Machine Learning in Data Science Project Week 14

Eva Aßmann, Paul Vogler, Katrin Böhler

Januar 2020

1 RL for the Taxi Problem

Reinforcement learning is a reward-based learning method based on the interaction with an environment. The basic principle comprises an agent taking actions within an environment in order to maximize the cumulative reward received by the environment for the performed actions. Reinforcement learning does not require labelled data or explicit performance correction but focuses on the efficient balance between exploration of the unexplored environment and exploitation of the so far gained knowledge.
The Taxi problem involves solving the Taxi v3 environment from the OpenAI gym library. The environment consists of 500 states that can be described as walls and locations on a 2 dimensional plane. Each location is surrounded by either passable or non-passable boundaries/walls (Fig. 1). There are four predefined locations Y, G, R, B, and on each reset of the environment one of these four is marked blue as a passenger pick-up location, as well as one being marked purple as the passenger drop-off location. The position of the taxi is marked as a yellow square without having picked up the passenger and a green square with the passenger on board.

Figure 1: The environment of the taxi problem.

The goal for the taxi is to pick up the passenger and drop him off at the drop-off location. For this, the agent has 6 actions available: move south, move north, move east, move west, pick up a passenger or drop off a passenger. The rewards for each action are: -1 for each action the agent takes, -10 for every incorrect pick-up or drop-off action and +20 for successfully picking up a passenger and dropping him off at the correct position. An episode ends when the 20 points for correct delivery are achieved.
Three reinforcement learning strategies were implemented to solve the taxi problem:

1.1 The random search strategy

In random search, each action the agent will perform next is sampled uniformly at random from the action space. This results in a policy with a uniform score of 1/n_actions for each state and action. After training

an agent using the random reinforcement learning strategy, the obtained solution was evaluated by letting the trained agent act in the environment until solving the taxi problem correctly 1.000 times. On average, it took the agent 2559 actions to successfully perform pick-up and drop-off (Fig. 2).

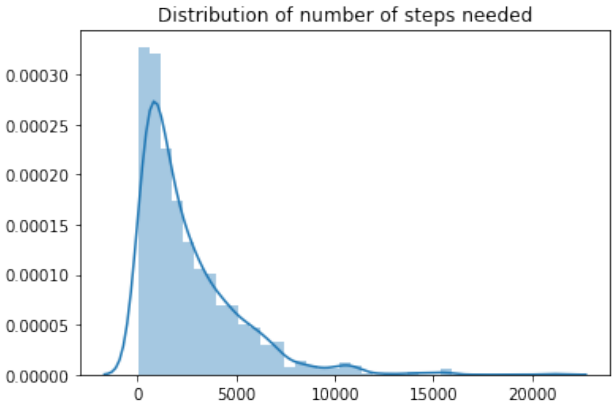

Figure 2: Histogram of the number of steps the agent took with a random search strategy. The average number of steps was 2559.

1.2 The value iteration strategy

When employing value iteration, the optimal policy is found by iteratively improving and updating the state values until they converge. For each state, the value is updated if the agent performed an action that yielded a higher state value than the current one. This is done until the value improvement over all states converges to a minimum threshold. For the learning experiment, a state-value table was initialized with zeros. Training was performed with value optimization threshold θ=0.0001 and discount factor γ=0.99. The discount factor defines the weight future rewards are included into the overall reward.
After learning was finished, the optimal policy was then extracted by using the estimated optimal value function over all states and actions. The solution obtained by the value iteration strategy was evaluated by letting the trained agent act in the environment until solving the taxi problem correctly 1.000 times. On average, it took the agent 13 actions to perform pick-up and drop-off successfully (Fig. 3).

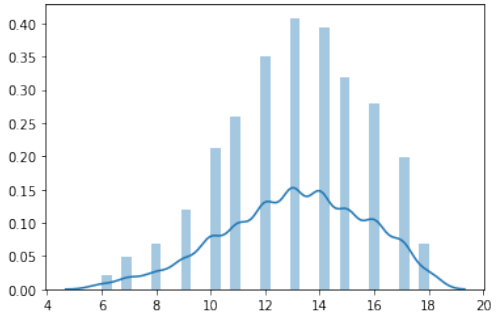

Figure 3: Histogram of the number of steps the agent took using the value iteration strategy. The average number of steps was 13.

1.3 The Q-learning strategy

In contrast to the value iteration strategy, where it is assumed that the agent has some prior knowledge about the environment, its state-transition-matrix and reward function, Q-learning does not require the agent to

have this information. As a model-free method, Q-learning derives an optimal policy from directly interacting with the environment. The main strategy is to approximate the Q-function that assigns a state-actions pair the total reward received by taking a specific action from a specific state. Approximation is done similarly to value iteration, by storing the Q-values the agent received for its actions in a Q-table and iteratively updating and maximizing them. In order to balance exploration and exploitation, the agent chooses its next action with probability ϵ at random or selects it based on the current estimates of Q-values with probability 1-ϵ. For the learning experiment, a Q-table was initialized with zero values. Learning was performed over 100.000 epochs with ϵ=0.1, a learning rate *alpha* of 0.2 and a discount rate γ of 0.95. The learning rate defines the extent to which the Q-values are being updated in every iteration.
After the Q-learning process was finished, an optimal policy was then extracted from the estimated Q-table. The solution obtained by the Q-learning strategy was evaluated by letting the trained agent act in the environment until solving the taxi problem correctly 1.000 times. On average, it took the agent 13 actions to perform pick-up and drop-off successfully (Fig. 4).

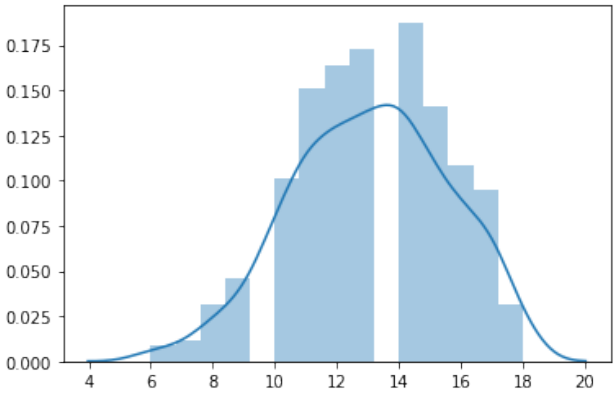

Figure 4: Histogram over the number of steps the agent took with a Q-learning iterated policy. The average number of steps was 13.

1.4 Comparison and discussion of the results

The random search was highly inefficient for this problem, making the agent requiring almost 200 times as many steps to solve the taxi problem as with the other two strategies. Both the value iteration and Q-learning strategy converged towards optimal policies that took 13 actions on average to solve the taxi problem.

2 Q Learning

2.1 The Space Invaders Problem

The environment for this reinforcement learning problem is the Atari2600 game Space Invaders (using the OpenAI gym SpaceInvaders-v0 environment). The environment consists of an output image frame of 210*160 RGB pixels. The action space consist of the action space noop (nothing operation), fire, right, left as well as rightfire and leftfire that combine the fire + left/right operation. The environment consists of enemies in a grid of 6x6, that move from left to right and slowly downwards (Fig. 5). The enemies can shoot downwards in a straight line and the agent can shoot straight upwards. If the enemies get hit they disappear and grant points in the order of 5*row-number. If the agent gets hit, he loses one of 3 lives. When the agent has been hit 3 times, the episode ends. The goal of the Space Invaders problem is to score the maximum point score within three lives.

Figure 5: The Space Invaders environment.

2.2 Deep Q-learning solution

At first, the image frame from the environment needs to be preprocessed in order to extract states. The frame is converted to a greyscale first, because the colour information is not used here. Then, the bottom part of the frame is cropped and the image is resized to a 110x84x1 frame.
Additionally, 4 frames are stacked in a first-in/first-out queue, that initially gets filled 4 times with the first frame. Every new step, a new frame gets queued and the frame that was stacked in the queue the longest gets dequeued, such that the resulting stack is of size 110x84x4.
The agent is then trained using Q-learning like described above in section 2, but instead of using a Q-table, a neural network is used. The 110x84x4 dimensional input is handed through 3 convolutional layers and then flattened. The next two layers are fully connected, such that in the end, one Q value is returned for each action. Learning was performed with learning rate α=0.00025, discount rate γ=0.9 and 100 training episodes. In this approach, the exploration parameter ϵ is not a constant parameter, but starts at 1 and exponentially decays at rate 0.00001 to a minimum value of 0.01.

2.3 Results

After training the agent over 60 episodes, the exploration probability was still at 0.67. The final point score varied between training runs and was at 110 for the last run.

```
Episode: 56 Total reward: 60.0 Explore P: 0.6803 Training Loss 1.8980
Episode: 57 Total reward: 75.0 Explore P: 0.6779 Training Loss 0.3959
Episode: 58 Total reward: 40.0 Explore P: 0.6752 Training Loss 627.4388
Episode: 59 Total reward: 110.0 Explore P: 0.6707 Training Loss 0.0435

INFO:tensorflow:Restoring parameters from ./models/model.ckpt
**************************************************
EPISODE  0
Score 20.0
```

Figure 6: Performance in training and testing: point score after 60 episodes with random policy (top) and after one episode without random policy (bottom)

The trained agent was tested using the tensorflow model checkpoints from the training run. For the test, one episode was run with no random choices allowed. This resulted in a point score of 20 after 757 actions, which is rather low compared to some of the training point scores (see Fig. 6).

2.4 Discussion

The Deep Q-learning network seems to still perform significantly worse without the random actions and will probably need a lot more training iterations, before the actions suggested by the Q-network yield a point score as high as the network with random actions yielded, or even higher.