UNIVERSITÀ
DEGLI STUDI
DI BRESCIA

INTELLIGENZA ARTIFICIALE

A.A. 2023-2024

Proff. Alfonso Gerevini

Reinforcement learning algorithms

studente :

RAGNOLI MICHELE

Sommario

1 Introduzione

Il reinforcement learning è una tecnica molto utilizzata in contesti dove si deve guidare un agente attraverso un ambiente sconosciuto o in costante cambiamento. Queste capacità lo rendono ottimale, ad esempio, per l'allenamento di robot.

Con questo lavoro voglio approfondire alcuni dei più importanti algoritmi utilizzati per applicare l'apprendimento per rinforzo e capire in quali contesti possano dare il loro meglio.

2 Background

Successivamente verranno definiti alcuni concetti essenziali per apprezzare al meglio il lavoro svolto.

2.1 Reinforcement Learning

Il reinforcement learning è una tecnica di apprendimento automatico in cui gli agenti imparano a risolvere un determinato problema interagendo con l'ambiente circostante. Tale approccio è utile soprattutto quando non dispongo di un modello completo dell'ambiente ed ho situazioni dinamiche e impreviste.

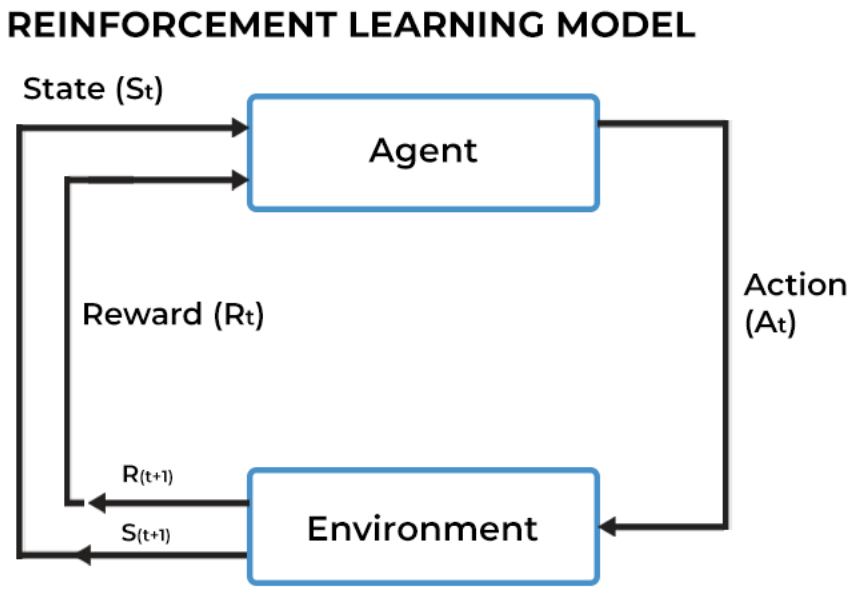

Figura 1: Schema funzionamento Reinforcement Learning.

Come possiamo vedere dalla figura 1 l'agente sulla base dello stato St sceglie l'azione più conveniente, ovvero che gli consente di ottenere un reward più alto. Inizialmente però l'agente non conosce l'azione migliore per lo stato corrente e dunque cerca di provare le azioni disponibili.

2.2 Policy

Nel reinforcement learning, una policy rappresenta la strategia che un agente adotta per selezionare l'azione da compiere in un determinato stato dell'ambiente. La policy può essere deterministica, ovvero assegna un'unica azione ad ogni stato, o stocastica, se assegna una probabilità a ciascuna delle possibili azioni. L'obiettivo dell'apprendimento per rinforzo è proprio quello di trovare la policy ottimale, ovvero quella che massimizza la ricompensa cumulativa ottenuta dall'agente nel lungo periodo.

In altre parole, la policy è il "cervello" dell'agente, che decide cosa fare in base a ciò che "vede" dell'ambiente circostante. È attraverso l'aggiornamento continuo della policy, guidato dalle ricompense ricevute, che l'agente impara a comportarsi in modo sempre più efficace.

L'aggiornamento della policy può avvenire in due modi:

- **On-Policy:** L'agente apprende direttamente dalla policy che sta attualmente utilizzando per interagire con l'ambiente. In altre parole, l'agente migliora la propria policy basandosi sulle esperienze raccolte seguendo quella stessa policy.
- **Off-Policy:** L'agente può apprendere da esperienze generate da una policy diversa dalla propria, permettendo una maggiore flessibilità e un uso più efficiente dei dati. Ho quindi una policy per il training ed una diversa per l'inferenza.

3 Algoritmi Reinforcement Learning

Per applicare l'apprendimento per rinforzo esistono diversi algoritmi, di seguito ne verranno approfonditi alcuni.

3.1 Algoritmo Q-Learning

Il Q-Learning è un metodo off-policy il cui obiettivo è allenare la sua action-value function ad ogni step, ovvero il valore di reward ricevuto una volta compiuta una certa azione partendo da un certo stato. Il metodo è inoltre definito value-based, ovvero trova la policy ottimale indirettamente effettuando il training della action-value function descritta precedentemente e che viene definita "Q-Function".

Come possiamo vedere in figura 2 la Q-Table contiene la Q-Function, ovvero la funzione che dato uno stato ed una possibile azione ritorna il rispettivo Q-Value. Essenzialmente alla fine otteniamo una tabella dove vi è una colonna per lo stato, una per la possibile azione da intraprendere ed infine il rispettivo Q-Value calcolato che si ottiene da tale combinazione stato-azione.

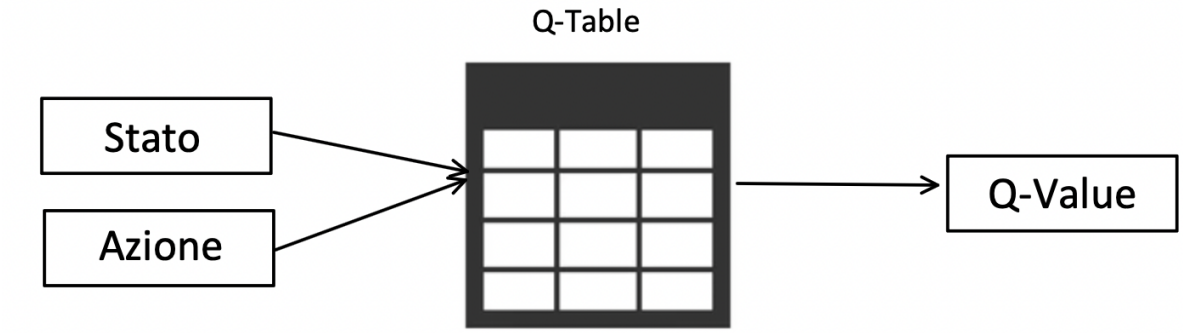

Figura 2: Rappresentazione Q-Function.

Solitamente la Q-Table è inizializzata a zero.

La Q-Function viene descritta in Figura 3.

$$Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma * \max(Q(s', a')) - Q(s, a)]$$

Figura 3: Q-Function per algoritmo Q-Learning.

Nella funzione rappresentata in Figura 3 al valore Q attuale che si ricava partendo da uno stato s ed applicando un'azione viene aggiunto un valore moltiplicato per un tasso di apprendimento alfa che varia tra 0 e 1. Il parametro r rappresenta la ricompensa immediata ottenuta eseguendo l'azione a partendo dallo stato s, s' rappresenta lo stato in cui arrivo dopo avere eseguito l'azione a dallo stato s ed infine Y rappresenta un fattore di sconto, compreso tra 0 e 1, che determina l'importanza delle ricompense future rispetto a quelle immediate.

La scelta dell'azione da intraprendere da uno stato avviene tramite un valore epsilon:

- Con probabilità epsilon viene scelta l'azione con il valore Q più alto.
- Con probabilità 1-epsilon invece viene scelta l'azione in modo casuale.

Il valore di epsilon viene impostato inizialmente alto per permettere all'algoritmo di esplorare e successivamente viene decrementato per seguire le strade migliori, ovvero le coppie stato-azione con un valore più alto. Il procedimento appena descritto rappresenta la policy seguita dall'algoritmo.

Questo algoritmo è particolarmente efficiente per ambienti semplici con azioni discrete.

3.1.1 Deep Q-Learning

Questo approccio va ad aggiungere una neural network per predire i vari Q-value per ogni possibile azione attuabile da un determinato stato.

Per allenare la rete viene definita un Q-Target che possiamo vedere in figura 4, dove alla ricompensa immediata viene aggiunta la ricompensa del prossimo stato moltiplicata per un fattore di discount.

$$R_{t+1} + \gamma max_a Q(S_{t+1}, a)$$

Figura 4: Funzione Q-Target Deep Q-Learning.

Viene inoltre definita una funzione di loss definita come possiamo vedere in figura 5, dove al valore target viene sottratto quello previsto dalla rete.

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

Figura 5: Funzione di Loss Deep Q-Learning.

Prima della fase di training avviene una fase di sampling dove effettuo delle azioni e salvo i risultati in delle tuple. Da queste tuple verranno poi estratti batch per l'addestramento.

3.2 Algoritmo SARSA

L'algoritmo SARSA (State-Action-Reward-State-Action) è un metodo di apprendimento per rinforzo che si basa come nel Q-Learning sull'aggiornamento di una funzione di action-value Q.

Come nel precedente algoritmo viene ricavata una tabella dove per ogni stato-azione viene salvato il rispettivo Q-value che si può ottenere.

La differenza principale con il Q-Learning sta nel fatto che questo algoritmo passa ad un approccio on-policy, ovvero utilizza la stessa politica sia per selezionare le azioni da eseguire nell'ambiente che per aggiornare la funzione Q. Ovvero viene sempre utilizzata una policy epsilon-greedy.

Di conseguenza la Q-Function viene modificata come possiamo vedere in figura 6, dove possiamo notare come la funzione di max sia stata rimossa.

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

Figura 6: Q-function per algoritmo SARSA.

3.3 Algoritmo PPO

L'algoritmo sviluppato da OpenAi[1] PPO (Proximal Policy Optimization) è una delle tecniche più avanzate e utilizzate nel Reinforcement Learning. Come SARSA si basa su un approccio on-policy ed utilizza un metodo policy gradient, ottimizzando così direttamente la politica di un agente.

PPO utilizza inoltre un aggiornamento della politica limitato, che evita cambiamenti drastici nella policy approssimativa, mantenendo l'aggiornamento in una "zona di sicurezza". In questo modo riesco a recuperare più velocemente in caso di errori.

Il punto chiave di PPO è l'uso di una funzione di ratio, che considera il rapporto di probabilità rt(θ), dato dal rapporto tra la probabilità dell'azione corrente con il parametro attuale θ e quella precedente θ(old); in figura 7 possiamo vedere la formula completa.

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$$

Figura 7: Formula rapporto di probabilità PPO.

Sempre nella figura 7 πθ(a(t)|s(t)) rappresenta la politica parametrizzata dalla rete neurale con i parametri θ, che produce la probabilità di eseguire l'azione a(t) nello stato s(t).

Un altro aspetto importante è che la funzione di perdita di PPO è definita come una funzione clippata che limita il rapporto rt(θ) in un intervallo prestabilito, solitamente [1−ϵ,1+ϵ], per evitare cambiamenti troppo drastici nella politica e dove epsilon è di solito 0,20; in figura 8 possiamo vedere la formula completa dove A^t è il vantaggio stimato (advantage), e la funzione di clipping impedisce che il rapporto di probabilità si discosti troppo dal valore 1.

$$L^{\text{CLIP}}(\theta) = \mathbb{E}_t \left[\min \left(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t \right) \right]$$

Figura 8: Funzione di perdita PPO.

Per concludere dunque PPO si basa quindi su un bilanciamento tra esplorazione e stabilità, dove la politica viene aggiornata solo se il miglioramento è abbastanza significativo, mantenendo comunque la coerenza con le iterazioni precedenti. Questo approccio lo rende efficace e meno soggetto a instabilità rispetto a metodi che permettono cambiamenti più drastici nella politica.

4 Test Performance Algoritmi

Di seguito verranno riportati i risultati degli algoritmi precedentemente descritti applicati ad alcuni ambienti di simulazione presi da Gymnasium[2].

4.1 Ambiente FrozenLake

Il seguente ambiente di simulazione si pone come obiettivo di fare raggiungere ad un personaggio il suo pacco regalo. Come possiamo vedere dalla Figura 9 lungo il percorso sono previsti dei buchi di ghiaccio dentro cui il personaggio potrebbe cadere.

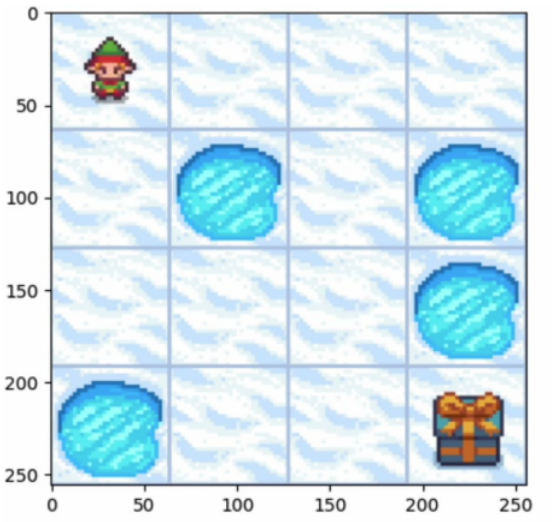

Figura 9: Immagine ambiente di simulazione FrozenLake.

Lo spazio delle azioni è discreto e permette al personaggio di muoversi in alto, a destra, a sinistra o in basso.

Lo spazio delle osservazioni rappresenta la posizione corrente del personaggio ed è composto da un valore calcolato come riga_corrente*numero_righe + colonna_corrente.

Il personaggio riceve un premio di un punto se raggiunge l'obiettivo, altrimenti zero.

La simulazione termina se il personaggio cade in un buco oppure riesce a raggiungere il goal.

Durante i test è stata variata la dimensione della griglia ed il parametro di slippery è stato disattivato.

4.1.1 Test Algoritmo Deep Q-Learning

Come possiamo vedere dalla figura 10 il valore di epsilon è stato inizialmente impostato ad uno e questo significa che inizialmente l'algoritmo esplorerà le azioni disponibili. Con il passare degli episodi tale valore viene decrementato. Sempre dalla Figura 10 notiamo come il reward continui a salire velocemente fino a raggiungere un picco all'episodio 1000 e rimanere poi stabile.

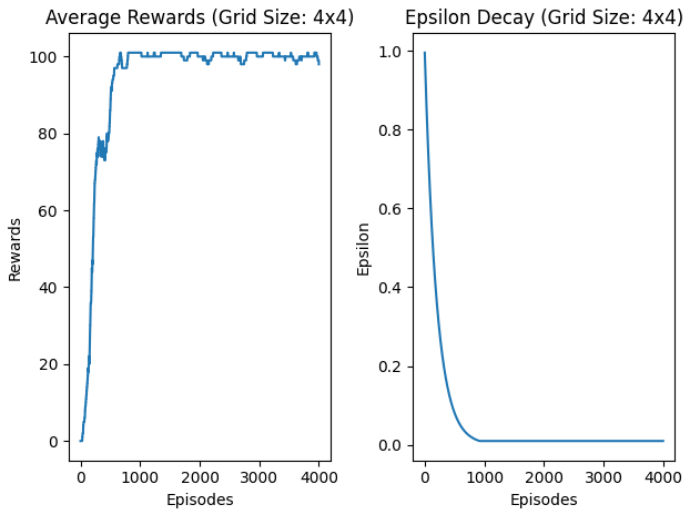

Figura 10: Reward ed epsilon rapportati con il numero di episodi griglia 4x4.

Successivamente in figura 11 possiamo invece vedere il numero di steps effettuati dal personaggio per ogni episodio prima che venisse interrotto per avere incontrato un buco o raggiunto un goal.

Possiamo notare come inizi a stabilizzarsi dai 600 episodi in poi circa, evidenziando così l'apprendimento di un percorso ottimale. I vari picchi dopo i 600 episodi rappresentano episodi in cui l'agente ha esplorato azioni che lo hanno portato lontano dal goal.

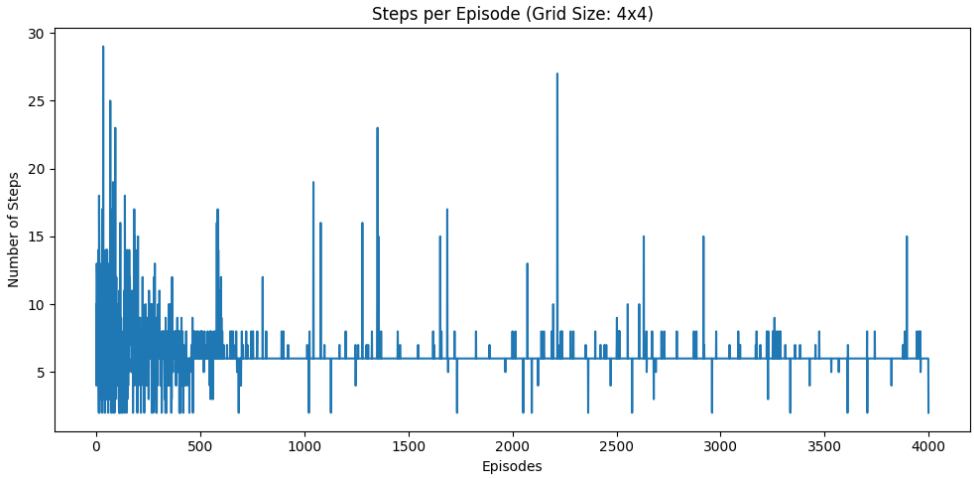

Figura 11: Numero di steps rapportati per numero episodi griglia 4x4.

Queste ultime comunque decrementano con il diminuire del parametro epsilon.

Successivamente ho deciso di incrementare le dimensioni della griglia passando ad una 7x7.

Come possiamo vedere in figura 12, questa volta l'incremento nei rewards è più pronunciato e possiamo notare come la convergenza al reward massimo avvenga prima.

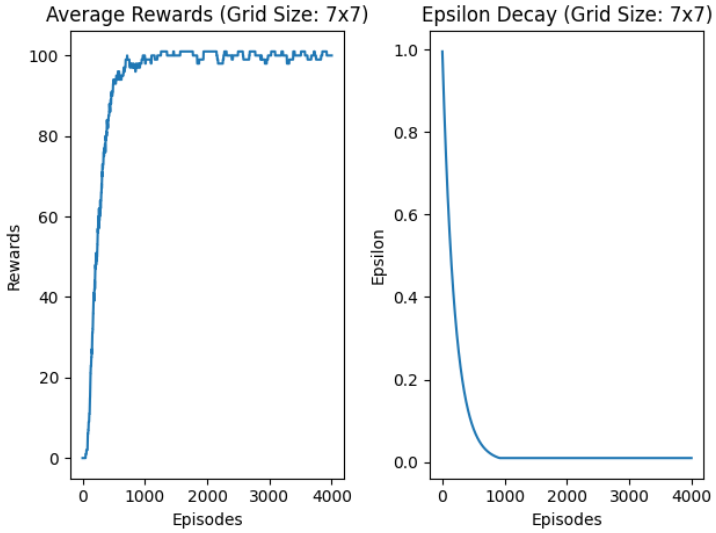

Figura 12: Reward ed epsilon rapportati con il numero di episodi griglia 7x7.

Anche il numero di steps ha subito cambiamenti, come possiamo vedere in figura 13. La fase di stabilizzazione avviene leggermente dopo.

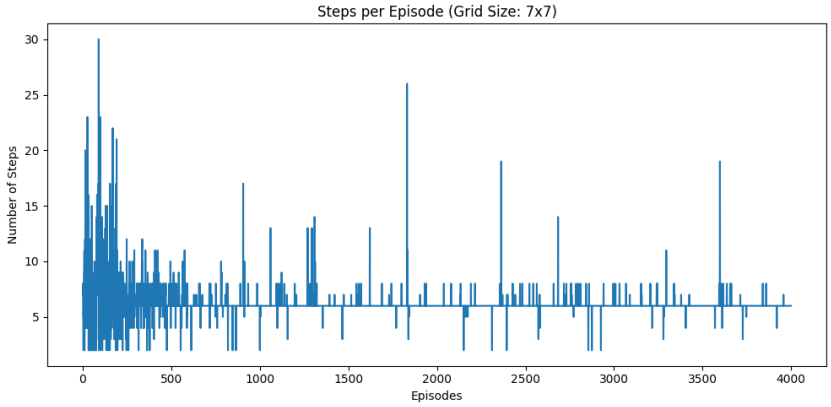

Figura 13: Numero di steps rapportati per numero episodi griglia 7x7.

4.1.2 Test Algoritmo SARSA

Come possiamo vedere dalla figura 14 l'andamento dei rewards al passare degli episodi è molto simile al caso 4x4 dell'algoritmo deep q-learning. Un aspetto interessante sembra però essere il calo nei pressi dei 2000 episodi dove probabilmente l'agente ha scelto azioni sbagliate.

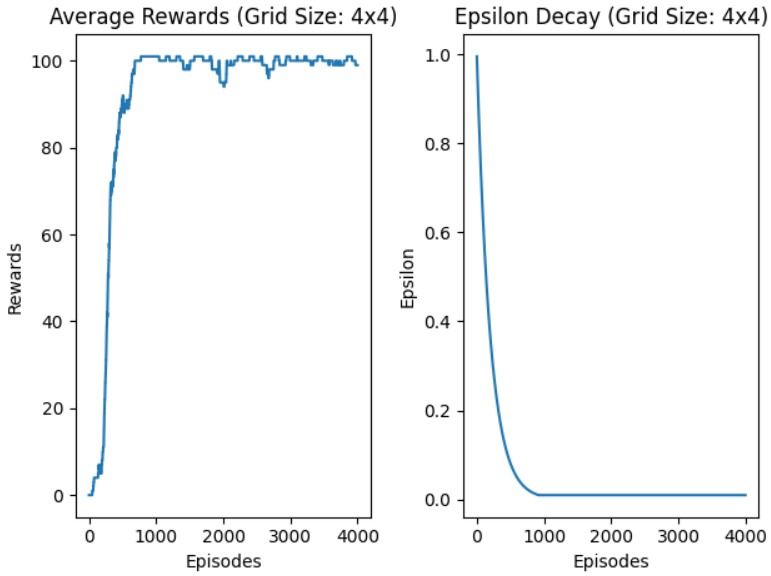

Figura 14: Reward ed epsilon rapportati con il numero di episodi griglia 4x4.

Come possiamo vedere dalla Figura 15 numero di steps per episodi come nell'algoritmo precedente tende a stabilizzarsi intorno ai 600 episodi ma questa volta presenta meno rumore sia prima che dopo la

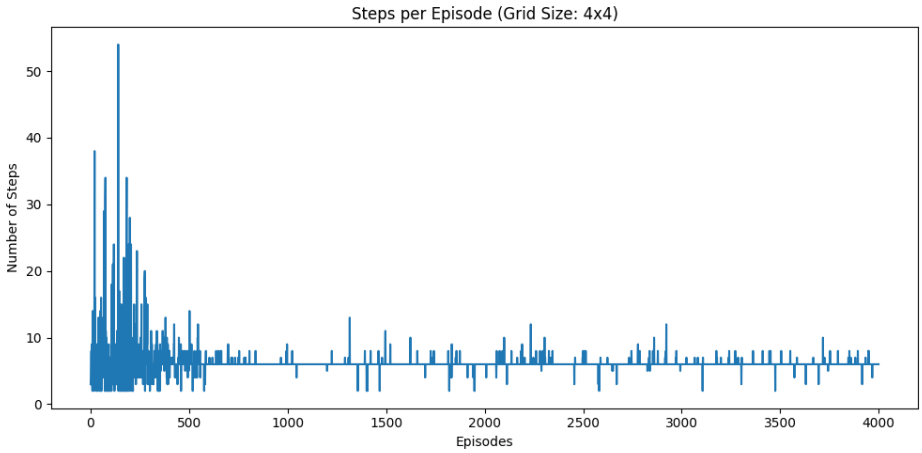

Figura 15: Numero di steps rapportati per numero episodi griglia 4x4.

stabilizzazione.

Questo dimostra un apprendimento migliore.

Successivamente ho deciso ancora di incrementare le dimensioni della griglia passando ad una 7x7.

Dalla figura 16 possiamo notare come l'andamento dei rewards sia molto simile al caso 7x7 dell'algoritmo deep q-learning.

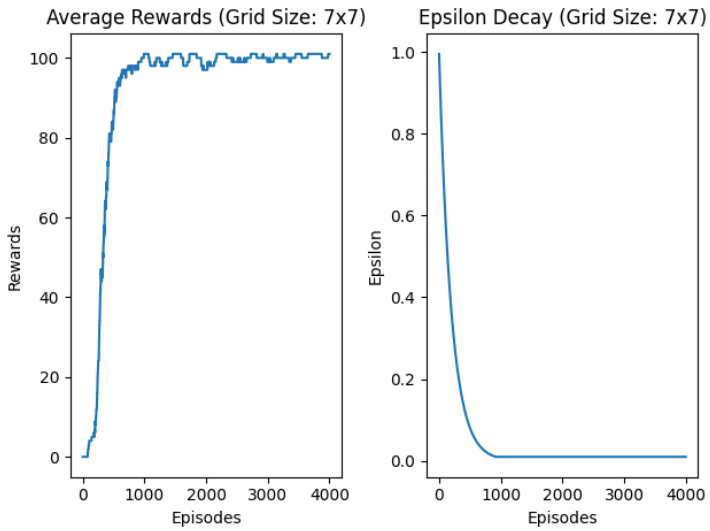

Figura 16: Reward ed epsilon rapportati con il numero di episodi griglia 7x7

Dalla figura 17 possiamo notare come rispetto al caso 4x4 il numero di steps questa volta inizi a stabilizzarsi prima.

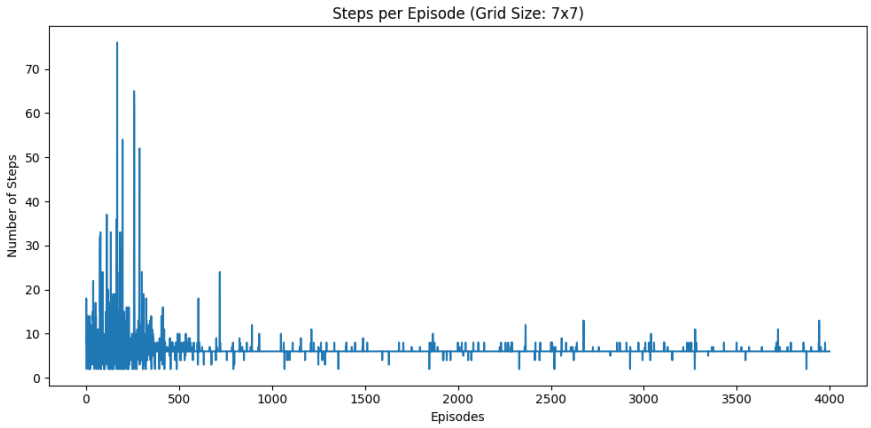

Figura 17: Numero di steps rapportati per numero episodi griglia 7x7.

Inoltre, successivamente alla stabilizzazione il rumore sembra essere diminuito ulteriormente.

4.1.3 Test Algoritmo PPO

A differenza dei precedenti algoritmi, PPO non utilizza il parametro epsilon e quindi avremo un grafico in meno.

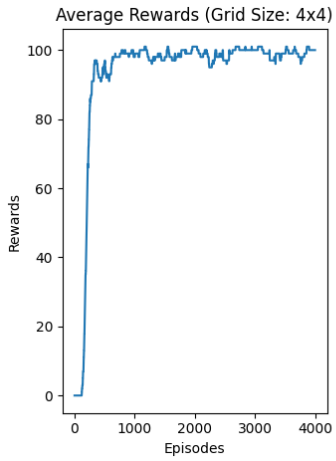

Figura 18: Reward rapportato con il numero di episodi griglia 4x4.

Come fatto in precedenza partiamo dalla griglia 4x4.

Come possiamo vedere dalla figura 18 l'algoritmo raggiunge il massimo numero di rewards a circa 500 episodi e quindi prima dei casi precedentemente studiati. Successivamente il valore rimane stabile.

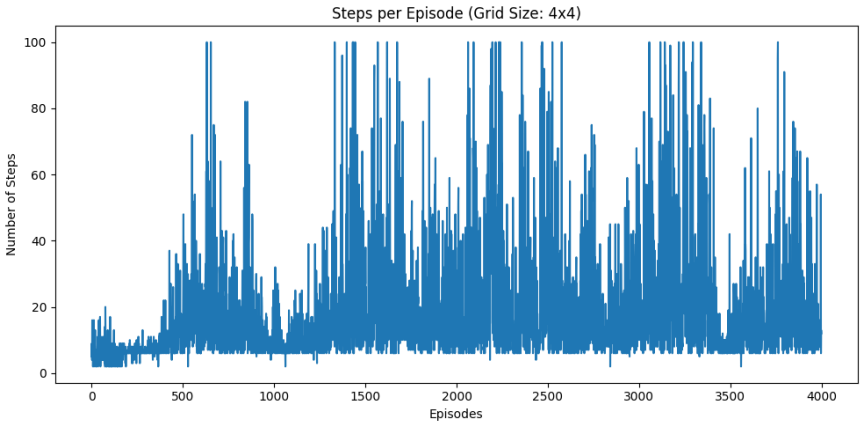

Figura 19: Numero di steps rapportati per numero episodi griglia 4x4.

Per quanto riguarda il numero di steps per episodio possiamo notare dalla Figura 19 come l'algoritmo questa volta non converga all'avvicinarsi dei 4000 episodi. L'agente dunque trova una soluzione complessa o non riesce a raggiungere il goal.

Passiamo ora alla griglia 7x7.

Come possiamo vedere dalla Figura 20 questa volta il rewards inizia a salire rapidamente a 2000 episodi ed in precedenza rimane piatto. Interessante, inoltre, il crollo nelle prossimità di 3500 episodi.

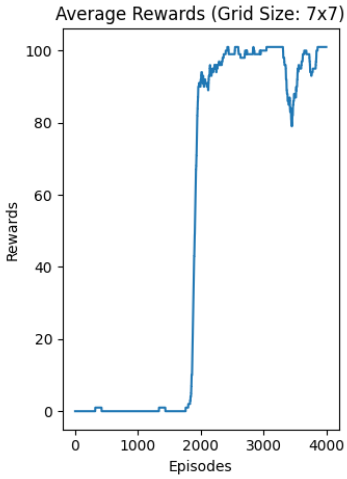

Figura 20: Reward rapportato con il numero di episodi griglia 7x7.

Infine, in Figura 21 possiamo notare come questa volta l'agente parta subito raggiungendo un numero elevato di steps, questo è attribuibile ad un'esplorazione più accurata. Successivamente il numero di steps crolla per poi risalire verso la fine degli episodi.

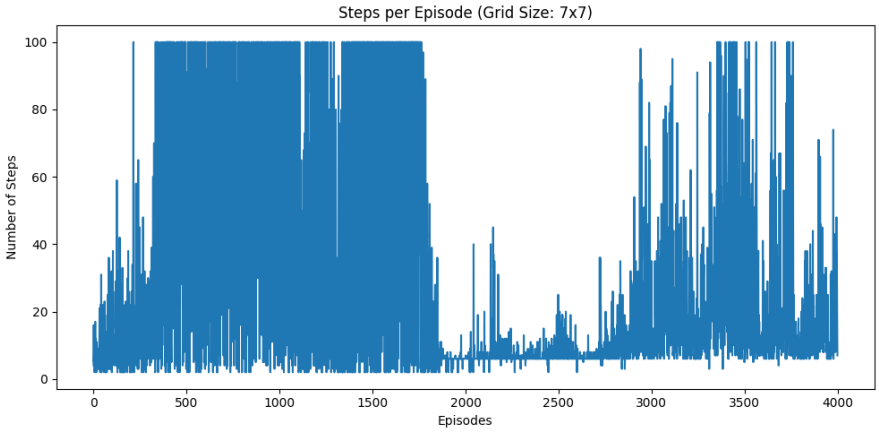

Figura 21: Numero di steps rapportati per numero episodi griglia 7x7.

4.1.4 Aggiunta Slippery

Infine, ho voluto abilitare il parametro di slippery, ovvero l'agente si muoverà nella direzione desiderata con una probabilità di 1/3, per le restanti volte si muoverà in modo perpendicolare.

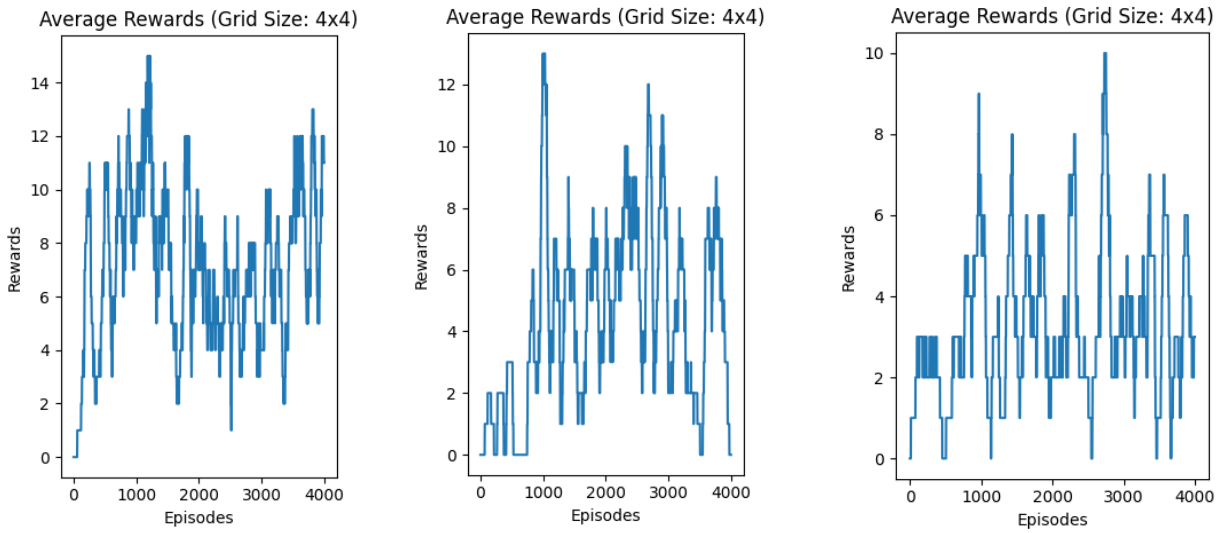

Figura 22: Rewards su episodi rispettivamente per DPQ, SARSA e PPO.

Come possiamo vedere in figura 22 l'aspetto più interessante dopo l'inserimento di questo parametro sia come per ogni algoritmo non si superi il valore 15 di reward e l'andamento sia molto oscillante rispetto a prima. Questo, evidenza sicuramente un incremento di volte in cui l'agente finisce in un buco.

4.2 Ambiente Mountain Car

Il seguente ambiente di simulazione si pone come obiettivo di fare raggiungere ad un carrello la cima di una collina. Come possiamo vedere dalla Figura 23 il carrellino inizialmente è posto in una conca.

Figura 23: Immagine ambiente di simulazione Mountain Car

Lo spazio delle azioni questa volta è continuo e rappresenta la forza direzionale applicata al carrello.

Lo spazio delle osservazioni è anche esso continuo e rappresenta un vettore di due elementi, dove il primo rappresenta la posizione del carrello sull'asse X mentre il secondo la velocità del carrello.

Il personaggio riceve un premio di 100 punti se raggiunge la cima, altrimenti viene penalizzato di -0,1*action^2 ad ogni timestamp.

La simulazione termina al raggiungere dei mille episodi.

4.2.1 Test Algoritmo SARSA

Un primo aspetto importante è stato il tempo di allenamento che si è rivelato questa volta piuttosto lungo rispetto all'ambiente discreto.

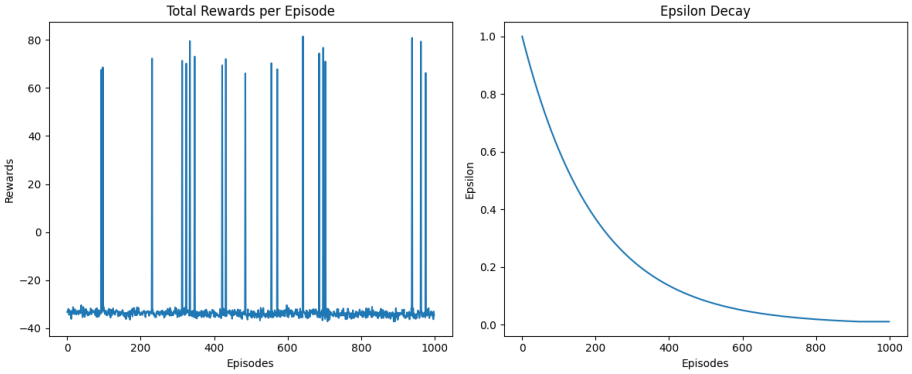

Figura 24: Rewards su episodi e decadimento epsilon per ambiente continuo.

A parte tale aspetto, dalla figura 24 possiamo notare come quasi sempre l'algoritmo venga fortemente penalizzato, infatti sembrerebbe tendere ad imporre valori piuttosto bruschi al carrello. Questa scelta a volte si rivela vincente, ma spesso no. L'algoritmo non riesce comunque a raggiungere la cima della collina.

4.2.2 Test Algoritmo PPO

Il tempo di esecuzione dell'algoritmo a pari parametri si è rivelato molto più veloce del caso precedente.

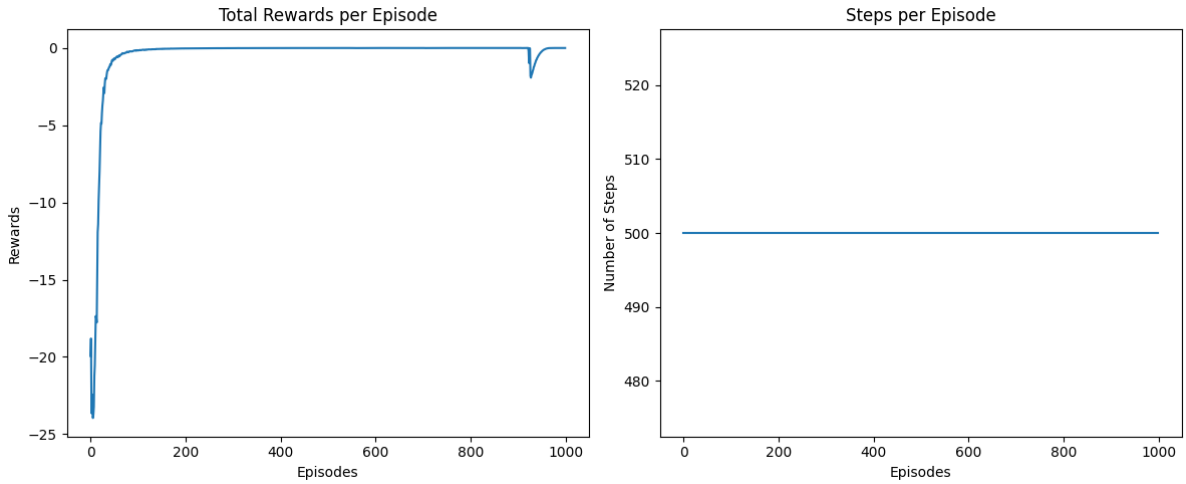

Figura 25: Rewards su episodi e numero steps per episodi per ambiente continuo.

Inoltre, dalla figura 25 possiamo notare come l'algoritmo impari abbastanza velocemente che azioni impulsive portano ad una forte penalizzazione e quindi il grafico è fortemente negativo solo negli episodi iniziali. Per i restanti episodi invece il valore di rewards rimane stabile a zero. Come indicato dal grafico a destra della Figura 25 il numero di steps impostato non si è rivelato sufficiente a studiare in modo più approfondito il comportamento dell'algoritmo negli episodi, per questo motivo la curva dell'immagine a sinistra, potrebbe superare il valore zero di rewards.

5 CONCLUSIONI

In conclusione, tutti gli algoritmi studiati si sono comportati in modo più o meno simile per il caso di un ambiente con osservazioni e stati discreti. L'algoritmo PPO ha un po' sofferto a convergere nel numero di steps impostati all'obiettivo in poche mosse.

Con l'ambiente continuo invece non sono riuscito ad applicare il replay buffer per l'algoritmo DQN ma ho potuto notare come PPO si comporti molto meglio ed in maniera più intelligente rispetto a SARSA, dove quest'ultimo tende a provare valori di azione casuali e grossolani.

6 BIBLIOGRAFIA

[1] «OpenAI». [Online]. Disponibile su: https://openai.com
[2] *Gymnasium*. (12 settembre 2024). [Online]. Disponibile su: https://gymnasium.farama.org