission and traction control.

Teaching the agent to behave: The reward scheme

- Possibly the most crucial part. All rewards serve their own important purposes.
  1. Course progress R_{cp}. $Rcp(s, s') \equiv s'_l - s_l$. $s_l =$ distance of point s along centreline from the start of the track.
  2. Offcourse penalty $R_{soc} \equiv -(s'_o - s_o)(v'_{kph})$, $s_o =$ cumul. offcourse time.
  3. Wall penalty $R_w \equiv -(s'_w - s_w)(v'_{kph})$, similar to R_{soc}.
  4. Tyreslip penalty R_{ts} - sum of some tire-related parameters over the 4 tires.
  5. Passing bonus R_{ps}. $R_{ps}(s, s') \equiv \sum_i (s_{L_i} - s'_{L_i}) \max(\mathbb{1}_{(b,f)} s_{L_i}, \mathbb{1}_{(b,f)} s'_{L_i})$; $s_{L_i} =$ signed distance along the centreline to opponent i.
  6. Collision penalty $R_c(s, s') \equiv -\max_i s_{c,i}$; $s_{c,i} = \mathbb{1}_{\text{opponent } i \text{ collides with agent}}$.
  7. Rear-end penalty $R_r(s, s') \equiv -\sum_i s'_{c,i} \mathbb{1}_{s'_{L_i} > 0} |v' - v'_i|^2$.
  8. Unsporting-collision penalty R_{uc}. Negative unit reward, fires when the agent makes "unsporting" collision.

- Total reward is a linear comb. of the above for each track:

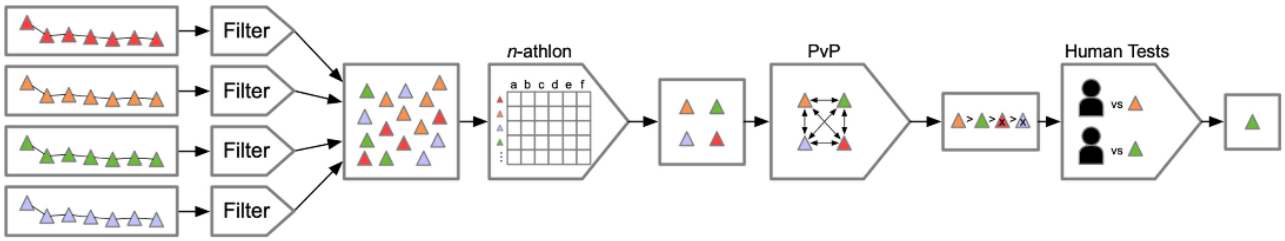
Course	R_{cp}	R_{soc}	R_{loc}	R_w	R_{ts}	R_{ps}	R_c	R_r	R_{uc}
Seaside	1	0.01	0	0.01	0.25	0.5	5	0.1	0
Maggiore	1	0.01	0	0.01	0.25	0.5	4	0.1	0
Sarthe	1	0	5	0.01	0	0.5	5	0.1	5

Training details

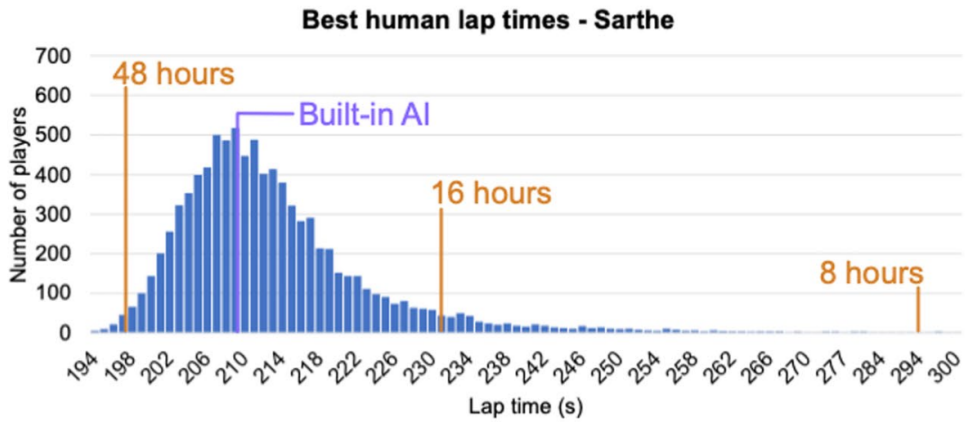
- Humans are apparently really good at racing. Exhorbitant amount of tuning and intervention required to beat absolute top racers.
- While training: randomized track positions, start speeds, spacing between cars and opponent policies. *Mistake learning* to avoid catastrophes.
- Policy selection: Policies saved at intervals. Each saved policy competed against other agents, filtering to a smaller set. These policies competed in an *n*-athlon.Performance reviewed by human committee to select a small set of policies that seemed the most competitive *and* best behaved. Round-robin on these to choose the final policies. (See next pg.)

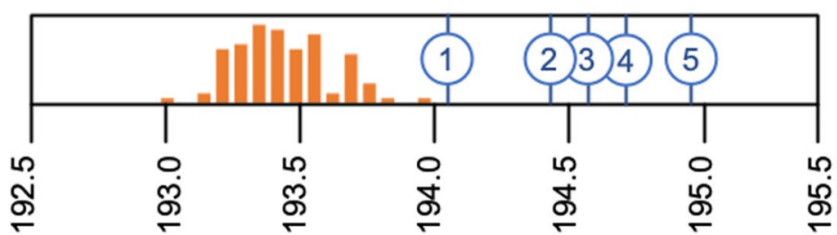
Training details

- Choosing final policy:



- Performance on Sarthe. Lower figure: histogram of 100 laps of the fully trained agent.





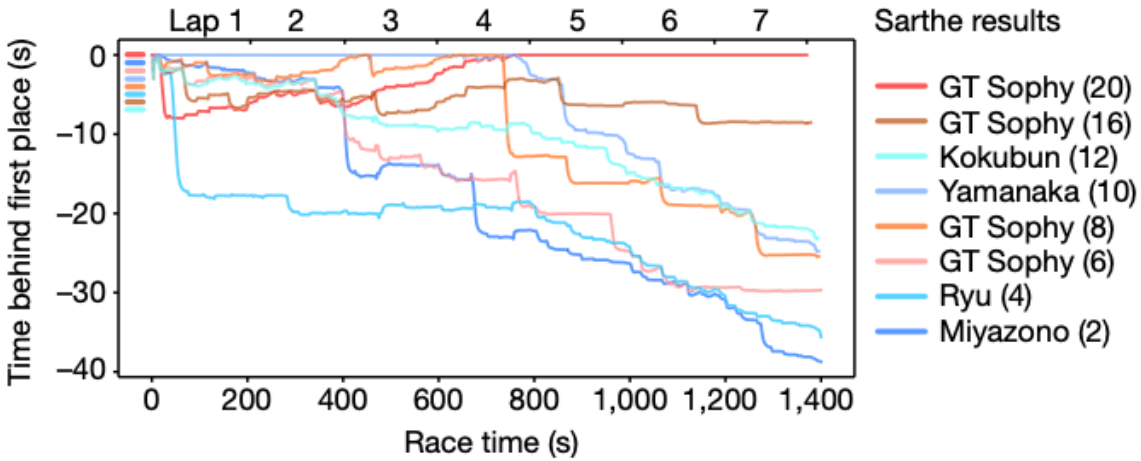
Racing with real people: July and Oct'21

- July: GT Sophy won time-trial on all three tracks, but lost 4v4 h2h 70-86.

Time-trial results

	Driver	Best time (s)	GT Sophy (s)
Seaside	Emily Jones	107.964	106.417
Maggiore	Valerio Gallo	114.466	114.249
Sarthe	Igor Fraga	194.888	193.080

- Oct (rematch): GT Sophy won h2h 104-52.



Fairness: Was it fair?

- Competitions between humans and AI systems cannot be made entirely fair; computers and humans think in different ways and with different hardware. Try to make it "fair enough".

- In racing, differences between humans and the agent:
  1. **Perception**. GT Sophy: precise map of the course. Humans: vision. Flipside: Humans: can see track outside boundaries. GT Sophy: need to drive and check.
  2. **Opponents**. GT Sophy: exact positions of all opponents. Humans: vision, but full 3D view.
  3. **Vehicle control**. GT Sophy: precise control. Humans: less precise information, but more control types - transmission etc.
  4. **Action frequency**. GT Sophy: 10 Hz. Humans: 60 Hz. Reaction time: GT Sophy: 20-30 ms, whereas humans have a much longer time of 100-200 ms. Similar delay tried in GT Sophy, still wins.

Conclusions: What does this hold for the future?

- Firstly, expert GT drivers are *really* good. Took a huge lot to beat them. Link to race videos.
- Agent able to perform several types of corner passing, it uses the slipstream, etc. It was also able to do these in many different scenarios - generalizability of the policy.
- Many areas for improvement: particularly in strategic decision-making - GT Sophy takes first opportunity to pass on a straightaway, sometimes opponent can re-overtake in time.
- Success of deep RL in this environment suggests that it may soon be used on real-world systems - collaborative robotics, aerial drones etc.