Multi Agent Systems

- Lab 5 -

Q-Learning and SARSA

Value Iteration Recap

- **Policy** and **Value Iteration** algorithms require knowledge of **environment dynamics**

$$V^*(s) = \max_a [R(s,a) + \gamma \sum_{s'} P(s'|s,a) V^*(s')]$$

  - *P(s' | s, a)* has to be known

- In many real world problems the environment dynamics is not known before hand => the agent has to *estimate rewards* and *improve its policy* based on direct interaction with the environment

Q-Function

- Instead of a *state value* function, we are more explicit, in storing the value of executing *an action* in a *given state*

$$q^{\pi}(s,a)=E_{\pi}[R_{t+1}+\gamma R_{t+2}+\gamma^2 R_{t+3}+\ldots|S_t=s,A_t=a]=E_{\pi}\left[\sum_{\tau=t+1}\gamma^{\tau-t-1}R_{\tau}|S_t=s,A_t=a\right]$$

- Bellman equation for q-function

$$q^{\pi}(s,a)=E_{\pi}[R_{t+1}+\gamma q^{\pi}(S_{t+1},A_{t+1})|S_t=s,A_t=a]=\sum_{s'\in S}\sum_{r\in R}p(s',r|s,a)[r+\gamma q^{\pi}(s',\pi(s'))]$$

Q-Learning Algorithm

Agent learns by observing consequences of actions it takes in the environment

Q-values adjusted through **temporal differences**

Learning is **off-policy**

Learning policy is greedy

Play policy allows for exploration

procedure ε-Greedy (s, q, ε)
with prob ε: return *random(A)*
with prob 1-ε: return $\underset{a}{argmax}\, q(s, a)$
end

procedure Q-Learning (<S, A, γ>, ε)
for all s in S, a in A **do**
q(s,a) ← 0 // set initial values to 0
end for
for all episodes **do**
s ← initial state
while s not final state **do**
pick action *a* using ε-Greedy (s, q, ε)
execute *a* → get reward r and next state *s'*
$q(s, a) \leftarrow q(s, a) + \alpha(r + \gamma \max_{a'} q(s', a') - q(s, a))$
s ← s'
end while
end for
for all *s* in S **do**
$\pi(s) \leftarrow argmax_{a \text{ in } A}\, q(s, a)$
end for
return *π*

Q-Learning Algorithm

Agent learns by observing consequences of actions it takes in the environment

Q-values adjusted through **temporal differences**

Learning is **off-policy**

Learning policy is greedy

Play policy allows for exploration

procedure ε-Greedy (s, q, ε)
with prob ε: return *random(A)*
with prob 1-ε: return $\underset{a}{argmax}\, q(s,a)$
end

procedure Q-Learning (<S, A, γ>, ε)
for all s in S, a in A **do**
q(s,a) ← 0 // set initial values to 0
end for
for all episodes **do**
s ← initial state
while s not final state **do**
pick action *a* using ε-Greedy (s, q, ε)
execute *a* → get reward r and next state *s'*
$q(s, a) \leftarrow q(s, a) + \alpha(r + \gamma \max_{a'} q(s',a') - q(s, a))$
s ← s'
end while
end for
for all *s* in S **do**
$\pi(s) \leftarrow argmax_{a \text{ in } A}\, q(s, a)$
end for
return *π*

Q-Learning Algorithm

Agent learns by observing consequences of actions it takes in the environment

Q-values adjusted through **temporal differences**

Learning is **off-policy**

Learning policy is greedy

Play policy allows for exploration

procedure ε-Greedy (s, q, ε)
with prob ε: return *random(A)*
with prob 1-ε: return $\underset{a}{argmax}\, q(s,a)$
end

procedure Q-Learning (<S, A, γ>, ε)
for all s in S, a in A **do**
q(s,a) ← 0 // set initial values to 0
end for
for all episodes **do**
s ← initial state
while s not final state **do**
pick action *a* using ε-Greedy (s, q, ε)
execute *a* → get reward r and next state *s'*
$q(s,a) \leftarrow q(s,a) + \alpha(\boldsymbol{r + \gamma \max_{a'} q(s',a') - q(s,a)})$
s ← s'
end while
end for
for all *s* in S **do**
$\pi(s) \leftarrow argmax_{a \in A}\, q(s,a)$
end for
return *π*

Q-Learning Algorithm

Agent learns by observing consequences of actions it takes in the environment

Q-values adjusted through **temporal differences**

Learning is **off-policy**

- Learning policy is greedy
- Play policy allows for exploration

procedure ε-Greedy (s, q, ε)
    with prob ε: return *random(A)*
    with prob 1-ε: return $\underset{a}{argmax}\, q(s,a)$
end

procedure Q-Learning (<S, A, γ>, ε)
    for all s in S, a in A **do**
        q(s,a) ← 0   // set initial values to 0
    end for
    for all episodes **do**
        s ← initial state
        while s not final state **do**
            pick action *a* using ε-Greedy (s, q, ε)
            execute *a* → get reward r and next state *s'*
            $q(s,a) \leftarrow q(s,a) + \alpha(r + \gamma \max_{a'} q(s',a') - q(s,a))$
            s ← s'
        end while
    end for
    for all *s* in S **do**
        $\pi(s) \leftarrow argmax_{a \text{ in } A}\, q(s,a)$
    end for
    return *π*

SARSA Algorithm

Agent learns by observing consequences of actions it takes in the environment

Q-values adjusted through **temporal differences**

Learning is **on-policy**

Action used to play/explore = action used for updating the q-values

procedure ε-Greedy (s, q, ε)
    with prob ε: return *random(A)*
    with prob 1-ε: return $\underset{a}{argmax}\, q(s,a)$
end

procedure SARSA (<S, A, γ>, ε**)**
    for all s in S, a in A **do**
        q(s,a) ← 0   // set initial values to 0
    end for
    for all episodes **do**
        s ← initial state
        pick action *a* from *s* using ε-Greedy (s, q, ε)
        while s not final state **do**
            execute *a* → get reward r and next state *s'*
            Pick action a' from s' using ε-Greedy (s', q, ε)
            q(s, a) ← q(s, a) + α(r + γ q(s',a') - q(s, a))
            s ← s', a ← a'
        end while
    end for
    for all *s* in S **do**
        $\pi(s) \leftarrow argmax_{a\ in\ A}\ q(s, a)$
    end for
    return *π*

OpenAI Gym Environments

- Remember the Taxi-v3 and FrozenLake-v1 environment in OpenAI Gymnasium:

- **Task**
  - Implement a Q-Learning agent and a SARSA agent for these environments
  - Create the **reward per training epoch** plot for both **Q-Learning** and **SARSA**
    - Plot the reward evolution for Q-Learning and SARSA **on the same diagram**
    - Every X (e.g. 50, 100) epochs do an **evaluation run** (run the *currently learned policy* for 50 epochs and report the *average reward*)
  - Compare the convergence speed and highest reward metrics of each algorithm under **different hyperparameter settings (see next slide)**

OpenAI Gym Environments

- **Task (details)**
  - **Vary the main parameters influencing learning:**
    - γ – 0.5, 0.9
    - ε – 0.1, 0.5, 0.8
    - α (lr) – 0.1, 0.5, 0.9
  - **Analyse the results by:**
    - **Keeping two parameters constant** (e.g. γ=0.9, ε – 0.1) and **varying the third;**
    - **plot all variations of a parameter on the same graph** (e.g. all variations of **α** are displayed on the same graph to better observe the influence of that single parameter)