Lunar Lander

Introduzione problema

- Lunar lander è un classico problema di ottimizzazione della traiettoria di un razzo.
- Stato delle azioni discreto perché, secondo il principio del massimo di Pontryagin, è ottimale accendere il motore a pieno regime o spegnerlo. (4 azioni)
- Lo stato è un vettore a 8 dimensioni: le coordinate del lander in x e y, le sue velocità lineari in x e y, il suo angolo, la sua velocità angolare e due valori booleani che rappresentano se ciascuna gamba è a contatto con il suolo o meno.
- Per ogni passo, la ricompensa:
    - aumenta/diminuisce quanto più il lander è vicino/lontano dalla piattaforma di atterraggio.
    - aumenta/diminuisce quanto più il lander si muove lentamente/velocemente.
    - diminuisce quanto più il lander è inclinato (angolo non orizzontale).
    - aumenta di 10 punti per ogni gamba a contatto con il suolo.
    - diminuisce di 0,03 punti per ogni fotogramma in cui un motore laterale è acceso.
    - diminuisce di 0,3 punti per ogni fotogramma in cui il motore principale è acceso.
    - L'episodio riceve una ricompensa aggiuntiva di -100 o +100 punti rispettivamente per l'impatto o l'atterraggio in sicurezza.
    - Un episodio è considerato una soluzione se ottiene almeno 200 punti.
- La navicella inizia con una forza iniziale casuale applicata al suo centro di massa
- L’episodio termina se il lander si schianta, esce dalla visuale o non è sveglio

Algoritmi

- REINFORCE
- REINFORCE con baseline
- TD(0) Actor Critic
- Actor Critic con elagibility trace
- DQN

Policy parametrizzata

- Le policy sono parametrizzate tramite rete neurale
- E’ caratterizzata da un livello da 128 neuroni, input dimensione dello spazio di osservazione, output dimensione spazio delle azioni
- Funzione di attivazione ReLu
- Ottimizzatore Adam
- Per REINFORCE e Actor-Critic l’azione viene campionata da una distribuzione discreta creata a partire dalle probabilità restituite dalla rete
- Per DQN l’azione è quella con valore più alto, è deterministico

REINFORCE

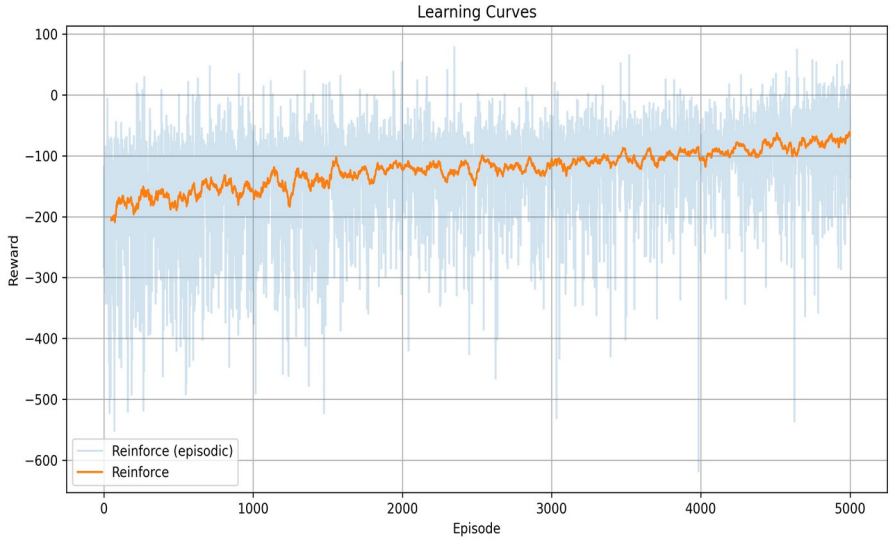
- Uno dei primi metodi policy gradient
- Ottimizza la policy direttamente, è addestrato per massimizzare la probabilità di rendimenti Monte Carlo.

$$\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(a_t|s_t) G_t$$

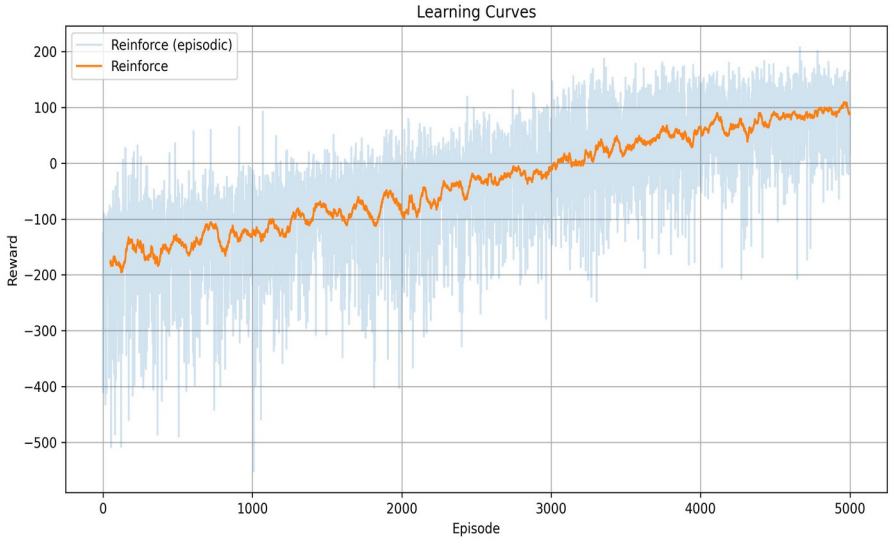
- Per ridurre la varianza si sottrae una baseline, il valore stimato $V(s_t)$
- Viene calcolato tramite un ulteriore NN
- MSE rispetto a G come loss

REINFORCE

REINFORCE



REINFORCE with baseline



Actor critic

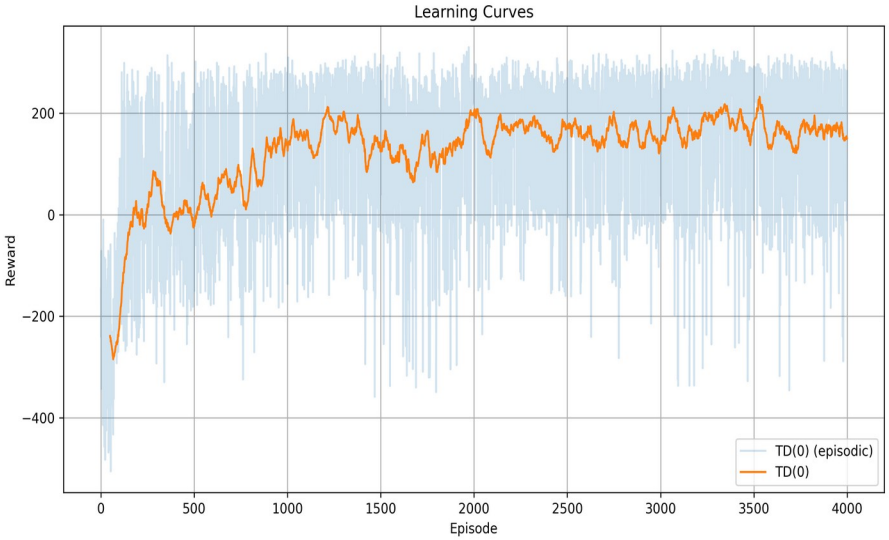
- Actor implementa la policy
- Critic valuta la policy
- Bootstrapping usando TD(0)

$$\begin{cases} \delta_t = r_{t+1} + \gamma V(s_{t+1}; w) - V(s_t; w) \\ w \leftarrow w + \alpha_c \, \delta_t \, \nabla_w V(s_t; w) \\ \theta \leftarrow \theta + \alpha_a \, \delta_t \, \nabla_\theta \ln \pi(a_t | s_t; \theta) \end{cases}$$

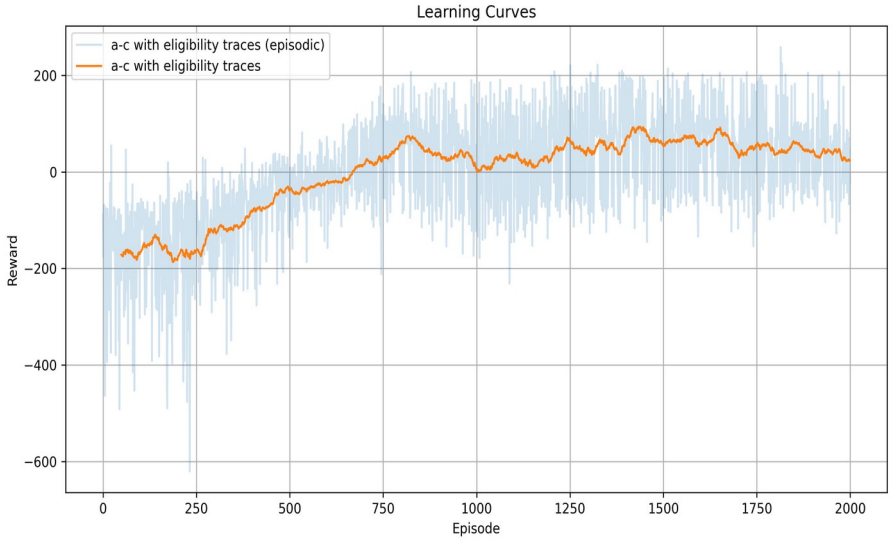
- Introduzione traccie d’elegibilità per compromesso tra Monte Carlo e TD(0)

Actor-Critic

TD(0) Actor-critic



Actor-critic with elagibility trace



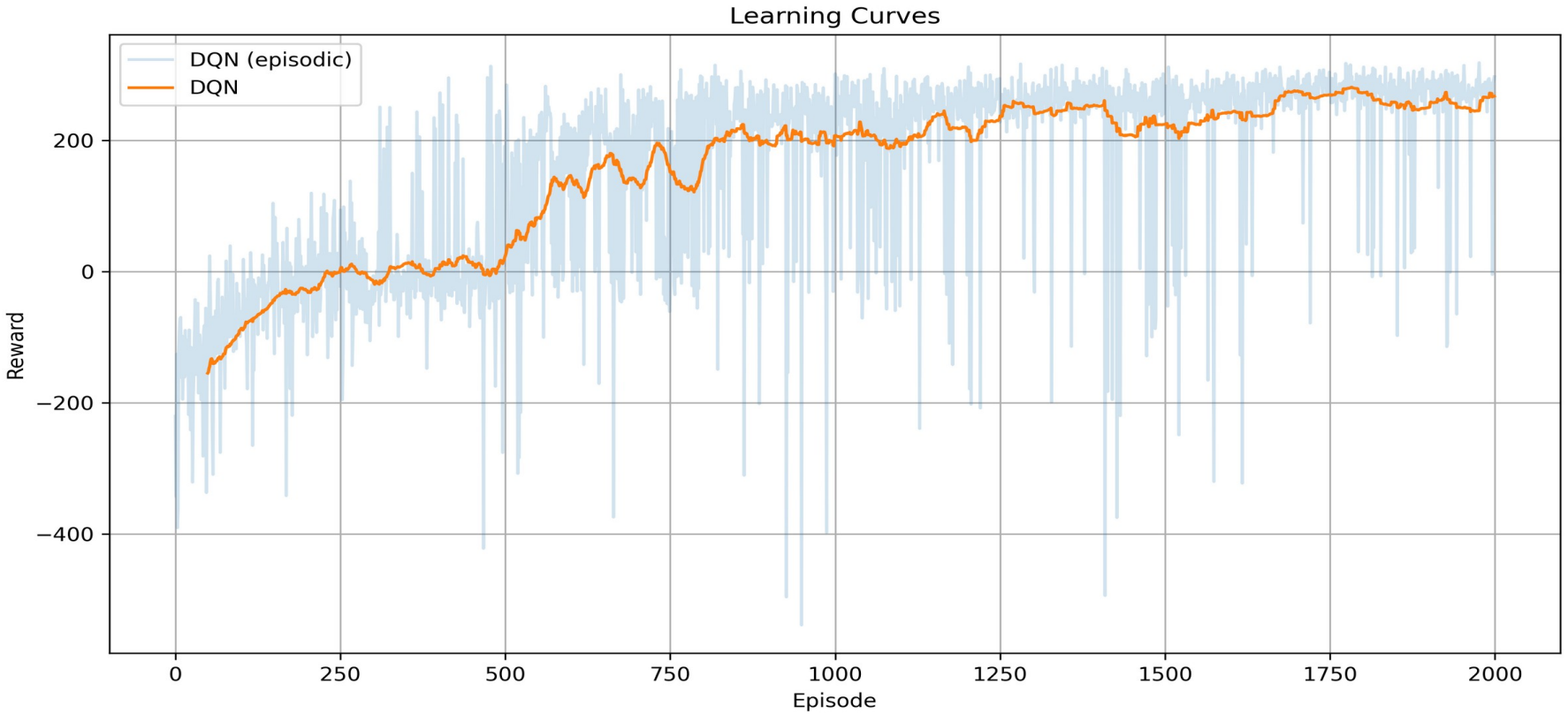
DQN

- Q-learning è indipendente dalla policy che viene seguita
- Stima la ricompensa attesa partendo dallo stato s eseguendo l’azione a e seguendo la politica ottimale

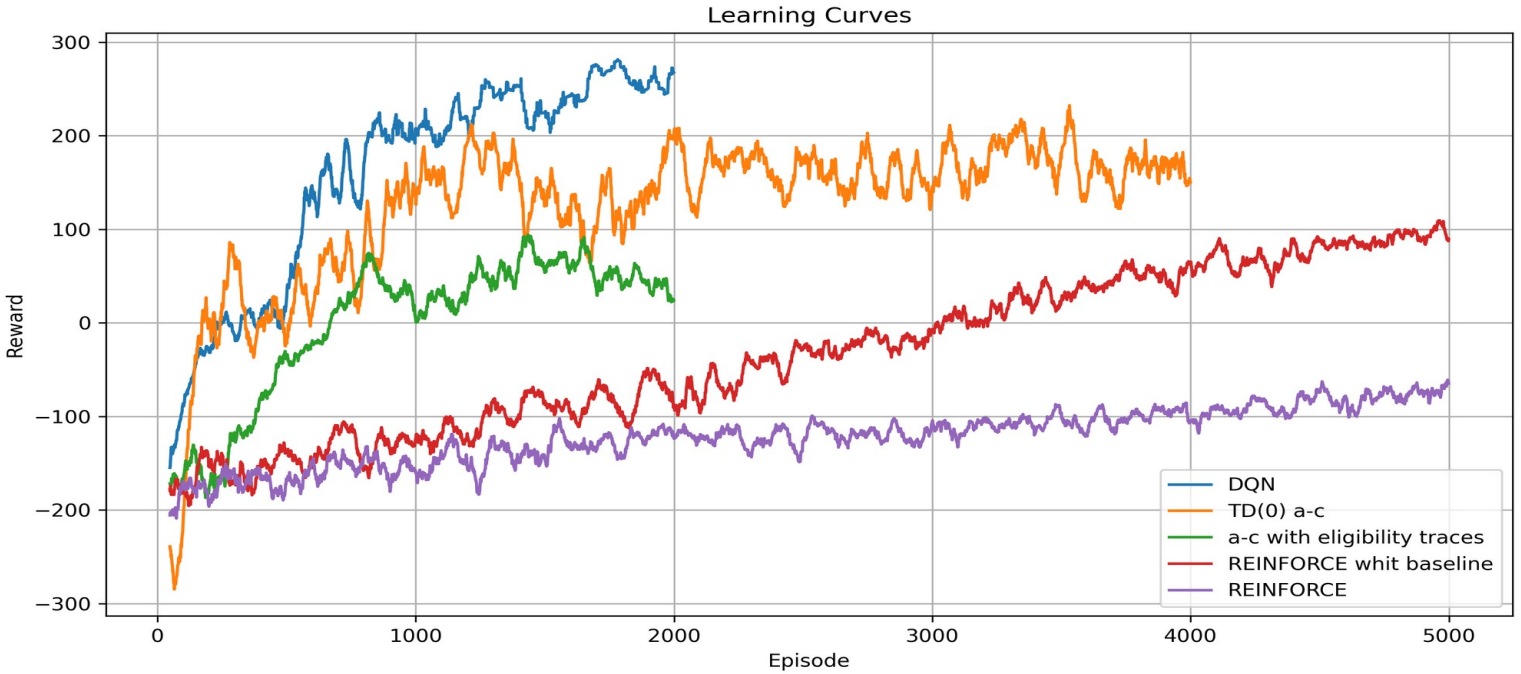
$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t))$$

- DQN:
  - Qnetwork per predirre il valore Q
  - Target Network, rete fissa per stabilizzare l’apprendimento
  - Replay buffer per memorizzare le transizioni
- Epsilon-greedy per la policy in fase di addestramento

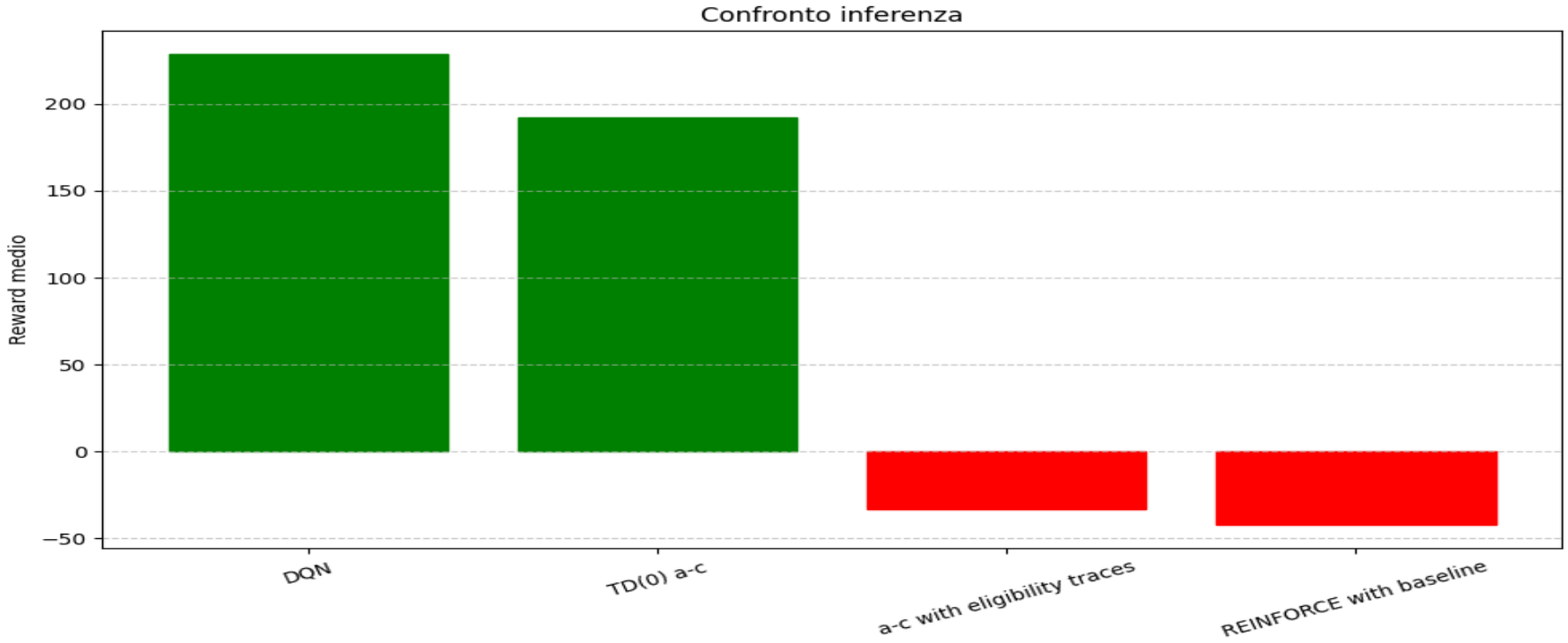
DQN


Learning Curves
DQN (episodic)
DQN
200
0
−200
−400
Reward
0
250
500
750
1000
1250
1500
1750
2000
Episode

Confronto training

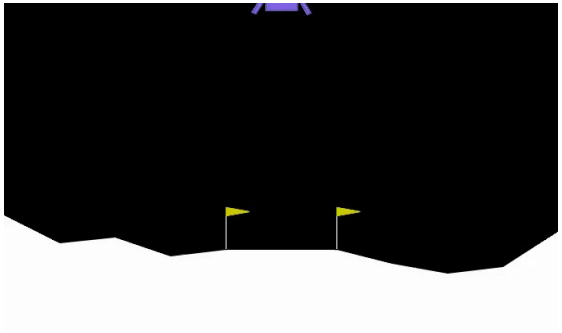


Confronto inferenza

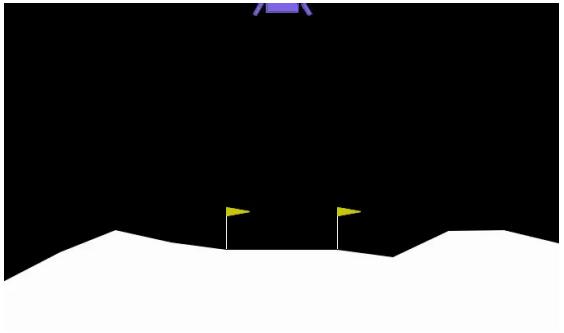


Inferenza

DQN

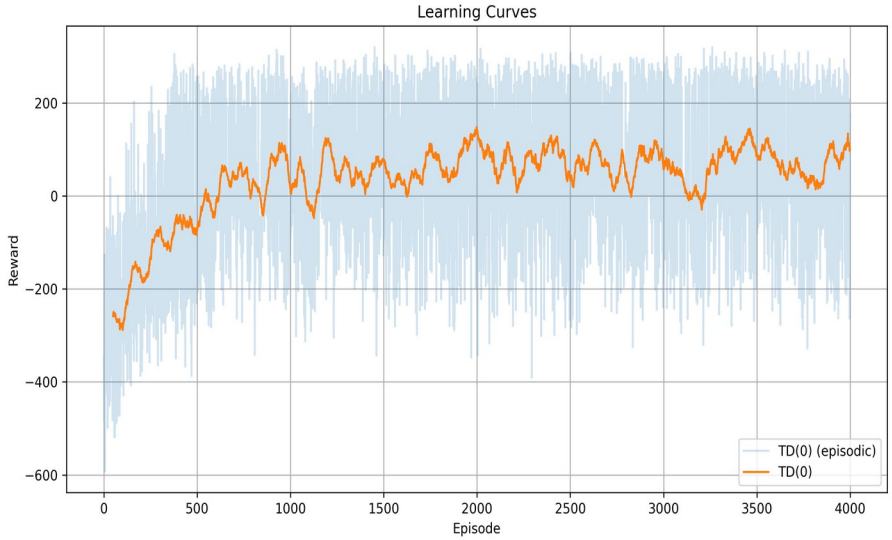


TD(0) A-C

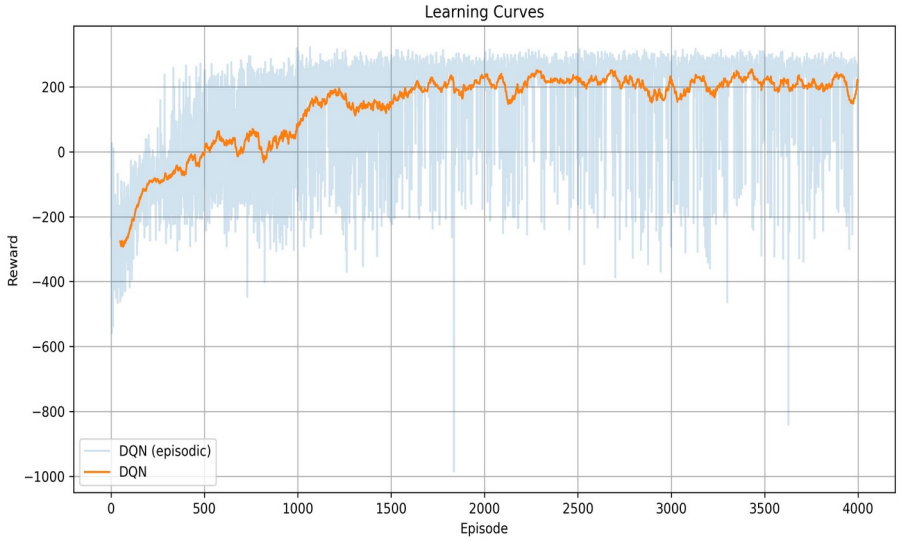


Introdizione vento

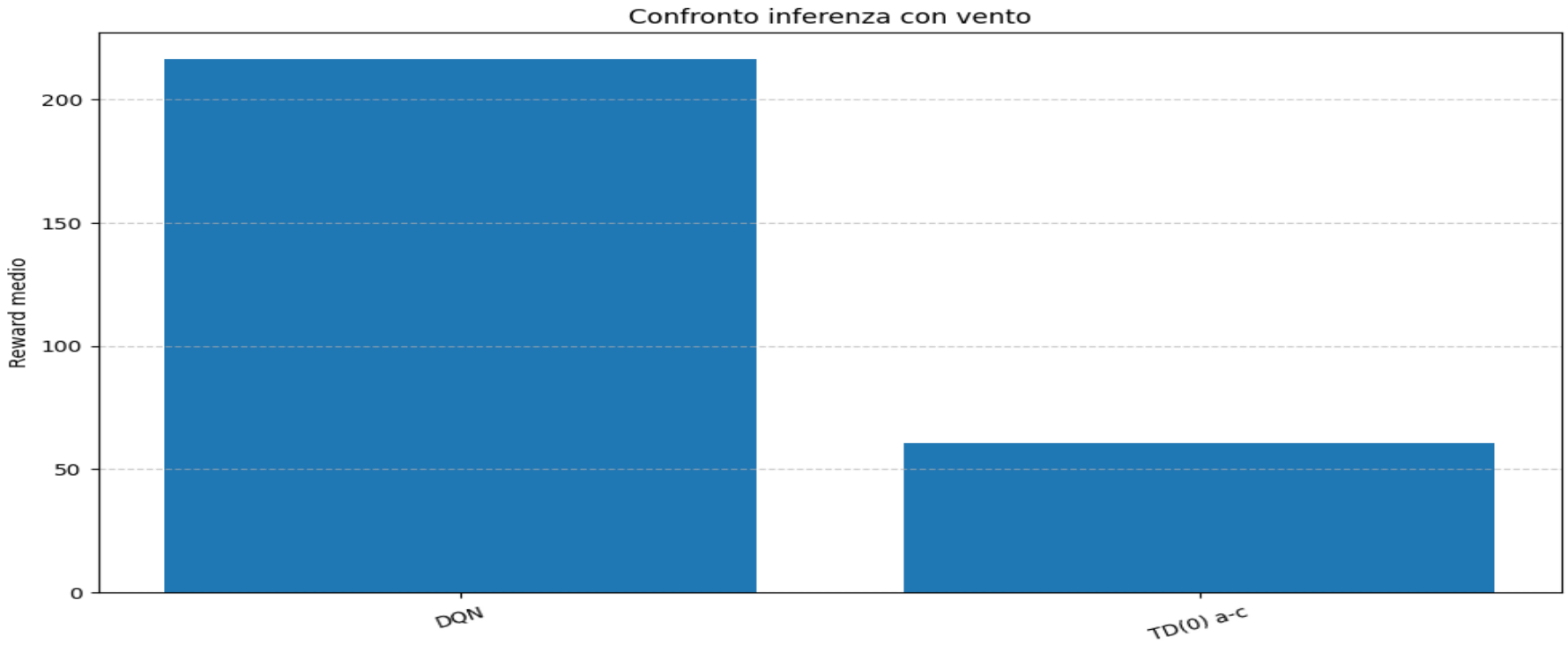
TD(0) Actor-critic



DQN

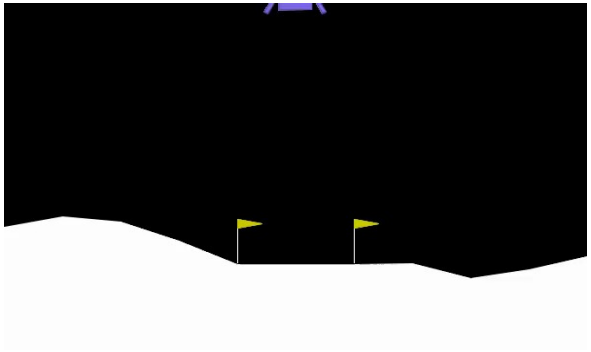


Confronto inferenza con vento

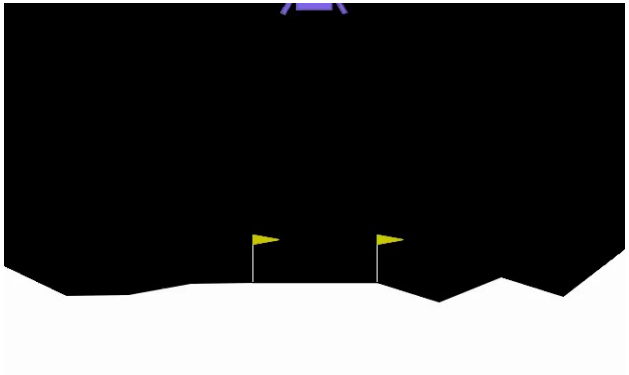


Inferenza

DQN



TD(0) A-C



FINE