Reinforcement Learning Pseudo Code

Wouter Deketelaere

September 23, 2022

1 Agent Learning Algorithm

Algorithm 1 Agent Learning Algorithm

```
strategy ← LearningStrategy                                   ▷ strategy = Learning Algorithm
environment ← Environment                                     ▷ OpenAI Gym Environment

function Learn(n_episodes)                                    ▷ maximum number of episodes
    while episode_count < n_episodes do
        episode ← Episode()                                   ▷ Create a new Episode
        state ← environment.state                             ▷ Where is the Agent?
        while episode not done do
            action ← strategy.next_action                     ▷ The Agent chooses its next action
            percept ← environment.step(action)                ▷ The Agent ends up in new state
            episode.add(percept)                              ▷ Add Percept to Episode
            strategy.learn(episode)                           ▷ The Agent learns from Percepts in the Episode
            state = percept.next_state
        end while
    end while
end function
```

2 Learning Strategy

Algorithm 2 LearningStrategy

```
λ                                 ▷ exponential decay rate used for exploration/exploitation
ε                                 ▷ (decaying) probability of selecting random action
ε_max = 1.0                       ▷ exploration probability at start
ε_min = 0.01                      ▷ minimum exploration probability

function Learn(episode: Episode)  ▷ abstract method
    Subclass implementation       ▷ here we insert the actual learning algorithm
    t ← t + 1                     ▷ increase episode time step
    τ ← τ + 1                     ▷ increase overall time step
end function
```

3 Tabular Learner

Algorithm 3 TabularLearner implements LearningStrategy

α ▷ learning rate
$\boldsymbol{\pi}$ ▷ 2D Numpy array holding the $\pi(s)$ values
$\mathbf{v}$ ▷ 1D Numpy array holding $v(s)$-values
$\mathbf{q}$ ▷ 2D Numpy array holding all $q(s,a)$-values

function INIT( )
  $\boldsymbol{\pi} \leftarrow$ uniform distributed policy table
  $\forall s : v(s) \leftarrow 0$
  $\forall s, a : q(s,a) \leftarrow 0$ ▷ $q(s,a)$ is the q-value of state s and action a
end function

function LEARN(episode: Episode)
  (SUBCLASS IMPLEMENTATION) ▷ subclasses insert their own code at this point
  EVALUATE( ) ▷ evaluate policy
  IMPROVE( ) ▷ improve policy
end function

function NEXTACTION( ) **return** action
  $a = sample(\pi(s))$ ▷ use Python's random.choice with p-parameter and add tie-breaking
  return a
end function

function EVALUATE( )
  for each $s \in \mathcal{S}$ **do** ▷ $v(s)$ is equal to the maximum $q(s,a)$-value in state s
    $v(s) \leftarrow max_a \; \mathbf{q(s)}$ ▷ $\mathbf{q(s)}$ is a 1D Numpy array holding all $q(s,a)$-values for state s
  end for
end function

function IMPROVE
  for each $s \in \mathcal{S}$ **do**
    $$a_* = \arg\max_a \mathbf{q(s)}$$ ▷ Numpys' argmax doesn't do tie-breaking, add this yourself
    $$\pi(a|s) = \begin{cases} 1 - \varepsilon + \frac{\varepsilon}{|A(s)|}, & \text{if a} = a_* \\ \frac{\varepsilon}{|A(s)|}, & \text{if a} \neq a_* \end{cases}$$ ▷ $|A(s)|$ is the number of actions in state s
  end for
  $\varepsilon(\tau) = \varepsilon_{min} + (\varepsilon_{max} - \varepsilon_{min}) \cdot e^{-\lambda \cdot \tau}$ ▷ τ increases only after the episode ends
end function

4 Q-learning

Algorithm 4 Q-Learning implements TabularLearner

```
function LEARN(episode: Episode)
    p ← last Percept in Episode                                         ▷ s = p.s en a = p.a
    q(s,a) ← q(s,a) + α · [r(s,a) + γ · max_a'(q(s',a')) − q(s,a)]      ▷ Update rule
                                   ▷ max_a' = maximum q-value over the next action a'
                                          ▷ r(s,a) = reward for taking action a in state s
    SUPER( )                       ▷ now call EVALUATE and IMPROVE in the superclass

end function
```

5 N-step Q-learning

Algorithm 5 N-Step Q-Learning implements TabularLearner

```
N                                                              ▷ N = number of steps

function LEARN(episode: Episode)

    if |ℰ| ≥ N then                       ▷ Do we have enough Percepts in the Episode ℰ?
        for each p ∈ 𝒫_N do                             ▷ P_N = N last percepts in Episode

            q(s,a) ← q(s,a) − α · (q(s,a) − [r(s,a) + γ · max_a'(q(s',a'))])   ▷ Update rule
                                                                 ▷ s = p.s en a = p.a
        end for
    end if
    SUPER( )                       ▷ now call EVALUATE and IMPROVE in the superclass
end function
```

6 Monte Carlo

Monte Carlo is equal to N-step Q-learning with N = length of the entire episode. By subclassing N-step Q-learning, you can reuse the code for N-step Q-learning.

7 Deep Q-learning (DQN)

Algorithm 6 Deep Q-Learning implements LearningStrategy

```
batch_size                                      ▷ batch size used by NNs

Q1(θ1) ← Keras NN Model      ▷ Q1 = Neural Network Model with weights θ1
Q2(θ2) ← Keras NN Model      ▷ Q2 = Neural Network Model with weights θ2

C                        ▷ Ring counter value for updating weights of Q2
function LEARN(episode: Episode)

    if |ℰ| ≥ batch_size then           ▷ Enough percepts in Episode ℰ?
        𝒫 ← ℰ.sample()            ▷ Sample random Percepts from Episode ℰ
        LEARNFROMBATCH(𝒫)
    end if
end function

function LEARNFROMBATCH(𝒫)
    count ← 0

    𝒟 =BUILDTRAININGSET(𝒫)

    TRAINNETWORK(𝒟)

    count ← count + 1
    if count mod C then   ▷ Copy weights θ1 from Q1 into θ2 of Q2 every C times
        θ2 = θ1
    end if
end function

function BUILDTRAININGSET(𝒫) return 𝒟
    𝒟                                        ▷ 𝒟 = empty trainingsset
    for each p ∈ 𝒫 do                        ▷ P = sample of Percepts
        q(s) = Q1(s; θ1)   ▷ Predict q-values (numpy array) for state s (numpy array)
        q* = max(Q2(s'; θ2))           ▷ q* = max q-waarde according to Q2
        q(s,a) = { r(s,a),              if episode done
                 { r(s,a) + γ · q*,     otherwise          ▷ q(s,a) ≠ array
        𝒟.append(s, q(s,a))
    end for
    return 𝒟
end function

function TRAINNETWORK(𝒟)
    for each (s, q(s,a) ∈ 𝒟 do
        θ1 = θ1 + ∇θ1 Q1(θ1)   ▷ Train NN Q1 on (s, q(s,a)) using fit-methode
    end for
end function
```

8 Double Deep Q-learning (DDQN)

Algorithm 7 Deep Q-Learning

```
batch_size                                          ▷ batch size voor de NNs

Q_1(θ_1) ← Keras NN Model                           ▷ Q_1 = Neural Network Model with weights θ_1
Q_2(θ_2) ← Keras NN Model                           ▷ Q_2 = Neural Network Model with weights θ_2

C                                                   ▷ Ring counter value for updating weights of Q_2
function LEARN(episode: Episode)

    if |E| ≥ batch_size then                        ▷ Enough percepts in Episode E?
        P ← E.sample()                              ▷ Sample random Percepts from Episode E
        LEARNFROMBATCH(P)
    end if
end function

function LEARNFROMBATCH(P)
    count ← 0

    D =BUILDTRAININGSET(P)

    TRAINNETWORK(D)

    count ← count + 1
    if count mod C then                             ▷ Copy weights θ_1 from Q_1 into θ_2 of Q_2 every C times
        θ_2 = θ_1
    end if
end function

function BUILDTRAININGSET(P) return D
    D                                               ▷ D = empty trainingsset
    for each p ∈ P do                               ▷ P = sample of percepts
        q(s) = Q_1(s; θ_1)                          ▷ Predict q-values (numpy array) for state s
        a_* = arg max_a Q_1(s'; θ_1)                ▷ Best action a_* as per Q_1
        q_* = Q_2(s'; θ_2)[a_*]                     ▷ q-value q_* as per Q_2 for that best actie a_*
        q(s, a) = { r(s, a),            if episode done
                  { r(s, a) + γ · q_*,  otherwise   ▷ q(s,a) ≠ array
        D.append(s, q(s, a))
    end for
    return D
end function

function TRAINNETWORK(D)
    for each (s, q(s, a) ∈ D do
        θ_1 = θ_1 + ∇_{θ_1} Q_1(θ_1)                ▷ Train NN Q_1 on (s, q(s, a)) using fit-method
    end for
end function
```

9 Next action voor (D)DQN

Algorithm 8 Next Action (D)DQN

function NEXTACTION( ) **return** action

$$a = \begin{cases} sample(A(s)), & \text{random ¡}\varepsilon \\ \arg\max_a Q_1(s;\theta_1), & \text{otherwise} \end{cases}$$ ▷ $Q_1(s;\theta_1)$ = predict q-values using Q_1

return a

end function