10th May, 2018

SELF-DRIVING CAR

USING DEEP REINFORCEMENT LEARNING

Prepared by: Yahya Alaa Massoud
Reg. Number: 14 10 20 70

SELF-DRIVING CARS

What’s the future of personal transportation? Well, you’ll likely be spending a lot less time behind the wheel, for one.

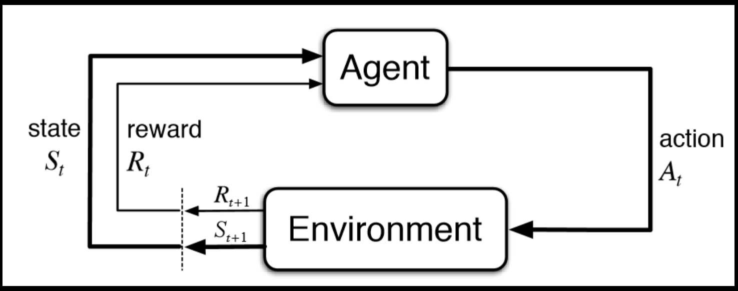
Cars today already include many semi-autonomous features, like assisted parking and self-braking systems. And completely autonomous vehicles - able to operate without human control - are rapidly becoming more of a reality.

Thanks, to Artificial Intelligence and Data Science, we have reached new ways of making cars learn how to drive more safely and more human-like.

Reinforcement Learning is a method that lets us train an agent to perform a specific task in an environment, which in this case, a car tries to drive safely in the streets of the city. Let’s talk more about Reinforcement Learning.

REINFORCEMENT LEARNING

Reinforcement learning is mainly based on trying to make an agent perform some actions in a specific environment, and for every action it takes, the environment gives a feedback - called a Reward - to the agent telling it how much was his action good.



Reinforcement Learning makes the agent learns an optimum policy after taking many actions in the environment and getting rewards for each of his actions.

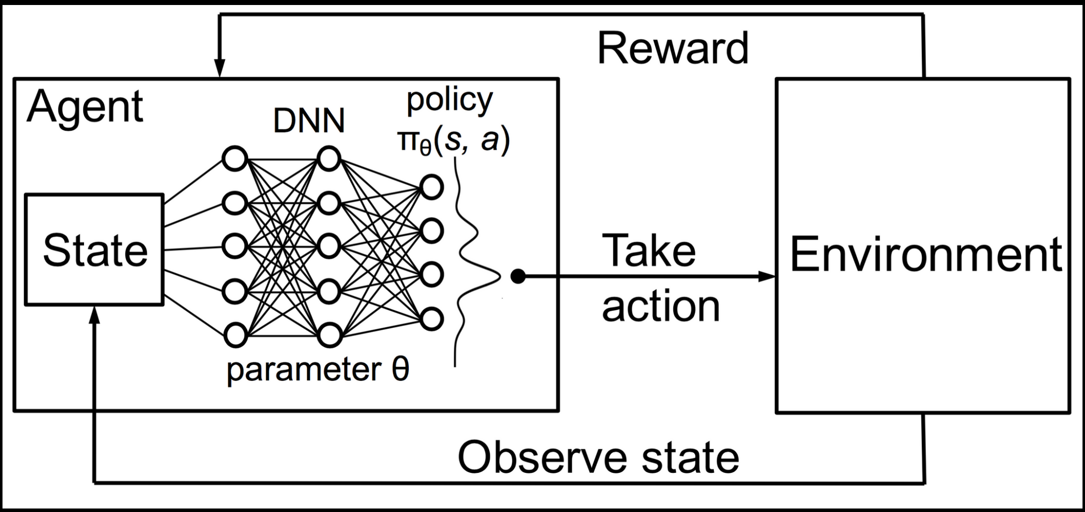
At each state, the agent takes an action and gets a reward as a feedback. At every state the agent visits, it has to choose the action that will maximize the expected future reward, as it always wants to perform optimally in the given task.

An expected future reward for a specific action is called ”Quality Value”, and Q-learning is the way that we can make an agent predict the Q-value of a given action in a certain state.

DEEP REINFORCEMENT LEARNING

One way to predict the Q-value of a given state is to do dynamic programming in a table-like data structure, to be able to update visited states' Q values. But in a such a complex state space like a self-driving car, which has its state as an image (a lot of RGB pixels), we have to find another way to predict the Q-value of a chosen action by the agent.

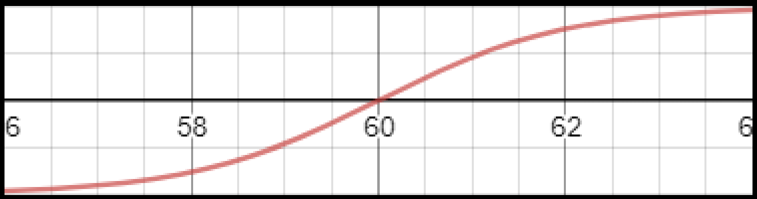
Here comes the power of Deep Neural Nets as function approximators, instead of using a table-like data structure for predicting the Q-values for states, we feed a state into a neural network as input, we pass it through many layers, and then get the output as a linear function that tells us the Quality Value of each action in the agent's action space.



OUR SELF-DRIVING R/C CAR

In our project, we used Deep Reinforcement Learning to train an R/C car to avoid obstacles while driving, by capturing images that describe the current view of the car as the ”State”, and where the action space is the valid car movements: Right, Left, and Forward.

But there is still a missing part, which is the car’s reward function, that was a challenging part of our project. We used an Ultrasonic sensor to be able to calculate the distance between the front of the car and the objects in front of it, after calculating this distance, we had to map it to a value from -1 to 1, making this mapping makes the algorithm converge faster than having large reward values for each state. We fed the calculated distance into a function we developed, which is a modified version of the Sigmoid function. Where the threshold number (60) means to the agent that a distance of 60 is the minimum distance to reach, more than 60 is safe, less than 60 is dangerous and collision is more probable.



HARDWARE COMPONENTS

- R/C car.
- Ultrasonic Sensor.
- Relays.
- Raspberry Pi.

SOFTWARE COMPONENTS

Python code that fully controls the Raspberry Pi:
captures images as current states,
calculates distance,
changes moving direction while driving (R, L, or F).

Python RESTful API for Reinforcement Learning, that we have developed during the project, all the agent have to do is to send a POST request that has the (current state, reward) as data in the request's body to the server, and since the API has already all of the Q-learning logic, it will predict the Q-values for each action based on the current state, and returns the best action to take, this makes it easier for the user, as he is only responsible for sending the current state and defining a good reward function to be able to judge the agent's moves fairly, while everything in Q-learning algorithm's logic happens behind the scenes on the server side.