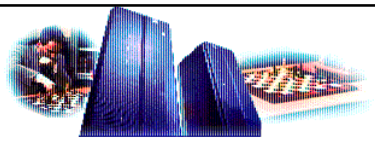
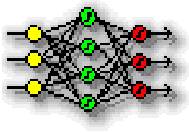




Sistemi Intelligenti
Reinforcement Learning: Temporal Difference

Alberto Borghese

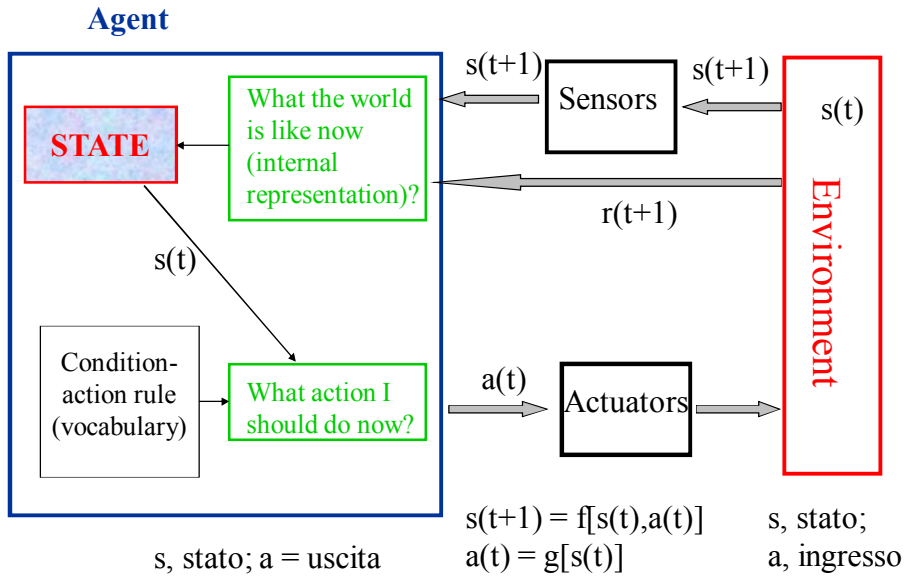
Università degli Studi di Milano
Laboratorio di Sistemi Intelligenti Applicati (AIS-Lab)
Dipartimento di Informatica
alberto.borghese@unimi.it

Sommario

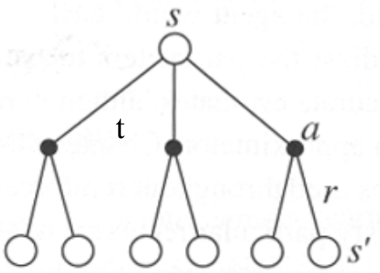
Temporal differences

SARSA

Schematic diagram of an agent



Tecnica full-backup

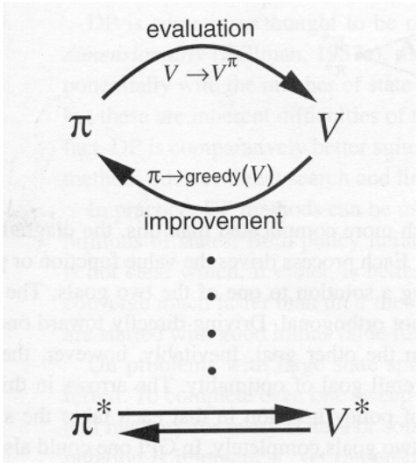


Back-up

$\pi(s,a)$ fissata

t+1

Conosciamo $V_k(s_t)$ $\forall s_t$, anche per s'_{t+1} quindi:
Analizziamo la transizione da s_t, a_t -> (s'_{t+1})
Calcoliamo un nuovo valore di V per s: $V_{k+1}(s_t)$ congruente con:
$V_k(s_{t+1})$ ed r_{t+1}
Full backup se esaminiamo tutti gli s', e tutte le a (cf. DP).
Da s' mi guardo indietro ed aggiorno V(s).



Schema di Apprendimento

Generalized Policy iteration { Policy iteration, Value iteration

Competizione e cooperazione -> V corretta e policy ottimale.

World RL competition

Started in NIP2006.

It became very popular and started a workshop on its own. Visit:

http://rl-competition.org

How About Learning the Value Function?

Facciamo imparare all'agente la value function, per una certa politica: V^{π}:

$$V^{\pi}(s) = \sum_{a_j} \pi(s, a_j) \sum_{s'} P_{s \to s' | a_j} \left[R_{s \to s' | a_j} + \gamma V^{\pi}(s') \right]$$

È una funzione dello stato.

Una volta imparata la value function, V^*, l'agente seleziona la policy ottima passo per passo, "one step lookahead":

$$\pi^*(s) = \arg\max_a \sum_{s'} P^a_{s \to s'} \left[R^a_{s \to s'} + \gamma V^{\pi}(s') \right]$$

Full backup, for all states

Value iteration

Facciamo imparare all'agente la value function, per una certa politica: V^{π}, analizzando quello che succede in uno step temporale:

$$V_{k+1}(s) = \sum_{a_j} \pi(a_j, s) \sum_{s'} P_{s \to s' | a_j} \left[R_{s \to s' | a_j} + \gamma V_k(s') \right]$$

Invece di considerare una policy stocastica, consideriamo l'azione migliore:

L'apprendimento della policy si può inglobare nella value iteration:

$$V_{k+1}(s) = \max_{a_j} \sum_{s'} P_{s \to s' | a_j} \left[R_{s \to s' | a_j} + \gamma V_k(s') \right] \qquad \forall s$$

Problema legato alla conoscenza della risposta dell'ambiente

$$V_{k+1}(s) \leftarrow \max_{a_j} \sum_{s'} P_{s \to s'|a_j} \left[R_{s \to s'|a_j} + \gamma V_k(s') \right]$$

Full backup, single state, s, all future states s'

Fino a questo punto, è noto un modello dell'ambiente:
- R(.)
- P(.)

Environment modeling -> Value function computation -> Policy optimization.

Osservazioni

Iterazione tra:
- Calcolo della Value function

$$V_{k+1}(s) = \sum_{a_j} \pi(s) \sum_{s'} P_{s \to s'|a_j} \left[R_{s \to s'|a_j} + \gamma \left[V_k(s') \right] \right]$$

- Miglioramento della policy

$$= \arg\max_{a_j} \sum_{s'} P_{s->s'|a_j} \left[R_{s->s'|a_j} + \gamma V^{\pi}(s') \right]$$

Non sono noti

Background su Temporal Difference (TD) Learning

Al tempo t abbiamo a disposizione:

$r_{t+1} = r'$ $\quad R_{s \to s'|a_j}$

$s_{t+1} = s'$ $\quad P_{s \to s'|a_j}$

Reward certo
Transizione certa
vengono misurati dall'ambiente

Come si possono utilizzare per apprendere?

Confronto con il setting associativo

$$Q_{k+1} = Q_k - \frac{Q_k}{N_{k+1}} + \frac{r_{k+1}}{N_{k+1}} = \boxed{Q_k + \alpha[r_{k+1} - Q_k]}$$

Occupazione di memoria minima: Solo Q_k e k.
NB k è il numero di volte in cui è stata scelta a_j.

Questa forma è la base del RL. La sua forma generale è:

NewEstimate = OldEstimate + StepSize [Target – OldEstimate]
*NewEstimate = OldEstimate + StepSize * Error.*

StepSize = $\alpha = 1/k$ $\quad$ *a = cost*
Rewards weight w = 1 $\quad$ *Weight of i-th reward at time k:* $w = (1\text{-}a)^{k-i}$

Qual è la differenza introdotta dall'approccio DP?

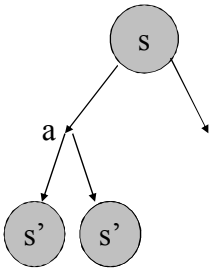
Un possibile aggiornamento

In iterative policy evaluation ottengo questo aggiornamento:

$$V_{k+1}(s) = \sum_{a_j} \pi(s, a_j) \sum_{s'} P_{s \to s'|a} \left[R_{s \to s'|a} + \gamma V_k(s') \right]$$

Ad ogni istante di tempo di ogni trial aggiorno la Value function:

$$V_{k+1}(s) = \left[r' + \gamma V_k(s) \right]$$



Qual'è il problema?

Un possibile aggiornamento di Q(s,a)

$$Q_{k+1} = Q_k - \frac{Q_k}{N_{k+1}} + \frac{r_{k+1}}{N_{k+1}} = \quad Q_k + \alpha \left[r_{k+1} - Q_k \right] = Q_k + \Delta Q_k$$

Quanto vale α?

$$V_k(s) = V_k(s) + \Delta V_k(s)$$

Come calcolo $\Delta V_k(s)$?

TD(0) update

Ad ogni istante di tempo di ogni trial aggiorno la Value function:

$$V_{k+1}(s_t) = V_k(s_t) + \alpha\left[r_{t+1} + \gamma V_k(s_{t+1}) - V_k(s_t)\right]$$

Da confrontare con la iterative policy evaluation:

$$V_{k+1}(s) = \sum_{a_j} \pi(s, a_j) \sum_{s'} P_{s \to s' | a_j} \left[R_{s \to s' | a_j} + \gamma V_k(s') \right]$$

Sample backup

E con il valore di uno stato sotto la policy π(s,a):

$$V^{\pi}(s) = E_{\pi}\{R_t \mid s_t = s\} = E_{\pi}\{r_{t+1} + \gamma V^{\pi}(s') \mid s_t = s\}$$

Quanto vale α?

Confronto con il setting associativo

$$Q_{k+1} = Q_k - \frac{Q_k}{N_{k+1}} + \frac{r_{k+1}}{N_{k+1}} = Q_k + \alpha\left[r_{k+1} - Q_k\right]$$

Occupazione di memoria minima: Solo Q_k e k.
NB k è il numero di volte in cui è stata scelta a_j.

Questa forma è la base del RL. La sua forma generale è:

NewEstimate = OldEstimate + StepSize [Target – OldEstimate]
*NewEstimate = OldEstimate + StepSize * Error.*

StepSize = α = *1/k* *a = cost*

Setting α value

$\alpha(s_t, a_t, s_{t+1}) = 1/k(s_t, a_t, s_{t+1})$, where k represents the number of occurrences of s_t, a_t, s_{t+1}. With this setting the estimated Q tends to the expected value of Q(s,a).

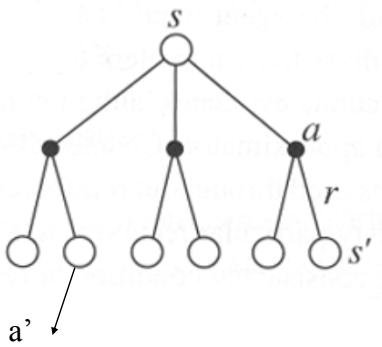
Per semplicità si assume solitamente $\alpha < 1$ costante. In questo caso, Q(s,a) assume il valore di una media pesata dei reward a lungo termine collezionati da (s,a), con peso: $(1-\alpha)^k$: *exponential recency-weighted average*.

Sommario

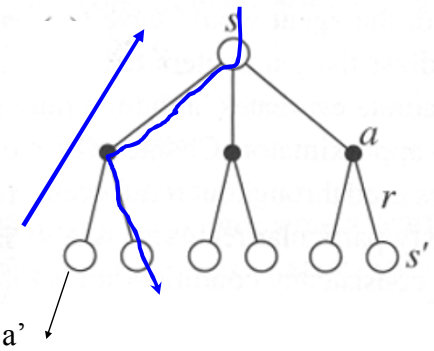
Temporal differences

SARSA

Sample backup

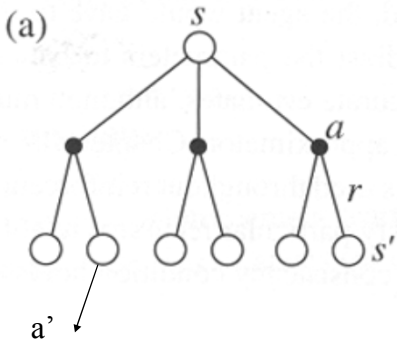


Full backup



Single sample is evaluated

Sample backup



SARSA algorithm

State –
Action –
Reward –
State (next) –
Action (next)

Campionamento di s_{t+1} e di r_{t+1}

TD(0)

Inizializziamo V(s) = 0.
Inizializziamo la policy: π(s,a).
Repeat
{ s = s0;
  Repeat // For each state until terminal state, analyze an episode
  { a = π(s);
    s_next = NextState(s, a);
    reward = Reward(s, s_next, a);
    V(s) = V(s) + α [reward + γ V(s next) – V(s)];
     s = s_next; }
  until TerminalState }
until convergence of V(s) for policy π(s,a)

Esempio: valutazione della policy TD

Stato	Tempo percorrenza stimato del segmento	Tempo percorrenza attuale del segmento	Tempo totale previsto in precedenza $Q^k(s,a)$	Tempo totale previsto aggiornato $Q^{k+1}(s,a)$	Increase or decrease Q(s,a)
Esco dall'ufficio	0	0	30	30 + (10 + 25 – 30)	>
Salgo in auto	5	10	25	25 + (15 + 10 - 25)	=
Esco dall'autostrada	15	15	10	10+ (10 + 5 - 10)	>
Esco su strada secondaria	5	10	5	5+ (2 + 3 - 5)	=
Entro Strada di casa	2	2	3	3 + (1 + 0 - 3)	<
Parcheggio	3	1	0	0	

Q(s,a) è l'expected "Time-to-Go" - $\gamma = 1$ $\alpha = 1$

Alcuni passi di apprendimento di TD(0)

Alcuni passi di iterazione per TD(0)

$V(0) = V(0) + \alpha\,(r1 + \gamma V(1) - V(0)) = 30 + \alpha\,(5 + 35 - 30) = 30 + \alpha * \Delta$ Stima iniziale del tempo di percorrenza totale: 30m
Tempo di percorrenza fino all'auto: 5m
Stima del tempo di percorrenza dal parcheggio: 35m

$V(1) = V(1) + \alpha\,(r1 + \gamma V(2) - V(1)) = 35 + \alpha\,(20 + 15 - 35) = 35 + \alpha * \Delta$
Stima iniziale del tempo di percorrenza dal parcheggio: 35m
Tempo di percorrenza fino ad uscita autostrada: 20m
Tempo di percorrenza fino ad uscita autostrada: 20m
Stima del tempo di percorrenza dall'uscita autostrada: 15m

Ruolo di α

$Q(1,a_1) = Q(1,a_1) + \alpha\,(r_1 + \gamma Q(2,a_2) - Q(1,a_1)) = 30 + \alpha\,(10+25 - 30) = 30 - \alpha*5$
Stima iniziale del tempo di percorrenza dal parcheggio: 30m
Tempo per raggiungere l'auto: 10m
Stima del tempo di percorrenza dall'uscita Dal parcheggio: 25m

$\alpha < 1$.
If $\alpha << 1$ aggiorno molto lentamente la value function.

If $\alpha = 1/k$ aggiorno la value function in modo da tendere al valore atteso. Devo memorizzare le occorrenze dello stato s.

If α = cost. Aggiorno la value function, pesando maggiormente i risultati collezionati dalle visite dello stato più recenti.

Proprietà del metodo TD

Non richiede conoscenze a priori dell'ambiente.
L'agente stima dalle sue stesse stime precedenti (bootstrap).
Si dimostra che il metodo converge asintoticamente.

Batch vs trial learning.

Converge!!

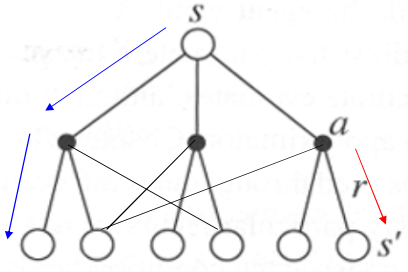
$$V^{\pi}(s_t) = V^{\pi}(s_t) + \alpha\left[r_{t+1} + \gamma V^{\pi}(s_{t+1}) - V^{\pi}(s_t)\right]$$

Single backup, single state, s_t, single future state s_{t+1}

Rimpiazza iterative Policy evaluation.
Rimane il passo di Policy iteration (improvement).

Le value function

$$V^{\pi}(s) = E_{\pi}\{R_t \mid s_t = s\} = E_{\pi}\left\{\sum_{k=0}^{\infty}\gamma^k r_{t+k+1} \,\middle|\, s_t = s\right\} = \left[\sum_{a_j}\pi(a_j, s)\right]\sum_{s'}P_{s\to s'|a_j}\left[R_{s\to s'|a_j} + \gamma V^{\pi}(s')\right]$$



$$Q^{\pi}(s,a) = E_{\pi}\{R_t \mid s_t = s,\ a_t = a\} = E_{\pi}\left\{\sum_{k=0}^{\infty}\gamma^k r_{t+k+1} \,\middle|\, s_t = s, a_t = a\right\}$$

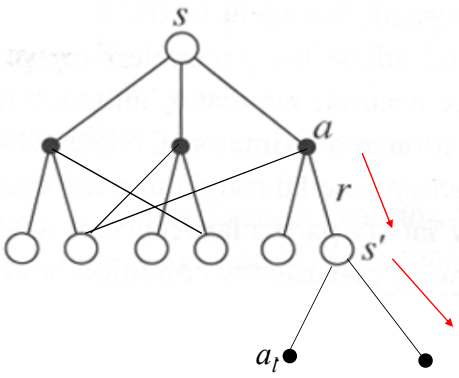
$$= \sum_{s'}P_{s\to s'|a_j}\left[R_{s\to s'|a_j} + \gamma V^{\pi}(s')\right]$$

Serve davvero la Value Function?

La Value Function deriva dalla visione della Programmazione Dinamica.

Ma è proprio necessario conoscere la Value function (che riguarda gli stati)? In fondo a noi interessa determinare la Policy.

Calcolo ricorsivo della value function Q



$$Q^{\pi}(s,a) = E_{\pi}\{R_t \mid s_t = s,\ a_t = a\} = E_{\pi}\left\{\sum_{k=0}^{\infty}\gamma^k r_{t+k+1}\,\middle|\, s_t = s, a_t = a\right\}$$

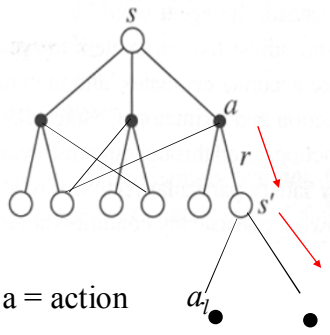
$$= \sum_{s'} P_{s\to s'|a}\left[R_{s\to s'|a} + \gamma \sum_{l}\pi(s',a_l)Q^{\pi}(s',a_l)\right]$$

Come apprendere Q: SARSA

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

1) Apprendiamo il valore di Q per una policy data (***on-policy***).

2) Dopo avere appreso la funzione Q, possiamo modificare la policy in modo da migliorarla (**policy improvement**)



s = state, a = action, r = reward, s = state, a = action è di tipo TD(0)

SARSA Algorithm (progetto)

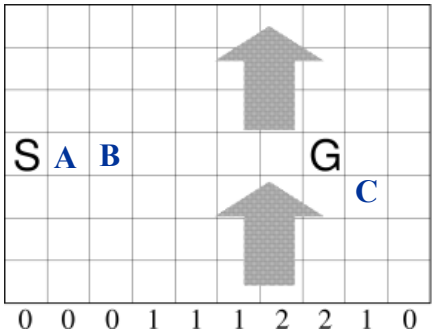
```
Q(s,a) = rand();   // ∀s, ∀a, eventualmente Q(s,a) = 0
Repeat                          // for each episode
{       s = s0;
        Repeat                  // for each step of the single episode
        {
                a = Policy(s);          // ε-greedy??
                s_next = NextState(s,a);
                reward = Reward(s,s_next,a);
                a_next = Policy(s_next);        // ε-greedy?
                Q(s,a) = Q(s,a) + α [reward + γ Q(s_next, a_next) – Q(s,a)];
                s = s_next;
        }                       // until last state
}                               // until the end of learning
```

1) Apprendiamo il valore di Q per una policy data (on-policy).
2) Dopo avere appreso la funzione Q, possiamo modificare la policy in modo da migliorarla.

Come integrare i due passi?

Esempio

From Start to Goal.

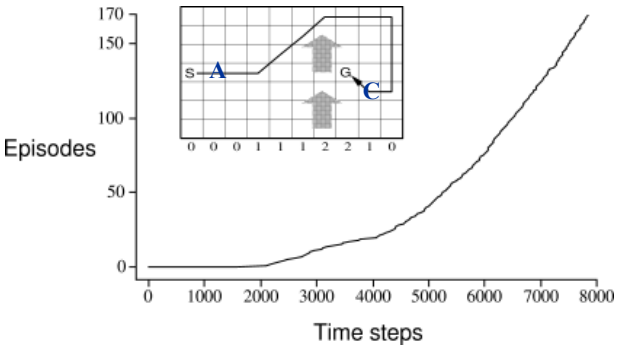


standard moves

Upwards wind

Q(s,a) iniziale = 0.
r = 0 se s' = G; altrimenti r = -1.
π(s,a) data.

Esempio - risultato



Policy π, greedy or ε-greedy

ε = 0.1
α= 0.5
γ = 1

Per trial or per epoch

Al termine, policy improvement.

Correzione di Q ad un passo:
Q(S, east) = 0 + 0.5 [-1 + 0 – 0] = -0.5
Q(A, east) = 0 + 0.5 [-1 + 0 –0] = -0.5
Q(C, west) = 0 + 0.5 [0 + 0 – 0] = 0; (NB c'è il vento verso l'alto di 1)

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

Sommario

Temporal differences

SARSA