

# Self-Driving Cars

## Exercise 2 - Reinforcement Learning

Kashyap Chitta

Autonomous Vision Group  
MPI-IS / University of Tübingen

November 22, 2019



University of Tübingen  
MPI for Intelligent Systems  

---

**Autonomous Vision Group**



# Exercise Setup

Download `sdc_exercise_02_reinforcement_learning.zip` which contains:

- ▶ Exercise sheet & slides
- ▶ Code template

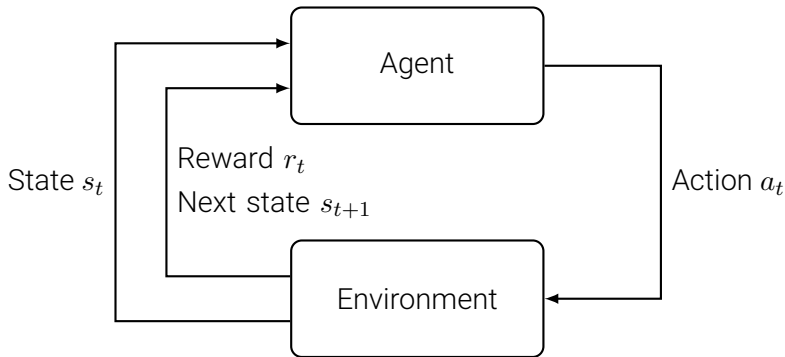
Submit `.zip` folder which contains:

- ▶ Your report of up to 3 pages (`.pdf`)
- ▶ Your best model (`agent.pt`)
- ▶ Your Python code (`.py`)

Deadline: **Wed, 18. December 2019 - 21:00**

# Reinforcement Learning

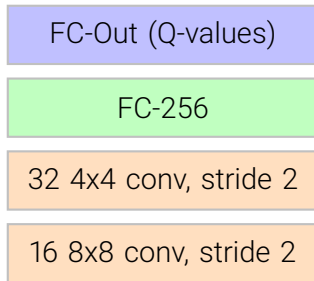
# Reinforcement Learning



- ▶ Agent observes environment state  $s_t$  at time  $t$
- ▶ Agent performs action  $a_t$  at time  $t$
- ▶ Environment returns the reward  $r_t$  and its new state  $s_{t+1}$  to the agent

# Deep Q-network

Use a deep neural network with weights  $\theta$  to estimate  $Q(s, a; \theta) \approx Q^*(s, a)$ :



# Deep Q-Learning

Training a deep Q-network using **experience replay** and **fixed Q-targets**

- ▶ Take action  $a_t$  according to  $\epsilon$ -greedy policy
- ▶ Store transition  $(s_t, a_t, r_t, s_{t+1})$  in replay memory  $D$
- ▶ Sample random mini-batch of transitions  $(s, a, r, s')$  from  $D$
- ▶ Compute Q-learning targets w.r.t. old, fixed parameters  $\theta^-$
- ▶ Optimize MSE between Q-network and Q-learning targets:

$$L_i(\theta_i) = \mathbb{E}_{s,a,r,s' \sim D_i} \left[ \left( r + \gamma \max_{a'} Q(s', a'; \theta_i^-) - Q(s, a; \theta_i) \right)^2 \right]$$

using a variant of stochastic gradient descent

1.1

Base Implementation

# Code Template

The provided code template contains:

- ▶ Deep Q-network (**to-do**)
- ▶ Deep Q-learning (**to-do**)
- ▶ Action selection (**to-do**)
- ▶ Training (implemented, incl. replay buffer, exploration)
- ▶ Evaluation (implemented)



## a) Deep Q-network

Implement a deep Q-network and its forward pass:

- ▶ Start with a simple network architecture
  - ▶ Some convolution + fully connected layers
  - ▶ Probably no need for batch normalization, dropout or residual architectures
  - ▶ Use a single frame as input to the network
  - ▶ You may again use the `extract_sensor_values` function
- ▶ Get inspired by the original DQN architecture for playing Atari
- ▶ Try to adapt your network architecture from Exercise 1

## b) Deep Q-learning

Implement the deep Q-learning update step (see *DQN Nature paper for details*):

1. Sample transitions from replay buffer
2. Compute  $Q(s_t, a)$
3. Compute  $\max_a Q(s_{t+1}, a)$  for all next states
4. Mask next state values where episodes have terminated
5. Compute the target and loss
6. Calculate and **clip** the gradients
7. Optimize the model

Implement the target update

## c) Action selection

Implement selecting an exploratory  $\epsilon$ -greedy or greedy action:

- ▶ With probability  $\epsilon$  choose an action at random
- ▶ With probability  $1 - \epsilon$  choose the greedy action:

$$\pi(a|s) = \begin{cases} 1 & \text{if } a = \operatorname{argmax}_{a \in \mathcal{A}} Q(s, a; \theta) \\ 0 & \text{otherwise} \end{cases}$$

## d) Training

Train a deep Q-learning agent:

- ▶ Use script `train_racing.py` to train locally
- ▶ Use script `train_racing_cluster.py` to train on the TCML cluster
  - ▶ No support for GPU training when running on the cluster
  - ▶ Do not use `--nv` in your `.sbatch` file
  - ▶ Edit your `.sbatch` file to use `day` partition and `1-0` time
- ▶ Start with the provided default parameters
- ▶ Training produces two plots
  - ▶ Loss curve
  - ▶ Episode rewards

## e) Evaluation

Evaluate the trained deep Q-learning agent:

- ▶ Use script `evaluate_racing.py` to evaluate locally
- ▶ Use script `evaluate_racing_cluster.py` to evaluate on the TCML cluster
  - ▶ Again, no GPU support - do not use `--nv` option in your `.sbatch` file
- ▶ Script will output preliminary leaderboard score
- ▶ Visual evaluation on local machine should be performed

**Important:** Make sure you have a working baseline implementation (i.e. agent is able to take some corners) before moving on to work on the next part of the exercise.

## 1.2

### Further Investigations and Extensions

## a) Discount Factor

Investigate the influence of the discount factor  $\gamma$ :

- ▶ Why do we use a discount factor  $\gamma$  in general?
- ▶ In which cases would it be a problem not to use a discount factor (i.e.  $\gamma = 1$ )?
- ▶ What happens if you increase / decrease  $\gamma$  from its default of 0.99?
- ▶ Any effects on the behavior and the evaluation score of the agent?

## b) Action Repeat Parameter

Investigate the influence of the `action_repeat` parameter:

- ▶ By default, an action is selected on every 4th frame and performed 4 times
- ▶ Why might this be helpful?
- ▶ What happens if you increase / decrease this parameter?



## c) Action Space

Investigate the influence of adding more actions:

- ▶ By default, the agent is trained with a set of 4 actions
- ▶ What happens if more actions are added?
- ▶ Why are we limited in DQN to a discrete set of actions?
- ▶ Why might adding more actions not always be helpful?

## d) Double Q-learning

Implement double Q-learning:

- ▶ Why does standard deep Q-learning overestimate  $Q$ -values?
- ▶ How does double Q-learning solve this problem?
- ▶ What is the effect on the training and performance of your agent?

**Important:** Include your double Q-learning implementation in your code submission

## e) Best Solution

Put together your findings and fine-tune your agent:

- ▶ What changes did you make to the baseline agent?
- ▶ Where does it improve upon the baseline agent?
- ▶ How does it compare to your imitation learning agent?
- ▶ Which aspects could still be improved (and maybe how)?

**Submit** your best code and model along with your report

Competition

# Competition

Submit your evaluation scores to the leaderboard:

- ▶ See exercise sheet for submission and leaderboard URL
- ▶ Make sure not to overfit on the provided evaluation tracks
- ▶ Final evaluation will be performed by us on a secret set of tracks
- ▶ Winners will present their approach in the last lecture

Advice

# Advice

**Read the DQN papers** on playing Atari games

- ▶ In particular, the Nature paper by Mnih et al. (2015)
- ▶ Check their pseudo-code

**Start early**

- ▶ Training a reinforcement learning takes time (several hours)
- ▶ You will need to train and evaluate several agents for this exercise
- ▶ Don't start too late or you will run out of time

Questions?