Self-Driving Cars

Exercise 3 - Reinforcement Learning

Joo Ho Lee
Autonomous Vision Group
University of Tübingen

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

ellis
European Laboratory for Learning and Intelligent Systems

Exercise Setup

Download `exercise_03_reinforcement_learning.zip` which contains:

- Exercise sheet & slides
- Code template

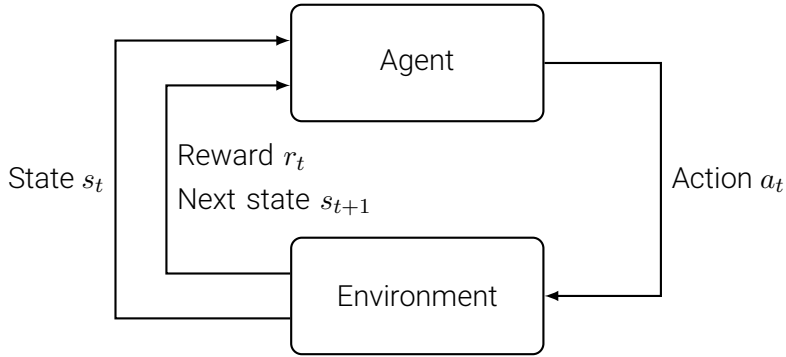
Submit `.zip` folder which contains:

- Your best model (`agent.t7`)
- Your Python codes (`.py`)
- (Optional) Your action file (`best_actions.txt`)

Deadline: **Tue, 21. December 2021 - 21:00**

Reinforcement Learning

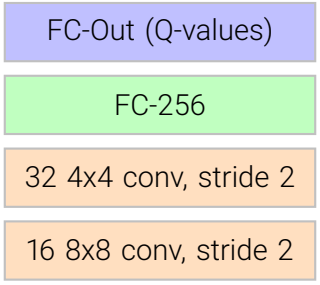
Reinforcement Learning



- Agent oberserves 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 θ 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 ϵ-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 θ^-
- 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 ϵ-greedy or greedy action:

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

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

d) Training

Train a deep Q-learning agent:

- Use script `train_racing.py` to train locally
- Add argument `--display` to show the window
  - On the cluster, call `python train_racing.py`
  - Set `--nv` in your `.sbatch` file if you want to use a GPU unit for training.
- 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 argument `--cluster` when training on the cluster.
- Read descriptions of other arguments.
- 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 γ:

- Why do we use a discount factor γ 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 γ 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
- You can change it by modifying `default_actions.txt` or giving another text file with new actions (use `--action_filename`).
- 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:

- Implement it in `learning.py`.
- Use argument `--use_doubleqlearning` to enable 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

Job Training

You can run experiments above by using script `submit_training_tasks.py`:

- It runs experiemnts written in `carracing_hyperparams.json`.
- The slurm setting is specified in `slurm_settings.json`.
- Use argument `--cluster` to submit jobs in the cluster.
- See help messages before using this script.

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

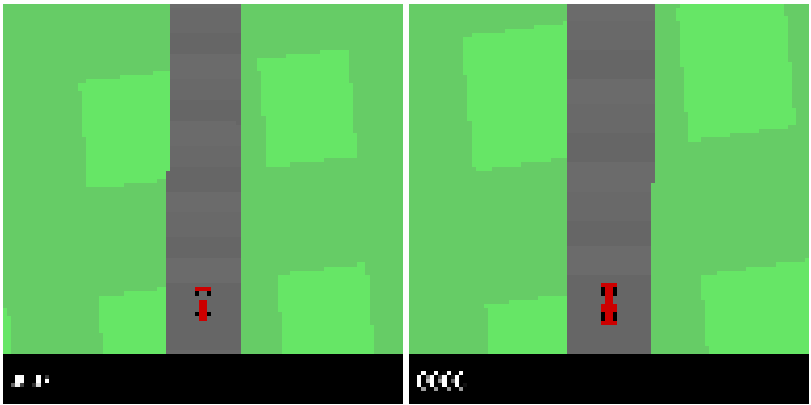
- Training a reinforcement learning takes time (several hours)
- You will need to train and evaluate several agents for this exercise
- Don’t start too late or you will run out of time

Advice

- Refer to the **benchmark** and check how your training environemt performs.
- We recommend you to run experiments on your local machine.
- Plan how to train agents efficiently.
- We will evaluate your best agent on the provided **singularity file** containing the following libraries.
  - Python 3.8.10
  - GYM 0.21.0
  - CUDA 11.3

Version Difference

There are some differences in terms of car shape, box size, and numbers (left: V.0.10.8, right: V.0.21.0).



Questions?