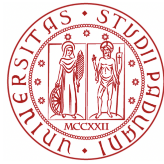

UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

CORSO DI LAUREA IN
INGEGNERIA DELL'INFORMAZIONE

Analisi di giochi di jamming su canali AWGN

Relatore:
LEONARDO BADIA

Laureando:
ALESSANDRO BALDO
1195762

Anno Accademico 2021/2022

Abstract

In questo lavoro verranno espansi i modelli di analisi di giochi di jamming dinamici, in contrasto con i modelli classici caratterizzati da nodi statici. I giochi considerano giocatori mobili il cui obiettivo è di massimizzare o minimizzare l'SNR del giocatore ricevitore, muovendosi su un canale fisico AWGN in condizioni di attenuazione di spazio libero. Nello specifico, verranno considerati tre spazi di gioco distinti, caratterizzati da diverse complessità di movimento e corredati di analisi di strategie convergenti ad un equilibrio dinamico tramite tecniche di reinforcement learning e un innovativo approccio tramite reti neurali ricorrenti.

Indice

1 Introduzione **1**

2 Modello fisico **3**

2.1 Capacità di canale 3

2.2 Capacità di canale in presenza di jammer 4

2.3 Parametri di analisi 5

2.4 Spazio di gioco 5

3 Formalizzazione del gioco **7**

3.1 Introduzione teorica 7

3.2 Il gioco dinamico 7

3.3 Spazio 1x5 8

3.4 Spazio 3x3 9

3.5 Spazio radiale 9

4 Progettazione dell'intelligenza artificiale **11**

4.1 Q-learning 11

4.2 Reti neurali 12

4.3 Processo di apprendimento 13

4.4 Reti neurali ricorrenti 13

4.5 L'architettura completa 14

4.6 Configurazione di apprendimento 16

5 Risultati **19**

5.1 Metodo di analisi 19

5.2 Gioco 1x5 19

5.3 Gioco 3x3 21

5.4 Gioco radiale 21

6 Conclusioni **25**

7 Codice **27**

Bibliografia **39**

Capitolo 1

Introduzione

Un comune ambito di analisi di sistemi di telecomunicazioni tramite la teoria dei giochi è quello del jamming [1]. Per esempio, è facilmente modellabile lo scenario in cui un utente cerca di ricevere dati da un trasmettitore attraverso un canale wireless, ma la comunicazione viene disturbata da un jammer malevolo [2], [3].

Scopo di questa tesi è di espandere i modelli di questo tipo analizzati in [4] e [5], in particolare concentrandosi sulle formulazioni dei giochi dinamici simultanei, nei quali i giocatori hanno conoscenza incompleta dello stato corrente, perché maggiormente comparabili con situazioni di jamming reale e più interessanti per l'analisi tramite intelligenze artificiali.

Nonostante il setup di base sia comunemente analizzato in letteratura [2], [3], [6], [7], infatti, vi un numero limitato di lavori che analizzano il sistema dal punto di vista dinamico [4], [5], sebbene rappresenti un'applicazione facilmente applicabile a sistemi reali, nei quali è comune la mobilità dei nodi per sopperire alla scarsa qualità di ricezione [8].

A causa dell'alta dinamicità dei giochi, gli spazi da analizzare esplodono facilmente, risultando nell'impossibilità di una pura analisi di game-theory del sistema [9]. Per questo in [4] e [5] viene esposta l'analisi tramite reinforcement learning, argomentando che i risultati ottenuti permettano di estrapolare informazioni importanti sulle strategie possibili da effettuare per i giocatori.

All'approccio tramite apprendimento automatico in Q-learning, si è scelto di espandere l'analisi operando in maniera diametralmente contrastante e innovativa, sviluppando una rete neurale ricorrente istruita tramite apprendimento supervisionato.

I due sistemi verranno confrontati reciprocamente nel ruolo di agente e di jammer, per valutare le differenze di rendimento dei diversi approcci.

Un ulteriore espansione sui modelli discussi è quella dell'aggiunta di un nuovo spazio di gioco, che unisce le caratteristiche degli spazi di gioco di [4] e [5] risultando in una complessità notevolmente maggiore. Questo spazio risulta utile nel rapportare l'analisi tramite intelligenze artificiali ad uno spazio più generalizzato. Il diverso rendimento delle IA nei tre differenti spazi di gioco fornirà utili informazioni sulle generalizzazioni dei risultati.

Infine, è stata ampliata l'analisi comparando il rendimento delle IA in relazione ad un'intelligenza casuale, che si muove nello spazio senza cercare di massimizzare o minimizzare il rendimento ma scegliendo mosse a caso. Questo elemento trova il parallelo in un giocatore ignaro dello stato e degli obiettivi del gioco, come per esempio un jammer "innocente" (una persona ignara del suo rumore generato a scapito degli altri giocatori), da non confondere con il *friendly jamming* [7], [10], cioè jamming usato per impedire a entità malevole di accedere alla rete.

La tesi procederà come segue: in primis un'introduzione teorica sullo spazio fisico collegato al gioco. Obiettivo dell'agente è infatti di massimizzare l'SNR attraverso un canale AWGN in presenza del jammer, che necessità di importanti semplificazioni per essere analizzata tramite le IA. Successivamente, la descrizione in teoria dei giochi degli spazi da analizzare, fondamentale per formalizzare la successiva descrizione delle azioni delle IA negli spazi di gioco. A seguire, la discussione dei due differenti approcci di IA per operare nelle ipotesi formalizzate precedentemente. Infine, i risultati verranno discussi e commentati, avanzando proposte di lavori futuri per espandere ulteriormente l'analisi in ambito di jamming dinamico.

Capitolo 2

Modello fisico

2.1 Capacità di canale

Prima di procedere con l'analisi computazionale, è necessario sviluppare un modello fisico su cui poter definire i giochi da esaminare.

Dalla teoria dell'informazione, possiamo modellare la comunicazione tra trasmettitore e ricevitore attraverso un canale rumoroso come un sistema statistico, descritto dall'informazione mutua per simbolo $I_s(t, r)$ tra messaggio inviato t e messaggio ricevuto r.

Detto F il tasso di simbolo, possiamo definire il tasso di informazione R attraverso un canale:

$$R(t) = FI_s(t, r) \tag{2.1}$$

E' necessario massimizzare la quantità di informazione comunicata attraverso il canale, si definisce quindi la capacità di canale C secondo Shannon [11]:

$$C = \max_t R(t) \tag{2.2}$$

Ipotizziamo ora di essere in presenza di un canale memoryless con rumore gaussiano bianco (canale AWGN), descritto in figura 2.1.

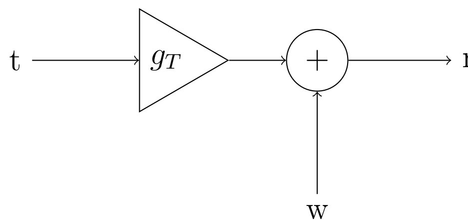

Figura 2.1

In assenza di memoria, possiamo scrivere $I_s(t,r) = I(t,r)$. Per massimizzare la capacità, scegliamo t gaussiano bianco a media nulla, r quindi è somma di due gaussiani a media nulla. Risulta:

$$\max_t I(t,r) = [\cdots] = \max_t \frac{1}{2}\log_2(1+\Gamma) \tag{2.3}$$

Con Γ l'SNR attraverso il canale. Allora:

$$C = \max_t R(t) = \max_t F I(t,r) = \frac{1}{T}\frac{1}{2}\log_2(1+\Gamma) \tag{2.4}$$

Ponendo la banda $B = \frac{1}{2T}$ (limite per Nyquist [12]), possiamo concludere:

$$C = B\log_2(1+\Gamma) \tag{2.5}$$

2.2 Capacità di canale in presenza di jammer

La presenza di un jammer che disturba la comunicazione può essere modellata come l'aggiunta di un'ulteriore segnale trasmesso j che funge da rumore ulteriore per il canale:

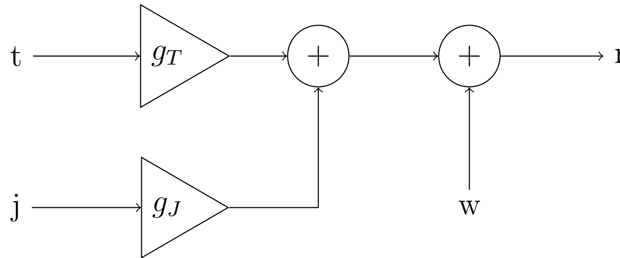

Figura 2.2

Supponendo che il rumore generato dal jammer sia preponderante rispetto al rumore del canale come in [4], si ottiene infine il modello di figura 2.3,

Al posto dell'SNR Γ possibile quindi valutare l'SNJR (Signal to Noise Ratio in presenza di Jammer) Γ_J:

$$\Gamma_J = \frac{g_R P_{tx}}{g_J P_J} \tag{2.6}$$

Ottenendo:

$$C = B\log_2(1+\Gamma_J) \tag{2.7}$$

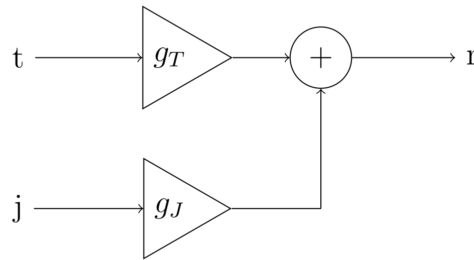

Figura 2.3

2.3 Parametri di analisi

Ai fini dell'analisi, è necessario semplificare l'equazione 2.7 per ottenere un parametro u (tale che $\Gamma_J \propto u$) da massimizzare, calcolabile con basso costo computazionale.

Innanzitutto consideriamo l'efficienza spettrale $\frac{C}{B}$, in luogo della capacità del canale, o equivalentemente consideriamo la banda B unitaria. Essendo $\log(1+x)$ monotona crescente, è possibile quindi studiare unicamente l'SNJR Γ_J definita nell'equazione 2.6. Supponiamo infine equivalenti le potenze di trasmissione del trasmettitore e del jammer, ottenendo:

$$u = \frac{g_R}{g_J} \tag{2.8}$$

Si possono considerare ora le funzioni g_R, g_T:

$$g_T \propto d(T,U)^{-\alpha} \quad , \quad g_J \propto d(J,U)^{-\alpha} \tag{2.9}$$

con $d(X,Y)$ la distanza tra trasmettitore X e ricevitore Y, T il trasmettitore del sistema, U il ricevitore e J il jammer. Assumendo di essere in condizioni di attenuazione di spazio libero, poniamo $\alpha = 2$.

A meno di moltiplicazioni per costanti, per valutare la SNJR è possibile quindi considerare:

$$u = \frac{d(U,J)^2}{d(U,T)^2} \tag{2.10}$$

2.4 Spazio di gioco

Per facilitare la computazione dei parametri, possiamo definire uno spazio infinito discretizzato come in griglia di figura 2.4, centrato nel trasmettitore. Definiamo

quindi la funzione $d(X, Y)$ come la distanza euclidea tra i punti X(x,y) e Y(x,y) identificati dalle coordinate della griglia.

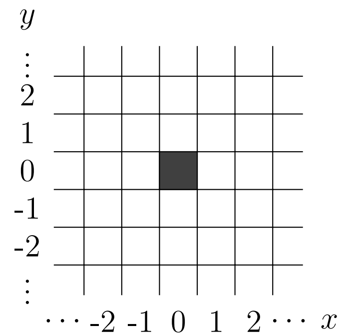

Figura 2.4: Discretizzazione dello spazio di gioco. Il punto $(0, 0)$ è occupato dal trasmettitore

Nei giochi che verranno analizzati, questo spazio verrà ulteriormente limitato come descritto nel capitolo 3.

Capitolo 3

Formalizzazione del gioco

3.1 Introduzione teorica

Per trasformare l'analisi del modello fisico in un analisi computazionale delle strategie migliori per agente e jammer, è necessario formalizzare il sistema attraverso un modello di gioco [1]. Tramite la teoria dei giochi, infatti, è possibile modellare e analizzare strutture in cui più elementi, chiamati giocatori, interagiscono tra di loro tentando di massimizzare dei parametri attuando diverse strategie.

Possiamo distinguere diversi tipi di giochi, in particolare qui analizzeremo giochi dinamici tra due giocatori a somma zero e informazione incompleta, in quanto l'analisi delle principali caratteristiche dei corrispondenti giochi statici è stata effettuata in [4] e [5].

Nello specifico, sono giochi nei quali:

- l'interazione tra i giocatori avviene in maniera dinamica, con un susseguirsi di stati che determinano l'evolversi delle strategie;
- il profitto di un giocatore è equivalente alla perdita del giocatore antagonista;
- i giocatori non possiedono informazioni complete relativamente allo stato corrente.

3.2 Il gioco dinamico

L'interazione tra utente e jammer può essere formalizzata come un gioco con le seguenti assunzioni:

- U e J sono giocatori che possono occupare una posizione nello spazio, lo spazio delle azioni viene denotato con $\mathcal{A}_U = \mathcal{A}_J = \mathcal{A}$;

- ad ogni step del gioco, i giocatori possono compiere un movimento simultaneo per spostarsi nello spazio, l'elenco delle mosse possibili viene denotato con $\mathcal{M}_U = \mathcal{M}_J = \mathcal{M}$;

- ogni interazione tra U e J risulta in un profitto $u_U = u$ per U, equivalente al valore u ricavato nell'equazione 2.10, e un profitto $u_J = -u_U$ per J;

- i giocatori possiedono informazioni dello stato corrente limitate, corrispondenti alla propria posizione (non quella dell'avversario) e al profitto attuale.

Le posizioni dei giocatori nello stato iniziale del gioco vengono stabilite casualmente tra le posizioni possibili nello spazio con una distribuzione uniforme.

3.3 Spazio 1x5

Lo spazio delle azioni è $\mathcal{A} = \{0, 1, \cdots, 4\}$ corrispondenti alle posizioni in figura 3.1.

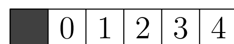

Figura 3.1

Le mosse disponibili ai due giocatori sono $\mathcal{M} = \{$

1. Muoversi di una casella a sinistra,

2. Muoversi di una casella a destra,

3. Rimanere nella stessa posizione $\}$

Il numero di stati possibili per il gioco è $|\mathcal{A}|^2 = 5^2 = 25$ e il profitto massimo possibile per l'agente è facilmente dimostrabile essere $u_U = 16$, corrispondente a U posizionato in 0 e J in 4.

Si tratta dello spazio di gioco più semplice qui analizzato (preso da [4]) che modella una situazione in cui i giocatori sono posizionati lungo una strada.

3.4 Spazio 3x3

Lo spazio delle azioni è $\mathcal{A} = \{0, 1, \cdots, 7\}$ corrispondenti alle posizioni in figura 3.2.

0	1	2
7	■	3
6	5	4

Figura 3.2

Le mosse disponibili ai due giocatori sono $\mathcal{M} = \{$

1. muoversi di una casella in senso antiorario,
2. muoversi di una casella in senso orario,
3. rimanere nella stessa posizione $\}$

Il numero di stati possibili per il gioco è $|\mathcal{A}|^2 = 8^2 = 64$ e il profitto massimo possibile per l'agente è facilmente dimostrabile essere $u_U = 5$, corrispondente allo stato in cui U è disposto in una casella adiacente al trasmettitore e J su un vertice nel lato opposto dello spazio (per esempio 1 e 4).

Si tratta di uno spazio (preso da [5]) con un aumento di complessità di stati rispetto allo spazio 1x5, mantenendo comunque lo stesso numero di mosse disponibili. Un'interessante differenza è che i due giocatori possono sempre effettuare tre mosse in qualsiasi stato essi si trovino, a differenza dello spazio 1x5 in cui agli estremi vengono limitate a causa della non ciclicità dello spazio. Questo potrebbe avere conseguenze particolari nello sviluppo delle strategie per le intelligenze artificiali.

3.5 Spazio radiale

Lo spazio delle azioni è $\mathcal{A} = \{0, 1, \cdots, 11\}$ corrispondenti alle posizioni in figura 3.3.

Le mosse disponibili ai due giocatori sono $\mathcal{M} = \{$

1. muoversi in senso antiorario di una casella mantenendo la stessa distanza dal trasmettitore,

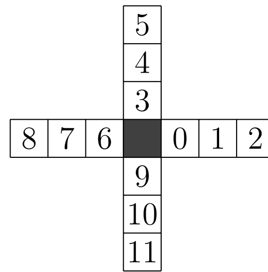

Figura 3.3

2. muoversi in senso orario di una casella mantenendo la stessa distanza dal trasmettitore,
3. allontanarsi dal trasmettitore,
4. avvicinarsi al trasmettitore,
5. rimanere nella stessa posizione }

Il numero di stati possibili per il gioco è $|\mathcal{A}|^2 = 12^2 = 144$ e il profitto massimo possibile per l'agente è facilmente dimostrabile essere $u_U = 16$, relativo allo stato in cui U si trova in una casella adiacente al trasmettitore e J si trova nella casella più lontana dal trasmettitore e da U lungo lo stesso asse (per esempio 0 e 8).

Si tratta dello spazio di gioco più complesso qui analizzato, che unisce le caratteristiche dei due spazi precedenti, ampliando notevolmente le possibilità di movimento concesse ai due giocatori.

Capitolo 4

Progettazione dell'intelligenza artificiale

4.1 Q-learning

In primo luogo, sviluppiamo l'approccio di apprendimento tramite Q-learning, un approccio di tipologia reinforcement learning per l'apprendimento automatico. L'algoritmo si basa sull'aggiornamento ricorsivo della funzione $Q(a, s)$ che restituisce una previsione del rendimento ottenuto effettuando una determinata mossa $a \in \mathcal{M}$ nello stato $s \in \mathcal{A}$. Nello specifico, è strutturato come segue:

1. viene scelta la mossa a_t che massimizza la funzione Q nello stato corrente s_t;

2. dopo aver mosso entrambe le entità, viene calcolato il rendimento ottenuto r dallo stato s_{t+1};

3. viene aggiornata la funzione Q secondo la seguente formula:

$$Q(a_t, s_t) = (1 - \rho)Q(a_t, s_t) + \rho \left(r + \gamma \max_a Q(a, s_{t+1})\right) \tag{4.1}$$

L'efficacia dell'algoritmo anche in questi giochi dinamici, che complicano la convergenza alla strategia migliore a causa della modifica continua dello scenario di gioco da parte di un'entità avversa, è stata già mostrata in [4] e [5].

4.2 Reti neurali

Prima di procedere con lo sviluppo della rete neurale, introduciamo i concetti principali di questo approccio. Partiamo dalla rete più semplice, la feedforward neural network [13], così chiamata perchè formata da più layer collegati uno dopo l'altro. Il singolo neurone del sistema è mostrato in figura 4.1 ed è un'entità così descritta:

- riceve n valori di input $[x_1, \cdots, x_n] = \mathbf{x}$;
- determina la somma $\sum_{i=1}^{n} w_i \cdot x_i + b = \mathbf{w} \cdot \mathbf{x} + b = y$, con $\mathbf{w}$ il vettore dei pesi e b il bias;
- restituisce in output $f(y)$, dove f è una funzione non lineare scelta a priori.

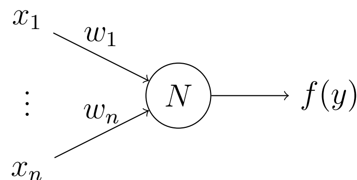

Figura 4.1: Singolo neurone di una rete neurale

Per formare un layer della rete, più neuroni vengono collegati in parallelo agli stessi input come in figura 4.2.

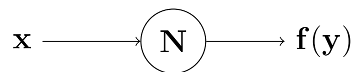

Figura 4.2: Singolo layer di una feedforward neural network. $\mathbf{N}$ rappresenta un vettore di neuroni

Infine, più layer possono essere connessi in serie, in tal modo ogni neurone di un layer possiede l'input collegato agli output di tutti i neuroni del layer precedente e l'output collegato a tutti gli input dei neuroni del layer successivo, come schematizzato in figura 4.3.

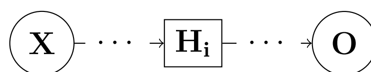

Figura 4.3: Struttura di un feedforward neural network. $\mathbf{X}, \mathbf{H_i}$ e $\mathbf{O}$ rappresentano vettori di neuroni

In questo caso si possono distinguere tre tipologie di layer:

1. il layer di input **X**, nel quale i valori di output dei neuroni vengono settati manualmente a partire dalle caratteristiche dell'input che si vuole analizzare;

2. i layer intermedi **H**, detti hidden layer;

3. il layer di output **O**, le cui uscite vengono utilizzate per determinare i parametri decisionali della rete in relazione all'input inserito.

4.3 Processo di apprendimento

Le reti così costruite permettono di associare ad ogni input un output tramite la combinazione di pesi e bias dei neuroni, l'obiettivo è modificare questi valori attraverso un processo di apprendimento automatico per ottenere l'approssimazione migliore della funzione cercata.

In questo caso, si è scelto di usare una tecnica di apprendimento supervisionato, cioè l'uscita della rete ad ogni iterazione viene comparata con l'output desiderato a partire dallo stato corrente per determinare come modificare i parametri della rete tramite la retropropagazione dell'errore. Per fare ciò, sono necessari due modelli:

- una funzione di perdita, che a partire dall'output ottenuto e quello desiderato, quantifica l'errore commesso dalla rete;

- un algoritmo di ottimizzazione, che a partire dall'errore calcolato modifica i pesi e bias della rete per minimizzarlo.

La rete viene quindi interfacciata con un numero di coppie (stato,uscita desiderata) considerevole, per tentare di minimizzare il più possibile l'errore di output nell'insieme di apprendimento.

Infine, la rete viene testata in assenza di retropropagazione dell'errore contro un insieme di stati non presenti nell'insieme di apprendimento, per valutarne il rendimento in una generalizzazione del problema.

4.4 Reti neurali ricorrenti

Per l'approccio tramite rete neurale, si è scelto di usare una RNN (Recurrent Neural Network), cioè una rete neurale che mantiene una memoria dello stato precedente per influenzare le scelte nello stato corrente

Negli ultimi anni la ricerca nell'ambito delle RNN si è concentrata soprattutto sulle LSTM (Long Short-Term Memory) [14], in particolare per problemi di linguistica computazionale [15] [16] [17]. Si è scelto di non optare per questa soluzione perché la limitata numerosità di stati da analizzare dei giochi in esame non è comparabile con i problemi risolti dalle LSTM e comporterebbe un notevole aumento di complessità computazionale per la rete.

L'architettura di RNN invece scelta è la rete Elman [18], la più semplice applicazione di una RNN costituita da tre layer di uguale dimensione: un layer di input, uno di output e un hidden layer intermedio. Tra uno step e l'altro vengono memorizzati gli stati relativi all'hidden layer prima che avvenga la retropropagazione dell'errore, questi vengono utilizzati come input ulteriori per l'hidden layer nello step successivo, espandendo i neuroni del layer di input. E' possibile schematizzare la struttura neurale nel tempo come in figura 4.4.

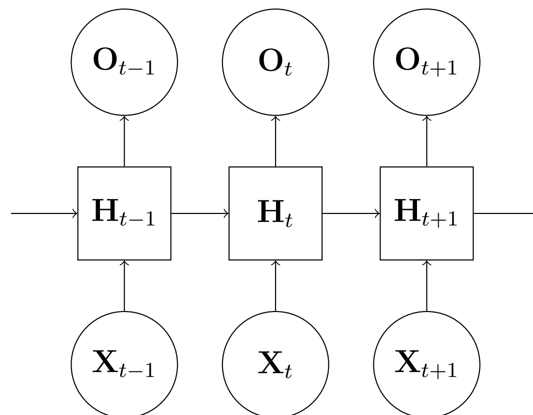

Figura 4.4: Schema dell'evoluzione di un RNN nel tempo. $\mathbf{X}_t, \mathbf{H}_t, \mathbf{O}_t$ rappresentano i layer della rete allo step t

4.5 L'architettura completa

L'IA tramite rete neurale viene quindi così strutturata su 3 layer:

1. Un layer di input di dimensione $|\mathcal{A}|$;
2. L'hidden layer del RNN di dimensione $|\mathcal{A}|$;
3. Un layer di output di dimensione $|\mathcal{M}|$ per convertire l'output del RNN nelle mosse disponibili ai giocatori.

L'input della rete mappa il gioco ad ogni iterazione nel seguente modo:

- il neurone relativo alla posizione del giocatore ha valore equivalente al suo profitto nello stato corrente;
- gli altri neuroni hanno valore -1.

I neuroni dell'output della rete invece corrispondono alla confidenza con cui l'IA ritiene che la relativa mossa sia quella migliore nello stato corrente.

L'avanzamento della rete avviene tramite apprendimento supervisionato secondo il seguente procedimento:

1. viene scelta la mossa ritenuta migliore dalla rete nello stato corrente;
2. dopo aver mosso entrambe le entità, viene calcolata la mossa che avrebbe restituito il profitto migliore, supponendo di conoscere il movimento effettuato dall'entità antagonista;
3. a partire dalla mossa determinata, viene calcolata la funzione di perdita relativa all'uscita della rete rispetto all'output così strutturato:
    - il neurone relativo alla mossa da attuare ha valore 1;
    - gli altri neuroni hanno valore 0;
4. viene effettuata la retropropagazione dell'errore e l'ottimizzazione della rete sulla base della funzione di perdita calcolata.

La funzione di perdita scelta è l'errore quadratico medio e l'algoritmo di ottimizzazione è l'RMSprop, comunemente usato nell'ambito delle reti neurali [19].

Questa struttura di rete dovrebbe permettere due vantaggi principali rispetto all'apprendimento tramite reinforcement learning:

- la possibilità di reagire allo stato corrente con l'influenza dello stato precedente, fondamentale in un gioco di mosse sequenziali come quelli da analizzare;
- la visione dell'IA di uno stato "analogico" che può risultare critica nell'intuizione della posizione dell'entità antagonista. L'informazione aggiuntiva sarebbe impossibile da implementare tramite Q-learning, in quanto implicherebbe la memorizzazione di una matrice Q di dimensione infinita.

Al contempo, l'apprendimento supervisionato potrebbe risultare negativamente nel rendimento nel test non supervisionato, in quanto potrebbe non permettere alla rete di adattarsi a delle tattiche di gioco variabili.

4.6 Configurazione di apprendimento

Le due IA verranno allenate per 7×10^5 iterazioni totali, resettando lo stato del gioco ad una posizione casuale per entrambi i giocatori ogni 10^4 iterazioni (per evitare lo stabilirsi di loop infiniti nelle mosse dei due giocatori che impedirebbero l'analisi completa delle tattiche attuabili).

Per entrambi i sistemi, si è scelto di adottare una politica di esplorazione (le mosse attuate vengono scelte o casualmente tra le mosse legali nello stato corrente con probabilità ε o seguendo la mossa ritenuta migliore dall'IA con probabilità $(1 - \varepsilon)$ attuata come segue e rappresentata in figura 4.5:

1. $\varepsilon = 1$ per le prime 10^5 iterazioni;
2. ε con decadimento esponenziale fino a $\varepsilon = 0.01$ durante le 5×10^5 iterazioni successive;
3. $\varepsilon = 0.01$ per le ultime 10^5 iterazioni (fase di test).

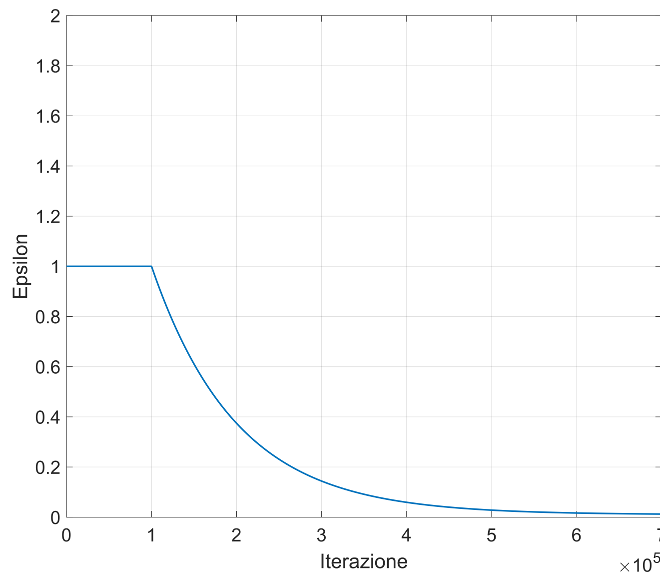

Figura 4.5: Andamento di ε in funzione dell'iterazione corrispondente

Ogni iterazione avverrà nel seguente modo:

1. lo stato del sistema verrà passato alle due IA secondo le rispettive modalità;
2. le due entità muoveranno attraverso la mossa scelta dalle IA corrispondenti in base alla politica di esplorazione corrente;

3. il nuovo stato del sistema con il relativo profitto per le due entità verrà calcolato e passato alle IA per procedere con i rispettivi algoritmi di apprendimento.

Per valutare l'efficacia dell'IA così istruita verranno utilizzate le ultime 10^5 iterazioni, in particolare non verrà più effettuato il processo di apprendimento della rete neurale per poterne valutare il profitto in situazioni reali non supervisionate.

Per permettere la valutazione oggettiva del rendimento dei due differenti approcci, si è scelto di introdurre un'ulteriore entità con politica totalmente esplorativa ($\varepsilon = 1$) durante tutte le iterazioni di apprendimento. Definendo con R quest'ultima, N l'IA tramite rete neurale e Q l'IA tramite Q-learning, per ogni spazio di gioco verranno simulati i processi di apprendimento relativi a tutte le permutazioni possibili delle tre entità nel ruolo di agenti e di jammer.

I processi che coinvolgono R verranno poi usati come benchmark per valutare il comportamento delle IA, in quanto un profitto medio inferiore a quello ottenuto con solo mosse casuali indicherebbe un'applicazione controproducente dell'analisi computazionale.

Capitolo 5

Risultati

5.1 Metodo di analisi

Nei seguenti grafici verranno tracciati i rendimenti degli agenti (ricordando che i rendimenti dei jammer sono uguali in modulo con segno opposto) durante il processo di apprendimento nelle iterazioni da 1×10^5 a 6×10^5 (*training*), processati attraverso una media mobile con una finestra di dimensione 5×10^4, e i rendimenti mediati sulle ultime 10^5 iterazioni (*test*), per confrontare l'efficacia dei diversi approcci.

Per analizzare i dati, verranno considerate le coppie agente - jammer con in comune una delle due entità, valutando la resa delle IA secondo le seguenti regole:

- A parità di agente, il jammer migliore sarà quello che minimizza il rendimento medio;
- A parità di jammer, l'agente migliore sarà quello che massimizza il rendimento medio.

Le coppie verranno indicate come $X - Y$, con X l'entità agente e Y l'entità jammer, secondo la nomenclatura N, Q, R descritta nella sezione 4.6.

5.2 Gioco 1x5

I risultati del gioco 1x5 sono mostrati nelle figure 5.1, 5.2, 5.3.

Contro l'entità R, entrambe le IA traggono vantaggio sia nel ruolo di jammer che di agente. In entrambi i casi, l'entità Q risulta migliore di N. Ponendo le IA una contro l'altra, si evidenziano dei risultati particolari:

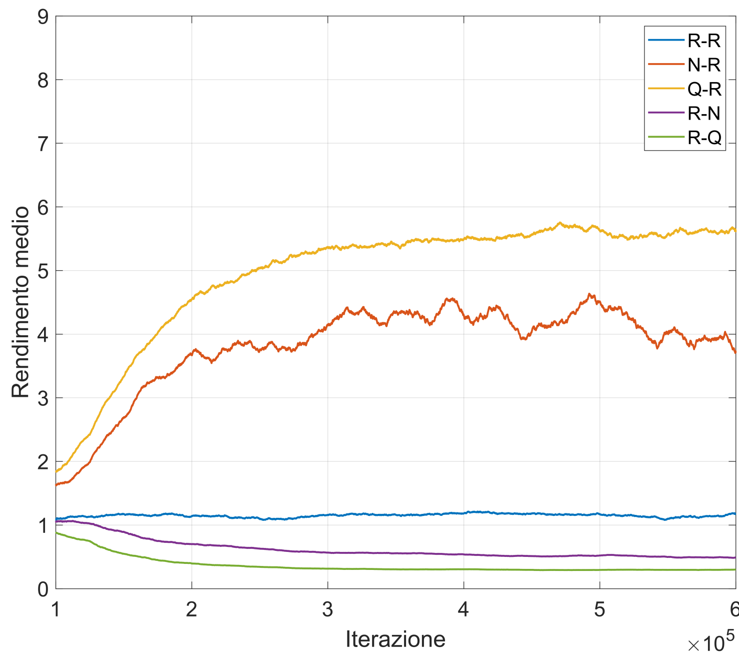

Figura 5.1: Training delle IA contro l'entità R nel gioco 1x5

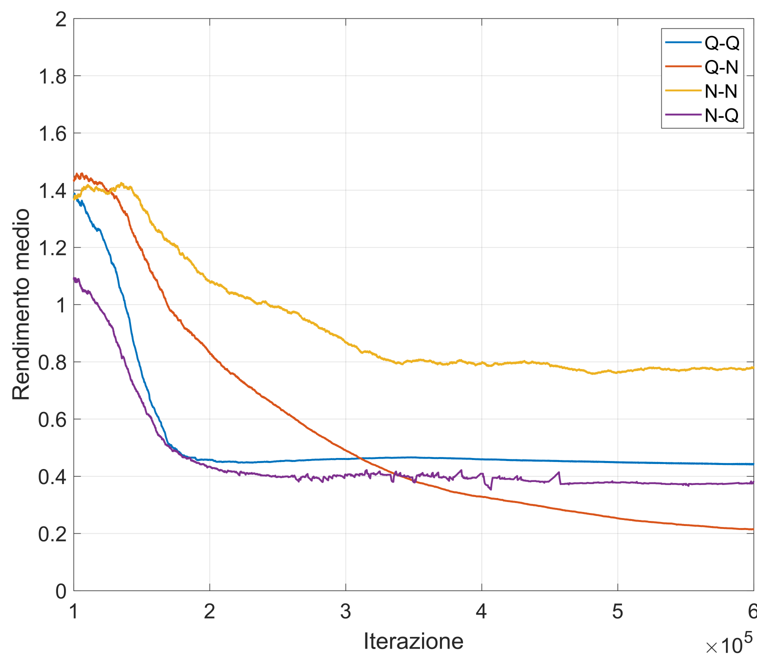

Figura 5.2: Training mutuo delle IA nel gioco 1x5

- Contro i jammer N e Q, risulta migliore l'agente Q

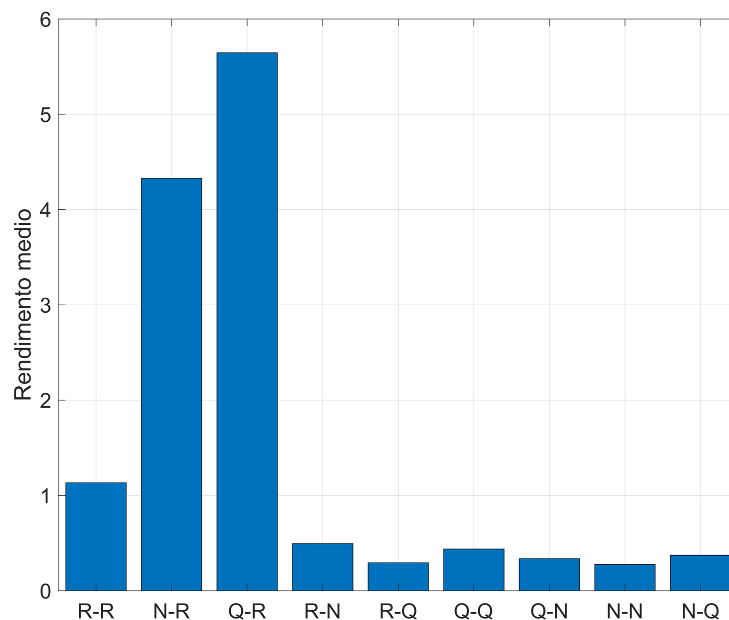

Figura 5.3: Risultati di test delle IA nel gioco 1x5

- Contro gli agenti Q e N, risulta migliore il jammer N

Non risulta quindi un IA preponderante rispetto all'altra.

5.3 Gioco 3x3

I risultati del gioco 3x3 sono mostrati nelle figure 5.4, 5.5, 5.6.

Contro l'entità R, entrambe le IA traggono vantaggio sia nel ruolo di jammer che di agente. In entrambi i casi, l'entità N risulta migliore di Q. Ponendo le IA una contro l'altra, sia nel ruolo di agente che nel ruolo di jammer risulta migliore l'entità N.

5.4 Gioco radiale

I risultati del gioco radiale sono mostrati nelle figure 5.7, 5.8, 5.9.

Contro l'entità R, entrambe le IA traggono vantaggio sia nel ruolo di jammer che di agente. E' interessante come nel ruolo di agente risulti migliore l'entità N, mentre nel ruolo di jammer risulti migliore l'entità Q. Ponendo le IA una contro

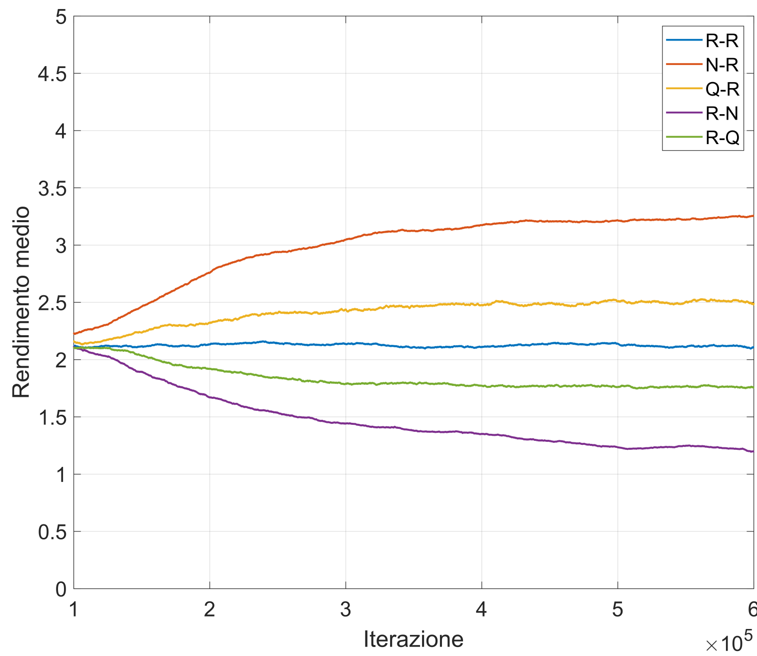

Figura 5.4: Training delle IA contro l'entità R nel gioco 3x3

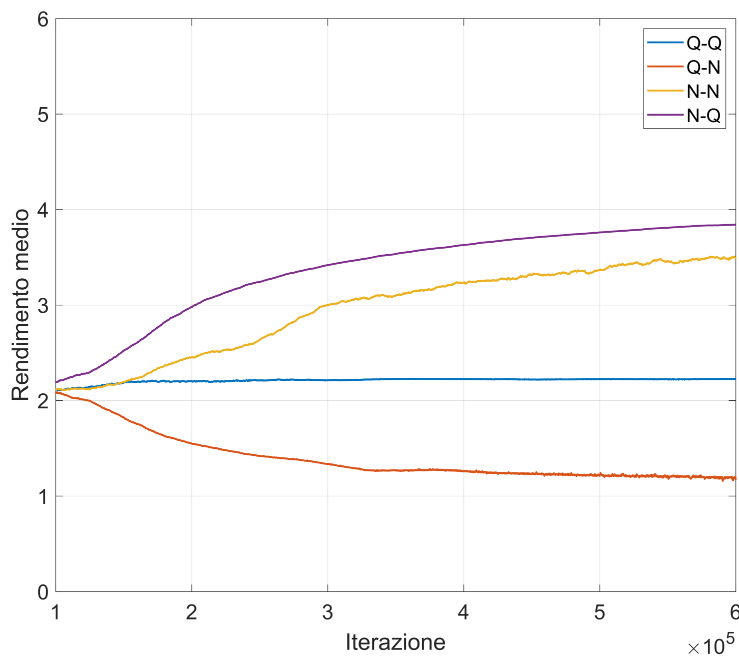

Figura 5.5: Training mutuo delle IA nel gioco 3x3

l'altra, sia nel ruolo di agente che nel ruolo di jammer risulta migliore l'entità N come nel caso 3x3.

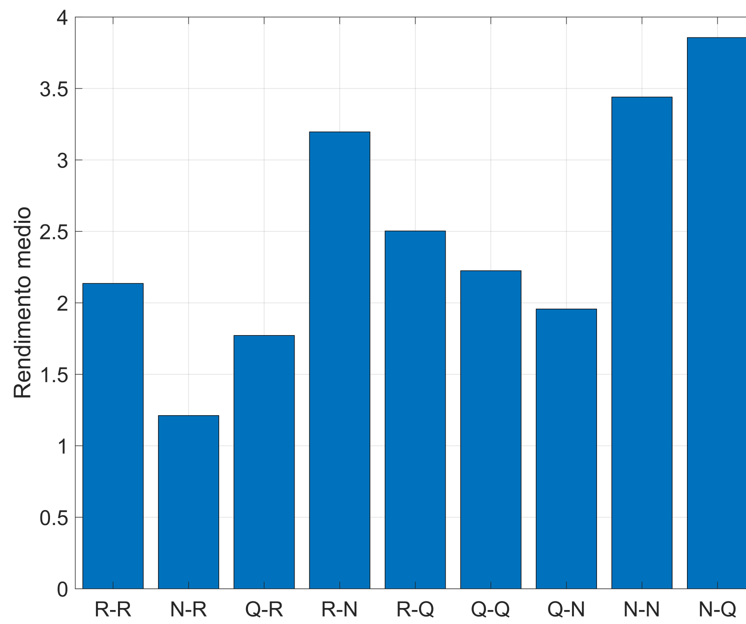

Figura 5.6: Risultati di test delle IA nel gioco 3x3

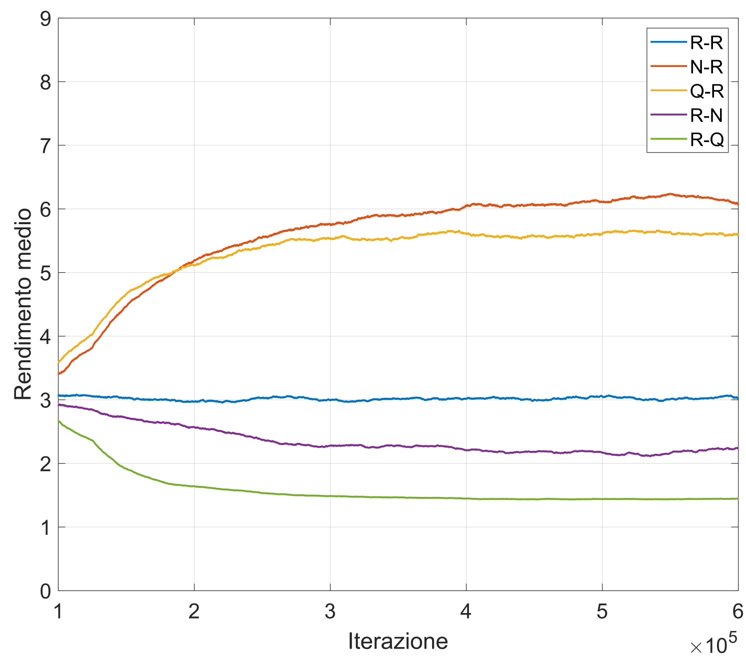

Figura 5.7: Training delle IA contro l'entità R nel gioco radiale

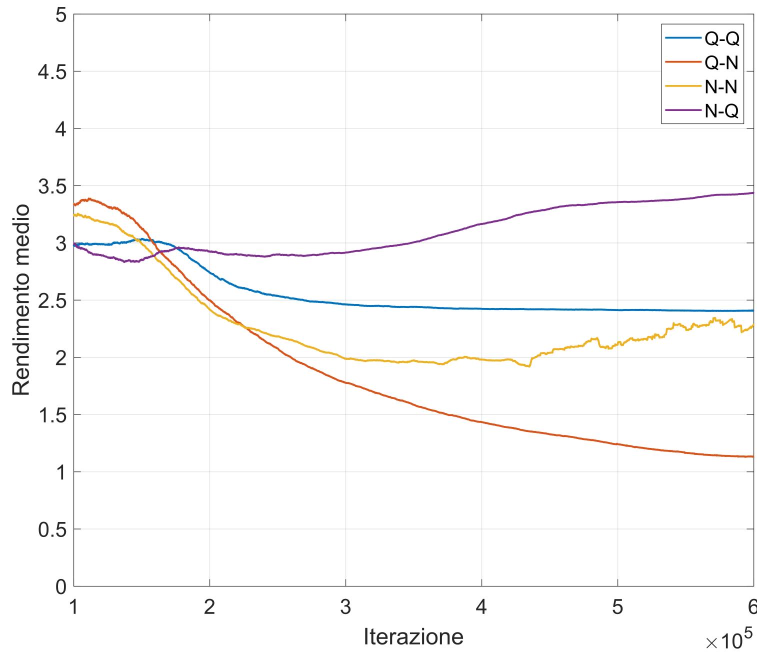

Figura 5.8: Training mutuo delle IA nel gioco radiale

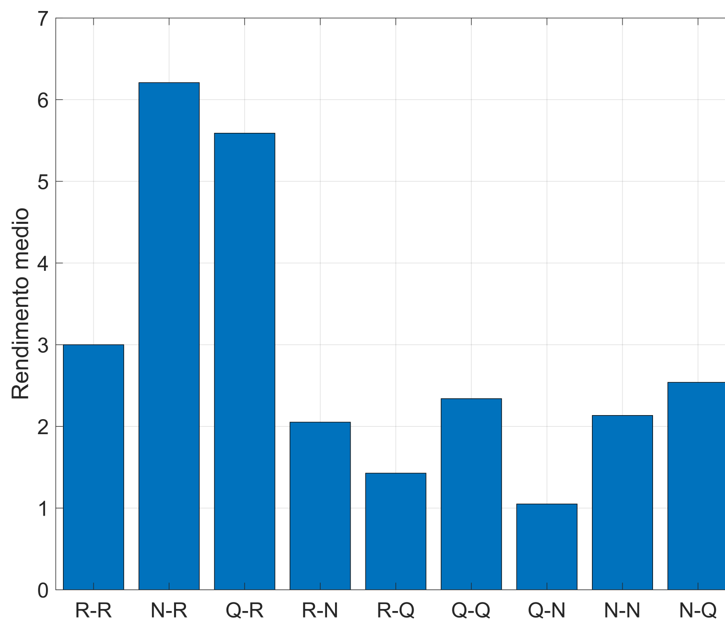

Figura 5.9: Risultati di test delle IA nel gioco radiale

Capitolo 6

Conclusioni

I risultati ottenuti in questa tesi tramite l'utilizzo delle reti neurali, confermano la validità dell'innovativo approccio utilizzato per studiare le configurazioni di gioco qui descritte.

Infatti, comparando i rendimenti delle due IA si possono trarre le seguenti considerazioni:

- l'approccio tramite IA risulta sempre vantaggioso, sia per l'agente che per il jammer;
- in generale, l'utilizzo della rete neurale consente risultati paragonabili e in alcuni casi anche migliori rispetto all'apprendimento tramite reinforcement learning.

La rete neurale ricorrente applicata a questo problema quindi ha dato buoni risultati nella ricerca delle strategie di gioco migliori, ma urge uno studio più approfondito sulla struttura di rete e di apprendimento ottimale. In particolare, l'effettivo vantaggio conferito dalla presenza di memoria nella rete e una possibile estensione dell'apprendimento per renderlo automatico senza supervisione, garantendo un adattamento all'entità avversaria in ogni situazione.

Sorge comunque spontaneo chiedersi gli sviluppi futuri per espandere le soluzioni trovate in ambiti di analisi diversi.

In primo luogo, dai dati si può ipotizzare un'aumento della differenza di rendimento tra rete neurale e Q-learning al crescere della complessità delle mosse e dello spazio delle azioni, ipotesi che potrebbe essere studiata ampliando ulteriormente lo spazio di gioco all'interno dello spazio discretizzato.

Un approccio simile potrebbe essere applicato all'analisi di un ampliamento completo dello spazio, rendendolo infinito, particolarmente complesso se non

impossibile da analizzare tramite Q-learning, ma generalizzabile con qualche accorgimento alla struttura di RNN utilizzata in questa tesi.

Un'altra possibilità è quella di ampliare le entità in gioco, facendo interagire un agente con jammer multipli, estremamente complicato da analizzare teoricamente ma simulabile con le tecniche qui discusse.

Ancora, è interessante analizzare la diversità di strategie ottenibili dalla rete variando le informazioni proposte al giocatore, per esempio in funzione della sua vicinanza o meno al giocatore avversario entro un certo raggio, oppure della presenza effettiva o meno di agenti avversari.

La ricerca in merito non è quindi affatto conclusa, e certamente i risultati di ricerche che ora affrontano i problemi qui proposti, e quelli a loro affini, possono risultare fondamentali nello studio di innovative strategie di comunicazione diverse nelle nuove generazioni di sistemi di trasmissione.

Capitolo 7

Codice

Per sviluppare il software, si è scelto di utilizzare Python con la libreria pytorch per la sua facilità di utilizzo e il suo estensivo uso nella comunità scientifica.

Vengono qui illustrati i codici relativi all'analisi dello spazio 3x3.

```
1  class SimpleRNN(nn.Module):
2      #Classe del RNN
3      def __init__(self, inSize, hidSize, numLayers, actions):
4          self.actions = actions
5          self.hidSize = hidSize
6          self.numLayers = numLayers
7  
8          super(SimpleRNN, self).__init__()
9          #Struttura del RNN
10         self.rnn1 = nn.RNN(inSize, hidSize, numLayers,
   batch_first=True,nonlinearity='tanh')
11         self.out = nn.Linear(hidSize, len(actions))
12         self.hid = torch.zeros(numLayers,1,hidSize)
13 
14     def resetHidden(self):
15         #Reset dell'hidden layer
16         self.hid = torch.zeros(self.numLayers,1,self.hidSize)
17 
18     def forward(self, x):
19         #Output del nn a partire dall'hidden layer precedente hid
    e dall'input x
20         self.hid = self.hid.detach()
21         x, self.hid = self.rnn1(x, self.hid)
22         return self.out(x)
```

Listing 7.1: RNN.py

```
#Inizializzazione costanti
N_TEST = "21"

SIZE = 8
MAXPO = 5

HIDSIZE = SIZE
HIDLAYERS = 1
ACTIONS = ('cc', 'stay', 'cw')

EPS_START = 0.9
GAMMA = 0.9
ALPHA = 2

EPS_START = 1
EPS_END = 0.01
EPS = EPS_START

EPOCH1 = 1e5
EPOCH2 = 6e5
EPOCHS = 700_000

Aloss_plot = []
Jloss_plot = []
avg_payoff = []


class Point():
    #Classe punto
    def __init__(self,x,y):
        self.x = x
        self.y = y
    #Distanzza Euclidea
    def dist(self,p):
        return sqrt((self.x-p.x)**2 + (self.y-p.y)**2)
    #Da punto a indice dell'array
    def toList(self):
        if self.y == 0:
            return self.x
        elif self.y == 1:
            if self.x == 2: return 3
            elif self.x == 1: return 7
        elif self.y == 2:
            return 6 - self.x
```

```
45
46 TX = Point(1,1) #Posizione trasmettitore
47
48 ### POINT ###
49 # 0,0 1,0 2,0
50 # 0,1 1,1 2,1
51 # 0,2 1,2 2,2
52
53 ### ARRAY ###
54 #  0 1 2
55 #  7 _ 3
56 #  6 5 4
57 # SIZE = 8
58
59 #Da indice array a punto
60 def toPoint(l):
61     if l < 3: return Point(l,0)
62     elif l == 3: return Point(2,1)
63     elif l == 7: return Point(0,1)
64     elif l > 3: return Point(6-l,2)
65
66 #Rendimento con le posizioni xa, xj per i giocatori
67 def payoff(xa,xj):
68     pa = toPoint(xa)
69     pj = toPoint(xj)
70     return (pa.dist(pj)/(pa.dist(TX)))**ALPHA
71
72 #Azioni possibili nella posizione corrispondente all'indice l
73 def legalActions(l):
74         if (l == 0):
75             return [7,0,1]
76         elif (l == 7):
77             return [6,7,0]
78         return [l-1,l,l+1]
79
80 #Azione migliore per l'agente con le posizioni xa, xj per i
    giocatori
81 def bestAgent(xa,xj):
82     actions = legalActions(xa)
83     bestPO = 0
84     bestAction = torch.zeros(3)
85     bestNextState = xa
86     for i in actions:
87         po = payoff(i,xj)
```

```
88          if po > bestPO:
89              bestPO = po
90              bestNextState = i
91      aX = bestNextState - xa + 1
92      if aX == 8: aX = 0
93      if aX == -6: aX = 2
94      bestAction[aX] = 1
95      return bestAction
96
97  #Azione migliore per il jammer con le posizioni xa, xj per i
        giocatori
98  def bestJammer(xa, xj):
99      actions = legalActions(xj)
100     bestPO = 0
101     bestAction = torch.zeros(3)
102     bestNextState = xj
103     for i in actions:
104         po = MAXPO - payoff(xa, i)
105         if po > bestPO:
106             bestPO = po
107             bestNextState = i
108     aX = bestNextState - xj + 1
109     if aX == 8: aX = 0
110     if aX == -6: aX = 2
111     bestAction[aX] = 1
112     return bestAction
113
114 #Inizializzazione nn
115 agent = SimpleRNN(SIZE,HIDSIZE,HIDLAYERS,ACTIONS)
116 jammer = SimpleRNN(SIZE,HIDSIZE,HIDLAYERS,ACTIONS)
117 state = ''
118
119 #Algoritmo di ottimizzazione e funzione di perdita
120 learning_rate = 1e-3
121 Aoptim = optim.RMSprop(agent.parameters(), lr = learning_rate)
122 Joptim = optim.RMSprop(jammer.parameters(), lr = learning_rate)
123 Floss = nn.MSELoss(reduction='sum')
124
125 #Inizializzazione delle entita
126 ax = randint(0,7)
127 Aq = Qlearn(ax, SIZE, 0.01, 0.99)
128 An = Entity(ax)
129 Ar = Rand(ax)
130
```

```
131 jx = randint(0,7)
132 Jq = Qlearn(jx,SIZE,0.01,0.99)
133 Jn = Entity(jx)
134 Jr = Rand(jx)
135
136 Aq.updateState(payoff(Aq.x, Jq.x))
137 An.updateState(payoff(Aq.x, Jq.x))
138 Ar.updateState(payoff(Aq.x, Jq.x))
139 Jq.updateState(MAXPO - payoff(Aq.x,Jq.x))
140 Jn.updateState(MAXPO - payoff(Aq.x,Jq.x))
141 Jr.updateState(MAXPO - payoff(Aq.x,Jq.x))
142
143 states = []
144 preds = []
145
146 #Loop principale
147 for e in range(EPOCHS):
148     if (e % 10_000 == 0):
149         #RESET ogni 10000 iterazioni
150         print('RESET')
151         ax = randint(0,7)
152         Aq.resetState(ax)
153         An.resetState(ax)
154         Ar.resetState(ax)
155         jx = randint(0,7)
156         Jq.resetState(jx)
157         Jn.resetState(jx)
158         Jr.resetState(jx)
159         Aq.updateState(payoff(Aq.x, Jq.x))
160         An.updateState(payoff(Aq.x, Jq.x))
161         Ar.updateState(payoff(Aq.x, Jq.x))
162         Jq.updateState(MAXPO - payoff(Aq.x,Jq.x))
163         Jn.updateState(MAXPO - payoff(Aq.x,Jq.x))
164         Jr.updateState(MAXPO - payoff(Aq.x,Jq.x))
165         agent.resetHidden()
166         jammer.resetHidden()
167
168     #Funzione eps
169     if (e > EPOCH2):
170         EPS = EPS_END
171     elif (e > EPOCH1):
172         EPS = (EPS_START-EPS_END)*math.exp(-(e-EPOCH1)/EPOCH1)+
    EPS_END
173
```

```
174     #Predizione delle IA
175     An.pred = agent(Aq.state.view(-1,1,8))
176     Aq.pred = An.pred
177     Ar.pred = An.pred
178     Jn.pred = jammer(Jq.state.view(-1,1,8))
179     Jq.pred = Jn.pred
180     Jr.pred = Jn.pred
181 
182     #Movimento dei giocatori
183     if (random() > EPS):
184         a = 'n' # per agente Q a = 'q', per agente N a = 'n', per
        agente R a = 'r'
185     else: a = 'r'
186 
187     if (a == 'q'):
188         Aq.move()
189         An.movex(Aq.x)
190         Ar.movex(Aq.x)
191     elif (a == 'n'):
192         An.move()
193         Aq.movex(An.x)
194         Ar.movex(An.x)
195     elif (a == 'r'):
196         Ar.move()
197         Aq.movex(Ar.x)
198         An.movex(Ar.x)
199 
200     if (random() > EPS):
201         j = 'n' # per jammer Q j = 'q', per jammer N j = 'j', per
        jammer R j = 'r'
202     else:
203         j = 'r'
204 
205     if (j == 'q'):
206         Jq.move()
207         Jn.movex(Jq.x)
208         Jr.movex(Jq.x)
209     elif (j == 'n'):
210         Jn.move()
211         Jq.movex(Jn.x)
212         Jr.movex(Jn.x)
213     elif (j == 'r'):
214         Jr.move()
215         Jq.movex(Jr.x)
```

```
216            Jn.movex(Jr.x)
217
218        if (e < EPOCH2):
219            #Determinazione e retropropagazione dell'errore
220
221            An.correct = bestAgent(Aq.oldx, Jq.x)
222            Jn.correct = bestJammer(Aq.x, Jq.oldx)
223
224            Aloss = Floss(An.pred[0,0], An.correct)
225            Jloss = Floss(Jn.pred[0,0], Jn.correct)
226
227            Aoptim.zero_grad()
228            Joptim.zero_grad()
229
230            Aloss.backward()
231            Jloss.backward()
232
233            Aoptim.step()
234            Joptim.step()
235
236        #Aggiornamento dello stato
237        Aq.updateState(payoff(Aq.x, Jq.x))
238        An.updateState(payoff(Aq.x, Jq.x))
239        Ar.updateState(payoff(Aq.x, Jq.x))
240        Jq.updateState(MAXPO - payoff(Aq.x,Jq.x))
241        Jn.updateState(MAXPO - payoff(Aq.x,Jq.x))
242        Jr.updateState(MAXPO - payoff(Aq.x,Jq.x))
243
244        #Salvataggio di una mappa testuale della successioni di stati
245        map = ['_'] * 8
246        map[Aq.x] = 'A'
247        map[Jq.x] = 'J' if (map[Jq.x] == '_') else 'B'
248        states.append(map)
249        preds.append(An.pred[0,0].tolist())
250
251        #Debug dell'apprendimento
252        if e % 1000 == 0: print(e, Aloss.data)
```

Listing 7.2: train.py

```
1 #Azioni possibili nella posizione corrispondente all'indice l
2 def legalActions(l):
3         if (l == 0):
4             return [7,0,1]
5         elif (l == 7):
```

```
            return [6,7,0]
        return [l-1,l,l+1]

#Entita Q
class Qlearn():
    #Inizializzazione
    def __init__(self,x,L,p,y):
        self.Q = [[0] * L] * L
        self.pred = torch.zeros(3)
        self.correct = torch.zeros(3)
        self.x = x
        self.oldx = x
        self.state = torch.zeros(8)
        self.state.fill_(-1)
        self.p = p
        self.y = y

    #Azione migliore scelta da Q
    def best(self,actions):
        action = self.x
        ev = -1
        for a in actions:
            q = self.Q[self.x][a]
            if q > ev:
                ev = q
                action = a
        return action

    #Rendimento migliore da Q
    def bestEv(self, actions):
        ev = -1
        for a in actions:
            q = self.Q[self.x][a]
            if q > ev:
                ev = q
        return ev

    #Funzioni di movimento
    def move(self):
        self.oldx = self.x
        actions = legalActions(self.x)
        self.x = self.best(actions)
    def movex(self,nx):
        self.oldx = self.x
```

```
50          self.x = nx
51
52      #Reset dello stato
53      def resetState(self,x):
54          self.x = x
55          self.oldx = x
56          self.state = torch.zeros(8)
57          self.state.fill_(-1)
58          self.correct = torch.zeros(8)
59          self.pred = torch.zeros(3)
60
61      #Update dello stato
62      def updateState(self, payoff):
63          self.Q[self.oldx][self.x] = self.Q[self.oldx][self.x] *
        (1-self.p) + self.p * (payoff + self.y * self.bestEv(
        legalActions(self.x)))
64          self.state = torch.zeros(8)
65          self.state.fill_(-1)
66          self.state[self.x] = payoff
67
68  #Entita R
69  class Rand():
70      #Inizializzazione
71      def __init__(self,x):
72          self.pred = torch.zeros(3)
73          self.correct = torch.zeros(3)
74          self.x = x
75          self.oldx = x
76          self.state = torch.zeros(8)
77          self.state.fill_(-1)
78
79      #Funzioni di movimento
80      def move(self):
81          self.oldx = self.x
82          actions = legalActions(self.x)
83          self.x = choice(actions)
84      def movex(self,nx):
85          self.oldx = self.x
86          self.x = nx
87
88      #Reset dello stato
89      def resetState(self,x):
90          self.x = x
91          self.oldx = x
```

```
92          self.state = torch.zeros(8)
93          self.state.fill_(-1)
94          self.correct = torch.zeros(3)
95          self.pred = torch.zeros(3)
96
97      #Update dello stato
98      def updateState(self, payoff):
99          self.state = torch.zeros(8)
100         self.state.fill_(-1)
101         self.state[self.x] = payoff
102
103 #Entita N
104 class Entity():
105     #Inizializzazione
106     def __init__(self,x):
107         self.pred = torch.zeros(3)
108         self.correct = torch.zeros(3)
109         self.x = x
110         self.oldx = x
111         self.state = torch.zeros(8)
112         self.state.fill_(-1)
113
114     #Funzioni di movimento
115     def move(self):
116         self.oldx = self.x
117         self.x = self.x + torch.argmax(self.pred) - 1
118         if (self.x < 0): self.x = 7
119         elif (self.x > 7): self.x = 0
120
121     def movex(self,nx):
122         self.oldx = self.x
123         self.x = nx
124
125     #Reset dello stato
126     def resetState(self,x):
127         self.x = x
128         self.oldx = x
129         self.state = torch.zeros(8)
130         self.state.fill_(-1)
131         self.correct = torch.zeros(3)
132         self.pred = torch.zeros(3)
133
134     #Update dello stato
135     def updateState(self, payoff):
```

```
136         self.state = torch.zeros(8)
137         self.state.fill_(-1)
138         self.state[self.x] = payoff
```

Listing 7.3: utilities.py

Bibliografia

[1] L. A. DaSilva, H. Bogucka, and A. B. MacKenzie, *Game theory in wireless networks*, IEEE Commun. Mag., vol. 49, no. 8, pp. 110–111, 2011.

[2] E. Altman, K. Avrachenkov, and A. Garnaev, *A jamming game in wireless networks with transmission cost*, in Proc. Int. Conf. Netw. Control Optimiz. Springer, 2007, pp. 1–12.

[3] C. W. Commander, P. M. Pardalos, V. Ryabchenko, S. Uryasev, G. Zrazhevsky, *The wireless network jamming problem*, J. Comb. Optim., vol. 2007, no. 14, pp. 481498, 2007.

[4] G. Perin, L. Badia, *Reinforcement Learning for Jamming Games over AWGN Channels with Mobile Players*, Proc. IEEE 26th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD), 2021.

[5] Giovanni Perin, Alessandro Buratto, Nicolò M. Anselmi, Shruti Wagle, Leonardo Badia, *Adversarial Jamming and Catching Games over AWGN Channels with Mobile Players*, Proc. IEEE International Conference on Wireless and Mobile Computing, Networking And Communications (WiMob), 2021.

[6] M. Scalabrin, V. Vadori, A. V. Guglielmi, and L. Badia, *A zero-sum jamming game with incomplete position information in wireless scenarios*, Proc. European Wireless, 2015.

[7] L. Badia and F. Gringoli. *A game of one/two strategic friendly jammers versus a malicious strategic node*, IEEE Networking Letters, vol. 1, no. 1, pp: 6-9, 2019.

[8] W. Xu, W. Trappe, and Y. Zhang, *Channel surfing: defending wireless sensor networks from interference*, Proc. IPSN, pp. 499–508, 2007.

[9] Daskalakis, C., Goldberg, P. W., Papadimitriou, C. H., *The complexity of computing a Nash equilibrium*, SIAM Journal on Computing, vol. 39, no. 1, pp. 195-259, 2009

[10] H. Zhu, M. Ninoslav, M. Debbah, A. Hjørungnes, *Physical layer security game: How to date a girl with her boyfriend on the same table*", Proc. IEEE Gamenets, 2009.

[11] C. E. Shannon, *A Mathematical Theory of Communication*, The Bell System Technical Journal, vol. 27, no. 4, pp. 379–423, 623–656, 1948.

[12] H. Nyquist, *Certain Topics in Telegraph Transmission Theory*, Transactions of the American Institute of Electrical Engineers, vol. 47, no. 2, pp. 617-644, 1928.

[13] G. Bebis, M. Georgiopoulos, *Feed-forward neural networks*, IEEE Potentials, vol 13, no. 4, pp 27-31, 1994.

[14] S. Hochreiter, J. Schmidhuber, *Long Short-Term Memory*, Neural Computation, vol. 9, no. 8, pp. 1735-1780, 1997.

[15] Y. He, T. N. Sainath, R. Prabhavalkar, I. McGraw, R. Alvarez, D. Zhao, D. Rybach, A. Kannan, Y. Wu, R. Pang, Q. Liang, D. Bhatia, Y. Shangguan, B. Li, G. Pundak, K. C. Sim, T. Bagby, S. Chang, K. Rao, A. Gruenstein, *Streaming end-to-end speech recognition for mobile devices*, Proc. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2019.

[16] H. Zen, Y. Agiomyrgiannakis, N. Egberts, F. Henderson, P. Szczepaniak, *Fast, Compact, and High Quality LSTM-RNN Based Statistical Parametric Speech Synthesizers for Mobile Devices*, Proc. Interspeech, 2016.

[17] G. Pundak, T. Sainath, *Highway-LSTM and Recurrent Highway Networks for Speech Recognition*, Proc. Interspeech, 2017.

[18] J. L. Elman, *Finding Structure in Time*, Cognitive Science, vol. 14, no. 2, pp. 179-211, 1990.

[19] R. Zaheer, H. Shaziya, *A Study of the Optimization Algorithms in Deep Learning*, Prop. International Conference on Inventive Systems and Control (ICISC), 2019.