Autonomous Vision Group
Prof. Dr.-Ing. Andreas Geiger

Eberhard Karls
Universität
Tübingen

Self-Driving Cars

Ex. 3 – Reinforcement Learning

Deadline: Tue 21. December 2021; 8pm

For this exercise you need to submit a **.zip** folder containing your best performing model as a `agent.t7` file, the filled out submission form `submission.txt`, all `.py` code files and the file `best_actions.txt` with actions associated with the best trained model. Please use the provided code templates for these exercises. Submissions with changes to the gym environment or additionally installed packages will not be considered. You may choose discrete or continuous actions as output of your agent. You must provide a single model, i.e., ensembles are not allowed. Your method must be a (double) deep Q-learning method, i.e., you are not allowed to use any other RL or IL methods or handcrafted (i.e., PID-) controllers based on waypoint predictions. You may use (prioritized) replay buffers.
Comment your code clearly, use docstrings and self-explanatory variable names and structure your code well. Give a description of what you implement in the text file. For the challenge we only evaluate your trained model. But we also provide some questions and tasks to deepen your understanding of the topic.

Note of caution: Training a reinforcement learning agent takes time. As this exercise requires you to train several agents, it is important to start working on this exercise sheet early on.

3.1 Base Implementation

The aim of this exercise is to implement a Deep Q-Learning agent for the CarRacing environment.

a) Deep Q-Network: Implement a Deep Q-Network and its forward pass in the `DQN` class in `model.py`. Your network should take a single frame as input. In addition, you may again utilize the `extract_sensor_values` function.

- **i)** Would it be a problem if we only trained our network on a crop of the game pixels, which would not include the bottom status bar and would not use the extracted sensor values as an additional input to the network? *Hint: Think about the Markov assumption.*
- **ii)** Why do we not need to use a stack of frames as input to our network as proposed to play Atari games in the original Deep Q-Learning papers [1][2]?

b) Deep Q-Learning: Implement the functions `perform_qlearning_step` and `update_target_net` in `learning.py`. The first should run a single Deep Q-Learning update step while the second should update the target network with the weights of the policy network. *Hint: Make sure to clip your gradients* [1].

- **i)** Why do we utilize fixed targets with a separate policy and target network?
- **ii)** Why do we sample training data from a replay memory instead of using a batch of past consecutive frames?

c) Action selection: In `action.py`, implement the functions `select_exploratory_action` and `select_greedy_action` which select an action according to an exploratory ϵ-greedy strategy or a greedy strategy (i.e. the action which maximizes the Q-values of your policy network).

- **i)** Why do we need to balance exploration and exploitation in a reinforcement learning agent and how does the ϵ-greedy algorithm accomplish this?

d) Training: Train a Deep Q-Learning agent using the `train_racing.py` file with the provided default parameters. Describe your observations of the training process. In particular, show the generated loss and reward curves and describe how they develop over the course of training. Some points of interest to describe should be: How quickly is the agent able to consistently achieve positive rewards? What is the relationship between the ϵ-greedy exploration schedule and the development of the cumulative reward which the agent achieves over time? How does the loss curve compare to the loss curve that you would expect to see on a standard supervised learning problem?

e) Evaluation: Evaluate the trained Deep Q-Learning agent by running the `evaluate_racing.py` script. Observe the performance of the agent by running the script on your local machine. Where does the agent do well and where does it struggle? How does its performance compare to the imitation learning agent you have trained for Exercise 1? Discuss possible reasons for the observed improvement/decline in performance compared to your imitation learning agent from Exercise 1.

Important: Make sure you have a working baseline implementation, in which the trained agent is often able to achieve positive scores as well as take some corners before moving on to work on Section 3.2.

3.2 Further Investigations and Extensions

Please note that all further investigations and extensions should be made w.r.t. the baseline agent which you have implemented in the previous section. Except for your best solution in part **e)**, you should not combine subsequent extensions but investigate the effect of each aspect separately on its own.

a) Discount factor γ: Investigate the influence of the discount factor γ. Have a look at reward curves that demonstrate the effect of an increase/decrease of γ from its default of 0.99 on your agent.

- **i)** Why do we typically use a discount factor in reinforcement learning problems and in which cases would it be a problem not to use a discount factor (i.e. $\gamma = 1$)?

b) Action repeat parameter: What is the reasoning behind the use of an `action_repeat` parameter? By default, this value is set to 4. What is the effect on the training progress (look at your training plots) and your evaluation performance if this parameter is increased or decreased? Discuss and interpret your findings.

- **i)** Why might it be helpful to repeat each action several times?

c) Action space: By default, the agent uses a 4-dimensional action set of left-turn, right-turn, brake and acceleration (see `get_action_set` function in `action.py`). Investigate the addition of a null action ([0,0,0]) as well as more fine-grained turning, braking or acceleration actions. What is the effect on the agent's driving style as well as its evaluation score? Which additional actions lead to an improvement in the agent's performance and which do not?

- **i)** Why might it not always be helpful to increase an agent's action space?
- **ii)** In general, why are Deep Q-Networks limited to a discrete set of actions and what solutions exist to overcome this limitation?

d) Double Q-learning: One problem with the standard Deep Q-Learning approach is an overestimation of Q-values. A proposed solution to this problem is the double Q-learning algorithm [3]. Read this paper, implement the double Q-learning algorithm and evaluate its effect on the training and evaluation performance of your agent.

- **i)** What is the reason for the overestimation of Q-values in standard DQN?
- **ii)** How does double Q-learning solve this issue?

e) Best solution: Finally, putting together your previous findings as well as any more ideas you might have, fine-tune and evaluate your best-performing agent. How is this agent constructed and trained? In which aspects has its performance improved over your baseline agent from Section 3.1 and where does it still exhibit sub-optimal behavior? Save the weights of your best performing agent as `agent.t7` and include it in your submission.

3.3 Competition

With each coding exercise sheet you are welcome to participate in our competition! The winners for each of the 3 exercise sessions will be given the opportunity to present their approach in the very last lecture. The evaluation works as follows: the models are tested on a set of validation-tracks. For each track, the reward after 600 frames is used as performance measure and the mean reward from all validation tracks then forms the overall score. For the final ranking, we will run that evaluation on a set of secret tracks for every submission. The reward is -0.1 every frame and +1000/N for every track tile visited, where N is the total number of tiles in track. Good luck!

3.4 References

[1] https://storage.googleapis.com/deepmind-media/dqn/DQNNaturePaper.pdf
[2] https://arxiv.org/pdf/1312.5602.pdf
[3] https://arxiv.org/pdf/1509.06461.pdf