RELAZIONE DI PROGETTO
DI
"INTELLIGENT ROBOTIC SYSTEMS"

Firefighter Robot

Repository

Zandoli Silvia
950299
silvia.zandoli2@studio.unibo.it

Vignola Francesco
895956
francesco.vignola@studio.unibo.it

21 marzo 2023

Indice

1 Descrizione del problema

Si vuole progettare ed implementare un sistema composto da due robot autonomi ed eterogenei nell'ambito della ricerca e soccorso di persone in situazioni di pericolo e in ambienti sconosciuti.

Come scenario si è scelta una casa che si immagina essere in fiamme che verrà simulata, in ARGoS, tracciando il perimento dell'edificio.
I due robot hanno comportamenti diversi. In particolare:

- robot *firefighter*: ha il compito di esplorare l'ambiente per trovare il robot *survivor*.
- robot *survivor*: aspetta di essere soccorso.

Si immagina che il robot durante lo svolgimento del suo compito sia in grado di schivare le sorgenti luminose ed eventuali ostacoli che incontrerà.

2 Analisi del problema

2.1 Reinforcement Learning

Il Reinforcement Learning è un processo caratterizzato da due entità: un ambiente (environment) che semplifica una certa realtà e un agente che interagisce con l'ambiente.
Questa tecnica ben si può applicare al nostro problema: l'agente in questo caso è il **robot firefighter** che non disponendo di una mappa dell'edificio deve raggiungere il **survivor** nel minor tempo possibile.
In un approccio classico per calcolare il valore atteso delle ricompense future per ogni azione in ogni possibile stato e permettere all'agente di scegliere la migliore azione in ogni stato, viene utilizzata la Q-Table. A volte lo spazio degli stati e delle azioni può essere troppo grande, per cui un approccio tabulare non è sufficiente. Per questo si è deciso di utilizzare il Q-learning con la function approximation.

2.2 Q-learning con la function approximation

Tale tecnica permette di modellare lo stato come un insieme di feature attive. L'obiettivo è di utilizzare questo insieme di "feature" per generalizzare la stima del valore in stati che hanno caratteristiche simili. Questo approccio non troverà mai il vero valore di uno stato, ma un'approssimazione di esso.
Sicuramente la performance dipenderà molto dalla qualità delle feature scelte.
I metodi che utilizzano queste approssimazioni sono chiamate "Function Approximator" e sfruttano la discesa del gradiente per ottenere il risultato migliore. Dovendo essere delle funzioni differenziabili la scelta della funzione di approssimazione ricade sulla combinazione lineare o sulla rete neurale.
Consideriamo il primo caso:
Per calcolare i Q-values con la funzione di approssimazione bisognerà trovare delle feature che sono funzioni degli stati e delle azioni. Avremo quindi una combinazione lineare delle feature ϕ:

$$Q(s,a) = \theta_0 \cdot 1 + \theta_1\phi_1(s,a) + \cdots + \theta_n\phi_n(s,a) = \theta^T\phi(s,a)$$

Nella progettazione delle feature è importante considerare anche le azioni oltre agli stati perchè possono avere un grande impatto sulla reward ottenuta. Pertanto la progettazione delle feature è avvenuta considerando solo lo stato ed è stato adottato il *dimensional scaling trick* che distingue esplicitamente le diverse azioni per ogni stato.

Let $\mathbf{w}$ and $\mathbf{e}$ be vectors with one component for each possible feature
Let $\mathcal{F}_a$, for every possible action a, be a set of feature indices, initially empty
Initialize $\mathbf{w}$ as appropriate for the problem, e.g., $\mathbf{w} = \mathbf{0}$
Repeat (for each episode):
$\mathbf{e} = \mathbf{0}$
$S \leftarrow$ initial state of episode
Repeat (for each step of episode):
For all $a \in \mathcal{A}(S)$:
$\mathcal{F}_a \leftarrow$ set of features present in S, a
$Q_a \leftarrow \sum_{i \in \mathcal{F}_a} w_i$
$A^* \leftarrow \arg\max_a Q_a$
$A \leftarrow A^*$ with prob. $1 - \varepsilon$, else a random action $\in \mathcal{A}(S)$
If $A \neq A^*$, then $\mathbf{e} = \mathbf{0}$
Take action A, observe reward, R, and next state, S'
$\delta \leftarrow R - Q_A$
For all $i \in \mathcal{F}_A$:
$e_i \leftarrow e_i + 1$ (accumulating traces)
or $e_i \leftarrow 1$ (replacing traces)
If S' is terminal, then $\mathbf{w} \leftarrow \mathbf{w} + \alpha\delta\mathbf{e}$; go to next episode
For all $a \in \mathcal{A}(S')$:
$\mathcal{F}_a \leftarrow$ set of features present in S', a
$Q_a \leftarrow \sum_{i \in \mathcal{F}_a} w_i$
$\delta \leftarrow \delta + \gamma \max_{a \in \mathcal{A}(S')} Q_a$
$\mathbf{w} \leftarrow \mathbf{w} + \alpha\delta\mathbf{e}$
$\mathbf{e} \leftarrow \gamma\lambda\mathbf{e}$
$S \leftarrow S'$

Figura 1: Versione lineare e a discesa del gradiente dell'algoritmo $Q(\lambda)$ di Watkins con feature binarie ed $\epsilon-$greedy policy

3 Soluzione

Il problema è stato affrontato utilizzando l'approccio detto di **shaping**, ovvero è stato rafforzato il comportamento in modo incrementale fino ad arrivare a quello finale.

Il task di base che il robot deve apprendere è come raggiungere il robot *survivor*.

3.1 Inseguimento del Survivor

L'inseguimento del *survivor* è stato implementato sulla base della ricezione del messaggio di aiuto inviato dal *survivor*. In un primo momento si è sfruttato il positioning affinchè il robot apprendesse come raggiungere il survivor solo in funzione della distanza euclidea. Successivamente si è utilizzato il range_and_bearing con cui è possibile accedere alle informazioni relative alla distanza e alla direzione del *survivor* rispetto al *firefighter*

L'applicazione del Q-learning con Linear Function Approximation ha richiesto la progettazione della **funzione di reward** e delle **feature**.

3.1.1 Progettazione delle feature

Le feature sono state progettate considerando unicamente la direzione del *survivor* rispetto al *firefighter*.

In particolare, essendo le feature binarie e l'angolo espresso in radianti è stato necessario binarizzare l'input utilizando delle soglie: se l'angolo è contenuto in un certo range allora si imposta la specifica feature ad 1, altrimenti a 0.

Le feature ottenute sono 7 in quanto è stata considerata solo la parte "frontale" del robot:

- 6 feature che si attivano se il da direzione del *firefighter* rispetto al *survivor* è compresa nei range [0, 15°], [15°, 30°], [30°, 45°], [45°, 60°], [60°, 75°] e [75°, 90°]
- 1 feature con bias costante, pari a 1.

```
function signal_detection_15()
        local message = nearest_robot_message()
        if not next(message) then
                return 0
        end

        local transmitter_angle = message.horizontal_bearing
        local result = 0 <= transmitter_angle and transmitter_angle <= math.pi/12
        return result and 1 or 0
end
```

3.1.2 Progettazione della Funzione di Reward

La funzione di Reward è stata implementata considerando:

- la direzione del *firefighter* rispetto al *survivor*:
  - viene dato un premio di 100 se il *firefighter* è in direzione del *survivor*;
  - altrimenti una penalità di -100;
- la distanza del *firefigter* dal *survivor* rispetto al passo precedente:
  - se la distanza è maggiore rispetto al passo precedente, allora viene dato un premio di 100;
  - se la distanza è minore rispetto al passo precedente, allora viene data una penalità di -100;
  - altrimenti viene dato 0.

Inoltre, il *firefighter* viene rinforzato maggiormente se si avvicina sempre di più (la distanza si riduce continuamente) al *survivor* e mantiene la sua direzione.

```
local reward = function()
        local robot_position = nearest_robot_message()
        local current_distance = robot_position.range
        local reward = 0

        if next(robot_position) and previous_distance ~= nil then
                local is_closer = current_distance < previous_distance
                local direction = robot_position.horizontal_bearing
                local in_survivor_direction =
                    -math.pi/18 <= direction and direction <= math.pi/18

                if in_survivor_direction and is_closer then
                        direction_reward = 100
                        position_reward = 100
                else
                        direction_reward = -100
                        position_reward = (current_distance > previous_distance) and -100 or 0
                end
                reward = position_reward+direction_reward
        end
        previous_distance = current_distance
        return reward
end
```

4 Risultati

Sono stati fatti numerosi test e il robot ha dimostrato un comportamento abbastanza robusto ed è sempre riuscito ad arrivare al goal. Di seguito si riporta la descrizione di alcuni risultati ottenuti.

4.1 Arena e Condizioni iniziali

L'arena è la rappresentazione di una casa avente le pareti che deliminano le stanze e le porte che le connettono.

Per il primo esperimento è stata utilizzata solo una stanza della casa, quella più grande, ovvero la "living room".

Il robot *survivor* è stato posizionato in un angolo della *living room* e tenuto fisso per tutti gli esperimenti, sia in termini di posizione che di orientamento. Diversamente, il robot *firefighter* è stato posizionato nell'angolo opposto al *survivor* per ogni esperimento, mantenendo fissa sia la posizione che la direzione.

Gli esperimenti sono stati eseguiti inizialmente senza alcun rumore, in modo da valutare il comportamento del robot in condizioni ottimali. Successivamente è stato introdotto in modo incrementale sia sul *survivor* che sul *firefighter* in modo da testare la robustezza del comportamento del robot.

4.2 Analisi senza rumore

Il comportamento osservato dalla calibrazione manuale dei parametri rappresenta un buon risultato.

In Tabella 1 sono riportati i valori degli iperparametri che hanno prodotto due comportamenti che abbiamo ritenuto importante mostrare. Mentre, in Figura 2 e in Figura 3 i grafici corrispondenti.

Il grafico vede sull'asse delle ascisse il numero di episodi, mentre sull'asse delle ordinate la media delle reward di ciascun episodio.

	α	γ	λ	ϵ	eposides
Comportamento 1	0.2	0.9	0.8	0.1	400
Comportamento 2	0.2	0.9	0.8	0.9->0.1	400

Tabella 1: Valori degli iperparametri calibrati

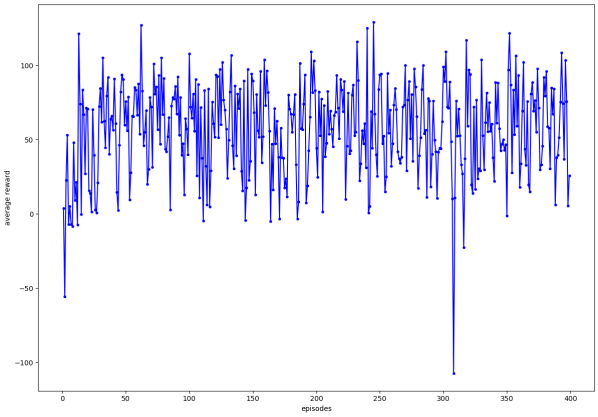

Figura 2: Average Reward con epsilon fisso

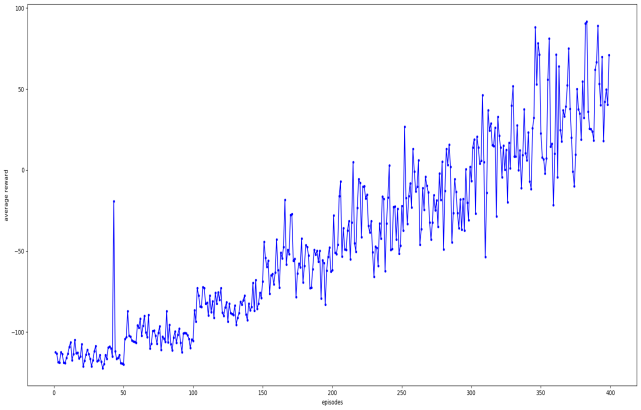

Figura 3: Average Reward con epsilon decrescente per batch di episodi

Osservazioni sul comportamento

Figura 1: Il robot raggiunge il *survivor* con qualche incertezza, fermandosi in vari momenti o cambiando direzione.

Figura 2: Il robot raggiunge il *survivor* con un avvio più lento rispetto al primo comportamento. Anche in questo caso sono presenti momenti di incertezza che portano il robot a fermarsi per capire quale direzione intraprendere.

4.3 Analisi con rumore

Per verificare la robustezza del comportamento, il robot è stato addestrato, in un secondo momento, aggiungendo rumore sui sensori e sugli attuatori.

In particolare, il rumore è stato aggiunto gradualmente sia sul *survivor* che sul *firefighter*.

Di seguito vengono riportati i grafici per i due comportamenti, considerando il caso con rumore=0.5 su entrambi i robot.

Osservazioni sul comportamento

Figura 1: Il robot spesso arriva a destinazione percorrendo una supercifie della stanza più ampia anziché andare diretto verso il *survivor*. Di conseguenza, il tempo con cui arriva a destinazione è maggiore rispetto allo scenario senza rumore. A volte, essendoci delle aperture tra le varie camere, tende ad andare nella camera etichettata "garden".

Figura 2: Anche in questo caso il robot percorre una superficie più ampia, mostrando un comportamento di ispezione, ma riesce ad arrivare a destinazione.

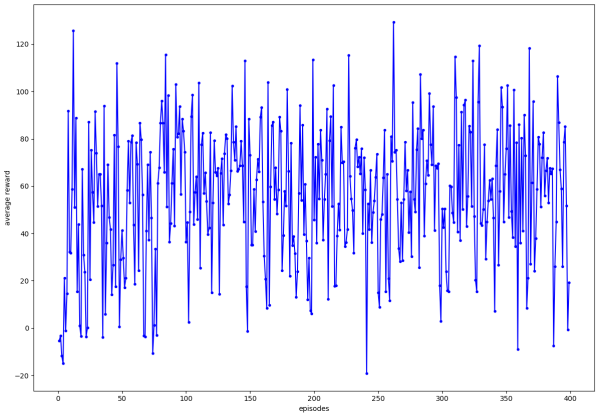

Figura 4: Average Reward con epsilon fisso

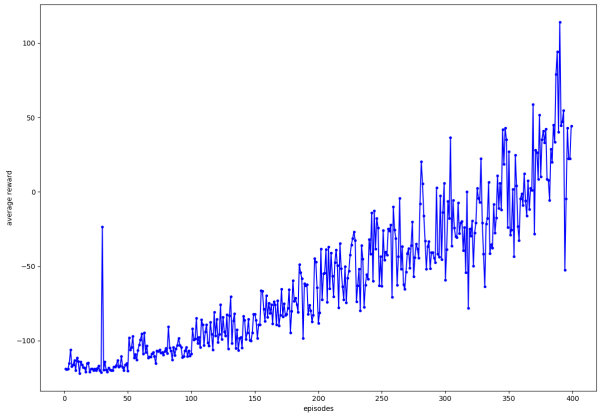

Figura 5: Average Reward con epsilon decrescente per batch di episodi

5 Sviluppi Futuri

Col presente progetto è stata affrontata la versione base del problema finale.
Gli sviluppi futuri prevedono l'aggiunta graduale di ulteriori requisiti affinché il robot possa:

- ampliare le proprie capacità ed apprendere come evitare gli ostacli (pareti, box e fuoco);
- portare in salvo il root *survivor*, ragiunggendo lo spot.

Inoltre, si deve implementare anche la logica per il *survivor* affinché segua il robot allo spot.
Infine, si possono testare anche altre tecniche, come la Rete Neurale per apprendere la funzione di approssimazione, in modo da evitare le restrizione della combinazione lineare e non imporre vincoli restringenti in fase di progettazione.

Bibliografia principale

[Sei16] Daniel Seita. *Understanding Q-learning and linear approximation*. https://danieltakeshi.github.io/2016/10/31/going-deeper-into-reinforcement-learning-understanding-q-learning-and-linear-function-approximation/. Ott. 2016.

[Was16] Dan Weld/University of Washington. «Approximate Q-Learning». In: (2016). URL: https://courses.cs.washington.edu/courses/cse473/16au/slides-16au/18-approx-rl2.pdf.

[R S18] A. Barto R. Sutton. *Reinforcement Learning: an Introduction. Second Edition*. The MIT Press, 2018.

[Mar20] Mario Martin. «Reinforcement Learning - Function Approximation». In: (2020). URL: https://www.cs.upc.edu/~mmartin/URL/Lecture3.pdf.

[Fer] Alan Fern. *RL for Large State Spaces: Value Function Approximation*. https://oregonstate.instructure.com/files/67025084/download?download_frd=1.

[SB] S.B. *Control with function Approximation*. http://www.incompleteideas.net/book/ebook/node89.html.