Gridworld

Jarod DURET
Jonathan HENO

19 janvier 2021

1 Architecture

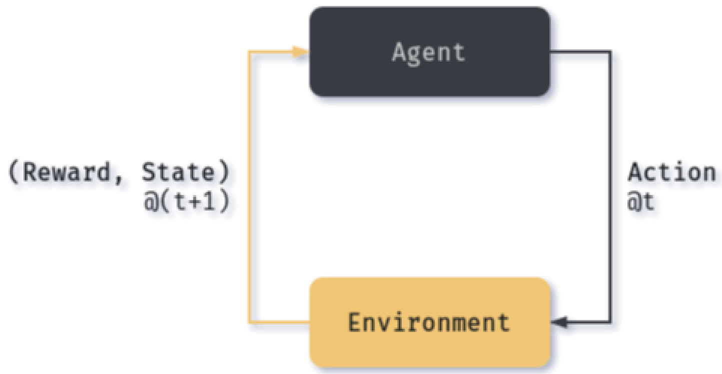

Figure 1 – Interactions between the decision maker and the environment

2 ϵ-greedy policy

There is a chance (valued at $\epsilon \in [0; 1]$) that an action initially chosen by the agent will not take effect. In our code we call it the **disobey factor**.

$$p_\pi(s, a) \triangleq \begin{cases} 1 - \epsilon & \text{if } a \in \pi(s) \\ \epsilon/(Card(A) - 1) & \text{otherwise} \end{cases} \tag{2.1}$$

It helps weighting the decision process given the fact that an action can fail and lead to another random decision, and thus to an unplanned reward. It also helps the agent to widen its field of view, to explore a wider space.

3 Markov Decision Process

The rewards progressively propagate accross the world and the optimal policy is questioned at each time step, by gauging every possible actions for each state.

INPUT : $S(\{i|i = 0..N^2\}),\ A(\{N, E, S, W\}),\ T(S, A) \in S,\ p_\pi,\ \gamma,\ \epsilon,\ \delta$
OUTPUT : $\pi^\star(S), \mathbf{v}^\star_\pi(S)$

\# Initialization
$\pi^\star_{\mathbf{0}}(S) \leftarrow (rand(A))_{s\in S}$
$\mathbf{v}^\star_{\pi^\star,\mathbf{0}}(S) \leftarrow \mathbf{0}$

FOR $t = 1$ TO t_{obs} DO
\# Policy evaluation
$$\mathbf{v}_{\pi^\star,\mathbf{t}}(S) \leftarrow \left(\sum_{s'\in T(s,A(s))} p_{\pi^\star_{\mathbf{t-1}}}(s,a) \left(r(s') + \gamma v^\star_{\pi^\star,t-1}(s') \right) \right)_{a\in\pi^\star_{\mathbf{t-1}}(S)}$$
\# Policy improvement
\# We value all actions available for each state $s \in S$
$$\pi^\star_{\mathbf{t}}(S) \leftarrow \left(\underset{a\in A}{argmax}\ \mathbf{v}_{\mathbf{a},\mathbf{t}}(s) \right)_{s\in S}$$
$$\mathbf{v}^\star_{\pi^\star,\mathbf{t}}(S) \leftarrow \left(\underset{a\in A}{max}\ \mathbf{v}_{\mathbf{a},\mathbf{t}}(s) \right)_{s\in S}$$
IF $\max_{s\in S} \left(\mathbf{v}^\star_{\pi^\star,\mathbf{t}}(s) - \mathbf{v}^\star_{\pi^\star,\mathbf{t-1}}(s) \right) \leq \delta$ THEN
RETURN $\pi^\star_{\mathbf{t}}(S), \mathbf{v}^\star_{\pi^\star,\mathbf{t}}(S)$
END IF
END FOR
RETURN $\pi^\star_{\mathbf{t_{obs}}}(S), \mathbf{v}^\star_{\pi^\star,\mathbf{t_{obs}}}(S)$

The other methods rely more on the policy than the previous one. At each time step, for a given epoch, each action taken, with respect to current policy, is weighted and learned throught environment feedbacks.

4 State Action Reward State Action (SARSA)

The key formulae of this algorithm is the following equation :

$$Q_{s_t,a_t} = Q_{s_t,a_t} + \alpha(R_{s_t,a_t} + \gamma Q_{s_{t+1},a_{t+1}} - Q_{s_t,a_t}) \tag{4.1}$$

With this in mind, in every iteration, we should generate two actions, with respect to the ongoing policy, and get the arrival state of both actions. We then update the Q matrix with the reward associated with the first action taken in the process : the optimal state is built upon the action value function, by mapping the optimal action given a fixed state.

Input : $S(\{i|i = 0..N^2\}),\ s_0 \in S,\ A(\{N, E, S, W\}),\ T(S, A) \in S,\ p_\pi,\ \gamma,\ \epsilon,\ \alpha,\ n_{u,max}$
Output : $\pi^\star(S), Q(S)$

  # Initialization
  $Q^{(0)}, \pi^\star_{\mathbf{0}}(S) \leftarrow (0)_{a\in A, s\in S}, (rand(A))_{s\in S}$
  $s_0, n_{unchanged} \leftarrow s_0, 0$

  For $t = 0$ to $(t_{obs} - 1)$ do
    # Generating first action with respect to policy $\pi^\star_{\mathbf{t}}$ at state s_t
    $a_t, s' \leftarrow$ `gen_move(`$s_t, \pi^\star_{\mathbf{t}}$`)`
    # Generating second action with respect to ongoing policy $\pi^\star_{\mathbf{t}}$ from state s_{t+1}
    $a_{t+1}, s'' \leftarrow$ `gen_move(`$s', \pi^\star_{\mathbf{t}}$`)`

    # Correcting model given the actions taken
    $Q^{(t+1)}_{s_t,a_t} \leftarrow Q^{(t)}_{s_t,a_t} + \alpha(R_{s_t,a_t} + \gamma Q^{(t)}_{s',a_{t+1}} - Q^{(t)}_{s_t,a_t})$
    # Improving policy given the update of the Q-values for state s_t
    $\pi^\star_{\mathbf{t+1}}(s_t) \leftarrow \underset{a\in A(s_t)}{argmax}\ \mathbf{Q_{s_t,a}}(s)$

    # Updating next state for next iteration
    $s_{t+1} \leftarrow s'$

    If $\pi^\star_{\mathbf{t+1}} = \pi^\star_{\mathbf{t}}$ then
      $n_{unchanged} \leftarrow n_{unchanged} + 1$
      If $n_{unchanged} = n_{u,max}$ then
        Return $\pi^\star_{\mathbf{t+1}}(S), Q^{(t+1)}$
      End If
    Else
      $n_{unchanged} \leftarrow 0$
    End If
  End For
  Return $\pi^\star_{\mathbf{t_{obs}}}(S), Q^{(t_{obs})}$

And voilà !

5 Q-learning

The logic is more or less the same than the previous, except the fact that we are valuing the best action value function in the landing state to optimize the model :

$$Q_{s_t,a_t} = Q_{s_t,a_t} + \alpha \left(R_{s_t,a_t} + \gamma \left(\max_{a \in A(s_{t+1})} Q_{s_{t+1},a} \right) - Q_{s_t,a_t} \right) \tag{5.1}$$

INPUT : $S(\{i | i = 0..N^2\}),\ s_0 \in S,\ A(\{N, E, S, W\}),\ T(S, A) \in S,\ p_\pi,\ \gamma,\ \epsilon,\ \alpha,\ n_{u,max}$
OUTPUT : $\pi^\star(S), Q(S)$

```
# Initialization
Q^(0), π*_0(S) ← (0)_{a∈A,s∈S}, (rand(A))_{s∈S}
s_0, n_unchanged ← s_0, 0

FOR t = 0 TO (t_obs − 1) DO
  # Generating first action with respect to policy π*_t at state s_t
  a_t, s' ← gen_move(s_t, π*_t)

  # Correcting model given the actions taken
  Q^(t+1)_{s_t,a_t} ← Q^(t)_{s_t,a_t} + α ( R_{s_t,a_t} + γ ( max_{a∈A(s')} Q^(t)_{s',a} ) − Q^(t)_{s_t,a_t} )
  # Improving policy given the update of the Q-values for state s_t
  π*_{t+1}(s_t) ← argmax_{a∈A(s_t)} Q_{s_t,a}(s)

  # Updating next state for next iteration
  s_{t+1} ← s'

  IF π*_{t+1} = π*_t THEN
    n_unchanged ← n_unchanged + 1
    IF n_unchanged = n_{u,max} THEN
      RETURN  π*_{t+1}(S), Q^(t+1)
    END IF
  ELSE
    n_unchanged ← 0
  END IF
END FOR
RETURN  π*_{t_obs}(S), Q^(t_obs)
```