A Modified Deep Q-Network Algorithm Applied to the Evacuation Problem

Marcin Skulimowski[0000−0002−7087−6281]

Faculty of Physics and Applied Informatics, University of Lodz
Pomorska 149/153, 90-236 Lodz, Poland
marcin.skulimowski@uni.lodz.pl

Abstract. In this paper, we consider reinforcement learning applied to the evacuation problem. Namely, we propose a modification of the deep Q-network (DQN) algorithm, enabling more than one action to be taken before each update of the Q function. To test the algorithm, we have created a simple environment that enables evacuation modelling. We present and discuss the results of preliminary tests. In particular, we compare the performance of the modified DQN algorithm with its regular version.

Keywords: Reinforcement Learning · Deep Q-Networks · Evacuation

1 Introduction

Reinforcement Learning (RL) is a subdomain of machine learning that allows the agent to gain experience and achieve goals only by interacting with its environment (without any "teacher"). There has been considerable interest in RL and its applications in many domains in recent years, e.g. games, robotics, finance and optimization (see [1, 2]). In this paper, we consider RL applied to the *evacuation problem*, i.e. to model humans in evacuation scenarios. The problem has been widely studied, mainly using cellular automata (CA) [3, 4]. Developments in RL have led to approaches based on various RL algorithms [5–7, 9] also in conjunction with CA [8]. In the most common RL approach to the evacuation problem, computationally expensive multi-agent methods are applied (see, e.g. [5, 9]). It means that each person is considered as a single agent. The approach discussed in this paper is different. Namely, we consider one agent that must evacuate all people from the room as quickly as possible through any available exit. The approach is similar to the one proposed by Sharma et al. [6]. The difference is that we consider evacuation from one room and Sharma et al. consider evacuation from many rooms modelled as a graph. Moreover, they use a deep Q-network algorithm and their agent acts by moving one person at a time step. This paper presents some modification of the deep Q-network (DQN) algorithm, enabling more than one action to be taken before each Q function update. Namely, our agent takes as many actions as is the number of persons in the room. Only then the Q function is updated. To test the proposed algorithm, we have

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77964-1_20

created a grid-based evacuation environment that enables modelling evacuation from rooms with many exits and additional interior walls. The paper is organized as follows. Main RL concepts and deep Q-networks are shortly presented in the next section. Section 3 describes, in short, the created evacuation environment. The proposed modification of the deep Q-network algorithm is presented in Section 4. Some conclusions are drawn in the final section.

2 Q-Learning and Deep Q-Networks

In RL the sequential decision problem is modelled as a *Markov decision process* (MDP). Under this framework, an *agent* interacts with its *environment*, and at each time step t, it receives information about the environment's *state* $S_t \in \mathcal{S}$. Using this state, the agent selects an *action* $A_t \in \mathcal{A}$ and then receives a *reward* $R_{t+1} \in \mathbb{R}$. The action changes the environment's state to $S_{t+1} \in \mathcal{S}$. The agent behaves according to a policy $\pi(a|s)$, which is a probability distribution over the set $\mathcal{S} \times \mathcal{A}$. The goal of the agent is to learn the *optimal policy* that *maximizes the expected total discounted reward*. The *value* of taking action a in state s under a policy π is defined by:

$$Q_\pi(s,a) \equiv \mathbb{E}[R_1 + \gamma R_2 + \ldots | S_o = s, A_o = a, \pi]$$

where $\gamma \in [0,1]$ is a *discount factor*. We call $Q_\pi(s,a)$ the *action-value function for policy* π. The *optimal value* is defined as $Q_*(s,a) = max_\pi Q_\pi(s,a)$. An *optimal policy* is derived by selecting the highest valued action in each state. *Q-learning* algorithm allows obtaining estimates for the optimal action values [10]. In cases where the number of state-action pairs is so large that it is not feasible to learn $Q(s,a)$ for each pair (s,a) separately, the agent can learn a parameterized value function $Q(s,a;\boldsymbol{\theta}_t)$. After taking action A_t in state S_t, obtaining the reward R_{t+1} and observing the next state S_{t+1} parameters are updated as follows [11]:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha(Y_t^Q - Q((S_t, A_t; \boldsymbol{\theta}_t))\nabla_{\boldsymbol{\theta}_t} Q(S_t, A_t; \boldsymbol{\theta}_t)) \tag{1}$$

where α is a *learning rate* and the *target value* Y_t^Q is defined as:

$$Y_t^Q \equiv R_{t+1} + \gamma \underset{a}{max} Q(S_{t+1}, a; \boldsymbol{\theta}_t) \tag{2}$$

During the learning, the agent selects actions according to the so-called ε-greedy policy: with probability $1-\varepsilon$, it selects the highest valued action, and with probability ε, it selects a random action. Learning takes place in the following loop (we omit subscript t for simplicity):

Initialize the value function Q;
foreach *episode* **do**
  Initialize S;
  while S *is not the terminal state* **do**
    Select action A according to an ε-greedy policy derived from Q;
    Take action A, observe R and the next state S';
    Update the value function $Q(S, A; \boldsymbol{\theta})$ towards the target value Y^Q;
    $S \rightarrow S'$
  end
end

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77964-1_20

A *deep Q network (DQN)* is a multi-layered neural network that for a given input state s returns $Q(s;\cdot;\boldsymbol{\theta})$, where $\boldsymbol{\theta}$ are network parameters [11]. Using the neural network as an approximator can cause learning instability. To prevent this Mnih et al. [2], proposed using the so-called *target network* (with parameters $\boldsymbol{\theta}^-$), which is the same as the *online network* (with parameters $\boldsymbol{\theta}$). Parameters $\boldsymbol{\theta}^-$ are copied periodically (each τ time steps) from the online network, thus reducing undesirable correlations with the target:

$$Y_t^{DQN} \equiv R_{t+1} + \gamma \underset{a}{max}(S_{t+1}, a; \boldsymbol{\theta}_t^-) \tag{3}$$

The second modification to improve the learning stability is the so-called *memory replay* [2], in which the agent's experiences $(s_t, a_t, r_{t+1}, s_{t+1})$ at each time-step are stored. A mini-batch of experiences is sampled uniformly from the memory to update the online network during the learning. Finally, one more improvement of DQN called *Double DQN* (DDQN), which eliminates overestimation of Q values (see [11]). The solution is to use two different function approximators (networks): one (with parameters $\boldsymbol{\theta}$) for selecting the best action and the other (with parameters $\boldsymbol{\theta}^-$) for estimating the value of this action. Thus the target in Double DQN has the following form:

$$Y_t^{DoubleDQN} \equiv R_{t+1} + \gamma Q(S_{t+1}, \underset{a}{argmax} Q(S_{t+1}, a; \boldsymbol{\theta}); \boldsymbol{\theta}_t^-) \tag{4}$$

3 The Room Evacuation Environment

In order to test RL algorithms we have created the following evacuation environment.

A **state** refers to a room with walls (W), exits (E) and people (P) (see Figure 1). A room of size $x \times y$ can be represented as a $3 \times x \times y$ tensor consisting of 3 matrices $x \times y$. Each matrix is an $x \times y$ grid of 1s and 0s, where digit 1 indicates the position of W, E or P (see Fig. 1). *The terminal state* is an empty room.

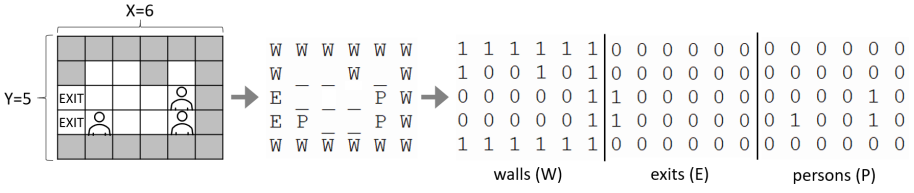

Fig. 1. A tensor representation of a room with 3 persons and 2 exits.

An agent takes **action** by selecting one field from $x \times y$ fields for a given situation in a room. Thus, the number of possible actions at a time step is equal to $x \times y$. It is worth noting that selecting a field is not equivalent to moving to that field because the move is not always possible. If there is one person near

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77964-1_20

the field selected by the agent then this person moves to this field. If there are two or more persons near the selected field, the move is made by a person who comes closer to the exit after the move or by random person.

After each action, the agent receives a **reward**. The reward depends on whether the move on a selected field is possible or not. There are three cases possible:

1. The move to a selected field is possible because the field is empty, and there is a person on an adjacent field who can move. The agent receives $R = +1$ when the selected field is an exit (Fig. 2A) and $R = -1$ in other cases (Fig. 2B).

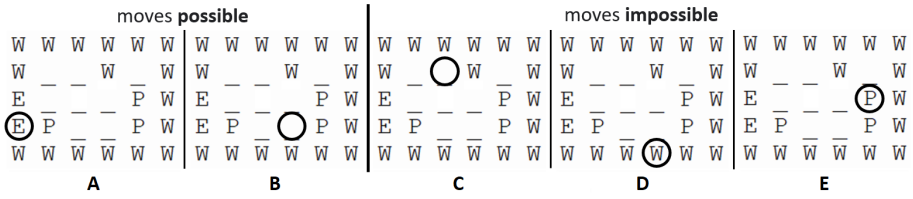

Fig. 2. Possible moves (left side) and impossible moves (right side).

2. The move to a selected field cannot be made because on an adjacent field there is no person that can move (Fig. 2C) - the agent receives $R = -1$.
3. The move to a selected field is not possible because it is occupied by another person or a wall (Fig. 2D-E) - the agent receives $R = -1$.

We used the above values of rewards in our tests, but they can be easily changed. The environment can be *static* or *dynamic*. In the static case, the initial number and positions of persons in the room are the same in each learning episode. The dynamic case is more challenging than static - the initial persons' positions in the room are random. During our tests, the dynamic version of the environment was used. The agent aims to evacuate all the people from the room quickly. What is important, the agent has no initial knowledge about the positions of exits and walls in the room.

4 Modified Deep Q-Network and its Tests

In standard DQN algorithm, the agent, at each time step, selects one action using an ε-greedy policy derived from Q. Next, the agent takes this action, observes a reward and the next state. After that, the update of Q is performed. However, in the evacuation scenario, people in the room can move simultaneously. It means that more than one action can be taken at each time step. The crucial point is that $x \times y$-dimensional vector $Q(S, \cdot; \boldsymbol{\theta})$ contains values of all $x \times y$ (possible and impossible) actions in the room. Consequently, instead of one action, we can select p actions $A_1, A_2, ..., A_p$ from $Q(S, \cdot; \boldsymbol{\theta})$ for a given state S, where p is the number of persons in the room. Namely, we can apply an ε-p-greedy policy during

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77964-1_20

the learning, i.e. with probability $1-\varepsilon$ we select p highest valued actions, and with probability ε we select p random actions. Note that actions $A_1, A_2, ..., A_p$ are in some sense *mutually independent* because they correspond to different fields of room. After taking all actions $A_1, A_2, ..., A_p$ the state S changes to S'. Then, we can update Q. The while loop in the proposed algorithm has the following form:

while *S is not the terminal state* **do**
  Select actions $A_1, A_2, ..., A_p$ according to an ε-*n*-greedy policy derived from Q;
  Take actions $A_1, A_2, ..., A_p$, where p is the number of people in the room, observe rewards $R_1, R_2, ..., R_p$ and the next state S';
  Find the targets:

$$Y_i = R_i + \gamma \underset{a}{max} Q(S', a; \boldsymbol{\theta})$$

  where $i = 1, 2, ..., p$;
  Perform a gradient descent step on $(Y_i - Q(S, A_i, \boldsymbol{\theta}))^2$;
end

Note that the number of people in the room decreases over time and, consequently, the agent's actions. For $p = 1$ the above algorithm is simply Q-learning.

To test the proposed algorithm, we consider a room of size 8×8 with two exits and an additional wall inside containing 18 people placed randomly at each episode's start (see Fig. 4). We implemented a convolutional neural network with the following configuration:

```
Conv2d(3,280,kernel_size=(5,5),stride=(1,1),padding=(2,2))
Conv2d(280,144,kernel_size=(3,3),stride=(1,1),padding=(1,1))
Conv2d(144,64,kernel_size=(3,3),stride=(1,1),padding=(1,1))
Conv2d(64,64,kernel_size=(3,3),stride=(1,1),padding=(1,1))
Conv2d(64,32,kernel_size=(3,3),stride=(1,1),padding=(1,1))
Flatten(start_dim=1,end_dim=-1)
Linear(in_features=2048,out_features=64)
```

The ReLU function is used for all layers, except the output layer, where a linear activation is used. The Adagrad optimizer with default parameters and a learning rate equal to 0.01 is used for training. We tested two agents. The first agent (DQN agent) uses the double DQN algorithm with the target network and memory replay. The second agent (MDQN agent) uses the *modified* double DQN algorithm with the target network. Both agents have the same values of the parameters: $\varepsilon = 0.1$, $\gamma = 0.9$, $\tau = 40$ (target network update frequency). Moreover, we set *memory replay* $= 500$ and *batch size* $= 100$ for the DQN agent. Both agents were trained for 1300 episodes. Figure 3 (left side) shows changes in total reward per episode. We can see that the DQN agent learns faster than the MDQN agent. The reason for this seems rather obvious. The DQN agent learns to take only one action at each time step. The MDQN agent has to learn to take more actions; consequently, it learns slower. On the right side of Figure 3, we can see that the number of time steps required to evacuate all persons from the room is much smaller in the case of MDQN. Figure 4 indicates a significant difference between both agents. The DQN agent evacuates persons one by one. It takes one person and moves it from its initial position to the exit. Then the

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77964-1_20

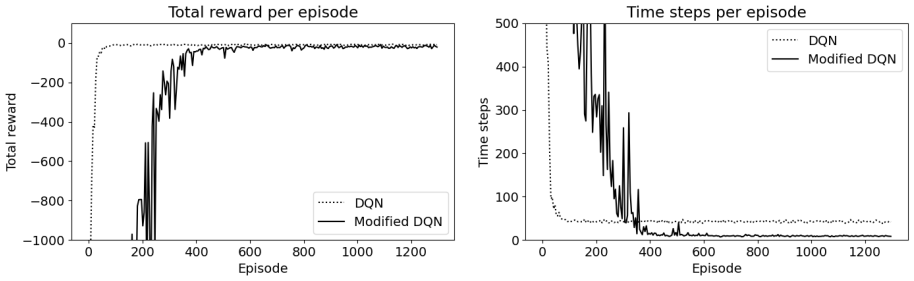

Fig. 3. The evacuation of 18 persons through 2 exits: total reward (left side), time steps (right side) - averages over 5 tests.

next person is evacuated. We can see in Figure 4A that three persons marked with frames stay in place, waiting for their turns. In the MDQN agent's case (Figure 4B), more than one person is moved to exits each time step. Figure 4B also shows that the MDQN agent evacuates persons via the nearest exit even if there are two exits located on the opposite walls.

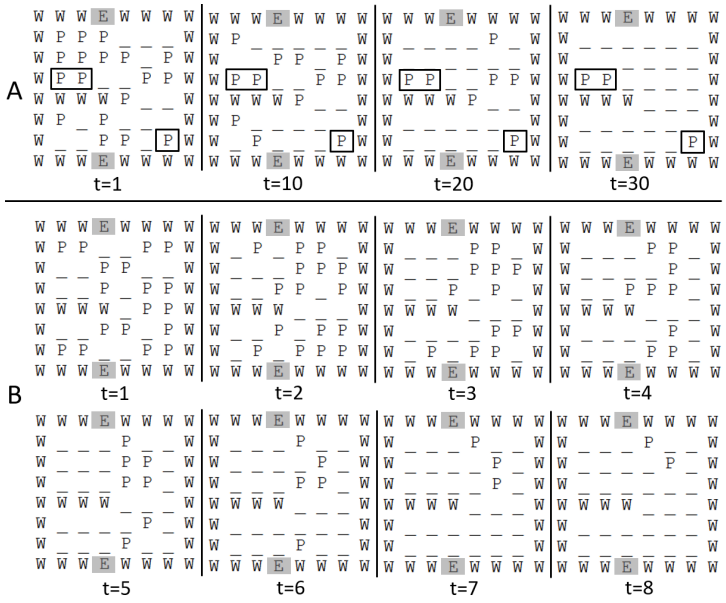

Fig. 4. The evacuation of 18 persons through 2 exits: (A) DQN, (B) Modified DQN.

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77964-1_20

5 Conclusions

Our preliminary results show that the proposed modified DQN algorithm works quite well in the evacuation scenario. Further work needs to be done to evaluate the algorithm in more complicated cases, e.g. for larger rooms, more people, obstacles and fire spreading. An interesting issue to resolve for future research is also finding out to what other problems than evacuation the proposed modification of DQN can be applied to. We can see a similarity between the proposed algorithm results and the result of multi-agent systems (see Fig.4B). Considering that the multi-agent RL methods are computationally expensive, the question arises whether the proposed algorithm can be an alternative to the multi-agent approach in some cases.

References

1. Barto, A., P. S. Thomas and R. Sutton.: Some Recent Applications of Reinforcement Learning (2017).
2. Mnih, V., Kavukcuoglu, K., Silver, et al.: Human-level control through deep reinforcement learning. Nature, 518 (7540), 529–533 (2015).
3. Gwizdałła, T. M.: Some properties of the floor field cellular automata evacuation model. Physica A: Statistical Mechanics and its Applications, Vol. 419, 718-728 (2015).
4. Jun, H., Xiaoling, G., Juan, W., Yangyong, G., Mei, L., Jierui, W.: The cellular automata evacuation model based on Er/M/1 distribution. Physica Scripta. 95, (2019).
5. A. Wharton, Simulation and Investigation of Multi-Agent Reinforcement Learning for Building Evacuation Scenarios, 2009. https://www.robots.ox.ac.uk/ ash/4YP
6. Sharma, J., Andersen, P., Granmo, O. and Goodwin, M.: Deep Q-Learning With Q-Matrix Transfer Learning for Novel Fire Evacuation Environment. IEEE Transactions on Systems, Man, and Cybernetics: Systems (2020).
7. Zhenzhen, Y., Guijuan Z., Dianjie L., Hong L.: Data-driven crowd evacuation: A reinforcement learning method. Neurocomputing. Vol. 366, 314-327, (2019).
8. Ruiz S., Hernández B.: A Hybrid Reinforcement Learning and Cellular Automata Model for Crowd Simulation on the GPU. In: Meneses E., Castro H., Barrios Hernández C., Ramos-Pollan R. (eds) High Performance Computing. CARLA 2018. Communications in Computer and Information Science, Vol. 979. Springer, Cham (2019).
9. Martinez-Gil, F., Lozano, M., Fernandez, F.: Marl-ped: A multi-agent reinforcement learning based framework to simulate pedestrian groups. Simulation Modelling Practice and Theory 47 (Complete), 259-275 (2014).
10. Watkins, C.J.C.H., Dayan, P. Q-learning. Mach Learn 8, 279–292 (1992).
11. van Hasselt, H., Guez, A., and Silver, D.: Deep reinforcement learning with double Q-learning. arXiv:1509.06461 (2015).

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77964-1_20