Introduction to Reinforcement Learning

A mini course @ HCMUS, Vietnam

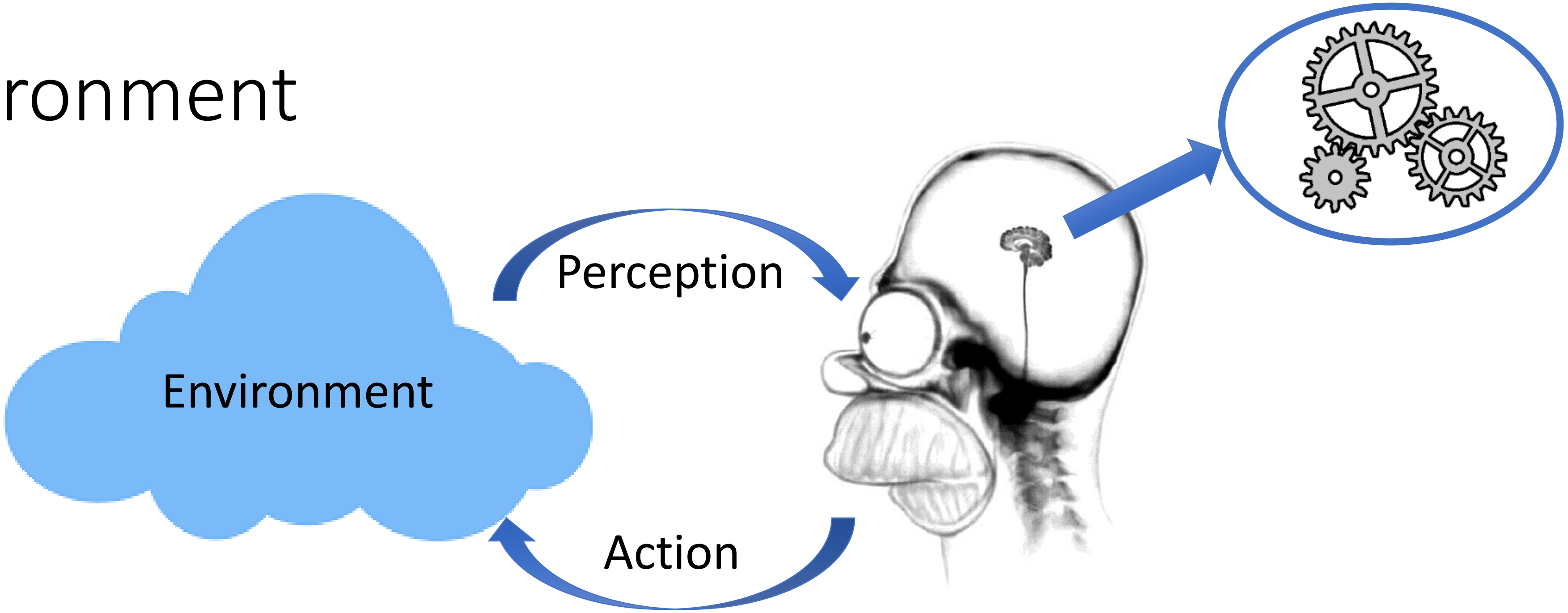
Lectures 1-3 (cont’d)

Long Tran-Thanh

Long.tran-thanh@warwick.ac.uk

Model-free approaches

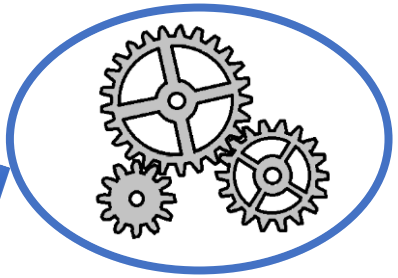
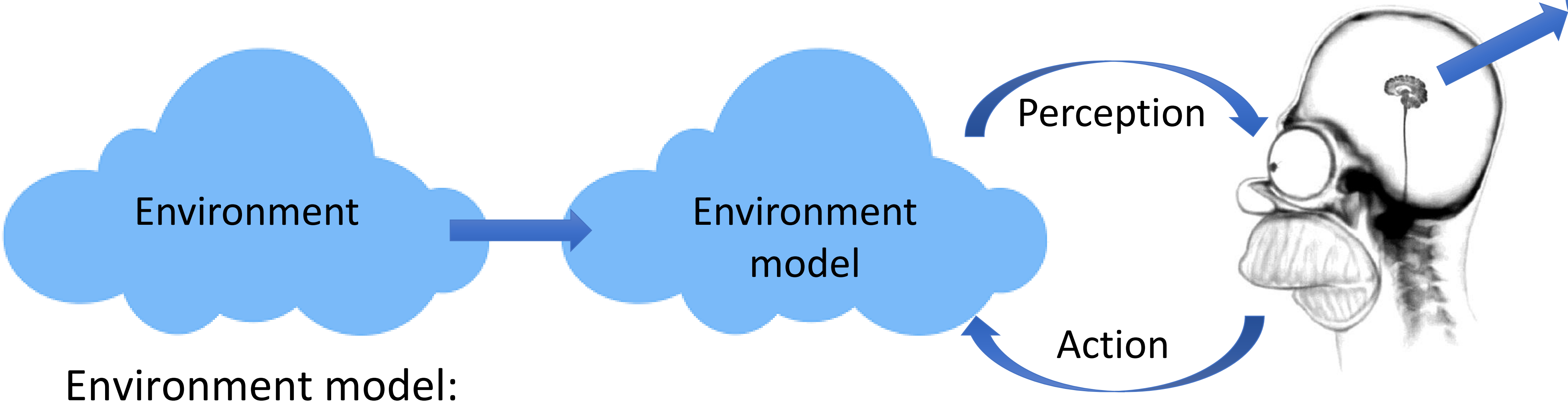
Agent + environment



Abstract model:

- Agent
- Environment

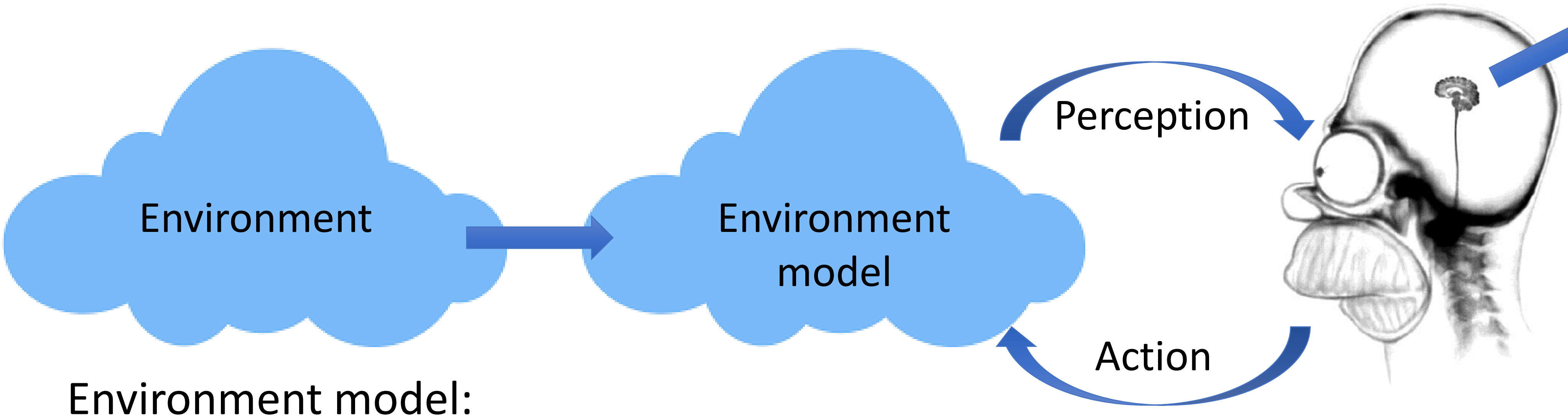
Environment model

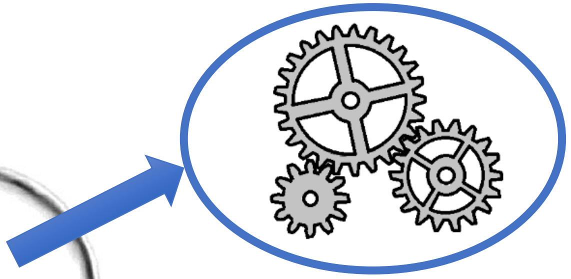



Environment model:

- Having access to this model gives us *P* and *R*
- Using DP + Bellman operator would work very well

Environment model



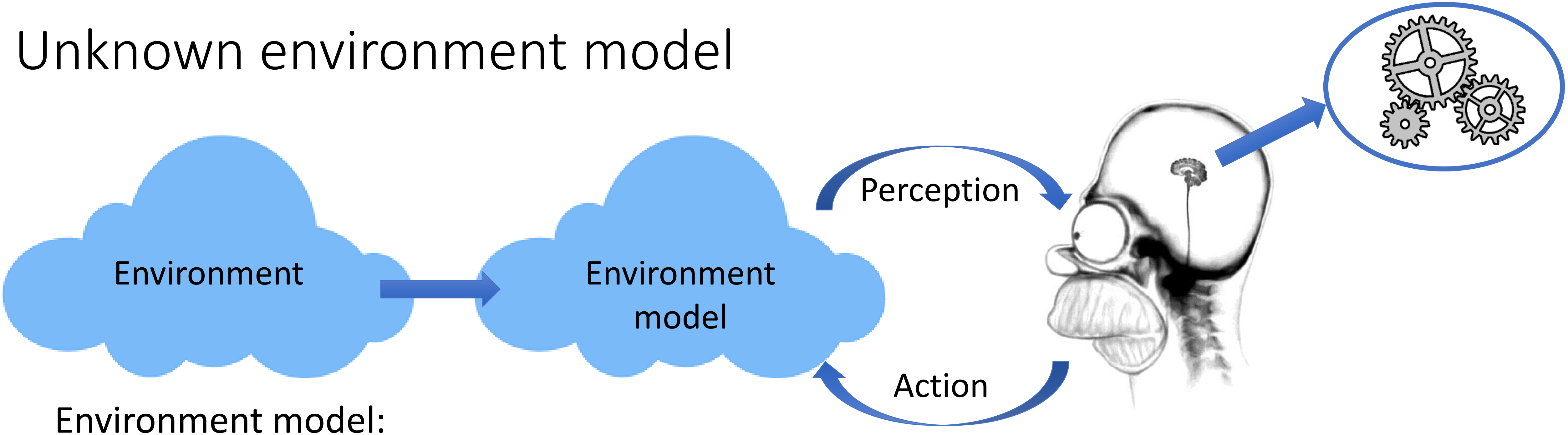


Environment model:

- Knowing this model gives us full information about *P* and *R*
- Using DP + Bellman operator would work very well

What happens if we don’t know *P* and/or *R*?

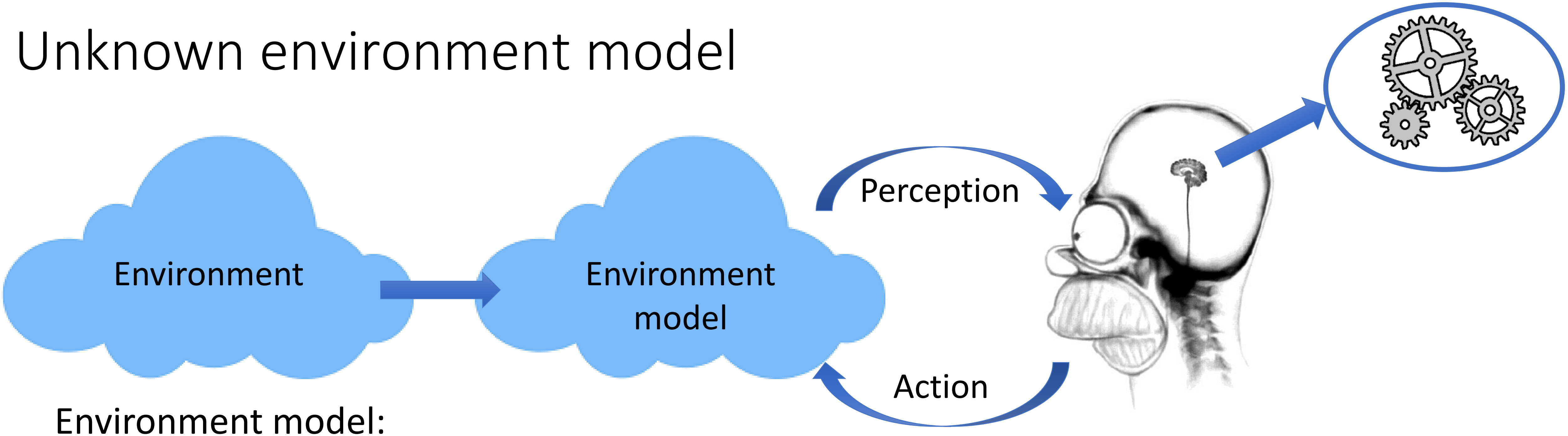
Unknown environment model



Environment model:

- We don’t know the model
- But we have access to it through interactions
- We can observe what happens if we make actions

Unknown environment model



Environment model:

- We don’t know the model
- But we have access to it through interactions
- We can observe what happens if we make actions

MDP: we assume full observability (received reward, new state)
POMDP: partially observable MDP (some feedback is missing)

- In this course we only focus on MDPs

Covered methods in this lecture

- Monte Carlo policy
- Temporal Difference: TD[0], TD[lambda]
- SARSA
- Q-learning

Monte Carlo (MC) method

Main question: how to evaluate a policy's value if we don't know the MDP model

Idea:

- Just follow the policy, let the randomness of the model and policy guide us
- Observe the trajectory *T* of the policy starting from certain state *s*
- Repeat this many times, and take the average total returns -> estimate for the value of *s*

Monte Carlo (MC) method

Main question: how to evaluate a policy's value if we don't know the MDP model

Idea:

- Just follow the policy, let the randomness of the model and policy guide us
- Observe the trajectory *T* of the policy starting from certain state *s*
- Repeat this many times, and take the average total returns -> estimate for the value of *s*

Justification:

$$G_t = r_t + \gamma t_{t+1} + \gamma^2 r_{t+2} + \dots$$

$$V^\pi(s) = \mathbb{E}_{T\sim\pi}[G_t | s_t = s]$$

Monte Carlo (MC) method

This is called the MC method:

- Follow the policy, let the randomness of the model and policy guide us
- Observe the trajectory *T* of the policy starting from certain state *s*
- Repeat this many times, and take the average total returns -> estimate for the value of *s*

We need to know when to stop following the policy -> works for finite trajectories only
-> MC method can **only be applied to episodic MDPs** (finite horizon H)

Monte Carlo (MC) method

Goal: estimate the value $V^\pi(s)$ of each state *s* under policy π

- Observe random state-action sequence by following the policy with episode length *H*:

$$s_1, a_1, s_2, a_2, \ldots, s_H, a_H, s_{H+1}$$

- Calculate return $G_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots$ for each *t*

- Repeat this for multiple episodes

- For each *s*: MC computes the empirical mean of relevant returns

Monte Carlo (MC) method

Goal: estimate the value $V^{\pi}(s)$ of each state *s* under policy π

- Observe random state-action sequence by following the policy:

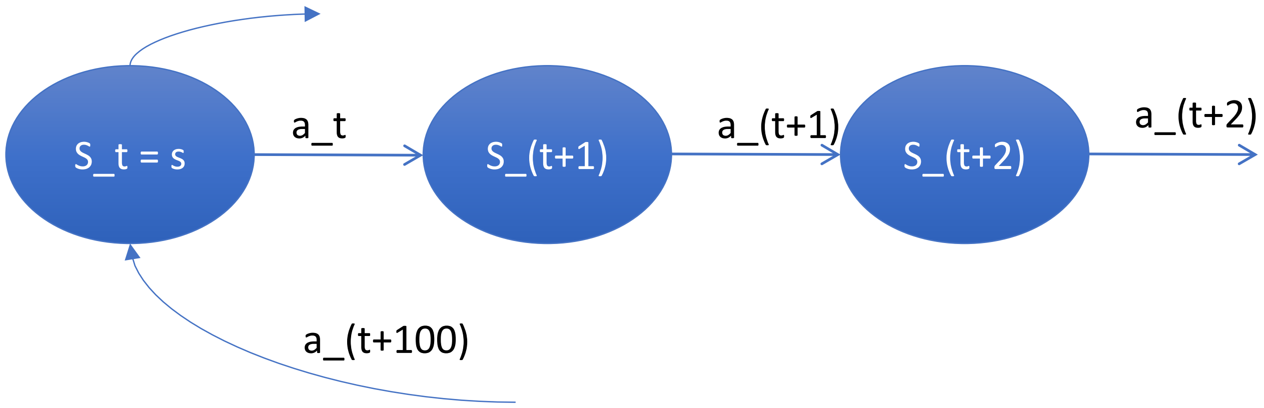
$$s_1, a_1, s_2, a_2, \ldots, s_H, a_H, s_{H+1}$$

- Calculate return $G_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots$ for each *t*

- For each *s*: MC computes the empirical mean of relevant returns

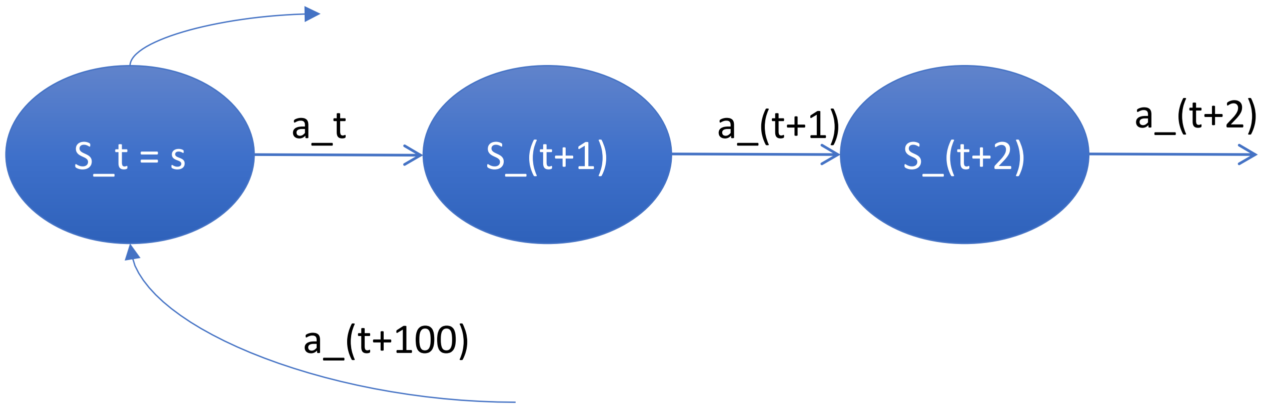
What are the **relevant** return values?

Monte Carlo (MC) method



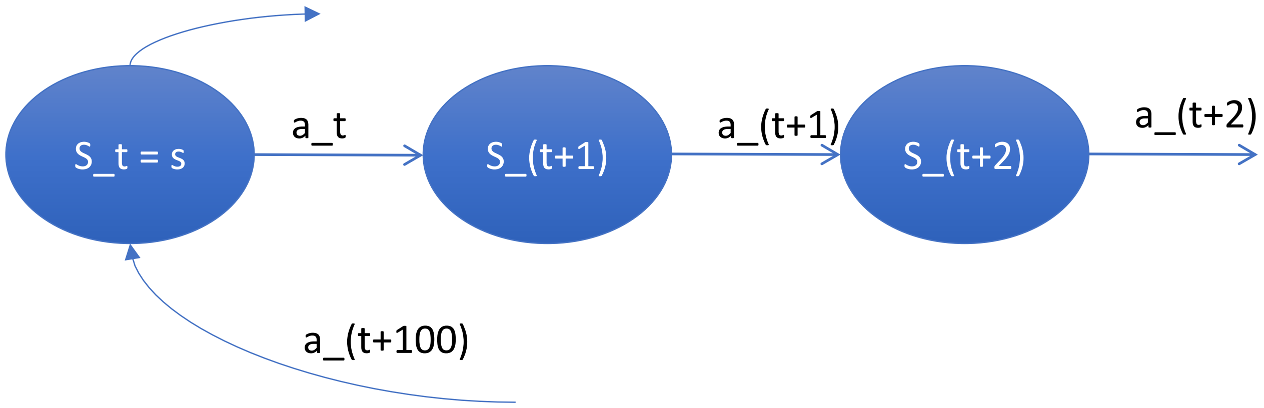
- During the process, we may visit *s* multiple times
  - Include the corresponding return of every time we visit s?
  - Include the return of first time visit only?

Monte Carlo (MC) method



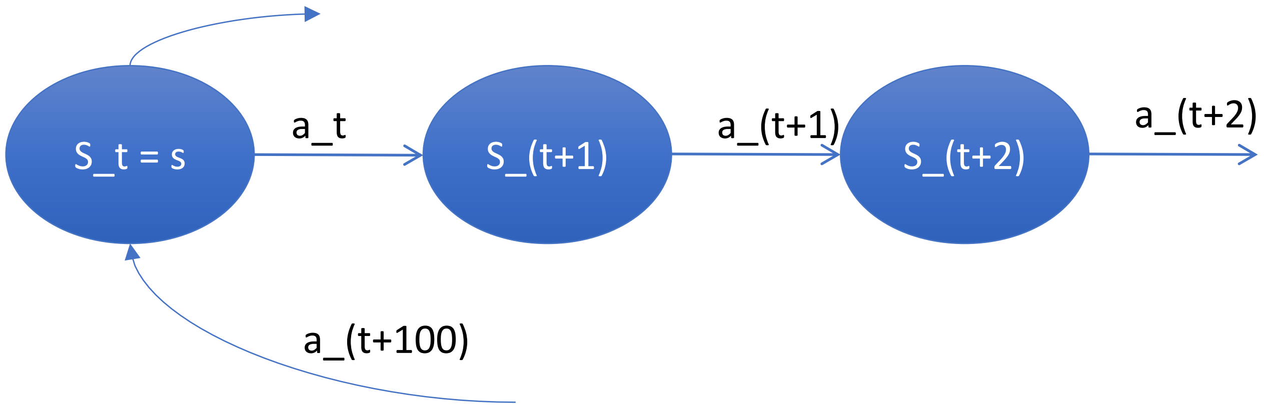
- First-visit MC method:
  - Aim: to evaluate value of state *s*
  - For each episode *i*: The **first time step *t*** that state *s* is visited in the episode:
    - Increment counter $N(s) = N(s) + 1$
    - Increment total return $G(s) = G(s) + G_{i,t}$
  - Value is estimated by empirical mean return $\hat{V}(s) = G(s)/N(s)$
  - By law of large numbers, $\hat{V}(s) \rightarrow V(s)$ as $N(s) \rightarrow \infty$

Monte Carlo (MC) method



- Every-visit MC method:
  - Aim: to evaluate value of state *s*
  - For each episode *i*: for **every time step *t*** that state *s* is visited in the episode:
    - Increment counter $N(s) = N(s) + 1$
    - Increment total return $G(s) = G(s) + G_{i,t}$
  - Value is estimated by empirical mean return $\hat{V}(s) = G(s)/N(s)$
  - By law of large numbers, $\hat{V}(s) \rightarrow V(s)$ as $N(s) \rightarrow \infty$

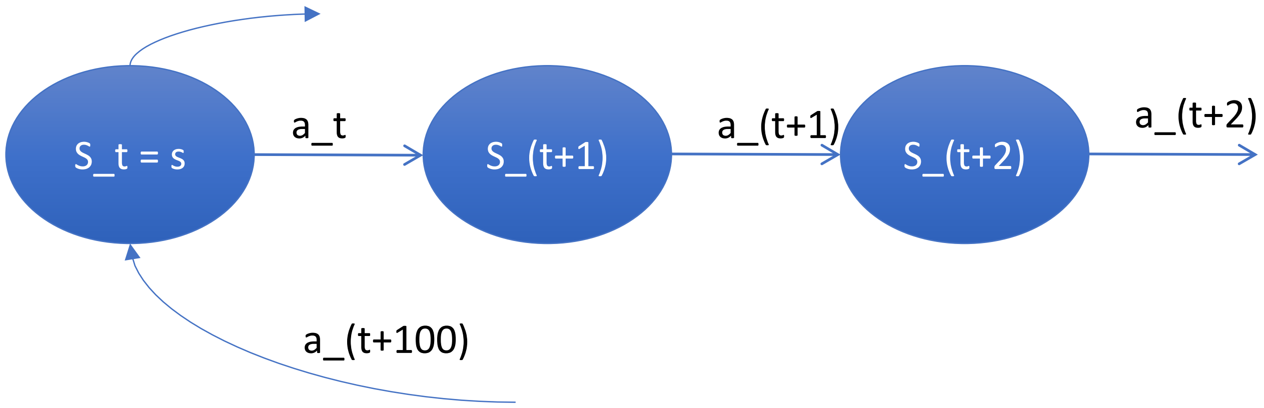
Monte Carlo (MC) method



- First-visit MC
  - Unbiased estimator of true value
  - Slower convergence (slower updates)

- Every-visit MC:
  - Duplicated rewards (introduce estimation bias)
  - Faster convergence

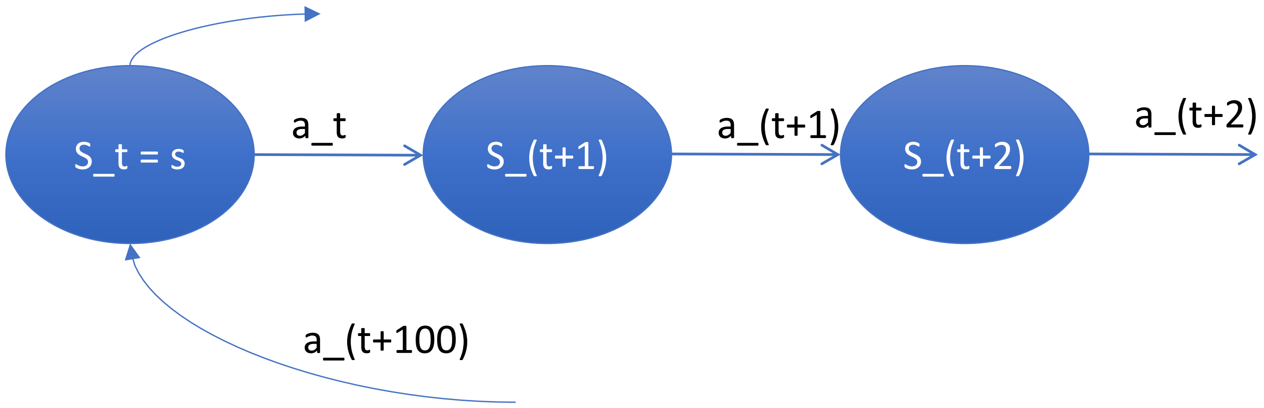
Monte Carlo (MC) method



- Third option - Incremental MC:
  - Idea: the value update is in fact the same as the following incremental update:

$$\hat{V}(s) = \hat{V}(s)\frac{N(s)-1}{N(s)} + \frac{1}{N(s)}G_{i,t}$$

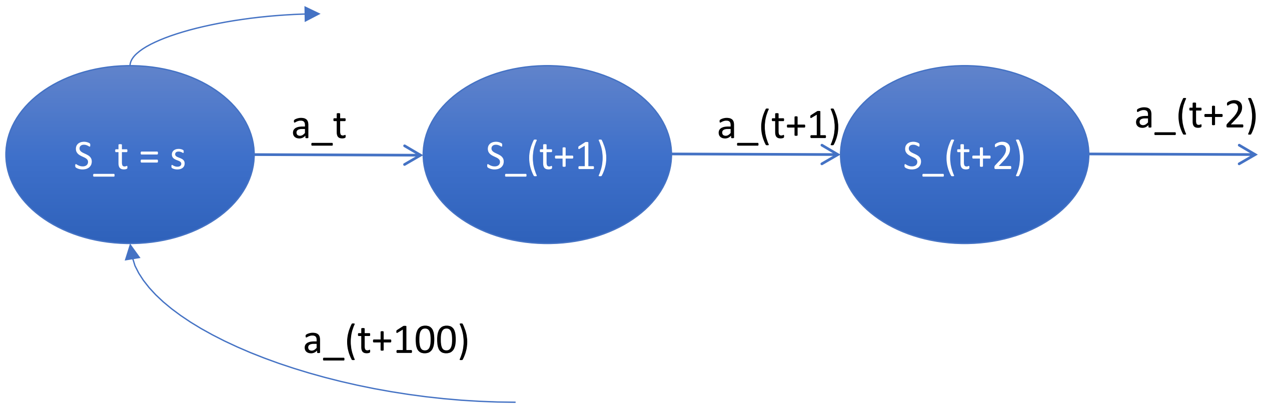
Monte Carlo (MC) method



- Third option - Incremental MC:
  - Idea: the value update is in fact the same as the following incremental update:

$$\hat{V}(s) = \hat{V}(s)\frac{N(s)-1}{N(s)} + \frac{1}{N(s)}G_{i,t}$$
$$= \hat{V}(s) + \frac{1}{N(s)}\left(G_{i,t} - \hat{V}(s)\right)$$

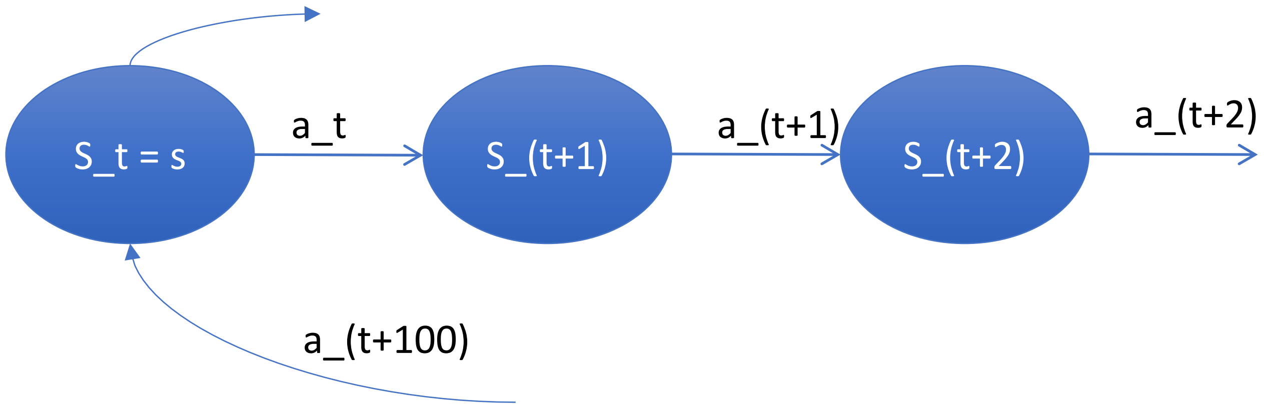
Monte Carlo (MC) method



- Third option - Incremental MC:
  - Idea: the value update is in fact the same as the following incremental update:

$$\hat{V}(s) = \hat{V}(s)\frac{N(s)-1}{N(s)} + \frac{1}{N(s)}G_{i,t}$$
$$= \boxed{\hat{V}(s) + \frac{1}{N(s)}\left(G_{i,t} - \hat{V}(s)\right)}$$

Monte Carlo (MC) method



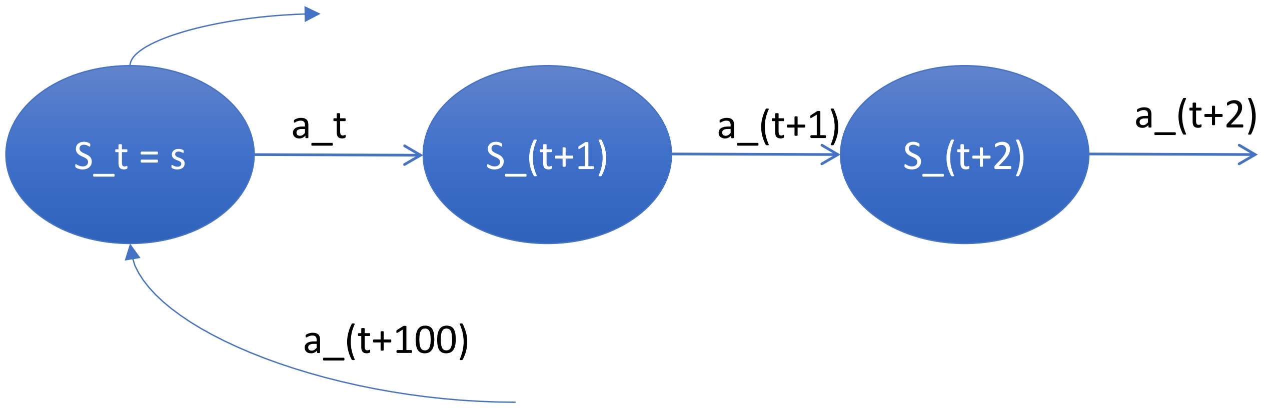
- Third option - Incremental MC:
  - Idea: the value update is in fact the same as the following incremental update:

$$\hat{V}(s) = \hat{V}(s)\frac{N(s)-1}{N(s)} + \frac{1}{N(s)}G_{i,t}$$

$$= \hat{V}(s) + \frac{1}{N(s)}\left(G_{i,t} - \hat{V}(s)\right)$$

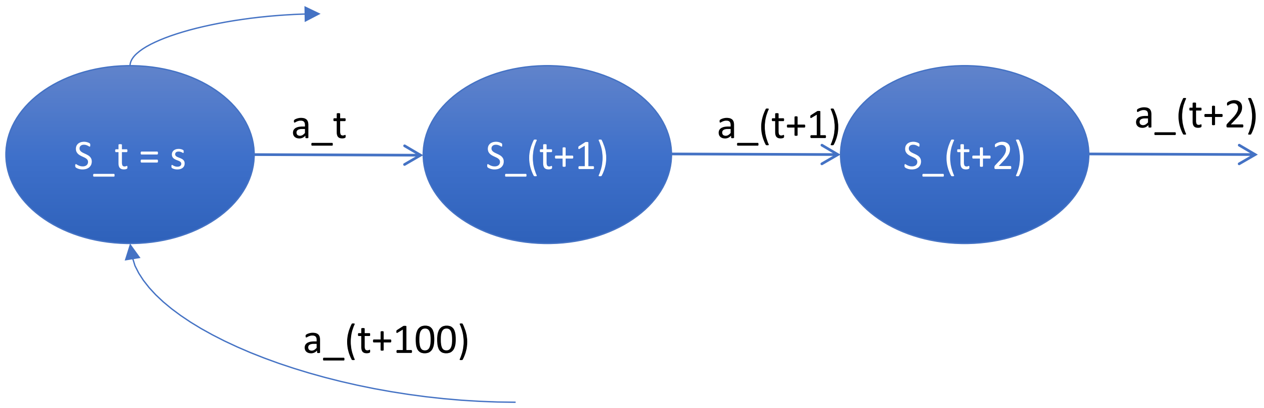
  - More general update rule: $\hat{V}(s) = \hat{V}(s) + \alpha\left(G_{i,t} - \hat{V}(s)\right)$

Monte Carlo (MC) method



- Third option - Incremental MC:
  - Every-visit MC + general update rule $\hat{V}(s) = \hat{V}(s) + \alpha\left(G_{i,t} - \hat{V}(s)\right)$

Monte Carlo (MC) method

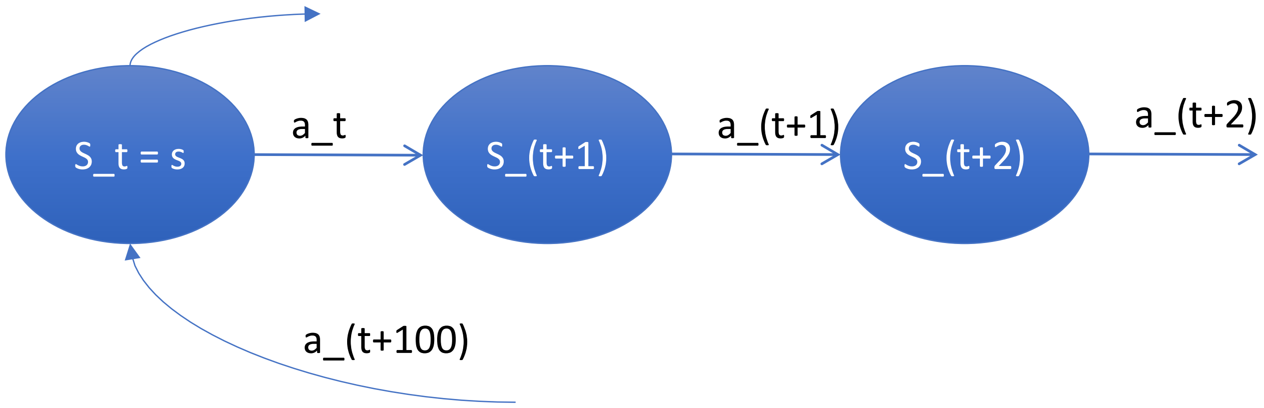


- Third option - Incremental MC:
  - Every-visit MC + general update rule $\hat{V}(s) = \hat{V}(s) + \alpha\left(G_{i,t} - \hat{V}(s)\right)$

- If $\alpha = 1/N(s)$ then incremental MC = every-visit MC

- $\alpha > 1/N(s)$ is useful when MDP is not stationary (i.e., we prioritise more recent updates)

Summary of the MC method



- Simple: Estimates expectation by empirical average (given episodes sampled from policy of interest)
- Updates V estimate using sample of return to approximate the expectation
- **Does not assume Markov property** (why?)
- Converges to true value under some (generally mild) assumptions
- But: **works for finite/episodic MDPs only**

Temporal difference (TD) learning

Recall:

- Dynamic programming: efficient calculations + can handle infinite horizon, but needs to know MDP (i.e., model-based)
- MC method/learning: model-free, but requires episodic nature

Idea: Why not combine these 2 and take the best of each?

Temporal difference (TD) learning

Recall:

- Dynamic programming: efficient calculations + can handle infinite horizon, but needs to know MDP (i.e., model-based)
- MC method/learning: model-free, but requires episodic nature

Idea: Why not combine these 2 and take the best of each? – this is what TD does

“If one had to identify one idea as central and novel to reinforcement learning, it would undoubtedly be temporal-difference (TD) learning." - Sutton and Barto 2017

Temporal difference (TD) learning

TD learning (or more precisely, TD(0) learning)

- Bellman operator: $B^\pi V^\pi(s) = r(s, \pi(s)) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, \pi(s)) V^\pi(s')$
- Incremental MC method: $\hat{V}(s) = \hat{V}(s) + \alpha \left(G_{i,t} - \hat{V}(s) \right)$

Temporal difference (TD) learning

TD learning (or more precisely, TD(0) learning)

- Bellman operator: $B^\pi V^\pi(s) = r(s,\pi(s)) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,\pi(s)) V^\pi(s')$
- Incremental MC method: $\hat{V}(s) = \hat{V}(s) + \alpha \left(G_{i,t} - \hat{V}(s) \right)$
- From the latter, we can write: $\hat{V}^\pi(s) = \hat{V}^\pi(s) + \alpha \left([r_t + \gamma \hat{V}^\pi(s_{t+1})] - \hat{V}^\pi(s) \right)$
  - $[r_t + \gamma \hat{V}^\pi(s_{t+1})]$ is a 1-step look-ahead estimate of $G_{i,t}$
  - Note that in MC we only do this update at the end of each epoch (but then we do for each state *s*)

Temporal difference (TD) learning

TD learning (or more precisely, TD(0) learning)

- Bellman operator: $B^\pi V^\pi(s) = r(s,\pi(s)) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,\pi(s)) V^\pi(s')$
- Incremental MC method: $\hat{V}(s) = \hat{V}(s) + \alpha \left(G_{i,t} - \hat{V}(s) \right)$
- From the latter, we can write: $\hat{V}^\pi(s) = \hat{V}^\pi(s) + \alpha \left([r_t + \gamma \hat{V}^\pi(s_{t+1})] - \hat{V}^\pi(s) \right)$
- In TD - do this update, but straight after we visit a state (and don’t wait until the end of the episode):

$$\hat{V}^\pi(s_t) = \hat{V}^\pi(s_t) + \alpha \left([r_t + \gamma \hat{V}^\pi(s_{t+1})] - \hat{V}^\pi(s_t) \right)$$

Temporal difference (TD) learning

TD learning (or more precisely, TD(0) learning):

- Can immediately update after each visit of *s*
- Therefore, there's no need for episodic setting
- If we repeat this many times, it resembles the Bellman operator (why?)

Temporal difference (TD) learning

TD learning (or more precisely, TD(0) learning):

- Can immediately update after each visit of *s*
- Therefore, there's no need for episodic setting
- If we repeat this many times, it resembles the Bellman operator (why?)

- TD target: $[r_t + \gamma \hat{V}^\pi(s_{t+1})]$

- TD error: $\delta_t = [r_t + \gamma \hat{V}^\pi(s_{t+1})] - \hat{V}^\pi(s_t)$

Temporal difference (TD) learning

TD learning (or more precisely, TD(0) learning):

Input: α
Initialisation: $\forall s \in \mathcal{S} : \quad \hat{V}(s) = 0$

While (interacting with environment):

Sample tuple (s_t, a_t, r_t, s_{t+1})
Update $\hat{V}^\pi(s_t) = \hat{V}^\pi(s_t) + \alpha \left([r_t + \gamma \hat{V}^\pi(s_{t+1})] - \hat{V}^\pi(s_t) \right)$

From TD(0) to TD(lambda)

Recall that in TD(0): $\hat{V}^{\pi}(s_t) = \hat{V}^{\pi}(s_t) + \alpha \left(\underbrace{[r_t + \gamma \hat{V}^{\pi}(s_{t+1})]}_{\text{1-step estimate of return } G_{i,t}} - \hat{V}^{\pi}(s_t) \right)$

From TD(0) to TD(lambda)

Recall that in TD(0): $\hat{V}^{\pi}(s_t) = \hat{V}^{\pi}(s_t) + \alpha \left(\underbrace{[r_t + \gamma \hat{V}^{\pi}(s_{t+1})]}_{\text{1-step estimate of return } G_{i,t}} - \hat{V}^{\pi}(s_t) \right)$

We can use n-step estimates as well

$$G_{i,t}^{(1)} = r_t + \gamma \hat{V}^{\pi}(s_{t+1})$$

$$G_{i,t}^{(2)} = r_t + \gamma r_{t+1} + \gamma^2 \hat{V}^{\pi}(s_{t+2})$$

$$\vdots$$

$$G_{i,t}^{(\infty)} = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots$$

From TD(0) to TD(lambda)

Recall that in TD(0): $\hat{V}^\pi(s_t) = \hat{V}^\pi(s_t) + \alpha\left(\underbrace{[r_t + \gamma\hat{V}^\pi(s_{t+1})]}_{\text{1-step estimate of return } G_{i,t}} - \hat{V}^\pi(s_t)\right)$

We can use n-step estimates as well

$$G_{i,t}^{(1)} = r_t + \gamma\hat{V}^\pi(s_{t+1})$$

$$G_{i,t}^{(2)} = r_t + \gamma r_{t+1} + \gamma^2\hat{V}^\pi(s_{t+2})$$

$$\vdots$$

$$G_{i,t}^{(\infty)} = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots$$

Which n would be the best?

- Small n = worse accuracy but faster update
- Large n = the opposite

Idea: why not combine return of different n-steps?

- E.g., take the average of n = 2 and n = 5
- How to efficiently combine these together?

From TD(0) to TD(lambda)

Recall that in TD(0): $\hat{V}^\pi(s_t) = \hat{V}^\pi(s_t) + \alpha \left(\underbrace{[r_t + \gamma \hat{V}^\pi(s_{t+1})]}_{\text{1-step estimate of return } G_{i,t}} - \hat{V}^\pi(s_t) \right)$

We can use n-step estimates as well

$$G_{i,t}^{(1)} = r_t + \gamma \hat{V}^\pi(s_{t+1})$$

$$G_{i,t}^{(2)} = r_t + \gamma r_{t+1} + \gamma^2 \hat{V}^\pi(s_{t+2})$$

$$\vdots$$

$$G_{i,t}^{(\infty)} = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots$$

Lambda-return $G_t^{(\lambda)}$:

$$G_t^{(\lambda)} = (1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1} G_t^{(n)}$$

TD(λ):

$$\hat{V}^\pi(s_t) = \hat{V}^\pi(s_t) + \alpha(G_t^\lambda - \hat{V}^\pi(s_t))$$

Forward-view TD(lambda)

TD(λ): $\hat{V}^\pi(s_t) = \hat{V}^\pi(s_t) + \alpha(G_t^\lambda - \hat{V}^\pi(s_t))$

How to calculate G_t^λ ?

Forward-view:

- Look into the future and calculate return for each n
- This has restrictions similar to MC: only works for finite episodes

Backward-view TD(lambda)

TD(λ): $\hat{V}^{\pi}(s_t) = \hat{V}^{\pi}(s_t) + \alpha(G_t^{\lambda} - \hat{V}^{\pi}(s_t))$

How to calculate G_t^{λ} ?

Backward-view:

- We update the returns of states visited in the past instead
- This sounds good, but how to implement it in practice?

Backward-view TD(lambda)

TD(λ): $\hat{V}^\pi(s_t) = \hat{V}^\pi(s_t) + \alpha(G_t^\lambda - \hat{V}^\pi(s_t))$

How to calculate G_t^λ ?

Backward-view:

- We update the returns of states visited in the past instead
- This sounds good, but how to implement it in practice?

Some heuristics:

- Frequency heuristic: assign current value to most frequent states
- Recency heuristic: assign current value to most recent states

Backward-view TD(lambda)

TD(λ): $\hat{V}^{\pi}(s_t) = \hat{V}^{\pi}(s_t) + \alpha(G_t^{\lambda} - \hat{V}^{\pi}(s_t))$

How to calculate G_t^{λ} ?

Backward-view:

- We update the returns of states visited in the past instead
- This sounds good, but how to implement it in practice?

Some heuristics:

- Frequency heuristic: assign current value to most frequent states
- Recency heuristic: assign current value to most recent states

Eligibility trace: $E_0(s) = 0 \qquad E_t(s) = \gamma\lambda E_{t-1}(s) + \mathbb{I}(s_t = s)$

Backward-view TD(lambda)

TD(λ): $\hat{V}^{\pi}(s_t) = \hat{V}^{\pi}(s_t) + \alpha(G_t^{\lambda} - \hat{V}^{\pi}(s_t))$

Backward-view:

- Eligibility trace: $E_0(s) = 0 \qquad E_t(s) = \gamma\lambda E_{t-1}(s) + \mathbb{I}(s_t = s)$
- Recall TD error: $\delta_t = [r_t + \gamma\hat{V}^{\pi}(s_{t+1})] - \hat{V}^{\pi}(s_t)$
- Value update: $\hat{V}^{\pi}(s) = \hat{V}^{\pi}(s) + \alpha\delta_t E_t(s_t)$

Backward-view TD(lambda)

TD(λ): $\hat{V}^\pi(s_t) = \hat{V}^\pi(s_t) + \alpha(G_t^\lambda - \hat{V}^\pi(s_t))$

Backward-view:

- Eligibility trace: $E_0(s) = 0 \qquad E_t(s) = \gamma\lambda E_{t-1}(s) + \mathbb{I}(s_t = s)$
- Recall TD error: $\delta_t = [r_t + \gamma\hat{V}^\pi(s_{t+1})] - \hat{V}^\pi(s_t)$
- Value update: $\hat{V}^\pi(s) = \hat{V}^\pi(s) + \alpha\delta_t E_t(s_t)$

Theorem: The sum of the updates is identical for forward-view and backward-view **in the offline setting** (i.e., episodic/batch)

Backward-view TD(lambda)

TD(λ): $\hat{V}^\pi(s_t) = \hat{V}^\pi(s_t) + \alpha(G_t^\lambda - \hat{V}^\pi(s_t))$

Backward-view:

- Eligibility trace: $E_0(s) = 0 \qquad E_t(s) = \gamma\lambda E_{t-1}(s) + \mathbb{I}(s_t = s)$
- Recall TD error: $\delta_t = [r_t + \gamma\hat{V}^\pi(s_{t+1})] - \hat{V}^\pi(s_t)$
- Value update: $\hat{V}^\pi(s) = \hat{V}^\pi(s) + \alpha\delta_t E_t(s_t)$

Theorem: The sum of the updates is identical for forward-view and backward-view **in the offline setting** (i.e., episodic/batch)

Theorem: With a slightly different eligibility trace, backward = forward-view in online setting (ICML 2014)

From policy evaluation to finding optimal policy

So far what we have done:

Given a policy π , we want to compute its value function

- DP: relies on Bellman operator, needs knowledge of MDP
- MC method: model-free, but episodic
- TD learning: model-free, works for both episodic and infinite horizon

From policy evaluation to finding optimal policy

So far what we have done:

Given a policy π , we want to compute its value function

- DP: relies on Bellman operator, needs knowledge of MDP
- MC method: model-free, but episodic
- TD learning: model-free, works for both episodic and infinite horizon

Next question: Can we find the optimal policy without the knowledge of the MDP model?

From policy evaluation to finding optimal policy

Recall: in case of known MDP model -> policy iteration

- State-action value function + policy improvement

$$Q^{\pi_i}(s,a) = R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) V^{\pi_i}(s')$$

$$\forall s \in \mathcal{S}: \quad \pi_{i+1}(s) = \arg\max_{a \in \mathcal{A}} Q^{\pi_i}(s,a)$$

Can we do the same without knowing *R* and *P*?

Idea: combine the model-free policy evaluation techniques (MC, TD) with policy iteration -> **model-free policy iteration**

Model-free policy iteration

Ideal case:

- Initialise policy π_0
- Repeat:
  - Policy evaluation: compute Q^{π_i}
  - Policy improvement: compute π_{i+1} given Q^{π_i}

Model-free policy iteration

Ideal case:

- Initialise policy π_0
- Repeat:
  - Policy evaluation: compute Q^{π_i}
  - Policy improvement: compute π_{i+1} given Q^{π_i}

Possible issues:

- If π_i is deterministic then we cannot compute $Q^{\pi_i}(s,a)$ for all $a \neq \pi(s)$

Model-free policy iteration

Ideal case:
- Initialise policy π_0
- Repeat:
  - Policy evaluation: compute Q^{π_i}
  - Policy improvement: compute π_{i+1} given Q^{π_i}

Possible issues:
- If π_i is deterministic then we cannot compute $Q^{\pi_i}(s, a)$ for all $a \neq \pi(s)$

How to visit other $a \neq \pi(s)$?

Idea: add some extra exploration (e.g., add random noise to the action choice)

Epsilon-greedy policy iteration

Initialise policy π_0 , $\varepsilon \in [0, 1]$

Repeat:

- Policy evaluation: compute Q^{π_i} (MC or TD)
- Policy improvement:

$$\pi_{t+1}(s_t) = \arg\max_{a \in \mathcal{A}} Q(s_t, a) \text{ with probability } (1 - \varepsilon)$$

$$= \text{arbitrary action } a \text{ with probability } \frac{\varepsilon}{|\mathcal{A}|} \text{ (i.e., uniformly randomly pick } a\text{)}$$

SARSA: epsilon-greedy policy iteration + TD evaluation

Initialise epsilon-greedy policy π_0 , $\varepsilon \in [0, 1]$, *t = 0,* initial state $s_t = s_0$

Choose $a_t \sim \pi_t(s_t)$

Repeat:

- **Take** action a_t , observe (r_t, s_{t+1})
- Choose action $a_{t+1} \sim \pi_t(s_{t+1})$ // note: still using the same policy
- Update Q given tuple $(s_t, a_t, r_t, s_{t+1}, a_{t+1})$ // note: hence the name

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha \left([r_t + \gamma Q(s_{t+1}, a_{t+1})] - Q(s_t, a_t) \right)$$

SARSA: epsilon-greedy policy iteration + TD evaluation

Initialise epsilon-greedy policy π_0 , $\varepsilon \in [0,1]$, *t = 0,* initial state $s_t = s_0$

Choose $a_t \sim \pi_t(s_t)$

Repeat:

- **Take** action a_t , observe (r_t, s_{t+1})
- Choose action $a_{t+1} \sim \pi_t(s_{t+1})$ // note: still using the same policy
- Update Q given tuple $(s_t, a_t, r_t, s_{t+1}, a_{t+1})$ // note: hence the name

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha \left([r_t + \gamma Q(s_{t+1}, a_{t+1})] - Q(s_t, a_t) \right)$$

- Policy improvement:

$\pi_{t+1}(s_t) = \arg\max_{a \in \mathcal{A}} Q(s_t, a)$ with probability $(1 - \varepsilon)$

$=$ arbitrary action *a* with probability $\frac{\varepsilon}{|\mathcal{A}|}$ (i.e., uniformly randomly pick *a*)

- $s_t = s_{t+1}, \quad a_t = a_{t+1}$, *t = t + 1*

Q-learning

Initialise epsilon-greedy policy π_0 , $\varepsilon \in [0,1]$, *t = 0*, initial state $s_t = s_0$

Repeat:

- **Take** action $a_t \sim \pi_t(s_t)$, observe (r_t, s_{t+1})
- Update Q:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha \left([r_t + \gamma \max_a Q(s_{t+1}, a)] - Q(s_t, a_t) \right)$$

Because of the max operator, we don't need to sample an additional action a_{t+1}

Q-learning

Initialise epsilon-greedy policy π_0 , $\varepsilon \in [0,1]$, *t = 0,* initial state $s_t = s_0$

Repeat:

- **Take** action $a_t \sim \pi_t(s_t)$, observe (r_t, s_{t+1})
- Update Q:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha \left([r_t + \gamma \max_a Q(s_{t+1}, a)] - Q(s_t, a_t) \right)$$

- Policy improvement:

$$\pi_{t+1}(s_t) = \arg\max_{a \in \mathcal{A}} Q(s_t, a) \text{ with probability } (1 - \varepsilon)$$

$$= \text{arbitrary action } a \text{ with probability } \frac{\varepsilon}{|\mathcal{A}|} \text{ (i.e., uniformly randomly pick } a\text{)}$$

- *t = t + 1*

Q-learning vs. SARSA

SARSA: $Q(s_t, a_t) = Q(s_t, a_t) + \alpha \left([r_t + \gamma Q(s_{t+1}, a_{t+1})] - Q(s_t, a_t) \right)$

- On-policy learning
  - Direct experience
  - Learn to estimate and evaluate a policy from experience obtained from following that policy (here a_{t+1} was sampled from the same policy)

Q-learning: $Q(s_t, a_t) = Q(s_t, a_t) + \alpha \left([r_t + \gamma \max_a Q(s_{t+1}, a)] - Q(s_t, a_t) \right)$

- Off-policy learning
  - Learn to estimate and evaluate a policy using experience gathered from following a different policy (here max_a is not part of the same policy)