Implementation of Markov Decision Processes into quantum algorithms for reinforcement learning

Francesca Stefano e Antonio Signorelli

HIGH PERFORMANCE COMPUTING

1 Obiettivo del progetto

Il presente progetto si prefigge di effettuare un'analisi approfondita del paper intitolato *"Implementation of Markov Decision Processes into quantum algorithms for reinforcement learning"*, focalizzandosi sui suoi elementi chiave e conducendo una valutazione accurata dei suoi pregi e dei suoi limiti. Inoltre, attraverso l'integrazione fra lo studio dell' articolo e l'analisi del codice proposto, si mira a sviluppare esempi di codice più complessi.

2 Analisi paper

Nel lavoro presentato, si è proposta una metodologia per implementare i classici Markov Decision Process in un paradigma quantistico, come primo passo per realizzare sistemi che eseguono task di Apprendimento per Rinforzo Quantistico in cui sia l' agente che l' ambiente sono espressi come programmi quantistici. Per fare questo, si è quindi analizzato il ciclo di iterazione tra l' agente e l' ambiente nell' apprendimento per rinforzo classico, cercando un metodo per mappare un MDP con spazio di stato discreto, insieme di ricompense e azioni, in un programma quantistico.

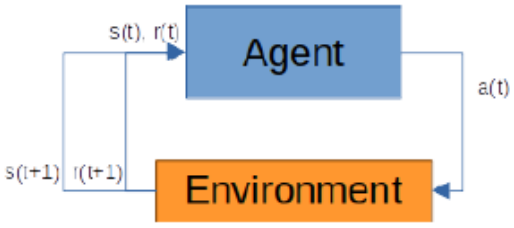


Figure 1: Ciclo di interazione

2.1 Introduzione

L'Apprendimento per Rinforzo Classico (RL) è una forma di Apprendimento Automatico in cui un agente intelligente apprende interagendo con un ambiente sconosciuto. Durante ogni iterazione tra agente e ambiente, l'ambiente si trova in uno stato **s(t)**. Successivamente, l'agente seleziona un'azione **a(t)** da un insieme di azioni disponibili e la esegue sull'ambiente, che si evolve dallo stato s(t) a **s(t+1)** e restituisce un valore scalare di ricompensa **r(t+1)** all'agente come feedback sulle sue prestazioni. L'obiettivo nell'RL è apprendere una politica $\pi(a|s)$per selezionare la migliore azione a in ogni stato s che massimizza la ricompensa accumulata a lungo termine.

Il Reinforcement Learning (RL) Classico si fonda sull'assunzione di Markov di primo ordine, dove l'ambiente è modellato come un Processo Decisionale di Markov (MDP). Un MDP è descritto da una tupla S, A, P, R, dove:

- S rappresenta un insieme di stati;
- A rappresenta un insieme di azioni;

- P è una funzione di transizione probabilistica che indica la probabilità di passare dallo stato s allo stato s' eseguendo l'azione a;
- R è una funzione di ricompensa che fornisce una valutazione scalare della ricompensa ottenuta quando si passa dallo stato s allo stato s' eseguendo l'azione a.

L'Apprendimento per Rinforzo Quantistico (QRL) si impegna a adattare i metodi tradizionali di Apprendimento per Rinforzo o a sviluppare nuove tecniche per il Calcolo Quantistico (QC). Si considerano quattro scenari diversi che combinano elementi classici e quantistici, riguardanti sia agenti che operano in ambienti classici che quantistici.

2.2 Implementazione

Il metodo generale proposto per implementare il ciclo di RL in QC, incluso il sottostante MDP, prevede i seguenti passaggi:

1. Preparazione dello stato quantistico: Il sistema inizia in uno stato iniziale dato s(t) = si, che viene codificato in uno stato quantistico $|\psi_s\rangle$ attraverso un'operazione unitaria US;
2. Selezione e codifica dell'azione: Un'azione deterministica a(t) = ak viene selezionata e codificata in uno stato quantistico $|\psi_a\rangle$ tramite un'operazione unitaria UA;
3. Evoluzione dell'ambiente: La funzione di transizione P(si, ak) fa evolvere l'ambiente dallo stato iniziale s(t) = si a un nuovo stato s(t + 1) = sj con una certa probabilità, utilizzando un'operazione unitaria controllata UT per calcolare la sovrapposizione degli stati dell'ambiente di destinazione;
4. Calcolo della ricompensa: La ricompensa r(t+1) = R(si, ak, sj) viene calcolata e codificata in uno stato quantistico $|\psi_r\rangle$ attraverso un'operazione unitaria controllata UR;
5. Restituzione delle informazioni all'agente: Lo stato successivo s(t+1) e la relativa ricompensa r(t+1) vengono restituiti all'agente, che nel caso classico può essere eseguito mediante misurazioni sui registri quantistici di stato e di ricompensa;

Tutti gli stati dell'ambiente, le azioni e le ricompense vengono memorizzati nelle ampiezze dei registri quantistici, consentendo una vasta gamma di valori possibili. Le operazioni di codifica e decodifica vengono realizzate attraverso circuiti parametrizzati contenenti porte $R_x(\theta)$ che dipendono dalla rappresentazione binaria degli stati e delle azioni. La trasformazione unitaria UT calcola la sovrapposizione degli stati di destinazione e le probabilità di misurazione, mentre la funzione di ricompensa viene implementata tramite operazioni controllate che impostano le ampiezze corrette nei registri delle ricompense quantistiche. Questo approccio consente una gestione efficiente delle informazioni e una robusta interazione tra agente e ambiente nel contesto del calcolo quantistico.

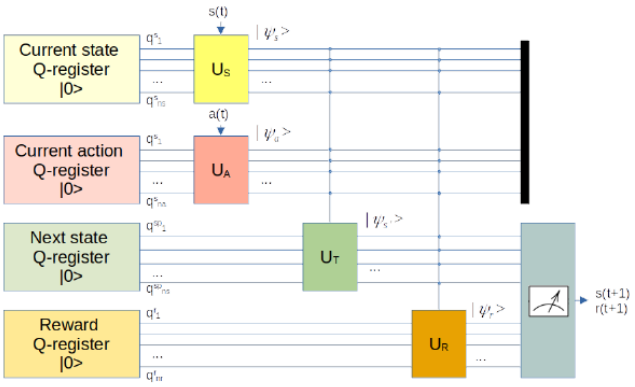


Figure 2: Implementazione di un MDP con quantum program

2.3 Proof of concept

Si è eseguito un esperimento di esempio per addestrare un agente classico utilizzando la procedura Q-Learning con $\gamma = 0.99$ in un numero massimo di T = 200 passaggi e tasso di apprendimento $\alpha = 0.2$ per testare la convergenza. Il MDP contiene quattro stati $S = \{s_1, s_2, s_3, s_4\}$, due possibili azioni per ogni stato $A = \{a_1, a_2\}$, e un insieme discreto con quattro ricompense che dipendono solo dallo stato di transizione finale $\{r(s_1) = 10, r(s_2) = -5, r(s_3) = 1, r(s_4) = -10\}$. Pertanto, il numero di qubit richiesti per rappresentare gli stati e le ricompense è ns = nr = 2. Il mapping proposto dagli stati e dalle ricompense ai vettori di base è

$$\begin{aligned} &s_1 \to |00\rangle, && r(\cdot,\cdot,s_1) \to |00\rangle, s_2 && \to |01\rangle, r(\cdot,\cdot,s_2) && \to |01\rangle, \\ &s_3 \to |10\rangle, && r(\cdot,\cdot,s_3) \to |10\rangle, s_4 && \to |11\rangle, r(\cdot,\cdot,s_4) && \to |11\rangle. \end{aligned}$$

, rispettivamente. Si assume che lo stato iniziale per eseguire l'ambiente sia s1. Per quanto riguarda le azioni, è richiesto un numero di na = 1 qubit secondo il mapping

$$\begin{aligned} a_1 &\to |0\rangle, \\ a_2 &\to |1\rangle. \end{aligned}$$

.

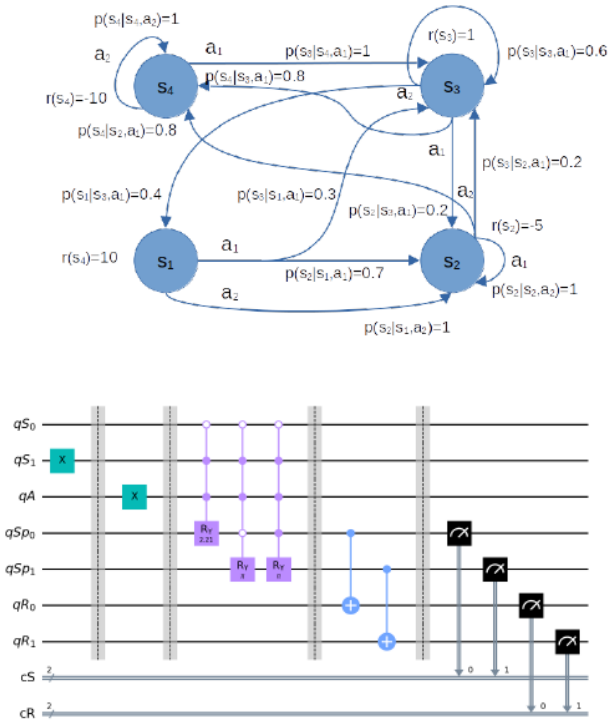


Figure 3: Esempio MPD e cricuito

3 Analisi del codice fornito

Da un punto di vista pratico, il codice fornito può essere suddiviso nei seguenti file Python sviluppati per testare le considerazioni teoriche sopracitate:

- *environments.py*: Il quale contiene un "toy example" di MDP incorporato in un ambiente classico (classe ClassicToyEnv) e in un ambiente quantistico (classe QuantumToyEnv);
- *algorithms.py*: contiene l'implementazione dei classici algoritmi Value Iteration e Q-Learning, adattati per utilizzare gli ambienti in environments.py;

- *ClassicValueIteration.py e ClassicQLearning.py* che contengono esempi di utilizzo dei metodi classici dell'iterazione del valore e del Q-Learning rispetto al classico;
- *QLearningQuantumEnv.py*:contiene esempi di utilizzo della procedura Q-Learning sull'ambiente quantistico.

3.1 environments.py

Il codice Python definisce una classe *ClassicToyEnv* che rappresenta un ambiente di gioco classico per un processo decisionale di Markov (MDP). La classe ClassicToyEnv ha diversi attributi:

- MaxSteps: Il numero massimo di interazioni ambiente/agente consentite per un episodio.
- nS, nA, nR: Il numero di stati, azioni e possibili ricompense nel MDP, rispettivamente.
- T: La funzione di transizione, rappresentata come un array numpy 3D. Definisce la probabilità di transizione da uno stato ad un altro dato un'azione.
- R: La funzione di ricompensa, rappresentata come un array numpy 1D. Definisce la ricompensa per ogni stato.
- currentState, currentStep: Lo stato corrente dell'ambiente e il numero di passaggi corrente in un episodio, rispettivamente.

La classe include anche diversi metodi:

- numberOfStates e numberOfActions: Questi metodi restituiscono il numero di stati e azioni nel MDP, rispettivamente.
- transitionProb: Questo metodo restituisce la probabilità di transizione da uno stato s e un'azione a ad un nuovo stato sp.
- rewardValue: Questo metodo restituisce il valore di ricompensa per una data transizione $(s, a) \rightarrow sp$.
- reset: Questo metodo reimposta l'ambiente al suo stato predefinito e restituisce lo stato corrente.
- step: Questo metodo esegue un'azione sull'ambiente, aggiorna lo stato corrente e il conteggio dei passaggi, e restituisce il prossimo stato e la ricompensa.
- StoppingCriterionSatisfied: Questo metodo controlla se il criterio di arresto (cioè, il conteggio dei passaggi correnti ha raggiunto o superato MaxSteps) è soddisfatto.

Viene poi sviluppata la classe QuantumToyEnv che estende ClassicToyEnv. Questa classe rappresenta un'implementazione quantistica di un ambiente classico per un Processo Decisionale di Markov (MDP).

- Il metodo init inizializza la classe con un numero massimo di passaggi per un episodio. Imposta i registri quantistici e classici per la rappresentazione dello stato, dell'azione e del premio. Inizializza anche un circuito quantistico per memorizzare un ciclo di un MDP e aggiunge varie funzioni a questo circuito. Queste funzioni includono funzioni di impostazione dello stato e dell'azione ($\mathbf{U}_s$ e $\mathbf{U}_a$), funzioni di transizione e premio ($\mathbf{U}_t$ e $\mathbf{U}_r$) e funzioni di misurazione per il prossimo stato e premio. Viene anche inizializzato il simulatore quantistico dal modulo Aer di Qiskit.
- Il metodo QSampleNextState è una funzione ricorsiva che codifica il prossimo stato dell'ambiente in base allo stato corrente e all'azione. Utilizza porte di rotazione controllate per codificare le probabilità di ciascuno stato.
- I metodi $\mathbf{U}_s$ e $\mathbf{U}_a$ vengono utilizzati per impostare rispettivamente lo stato e l'azione dell'ambiente di input. Aggiornano il circuito quantistico con i circuiti corrispondenti alla fine.
- Il metodo $\mathbf{U}_t$ implementa la funzione di transizione. Fa un ciclo su tutti gli stati e le azioni possibili e, per ogni coppia, applica la funzione di transizione al circuito quantistico.
- Il metodo $\mathbf{U}_r$ implementa la funzione di premio. Applica un NOT gate tra ogni qubit del prossimo stato e il qubit corrispondente del premio.
- Il metodo step esegue un'azione sull'ambiente e restituisce il prossimo stato e premio. Prima controlla se il criterio di arresto è soddisfatto. In caso contrario, codifica lo stato corrente e l'azione, simula il ciclo MDP e misura il prossimo stato e premio. Quindi aggiorna il conteggio dei passaggi e lo stato corrente

3.2 algorithms.py

Questo script Python contiene implementazioni di due algoritmi fondamentali di apprendimento per rinforzo: **Value Iteration** e **Q-Learning**. Entrambi gli algoritmi sono utilizzati per risolvere Processi Decisionali di Markov (MDP).

- La funzione **runEnvironment** viene utilizzata per testare una politica deterministica su un ambiente dato. Prende in input l'ambiente, la politica, il numero di iterazioni da eseguire e un flag booleano per mostrare le iterazioni. Restituisce la ricompensa totale ottenuta e il tempo impiegato;
- La funzione ValueIteration implementa l'algoritmo Value Iteration. Prende in input l'ambiente, il fattore di sconto gamma, il numero di iterazioni e la soglia di convergenza. Restituisce la funzione valore, il numero di iterazioni eseguite e un booleano che indica se l'algoritmo è convergente;
- La funzione **ExtractPolicyFromVTable** viene utilizzata per estrarre una politica da una data funzione valore. Prende in input l'ambiente, la funzione valore e il fattore di sconto gamma. Restituisce la politica ottimale;
- La funzione **eGreedyPolicy** implementa una politica epsilon-greedy. Prende in input l'ambiente, lo stato corrente, la politica dell'agente, la tabella Q e il valore epsilon. Restituisce un'azione da eseguire nell'ambiente;
- La funzione **AgentPolicy** seleziona l'azione con il massimo valore Q. Prende in input l'ambiente, lo stato corrente e la tabella Q. Restituisce un'azione da eseguire nell'ambiente;
- La funzione **QLearning** implementa l'algoritmo Q-Learning con una politica epsilon-greedy con decremento lineare di epsilon. Prende in input l'ambiente, il numero massimo di passaggi, i valori epsilon iniziali e finali, il numero di passaggi per diminuire epsilon, il tasso di apprendimento *alpha*, il fattore di sconto gamma e un flag booleano per mostrare le iterazioni. Restituisce la politica ottimale, il numero di iterazioni eseguite e un booleano che indica se l'algoritmo è convergente

3.3 QLearningQuantumEnv.py

Questo script Python è un'implementazione dell'algoritmo Q-Learning. Lo script è diviso in diverse sezioni: configurazione dell'ambiente, esecuzione del Q-Learning, visualizzazione dei risultati ed esecuzione della politica. Nella sezione di configurazione dell'ambiente, lo script importa i moduli necessari e inizializza l'ambiente quantistico utilizzando la classe QuantumToyEnv dal modulo environments. Imposta anche vari iperparametri per l'algoritmo Q-Learning, come il fattore di sconto (gamma), il numero massimo di iterazioni (MaxSteps), i valori iniziali e finali di epsilon per la politica epsilon-greedy (eps0 e epsf), il numero di passaggi per diminuire epsilon (epsSteps) e il tasso di apprendimento (alpha). La funzione QLearning restituisce la politica ottimale e il numero di iterazioni necessarie per trovarla.
La politica è una lista in cui ogni elemento rappresenta l'azione migliore da prendere per uno stato dato. Nella sezione di esecuzione della politica, lo script testa la politica ottenuta nell'ambiente utilizzando la funzione runEnvironment dal modulo algorithms. Esegue la politica per un numero specificato di iterazioni (MaxIterations) e ripete il test un numero specificato di volte (MaxTests). La ricompensa totale ottenuta in ogni test viene memorizzata in RewardSet. Risulta quindi essere un buon esempio di come implementare e testare un algoritmo Q-Learning in un ambiente personalizzato. Dimostra anche come utilizzare la politica epsilon-greedy e come regolare i suoi parametri nel tempo.

4 Pregi e difetti

Il paper propone una metodologia per implementare Processi Decisionali di Markov classici in un paradigma di Calcolo Quantistico, come primo passo per realizzare sistemi che eseguono Apprendimento per Rinforzo Quantistico dove sia l'agente che l'ambiente sono espressi come programmi quantistici. Questo approccio si basa sull'analisi del ciclo di interazione tra l'agente e l'ambiente nell'apprendimento per rinforzo classico, con l'obiettivo di mappare un Processo Decisionale di Markov con spazio di stati discreto, insieme di azioni e ricompense, in un programma quantistico.

Pregi:

1. Innovativo: Introduce un approccio innovativo per estendere l'apprendimento per rinforzo tradizionale al contesto del calcolo quantistico, aprendo nuove prospettive di ricerca e sviluppo.

2. Ponte tra campi: Fornisce un ponte tra l'apprendimento per rinforzo classico e quantistico, offrendo nuove possibilità per sfruttare le potenzialità del calcolo quantistico nell'ambito dell'apprendimento automatico.

3. Metodologia chiara: La metodologia proposta è ben definita e strutturata, consentendo una comprensione chiara dell'implementazione del ciclo di apprendimento per rinforzo in un ambiente quantistico.

4. Applicazione pratica: Mostra l'applicazione pratica dell'algoritmo Q-Learning in un ambiente quantistico utilizzando Qiskit, il che potrebbe essere un contributo significativo al campo dell'informatica quantistica applicata.

5. Disponibilità del codice sorgente: Il paper rende disponibile il codice sorgente utilizzato per le sperimentazioni, consentendo agli altri ricercatori di replicare gli esperimenti e di estendere il lavoro presentato.

Difetti:

1. Limitazioni di spazio: Il paper limita l'esperimentazione a un caso di prova.Potrebbe mancare una valutazione più ampia delle prestazioni e dell'efficacia dell'approccio proposto su una gamma più ampia di scenari e problemi.

2. Confronto limitato: Non fornisce un confronto dettagliato delle prestazioni tra l'apprendimento per rinforzo classico e quantistico su diversi scenari o problemi, che potrebbe essere utile per comprendere appieno il vantaggio quantistico.

3. Complessità implementativa: L'implementazione di un MDP quantistico richiede una comprensione avanzata della computazione quantistica e potrebbe essere complessa per gli sviluppatori meno esperti.

4. Limitazione degli stati completamente osservabili: Il paper assume che gli stati dell'ambiente siano completamente osservabili, il che potrebbe non essere realistico in molte situazioni reali. Questa assunzione semplificativa potrebbe non riflettere accuratamente la complessità delle situazioni del mondo reale, limitando così la generalizzabilità e l'applicabilità pratica dell'approccio proposto.

5. Assunzioni sull'ambiente quantistico: Il paper presume la disponibilità di un ambiente quantistico ideale e privo di rumore, che potrebbe non essere realistico nelle attuali implementazioni pratiche dei computer quantistici. Questa assunzione potrebbe rendere i risultati del paper difficili da riprodurre o generalizzare in ambienti reali con rumore e errori quantistici.

In sintesi, il paper propone un'interessante integrazione tra apprendimento per rinforzo classico e quantistico, ma potrebbe beneficiare di una valutazione più approfondita delle prestazioni e di una discussione più ampia sulle implicazioni pratiche e le sfide dell'implementazione.

5 Sviluppo esempi codice

5.1 eGreedyPolicy e DQLearning

file env2.py e algo2.py
Sulla base dello studio del paper e del codice fornito, si è proceduto con esempi implementativi più complessi per testare nuovi scenari. Per prima cosa si è pensato di andare a modificare la funzione **eGreedyPolicy** nel file di origine *algorithms.py*, aggiungendo un parametro di decadimento per epsilon, che riduce il valore di epsilon ad ogni chiamata della funzione, rendendo la politica meno casuale nel tempo. Inoltre, si è implementato anche un meccanismo di tie-breaking per gestire casi in cui ci sono più azioni con lo stesso valore Q massimo.

Un'ulteriore modifica introdotta nel file di origine `algorithms.py` riguarda dei cambiamenti introdotti nella funzione **QLearning**, trasformandola in **DQLearning**. L'introduzione di queste modifiche è stata pensata a partire dalle seguenti ipotesi:

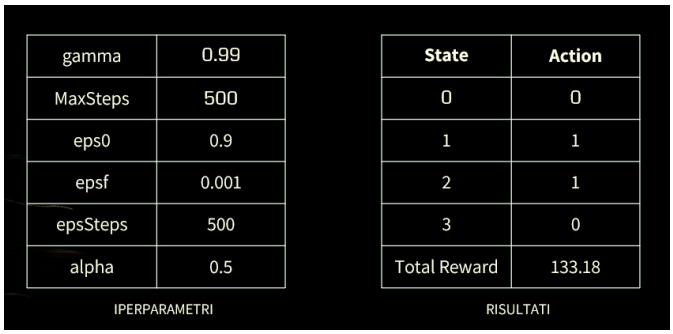
1. Implementare un algoritmo di apprendimento profondo (*Deep Q-Learning*, DQL), che è un metodo di apprendimento per rinforzo molto potente;

2. Modularità: la funzione *create model* è separata dalla funzione *DQLearning*, il che rende il codice più leggibile e riutilizzabile;

3. Uso di Deep Learning: utilizzo di un modello di rete neurale per approssimare la funzione Q. Questo permette di gestire spazi di stato e azione molto grandi che non potrebbero essere gestiti con metodi tabulari;
4. Aggiornamento del modello ad ogni passo, il che può aiutare l'agente ad adattarsi rapidamente a nuove informazioni.

In questi due primi casi di modifica, gli iperparametri dell'algoritmo di Q-Learning sono rimasti invariati rispetto a quelli forniti direttamente nel codice originale. Successivamente, per cercare di capire se potessero esserci miglioramenti, alcuni di questi sono stati modificati come, ad esempio, $\alpha = 0.5$, $\epsilon_0 = 0.9$, e sono stati aumentati il numero di *MaxTest* e *Iterations*.

Successivamente, le modifiche si sono concentrate sul file di `environments.py`, andando a manipolare le varie operazioni definite nei metodi *Us*, *uA*, *Ut*, e *Ur*, in particolare modificando le rotazioni degli assi.



gamma	0.99
MaxSteps	500
eps0	0.9
epsf	0.001
epsSteps	500
alpha	0.5

IPERPARAMETRI

State	Action
0	0
1	1
2	1
3	0
Total Reward	133.18

RISULTATI

Figure 4: Iperparametri e Risultati

5.2 MultiAgentQuantumToyEnv

file envMulti.py e algoMulti.py
Una seconda ipotesi formulata per le modifiche risluta essere l' idea di implementare una politica multi agente che ha diversi vantaggi rispetto a un ambiente di apprendimento automatico a singolo agente.

- Simulazione di scenari più realistici: In molti scenari del mondo reale, ci sono più di un agente che interagisce con l'ambiente. Ad esempio, in un gioco multiplayer o in un sistema di traffico, ci sono molteplici agenti che interagiscono tra loro. Un ambiente multiagente può simulare meglio questi scenari;
- Apprendimento cooperativo e competitivo: In un ambiente multiagente, gli agenti possono apprendere non solo dalle loro interazioni con l'ambiente, ma anche dalle interazioni con altri agenti. Questo può portare a strategie di apprendimento più complesse e potenzialmente più efficaci;
- Decentralizzazione: In un ambiente multiagente, ogni agente può operare indipendentemente, il che può portare a un sistema più robusto e resiliente. Se un agente fallisce, gli altri agenti possono continuare a operare;
- Parallelizzazione: In un ambiente multiagente, è possibile eseguire più azioni contemporaneamente, il che può portare a un' apprendimento più rapido;

Ogni agente esegue, quindi, un'azione in ogni passo di tempo e riceve un feedback (stato e ricompensa) indipendente. Questo permette agli agenti di apprendere in parallelo dalle loro esperienze individuali. Sulla base di queste modifiche, sono stati fatti dei cambiamenti nel file degli algoritmi, sottolineando in particolare le modifiche nell' algoritmo di QLearning. Nel codice specifico, la funzione MultiAgentQLearning crea una serie di politiche, una per ogni agente, utilizzando la funzione QLearning. Ogni agente apprende la propria politica indipendentemente, utilizzando la propria copia dell'ambiente. Questo significa che ogni agente può apprendere a ottimizzare il proprio comportamento senza essere influenzato dal comportamento degli altri agenti. Affinché il codice possa effettivamente funzionare è stato necessario modificare anche la funzione runEnvoirment sempre all' inetrno dello stesso file per adattarla alla nuova politica multiagente introdotta. Si vuole inoltre far notare che il codice è stato modificato per poter essere eseguito con 3 qubit per lo state representation (nel codice originale è 2), 2 qubit per l' action representation (nel codice originale è 1) e 3 qubit per la reward representation (nel codice originale è 2). Avere più qubit in ciascuna di queste categorie permette una rappresentazione più dettagliata e potenzialmente una maggiore capacità di apprendimento e di decisione.

Figure 5: Iperparametri e Risultati

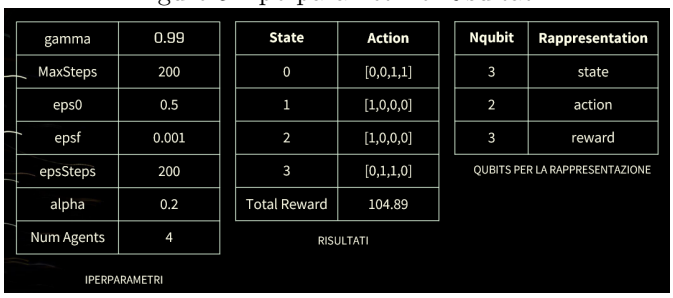


Figure 6: Iperparametri Risultati e qubits per la rappresentazione

5.3 Stati parzialmente osservabili e QuantumQLearning

file algorithms.py e environmets.py
Fra le ultime modifiche proposte per la realizzazione del progetto si è pensato di introdurre stati parzialmente osservabili per le seguenti ragioni:

1. Complessità ridotta: In molti ambienti reali, lo stato completo può essere estremamente grande o complesso. Utilizzando solo una parte dello stato, possiamo ridurre la complessità del problema di apprendimento;

2. Generalizzazione migliorata: Se alcune parti dello stato non sono rilevanti per l'azione ottimale, ignorarle può aiutare l'algoritmo a generalizzare meglio da un numero limitato di esempi;

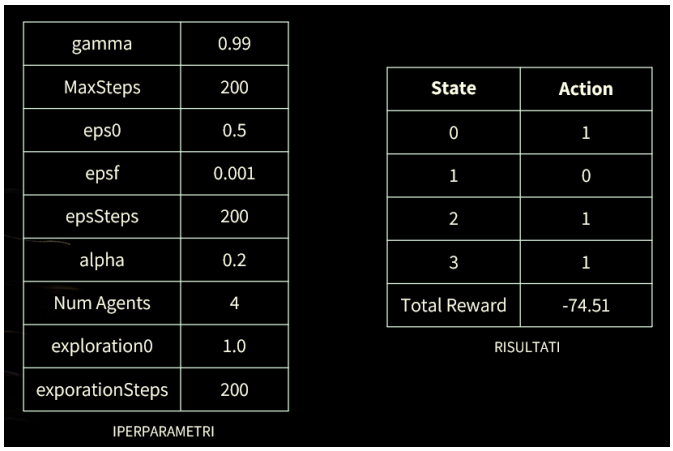


Figure 7: Iperparametri e Risultati

Invece per quanto riguarda il file della parte di algoritmi si è pensato di implementare un algoritmo di Q-Learning con l' utilizzo dell' algoritmo di Grover, che viene usato per influenzare la selezione dell' azione e per la ricerca non strutturata fornendo maggiore velocità rispetto alla ricerca classica.

6 Conclusioni e sviluppi futuri

In ultima analisi, il paper analizzato risulta essere una buona base per problemi di QRL.

- Innovazione e Integrazione: Il progetto ha esplorato l'innovativo campo dell'apprendimento per rinforzo quantistico, integrando concetti classici con metodi quantistici. L'implementazione di algoritmi di apprendimento per rinforzo in un contesto quantistico apre nuove prospettive di ricerca e sviluppo, offrendo un ponte tra l'apprendimento classico e quantistico.
- Sviluppo e Modifiche: Le proposte di sviluppo e modifiche al codice fornito hanno ampliato il campo di studio, introducendo concetti come Deep Q-Learning, politiche multi-agente e gestione di stati parzialmente osservabili. Queste modifiche non solo arricchiscono l'esperienza di apprendimento degli agenti, ma offrono anche una maggiore flessibilità e adattabilità agli ambienti di apprendimento.
- Sfide e Limitazioni: È emerso che l'implementazione di algoritmi quantistici comporta sfide complesse, come la gestione dell'entanglement, la codifica degli stati e delle azioni, e la considerazione degli errori quantistici. Inoltre, alcune delle modifiche proposte potrebbero richiedere una comprensione avanzata della computazione quantistica e potrebbero non essere immediatamente accessibili a tutti gli sviluppatori.
- Opportunità Future: Nonostante le sfide, il progetto suggerisce una serie di opportunità future per l'avanzamento dell'apprendimento per rinforzo quantistico. Queste includono la sperimentazione su scenari più complessi, l'ottimizzazione degli algoritmi per ambienti reali, e l'esplorazione di nuovi concetti e tecniche nel campo della computazione quantistica applicata all'apprendimento automatico.

In definitiva, il progetto rappresenta un passo significativo verso la comprensione e l'applicazione dell'apprendimento per rinforzo quantistico, offrendo un solido fondamento per futuri sviluppi e ricerche in questo campo promettente.