Deep Learning Lab – Assignment 4

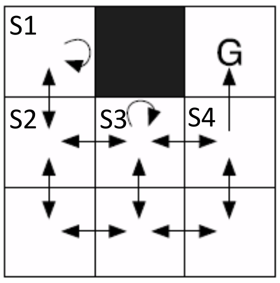
Theresa Eimer, Maryna Kapitonova, Hakan Yilmaz

Introduction

The goal of this assignment is the investigation and the implementation of a deep Q-Learning algorithm applied to a grid-world environment similar to that in the third assignment. Before the actual implementation and to strengthen the theoretical background of the Q-learning algorithm, we were challenged to answer the questions in the first part of the exercise sheet.

Q-Learning (on paper)

Prerequisites:



- Deterministic state transitions and absorbing goal state G (agent can never transition out of it)
- Transitions into the wall are not possible, attempts result in same position as before
- All transitions have immediate reward -1, only transitions within the goal-state have reward 0
- Discounting factor γ = 0.5

1.1:

Update rule of Q-Learning: $q(i,u) = q(i,u) + \alpha\big(r(i,u) + \gamma \cdot max_u q(j,u) - q(i,u)\big)$

Transitions to the goal state can be handled just like other "normal" transitions, they can additionally be rewarded (e.g. +5).

In order to make the goal state absorbing, all transitions within the goal state leave the agent where it is.

1.2:

1. $$\begin{aligned} q(S1,down) &= q(S1,down) + \alpha\big(r(S1,down) + \gamma \cdot max_u q(S2,u) - q(S1,down)\big) \\ &= 0 + 1(-1 + 0.5 \cdot 0 - 0) = -1 \end{aligned}$$

2. $$\begin{aligned} q(S2,right) &= q(S2,right) + \alpha\big(r(S2,right) + \gamma \cdot max_u q(S3,u) - q(S2,right)\big) \\ &= 0 + 1(-1 + 0.5 \cdot 0 - 0) = -1 \end{aligned}$$

3. $$\begin{aligned} q(S3,up) &= q(S3,up) + \alpha\big(r(S3,up) + \gamma \cdot max_u q(S3,u) - q(S3,up)\big) \\ &= 0 + 1(-1 + 0.5 \cdot 0 - 0) = -1 \end{aligned}$$

4. $$\begin{aligned} q(S3,right) &= q(S3,right) + \alpha\big(r(S3,right) + \gamma \cdot max_u q(S4,u) - q(S3,right)\big) \\ &= 0 + 1(-1 + 0.5 \cdot 0 - 0) = -1 \end{aligned}$$

5. $$\begin{aligned} q(S4,up) &= q(S4,up) + \alpha\big(r(S4,up) + \gamma \cdot max_u q(G,u) - q(S4,up)\big) \\ &= 0 + 1(-1 + 0.5 \cdot 0 - 0) = -1 \end{aligned}$$

All Q-values of state-action pairs that were not visited in this scenario remain unchanged and are therefore 0.

Deep Q-Learning (implementation)

Introduction

Different from Assignment 3, here, training data are not generated by an optimal agent. Instead, the agent is supposed to learn while interacting with the grid environment. Visited states, taken actions, resulting states and rewards (all referred to as "experience") are stored in a table. Data batches from this table (experience replay) are used for training using Q-learning with fixed Q-targets. The loss function to be optimized is

$$L_i(w_i) = E_{s,a,r,s'\sim D_i}\left[\left(r + \gamma \cdot max_{a'} Q(s', a', w_i) - Q(s, a, w_i)\right)^2\right]$$

Where $Q(s, a, w_i)$ denotes the underlying Q-function approximation using a state s, an action a and w_i, the weight parameters of the underlying function approximator at time step i.

In order to precisely be able to approximate the action-value function Q, a convolutional neural network (CNN) is implemented in Tensorflow with Python3 frontend. The final network architecture is as follows:

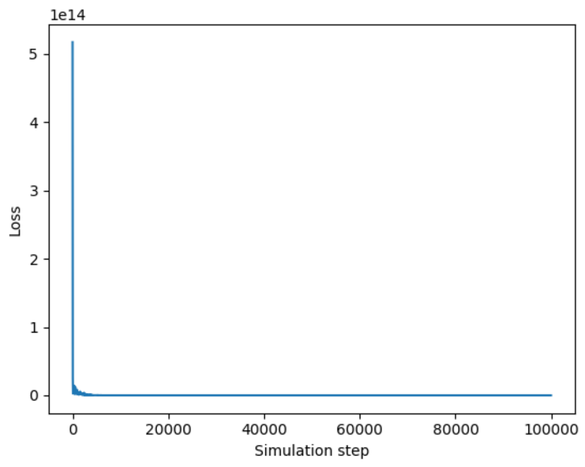
Convolutional Layer (3x3x64), ReLU → Max-Pooling (2x2, stride 2) → Convolutional Layer (3x3x64), ReLU → Max-Pooling (2x2, stride 2) → Fully connected layer (128 units) → Dropout (0.5 during training, 1 else) → Fully connected layer (5 units).

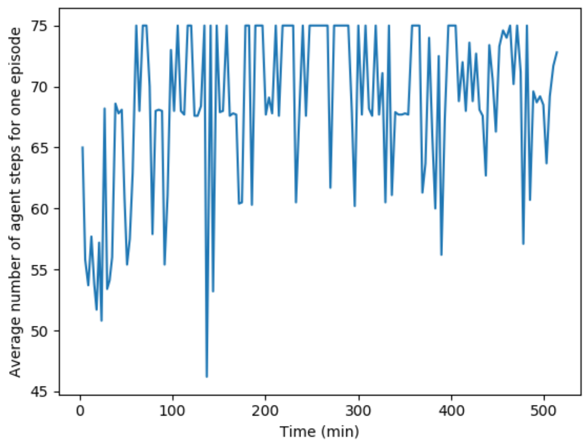
The ADAM algorithm with learning rate 0.0002 was used for optimization, optimal actions were chosen ε-greedily using $\varepsilon_0 = 0.2$ and a decay factor of $\gamma_\varepsilon = 0.9999$.

For all computations, a AMD A10-5750M CPU was used.

Results

Unfortunately, even after a long experimental session where various hyperparameters were tested, no hyperparameter combination seemed to deliver the expected performance.





While the loss steadily decreased, the average number of steps that the agent needed to reach the goal state did not decrease. Instead, this performance marker constantly zig-zagged around about 67 steps – a rather bad value with regards to the early stop value of 75.

These observations imply that, either the implemented network is extremely sensitive to hyperparameter tuning and needs more sophisticated methods to find an optimal hyperparameter set, or there is a bug in the code that could not be identified so far. In either way, both possibilities are going to be investigated. Additionally, also in the frame of the next and final assignment, more sophisticated methods are used to guarantee a stable increase of performance with respect to training time.

24.01.2018