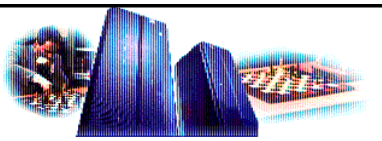
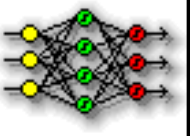


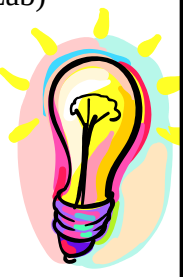


Sistemi Intelligenti
Reinforcement Learning:
Eligibility traces - Examples

Alberto Borghese

Università degli Studi di Milano
Laboratorio di Sistemi Intelligenti Applicati (AIS-Lab)
Dipartimento di Informatica
Alberto.borghese@unimi.it

Barto and Sutton, cap. 12.2, 12.7, 12.10

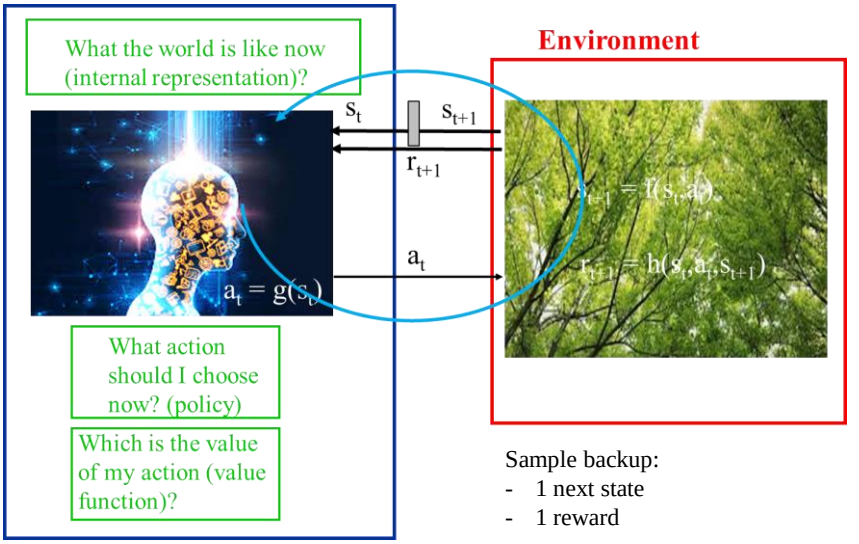


Sommario

Traccia

Fuzzy RL

Schematic diagram of an agent



Sample backup:
- 1 next state
- 1 reward

Formulazione di TD(0)

Correggo la stima corrente valutando l'"errore" **ad un passo.**

$$Q_{k+1}^{\pi}(s_t, a_t) = Q_k^{\pi}(s_t, a_t) + \alpha[r' + \gamma Q_k^{\pi}(s_{t+1}, a_{t+1}) - Q_k^{\pi}(s_t, a_t)]$$

$$\Delta Q_k^{\pi}(s_t, a_t) = \alpha \delta_k(s_t, a_t)$$

$$\delta_k(s_t, a_t) = [r' + \gamma Q_k^{\pi}(s_{t+1}, a_{t+1}) - Q_k^{\pi}(s_t, a_t)] \quad \text{"Errore"}$$

Come estendere l'orizzonte temporale dell'apprendimento?

Pollicino

Marca la strada da casa con dei sassolini

Cosa rappresenta la Eligibility trace

Buffer di memoria: contiene traccia di eventi passati (stati visitati, azioni…).

La traccia evapora nel tempo secondo un parametro λ: TD(0) -> TD(λ)

Definisce se uno stato è eleggibile e "quanto" sia **eleggibile**, cioè quanto aggiornamento meriti.

Quando viene calcolato un errore usando metodi basati su TD, la eligibility trace suggerisce quali variabili aggiornare e quanto aggiornarle (**credit assignement**)

Si suppone che gli stati meritino sempre meno aggiornamento tanto più sono passata.

Amplia l'orizzonte temporale sul quale fare l'aggiornamento a più di 1 passo.

NB L'errore modifica il valore di Q(s,a) ma questo modifica a sua volta il valore di $Q(s_{t-1}, a_{t-1})$ e così via ricorsivamente fino a $Q(s_0, a_0)$.

Funzionamento di base

Definiamo e(s,a) l'eleggibilità di una coppia stato-azione. E' associata alla valutazione del reward a lungo termine, Q(s,a).

Ogni volta che uno stato viene visitato, viene **aumentata** la sua eleggibilità: $e_{k+1}(s,a) = e_k(s,a) + 1$.

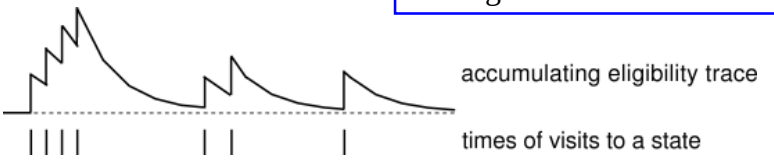
e(s,a) decresce esponenzialmente con il tempo. Ad ogni istante di tempo **l'eleggibilità di tutti gli stati viene decrementata** esponenzialmente secondo una costante $0 < \lambda < 1$: $e_{k+1}(s,a) = \lambda\, e_k(s,a)$.

Eligibility trace per la coppia (s_t-a_t)

$$e_t(s,a) = \begin{cases} \gamma\lambda e_{t-1}(s,a) + 1 & \text{if } s = s_t \text{ and } a = a_t; \\ \gamma\lambda e_{t-1}(s,a) & \text{otherwise.} \end{cases} \quad \text{for all } s, a$$

decay

Increases: depends only on visiting the state

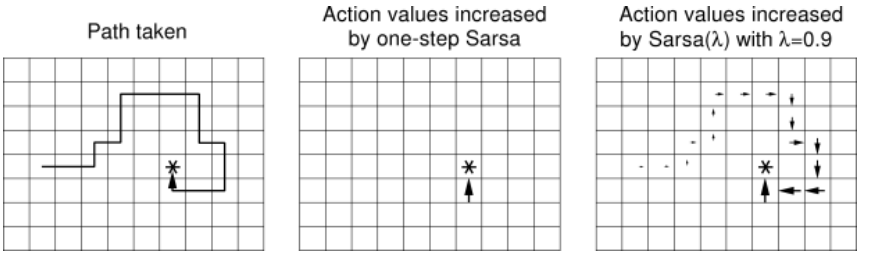


Alternativa:

$$e_t(s,a) = \begin{cases} 1 & \text{if } s = s_t \text{ and } a = a_t; \\ \gamma\lambda e_{t-1}(s,a) & \text{otherwise.} \end{cases} \quad \text{for all } s, a$$

e(s,a) = 0 at start, e(s,a) ≥ 0.

Esempio su Q(s,a)



Con il semplice costo di una variabile per ogni coppia stato-azione, ho un aggiornamento della funzione valore che decresce gradualmente negli stati precedenti.

$Q^{\pi}(s,a)$ inizializzati a un valore leggermente negativo.

r = 0 per ogni stato prossimo, tranne che per lo stato finale, per il quale r = +1.

Osservazioni

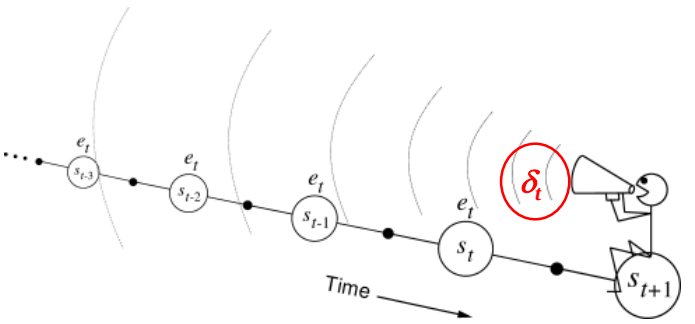


Figure 12.5: The backward or mechanistic view. Each update depends on the current TD error combined with traces of past events.

$$\Delta Q^{\pi}(s,a) = \alpha \delta_t(s_t,a_t)\, e(s,a)$$

$$\delta_t(s_t,a_t) = [r_{t+1} + \gamma Q^{\pi}(s_{t+1},a_{t+1}) - Q^{\pi}(s_t,a_t)]$$ "Errore"

Earlier states are given less credit for the TD(0) error

Come utilizzare la eligibility trace

$$Q^{\pi}_{k+1}(s_t, a_t) = Q^{\pi}_{k}(s_t, a_t) + \alpha[r' + \gamma Q^{\pi}_{k}(s_{t+1}, a_{t+1}) - Q^{\pi}_{k}(s_t, a_t)]$$

Errore: δ_t

TD(λ) Learning:

$$Q^{\pi}_{k+1}(s, a) = Q^{\pi}_{k}(s, a) + \alpha\delta_t(s_t, a_t)e_t(s, a)$$

Per tutte le coppie (s,a)

Eleggibilità: $e_t(s,a)$

Propagate error computed at time t to **all (s,a)**

SARSA(λ) – On policy

Inizialize Q(s,a) = 0; e(s,a) = 0; ∀s, ∀a
Repeat (for each episode):
  s = s_0;
  Repeat for each step of the episode:
    a = Policy(s)
    s_next = NextState(s,a)
    a_next = Policy(s_next) // Greedy or not greedy choice
    reward = Reward(s,a,s_next)
    $\delta(s,a) = [r' + \gamma Q^{\pi}(s\_next, a\_next) - Q^{\pi}(s,a)]$
    $e(s,a) = e(s,a) + 1$
    for all s and all a:
      $Q^{\pi}(s,a) = Q^{\pi}(s,a) + a * \delta$ (s,a) * e(s,a)
      $e(s,a) = \lambda * \gamma * e(s,a)$
    end
    s = s_next;
  until s = terminal state
until end of learning

Commenti

$$Q_{k+1}^{\pi}(s,a) = Q_k^{\pi}(s,a) + \alpha\delta_t(s,a)e_t(s,a)$$

Ruolo di λ.

Ogni volta che uno stato viene visitato, viene aumentata bruscamente la sua eleggibilità: e(s,a) = e(s,a) + 1

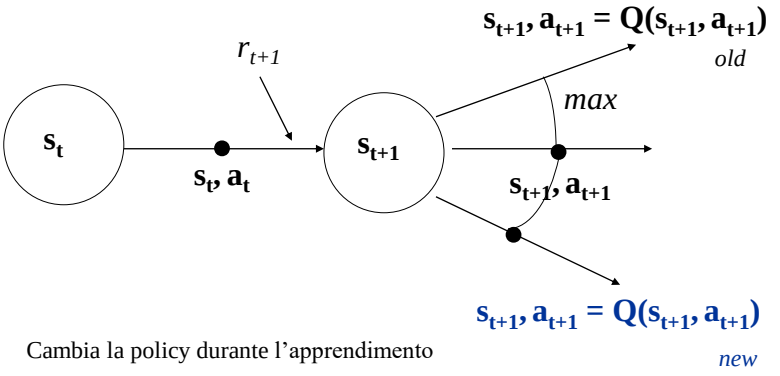
Ogni istante l'eleggibilità di tutti gli stati viene decrementata esponenzialmente secondo una costante 0 < λ < 1: $e(s,a) = \lambda\, e(s,a)$

λ = 0 ➔ e(s,a) = 0. Non c'è eleggibilità di coppie stato-azione diverse da quella visitata al tempo t. e(s,a) = 0 per $s \neq s_t$ e $a \neq a_t$.

λ = 1 ➔ e(s,a) = e(s,a) + 1. Ad ogni passo considero tutti gli stati visitati nel passato almeno una volta con un livello di eleggibilità pari al numero di volte in cui quella coppia stato-azione è stata visitata.

Q-learning

$$Q_{k+1}^{\pi}(s,a) = Q_k^{\pi}(s,a) + \alpha\left[r' + \gamma \max_{a'} Q_k^{\pi}(s',a') - Q_k^{\pi}(s,a)\right]$$

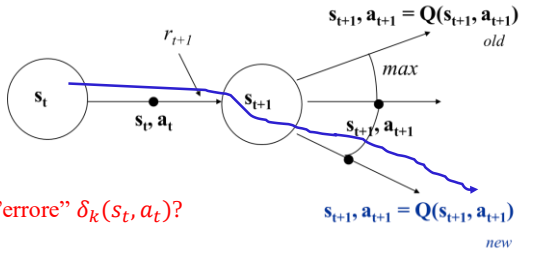


Cambia la policy durante l'apprendimento

Trace and Q(λ)

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha\left[r_{t+1} + \gamma \max_{a_{t+1}} Q(s_{t+1},a_{t+1}) - Q(s_t,a_t) \right]$$

$$\Delta Q_k^{\pi}(s_t,a_t) = \alpha \delta_k(s_t,a_t) \qquad \delta_k(s_t,a_t) = \left[r' + \gamma \max_{a'} (Q_k^{\pi}(s_{t+1},a') - Q_k^{\pi}(s_t,a_t) \right]$$



Quanto posso propagare all'indietro l'"errore" $\delta_k(s_t,a_t)$?

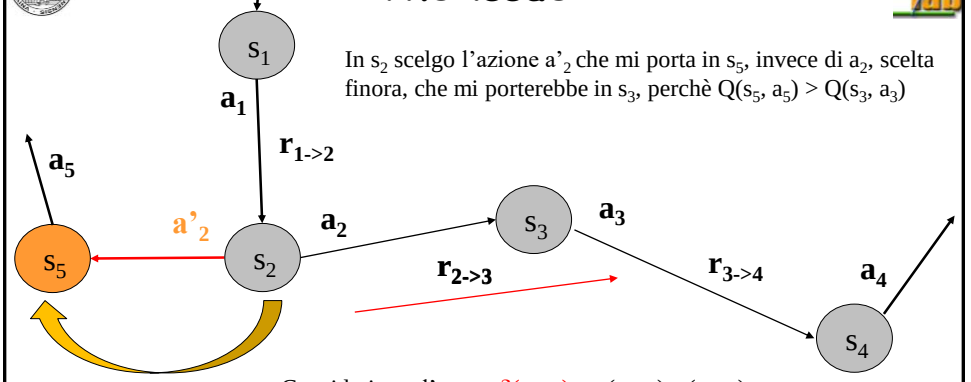
L'azione che scelgo può non essere la migliore, la policy viene modificata run-time e la funzione $Q(s_t,a_t)$ viene calcolata su un cammino diverso del grafo di transizione degli stati.

Se sceglie un'azione diversa, «Pollicino percorre una strada diversa da quella che aveva marcato fino al trial precedente»

Strategia

- L'eleggibilità viene aggiornata come in SARSA fino a quando l'azione scelta è l'azione prevista dalla policy.
- L'eleggibilità viene messa a zero ogni volta che si sceglie un'azione diversa da quella prescritta dalla policy.

The issue



In s_2 scelgo l'azione a'_2 che mi porta in s_5, invece di a_2, scelta finora, che mi porterebbe in s_3, perchè $Q(s_5, a_5) > Q(s_3, a_3)$

Consideriamo l'errore $\delta(s_3,a_3)$ tra (s_3,a_3) e (s_4,a_4):

$$\Delta Q_k^\pi(s_3,a_3) = \alpha e(s_3,a_3)\left[r_{3\to4,a3} + \gamma(Q_k^\pi(s_4,a_4) - Q_k^\pi(s_3,a_3)\right]$$

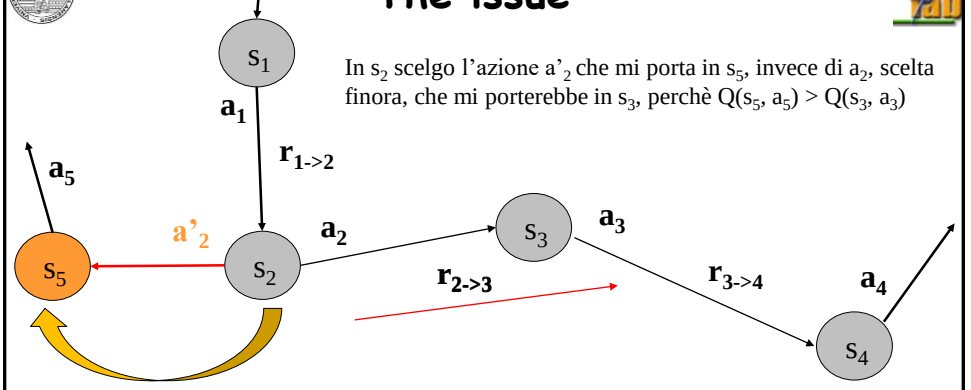
$$\Delta Q_k^\pi(s_2,a_2) = \alpha e(s_2,a_2)\left[r_{2\to3,a2} + \gamma(Q_k^\pi(s_3,a_3) - Q_k^\pi(s_2,a_2)\right]$$

Hanno senso

$$\Delta Q_k^\pi(s_1,\boldsymbol{a_1}) = e(s_1,a_1)\left[r_{1\to2,a1} + \gamma(Q_k^\pi(s_2,a_2) - Q_k^\pi(s_1,a_1)\right]$$

Ha senso se in s_2 scegliamo a'_2?

The issue



In s_2 scelgo l'azione a'_2 che mi porta in s_5, invece di a_2, scelta finora, che mi porterebbe in s_3, perchè $Q(s_5, a_5) > Q(s_3, a_3)$

$$\Delta Q_k^\pi(s_1,a_1) = e(s_1,a_1)\left[r_{1\to2,a1} + \gamma(Q_k^\pi(s_2,a_2) - Q_k^\pi(s_1,a_1)\right]$$

Non ha senso perché in s_2 scegliamo a'_2

L'aggiornamento potrebbe diventare:

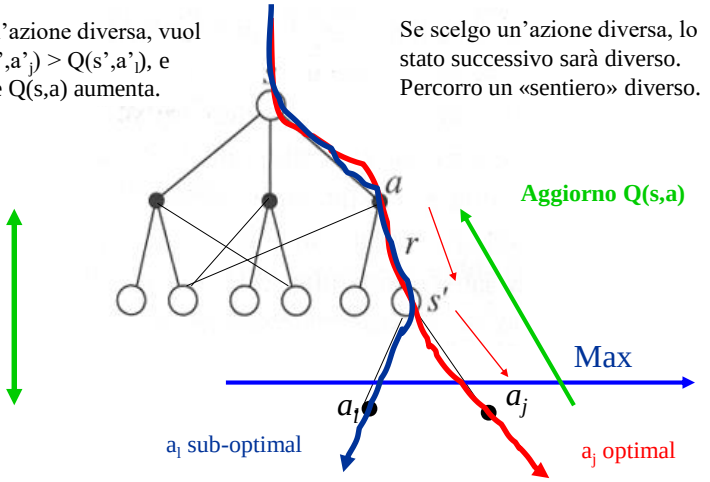
$$\Delta Q_k^\pi(s_1,a_1) = e(s_1,a_1)\left[r_{1\to2,a1} + \gamma(Q_k^\pi(s_2,a'_2) - Q_k^\pi(s_1,a_1)\right]$$

Ma come facciamo a calcolare l'eleggibilità dello stato successivo basandosi sull'eleggibilità calcolata fino a quel momento?

Analisi grafica delle mosse ε-eplorative

Se scelgo un'azione diversa, vuol dire che $Q(s',a'_j) > Q(s',a'_l)$, e quindi anche Q(s,a) aumenta.

Se scelgo un'azione diversa, lo stato successivo sarà diverso. Percorro un «sentiero» diverso.



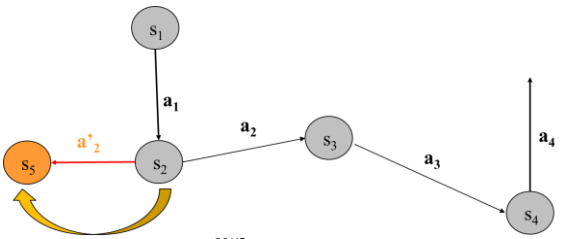
Cambia l'eleggibilità di tutti gli stati successivi: non so quali visiterò e in che sequenza. L'"errore" si può retro-propagare solo fino a quando si percorre la stessa strada.

Watkin's Q(λ)

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha\left[r_{t+1} + \gamma \max_{a_{t+1}} Q(s_{t+1},a_{t+1}) - Q(s_t,a_t)\right]$$

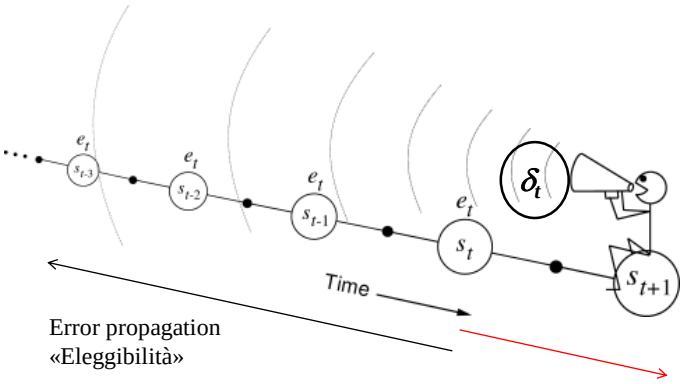
Posso sempre calcolare $Q(s_t, a_t)$, scegliendo il $\max(Q(s_{t+1}, a_{t+1}))$. Questo vuole dire ipotizzare di scegliere $a_{max} = \text{argmax}(\max(Q(s_{t+1}, a_{t+1}) \neq \pi(s'))$; in questo caso: $a_{max} \neq a'$.

Ma facendo questa operazione, l'eleggibilità degli stati successive deve essere calcolata da capo: viene percorso un diverso cammino nel grafo. Da lì in poi seleziono una sequenza diversa di transizioni di stato. Visito il grafo in modo diverso.



Osservazioni

When to stop propagation?



Error propagation
«Eleggibilità»

«Eleggibilità»?

Q-learning

Aggiorno Q:

$$Q_{k+1}(s,a) = Q_{k+1}(s.a) + \alpha\delta_t e_k(s,a)$$

$$\delta_t = r_{t+1} + \gamma \max_{a'} Q_t(s_{t+1}, a') - Q_t(s_t, a_t).$$

Scelta di a:

Se scelgo a_{max}, continuo come SARSA, altrimenti e(s,a) = 0.

Aggiorno e, ∀s ∀a, going backwards:

$$e_t(s,a) = \mathcal{I}_{ss_t} \cdot \mathcal{I}_{aa_t} + \begin{cases} \gamma\lambda e_{t-1}(s,a) & \text{if } Q_{t-1}(s_t,a_t) = \max_a Q_{t-1}(s_t,a); \\ 0 & \text{otherwise,} \end{cases}$$

1 or 0

Q-learning

Aggiorno Q:

$$Q_{k+1}(s,a) = Q_{k+1}(s.a) + \alpha\delta_t e_k(s,a)$$

$$\delta_t = r_{t+1} + \gamma \max_{a'} Q_t(s_{t+1}, a') - Q_t(s_t, a_t).$$

Aggiorno e, ∀s ∀a, going backwards:

$$e_t(s,a) = \mathcal{I}_{ss_t} \cdot \mathcal{I}_{aa_t} + \begin{cases} \gamma\lambda e_{t-1}(s,a) & \text{if } Q_{t-1}(s_t,a_t) = \max_a Q_{t-1}(s_t,a); \\ 0 & \text{otherwise,} \end{cases}$$

1 or 0

Se nello stato s, mantengo a -> aggiorno l'eleggibilità di s.
Se nello stato s, cambio a -> metto a 0 l'eleggibilità di s.

Algorithm for Watkin's Q(λ)

Inizialize Q(s,a) = 0; e(s,a) = 0; ∀s, ∀a
Repeat (for each episode):
  s = s_0;
  Repeat for each step of the episode:
    a = Policy(s)
    s_next = NextState(s,a)
    reward = Reward(s,a,s_next)
    a_pol = Policy(s_next) // Greedy or not greedy choice
    a_opt = $\max_b(Q(s_{next}, b))$
    if (a_opt != a_pol)
      a_next = a_opt
      UpdatePolicy(s_next, a_next)
    else
      a_next = a_pol
    endif
    $\delta = [r' + \gamma Q^{\pi}(s\_next, a\_next) - Q^{\pi}(s,a)]$
    $e(s,a) = e(s,a) + 1$
    for all s and all a:
      $Q^{\pi}(s,a) = Q^{\pi}(s,a) + a\ \delta\ e(s,a)$
      if (a_opt == a_next)
        $e(s,a) = \gamma\ \lambda\ e(s,a)$
      else
        $e(s,a) = 0$
      end
    end; s = s_next;
  until s = terminal state
until end of learning

Sommario

Traccia

Fuzzy RL

Clever Pac-man

Tohru Iwatani, formato arcade da sala, 1980.

N.A.Borghese, A.Rossini and C.Quadri (2012) Clever Pac-man, Proceedings of the 21st Italian Workshop on Neural Nets, WIRN2011, Frontiers in Artificial Intelligence and Applications, IOS Press (Apolloni, Bassis, Esposito, Morabito eds.), pp.11-19.

Applied Intelligent Systems Laboratory
Computer Science Department
University of Milano
http://ais-lab.di.unimi.it

Motivation

How can we make a computer agent play Pac-man?

The Pac-man game

Arcade computer game

- An agent that moves in a maze. The agent is a stilyzed yellow mouth that opens/closes.

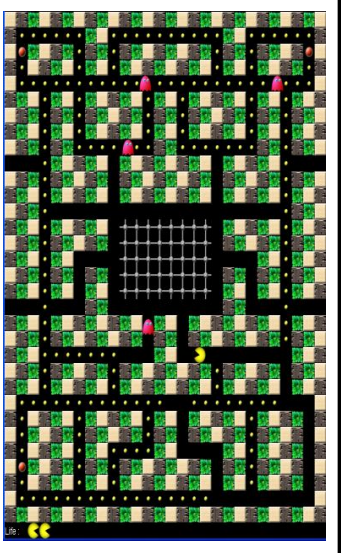
- The maze is constituted of corridors paved with (yellow) pills.

- When all pills are eaten the agent can move to the next game level.

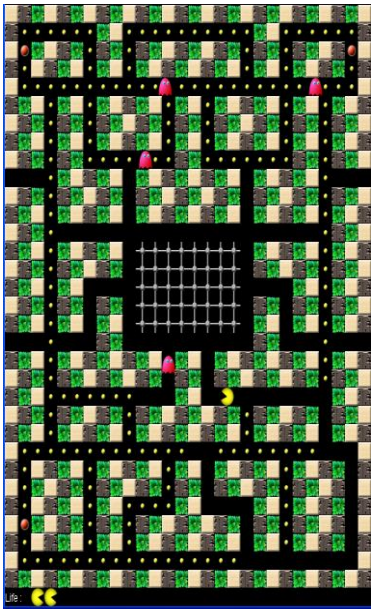
- Some enemies, with the shape of pink ghosts, are present, that go after the pacman.

- Special pills, called power pills (pink spheres) are present among the pills. They allow the pacman to eat the ghosts but their effect lasts for a limited amount of time.

- Each eaten pill is worth one point, while each eaten ghost is worth 200, 400, 800, 1600 points (first, second, third ghost).



Pac-man as a learning agent



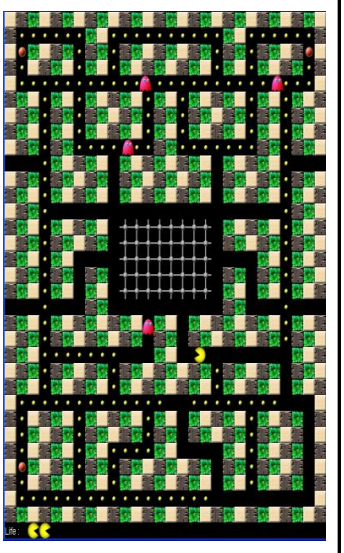
Environment:
- maze structure,

- ***State of the environment:***
- Ghosts position
- Ghosts motion direction (north-south-east-ovest): ghosts policy
- Pills position)
- Power pill position
- **Reward** for PacMan actions

Next state and reward are not known to the pac-man
→ **environment identification.**

Pacman – learning agent
No a-priori information is available to the pac-man.
Goal: maximize reward.
- Actions: move north-south-est-ovest
  Actions depend on actual situation (state)
- Estimates the value of each action in a given state.

Pac-man as a learning agent description



Environment description
- Ghost behavior
- Rewards

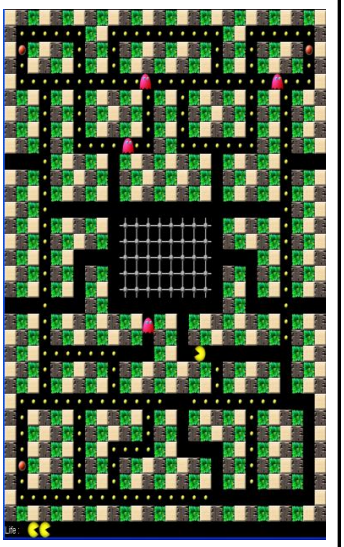
Pacman description
- Value function: Q(s,a)
- Policy: π(s,a).
- Learning machinery.
- Pacman behavior (motion strategy)

State definition
- Large number of cells (≅ 30 x 32 = 960) and situations
- **Fuzzy state definition** allows managing the large number of cells: **near, medium, far,** si utilizzano **delle distanza relative (to pac-man, ghost).**
- Minimum path has to be updated at each step as the Pac-man moves. The *Floyd-Warshall algorithm* is used to pre-compute the minimum path, distance between pairs of cells, for each cell of the maze, at game loading time.

Fuzzy state definition

State is constituted of:

- Ghosts position
- Ghosts motion direction (north-south-east-ovest): ghosts policy
- Pills position)
- Power pill position
- **Reward** for PacMan actions



State definition

- Large number of cells (≅ 30 x 32 = 960) and situations
- **Fuzzy state definition** allows managing the large number of cells: **near, medium, far,** si utilizzano **delle distanza relative (to pac-man, to ghost).**
- Minimum path has to be updated at each step as the Pac-man moves. The *Floyd-Warshall algorithm* is used to pre-compute the minimum path, distance between pairs of cells, for each cell of the maze, at game loading time.

The ghosts original behavior

In the original game design (*Susan Lammers: "Interview with Toru Iwatani, the designer of Pac-Man", Programmers at Work 1986*), the four ghosts had different personalities and behaviors when the power pill is not active:

a) Ghost #1, chases directly after Pac- man (red, «blinky»).
b) Ghost #2, positions himself a few dots in front of Pac-man mouth (if these two ghosts and the Pac-man are inside the same corridor a sandwich movement occurs).
c) Ghost #3 and #4, move randomly.

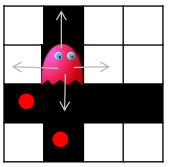
Moreover:
d) Ghosts have to escape the Pac-man when the power pill is active (fourth behavior).

In the present implementation all the four ghosts can assume all four possible behaviors depending on the situation (power pill active and closeness of pacman) of the game (the state) and is not assigned a-priori.

The more the game progresses the more the ghosts have to aim at the Pac-man.

The ghosts behavior::implementation

At each step each ghost has to decide if moving north, south, east, west **according to its behavior** decided by the game logic.



a) Hunting behavior. The ghost chooses the direction of the minimum path to the Pac-man.

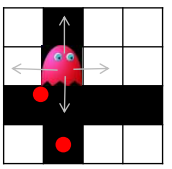
c) Random behavior. The ghost chooses an admissible direction randomly.

d) Shy behavior. The ghost moves away from the closest ghost. This allows distributing the ghosts inside the maze. When the power pill is active, the ghosts tend to move as far as possible from the Pac-man. ***The direction the maximize the increment of distance is chosen.*** When ties are present, the Pac-man makes a randomized choice to avoid stereotyped behavior.

Defence behavior. The ghosts go in the area in which the pills density is maximum. To this aim the maze is subdivided into nine partially overlapped areas: {0 - ½; ¼ - ¾; ½ - 1} and the ghost aims to the center of the area waiting for the Pac-man.

The Fuzzy behavior of Ghosts

At each step each ghost chooses among the four possible behaviors: *shy*, *random*, *hunting* and *defence*, according to a **policy** based on **fuzzy reasoning (FAM):**



Input state variables are computed according to:
• distance between the ghost and the Pac-man
• distance between the ghost and the nearest other ghost.
• frequency of the Pac-man eating pills.
• life time of the Pac-man (that is associated to its ability, the more the game progresses, the more aggressive become the ghosts - skill).
• power pill active

A set of rules have been designed for defining the next-state, according to the domain expert, like for instance:

· *If pacman_near AND skill_good, Then hunting_behavior*
· *If pacman_near AND skill_med AND pill_med, Then hunting_behavior*
· *If pacman_near AND skill_med AND pill_far, Then hunting_behavior*
· *If pacman_med AND skill_good AND pill_far, Then hunting_behavior*
· *If pacman_med AND skill_med AND pill_far, Then hunting_behavior*
· *If pacman_far AND skill_good AND pill_far, Then hunting_behavior*

Input class boundaries are chosen so that ghosts have hunting as preferred action (four times the other actions) in real game situations.

At start all ghosts are grouped in the center.

The agent and fuzzy Q-learning

Fuzzy description of the state is mandatory to avoid combinatorial explosion of the number of the states.

The state of the game for the Pac-man, is described by three (fuzzy) variables:
• minimum distance from the closest pill.
• minium distance from the closest power pill.
• minimum distance from a ghost.

Three fuzzy classes for each variable -> 27 fuzzy states. **The Agent (Pacman) can belong to more than one state** with a given membership value.

Fuzzy aggregated state	Closest ghost	Closest pill	Closest power pill
1	Low	Low	Low
2	Low	Low	Medium
3	Low	Low	High
4	Low	Medium	Low
5	Low	Medium	Medium
6	Low	Medium	High
7	Low	High	Low
8	Low	High	Medium
9	Low	High	High
10	Medium	Low	Low
11	Medium	Low	Medium
12	Medium	Low	High
13	Medium	Medium	Low
14	Medium	Medium	Medium
15	Medium	Medium	High
16	Medium	High	Low
17	Medium	High	Medium
18	Medium	High	High
19	High	Low	Low
20	High	Low	Medium
21	High	Low	High
22	High	Medium	Low
23	High	Medium	Medium
24	High	Medium	High
25	High	High	Low
26	High	High	Medium
27	High	High	High

Q-learning

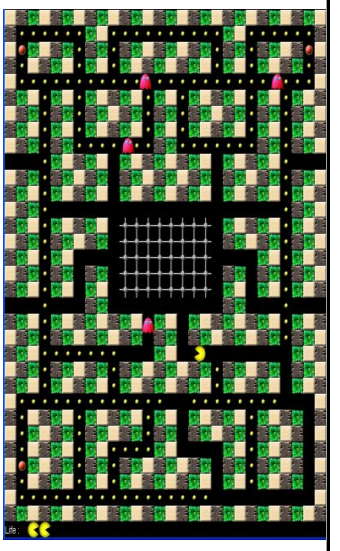
Agent – the pacman:

- State {s}: {(fuzzy state – relative distance) U PowerPill active}.
- Actions {a}: {Go to Pill, Go to Power Pill, Avoid Ghost, Go after Ghost}

The pacman learns (Q-learning):

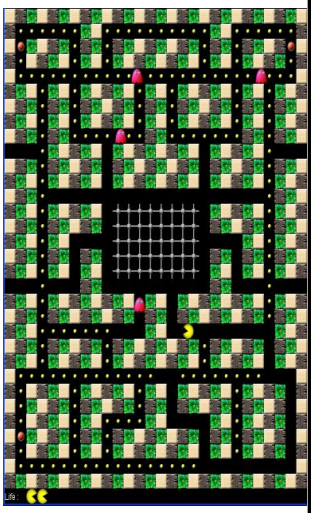
- Value function: $Q = Q(s_t, a_t)$
- Policy: $a_t = g(s_t)$

$Q(s_t,a_t) = Q(s_t,a_t) + \alpha[r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t,a_t)]$



Fuzzy State of the Pac-man

We measure the state:
-The distance from the closest ghost, c1 -> m(c1)
- The distance from the closest pill, c2 -> m(c2)
- The distance from the power pill, c3 -> m(c3)
Each element can fall in more than one fuzzy state at each time step



We compute the membership to a fuzzy state s_j , $\mu(s_j)$, as the average value of the membership to the three components of the state, $m(c_j)$ (we could have chosen the minimum):

$$\mu(s_j) = \frac{\sum_{i=1}^{3} m(c_i)}{3}$$

the three degrees of membership, $\mu(s_j)$ add to one.

More than one state can be active at each time step and

Fuzzy Q-learning

The value function for the state s_t (constituted of its 3 components, c_i, with their membership value, $m(c_i)$), in which the Pac-man choses an action, a_i, and moves from. The environment will drive the Pac-man in a new state.

$$Q(s_t^*, a_t) = \frac{1}{n} \sum_{i=1}^{n} \mu(s_{t,i})\ q(s_{t,i}, a_t)$$

where q(.) is the value function of the state components. It is updated using Q-learning strategy as:

$$q(s_{t,i}, a_t) = q(s_{t,i}, a_t) + \alpha_{s,a} \left[r + \gamma \cdot \frac{1}{N} \max_{a'} Q(s_{t+1}, a') - q(s_{t,i}, a_t) \right]$$

Where the value of each component q(.), is compared with the value of the state Q(.) and all components receive the state reward r.

α is chosen as: $\alpha_{s,a} = \frac{1}{\sum_{\tau=0}^{t-1} \mu(s_{\tau,i})}$

That is a natural extension of running average computation and it is inversely proportional to the cumulative membership of the component i, in all the states active at that time step.

For each fuzzy state, a different optimal action for the next state s*, is identified according to Q(s*,a*). The action implemented in the one associated to the maximum fitness of the associated fuzzy state.

Implementation issues of Pac-man policy

The agent has to choose a given action a_t in the state s_t according to its policy. a_t = {Go to Pill, Go to Power Pill, Avoid Ghost, Go to Ghost}

Policy: $a_t = \pi(s_t)$

Go to Pill. The Pac-man always goes to the closest pill, independently on the position of the ghosts. If ties occur the choice is randomized to avoid stereotyped behavior.

Go to Power Pill. Similar as above.

Go to Ghost. Similar as above.

Avoid Ghost. If only the closest ghost is considered, the Pac-man would easily run into a second ghost. The move the minimizes the weighted distance with all the ghost could be considered, but this would move the Pac-man in a small area close to the corners of the maze. We have implemented a weighted distance computed only inside a small area around the actual position of the Pac-man (that changes at each time step). Moreover, in case of ties, the Pac-man choses the direction the leads to the closest power pill (if still present in the maze).

Additional implementation issues

Few heuristics have been introduced to improve the policy:

Persistence *(cf. DeLooze, L.L.; Viner, W.R.; "Fuzzy Q-learning in a nondeterministic environment: developing an intelligent Ms. Pac-Management", Computational Intelligence and Games, 2009. CIG 2009. pp.162-169, 7-10 Sept. 2009).* Forcing the same action for n steps (n=5 here).

Persistence removal. When power pill effect ends. A brisk change of behavior is often observed.

Taboo. Inhibits the Pac-man to return in the previous state.

Parameters role

Rewards. The death of the Pac-man receives instant reward of -1000. A less negative reward was not enough to compensate all the positive points earned during a typical game. A more negative reward made the Pac-man "depressed" and little inclined to look for pills.

Fuzzy classes boundary: d=5, d=12 and d = 25 were assumed as maximum distance for the classes: low, medium and large. These values have been experimentally set analyzing the game results.

Pills reward: no particular effect was observed when the value was in the range [0.1 ÷ 1].

Greediness of the policy

Greediness of the policy: ε-greedy policy is fundamental to obtain very good results.

With random policy (blue) little points are gained.

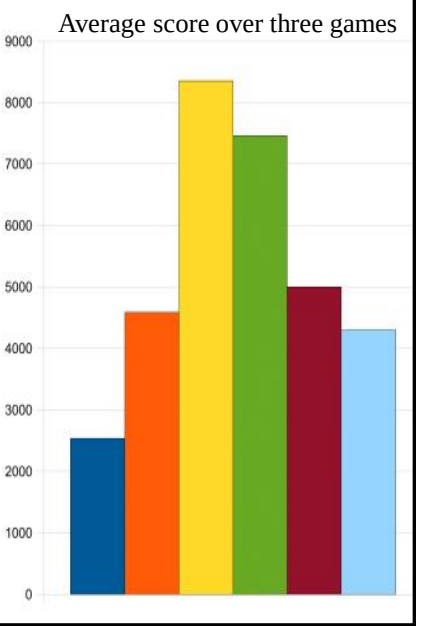
Some more points can be gained if the Pac-man always chooses "avoid ghosts" unless he has eaten the Power Pill (orange).

Maximum reward is obtained when Q-learning with ε-greedy policy with ε=0.1) choice is adopted and r = 0.1 per pill (yellow).

A high reward is obtained when Q-learning with ε-greedy policy with ε=0.1) choice is adopted and r = 1 per pill (green).

Less reward is obtained with Q-learning with greedy policy (brown).

An even small reward is obtained with Q-learning with greedy policy, when fuzzy classes boundaries are different: d = {6, 18, 30} (cyan).



Conclusion and further developments

- Highest score was around 4,500 and reported in *DeLooze, L.L.; Viner, W.R.; "Fuzzy Q-learning in a nondeterministic environment: developing an intelligent Ms. Pac-Man agent", Computational Intelligence and Games, 2009. CIG 2009. pp.162-169, 7-10 Sept. 2009.* We obtain here a large improvement in the score.
- Fuzzy approach has made RL approach feasible.

- We have only considered the bonus represented by power pills.
- A single scheme was used.
- Fuzzy classes boundaries were not optimized.
- A human player elaborates **strategies** both in chasing and escaping that are based on a global "view" of the game. This would require a much elaborate learning machinery than "simple" RL.

- Here is the Pac-man learning live....

Launch Fuzzy Pac-man

- Spostarsi nella cartella *bin dell'applicazione.*
- Lanciare il file main:

  java pacman.PacmanMAIN

Sommario

Traccia

Fuzzy RL