Deep Reinforcement Learning Autonomous Driving Using Lidar in the Physical World

Michael Bosello
michael.bosello@studio.unibo.it

Università di Bologna – Department of Computer Science and Engineering, Cesena, Italy

Autonomous systems

Outline

Next in Line...

Autonomous Driving

Three sub-problems

Recognition Identify environment's components

Prediction Predict the evolution of the surrounding

Decision making Take decisions and act to pursue the goal

Two approaches

- Single task handling
  - Use human ingenuity to inject knowledge about the domain
- End-to-end
  - Let the algorithm optimize towards the final goal without constrains

Environment

- Non-deterministic
- Partially observable
- Dynamic

End-to-end

Supervised Learning

- Based on imitation
- Current approach by major car manufacturer
- **Robotic drivers are expected to be perfect**

Drawbacks

Training data Huge amounts of labeled data or human effort

Covering all possible driving scenarios is very hard

Unsupervised Learning

- Learns behaviors by trial-and-error
  - like a young human driving student

- Does not require explicit supervision from humans.

- Mainly used on simulated environment
  - to avoid consequences in real-life
  - An agent trained in a virtual environment will not perform well in the real setting

Next in Line...

Reinforcement learning in the physical world

- Depart from the assumption of infallible self-driving vehicles
  - Granting some time to learn how to drive in certain scenarios
- Use of very realistic 1/10 scale car prototype
  - the agent still faces challenges of a real driving scene
  - inexpensive
  - safe
- We used Deep Q-Network (DQN) to understand the feasibility of the approach

Use of LIDAR

- Few works use LIDAR data in RL contexts
  - LIDAR measurements greatly affected by reflection [Ivanov et al., 2020]
  - Training of RL agent with real LIDAR data is considered an *open problem* [Ivanov et al., 2020]
  - 1D CNN used to process LIDAR data for the first time

Next in Line...

Agent-environment interaction

- In RL, an agent learns how to fulfill a task by interacting with its environment
- The interaction between the agent and the environment can be reduced to:

State	Every information about the environment useful to predict the future
Action	What the agent do
Reward	A real number that Indicates how well the agent is doing

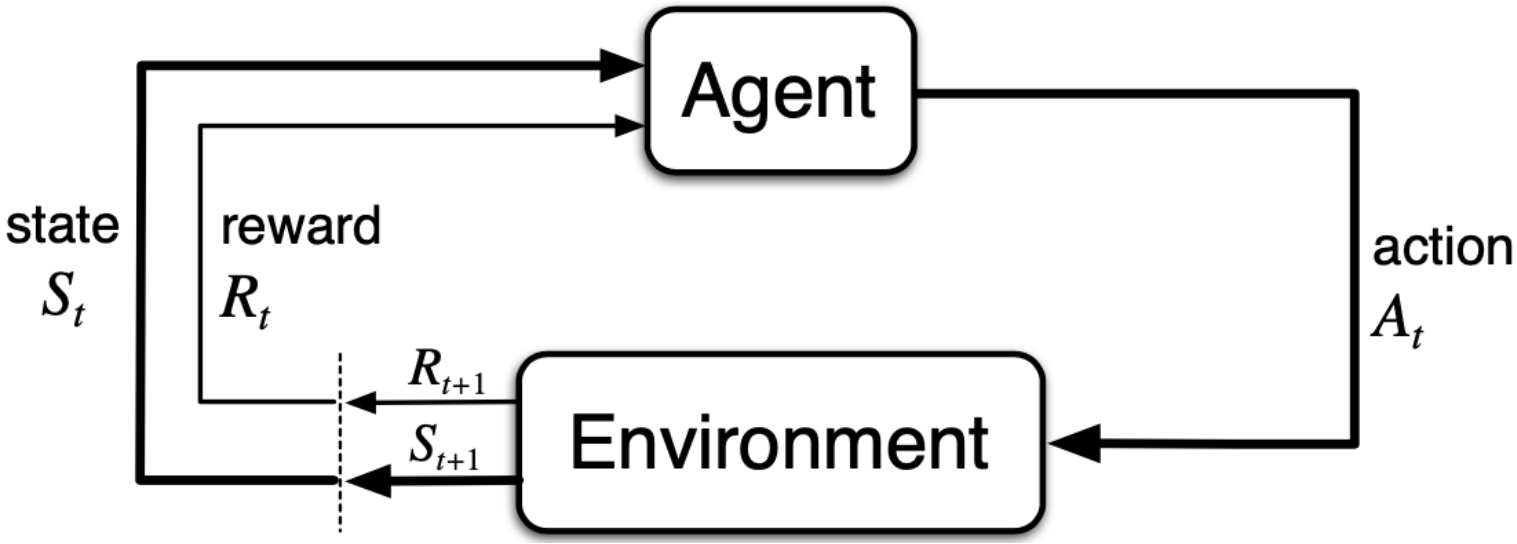


Figure: At each ***time step*** t the agent perform an action A_t and receives the couple S_{t+1}, R_{t+1} (source [Sutton and Barto, 2018])

Terminology

Policy

- A map from states to actions used by the agent to choose next action

Episode

- A complete run from one of the initial states to a final state

Markov Decision Process (MDP)

A RL problem can be formalized as a MDP. The 4-tuple:

- S set of states
- A_s set of actions available in state s
- $P(s'|s,a)$ state-transition probabilities (probability to reach s' given s and a)
- $R(s,s')$ reward distribution associate to state transitions

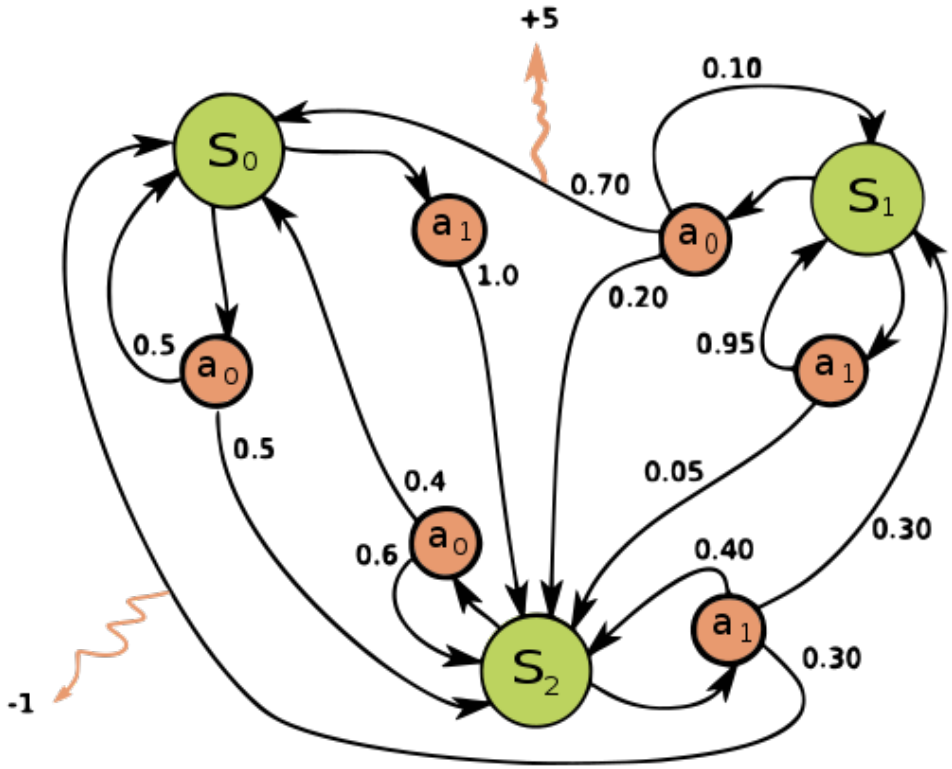


Figure: Example of a simple MDP (source Wikipedia)

Environment features

- MDPs can deal with non-deterministic environments (because they have probability distributions)
- But we need to cope with observability

Markov Property – Full observability

- Classical RL algorithms rely on the Markov property
- A state satisfy the property if it includes information about all aspects of the (past) agent-environment interaction that make a difference for the future

From states to *observations* – Partial observability

- Sometimes, an agent has only a partial view of the world state
- The agent gets information (observations) about some aspects of the environment
- Abusing the language, observations/history are called "state" and symbolized S_t

How to deal with partial observability?

- Approximation methods (e.g. NN) doesn't rely on the Markov property
- Sometimes we can reconstruct a Markov state using a history
  - Stacking the last n states
  - Using a RNN (e.g. LSTM)

Learning a policy

- The agent wants to maximize the cumulative reward

Cumulative reward

- The discounted sum of rewards over time

- A way to produce a policy is to estimate the action-value function
  - given the function, the agent needs only to perform the action with the greatest value

Action-value function

- (state, action) $\rightarrow$ expected return (the expectation of future rewards)

The exploration/exploitation dilemma

- The agent has to behave optimally but it needs to explore the environment to improve
- ϵ-greedy policy: the agent behave greedily but there is a (small) ϵ probability to select a random action.

Deep Q-Network

Q-learning

- Is an algorithm that estimates the action-value function
- At every iteration the estimation is refined thanks to the new experience.
- Every time the evaluation becomes more precise, the policy gets closer to the optimal one.

DQN

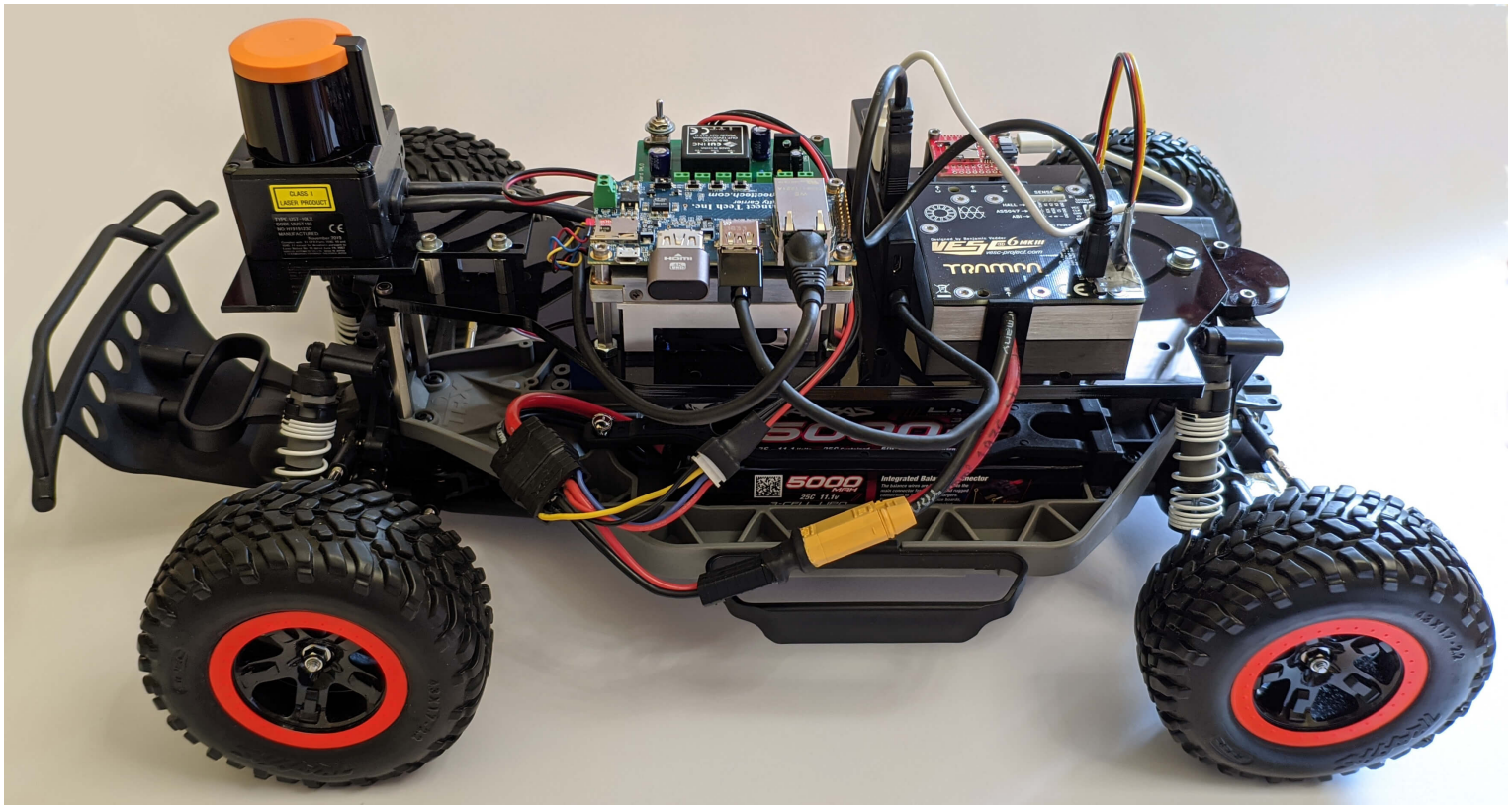
- When the number of states increases, it become impossible to use a table to store the Q-function
- A neural network can approximate the Q-function
- We also get better generalization and the ability to deal with non-Markovian envs

Design the driver agent

- Environment: states and actions, reward function
- Neural network
- Training cycle

Next in Line...

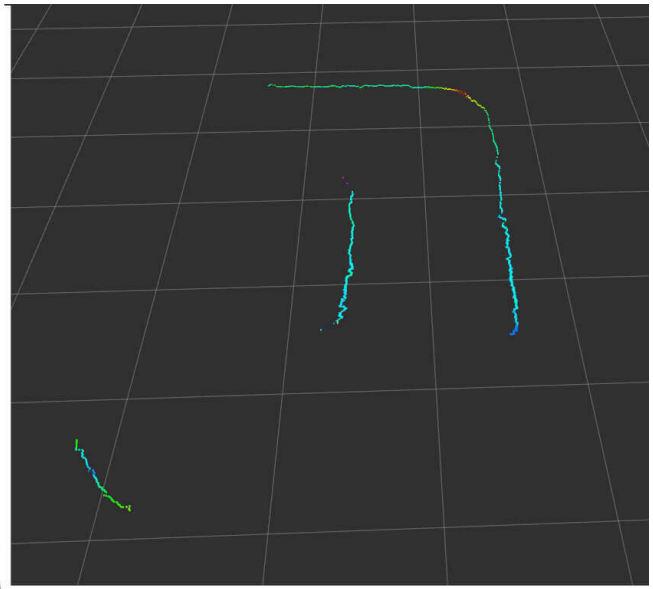
F1tenth



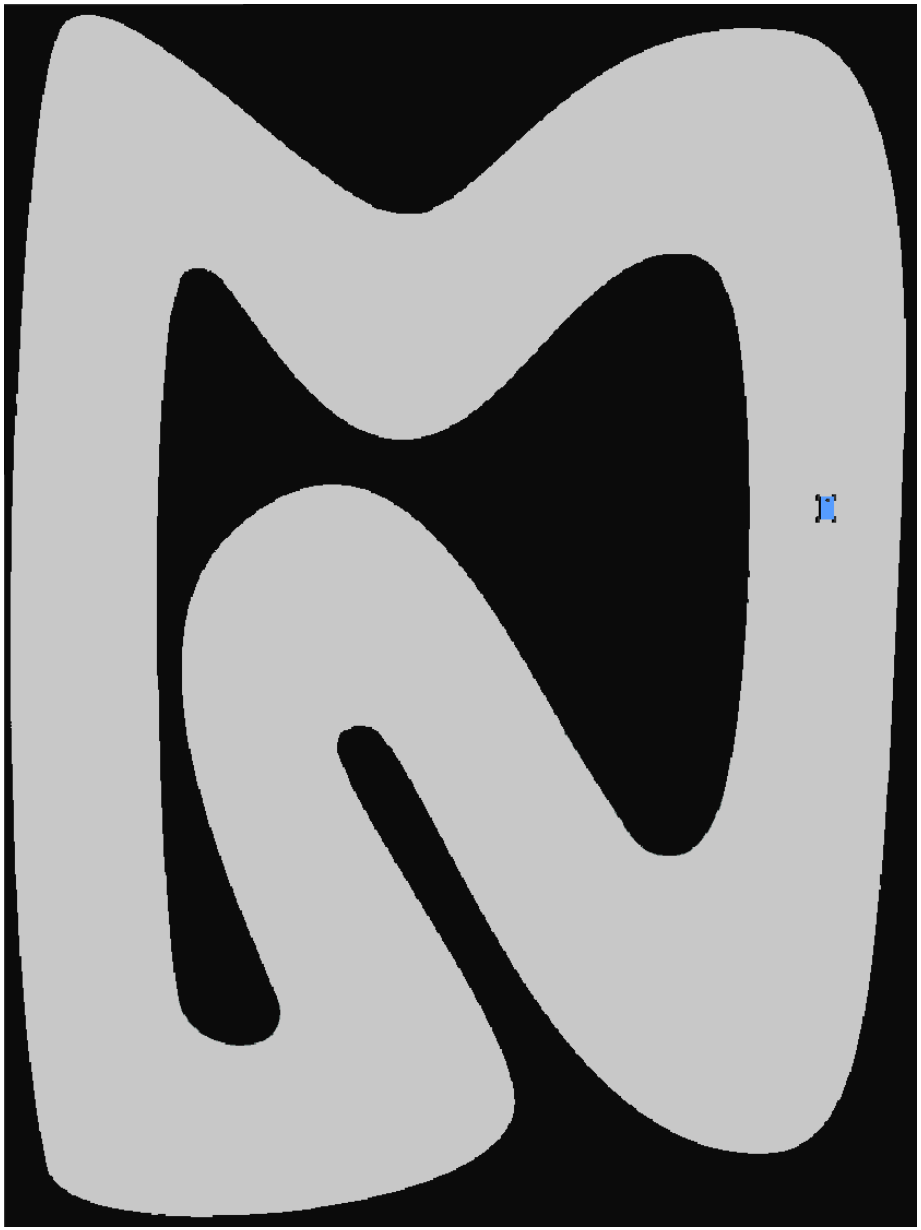
Realistic dynamics

- Ackermann steering
- High speeds

Physical racetrack



Simulated racetrack



RL environment I

ROS

- LIDAR data and actuator commands updated asynchronously
- Automatic emergency breaking
  - It has priority over RL decisions

State

- Two subsequent LIDAR measurements

Action

- Go forward
- Turn right
- Turn left

Episode

- An episode ends when automatic emergency breaking activates
- The car goes backward, then a new episode starts

RL environment II

Reward

Real car

- Automatic emergency breaking = -1
- Reward proportional to the car velocity (odometry), max = 0.09
- Additional bonus proportional to the distance to the nearest obstacle

Simulator

- Automatic emergency breaking = -1
- forward = 0.2
- turn = 0.05

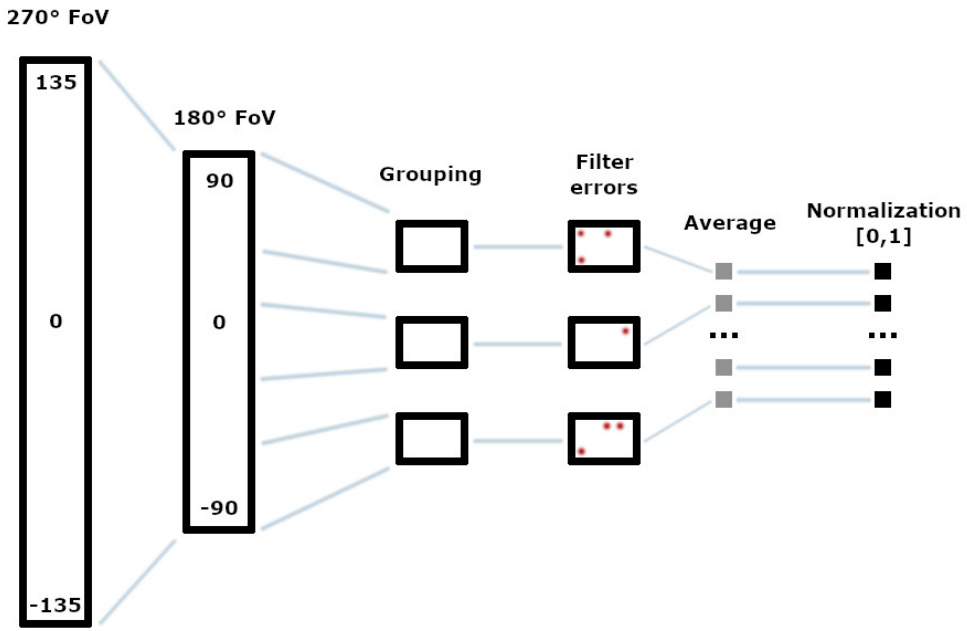
Next in Line...

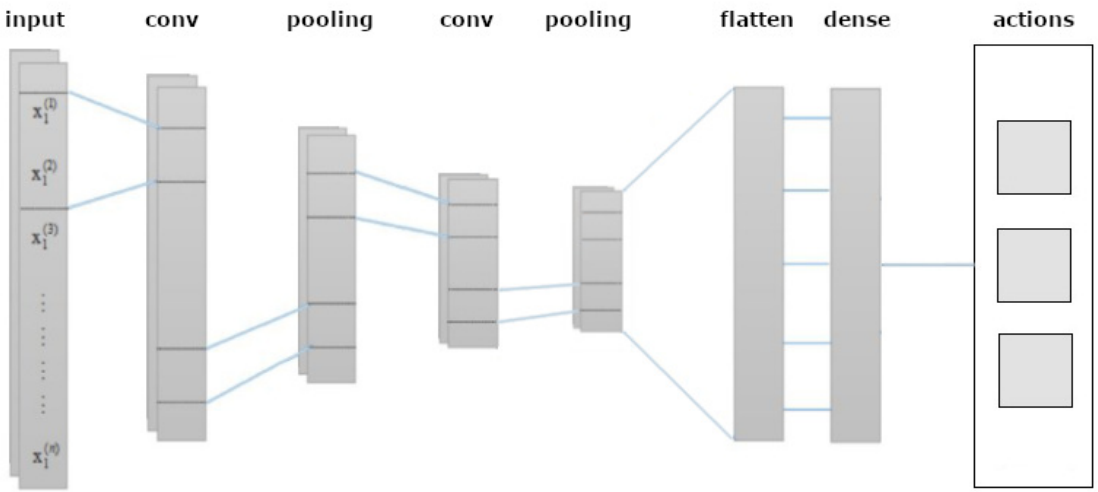
Driver agent software I

Two parts

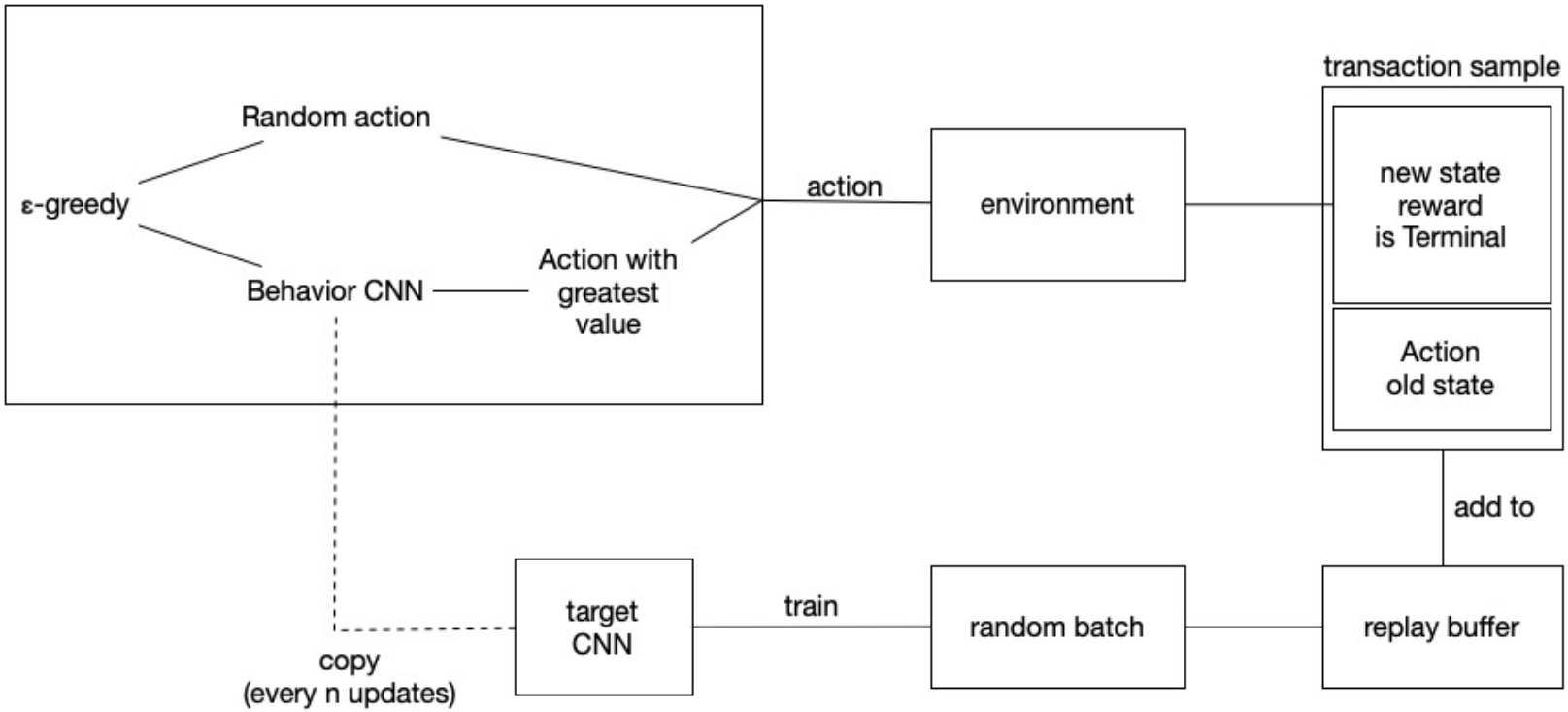
- Neural network
- Training cycle

Driver agent software II





Driver agent software III



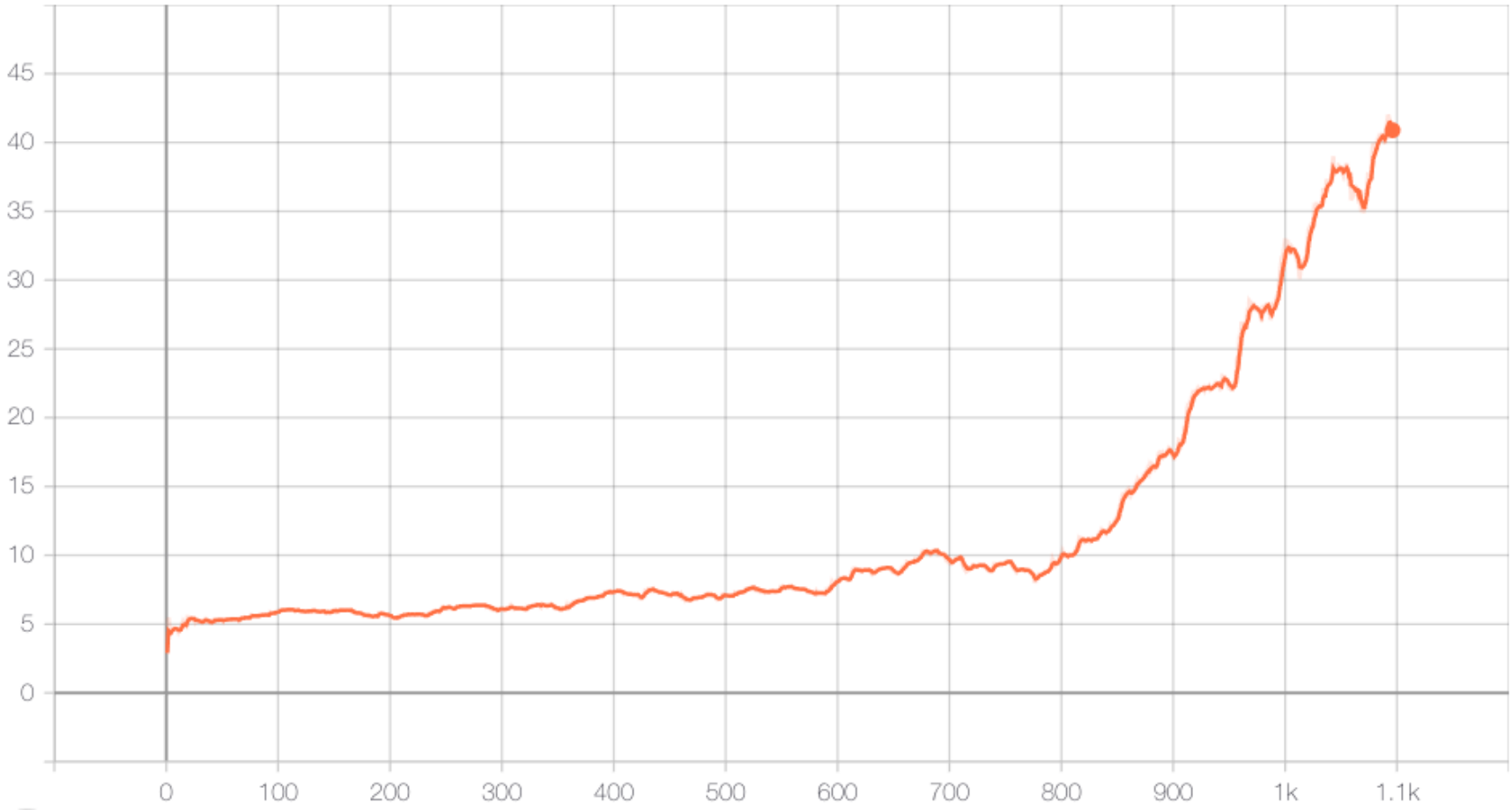
- Experience replay is needed to break temporal relation
- Transaction samples are used to compute the target update

Next in Line...

Conclusions I

Results

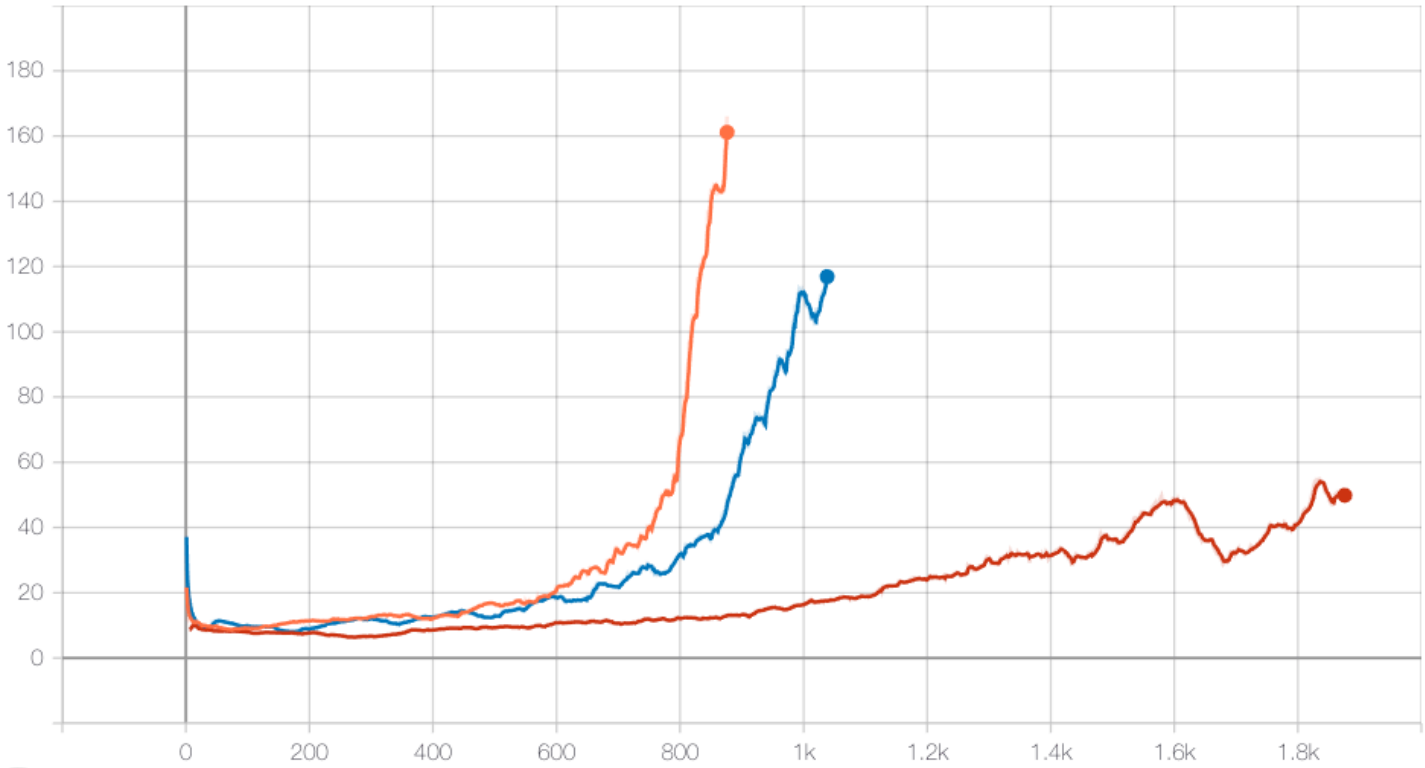
- The driver-agent successfully learned a control policy
  - using real LIDAR data



Conclusions II

NNs comparison

- CNNs perform better on structured and spatially related data
- 1D CNNs are very effective in processing LIDAR data
  - showing a behavior oriented to cut curves so as to minimize turning and maximize forwarding



- 1D CNN: orange
- Dense: blue
- 2D CNN: red

Results demo

[https://youtu.be/ardg7-7Pevw]

Next in Line...

Future works

- Camera vs LIDAR
- Continuous action space RL
- Meta-learning
- Cooperative learning
  - exchange of experience between cars with efficient communication [Kamp et al., 2018]
- Reinforcement learning with bounded risk [Geibel, 2001]
  - find the optimal policy with bounded risk
  - risk as the probability to enter in a fatal state
- Safe reinforcement learning [Shalev-Shwartz et al., 2016]
  - ensure functional safety through hard constrains
- From 1/10 to real vehicles
- **Move to an urban scenario and make the agent follows high-level directions**

Deep Reinforcement Learning Autonomous Driving Using Lidar in the Physical World

Michael Bosello
michael.bosello@studio.unibo.it

Università di Bologna – Department of Computer Science and Engineering, Cesena, Italy

Autonomous systems

References I


Geibel, P. (2001).
Reinforcement learning with bounded risk.
In *ICML*, pages 162–169.


Ivanov, R., Carpenter, T. J., Weimer, J., Alur, R., Pappas, G. J., and Lee, I. (2020).
Case study: Verifying the safety of an autonomous racing car with a neural network controller.
In *Proceedings of the 23rd International Conference on Hybrid Systems: Computation and Control*, HSCC '20, New York, NY, USA. Association for Computing Machinery.


Kamp, M., Adilova, L., Sicking, J., Hüger, F., Schlicht, P., Wirtz, T., and Wrobel, S. (2018).
Efficient decentralized deep learning by dynamic model averaging.


Shalev-Shwartz, S., Shammah, S., and Shashua, A. (2016).
Safe, multi-agent, reinforcement learning for autonomous driving.


Sutton, R. S. and Barto, A. G. (2018).
Reinforcement learning : an introduction.
The MIT Press.

Deep Reinforcement Learning Autonomous Driving Using Lidar in the Physical World

Michael Bosello
michael.bosello@studio.unibo.it

Università di Bologna – Department of Computer Science and Engineering, Cesena, Italy

Autonomous systems