TDDC17 Artificial Intelligence

LAB 5

Davgr686, Ottde625
Linköping Universitet

18/10/13

Part II:

Solution:

The update of the Q-table was done by implementing the algorithm explained in the slides explaining Q-learning. The update is done as follows:

/* Update-function for Qtable */

double value = Qtable.get(prev_stateaction) + (tmp_alpha * (previous_reward + GAMMA_DISCOUNT_FACTOR * getMaxActionQValue(new_state) - Qtable.get(prev_stateaction)));

/* Update the Qtable */

Qtable.put(prev_stateaction, value);

For this part we created 10 different states, depending on the angle of the rocket. The rocket is rewarded based on the current angle of it with the equation: 10 – (2*(Math.abs(angle)). After 100k iterations, the rocket faces straight up and uses the middle engine/the left and right engine to face up and fly. We discretize the angle by using the pre-defined function *discretize*, in order to get 10 states: discretize(angle, 10, -Math.PI, Math.PI). We use four different actions the rocket can execute: "Do nothing", use right engine, use middle engine, use left engine. At first we used 8 actions, the ones previously mentioned as well as combinations of two or more engines at the same time. The implementation using 8 different actions worked but caused problems in part III, which we will discuss later.

Our implementation works, but problems could occur since the rocket could be rewarded differently in the same state. A solution to this would be to reward the rocket based on the state it is in, rather than rewarding based on the angle.

7) **Question 1**

a) **Describe your choices of state and reward functions.**

We decided to use 10 states for the angle. Values less than -Pi are issued state 0 and values higher than Pi are issued state 10. Based on our observations, the angle equals -Pi/Pi when the rocket is facing down and 0 when facing up. Setting the limits in the discretization to -Pi and Pi means every angle the rocket can take is issued a state. Setting the limits closer to 0 would result in each angle lower than the min-limit being issued state 0 and higher than the max-limit being issued state 10.

The closer the rocket is to facing upwards (angle = 0), the higher the reward it receives. This results in the rocket wanting to face up at all times, which is the wanted behavior for the exercise.

b) **Describe in your own words the purpose of the different components in the Q-learning update that you implemented. In particular, what are the Q-values?**

The function: $Q(s, a) \leftarrow Q(s, a) + \alpha(R(s) + \gamma \max Q(s', a') - Q(s, a))$
The Q-table is a table of estimated utilities for executing action 'a' when in state 's'.

Q(s, a) is a Q-value or an entry in the Q-table; a sum of rewards for executing a specific action when in a specific state.

α is the learning rate. It calculates a value based on how many times a specific state + action has been tested. The lower the learning rate the faster the rocket will come up with a "solution" to the problem. If the learning rate is too low in comparison to the state space, the rocket might end up thinking a "bad" solution is the best. This is due to the state space not being sufficiently explored.

R(s) is the reward received when being in the previous state.

γ is the discount factor. In short it can be explained as how far ahead in time the rocket should search for rewards. A low discount factor means the rocket will search for rewards in the near future, while a high discount factor means it looks far ahead in time.

maxQ(s', a') is the expected max reward given the new state and all of the possible actions at that state. In other words, the highest Q-value given the new state and the possible actions at that state.

8) **Question 2:**

Try turning off exploration from the start before learning. What tends to happen? Explain why this happens in your report.

When turning off exploration in the start of the simulation, the rocket starts to fall and does not burst any engine. It keeps on doing this for the rest of the simulation. This happens because the rocket is calling ACTION 0. ACTION 0 correlates to not bursting any of the rockets so the rocket will keep on falling. All the utility values are default from the start and it has only discovered one state. The P_QVAL for that state and action will increase every time and the rocket will think that this is the most optimal action.

Part III:

Solution:

For this part of the lab, we created states based on the angle as well as the vx- and vy-values. We used 10 states for the angle, 3 for vx and 4 for vy, i.e. 120 states. The states are defined in the form "angle:vx:vy". The rocket is rewarded based on the angle it is facing, the value of vx and vy. The reward given increases the closer the rocket is to the goal (which is to hover). The actual rewards can be found in the code. At first 8 actions were used, which caused the rocket to sometimes "trying" to hover with minor movement and sometimes not hovering at all. The amount of actions was then changed to 4 (as in part II), which reduced the size of the Q-table. With a smaller Q-table (given fewer actions) the rocket is able to hover every time. We suspect that the Q-table grew to large with 8 actions (120 states, 8 actions), not being able to explore enough states+action before reaching 500k iterations.

4) Question 3

Would a linear model work, i.e. the Q-values are assumed a linear function of the state? Otherwise suggest some sufficiently powerful SL model/representation that could be used to learn a Q-function for this lab.

Supervised learning using a linear model does not work well for non-linear relations. In the case of the lab, for a linear model to be suitable, the Q-values would have to increase linearly depending on the (state, action). If this is not the case a linear model would not be suitable. Instead an Artificial Neural Network (ANN) could be used, which under some condition has a universal approximator to any function f(x). Potentially, this universal approximator would be better at fitting the actual model of the

data than a linear model, which would be a better representation of the actual data. The disadvantages with ANN are that there are a lot of tuning parameters to configure and it could be difficult to debug.

5) Question 4

We clearly cannot make a Q-table for all possible combinations of such pixel values. If we learn an approximation via supervised learning like in Question 3 above, what would a suitable model/representation be for supervised learning from image inputs?

Since making a Q-table for all possible combinations of each pixel value in an image would not be possible, it would be more suitable to use a Convolutional Neural Network (CNN). This type of neural network consists of convolutional layers, pooling layers and fully connected output layers. The basis of this architecture is to identify key parts of an image, rather than identifying the entire image. The architecture works by having a sliding window scanning the entire image, computing the sum of the specific part of the image. This sum is then used to "recognize" the structure of the image, rather than identifying the entire image as a whole. In the case of the lab, a CNN could be used to identify the positions of the wings and the nose of the rocket, in order to capture which direction the rocket is facing. The vx- and vy-values could be calculated by comparing the position of the rocket in one image to another (preferably "neighboring" images in a sequence).