
بسم الله الرحمن الرحیم

# Deep Learning

Mohammad Reza Mohammadi

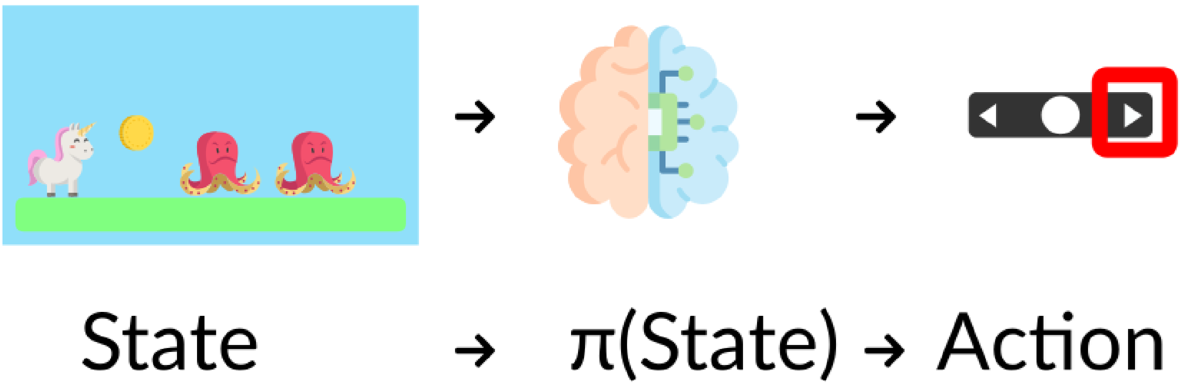
2021

# Policy

- There are two approaches to train our agent to find this optimal policy $\pi^*$:
  - Policy-based methods
    - Directly, teach the agent to learn which action to take, given the state is in
  - Value-based methods
    - Indirectly, teach the agent to learn which state is more valuable and then take the action that leads to the more valuable states

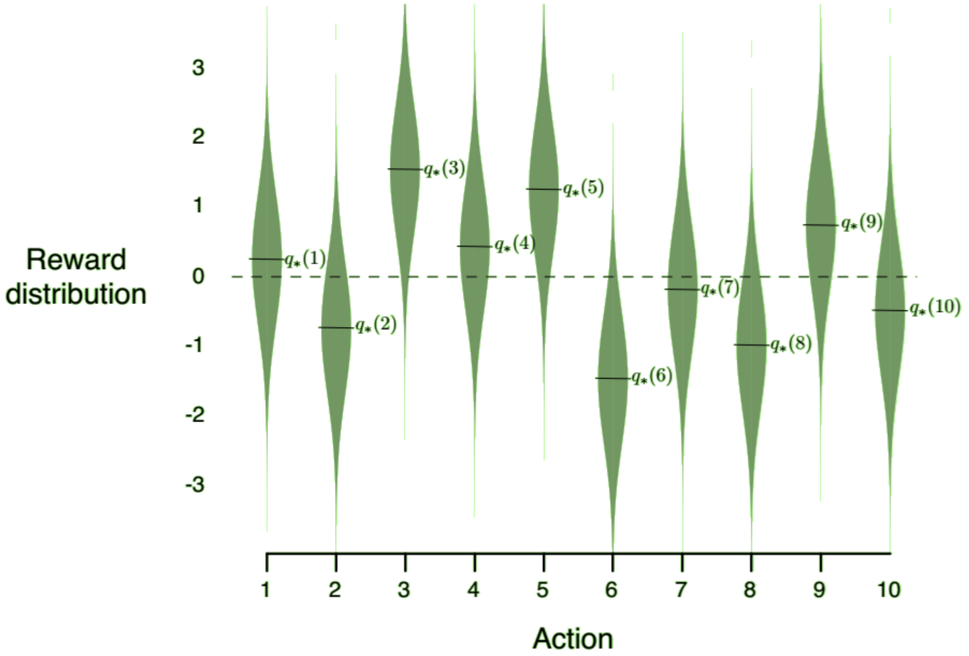
- The link between Value and Policy:

$$\pi^*(s) = \arg\max_a Q^*(s,a)$$



# $\varepsilon$-greedy policy

- In this method agent updates its initial estimates of actions on the basis of received rewards and balances exploration and exploitation by choosing exploratory action with ∊probability and optimal action rest of the time




Initialize, for $a = 1$ to $k$:

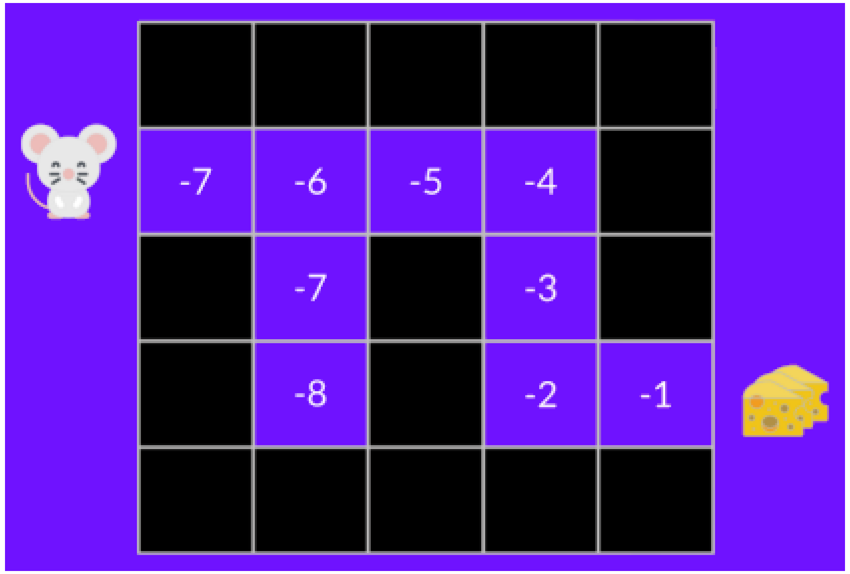
$$Q(a) \leftarrow 0$$
$$N(a) \leftarrow 0$$

Repeat forever:

$$A \leftarrow \begin{cases} \arg\max_a Q(a) & \text{with probability } 1-\varepsilon \quad \text{(breaking ties randomly)} \\ \text{a random action} & \text{with probability } \varepsilon \end{cases}$$
$$R \leftarrow bandit(A)$$
$$N(A) \leftarrow N(A) + 1$$
$$Q(A) \leftarrow Q(A) + \frac{1}{N(A)}\left[R - Q(A)\right]$$

# State-Value function

- The state value function under a policy $\pi$
- For each state, the state-value function outputs the expected return if the agent starts at that state, and then follow the policy forever after

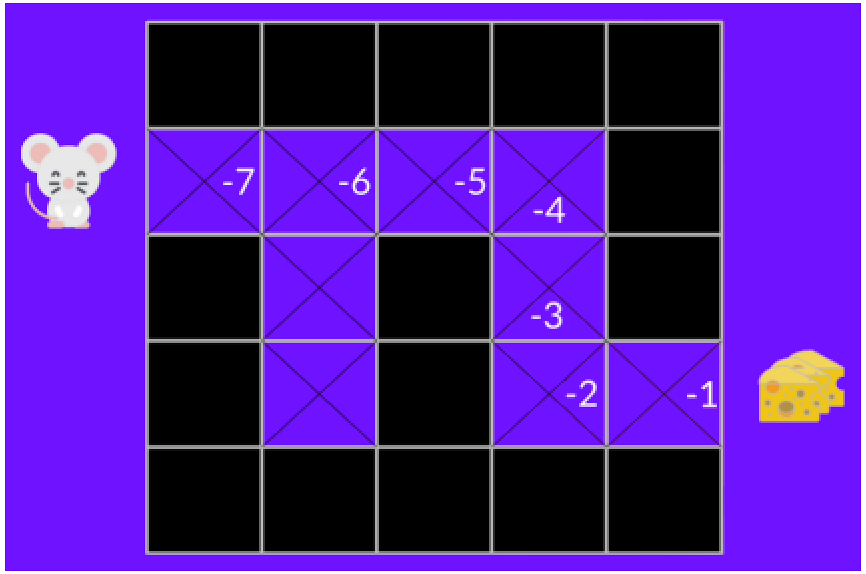



$$\underbrace{V_\pi(s)}_{\text{Value of state s}} = \underbrace{\mathbf{E}_{\boxed{\pi}}[G_t}_{\text{Expected return}} | \underbrace{S_t = s}_{\text{If the agent starts at state s}}]$$

And uses the policy to choose its actions for all time steps

# Action-Value function

- In the action-value function, for each state and action pair, the action-value function outputs the expected return, if the agent starts in that state and takes the action, and then follows the policy forever after
- The value of taking action a in state s under a policy $\pi$ is:

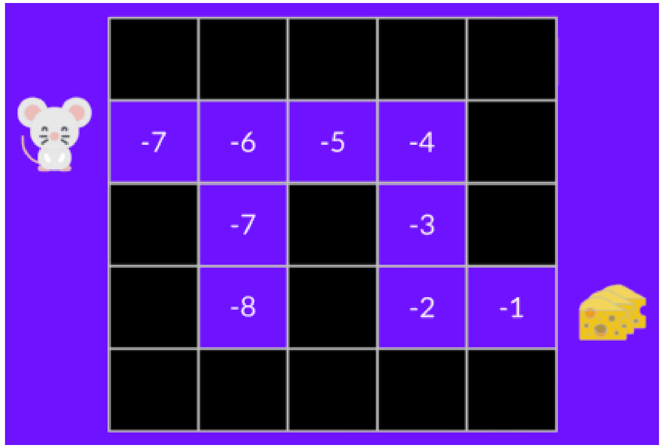


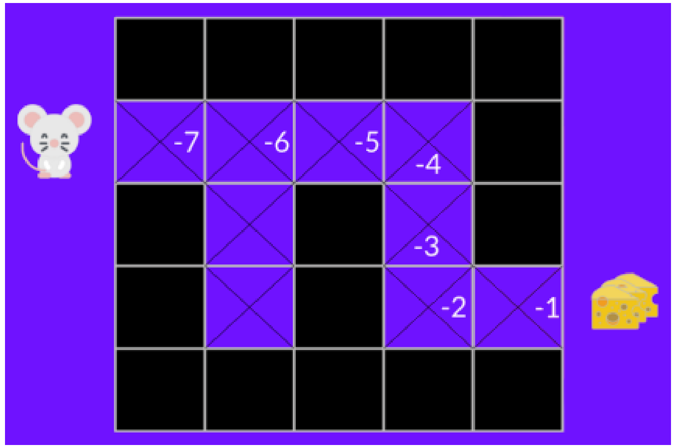

$$\underbrace{Q_\pi(s,a)}_{\text{Value of state-action pair s,a}} = \underbrace{\mathbf{E}_{\boxed{\pi}}[G_t}_{\text{Expected return}} | \underbrace{S_t = s,}_{\text{If the agent starts at state s}} \underbrace{A_t = a]}_{\text{and chooses action a}}$$

And then uses the policy to choose its actions for all time steps

# Value functions

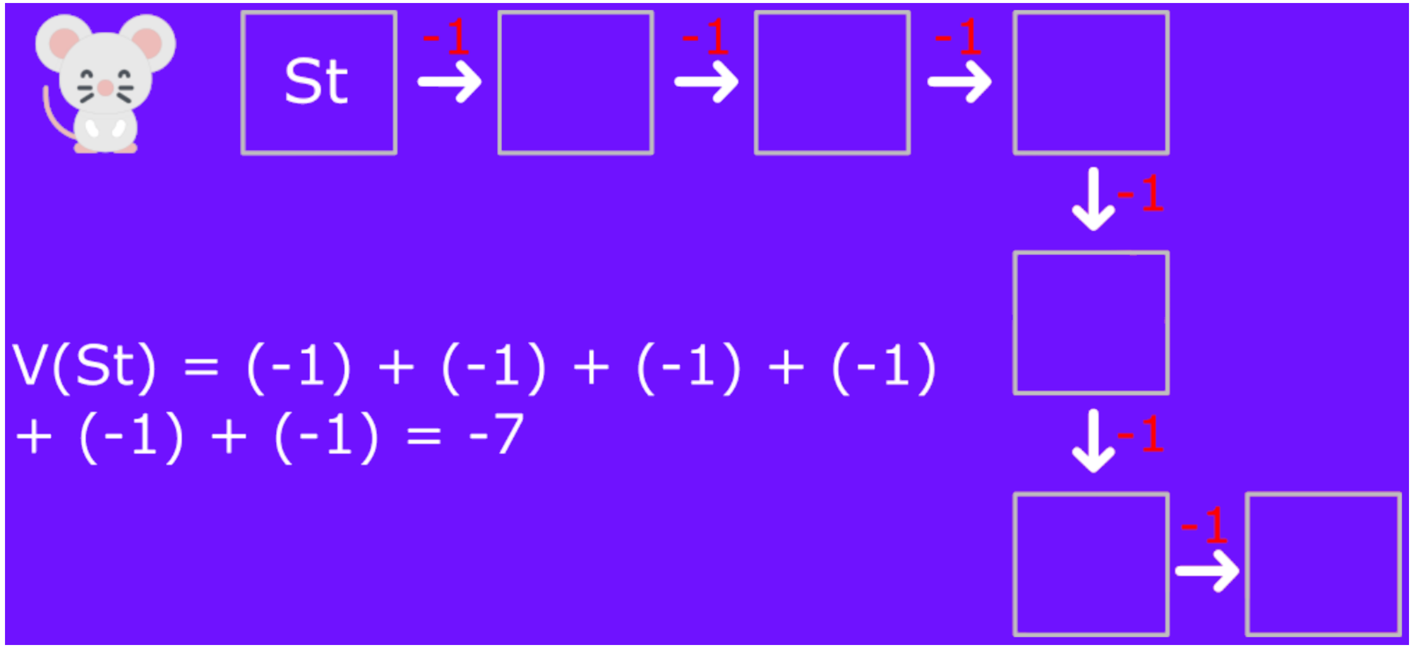
- In state-value function, we calculate the value of a state ($S_t$)
- In action-value function, we calculate the value of state-action pair ($S_t$, $A_t$) hence the value of taking that action at that state
- Whatever value function we choose (state-value or action-value function), the value is the expected return
- We need to sum all the rewards an agent can get if it starts at that state



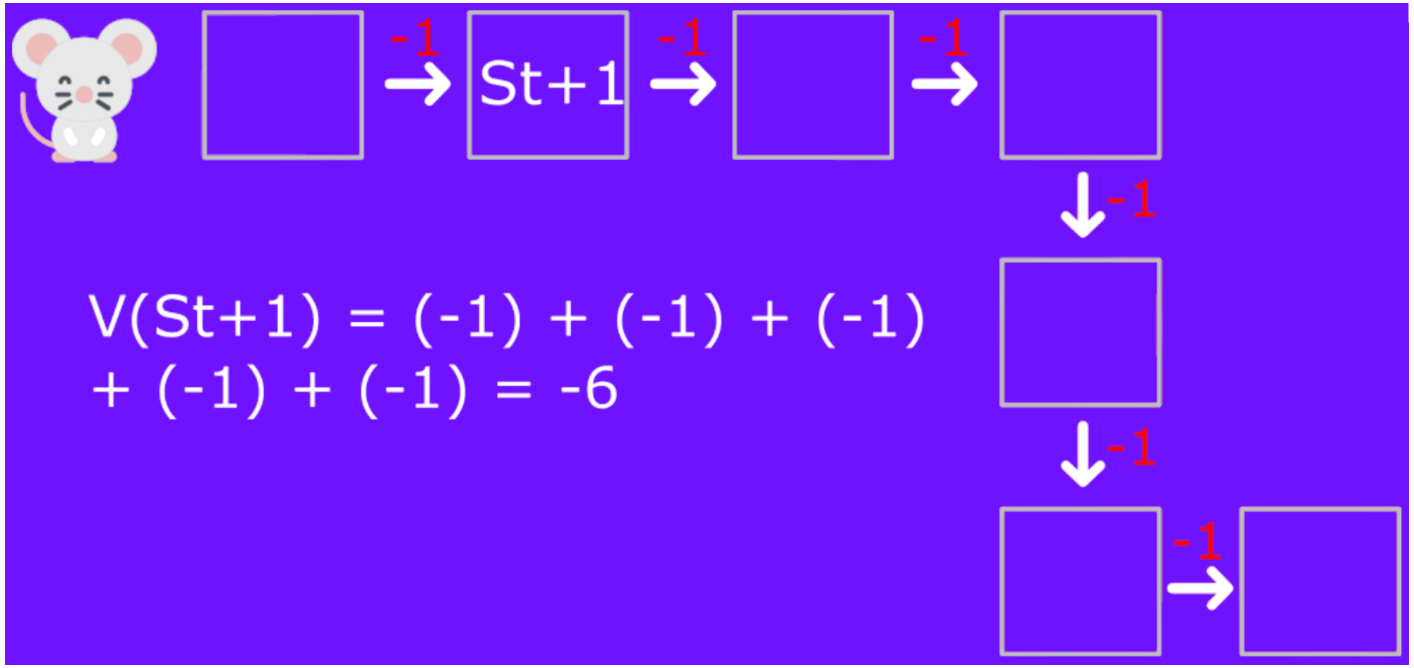
# Bellman equation

- The Bellman equation simplifies our value calculation
- With what we learned from now, we know that if we calculate the $V(S_t)$, we need to calculate the return starting at that state then follow the policy forever after
- So to calculate $V(S_t)$ we need to make the sum of the expected rewards



# Bellman equation

- Then, to calculate the $V(S_{t+1})$, we need to calculate the return starting at that state $S_{t+1}$
- That's a quite dull process if you need to do it for each state value or state-action value
- Instead of calculating for each state or each state-action pair the expected return, we can use the Bellman equation



# Bellman equation

- The Bellman equation is a recursive equation
  - Instead of starting for each state from the beginning and calculating the return, we can consider the value of any state as
- The immediate reward ($R_{t+1}$) + the discounted value of the state that follows ($\gamma * S_{t+1}$)

$$\underbrace{V_\pi(s)}_{\text{Value of state s}} = \underbrace{\mathbf{E}_{\boxed{\pi}}[R_{t+1}}_{\text{Expected value of immediate reward}} + \underbrace{\gamma * V_\pi(S_{t+1})}_{\text{+ the discounted value of next\_state}} \underbrace{|S_t = s]}_{\text{If the agent starts at state s}}$$

And uses the policy to choose its actions for all time steps

# Monte Carlo vs Temporal Difference Learning

- The idea of RL is that using the experience taken, given the reward, it will update its value or its policy
- Monte Carlo and Temporal Difference Learning are two different strategies on how to train our value function or our policy function
  - Both of them use experience to solve the RL problem

- Monte Carlo uses an entire episode of experience before learning
- Temporal Difference uses only a step $(S_t, A_t, R_{t+1}, S_{t+1})$ to learn

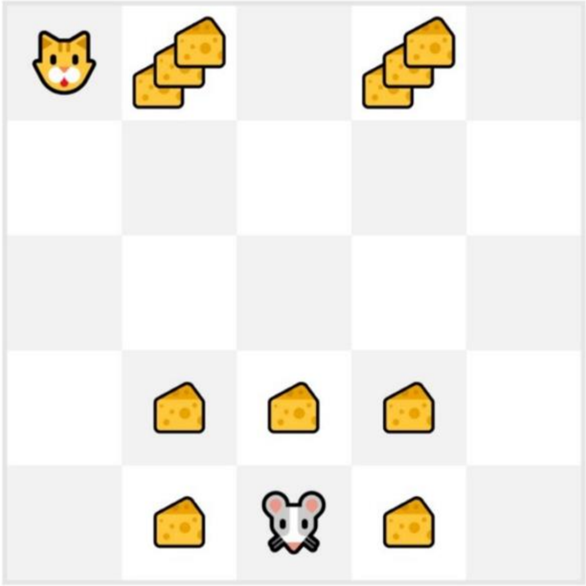
# Monte Carlo

- Monte Carlo waits until the end of the episode, then calculates $G_t$ (return) and uses it as a target for updating $V(S_t)$
  - It requires a complete entire episode of interaction before updating our value function

$$\underbrace{V(S_t)}_{\text{New value of state t}} \leftarrow \underbrace{V(S_t)}_{\substack{\text{Former estimation}\\ \text{of value of state t}\\ \text{(= Expected return}\\ \text{starting at that state)}}} + \underbrace{\alpha}_{\substack{\text{Learning}\\ \text{Rate}}}[\underbrace{G_t}_{\substack{\text{Return at}\\ \text{timestep}\\ \text{t}}} - \underbrace{V(S_t)}_{\substack{\text{Former estimation}\\ \text{of value of state t}\\ \text{(= Expected return}\\ \text{starting at that}\\ \text{state)}}}]$$

# Monte Carlo

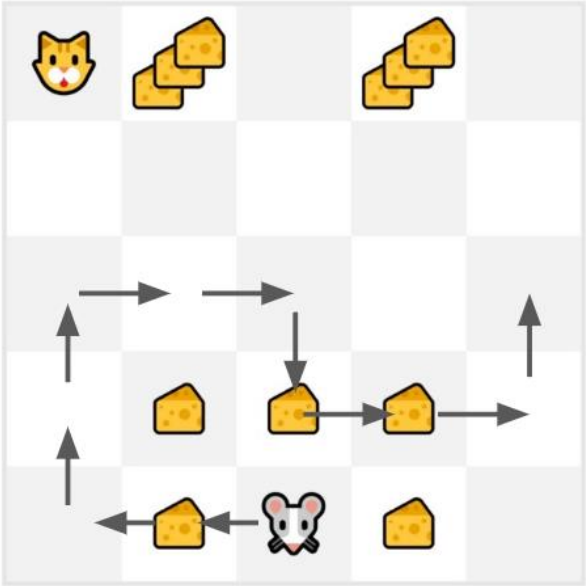
$$V(S_t) \leftarrow V(S_t) + \alpha[G_t - V(S_t)]$$



- We always start the episode at the same starting point
- We try actions using our policy
  - For example, epsilon greedy
- We get the reward and the next state
- We terminate if the cat eats us or if we move > 10 steps
  - We have a list of states, actions, rewards, and next states
- The agent will sum the total rewards $G_t$
- It will then update $V(S_t)$ based on the formula
- Then start a new game with this new knowledge

# Monte Carlo: Example

$$V(S_t) \leftarrow V(S_t) + \alpha[G_t - V(S_t)]$$



- Consider initial values $= 0$, $\alpha = 0.1$, $\gamma = 1$
- $G_t = R_{t+1} + \gamma\, R_{t+2} + \gamma^2\, R_{t+3} + \cdots$
- $G_0 = 1 + 0 + 0 + 0 + 0 + 0 + 1 + 1 + 0 + 0 = 3$
- $V(S_0) = 0 + 0.1\,[3 - 0] = 0.3$
- $G_1 = 0 + 0 + 0 + 0 + 0 + 1 + 1 + 0 + 0 = 2$
- $V(S_1) = 0 + 0.1\,[2 - 0] = 0.2$

- $G_9 = 0 = 0$
- $V(S_9) = 0 + 0.1\,[0 - 0] = 0$

# Temporal Difference Learning

- The idea is that with TD we update the $V(S_t)$ at each step
- But because we didn't play during an entire episode, we don't have $G_t$ (expected return), instead, we estimate $G_t$ by adding $R_{t+1}$ and the discounted value of next state

$$V(S_t) \leftarrow V(S_t) + \alpha[G_t - V(S_t)]$$

$$\underbrace{V(S_t)}_{\text{New value of state t}} \leftarrow \underbrace{V(S_t)}_{\text{Former estimation of value of state t}} + \underbrace{\alpha}_{\text{Learning Rate}}[\underbrace{\underbrace{R_{t+1}}_{\text{Reward}} + \underbrace{\gamma V(S_{t+1})}_{\text{Discounted value of next state}}}_{\text{TD Target}} - \underbrace{V(S_t)}_{}]$$

# Temporal Difference Learning

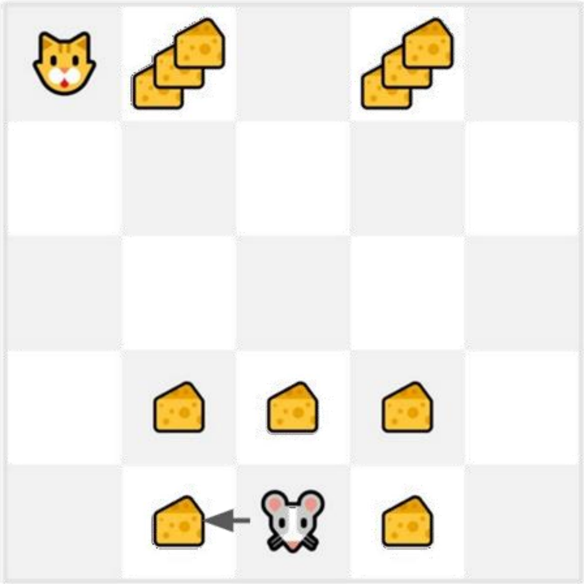
- TD waits for only one interaction (one step) $S_{t+1}$ to form a TD target and update $V(S_t)$ using $R_{t+1}$ and $\gamma\, V(S_{t+1})$
- We speak about bootstrap because TD bases its update part on an existing estimate $V(S_{t+1})$ and not a full sample $G_t$
- This method is called TD(0) or one step TD (update after any individual step)

$$\underbrace{V(S_t)}_{\text{New value of state t}} \leftarrow \underbrace{V(S_t)}_{\text{Former estimation of value of state t}} + \underbrace{\alpha}_{\text{Learning Rate}}\Big[\underbrace{\underbrace{R_{t+1}}_{\text{Reward}} + \underbrace{\gamma V(S_{t+1})}_{\text{Discounted value of next state}}}_{\text{TD Target}} - \underbrace{V(S_t)}\Big]$$

# TD: Example

$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$$

- Consider initial values $= 0$, $\alpha = 0.1$, $\gamma = 1$
- It gets a reward $R_{t+1} = 1$ since it eat a piece of cheese



- $V(S_0) = 0 + 0.1\ [1 + 1 \times 0 - 0] = 0.1$
- We just updated our value function for State 0
- Now we continue to interact with this environment with our updated value function

# Monte Carlo vs TD Learning

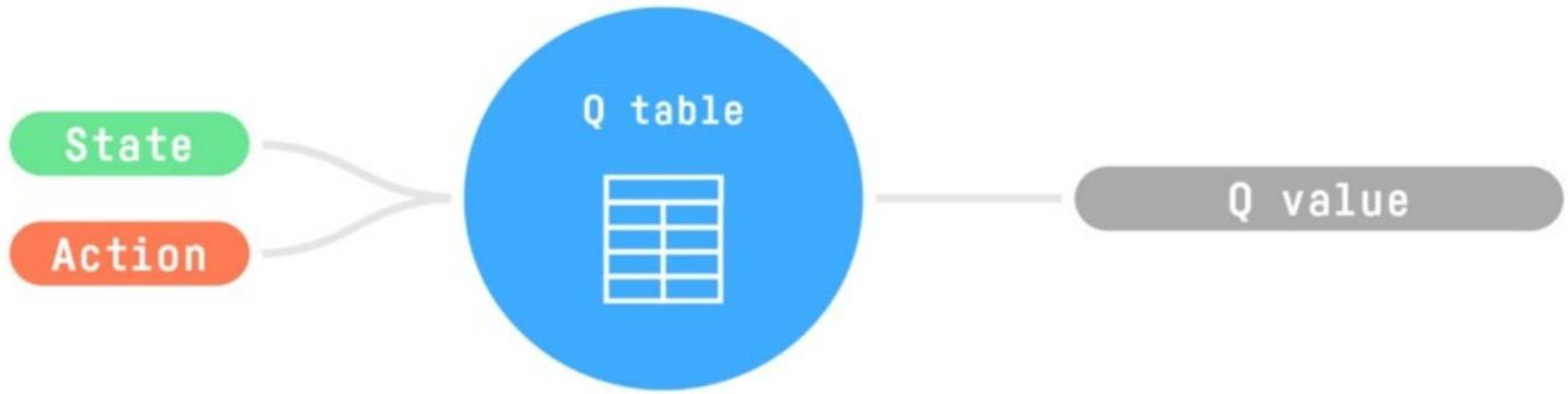
- With Monte Carlo, we update the value function from a complete episode and so we use the actual accurate discounted return of this episode
- With TD learning, we update the value function from a step, so we replace $G_t$ that we don't have with an estimated return called TD target

Monte Carlo: $V(S_t) \leftarrow V(S_t) + \alpha[G_t - V(S_t)]$

TD Learning: $V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$

# Q-Learning

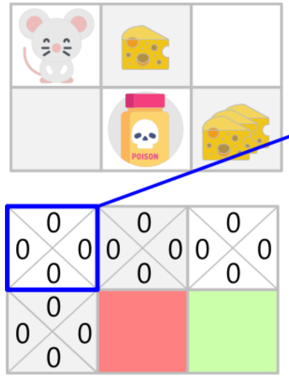
- Q-Learning is an off-policy value-based method that uses a TD approach to train its action-value function
- Q-Learning is the algorithm we use to train our Q-Function, an action-value function that determines the value of being at a certain state, and taking a certain action at that state



Q-Function

# Example

- The Q-Table (just initialized that's why all values are = 0), contains for each state, the 4 state-action values
- Q-Function contains a Q-table that contains the value of each state-action
- Given a state and action, our Q-Function will search inside its Q-table to output the value




Actions

States

| | ← | → | ↑ | ↓ |
|---|---|---|---|---|
| | 0 | 0 | 0 | 0 |
| | 0 | 0 | 0 | 0 |
| | 0 | 0 | 0 | 0 |
| | 0 | 0 | 0 | 0 |
| | 0 | 0 | 0 | 0 |
| | 0 | 0 | 0 | 0 |

# Example

- In the beginning, our Q-Table is useless since it gives arbitrary value for each state-action pair (most of the time we initialize the Q-Table to 0 values)
- But, as we'll explore the environment and update our Q-Table it will give us better and better approximations

| | ← | → | ↑ | ↓ |
|---|---|---|---|---|
| | 0 | 0 | 0 | 0 |
| | 0 | 0 | 0 | 0 |
| | 0 | 0 | 0 | 0 |
| | 0 | 0 | 0 | 0 |
| | 0 | 0 | 0 | 0 |
| | 0 | 0 | 0 | 0 |

$\Rightarrow$

| | ← | → | ↑ | ↓ |
|---|---|---|---|---|
| | 0 | 10.8 | 0 | 0 |
| | 0 | 9.9 | 0 | -10 |
| | 0 | 0 | 0 | 10 |
| | -10 | 0 | 0 | 0 |
| | 0 | 0 | 0 | 0 |
| | 0 | 0 | 0 | 0 |

# Q-Learning pseudocode

**Algorithm 14:** Sarsamax (Q-Learning)

**Input:** policy $\pi$, positive integer *num_episodes*, small positive fraction $\alpha$, 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., $\epsilon$-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

| | ← | → | ↑ | ↓ |
|---|---|---|---|---|
| | 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 $\pi$, positive integer $num\_episodes$, small positive fraction $\alpha$, 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., $\epsilon$-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

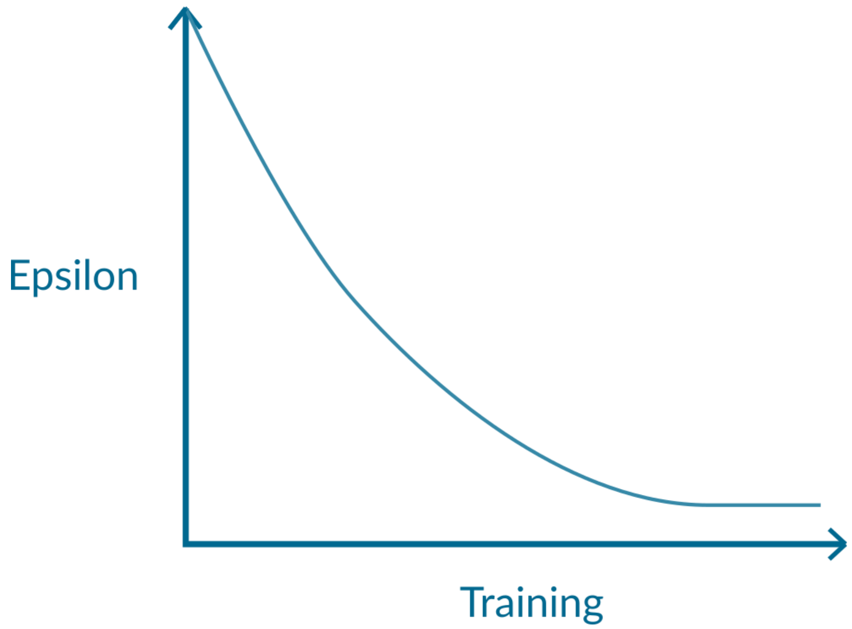
$\varepsilon$-greedy policy

$\varepsilon$ → Exploitation (select the **greedy** action)

$1-\varepsilon$ → Exploration (select the **random** action)

**Algorithm 14:** Sarsamax (Q-Learning)

**Input:** policy $\pi$, positive integer *num_episodes*, small positive fraction $\alpha$, 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., $\epsilon$-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** $Q$

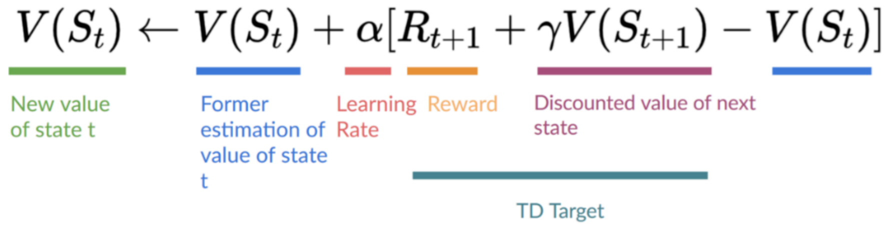


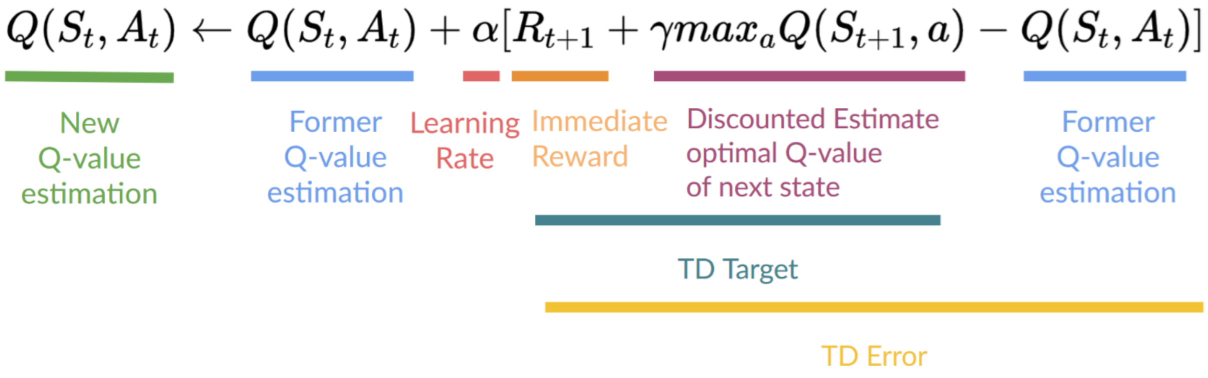
# Q-Learning

- To update $Q(S_t, A_t)$, we need $S_t$, $A_t$, $R_{t+1}$, $S_{t+1}$
- We use $R_{t+1}$ and to get the best next-state-action pair value, we select with a greedy-policy (so not our epsilon greedy policy) the next best action

**Algorithm 14:** Sarsamax (Q-Learning)

**Input:** policy $\pi$, positive integer *num_episodes*, small positive fraction $\alpha$, 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., $\epsilon$-greedy)
        Take action $A_t$ and observe $R_{t+1}, S_{t+1}$
        $\boxed{Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t))}$
        $t \leftarrow t + 1$
    **until** $S_t$ *is terminal*;
**end**
**return** $Q$



# 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 $\pi$, positive integer *num_episodes*, small positive fraction $\alpha$, 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., $\epsilon$-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 $\pi$, positive integer *num_episodes*, small positive fraction $\alpha$, 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 $S_0$
  Choose action $A_0$ using policy derived from $Q$ (e.g., $\epsilon$-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., $\epsilon$-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** $Q$