Reinforcement Learning

teaching material

Lorenzo Fiaschi

a.y. 2024-2025

Course: Symbolic and Evolutionary Artificial Intelligence
University of Pisa
lorenzo.fiaschi@ing.unipi.it

RL in one image

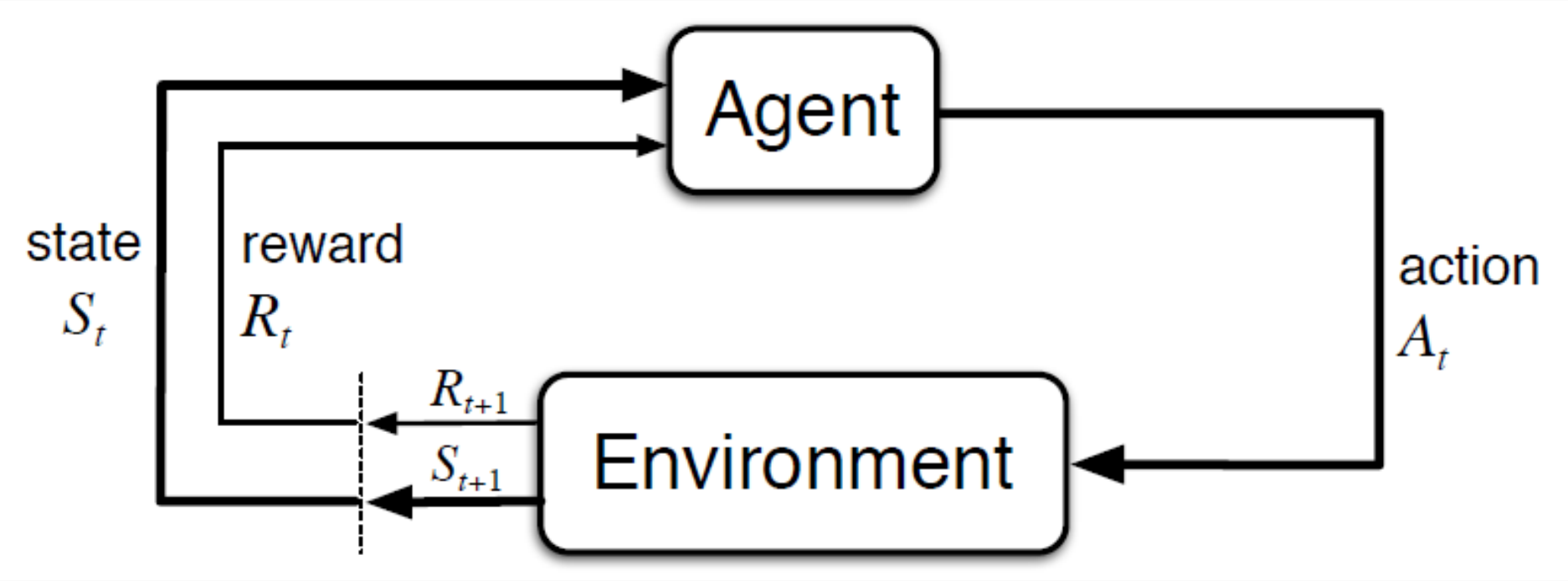
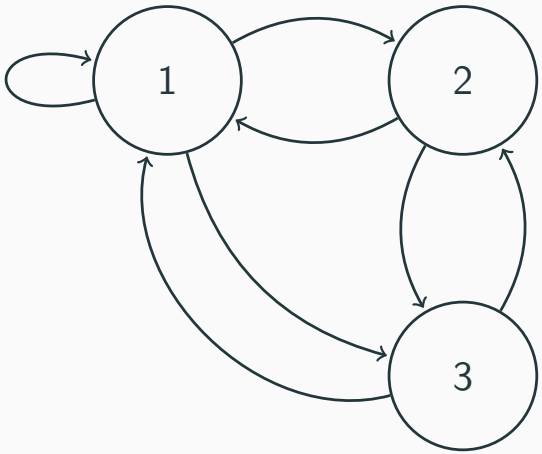


Figure 1: Idealization of an RL problem

Mathematical preliminaries

Markov Process (1/2)

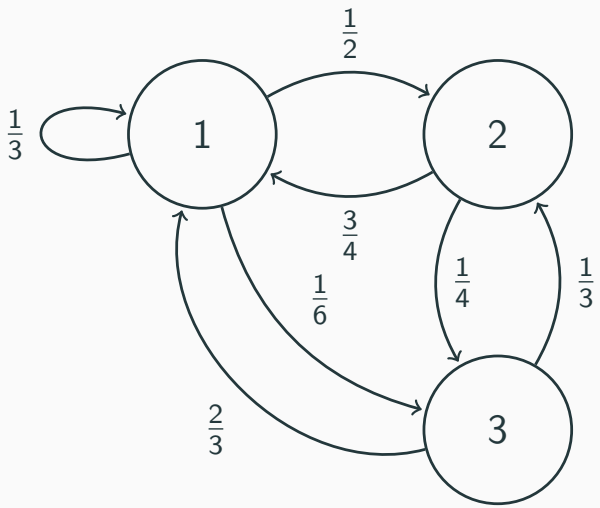


- State space S (discrete for now)
- Random walk among states
- $\mathbb{P}(s_t = \bar{s} \,|\, \bar{s}_{t-1}, \ldots, \bar{s}_1)$
- Markovian property:

  $$\mathbb{P}(s_t = \bar{s} \,|\, \bar{s}_{t-1}, \ldots, \bar{s}_1) = \mathbb{P}(s_t = \bar{s} \,|\, \bar{s}_{t-1})$$

- Transition kernel: $\mathbb{P}(s_t = \bar{s} \,|\, \bar{s}_{t-1})$
- $S = \{1, 2, 3\}$

Markov Process (2/2)



$$P = \begin{bmatrix} \frac{1}{3} & \frac{1}{2} & \frac{1}{6} \\ \frac{3}{4} & 0 & \frac{1}{4} \\ \frac{2}{3} & \frac{1}{3} & 0 \end{bmatrix}$$

$$s^* = \left[\frac{66}{127}, \frac{40}{127}, \frac{21}{127}\right]^T$$

- Transition matrix: $P \in \mathbb{R}^{|S| \times |S|}$
- $P \geq 0$, $\sum_{i=1}^{|S|} P_{ij} = 1 \; \forall j \in S$
- Starting distribution: $s^{(0)} \in \wp(S)$
- Markov process:

$$s^{(t+1)} = Ps^{(t)} = P^t s^{(0)}$$

- Stationary distribution:

$$s^* = \lim_{t \to \infty} s^{(t)} = Ps^*$$

- Eigenvector associated to $\lambda = 1$
- Ergodic theorem for existence and uniqueness

Agent, Environment and the Markovian hypothesis

- Idealization: an agent which interacts with an environment
- Environment: Markov Decision Process (MDP)
- MDP $= \langle S, A, P, R, \gamma \rangle$
- S: state space
- A: action space
- P: $S \times A \to S$ or P: $S \times A \to \wp(S)$, state transition function
- R: $S \times A \to \mathbb{R}$, reward function
- γ: discount factor
- Agent: $\langle \pi, V_\pi \rangle$ or $\langle \pi, Q_\pi \rangle$
- π: $S \to A$ or π: $S \to \wp(A)$, policy function
- V_π: $S \to \mathbb{R}$, value function
- Q_π: $S \times A \to \mathbb{R}$, action-value function

Analytical definitions

- $\pi(a|s) = \mathbb{P}(a_t = a|s_t = s)$
- $R(s,a) = R_s^a = \mathbb{E}[r_{t+1}|s_t = s,\ a_t = a]$
- $R_s^\pi = \sum_{a \in A} \pi(a|s) R_s^a = \mathbb{E}_A[R_s^a]$
- $P(s'|s,\ a) = P_{ss'}^a = \mathbb{P}(s_{t+1} = s'|s_t = s,\ a_t = a)$
- $P_{ss'}^\pi = \sum_{a \in A} \pi(a|s) P_{ss'}^a = \mathbb{E}_A[P_{ss'}^a]$
- $V_\pi(s) = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \,\middle|\, s_t = s \right]$
- $G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$, return
- $Q_\pi(s,a) = \mathbb{E}_\pi[G_t \,|\, s_t = s,\ a_t = a]$

Policy Evaluation

Policy evaluation paradigm

- **Evaluate** a policy means to infer **how worthy** it is
- V_π is **unknown**
- Idea: use π as environment **exploration** policy
- Keep **track** of the **rewards** achieved during time
- Use the rewards to **infer** V_π
- Use V_π as **measure** of π **quality**
- **Convergence** is **guaranteed** under mild hypotheses

Lemma (Bellman equation)

$G_t = r_{t+1} + \gamma G_{t+1}$

$V_\pi(s) = \mathbb{E}_\pi[r_{t+1} + \gamma V_\pi(s_{t+1}) \,|\, s_t = s]$

$Q_\pi(s, a) = \mathbb{E}_\pi[r_{t+1} + \gamma Q_\pi(s_{t+1}, a_{t+1}) \,|\, s_t = s,\ a_t = a]$

Proof.

$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} = r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} = r_{t+1} + \gamma G_{t+1}$$

The others are similar. □

Corollary (Bellman expectation)

$$V_\pi(s) = \sum_{a \in A} \pi(a|s) \left(R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a V_\pi(s') \right)$$
$$Q_\pi(s,a) = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a \sum_{a' \in A} \pi(a'|s') Q_\pi(s',a')$$

Proof.

Just a rewriting of the Lemma. □

Proposition (Closed-form solution of B.E.)

$$V_\pi = (I - \gamma P^\pi)^{-1} R^\pi$$

Proof.

From the Corollary one gets the Bellman equation matrix form:
$V_\pi = R^\pi + \gamma P^\pi V_\pi$. □

Policy Optimization

Towards the optimal agent (1/3)

- R^π is the vector whose entries are R^π_s
- P^π is the matrix whose entries are $P^\pi_{ss'}$
- The closed solution is a linear system
- For Q_π it is a tensor system (omitted for brevity)
- Impracticable to solve in real world scenarios

- Optimal agent = Optimal policy π^* (behaviour)
- π^* maximizes the expected return from any starting state

Definition (Optimal policy)

$$\pi^* := \arg\max_\pi \mathbb{E}_\pi[G_1 \mid s_1 \sim \wp(S)] = \arg\max_\pi \mathbb{E}_\pi[V_\pi(s_1) \mid s_1 \sim \wp(S)]$$

Towards the optimal agent (2/3)

Definition (Optimal value functions)

$$V^*(s) := \max_\pi V_\pi(s)$$

$$Q^*(s,a) := \max_\pi Q_\pi(s,a)$$

Theorem

For any MDP $\exists \pi^*$ which is deterministic. In addition,

$$\pi^*(a|s) = \begin{cases} 1 & \text{if } a = \arg\max_{a \in A} Q^*(s,a) \\ 0 & \text{otherwise} \end{cases}$$

- Goal: find the optimal (action-) value function
- Issue: it depends on a policy
- What about Bellman closed-form equation at optimality?

Towards the optimal agent (3/3)

Proposition (Bellman optimality equation)

$$V^*(s) = \max_{a \in A} R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a V^*(s')$$

$$Q^*(s,a) = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a \max_{a' \in A} Q^*(s',a')$$

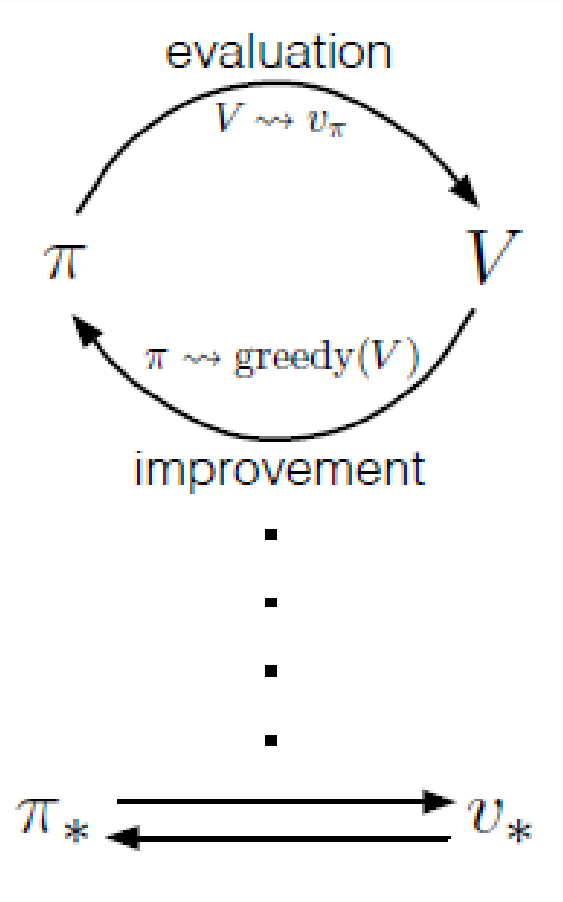
Proof.

By definition: $V^*(s) = \max_{a \in A} Q^*(s,a)$. *At the same time:* $Q^*(s,a) = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a V^*(s')$. *Substituting the definitions back and forth, one gets the thesis.* □

- These equations are not linear
- There is no closed solution (in general)
- Iterative approaches are needed

Policy optimization paradigm

- Infer the **optimal strategy** for the agent
- **Choose** an arbitrary **exploratory** policy π
- **Evaluate** it and get V_π
- **Improve** the current policy: $\pi \leftarrow \texttt{greedy}(V_\pi)$
- **Repeat** until π **converged** to π^*
- Each iteration is called GPI
- **Convergence** is **guaranteed** under mild hypotheses



Dynamic Programming

Dynamic Programming (DP)

- **Methodology** for **solving** complex **problems**
- Idea: **break** the problem into **subproblems** and solve them
- Different from "Divide-et-impera": **subproblems correlation**
- Optimal **solution** is a **subgame** perfect Nash **equilibrium**
- **Example**: Dijkstra shortest path algorithm
- **MDP** can be solved by **Dynamic Programming**
  - Bellman equation for recursive decomposition (backpropagation)
  - V_k lets one store and re-use results
- Assumption of **full environment knowledge**

Policy evaluation and Policy improvement with DP

Algorithm 1 Policy evaluation algorithm with Dynamic Programming

Input: S, A, R, P, π, ε

$V_0(s) = 0 \; \forall s \in S$

$k = 0$

while $k == 0$ **or** $||V_k - V_{k-1}|| > \varepsilon$ **do**

$$V_{k+1}(s) = \sum_{a \in A} \pi(a|s) \left(R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a V_k(s') \right)$$

$k = k + 1$

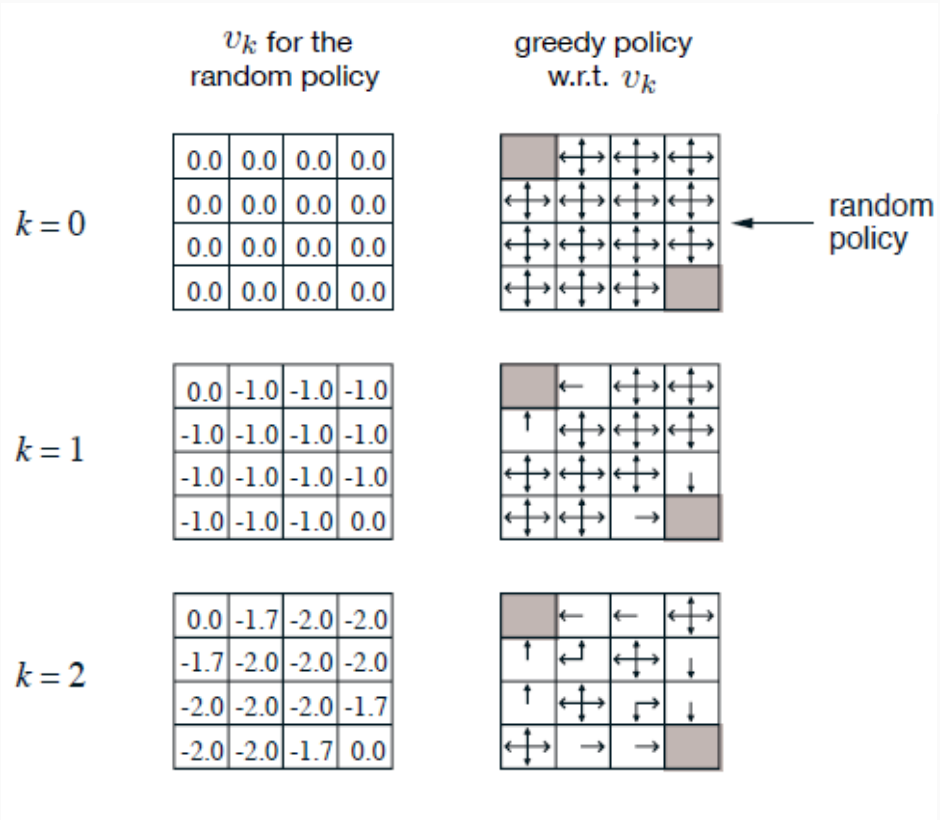
end while

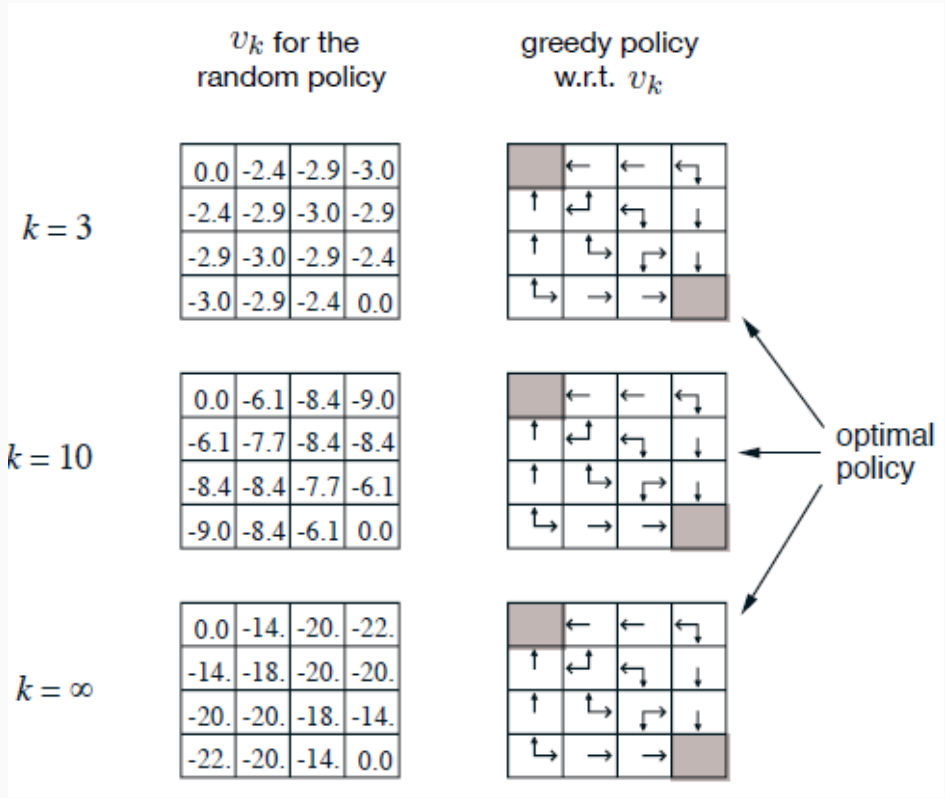
return V_k

- Policy improvement (ties are broken arbitrarily):

$$\pi(s) \leftarrow \arg\max_{a \in A} R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a V_k(s')$$

Policy evaluation at work: gridworld





- Value function takes a lot to converge
- Policy needs only 3 steps to converge
- Better updates of V_k may speed the convergence up
- Is the greedy choice always a good choice?

Policy improvement guarantee and Value iteration method

Theorem (Policy improvement)

Let $\pi' := \texttt{greedy}(V_\pi)$ *and* $\pi \neq \pi'$. *Then,* $V_{\pi'} > V_\pi$.

Proof.

By constr.: $Q_\pi(s, \pi'(s)) = \max_{a \in A} Q_\pi(s, a) \geq Q_\pi(s, \pi(s)) = V_\pi(s)$.
In case of equality, it holds $V_\pi(s) = \max_{a \in A} Q_\pi(s, a)$ *which implies* $V_\pi = V_{\pi'} = V^*$ *and therefore* $\pi = \pi' = \pi^*$. □

- How to speed V_k convergence up?
- Bellman optimality equation suggests a greedy approach
- Value iteration method

$$V_{k+1}(s) = \max_{a \in A} R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a V_k(s)$$

Partial updates and limits of DP

- Reduce update costs by **asynchronous backups**
- **In-place**: $V_{k+1}(s) = \texttt{update}(V_k(s))$
- **Prioritized**: states with highest Bellman optimality error first
- **Real-time**: only visited states are considered (need of a **well exploring** π)
- **Early stopping**: not to wait for convergence

- **DP** does **not scale** with action-state space **dimension**
- At each iteration **all** the states and actions are **considered**
- Solutions: **sample**-based approaches
  - Model-free approaches
  - Model-based approaches

Reinforcement Learning

teaching material

Lorenzo Fiaschi

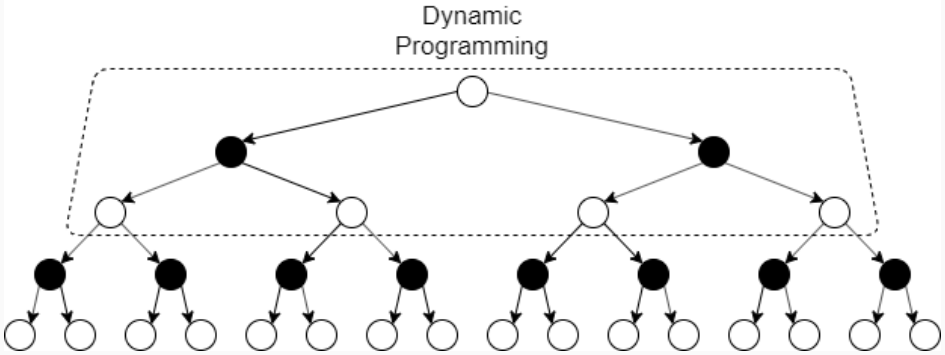
a.y. 2024-2025

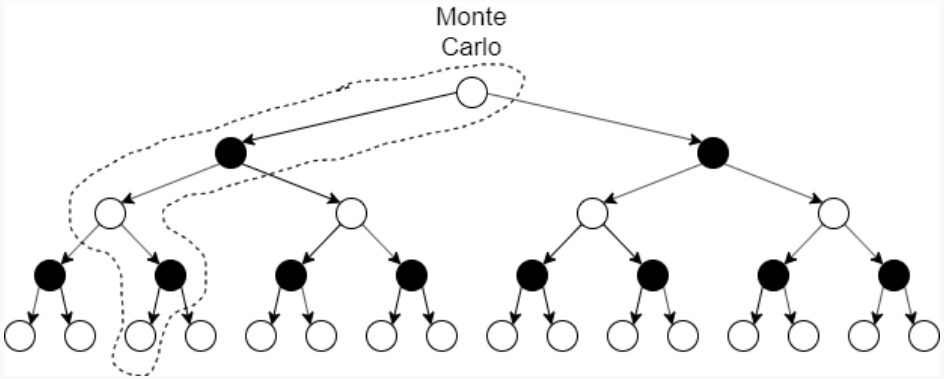
Course: Symbolic and Evolutionary Artificial Intelligence
University of Pisa
lorenzo.fiaschi@ing.unipi.it

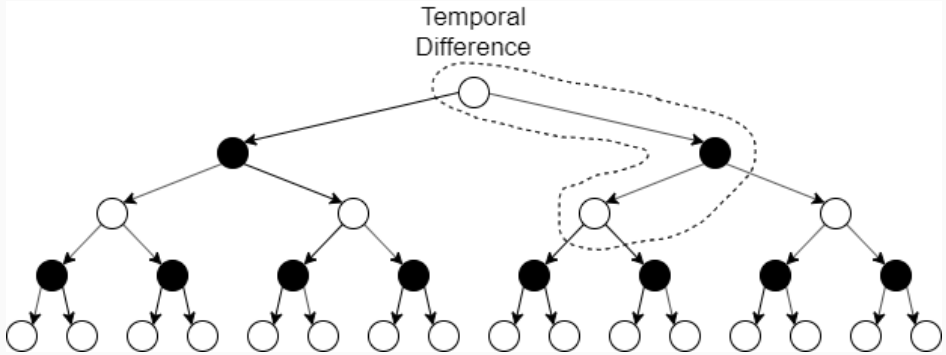
Model-free evaluation

Model-free sample-based approaches

- Monte Carlo RL
  - Learning from episodes
  - Off-line learning
  - Unbiased
  - High variance
- Temporal Difference RL
  - Learning from samples
  - Online learning
  - Biased (Bootstrapping)
  - Low variance







MC Policy evaluation (1/3)

- **Explore** the environment with π
- **Collect** $\langle S, A, R, S' \rangle$ **samples**
- An empirical **series of samples** is called **episode**
- T maximum number of actions (if not sure each episode ends)
- **Value function** computation

$$V_\pi(s) = \mathbb{E}_\pi[G_t^T \mid s_t = s], \qquad G_t^T = \sum_{k=t}^{T-1} \gamma^{k-t} r_{k+1}$$

- Two implementations
  - First-encounter
  - Every-encounter

MC Policy evaluation (2/3)

Algorithm 1 Every-encounter MC policy evaluation algorithm

Input: E

// State encountering counter

$N(s) = 0 \ \forall s \in S$

// Approximated return per state

$G(s) = 0 \ \forall s \in S$

// Approximated value function

$V(s) = 0 \ \forall s \in S$

for all $e \in E$ **do**

for $t = T - 1, \ldots, 0$ **do**

$N(s_t) = N(s_t) + 1$

$G(s_t) = G(s_t) + G_t$

end for

end for

$V(s_i) = \frac{G(s_i)}{N(s_i)}$

return V

- **Backpropagation** is used
- $\lim_{N(s)\to\infty} V(s) = V_\pi(s)$
- **Incremental mean** for algorithm optimization

$$\mu_k = \frac{1}{k}\sum_{i=1}^{k} x_i = \mu_{k-1} + \frac{1}{k}(x_k - \mu_{k-1})$$

MC Policy evaluation (3/3)

- **Using incremental mean**
  - You do not need to store $G(s)$
  - You can update $V(s)$ after each episode
  - Memory save
- **New update rule:**

$$V(s_t) = V(s_t) + \frac{1}{N(s_t)}(G_t - V(s_t))$$

- Case of **non-stationary MDP**
  - V do not converge to anything
  - **Exponential decay** is needed to forget about past

$$V(s_t) = V(s_t) + \nu(G_t - V(s_t)), \qquad \nu \in (0, 1)$$

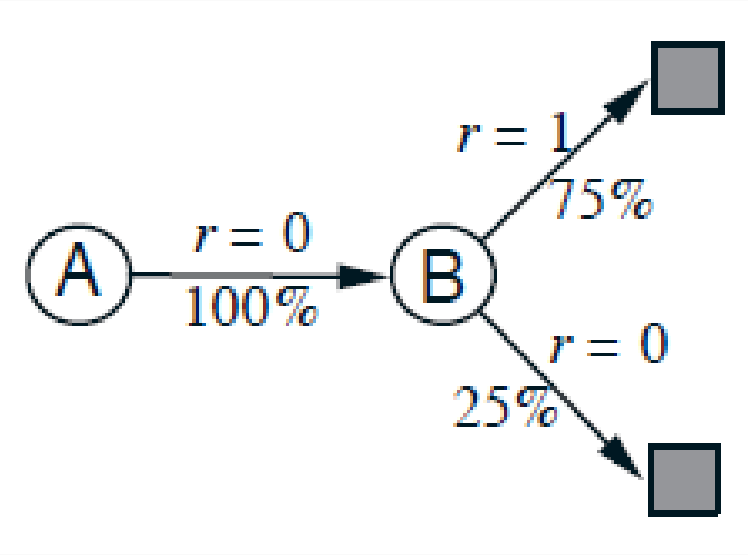
TD policy evaluation

- **Explore** the environment with π
- **Collect** and **immediately learn** from samples $\langle S, A, R, S' \rangle$
- **Incremental learning**
- **No** need to assume episodes **finite termination**
- Iterative procedure starting from $V(s) = 0 \; \forall s \in S$
- **Value function** computation

$$V(s_t) = V(s_t) + \nu(G_t - V(s_t)), \quad G_t = r_{t+1} + \gamma V(s_{t+1})$$

- Is convergence to optimality guaranteed? $0 < \nu \ll 1$
- Samples **order** affects the result a lot
- TD **converges faster** than MC in practice
- The particular choice of G_t labels the algorithm as **TD(0)**

Reasoning example



A, 0, B, 0	B, 1
B, 1	B, 1
B, 1	B, 1
B, 1	B, 0

MC approach

- $V(A) = 0$
- $V(B) = 0.75$

TD approach

- $V(A) = 0.99 * 0.75 = 0.74$
- $V(B) = 0.75$

MC vs TD in policy estimation

- **MC** converges to **minimum squared error** solution
- It tries to best **fit** the **observed returns**
- It does **not respect** the **Markovian** property

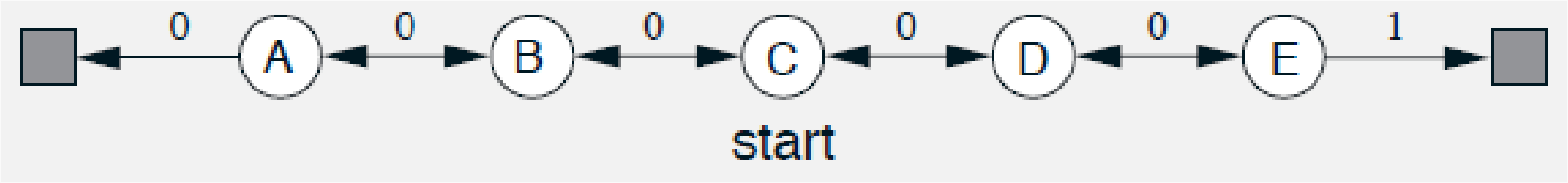
$$V_\pi = \arg\min_V \sum_{e=1}^{E} \sum_{t=0}^{T_e-1} (G_t - V(s_t))^2$$

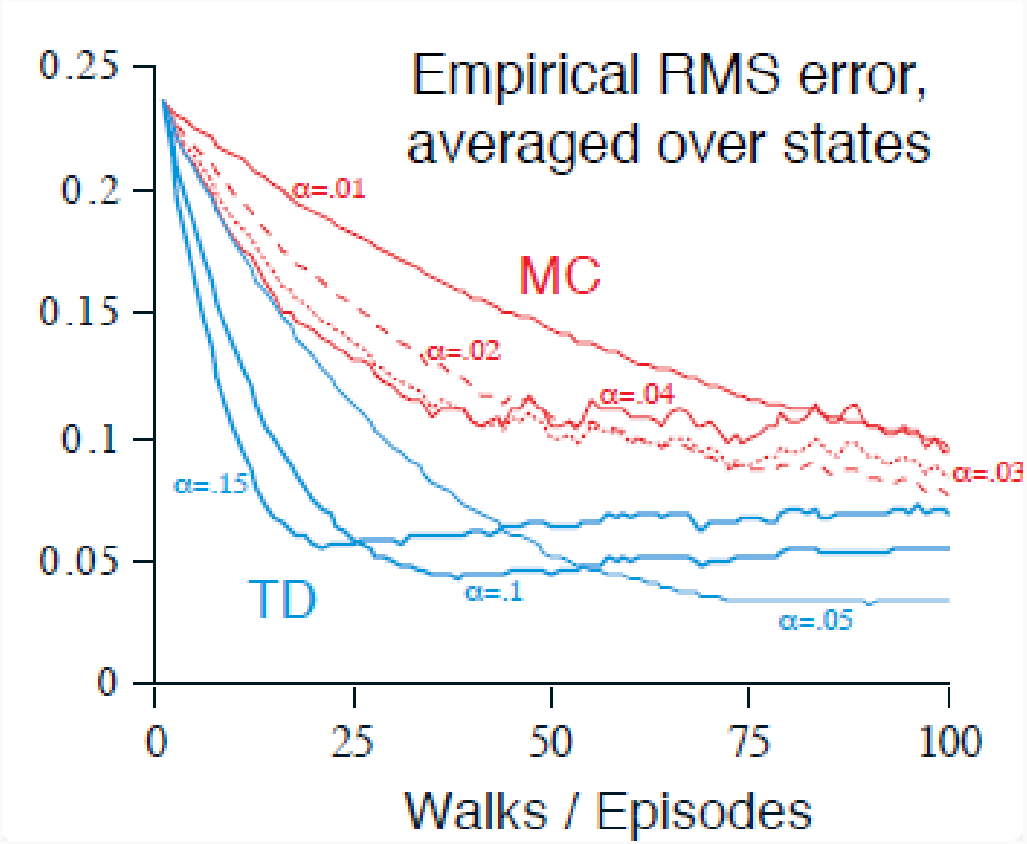
- **TD(0)** converges to **maximum likelihood** solution
- It tries to **learn transitions and rewards**
- It **respects** the **Makovian** property

$$\widehat{P}^a_{ss'} = \frac{1}{N(s,a)} \sum_{e=1}^{E} \sum_{t=0}^{T_e-1} \mathbb{1}(s_t^{(e)},\ a_t^{(e)},\ s_{t+1}^{(e)};\ s,\ a,\ s')$$

$$\widehat{R}^a_s = \frac{1}{N(s,a)} \sum_{e=1}^{E} \sum_{t=0}^{T_e-1} \mathbb{1}(s_t^{(e)},\ a_t^{(e)};\ s,\ a) r_t^{(e)}$$

Noise and convergence





Model-free on-policy optimization

Model-free on-policy optimization (1/4)

- Two philosophies: **on-** or **off-policy** learning
- On-policy: learn about π **sampling** from π's **experience**
- Off-policy: learn about π from **another** policy μ

- **First idea** for on-policy approach: **GPI**
  - $\pi_k \rightarrow V_{\pi_k}$
  - $V_{\pi_k} \rightarrow \pi_{k+1}$
  - $k \leftarrow k+1$ and repeat
- First issue: **lack environment knowledge**
  - Greedy improvement: $\pi_{k+1}(a|s) = \arg\max_{a \in A} R_s^a + P_{ss'}^a V_{\pi_k}(s')$
  - R_s^a and $P_{ss'}^a$ are not known
  - Solution: Q-learning: $\pi_{k+1}(a|s) = \arg\max_{a \in A} Q_{\pi_k}(s,a)$

Model-free on-policy optimization (2/4)

- Second issue: **no exploration**
  - π_0 is stochastic, π_1 is deterministic
  - Solution: ε-greedy policy
  - ε probability to explore, 1-ε to exploit

$$\pi_k(a|s) = \begin{cases} \frac{\varepsilon}{m} + (1 - \varepsilon) & \text{if } a = \arg\max_{a \in A} Q_\pi(s, a) \\ \frac{\varepsilon}{m} & \text{otherwise} \end{cases}$$

  - m is the number of actions at state s
- Is **convergence** still guaranteed?

Proposition (ε-greedy policy improvement)

The ε-greedy update generates a policy π_{k+1} such that $V_{\pi_{k+1}} \geq V_{\pi_k}$

Proof.

$$
\begin{aligned}
V_{\pi_k}(s) &= \sum_{a \in A} \pi_{k+1}(a|s) Q_{\pi_k}(s,a) = \\
&= \frac{\varepsilon}{m} \sum_{a \in A} Q_{\pi_k}(s,a) + (1-\varepsilon) \max_{a \in A} Q_{\pi_k}(s,a) \geq \\
&\geq \frac{\varepsilon}{m} \sum_{a \in A} Q_{\pi_k}(s,a) + (1-\varepsilon) \frac{\pi_k(a|s) - \frac{\varepsilon}{m}}{1-\varepsilon} Q_{\pi_k}(s,a) = \\
&= \sum_{a \in A} \pi_k(a|s) Q_{\pi_k}(s,a) = V_{\pi_k}(s)
\end{aligned}
$$

□

Model-free on-policy optimization (4/4)

- Bellman optimality equation demands for a **deterministic** policy
- ε-greedy improvement generates a **stochastic** one
- Is **convergence** to optimality still **guaranteed**?
- Greedy in the Limit with Infinite Exploration (**GLIE**) property

Definition (GLIE)

An ε-greedy improvement of π has the GLIE property if and only if

1. $\lim_{k\to\infty} N_k(s,a) = \infty$
   each state-action pair is explored infinitely many times
2. $\lim_{k\to\infty} \pi_k(a|s) = \mathbb{1}(a;\ \arg\max_{a'\in A} Q_k(s,a'))$
   policy converges to the greedy one
3. $\varepsilon_k \in \mathcal{O}(\frac{1}{k})$
   ε converges to zero as $\frac{1}{k}$

On-policy optimization algorithms

Model-free on-policy optimization algorithms

- All the algorithms presented have the **GLIE** property
- **MC optimization iteration**:
  - $N(s_t, a_t) \leftarrow N(s_t, a_t) + 1$
  - $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \frac{1}{N(s_t, a_t)}(G_t - Q(s_t, a_t))$
  - $\varepsilon \leftarrow \frac{1}{k}$
  - $\pi_{k+1} \leftarrow \varepsilon\text{−}\texttt{greedy}(Q)$
- **TD optimization** has multiple **variants**
  - SARSA
  - Expected SARSA
  - (Expected) SARSA(n) and SARSA(λ)
  - Backward (Expected) SARSA(λ)

TD optimization

- The idea is the same of TD evaluation
- This time is Q to be update rather than V
- **SARSA** samples: $\langle S, A, R, S', A' \rangle$

- **SARSA update rule**
  - $Q(s,a) \leftarrow Q(s,a) + \nu(r + \gamma Q(s',a') - Q(s,a))$
  - Robbins-Monro condition: $\sum_{t=1}^{\infty} \nu_t = \infty \wedge \sum_{t=1}^{\infty} \nu_t^2 < \infty$
- **Expected SARSA**
  - $Q(s,a) \leftarrow Q(s,a) + \nu \left(r + \gamma \sum_{a' \in A} \pi(a'|s') Q(s',a') - Q(s,a) \right)$
  - Fails when the codomain dimension of π is too large

Model-free off-policy optimization

Off-policy learning

- Why is it important?
  - Learn from **imitation**
  - **Re-use experience** generated by old policies
  - **Learn** the optimal policy while **following** an **exploratory** one
  - **Learn multiple** policies while following only one
  - **Avoid** real world **re-testing** (expensive and ineffective)
- Key ingredient: **Importance Sampling**
- Expectation estimation leveraging an external distribution

Definition (Importance Sampling)

$$\mathbb{E}_{x\sim P}[f(x)] = \sum_x P(x)f(x) =$$

$$= \sum_x Q(x)\frac{P(x)}{Q(x)}f(x) = \mathbb{E}_{x\sim Q}\left[\frac{P(x)}{Q(x)}f(x)\right]$$

Off-policy optimization algorithms (1/2)

- **MC optimization**
  - Two returns: G_t^μ and G_t^π
  - G_t^μ: return experienced by acting in accordance with μ
  - G_t^π: return inferred by π from μ's experience
  - $G_t^\pi = \prod_{i=t}^{T} \frac{\pi(a_i|s_i)}{\mu(a_i|s_i)} G_t^\mu$
  - Update rule: $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \nu(G_t^\pi - Q(s_t, a_t))$
- **Dramatic variance increment** by importance sampling
- π and μ must be **as close as possible**
- **TD optimization**
  - Update rule:
    $Q(s, a) \leftarrow Q(s, a) + \nu \left[\frac{\pi(a|s)}{\mu(a|s)}(r_{t+1} + \gamma Q(s', a')) - Q(s, a)\right]$
  - Only one importance sampling correction needed
  - This reduces a lot the variance

Off-policy optimization algorithms (2/2)

- **Q-learning**
  - Particular choice of π and μ
  - $\pi(s) = \arg\max_{a \in A} Q(s, a)$ (greedy policy)
  - $\mu(s) = \varepsilon{-}\texttt{greedy}(Q(s, a))$
  - No use of importance sampling
  - Update rule:

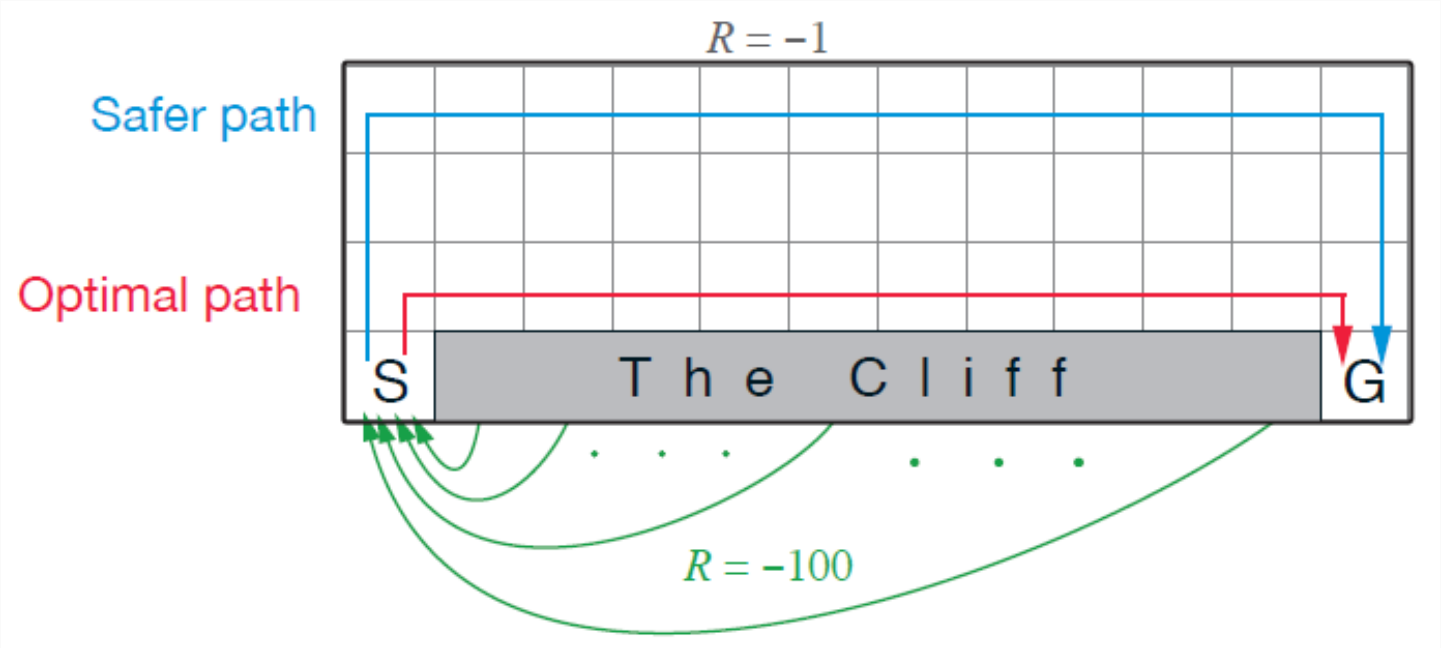
    $$Q(s, a) \leftarrow Q(s_t, a_t) + \nu(r^{\mu}_{t+1} + \gamma Q(s_{t+1}, a^{\pi}_{t+1}) - Q(s_t, a_t))$$
  - r^{μ}_{t+1} comes from μ, while a^{π}_{t+1} is chosen by π
  - Simplification of learning target

$$r^{\mu}_{t+1} + \gamma Q(s_{t+1}, a^{\pi}_{t+1}) = r^{\mu}_{t+1} + \gamma Q(s_{t+1}, \arg\max_{a' \in A} Q(s_{t+1}, a')) =$$

$$= r^{\mu}_{t+1} + \gamma \max_{a' \in A} Q(s_{t+1}, a')$$

- $Q(\lambda)$ generalization is straightforward

Expected SARSA vs. Q-learning



Beyond TD(0)

TD(n) and TD(λ)

- Faster and more precise with **more than one** look ahead
- TD(n): n **additional steps forward**

$$V(s_t) = V(s_t) + \nu(G_t^{(n)} - V(s_t)), \quad G_t^{(n)} = \sum_{k=0}^{n} \gamma^k r_{t+k+1} + \gamma^{n+1} V(s_{t+n+1})$$

- TD(λ): **average** of the $G_t^{(n)}$ (**weighted mean**)

$$V(s_t) = V(s_t) + \nu(G_t^{(\lambda)} - V(s_t)), \quad G_t^{(\lambda)} = (1-\lambda) \sum_{n=1}^{N} \lambda^{n-1} G_t^{(n)}$$

N as large as you want, $\lambda \in (0, 1)$, $\sum_{n=1}^{\infty} (1-\lambda)\lambda^{n-1} = 1$

- $G_t^{(\lambda)}$ and $G_t^{(n)}$ can be incrementally updated
- Both are **offline** approaches: they learn on **complete** episodes

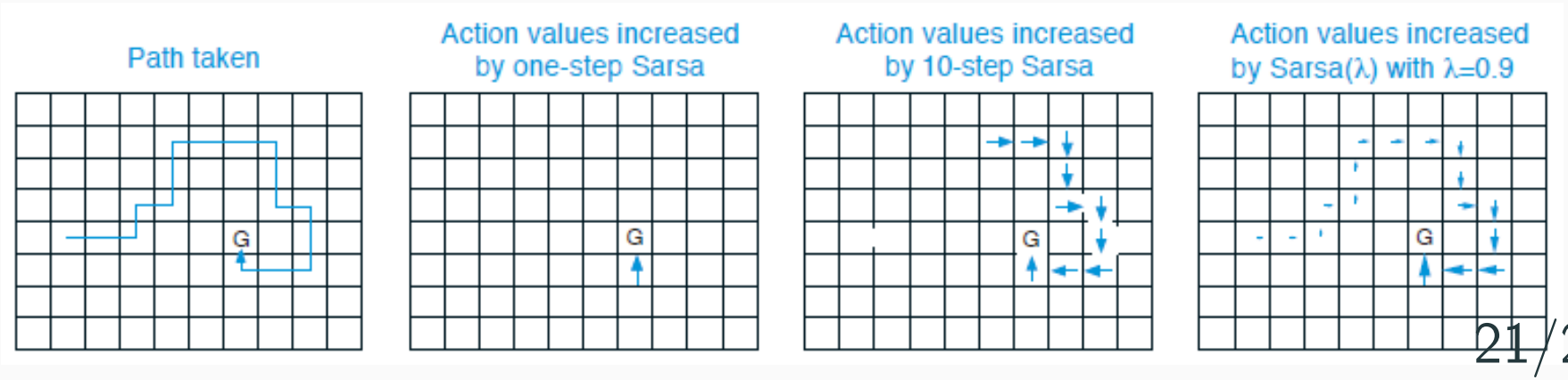
Eligibility traces and Backward TD(λ)

- Goal: implement TD(λ) from **partial episodes**
- Idea: **Eligibility function** $E\colon S \to \mathbb{R}^+$
- Problem of **assigning the cause** of something happened later
- It **measures** how far in the **future** a **state** is still **important**

$$E_0(s) = 0, \qquad E_t(s) = \gamma\lambda E_{t-1}(s) + \mathbb{1}(s_t;\, s)$$

- BTD(λ)

$$V(S) = V(S) + \nu\delta_t E_t(S), \quad \delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$$



Eligibility traces and Backward TD(λ)

$$\gamma = 1, \quad \lambda = 0.5, \quad \nu = 0.99, \quad r_{t+1} = 1$$

$$E_0(s) = 0, \qquad E_t(s) = \gamma\lambda E_{t-1}(s) + \mathbb{1}(s_t;\, s)$$

$$V(S) = V(S) + \nu\delta_t E_t(S), \quad \delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$$

$V(S)$		$E(S)$	δ	$V(S)$
$\begin{bmatrix}0.00\\0.00\\0.00\\0.00\end{bmatrix}$	$s_1 \to s_2$	$\begin{bmatrix}1.00\\0.00\\0.00\\0.00\end{bmatrix}$	1.00	$\begin{bmatrix}0.99\\0.00\\0.00\\0.00\end{bmatrix}$
$\begin{bmatrix}0.99\\0.00\\0.00\\0.00\end{bmatrix}$	$s_2 \to s_3$	$\begin{bmatrix}0.50\\1.00\\0.00\\0.00\end{bmatrix}$	1.00	$\begin{bmatrix}1.48\\0.99\\0.00\\0.00\end{bmatrix}$
$\begin{bmatrix}1.48\\0.99\\0.00\\0.00\end{bmatrix}$	$s_3 \to s_1$	$\begin{bmatrix}0.25\\0.50\\1.00\\0.00\end{bmatrix}$	2.48	$\begin{bmatrix}2.09\\1.60\\0.61\\0.00\end{bmatrix}$
$\begin{bmatrix}2.09\\1.60\\0.61\\0.00\end{bmatrix}$	$s_1 \to s_4$	$\begin{bmatrix}1.13\\0.25\\0.50\\0.00\end{bmatrix}$	-1.09	$\begin{bmatrix}0.87\\1.33\\0.09\\0.00\end{bmatrix}$

SARSA(n), SARSA(λ) & Co.

- **SARSA(n)**
  - $Q(s,a) \leftarrow Q(s,a) + \nu(r + \gamma Q_t^{(n)} - Q(s,a))$
  - $Q_t^{(n)} = \sum_{k=0}^{n} \gamma^k r_{t+k+1} + \gamma^{n+1} Q(s_{t+n+1}, a_{t+n+1})$
- **SARSA(λ)**
  - $Q(s,a) \leftarrow Q(s,a) + \nu(r + \gamma Q_t^{(\lambda)} - Q(s,a))$
  - $Q_t^{(\lambda)} = (1-\lambda) \sum_{n=1}^{N} \lambda^{n-1} Q_t^{(n)}, \quad \sum_{n=1}^{\infty} (1-\lambda)\lambda^{n-1} = 1$
- **Backward SARSA(λ)**
  - $Q(S,A) \leftarrow Q(S,A) + \nu \delta_t E_t(S,A)$
  - $\delta_t = r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)$
  - $E_t(s,a) = \gamma\lambda E_{t-1}(s,a) + \mathbb{1}(s_t, a_t;\, s, a),\ E_0(s,a) = 0$

Reinforcement Learning

teaching material

Lorenzo Fiaschi

a.y. 2024-2025

Course: Symbolic and Evolutionary Artificial Intelligence

University of Pisa
lorenzo.fiaschi@ing.unipi.it

Function approximators

Value function approximation (1/2)

- **Tabular methods** have some **limitations**
- Difficulties with **non-trivial state spaces**
  - Large spaces
  - Continuous spaces
- **Lack of generalization** properties
  - Incapability to recognize "similar" states
  - Similar states must behave similarly
- **State encoding**
- **Function approximator**
  - No more tabular approximation of V and Q
  - Parametric functions $\widehat{V}\colon S \times \Theta \to \mathbb{R}$, $\widehat{Q}\colon S \times \Theta \to \mathbb{R}^{|A|}$
  - Θ: parameter space; $\theta \in \Theta$ specific parameters choice
  - Capability to generalize from seen to unseen states

Value function approximation (2/2)

- **Type of approximators** (focus on differentiable methods)
  - Kernel/Feature maps
  - DNNs
  - Decision Trees
  - KNNs
  - Fourier/Wavelet transforms
- Issue 1: **Non-stationary data**
  - Policy modifies over time
  - The environment may change
- Issue 2: **Non-IID data**
  - Samples are necessarily correlated

Incremental learning (1/2)

- **Policy evaluation**: looking for V_π
- **Stochastic Gradient Descent**
  - Loss function MSE:
    $L(\theta) = \mathbb{E}[(V_\pi(s) - \widehat{V}(s,\theta))^2]$
- **Linear value approximation**
  - $\widehat{V}(s,\theta) = \phi(s)^T\theta$
  - $\phi(s)$: state space encoding function
  - Linear approximation is enough thanks to Representer Theorem
  - Linearity $\rightarrow$ $L(\theta)$ is quadratic
  - Gradient: $\Delta\theta = \nu(V_\pi(s) - \widehat{V}(s,\theta))\phi(s)$

Incremental learning (2/2)

- Issue: V_π is **not known**
- Solution: **replace** it **with** the **empirical target** G_t
- **MC evaluation**
  - G_t is unbiased but noisy
  - Supervised learning on the samples $(s_t,\ G_t)$
  - Strong convergence properties
- **TD(0) evaluation**
  - G_t is reliable but biased
  - Supervised learning on the samples $(s_t,\ r_{t+1} + \gamma \widehat{V}(s_{t+1}, \theta))$
  - Weak convergence properties
- **TD(λ)**
  - As in TD(0) but with G_t^λ
  - Eligibility is an overlapping of states encoding now
  - $E_t = \gamma \lambda E_{t-1} + x(s_t)$

Batched learning

- **Batched episodes** to generate more **stable gradients**
- **Gradient** from multiple samples (**training data**)
- **Non-stationarity is solved by**
  - State space encoding
  - Experience collection in datasets
- Non-IID is solved by **dataset sampling**
- **Experience replay buffer**
  - Collect agent experience in a dataset
  - Learn stochastically sampling from it
  - Transitions repetition is allowed
  - Increases stability of the process because data are not correlated
  - Random sampling, Prioritized sampling

DNN based approach

- **Deep Q-Network** (DQN)
  - DNN to approximate Q
  - Experience replay buffer
  - Fixed Q-targets to stabilize the learning
  - Max. bias: maximum estimate to estimate the maximum
  - Targets come from an old, fixed version of the network: θ^-
- **Simple DQN loss function**

$$L(\theta) = \mathbb{E}_{\mathcal{D}}[(r + \gamma \max_{a' \in A} \widehat{Q}(s', a', \theta^-) - \widehat{Q}(s, a, \theta))^2]$$

- **Double DQN loss function**

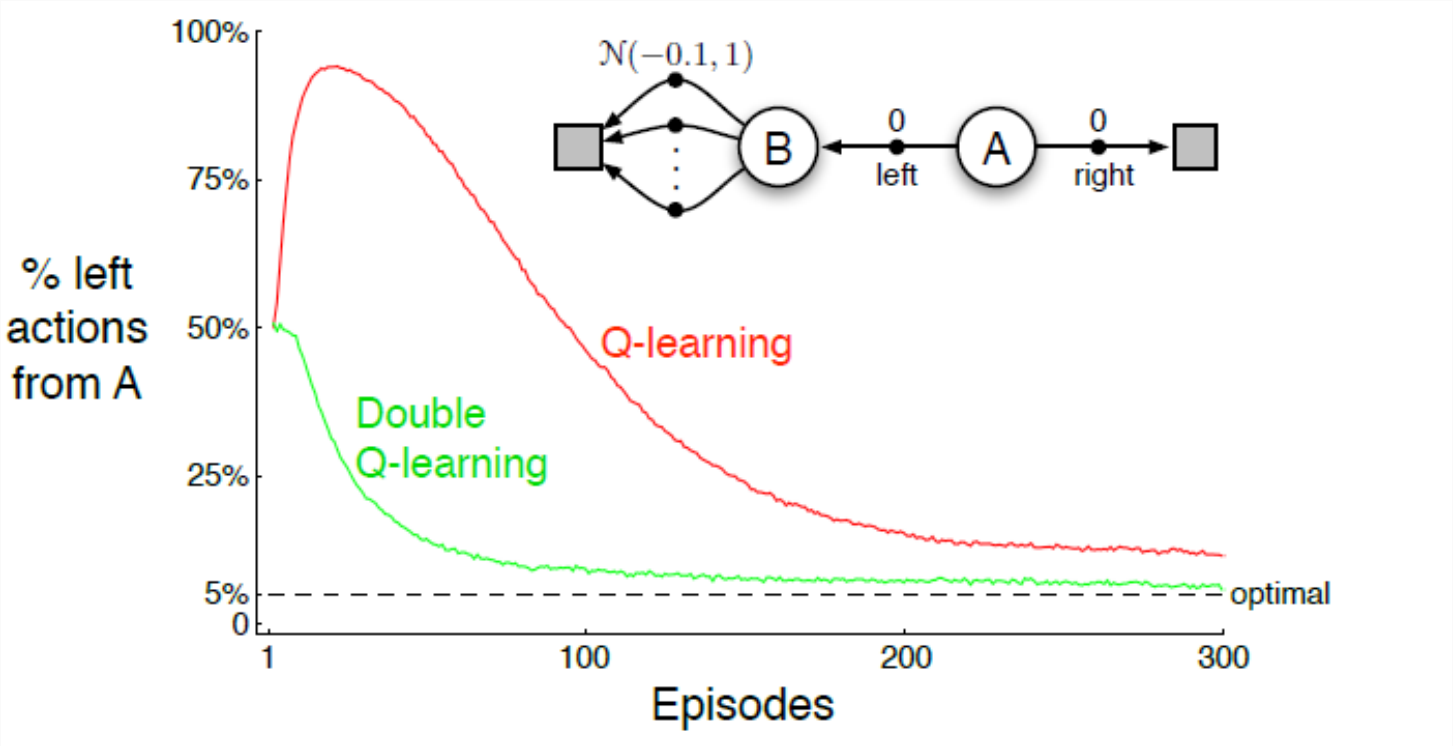
$$L(\theta) = \mathbb{E}_{\mathcal{D}}[(r + \gamma \widehat{Q}(s', \arg\max_{a' \in A} \widehat{Q}(s', a', \theta), \theta^-) - \widehat{Q}(s, a, \theta))^2]$$

- **Duelling networks**

$$\widehat{Q}(s, a, \theta) = \widehat{V}(s, \theta_V) + \left(\widehat{A}(s, a, \theta_A) - \max_{a' \in A} \widehat{A}(s, a', \theta_A)\right)$$

Effect of maximum bias

- Use of the maximum estimate as the estimate of the maximum
- A is the starting point
- Suppose to experience something positive going left from A
- The learning is biased towards B



Policy Gradient

Policy gradient optimization (1/3)

- π^* approximation **converges faster** than V^* one
- Idea: **approximate** it **directly**
- Policy approximation function: $\widehat{\pi}(a|s, \theta)$
- **Previously**, π^* approximation was **implicit**

Pros.

- Faster convergence
- Effective in non-trivial action spaces
- Stochastic policy learning is possible (Game Theory)

Cons.

- No guarantee of global optimum convergence
- Higher variance
- Inefficient evaluation
- Need of a differentiable π

Policy gradient optimization (2/3)

- How to **evaluate** a policy without V and Q?
- Estimating the **expected value** of the **starting state**
- **MC**
  - Deterministic case: $L(\theta) = \mathbb{E}_{\widehat{\pi}}[V(s_1)]$
  - Stochastic case: $L(\theta) = \mathbb{E}_{\widehat{\pi}}[V(s_1) \mid s_1 \sim \wp(S)]$
- Estimate the **average value** or **reward** among the states
- **TD**
  - Average value: $L(\theta) = \sum_{s \in S} d_{\widehat{\pi}}(s) V_{\widehat{\pi}} = \mathbb{E}_{\widehat{\pi}}[V(s) \mid s \sim d_{\widehat{\pi}}(S)]$
  - Average reward: $L(\theta) = \sum_{s \in S} d_{\widehat{\pi}}(s) \sum_{a \in A} \widehat{\pi}(a|s, \theta) R_s^a$
  - $d_{\widehat{\pi}}(S)$: stationary distribution of $\widehat{\pi}$

Policy gradient optimization (3/3)

- Differentiability
- Adaptive ν needed (ADAM)

Theorem (Policy Gradient)

Let $\widehat{\pi} \in C^1(\mathbb{R}^n)$. In any policy gradient optimization where $L(\theta) \propto \mathbb{E}_{\widehat{\pi}}[V(s)]$, it holds true that

$$\begin{aligned}\nabla_\theta L(\theta) \propto & \sum_{s\in S} d_{\widehat{\pi}}(s) \sum_{a\in A} \nabla_\theta \widehat{\pi}(a|s,\theta) Q_{\widehat{\pi}}(s,a) = \\ &= \mathbb{E}_{\widehat{\pi}}\left[\sum_{a\in A} \nabla_\theta \widehat{\pi}(a|S,\theta) Q_{\widehat{\pi}}(S,a) \mid d_{\widehat{\pi}}(S)\right] = \\ &= \mathbb{E}_{\widehat{\pi}}\left[\nabla_\theta \ln \widehat{\pi}(A|S,\theta) Q_{\widehat{\pi}}(S,A) \mid d_{\widehat{\pi}}(S,A)\right]\end{aligned}$$

- **Cost function redefinition**:

$$\nabla_\theta \widehat{\pi} = \widehat{\pi} \frac{\nabla_\theta \widehat{\pi}}{\widehat{\pi}} = \widehat{\pi} \nabla_\theta \ln \widehat{\pi}$$

MC policy gradient

- G_t is an **unbiased sample** of $Q_{\widehat{\pi}}(s_t, a_t)$
- **On-policy** approach:

$$\nabla_\theta L(\theta) \propto \underbrace{\underbrace{\frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T}\nabla_\theta \ln \widehat{\pi}(s_t^i, a_t^i, \theta)}_{\text{Maximum Likelihood Gradient}} G_t^i}_{\text{Policy Gradient}}$$

- G_t^i **encourages better** performing **trajectories**
- Likelihood maximization is a **supervised** problem **instead**
- **Off-policy**: importance sampling applies but very is noisy

Actor-Critic

Actor-Critic policy gradient (1/4)

- Goal: **reduce** MC policy gradient **variance**
- Tools: **both value** and **policy** learning
- Idea: $\widehat{Q}(s, a, \omega) \approx Q_{\widehat{\pi}}(s, a)$, already seen approach (MC, TD, ...)
- In case $\widehat{Q}(s, a, \omega)$ is DNN
  - $\omega \neq \theta$: two DNNs, **simple**, **stable**, no features sharing
  - $\omega = \theta$: one DNN, only **features sharing**, gradient conflicts
- Two entities now
  - **Actor**: $\widehat{\pi}$ generates experience and updating directions
  - **Critic**: $\widehat{Q}$ evaluates the quality of the actions taken

Theorem (Compatible function approximation)

In case of actor-critic approach, if

1. $\nabla_\omega \widehat{Q}(s, a, \omega) = \nabla_\theta \ln \widehat{\pi}(a|s, \theta)$ *(compatibility)*
2. $\mathbb{E}_{\widehat{\pi}}[(Q_{\widehat{\pi}} - \widehat{Q})^2] \xrightarrow{t\to\infty} 0$ *(unbiased property)*

then the Policy Gradient Theorem still holds true, i.e.,

$$\nabla_\theta L(\theta) \propto \mathbb{E}_{\widehat{\pi}}[\nabla_\theta \ln \widehat{\pi}(A|S, \theta) \widehat{Q}(S, A, \omega) \mid d_{\widehat{\pi}}(S, A)].$$

- Thus

$$\widehat{Q}(s, a, \omega) = \nabla_\theta \ln \widehat{\pi}(a|s, \theta)^T \omega$$

Actor-Critic policy gradient (3/4)

- **Variance** can be **reduced** further by the use of a **baseline** $B(\cdot)$
- $B(\cdot)$ represents **empirical knowledge on the model**
- **Use** of the Baseline:

$$\nabla_\theta L(\theta) \propto \mathbb{E}_{\widehat{\pi}} \left[\sum_{a \in A} \nabla_\theta \widehat{\pi} \left(Q_{\widehat{\pi}} - B(\cdot) \right) \mid d_{\widehat{\pi}}(S) \right]$$

- If B depends only on S it does **not alter** the **gradient**:

$$\mathbb{E}_{\widehat{\pi}} \left[\sum_{a \in A} \nabla_\theta \widehat{\pi} B(S) \right] = \mathbb{E}_{\widehat{\pi}} \left[B(S) \nabla_\theta \sum_{a \in A} \widehat{\pi} \right] = \mathbb{E}_{\widehat{\pi}} \left[B(S) \nabla_\theta 1 \right] = 0$$

- A **good choice** is $B(s) = V_{\widehat{\pi}}(s)$
- **Advantage function** $A_{\widehat{\pi}}(s, a) = Q_{\widehat{\pi}}(s, a) - V_{\widehat{\pi}}(s)$
- **Measures** the **value added** by action a to state s

Actor-Critic policy gradient (4/4)

- **Advantage Policy Gradient** theorem reformulation:

$$\nabla_\theta L(\theta) \propto \mathbb{E}_{\widehat{\pi}}[\nabla_\theta \ln \widehat{\pi}(A|S,\theta) A_{\widehat{\pi}}(S,A,\omega) \mid d_{\widehat{\pi}}(S,A)]$$

- In **Actor-Critic scenario** $A_{\widehat{\pi}}$ complicates a lot
- **Three sets of parameters** are needed

$$\widehat{A}(s,a,\omega,\rho) = \widehat{Q}(s,a,\omega) - \widehat{V}(s,\rho)$$

- However, **only** ρ is **truly needed**

$$Q_{\widehat{\pi}}(s,a) = \mathbb{E}_{s' \in S}[r + \gamma V_{\widehat{\pi}}(s') \mid s,a]$$

$$A_{\widehat{\pi}}(s,a) = \mathbb{E}_{s' \in S}[r + \gamma V_{\widehat{\pi}}(s') - V_{\widehat{\pi}}(s) \mid s,a]$$

- The degree of bootstrapping depends on the time-scale

Natural Policy Gradient

Natural Policy Gradient (1/3)

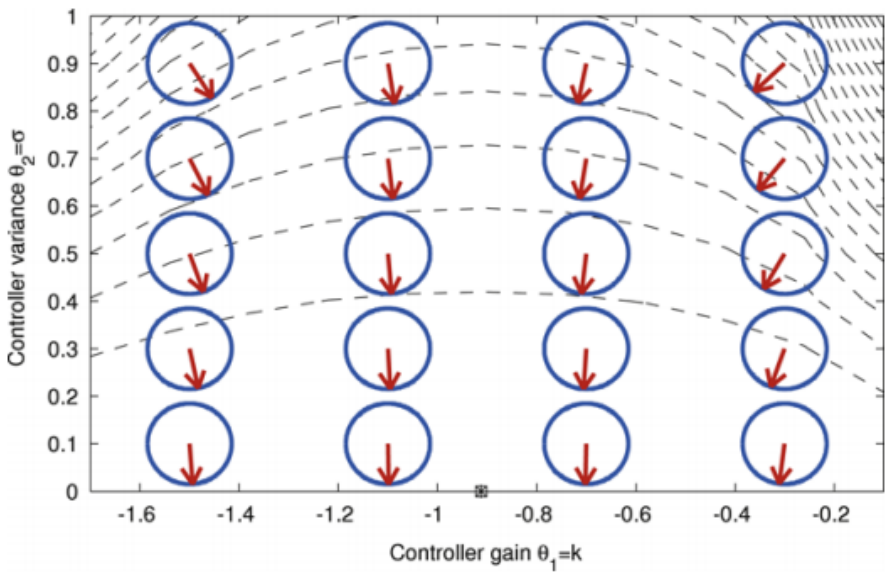
- Noisy policy generates **noisy data**
- Policy gradient approaches still **perform poorly**
- Because of **plateaus** and **discontinuities** in policy space
- **Slow** convergence or high **non-stationarity**
- Solution: make **changes** in θ proportional to **policy variation**
- Possible metric: **KL-divergence** between $\widehat{\pi}(\theta_t)$ and $\widehat{\pi}(\theta_{t+1})$

$$KL(\widehat{\pi}(\theta_t) \,||\, \widehat{\pi}(\theta_{t+1})) = \int_{S\times A} \widehat{\pi}(a|s,\theta_t) \ln\left(\frac{\widehat{\pi}(a|s,\theta_t)}{\widehat{\pi}(a|s,\theta_{t+1})}\right) \mathrm{d}(s,a)$$

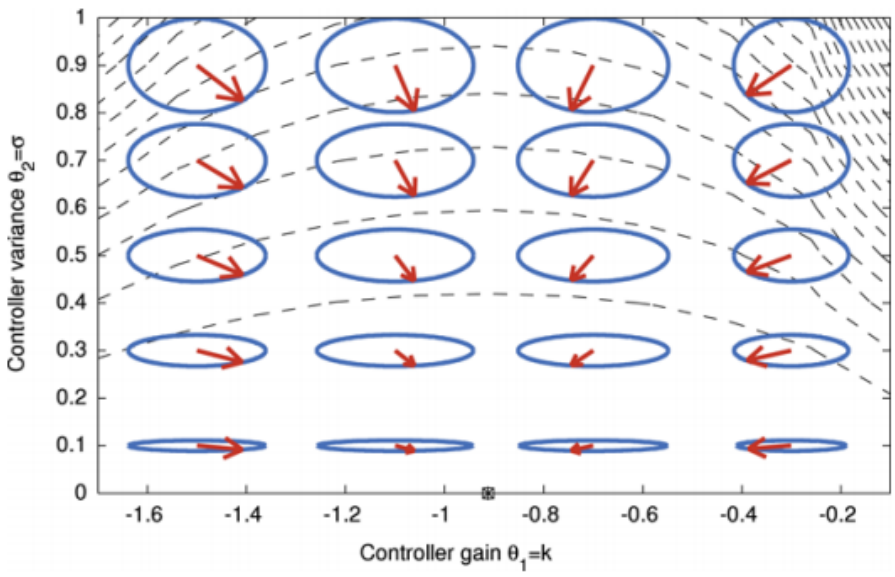
- It measures the **loss of information**
- **Gradient reformulation**

$$\nabla_\theta L(\theta_t) \leftarrow \text{optimal } \nabla_\theta L(\theta_t) \text{ s.t. } KL(\widehat{\pi}(\theta_t), || \, \widehat{\pi}(\theta_{t+1})) \leq \varepsilon$$

Natural Policy Gradient (2/3)



(a) Vanilla policy gradient.



(b) Natural policy gradient.

- **NPG** locally **alters** the **parameters space search**
- Feasible gradients: **ellipses contours**
- **Vanilla** gradient considers all **parameters equally important**
- Dashed line: expected return level curves
- **Plateau for small values** of θ_2

Natural Policy Gradient (3/3)

- Optimal improving direction:

$$\Delta\theta \propto G_\theta^{-1} \nabla_\theta L(\theta)$$

- G_θ is the **Fisher information matrix**

$$G_\theta = \mathbb{E}_{\widehat{\pi}} \left[\left\langle \nabla_\theta \ln \widehat{\pi},\, \nabla_\theta \ln \widehat{\pi}^T \right\rangle \right]$$

- By Policy Gradient Theorem:

$$\Delta\theta \propto G_\theta^{-1} \nabla_\theta \ln \widehat{\pi} Q_{\widehat{\pi}}$$

- By Compatibility:

$$\Delta\theta \propto G_\theta^{-1} \nabla_\theta \ln \widehat{\pi} \nabla_\theta \ln \widehat{\pi}^T \omega = G_\theta^{-1} G_\theta \omega = \omega$$

Reinforcement Learning

teaching material

Lorenzo Fiaschi

a.y. 2024-2025

Course: Symbolic and Evolutionary Artificial Intelligence
University of Pisa
lorenzo.fiaschi@ing.unipi.it

Model-based RL

Model-based Reinforcement Learning (1/3)

- **New paradigm**:
  - First, learn the model from experience
  - Then, learn V, Q, π (planning)
- Model learning is a **supervised problem**
- **Model uncertainty** as an additional reasoning perspective
- **Two sources of approximation error**
- $M_\eta = \langle P_\eta, R_\eta \rangle$

Model-based Reinforcement Learning (2/3)

- **Possible learning models**
  - Table lookup
  - Linear expectation
  - Gaussian expectation
  - Gaussian process (good for distributions)
  - Deep belief networks
- Quick look at the simplest one: **Table lookup**

$$P^a_{ss'} = \frac{1}{N(s,a)} \sum_{t=1}^{T} \mathbb{1}(s, a, s')$$

$$R^a_s = \frac{1}{N(s,a)} \sum_{t=1}^{T} \mathbb{1}(s, a) r_t$$

- Inferring a model **requires gathering experience**
- The **model** is used to generate **further experience**

Model-based Reinforcement Learning (3/3)

- Agent's **model** is just an **approximation** of the real one
- **Inaccurate models** lead to **sub-optimal policies**
- Solutions when **uncertainty** is **too high**:
  - Revert to model-free approaches
  - Introduce explicitly the model uncertainty in the reasoning
  - Midway-approach (integrating)
- **Integrating approaches**
  - Two sources of experience
  - Model learning from real experience (real source)
  - Planning from both model and experience (simulated and real)
- **Simulations** are used to explore and infer
- **Experience**: double check the supposed knowledge and act

Backward search

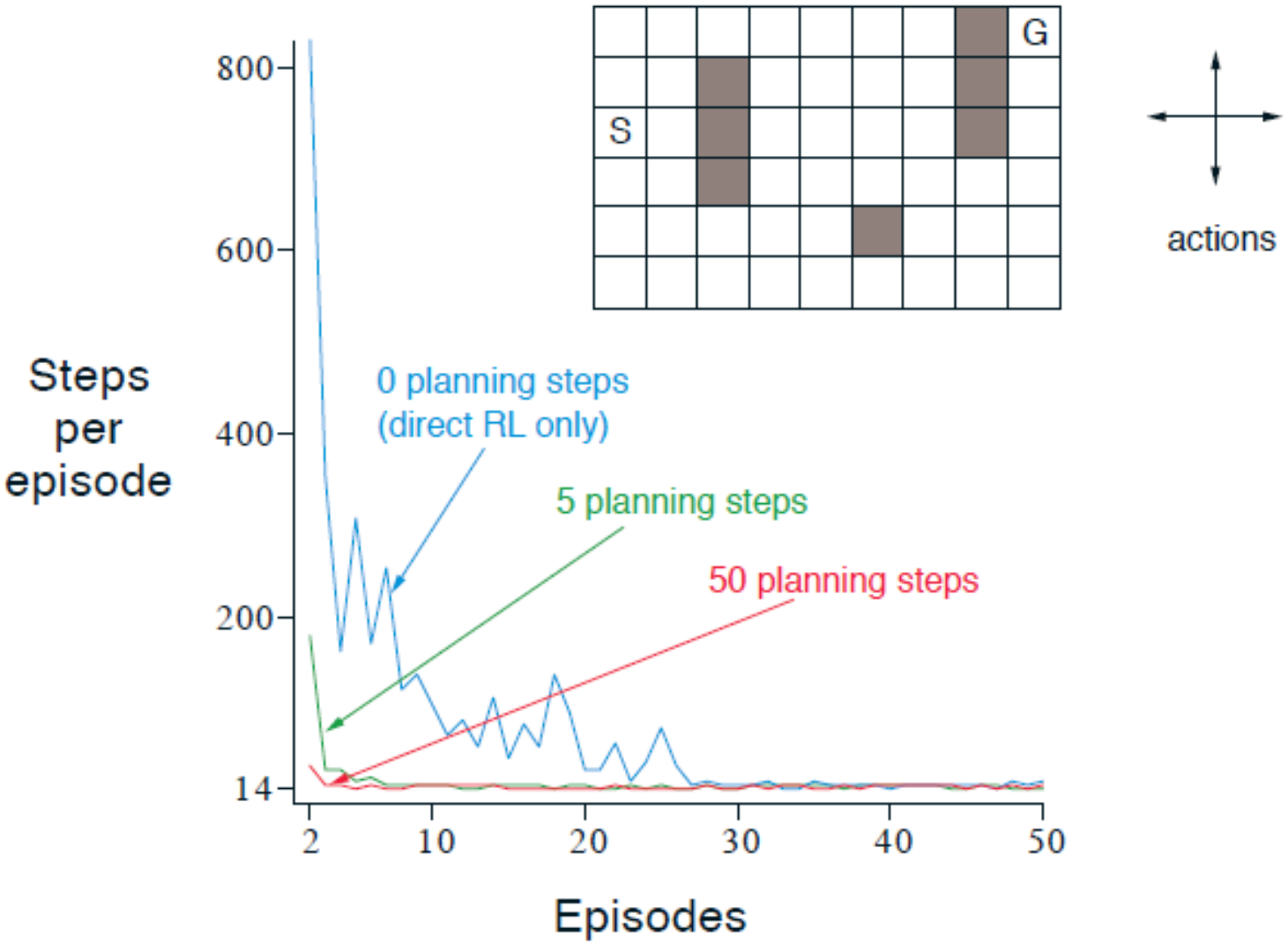
Dyna-Q algorithm

- **Two phases**: singleton experience and past multiple repetition
- Dyna-Q **automatically recovers** from **model variations**
- Dyna-Q+ draws samples from a **weighted distribution**
- Weights: $w_s(\tau) = \mathbb{E}_a[R_s^a] + k\sqrt{\tau}$

Algorithm 1 Dyna-Q algorithm

Input: S, A, γ
  Initialize Q and M
  while true do
    Choose $s \in S$
    $a \leftarrow \varepsilon-\texttt{greedy}(Q(s, \cdot))$
    Collect r and s'
    /* *Model update* */
    $M(s, a) \leftarrow \langle s', r' \rangle$
    TD(0) update of $Q(s, a)$
    for n times **do**
      Choose $s \in S$ from $\mathcal{H}$
      Choose $a \in A$ from $\mathcal{H}(s)$
      $\langle r, s' \rangle \leftarrow M(s, a)$
      TD(0) update of $Q(s, a)$
    end for
  end while

Dyna-Q at work



TD search and Dyna2

- **TD search** works online
  - SARSA is used because Q is needed
  - π is $\varepsilon-\texttt{greedy}(Q)$ optimized after each step
- **Dyna2**
  - Same ideas as Dyna
  - Uses two Q-functions
  - Persistent Q^P: updated only on experience
  - Transient Q^T: updated only on simulation and then cleared
  - Actions are chosen $\varepsilon-\texttt{greedy}(Q^P + Q^T)$

Forward search

Forward search

- **Select** action by **looking ahead**
- Construction of a **search tree**, current state as root
- It **recalls DP** breadth-search
- **Differences**:
  - Sample-based
  - Planning and not learning (data may be imperfect)
- **Naive MC search**
  - Does not improve π
  - Inefficient in the long run

Algorithm 2 Naive MC search

Input: M, S, A, s, π, k

for all $a \in A$ **do**

Simulate k episodes from s

$$Q(s,a) = \frac{1}{k} \sum_{t=1}^{k} G_t$$

end for

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

return a

MC tree search (1/2)

- Each step is constituted by **four phases:**

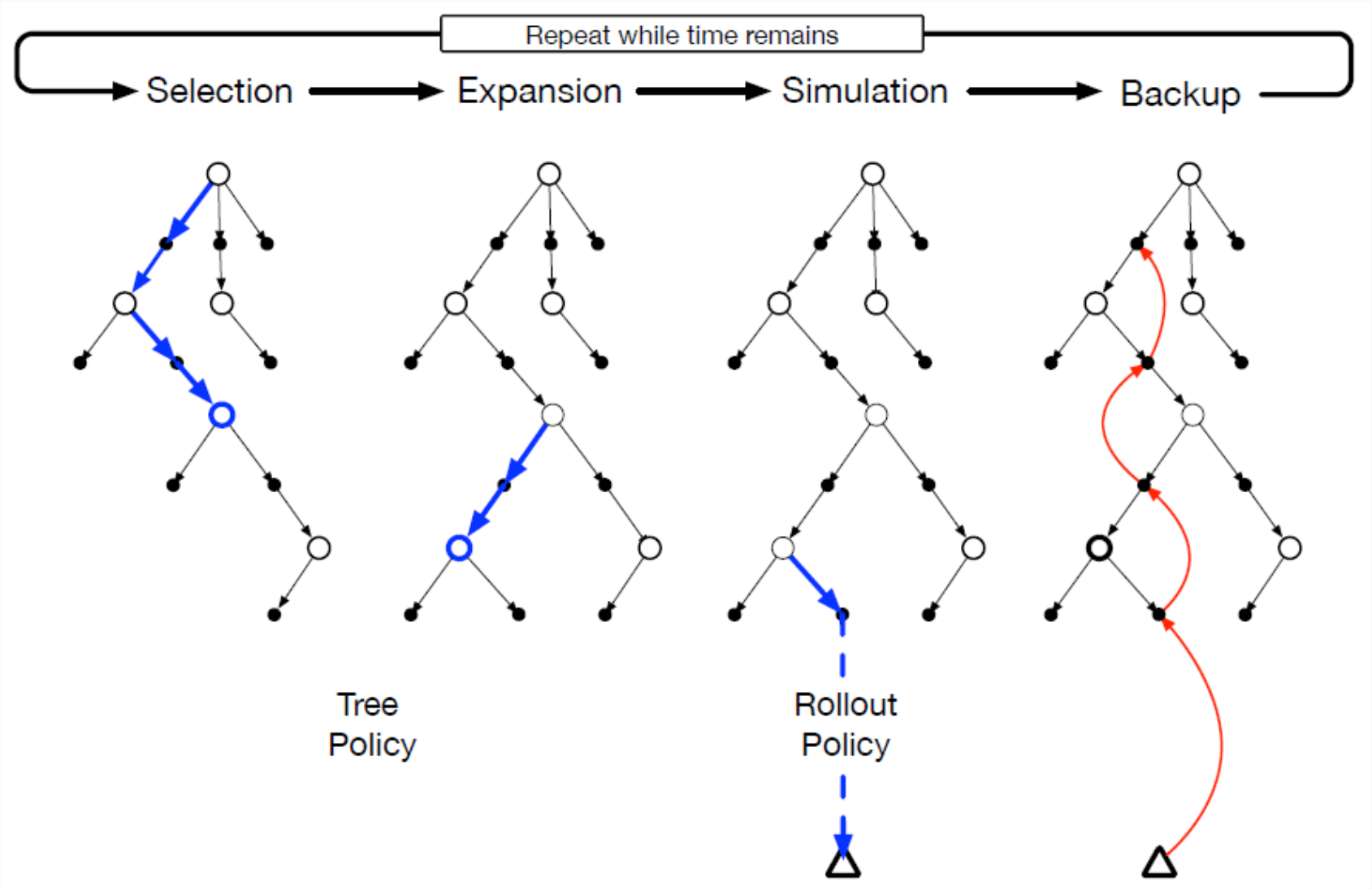
  - Selection - Expansion - Playout - Backprop.

- **Policy evaluation**

$$Q(s,a) = \frac{1}{N(s,a)} \sum_{k=1}^{K} \sum_{t=1}^{T} \mathbb{1}(s_t^k, a_t^k) G_t^k$$

- **Policy improvement** is $\varepsilon-\texttt{greedy}(Q)$
- **Action** taken: **greedy** one
- **Advantages**
  - Best-first search
  - Parallelizable
  - Exhaustive search may be impracticable

MC tree search (2/2)



Exploration-Exploitation tradeoff

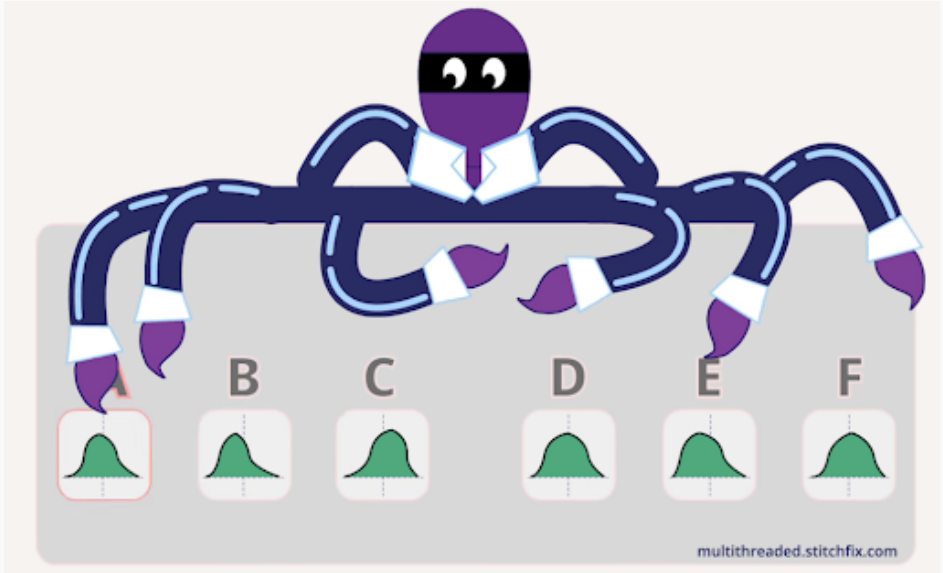
Exploration-Exploitation tradeoff

- **Forward search** asks for **good exploration**
- So far the **exploration** has been **random**
- **Noise injection** in greedy policy
- **Two more paradigms** are possible
- **Optimism**
  - Estimation of action-state value uncertainty
  - Exploration of more uncertain action-state pairs
- Use of **information state**
  - State-action space is enriched with history knowledge
  - Lookahead leverages such information to explore and to take actions

Multi-armed bandit scenario

- For now assume a **multi-armed bandit** scenario
- Deterministic starting state
- Multiple actions
- Each brings to a **different terminal state**
- **No state space**
- Goal: learn R^a





Regret (1/2)

- How to **measure exploration quality**?
- Asymptotic **regret**: loss w.r.t. the ideal optimal agent
- The lower the better
- Is the **same** as to **maximize** the expected **cumulative reward**
- Issue: optimal agent is **not known**
- It can be proved that:
  - Greedy algorithms $\rightarrow$ linear regret
  - ε-greedy algorithms $\rightarrow$ linear regret

Regret (2/2)

- **Other empirical approaches are possible**:
  - High-confidence elimination
  - Successive elimination
  - ...
- Goal: implement an oracle-less sublinear regret exploration
- **Hoeffding's inequality**: upper bound on the probability that the mean of a random variable deviates from its expected value by a certain amount

Proposition (Hoeffding's inequality)

Let $X_1, \ldots, X_t$ be i.i.d. random variables in [0,1] almost surely, and let $\bar{x}_t = \frac{1}{t}\sum_{i=1}^{t} x_i$ be the sample mean. Then,

$$\mathbb{P}(\mathbb{E}[x] - \bar{x}_t \geq u) = \mathbb{P}(\mathbb{E}[x] \geq \bar{x}_t + u) \leq e^{-2tu^2}$$

Optimism

- **Choose action** considering **two factors**
  - Expected reward
  - Variance (uncertainty, potential improvement)
- **Upper confidence** for action a: $U_t(a)$
- $U_t(a) \propto \frac{1}{N_t(a)}$, larger uncertainty for rarely taken actions
- Take action which **maximizes** the **upper confidence bound**:

$$a = \arg\max_a Q_t(a) + U_t(a)$$

Upper Confidence Bound algorithm (UCB)

- **Rewriting of Hoeffding's inequality**

$$\mathbb{P}(Q_t(a)^* \geq Q_t(a) + U_t(a)) \leq e^{-2N_t(a)U_t^2(a)}$$

- Ask for a **level of uncertainty** equal to p

$$e^{-2N_t(a)U_t^2(a)} = p \Longrightarrow U_t(a) = \sqrt{\frac{-\ln p}{2N_t(a)}}$$

- **Uncertainty** must **decay** in time (as ε)

$$p = t^{-4} \Longrightarrow U_t(a) = \sqrt{\frac{2 \ln t}{N_t(a)}}$$

- **Sublinear regret**
- Guarantee of asymptotic **convergence** to **optimal action**

Reinforcement Learning

teaching material

Lorenzo Fiaschi

a.y. 2024-2025

Course: Symbolic and Evolutionary Artificial Intelligence
University of Pisa
lorenzo.fiaschi@ing.unipi.it

Proximal Policy Gradient

Trust policy region is demanding

- **Goal**: limiting the policy variation
- Using KL divergence is demanding
- Exception: natural policy gradient
- **Idea**: use proxy for it
- Two algorithms:
  - PPO-Penalty
  - PPO-Clipping

Loss function

$$L(\theta) = \mathbb{E}_t \left[\frac{\pi_{\theta_k}(a_t|s_t)}{\pi_{\theta_{k-1}}(a_t|s_t)} A_t - \beta_n \, KL(\pi_{\theta_k}(\cdot|s_t), \, \pi_{\theta_{k-1}}(\cdot|s_t)) \right]$$

Weight update

$$\beta_{n+1} = \begin{cases} \frac{\beta_n}{2} & \text{if} \quad d < \frac{2}{3}\overline{d} \\ 2\beta_n & \text{if} \quad d > \frac{3}{2}\overline{d} \\ \beta_n & \text{otherwise} \end{cases}$$

where $\overline{d}$ is a user-defined threshold

Loss function

$$L(\theta) = \mathbb{E}_t \left[\min \left(\frac{\pi_{\theta_k}(a_t|s_t)}{\pi_{\theta_{k-1}}(a_t|s_t)}, \; g(\varepsilon, A_t) \right) A_t \right]$$

where

$$g(\varepsilon, A) := \begin{cases} (1+\varepsilon) & \text{if} \quad A > 0 \\ (1-\varepsilon) & \text{otherwise} \end{cases}$$

Deep Deterministic Policy Gradient

New hypotheses

$$\pi : S \times \Theta \to A, \qquad Q : A \times \Omega \to \mathbb{R}$$

Theorem (Deterministic Policy Gradient Theorem)

Let $L(\theta) = \mathbb{E}_{\pi_\theta}[R(s, \pi_\theta(s)) | s \sim d(S)]$, $\pi, R, P \in C^1$, *then*

$$\nabla_\theta L \sim \mathbb{E}_{\pi_\theta}[\nabla_\theta \pi_\theta(s) \nabla_a Q(s,a)|_{a=\pi_\theta(s)} \,|\, s \sim d(S)]$$

Theorem (Deterministic compatible function approximation)

Let $Q\colon A \times \Omega \to \mathbb{R}$, $Q \in C^1$, *then the Deterministic Policy Gradient Theorem continues to hold if*

- $\nabla_a Q_\omega(s,a)|_{a=\pi_\theta(s)} = \nabla_\theta \pi_\theta(s)^T \omega$
- $\mathbb{E}_{\widehat{\pi}}[(\nabla_a Q_\omega(s,a) - \nabla_a Q(s,a)|_{a=\pi_\theta(s)})^2] \xrightarrow{t\to\infty} 0$

From which one derives that

$$Q_\omega(s,a) = a^T \nabla_\theta \pi_\theta(s)^T \omega$$

Update rule

General case

$$\delta_t = r_t + \gamma Q_{\omega_t}(s_{t+1}, \pi_\theta(s_{t+1})) - Q_{\omega_t}(s_t, a_t)$$

$$\theta_{t+1} = \theta_t + \lambda_\theta \nabla_\theta \pi_\theta(s) \nabla_\theta \pi_\theta(s)^T \omega_t$$

$$\omega_{t+1} = \omega_t + \lambda_\omega \delta_t \nabla_\theta \pi_\theta(s) a$$

Modification for natural policy

$$\theta_{t+1} = \theta_t + \lambda_\theta \omega_t$$

Twin Delayed DDPG

Limitations of DDPG

- DDPG is sensitive to hyperparameters and other tunings
- DDPG tends to overestimate Q

Three tricks

- Target Policy Smoothing
- Clipped Double-Q Learning
- Delayed Policy Updates

Target Policy Smoothing

- TD3 adds noise to the action chosen
- This makes it harder for π to exploit Q errors
- Peaks are particularly detrimental

Action selection

$$a = \text{clip}(\pi(s) + \text{clip}(\varepsilon, -c, c), a_\ell, a_h)$$

where

$$a_\ell,\ a_h \in \mathbb{R},\ a_\ell < a_h,\ c \in \mathbb{R}^+,\ \varepsilon \in \mathcal{N}(0, 1)$$

Clipped Double-Q Learning

- TD3 learns two Q-functions instead of one (hence “twin”)
- It uses the smaller of the two values for the Bellman error

New loss function

$$L(\theta_i) = \mathbb{E}_{\mathcal{D}}\left[\left(r + \gamma \min_{j=1,2} Q(s', a', \omega^-) - Q(s, a, \omega)\right)^2\right]$$

Policy Learning

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{\pi}[Q(\cdot, \cdot, \omega_1)]$$

Delayed Policy Updates

- TD3 updates the π and ω^- less frequently than the Q-function
- Suggested ratio is 1:2
- This allows for better policy improvement directions
- Indeed, the critic suggests the improvements for the actor

Soft Actor Critic

Another way for exploration - exploitation tradeoff

Entropy regularization to enforce exploration

Definition (Entropy)

Given a probability density function P, the entropy H of a random variable x drawn from it is defined as

$$H(P) := -\mathbb{E}_{x\sim P}[\ln(P(x))]$$

Another way for exploration - exploitation tradeoff

Change in the RL paradigm

$$\pi^* = \arg\max \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \left(R(s_t, a_t) + \nu\, H(\pi(\cdot|s_t))\right)\right]$$

where

$$\nu \in \mathbb{R}^+$$

Entropy-based Bellmann equation

$$Q_\pi(s, a) = \mathbb{E}_{\substack{s' \sim P \\ a' \sim \pi}} \left[R(s, a) + \gamma \left(Q_\pi(s', a') - \nu\, \ln(\pi(a'|s'))\right)\right]$$

Hands on

- The expectation term is replaced by sampling a replay buffer
- SAC uses the clipped double-Q trick of TD3
- The entropy weight factor ν is decreased in time
- The learning of Q is a implemented through MSBE

Policy update

Loss function

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{a\sim\pi}\left[\min_{j=1,2} Q_{\omega_j}(s,a) - \nu \ln \pi_\theta(a|s)\right]$$

- The expectation is computed over actions depending on θ
- This is tedious to compute
- **Reparameterization trick**: move the dependence on an independent noise

$$a_\theta(s,\xi) := \tanh(\mu_\theta(s) + \sigma_\theta(s) \odot \xi)$$

given

$$\xi \sim \mathcal{N}(0,1), \quad \pi_\theta \sim \mathcal{N}(\mu_\theta, \sigma_\theta)$$

New loss function

Since

$$\mathbb{E}_{a\sim\pi}\left[\min_{j=1,2} Q_{\omega_j}(s,a) - \nu \ln \pi_\theta(a|s)\right] =$$

$$= \mathbb{E}_{\xi\sim\mathcal{N}}\left[\min_{j=1,2} Q_{\omega_j}(s,a_\theta(s,\xi)) - \nu \ln \pi_\theta(a_\theta(s,\xi)|s)\right]$$

the new loss function is the RHS