Università degli
Studi di Pavia

Deep Learning

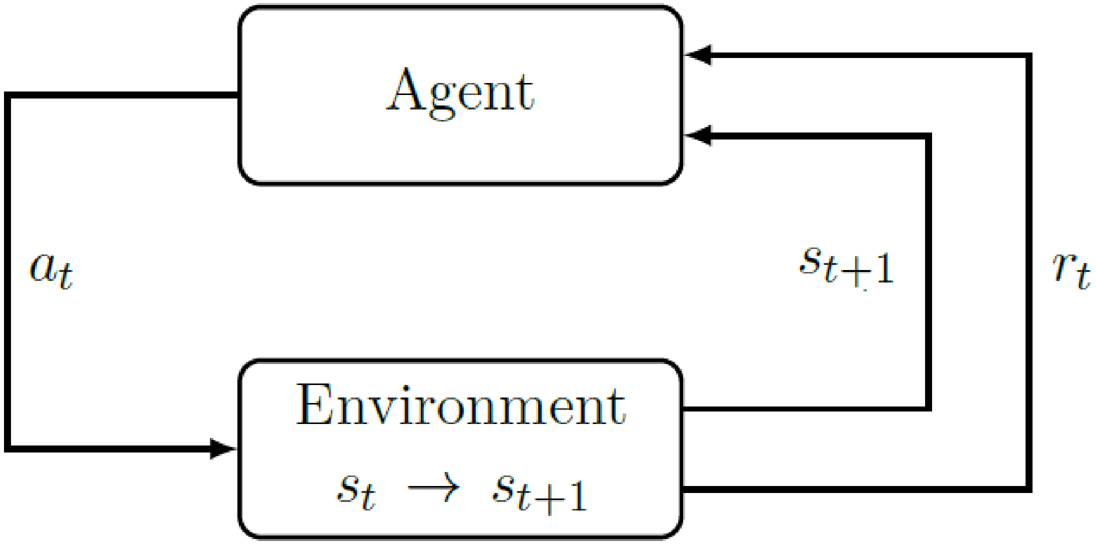
10 –Reinforcement Learning

Marco Piastra & Andrea Pedrini(*)

(*) Dipartimento di Matematica F. Casorati

Basic assumptions

[image from: https://arxiv.org/pdf/1811.12560.pdf]



The **Environment**: is in *state* s_t *time*

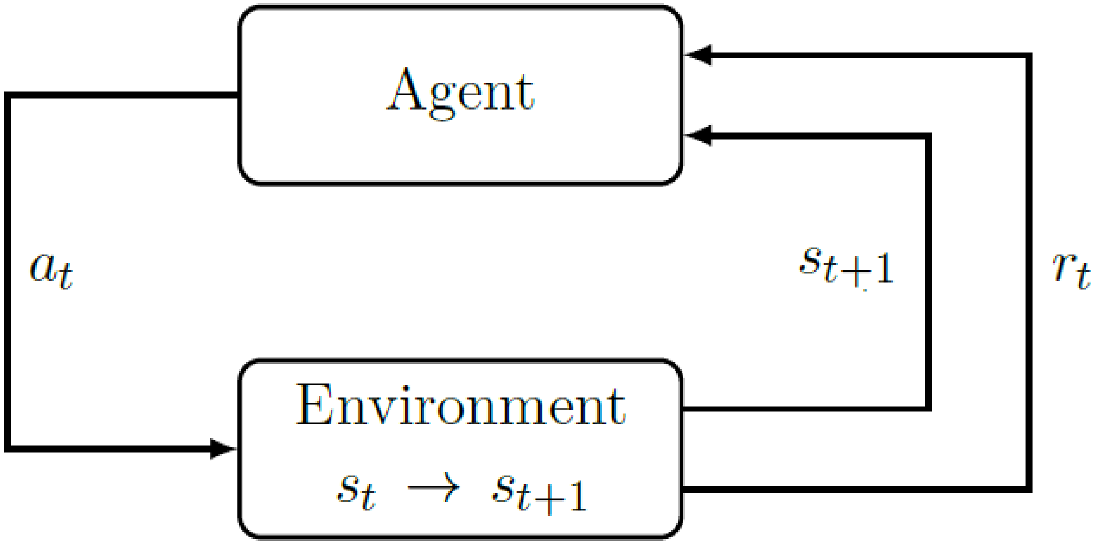
An **Agent** observes *state* s_t and performs *action* a_t

The **Environment** *state* transitions from $s_t \rightarrow s_{t+1}$

The **Agent** receives *reward* r_t

Basic assumptions

[image from: https://arxiv.org/pdf/1811.12560.pdf]



The **Environment**: is in *state* s_t *time*

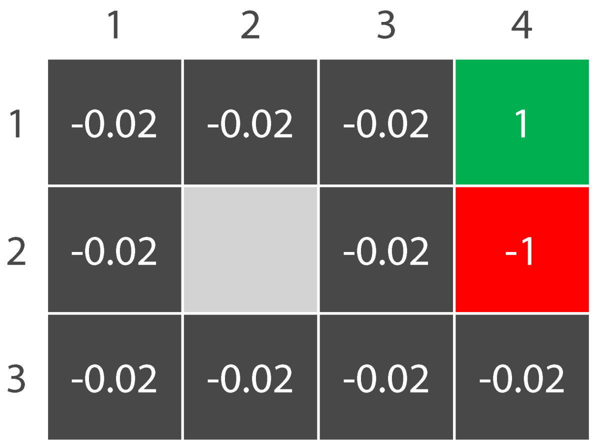
An **Agent** observes *state* s_t and performs *action* a_t

The **Environment** *state* transitions from $s_t \to s_{t+1}$

The **Agent** receives *reward* r_t

Cumulative reward: $$R := \sum_{t=0}^{\infty} r_t$$

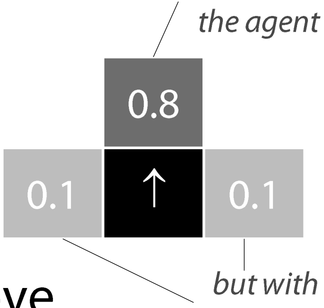
An example: *gridworld*



	1	2	3	4
1	-0.02	-0.02	-0.02	1
2	-0.02		-0.02	-1
3	-0.02	-0.02	-0.02	-0.02

The *state* of the agent is the position on the grid: e.g. (1,1), (3,4), (2,3)

At each time step, the agent can *move* one box in the directions ←↑↓→



with probability 0.8 the agent will end up here

but with probability 0.2 it might end up here

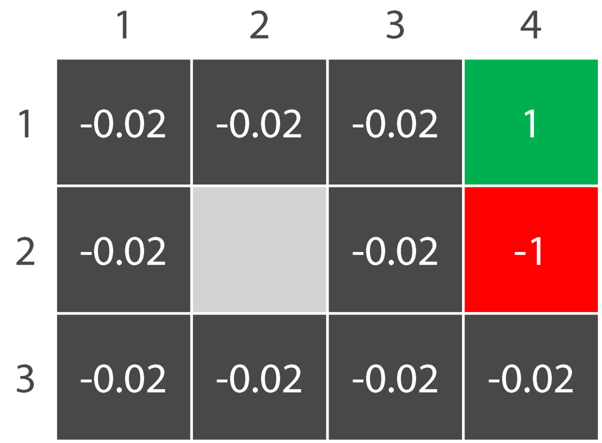
The effect of each move is somewhat stochastic, however: for example, a move ↑ has a slight probability of producing a different (*and perhaps unwanted*) effect

Entering each state yields the *reward* shown in each box above

There are two *absorbing states*: entering either the green or the red box means exiting the *gridworld* and completing the game

- What is the best (*i.e. maximally rewarding*) movement policy?

Markov Decision Process (MDP)



	1	2	3	4
1	-0.02	-0.02	-0.02	1
2	-0.02		-0.02	-1
3	-0.02	-0.02	-0.02	-0.02

Formalization and abstraction of the gridworld example

Markov Decision Process: $< \mathcal{S}, \mathcal{A}, r, P, \gamma >$

A set of *states* : $\mathcal{S} = \{s_1, s_2, \dots\}$

A set of *actions* : $\mathcal{A} = \{a_1, a_2, \dots\}$

A *reward function* : $r : \mathcal{S} \rightarrow \mathbb{R}$

A *transition probability distribution* : $P(S_{t+1} \mid S_t, A_t)$ (also called a *model*)

Markov property: the transition probability depends only on the previous state and action

$$P(S_{t+1} \mid S_t, A_t) = P(S_{t+1} \mid S_t, A_t, S_{t-1}, A_{t-1}, S_{t-2}, A_{t-2}, \dots)$$

A *discount factor* : $0 \leq \gamma < 1$

Markov Decision Process (MDP): policies and values

The agent is supposed to adopt a *deterministic* policy : $\pi : \mathcal{S} \to \mathcal{A}$

In other words, the agent always chooses its *action* depending on the *state* alone

Given a policy π , the ***state value function*** is defined, for each state s as:

$$V^{\pi}(s) := \mathbb{E}[r(S_t) + \gamma r(S_{t+1}) + \gamma^2 r(S_{t+2}) + \ldots \mid \pi, S_t = s]$$

Note the role of the *discount factor*: a value $\gamma < 1$ means that that future rewards could be weighted less (by the agent) than immediate ones

Note also that all states S_t must be described by *random variables* :
i.e. the policy is deterministic but the state transition is not

Note also that when the reward is *bounded*, i.e. $r(S) \leq r_{\max}$

$$\sum_{t=0}^{\infty} \gamma^t \, r(S_t) \;\leq\; r_{\max} \sum_{t=0}^{\infty} \gamma^t \;=\; r_{\max} \, \frac{1}{1-\gamma}$$

for $\gamma < 1$ this is the *geometric series*

Markov Decision Process (MDP): policies and values

The agent is supposed to adopt a *deterministic* policy : $\pi : \mathcal{S} \to \mathcal{A}$

In other words, the agent always chooses its *action* depending on the *state* alone

Given a policy π, the ***state value function*** is defined, for each state s as:

$$V^{\pi}(s) := \mathbb{E}[r(S_t) + \gamma r(S_{t+1}) + \gamma^2 r(S_{t+2}) + \ldots \mid \pi, S_t = s]$$

Note the role of the *discount factor*: a value $\gamma < 1$ means that that future rewards could be weighted less (by the agent) than immediate ones

Note also that all states S_t must be described by *random variables* :
i.e. the policy is deterministic but the state transition is not

In the *gridworld* example:

- The set of states is finite
- The set of actions is finite
- For every policy, each entire story is finite

  Sooner or later the agent will fall into one of the absorbing states

Bellman equations

By working on the definition of value function:

$$\begin{aligned} V^{\pi}(s) &:= \mathbb{E}[r(S_t) + \gamma r(S_{t+1}) + \gamma^2 r(S_{t+2}) + \ldots \mid \pi, S_t = s] \\ &= \mathbb{E}[r(S_t) + \gamma(r(S_{t+1}) + \gamma r(S_{t+2}) + \ldots) \mid \pi, S_t = s] \\ &= r(s) + \gamma \mathbb{E}[r(S_{t+1}) + \gamma r(S_{t+2}) + \ldots \mid \pi, S_t = s] \\ &= r(s) + \gamma \textstyle\sum_{s'} P(s' \mid s, \pi(s)) \cdot \mathbb{E}[r(S_{t+1}) + \gamma r(S_{t+2}) + \ldots \mid \pi, S_{t+1} = s'] \\ &= r(s) + \gamma \textstyle\sum_{S_{t+1}} P(S_{t+1} \mid s, \pi(s)) \cdot V^{\pi}(S_{t+1}) \end{aligned}$$

This means that in a Markov Decision Process:

$$V^{\pi}(s) = r(s) + \gamma \textstyle\sum_{S_{t+1}} P(S_{t+1} \mid s, \pi(s)) \cdot V^{\pi}(S_{t+1})$$

This is true for any *state*, so there is one such equation for each of those

If the set of states is finite, there are exactly $|S|$ *(linear) Bellman equations for* $|S|$ *variables:*
in general, for any deterministic policy , V^{π} *can be computed analytically*

Optimal policy – Optimal value function

- Basic definitions

$$V^*(s) := \max_\pi V^\pi(s),\ \forall s \in S$$
$$\pi^*(s) := \mathrm{argmax}_\pi V^\pi(s),\ \forall s \in S$$

Property: for every MDP, there exists such an optimal deterministic policy (*possibly non-unique*)

With Bellman Equations:

$$\begin{aligned}\max_\pi V^\pi(s) &= r(s) + \gamma \max_\pi \left(\sum_{S_{t+1}} P(S_{t+1} \mid s, \pi(s)) \cdot V^\pi(S_{t+1})\right)\\ V^*(s) &= r(s) + \gamma \max_\pi \left(\sum_{S_{t+1}} P(S_{t+1} \mid s, \pi(s)) \cdot V^*(S_{t+1})\right)\\ &= r(s) + \gamma \max_a \left(\sum_{S_{t+1}} P(S_{t+1} \mid s, a) \cdot V^*(S_{t+1})\right)\end{aligned}$$

Therefore:

$$\pi^*(s) = \mathrm{argmax}_a \left(\sum_{S_{t+1}} P(S_{t+1} \mid s, a) V^*(S_{t+1})\right)$$

Computing V^* *directly from these equations is unfeasible, however*

There are in fact $|A|^{|S|}$ *possible strategies*

However, once V^* *has been determined,* π^* *can be determined as well*

Optimal value function: value iteration

- Value iteration algorithm

Initialize: $V(s) := r(s), \ \forall s \in S$
Repeat:

Note that there is no policy:
all actions must be explored

1) For every state, update: $V(s) := r(s) + \gamma \max_a \sum_{s'} P(s' \mid s, a) V(s')$

Theorem: for every fair way (*i.e. giving an equal chance*) of visiting the states in S, this algorithm converges to V^*

Value iteration and optimal policy

	1	2	3	4
1	-0.02	-0.02	-0.02	1
2	-0.02		-0.02	-1
3	-0.02	-0.02	-0.02	-0.02

Initialize states
(e.g. using rewards as initial values)

Iterate and compute V^*

	1	2	3	4
1	0.86	0.90	0.93	1
2	0.82		0.69	-1
3	0.78	0.75	0.71	0.49

V^*

Define the optimal policy as:

$$\pi^*(s) := \mathrm{argmax}_a\left(\sum_{S_{t+1}} P(S_{t+1} \mid s, a) \cdot V^*(S_{t+1})\right)$$

	1	2	3	4
1	→	→	→	1
2	↑		↑	-1
3	↑	←	←	←

π^*

Optimal policy: policy iteration

- Policy iteration algorithm

Initialize $\pi(s), \forall s \in S$ at random
Repeat:

This step is computationally expensive:
either solve the equations or use value iteration
(with fixed policy π)

1) For each state, compute: $V(s) := V^{\pi}(s)$
2) For each state, define: $\pi(s) := \mathrm{argmax}_a \sum_{s'} P(s' \mid s, a) V(s')$

Theorem: for every fair way (*i.e. giving an equal chance*) of visiting the states in S, this algorithm converges to π^*

As with the value iteration algorithm, this algorithm uses partial estimates to compute new estimates.
It is also greedy, in the sense that it exploits its current estimate $V^{\pi}(s)$

Policy iteration converges with very few number of iterations,
but every iteration takes much longer time than that of *value iteration*

The tradeoff with value iteration is the action space:
when action space is large and state space is small, policy iteration could be better

Offline vs. *Online* learning

- *Value iteration* and *policy iteration* are offline algorithms

  The *model* , i.e. the Markov Decision Process is known

  What needs to be learn is the optimal policy π^*

  In the algorithms, *visiting states* just means considering: there is no agent actually playing the game.

- Different conditions: *learning by doing* …

  Suppose the *model* (i.e. the MDP) is NOT known, or perhaps known only in part

  Then the agent must learn by doing…

Action value function

An analogous of the value function V^π

Given a policy π, the ***action value function*** is defined, for each pair (s, a) as:

$$Q^\pi(s,a) := \sum_{S_{t+1}} P(S_{t+1} \mid s,a) \cdot V^\pi(S_{t+1})$$

$$= \sum_{S_{t+1}} P(S_{t+1} \mid s,a) \cdot \mathbb{E}[r(S_{t+1}) + \gamma r(S_{t+2}) + \ldots \mid \pi, S_{t+1}]$$

$$= \sum_{S_{t+1}} P(S_{t+1} \mid s,a) \cdot [r(S_{t+1}) + \mathbb{E}[\gamma r(S_{t+2}) + \ldots \mid \pi, S_{t+1}]]$$

$$= \sum_{S_{t+1}} P(S_{t+1} \mid s,a) \cdot [r(S_{t+1}) + \gamma Q^\pi(S_{t+1}, \pi(S_{t+1}))]$$

In other words, $Q^\pi(s,a)$ is the expected value of the reward in S_{t+1}
by taking action a in state s and then following policy π from that point on

Following a similar line of reasoning, the ***optimal*** *action value function* is

$$Q^*(s,a) = \sum_{S_{t+1}} P(S_{t+1} \mid s,a) \cdot [r(S_{t+1}) + \gamma \max_{a'} Q^*(S_{t+1}, a')]$$

Q-Learning

- Q-learning algorithm (ε-*greedy version*)

```
Initialize $\hat{Q}(s,a)$ at random, put the agent is in a random state $s$
Repeat:
    1) Select the action $\mathrm{argmax}_a \hat{Q}(s,a)$ with probability $(1-\varepsilon)$
       otherwise, select $a$ at random
    2) The agent is now in state $s'$ and has received the reward $r$
    3) Update $\hat{Q}(s,a)$ by
           $\Delta\hat{Q}(s,a) = \alpha[r + \gamma \max_{a'} \hat{Q}(s',a') - \hat{Q}(s,a)]$
```

Exponential Moving Average
(see later …)

$\mathcal{A}$

Q-Learning

- Q-learning algorithm

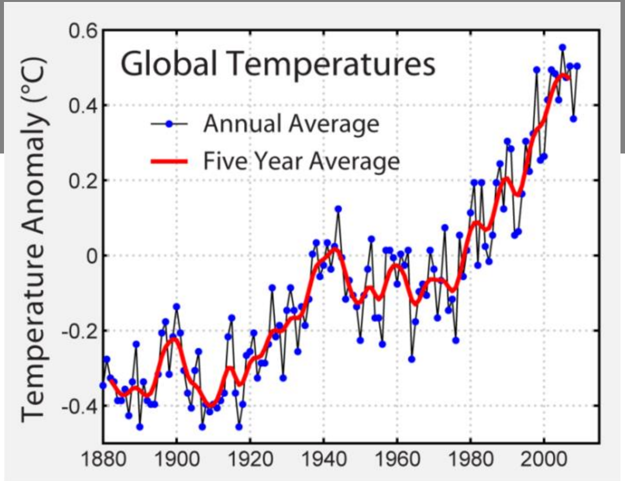
Theorem (Watkins, 1989): in the limit of that each action is played infinitely often and each state is visited infinitely often and $\alpha \rightarrow 0$ as experience progresses, then

$$\hat{Q}(s,a) \rightarrow Q^*(s,a)$$

with probability 1

The Q-learning algorithm bypasses the MDP entirely,
in the sense that the optimal strategy is learnt without learning the model $P(S_{t+1} \mid S_t, A_t)$

An aside: *moving averages*



[image from wikipedia]

Following non-stationary phenomena

- Average

  Definition: $$\overline{v}_T := \frac{1}{T}\sum_{k=1}^{T} v_k$$

  Running implementation:

$$\overline{v}_T = \frac{1}{T}(v_T + \sum_{k=1}^{T-1} v_k) = \frac{1}{T}(v_T + (T-1)\overline{v}_{T-1})$$

$$= \overline{v}_{T-1} + \frac{1}{T}(v_T - \overline{v}_{T-1}) = \frac{1}{T}\, v_T + (1 - \frac{1}{T})\,\overline{v}_{T-1}$$

"the weight of newer observations diminishes with time"

- Simple Moving Average (SMA)

$$\overline{v}_{T,n} := \frac{1}{n}\sum_{k=T-n}^{T} v_k$$

- Exponential Moving Average (EMA)

$$\overline{v}_{T,\alpha} := \alpha\, v_T + (1-\alpha)\,\overline{v}_{T-1,\alpha},\ \ \alpha \in [0,1]$$

"the weight of newer observations remains constant"

An aside: *moving averages*

- Exponential Moving Average (EMA)

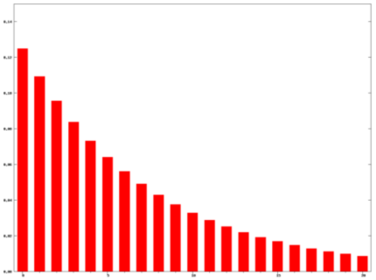
$$\overline{v}_{T,\alpha} := \alpha\, v_T + (1-\alpha)\,\overline{v}_{T-1,\alpha},\ \ \alpha \in [0,1]$$

Expanding:

$$\begin{aligned}\overline{v}_{t,\alpha} &= \alpha\, v_t + (1-\alpha)\,\overline{v}_{t-1,\alpha}\\ &= \alpha\, v_t + (1-\alpha)(\alpha\, v_{t-1} + (1-\alpha)\overline{v}_{t-2,\alpha})\\ &= \alpha\, v_t + (1-\alpha)(\alpha\, v_{t-1} + (1-\alpha)(\alpha\, v_{t-2} + (1-\alpha)\overline{v}_{t-3,\alpha}))\\ &= \alpha\,(v_t + (1-\alpha)\, v_{t-1} + (1-\alpha)^2\, v_{t-2}) + (1-\alpha)^3\,\overline{v}_{t-3,\alpha}\end{aligned}$$

$$(1-\alpha)^{\Delta_t}$$

"the weight of older observations diminishes with time"



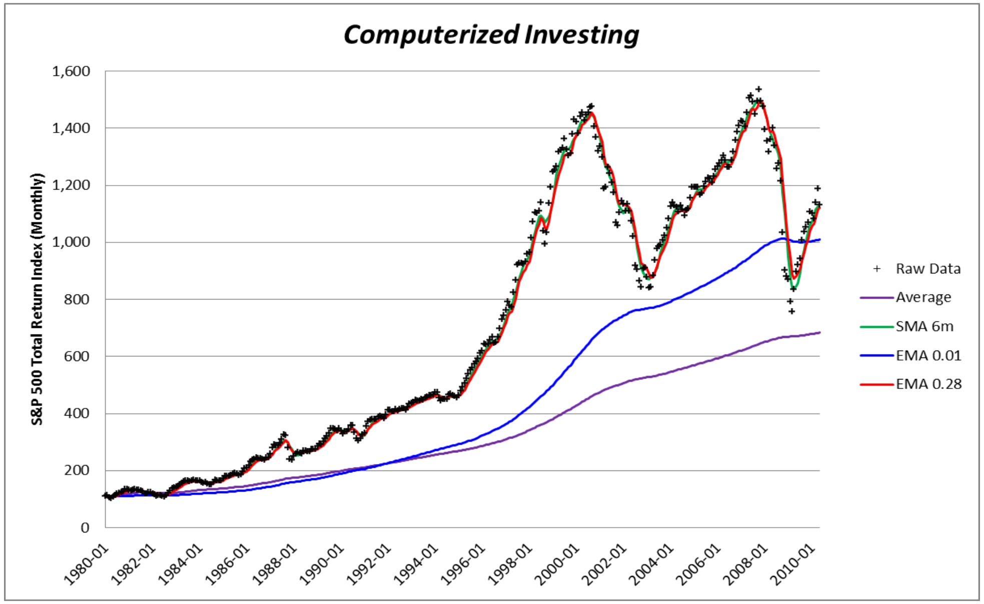
[image from wikipedia]

The weight of past contributions decays as

$$(1-\alpha)^{\Delta_t}$$

A SMA with n previous values is approximately equal to an EMA with

$$\alpha = \frac{2}{n+1}$$



Q-Learning revisited

- Q-learning algorithm (*ε-greedy version*)

off-policy

Initialize $\hat{Q}(s,a)$ at random, put the agent is in a random state s
Repeat:

1) Select the action $a = \mathrm{argmax}_a \hat{Q}(s,a)$ with probability $(1-\varepsilon)$
otherwise, select a at random
2) The agent is now in state s' and has received the reward r
3) Update $\hat{Q}(s,a)$ by

$$\Delta\hat{Q}(s,a) = \alpha[r + \gamma \max_{a'} \hat{Q}(s',a') - \hat{Q}(s,a)]$$

By rewriting step 3)

$$\hat{Q}(s,a) = \hat{Q}(s,a) + \Delta\hat{Q}(s,a) = \hat{Q}(s,a) + \alpha[r + \gamma \max_{a'} \hat{Q}(s',a') - \hat{Q}(s,a)]$$
$$= \underbrace{\alpha[r + \gamma \max_{a'} \hat{Q}(s',a')] + (1-\alpha)\hat{Q}(s,a)}_{\text{Exponential Moving Average}}$$

compare with (see before):

$$Q^*(s,a) = \underbrace{\sum_{S_{t+1}} P(S_{t+1} \mid s,a) \cdot [r(S_{t+1}) + \gamma \max_{a'} Q^*(S_{t+1},a')]}_{\text{Expectation}}$$

SARSA

- SARSA algorithm (ε-*greedy version*)

on-policy

Initialize $\hat{Q}(s,a)$ at random, put the agent is in a random state s
Repeat:

1) Select the action $a = \text{argmax}_a \hat{Q}(s,a)$ with probability $(1-\varepsilon)$
otherwise, select a at random
2) The agent is now in state s' and has received the reward r
3) Select the action $a' = \text{argmax}_a \hat{Q}(s',a)$ with probability $(1-\varepsilon)$
otherwise, select a' at random
4) Update $\hat{Q}(s,a)$ by

$$\Delta\hat{Q}(s,a) = \alpha[r + \gamma\hat{Q}(s',a') - \hat{Q}(s,a)]$$

No more 'max' here

Q-learning is a an *off-policy* algorithm: each update involves $\max_{a'} \hat{Q}(s',a')$
(i.e. *exploration* is not taken into account)
SARSA is a an *on-policy* algorithm: each update involves $\hat{Q}(s',a')$
(which involves the next policy action, *exploration* included)

SARSA vs Q-Learning

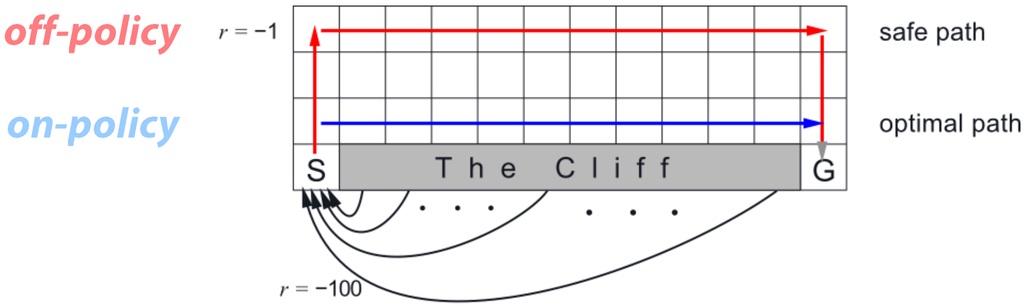
- Cliff World

  'S' is the start

  'G' is the goal

  Each white box has $r = -1$

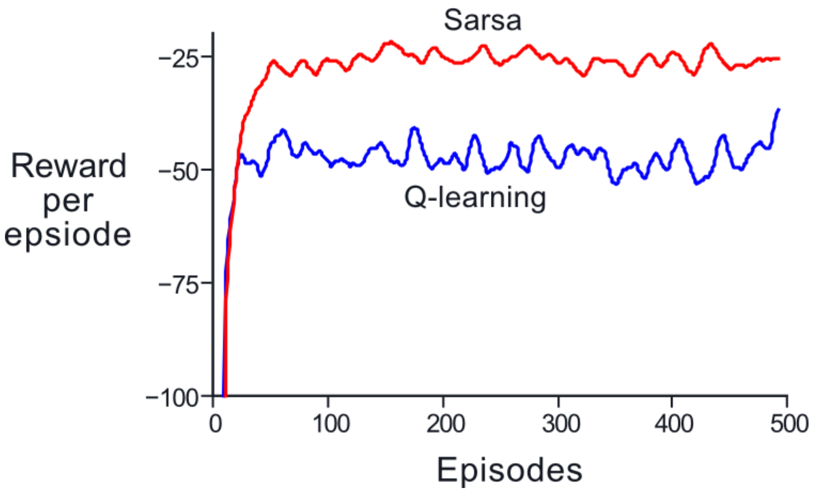
  'The Cliff' region has $r = -100$ and entails going back to 'S'



- Experimental Results

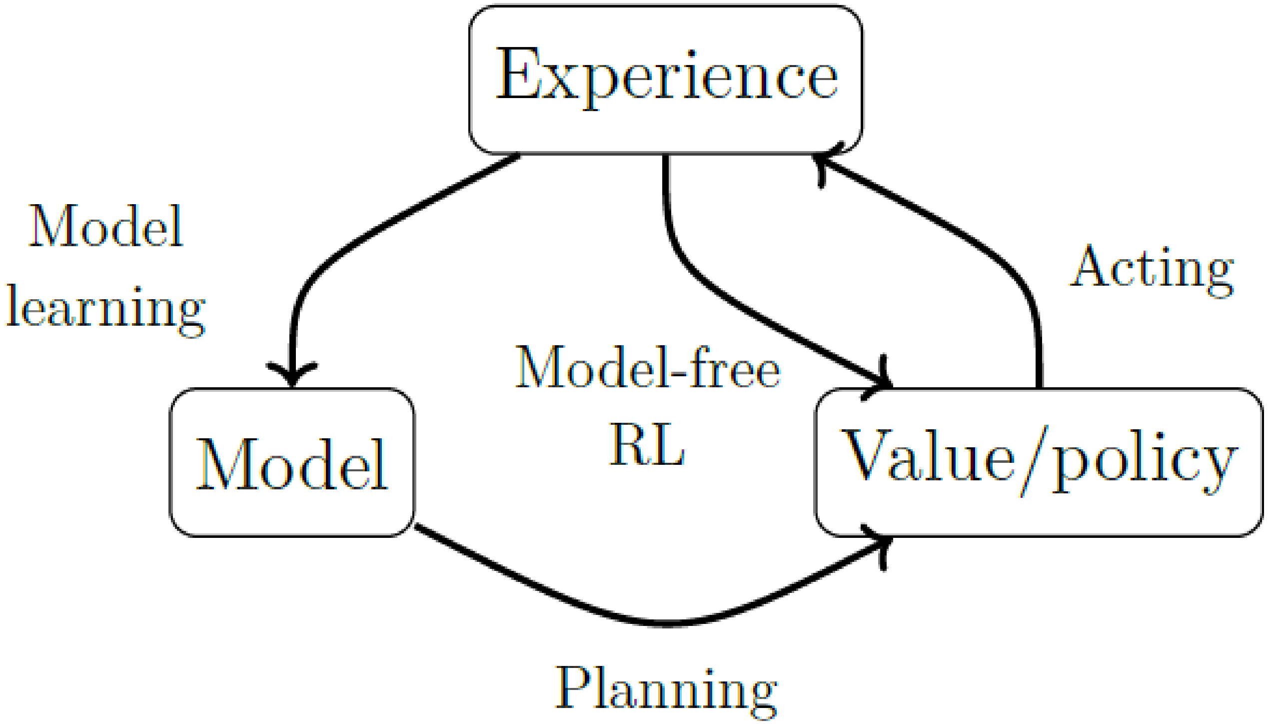
  SARSA finds a sub-optimal but safer path since its learning takes into account the ε risk of going off the cliff

  Q-learning finds the optimal path but, occasionally, it falls off the cliff during learning due to the ε-greedy strategy



Reinforcement Learning Methods

[image from: https://arxiv.org/pdf/1811.12560.pdf]



Experience

Model learning

Model-free RL

Acting

Model

Value/policy

Planning

<u>Deep</u> Reinforcement Learning

Game Playing with DRL

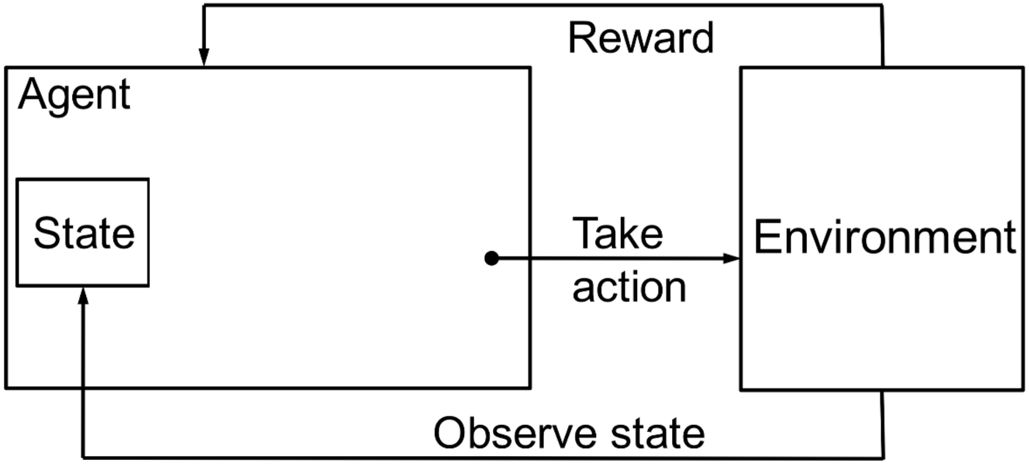
- Playing Atari with Deep Reinforcement Learning
  [2013, V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, M. Riedmiller, http://arxiv.org/abs/1312.5602, see also http://www.nature.com/nature/journal/v518/n7540/full/nature14236.html]

  - A software system only
    - Runs on virtually any Linux-based system, it contains optional provisions for GPU
  - *It's open source*
    - https://github.com/kuz/DeepMind-Atari-Deep-Q-Learner
  - Sophisticated machine-learning techniques
    - Uses deep reinforcement learning
    - in combination with convolutional neural networks (CNN)
  - Same configuration, multiple games
    - Same configuration applied to arcade games
    - It learned to play 7 (2013) or 49 (2015) different games
  - It is *autonomous*
    - *It learns by itself*, it receives no human expertise as input
    - In many cases, it outperforms human players

(from GitHub)

Deep Reinforcement Learning (DRL)

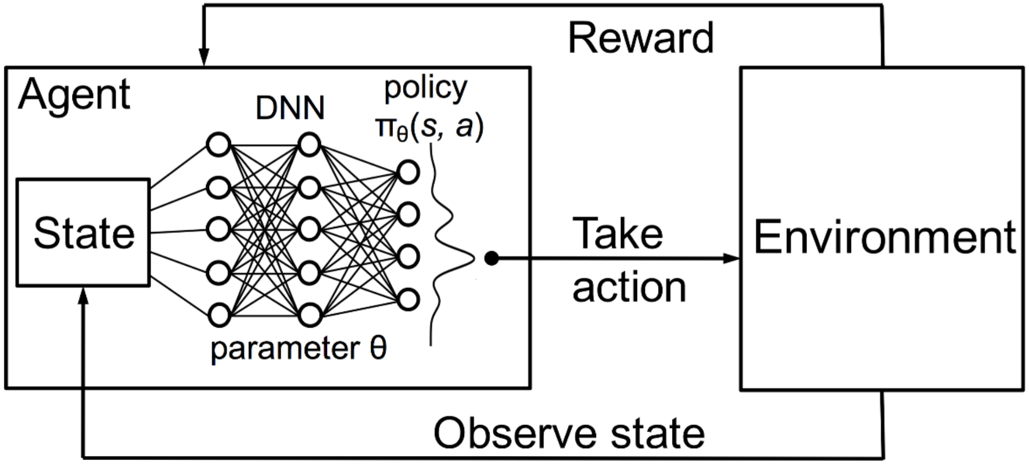
- ***Reinforcement Learning***



Deep Reinforcement Learning (DRL)

- ***Deep Reinforcement Learning***

  Using a deep neural network as the approximator $\hat{Q}(s, a)$



The optimal policy is learnt incrementally by using a deep neural network

Deep Reinforcement Learning

- ***Q-Learning Algorithm***

Initialize $\hat{Q}(s,a)$ at random, put the agent in a random state s
Repeat:

1) Select the action $\mathrm{argmax}_a \hat{Q}(s,a)$ with probability $(1-\varepsilon)$
   otherwise, select a at random
2) The agent is now in state s' and has received the reward r
3) Update $\hat{Q}(s,a)$ by

$$\Delta\hat{Q}(s,a) = \alpha[r + \gamma \max_{a'} \hat{Q}(s',a') - \hat{Q}(s,a)]$$

Fundamental Idea:

use a deep neural network to learn the approximator $\hat{Q}(s,a)$
from the examples collected while ***exploring*** *-* ***exploiting***