CS 4080-5080 - Reinforcement Learning
Homework 3 Report

Carlos E. L. P. X. Torres
University of Colorado at Colorado Springs
1420 Austin Bluffs Pkwy.
Colorado Springs, Colorado 80918
clopespi@uccs.edu

Abstract

Homework 3 Report for the class CS 4080-5080 - Reinforcement Learning due to 12/01/2021. On this assignment, it is implemented a reinforcement learning algorithm to teach an agent how to navigate in a 6x7 simplified parking lot environment from the entrance to any of the parking spots (P1 to P8). The algorithm used is Q-learning implemented with a neural network (DQN) to approximate the Q value. A simulation is presented showing the implemented algorithms in action and also showing a chart of the learning progress. The results obtained were slightly better than the previous homework, finding the policies with the highest possible rewards. The architecture of type 3 was responsible for the best results.

1 Introduction

Homework 3 Report for the class CS 4080-5080 - Reinforcement Learning. On this assignment, it is implemented a reinforcement learning algorithm to teach an agent how to navigate in a 6x7 simplified parking lot environment from the entrance to any of the parking spots (P1 to P8). The algorithm used is Q-learning implemented with a neural network to approximate the Q value. No reinforcement learning libraries are used, only some common libraries, like for numeric processing (numpy), math, random, and time. The neural network is implemented using TensorFlow Keras.

In section 2, it is presented a model of the problem, describing the various components required for the agent and the environment (states, rewards, actions, barriers, edges avoiding handling).

In section 3, it is presented the neural network architectures created to perform experiments and the structure of the replay memory used.

In section 4, it is presented how the algorithm was implemented, and a simulation was created to illustrate it.

In section 5, the results are presented with a chart showing the difference of the rewards obtained using the 3 different neural network architectures.

In conclusion, it is discussed the final considerations, problems faced, and how they were overcome during the work.

2 Problem Model

To model the 6x7 parking lot environment (Figure 1) as a reinforcement learning problem, we should formulate it in terms of a Markov Decision Process (MDP), consisting of a tuple $\{S, A, P, R, \gamma\}$ (Sutton and Barto 2018), where:

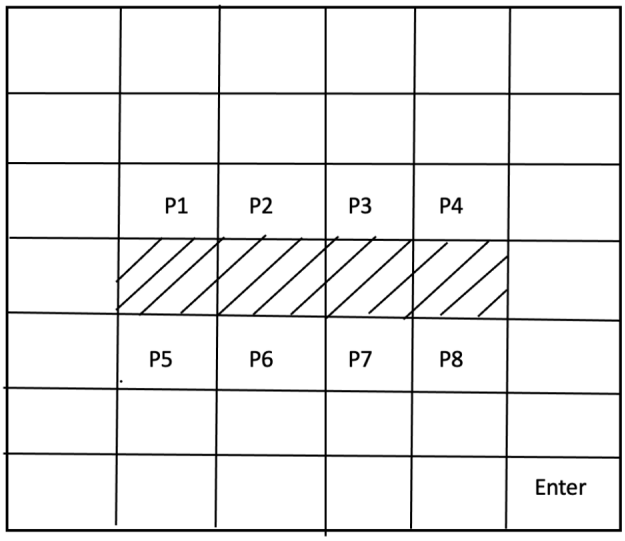

Figure 1: The 6x7 simplified parking lot environment

- *S* - The state space: a set of 42 states represented by a 6x7 2D matrix of cells (x, y) identifying each possible state inside the parking lot. The start state is represented as (5, 6) and the available parking spots states are P1 = (1, 2), P2 = (2, 2), P3 = (3, 2), P4 = (4, 2), P5 = (1, 4), P6 = (2, 4), P7 = (3, 4), P8 = (4, 4). But only one of the parking spots states are set as the target state each time.
- *A* - The action space: a set of 4 possible actions $\{N, S, E, W\}$ that the agent can take in each state inside the parking lot, each one being a directions (North, South, East, West). In this problem, not all actions are allowed in all states, due to the barrier states and the boundaries of the parking lot.
- *P* - The transaction probability function: defines how the agent can change states inside the parking lot. The function will serve to define the allowed actions inside each state (or cell) of the parking lot, associating probabilities to each state-action pair, and 0 (zero) for the ones that are not allowed. This will also serve to define the barriers in-

side the parking lot, by assigning 0 to an action that will lead to a barrier state. See Table 1 for the representation of this function.

State / Action	N	S	E	W
s_1 (0,0)	0	0.5	0.5	0
s_2 (1,0)	0	0.33	0.33	0.33
s_3 (2,0)	0	0.33	0.33	0.33
s_4 (3,0)	0	0.33	0.33	0.33
s_5 (4,0)	0	0.33	0.33	0.33
s_6 (5,0)	0	0.5	0	0.5
s_7 (0,1)	0.33	0.33	0.33	0
s_8 (1,1)	0.25	0.25	0.25	0.25
s_9 (2,1)	0.25	0.25	0.25	0.25
s_{10} (3,1)	0.25	0.25	0.25	0.25
s_{11} (4,1)	0.25	0.25	0.25	0.25
s_{12} (5,1)	0.33	0.33	0	0.33
s_{13} (0,2)	0.33	0.33	0.33	0
s_{14} (1,2)	0	0	0	0
s_{15} (2,2)	0	0	0	0
s_{16} (3,2)	0	0	0	0
s_{17} (4,2)	0	0	0	0
s_{18} (5,2)	0.33	0.33	0	0.33
s_{19} (0,3)	0.5	0.5	0	0
s_{20} (1,3)	0	0	0	0
s_{21} (2,3)	0	0	0	0
s_{22} (3,3)	0	0	0	0
s_{23} (4,3)	0	0	0	0
s_{24} (5,3)	0.5	0.5	0	0
s_{25} (0,4)	0.33	0.33	0.33	0
s_{26} (1,4)	0	0	0	0
s_{27} (2,4)	0	0	0	0
s_{28} (3,4)	0	0	0	0
s_{29} (4,4)	0	0	0	0
s_{30} (5,4)	0.33	0.33	0	0.33
s_{31} (0,5)	0.33	0.33	0.33	0
s_{32} (1,5)	0.25	0.25	0.25	0.25
s_{33} (2,5)	0.25	0.25	0.25	0.25
s_{34} (3,5)	0.25	0.25	0.25	0.25
s_{35} (4,5)	0.25	0.25	0.25	0.25
s_{36} (5,5)	0.33	0.33	0	0.33
s_{37} (0,6)	0.5	0	0.33	0
s_{38} (1,6)	0.33	0	0.33	0.33
s_{39} (2,6)	0.33	0	0.33	0.33
s_{40} (3,6)	0.33	0	0.33	0.33
s_{41} (4,6)	0.33	0	0.33	0.33
s_{42} (5,6)	0.5	0	0	0.5

Table 1: Transition probability function representation

The barrier states themselves (s_{20}, s_{21}, s_{22}, s_{23}) will have a probability of 0 (zero), to keep the integrity of the function. Similarly, the parking spot states (s_{14}, s_{15}, s_{16}, s_{17}, s_{26}, s_{27}, s_{28}, s_{29}) will also be considered with probability 0 (zero) because they can assume target (or terminal) states, once achieved, the agent should not go anywhere else, and the episode ends.

- *R* - The reward signal: represented by a set of rewards from the environment to the agent. For this problem, two types of rewards will be used. For every step in the parking lot, the agent receives a reward of -0.1. And when the agent reaches the goal state (one of the parking spots that were selected as target), a reward of 1 is given. The negative rewards will make the agent prefer a policy with fewer steps to achieve the goal state.
- γ - The discount factor: can assume values $0 \leq \gamma \leq 1$. It increases the interest of the agent in having earlier rewards instead of future or later rewards. For this problem, it will be set initially to $\gamma = 0.99$, and can change during the learning process to see how it affects the algorithm.

2.1 Algorithm

We used Algorithm 1 to implement a DQN, Deep Q-learning with Experience Replay (Mnih et al. 2013), to train our agent. Some modifications were made to adapt the algorithm to our problem.

The algorithm starts on line 1 by initializing the replay memory D and, on line 2, the state-value value function Q with random weights.

On line 3, starts the main loop of the algorithm, for all episodes until the maximum M. In each iteration, it initializes a sequence s_1 (Line 4) and starts the iterations through steps of each episode (Line 5), until the maximum T.

On lines 6 and 7, an action is selected using probability ϵ to choose a random action or the $max_a Q^*(\phi(S_t), a; \theta)$.

On line 8, the chosen action a is taken in the emulator, and it is obtained r_t and observation o_{t+1}.

On line 9, the next state s_{t+1} is set to s_t, a_t, o_{t+1} and ϕ_{t+1} is set to $\phi(s_{t+1})$.

On line 10, the transition is stored in the replay memory D. On lines 11 and 12, a random sample batch is obtained from D to be used to predict y_i and then train the target network with the Q-learning formula, if it is not a terminal state.

Finally, on line 13, the neural network gradient descent step is performed, calculating the loss using the Mean Squared Error function.

3 Neural Network Architectures

Three different neural network architectures were created to perform experiments. All of them are based on the Keras Sequential model. The input dimension of the first layer is what is being passed as input to the neural network. In this problem, a one-dimension array with two numbers representing the current coordinates $[x, y]$ of the agent in the environment. The output dimension of the last layer is what is being provided by the neural network. In this problem is 4, representing the four possible actions that can be performed. The activation function used in the last layer is *softmax*, that converts a vector of values to a probability distribution in range (0, 1) and sum to 1.

Below the three architecture types (1, 2, and 3) are described.

- Type 1 has 3 dense layers with 32, 16 and 4 nodes, respectively.

Algorithm 1: Deep Q-learning with Experience Replay

1: Initialize the replay memory D to capacity N
2: Initialize the state-action value function Q with random weights
3: **for all** $episode = 1 \in M$ **do**
4: Initialize the sequence $s_1 = o_1$ and the preprocessed sequence $\phi_1 = \phi(s_1)$
5: **for all** $t = 1 \in T$ **do**
6: With probability ϵ select a random action a_t,
7: Otherwise select $a_t = max_a Q^*(\phi(S_t), a; \theta)$
8: Execute the action a_t in the emulator and obtain the reward r_t and observation o_{t+1}
9: Set $s_{t+1} = s_t, a_t, o_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
10: Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D
11: Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D
12: Set $y_i = \begin{cases} r_j & \text{terminal } \phi_{j+1} \\ r_j + \gamma max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{non terminal } \phi_{j+1} \end{cases}$
13: Perform a gradient descent step on $(y_i - Q(\phi_j, a_j; \theta))^2$
14: **end for**
15: **end for**

- Type 2 has also 3 dense layers, but with 128, 128 and 4 nodes, respectively.
- Type 3 has 5 dense layers with 256, 128, 64, 32, and 4 nodes, respectively.

On tables 2, 3, and 4 the three model summaries are shown.

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 32)	96
dense_1 (Dense)	(None, 16)	528
dense_2 (Dense)	(None, 4)	68
Total params:	692	
Trainable params:	692	
Non-trainable params:	0	

Table 2: Type 1 model summary.

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 128)	384
dense_1 (Dense)	(None, 128)	16512
dense_2 (Dense)	(None, 4)	516
Total params:	17,412	
Trainable params:	17,412	
Non-trainable params:	0	

Table 3: Type 2 model summary.

3.1 Replay Memory

As mentioned on section 2.1, we are using a replay memory structure to help implement our DQN. A class named *ReplayBuffer* was created to be used by the *DQNAgent* class to interact with the replay memory. This class stores the structure $(state, next\_state, action, reward, terminal)$ in an array of a predetermined size, defined by the replay memory size on Table 5.

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 256)	768
dense_1 (Dense)	(None, 128)	32896
dense_2 (Dense)	(None, 64)	8256
dense_3 (Dense)	(None, 32)	2080
dense_4 (Dense)	(None, 4)	132
Total params:	44,132	
Trainable params:	44,132	
Non-trainable params:	0	

Table 4: Type 3 model summary.

There is a method named *store_transition* used to store the current transition, as shown in the Algorithm 1 on line 10. There is also another method named *sample_buffer* to obtain a random sample batch of transitions, as shown on line 11 of the algorithm.

4 Implementation and Simulation

The algorithms were implemented in Python 3 using native or common libraries like: random, time, sys, a numeric processing library called numpy, and a simulated environment, using a custom developed Open AI gym (Brockman et al. 2016) to model the simplified parking lot environment. For the neural network implementation, TensorFlow Keras was used.

It was implemented a class named *DeepQLearning*, that receives all the parameters for the algorithm, performs the training of the agent, and returns the generated data for the rewards and episodes created during the process for graph plotting purposes.

Another class named *DQNAgent* was created to implement the DQN for the Q-learning algorithm, and it is called by *DeepQLearning* class. *DQNAgent* creates the neural network model, the replay memory, and implements the learning process for the prediction and target networks.

The hyperparameters used to train the agent can be

viewed on Table 5 below. After running the training for the agent, it is generated a graph (Figure 4) with the results to compare the three types of neural network architectures used.

Hyperparameter	Value
Alpha (learning rate)	0.001
Gamma (discount factor)	0.99
Epsilon (exploration rate)	0.8
Epsilon min	0.001
Decay factor	0.067
Replay memory	1,000,000
Batch size	64
Max Number of Steps/Episode	670
Max Number of Episodes	1,000

Table 5: Hyperparameters of the DQN.

4.1 Policies Learned

After running the algorithm several times, the following policies were learned by the agent to take the car to each parking spot (P1 to P8). The parking spots are represented in Figure 2 and the policies for each one can be visualized in Figure 3.

Policy for P1:
$\pi(s_{42}) = N \rightarrow \pi(s_{36}) = N \rightarrow \pi(s_{30}) = N \rightarrow \pi(s_{24}) = N \rightarrow \pi(s_{18}) = N \rightarrow \pi(s_{12}) = W \rightarrow \pi(s_{11}) = W \rightarrow \pi(s_{10}) = W \rightarrow \pi(s_{09}) = W \rightarrow \pi(s_{08}) = S \rightarrow \pi(s_{14})$

Policy for P2:
$\pi(s_{42}) = N \rightarrow \pi(s_{36}) = N \rightarrow \pi(s_{30}) = N \rightarrow \pi(s_{24}) = N \rightarrow \pi(s_{18}) = N \rightarrow \pi(s_{12}) = W \rightarrow \pi(s_{11}) = W \rightarrow \pi(s_{10}) = W \rightarrow \pi(s_{09}) = S \rightarrow \pi(s_{15})$

Policy for P3:
$\pi(s_{42}) = N \rightarrow \pi(s_{36}) = N \rightarrow \pi(s_{30}) = N \rightarrow \pi(s_{24}) = N \rightarrow \pi(s_{18}) = N \rightarrow \pi(s_{12}) = W \rightarrow \pi(s_{11}) = W \rightarrow \pi(s_{10}) = S \rightarrow \pi(s_{16})$

Policy for P4:
$\pi(s_{42}) = N \rightarrow \pi(s_{36}) = N \rightarrow \pi(s_{30}) = N \rightarrow \pi(s_{24}) = N \rightarrow \pi(s_{18}) = W \rightarrow \pi(s_{17})$

Policy for P5:
$\pi(s_{42}) = W \rightarrow \pi(s_{41}) = W \rightarrow \pi(s_{40}) = W \rightarrow \pi(s_{39}) = W \rightarrow \pi(s_{38}) = N \rightarrow \pi(s_{32}) = N \rightarrow \pi(s_{26})$

Policy for P6:
$\pi(s_{42}) = W \rightarrow \pi(s_{41}) = W \rightarrow \pi(s_{40}) = W \rightarrow \pi(s_{39}) = N \rightarrow \pi(s_{33}) = N \rightarrow \pi(s_{27})$

Policy for P7:
$\pi(s_{42}) = W \rightarrow \pi(s_{41}) = W \rightarrow \pi(s_{40}) = N \rightarrow \pi(s_{34}) = N \rightarrow \pi(s_{28})$

Policy for P8:
$\pi(s_{42}) = W \rightarrow \pi(s_{41}) = N \rightarrow \pi(s_{35}) = N \rightarrow \pi(s_{29})$

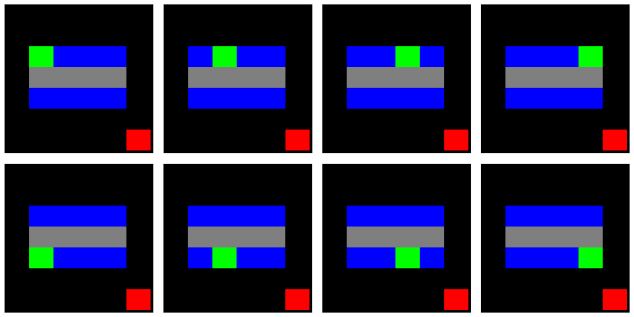

Figure 2: Parking lot spots representation in the simulation. From top left to bottom right, P1 to P8.

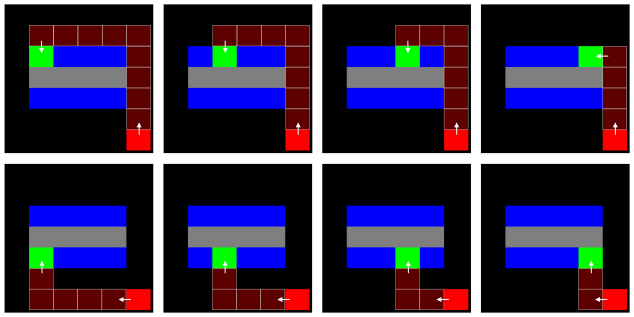

Figure 3: Policies learned for each spot represented in the simulation. From top left to bottom right, P1 to P8.

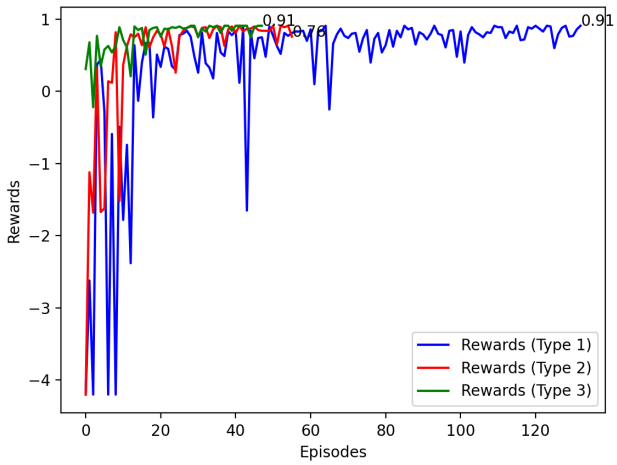

Figure 4: Cumulative rewards results for the three types of DQNs used. The P1 spot is used to illustrate.

5 Results

The algorithm was executed several times and the resulting policies are the best ones that were found. Notice that, for the policies P1 to P4, the algorithm found best to start going always up, accessing the top spots by the right side, which, indeed, generates the best policies, with maximum reward and shorter paths. These results were better than the previous homework, that implemented Q-learning using only a table.

The algorithm converges after a few episodes, depending on the DQN type used (Type 1, 2, or 3). The results can be viewed on Figure 4. Notice that when the algorithm used the DQN Type 3 (green) it has the best results, which means the highest rewards in the fewest episodes. The Type 1 (blue) shows the worst results, taking the most amount of episodes.

The results are shown for the P1 spot to illustrate the behavior of the algorithm. For the other seven spots, the results are similar with the same behavior, but different rewards, based on each spot.

6 Conclusion

The problem of the 6x7 simplified parking lot proposed on this work is a great example to use reinforcement learning to solve. It can be clearly formulated as an MDP and the initial policy created using the transition probability function. The algorithms selected to solve it was Deep Q-learning with Replay Memory, which uses a DQN, a neural network to approximate the Q values.

The implementation of the algorithm was generally straightforward, with a few problems faced but overcame after some thought. As described in section 4, two classes were created, DeepQLearning and DQNAgent, with the latter being used by the first to implement the Q-learning with a DQN.

The algorithm main parameters are presented on Table 5. The decay factor is applied to the ϵ (epsilon) parameter to dynamically decrease it after each episode.

References

Brockman, G.; Cheung, V.; Pettersson, L.; Schneider, J.; Schulman, J.; Tang, J.; and Zaremba, W. 2016. OpenAI Gym. *CoRR*, abs/1606.01540.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; and Riedmiller, M. A. 2013. Playing Atari with Deep Reinforcement Learning. *CoRR*, abs/1312.5602.

Sutton, R. S.; and Barto, A. G. 2018. *Reinforcement learning: An introduction*. MIT press.