Udacity Connect Machine Learning Capstone Project Proposal

Ramtin Raji Kermani, ramtin@asu.edu

April 15th, 2018

Proposal

Domain Background

One of the most interesting household product families in market today are the autonomous robotic vacuum cleaners such as iRobot Roomba from iRobot corporation which is sold over 10 million unit in 2017[1] and these machines are loved by thousands. One of the main challenges of household robotic systems is the problem of accurate indoor localization and path planning. Even though these robots are called “Smart” and “Autonomous”, these machines still are far from being intelligent and efficient. They lack basic learning capabilities even though they posses various sensors to sense the environment in which they operate. Most of the sensory data is used for reactive control (avoiding a fall or hitting a wall). No matter how many times the robot runs around a small house with basic floor plan, the next time it will still bump into the couch or assumes that all areas of the house have the same amount of “dirt” to be cleaned. These issues decrease the performance of the robots as well as their life-time.

Reinforcement learning is one if the most interesting branches of machine learning that trains an agent to be **“Good”** and **“Behave nicely”** and **“Achieve goals efficiently”** by giving it a chance to explore the environment and perform actions by “**trial and error”**. The agent, based on the outcome, is either “**Punished**” or “**Rewarded**”. The rewards or punishments are awarded based on agent’s behavior, achievements and weather or not it was able to achieve its **final goal**.

Depending on the state space, the agent needs to run thousands or hundreds of thousands of times to cover all the possible states and learn what is the appropriate action to take in each state. An example state/action (representing a policy) for iRobot roomba could be a tuple as follow:

State: (location=(3,4), trashLocation=[(0,1), (0,3), (3,4)], hazardLocation=[(3,6), (2,3)])

Action: (turn 30 degrees and Go up 10 grid units)

However in reality it's not always feasible to let the robot to run 24/7 for weeks just to learn about the house and simulation might be a better option for the following reasons:

1- Mechanical machinery have a limited lifetime

2- It will take a long time for the agent to be trained

3- When dealing with real world scenarios we are talking about continuous space which presents its own issues that we will talk about later.

That's where simulation and simulated environments and agents come into play. Simulated environments are not always the most accurate as they don't represent the environment and physical world one hundred percent, however the advantages they offer makes up for the small inaccuracies.

Once the agent is trained in a sufficiently similar environment to the real world equivalent, we can export the model into a real robotic agent and with minor fine tuning, achieve a great result.

Problem Statement

I'm proposing to use a basic simulated environment to train a robot to learn to efficiently (and as quickly as possible) clean a room while avoiding the hazards. The goal is to train the agent to understand how it can get to the dirty areas in shortest steps and learn that it needs to avoid areas that are marked as dangerous (in a real world scenario these areas could be stairs or water pits, for example, that may damage the robot).

One may say we can manually program the robot with static rules to perform these tasks which could be feasible for simple and static environments. However when it comes to a dynamic and complex environments, this approach won't work as the program will fail as soon as a slight change in the environment is introduced. Machine Learning in general and reinforcement learning in particular will help us greatly to overcome these type of issues (which is known as over-fitting to the environment).

Datasets and Inputs

Inputs to this application will be the environment specification which includes for simplicity the width and height of the environment and number of grids. Another set of parameters are location of dirt cells as well as hazard cells.

In Reinforcement Learning, the agent initially has minimum data about the environment, itself and its "**Purpose**". It starts by running and performing tasks more or less randomly. These random actions puts the agent in various states and based on the action taken in each step, is awarded rewards or punishments. These rewards or punishments, become the learning states and for later steps, the data to the agent. Also the data based on which the agent is rewarded or punished could be considered the initial data of this learning algorithm.

The environment will be a Web-based application and I will use Javascript to create a simplified simulated environment that the robot will reside in, play around and learn its objectives and purpose.

The environment will be a configurable grid of size (W square units x H square units). The number of **hazardous cells** (indicated with a fire icon) as well as the number of **dirt cells** (depicted with a trash icon) are configurable. However in every run, the positions of these cells are randomly chosen and placed. The following pictures show the initial development of this environment. The first image shows the beginning of what we will refer to "**The Game**" from now on.

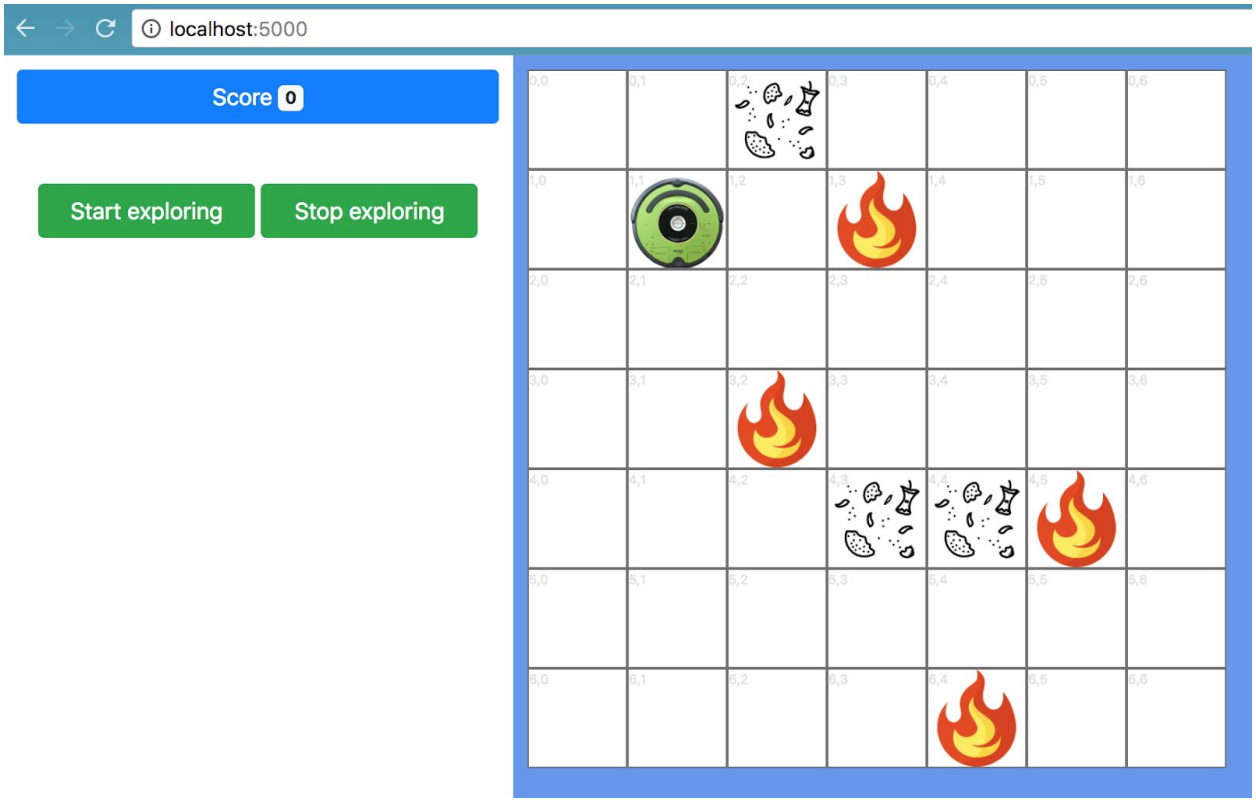

Figure 1) A basic demonstration of the initial environment

The second picture, shows a specific scenario where the agent has been able to collect all the trashes, has avoided the hazardous cells and has finished its job in time.

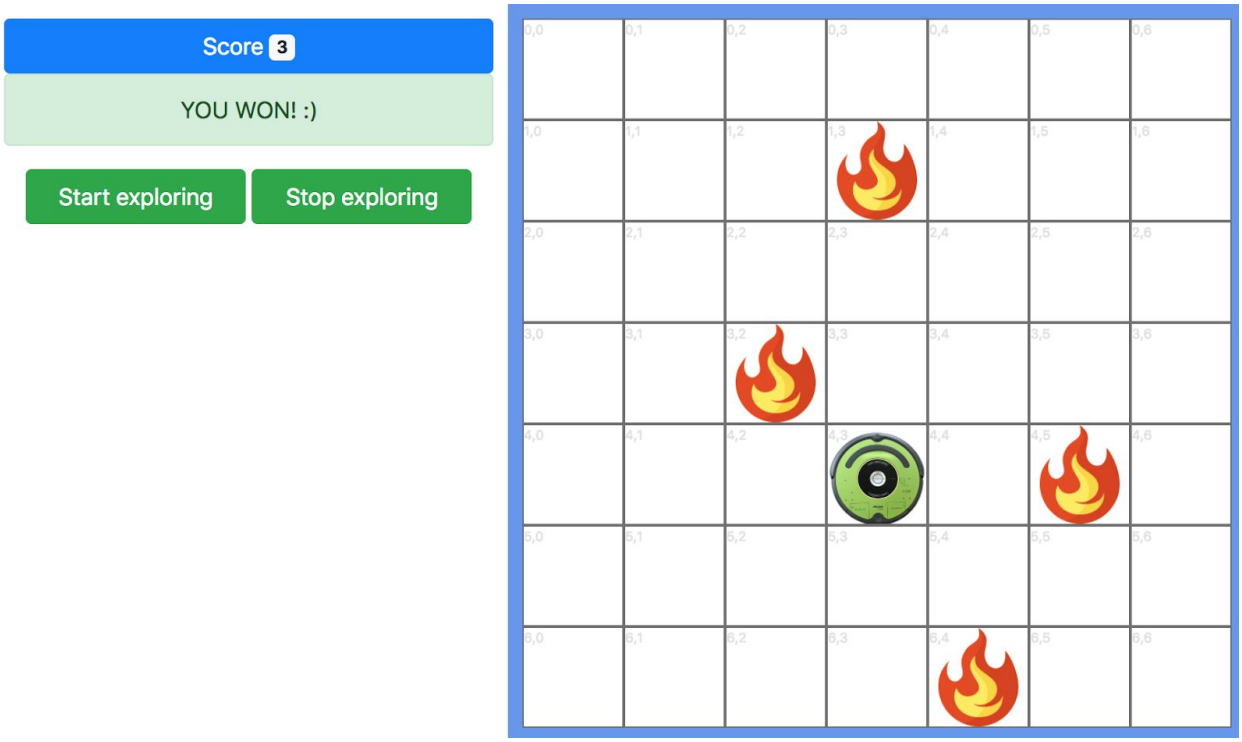

Figure 2) A basic demonstration of the environment, after the agent has collected all the trash cells and has "Won" the game.

Solution Statement

One of the most popular techniques in reinforcement learning is **Q Learning**[2] that is used to teach the agent learn about its purpose and goals and to perform efficiently. Initially, the agent does not know the rules and the fact that it's a vacuum cleaner and its goal is to collect all the dirts and avoid the hazardous cells. However, as it starts moving around and performing actions, it will learn to pick the most efficient action in each step. This is called the agent's **"Optimal Policy"**. These policies are stored in a table called the "**Q-Table**" which basically is a guide for the agent:

- "If you are in state X and perform action A => you receive 10 points"
- "If you are in state X and perform action B => you receive 5 points"
- "If you are in state X and perform action C => you lose 15 points"

And so on.

Each row in the Q-Table indicates a state and each column is one of the actions that agent can take in each state. The value of each cell is the "**Immediate Score**" the agent receives if it takes action A when in state X. The agent's can perform in one of the following modes:

- **Exploration**: Agent is learning. It takes many random actions to "fill" in the Q-Table and explore the opportunities. To learn "What happens if I take action A when I'm in state X".
- **Exploitation**: The agent has explored various scenarios enough that now has a relatively useful Q-table that it can exploit to actually perform the desired tasks. When in state X, it looks in the table and finds the action that gives the highest possible score and takes that action.

Initially we start learning by having the agent perform in the "Exporative" state. Once satisfied with the Q-Table and we have enough data per state per action, we can switch to "Exploitative" mode and actually utilize the learned Q-Table. However, every now and then (probably 5% of the times), we switch back to Explorative mode to see if there are better actions we can take in each step.

The overall goal is for the agent to learn to perform efficiently in the environment, to complete its job which is cleaning all the dirt and also avoid the hazardous cells. A time restriction is introduced to push the agent to finish the job as fast as possible. This basically implies that the agent needs to learn how to find the most efficient route (shortest) from its starting point to all the dirt cells while avoiding the hazardous areas.

To see if the agent is performing well, we will have a scoreboard that shows the agent's score and if it has won or lost the game. We will use this to evaluate how well the agent is performing.

Benchmark Model

For this problem, the benchmark would be the agent going around randomly in the environment and collecting dirt cells. Funnily enough this is almost the way most vacuum cleaners perform, well not quite randomly but not as intelligently as they could. Depending on the complexity and size of the environment, number of dirt cells and hazard cells, this benchmark will have different success rates, but in general, as we will

see, the success rate is very low, close to zero percent. We will include a functionality in the game to make the agent perform this way.

Evaluation Metrics

We will use binary scoring method to evaluate the performance of the agent in the environment, this means we only care about the final results in every round, not the immediate scores. Immediate scores are used to help the agent learn.

After the Exploration phase is done and we have finished filling the Q-Table, we start the Exploitation phase in which the agent keeps running the game for many rounds. The result of each round is either success or failure. The agent fails if it doesn't collect all the dirt cells in time or if it runs into the hazardous cells.

After say 1000 rounds of executions, we can evaluate the performance of the agent using the following formula:

Performance = number of times the agent has won the game / number of executions

Project Design

The project will be created in the following phases:

1- Creating the basic environment and agent model (80% done already)

The application is a Web application that can be hosted on a web server and the learning and agent performing, will run on the user's browser. The environment is created to resemble a simulation of a robot vacuum cleaner in a square room whose responsibility is to clean up all the dirt. This will be presented as a game that a player can Win or Lose.

2- Introducing the manual game play (100% done already)

A human can set the game to this mode and start playing using the arrow keys. This is intended to give a basic understanding of what is the robot is going to learn and how it is expected to perform.

3- Introducing the Random benchmark play (100% done already)

This mode, allows the robot to start running in the environment, randomly and we will see how (bad!) it performs. This is intended for us to have a base measure to compare with, after we have trained our agent.

4- Creating the Learning structure and mechanism

This is the most challenging part of the project. We need to define the “State”. Problem is if the State contains too many variables, the state space will be huge and it will take a long time for the agent to learn, also the Q-Table become huge and we may run into memory issues. Initially I thought about defining the state as the following tuple:

Even if the grid is of size 4x4, each cell can either of following four states:

1- Hazardous cell 2- Dirt Cell 3- Robot 4- empty cell

That will add up to 4 to the power of 16 which equals to 4294967296. This means we need to fill 4294967296 rows of the Q-Table for each move (Up, Down, Right, Left) which is going to take a very long time and probably we will run out of memory.

I’m proposing a different more simplified approach that reduces the number of states significantly.

Using Q-Learning we will be able to train this agent to efficiently perform in the environment and do its job. However, when dealing with continuous space and high dimensional spaces, there are more efficient algorithms and techniques to be used. One method that is used by **DeepMind** to train the computer to play Atari games is the **Deep Q-Networks**[3] which leverages the power of **Deep Neural Networks** to solve Reinforcement problems. I might need to look into using this technique if Q-Learning comes short solving this problem.

- [1] http://media.irobot.com/2017-07-25-iRobot-Reports-Strong-Second-Quarter-Financial-Results
- [2] https://en.wikipedia.org/wiki/Q-learning
- [3] https://deepmind.com/research/dqn/