A1822

09/07/2019

Autonomous Driving Car in Unity Simulation using Deep Reinforcement Learning

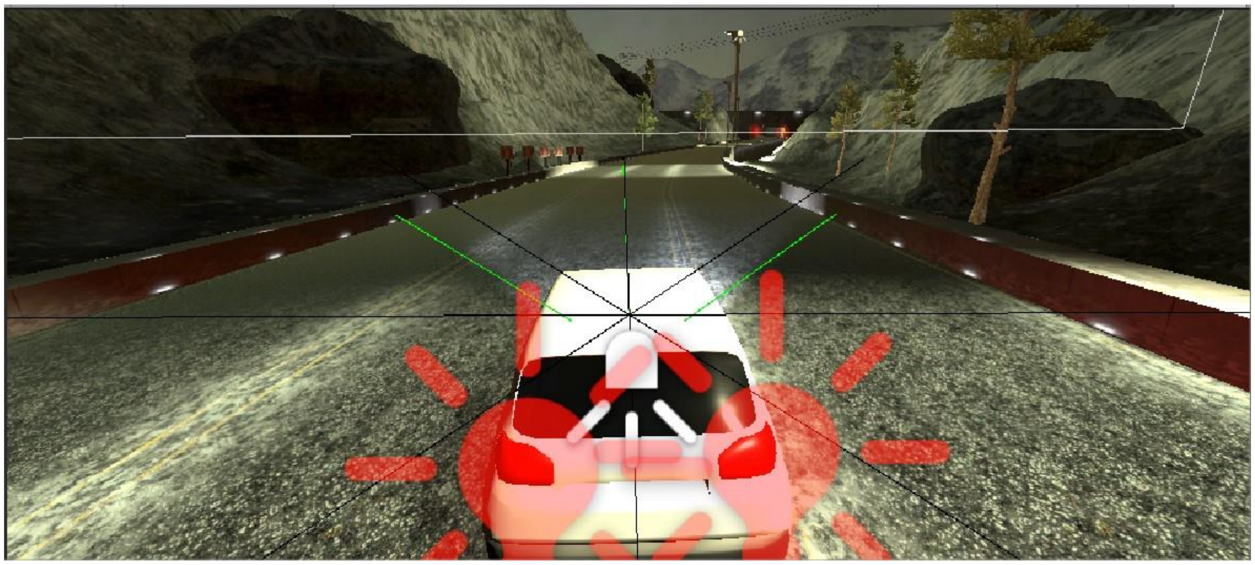


Figure 1

Video Link- https://www.youtube.com/watch?v=og1jGU8T8dE

Basic rundown-

RL was used to train a car in Unity simulation to drive a car and not crash for as long as possible. The car is physics based complete with suspension physics/tire physics. The car has constant acceleration at a set rate, speed is capped at 13.3m/s (47.9kmph). Output of the model is a float value between -1 and 1. -1 represents max steering angle to the left while 1 represents max steering to the right.

Inputs to the model-

- Three raycasts of length 100 units at the front (green lines shown in Fig1 and Fig-2), returns distance detected.
- Eight raycasts of length 20 units (black lines shown in Fig-1 and Fig 2), returns distance detected.
- Speed of the car.
- Drift value, which is calculated via dot product of car velocity direction and car forward direction. Measures if the car is oversteering/sliding or stable.
- Angle of the wheel.
- Velocity of the car is x-axis, y-axis and z axis.

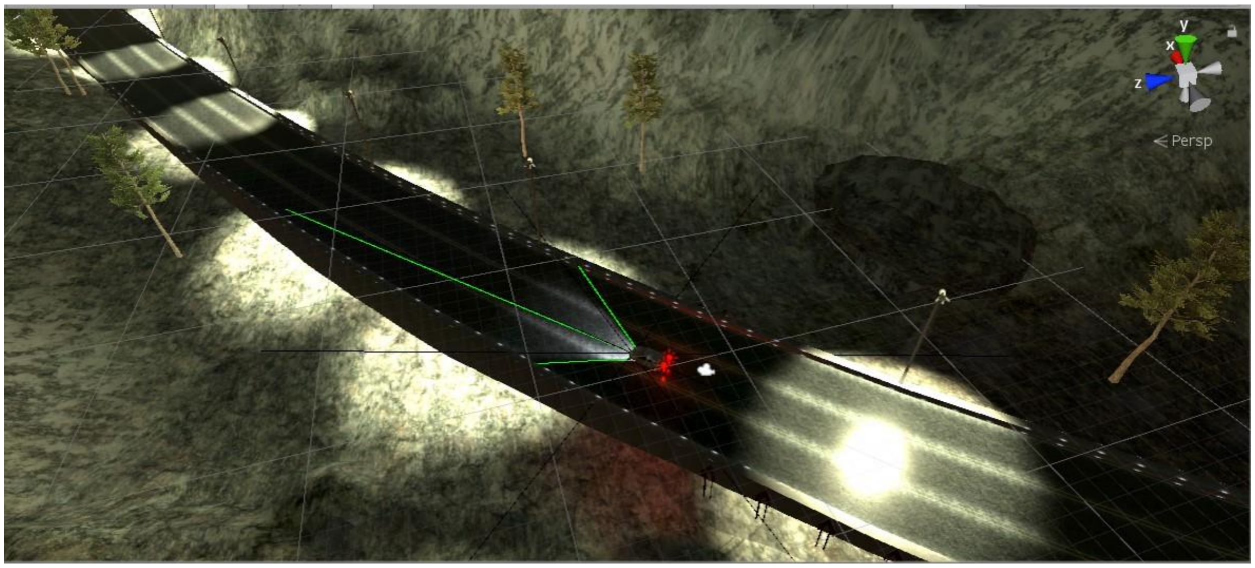


Figure 2

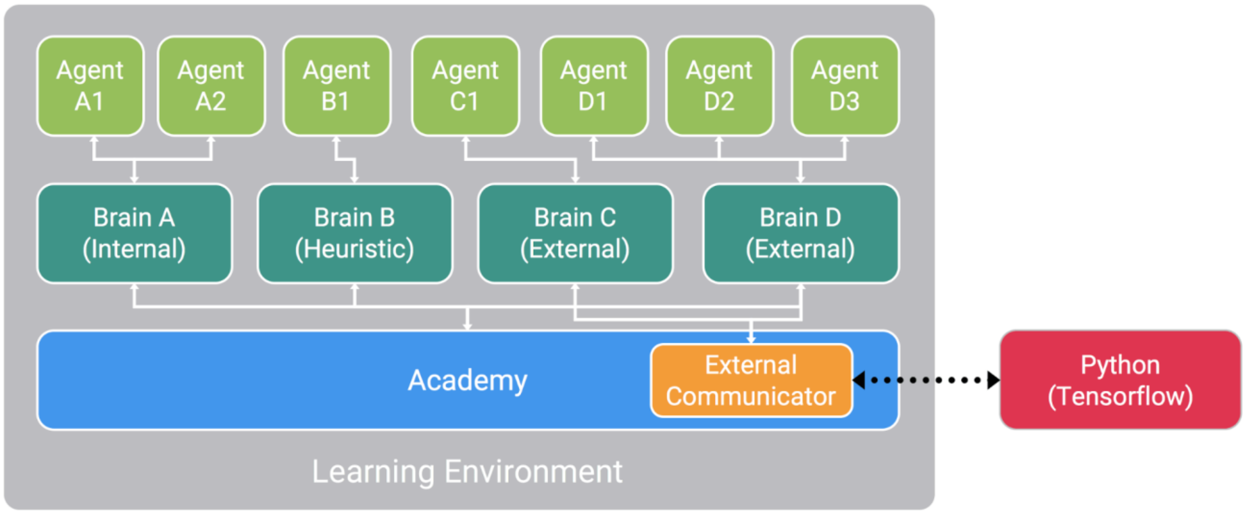
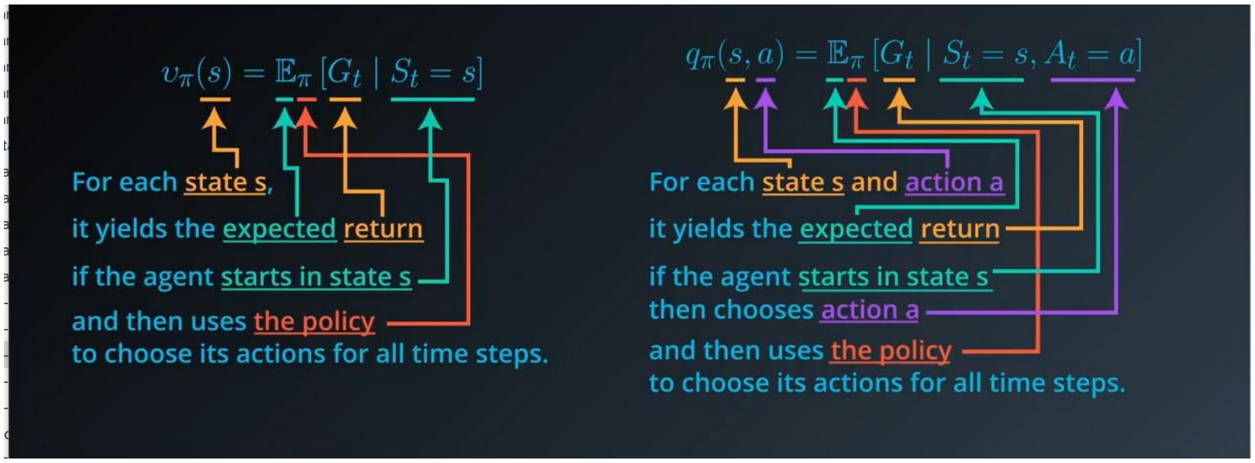


Figure 3 Unity ML-Agents



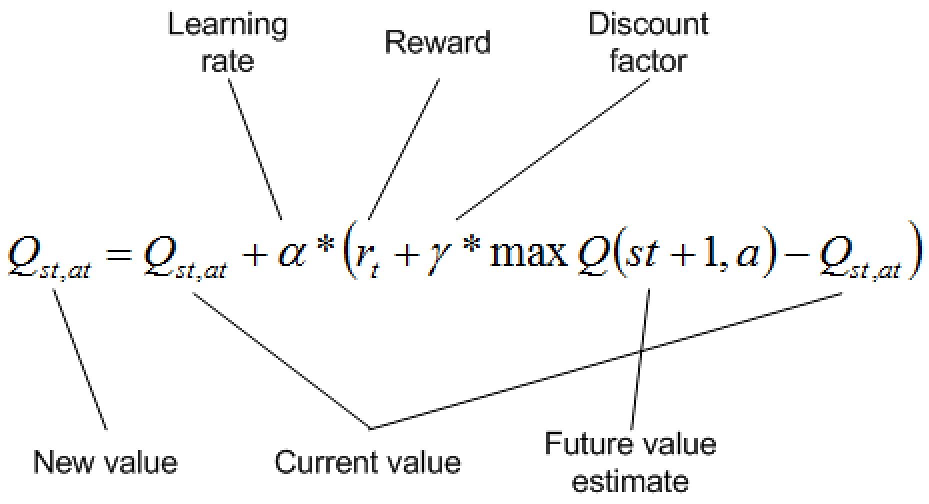
$$v_\pi(s) = \mathbb{E}_\pi\left[G_t \mid S_t = s\right]$$

For each state s,
it yields the expected return
if the agent starts in state s
and then uses the policy
to choose its actions for all time steps.

$$q_\pi(s, a) = \mathbb{E}_\pi\left[G_t \mid S_t = s, A_t = a\right]$$

For each state s and action a
it yields the expected return
if the agent starts in state s
then chooses action a
and then uses the policy
to choose its actions for all time steps.

Figure 4



$$Q_{st,at} = Q_{st,at} + \alpha * \left(r_t + \gamma * \max Q(st+1, a) - Q_{st,at}\right)$$

Figure 5

Rewards and Punishments-

Rewards-

- (forward ray distance/6000f)
- (forward left ray distance/10000f)
- (forward right ray distance/10000f)
- if(driftValue>0.90) then 0.01 reward
- if(car_Speed>5) then 0.001 reward
- if(car_Speed>10) then 0.1 reward

Punishments-

- (-(100f- forward ray distance)/50000f)
- (-(100f- forward left ray distance)/90000f)
- (-(100f- forward right ray distance)/90000f)
- if any forward green rays less than 4f then -0.08 reward
- if(driftValue<0.90) then -0.05 reward
- if any forward green rays less than 1f then resets reward to -1 for the current step and resets car position

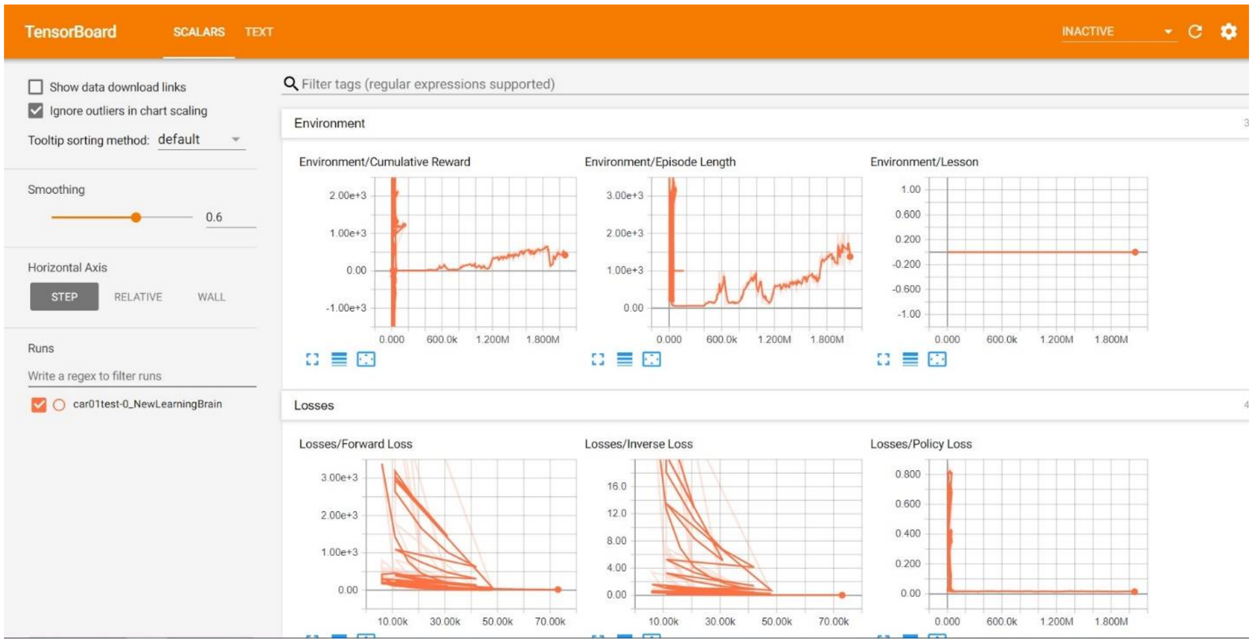
Training info-

Car agent was trained for 1.8 million “steps”. Car agent has 10000 steps to not crash, after 10000 or a crash, car agent respawns. Inputs, raycast angles/position, car agent spawn points were tweaked as necessary between steps as model was routinely saved. Trainer used was Proximal Policy Optimization (PPO).

Hyperparameters-

```
trainer: ppo

    beta: 5.0e-3

    epsilon: 0.2
    lambd: 0.95
    learning_rate: 3.0e-4

    memory_size: 256

    sequence_length: 64

    use_recurrent: false
    use_curiosity: false
    curiosity_strength: 0.01
    curiosity_enc_size: 128
    normalize: true
    num_epoch: 3
    time_horizon: 1000
    batch_size: 2024
    buffer_size: 20240
    gamma: 0.995
    max_steps: 5e6
    summary_freq: 3000
    num_layers: 3
    hidden_units: 512
```

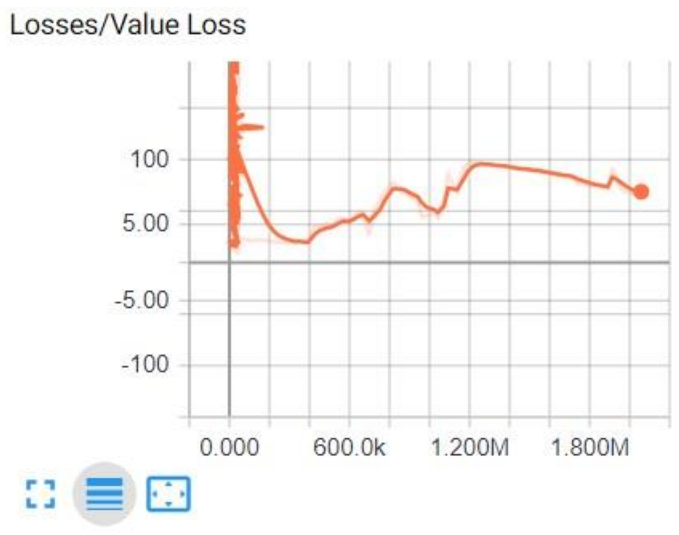
Tensorboard-


☐ Show data download links
☑ Ignore outliers in chart scaling
Tooltip sorting method: default

Smoothing 0.6

Horizontal Axis [✓ STEP] RELATIVE WALL

Runs
Write a regex to filter runs
☑ car01test-0_NewLearningBrain



Overfitting in RL-

Overfitting is “to fall into a local minimum”. In this sim the RL models generates its own data. An agent looks for an optimal strategy, interacting with the environment. If environment reacts “kindly”, the strategy is reinforced, the agent is “motivated” to repeat the same actions over and over aiming to get higher reward. If there is an exploit in the rewards, the agent inevitably learns to exploit it. To avoid this the rewards and punishments were tweaked until the agent learned to generalize.

Video link-

https://www.youtube.com/watch?v=og1jGU8T8dE

Project executable link-

https://drive.google.com/open?id=16Vgichx38PfQy13Udjuevkn3HS5ajYRY

Assets Used-

- Unity Vehicle Pack
- Mountain Road

Conclusion-

In RL the most difficult thing to achieve is the right balance of reward function. Doing this project resulted in a better understanding of that. This project maybe further improved by addition of things such as tire pressure, gravity measurements, virtual cameras, traffic, off-road environments, pedestrians, giving the agent acceleration control, braking control and so on.