Exercise 11

Kai Schultz

May 31. 2024

I forked the GitHub repo so all the code can be found on here: Github.

Task 1 (2 points): Learning the Q Way

States, Actions and Rewards

The state space should capture all relevant information needed to make a decision. In our case we have 7 different factors that should be included in the state. First we have the weather station that reports the sunshine level outside. Then we have the two light sensors inside, that report the illuminance level on a scale of 0 - 3. Then we have the status of the two lights and the two blinds. So we can find the total state space by just multiplying each possible state for each factor: 4^3 (for the 4 possible states for the two light sensors inside, and the converted sunshine value) * 2^4 (two lights that can be either on / off and two blinds that are either up / down). Therefore we have a total state space of 1024. Each state should then have the correct value for all 7 factors. As given in the later implementation, the most intuitive way to design this is to just create an array where each element represents the value of that factor for the given state.
Actions represent what the agent can actually influence in the environment. Here we can see. That we can turn the lights on / off and we can put the blinds up / down. Since we again have 2 lights and 2 blinds, we can do a total of 8 different actions. Technically the action space would be $2^4 = 16$, but since the actions are mutually exclusive (can either turn off when it is on or turn on when it is off), we have an actual action space of 8.
The reward function, should reflect the goal of achieving the wanted light levels, meaning that once we have achieved a state that satisfies the light levels, we give a reward. In my implementation I used 100 as a reward, since it is quite a big state space and this way the paths to valid end states propagate faster. If we get to a state that is not an end state, I simply set -1 as the reward, so that if we end up at the same point again, we choose a different path and do not get a circular behaviour.
Arguments for RL in this use case:
Since the states and actions are clearly definable and reward behaviour can be sensibly applied to this use case, it seems like a good case for reinforcement learning. Since the environment is ever changing with different light levels and the requirements for the wanted light level also can change over time, but the learnings can still be applied, because the way illuminance changes when you put the blinds down is always the same, it indicates that RL is a good fit. Additionally, if there were bigger changes in the environment, the RL agent would adapt through the changing Q values.
Arguments against RL in this use case:
The state space is quite large, currently it is doable because the illuminance level is condensed into only 4 ranks, but the more accurate the illuminance level gets, the larger the state space. A problem I already see though, is in the implementation, since we have to query an http endpoint, the training can overload the endpoint. So the sheer volume is already a slight hinderance. And the general challenge with RL is finding the correct values for exploration vs exploitation, the discount factor, the learning factor and even the reward values. Finding the correct values there can be very time consuming, and costly if not implemented correctly. Alternatives to RL could be simpler rule-based systems that just react on certain

thresholds.
Handling additional constraint:
To enable our model to work with this constraint, we need to expand the state and action spaces. First we would need to add a human boolean to the state, so that we would know if a human is currently in the lab. Then we would need to modify the available actions for states where the outdoor light level is 3 and a human is present. Here we should only make these actions available, that result in a state where the blinds are down. This we we can guarantee, that if a human is present and it is too bright outside, all actions will result with the blinds being down.

Task 2 & Task 3

There is not that large of a difference between the two tasks, since all that really changes is the url that we need to pass. In my current implementation, the agent has a real(boolean) initial belief, that can be either true or false. Depending on that boolean, when the makeTDArtifact plan is executed either the learning artefact will be created or the real one. Additionally, because we cannot guarantee, that with the readProperty the key and value arrays will be in the same order for both artefacts, I used a map to get the correct values. The main pitfall that I encountered, was that I realised very late after many hours of debugging, that I forgot to set the reward to a negative value in case we are at a state that is not a goal state, because I had it set to simply 0, the RL agent kept using the same invalid state paths. Once I added the -1 for the immediate reward the training worked out great.