Summary

EL 2805 – Reinforcement Learning

Alexandre Proutiere

KTH, The Royal Institute of Technology

Outline

1. Markov Decision Processes
2. RL problems
3. Stochastic Approximation and Stochastic Gradient Descent Algorithms
4. Policy evaluation and TD learning
5. Optimal control in RL
6. RL with function approximation
7. Policy Gradient methods
8. Actor-critic methods

1. Markov Decision Processes

1. MDPs

State space. S (finite). A state is available to the decision maker, and leads to Markovian dynamics.

Action space. For any $s \in S$, the set of available actions is A_s. $A = \cup_{s \in S} A_s$.

Dynamics. $\mathbb{P}[s_{t+1} = s' | s_t = s, a_t = a] = p_t(s'|s, a)$. Stationary dynamics if $p_t(s'|s, a) = p(s'|s, a)$.

Rewards. At time t: $r_t(s, a)$ when $(s_t, a_t) = (s, a)$ (deterministic). Stationary rewards if $r_t(s, a) = r(s, a)$.

Finite Horizon MDP

A policy $\pi = (\pi_1, \ldots, \pi_T)$ with $\pi_t : S \to A$ (or $\mathcal{P}(A)$).

Objective. Find a policy π (Markovian and deterministic) maximizing the expected reward accumulated in T rounds given the initial state s_1:
$\mathbb{E}[R(s_1, a_1^\pi, s_1^\pi, \ldots, s_T^\pi, a_T^\pi)] = \sum_{t=1}^T \mathbb{E}[r_t(s_t^\pi, a_t^\pi) | s_1^\pi = s_1]$.

Policy evaluation (Dynamic Programming). The state value function of π is
$V^\pi(s) = \sum_{t=1}^T \mathbb{E}[r_t(s_t^\pi, a_t^\pi) | s_1^\pi = s]$.
Define the rewards to go: $u_t^\pi(s) = \mathbb{E}\left[\sum_{u=t}^T r_u(s_u^\pi, a_u^\pi) \middle| s_t^\pi = s\right]$

- Start with: $u_T^\pi(s_T) = r_T(s_T, \pi(s_T))$ for all s_T
- Backward recursion to compute u_{t-1}^π from u_t^π

$$u_{t-1}^\pi(s_{t-1}) = r_{t-1}(s_{t-1}, a) + \sum_{j \in S} p_{t-1}(j | s_{t-1}, a) u_t^\pi(j)$$

We obtain: $V^\pi(s) = u_1^\pi(s)$ for any s.

Finite Horizon MDP

Optimal control. The value function $V^\star$ is the state value function of the best policy. It satisfies Bellamn's equations:

- For all s_T, $u_T^\star(s_T) = \max_a r_T(s_T, a)$
- For all $t = T-1, T-2, \ldots, 1$

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

Value function: $V_T^\star(s) = u_1^\star(s)$, $\forall s \in S$.
An optimal policy π is obtained by selecting $\pi_t(s_t)$ at time t such that

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

Infinite Horizon Discounted MDP

Stationary dynamics and rewards. Stationary policy $\pi : S \to A$.

Objective. Find a policy π (Markovian and deterministic) maximizing the expected discounted reward given the initial state s_1: $\sum_{t=1}^{\infty} \mathbb{E}[\lambda^{t-1} r(s_t^\pi, a_t^\pi) | s_1^\pi = s_1]$.

Policy evaluation. the state value function V^π is the average reward under π given the initial state, $V^\pi(s)$. It satisfies:

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

(state, action)-value function $Q^\pi(s, a)$: the average reward under π given that the first action is a,

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

Infinite Horizon Discounted MDP

Optimal control. 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:

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

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.

Value Iteration

Algorithm: Value Iteration Input. Precision ϵ, discount factor λ

1. **Initialization.** Select a value function $V_0 \in \mathbb{R}^S$, $n = 0$, $\delta \gg 1$
2. **Value improvement.** While $(\delta > \frac{\epsilon(1-\lambda)}{\lambda})$ do
   (a) $V_{n+1} = \mathcal{L}(V_n)$, i.e., $\forall s \in S$, $V_{n+1}(s) = \max_{a \in A_s}(r(s,a) + \lambda \sum_j p(j|s,a)V_n(j))$
   (b) $\delta = \|V_{n+1} - V_n\|$, $n \leftarrow n+1$
3. **Output.** Policy π with $\forall s \in S$, $\pi(s) \in \arg\max_{a \in A_s}(r(s,a) + \lambda \sum_j p(j|s,a)V_n(j))$

The above VI algorithm returns an ϵ-optimal policy π, i.e., for any $s \in \mathcal{S}$, $V^\pi(s) \geq V^\star(s) - \epsilon$.

Policy Iteration

Algorithm: Policy Iteration

1. **Initialization.** Select a policy π_0 arbitrarily, $n = 0$
2. **Policy evaluation.** Evaluate the state value function V_n of π_n by solving:

$$\forall s \in S,\ V_n(s) = r(s, \pi_n(s)) + \lambda \sum_j p(j|s, \pi_n(s)) V_n(j)$$

3. **Policy improvement.** Update the policy:

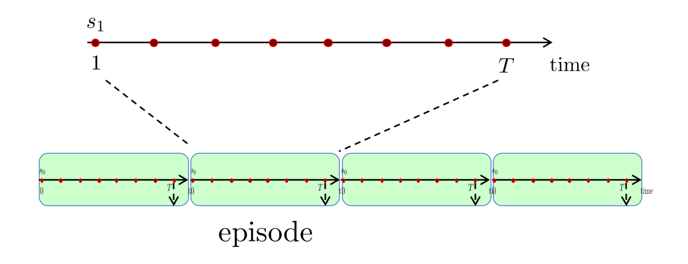
$$\forall s \in S,\ \pi_{n+1}(s) \in \arg\max_{a \in A_s} (r(s,a) + \lambda \sum_j p(j|s,a) V_n(j))$$

4. **Stopping criterion.** If $\pi_{n+1} = \pi_n$, return π_n.
   Otherwise $n \leftarrow n + 1$, and go to 2.

The **policy improvement theorem** states that under the PI algorithm, V_n is an increasing sequence, in the sense that for any $s \in \mathcal{S}$, $V_{n+1}(s) \geq V_n(s)$. In other words π_{n+1} is better than π_n. When the PI stops, it returns an optimal policy.

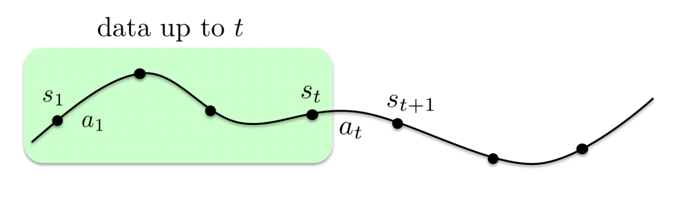
2. RL problems

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 $\pi^\star$

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.
Example: Q-learning.

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.
Example: SARSA.

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.

In RL, we use randomized policies.

ϵ-**soft policies.** Select actions uniformly at random with probability $\epsilon > 0$.

ϵ-**greedy policy w.r.t.** R**.** Let $R : S \times A \to \mathbb{R}$ a (state, action) value function. π is ϵ-greedy policy w.r.t. R iff in state s, it selects $\arg\max_a R(s, a)$ w.p. $1 - \epsilon$, and actions uniformly at random with probability ϵ.

3. Stochastic Approximation and Stochastic Gradient Descent Algorithms

Stochastic Approximation algorithm

Let $h : \mathbb{R}^d \to \mathbb{R}^d$ be a lipschitz continuous function. Root of h, i.e., $x^\star$ such that $h(x^\star) = 0$. To this aim, we have access to noisy estimates of h, i.e., for a given x, we can get from an Oracle a random variable $Y(x)$ with expectation $h(x)$.

Robbins-Monro Stochastic Approximation algorithm:

1. **Initialization:** $x^{(0)} \in \mathbb{R}^d$
2. **Iterations:** for $k \geq 0$, $x^{(k+1)} = x^{(k)} + \alpha_k [Y(x^{(k)})]$

Stochastic Approximation algorithm

Noise process: $M_{k+1} = Y(x^{(k)}) - h(x^{(k)})$.

A1. (Martingale difference) $\forall k$, $\mathbb{E}[M_{k+1}|x^{(k)}, \dots, x^{(1)}] = 0$ and $\forall k$, $\mathbb{E}[\|M_{k+1}\|_2^2 \mid x^{(k)}, \dots, x^{(1)}] \leq c_0(1 + \|x^{(k)}\|^2)$.

A2. (Stability) $\dot{x} = h(x)$ has a unique globally stable equilibrium $x^\star$. $\forall x$, $h_\infty(x) = \lim_{c \to \infty} \frac{h(cx)}{c}$ exists and 0 is the only globally stable point of $\dot{x} = h_\infty(x)$.

Convergence for decreasing learning rates. Under A1 and A2, if the learning rates α_k satisfy $\sum_{k=0}^{\infty} \alpha_k = \infty$ and $\sum_{k=0}^{\infty} \alpha_k^2 < \infty$, then $\lim_{k \to \infty} x^{(k)} = x^\star$ almost surely where $x^\star$ is the unique globally stable point of $\dot{x} = h(x)$.

Convergence for fixed learning rates. When $\alpha_k \alpha$ for all k, the algorithm does not converge, but is guaranteed to be in a neighborhood of $x^\star$ at the limit. The neighborhood is of size proportional to α.

SGD algorithm

Let $f : \mathcal{C} \to \mathbb{R}$ be a convex function defined over the convex set $\mathcal{C}$. We wish to find the minimizer of f. To this aim, we can not evaluate the gradients of f, but get only unbiased estimates of these gradients. Specifically, for any $x \in \mathcal{C}$, an Oracle can reveal $g(x)$, a r.v. such that $\nabla f(x) = \mathbb{E}[g(x)]$.

Kiefer-Wolfowitz SGD Algorithm:

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

SGD can enjoy the same convergence guarantees as those of the SA algorithm, if

$$\mathbb{E}[g(x^{(k)})|x^{(k)}, \dots, x^{(1)}] = 0.$$

4. Policy evaluation and TD learning

Finite RL problems

To evaluate a policy π, we can use Monte Carlo methods (just simulate). First visit MC algorithm:

Monte Carlo prediction algorithm:

1. **Initialization:** $\forall s$, $V^{(0)}(s) = 0$
2. **Iterations:** for episode $i = 1, \dots, n$
   generate $\tau_i = (s_1, a_{1,i}, r_{1,i}, \dots, s_{T,i}, a_{T,i}, r_{T,i})$ under π
   $G = 0$
   for $t = T, T-1, \dots, 1$:
   a. $G = r_{t,i} + G$
   b. Unless $s_{t,i}$ appears in $\{s_{1,i}, \dots, s_{t-1,i}\}$
      $V^{(i)}(s_{t,i}) = V^{(i-1)}(s_{t,i}) + \frac{1}{i}(G - V^{\pi}(s_{t,i}))$

Discounted RL problems: TD learning

Observe that the state value function of π satisfies $h(V^\pi) = 0$ where

$$\forall s, \quad h(V)(s) = r(s, \pi(s)) + \lambda \sum_j p(j|s, \pi(s)) V(j) - V(s).$$

TD(0) algorithm

1. **Initialization.**
   Select a value function $V^{(1)}$
   Initial state s_1
   Number of visits: $\forall s$, $n_s^{(1)} = 1_{(s=s_1)}$
2. **Value function updates.** For all $t \geq 1$, select action $\pi(s_t)$ and observe the new state s_{t+1} and reward r_t.
   Update the value function estimate: for all s,

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

   Update for all s, $n_s^{(t+1)} = n_s^{(t)} + 1_{(s=s_t)}$

Discounted RL problems: TD learning

Remarks.

1. TD learning is an *asynchronous* Stochastic Approximation algorithm.
2. The learning rates α_n are typically chosen as for the SA algorithm.
3. In TD methods, the term $r_t + \lambda V^{(t)}(s_{t+1})$ is often referred to as the **target**.
4. Note that TD learning is a **bootstrapping** method because the targeted value in each iteration depends on the current estimate of V^π.

5. Optimal control in RL

Q-learning

Off-policy learning of the optimal policy; behavior policy π_b. The Q-function solves:

$$\forall (s,a), \quad Q(s,a) = r(s,a) + \lambda \sum_j p(j|s,a) \max_{b \in \mathcal{A}_b} Q(j,b).$$

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: $\forall s, a,$

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

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

Q-learning

This is an asynchronous Stochastic Approximation algorithm because:

$$\mathbb{E}[r_t + \lambda \max_{b \in \mathcal{A}} Q^{(t)}(s_{t+1}, b) | s_t] = r(s_t, a_t) + \lambda \sum_j p(j|s_t, a_t) \max_{b \in \mathcal{A}_b} Q(j, b)$$

Convergence.

1. if the learning rates are appropriate;
2. the behavior policy explores enough: each (state, action) pairs should be visited infinitely often; example: under any ϵ-soft policies, e.g., π_b can be ϵ-greedy w.r.t. $Q^{(t)}$.

SARSA

SARSA is an on-policy learning algorithm. The algorithm maintains in step t both a policy π_t and its estimated (state, action) value function $Q^{(t)}$. In each iteration, SARSA

- updates π_t (the policy improvement step by taking the ϵ-greedy policy w.r.t. $Q^{(t)}$;
- updates $Q^{(t)}$ by applying a TD-learning step to evaluate the (state, action) value function of π_t.

The policy improvement step actually improves the policy (in average) according to the **policy improvement theorem**.

SARSA means (State, Action, Reward, State, Action), because the algorithm uses experiences of the type (s, a, r, s', a') generated under the policy π_t.

SARSA algorithm

Parameters. Step sizes (α_t), Exploration rate $\epsilon > 0$

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: $\forall s, a$,

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

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

Under SARSA, π_t does not converge towards an optimal policy, because π_t is ϵ-soft. When ϵ is very small, π_t approximates an optimal policy when t is large.

6. RL with function approximation

Function approximation

Beyond tabular MDPs. With large state and action spaces, Q-learning and SARSA converge very slowly (need to visit each (state, action) pairs a large number fo times).

Function approximation. Functions of interest such as the value function, the state value function of a policy, the Q-function belong to a set of parametrized functions.
For example, $V^\star \approx V_\theta \in \mathcal{V} = \{V_\mu : \mu \in \mathbb{R}^d\}$. One just needs to learn the parameter θ.

Examples.

1. Linear function approximation. In this case, we work with a basis of functions $\phi_1, \dots, \phi_d : \mathbb{R}^S \to \mathbb{R}$, and $V_\theta(s) = \sum_{i=1}^d \theta_i \phi_i(s)$.
2. Deep learning. Here the value of the function is the output of a neural network. The function is hence parametrized by the weights of the network: $V_w(s)$.

Policy evaluation with function approximation

Stationary policy π for an infinite horizon discounted MDP. The goal is then to identify the parameter θ such that V^π and V_θ are as close as possible. We minimize over θ the mean TD square error:

$$J(\theta) = \frac{1}{2}\mathbb{E}_{s\sim\mu}[(r(s,\pi(s)) + \lambda \sum_j p(j|s,\pi(s))V_\theta(j) - V_\theta(s))^2].$$

Semi-gradient method. The target is fixed and the gradient is taken w.r.t. $V_\theta(s)$ only.

TD(0) algorithm with function approximation

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

SARSA with function approximation

The previous TD algorithms with function approximation can be applied to provide an approximation of the (state, action) value function of a given policy π. This can be implemented in the policy evaluation step of SARSA algorithm.

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 **(policy improvement)**
   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: **(policy evaluation)**

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

Q-learning with function approximation

Function approximation in Q-learning: minimize the mean square Bellman error.

Bellman error. If $\tilde{Q}$ is an estimated Q-function, the corresponding Bellman error is defined as:

$$BE(s,a) = r(s,a) + \lambda \sum_j p(j|s,a) \max_b \tilde{Q}(j,b) - \tilde{Q}(s,a).$$

Objective function.

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

where μ_b is the stationary distribution of (s,a) under the behavior policy π_b.

Semi-gradient method. The semi-gradient is

$$-\mathbb{E}_{(s,a)\sim\mu_b}\Big[\underbrace{(r(s,a) + \lambda \sum_j p(j|s,a) \max_b Q_\theta(j,b)}_{\text{target}} - Q_\theta(s,a))\nabla_\theta Q_\theta(s,a)\Big]$$

Observing an experience (s, a, r, s') generated under π_b. The semi-gradient can be estimated (without bias) by:

$$-(r + \lambda \max_b Q_\theta(s', b) - Q_\theta(s, a))\nabla_\theta Q_\theta(s, a))$$

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 and Fixing the Target

Issues of Q-learning with function approximation. (a) The successive updates are strongly correlated; (b) the target is not fixed, and the algorithm struggles to follow this moving target.

Experience replay. We maintain a buffer B of previous experiences (s, a, r, s'). At time t, we store the current experience in the buffer, but to update the Q-function parameter, we sample mini-batches of fixed size k from B uniformly at random. At time t Sample, we perform k updates: for $i = 1, \dots, k$, the experience (s_i, a_i, r_i, s'_i) is sampled uniformly at random from B, and we do:

$$\theta \leftarrow \theta + \alpha(r_i + \lambda \max_b Q_\theta(s'_i, b) - Q_\theta(s_i, a_i))\nabla_\theta Q_\theta(s_i, a_i)$$

Fixed target. We use a second parameter ϕ for the target. The target is fixed for C successive steps, and then aligned to θ.

Q-learning with function approximation, 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 $i = 1, \dots, k$:

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

   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: $\phi \leftarrow \theta$

7. Policy Gradient methods

Policy Gradient methods

Main principles.

1. Parametrize your (randomized) policy: $\pi \in \Pi = \{\pi_\theta : \theta \in \mathbb{R}^d\}$;
2. Formulate an objective to be maximized: $J(\theta) = \mathbb{E}_{s_1 \sim p}[V^{\pi_\theta}(s_1)]$;
3. Compute the gradient of $J(\theta)$ (the policy gradient theorem);
4. Implement a Stochastic Gradient Ascent algorithm.

Episodic RL problems

1. and 2. Fixed time horizon T. Initial state distribution $s_1 \sim p$.
Objective function:

$$J(\theta) = \mathbb{E}_{s_1 \sim p_1}[V^{\pi_\theta}(s_1)].$$

3. Policy gradient theorem. Rewrite $V^{\pi_\theta}(s_1)$ as:

$$V^{\pi_\theta}(s_1) = \mathbb{E}_{\pi_\theta}[\sum_{t=1} r(s_t, a_t)] = \sum_{\tau} \pi_\theta(\tau) R(\tau),$$

where τ denotes the random trajectory of an episode, $\pi_\theta(\tau)$ is the probability to observe this trajectory under π_θ, and $R(\tau)$ is the total reward collected during the episode τ. we have:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(\tau) R(\tau)].$$

In the above expectations, $\mathbb{E}_{\pi_\theta}$ just indicates that the episode is generated under π_θ.

$\nabla \log \pi_\theta(\tau) = \sum_{t=1}^{T} \nabla \log \pi_\theta(s_t, a_t)$ is referred to as the **score function**.

Episodic RL problems

4. The policy gradient theorem states that when generating τ under π_θ, then $\nabla_\theta \log \pi_\theta(\tau) R(\tau)$ constitutes an unbiased estimator of $\nabla_\theta J(\theta)$. This naturally leads to REINFORCE algorithm:

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

Episodic RL problems

REINFORCE is a SGD algorithm, and finds a local maxmizer of $J(\theta)$ (under the usual convergence conditions). In practice, the algorithm offers poor performance, because of the high variance of the gradient estimator $\nabla_\theta \log \pi_\theta(\tau) R(\tau)$. Three techniques can be used to reduce this variance:

- **Batches.** Generate n episodes (instead of 1) before updating θ; this divides the variance by $1/n$.
- **Reward to go.** Instead of $\nabla_\theta \log \pi_\theta(\tau) R(\tau)$, use $\sum_{t=1}^T \nabla \log \pi_\theta(s_t, a_t) \sum_{u=t}^T r(s_u, a_u)$; this estimator remains Unbiased.
- **Baseline.** Adding a baseline helps and does not introduce any bias. When n episodes are generated under the same θ, a natural baseline is the empirical rewards observed in the n episodes:
$$b = \frac{1}{n} \sum_{i=1}^n \sum_{t=1}^T r(s_{t,i}, a_{t,i})$$
The update is computed using $\nabla_\theta \log \pi_\theta(\tau)(R(\tau) - b)$.

In practice, you may consider combining the three above techniques.

Discounted RL problems

1. and 2. Infinite time horizon discounted MDP. The objective is to maximize $J(\theta) = \mathbb{E}_{s_1 \sim p}[V^{\pi_\theta}(s_1)]$ over all possible θ.

3. Policy gradient theorem. Introduce the discounted stationary distribution ρ_θ under π_θ:

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

We have:

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

There are two difficulties in using the above formula towards a SGD algorithm.
(a) We need a critic to estimate Q^{π_θ} (see the next subsection);
(b) Sampling s according to ρ_θ is not easy.

Sampling from the discounted stationary distribution

Existing literature. Most algorithms implicitly assume that ρ_θ is the stationary distribution of the state under π_θ: they are wrong! Indeed, ρ_θ depends on the discount factor.

With restarts. When you may restart the system when you wish, you can sample according to ρ_θ as follows. Generate $s_1 \sim p_1$; for $t \geq 1$, $a_t \sim \pi_\theta(s_t, \cdot)$, s_{t+1} drawn from $p(\cdot|s_t, a_t)$ with probability λ, and from p_1 with probability $1 - \lambda$. Then sampling a state randomly from the constructed trajectory corresponds to sampling from ρ_θ.

8. Actor-critic methods

Actor-critic algorithms

A policy gradient method with policy evaluation.

Policy gradient theorems.

Stationary MDPs with terminal state: $\nabla J(\theta) = \mathbb{E}_{(s,a)\sim\mu_\theta}\left[\nabla \log \pi_\theta(s,a) Q^{\pi_\theta}(s,a)\right];$

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

Policy evaluation. The (state, action) value function of a stationary policy π satisfies:

$$\forall (s,a), Q^{\pi_\theta}(s,a) = r(s,a) + \sum_j p(j|s,a) \sum_b \pi_\theta(j,b) Q^{\pi_\theta}(j,b).$$

To evaluate Q^{π_θ}, we can use TD learning and function approximation $Q^{\pi_\theta} \approx Q_\phi$: when the experience (s,a,r,s',a') is observed, we update ϕ following a semi-gradient descent algorithm:

$$\phi \leftarrow \phi + \beta(r + Q_\phi(s',a') - Q_\phi(s,a)) \nabla_\phi Q_\phi(s,a).$$

Q Actor-critic algorithms

QAC Algorithm:

1. **Initialization:** θ, ϕ, state $s \leftarrow s_1 \sim p_1$
2. **Iterations:** Loop
   If $s = \emptyset$, $s \leftarrow s_1 \sim p_1$
   Take action $a \sim \pi_\theta(s, \cdot)$ and observe r, s' (reward, next state)
   Sample the next action $a' \sim \pi_\theta(s', \cdot)$
   Update the parameters

$$\begin{aligned} \phi &\leftarrow \phi + \beta(r + Q_\phi(s', a') - Q_\phi(s, a))\nabla_\phi Q_\phi(s, a) \\ \theta &\leftarrow \theta + \alpha\left(\nabla_\theta \log \pi_\theta(s, a) Q_\phi(s, a)\right) \end{aligned}$$

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

Note that the learning rates α and β at which θ and ϕ are updated may differ. Typically, we wish to keep the same policy π_θ for a period long enough so as to be able to estimate Q^{π_θ}. Hence, typically, α is chosen much smaller than β.

Actor-Critic algorithm with a baseline

Use a baseline to enhance the convergence properties of the algorithm.
The natural baseline is $V^{\pi_\theta}(s)$ since $V^{\pi_\theta}(s) = \mathbb{E}_{a\sim\pi_\theta(s,\cdot)}[Q^{\pi_\theta}(s,a)]$.

$$\text{Advantage function: } A^{\pi_\theta}(s,a) = Q^{\pi_\theta}(s,a) - V^{\pi_\theta}(s).$$

The policy gradient theorem states that:

$$\nabla_\theta J(\theta) = \mathbb{E}_{(s,a)\sim\mu_\theta}\left[\nabla \log \pi_\theta(s,a) A^{\pi_\theta}(s,a)\right].$$

Now when (s,a,r,s',a') is observed under π_θ, we get:

$$A^{\pi_\theta}(s,a) = r + \mathbb{E}[V^{\pi_\theta}(s')] - V^{\pi_\theta}(s)$$

Hence we can use and fit $V^{\pi_\theta} \approx V_\phi$ only! Using TD learning we get the following update:

$$\phi \leftarrow \phi + \beta(r + V_\phi(s') - V_\phi(s))\nabla_\phi V_\phi(s).$$

Actor-Critic algorithm with a baseline

The following two versions of the A2C (Advantage Actor-Critic) algorithms implement these ideas.

A2C Algorithm (TD version)

1. **Initialization:** θ, ϕ, state $s \leftarrow s_1 \sim p_1$
2. **Iterations:** Loop
   If $s = \emptyset$, $s \leftarrow s_1 \sim p_1$
   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

$$\begin{aligned}\phi \leftarrow& \phi + \beta(r + V_\phi(s') - V_\phi(s))\nabla_\phi V_\phi(s) \\ \theta \leftarrow& \theta + \alpha\left(\nabla_\theta \log \pi_\theta(s, a)(r + V_\phi(s') - V_\phi(s))\right)\end{aligned}$$

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

An interesting blog

Lilian Weng (Open AI)

RL overview: https://lilianweng.github.io/posts/2018-02-19-rl-overview/

Policy gradients: https://lilianweng.github.io/posts/2018-04-08-policy-gradient/