Generating balanced multiplayer game levels with GANs and RL

Grzegorz Opoka (grzego.opoka@gmail.com)

AGH

Problem

Given players with different abilities, can we generate game levels that do not favor any of them?

Idea

Use RL agents to evaluate *fairness* (i.e. how balanced the level is) of given game level and then train GANs to generate game levels that optimize for it.

What's *fairness*?

Assuming all players are equally skilled. We say that a level is fair if it is not biased toward any player. We estimate *fairness* with following equation:

$$\text{fairness}(b) = -\sum_{a \in A} \left| \underbrace{\left(\frac{1}{N} \sum \text{game}(b, a) \right)}_{\substack{\text{games won by} \\ \text{player } a \in A \text{ on average}}} - \underbrace{\frac{1}{|A|}}_{\substack{\text{fair win-rate} \\ \text{of game level}}} \right|$$

The closer the value of *fairness* is to 0 the fairer the level is.

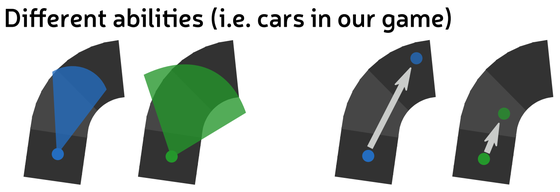
How to optimize *fairness*?

- Assuming RL agents are trained properly, they can be used as a replacement for human players. We can therefore use them to estimate *fairness*.
- We then simultaneously train Discriminator to predict the *fairness* estimated by Agents, and use it's knowledge about what makes levels fair to train Generator.

Agents

- Serve as replacement for slow evaluation by human players
- Used to estimate *fairness* of given game level
- Trained by PPO (Proximal Policy Optimization) algorithm to drive as best as possible

Different abilities (i.e. cars in our game)



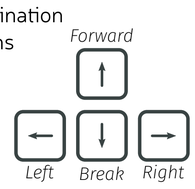
angle of rotation *speed*

Environment (Game)

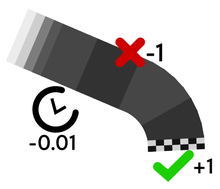
- Basic race game where each players is starting from the same position and has to finish the race as fast as possible.

Actions

- Any combination of 4 actions (+ noop)

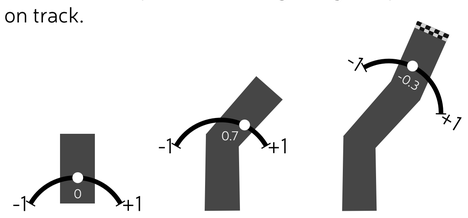


Rewards



Racetracks

- Are represented as vectors of numbers. Each number corresponds to an angle at given point on track.

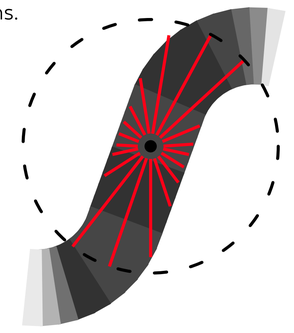


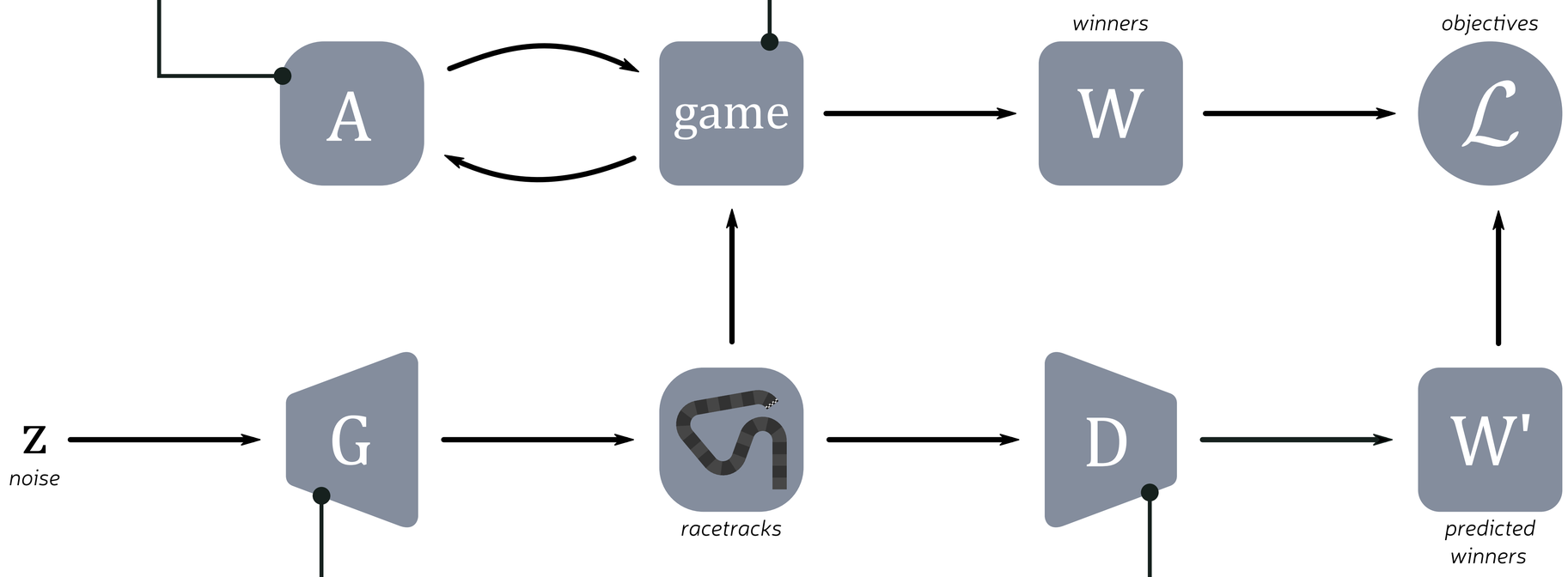
$$[0.0, 0.7, -0.3]$$

Vector representing above racetrack

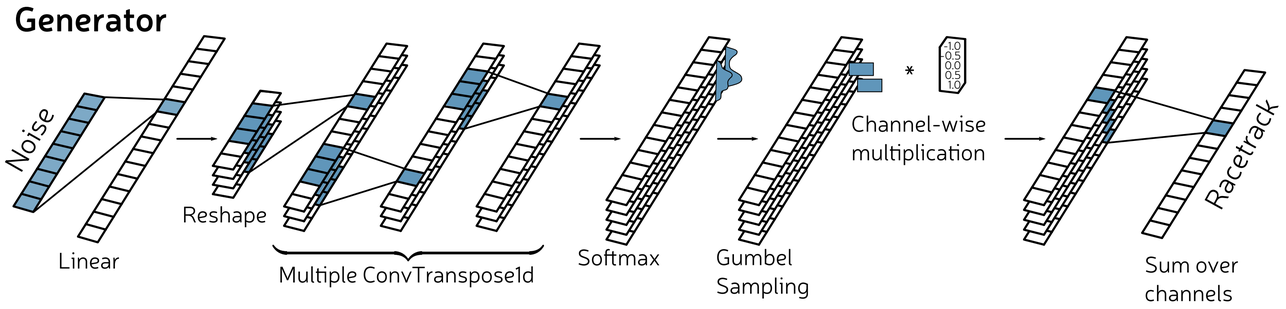
Observation (Input)

- Distances in 19 different directions.





Generator

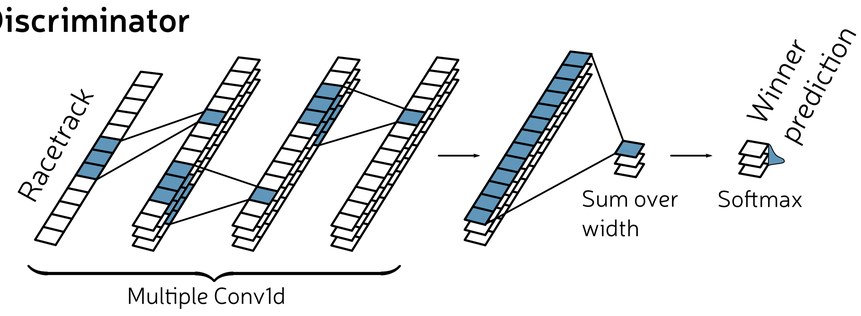


Objective

- Generator goal is to maximize *fairness* of generated levels by minimizing following loss:

$$\mathcal{L}^G(\theta) = \mathbb{E}_{\mathbf{z} \sim \mathbb{N}} \left[-\frac{1}{|A|} \cdot \log(D(G_\theta(\mathbf{z})) \right]$$

Discriminator



Objective

- Discriminator goal is to predict winner of given game level and by that approximate *fairness*. This is achieved by minimizing:

$$\mathcal{L}^D(\theta) = \mathbb{E}_{\mathbf{z} \sim \mathbb{N}} \left[-\text{game}(b, A) \cdot \log(D_\theta(G(\mathbf{z}))) \right]$$

Algorithm

Initialize all networks, A, G, D
while agents not trained **do**
  Reset environment
  while episode not finished **do**
    Perform actions decided by agents, A
    Gather rewards received by agents
    Update state of environment
  end while
  Use PPO to update agents, A, policies with data gathered during episode.
end while
for number of training iterations **do**
  Sample minibatch of noise vectors and generate boards, $\mathbf{z} \sim \mathbb{N}, b = G_\theta(\mathbf{z})$
  Simulate game and obtain winners, $w = \text{game}(b, A)$
  Update the discriminator, D_θ, parameters by descending gradient of:

$$\nabla \mathbb{E}_b \left[-w \cdot \log(D_\theta(b)) \right]$$

  Sample noise vectors, $\mathbf{z} \sim \mathbb{N}$
  Update the generator, G_θ, parameters by descending gradient of:

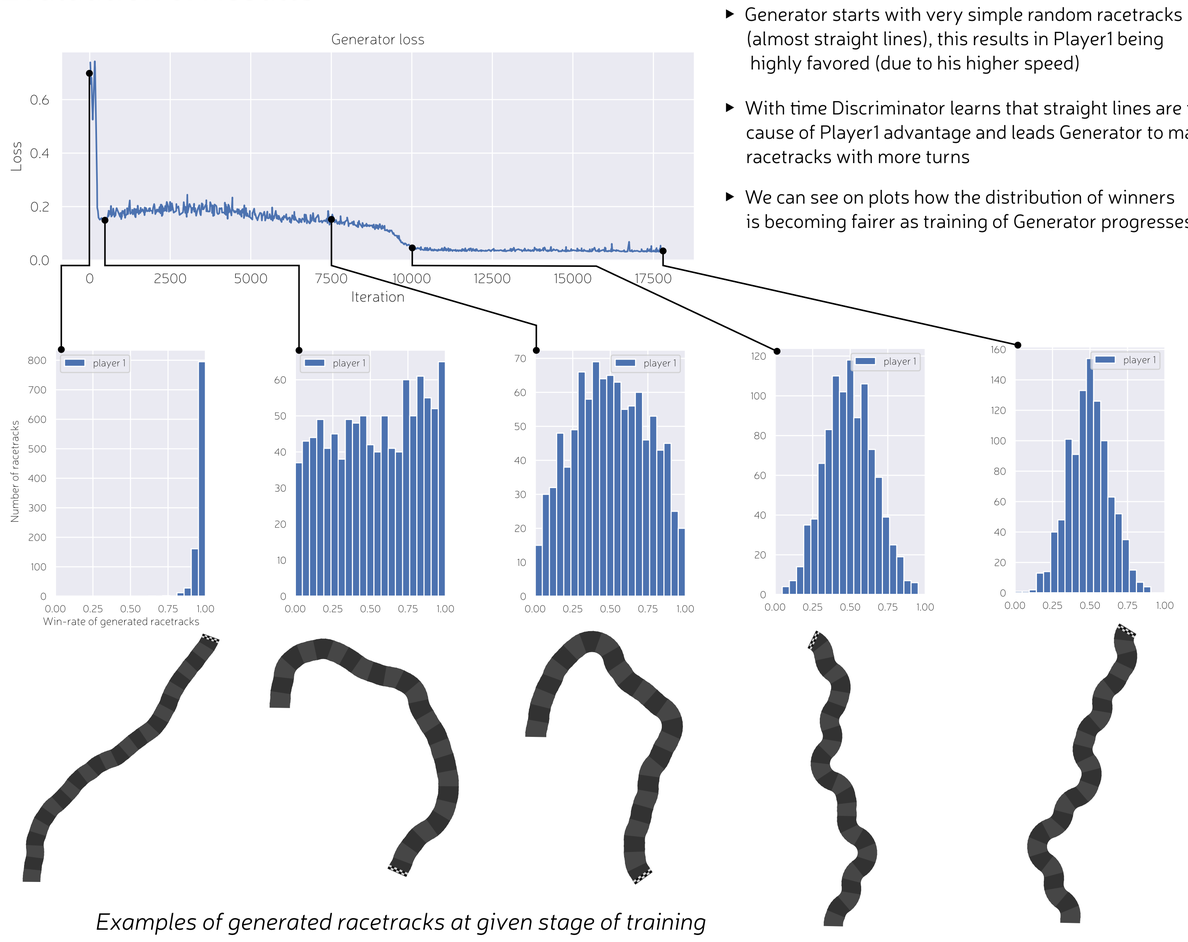
$$\nabla \mathbb{E}_\mathbf{z} \left[-\frac{1}{|A|} \cdot \log(D(G_\theta(\mathbf{z})) \right]$$

end for

Further work

- Due to catastrophic forgetting Agents are not trained simultaneously with Generator & Discriminator, maybe there is a way to avoid it and train them together.
- Find other ways to include Agents feedback in generation process.
- Investigate possible extensions to generator loss function to encourage more diverse results.

Evaluation & Results



Examples of generated racetracks at given stage of training

- Generator starts with very simple random racetracks (almost straight lines), this results in Player1 being highly favored (due to his higher speed)
- With time Discriminator learns that straight lines are the cause of Player1 advantage and leads Generator to make racetracks with more turns
- We can see on plots how the distribution of winners is becoming fairer as training of Generator progresses