Summary

EL2805 - Reinforcement Learning

Alexandre Proutiere and Cristian Rojas

KTH, The Royal Institute of Technology

Optimal control of systems with known dynamics and rewards

- MDPs: a generic model for controlled Markovian systems
  An MDP is defined through:

$$\{T, S, (A_s, p_t(\cdot|s,a), r_t(s,a), 1 \le t \le T, s \in S, a \in A_s)\}$$

- Finite-time horizon MDPs
  - Policy evaluation: computing the average reward of a policy $\pi = (\pi_1, \dots, \pi_T)$ starting at s can be done using DP: $u_T(s) = r_T(s, \pi_T(s))$, and for $t = T-1, \dots, 1$,

$$u^\pi_{t-1}(s_{t-1}) = r_{t-1}(s_{t-1}, a) + \sum_{j \in S} p_{t-1}(j|s_{t-1}, a) u^\pi_t(j)$$

    We obtain: $V^\pi_T(s) = u^\pi_1(s)$

Part 2: MDPs

- Value function and optimal policy: $V_T^\star(s) = \sup_{\pi \in MD} V_T^\pi(s)$ obtained by solving **Bellman's equations** with **Dynamic Programming**:
  For all s_T, $u_T^\star(s_T) = \max_a r_T(s_T, a)$
  For all $t = T-1, T-2, \ldots, 1$

$$u_t^\star(s_t) = \max_{a \in A_{s_t}} \underbrace{\left[r_t(s_t, a) + \sum_{j \in S} p_t(j|s_t, a) u_{t+1}^\star(s_t, a, j) \right]}_{Q_t(s_t,a) \text{ optimal reward from } t \text{ if } a \text{ selected}}$$

  An optimal policy π is obtained by selecting $\pi_t(s_t)$ at time t such that

$$Q_t(s_t, \pi_t(s_t)) = \max_{a \in A_{s_t}} Q_t(s_t, a)$$

Part 2: MDPs

- Discounted intinite-horizon MDPs
  - Policy evaluation: computing the average reward of a stationary policy $\pi = (\pi_1, \pi_1, \ldots)$ starting at s can be done solving the linear system:

$$\forall s, \quad V^\pi(s) = r(s, \pi_1(s)) + \lambda \sum_j p(j|s, \pi_1(s)) V^\pi(j)$$

  - Value function and optimal policy: $V^\star(s) = \sup_{\pi \in MD} V^\pi(s)$ obtained by solving **Bellman's equations** through **VI or PI algorithm**:

$$\forall s, \quad V^\star(s) = \max_{a \in A_s} \underbrace{\left[r(s,a) + \lambda \sum_{j \in S} p(j|s,a) V^\star(j) \right]}_{Q(s,a) \text{ optimal reward from state } s \text{ if } a \text{ selected}}$$

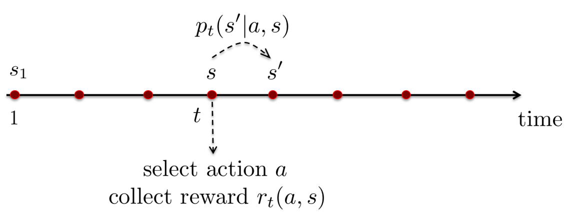
An optimal policy π is stationary $\pi = (\pi_1, \pi_1, \ldots)$ where $\pi_1 \in MD_1$ is defined by: for any s,

$$\pi_1(s) = \arg\max_{a \in A_s} Q(s,a)$$

Q is referred to as the Q-function.

Part 3: RL problems

From finite-horizon MDP to episodic RL problems

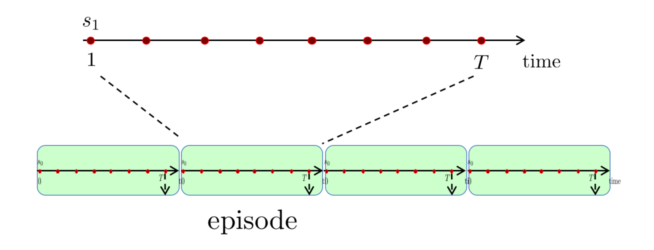


- State space: S, actions available in state $s \in S$, A_s $(A \cup_{s \in S} A_s)$
- **Unknown** transition probabilities at time t: $p_t(s'|s,a)$
- **Unknown** reward at time t: $r_t(a,s)$
- Objective: *quickly* learn a policy $\pi^\star$ maximizing over $\pi \in MD$

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

from the *data*

Part 3: RL problems

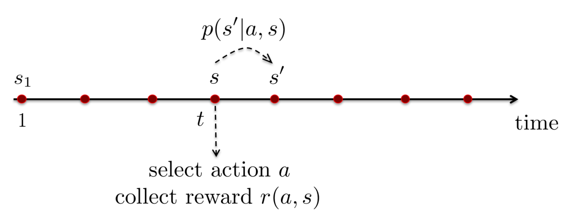


Episodic RL problems

- 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 $\pi^\star$

Part 3: RL problems

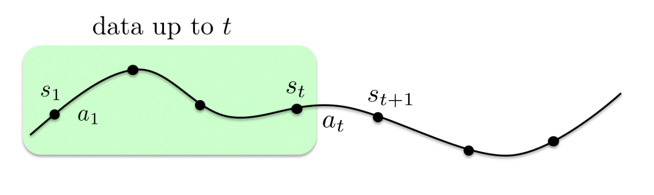
From Infinite-horizon discounted MDP to discounted RL problems



- **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)

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

Part 3: RL problems



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 $\pi^\star$

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.

Part 4: MC methods and TD learning

Model-free policy evaluation:
How can we evaluate the value function of a policy π by observing trajectories or episodes generated under π?

- Monte Carlo methods: Evaluating policies through sampling
- Robbins-Monro's stochastic approximation algorithm
- TD (Time Difference) learning: Evaluating policies through sampling and bootstrapping

Robbins-Monro Algorithm. If x is selected by the algorithm, one observes $Y(x)$ a r.v. bounded in magnitude by G and such that $\mathbb{E}[Y(x)] = h(x)$. The algorithm finds the root of h under some conditions ...

Robbins-Monro Algorithm:

1. **Initialization:** $x^{(0)}$
2. **Iterations:** for $k \geq 0$,

$$x^{(k+1)} = x^{(k)} - \alpha_k Y(x^{(k)}).$$

Part 4: MC methods and TD learning

Model-free policy evaluation can be made using:

- Monte Carlo methods in episodic RL problems. After each episode k, if s appears in the episode:

  $$V^{(k)}(s) = V^{(k-1)}(s) + \frac{1}{k}(G_k(s) - V^{(k-1)}(s))$$

  where $G_k(s)$ is the return observed from state s in episode k.
  Convergence almost sure towards V^π.
  High variance, slow convergence.
- TD learning in episodic and discounted RL problems. After each step t, the estimated value of state s_t only is updated:

  $$V^{(t+1)}(s_t) = V^{(t)}(s_t) + \alpha_{n_{s_t}^{(t)}} \left(r_t + \lambda V^{(t)}(s_{t+1}) - V^{(t)}(s_t)\right)$$

  Convergence almost sure to V^π for decreasing step sizes or to a neighborhood of V^π in expectation for fixed step size.
  Generally better than MC methods.

MC control with ϵ-soft policies (on-policy algorithm)

Monte Carlo for ϵ-soft policies:

1. **Initialization:** π ϵ-soft, $\forall s, a$, $Q(s,a) = 0$
   Returns$(s,a) \leftarrow$ empty list, $\forall s, a$
2. **Iterations:** for episode $i = 1, \ldots, n$
   generate $\tau_i = (s_{1,i}, a_{1,i}, r_{1,i}, \ldots, s_{T_i,i}, a_{T_i,i}, r_{T_i,i})$ under π
   $G = 0$
   for $t = T_i, T_i - 1, \ldots, 1$:
   a. $G = r_{t,i} + \lambda G$
   b. Unless $(s_{t,i}, a_{t,i})$ appears before in the episode:
      append G to Returns$(s_{t,i}, a_{t,i})$
      $Q(s_{t,i}, a_{t,i}) \leftarrow$ average(Returns$(s_{t,i}, a_{t,i})$)
      update π: $a \sim \pi(s_{t,i}, \cdot)$ such that

$$a = \begin{cases} \arg\max_b Q(s_{t,i}, b) & \text{w.p. } 1 - \epsilon, \\ \text{uniform}(\mathcal{A}_{s_{t,i}}) & \text{w.p. } \epsilon \end{cases}$$

Q-learning algorithm

Parameter. Step sizes (α_t)

1. Initialization. Select a Q-function $Q^{(0)} \in \mathbb{R}^{S\times A}$

2. Observations. (s_t, a_t, r_t, s_{t+1}) under the behavior policy π_b

3. Q-function improvement. For $t \geq 0$. Update the estimated Q-function as follows: $\forall s, a$,

$$Q^{(t+1)}(s,a) = Q^{(t)}(s,a)$$

$$+ 1_{(s_t,a_t)=(s,a)} \alpha_{n^{(t)}(s_t,a_t)} \left[r_t + \lambda \max_{b \in \mathcal{A}} Q^{(t)}(s_{t+1}, b) - Q^{(t)}(s_t, a_t) \right]$$

where $n^{(t)}(s,a) := \sum_{m=1}^{t} 1[(s,a) = (s_m, a_m)]$.

Part 5: Q-leanring and SARSA

SARSA with ϵ-greedy

Parameter. Step sizes (α_t)

1. Initialization. Select a Q-function $Q^{(0)} \in \mathbb{R}^{S \times A}$

2. Observations. $(s_t, a_t, r_t, s_{t+1}, a_{t+1})$ under π_t ϵ-greedy w.r.t. $Q^{(t)}$

3. Q-function improvement. For $t \geq 0$. Update the estimated Q-function as follows: $\forall s, a,$

$$Q^{(t+1)}(s, a) = Q^{(t)}(s, a)$$

$$+ 1_{(s_t, a_t) = (s, a)} \alpha_{n^{(t)}(s_t, a_t)} \left[r_t + \lambda Q^{(t)}(s_{t+1}, a_{t+1}) - Q^{(t)}(s_t, a_t) \right]$$

where $n^{(t)}(s, a) := \sum_{m=1}^{t} 1[(s, a) = (s_m, a_m)]$.

Part 6: Policy gradients

Policy gradient algorithms assume that:

- Policies are parametrized: $\pi \in \{\pi_\theta : \theta \in \Theta\}$
- Maximize $J(\theta) = \mathbb{E}_{s_1 \sim p}[V^{\pi_\theta}(s_1)]$
- $J(\theta)$ must be smooth in θ, and preferably concave!
- PG algorithms are SGD algorithms to find a maximizer of J

Policy gradient theorems:

- Episodic RL:
  $\nabla J(\theta) = \mathbb{E}_{\pi_\theta}\left[\left(\sum_{t=1}^T \nabla \log \pi_\theta(s_t, a_t)\right)\left(\sum_{t=1}^T r(s_t, a_t)\right)\right]$
- ∞-horizon RL:
  $\nabla J(\theta) = \frac{1}{1-\lambda}\mathbb{E}_{s \sim \rho_\theta, a \sim \pi_\theta(s,\cdot)}\left[\nabla \log \pi_\theta(s,a) Q^{\pi_\theta}(s,a)\right]$

Stochastic Gradient Descent algorithm

- Let $f : \mathcal{C} \to \mathbb{R}$ be a convex function.
- Unbiased estimator of the gradient: $g(x)$ is a r.v. such that $\nabla f(x) = \mathbb{E}[g(x)]$

SGD Algorithm:

1. **Initialization:** $x^{(0)}$
2. **Iterations:** for $k \geq 0$,

$$x^{(k+1)} = x^{(k)} - \alpha_k g(x^{(k)})$$

REINFORCE algorithm. If we generate an episode following π_θ: $(s_1 = s, a_1, r_1 \ldots s_T, a_T, r_T)$ where $r_t = r(s_t, a_t)$ is the observed reward, the quantity $\left(\sum_{t=1}^{T} \nabla \log \pi_\theta(s_t, a_t)\right)\left(\sum_{t=1}^{T} r_t\right)$ is an unbiased estimator of $\nabla J(\theta)$.

REINFORCE Algorithm:

1. **Initialization:** select $\theta^{(0)}$ arbitrarily
2. **Iterations:** For all $k \geq 0$, for episode k, generate a trajectory under $\pi_{\theta^{(k)}}$: $(s_{1,k} = s, a_{1,k}, r_{1,k}, \ldots s_{T,k}, a_{T,k}, r_{T,k})$
   Update the parameter

$$\theta^{(k+1)} = \theta^{(k)} + \alpha_k \left(\sum_{t=1}^{T} \nabla \log \pi_\theta(s_{t,k}, a_{t,k})\right)\left(\sum_{t=1}^{T} r_{t,k}\right)$$

Part 6: Policy gradients

The estimator for the gradient has high variance. Variance reduction techniques:

- baseline
- batches
- causality principle (for episodic RL problems)

Part 7: Learning with function approximation and Deep RL

Why do we need function approximation?

- The best regret and sample complexity scale as $S \times A$
- Continuous action and state spaces

Video games: state = image ($S = ((255)^3)^{250000}$)

Part 7: Learning with function approximation and Deep RL

Idea: restrict our attention to learning functions belonging to a parametrized family of functions. Generally, we approximate state value functions, (state, action) value functions, or policies.

Examples: Value function and Q-function

1. Linear functions: $\mathcal{V} = \{V_\theta, \theta \in \mathbb{R}^M\}$ and $\mathcal{Q} = \{Q_\theta, \theta \in \mathbb{R}^M\}$,

   $$V_\theta(s) = \sum_{i=1}^M \phi_i(s)\theta_i = \phi(s)^\top \theta, \quad Q_\theta(s,a) = \sum_{i=1}^M \phi_i(s,a)\theta_i = \phi(s,a)^\top \theta$$

   where the ϕ_i's are linearly independent.
2. Deep networks: $\mathcal{V} = \{V_{\mathbf{w}}, \mathbf{w} \in \mathbb{R}^M\}$ and $\mathcal{Q} = \{Q_{\mathbf{w}}, \mathbf{w} \in \mathbb{R}^M\}$. $V_{\mathbf{w}}(s)$ (resp. $Q_{\mathbf{w}}(s,a)$) is given as the output of a neural network with weights $\mathbf{w}$ and inputs s (resp. (s,a)).

Part 7: Learning with function approximation and Deep RL

Policy evaluation: minimize the mean square TD error (∞ horizon)

$$J(\theta) = \frac{1}{2}\mathbb{E}_\pi[(r(s,a) + \lambda V_\theta(s') - V_\theta(s))^2]$$

TD(0) learning with function approximation is a stochastic **semi-gradient** descent algorithm:

TD(0) algorithm

1. **Initialization.** θ, initial state s_1
2. **Iterations:** For every $t \geq 1$, observe s_t, a_t, r_t, s_{t+1} under π. Update θ as:

$$\theta \leftarrow \theta + \alpha(r_t + \lambda V_\theta(s_{t+1}) - V_\theta(s_t))\nabla_\theta V_\theta(s_t)$$

Part 7: Learning with function approximation and Deep RL

On-policy control: SARSA with function approximation ((state, action) value function evaluation + policy improvement)

SARSA algorithm with function approximation

1. **Initialization.** θ, initial state s_1
2. **Iterations:** For every $t \geq 1$,
   compute π_t the ϵ-greedy policy w.r.t. Q_θ
   take action a_t according to π_t, and observe r_t, s_{t+1}
   (alternative: select the "a_{t+1}" of the previous step as a_t)
   sample a_{t+1} according to π_t
   update θ as:

$$\theta \leftarrow \theta + \alpha(r_t + \lambda Q_\theta(s_{t+1}, a_{t+1}) - Q_\theta(s_t, a_t))\nabla_\theta Q_\theta(s_t, a_t)$$

Off-policy control: Q learning with function approximation objective: a semi-gradient descent to minimize the Mean Square Bellman Error:

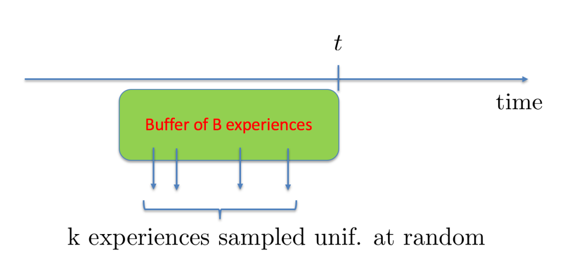
$$J(\theta) = \frac{1}{2}\mathbb{E}_{(s,a)\sim\mu_b}[(r(s,a) + \lambda \sum_j p(j|s,a) \max_b Q_\theta(j,b) - Q_\theta(s,a))^2]$$

Q-learning with function approximation

1. **Initialization.** θ, initial state s_1
2. **Iterations:** For every $t \geq 1$,
   compute π_t the ϵ-greedy policy w.r.t. Q_θ
   take action a_t according to π_t, and observe r_t, s_{t+1}
   update θ as:

$$\theta \leftarrow \theta + \alpha(r_t + \lambda \max_b Q_\theta(s_{t+1}, b) - Q_\theta(s_t, a_t))\nabla_\theta Q_\theta(s_t, a_t)$$

Experience replay: maintain a buffer B of previous experiences (s, a, r, s'). Sample mini-batches of fixed size k from B uniformly at random, and update θ accordingly.



$$\theta \leftarrow \theta + \alpha(\underbrace{r_t + \lambda \max_b Q_\theta(s_{t+1}, b)}_{\text{non-stationary target}} - Q_\theta(s_t, a_t))\nabla_\theta Q_\theta(s_t, a_t)$$

The target evolves as θ is constantly updated – it moves too fast to get tracked.

Solution: fix the target for C successive steps.
For every step:

$$\theta \leftarrow \theta + \alpha(r_t + \lambda \max_b Q_\phi(s_{t+1}, b) - Q_\theta(s_t, a_t))\nabla_\theta Q_\theta(s_t, a_t)$$

Every C steps, update the target: $\phi \leftarrow \theta$

DQN: Q-learning with ER and fixed targets

1. **Initialization.** θ and ϕ, replay buffer B, initial state s_1
2. **Iterations:** For every $t \geq 1$,
   compute π_t the ϵ-greedy policy w.r.t. Q_θ
   take action a_t according to π_t, and observe r_t, s_{t+1}
   store (s_t, a_t, r_t, s_{t+1}) in B
   sample k experiences (s_i, a_i, r_i, s'_i) from B
   for $i = 1, \ldots, k$:

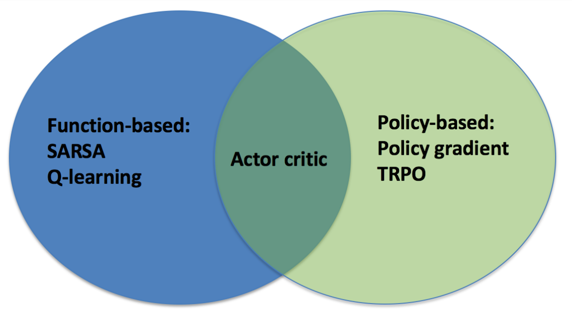
$$y_i = \begin{cases} r_i & \text{if episode stops in } s'_i \\ r_i + \lambda \max_b Q_\phi(s'_i, b) & \text{otherwise} \end{cases}$$

   update θ as:

$$\theta \leftarrow \theta + \alpha(y_i - Q_\theta(s_i, a_i))\nabla_\theta Q_\theta(s_i, a_i)$$

   every C steps: $\phi \leftarrow \theta$

Part 8: Actor-critic methods



- Function-based methods: evaluate the Q-function or the (state, action) value function of a policy to be improved
- Policy-based methods: a direct gradient on the policy
- Actor-critic methods: a policy-gradient method where function evaluation is needed

Part 8: Actor-critic methods

Policy gradient objective: maximize $J(\theta) = \mathbb{E}_{s_1 \sim p}[V^{\pi_\theta}(s_1)]$
Discounted stationary distribution ρ_θ under π_θ:

$$\forall s \in \mathcal{S}, \quad \rho_\theta(s) = (1 - \lambda) \sum_{s'} p(s') \sum_{k=0}^{\infty} \lambda^k \mathbb{P}_{\pi_\theta}[s_k = s | s_1 = s']$$

$$\nabla J(\theta) = \frac{1}{1 - \lambda} \mathbb{E}_{s \sim \rho_\theta, a \sim \pi_\theta(s, \cdot)} \left[\nabla \log \pi_\theta(s, a) Q^{\pi_\theta}(s, a) \right]$$

AC algorithm: combines policy gradient and TD algorithms

Actor-critic algorithm with baseline

AC Algorithm:

1. **Initialization:** θ, ϕ, state $s = s_1$
2. **Iterations:** Loop
   Take action $a \sim \pi_\theta(s, \cdot)$
   Observe r, s' (reward, next state)
   Sample the next action $a' \sim \pi_\theta(s', \cdot)$
   Update the parameters

$$\phi \leftarrow \phi + \beta(r + \lambda V_\phi(s') - V_\phi(s))\nabla_\phi V_\phi(s)$$
$$\theta \leftarrow \theta + \alpha\left(\nabla_\theta \log \pi_\theta(s, a)(r + \lambda V_\phi(s') - V_\phi(s))\right)$$

   $s \leftarrow s'$, $a \leftarrow a'$