Reinforcement learning based single-agent and multi-agent systems for cleaning robots

Mika Sipilä

30.3.2021

1 Introduction

Cleaning robots have been on market for at least two decades already. With hard-coded algorithms or beforehad defined moving patterns, it is hard to create a cleaning robot which can successfully clean the entire environment and perform well in many different environments. In this paper reinforcement learning is suggested as a method to train cleaning robots to clean their environment as efficiently as possible.

In recent years reinforcement learning methods have been researched increasingly in context of single- and multi-agent systems. Deep Q-learning is one of the reinforcement learning methods that could be suitable for solving this kind of problem. In this paper I will define how cleaning robots will interact with their environment, what they know of the environment, how the robots communicate with each other, how they choose their actions and how they learn to accomplish cleaning tasks alone and together with other cleaning robots in many different environments. I will also give an example how deep Q-learning could be implemented to solve this task. The idea is that similar approach could be used with other reinforcement learning methods aswell.

In background and theory -section I will go briefly through basic idea behind deep Q-learning and explain why I chose that method. In implemetion-section I will define the environment, agent's states and actions, how agent is rewarded for it's actions and how agent learns. I will also go though a single-agent scenario and a multi-agent scenario and discuss about the differences between them and about the difficulties in multi-agent scenario. Lastly I will give brief conclusions and discuss about possbile future work.

2 Background and theory

In this paper deep Q-learning is propsed as a reinforcement learning technique to train cleaning robots to successfully clean their environment. In cleaning task, agents interact with the environment E in sequence of actions, observations and rewards. At each time step each of the agents takes an action from the set of possible actions $A = \{0, 1, ..., K\}$ based on agent's state $s \in S$. The action is then passed to the environment, and it may modify the environment. In addition agent gets reward based on it's action.

I decided to consider deep Q-learning instead of regular Q-learning, because goal is to have agent that is capable of working in multiple different environments. Given the fact that the agent's state is very high dimensional, the regular Q-learning would require too large Q-table to handle all the possible states. In deep Q-learning, Q-function is estimated by deep neural network and the idea is that agent would be able to perform well in totally new situations if it has experience from something similar in the past. The basic idea of deep Q-learning is briefly covered in this section.

Deep Q-Network (DQN) based agent has two important properties: Q-function with weights w and experience replay memory. At each time step weights w of the Q-function are updated instead of updating Q-table. This can be done for example by using neural network with stochastic gradient descent and back propagation, where the prediction error is based on Bellman's equation:

$$Error(s, a, r, s', w) = r + \gamma max_A Q(s', a', w) - Q(s, a, w),$$

where s is initial state, a is the executed action, r is the initial reward for the action, s' is the state after the action and w is current weights of Q-network. γ is discount factor for future rewards, which is used to manipulate how much weight is put on future rewards compared to initial reward. $max_A Q(s', a', w)$ means taking maximum value of Q-function of any possible next action from action space A. (Arango, 2018)

On each time step the transition tuple (s, a, r, s') is stored in the replay memory. Training is done on each time step, but not based on that transaction but the random sample of batch size b from the replay memory. This solves the problem of correlated observations, because the replay memory grows really large really fast. To reduce non-stationarity, the separate Q-network $\hat{Q}$ is often used with fixed weigths $w*$ to calculate targets:

$$Target = r + \gamma max_A \hat{Q}(s', a', w*).$$

The weights from original Q-network is cloned to target network $\hat{Q}$ for every C time steps, where C is a hyperparameter. (Arango, 2018)

3 Implementation

In this paper cleaning robots, or *agents*, are considered as vacuum cleaner robots that clean the floor by sucking dust and dirt while moving around around. Goal of this kind of robots is to clean the whole floor of the house as effeciently as possible. This means that the robots have to visit same places as little as possible before every square of the house is cleaned. When there is more than one robot cleaning the house, the robots have to be aware of each other or communicate somehow to avoid cleaning the already cleaned places again. If robots end up next to each other, they must be capable of communicating and moving towards different uncleaned areas.

This solution in based on assumption that the robots are able to know their position in the house based on wi-fi, bluetooth and GPS. The robots are saving the positions they visited, walls and obstacles in the memory based on their location, direction and distance moved. If the loation is not exactly precise, the robots can modify the memory and get the idea of their surroundings more accurate each time they move in it. This paper does not cover more deeply the possible solutions for tracking robots' locations, but rather consentrates on solution for how robots can get familiar with their environment, how they can they clean efficiently and how they can communicate with each other based on this assumption.

3.1 The environment

The environment where the robots live, is some kind of house, room or other flat floored place where it is possible to move around. For the sake of simplicity, the environment is considered to have only rectangular shapes and obstacles. In the house it is possible that someone moves furniture around, so environment may change over time. The environment can be at any size and it can have one or multiple rooms and any kind of rectangular shaped obstacles. That's why the robots have to be capable of successfully cleaning in different environments.

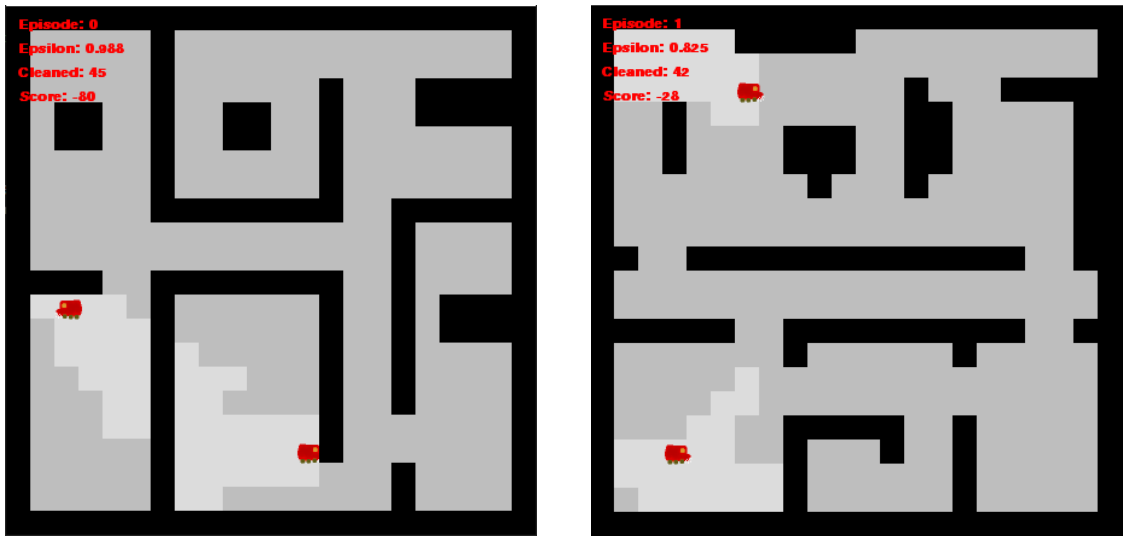

Figure 1: Two examples of the possible environments for the cleaning robots described from above.

3.2 Memory, actions and states

The robots are blind and they start in a new environment with full ignorance. As they move, they continuously save information about their environment and start to learn about it.

The robots have a memory of the places they have visited. The memory is shared between robots and each robot updates the memory as the robot explores the environment. For each position of environment, the memory contains the information of whether the position is *unknown*, *available* or *blocked* by some obstacle such as a piece of furniture or a wall. For the available positions, the memory also contains the information whether the position is cleaned or not and how recently the position is cleaned. Each position is the size of robot, for example 40cm x 40cm.

The information of the explored positions is stored as decimal numbers in two dimensional matrix (large enough to store values even for big environments). If the position is blocked, it is stored as -1, if it is available, it is stored as decimal number from 0 to 0.5 depending on how recently it has been cleaned. If it just cleaned, it has the value 0.5 and the value decreases by $\frac{1}{C}$ each time step. C is the number of time steps in which the position is considered totally dirty again. If the position is unknown for the robots, it is stored as -0.2. If there is a robot in that position, it is stored as 1.

The cleaning robot is continuously sucking dirt and dust while it moves around. That's why it only has to decide which way it will move at any given state. Our environment is simplified version of the real life environment and it is made of rectangular shaped blocks, which is why we have to define only four possible action. The actions are *0: move up*, *1: move right*, *2: move down* and *3: move left*. The actions corresponds to directions in environment described from above.

For the logic behind an agent taking an action, I considered three different possibilities. One was to use robot's memory to find the shortest path to nearest possible unknown position and take an action accordingly. Agent would find the shortest path at any time using path finding algortihm like Dijkstra's algorithm (Dijkstra, 1959). How ever this would have been problematic for multi-agent scenario, because agents might have the same nearest unvisited position and they might end up following each other. This also requires a lot of computational power in cases where the nearest unknown position is really far away.

The other two logics are reinforcement learning based, or more specifically Deep Q-Learning based solutions. Deep Q-Network (DQN) takes agents state as input and decides the output, the action, based on what the network thinks would bring the greatest future reward. One possibility for determining agent's state is with multiple sensors that would sense the distances to obstacles for multiple directions of the robot. This solution is more expensive as it requires many expensive sensors. The solution, which i ended up with, is to determine agent's state based on the robot's memory. Agent takes the state from the memory as the large image-like square of it's surroundings, where it's own position is in the middle of it. In this solution, there is no need for any extra sensors. The state in this case is image-like information encoded to digits and hence is suitable as input for convolutional neural network.

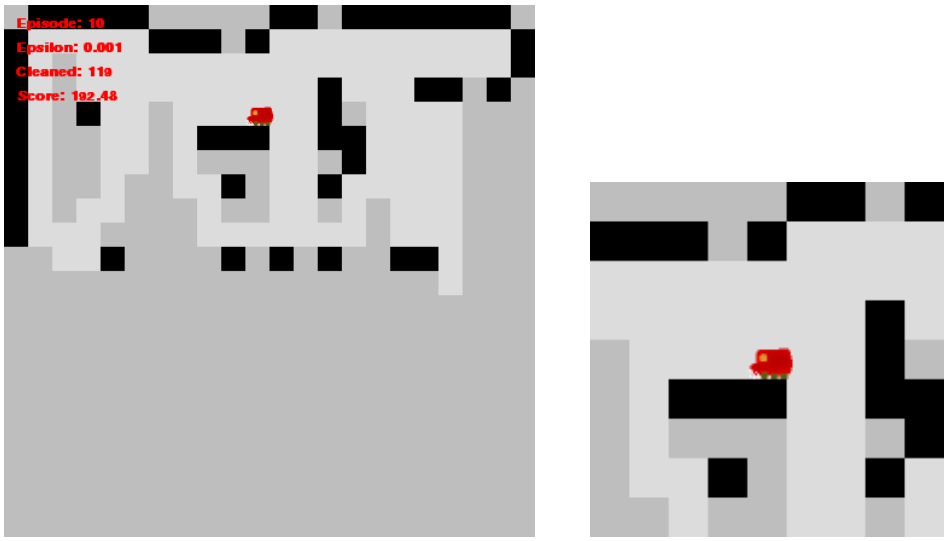

Figure 2: A representation of agents memory (left) in case where the agent haven't explored it's whole environment. The darker grey represents the area that is unknown for agent, black represents obstacles and lighter gray represents visited area. The picture at right is a representation of what agent sees from the environment at left picture. The state of the agent is picture at right encoded to digits. The area which agent sees could also be determined to be larger.

3.3 Reward function

The reward function r has to be defined in a way that it encourages agents to take actions towards achieving their goal. The goal is to clean as much as possible as efficiently as possible so we have to give agent positive reward for cleaning dirty position and give negative reward for moving to already cleaned position, to wall or to obstacle. Given the fact that agent starts with full ignorance, we also have to give agent positive reward for visiting new position for the first time or hitting the wall or obstacle for the first time. Let's define the reward as following:

$$r(a,s) = \begin{cases} 2 - 2 * \text{next\_pos.value}, & \text{agent cleans uncleaned position} \\ -1, & \text{agent takes an action} \\ 2, & \text{when agent moves to new position or hits obstacle,} \end{cases}$$

where next_pos.value indicates the value stored in memory of the position agent is about to move. The idea behind this reward function is to encourage agent to explore the environment as much as possible and when the most of the environment is explored it encourages agent to start reclean the most dirty positions. By penaltizing each move agent takes, the total reward will be highest when the agent plans it's way by avoiding already cleaned positions as much as possible. When the agent ends up in a situation where all of it's surroudings are already cleaned, the reward function encourages agent to move in a direction where the positions have been cleaned longest time ago. If agent can learn this, it prevents agent from getting in a loop where it goes back and forth without knowing what to do next.

3.4 The structure of deep Q-network, and agent's learning

In single-agent scenario it is much more straight forward to determine how agent learns as there is no need for determining how agents communicate with each other, what information they share and how they use the shared information. In single-agent case the input for Q-network is simply the state of the

agent i.e. the information of agents surroundings encoded as digits. The Q-network has the structure of two dimensional convolutional layers lead into fully connected layers. The last layer contains four values, one for each possible action and the output is network's estimation of Q-values for each action. The agent executes the action which has the highest Q-value.

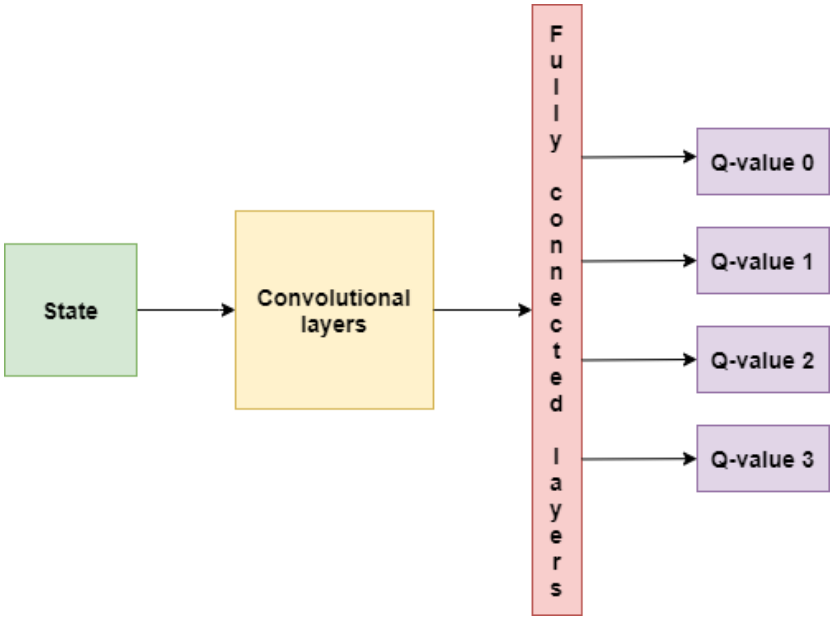

Figure 3: The general structure for deep Q-network in single-agent scenario. The number after Q-value indicates the number of action. The agent chooses the action which has the largest Q-value.

The network should be deep enough to capture many different features from the states and to successfully learn how to perform well in different situation. The dropout layers could be considered for fully connected layers to avoid overfitting to training environments. It is not reasonable to define the structure of Q-network more in detail in this theorethical approach, the most suitable structure must be searched when training the agents.

In multi-agent scenario, the challenge is to define how agents communicate with each other, how do they share information and what information do they share. Communication happens via agents shared memory, which is accessible by wi-fi or bluetooth etc. They could share their full states with each other or alternatively a processed vector of information of their state. The processing would happen with neural network and the goal of it is reduce dimesions of shared information and extract only meaningful parts of it.

For cooperating agents, I considered two possible approaches. The first one is that all the agents would be controlled by one neural network, which chooses the actions for all of the robots and then gets the reward which is the sum of individual agents' rewards. The updating of DQN would then go as in single-agent case except that maxmium of Q-values would be taken over joint action space $A_1 \times A_2 \times ... \times A_n$, where n is the number of agents. In this techinque the problem is that agents doesn't get information about the location of other agents unless they are close enough that they mark each other to their input states. Hence the states of other agents might be meaningless when choosing actions.

Another approach is that agents share their information only when they are near enough of each other so that they can get information of each other via their own input aswell. The agents would share processed vector of their information, which is meant to represent the cleaning plan of that agent. Then the processed inputs of nearby agents are put into deep Q-network together with agent's own state and the action is chosen based on all the information. In this case agents share the reward function if they are close to each other. Othewise they will be rewarded individually. For example, if two agents are in range of seeing each other, the reward will be $r_1 + r_2$ for both of the agents. This will encourage agents to work together when they are close to each other.

The idea behind this approach is that agents would learn to avoid each other i.e. plan their way to different direction if they see each other. If there is nothing to clean in other direction, then agents may end up cleaning same area side by side. In this scenario agents would get information of other agents' plan and can adapt accordingly. Because the agent will get rewarded even if the other agent cleans the area, it will bring larger reward to head for a different positions.

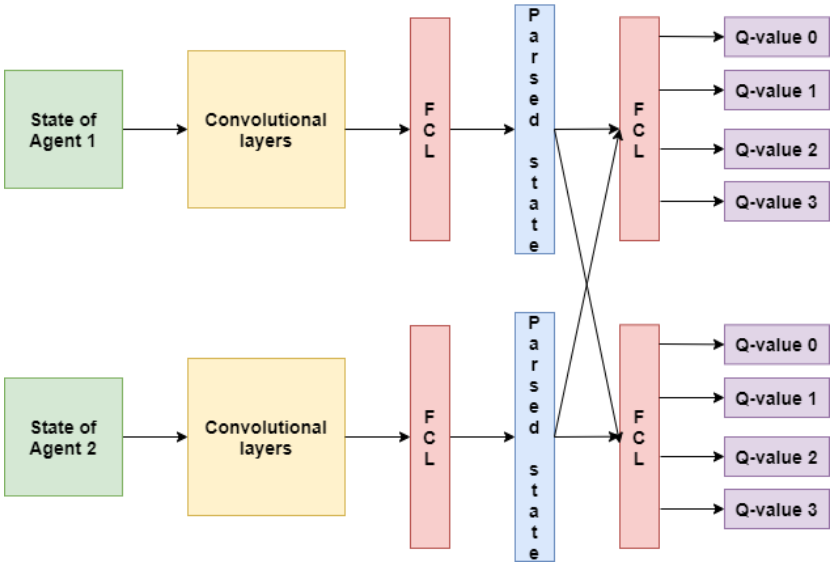

Figure 4: The general structure for deep Q-network in multi-agent scenario. The figure is an example of two agents communicating with each other. The number after Q-value indicates the number of action. The agent chooses the action which has the largest Q-value. FCL means fully connected layers, in one FCL block there can be multiple fully connected layers.

In both, single-agent and multi-agent cases, agents would be trained in virtual environment where it is easy to create thousands of different environments. Environments would vary on sizes and room plans. There can be also specific environments for some harder tasks like cleaning a single room together as efficiently as possible. There can also be scenarios where agents have full ignorance and scenarios where agents have already memorized the environment. Some stopping criteria needs to be defined for each environment i.e. when one training episode ends. This can be e.g. to *stop when the whole environment is cleaned or if agents gets stuck/running in circle for some period of time.*

The learning starts with epsilon-greedy method, which means that agent takes only random actions in start to get an idea of the environment. The value of epsilon is decreased a little each time the agent takes an action and when the value of epsilon is decreased, agent starts to take actions based on deep Q-network. When epsilon reaches zero (or almost zero) agent is learning only based on it's own

decisions and not random actions. The fact that agent takes also random actions when it is learning, leads more likely to global optimum instead of some local optimum.

Goal is that agents learn really robust behavior of how they can accomplish cleaning tasks effectively in different environments. If results doesn't satisfy as well as needed, the one possibility is to let agent train and fine-tune it's parameters in it's environment in real world after training in virtual world.

4 Conclutions and future work

In this paper deep Q-learning is proposed as reinforcement learning method for single-agent and multi-agent systems for agents to accomplish cleaning tasks. Goal is to get virtually trained cleaning robots perform well in any environment in the real world. The most important part of this research is not how to implement deep Q-learning for this task, but how to define how agents interact with their environment, what informations they get from it and how they interact with each other. The important part is to give an idea how this kind of task could be done with agents learned with any reinforcement learning method.

There is much more advanced reinforcement algorithms already that might be more suitable for this kind of task. It is possible that deep Q-learning as it is presented here, is not good enough techinque to get really robust and well performing agents. In future it would be interesting to develop a virtual environment to test out many different reinforcement learning approaches and deep neural network structures for this task. I strongly believe that with more research and some practical testing on this topic, it is possible to use reinforcement learning methods to train really well performing and cooperating agents who success well in tasks like cleaning environment as effeciently as possible.

References

Arango, M. (2018). Deep Q-Learning Explained. 10.13140/RG.2.2.22983.14241.

Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. Numerische Mathematik, 1(1), 269–271.