بسم الله الرحمن الرحیم

Deep Learning

Mohammad Reza Mohammadi

2021

Off-policy vs On-policy

- Off-policy: using a different policy for acting and updating
- On-policy: using the same policy for acting and updating

Algorithm 14: Sarsamax (Q-Learning)

Input: policy π, positive integer *num_episodes*, small positive fraction α, GLIE $\{\epsilon_i\}$
Output: value function Q ($\approx q_\pi$ if *num_episodes* is large enough)
Initialize Q arbitrarily (e.g., $Q(s,a) = 0$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$, and $Q(\textit{terminal-state}, \cdot) = 0$)
for $i \leftarrow 1$ ***to*** *num_episodes* **do**
  $\epsilon \leftarrow \epsilon_i$
  Observe S_0
  $t \leftarrow 0$
  repeat
    Choose action A_t using policy derived from Q (e.g., ϵ-greedy)
    Take action A_t and observe R_{t+1}, S_{t+1}
    $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t))$
    $t \leftarrow t + 1$
  until S_t *is terminal*;
end
return Q

Algorithm 13: Sarsa

Input: policy π, positive integer *num_episodes*, small positive fraction α, GLIE $\{\epsilon_i\}$
Output: value function Q ($\approx q_\pi$ if *num_episodes* is large enough)
Initialize Q arbitrarily (e.g., $Q(s,a) = 0$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$, and $Q(\textit{terminal-state}, \cdot) = 0$)
for $i \leftarrow 1$ ***to*** *num_episodes* **do**
  $\epsilon \leftarrow \epsilon_i$
  Observe S_0
  Choose action A_0 using policy derived from Q (e.g., ϵ-greedy)
  $t \leftarrow 0$
  repeat
    Take action A_t and observe R_{t+1}, S_{t+1}
    Choose action A_{t+1} using policy derived from Q (e.g., ϵ-greedy)
    $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t))$
    $t \leftarrow t + 1$
  until S_t *is terminal*;
end
return Q

Example

- Always start at the same starting point
- The goal is to eat the big pile of cheese at the bottom right-hand corner, and avoid the poison
- The episode ends if we eat the poison, eat the big pile of cheese or if we spent more than 5 steps
- The learning rate is 0.1
- The gamma (discount rate) is 0.99

Example

- The reward function goes like this:
  - 0: Going to a state with no cheese in it
  - +1: Going to a state with a small cheese in it
  - +10: Going to the state with the big pile of cheese
  - -10: Going to the state with the poison and thus die

Example

- Initialize the Q-Table

	←	→	↑	↓
	0	0	0	0
	0	0	0	0
	0	0	0	0
	0	0	0	0
	0	0	0	0
	0	0	0	0

Algorithm 14: Sarsamax (Q-Learning)

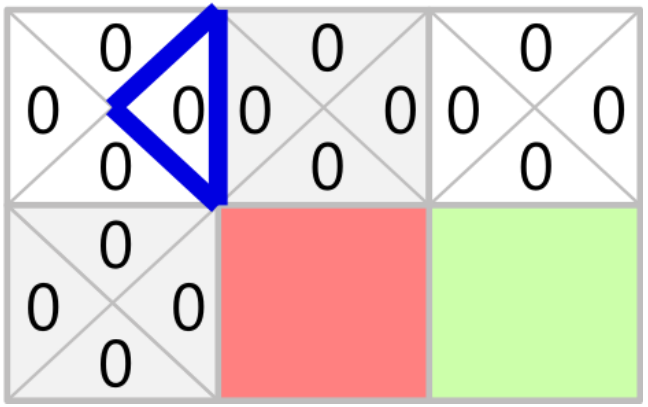
Input: policy π, positive integer *num_episodes*, small positive fraction α, GLIE $\{\epsilon_i\}$
Output: value function Q ($\approx q_\pi$ if *num_episodes* is large enough)
Initialize Q arbitrarily (e.g., $Q(s,a) = 0$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$, and $Q(\textit{terminal-state}, \cdot) = 0$)
for $i \leftarrow 1$ *to num_episodes* **do**
    $\epsilon \leftarrow \epsilon_i$
    Observe S_0
    $t \leftarrow 0$
    repeat
        Choose action A_t using policy derived from Q (e.g., ϵ-greedy)
        Take action A_t and observe R_{t+1}, S_{t+1}
        $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t))$
        $t \leftarrow t + 1$
    until S_t *is terminal*;
end
return Q

Example

- Choose action
  - Because epsilon is big = 1.0, I take a random action, in this case I go right

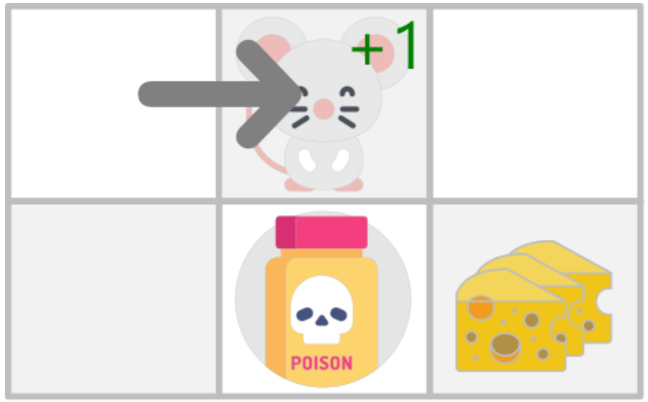
Algorithm 14: Sarsamax (Q-Learning)

Input: policy π, positive integer *num_episodes*, small positive fraction α, GLIE $\{\epsilon_i\}$
Output: value function Q ($\approx q_\pi$ if *num_episodes* is large enough)
Initialize Q arbitrarily (e.g., $Q(s,a) = 0$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$, and $Q(\textit{terminal-state}, \cdot) = 0$)
for $i \leftarrow 1$ ***to*** *num_episodes* **do**
  $\epsilon \leftarrow \epsilon_i$
  Observe S_0
  $t \leftarrow 0$
  repeat
    Choose action A_t using policy derived from Q (e.g., ϵ-greedy)
    Take action A_t and observe R_{t+1}, S_{t+1}
    $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t))$
    $t \leftarrow t + 1$
  until S_t *is terminal*;
end
return Q



Example

- Perform action



Algorithm 14: Sarsamax (Q-Learning)

Input: policy π, positive integer *num_episodes*, small positive fraction α, GLIE $\{\epsilon_i\}$
Output: value function Q ($\approx q_\pi$ if *num_episodes* is large enough)
Initialize Q arbitrarily (e.g., $Q(s,a) = 0$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$, and $Q(\textit{terminal-state}, \cdot) = 0$)
for $i \leftarrow 1$ ***to*** *num_episodes* **do**
    $\epsilon \leftarrow \epsilon_i$
    Observe S_0
    $t \leftarrow 0$
    repeat
        Choose action A_t using policy derived from Q (e.g., ϵ-greedy)
        $\boxed{\text{Take action } A_t \text{ and observe } R_{t+1}, S_{t+1}}$
        $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t))$
        $t \leftarrow t + 1$
    until S_t *is terminal*;
end
return Q

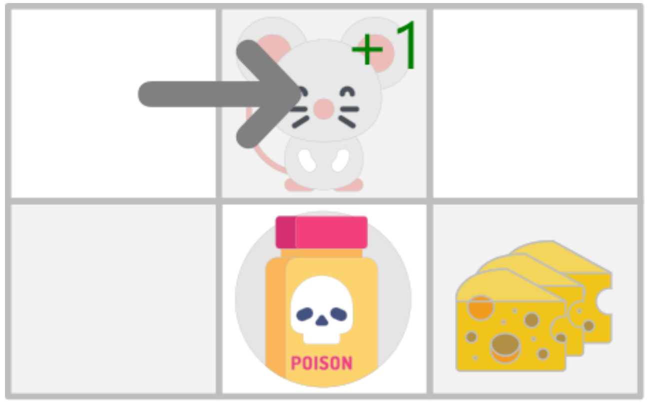
Example

- Update $Q(S_t, A_t)$
  - $Q(\text{State1}, \text{Right}) = 0 + 0.1 \times [1 + 0.99 \times 0 - 0] = 0.1$

	←	→	↑	↓
	0	0.1	0	0
	0	0	0	0
	0	0	0	0
	0	0	0	0
	0	0	0	0
	0	0	0	0

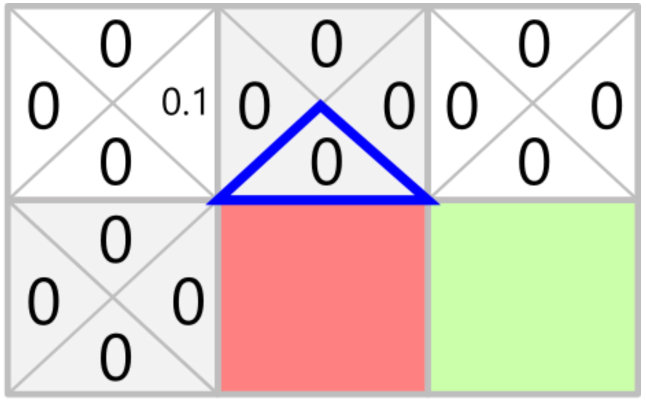
Algorithm 14: Sarsamax (Q-Learning)

Input: policy π, positive integer *num_episodes*, small positive fraction α, GLIE $\{\epsilon_i\}$
Output: value function Q ($\approx q_\pi$ if *num_episodes* is large enough)
Initialize Q arbitrarily (e.g., $Q(s,a) = 0$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$, and $Q(\textit{terminal-state}, \cdot) = 0$)
for $i \leftarrow 1$ ***to*** *num_episodes* **do**
  $\epsilon \leftarrow \epsilon_i$
  Observe S_0
  $t \leftarrow 0$
  repeat
    Choose action A_t using policy derived from Q (e.g., ϵ-greedy)
    Take action A_t and observe R_{t+1}, S_{t+1}
    $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t))$
    $t \leftarrow t + 1$
  until S_t *is terminal*;
end
return Q



Example

- Choose action
  - I take again a random action, since epsilon is really big 0.99



Algorithm 14: Sarsamax (Q-Learning)

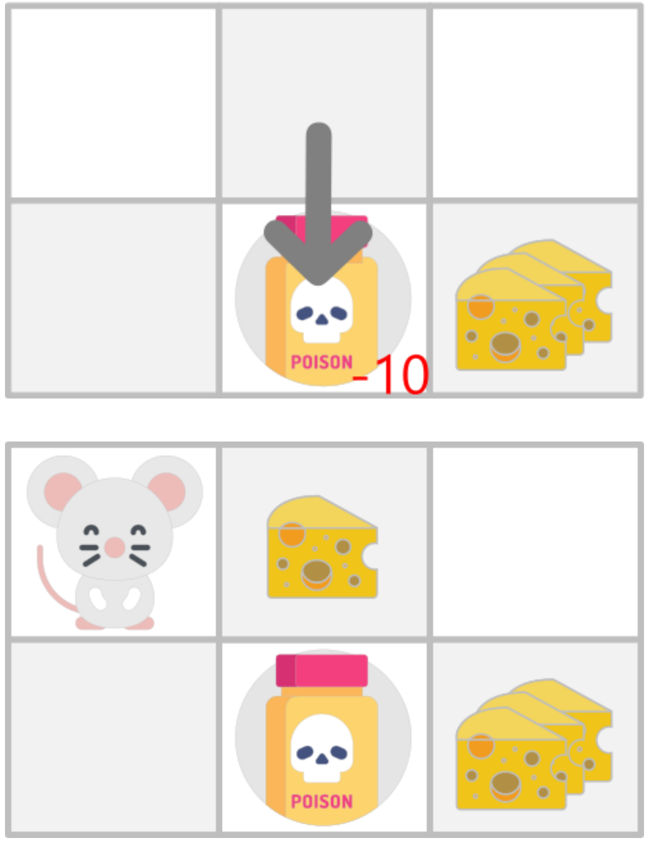
Input: policy π, positive integer *num_episodes*, small positive fraction α, GLIE $\{\epsilon_i\}$
Output: value function Q ($\approx q_\pi$ if *num_episodes* is large enough)
Initialize Q arbitrarily (e.g., $Q(s,a) = 0$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$, and $Q(\textit{terminal-state}, \cdot) = 0$)
for $i \leftarrow 1$ ***to*** *num_episodes* **do**
    $\epsilon \leftarrow \epsilon_i$
    Observe S_0
    $t \leftarrow 0$
    repeat
        Choose action A_t using policy derived from Q (e.g., ϵ-greedy)
        Take action A_t and observe R_{t+1}, S_{t+1}
        $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t))$
        $t \leftarrow t + 1$
    until S_t *is terminal*;
end
return Q

Example

- Perform action

Algorithm 14: Sarsamax (Q-Learning)

Input: policy π, positive integer $num\_episodes$, small positive fraction α, GLIE $\{\epsilon_i\}$
Output: value function Q ($\approx q_\pi$ if $num\_episodes$ is large enough)
Initialize Q arbitrarily (e.g., $Q(s,a) = 0$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$, and $Q(terminal\text{-}state, \cdot) = 0$)
for $i \leftarrow 1$ ***to*** $num\_episodes$ **do**
    $\epsilon \leftarrow \epsilon_i$
    Observe S_0
    $t \leftarrow 0$
    repeat
        Choose action A_t using policy derived from Q (e.g., ϵ-greedy)
        Take action A_t and observe R_{t+1}, S_{t+1}
        $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t))$
        $t \leftarrow t + 1$
    until S_t *is terminal*;
end
return Q



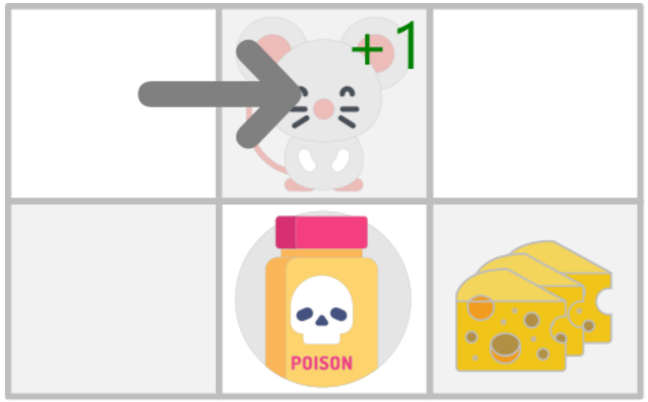
Example

- Update $Q(S_t, A_t)$
  - $Q(\text{State2}, \text{Down}) = 0 + 0.1 \times [-10 + 0.99 \times 0 - 0]$
    $= -1$

	←	→	↑	↓
	0	0.1	0	0
	0	0	0	-1
	0	0	0	0
	0	0	0	0
	0	0	0	0
	0	0	0	0

Algorithm 14: Sarsamax (Q-Learning)

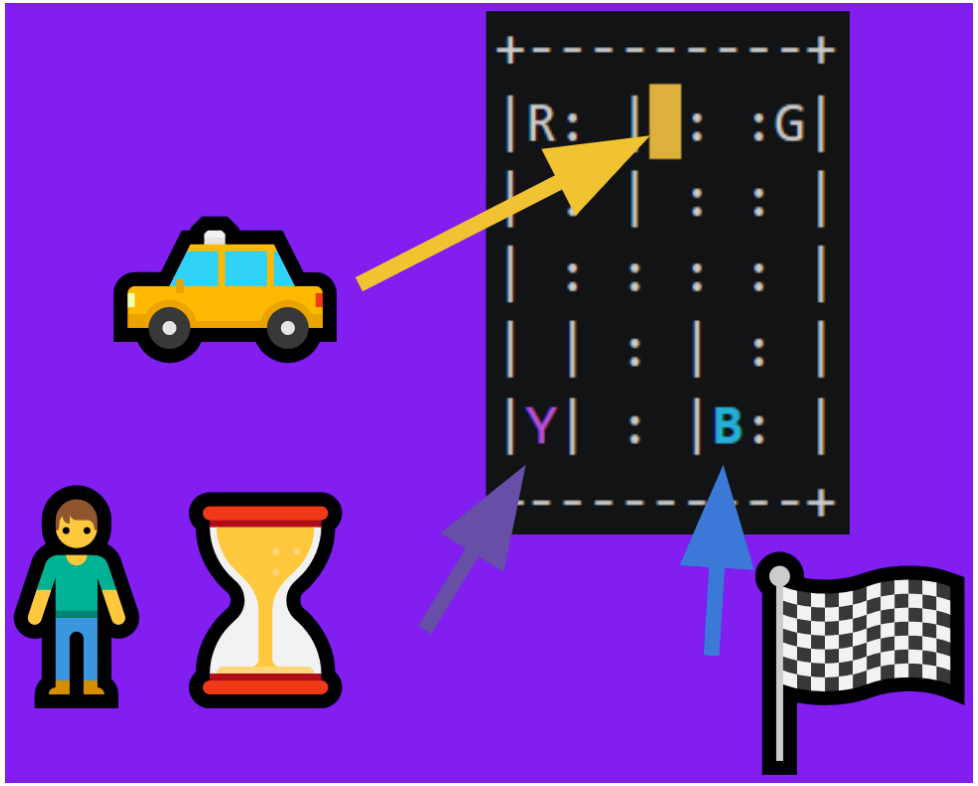
Input: policy π, positive integer *num_episodes*, small positive fraction α, GLIE $\{\epsilon_i\}$
Output: value function Q ($\approx q_\pi$ if *num_episodes* is large enough)
Initialize Q arbitrarily (e.g., $Q(s,a) = 0$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$, and $Q(\textit{terminal-state}, \cdot) = 0$)
for $i \leftarrow 1$ ***to*** *num_episodes* **do**
    $\epsilon \leftarrow \epsilon_i$
    Observe S_0
    $t \leftarrow 0$
    repeat
        Choose action A_t using policy derived from Q (e.g., ϵ-greedy)
        Take action A_t and observe R_{t+1}, S_{t+1}
        $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t))$
        $t \leftarrow t + 1$
    until S_t *is terminal*;
end
return Q



Q-Learning - Taxi agent

- The goal here is to train a taxi agent to navigate in this city to transport its passengers from point 1 to point 2

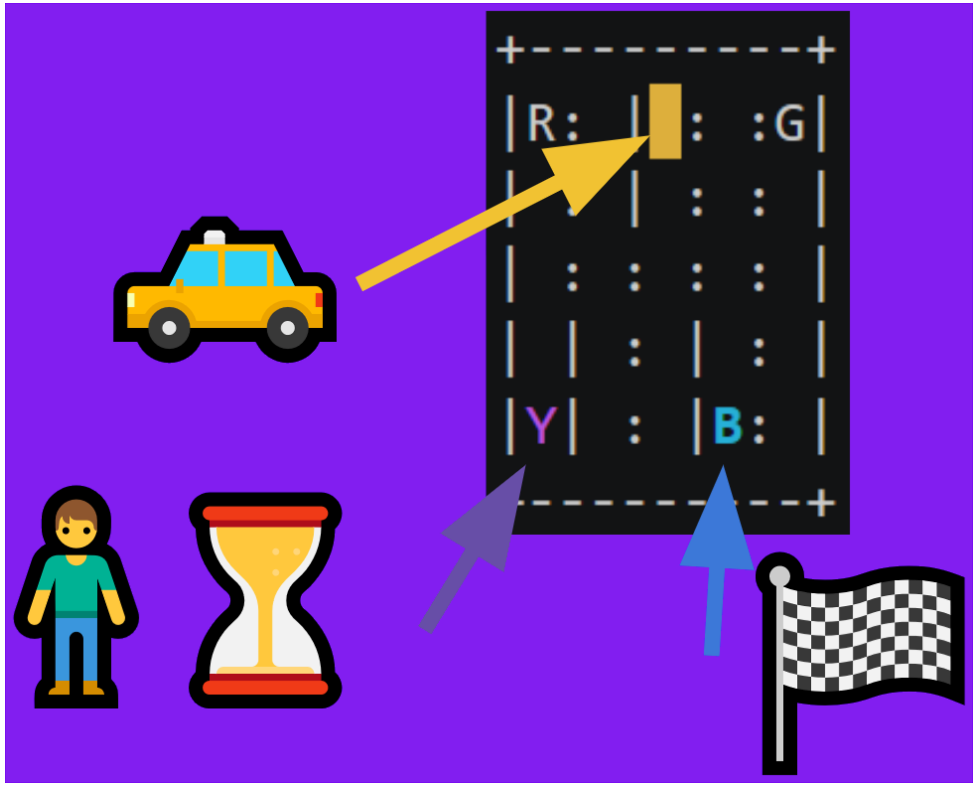
- Discrete state space (500):
  - 25 squares (5 × 5 grid world)
  - 5 different locations for the passenger
    - R, G, B, Y, or in the taxi
  - 4 destinations
    - R, G, B, Y



Q-Learning - Taxi agent

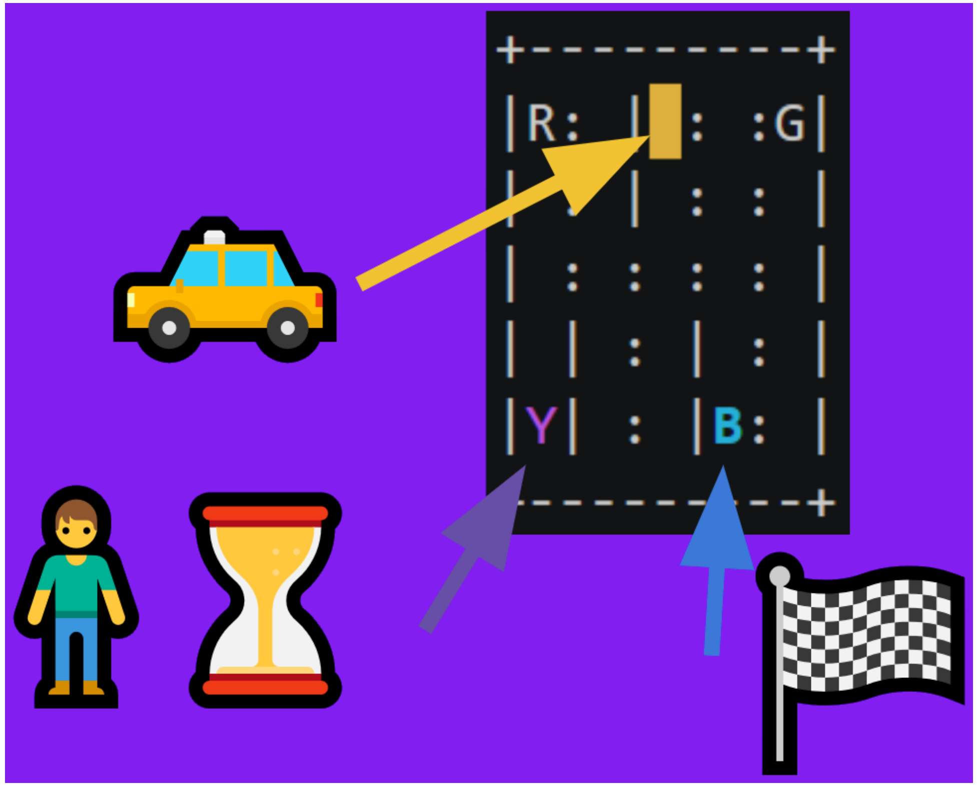
- Your task is to pick up the passenger at one location and drop him off in its desired location (selected randomly)
  - Get the passenger
  - Deliver him to the destination

- Discrete action space:
  - 4 directions (N, S, W, E)
  - Pickup
  - Put down



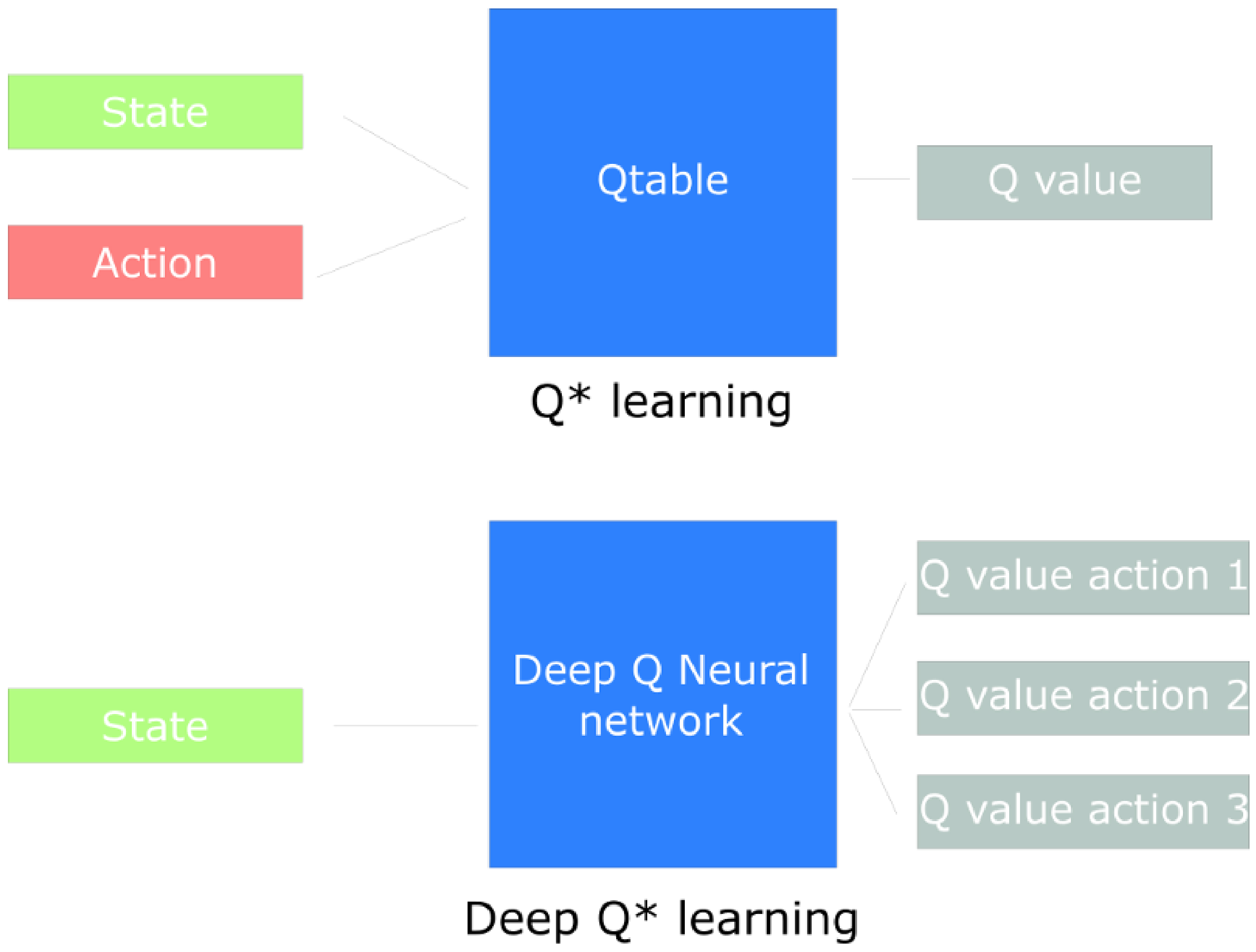
Q-Learning - Taxi agent

- The reward system:
  - -1 for each time step
  - +20 for successfully deliver the passenger
  - -10 if put down or pickup outside of the passenger or destination location



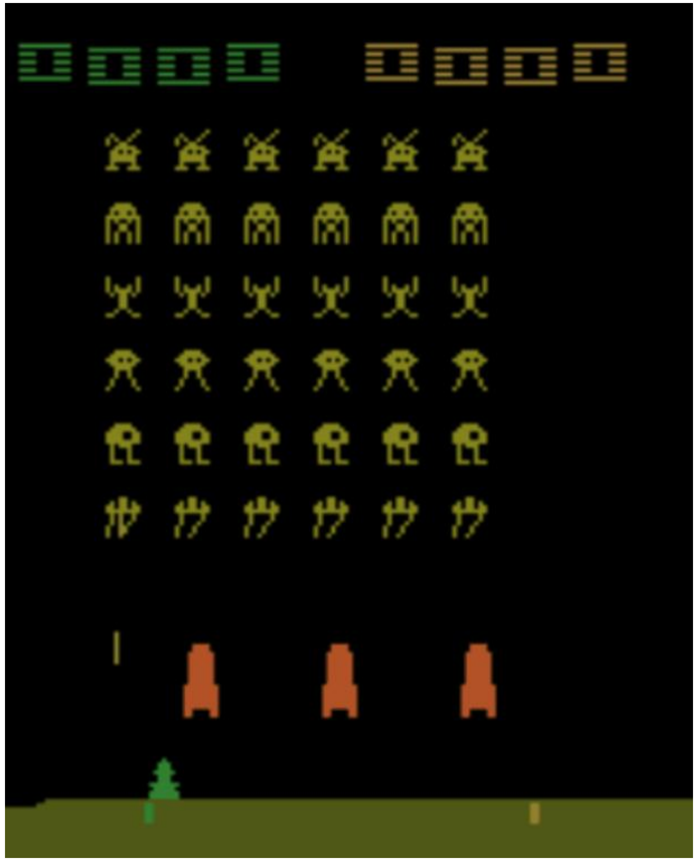
Deep Q-Learning

- Producing and updating a Q-table can become ineffective in big state space environments
- Instead of using a Q-table, we'll implement a Neural Network that takes a state and approximates Q-values for each action based on that state



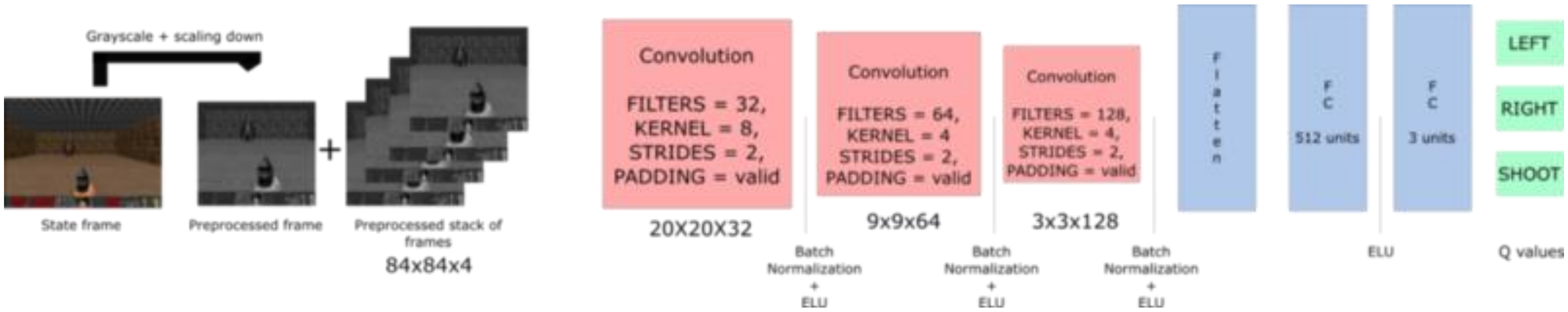
Deep Q-Learning

- We'll create an agent that learns to play Space Invaders
  - is a big environment with a gigantic state space
- Creating and updating a Q-table for that environment would not be efficient at all
- Create a neural network that will approximate, given a state, the different Q-values for each action



Deep Q-Learning

- This will be the architecture of our Deep Q Learning
- Our Deep Q Neural Network takes a stack of four frames as an input
- These pass through its network, and output a vector of Q-values for each action possible in the given state
- We need to take the biggest Q-value of this vector



Preprocessing part

- We want to reduce the complexity of our states to reduce the computation time needed for training
  - First, we can grayscale each of our states
  - Then, we crop the frame
  - Then, we reduce the size of the frame
  - And stack four sub-frames together