Exercise 4: Visual Planning with Deep Q-Learning

Badhreesh M Rao, David-Elias Künstle

22/01/2018

1 Q-Learning

1.1 Update Rule

Given a transition from state i to state j, using action u and observing immediate reward *r(i,u)*, the update rule for Q-Learning would be:

$$Q(i,u) = Q(i,u) + \alpha(r(i,u) + \gamma(\max_u Q(j,u) - Q(i,u))$$

where α is the learning rate and γ is the discount factor. To handle transitions to or within the goal state, one can set a loop with maximum reward(in this case, 0)for the goal state, so that the agent remains there forever.

1.2 Grid World

The zero-initialized Q-function is represented as a zero matrix of size *(9 × 4)*, where 9 is the number of states(in this case, the cells in the grid world) and 4 is the number of possible actions the agent can take. The order of actions in the columns will be up, down, left, right. The grid world has been named columnwise from A-I, such that the goal state will be named as G. After following the episode, the following Q-values will be updated in the matrix:

$$Q(A, down), Q(B, right), Q(E, up), Q(E, right), Q(H, up)$$

The improved Q matrix after this initial episode will be:

$$Q(s,a) = \begin{bmatrix} 0 & \boxed{\text{-1}} & 0 & 0 \\ 0 & 0 & 0 & \boxed{\text{-1}} \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \boxed{\text{-1}} & 0 & 0 & \boxed{\text{-1}} \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \boxed{0} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

The boxed values are the Q-values that were updated during this episode.

2 Deep Q-Learning

Applications of Q-learning beyond a toy example like in subsection 1.2 is not possible using a lookup table representing $Q(s,a)$. Due to the course of dimensionality, the table becomes infeasible large for bigger state-action-space.

A major development is, representing $Q(s,a)$ in a compressed, continuous format by using a *function approximator*. In the report of *Exercise 3*, we showed, that a supervised, convolutional deep network can map a state history in a maze task to probabilities, recommending the best action to choose for reaching a target state.

Here we present an online learning solution to the maze task without pregenerated training data. The agent is learning of rewards remembered from exploration and exploitation of the environment using *Q-learning* (subsection 1.1). A convolutional network similar to the one in *Exercise 3* is used, here representing the state-action-value $Q(s,a)\ \forall a$ instead of the action-probability $P(a|s)\ \forall a$.

2.1 Maze Task

Similar to *Exercise 3*, an agent is spawned at a random position in a discrete 2d maze. The agent can change the position by choosing *up, down, left, right, stay* actions according to a policy and a history of states (local maze cutouts). For each movement the agent receives a small punishment (negative reward, $r = -0.04$), if it tries to move on a wall it receives a big punishment ($r = -1$) and stays. Only for moving to the fix positioned goal state in the middle of the maze, the agent receives positive reward ($r = 1$). From time to time, some of the past transition information (*state, action, next state, reward*) are used, to improve the agent's policy.

Algorithm 1 Agent learns finding a target in a maze while exploration and exploitation (Pseudocode).

```
for all $t \in$ steps do
    action$_t \leftarrow$ agent.policy(state$_{t-n,\cdots,t}$)
    state$_{t+1}$, reward $\leftarrow$ maze.act(action$_t$)
    memory.store(state$_t$, action$_t$, state$_{t+1}$, reward)
    if $t \in$ learnsteps then
        statebatch, actionbatch, nextstatebatch, rewardbatch $\leftarrow$ memory.samplebatch()
        agent.learn(statebatch, actionbatch, nextstatebatch, rewardbatch)
    if maze.episodesteps $>$ earlystop or state$_t \in$ terminal then
        maze.newepisode()
```

2.2 Agent

The agent we use for solving the maze task should learn from scratch to reach the target as fast as possible by exploration of the environment. Therefore the agent builds up an estimation of the reward until reaching the target for being in a state and choosing an action (*Q-learning*, $Q(s, a)$) by the perceived reward of a state transition.

The Q-function is represented as a convolutional network (Table 1).

Convolution Layer	(8 filters, size 64)
Convolution Layer	(16 filters, size 32)
Convolution Layer	(16 filters, size 16)
Convolution Layer	(32 filters, size 4)
Dense Layer	(5 outputs, linear activation)

Table 1: Network architecture. A convolution layer here includes ReLu activation function and max pooling (size 2, stride 2).

Policy

Each step in the environment, the agent chooses the next action according to an *ϵ-greedy policy.* Therefore by a probability of ϵ a random action is used to explore new transitions in the environment, else we greedily follow the Q-value by $\arg\max_a Q(s, a)$.

At the beginning of training where we assume Q to be a bad estimation of the cost, we always explore ($\epsilon = 1$). To focus training to useful paths, we want to reduce exploration while the Q-value estimation improves. Therefore we linearly reduce ϵ to 0.1 after the first half of training steps.

Training

The estimation of the Q-function can be improved similar to supervised training of an network, using back-propagation and a *sum-of-square-differences loss.* However the difference is not between a $Q(s, a)$ and a true $Q^*(s, a)$. Since we don't know $Q^*(s, a)$, we use the observed transition reward to get at least a slightly better estimation of $Q(s, a)$ like in the Q-learning formula in subsection 1.1. Therefore we could also call the loss section 2.2 *sum-of-square-temporal-differences loss.*

$$L = \sum_i (Q(s_i, a_i) - (r + \gamma \cdot \max_{a'} Q(s'_i, a'_i)))^2$$

2.3 Evaluation

While training without any training data, we have no direct measurement of the agents performance. The loss was the only indication of the agent's progress. A loss close to zero indicates, that the $Q(s, a)$ value converged close to the true $Q^*(s, a)$. In the beginning, the loss increased to several thousand per batch. Still, after several thousand environment steps (over one thousand training steps) the loss came down to two-digits and finally smaller than one.

Finally we test the trained agent in the maze task, with a low exploration rate ($\epsilon = 0.1$). Out of 100 episodes, the agent reached the target 96 times before an early stop (75 steps). In visual inspection of the episodes we see, that if the agent gets stuck, it usually gets stuck in the lower left corner. We assume that the Q-function is not close to Q^* for these states, because they weren't trained enough.