UPORTO
FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

Agents and Multi-Agent Systems

Multi-Agent Decision Making

Reinforcement Learning

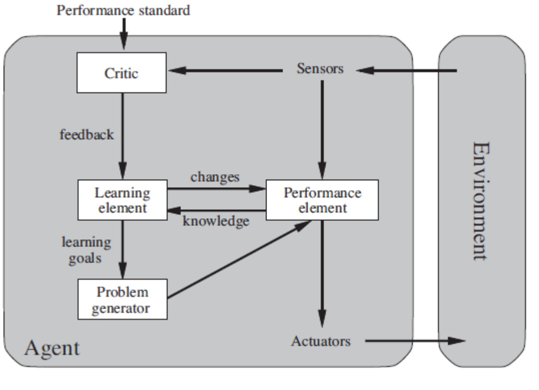
2023/2024

Ana Paula Rocha, Henrique Lopes Cardoso

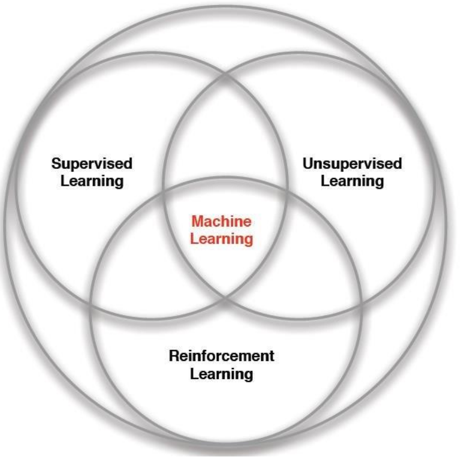
UPORTO
FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

Learning Agents

- Operate in initially unknown environments and become more competent than their initial knowledge, through learning

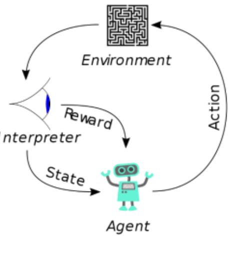


Machine Learning



What is Reinforcement Learning?

- Reinforcement Learning (RL) is focused on goal-directed learning from interaction
- RL is learning what to do – how to map situations to actions – so as to maximize a numerical reward signal
  - The learner is not told which actions to take: it must discover which actions yield the most reward by trying them
  - Typically, actions may affect not only immediate reward but also the next situation and subsequent rewards



➢ Goal can be described by the maximization of expected cumulative reward

U.PORTO FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

Formulating RL

- World described by a set of states and actions
- At every time step t, we are in a state s_t, and we:
  - Take an action a_t (possibly null action)
  - Receive some reward r_{t+1}
  - Move into a new state s_{t+1}

- RL include the following elements:
  - Policy π: behaviour function
  - Reward r: environment's feedback
  - Value function: how good is each state and/or action
  - Model (if model-based): representation of the environment

→ We seek actions that bring about states of highest value, not highest reward, because these actions obtain the greatest amount of reward over the long run

U.PORTO FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

Elements of RL

- Policy π
  - How should the agent behave over time?
  - It is a selection of which action to take, based on the current state
  - Deterministic policy: $a = \pi(s)$
  - Stochastic policy: $\pi(a|s) = P[a_t = a|s_t = s]$

- Reward signal r
  - Defines the *goal* of the RL problem
  - On each time step, the environment sends a reward to the RL agent

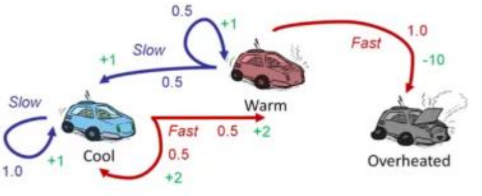
- Value function v
  - Specifies what is good in the long run
  - The value of a state is the *total amount of reward* an agent can expect to accumulate from that state onwards (it takes into account future rewards)

Elements of RL

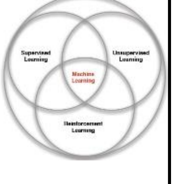
- Model of the environment
  - In model-based methods, allows inferences about how the environment will behave
  - The model describes the environment by a distribution over rewards and state transitions:

$$P(s_{t+1} = s' \,,\, r_{t+1} = r' \mid s_t = s \,, a_t = a)$$

  - We assume the Markov property: the future depends on the past only through the current state



RL vs (Un)Supervised Learning



- Different from supervised learning
  - In interactive problems it is impractical to obtain examples of desired behavior
  - In uncharted territory, an agent must learn from its own experience
- Different from unsupervised learning
  - In RL we try to maximize a reward signal, we do not seek to find hidden structure in collections of unlabeled data
- RL explicitly considers the *whole* problem of a goal-directed agent interacting with an uncertain environment
  - Creating a behavior model while applying it in the environment
- RL is the closest form of ML to the kind of learning humans do

Learning to Play Tic-Tac-Toe

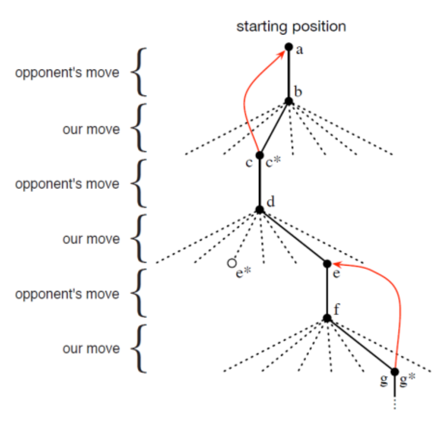
- *Rule-based approach*
  - Need to hardcode rules for each possible situations that might arise in a game
- *Minimax*
  - Assumes a particular way of playing by the opponent

X	O	O
O	X	X
		X

- *Dynamic programming* can compute an optimal solution for any opponent
  - But requires as input a complete specification of that opponent (state/action probabilities)
- Can we obtain such information *from experience*?
  - Play many games against the opponent!

Learning to Play Tic-Tac-Toe

- States
  - Possible configurations of the board
- Actions
  - Possible moves to make
- Policy
  - Which action should I play in each state?
- Reward
  - How good was the chosen action?



U.PORTO FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

Sequential Decision Making

- Goal: select actions that maximize total future reward

- Actions may have long term consequences
- Reward may be delayed
- It may be better to sacrifice immediate reward to gain more long-term reward

- Examples:
  - A financial investment (may take months to mature)
  - Refueling a helicopter (might prevent a crash in several hours)
  - Blocking opponent moves (might help winning chances later on)

12

U.PORTO FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

Exploration vs Exploitation

- How can an agent find the best actions while maximizing the expected cumulative reward?

- Exploitation
  - Prefer actions known (or estimated) to be effective (in producing reward)
  - Higher short-term reward
- Exploration
  - Try actions not selected before
  - Improve estimates on action values (particularly in stochastic tasks)
    - Lower reward in the short run, but higher in the long run

➢ The exploration-exploitation tradeoff
  - Neither exploration nor exploitation can be pursued exclusively
  - Try a variety of actions and progressively favor those that appear to be best

14

U.PORTO FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

Reinforcement Learning

MULTI-ARMED BANDITS

15

U.PORTO FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

Bandit Problems

- A simple setting with a single state

- *K*-armed bandit problem
  - There are *k* different actions
  - After each action a numerical reward is received from a stationary probability distribution
  - Each action has a *value* – its expected or mean reward, not known by the agent: $q_*(a) \doteq \mathbb{E}[R_t | A_t = a]$
  - The agent *estimates*, at time step t, the value of an action a: $Q_t(a)$

16

U.PORTO FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

Estimating Action Values

- Sample average:

$$Q_t(a) \doteq \frac{\text{sum of rewards when } a \text{ taken prior to } t}{\text{number of times } a \text{ taken prior to } t} = \frac{\sum_{i=1}^{t-1} R_i \cdot 1_{A_i=a}}{\sum_{i=1}^{t-1} 1_{A_i=a}}$$

- *Update rule:*

$$Q_{n+1} = Q_n + \frac{1}{n}[R_n - Q_n]$$

$$\boxed{NewEstimate \leftarrow OldEstimate + StepSize[Target - OldEstimate]}$$

  - The target indicates a desirable direction in which to move
  - The *step-size* parameter changes from time step to time step

- Giving more weight to recent rewards – constant *step-size* parameter:

$$Q_{n+1} \doteq Q_n + \alpha[R_n - Q_n]$$

where $\alpha \in (0,1]$

17

U.PORTO FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

Action Selection

- *Greedy* action selection (always exploits): $A_t \doteq \underset{a}{\operatorname{argmax}}\, Q_t(a)$

- *ε-greedy* action selection: behave greedily most of the time, but with small probability ε select randomly from among all the actions
  – $Q_t(a)$ will converge to $q_*(a)$ if a is selected sufficiently often

- *Soft-max* action selection (Boltzmann distribution):

$$\Pr\{A_t = a\} \doteq \frac{e^{Q_t(a)/\tau}}{\sum_{b=1}^{k} e^{Q_t(b)/\tau}}$$

where τ is a temperature parameter:

– if high, actions will tend to be equiprobable;

– if low, action values matter more;

– if $\tau \to 0$, then we have greedy action selection

18

U.PORTO
FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

Bandit Algorithm

A simple bandit algorithm

Initialize, for $a = 1$ to k:
$Q(a) \leftarrow 0$
$N(a) \leftarrow 0$

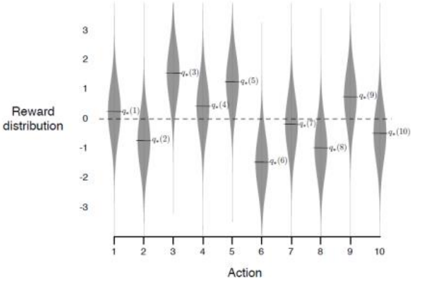
Loop forever:

$$A \leftarrow \begin{cases} \arg\max_a Q(a) & \text{with probability } 1 - \varepsilon \quad \text{(breaking ties randomly)} \\ \text{a random action} & \text{with probability } \varepsilon \end{cases}$$

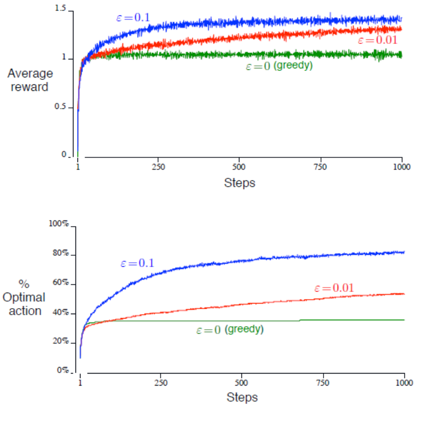
$R \leftarrow bandit(A)$
$N(A) \leftarrow N(A) + 1$
$Q(A) \leftarrow Q(A) + \frac{1}{N(A)}\left[R - Q(A)\right]$

U.PORTO
FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

The 10-armed Testbed



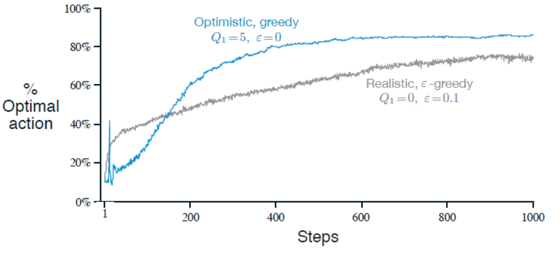
averages over 2000 runs



Optimistic Initial Values

- Methods for action selection are dependent on the initial action-value estimates
  - They are *biased* by their initial estimates
- Initial estimates are useful to:
  - Supply some prior knowledge about what level of rewards can be expected
  - Encourage initial exploration

- Using optimistic initialization: $Q_1(a) = +5$, for all a



Reinforcement Learning

MARKOV DECISION PROCESSES

U.PORTO FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

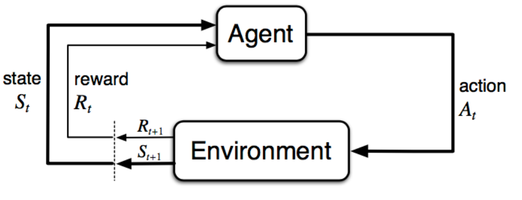
Markov Decision Processes

- In the general setting we have many states
- Markov Decision Processes (MDP) are a classical formalization of sequential decision making
  - The environment is fully observable
  - Actions influence not just immediate rewards, but also subsequent situations (states) and thus future rewards
- In a finite MDP, there is a finite number of states, actions and rewards
- In MDPs we estimate the value $q_*(s,a)$

23

U.PORTO FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

Agent-Environment Interface



- Dynamics of the MDP:

$$p(s',r|s,a) = \Pr\{s_t = s', r_t = r \mid s_{t-1} = s, a_{t-1} = a\}$$

  - The probability of each possible value for s' and r depends only on the immediately preceding state s and action a
  - The state must include all relevant information about the past agent-environment interaction – Markov property

24

Example: Recycling Robot

- A robot has to decide whether it should (1) actively search for empty soda cans, (2) wait for someone to bring it a can, or (3) go to home base and recharge
- Searching is better (higher **probability** of getting a can) but runs down *battery*; if out of battery, the robot has to be rescued
- Decisions made on the basis of current energy level: high, low
- **Reward** is mostly **zero**, **positive** when getting a can, and **negative** if out of battery

$$\mathcal{S} = \{high, low\}$$

$$\mathcal{A}(high) = \{search, wait\}$$

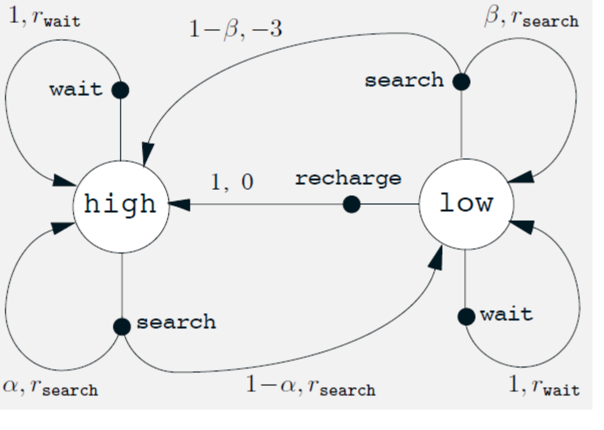
$$\mathcal{A}(low) = \{search, wait, recharge\}$$

$$r_{search} > r_{wait}$$

s	a	s'	$p(s' \mid s, a)$	$r(s, a, s')$
high	search	high	α	$r_{\texttt{search}}$
high	search	low	$1-\alpha$	$r_{\texttt{search}}$
low	search	high	$1-\beta$	-3
low	search	low	β	$r_{\texttt{search}}$
high	wait	high	1	$r_{\texttt{wait}}$
high	wait	low	0	-
low	wait	high	0	-
low	wait	low	1	$r_{\texttt{wait}}$
low	recharge	high	1	0
low	recharge	low	0	-

Example: Recycling Robot

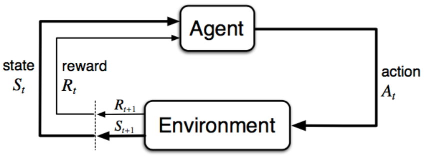
- Transition graph



U.PORTO FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

Goals and Rewards

A reward signal, from the environment to the agent, is used to define the goal of the agent



- Learning to walk: reward on each time step proportional to the robot's forward motion
- Learning to Escape from a maze: reward -1 for any state prior to escape (encourage escaping as quickly as possible)
- Learning to find empty soda cans for recycling: reward of 0 most of the time, +1 for each can collected
- Learning to play checkers or chess: reward +1 for winning, -1 for losing, and 0 for drawing and nonterminal positions

U.PORTO FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

Goals and Rewards

- Provide rewards in such a way that by maximizing them the agent will also achieve the goal
  - The agent's goal is to maximize the cumulative reward it receives in the long run
- It is critical that the rewards we set up truly indicate what we want accomplished

→ The reward signal is a way of communicating to the agent *what* you want it to achieve, not *how* – it is ***not*** meant to encode prior knowledge (it is part of the environment, not the agent!)

U.PORTO FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

Returns and Discounting

- Agent wants to maximize expected return

  $$G_t = r_{t+1} + r_{t+2} + r_{t+3} + \cdots + r_T$$

- Adding discounting: agent wants to maximize expected discounted return

  $$G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} = r_{t+1} + \gamma G_{t+1}$$

  – $0 \leq \gamma \leq 1$ is the discount rate: the present value of future rewards
    - The value of receiving reward r after $k+1$ steps is $\gamma^k r$
    - If $\gamma = 0$ the agent is "myopic" (only immediate reward matters)
    - As γ approaches 1, the agent becomes more farsighted (strongly considers future rewards)

  – G_t is now finite (if $\gamma < 1$), even if summing an infinite number of terms
    - for instance, if reward is always +1: $G_t = \sum_{k=0}^{\infty} \gamma^k = \frac{1}{1-\gamma}$

U.PORTO FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

Episodic/Continuing Tasks

- Episodic tasks: when the agent-environment interaction breaks naturally into subsequences – episodes
  – From a *starting state* to a *terminal state*
  – Followed by a reset to another starting state, chosen independently of how the previous episode ended

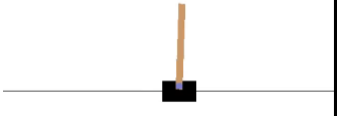
- Continuing tasks do not break naturally into identifiable episodes (*e.g.,* on-going process-control)
  – The final timestep is ∞, so we can't really compute a useful G_t
  – Problem with calculating G_t:
    - $T = \infty$
    - G_t could also be infinite (if rewards are positive at each time step)

Example: Pole Balancing

- Move a cart so as to keep a pole from falling over
  - Failure if the pole falls past a given angle or if the cart runs off the track
  - The pole is reset to vertical after each failure

- Episodic task: repeated attempts to balance the pole
  - reward +1 except when failure
  - return is the number of steps until failure

- Continuing task, using discounting:
  - reward –1 on each failure and 0 otherwise
  - return is $-\gamma^K$, where K is the number of steps before failure



Value Functions

- Most RL algorithms involve estimating value functions
  - How good (in terms of expected return) is it to be in a given state?
  - How good is it to perform a given action in a given state?

- Bellman Equation: the value function can be recursively decomposed into two parts
  - Immediate reward R_{t+1}
  - Discounted value of successor state $\gamma v(S_{t+1})$

$$
\begin{aligned}
v(s) &= \mathbb{E}\left[G_t \mid S_t = s\right] \\
&= \mathbb{E}\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots \mid S_t = s\right] \\
&= \mathbb{E}\left[R_{t+1} + \gamma\left(R_{t+2} + \gamma R_{t+3} + \dots\right) \mid S_t = s\right] \\
&= \mathbb{E}\left[R_{t+1} + \gamma G_{t+1} \mid S_t = s\right] \\
&= \mathbb{E}\left[R_{t+1} + \gamma v(S_{t+1}) \mid S_t = s\right]
\end{aligned}
$$

U.PORTO
FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

Policies and Value Functions

- Future rewards depend on the choice of actions
  - Value functions are defined with respect to policies (ways of acting)
  - Policy: a mapping from states to probabilities of selecting each possible action
    - $\pi(a|s) = \Pr(A_t = a|S_t = s)$
- State-value function $v_\pi(s)$
  - Expected return when starting in s and following π thereafter
  - $v_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s]$
- Action-value function $q_\pi(s, a)$
  - Expected return when taking action a in state s, and following π thereafter
  - $q_\pi(s, a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a]$

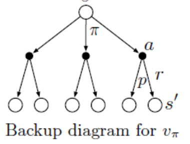
33

U.PORTO
FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

Bellman Equation

- *Bellman equation* for v_π: looking ahead from a state to its possible successor states

$$
\begin{aligned}
v_\pi(s) &\doteq \mathbb{E}_\pi[G_t \mid S_t = s] \\
&= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s] \\
&= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a) \Big[r + \gamma \mathbb{E}_\pi[G_{t+1}|S_{t+1} = s'] \Big] \\
&= \sum_a \pi(a|s) \sum_{s', r} p(s', r|s, a) \Big[r + \gamma v_\pi(s') \Big], \quad \text{for all } s \in \mathcal{S}
\end{aligned}
$$



Backup diagram for v_π

*Expected return as a sum over possible action choices made by the agent in state **s***

*Summation of all joint probabilities of possible rewards and next states condition on state **s** and action **a***

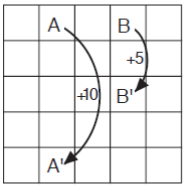
*Sum of immediate reward and expected future returns from the next state **s'***

  - Averages over all the possibilities, weighting each by its probability of occurring

34

Example

- Example: using a random policy, with $\gamma = 0.9$:



Gridworld


Actions

- Off-grid actions have no effect, with $r = -1$
- Any action from A gets to A', with $r = +10$
- Any action from B gets to B', with $r = +5$

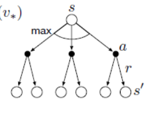
3.3	8.8	4.4	5.3	1.5
1.5	3.0	2.3	1.9	0.5
0.1	0.7	0.7	0.4	-0.4
-1.0	-0.4	-0.4	-0.6	-1.2
-1.9	-1.3	-1.2	-1.4	-2.0

v_π

Optimal Policy and Value Function

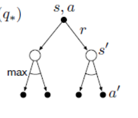
- Optimal state-value function
  - The maximum value function over all policies

$$v_*(s) = \max_\pi v_\pi(s)\,, \forall s \in \mathcal{S}$$

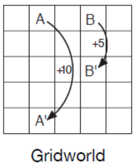


- Optimal action-value function
  - The maximum action-value function over all policies

$$q_*(s,a) = \max_\pi q_\pi(s,a), \forall s \in \mathcal{S}, \forall a \in \mathcal{A}$$



- Once we know v_* or q_*, the optimal policy π_* is greedy
  - The expected return is greater than any other policy

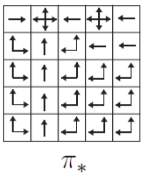


Gridworld


Actions

22.0	24.4	22.0	19.4	17.5
19.8	22.0	19.8	17.8	16.0
17.8	19.8	17.8	16.0	14.4
16.0	17.8	16.0	14.4	13.0
14.4	16.0	14.4	13.0	11.7

v_*


π_*

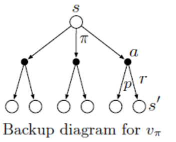
U.PORTO
FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

Policy Evaluation via DP

- Policy evaluation: computing the state-value function v_π for an arbitrary policy π

- Turning Bellman equation into an update:

$$\begin{aligned} v_{k+1}(s) &\doteq \mathbb{E}_\pi[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s] \\ &= \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)\Big[r + \gamma v_k(s')\Big] \end{aligned}$$



Backup diagram for v_π

- Iterative solution method: dynamic programming
  - We can maintain two arrays
    - One for the old values $v_k(s)$, one for the new values $v_{k+1}(s)$
  - Or make changes "in place", using a single array (faster convergence)

37

U.PORTO
FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

Iterative Policy Evaluation

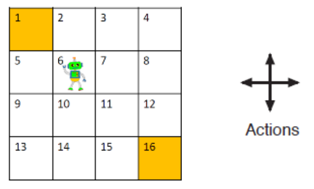
Iterative Policy Evaluation, for estimating $V \approx v_\pi$

Input π, the policy to be evaluated
Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

Loop:
  $\Delta \leftarrow 0$
  Loop for each $s \in \mathcal{S}$:
    $v \leftarrow V(s)$
    $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)\big[r + \gamma V(s')\big]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$

38

Grid World (example)



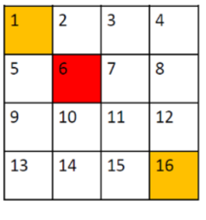
- A bot is required to traverse a grid of 4×4 dimensions to reach its goal (1 or 16)
- Deterministic actions $\mathcal{A} = \{\text{up}, \text{down}, \text{right}, \text{left}\}$
- There are 2 terminal states (1 and 16) and 14 non-terminal states (2 to 15)
- Each step is associated with a reward of -1
- Consider a **random policy**: $\pi(a|s) = 0.25, \forall s \in \mathcal{S}, \forall a \in \mathcal{A}$
- Initialize v_1 for the random policy with all 0s

Grid World: Policy Evaluation (example)

- Turning Bellman equation into an update:

$$\begin{aligned} v_{k+1}(s) &\doteq \mathbb{E}_\pi[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s] \\ &= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\Big[r + \gamma v_k(s')\Big] \end{aligned}$$

- Step 1



$$\begin{aligned} v_1(6) &= \sum_{a\in\{u,d,l,r\}} \pi(a|6) \sum_{s',r} p(s', r|6, a)[r + \gamma v_0(s')] \\ &= \sum_{a\in\{u,d,l,r\}} \underbrace{\pi(a|6)}_{= 0.25\ \forall a} \sum_{s'} p(s'|6, a)[\underbrace{r}_{=-1} + \gamma \underbrace{v_0(s')]}_{=0\ \forall s'} \\ &= 0.25 * \{-p(2|6, u) - p(10|6, d) - p(5|6, l) - p(7|6, r)\} \\ &= 0.25 * \{-1 - 1 - 1 - 1\} \\ &= -1 \end{aligned}$$

$$\Rightarrow v_1(6) = -1$$

Grid World: Policy Evaluation (example)

- For non-terminal states, $v_1(s) = -1$
- For terminal states, $p(s'|s,a) = 0$
  - and hence $v_k(1) = v_k(6) = 0$, for all k

$v_1 =$

0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0

U.PORTO
FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

Grid World: Policy Evaluation (example)

- Step 2, for red states in figure, with discount factor $\gamma = 1$

$$v_2(6) = \sum_{a\in\{u,d,l,r\}} \underbrace{\pi(a|6)}_{=0.25\ \forall a} \sum_{s'} p(s'|6,a)[\underbrace{r}_{=-1} + \gamma \underbrace{v_1(s')}_{=\begin{cases}-1, s'\in S \\ 0, s'\in S^+\backslash S\end{cases}}]$$

$$= 0.25 * \{p(2|6,u)[-1-\gamma] + p(10|6,d)[-1-\gamma] + p(5|6,l)[-1-\gamma] + p(7|6,r)[-1-\gamma]\}$$

$$\overset{\gamma=1}{=} 0.25 * \{-2-2-2-2\}$$

$$= -2$$

- Step2, for other states (2, 5, 12, 15):

$$v_2(2) = \sum_{a\in\{u,d,l,r\}} \underbrace{\pi(a|2)}_{=0.25\ \forall a} \sum_{s'} p(s'|2,a)[\underbrace{r}_{=-1} + \gamma \underbrace{v_1(s')}_{=\begin{cases}-1, s'\in S \\ 0, s'\in S^+\backslash S\end{cases}}]$$

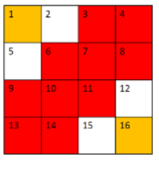
$$= 0.25 * \{p(2|2,u)[-1-\gamma] + p(6|2,d)[-1-\gamma] + p(1|2,l)[-1-\gamma*0] + p(3|2,r)[-1-\gamma]\}$$

$$\overset{\gamma=1}{=} 0.25 * \{-2-2-1-2\}$$

$$= -1.75$$

$$\Rightarrow v_2(2) = -1.75$$

For all red states, $v_2(s) = -2$

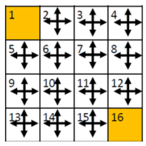


$v_2 =$

0.0	-1.7	-2.0	-2.0
-1.7	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.7
-2.0	-2.0	-1.7	0.0

U.PORTO FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

Grid World: Policy Evaluation (example)



Random policy

$k = 0$

0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0

$k = 1$

0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0

$k = 2$

0.0	-1.7	-2.0	-2.0
-1.7	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.7
-2.0	-2.0	-1.7	0.0

$k = 3$

0.0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0.0

…

$k = 10$

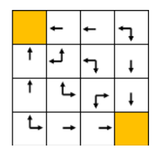
0.0	-6.1	-8.4	-9.0
-6.1	-7.7	-8.4	-8.4
-8.4	-8.4	-7.7	-6.1
-9.0	-8.4	-6.1	0.0

…

$k = \infty$

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0

$\leftarrow v_\pi$



Optimal policy

U.PORTO FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

Approximation

- Solving the Bellman optimality equation is equivalent to exhaustive search
  - Impractical for large state spaces
- RL methods can be understood as approximately solving it, using actual experienced transitions in place of knowledge of the expected transitions
  - Model-free approaches
- Optimal policies are computationally costly to find – we can only approximate
  - In tasks with small, finite state sets: tabular methods
  - Otherwise: function approximation using a more compact parameterized function representation (*e.g.* using neural networks)

→ The online nature of RL allows us to *put more effort into learning to make decisions for frequently encountered states*

U.PORTO
FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

Reinforcement Learning

TEMPORAL-DIFFERENCE LEARNING

U.PORTO
FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

Model-Free RL

- In model-free methods, we are not given the MDP
  - I.e., we do not assume complete knowledge of the environment
  - We learn directly from *actual* experience, by interacting with the environment

- Two main approaches:
  - Monte Carlo learning
    - Average returns from full sample sequences of states, actions, and rewards (episodic MDPs)
  - Temporal-Difference learning
    - Update estimates based in part on other learned estimates, without waiting for a final outcome
    - Combination of *Monte Carlo* ideas and *Dynamic Programming* ideas

TD(λ)

- Unifies both approaches

U.PORTO FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

Monte Carlo

- Value of a state is *estimated from experience*, by average returns observed after visits the state
  - First-visit MC: average of the returns following first visits to state
  - Every-visit MC: average the returns following all visits to s

First-visit MC prediction, for estimating $V \approx v_\pi$

Input: a policy π to be evaluated
Initialize:
  $V(s) \in \mathbb{R}$, arbitrarily, for all $s \in \mathcal{S}$
  $Returns(s) \leftarrow$ an empty list, for all $s \in \mathcal{S}$

Loop forever (for each episode):
  Generate an episode following π: $S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T$
  $G \leftarrow 0$
  Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:
    $G \leftarrow \gamma G + R_{t+1}$
    Unless S_t appears in $S_0, S_1, \ldots, S_{t-1}$:
      Append G to $Returns(S_t)$
      $V(S_t) \leftarrow \text{average}(Returns(S_t))$

Every-visit MC: without this checking

U.PORTO FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

Temporal-Difference Learning

- TD methods update estimates based on immediately observed reward and state
- Update rule: $V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$
- Because TD bases its update in part on an existing estimate (incomplete episodes), it is a *bootstrapping* method

Tabular TD(0) for estimating v_π

Input: the policy π to be evaluated
Algorithm parameter: step size $\alpha \in (0, 1]$
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    $A \leftarrow$ action given by π for S
    Take action A, observe R, S'
    $V(S) \leftarrow V(S) + \alpha[R + \gamma V(S') - V(S)]$
    $S \leftarrow S'$
  until S is terminal

U.PORTO FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

Temporal-Difference Learning

- TD vs Dynamic Programming methods
  - TD methods do not require a model of the environment's dynamics (rewards and next-state probability distributions)

- TD vs Monte Carlo methods
  - TD methods are naturally implemented in an online, fully incremental fashion, while MC methods must wait until the end of an episode
    - Useful if episodes are very long, or in continuing tasks (that have no episodes at all)

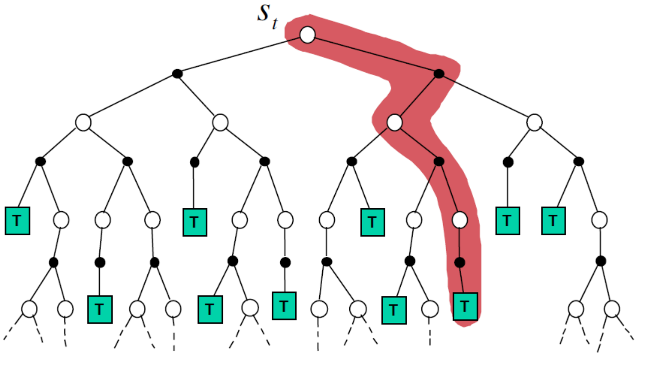
- TD combines the sampling of Monte Carlo with the bootstrapping of DP

- Usually, TD methods converge faster than MC methods on stochastic tasks

U.PORTO FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

Monte-Carlo Backup

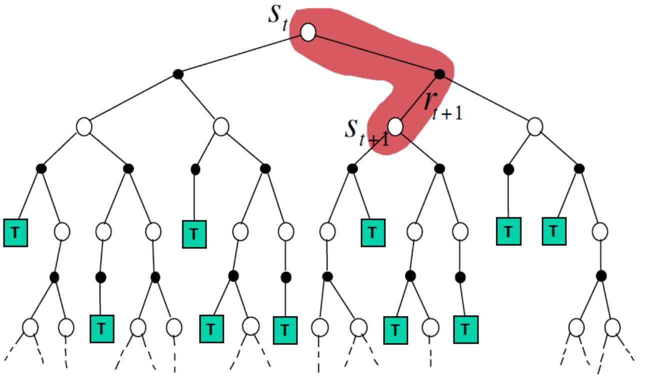
$$V(S_t) \leftarrow V(S_t) + \alpha\,(G_t - V(S_t))$$



U.PORTO FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

Temporal-Difference Backup

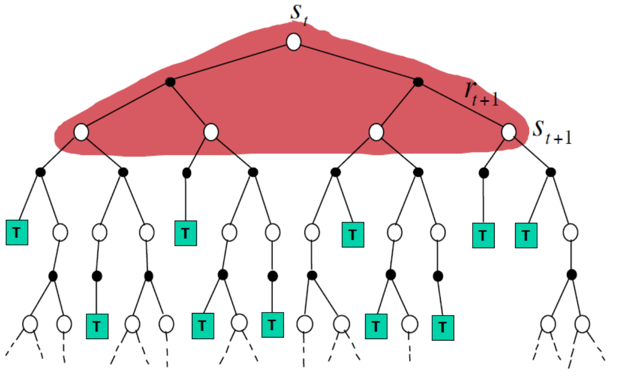
$$V(S_t) \leftarrow V(S_t) + \alpha \left(R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\right)$$



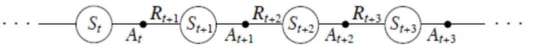
U.PORTO FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

Dynamic Programming Backup

$$V(S_t) \leftarrow \mathbb{E}_\pi \left[R_{t+1} + \gamma V(S_{t+1})\right]$$



Sarsa: On-policy TD Control



- Update rule:

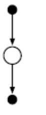
$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$$

  - This rule uses every element of the quintuple of events $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$

Sarsa (on-policy TD control) for estimating $Q \approx q_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(\mathit{terminal}, \cdot) = 0$

Loop for each episode:
  Initialize S
  Choose A from S using policy derived from Q (e.g., ε-greedy)
  Loop for each step of episode:
    Take action A, observe R, S'
    Choose A' from S' using policy derived from Q (e.g., ε-greedy)
    $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma Q(S', A') - Q(S, A)]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until S is terminal



Sarsa

- Converges to optimal policy and action-value function if all state-action pairs are *visited infinitely* and *policy converges to greedy* (*e.g.* using ε-greedy with $\varepsilon = 1/t$)

Q-learning: Off-policy TD Control

- Update rule:

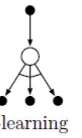
$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\right]$$

  - The target policy π is greedy w.r.t. $Q(S, A)$

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(\mathit{terminal}, \cdot) = 0$

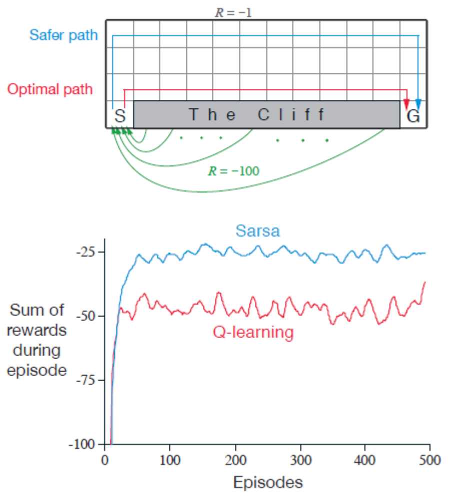
Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \max_a Q(S', a) - Q(S, A)]$
    $S \leftarrow S'$
  until S is terminal



Q-learning

- The learned action-value function Q directly approximates q_*, independently of the policy being followed

Example: Cliff Walking



- Sarsa and Q-learning with ε-greedy action selection ($\varepsilon = 0.1$)
  - Q-learning learns values for the optimal policy
  - Sarsa takes action selection into account and learns the longer but safer path
  - Given exploration, Q-learning occasionally falls off the cliff, hence the lower online performance
- If ε is gradually reduced, both methods converge to the optimal policy

Reinforcement Learning

POLICY GRADIENT METHODS

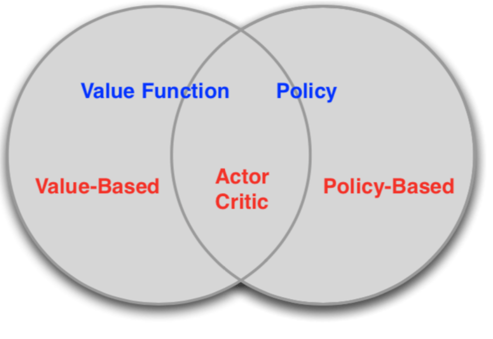
Policy Gradient Methods

- Action-value methods select actions based on action value estimates
  - Using approximation: $\hat{q}(s, a, \mathbf{w}) \approx q_\pi(s, a)$
  - A policy is generated directly from the value function, e.g., using ε-greedy
- Policy gradient methods learn a parameterized policy directly (without consulting a value function)
  - ➢ Search directly in the policy space (an optimization problem)
  - Action selection:

$$\pi(a|s, \theta) = \Pr\{A_t = a | S_t = s, \theta_t = \theta\}$$

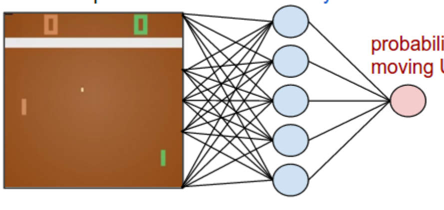
Value-Based and Policy-Based RL

- Value Based
  - Learned Value Function
  - Implicit policy (e.g., ε-greedy)
- Policy Based
  - No Value Function
  - Learned Policy
- Actor-Critic
  - Learned Value Function
  - Learned Policy



Policy Gradient Methods

- Learn the policy parameters based on the gradient of some scalar performance measure $J(\theta)$
  - Seek to maximize performance: approximate gradient ascent in J:
$$\theta_{t+1} = \theta_t + \alpha\widehat{\nabla J(\theta_t)}$$
  - $\widehat{\nabla J(\theta_t)}$ is the policy gradient
  - θ can be the connection weights in a deep neural network



Policy Approximation

- Learning parameterized numerical preferences $h(s, a, \theta)$ for each state-action pair
  - Actions with highest preferences get higher probabilities of being selected (*softmax*)
$$\pi(a|s,\theta) = \frac{e^{h(s,a,\theta)}}{\sum_b e^{h(s,b,\theta)}}$$
- Advantages
  - Can learn stochastic policies with arbitrary probabilities
    - Rock-paper-scissors example: a uniform random policy is optimal (Nash equilibrium)
    - Action preferences are different from action-values: they are driven to produce the optimal stochastic policy
  - Better convergence properties
  - Effective in high-dimensional or continuous action spaces
  - The policy may be a simpler function to approximate, compared to action-values
  - Policy parameterization may be a good way of injecting prior knowledge about the desired form of the policy

U.PORTO
FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

REINFORCE: Monte Carlo Policy Gradient

- A classical algorithm whose update at time t involves just the action A_t taken
- REINFORCE is a Monte Carlo algorithm
  - Uses the complete return G_t from time t, including all future rewards until the end of the episode

$$\theta_{t+1} = \theta_t + \alpha G_t \frac{\nabla\pi(A_t|S_t,\theta_t)}{\pi(A_t|S_t,\theta_t)}$$

REINFORCE: Monte-Carlo Policy-Gradient Control (episodic) for π_*

Input: a differentiable policy parameterization $\pi(a|s,\boldsymbol{\theta})$
Algorithm parameter: step size $\alpha > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ (e.g., to $\mathbf{0}$)

Loop forever (for each episode):
  Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot,\boldsymbol{\theta})$
  Loop for each step of the episode $t = 0, 1, \ldots, T-1$:
    $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$ $\qquad (G_t)$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\gamma^t G \nabla \ln \pi(A_t|S_t,\boldsymbol{\theta})$

  - The aim is to maximize the expected cumulative reward by adjusting the policy parameters.

U.PORTO
FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

Proximal Policy Optimization

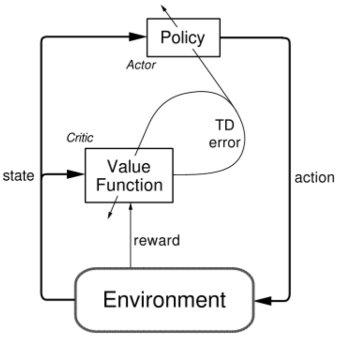
- Proximal Policy Optimization (PPO) works by iteratively improving its policy.

- Like REINFORCE, PPO updates its policy:
  - trying to increase the probability of actions that have higher than average advantage
  - trying decrease the probability of actions that have lower than average advantage.

- However, to prevent the policy from changing too much, PPO adds a penalty term to the objective function, limiting the change made to the policy (stability)

Actor-Critic Methods

- Monte Carlo policy gradient has high variance and tends to learn slowly
- Actor–critic methods: learn approximations to both policy and value functions
- 'Actor' is a reference to the learned policy: updates policy parameters θ
  - Decides which action to take
  - Adjusts a policy based on information (TD error) it receives from the critic
- 'Critic' refers to the learned value function
  - Helps on reducing variance and accelerate learning
  - Tells the actor how good its action was (based on reward signal and the current change in its estimate of state values)



$$\theta_{t+1} = \theta_t + \alpha\big(R_{t+1} + \gamma\hat{v}(S_{t+1},\mathbf{w}) - \hat{v}(S_t,\mathbf{w})\big)\frac{\nabla\pi(A_t|S_t,\theta_t)}{\pi(A_t|S_t,\theta_t)}$$

Reinforcement Learning

ALGORITHMS, RL IN GAMES, ENVIRONMENTS

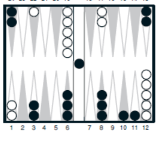
RL Algorithms

Algorithm	Description	Policy	Action Space	State Space
Monte Carlo	Every visit to Monte Carlo	Either	Discrete	Discrete
Q-learning	State–action–reward–state	Off-policy	Discrete	Discrete
SARSA	State–action–reward–state–action	On-policy	Discrete	Discrete
Q-learning - Lambda	Q-learning with eligibility traces	Off-policy	Discrete	Discrete
SARSA - Lambda	SARSA with eligibility traces	On-policy	Discrete	Discrete
DQN [Mnih et al., 2013]	Deep Q Network	Off-policy	Discrete	Continuous
DDPG [Lillicrap et al., 2016]	Deep Deterministic Policy Gradient	Off-policy	Continuous	Continuous
A3C [Mnih et al., 2016]	Asynchronous Advantage Actor-Critic	On-policy	Continuous	Continuous
NAF [Gu et al., 2016]	Q-Learning with Normalized Advantage Functions	Off-policy	Continuous	Continuous
TRPO [Schulman et al., 2015]	Trust Region Policy Optimization	On-policy	Continuous	Continuous
PPO [Schulman et al., 2017]	Proximal Policy Optimization	On-policy	Continuous	Continuous
TD3 [Fujimoto et al., 2018]	Twin Delayed Deep Deterministic Policy Gradient	Off-policy	Continuous	Continuous
SAC [Haarnoja et al., 2018]	Soft Actor-Critic	Off-policy	Continuous	Continuous

65

RL in Games

- TD-Gammon [Tesauro, 1995]
  - Neural Network trained with self-play reinforcement learning

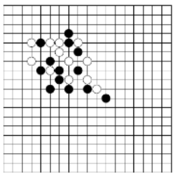


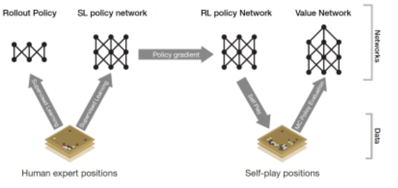
- Atari 2600 Games [DeepMind, 2013]
  - Learn control policies directly from high-dimensional sensory input using reinforcement learning
  - Input is raw pixels and output is a value function estimating future rewards

66

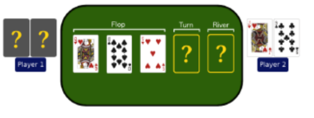
RL in Games

- AlphaGo [Google DeepMind, 2016]
  - Convolutional Neural Networks trained with human expert data
  - Deep Reinforcement Learning with fictitious self-play



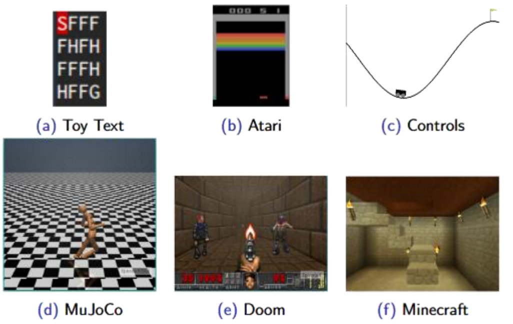


- Poker: Heads-Up Limit Texas Hold'em – NFSP [UCL, 2016]
  - Deep Reinforcement Learning with fictitious self-play
  - No prior knowledge



OpenAI Gym

- OpenAI Gym is a toolkit for developing and comparing RL algorithms
- The gym library is a collection of test problems with a shared interface — environments — that you can use to work out your RL algorithms
  - See also Gymnasium



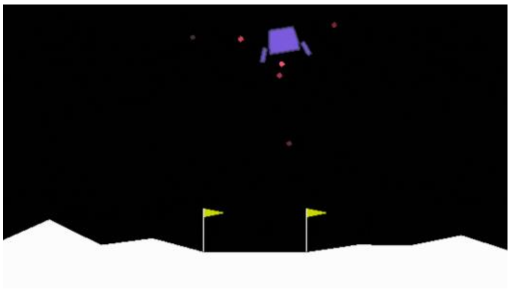
(a) Toy Text (b) Atari (c) Controls

(d) MuJoCo (e) Doom (f) Minecraft

U.PORTO
FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

Stable Baselines

- [OpenAI Baselines] is a set of high-quality implementations of RL algorithms
- [Stable Baselines 3]
  - Stable baselines 3 is for RL what scikit-learn is for ML
  - Tutorial: [Reinforcement Learning in Python with Stable Baselines 3]

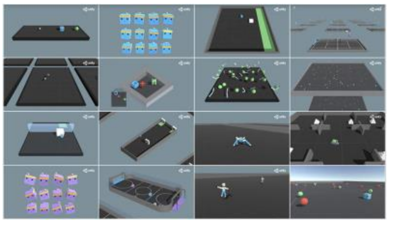


69

U.PORTO
FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

Unity ML-Agents

- With Unity Machine Learning Agents (ML-Agents), you teach intelligent agents through a combination of deep reinforcement learning and imitation learning



70

Conclusions

- RL enables to learn intelligent behavior in complex environments
- Large number of algorithms and approaches
- Amazing results in vintage Atari Games, AlphaGo and AlphaZero
- Very fast evolution in the last few years
- Impact in diverse areas, from games to robotics to language models (e.g., ChatGPT)

71

Further Reading

- Sutton, R. S. and Barto, A. G. (2018). *Reinforcement Learning – An Introduction*, 2nd ed., The MIT Press: Chap. 1-3, 6, 13
- UCL Course on RL (David Silver)
- Tutorial Videos for Deep RL:
  - A friendly introduction to deep reinforcement learning, Q-networks and policy gradients
  - An introduction to Reinforcement Learning
  - Policy Gradient methods and Proximal Policy Optimization (PPO)

72