Progetto OSC I

Confronto tra RL e rete neurale RBF

A cura di:

Emanuele Alfano
Filippo Badalamenti
Gabriele Vitti

Introduzione alle funzioni RBF

Abstract progetto

Lo scopo del progetto consiste nel mostrare un differente approccio alla discretizzazione degli stati del Reinforcement Learning in contesti di ridotta memoria disponibile per il mantenimento degli stati.

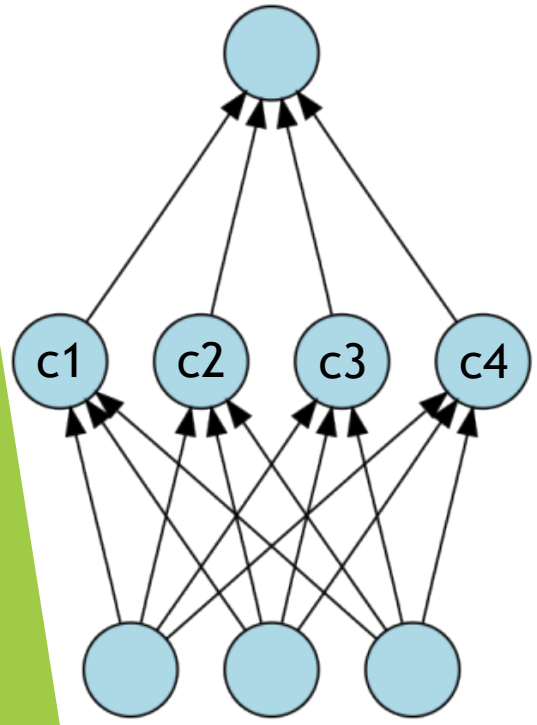
È stato utilizzato lo schema di Policy Improvement con la fase di Evaluation eseguita con Temporal Difference, che permette al sistema di imparare in assenza di un modello e aggiorna i valori correnti anche in base ad infromazioni stimate.

Tra i possibili approcci, è stato scelto l'On-Policy TD: SARSA.

L'obiettivo di minimizzare l'uso della memoria è stato raggiunto tramite l'utilizzo di una Radial Basis Function Network (RBF Network), ovvero un approssimatore lineare.

Radial basis function (RBF) Network

Una rete neurale a base radiale o rete di funzione di base radiale è una rete neurale artificiale che usa le funzione di base radiale come funzioni d'attivazione.



Output y $\varphi : \mathbb{R}^n \to \mathbb{R}$ $\varphi(x) = \sum_{i=1}^{N} w_i \rho(\|x - c_i\|)$

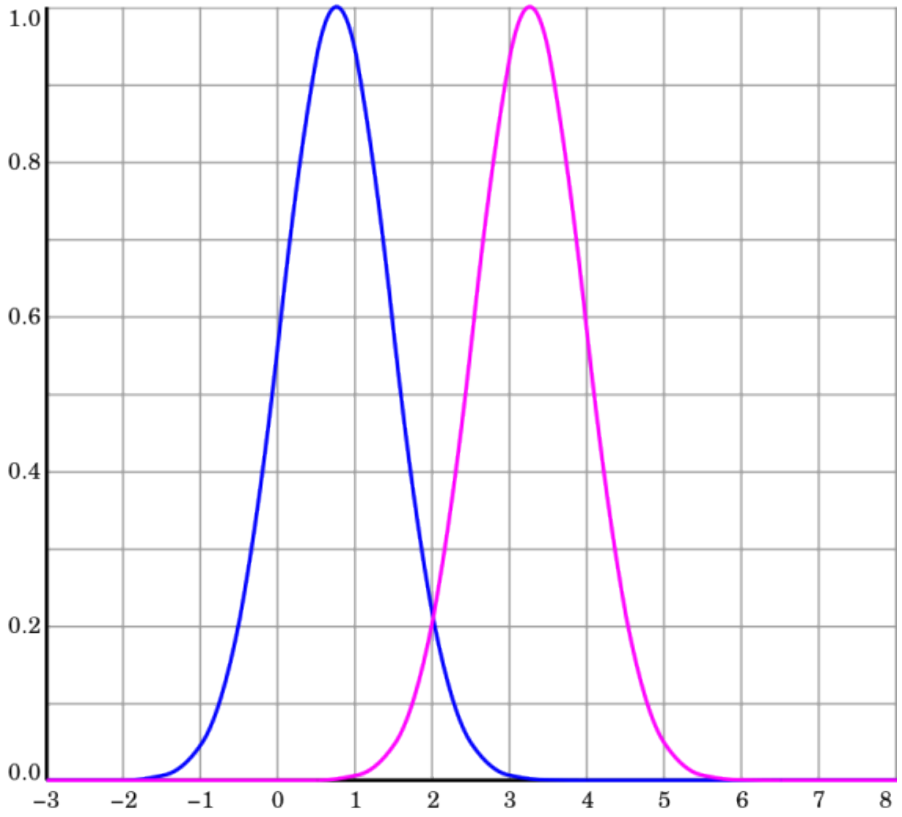
Linear weights

Radial basis functions $\rho\left(\|\mathbf{x} - \mathbf{c}_i\|\right) = \exp\left[-\beta\|\mathbf{x} - \mathbf{c}_i\|^2\right]$

Weights

Input x $\mathbf{x} \in \mathbb{R}^n$

Radial basis function (RBF)



Due funzioni RBF monodimensionali, i cui centri sono situati in *c1=0.75* e *c2=3.25*.

Una funzione RF è caratterizzata dal fatto che il suo valore dipende solo dalla distanza tra il punto considerato e il suo centro fissato

$$\varphi(\mathbf{x}) = \varphi(\|\mathbf{x} - \mathbf{c}\|)$$

Una delle proprietà delle funzioni RBF è che, essendo una forma Gaussiana, ha come proprietà che, all'aumentare della distanza, il valore di attivazione deve tendere a zero.

$$\lim_{||x|| \to \infty} \rho(\|\mathbf{x} - \mathbf{c}_i\|) = 0$$

RBF Network Interpolation

Le reti RBF possono essere usate per interpolare una funzione di cui siano noti i valori dei centri di approssimazione prima definiti:

$$y(\mathbf{x}_i) = b_i, i = 1, \ldots, N$$

Definendo l'elemento: $g_{ij} = \rho(||\mathbf{x}_j - \mathbf{x}_i||)$

È possibile creare la il seguente sistema lineare nell'incognita W (pesi sui valori degli attivatori dei nodi):

$$\begin{bmatrix} g_{11} & g_{12} & \cdots & g_{1N} \\ g_{21} & g_{22} & \cdots & g_{2N} \\ \vdots & & \ddots & \vdots \\ g_{N1} & g_{N2} & \cdots & g_{NN} \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_N \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_N \end{bmatrix}$$

Con: $\mathbf{w} = \mathbf{G}^{-1}\mathbf{b}$,si dimostra che purché tutti i centri siano distinti la matrice G è sempre invertibile

È così possibile calcolare l'output della rete

Simulazione rete interpolante

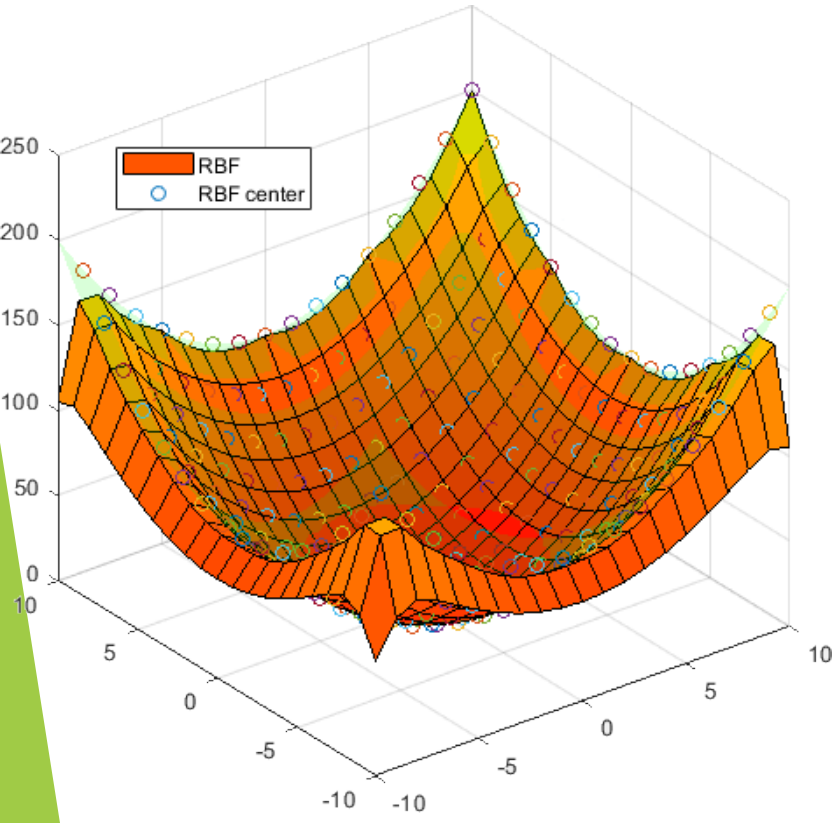
RBF Algorithm

Nei nostri test, oltre all'applicazione classica della rete, abbiamo provato altre varianti nel tentativo di diminuire il tempo di elaborazione, senza però far calare eccessivamente la «qualità» dell'interpolazione.

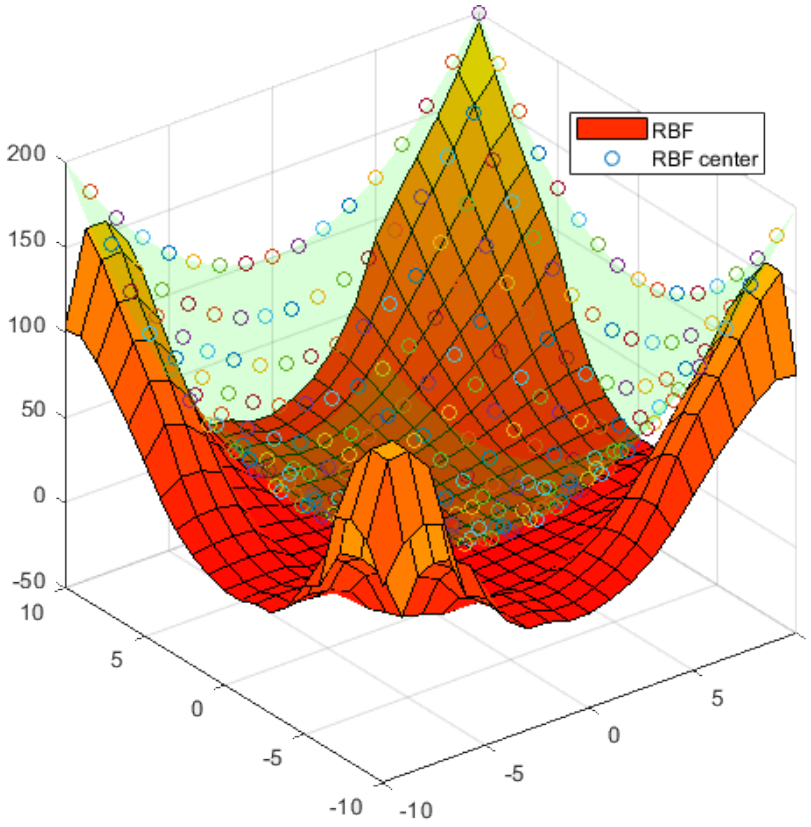
I tre metodi usati sono:

1. Full:
   Viene calcolato il valore di tutti i centri della rete per il punto in esame, ed in base ai pesi dell'interpolazione viene calcolato il risultato della rete.
2. Speed:
   Si interpolano solo i centri vicini entro una certa distanza del punto in esame, conseguentemente solo questi ultimi vengono calcolati e sommati; l'idea di base è che proprio i punti più vicini siano quelli più influenti nel calcolo della soluzione
3. Trunc:
   Supponendo che l'ottimizzazione di MATLAB riconosca la moltiplicazione per 0, si mettono a 0 il 50% dei nodi con pesi più bassi (purtroppo MATLAB non ottimizza e fa comunque eseguire alla FPU i calcoli) è tuttavia interessante osservare l'interpolazione ottenuta.

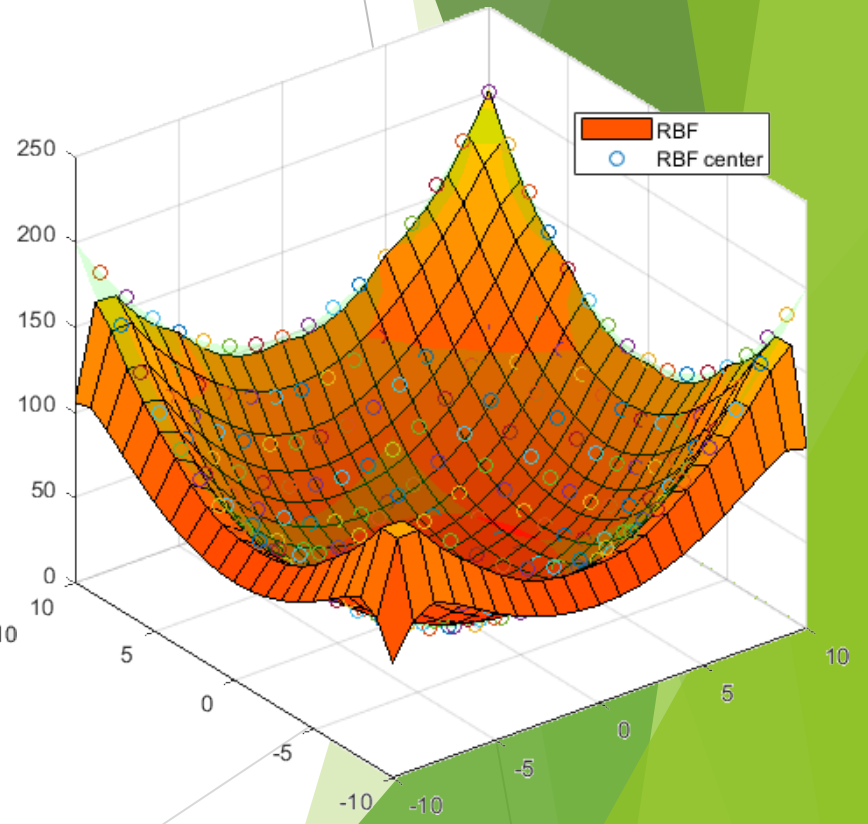
$f(x,y) = x^2 + y^2$ Paraboloide
16 Centri per Lato



RBF Full

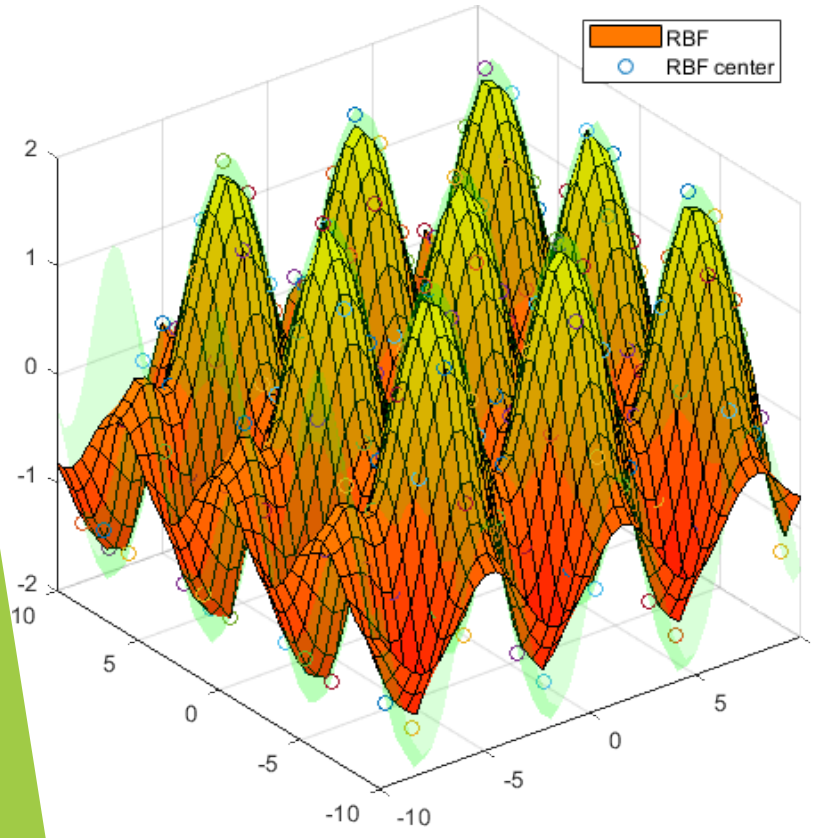


RBF Speed

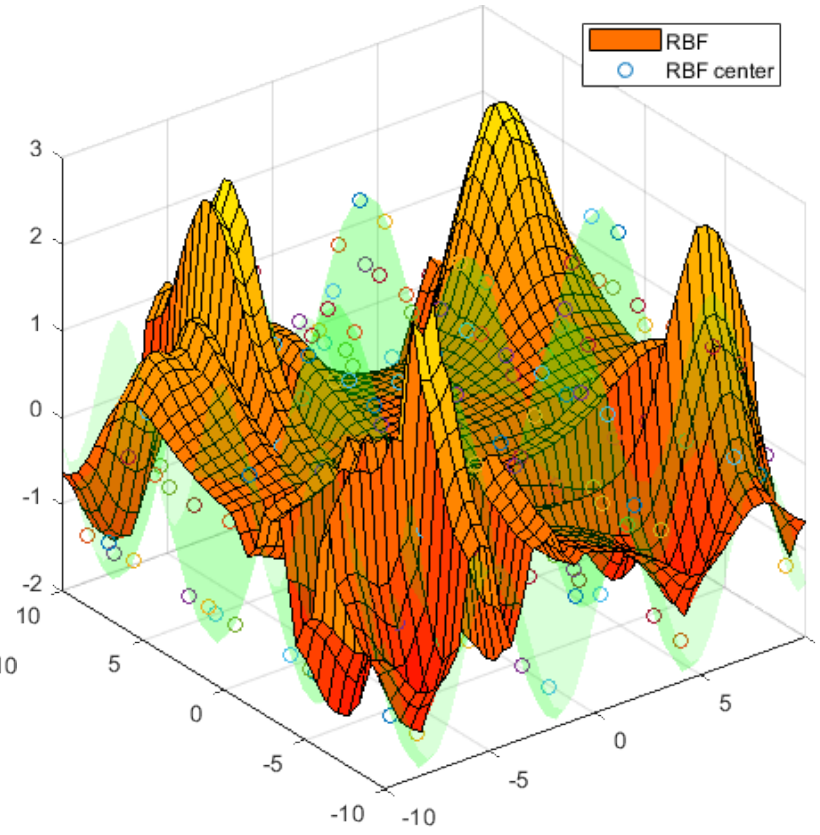


RBF Trunc

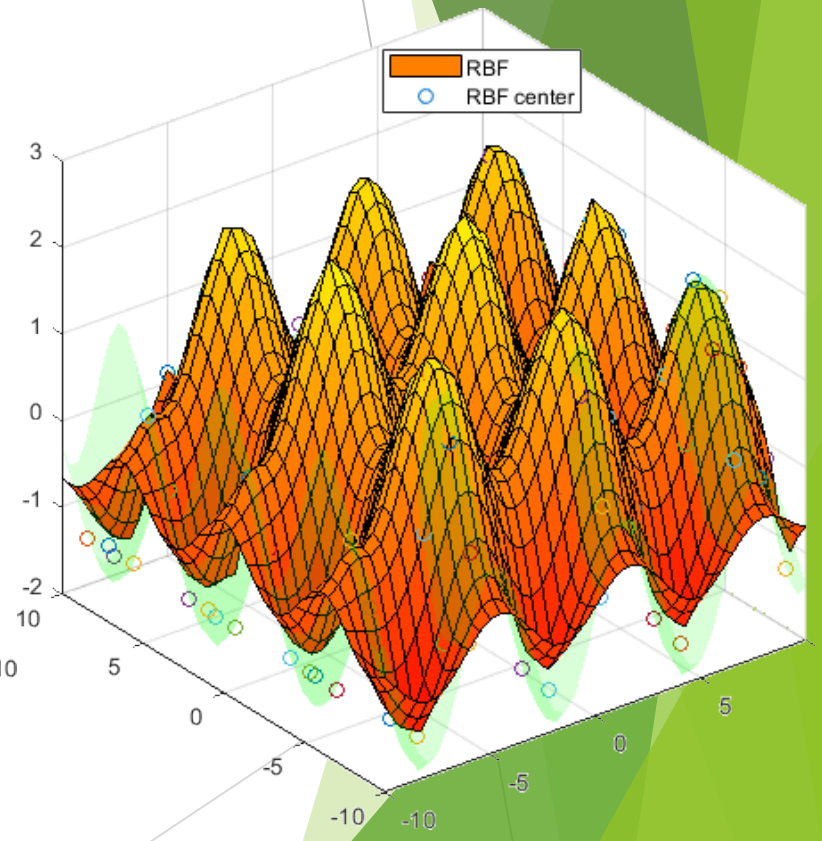
$f(x,y) = \sin(x) + \cos(y)$ Sinusoidale
16 Centri per Lato



RBF Full

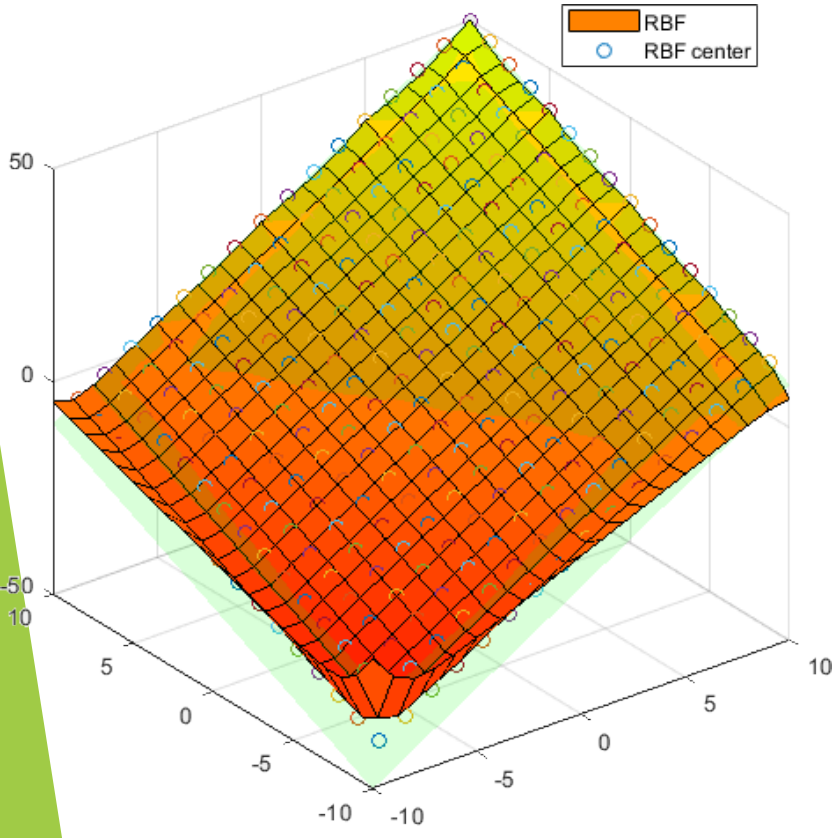


RBF Speed

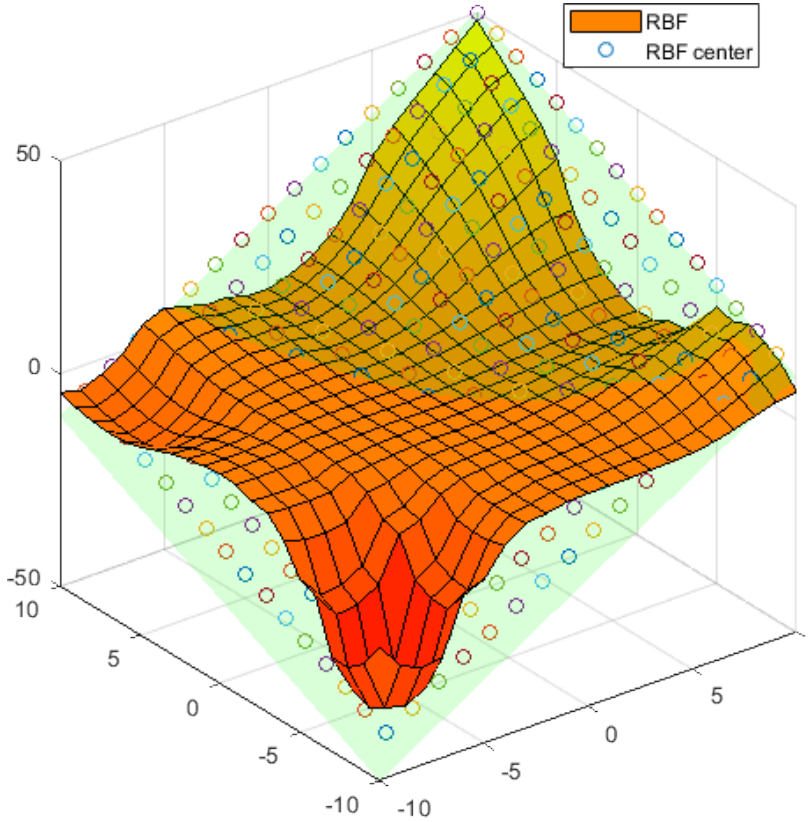


RBF Trunc

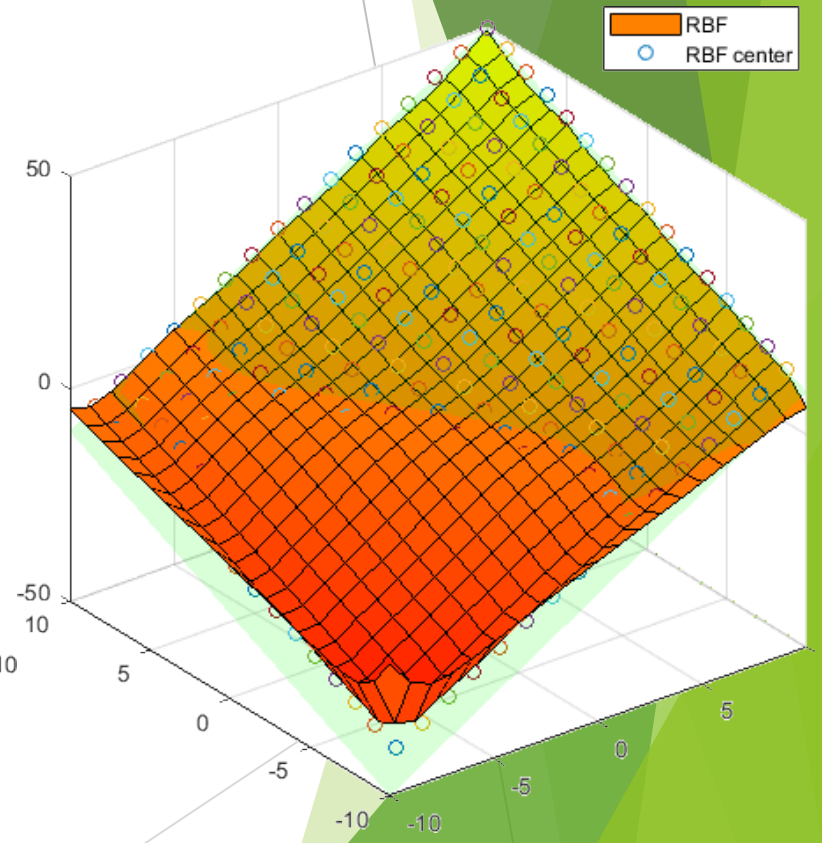
$f(x,y) = 3x + 2y$ Piano
16 Centri per Lato



RBF Full

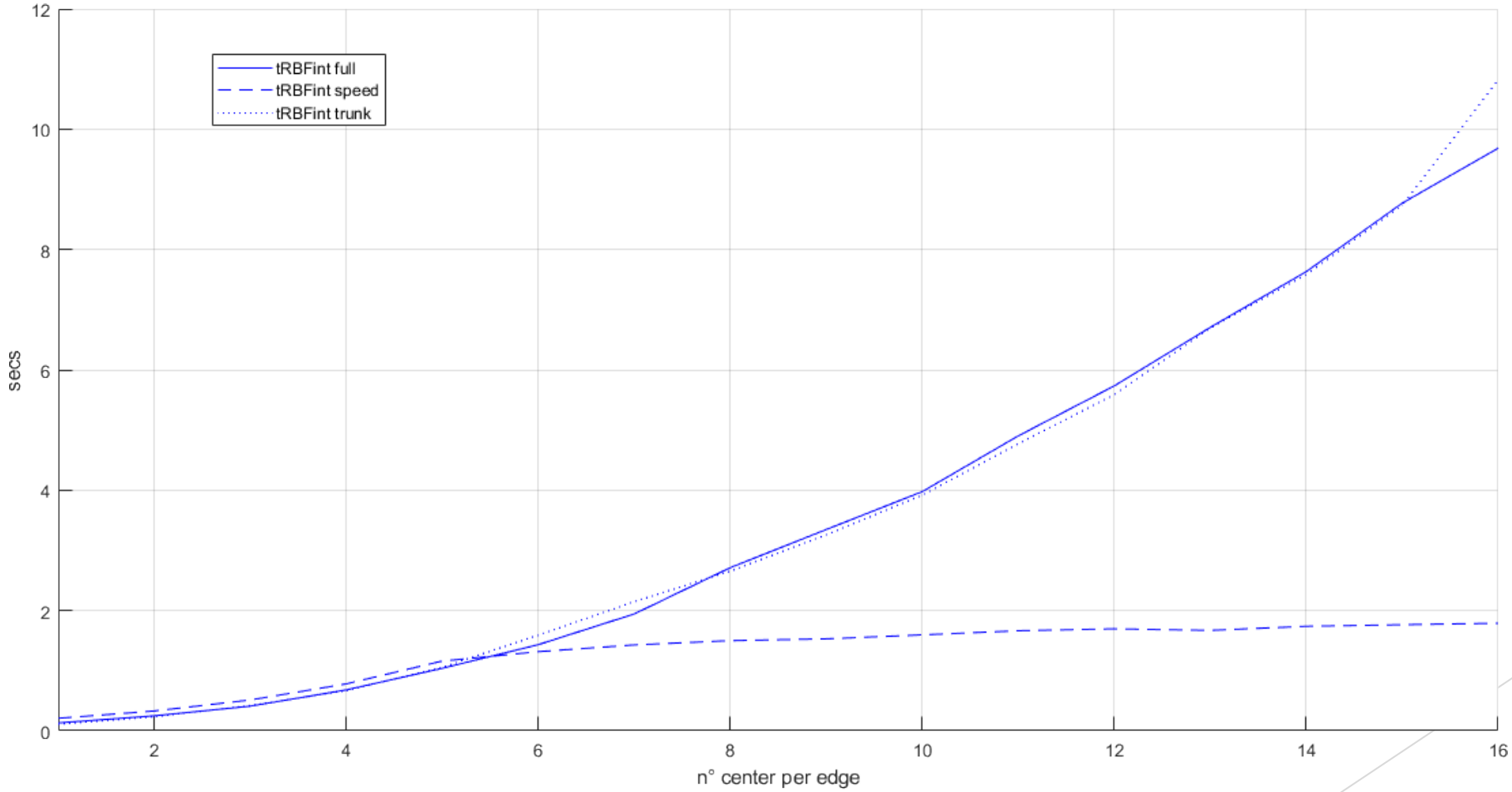


RBF Speed



RBF Trunc

Confronto Temporale Tipologie RBF



Applicazione RL con RBF su Pong

Pong in solitario

Lo scenari di gioco è inspirato al pong, ma invece di avere 2 giocatori, è presente solo uno e dall'altro lato ci sono degli «hot-spot» che se colpiti fanno guadagnare punti.

L'obiettivo della AI è scegliere ogni tempo di campionamento il controllo da attuare sulla barra, per evitare di perdere e massimizzare il punteggio:

1. Salire di +1
2. Stare fermo 0
3. Scendere di -1

Il campo è discretizzato in maniera più rarefatta allo scopo di diminuire gli stati da memorizzare:

Stato: < X, Y, y_Barra, V_xBall, V_yBall>

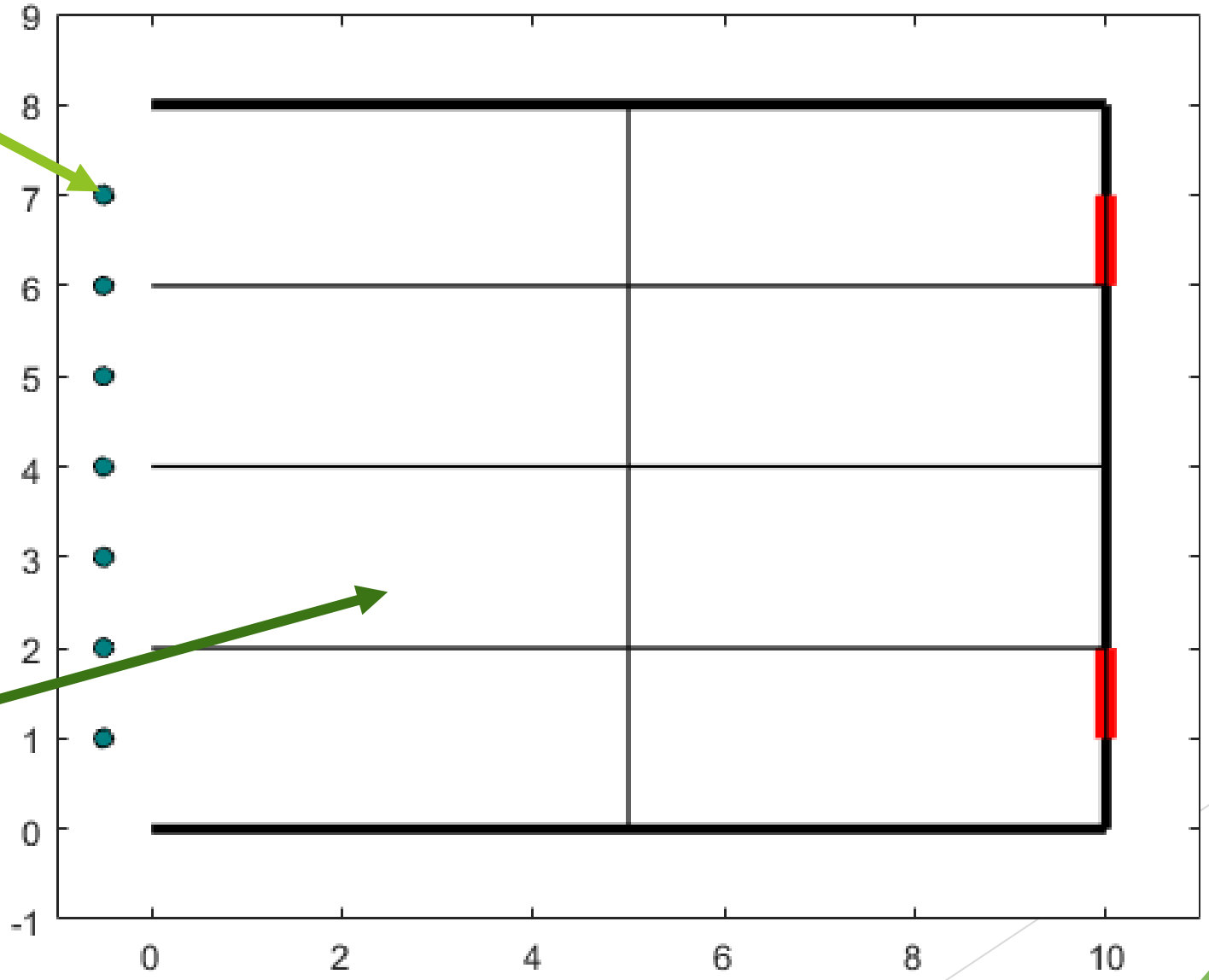
Dallo stato mediante una funzione row-major ci si riporta su una matrice 5D.

In base allo stato si rientra in uno dei settori della matrice, per il calcolo di questo ultimo si usa un metodo di indicizzazione rowMajor, così da associare a ogni tupla un numero univoco.

Visualizzazione dello stato nel campo di Gioco

Stati Discretizzati della Barra

Stati Discretizzati del campo



La pallina può avere 3 colori in base alla V_yBall:

- Rosso:= Sale
- Nero:= circa orizzontale
- Verde:= Scende

Implementazione dell'RBF sul RL di base

- Come visto nel paragrafo precedente una RBF è un approssimatore di funzione, nel nostro caso quindi vogliamo provare ad approssimare la: «Funzione Valore di Stato/Azione» $Q(x_k, u_k)$.

Poiché i controlli sono 3, e non sono correlati tra di loro, per evitare interferenze abbiamo implementiamo 3 diverse reti neurali per renderli indipendenti.

Come risultato finale si hanno:

- $Q_{up}(x_k, +1)$
- $Q_{still}(x_k, 0)$
- $Q_{down}(x_k, -1)$

Ognuna di queste matrici è 5D e occupa in memoria:

$$slotMemory = n_x * n_y * n_{Barra} * n_{v_xBall} * n_{v_yBall}$$

Nel nostro caso di studi abbiamo usato:

- $n_x = 2$
- $n_y = 4$
- $n_{Barra} = 7$
- $n_{v_xBall} = n_{v_yBall} = 3$

$\rightarrow SlotMemory = 504 \; \forall \; Q_i$
Supponendoli memorizzati come double = **3,93KiB**

Con il Pong di esempio il costo era di **32KiB** ogni Funzione Valore Stato/Azione con una riduzione di memoria conseguentemente dell'**88%**

Dati sperimentali

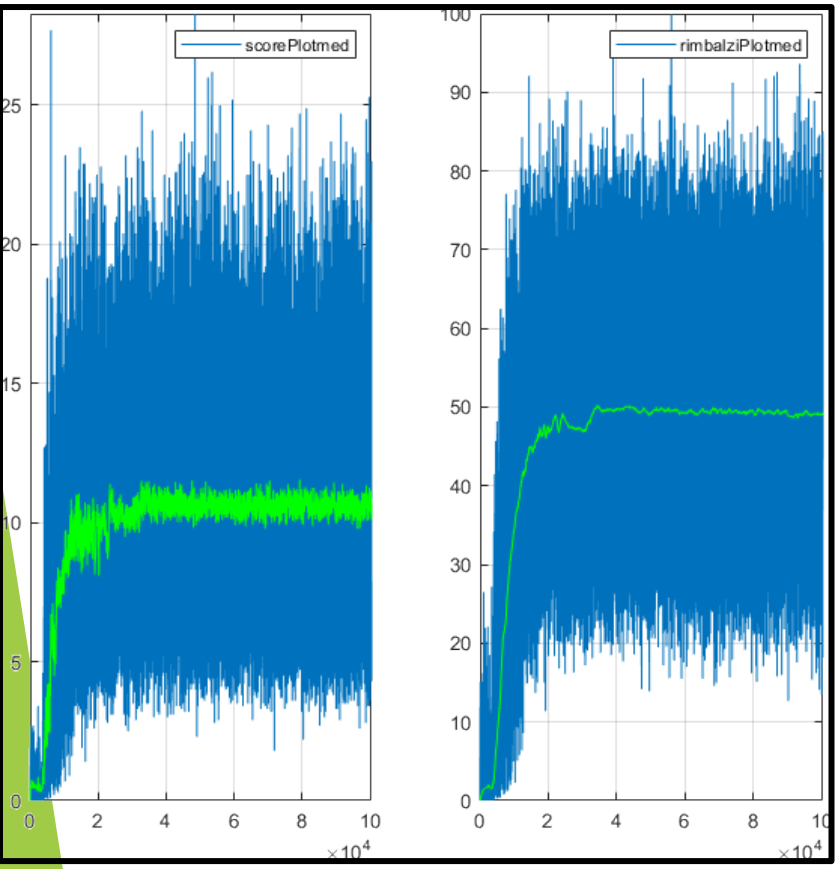
Abbiamo portato avanti, sullo stesso campo di gioco 3 diverse varianti di RL:

1. Classica rete RL a stati ridotti
   Scopo di questa simulazione è avere dei dati di confronto
2. Rete RBF usando come centri i centri degli stati discretizzati:
   Per il valore della funzione si è usata la stima Q_i calcolata fino a quel punto.
   1. RBF_{speed}
      A motivo dell'eccessivo tempo di computazione richiesto abbiamo addestrato la rete usando questa variante con distanza 1 (copre la combinazione dei 18 stati stati più vicini, facendo risparmare alla rete il 97% dei calcoli)
   2. RBF_{full} mix RL
      Sapendo che la rete impara di più quando α è alta, si è scelto un profilo a dente di sega per la α in cui nel primo 5% del tempo viene usata la rete RBF_FULL per cercare di imparare il più possibile, e il restante 95% si usa la classica rete RL per propagare le informazioni apprese.

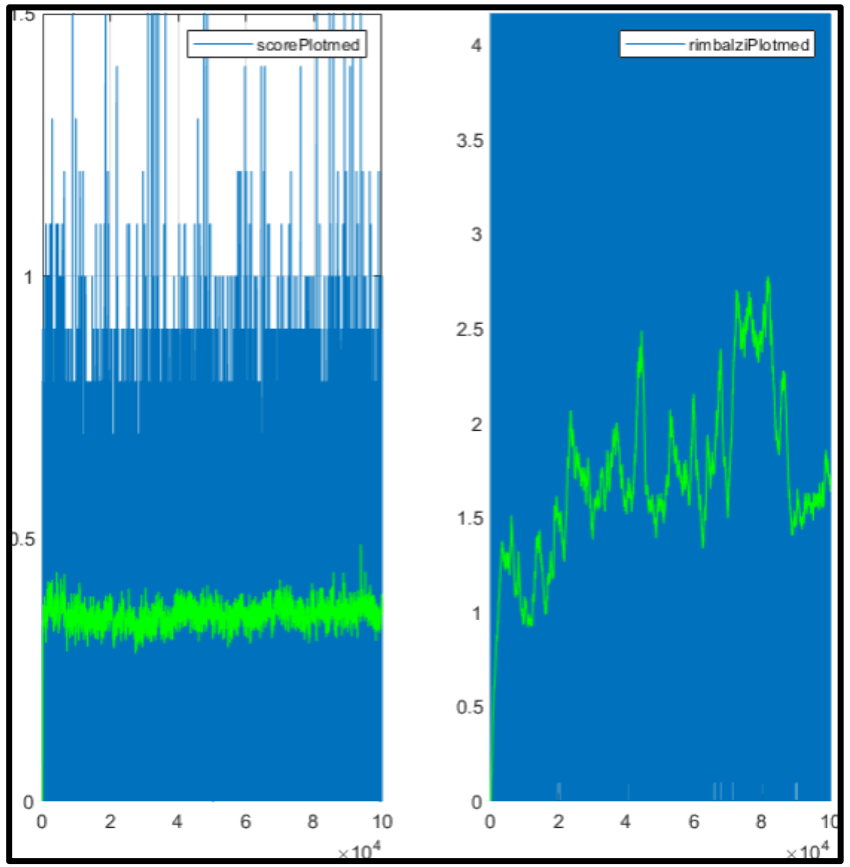
Da osservare che la rete RL è sempre molto più rapida (anche nella versione compilata) rispetto all'onere computazionale di far calcolare il valore interpolato alla rete neurale.

Evoluzione dell'AI

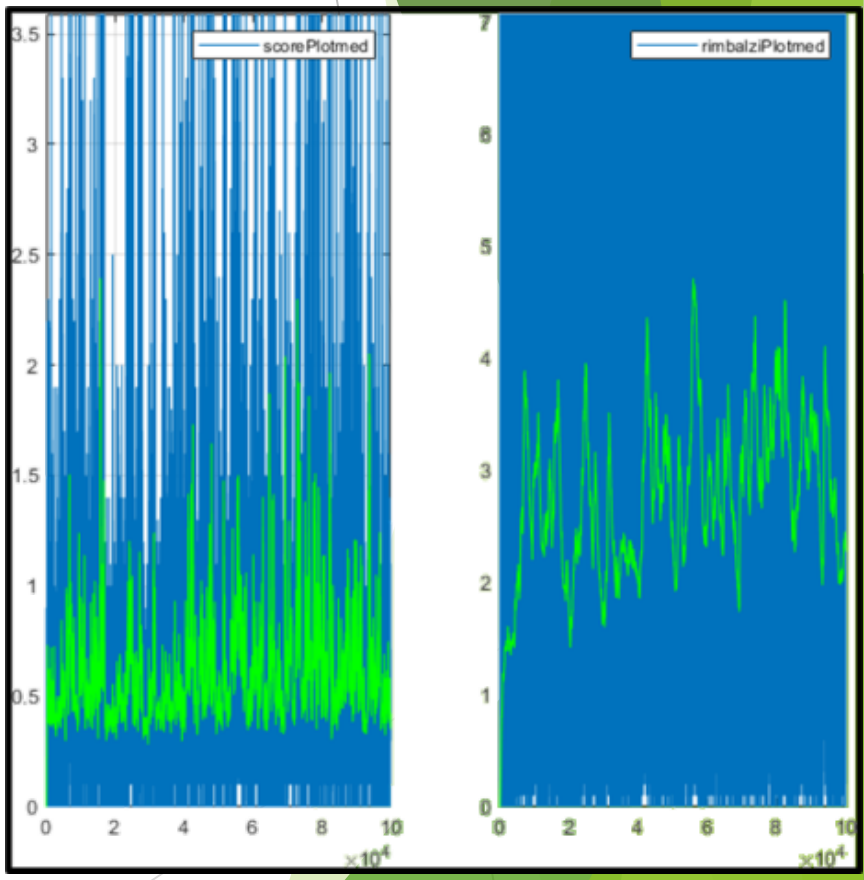
$$\gamma_{start} = 0{,}1\ ;\ \gamma_{end} = 0{,}85\ ;\ \alpha_{start} = 1\ ;\ Max_{rimbalzi} = 100;\ N°_{simulazioni} = 10^6$$



RL Classic



RBF Speed



RBF Full mix RL

Considerazioni finali

I risultati sono stati ottenuti utilizzando il tool «MATLAB Coder» che si occupa di ricompilare in codice C le funzioni scritte in MATLAB e ciò al fine di accelerare il calcolo numerico della simulazione.

Si è osservato un grande incremento del throughput dovuto sia al parallelismo consentito dal codice compilato (dipendente dal N° Thread della macchina) che dalle ottimizzazioni sugli accessi in memoria.

È stato osservato che la stessa simulazione del RBF-speed eseguita full-Matlab è durata 4gg esatti, mentre la stessa, in versione compilata, appena 2,5h con un risparmio del 97,4% di tempo!

La simulazione grafica è invece in linguaggio interpretato a causa di funzioni interne a MATLAB che necessitano della sua macchina virtuale.

Alla luce dei risultati ottenuti, non è stata implementata la RBF_trunc, in quanto il vantaggio temporale sulla full non è rilevante, e le sue prestazioni di apprendimento saranno sempre inferiori a quest'ultima.

Sitografia e Bibliografia

- https://en.wikipedia.org/wiki/Radial_basis_function_network
- https://en.wikipedia.org/wiki/Radial_basis_function
- Slide corso OSC I