ALMA MATER STUDIORUM · UNIVERSITÀ DI BOLOGNA

SCUOLA DI SCIENZE

Corso di Laurea in Informatica

Gestione di un sistema HSS tramite Maximum Entropy Inverse Reinforcement Learning

Relatore:
Chiar.mo Prof.
Andrea Asperti

Presentata da:
Giacomo Fornaciari

Correlatore:
Chiar.mo
Francesco Mercuri

Sessione I
Anno Accademico 2024/2025

Indice

Capitolo 1

Introduzione

Negli ultimi anni, si cerca sempre maggiormente di utilizzare fonti rinnovabili per soddisfare la domanda energetica del mercato e di minimizzare lo spreco energetico. Per questo motivo, fonti come l'idrogeno stanno pian piano ottenendo più importanza per virare verso sistemi energetici più sostenibili. [1]

Infatti, l'idrogeno è al centro di numerose strategie per la decarbonizzazione, poiché consente di accumulare l'energia prodotta in eccesso da fonti intermittenti, come il solare e l'eolico. Successivamente, questa energia può essere riutilizzata quando serve, come quando la produzione non è sufficiente a coprire la domanda.

L'idea di un'economia basata sull'idrogeno non è recente, ed è stata esplorata approfonditamente da Ball et al. [2], che ne analizzano tanto le opportunità quanto le sfide strutturali e politiche ancora da superare. Inoltre, tra le molteplici tecnologie di produzione disponibili, un'analisi comparativa condotta da Nikolaidis et al. [3] evidenzia come i processi elettrochimici (come l'elettrolisi) rappresentino una delle opzioni più promettenti per una produzione sostenibile di idrogeno.

I sistemi energetici moderni stanno attraversando una fase di transizione, caratterizzata dalla crescente integrazione di fonti rinnovabili e dalla necessità di gestire in modo efficiente la variabilità della produzione e della domanda [4]. In questo scenario, il machine learning sta emergendo come uno strumento chiave per ottimizzare la gestione dell'energia soprattutto nel contesto delle smart grid [5] e migliorare la resilienza delle reti intelligenti [6].

Un esempio di utilizzo del Reinforcement Learning (RL) nella gestione energetica delle abitazioni è lo studio di Sheikhi et al. [7], che introduce il sistema Smart Energy Hub (SEH). La proposta unisce numerose sorgenti energetiche, impianti di accumulo e apparati di controllo intelligenti, ottimizzando il consumo energetico attraverso un agente addestrato con RL. Gli autori mostrano che l'approccio permette un sensibile abbassamento dei costi

e dei carichi di picco.

Inoltre l'efficacia dell'apprendimento per rinforzo nella gestione dell'energia è stata già dimostrata in scenari come il controllo intelligente di boiler elettrici [8] e in sistemi di gestione energetica distribuita [9], confermando l'utilità di agenti RL per ottimizzare decisioni locali in ambienti dinamici.

Nell'ultimo decennio, sono stati svolti vari esperimenti di uso di sofisticate tecniche di learning imitation nei contesti esistenti di gestione dell'energia, in particolare nelle stazioni di carburanti per idrogeno (HRS). Tra questi, un membro del gruppo dei procedimenti che ha presentato particolari risultati interessanti è stato il Generative Adversarial Imitation Learning (GAIL), che è stato molto utile nella sua abilità di imparare strategie decisionali complesse secondo dati esperti. In particolare, nella ricerca di Truong et al. [10], l'applicazione di GAIL ha determinato un netto aumento dell'efficienza di funzionamento.

Nonostante tutto, l'adozione di GAIL ha anche un certo livello di complessità: è basata infatti su un modello di addestramento avversario richiedente più risorse di calcolo e una progettazione più articolata rispetto ad altri metodi.

L'argomento di questa tesi riguarda la gestione economica di un sistema HSS (hydrogen storage system) con l'utilizzo di tecniche di apprendimento automatico. In particolare viene studiato un modello di inverse reinforcement learning a agente singolo che si basa sul principio di massima entropia della distribuzione sulle traiettorie svolte durante l'addestramento (Maximum Entropy IRL). Questo metodo permette di affrontare l'incertezza sul comportamento dell'agente.

Al modello vengono fornite traiettorie semi-esperte con le quali deve riuscire ad apprendere una funzione di ricompensa senza che sia fornita esplicitamente. Questo aspetto può risultare particolarmente utile in situazioni dinamiche come nel caso di sistemi energetici.

Una volta che il modello trova la funzione di ricompensa, viene utilizzato un modello di reinforcement learning per apprendere una politica ottimale su di essa.

Lo scopo della tesi è di riuscire ad addestrare un agente in grado di massimizzare l'efficienza economica di un sistema HSS rappresentato attraverso un ambiente formato da un elettrolizzatore, una cella a combustibile e un serbatoio per lo stoccaggio di idrogeno.

L'elettrolizzatore è un dispositivo a cui viene fornita energia elettrica per dividere l'acqua in ossigeno e idrogeno utilizzando un processo chiamato elettrolisi. Ogni volta che c'è più energia elettrica disponibile, ad esempio da fonti rinnovabili, viene usata per caricare l'elettrolizzatore.

Viceversa, la cella a combustibile consuma l'idrogeno stoccato e lo combina con l'ossigeno dell'aria per produrre elettricità e acqua. In questo modo si può fornire alla

rete elettrica energia prodotta direttamente a partire dall'idrogeno accumulato nel sistema HSS quando l'energia in ingresso non è sufficiente per soddisfare la domanda o quando risulta temporaneamente più costosa la produzione.

Questo meccanismo permette di accumulare energia sotto forma di idrogeno e riutilizzarla in forma elettrica quando occorre, rendendo il sistema energetico più ottimizzato e sostenibile.

L'agente viene dotato della politica per la gestione delle risorse in grado di bilanciare produzione, stoccaggio e vendita d'energia. Sarà quindi importante regolare l'attivazione e lo spegnimento dell'elettrolizzatore e della cella a combustibile in base alle situazioni verificabili e alle rewards da ottenere.

Inoltre, volendo sviluppare un sistema sostenibile, bisogna poter considerare come energia in ingresso, che alimenta l'elettrolizzatore, fonti rinnovabili come energia solare ed energia eolica. Ciò può portare ad alcune problematiche dovute all'incertezza nella previsione della disponibilità di energia, dovuta principalmente all'instabilità delle condizioni meteorologiche. Per evitare sprechi energetici è bene sapere con esattezza la quantità massima immagazzinabile di idrogeno.

Viene preferito un approccio misto tra un modello di inverse reinforcement learning e uno di reinforcement learning piuttosto che usare direttamente il secondo poiché quest'ultimo soffre di un'esplorazione iniziale instabile, che può causare costosi danni ai sistemi HSS e condizioni interne inaccettabili [11].

L'uso di un modello di IRL riduce la natura instabile di tentativi ed errori esplorativi del RL trovando una migliore politica ottimale [11].

Per verificare ciò è stato sviluppato anche un environment puramente di reinforcement learning all'interno del quale è stato addestrato un agente utilizzando l'algoritmo PPO. Le due politiche ottimali ottenute sono state confrontate in merito alle perdite di energia, alla resa economica e alla possibilità di soddisfare la domanda energetica.

Per appurare che l'algoritmo PPO fosse il più adatto ad addestrare l'agente è stato anche utilizzato un algoritmo DQN e poi messo a confronto nei risultati con il primo.

La ricerca analizzata in questo lavoro fa parte delle iniziative sostenute dal Piano Nazionale di Ripresa e Resilienza (PNRR). Infatti, è stata finanziata con fondi pubblici dedicati allo studio dell'idrogeno verde e alla transizione energetica. Queste attività in Italia sono coordinate da enti come ENEA, CNR e RSE.

L'obiettivo principale dello studio è sviluppare nuove tecniche come il reinforcement learning e l'inverse reinforcement learning, per gestire in modo più efficiente i sistemi di stoccaggio dell'idrogeno (HSS). L'idea è di migliorare la sostenibilità economica e

contribuire alla riduzione delle emissioni che possono causare cambiamenti climatici, tema cruciale per il futuro energetico dell'Italia e dell'Europa.

In questa ricerca sono stati utilizzati esclusivamente dati simulati, creati attraverso funzioni matematiche randomizzate, così da riprodurre in modo realistico la potenza energetica prodotta da fonti rinnovabili, l'andamento della domanda e i relativi prezzi di mercato.

Questa scelta è stata adottata poiché non sono disponibili dataset reali e pubblici su impianti HSS su scala industriale. Le ragioni sono legate alla riservatezza industriale, alla sicurezza operativa, e al fatto che sperimentare algoritmi ancora in fase di apprendimento direttamente su impianti reali, comporterebbe rischi tecnici ed economici troppo elevati.

Nel capitolo 2 si vogliono introdurre i principali componenti del sistema HSS e la loro realizzazione. Successivamente viene presentato come sono stati sviluppati i due environment di RL e IRL, focalizzando l'attenzione sulla descrizione del modello IRL adottato, e successivamente sull'addestramento degli agenti. Inoltre sono definite le varie tecnologie e strumenti utilizzati.

Nel capitolo 3 si vogliono mostrare i risultati e le valutazioni relative ai due modelli addestrati e al loro confronto.

Nel capitolo 4 sono state inserite le conclusioni relative al lavoro svolto e come può essere integrato maggiormente in futuro.

Capitolo 2

Implementazione del sistema e delle strategie di ottimizzazione

2.1 Introduzione

In questo capitolo vengono illustrate le scelte tecniche e progettuali per lo sviluppo dell'ambiente simulato e dell'algoritmo di Inverse Reinforcement Learning utilizzato durante la sperimentazione. Viene introdotta la struttura del sistema da modellare e le sue caratteristiche principali, come stati, azioni e funzioni di ricompensa. Successivamente si descrive in modo dettagliato i metodi di RL utilizzati specificandone l'idea alla base e le varie fasi dell'addestramento. L'obiettivo di questo capitolo è di offrire una visione chiara e comprensibile del processo di implementazione, in modo da specificare le scelte adottate e dare la possibilità di replicare facilmente il lavoro svolto.

2.2 Architettura generale del sistema HSS

Il sistema HSS è principalmente formato da 3 componenti:

- Elettrolizzatore
- Serbatoio per l'idrogeno
- Cella a combustibile

Il modello è a agente singolo per rendere inferiore la complessità del sistema.

L'idrogeno viene immagazzinato allo stato gassoso e quindi viene misurato in m^3.

La quantità di idrogeno che viene prodotta dall'elettrolizzatore quando è in funzione è la seguente:

$$\mathrm{H} = \mathrm{P} \cdot \kappa_{H_2} \cdot \eta \tag{2.1}$$

dove:

- H è la quantità di idrogeno prodotta $[m^3]$,
- P è la potenza elettrica in ingresso $[kWh]$,
- κ_{H_2} rappresenta una costante di conversione dalla potenza elettrica alla quantità di idrogeno prodotta e corrisponde indicativamente a 0.2 $[m^3/kWh]$
- η è l'efficienza dell'elettrolizzatore $[0 \leq \eta \leq 1]$.

Invece, la quantità di potenza elettrica prodotta da una cella a combustibile partendo dall'idrogeno immagazzinato è la seguente:

$$H_{\text{richiesto}} = \frac{D_{\text{elec}}}{c_{\mathrm{H_2}} \cdot \eta} \tag{2.2}$$

dove:

- $H_{\text{richiesto}}$: quantità di idrogeno necessaria $[m^3]$,
- D_{elec}: domanda elettrica $[kWh]$,
- $c_{\mathrm{H_2}}$: consumo specifico di idrogeno che corrisponde circa a 3.0$[m^3/kWh]$,
- η: è l'efficienza della cella $[0 \leq \eta \leq 1]$.

La potenza elettrica in ingresso al sistema proviene da fonti eoliche e viene misurata in kWh. Poiché la produzione eolica è fortemente influenzata da fattori atmosferici variabili, come la velocità del vento, la turbolenza e la pressione, essa si caratterizza per una forte variabilità.

Per tenere conto di questa incertezza nel modello, si è scelto di simulare la potenza disponibile tramite una funzione sinusoidale randomizzata, capace di riprodurre l'alternanza tipica tra fasi di maggiore e minore disponibilità energetica, come accade nei cicli giornalieri o stagionali.

Tale funzione è determinata dalla seguente equazione:

$$\mathrm{P}(t) = \frac{\text{max\_P} + \text{min\_P}}{2} + \frac{\text{max\_P} - \text{min\_P}}{2} \cdot \sin\left(\frac{2\pi}{\text{period}} \cdot t\right) \tag{2.3}$$

Sono stati inseriti anche dei valori di massima e minima potenza per far sì che i valori generati siano all'interno di un determinato intervallo. Il periodo simula una quantità di tempo reale pari a 10 ore.

Invece, la domanda energetica viene generata in maniera randomica seguendo una funzione gaussiana con intervalli uguali alla funzione sinusoidale della potenza in input. L'utilizzo della distribuzione gaussiana per generare il carico energetico è dovuto dall'esigenza di riprodurre una variabilità continua e reale dei carichi energetici. In realtà, l'elevato numero dei fattori che interessano il consumo elettrico in un sistema lo porta ad assumere comportamenti associabili a una distribuzione normale, nell'approssimare l'insieme delle variabili che lo influenzano. Inoltre, l'uso della gaussiana permette di controllare media e deviazione standard del segnale di richiesta, consentendo così la simulazione di condizioni di picco, carico medio o bassa domanda. La funzione utilizzata è la seguente:

$$\mathrm{X} = \mathrm{TruncNorm}\left(\frac{\min - \mu}{\sigma},\ \frac{\max - \mu}{\sigma};\ \mu = \frac{\min + \max}{2},\ \sigma = \frac{\max - \min}{2}\right) \tag{2.4}$$

Si tratta di una distribuzione normale troncata con un minimo e un massimo per far sì che i valori siano compresi in un intervallo definito.

2.3 Problema decisionale di Markov

L'interazione tra agente a ambiente nel progetto è stata descritta da un problema decisionale di Markov (MDP). E' un framework matematico per modellare decisioni sequenziali con l'obiettivo di massimizzare una ricompensa cumulativa nel tempo.

Un MDP è definito da un insieme di elementi chiave:

1. Stati S:
    - Rappresentano tutte le possibili configurazioni in cui l'ambiente può trovarsi.
2. Azioni A:
    - Rappresentano le decisioni che l'agente può prendere in ogni stato.
3. Transizioni $P(s'|s, a)$:
    - Definiscono la probabilità di passare da uno stato s a un nuovo stato s', eseguendo un'azione a.
    - Questo elemento riflette la dinamica dell'ambiente.

4. Ricompense $R(s, a, s')$:

    - Rappresentano il guadagno immediato che l'agente riceve eseguendo un'azione a nello stato s e finendo nello stato s'.
    - La ricompensa guida l'agente a prendere decisioni ottimali.

5. Fattore di sconto γ:

    - Un valore compreso tra 0 e 1 che determina l'importanza delle ricompense future rispetto a quelle immediate.
    - Ad esempio, γ=0.9 dà più peso alle ricompense immediate, ma considera anche quelle future.

Nel mio sistema HSS, lo stato s è rappresentato da un vettore continuo di 7 valori, che descrive le principali grandezze operative del sistema HSS:

- Quantità di idrogeno stoccato $[0 - 500]$ m^3
- Potenza elettrica in entrata $[0 - 600]$ kWh
- Prezzo di vendita dell'elettricità $[1 - 5]$ €/kWh
- Prezzo dell'idrogeno $[5 - 10]$ €/m^3
- Domanda elettrica $[0 - 600]$ kWh
- Stato dell'elettrolizzatore (0: spento, 1: acceso)
- Stato della cella a combustibile (0: spenta, 1: accesa)

I valori massimi della potenza elettrica in entrata e della domanda elettrica sono poi stati aumentati per verificare se, con quantità maggiori e quindi più complesse da gestire, i modelli funzionassero ugualmente e se differissero nella politica ottimale adottata.

I valori del prezzo dell'idrogeno e dell'elettricità vengono generati casualmente ad ogni step di addestramento all'interno di intervalli prestabiliti.

Gli stati sono continui poiché rappresentano grandezze fisiche che variano su scale numeriche dettagliate e non su valori discreti. Questa scelta permette di simulare in modo più realistico il comportamento del sistema e di fornire all'agente dati più precisi su cui basare l'apprendimento.

Per le azioni è stato scelto di usare un array discreto con 5 valori booleani che indicano se l'azione viene eseguita (1) o è disabilitata (0):

- Produzione di idrogeno
- Vendita di potenza generata dall'idrogeno stoccato
- Vendita diretta della potenza in input nel sistema
- Blocco della produzione di idrogeno
- Blocco della vendita di potenza generata dall'idrogeno

Inizialmente le azioni scelte erano discrete, ovvero poteva verificarsi una sola azione ad ogni passo. Nonostante i risultati fossero buoni il sistema non era del tutto realistico poiché il sistema deve poter essere in grado di svolgere anche azioni contemporaneamente, come ad esempio, la vendita di potenza in ingresso e la produzione di idrogeno usando la potenza rimanente. Un altro caso è la vendita contemporanea della potenza in ingresso e dell'energia prodotta da idrogeno nel caso in cui la domanda non possa essere soddisfatta.

Quindi è stato necessario l'impiego di un campo di azioni multidiscrete. Per escludere azioni che non possono verificarsi contemporaneamente è stata data una reward negativa nel caso che il modello scelga tali combinazioni.

Le transizioni e il fattore di sconto γ vengono definiti direttamente dal modello RL utilizzato per ottenere la policy ottimale, ovvero il Proximal Policy Optimization (PPO). Il valore di γ prestabilito è pari a 0.99 quindi, nonostante la ricompensa immediata pesi di più, le ricompense future hanno un impatto significativo.

Le ricompense dipendono dall'environment utilizzato. L'ambiente RL permette di dare una ricompensa diretta ad ogni passo in base ai valori ottenuti grazie alle azioni scelte.

Al contrario, nell'ambiente IRL le ricompense non possono essere direttamente assegnate alle azioni. Saranno invece direttamente proporzionali a quanto il modello riesce a seguire il comportamento delle traiettorie esperte.

2.4 Environment RL

Nel modello sviluppato, la funzione di ricompensa è fondamentale per guidare l'apprendimento dell'agente, poiché stabilisce il criterio con cui le sue azioni vengono valutate a ogni passo temporale. In particolare, l'obiettivo è quello di massimizzare il profitto netto del sistema HSS, tenendo conto sia dei ricavi derivanti dalla vendita di energia, sia delle penalizzazioni legate a perdite o mancata soddisfazione della domanda.

La ricompensa è calcolata secondo la seguente formula:

$$\text{reward} = R_{\text{vendite}} - R_{\text{perdite}} - R_{\text{nosodd}} \tag{2.5}$$

dove:

- R_{vendite} rappresenta i ricavi ottenuti dalla vendita dell'energia elettrica, sia quella generata da fonti rinnovabili sia quella prodotta tramite la cella a combustibile.
- R_{perdite} penalizza l'energia disponibile che non viene utilizzata, calcolata come energia persa moltiplicata per il prezzo dell'elettricità.
- R_{nosodd} tiene conto della parte di domanda che non viene soddisfatta, anch'essa penalizzata in funzione del prezzo dell'elettricità.

Questi valori vengono aggiornati dinamicamente in base allo stato del sistema e alle azioni selezionate. Il calcolo considera la produzione effettiva, l'energia venduta, le eventuali perdite per inattività e i deficit rispetto alla domanda elettrica prevista. Non viene però considerato il caso in cui la domanda non possa essere soddisfatta dalla potenza totale nel sistema, poiché porterebbe il modello all'eccessivo accumulo iniziale di idrogeno non permettendo la vendita della potenza in input a discapito dei guadagni economici. Nel caso in cui vengano eseguite combinazioni di azioni non valide; ad esempio, attivare e bloccare contemporaneamente la produzione; l'agente riceve una penalità fissa pari a:

$$\text{reward} = -100 \tag{2.6}$$

Questa struttura di ricompensa consente all'agente di apprendere un comportamento orientato all'efficienza economica, minimizzando sprechi e penalità, e ottimizzando l'utilizzo delle risorse disponibili per soddisfare la domanda energetica.

2.5 Maximum Entropy IRL

Il Maximum Entropy Inverse Reinforcement Learning (MaxEnt IRL) è un metodo per apprendere una funzione di ricompensa a partire da dimostrazioni esperte, mantenendo il principio di massima entropia per gestire l'incertezza sui comportamenti.

I metodi classici di IRL cercano di trovare una funzione di ricompensa i cui valori medi delle features coincidano con quelli delle traiettorie esperte. [12] [13]

Tuttavia:

- più reward possono generare lo stesso comportamento,
- i dati possono essere rumorosi o imperfetti,
- non è garantita l'unicità della soluzione.

Per superare questi limiti, Ziebart et al. propongono l'utilizzo del metodo di Maximum Entropy Inverse Reinforcement Learning [14].

I vantaggi principali di usare il Maximum Entropy IRL sono:

- Evita soluzioni arbitrarie (zero-reward o overfitting),
- Gestisce dati rumorosi e traiettorie sub-ottimali,
- Produce una politica stocastica robusta,
- Ha una funzione obiettivo convessa e ottimizzabile con metodi standard,
- Si integra bene con modelli probabilistici complessi (es. obiettivi nascosti, destinazioni ignote).

Oltre alla formulazione classica di Ziebart et al., Wulfmeier et al. (2015) hanno esteso il paradigma di massima entropia all'uso di reti neurali profonde, dimostrando la capacità di modellare reward complesse in grandi spazi di stato con una complessità computazionale più controllata [15].

Il MaxEnt IRL assume che l'esperto segua una distribuzione probabilistica sulle traiettorie proporzionale all'esponenziale della loro ricompensa totale (distribuzione di Boltzmann):

$$P(\tau) \propto e^{R(\tau)} \tag{2.7}$$

dove $R(\tau)$ è la somma delle ricompense lungo la traiettoria:

$$R(\tau) = \sum_{t=1}^{T} R(s_t) \tag{2.8}$$

dove:

- τ indica la traiettoria considerata,
- T è la lunghezza della traiettoria (numero di step),
- s_t è lo stato visitato al tempo t,
- $R(s_t)$ è la ricompensa associata allo stato s_t.

Questo evita di assumere che l'esperto agisca in modo completamente deterministico, permettendo anche piccole variazioni nelle traiettorie.

Per impedire che il modello faccia assunzioni troppo rigide sulle dimostrazioni, si massimizza l'entropia della distribuzione sulle traiettorie:

$$\max H(P) = -\sum_{\tau} P(\tau) \log P(\tau) \tag{2.9}$$

soggetto al vincolo che la distribuzione delle features osservate nelle traiettorie generate dal modello deve essere uguale a quella osservata nei dati esperti.

La probabilità di una traiettoria τ può essere scritta come:

$$P(\tau) = P(s_1) \prod_{t=1}^{T} P(a_t \mid s_t) P(s_{t+1} \mid s_t, a_t) \tag{2.10}$$

in cui:

- $P(s_1)$ è la probabilità di iniziare nello stato s_1 (distribuzione iniziale),
- $\prod_{t=1}^{T}$ è il prodotto su tutti i passi temporali da $t = 1$ a T,
- $P(a_t \mid s_t)$ è la probabilità che l'agente scelga l'azione a_t nello stato s_t, secondo la propria politica,
- $P(s_{t+1} \mid s_t, a_t)$ è la probabilità che l'ambiente transiti nello stato s_{t+1} dopo aver eseguito l'azione a_t nello stato s_t.

La probabilità dell'azione in uno stato è modellata come:

$$P(a_t \mid s_t) = \frac{e^{Q(s_t, a_t)}}{\sum_{a'} e^{Q(s_t, a')}} \tag{2.11}$$

dove:

- $Q(s_t, a_t)$ è il valore dell'azione, cioè il valore atteso cumulato eseguendo l'azione a_t nello stato s_t,
- $e^{Q(s_t, a_t)}$ è l'esponenziale del valore dell'azione, utilizzato per dare maggiore peso alle azioni con valore più alto,
- $\sum_{a'} e^{Q(s_t, a')}$ è la normalizzazione su tutte le azioni possibili nello stato s_t.

Questa funzione permette di bilanciare esplorazione ed exploitazione, assegnando probabilità più alte alle azioni più promettenti ma mantenendo una scelta casuale. I pesi w

vengono appresi massimizzando la log-verosimiglianza delle traiettorie dimostrate:

$$\max_{w} \sum_{\tau \in D} \log P(\tau) \tag{2.12}$$

Questo significa che il modello aggiorna w fino a che le statistiche delle features delle traiettorie generate corrispondono a quelle osservate nelle dimostrazioni.

Per fare questo si può usare un gradiente ascendente:

$$\frac{\partial}{\partial w} \sum_{\tau \in D} \log P(\tau) = E_D[\phi(s)] - E_P[\phi(s)] \tag{2.13}$$

in cui:

- $\tau \in D$ è l'insieme delle traiettorie esperte osservate,
- $\phi(s)$ è il vettore delle features associate allo stato s,
- $E_D[\phi(s)]$ è la media delle features osservate nelle traiettorie esperte,
- $E_P[\phi(s)]$ è la media delle features secondo la distribuzione del modello corrente.

Data la mancanza di dati reali da utilizzare per addestrare il modello, le traiettorie usate non sono del tutto esperte, bensì è stata sviluppata una policy esperta euristica che decide le azioni ottimali in base a soglie su domanda, prezzo e capacità di stoccaggio. Questa policy è utilizzata per generare traiettorie esperte, ovvero sequenze di coppie stato-azione, salvate in un file .csv che serve da base per il training IRL.

L'algoritmo Maximum Entropy Inverse Reinforcement Learning funziona su uno spazio degli stati discreto, perché la stima delle probabilità lungo le traiettorie è definita solo su un insieme finito di stati. Tuttavia, nel mio ambiente, lo stato del sistema è descritto da variabili continue che rappresentano valori su intervalli reali.

Per adattare l'IRL, si è dovuto discretizzare lo spazio degli stati, suddividendo ciascuna variabile in intervalli (bins). In questo modo, ogni valore continuo viene assegnato a una classe discreta, creando una griglia che approssima lo spazio continuo.

Essendo che lo stato è formato da valori prodotti casualmente ad ogni passo, come la potenza in input e la domanda, non è necessario inserire tali valori nelle traiettorie esperte. Infatti, si rischia solo di depistare il modello in caso che non venga ricoperto tutto lo spazio degli stati, richiedendo un eccessivo numero di traiettorie esperte che rallenterebbero l'addestramento.

Invece, i valori che è importante analizzare nelle traiettorie esperte sono la quantità di idrogeno stoccato, la potenza persa e la domanda non soddisfatta. Se le traiettorie sono

veritiere la potenza persa e la domanda non soddisfatta saranno ad ogni passo pari a 0. In questo modo il modello trova una funzione di ricompensa in grado di stabilizzare la quantità di idrogeno stoccato in base ai dati in input e a minimizzare le perdite e i deficit della domanda. In caso gli ultimi due valori non siano nulli o vicini a 0, il modello impara a dare in automatico una reward negativa.

Per ottenere delle ricompense abbastanza precise da permettere un addestramento ottimale del modello, vengono create un totale di 300 traiettorie esperte dalla lunghezza di 5000 passi. I valori iniziali di quantità di idrogeno stoccato, potenza in ingresso e domanda energetica, vengono generati in maniera casuale per far sì che le traiettorie esplorino maggiormente il campo di stati possibili.

Altrimenti, il modello si concentrerebbe più sui valori iniziali delle traiettorie dandogli ricompense maggiori.

Quindi, il metodo Maximum Entropy IRL crea delle ricompense apprese per ogni intervallo di valori di idrogeno. Tali valori saranno per la maggior parte valori negativi tranne per gli intervalli in cui il modello si deve concentrare. Essendo che la potenza e la domanda in input sono valori casuali, è facile che il modello in alcuni passi esca dal range di idrogeno stoccato designato ricevendo una reward negativa. Questo comportamento può portare il modello a cercare sempre nuove traiettorie non permettendogli di convergere. Per evitare tale problema è stata applicata una trasformazione in modo che tutte le reward risultino positive:

$$R'_i = R_i - \min(R) \quad \text{per ogni } R_i \in R \tag{2.14}$$

L'algoritmo quindi:

1. crea o riceve delle traiettorie esperte precedentemente generate,
2. assegna reward casuali agli stati iniziali,
3. stima una politica probabilistica tramite softmax,
4. aggiorna le reward per ridurre la differenza tra la distribuzione esperta e quella stimata,
5. normalizza e raggruppa le reward in range per semplificare l'analisi.

Il risultato è una mappa stato-reward, che guida l'agente PPO nell'ottimizzazione del comportamento simulato.

2.6 Apprendimento dell'agente tramite PPO

Per quanto riguarda l'addestramento di agenti in ambienti complessi e continui, una delle tecniche più recenti e affidabili è il Proximal Policy Optimization (PPO), introdotto da Schulman et al. [16].

Proximal Policy Optimization (PPO) è una famiglia di algoritmi di ottimizzazione per il reinforcement learning. Il suo obiettivo è di combinare l'efficienza dei metodi di policy gradient con la stabilità garantita dagli approcci a trust region, come TRPO (Trust Region Policy Optimization), mantenendo al tempo stesso una maggiore semplicità implementativa. La loss function di base da cui parte PPO è quella classica del policy gradient:

$$L^{PG}(\theta) = E_t[\log \pi_\theta(a_t \mid s_t) \cdot \hat{A}_t] \tag{2.15}$$

in cui:

- $L^{PG}(\theta)$ è la policy gradient loss function che vogliamo massimizzare,
- θ sono i parametri della politica ovvero i pesi del modello,
- E_t è il valore atteso su tutti i passi temporali t di un episodio,
- $\log \pi_\theta(a_t \mid s_t)$ è il calcolo logaritmico sulla probabilità che l'agente allo stato s_t scelta l'azione a_t. Il logaritmo serve per rendere il calcolo del gradiente più stabile e trattabile.
- $\hat{A}_t$ è il vantaggio stimato al tempo t, ovvero una misura di quanto l'azione scelta sia stata migliore rispetto alla media attesa in quello stato.
  - Se $\hat{A}_t > 0$ allora quell'azione è meglio della media e il gradiente rafforzerà la probabilità di sceglierla.
  - Se $\hat{A}_t < 0$ allora è peggiore della media quindi il gradiente cercherà di ridurre la sua probabilità.

Gli algoritmi di policy gradient classici aggiornano i parametri di una politica stocastica massimizzando una funzione obiettivo basata sull'esperanza del prodotto tra il gradiente logaritmico della politica e una stima della funzione vantaggio.

Tuttavia, questi metodi possono produrre aggiornamenti instabili, soprattutto quando si effettuano più passi di ottimizzazione a partire dalla stessa traiettoria.

PPO affronta questo compromesso introducendo un approccio di ottimizzazione più semplice e robusto, fondato su una funzione obiettivo con clipping:

$$r_t(\theta) = \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)} \tag{2.16}$$

$$L^{\text{CLIP}}(\theta) = \mathbb{E}_t \left[\min \left(r_t(\theta) \cdot \hat{A}_t,\ \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon) \cdot \hat{A}_t \right) \right] \tag{2.17}$$

dove:

- $r_t(\theta)$ rappresenta quanto la nuova politica si discosta dalla vecchia,
- ϵ è il parametro che definisce un intervallo entro cui sono accettati piccoli cambiamenti della policy.

La funzione di clip serve proprio a limitare $r_t(\theta)$ impedendo che gli aggiornamenti siano troppo aggressivi.

Quindi:

- Se l'azione scelta è stata utile $(\hat{A}_t)$ e la nuova politica non cambia troppo, allora l'aggiornamento viene accettato,
- Se la nuova politica cambia troppo allora la funzione limita il rischio di instabilità.

Questo meccanismo permette all'agente di apprendere in modo più stabile e regolare, anche quando l'ottimizzazione viene eseguita su più episodi consecutivi.

PPO è stato validato su diversi ambienti dimostrandosi efficace, stabile e semplice da implementare. La sua capacità di ottenere buone performance con un ridotto numero di iperparametri lo ha reso uno degli algoritmi più diffusi nella pratica del reinforcement learning moderno.

2.7 Apprendimento dell'agente tramite DQN

Il Deep Q-Network(DQN), introdotto da Mnih et al. nel 2013 [17] [18], è un algoritmo sviluppato per superare alcune limitazioni del Q-learning classico, soprattutto nei casi in cui l'ambiente ha uno spazio degli stati ampio o continuo. Il Q-learning tradizionale utilizza una tabella in cui vengono memorizzati i valori $Q(s, a)$, che rappresentano la stima di ritorno atteso eseguendo una certa azione a in uno stato s. Però questa soluzione diventa inutile se gli stati sono troppi, o se variano in modo continuo.

Il DQN affronta questo problema usando una rete neurale al posto della tabella. La rete è addestrata per approssimare la funzione Q, e riceve in ingresso lo stato corrente

dell'ambiente. Come output, restituisce i valori $Q(s, a; \theta)$ per tutte le azioni possibili, dove θ indica i parametri (i pesi) della rete. In questo modo, anche in ambienti molto complessi, l'agente riesce comunque ad apprendere una buona strategia, sfruttando la capacità della rete di generalizzare tra stati simili.

L'addestramento della rete neurale si basa sull'equazione di Bellman, che definisce una relazione ricorsiva tra i valori delle coppie stato-azione. L'equazione di Bellman per la funzione di valore ottimale assume la forma:

$$Q(s, a) = \mathbb{E}_{s'} \left[r + \gamma \max_{a'} Q(s', a') \right]$$

dove:

- s è lo stato corrente,
- a è l'azione eseguita,
- r è la ricompensa immediata ricevuta,
- s' è lo stato successivo dopo aver eseguito l'azione,
- $\gamma \in [0, 1]$ è il fattore di sconto che bilancia ricompense presenti e future,
- a' rappresenta le possibili azioni nello stato futuro.

Questa equazione afferma che il valore $Q(s, a)$ deve essere uguale alla somma della ricompensa ricevuta più il valore atteso massimo ottenibile nello stato successivo s'. In pratica, l'algoritmo cerca di apprendere una funzione Q che rispetti questa relazione, aggiornando i valori osservati per ridurre la differenza tra il lato sinistro e il lato destro dell'equazione.

Nel DQN, questa idea viene applicata usando una rete neurale per approssimare la funzione Q, e l'apprendimento si basa sulla minimizzazione della differenza (errore quadratico) tra il valore corrente e il target calcolato secondo l'equazione di Bellman.

Ogni transizione osservata dall'ambiente viene descritta da una quaterna (s, a, r, s'), ovvero stato attuale, azione eseguita, ricompensa ricevuta e nuovo stato. Per ogni quaterna viene calcolato un valore target y:

$$y = r + \gamma \max_{a'} Q(s', a'; \theta^-) \tag{2.18}$$

in cui:

- r è la ricompensa immediata ricevuta dopo l'azione.

- $\gamma \in [0, 1]$ è il fattore di sconto che bilancia l'importanza delle ricompense future rispetto a quelle immediate. Più è vicino a 1, maggiore sarà l'importanza delle nuove esplorazioni.

- $Q(s', a'; \theta^-)$ è il valore massimo previsto per lo stato per lo stato successivo s', calcolato usando una rete neurale con parametri θ^-, che viene aggiornata meno frequentemente per evitare instabilità durante l'apprendimento.

Il modello cerca quindi di minimizzare la differenza tra il valore predetto della rete corrente e il target y. Quindi minimizza la seguente funzione di perdita:

$$L(\theta) = E_{(s,a,r,s')\sim D}[(y - Q(s, a; \theta))^2] \tag{2.19}$$

dove D è un buffer chiamato experience replay, che memorizza una grande quantità di transizioni passate. Durante l'addestramento, i dati vengono prelevati in modo random da questo buffer, in modo da rompere la dipendenza temporale tra le esperienze e stabilizzare il processo di aggiornamento.

Per decidere quale azione eseguire, DQN utilizza una strategia detta ϵ-greedy. In pratica con probabilità ϵ l'agente sceglie un'azione casuale di esplorazione o con probabilità $1 - \epsilon$ seleziona l'azione che massimizza il valore stimato $Q(s, a)$. All'inizio dell'addestramento ϵ è impostato a un valore alto per favorire l'esplorazione dell'ambiente, ma viene progressivamente ridotto nel tempo per privilegiare l'utilizzo delle conoscenze apprese. Sono state necessarie alcune modifiche all'ambiente basato su Maximum Entropy Inverse Reinforcement Learning per poter utilizzare l'algoritmo DQN.

Infatti è stato necessario convertire lo spazio delle azioni da una rappresentazione MultiDiscrete a una Discrete. Originariamente, le azioni erano rappresentate come una combinazione di cinque azioni binarie indipendenti (produrre, vendere idrogeno, vendere elettricità, bloccare la produzione, bloccare la vendita), codificate come MultiDiscrete([2, 2, 2, 2, 2]), per un totale di 2^5 = 32 combinazioni possibili. Tuttavia, l'implementazione classica di DQN fornita da Stable-Baselines3 supporta solo spazi d'azione discreti. Per risolvere il problema, ogni combinazione binaria è stata codificata come un intero tra 0 e 31, creando così uno spazio d'azione Discrete(32). Durante l'interazione dell'agente con l'ambiente, l'intero viene riconvertito nella sua rappresentazione binaria a 5 bit, così da ricostruire l'azione originale. Questo accorgimento ha permesso di mantenere la struttura logica delle azioni complesse pur rispettando le limitazioni dell'algoritmo DQN.

Inoltre, la reward negativa in caso l'agente trovasse un nuovo stato non presente nelle traiettorie esperte, è stata abbassata da -10 a -6 poiché risultava troppo bassa e il

modello tendeva a trovare una politica subottimale pur di non provare nuovi tentativi di esplorazione.

Infine, per rendere il processo di apprendimento più robusto e realistico, è stata introdotta una quota di traiettorie subottimali, pari al 20% del totale. Queste traiettorie rappresentano comportamenti meno efficaci rispetto alla politica esperta e includono, ad esempio, azioni casuali o strategie non produttive.

L'aggiunta di questi esempi ha l'obiettivo di fornire all'algoritmo IRL un termine di paragone tra comportamenti efficienti e inefficaci, consentendo di apprendere una funzione di ricompensa in grado di distinguere meglio tra stati desiderabili e indesiderabili. Ciò si è rivelato particolarmente utile in fase di addestramento con l'algoritmo DQN, in quanto una funzione di reward più informativa e ben bilanciata ha favorito una convergenza più stabile e coerente verso una politica ottimale.

2.8 Strumenti e tecnologie utilizzate

Per sviluppare e testare l'agente all'interno di un ambiente simulato, ho utilizzato Gymnasium, una libreria ampiamente adottata nel campo del reinforcement learning. Gymnasium (in precedenza conosciuta come OpenAI Gym) offre un'interfaccia standard per definire ambienti in cui un agente può osservare uno stato, compiere un'azione e ricevere una ricompensa in risposta. Questo paradigma è noto come loop agente-ambiente, ed è alla base di gran parte degli algoritmi di apprendimento per rinforzo.

Gym si occupa di tenere traccia dello stato attuale, gestisce il passaggio al nuovo stato dopo un'azione e permette di "resettare" l'ambiente per iniziare una nuova simulazione.

Questo permette di addestrare agenti in modo automatizzato, ripetendo decine di migliaia di volte il ciclo osservazione-azione-risultato, fino a quando non emergono strategie ottimali che fanno convergere il modello.

Grazie alla flessibilità di Gymnasium, ho potuto modellare un ambiente complesso come un sistema HSS mantenendo una struttura chiara.

Per addestrare l'agente all'interno dell'ambiente simulato del sistema di stoccaggio di idrogeno, ho utilizzato la libreria Stable-Baselines3. È una raccolta di algoritmi di reinforcement learning scritti in Python e basati su PyTorch, molto utile perché semplice da usare, modulare e ben compatibile con ambienti come quelli forniti da Gym.

Grazie a questa libreria, ho potuto integrare facilmente l'algoritmo PPO nel mio progetto con poche righe di codice. Dopo aver definito l'ambiente, ho verificato che fosse compatibile con Stable-Baselines3. Successivamente ho avviato l'addestramento dell'agente con la funzione PPO(), aggiungendo anche un callback per monitorare il

progresso. Durante il processo, l'agente ha interagito ripetutamente con l'ambiente, imparando poco a poco quali azioni portano a risultati migliori.

Inoltre ho potuto salvare il modello addestrato in modo da riprenderne l'addestramento se necessario.

Usare Stable-Baselines3 ha permesso di testare diverse configurazioni con facilità, definire con precisione su quanti passi dovesse essere svolto l'addestramento, controllare quando bene l'agente stava imparando, e confrontare i risultati ottenuti con quelli derivanti dal comportamento esperto appreso tramite IRL.

Tale libreria è stata usata anche per l'utilizzo dell'algoritmo DQN applicato all'ambiente basato su IRL.

Nel contesto dell'addestramento di agenti tramite Deep Q-Network (DQN), l'impostazione degli iperparametri gioca un ruolo fondamentale nel determinare l'efficacia e la stabilità del processo di apprendimento. Sebbene la libreria Stable-Baselines3 fornisca dei valori di default, è spesso necessario adattarli al proprio ambiente specifico per ottenere buoni risultati.

In particolare, nel mio caso, alcuni parametri sono stati calibrati manualmente in funzione della natura della reward appresa tramite IRL, della dinamica dell'ambiente e della convergenza osservata durante l'allenamento. Di seguito vengono riportati gli iperparametri adottati e il significato di ciascuno.

- learning_rate = 1e-4: tasso di apprendimento della rete neurale. Un valore basso come 10^{-4} rende l'aggiornamento dei pesi più stabile, evitando grandi oscillazioni ma rallentando leggermente la convergenza.

- buffer_size = 20000: numero massimo di esperienze salvate nel replay buffer. Un buffer di dimensioni medie permette una buona varietà di transizioni passate da cui campionare senza richiedere troppa memoria.

- learning_starts = 1000: l'addestramento inizia solo dopo che l'agente ha accumulato almeno 1000 esperienze. Questo assicura che ci sia una base minima di dati nel buffer prima di aggiornare la rete.

- batch_size = 64: numero di campioni presi dal buffer ad ogni aggiornamento. Un valore standard che bilancia accuratezza della stima del gradiente e efficienza computazionale.

- gamma = 0.99: fattore di sconto che determina l'importanza delle ricompense future. Un valore vicino a 1 spinge l'agente a pianificare nel lungo termine. Al

contrario, un valore che si avvicina allo zero non darebbe la giusta rilevanza a ricompense future.

- train_freq = 4: ogni 4 passi di interazione con l'ambiente viene eseguito un aggiornamento dei pesi. Questa frequenza è utile per mantenere un buon compromesso tra esplorazione e apprendimento.
- target_update_interval = 1000: ogni 1000 aggiornamenti, i pesi della rete target vengono sincronizzati con quelli della rete principale, migliorando la stabilità dell'apprendimento.
- exploration_fraction = 0.4: per il 40% dell'addestramento, il valore ϵ decresce linearmente, favorendo l'esplorazione iniziale e gradualmente passando allo sfruttamento delle strategie apprese.
- exploration_final_eps = 0.1: valore minimo di ϵ, ovvero la probabilità residua di scegliere un'azione casuale anche a training inoltrato. Un ϵ finale di 0.1 mantiene una leggera componente di esplorazione.
- verbose = 0: disattiva l'output durante il training. Impostazione utile per ridurre il rumore su console quando l'addestramento viene eseguito in batch o in background. Questo parametro permette di rendere inferiore il tempo per terminare gli steps del modello.

Le operazioni numeriche, come il calcolo della distribuzione di probabilità o la gestione dello stato, sono state realizzate con il supporto della libreria NumPy, standard per la manipolazione di array multidimensionali in Python.

La gestione dei dati, in particolare la lettura e scrittura di traiettorie esperte e reward apprese, è stata effettuata tramite Pandas, una libreria Python ottimizzata per il data processing. È stata molto utile anche per la fase di debugging manuale per verificare se le traiettorie utilizzate dal modello fossero ottimali.

Inoltre, i risultati dell'addestramento sono stati visualizzati con Matplotlib, una libreria grafica ampiamente utilizzata per generare plot in ambito scientifico.

Le azioni dell'agente sono state definite tramite l'utilizzo della classe Enum di Python, che consente di strutturare insiemi di valori in modo leggibile.

Per simulare in modo realistico la domanda energetica e per il calcolo di funzioni logaritmiche stabilizzate numericamente (come la funzione logsumexp per il calcolo della massima entropia), è stata utilizzata la libreria SciPy.

Capitolo 3

Risultati e Valutazione

In questa parte della tesi vengono presentati e valutati i risultati ottenuti dall'addestramento degli agenti nei vari ambienti sviluppati.

L'obiettivo è capire quanto siano efficaci e applicabili le strategie adottate, mettendo a confronto approcci basati su reinforcement learning diretto (RL) e inverse reinforcement learning (IRL), considerando anche le differenze dei modelli utilizzati.

Vengono analizzati 3 confronti:

- Per prima cosa è stato utile osservare il comportamento dell'agente in un ambiente con spazio delle azioni discreto rispetto a uno con azioni multidiscrete. Questo passaggio serve soprattutto a dimostrare come una semplificazione eccessiva, come nel caso delle azioni completamente discrete, renda il modello poco realistico e quindi poco utile per simulare un sistema complesso come un sistema di stoccaggio di idrogeno.

- Successivamente, viene analizzata la differenza tra due modelli di Maximum Entropy IRL addestrati con algoritmi diversi, ovvero il PRoximal Policy Optimization (PPO) e il Deep Q-Network (DQN). Si tratta di due approcci molto usati nel reinforcement learning, ma che si comportano in modo piuttosto diverso, soprattutto in ambienti complessi come quelli che si basano su reward apprese da traiettorie esperte.

  Viene quindi valutato quale dei due riesca ad apprendere politiche più stabili, coerenti ed efficienti.

- Infine, si confrontano i risultati ottenuti da un agente addestrato con IRL con quelli di un agente addestrato direttamente con RL, entrambi usando lo stesso algoritmo (PPO).

Questo confronto è fondamentale per capire se un modello che apprende indirettamente la reward tramite esempi esperti può raggiungere prestazioni pari, se non superiori, a quelle di un agente che apprende da una reward definita esplicitamente.

Per ogni esperimento, verranno mostrati grafici relativi alle principali grandezze coinvolte: idrogeno prodotto, energia venduta da fonte primaria o dall'uso dell'idrogeno, perdite di potenza e andamento della reward.

L'obiettivo non è solo confrontare numeri, ma anche capire il comportamento degli agenti, quanto le loro decisioni siano coerenti e in che misura riflettano strategie davvero utili in uno scenario reale.

3.1 RL con azioni discrete vs. multidiscrete

Per comprendere meglio come l'agente apprende e prende decisioni, è stato utile confrontare due configurazioni dell'ambiente: una in cui le azioni sono discrete (Figura 3.1) e una in cui sono multidiscrete (Figura 3.2). In entrambi i casi, l'addestramento è stato effettuato utilizzando l'algoritmo PPO.

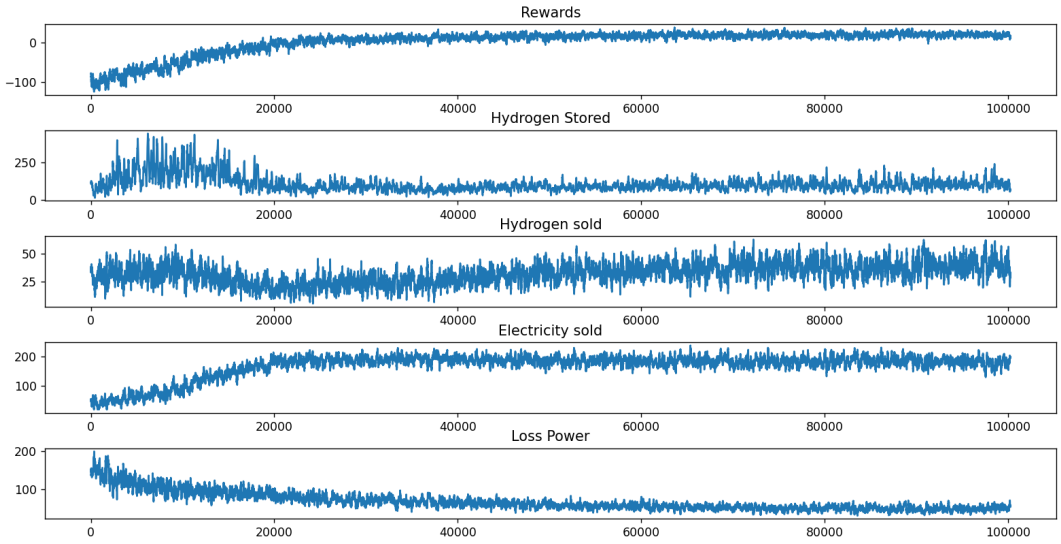

Figura 3.1: Andamento dell'addestramento nell'ambiente con azioni discrete

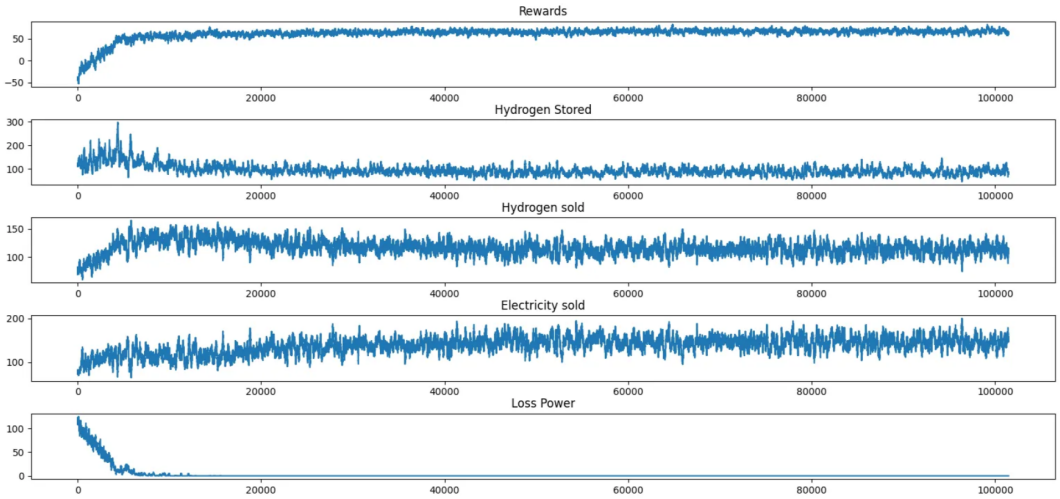

Figura 3.2: Andamento dell'addestramento nell'ambiente con azioni multidiscrete

Nel primo ambiente, l'agente può scegliere una sola azione alla volta tra tutte le possibili combinazioni di comportamento. Questo implica che ad ogni passo temporale venga eseguita un'unica azione predefinita, che non può includere contemporaneamente più operazioni (ad esempio, produrre idrogeno e vendere elettricità).

Invece, nell'ambiente con azioni multidiscrete ogni decisione è scomposta in cinque sottocomponenti binarie, che rappresentano la possibilità di: produrre idrogeno, vendere idrogeno, vendere elettricità, bloccare la produzione e bloccare la vendita. Questo consente una maggiore flessibilità nella definizione della politica, dato che le azioni non sono più mutualmente esclusive, ma possono essere combinate tra loro in maniera più naturale e coerente con il funzionamento reale di un sistema di produzione e stoccaggio di idrogeno.

L'analisi dei grafici mostra chiaramente come l'ambiente con azioni multidiscrete consenta un apprendimento più stabile ed efficace. Le ricompense medie crescono in modo più netto e regolare rispetto all'ambiente discreto, e la produzione energetica è distribuita in modo più efficiente: l'idrogeno immagazzinato si stabilizza a livelli sostenibili, mentre la quantità venduta cresce progressivamente. Al contrario, nel caso discreto, il comportamento dell'agente risulta più disordinato: l'idrogeno accumulato aumenta bruscamente in una prima fase, per poi ridursi senza mostrare una chiara strategia, e anche la vendita risulta poco coordinata.

Anche la power loss (energia prodotta ma non utilizzata) risulta significativamente più elevata nel caso discreto, segno di una gestione meno ottimale delle risorse. Al contrario, l'ambiente multidiscreto riesce a minimizzare le perdite quasi completamente già dopo poche migliaia di passi di addestramento.

In conclusione, l'esperimento dimostra che, per ambienti reali in cui le azioni sono

composte da più decisioni simultanee, l'uso di uno spazio delle azioni multidiscrete è non solo più coerente dal punto di vista modellistico, ma anche decisamente più efficace in termini di apprendimento e prestazioni finali.

3.2 IRL con PPO vs. IRL con DQN

Per valutare l'efficacia degli algoritmi di reinforcement learning utilizzati nell'addestramento di un ambiente basato su reward apprese tramite Maximum Entropy Inverse Reinforcement Learning sono stati eseguiti e confrontati due esperimenti paralleli: uno usando il Proximal Policy Optimization (PPO) e uno con Deep Q-Network (DQN). Nonostante entrambi partano dallo stesso ambiente e dalle stesse ricompense apprese, il comportamento degli agenti addestrati è risultato differente, come si può osservare dai grafici sotto riportati.

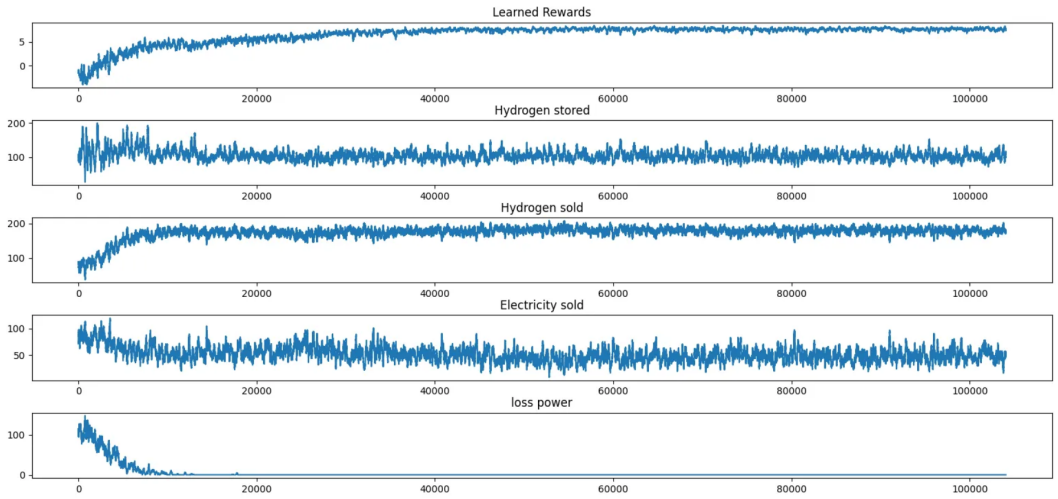

Figura 3.3: Andamento dell'addestramento nell'ambiente con PPO

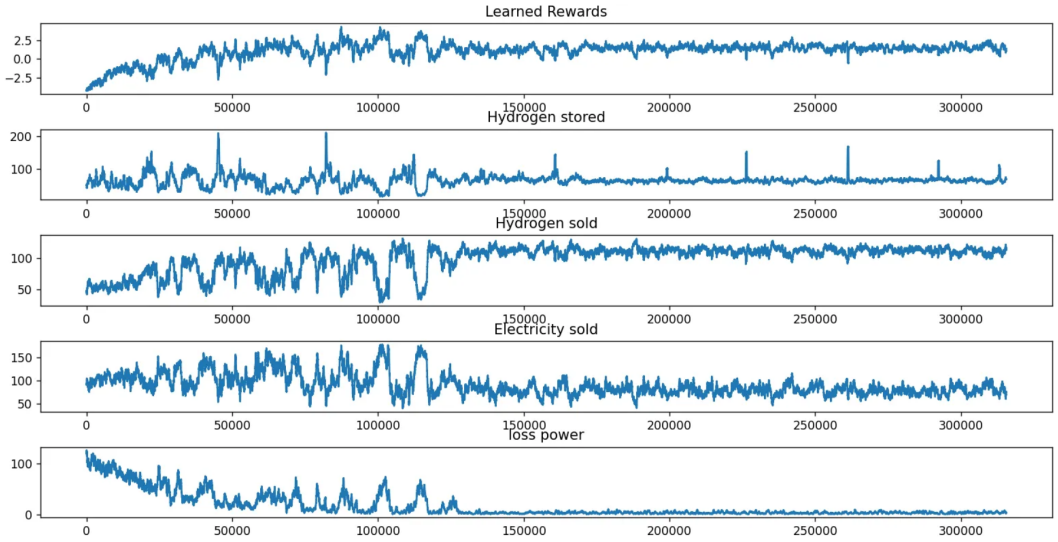

Figura 3.4: Andamento dell'addestramento nell'ambiente con DQN

Nel primo caso, l'agente addestrato con PPO mostra un apprendimento coerente e progressivo. Le rewards aumentano rapidamente nei primi step fino a stabilizzarsi su valori positivi, indice di un comportamento ottimizzato in base alla struttura delle ricompense. La curva dell'idrogeno stoccato risulta regolare e le vendite di energia prodotta dall'idrogeno e di energia in ingresso seguono andamenti ordinati.

Importante è l'andamento della curva della perdita di energia (loss power) che si abbassa rapidamente fino ad annullarsi in appena 10000 steps. Ciò dimostra che l'agente ha imparato efficacemente a sfruttare a pieno l'energia disponibile per massimizzare la ricompensa appresa.

Invece, l'agente addestrato con DQN presenta un comportamento meno stabile. Anche se in fase iniziale l'addestramento sembra avvenire, la curva delle rewards tende a rimanere più bassa e presenta un maggiore livello di rumore, sintomo di una strategia utilizzata dall'agente meno efficace.

Le curve delle variabili come la produzione e la vendita, mostrano maggiori oscillazioni e sono meno coerenti con l'obiettivo di minimizzare le perdite. In particolare, l'energia persa non scende mai agli stessi valori minimi osservati con PPO, segno che l'agente non è riuscito a sfruttare in modo efficace l'energia disponibile. Ciò porta anche a una gestione meno efficace dell'idrogeno immagazzinato e distribuito.

Queste differenze possono essere spiegate da alcune diversità tra i due algoritmi. Infatti, DQN opera in uno spazio d'azione discreto e richiede una funzione di valore approssimata, mentre PPO lavora direttamente con politiche stocastiche e continua ad aggiornare la sua strategia in modo più flessibile, rendendolo più adatto ad ambienti complessi come quello IRL. Inoltre, DQN è più sensibile alla qualità della discretizzazione degli stati e

all'esplorazione, mentre PPO può gestire con maggiore stabilità anche dinamiche continue e azioni multidiscrete.

Un ulteriore vantaggio di PPO è che, essendo un algoritmo on-policy, aggiorna la sua politica passo dopo passo sulla base delle nuove rewards. DQN, invece, è off-policy ovvero apprende da esperienze passate salvate in memoria (replay buffer), e per questo è più soggetto all'overfitting quando le rewards non sono ben definite o quando cambiano nel tempo.

In sintesi, i risultati ottenuti confermano che l'utilizzo di PPO per addestrare un agente in ambiente IRL consente di ottenere prestazioni più stabili, coerenti e in linea con gli obiettivi appresi tramite l'inferenza delle rewards. DQN, pur funzionando in alcuni scenari, mostra maggiori limiti quando si applica in contesti complessi e appresi indirettamente come quelli dell'IRL.

3.3 IRL vs RL su PPO

Per valutare le prestazioni e le differenze tra i modelli IRL (Inverse Reinforcement Learning) e RL (Reinforcement Learning), sono stati svolti tre esperimenti diversi, ognuno dei quali presentava differenti dati in input in termini di potenza elettrica disponibile e di domanda energetica. Le condizioni oggetto di analisi sono basate su tre tipologie progressivi di input energetico e di richiesta: 600 kWh, 1800 kWh e 2500 kWh. Tutti e due i modelli sono stati addestrati per 100.000 step e sono stati però testati con le stesse condizioni iniziali per trattenerne le prestazioni.

I grafici che seguono rappresentano l'evoluzione di cinque indicatori:

- Profitto economico
- Energia venduta prodotta partendo dall'idrogeno stoccato nel sistema
- Idrogeno stoccato
- Energia venduta partendo dalla potenza in input
- Potenza persa, ovvero non utilizzata dal modello

I risultati relativi ai tre scenari sono riportati nelle Figure 3.5, 3.6 e 3.7.

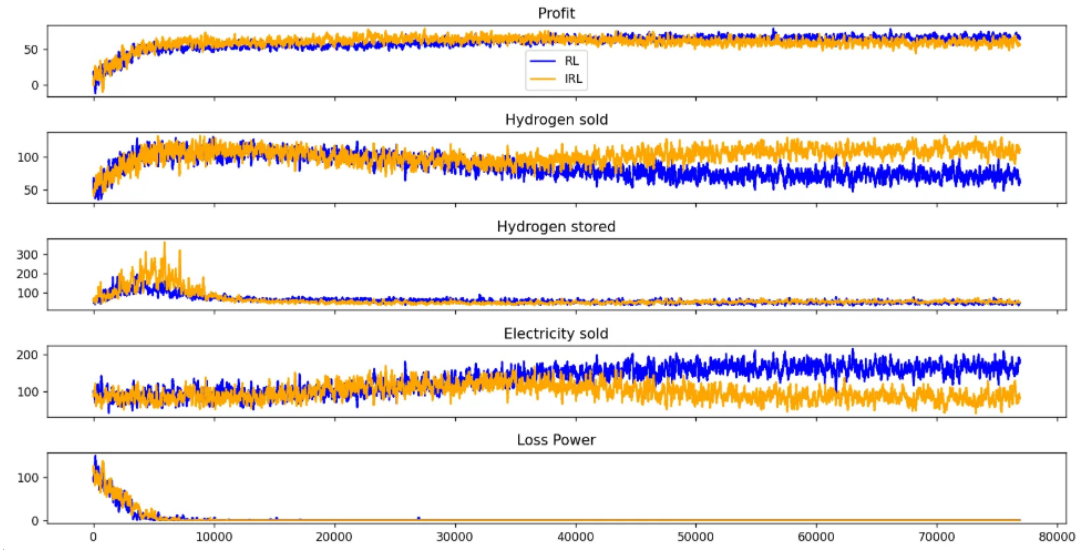

Figura 3.5: Confronto tra gli andamenti di RL e IRL con 600 kWh come potenza energetica in ingresso e domanda energetica

Il modello IRL mostra un profitto leggermente superiore e più stabile rispetto al modello RL, suggerendo una strategia di ottimizzazione più efficace nel lungo periodo.

Anche in termini di idrogeno venduto, IRL risulta più performante, mantenendo valori mediamente più alti.

Tuttavia, per quanto riguarda la vendita di elettricità, il modello RL mostra prestazioni migliori, probabilmente adottando una politica più orientata alla monetizzazione immediata dell'energia disponibile.

La quantità di idrogeno immagazzinato decresce progressivamente in entrambi i casi, indicando una sempre maggiore capacità di allocare in modo efficiente la produzione.

Infine, la metrica relativa all'energia persa si riduce drasticamente per entrambi i modelli già nelle fasi iniziali, attestandosi su valori prossimi allo zero, segno di un buon adattamento alle dinamiche del sistema. Complessivamente, entrambi i modelli dimostrano un apprendimento efficace, con IRL che mostra una leggera superiorità in termini di efficienza complessiva e stabilità operativa.

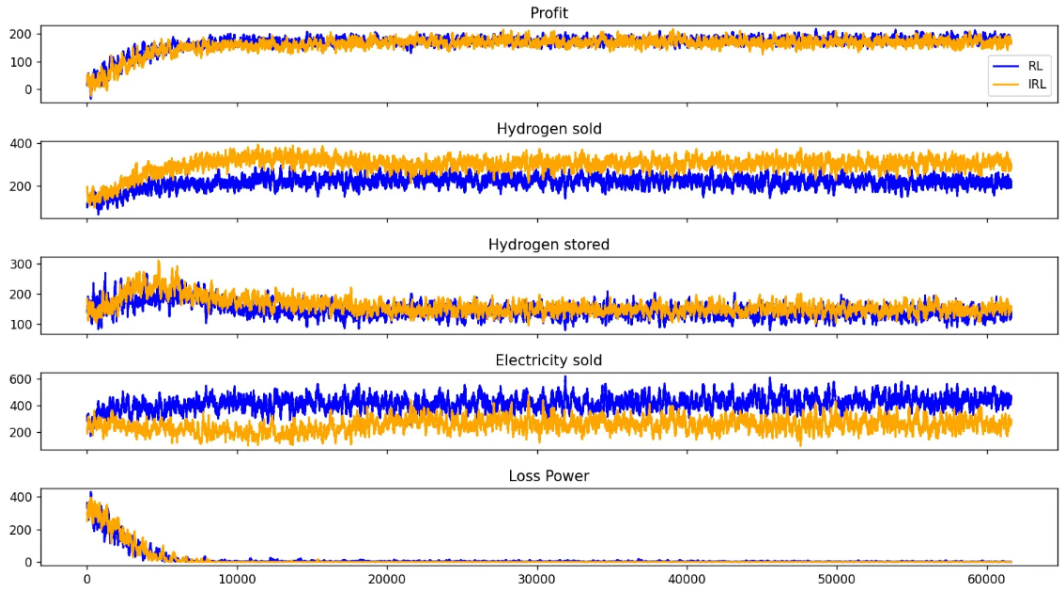

Figura 3.6: Confronto tra gli andamenti di RL e IRL con 1800 kWh come potenza energetica in ingresso e domanda energetica

Nel secondo scenario, caratterizzato da un incremento della potenza in ingresso e della domanda energetica a 1800 kWh, l'analisi dei risultati evidenzia un'evoluzione delle prestazioni dei modelli RL e IRL. Il profitto totale aumenta in modo consistente rispetto al caso precedente, raggiungendo valori più elevati e stabili. Entrambi i modelli si comportano in maniera simile, ma IRL continua a mantenere una leggera superiorità in termini di profitto medio e minore variabilità, segno di un comportamento più regolare ed efficace.

Per quanto riguarda l'idrogeno venduto, IRL mostra un netto miglioramento rispetto a RL, con valori mediamente superiori lungo tutta la durata dell'addestramento. Questo suggerisce che IRL riesce a sfruttare meglio l'energia disponibile per produrre e commercializzare idrogeno. Anche l'idrogeno immagazzinato presenta una tendenza interessante: sebbene entrambi i modelli mantengano una quantità stabile nel tempo, IRL sembra gestire le scorte con maggiore flessibilità, accumulando e rilasciando in modo più dinamico a seconda delle condizioni.

Sul fronte dell'elettricità venduta, RL si conferma più aggressivo, mantenendo valori mediamente più alti rispetto a IRL. Questo comportamento potrebbe indicare una strategia meno conservativa, mirata a massimizzare i ricavi nel breve periodo attraverso la vendita diretta dell'energia. Infine, la metrica relativa all'energia persa mostra un'evoluzione simile a quella osservata nel primo scenario: le perdite diminuiscono rapidamente entro i primi 10.000 step e si mantengono su livelli molto bassi per entrambi i modelli, a testimonianza di un'efficace ottimizzazione del sistema.

L'aumento della disponibilità energetica consente ai modelli di esprimere meglio il proprio potenziale, con IRL che continua a dimostrarsi leggermente più efficiente e stabile, mentre RL mantiene un approccio più orientato alla vendita diretta dell'elettricità.

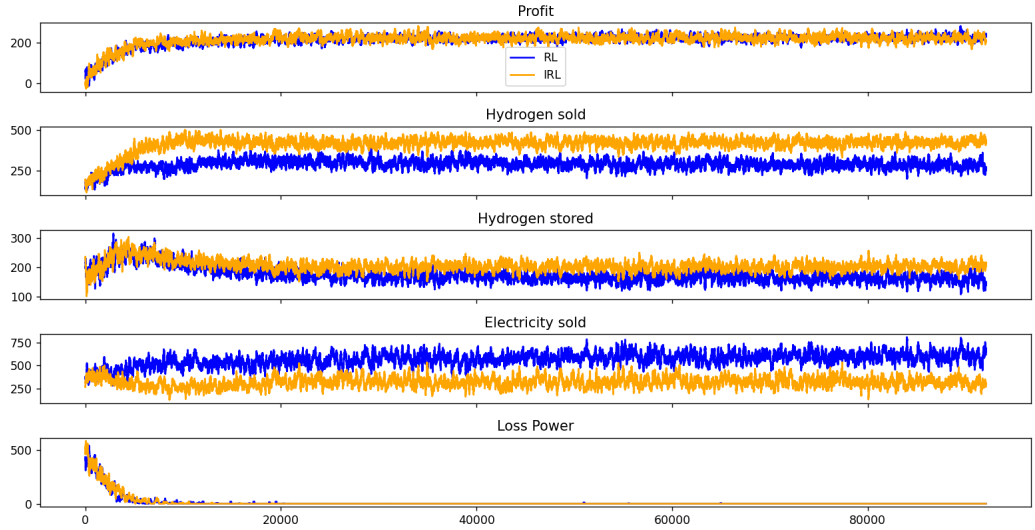

Figura 3.7: Confronto tra gli andamenti di RL e IRL con 2500 kWh come potenza energetica in ingresso e domanda energetica

Nel terzo e ultimo scenario, corrispondente a una configurazione con potenza in ingresso e domanda energetica pari a 2500 kWh, si osserva un ulteriore incremento delle prestazioni dei modelli RL e IRL, confermando la correlazione positiva tra disponibilità energetica e rendimento complessivo del sistema. Il profitto totale raggiunge i valori più alti tra tutti gli scenari analizzati, con una curva che si stabilizza rapidamente oltre i 200 unità. In questo caso, le prestazioni dei due modelli risultano molto ravvicinate, ma IRL continua a mostrare un comportamento più regolare, con oscillazioni più contenute.

Anche l'idrogeno venduto aumenta sensibilmente rispetto agli scenari precedenti. Il modello IRL si distingue ancora una volta, registrando valori mediamente superiori a quelli di RL. Questo conferma l'efficacia di IRL nel convertire l'energia disponibile in idrogeno commerciabile in modo ottimale. In parallelo, la quantità di idrogeno immagazzinato segue un andamento coerente, con livelli stabili e leggermente più elevati per IRL, a testimonianza di una strategia di accumulo ben bilanciata e pronta a rispondere alle variazioni della domanda.

Per quanto riguarda l'elettricità venduta, il modello RL mantiene un vantaggio evidente anche in questo scenario, con picchi che superano stabilmente i 700 unità. Questo conferma la sua inclinazione verso politiche orientate alla massimizzazione immediata del profitto tramite la vendita diretta di energia elettrica. Infine, l'energia persa si riduce rapidamente entro i primi 10.000 step, stabilizzandosi su livelli prossimi allo zero, come già

osservato nei casi precedenti. Entrambi i modelli dimostrano dunque un'ottima capacità di ottimizzazione anche in presenza di input energetici più abbondanti.

Tuttavia in alcuni casi, specialmente durante i test con scenari caratterizzati da potenza in ingresso e domanda energetica elevate (es. 1800 o 2500 kWh), il modello IRL può incontrare difficoltà nel convergere verso politiche pienamente ottimali. Sebbene questo comportamento si manifesti raramente, tende a emergere quando i valori di potenza e domanda, generati casualmente, risultano particolarmente sbilanciati per diversi step consecutivi. In tali situazioni, l'algoritmo può preferire strategie stabili ma subottimali, come lo stoccaggio completo dell'idrogeno anche in assenza di un'effettiva necessità, oppure la vendita diretta della potenza in ingresso senza considerare l'accumulo. Per mitigare tali effetti, un potenziale miglioramento potrebbe consistere nell'utilizzo di dati reali per la generazione delle traiettorie esperte, al fine di ridurre l'eterogeneità delle situazioni simulate e riflettere con maggiore precisione le dinamiche operative di un contesto reale.

Nel complesso, l'aumento della potenza a 2500 kWh consente di sfruttare pienamente le potenzialità di entrambi i modelli. IRL conferma la propria efficacia nella gestione dell'idrogeno e nella stabilità delle prestazioni, mentre RL si afferma nel preferire la vendita diretta dell'elettricità, mostrando strategie più conservative ma anch'esse efficienti.

Capitolo 4

Conclusioni

Nel corso di questa tesi ho approfondito l'utilizzo di tecniche di Reinforcement Learning (RL) e Inverse Reinforcement Learning (IRL) per migliorare la gestione di un sistema di stoccaggio di idrogeno, con l'obiettivo di ottimizzarne le prestazioni economiche. L'approccio basato sul metodo Maximum Entropy IRL si è rivelato particolarmente promettente, riuscendo a fornire all'agente una strategia più stabile ed efficace, soprattutto in scenari complessi e variabili.

Nel confronto tra i due approcci, IRL ha mostrato una gestione più equilibrata delle risorse, con performance solide in termini di profitto e riduzione delle perdite. RL, d'altra parte, ha adottato politiche più aggressive nella vendita dell'elettricità, ottenendo comunque risultati interessanti sul piano economico, anche se leggermente meno stabili.

L'algoritmo PPO si è dimostrato quello più adatto al contesto, garantendo un apprendimento coerente sia nel caso di reward esplicite (RL) sia nel caso di reward apprese (IRL). Al contrario, l'algoritmo DQN ha faticato di più ad adattarsi, evidenziando una maggiore sensibilità alla struttura dell'ambiente e alla qualità delle traiettorie.

A livello di progettazione, il passaggio a uno spazio delle azioni multidiscreto ha reso il comportamento dell'agente molto più realistico, permettendogli di prendere decisioni complesse e combinate, come avviene in un vero sistema energetico. Anche la scelta di creare traiettorie esperte e di normalizzare le ricompense ha avuto un impatto positivo sull'apprendimento.

In conclusione, il lavoro ha confermato che l'integrazione tra RL e IRL può offrire soluzioni efficaci per la gestione intelligente dell'energia, in particolare in contesti dove si utilizzano fonti rinnovabili e sistemi di accumulo. In futuro, sarebbe interessante applicare questi modelli a dati reali, esplorare scenari multi-agente e valutare l'integrazione all'interno di reti intelligenti su scala più ampia.

Guardando a studi futuri, ci sono diversi spunti interessanti che potrebbero arricchire

e potenziare il lavoro svolto. Un primo passo potrebbe essere quello di testare l'approccio IRL con algoritmi alternativi, come SAC [19] [20], TD3 o A2C, per valutarne l'efficacia in contesti più dinamici e confrontarli con i risultati ottenuti con PPO e DQN. Un altro sviluppo rilevante sarebbe l'utilizzo di dati reali per costruire traiettorie esperte più inerenti alle condizioni operative concrete; ciò potrebbe portare a un ulteriore miglioramento del modello IRL. Inoltre, introdurre vincoli energetici realistici, come il tempo di ricarica o l'usura degli impianti, permetterebbe di simulare scenari ancora più vicini al mondo reale. In aggiunta si potrebbe anche considerare la vendita diretta dell'idrogeno passando da un sistema di stoccaggio (HSS) a una stazione di rifornimento di idrogeno (HRS). Infine, l'estensione del modello a un sistema multi-agente, con la presenza di più utenti, elettrolizzatori, serbatoi per l'idrogeno e celle a combustibile, aprirebbe interessanti possibilità in termini di coordinamento e ottimizzazione distribuita.

Bibliografia

[1] International Energy Agency, «Global Hydrogen Review 2022,» IEA, Paris, rapp. tecn., 2022, Rapporto annuale IEA su infrastrutture, produzione e mercati dell'idrogeno. indirizzo: https://www.iea.org/reports/global-hydrogen-review-2022.

[2] M. Ball e M. Weeda, «The hydrogen economy – Vision or reality?» *International Journal of Hydrogen Energy*, vol. 40, n. 25, pp. 7903–7919, 2015. DOI: 10.1016/j.ijhydene.2015.04.032. indirizzo: https://doi.org/10.1016/j.ijhydene.2015.04.032.

[3] P. Nikolaidis e A. Poullikkas, «A comparative overview of hydrogen production processes,» *Renewable and Sustainable Energy Reviews*, vol. 67, pp. 597–611, 2017. DOI: 10.1016/j.rser.2016.09.044. indirizzo: https://doi.org/10.1016/j.rser.2016.09.044.

[4] X. Li, Y. Li, X. Song et al., «Operation of Distributed Battery Considering Demand Response Using Deep Reinforcement Learning in Grid Edge Control,» *Applied Energy*, vol. 304, p. 117 773, 2021. DOI: 10.1016/j.apenergy.2021.117773.

[5] S. D. Ramchurn, P. Vytelingum, A. Rogers e N. R. Jennings, «Putting the 'smarts' into the smart grid: a grand challenge for artificial intelligence,» *Communications of the ACM*, vol. 55, n. 4, pp. 86–97, 2012. DOI: 10.1145/2133806.2133825.

[6] S. I. Muhammad, D. Wei e Y. Qiang, «Machine learning driven smart electric power systems: Current trends and new perspectives,» *Applied Energy*, 2020. DOI: https://doi.org/10.1016/j.apenergy.2020.115237.

[7] A. Sheikhi, M. Rayati e A. Ranjbar, «Dynamic load management for a residential customer: Reinforcement Learning approach,» *Sustainable Cities and Society*, vol. 24, pp. 42–51, 2016. DOI: https://doi.org/10.1016/j.scs.2016.04.001.

[8] F. Ruelens, B. J. Claessens, S. Vandael, B. De Schutter, R. Babuska e R. Belmans, «Reinforcement learning applied to an electric water heater: From theory to practice,» *IEEE Transactions on Smart Grid*, vol. 9, n. 4, pp. 3792–3800, 2018. DOI: 10.1109/TSG.2016.2596817.

[9] Y. Yu, Z. Yang, N. Zhang, W. Tan, Y. Zhang e Y. Zhang, «A Review of Deep Reinforcement Learning for Smart Building Energy Management,» *IEEE Internet of Things Journal*, vol. 8, n. 20, pp. 15 439–15 452, 2021. DOI: 10.1109/JIOT.2021.3076583.

[10] T. H. B. Huy, N. T. M. Duy, P. V. Phu, T.-D. Le, S. Park e D. Kim, «Robust real-time energy management for a hydrogen refueling station using generative adversarial imitation learning,» *Applied Energy*, vol. 373, 2024. DOI: https://doi.org/10.1016/j.apenergy.2024.123847.

[11] D. Sourav, M. Thibault e G. Henze, «Inverse reinforcement learning control for building energy management,» *Energy & Buildings*, 2023. DOI: https://doi.org/10.1016/j.enbuild.2023.112941.

[12] A. Y. Ng e S. J. Russell, «Algorithms for Inverse Reinforcement Learning,» in *Proceedings of the Seventeenth International Conference on Machine Learning (ICML)*, Morgan Kaufmann, 2000, pp. 663–670.

[13] P. Abbeel e A. Y. Ng, «Apprenticeship Learning via Inverse Reinforcement Learning,» in *Proceedings of the Twenty-First International Conference on Machine Learning (ICML)*, ACM, 2004, pp. 1–8. DOI: 10.1145/1015330.1015430.

[14] B. D. Ziebart, A. L. Maas, J. A. Bagnell e A. K. Dey, «Maximum Entropy Inverse Reinforcement Learning,» in *Proceedings of the 23rd AAAI Conference on Artificial Intelligence*, 2008, pp. 1433–1438.

[15] M. Wulfmeier, P. Ondrúška e I. Posner, «Maximum Entropy Deep Inverse Reinforcement Learning,» *arXiv preprint arXiv:1507.04888*, 2015.

[16] S. John, W. Filip, D. Prafulla, R. Alec e K. Oleg, «Proximal Policy Optimization Algorithms,» *arXiv*, 2017. DOI: https://doi.org/10.48550/arXiv.1707.06347.

[17] V. Mnih, K. Kavukcuoglu, D. Silver et al., «Playing Atari with Deep Reinforcement Learning,» *arXiv preprint arXiv:1312.5602*, 2013.

[18] V. Mnih, K. Kavukcuoglu, D. Silver et al., «Human-level control through deep reinforcement learning,» *Nature*, vol. 518, pp. 529–533, 2015. DOI: 10.1038/nature14236. indirizzo: https://www.nature.com/articles/nature14236.

[19] T. Haarnoja, A. Zhou, P. Abbeel e S. Levine, «Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor,» in *Proceedings of the 35th International Conference on Machine Learning (ICML)*, vol. 80, PMLR, 2018, pp. 1861–1870. indirizzo: http://proceedings.mlr.press/v80/haarnoja18b.html.

[20] S. Fujimoto, H. van Hoof e D. Meger, «Addressing Function Approximation Error in Actor-Critic Methods,» in *Proceedings of the 35th International Conference on Machine Learning (ICML)*, vol. 80, PMLR, 2018, pp. 1587–1596. indirizzo: http://proceedings.mlr.press/v80/fujimoto18a.html.