Part 3: Reinforcement Learning problems

EL2805 – Reinforcement Learning

Alexandre Proutiere

KTH, The Royal Institute of Technology

Objectives of this lecture

- Introduce the different classes of RL problems
- Introduce the notion of on and off-policy learning
- Introduce regret and sample complexity, the two main performance metrics for RL algorithms
- Provide a classification of RL algorithms

References

- Read chapter 1 of Sutton-Barto's book
- Introduction lectures by Levine and Silver (youtube – slides of lecture 1)

Part 3: Outline

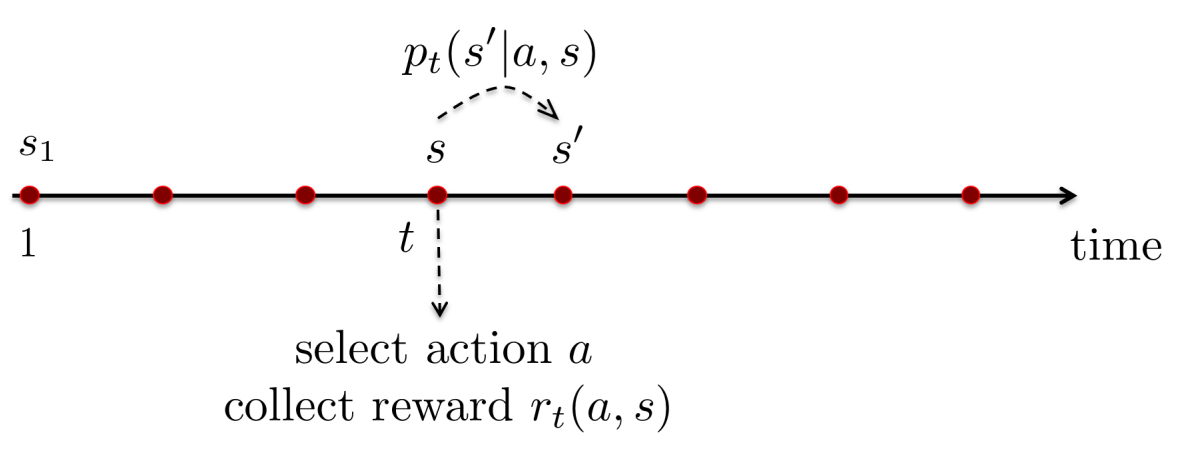
1. Different classes of RL problems
    a. Episodic problems
    b. Discounted problems
2. On vs Off-policy learning
3. Sample complexity and regret
4. Classification of RL algorithms

1. Difference classes of RL problems

Finite-time horizon MDPs $\Longrightarrow$ Episodic RL problems

Discounted infinite-horizon MDPs $\Longrightarrow$ Discounted RL problems

1.a From finite-horizon MDP to episodic RL problems

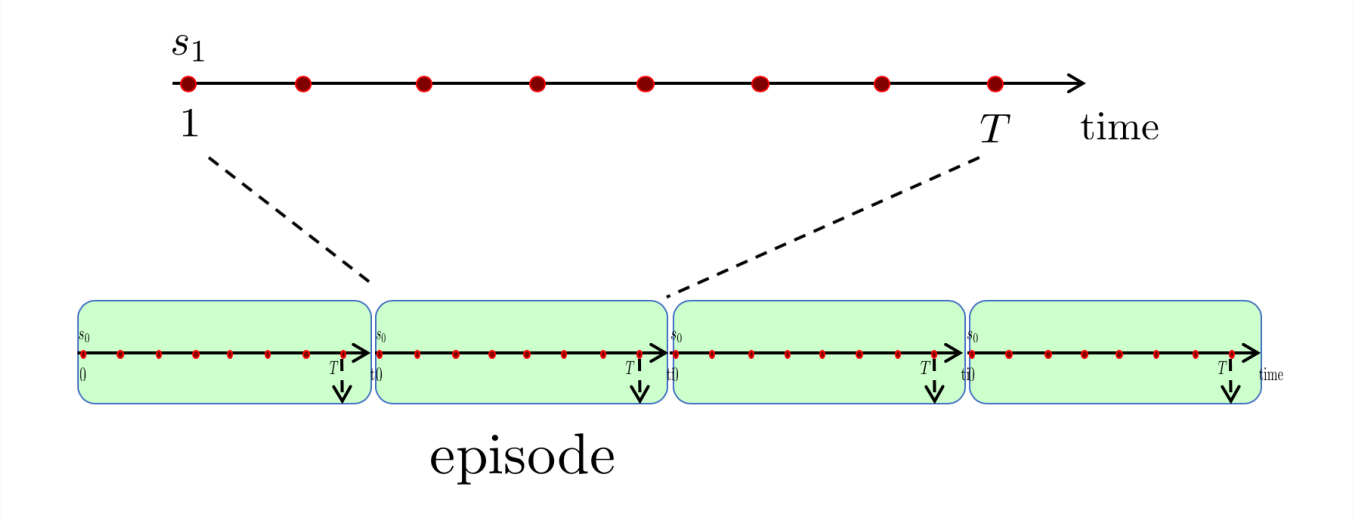


- State space: S, actions available in state $s \in S$, A_s ($A \cup_{s \in S} A_s$)
- **Unknown** transition probabilities at time t: $p_t(s'|s,a)$
- **Unknown** reward at time t: $r_t(a,s)$
- Objective: *quickly* learn a policy $\pi^\star$ maximizing over $\pi \in MD$

$$V_T^\pi(s) = \mathbb{E}\left[\sum_{t=1}^{T} r_t(s_t^\pi, a_t^\pi) | s_1^\pi = s\right]$$

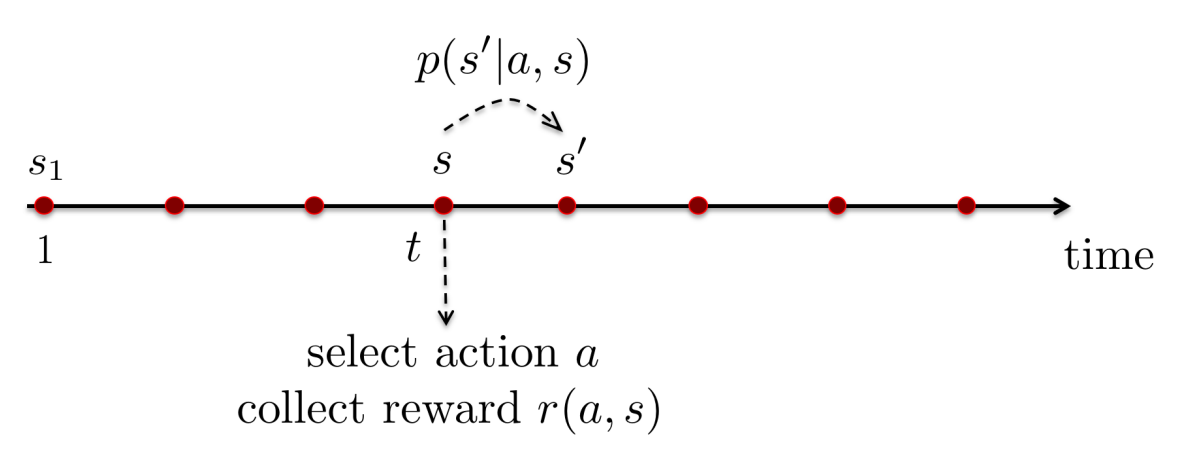
from the *data*

Episodic RL problems



- Data: K episodes of length T (actions, states, rewards)
- Learning algorithm π : data $\mapsto \pi_K \in MD$
- Performance of π: how close π_K is from the optimal policy $\pi^\star$

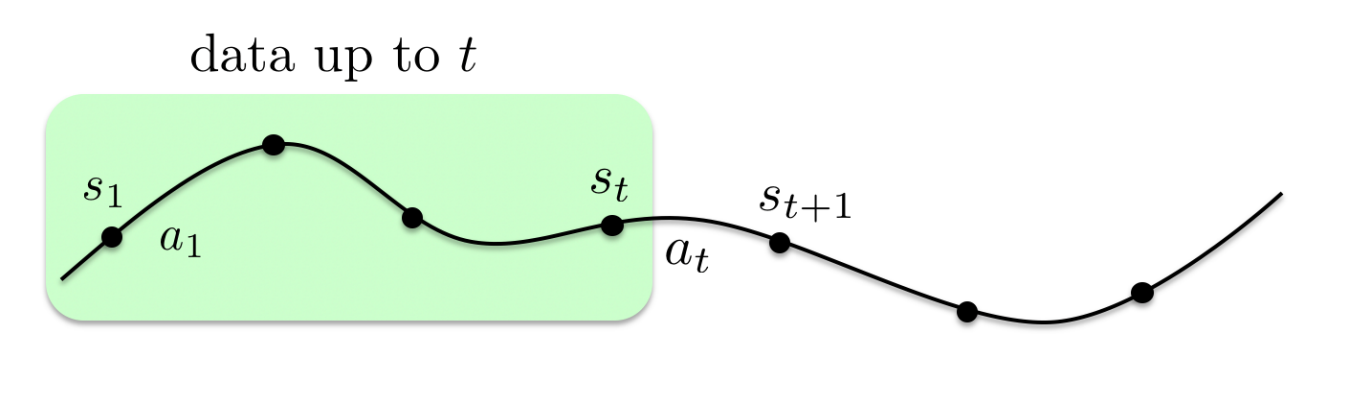
1.b From Infinite-horizon discounted MDP to discounted RL problems



- **Unknown** stationary transition probabilities $p(s'|s,a)$ and rewards $r(s,a)$, uniformly bounded: $\forall a, s,\ |r(s,a)| \leq 1$
- Objective: for a given discount factor $\lambda \in [0,1)$, from the data, find a policy $\pi^\star \in MD$ maximizing (over all possible policies)

$$V^\pi(s) = \mathbb{E}\left[\sum_{t=1}^{T} \lambda^{t-1} r(s_t^\pi, a_t^\pi,)|s_1^\pi = s\right]$$

Discounted RL problems



- Data: trajectory of the system up to time t (actions, states, rewards)
- Learning algorithm π : data $\mapsto \pi_t \in MD$
- Performance of π: how close π_t is from the optimal policy $\pi^\star$

2. On vs. Off-policy learning

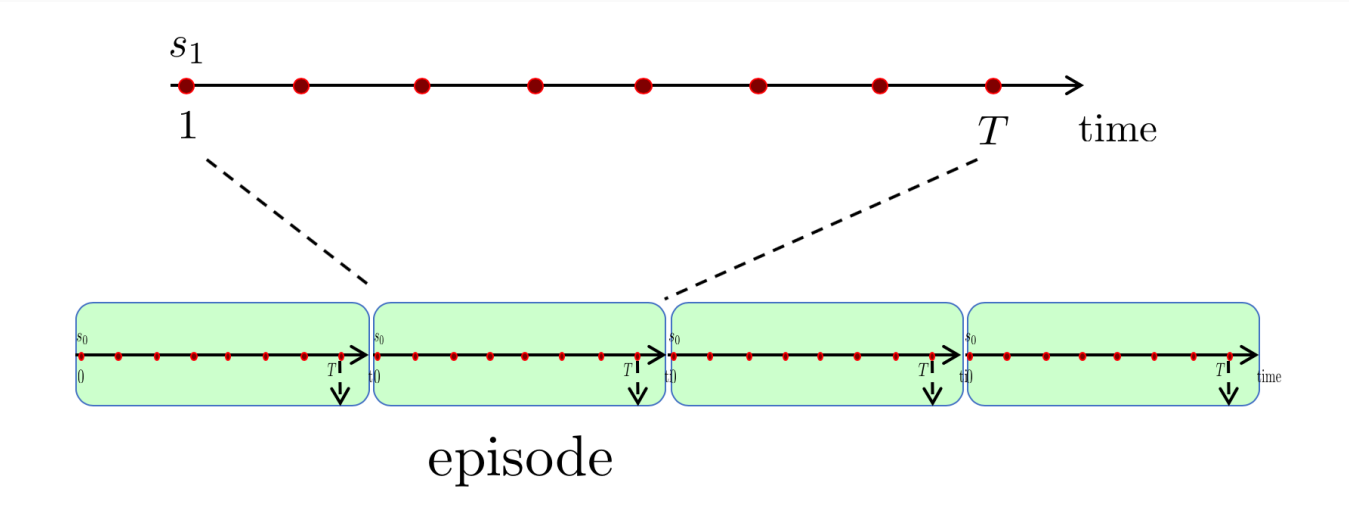
An **off-policy** learner learns the value of the optimal policy independently of the agent's actions.
The policy used by the agent is often referred to as the **behavior** policy, and denoted by π_b.

An **on-policy** learner learns the value of the policy being carried out by the agent. The policy used by the agent is computed from the previous collected data. It is an *active learning* method as the gathered data is controlled.

Off-policy learning

Data: the observations made by the agent along the trajectory of the dynamical system under the behavior policy π_b.

Episodic RL problems:
The K episodes are generated under π_b.



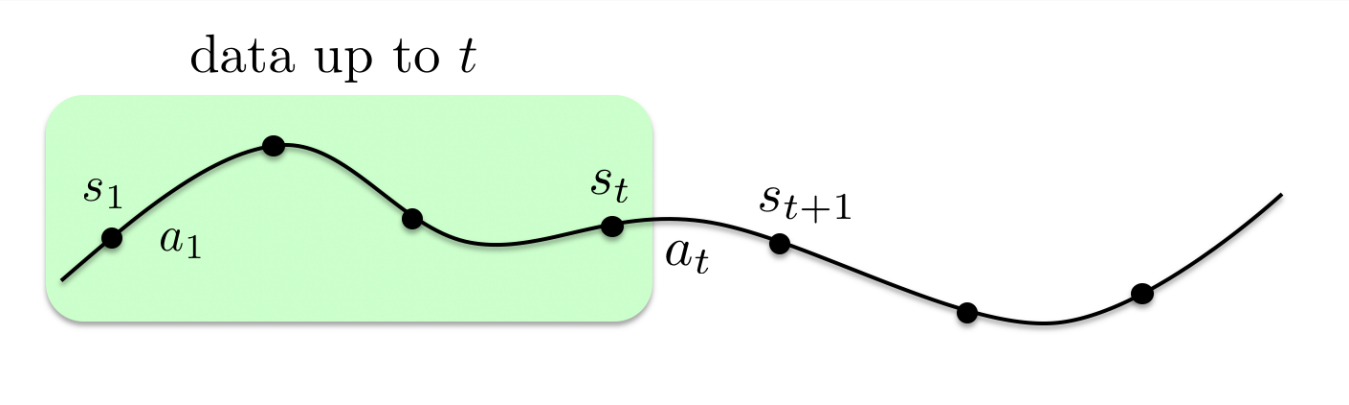
Off-policy learning

Data: the observations made by the agent along the trajectory of the dynamical system under the behavior policy π_b.

Discounted RL problems:

$a_t \sim \pi_b(s_t)$

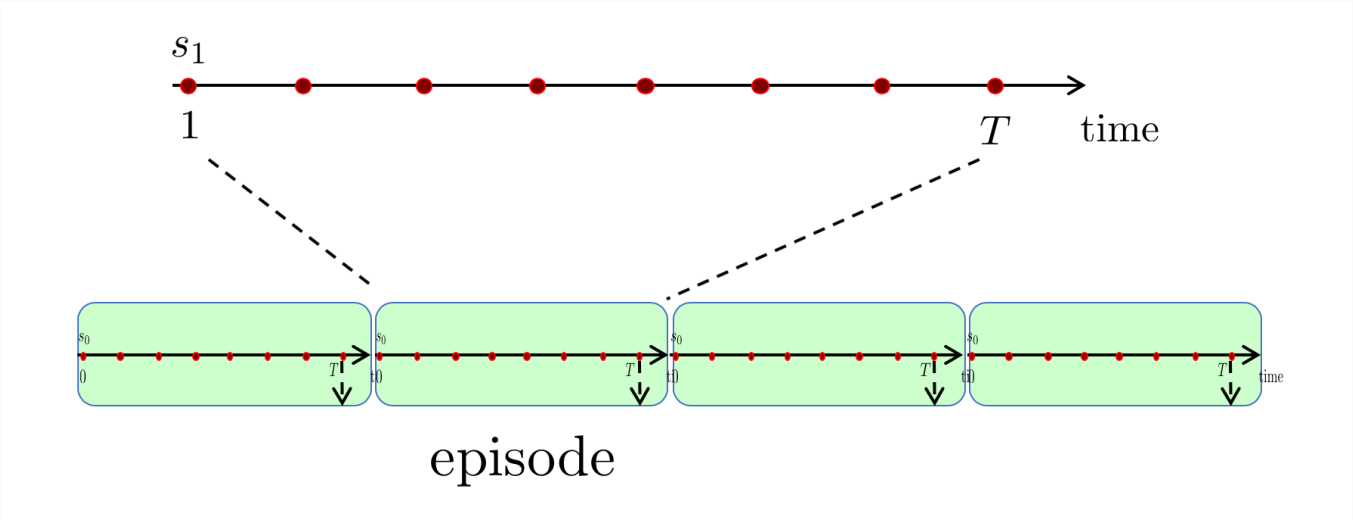
$s_{t+1} \sim p(\cdot|s_t, a_t)$



On-policy learning

Data: the observations made by the agent in episode k are generated under the policy π_k. π_k is the policy supposed to converge to $\pi^\star$.

Episodic RL problems:
Episode k is generated under π_k.



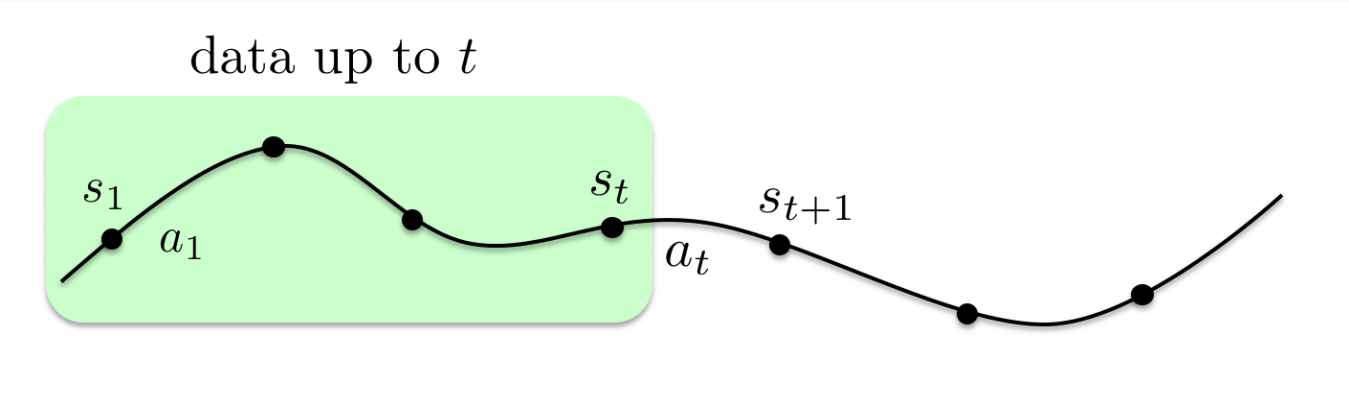
On-policy learning

Data: the observations made by the agent in step t along the trajectory of the dynamical system are generated under the policy π_t. π_t is the policy supposed to converge to $\pi^\star$.

Discounted RL problems:

$a_t \sim \pi_t(s_t)$

$s_{t+1} \sim p(\cdot|s_t, a_t)$



Examples

1. Learning to play chess from a dataset consisting of past historical games.
Off-policy learning

2. Build your robot, and teach it to walk.
On-policy learning (could also be off-policy)

Exploration in RL problems

Exploration: RL is like trial-and-error: all actions in all states should be tested.
Off-policy learning: π_b should explore all actions.
On-policy learning: π_t should explore all actions, at **any** time t.

Why always exploring?

Exploration in RL problems

Exploration: RL is like trial-and-error: all actions in all states should be tested.
Off-policy learning: π_b should explore all actions.
On-policy learning: π_t should explore all actions, at **any** time t.

Why always exploring? Because at time t, π_t is decided inspecting a finite amount of data (a trajectory of length $t-1$)

$$\mathbb{P}[\pi_t = \pi^\star] = 1 - \epsilon < 1$$

Errors always occur (with decreasing probability as the data size grows)

Example

No state, two actions:

'1' with Gaussian reward $\sim \mathcal{N}(\mu_1, 1)$

'2' with Gaussian reward $\sim \mathcal{N}(\mu_2, 1)$

$\mu_1 > \mu_2$: the optimal strategy is to play '1'.

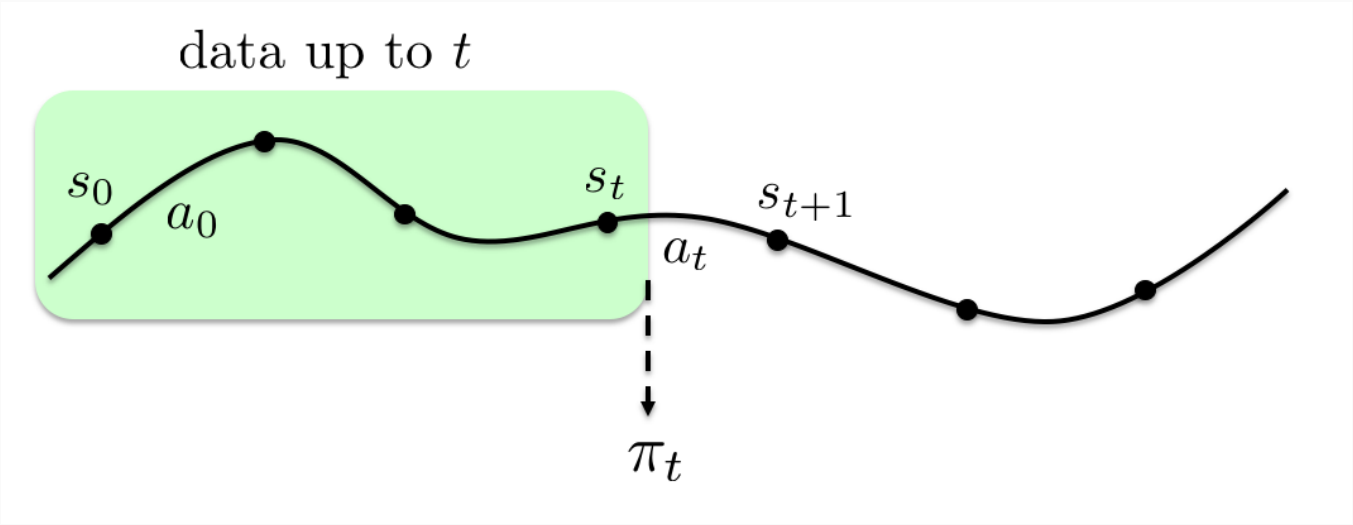
Up to time $2t$, '1' and '2' have been played t times. $\hat{S}_1 = \frac{1}{t}\sum_{i=1}^{t} R_{1,i}$, $\hat{S}_2 = \frac{1}{t}\sum_{i=1}^{t} R_{2,i}$.
Error probability:

$$\mathbb{P}\left[\hat{S}_1 < \hat{S}_2\right] = \frac{1}{\sqrt{4\pi}} \int_{-\infty}^{\sqrt{n}(\mu_2-\mu_1)} e^{-v^2/4} dv$$

Exploration vs. exploitation trade-off

Off-policy learning: works if the behavior policy explores.

On-policy learning: works e.g. with ϵ-**greedy** policies



$$\pi_t(s) = \begin{cases} \text{best empirical action in } s & \text{w.p. } 1-\epsilon \\ \text{a random action} & \text{w.p. } \epsilon \end{cases}$$

In RL, to enforce exploration, we generally use randomized policies.

3. Sample complexity and regret

How can we measure the performance of various learning algorithms?

Sample complexity. Defined as the time required to find an approximately optimal policy. Well defined for any kind of RL problems. A Probably Approximately Correct (PAC) framework.

1. Episodic RL. An on-policy algorithm returns, after the end of the $(k-1)$-th episode, a policy π_k to be applied in the k-th episode.

> *The sample complexity of π is the minimum number SP^π of episodes such that for all $k \geq SP^\pi$, π_k is ϵ-optimal with probability at least $1-\delta$, i.e., for $k \geq SP^\pi$,*
>
> $$\mathbb{P}\left[V_T^{\pi_k} \geq V_T^{\pi^\star} - \epsilon\right] \geq 1 - \delta$$

Sample complexity

Sample complexity. Defined as the time required to find an approximately optimal policy. Well defined for any kind of RL problems.

2. Discounted and ergodic RL. An on-policy algorithm π returns, after the end of the $(t-1)$-th step, a policy π_t to be applied in the t-th step.

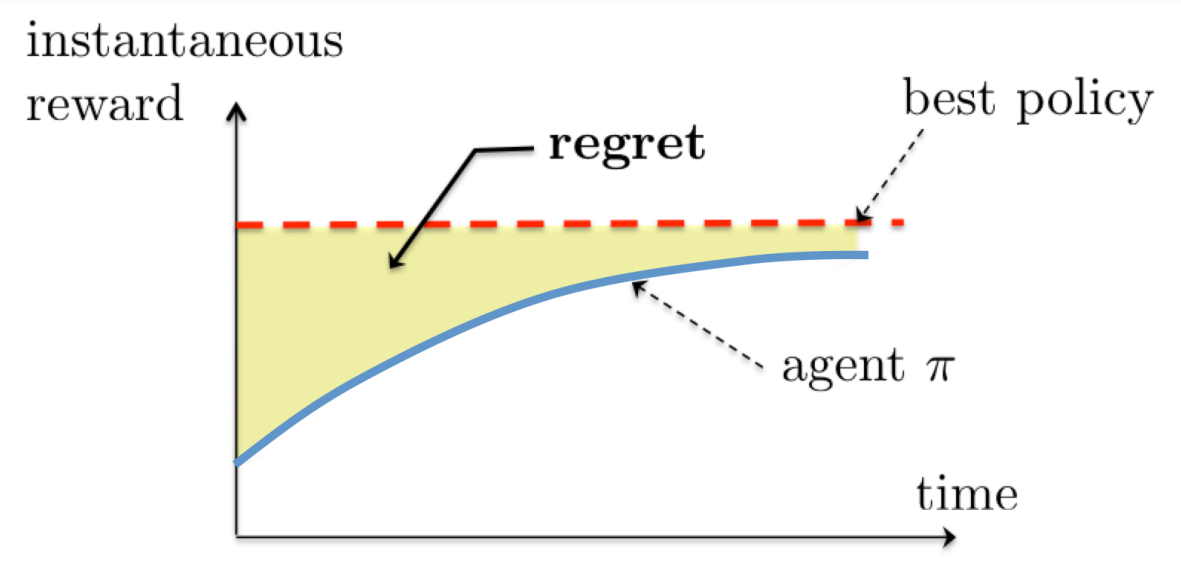
> *The sample complexity of π is the minimum number SP^π of steps such that for any $t \geq SP^\pi$, π_t is ϵ-optimal when starting in the current state s_t^π with probability at least $1-\delta$, i.e., for any $t \geq SP^\pi$,*
>
> $$\mathbb{P}\left[V^{\pi_t}(s_t^\pi) \geq V^{\pi^\star}(s_t^\pi) - \epsilon\right] \geq 1-\delta$$

Sample complexity can be defined for off-policies analogously.

Regret

A more appropriate performance metrics for on-policies? Capture the exploration vs. exploitation trade-off.

Regret of an algorithm π. Defined as the difference between the cumulative reward of the optimal policy and that gathered by π.



Regret

Regret of an algorithm π. Defined as the difference between the cumulative reward of the optimal policy and that gathered by π.

1. Episodic RL. An on-policy algorithm π returns, after the end of the $(k-1)$-th episode, a policy π_k to be applied in the k-th episode.

The regret of π after K episodes is:

$$R^{\pi}(K) = KV_T^{\pi^\star} - \sum_{k=1}^{K} \mathbb{E}[V_T^{\pi_k}]$$

Regret

Regret of an algorithm π. Defined as the difference between the cumulative reward of the optimal policy and that gathered by π.

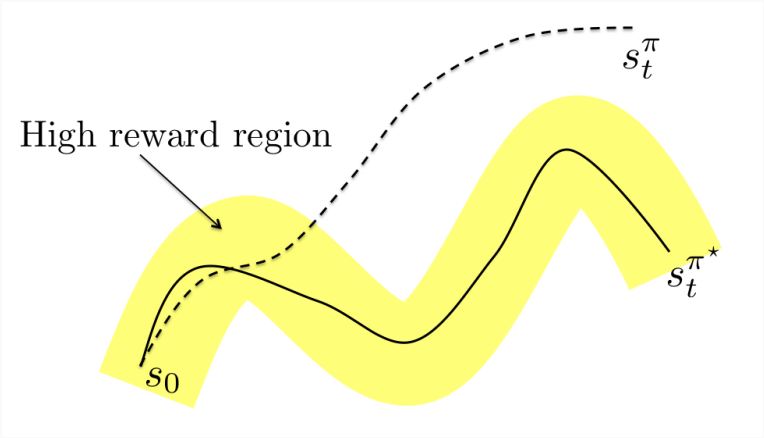
2. Discounted RL. An on-policy algorithm π returns, after the end of the $(t-1)$-th step, a policy π_t to be applied in the t-th step. The regret is difficult to define as the cumulative reward is bounded (no scaling in T)

A tentative definition. The regret of π after T steps is:

$$R^{\pi}(T) = \sum_{t=1}^{T} \left(V^{\pi^\star}(s_t^{\pi^\star}) - V^{\pi_t}(s_t^{\pi}) \right)$$

A remark on these definitions

For discounted RL problems, along which trajectory should we define the performance metrics?



Sample complexity:

$$\mathbb{P}\left[V^{\pi_t}(s_t^\pi) \geq V^{\pi^\star}(s_t^\pi) - \epsilon\right] \geq 1-\delta$$

Following the trajectory of the algorithm can be misleading: Due to exploration, we can end up in states with very low rewards, and being optimal from there does not mean much – especially for discounted problems.

Fundamental performance limits

- Performance metrics:
  - Sample complexity: amount of data required to learn a near-optimal policy (for off and on-policy learning)
  - Regret: cumulative rewards loss due to the need of learning, quantifies the exploration-exploitation trade-off (for on-policy learning)
- Fundamental performance limits: e.g. regret
  Denote by $M = (\mathcal{S}, \mathcal{A}, (p(\cdot|s,a), r(s,a))_{s\in S, a\in A})$ the MDP of interest. Two types of regret lower bounds:

  Problem-specific lower bound: for all M, $\forall$ algorithm π, $R^\pi(T) \geq F(M,T)$
  (most often in the form of $\liminf_{T\to\infty} \frac{R^\pi(T)}{f(T)} \geq c(M)$)
  Minimax lower bound: $\exists M$ such that $\forall$ algorithm π, $\forall T$, $R^\pi(T) \geq G(S,A,T,...)$

Episodic RL problems: State-of-the-art

- **Regret minimization**
  - Minimax lower bound $\Omega(\sqrt{TSAK})$

    No problem-dependent lower bound is derived so far

- **Sample complexity**
  - Minimax lower bound $\Omega\left(\frac{K^2SA}{\varepsilon^2}\log\delta^{-1}\right)$

    No problem-dependent lower bound is derived so far

Discounted RL problems: State-of-the-art

No regret analysis for this class of RL problems.

- **Sample complexity:**
  - Minimax lower bound: $\Omega\left(\frac{SA}{(1-\lambda)^3\varepsilon^2}\log\delta^{-1}\right)$
    No problem-dependent lower bound is derived so far

4. Classification of RL algorithms

(To fix ideas, discounted RL problems only are discussed below.)

For a given MDP, the **key quantities** are:

- $M = (r(s,a), p(\cdot|s,a), s \in \mathcal{S}, a \in \mathcal{A})$
- Value function: $V^\star(s) = \max_a (r(s,a) + \lambda \sum_j p(j|s,a) V^\star(j))$
- Q-function: $Q(s,a) = r(s,a) + \lambda \sum_j p(j|s,a) V^\star(j)$
- For a policy π, state value function:

$$V^\pi(s) = \mathbb{E}_{a \sim \pi(s)}[r(s,a) + \lambda \sum_j p(j|s,a) V^\pi(j)]$$

(state, action) value function:

$$Q^\pi(s,a) = r(s,a) + \lambda \sum_j p(j|s,a) V^\pi(j)$$

The knowledge of some of these quantities is enough to find optimal policies.

Classification of RL algorithms

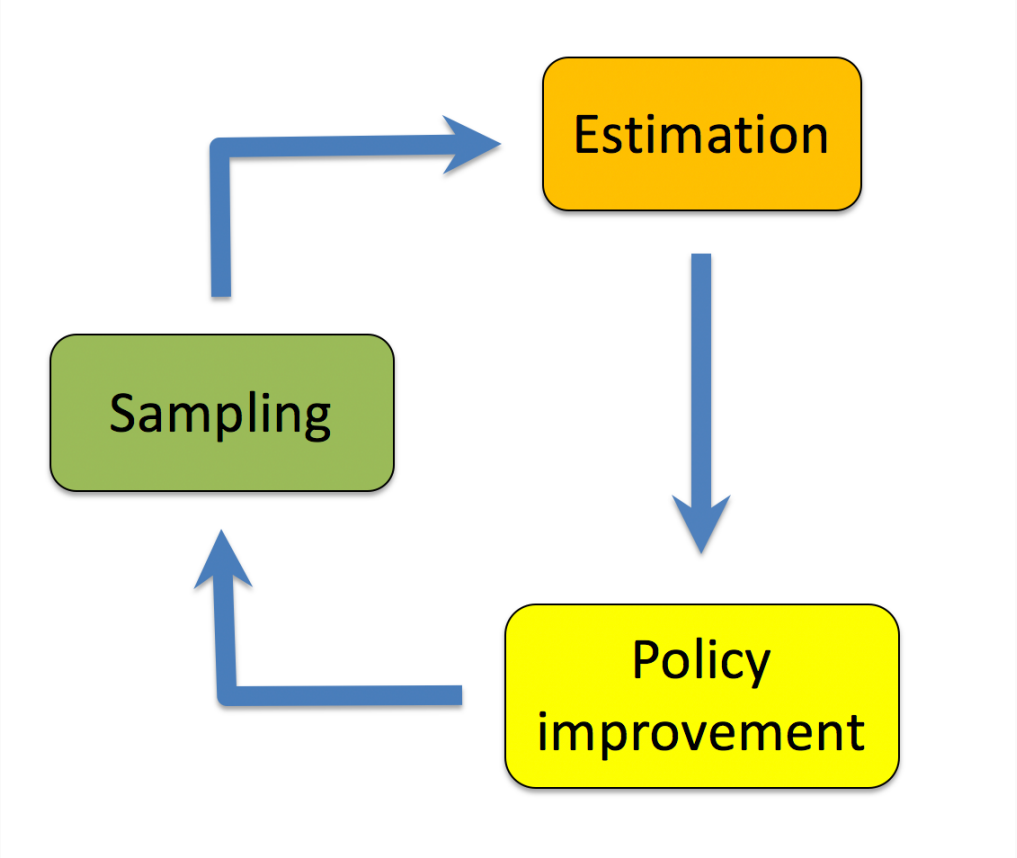
Knowing M: Solve Bellman's equations to get the value function and the optimal policy.

Knowing $V^\star$ or Q: the optimal decision in state s is $a(s) \in \arg\max_b Q(s, b)$

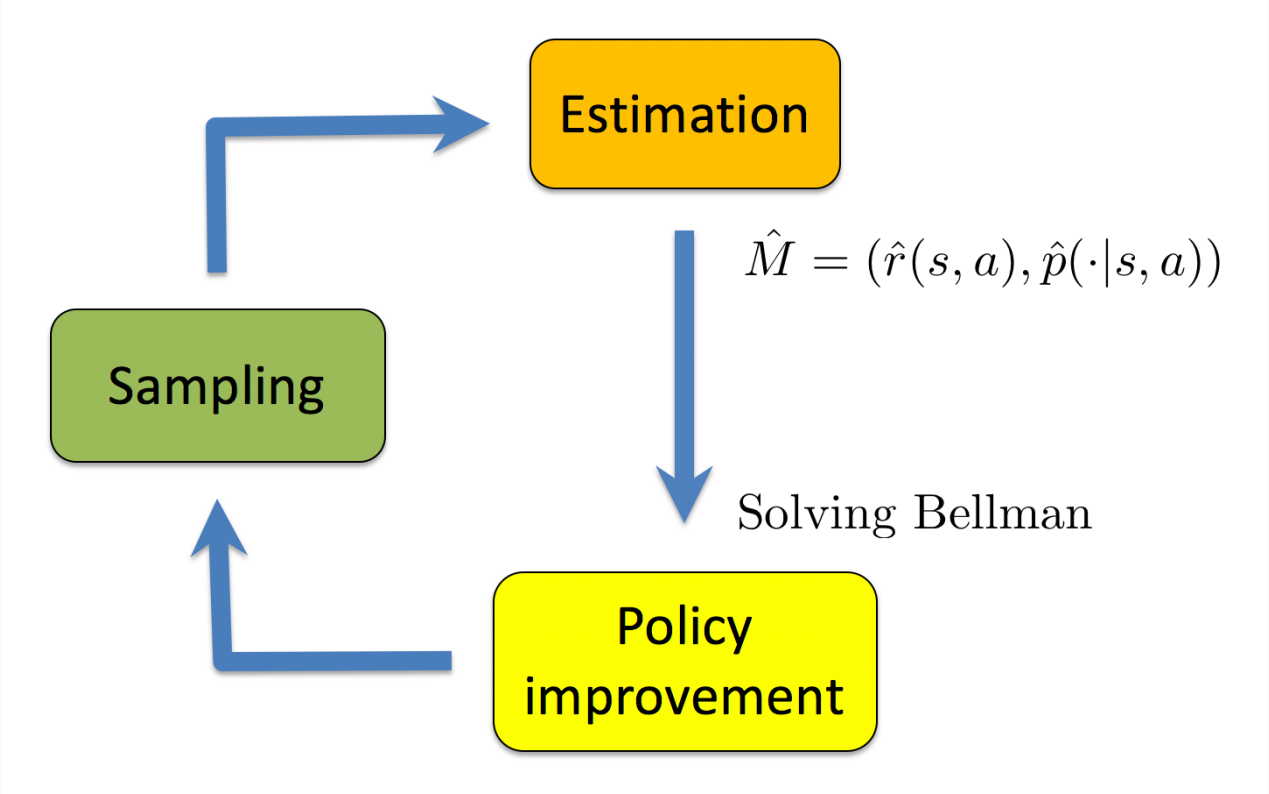
Knowing V^π or Q^π: the policy can be improved (i.e., as in the PI algorithm). π' is better than π if $\pi'(s) \in \arg\max_a Q^\pi(s, a)$

Without the knowledge of these quantitities, RL algorithms rely on **sampling** (generate trajectories to estimate them).

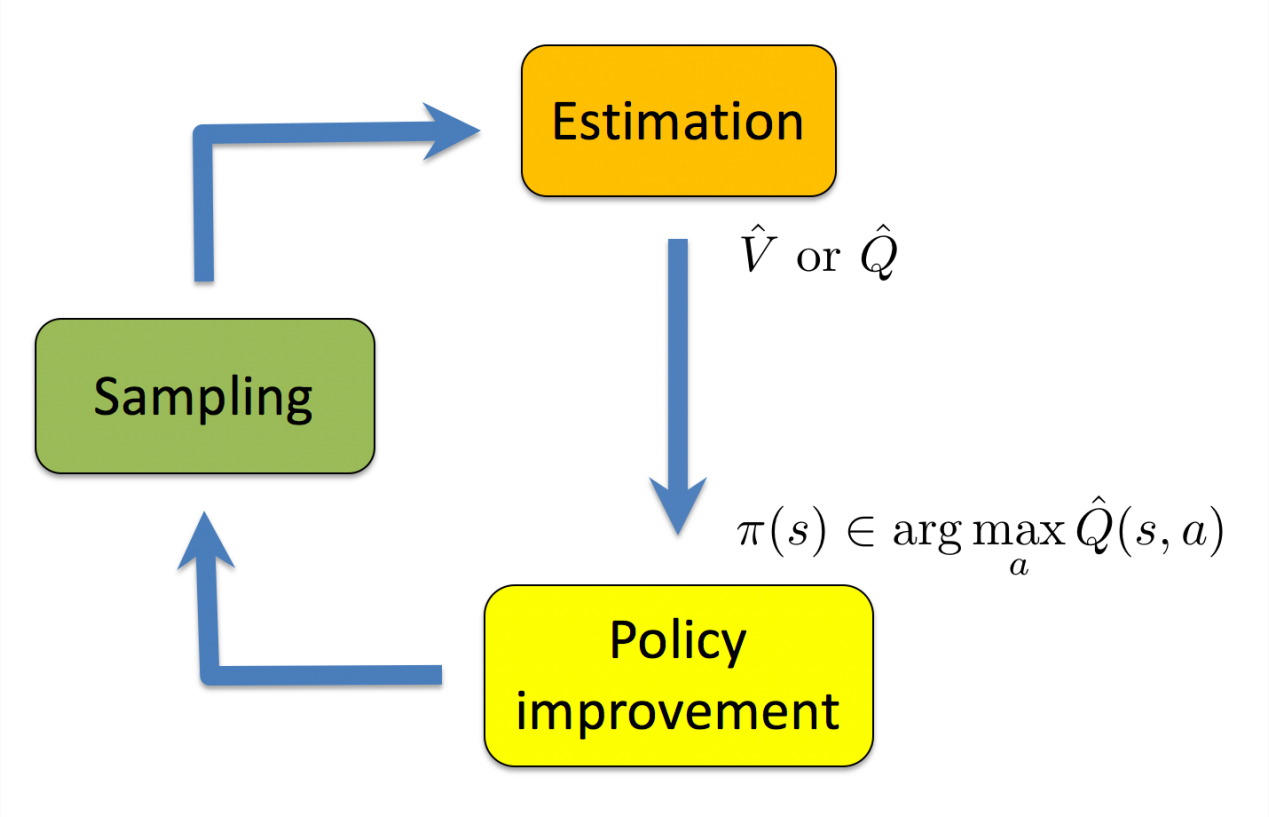
RL algorithms



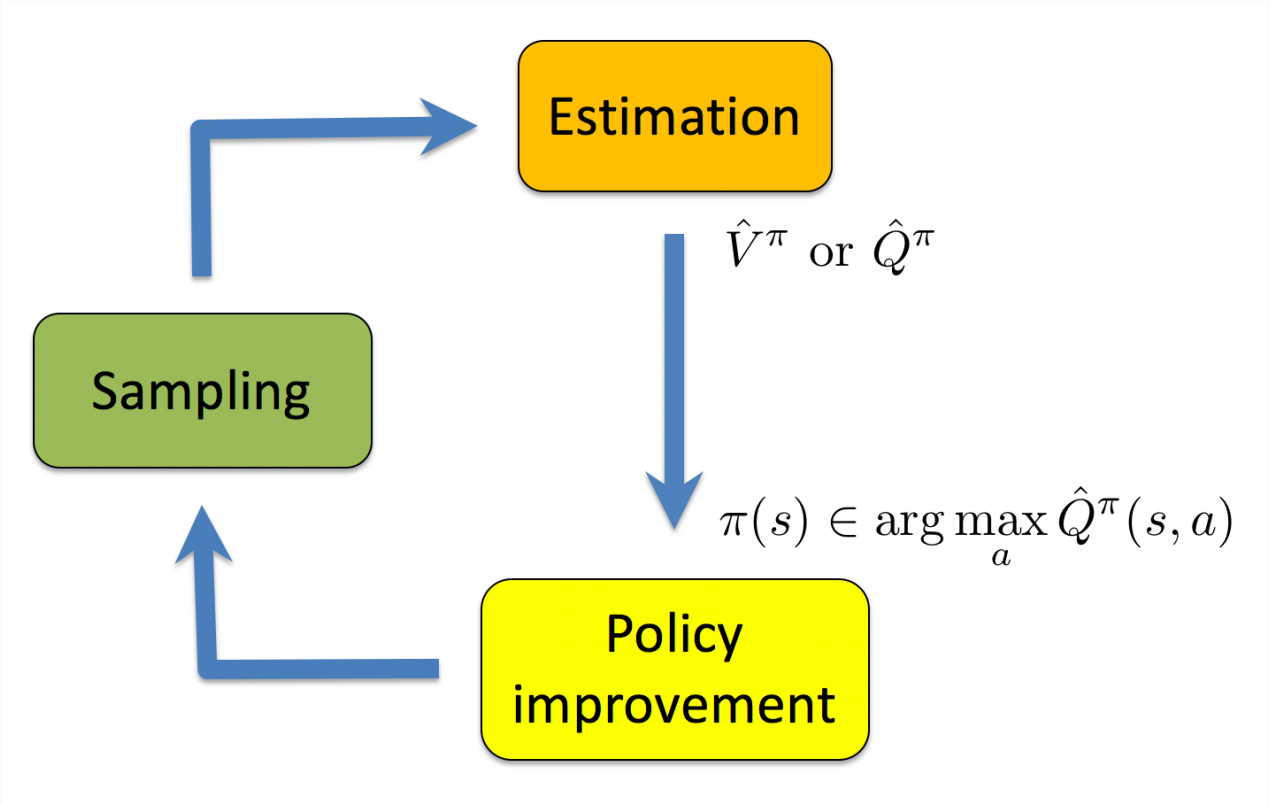
RL algorithms: model-based



RL algorithms: value function-based (parts 4-5)

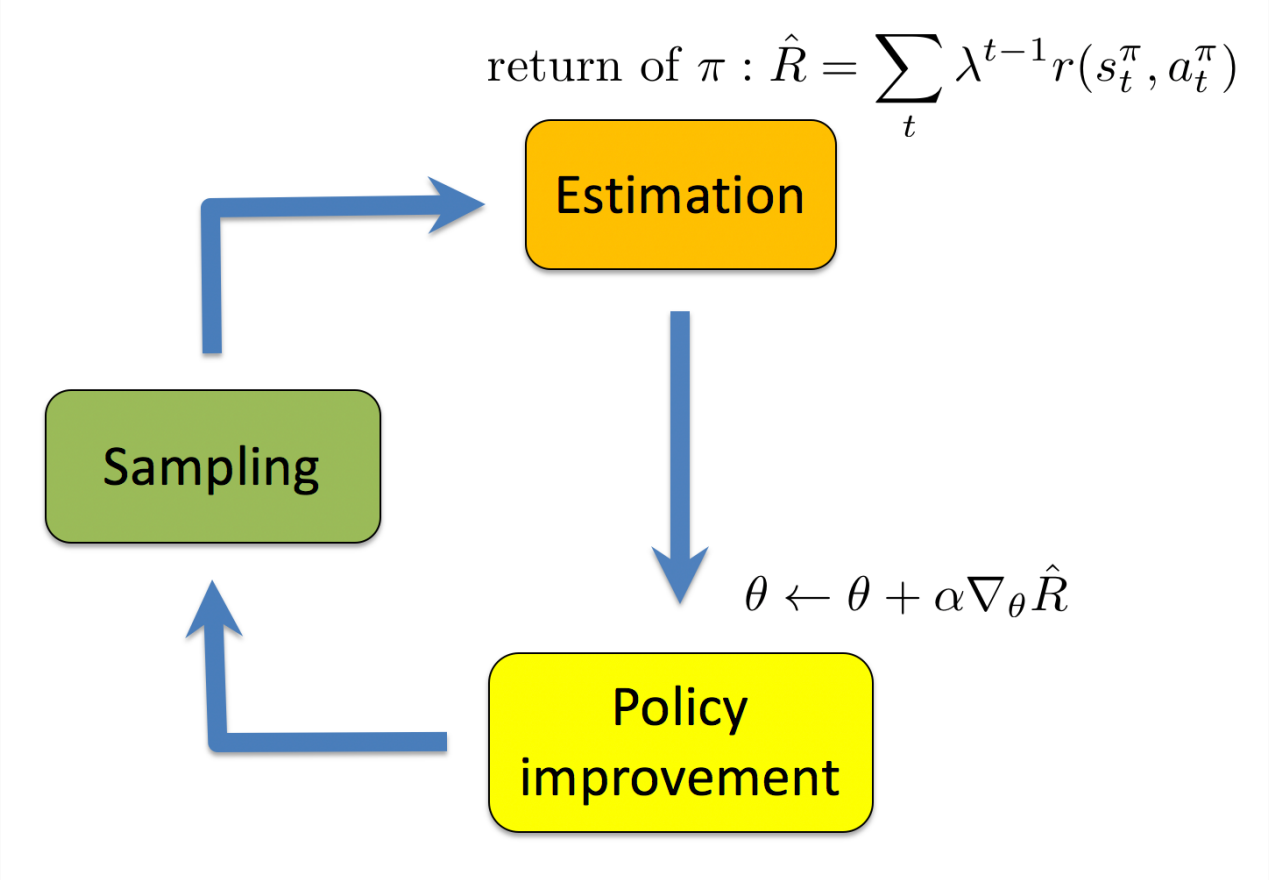


RL algorithms: value function-based (parts 4-5)



RL algorithms: policy-based (part 6)

Parametrized policies: $\pi = \pi_\theta$



RL algorithms: actor-critic (part 8)

Parametrized policies: $\pi = \pi_\theta$