Deep Learning Course 2017
Autonomous Intelligent Systems
University Freiburg

Ilia Dobrusin
Archit Basal
Due Date: 22.01.2017

Deep Learning Lab Course 2017 - Exercise 4

This assignment consists of two parts: manually simulating Q-learning and implementing Deep Q-learning using Tensorflow.

1 Manual Q-Learning

Q-function update rule

Update rule for the Q function for any state-action pair:

$$Q(i, u) = (1 - \gamma) * Q(i, u) + \gamma * (r(i, u) + \underset{u'}{\operatorname{argmin}} Q(j, u'))$$

where i is the initial state, u is the action taken in i, j the final state reached from i by action u, and r is the reward function.

The goal state can be made an absorbing state by ensuring that all actions leading away from the goal state return the agent to the goal state, i.e. for all u available in g, where g is the goal state, the transition function T is defined as $T(g, u) = g$. Another way to deal with this is to introduce an action to *stay* in the current state, and ensure that all actions except the *stay* action have a large penalty whereas the *stay* action has a reward of 0. Although it must be noted that the *stay* action can be considered a *noop* that can lead to other complications unless properly handled, for example, the *stay* action should result in a small penalty in all non-goal states, and not be 0.

Moving in maze

Assuming that the cell the agent starts in is represented by $(0, 0)$, such that the goal state is the cell $(2, 0)$, the wall is the cell $(1, 0)$ and the cell immediately below $(0, 0)$ is $(0, 1)$, the following sequence of state-action pairs would be updated:

$$< (0, 0), D >, < (0, 1), R >, < (1, 1), U >, < (1, 1), R >, < (2, 1), U >$$

where the actions available to the agent are to move up, down, left or right, represented with U, D, L and R respectively.

Since the learning rate is 1, the old values of the Q-function will be completely ignore during the update step and only the sum of the reward and the Q-value of the next state and it's greedily-selected best action will be retained. Assuming the Q-function was initialized with zeroes, the updated Q-values will be:

$$\begin{gathered} Q((0, 0), D) = -1 \\ Q((0, 1), R) = -1 \\ Q((1, 1), U) = -1 \\ Q((1, 1), R) = -1 \\ Q((1, 2), U) = -1 \end{gathered}$$

and Q(i, u) = 0 for all other valid combinations of i and u.

2 Deep Q-learning

The second part of the assignment consists of implementing Deep Q-learning using a neural network for predicting the Q-value for a given state. The network is similar to the approach of the visual planner from assignment 3, although the architecture is simplified, with fewer layers and no max-pooling layers. Figure 4 shows the architecture of the network.

Setup and Hyperparameters

The agent moved in the environment at random for 1000 steps in order to build a up a transition table prior to training. In addition, the agent performs random movements with a certain probability, which decayed after each training step, resulting in less random movements of the agent as the training progresses. The network was trained with the hyperparameters shown in table 1.

Parameter	**Value**
Discount factor	0.99
Max. exploration probability	0.7
Min. exploration probability	0.2
Exploration probability decay	0.99999
	(steps)
Exploration steps prior to training	1,000
Total training steps	20,000

Table 1: Hyperparameters used during training

2.1 Issues during implementation

At first, we implemented DQN using Keras, which resulted in suboptimally trained model where the agent would reach state next to goal state without moving to the goal state. In the end, we implemented the network using Tensorflow, which lead to an agent who could reach the goal. The training process in itself is instable in the sence that small changes in the architecture lead to significantly worse results, were the calculated Q-loss explodes to values up to a range of 10^{20}, while for a successful run the loss is in the range between 0 and 20.

Our first approach consisted of a target network as described by Wang et al [1], but due to changing from Keras to Tensorflow we did not reimplement it the final architecture. In the current state, the architecture is not yielding good results on every run.

Results

The network was trained using hyperparameters in table 1. Figure 2 shows the accumulated reward during training. As the accumulated reward does not increase over time, it is possible that the model did not yet converge. We would expect the curve to rise after the model is trained sufficiently, although overfitting could occur. In figure 1 the loss during training of the first 1000 episodes after the exploration phase is displayed. Figure 3 shows the resulting accumulated rewards during testing, where the agent performed 10000 steps, reaching the goal in 96,71% episodes.

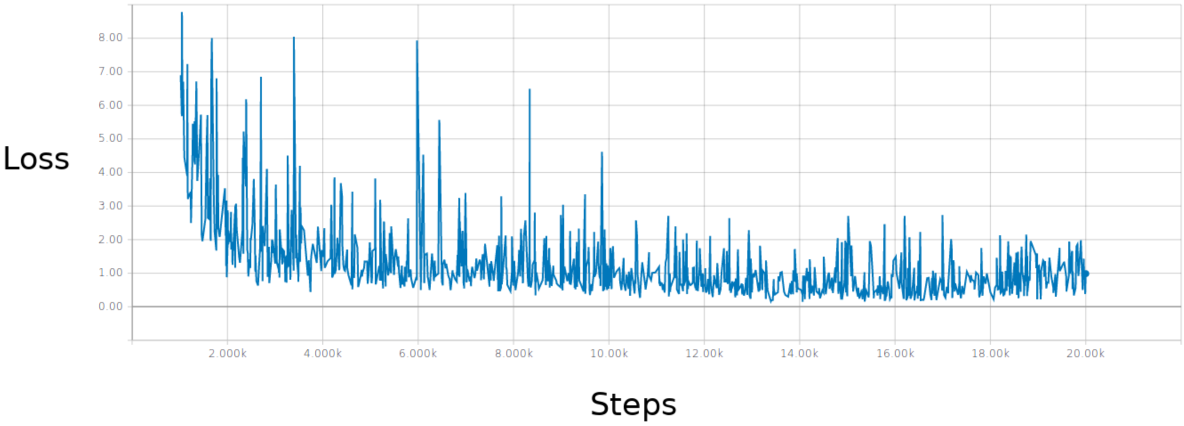

Figure 1: Loss after each training step over the training time.

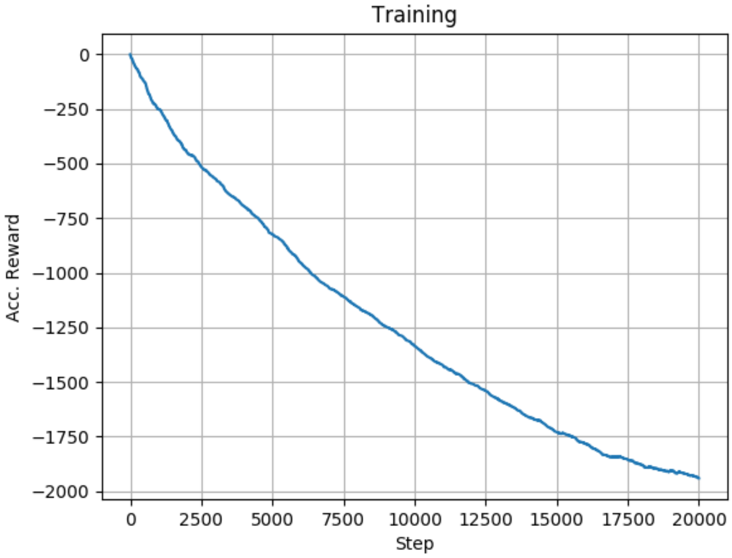

Figure 2: Accumulated reward during training

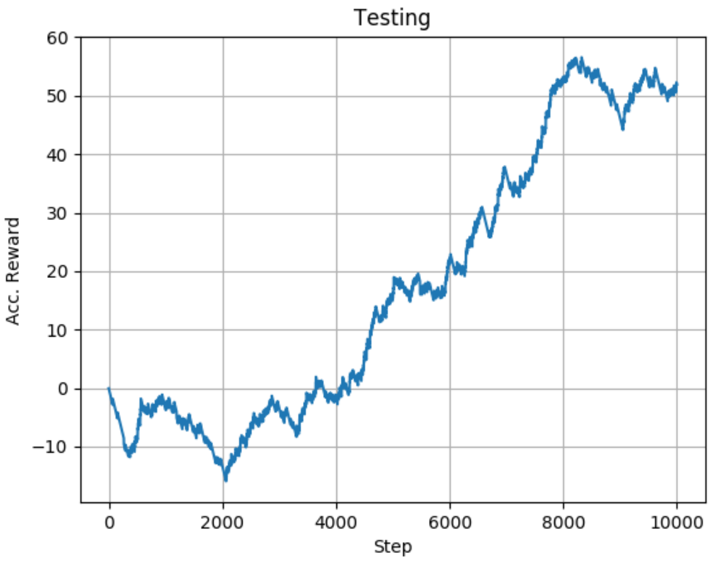

Figure 3: Accumulated reward during testing

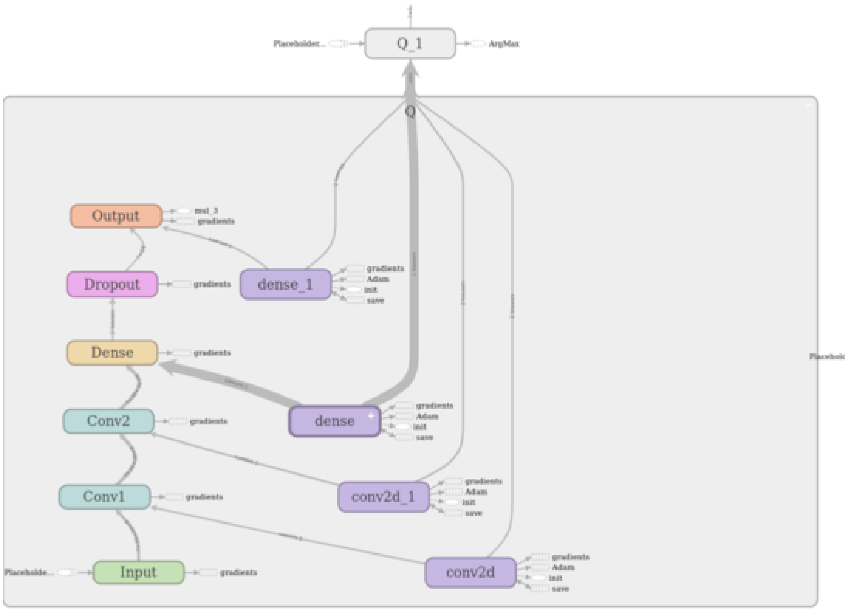

Figure 4: The architecture of the neural network used in this exercise

References

[1] Z. Wang, N. de Freitas, and M. Lanctot. Dueling network architectures for deep reinforcement learning. *CoRR*, abs/1511.06581, 2015.