POLITECNICO DI TORINO

Corso di Laurea Magistrale in Ingegneria Meccanica

Tesi di Laurea Magistrale

Studio parametrico di un algoritmo di Deep Reinforcement Learning per la gestione energetica di un veicolo ibrido elettrico

Relatori:

Prof. Ezio Spessa

Dott. Claudio Maino

Dott. Matteo Acquarone

Candidato:

Luigi Lacaita

Anno Accademico 2021-22

Sommario

Nei veicoli ibridi elettrici (HEV) l'introduzione di una o più sorgenti di potenza, rispetto ad un veicolo convenzionale, aumentano la complessità del powertrain e di conseguenza il sistema di gestione energetica del veicolo. Pertanto, negli ultimi anni la ricerca sull' *energy management* è diventata un'area sempre più importante negli studi degli HEV. All'interno di questo lavoro di tesi è stato applicato un algoritmo di *Deep Reinforcement Learning*, ossia il *Deep Q-Network* (DQN), che combina i concetti dell'apprendimento per rinforzo con l'utilizzo di reti neurali artificiali. Questo algoritmo è utilizzato per gestire la ripartizione dei flussi di potenza tra il motore termico (ICE) e la macchina elettrica (EM), controllando contemporaneamente i consumi di combustibile (FC) e lo stato di carica della batteria (SOC). Nel dettaglio il nostro caso studio è una *passenger car* ibrida con architettura parallela p2. Il sofware utilizzato si compone di tre ambienti: il *Simulator*, *l'Environment* e *l'Agent*. Il *Simulator* rappresenta il modello del veicolo e di tutti i suoi sottocomponenti, ed ha lo scopo di riprodurre una rappresentazione virtuale di un HEV e di fornire all'utente una valutazione realistica delle prestazioni dell'agente sull'attività di controllo dell'HEV. Il *Simulator* è completamente sviluppato in ambiente Matlab e comunica con un Master implementato in Python, in cui sono costruiti l'A*gent* e *l'Environment*. L'obbiettivo di questa tesi è quello di testare un algoritmo DQN al variare dei suoi principali parametri di training su ciclo omologativo WLTP. Dopo aver trovato una soluzione sub-ottimale, sono stati eseguiti dei test su altri cicli guida (cicli clust) per verificare la robustezza di un agente DQN al variare della missione di guida. I risultati ottenuti mostrano come tale strategia di gestione dell'energia possa ridurre notevolmente i consumi di carburante, ma al contempo evidenziano uno scarso adattamento dell'agente DQN al variare delle condizioni di guida, pertanto è richiesta una calibrazione dei principali parametri di addestramento al variare dell'ambiente.

Indice

Indice delle figure

Indice delle tabelle

1. Introduzione

1.1 Motivazioni

Per far fronte all'inquinamento e al riscaldamento globale causato principalmente dalla CO_2 attualmente è in atto una vera e propria transizione energetica per quanto riguarda il settore dei trasporti. L'evoluzione tecnologica è dettata principalmente dal contesto normativo, perciò anche per via delle stringenti limitazioni emissive imposte dagli Enti Regolatori , come l'Unione Europea (UE), oggigiorno stiamo assistendo ad uno *switch* verso l'elettrificazione. Poiché il contesto normativo (EURO6) regolamenta le emissioni *Tank-To-Wheel* (TTW), negli ultimi anni si è assistito ad un prepotente ingresso sul mercato dei veicoli puramente elettrici (BEV) ed ibridi (PHEV/FHEV). L'UE ha fissato una serie di target per ridurre progressivamente le emissioni dei gas serra con l'obbiettivo di raggiungere la neutralità climatica entro il 2050. A partire dal 2020 i costruttori in fase di omologazione devono garantire una media di 95 gCO_2/km, calcolata sull'intera flotta del venduto. Tale target è destinato ad abbassarsi, infatti è prevista una riduzione della CO_2 media del 15% nel 2025 e del 37,5% nel 2030 rispetto alla *baseline* del 2021.
L'introduzione del motore elettrico a bordo del veicolo, viene utilizzato per affiancare o sostituire il motore termico, permettendo così di ridurre le emissioni inquinanti (HC,CO,NO_X,PM) allo scarico del motore ed i gCO_2/km. In particolare viste le attuali problematiche che presentano i BEV, come ad esempio l'autonomia ridotta, i tempi di ricarica eccessivamente lunghi, l'assenza di un'adeguata infrastruttura di ricarica e ovviamente dei prezzi non accessibili a tutti, i *powertrain* ibridi stanno ricevendo parecchie attenzioni da parte di tutte le case automobilistiche, diffondendosi sempre di più su tutto il mercato mondiale.

1.2 Perché veicoli ibridi elettrici?

1.2.1 Descrizione del sistema di propulsione ibrido

Il veicolo ibrido combina due o più sorgenti di potenza che possono fornire direttamente o indirettamente propulsione, ossia energia alle ruote. Nel caso di ibrido di tipo elettrico abbiamo un motore a combustione interna (ICE) che ha come sorgente di energia primaria l'energia chimica stoccata nel combustibile, mentre la seconda sorgente di potenza è di tipo elettrico, quindi abbiamo uno o più motori elettrici.

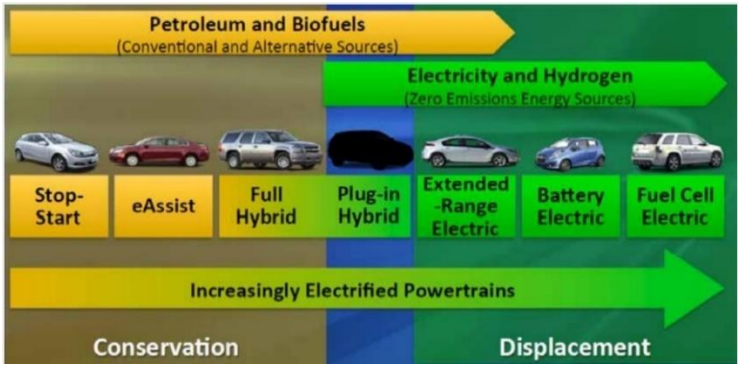

Figura 1.1: Classificazione veicoli sulla base della loro sorgente di energia

Oggigiorno non esiste un'alternativa unica per sostituire il petrolio come sorgente di energia primaria nel trasporto su strada ma coesistono una serie di soluzioni, ad esempio l'impiego di combustibili alternativi (gas naturale o biocombustibili) oppure l'elettrificazione del *powertrain*. Tra le ibride si possono identificare tre diverse macro-categorie: al primo posto in ordine di complessità vi sono le *Mild Hybrid* (MHEV), ossia le ibride leggere, le *Full Hybrid* (FHEV), le ibride complete ed infine le *Plug-in Hybrid* (PHEV), ovvero le ibride che possono essere collegate ad una presa elettrica. La principale differenza tra un PHEV ed un FHEV è che il *plug-in* può potenzialmente utilizzare il motore elettrico come fonte di alimentazione primaria, mentre l' ICE per fornire potenza aggiuntiva. Attualmente l'ibrido è una soluzione molto interessante perché si presenta come oggetto di transizione tra il veicolo convenzionale e l'elettrico, ma al di là di questo l'introduzione di *powertrain* ibridi introduce una serie di miglioramenti:

- **Frenata rigenerativa:** quando il veicolo è in frenata l'energia cinetica può essere recuperata utilizzando la macchina elettrica come generatore per ricaricare il pacco batteria.
- **Eliminazione minimo motore termico (*idle*):** il motore termico può essere spento in condizioni di stop o bassa velocità (ad esempio meno di 20 km/h).
- **Miglioramento efficienza motore:** il motore elettrico assiste il motore termico, spostando i suoi punti di funzionamento in punti di maggiore efficienza, in questo modo il motore produrrebbe una coppia necessaria per la trazione più un ΔC per ricaricare la batteria *(battery-charging*). Invece per i punti di funzionamento che si trovano ad un *Engine power output* molto basso si può lavorare in *full-electric*, quindi spegnendo il motore termico nei punti a bassissima efficienza.
- **ICE Downsizing/Downspeeding:** grazie all'assistenza del motore elettrico è possibile utilizzare un motore termico (ICE) di taglia più piccola (*Downsizing*) e farlo girare a velocità più basse (*Downspeeding*), garantendo una maggiore efficienza del sistema.
- **Accessori comandati elettricamente:** permette di realizzare un motore termico senza giro cinghia (elemento altamente inefficiente).
- In alcuni casi l'ibrido può essere utilizzato per **gestire i transitori del motore termico,** facendo in modo che questi vengano gestiti più dalla parte elettrica che dalla parte termica, portando a dei vantaggi in termini di consumi ma anche in termini di emissioni di inquinanti e di guidabilità. [1]

Ovviamente l'introduzione di *powertrain* ibridi introduce anche una serie di problematiche, quali:

- Aumento dei costi dei componenti (ad es. batteria, macchina elettrica).
- Maggior peso del veicolo, data la presenza di due *powertrain* a bordo.
- Sistema di controllo per l'ottimizzazione dei consumi più complesso.

Quindi nei veicoli ibridi, avendo aumentato il numero delle sorgenti di potenza installate a bordo, diventa di cruciale importanza la gestione energetica al fine di minimizzare i consumi e quindi le emissioni inquinanti.

1.2.2 Architettura *powertrain* nei veicoli ibridi

I principali componenti di propulsione di un ibrido elettrico sono:

- **Motore termico (ICE):** i veicoli ibridi più diffusi oggi sul mercato sono motorizzati da un motore a combustione interna. Tuttavia, il motore su un ibrido è più piccolo che su un veicolo convenzionale ed utilizza tecnologie avanzate per ridurre le emissioni e aumentare l'efficienza.
- **Serbatoio:** in un ibrido, è il dispositivo di accumulo di energia primaria a bordo.
- **Macchina elettrica:** su un'auto ibrida possiamo avere una o più macchine elettriche. Queste possono agire come motori o come generatori, grazie ai componenti di elettronica di potenza montati a bordo (inverter), in modo tale da trarre energie dalle batterie per accelerare il veicolo o recuperare energia dall'auto in fase di frenata e stoccarla nelle batterie.
- **Batterie:** rappresentano il dispositivo di accumulo di energia secondaria a bordo. A differenza del carburante presente del serbatoio, che può alimentare solo il motore termico, le macchine elettriche possono dare e ricevere energia alle/dalle batterie.
- **Trasmissione:** la trasmissione è un componente chiave di un veicolo ibrido, permettendo di trasmettere energia dalle sorgenti di potenza alle ruote.

A loro volta i veicoli HEV possono presentare due differenti configurazioni possibili, in particolare vi sono due tipologie di architetture utilizzate: Serie e Parallelo.

- **Architettura Serie:** una tipica configurazione di un ibrido Serie è mostrata nella Figura 1.2. Il motore a combustione interna è collegato meccanicamente alla macchina elettrica ed è utilizzato per fornire potenza al motore elettrico o per ricaricare il pacco batterie, quindi il motore termico non può fornire direttamente energia meccanica alle ruote. Potenzialmente un ibrido Serie ci consente di far lavorare il motore termico sempre nel punto di massimo rendimento, in quanto le sue condizioni di funzionamento sono svincolate da quelle del veicolo. Questo sistema inoltre elimina la necessità di avere frizioni o trasmissioni convenzionali, in quanto la caratteristica meccanica di un motore elettrico è già adatta alla trazione. In questa architettura va considerata la multi conversione dell'energia, in quanto l'energia meccanica del motore termico viene convertita in energia elettrica e quest'ultima nuovamente in energia meccanica alle ruote. Ogni conversione di potenza, e quindi di energia produce delle perdite, facendo diminuire l'efficienza del *powertrain*.

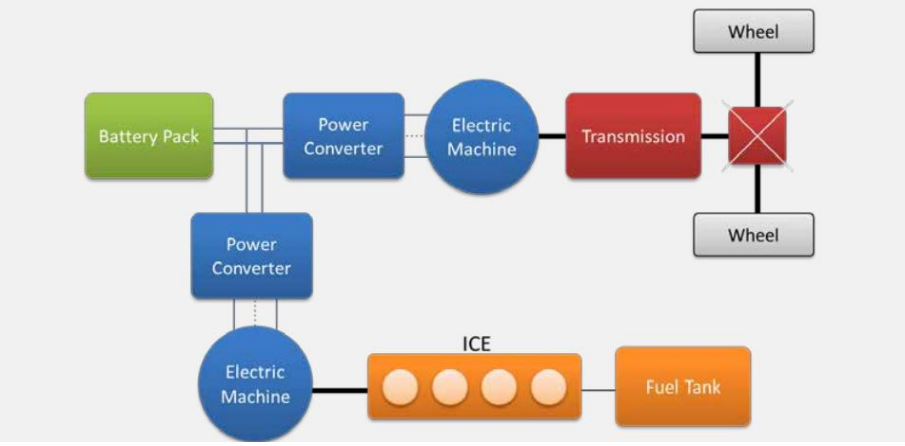

Figura 1.2 Schema di un'architettura ibrido Serie

Nei sistemi di moto propulsione di tipo ibrido è possibile definire un grado di ibridazione R_h, dato dal rapporto tra la potenza erogata dal motore termico e la potenza erogata dal motore collegato direttamente alle ruote, quindi il motore elettrico nel caso di un ibrido serie.

$$R_h = \frac{P_{ICE}}{P_{motor}}$$

Tale rapporto può variare tra 0 ed 1, in particolare abbiamo $R_h = 0$ nel caso di un veicolo puramente elettrico e $R_h = 1$ nel caso di un veicolo con trasmissione elettrica. Nel mezzo abbiamo dei casi intermedi come: *Range Extender, Load Follower* e *Full Performance.*

Infine da Figura 1.3 è possibile notare come questa architettura richieda un sistema di accumulo di energia grande e pesante a bordo, che aumenta notevolmente i costi e riduce le prestazioni del veicolo.

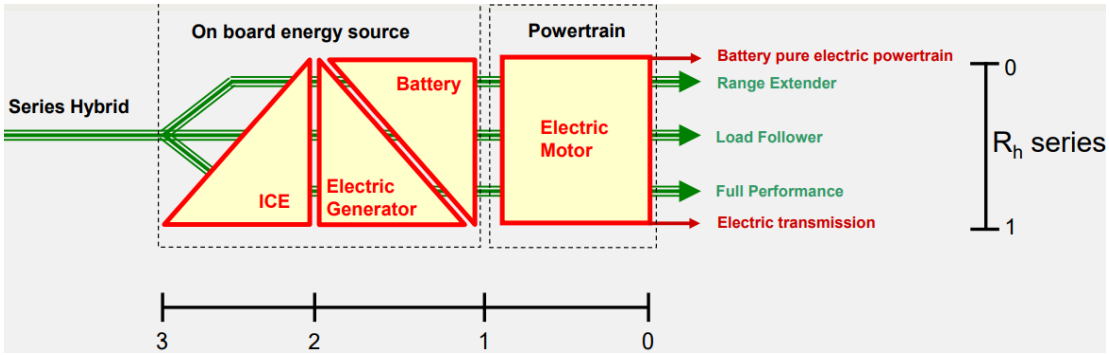

Figura 1.3 Schema grado ibridazione ibrido Serie

- **Architettura Parallelo:** a differenza dell'ibrido Serie, nell'ibrido Parallelo sia il motore termico che quello elettrico sono in grado di fornire potenza meccanica direttamente alle ruote motrici, come visibile in Figura 1.4.

  I principali vantaggi introdotti da questa configurazione rispetto all'ibrido serie sono:

  1. La presenza di un'unica macchina elettrica
  2. Motore di trazione più piccolo. Poiché sia il motore termico che la macchina termica possono contestualmente fornire energia meccanica alle ruote, nessuna delle due macchine è dimensionata per fornire la massima potenza. Ciò determina una riduzione delle taglie dei motori dei trazione, abilitando quindi il *Downsizing* per il motore termico con conseguente migliore efficienza.
  3. Non è necessaria la multi conversione della potenza dal motore termico alle ruote.

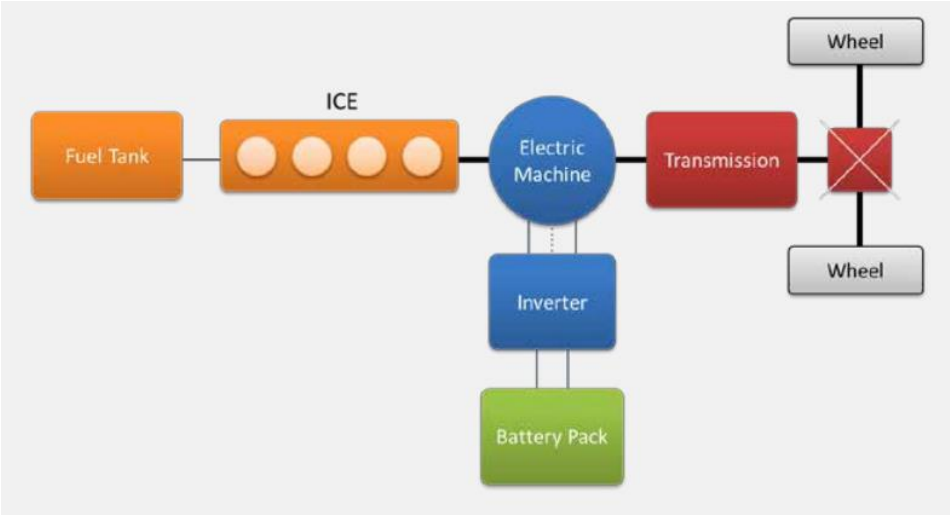

Figura 1.4 Schema di un'architettura ibrido Parallelo

Quindi, complessivamente l'efficienza del sistema può essere maggiore rispetto al caso ibrido serie. Tuttavia, si ha una maggiore complessità dal punto di vista del controllo. Anche in questo caso è possibile definire un grado si ibridazione R_h ed ha lo stesso significato di quello visto per l'ibrido serie.

$$R_h = \frac{P_{ICE}}{P_{ICE} + P_{motor}}$$

Tale rapporto varia tra 0, nel caso di un veicolo convenzionale e 1 nel caso di un veicolo puro elettrico. Tutti i valori intermedi rappresentano ulteriori configurazioni, come: *Minimal Hybrid, Mild Hybrid e Full Performance*.

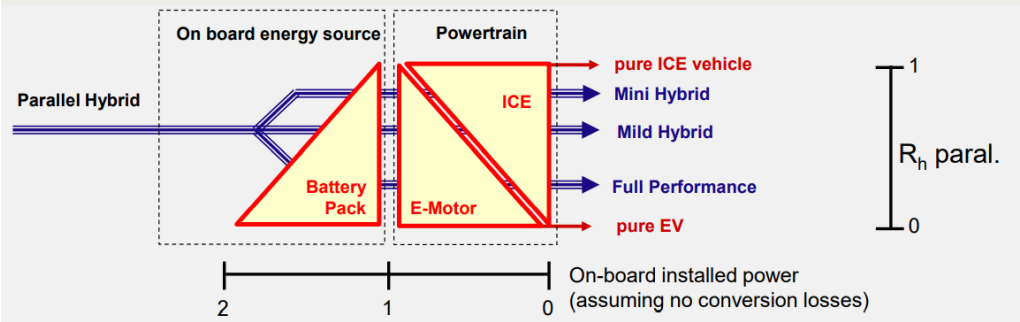

Figura 1.5 Schema grado ibridazione ibrido parallelo

1.2.3 Classificazione ibrido parallelo in base alla posizione della macchina elettrica

Entrando nel dettaglio è possibile individuare tre differenti architetture di ibrido parallelo, come: *Double drive*, *Double shaft* e *Single shaft.* Nel *Double Drive* ho un collegamento meccanico a livello delle ruote, nel *Double shaft* il collegamento meccanico avviene a livello della trasmissione ed infine

nel *Single shaft* il motore termico è collegato meccanicamente con quello elettrico attraverso giro cinghia, ruote dentate ecc.

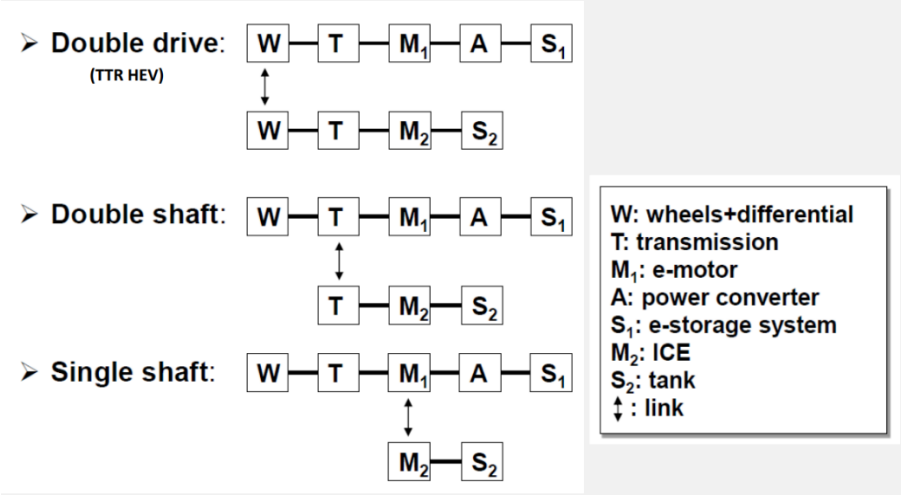

Figura 1.6: Possibili architetture ibrido Parallelo

Nel *Single-shaft* il collegamento meccanico tra i due motori può avvenire in due modi, attraverso una configurazione coassiale o non coassiale. Nella soluzione coassiale i due motori girano alla stessa velocità perché appunto coassiali, permettendo di trasmettere coppie maggiori. Nella configurazione non coassiale invece ho solitamente un collegamento con giro cinghia tra i due motori che pone dei limiti sulla massima coppia scambiabile tra i due assali.

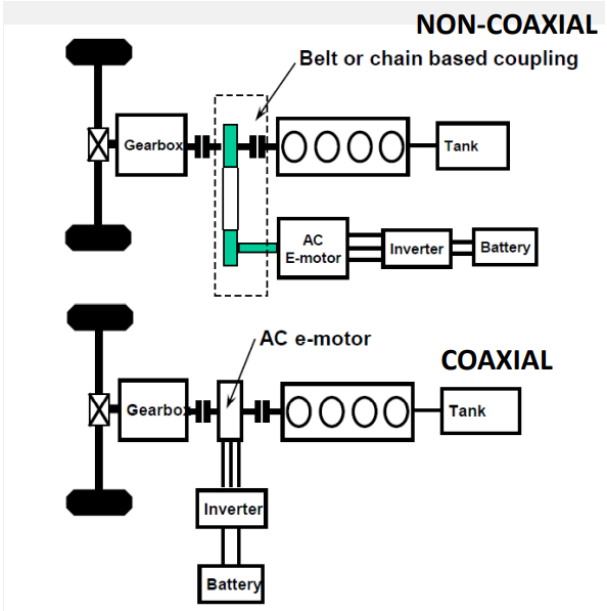

Figura 1.7: Schema single-shaft coassiale e non-coassiale

In una configurazione Parallela può essere fatta un'ulteriore classificazione sulla base della posizione della macchina elettrica.

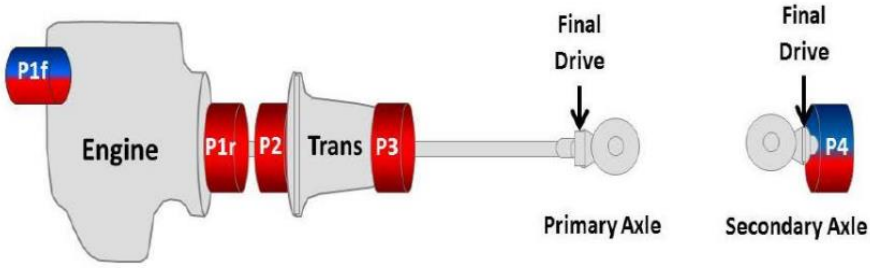

Figura 1.8: Schema posizione macchina elettrica

P1: macchina elettrica collegata direttamente al motore (f = anteriore e r = posteriore considerando una posizione longitudinale del motore). In particolare P1f potrebbe essere un *Single-shaft non-coassiale* mentre P1r un *Single-shaft coassiale* senza frizione tra motore e macchina elettrica.

P2: la macchina elettrica è collegata alla trasmissione lato motore, con la possibilità di disaccoppiarla dal motore tramite una frizione dedicata. P2 rappresenta un *Single-shaft coassiale* con frizione

P3: la macchina elettrica è posizionata tra la trasmissione e il differenziale. P3 rappresenta un *Single-shaft coassiale* dove però il collegamento avviene a valle della trasmissione.

P4: la macchina elettrica è posizionata sull'assale secondario (motore termico sull'asse primario). Questa rappresenta un'architettura *Double-drive.*

1.3 Principali strategie di gestione dell'energia nei veicoli ibridi

Nei veicoli ibridi l'introduzione di una o più sorgenti di potenza rispetto ad un veicolo convenzionale introduce una maggiore complessità del sistema di gestione energetica del veicolo. Diventa di fondamentale importanza determinare in ogni istante di tempo di utilizzo del veicolo la frazione di potenza fornita dal motore a combustione interna e quella dal motore elettrico, al fine di minimizzare il consumo di combustibile (FC) e di mantenere lo stato di carica della batteria (SOC) all'interno di un range prefissato nell'intorno del suo valore di partenza. La strategia di controllo consiste in un algoritmo che regola le condizioni di funzionamento del *powertrain,* in particolare converte una serie di dati di input (come velocità del veicolo, accelerazione ecc.) in decisioni sui flussi di potenza richiesti ai componenti del *powertrain* per soddisfare la coppia richiesta dal conducente.

Le principali strategie di controllo si dividono in quattro macro-categorie:

- **Metodi di ottimizzazione globale:** tali algoritmi non possono essere usati in applicazioni *on-board* poiché utilizzano un approccio di tipo *backworld*, ossia richiedono la conoscenza pregressa del ciclo guida e risultano molto pesanti dal punto di vista computazionale, quindi vengono utilizzati principalmente per risolvere complessi problemi di *benchmark* off-line. *Dynamic Programming* (DP) è uno dei più famosi metodi di ottimizzazione globale che può essere utilizzato per identificare l'ottimale strategia di controllo in problemi discretizzati nei veicoli ibridi.

- **Metodi di ottimizzazione istantanea:** tali metodi identificano l'ottimale gestione di potenza del motore termico (ICE) e della macchina elettrica (EM) attraverso la minimizzazione di una funzione di costo istantanea, cioè ad ogni istante di tempo della missione di guida e quindi possono essere teoricamente applicati on-line. Se la funzione di costo è opportunatamente definita, potremmo ottenere una soluzione molto vicina all'ottimo globale. La più comune di queste strategie è *Equivalent Consumption Minimization Strategy* (ECMS), che prevede la minimizzazione del consumo di combustibile e la conversione dell'energia erogata dalla batteria in un consumo di combustibile equivalente, utilizzando un opportuno fattore di conversione. Tuttavia affinché tale metodo dia una strategia di controllo approssimativamente simile a quella ottimale tale fattore deve essere calcolato su una determinata missione di guida.
- **Metodi euristici:** tali metodi includono tecniche di controllo, come logiche fuzzy e metodi basati su regole fisse. Questi metodi di solito non forniscono una strategia ottimale di controllo ma hanno il grosso vantaggio di essere facilmente implementabili e applicabili on-line sul veicolo, per questa ragione attualmente questi sono gli algoritmi più utilizzati nella pratica nei veicoli ibridi.
- **Machine learning:** con il termine *Machine learning* (ML) si indicano un insieme di tecniche dell'intelligenza artificiale (IA). Tali sistemi sono in grado di generare un livello di autonomia in modo tale di gestire situazioni molto complicate a valle di un'operazione di allenamento. Inoltre gli algoritmi di intelligenza artificiale sono applicabili *on-board* e risultano essere tendenti all'ottimo, cioè sono in grado di imparare nel corso delle prove e quindi sono in grado di fornire nel corso del tempo un miglioramento progressivo delle proprie prestazioni. In letteratura sono stati definiti metodi molto diversi per consentire l'apprendimento automatico delle macchine, ma in generale si possono riassumere tre principali famiglie di algoritmi: *Unsupervised ML algorithms, Supervised ML algorithm, Reinforcement learning algorithms.* Nell'approccio *Unsupervised* i dati di *training* contengono solo l'input, e non il corrispondente output, quindi l'algoritmo cerca *pattern* e similitudini fra i dati forniti, scegliendo in autonomia i criteri di raggruppamento. Nel *Supervised* l'algoritmo impara ad associare ad ogni input l'output corretto tramite *labeled data* (dati di allenamento composti da coppie input-output). Infine nel *Reinforcement learning (RL)* il sistema esegue della azioni in un ambiente in modo tale da massimizzare una ricompensa definita dallo sviluppatore.
  Negli ultimi anni gli algoritmi di *Reinforcement Learning* stanno trovando sempre più spazio nel mondo della mobilità per quanto riguarda l'ottimizzazione energetica nei veicoli ibridi, in particolare possiamo distinguere il *Q-Learning* (QL) in cui le variabili utilizzate sono discrete e il *Deep Reinforcement Learning,* che combina i concetti del RL con quelli del *Deep Learning,* cioè con l'utilizzo di reti neurali. Come evoluzione del QL tabulare sono quindi stati utilizzati algoritmi come *Deep Q-Network* (DQN) e *Double Deep Q-Network* (DDQN), grazie ai quali è possibile utilizzare delle variabili continue, eliminando gli errori dovuti alla discretizzazione.

1.4 Scopo

Obbiettivo di questa tesi è quello di testare un algoritmo di *Deep Q-Network* (DQN) in un'applicazione di *energy management* per una *passenger car* ibrida con architettura parallela p2 su ciclo guida WLTP e su altri cicli guida reali (cicli clust), i quali non vengono usati nelle procedure omologative ma sono molto utili per fare analisi in ambito di simulazione. L'introduzione di più sorgenti di potenza installate a bordo aumenta i gradi di libertà e quindi rende più complesso il sistema di ottimizzazione energetica. In particolare bisogna stabile in ogni istante temporale la ripartizione dei flussi di potenza erogati dal motore termico e quello elettrico, al fine di minimizzare i consumi di carburante lavorando comunque in modalità *charge-sustaining*, quindi controllando lo stato di carica della batteria. Nel dettaglio è stata valutata l'influenza dei vari iperparametri della rete neurale sulle performance di un agente DQN. Sulla base delle variabili di controllo sono stati opportunatamente selezionati gli stati che meglio caratterizzano l'ambiente. Vi è stata anche un'adeguata calibrazione dei coefficienti che pesano il SOC (*state of charge*) e l'FC (*fuel consumption*) presenti all'interno della funzione di *reward*, con l'obbiettivo di ottenere una ricompensa quanto più possibile *FC-oriented*, ed infine è stato individuato un opportuno rapporto *exploration-exploitation*. Dopo aver trovato una soluzione sub-ottimale al problema sul WLTP l'agente è stato testato anche su altri cicli guida reali (ciclo clust4, clust7, clust11, clust12) per verificarne l'adattabilità al variare delle condizioni di guida.

2. *Reinforcement Learning*

2.1 Concetti chiave del *Reinforcement Learning*

In questa sezione esaminiamo in modo più approfondito il *Reinforcement learning*, poiché il DQN rientra proprio in questa categoria.
Il RL si basa sul concetto di interazione tra agente e ambiente, in particolare l'agente è definito come l'ente che ha il compito di scoprire in totale autonomia il corretto comportamento da seguire, mentre l'ambiente è definito come tutto ciò che circonda l'agente e con cui esso interagisce. Ad ogni *time step* l'agente riceve un' osservazione o_t, che nel nostro caso la considereremo equivalente allo stato s_t, e un reward r_t. In seguito l'agente compie un'azione a_t, che modifica l'ambiente dallo stato s_t a s_{t+1}. Inoltre alla transizione (s_t, a_t, s_{t+1}) è associata una nuova ricompensa r_{t+1}. Questo ciclo continua fino al termine dell'episodio e lo scopo dell'agente è quello di massimizzare la somma di tutte le ricompense ricevute (*Cumulative reward*).

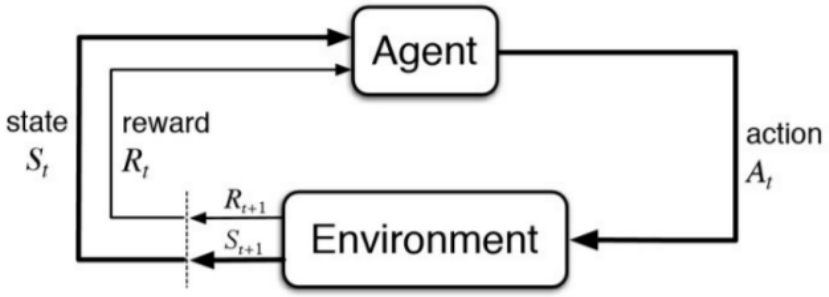

Figura 2.1 Schema generico di un algoritmo RL

2.1.1 Processo di decisione Markoviano (MDP)

Se si vuole utilizzare un metodo di apprendimento automatico bisogna dare una descrizione formale dell'ambiente. Non interessa sapere esattamente com'è fatto l'ambiente, interessa piuttosto fare delle ipotesi generali sulle proprietà che l'ambiente possiede. Nel RL si assume di solito che l'ambiente possa essere descritto da un Processo di Decisione Markoviano (*Markov Decision Process* o MDP). Un *Markov Decision Process* è formalmente definito da:

1. Uno spazio degli stati **S**,
2. Uno spazio delle azioni **A** che possono essere intraprese in funzione dello stato s,
3. $P_a(s, s') = P_r(s_{t+1} = s' \mid s_t = s, a_t = a)$, che esprime la probabilità che l'azione a nello stato s al tempo t conduca allo stato s' al tempo t +1,
4. $R_a(s, s')$, indica la ricompensa ricevuta a seguito del passaggio dallo stato s allo stato s', tramite l'azione a.

Uno stato s_t è detto Markoviano, cioè goda della proprietà di Markov se:

$$P[s_{t+1}|s_t] = P[s_{t+1}|s_1, \dots, s_t]$$

o in altre parole se il futuro è indipendente dal passato dato il presente, ciò implica che tutte le informazioni rilevanti del passato sono catturate nello stato attuale.

2.1.2 Return

Nei problemi di RL l'obbiettivo dell'agente è quello di individuare una *policy* che massimizzi la somme delle ricompense ottenute, perciò in questa sezione viene introdotto il concetto di *Return*. Nel caso di orizzonte finito il *Return* è definito come una semplice somma dei *reward* puntuali ottenuti in ciascuno istante temporale *t*:

$$G_t = \sum_{t=0}^{T} r_t$$

In problemi continuativi, ossia nel caso di orizzonte infinito, è necessario introdurre il *discounted factor* $\gamma \in (0,1)$, in modo tale da rendere finito il *Return* al fine di poter essere massimizzato. In questo caso si parla di *Discounted return* dove *i reward* sono scontati di un fattore γ. Al diminuire di tale fattore diminuisce l'incidenza dei *reward* futuri sul *Return,* privilegiando le ricompense immediate a scapito di quelle future. Un tipico valore di γ è 0.99, che da molta importanza alle ricompense future ma predilige leggermente quelle immediate.

$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$$

2.1.3 Funzione Valore e *policy*

In questa sezione introduciamo il concetto di Valore V o *Value Function* di un certo stato s_t, che quantifica quanto è conveniente essere in un certo stato in relazione alla sequenza di tutti gli stati futuri che verranno raggiunti a partire da s_t seguendo una determinata *policy* π.

$$V_\pi(s) = E_\pi\left(\sum_{k=0}^{\infty} \gamma^k r_{t+k} \mid s_t = s\right)$$

E_π rappresenta *l' Expectation*, ovvero il *Return* atteso seguendo la policy π e non quello realmente ottenuto. E' possibile definire anche un' *action Value Function***,** chiamata anche *Q-function***,** in modo tale da conoscere quanto è 'buono' intraprendere una determinata azione a_t al tempo t a partire da uno stato s_t seguendo una *policy* π.

$$Q_\pi(s,a) = E_\pi\left(\sum_{k=0}^{\infty} \gamma^k r_{t+k} \mid s_t = s, a_t = a\right)$$

L'obbiettivo di un agente nel RL è quello di individuare una *policy* che porti alla massimizzazione del *Return* atteso. La scelta di un'azione è data dalla policy ed indichiamo quella ottimale con π^*, in particolare l'azione migliore è quella che massimizza il *Q-value* in ogni istante di tempo.

$$\pi^*(s) = argmax\, Q_{\pi^*}(s,a) \qquad \forall\, s \in S$$

2.1.4 Equazione di Bellman

E' possibile esprimere il *Q-value* di una *policy* generica π in modo ricorsivo tramite l' equazione di Bellman, sommando al *reward* atteso il *discounted factor* moltiplicato per il *Q-value* dello stato successivo ed associato all'azione a'seguendo la stessa *policy* π:

$$Q_\pi(s,a) = E_\pi(r_t + \gamma Q_\pi(s_{t+1}, a') \mid s_t = s, a_t = a)$$

L'equazione di Bellman può essere usata anche per definire il *Q-value* che segue la *policy* migliore π^*. In particolare il valore Q per un'azione a a partire dallo stato s è pari alla ricompensa immediata per quello stato e quell'azione, a cui sommiamo il massimo valore possibile conosciuto tra tutte le azioni dello stato successivo moltiplicato per il fattore di sconto. Di seguito viene riportata l'equazione di ottimalità di Bellman:

$$Q_{\pi^*}(s,a) = E_{\pi^*}(r_t + \gamma \max_{a'} Q_{\pi^*}(s_{t+1}, a') \mid s_t = s, a_t = a)$$

2.1.5 Strategia di esplorazione

In un problema di RL assume un ruolo fondamentale la strategia di esplorazione, ossia la determinazione della *policy* π con cui l'algoritmo sceglie le azioni da compiere.

Solitamente viene utilizzata la $\varepsilon - greedy$ policy, che consiste nello scegliere la mossa ritenuta migliore con probabilità 1- ε , e scegliere una mossa casuale con probabilità ε :

$$\pi(s_t) = \begin{cases} azione\ randomica\ a \in A\ con\ probabilità\ \varepsilon \\ arg\max_{a' \in A} Q(s_t, a')\ con\ probabilità\ 1 - \varepsilon \end{cases}$$

E' opportuno controllare il parametro ε in modo tale da avere un giusto rapporto tra *exploration* e *exploitation*, garantendo un elevato grado di esplorazione nelle fasi iniziali dell'allenamento e successivamente avere un quasi completo sfruttamento. Per ridurre la possibilità che l'algoritmo rimanga intrappolato in massimi locali è necessario lasciare comunque una piccola probabilità di esplorazione anche quando l'agente crede di aver trovato una strategia efficace.

2.2 Q-Learning

Uno degli algoritmi più famosi è sicuramente il *Q-Learning*, in particolare dato uno stato l'algoritmo associa ad ogni possibile azione un *Q-value*, ossia un valore numerico che fornisce la bontà dell'azione stessa. Un'azione risulta tanto più efficace quanto più è alto il *Q-value*. Tutti i *Q-values* che seguono la *policy* migliore vengono salvati in una tabella, dove ogni riga corrisponde ad uno stato e ogni colonna ad un'azione.
L'obbiettivo è quello di approssimare questi *Q-values* tramite l'equazione di ottimalità di Bellman, eseguendo molti episodi e aggiornando la tabella. In particolare nel *Q-Leaning* l'aggiornamento dell'*action value function* avviene tramite il metodo *Temporal-Difference* (TD):

$$Q(s_t, a_t) \Leftarrow old\ Q\ value - TD\ error$$

$$Q(s_t, a_t) \Leftarrow old\ Q\ value + (\ new\ Q\ value - old\ Q\ value)$$

$$Q^{new}(s_t, a_t) \Leftarrow Q(s_t, a_t) + \alpha \cdot \left(r_t + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t)\right)$$

dove $\alpha \in (0,1)$ rappresenta il tasso di apprendimento (*learning rate*), ossia dice quanto velocemente impara l'agente. In particolare un fattore pari a 0 non permetterebbe all'agente di apprendere, mentre un fattore pari a 1 farebbe si che l'agente si interessi solo delle informazioni recenti.
Come detto precedentemente il *Q-Learning* si basa su una *Q-table*, tuttavia in applicazioni reali il numero di stati potrebbe essere enorme rendendo il problema proibitivo dal punto di vista computazionale. Inoltre, un'altra limitazione del *Q-Learning* tabulare è l'operare in spazi discreti per descrivere fenomeni in cui lo spazio è continuo. Di conseguenza affinché il problema sia valido dal punto di vista fisico occorre un'adeguata discretizzazione degli stati e delle azioni prese dall'agente, rendendo in alcune circostanze impossibile l'adozione di una forma tabellare, poiché porta al cosiddetto problema della *curse of dimensionality.*

2.3 Reti neurali

Per superare le limitazioni del *Q-Learning* tabellare e quindi risolvere problemi del mondo reale sono stati introdotti degli approssimatori di funzione. Una possibile soluzione possono essere le reti neurali che permettono di operare in spazi continui. In particolare le reti neurali utilizzano una funzione Q piuttosto che una *Q-table*, grazie alla quale sono in grado di approssimare i *Q-values* di ogni possibile azione dato uno stato.
Una rete neurale (artificiale in inglese *artificial neural network*, abbreviato come ANN o NN) è un modello matematico composto da "neuroni" artificiali, che si ispira ad una rete neurale biologica.

Possiamo considerare una rete neurale come una scatola nera, con degli input (*input layer*), degli strati intermedi (*hidden layers*) in cui "succedono le cose", e degli output (*output layer*) che forniscono il risultato finale. Ogni strato delle rete neurale è costituita da un certo numero di neuroni ed ogni neurone è connesso con ciascun neurone presente nullo strato ad esso successivo.

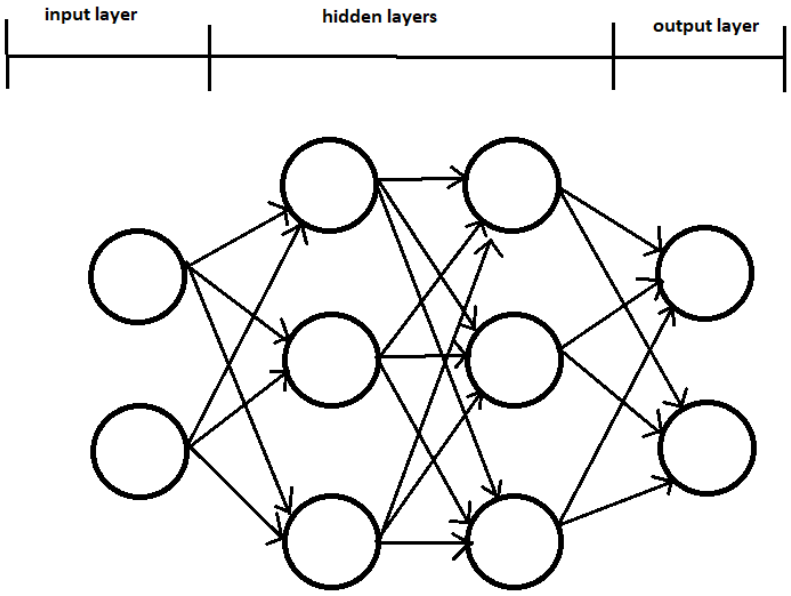

Figura 2.2 : modello di una generica rete neurale artificiale

2.3.1 *Forward propagation*

Il processo di predizione della rete neurale prende il nome di *Forward propagation*, ossia abbiamo un flusso di informazioni che parte dall'*input layer,* si propaga nell'*hidden layer* ed infine giunge all'*output layer*. Durante tale processo ogni neurone compie la stessa operazione, schematizzata in Figura 2.3. Il generico neurone calcola il valore di uscita z_j facendo una somma pesata degli input X_i, alla quale viene aggiunto il *bias (b)***.** A questo risultato viene applicata una funzione di attivazione (*f*), che non fa altro che trasformare matematicamente il valore prima di passarlo allo strato successivo. Inizialmente i pesi W_i sono scelti casualmente dalla rete e l'obbiettivo principale è quello di regolare correttamente i pesi e bias al fine di predire correttamente i *Q-values*.

$$z_j = \sum_i X_i W_i + b$$

$$a_j = f(z_j)$$

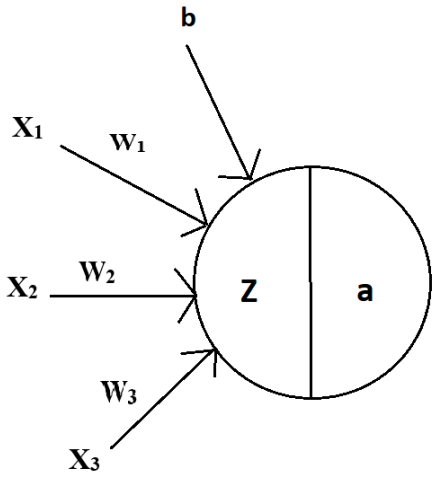

Figura 2.3: modello di un generico neurone artificiale

2.3.2 Funzioni di attivazione

Lo scopo di una rete neurale è quello di approssimare qualsiasi funzione, perciò è necessario introdurre un fattore di non linearità, ossia la funzione di attivazione. Le funzioni di attivazioni più popolari per i problemi di RL sono: *sigmoide, tanh* e ReLU.

Sigmoide: $f(x) = \frac{1}{1+e^{-x}}$

Il vantaggio di questa funzione è quello di essere differenziabile in ogni suo punto e di comprimere i valori di output in un range tra 0 e 1. Tuttavia per valori di input molto grandi la convergenza è molto lenta, in quanto la sua derivata tende a zero causando problemi di *vanisching gradient.*

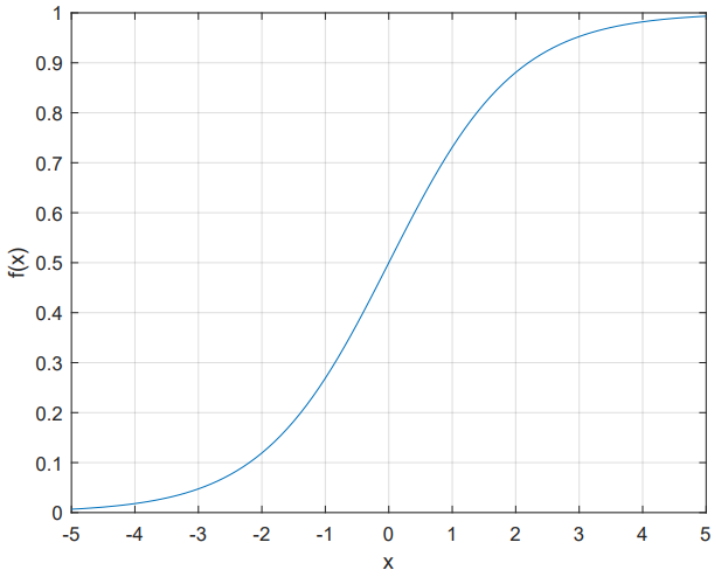

Figura 2.4: Funzione sigmoide

Tanh: $f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$

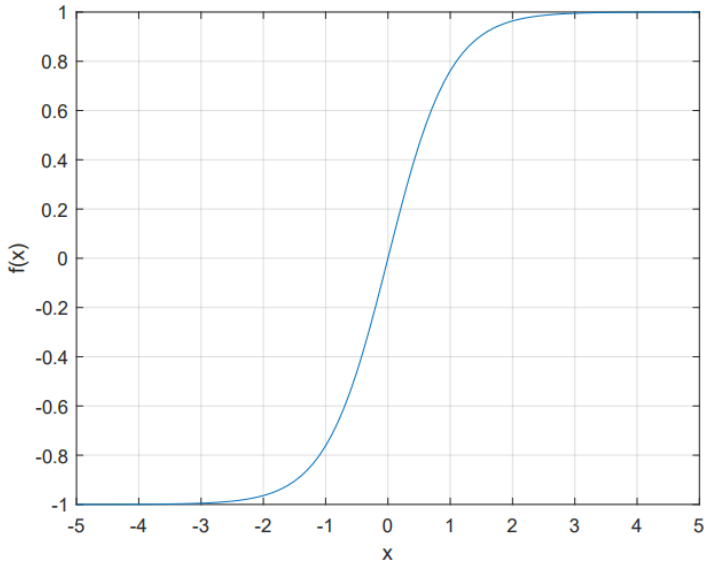

Figura 2.5: Funzione tanh

La funzione tangente iperbolica è simile alla funzione sigmoide, dalla quale differisce per il semplice fatto che il suo codominio va da -1 a 1.

ReLU: $f(x) = \max(0, x)$

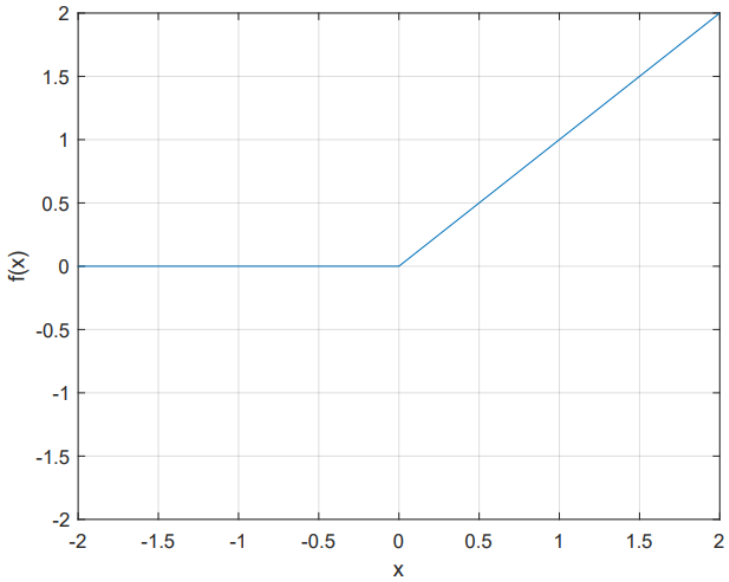

Figura 2.6: Funzione ReLU

Tale funzione è tra le più utilizzate nell'ambito DQN, restituisce zero nel caso in cui la somma pesata dei segnali in input è minore 0, oppure lascia tutto invariato per valori uguali o maggiori di zero. Il calcolo della derivata di tale funzione è molto semplice: per tutti i valori negativi è uguale a 0, mentre per quelli positivi è uguale a 1. Nel punto angoloso in corrispondenza dell'origine la derivata invece è indefinita, ma viene comunque impostata a zero per convenzione. Il codominio della funzione spazia in questo caso da zero ad infinito.

2.3.3 Funzione di costo e *backpropagation*

L'ottimizzazione di una rete neurale si basa sul concetto di una funzione di costo J, definita come la differenza tra valore di output previsto e atteso elevata al quadrato:

$$J = (previsione\ Qvalue - target\ Qvalue)^2$$

La rete neurale quindi apprende generando un segnale di errore quadratico che utilizza per aggiornare i propri pesi, in modo tale da minimizzare questa funzione di costo ed ottenere delle previsioni quanto più accurate possibili.
La funzione di costo J è funzione dei pesi e dei *bias* della rete stessa, perciò dopo aver calcolato la derivata parziale di J per ogni peso, l'ottimizzazione avviene con l'algoritmo *gradient descent*, che aggiorna ogni peso nel seguente modo:

$$W_i \Leftarrow W_i - \alpha \frac{\partial J}{\partial W_i}$$

Ossia andiamo a sottrarre al vecchio peso una frazione della pendenza della retta tangente alla funzione costo. In particolare $\alpha \in (0.1)$ prende il nome di *learning rate* ed indica lo *step size* di modifica dei pesi, quindi controlla la velocità di apprendimento. Più nel dettaglio dei valori di α elevati permetterebbero alla rete di apprendere più velocemente col rischio però di saltare il minimo, ottenendo un set di pesi subottimali, mentre un α troppo basso mi permette di ricavare un set di pesi ottimali provocando però un allungamento dei tempi di allenamento. Perciò il *learning rate* rappresenta uno degli iperparametri più complicati da regolare per ottenere delle buone performance. Questo aggiornamento dei pesi della rete, che procede dall'*output layer* fino *all'inner layer*, prende il nome di *backpropagation*, ossia propagazione "all'indietro" dell'errore, permettendomi di ridurre la funzione di costo.

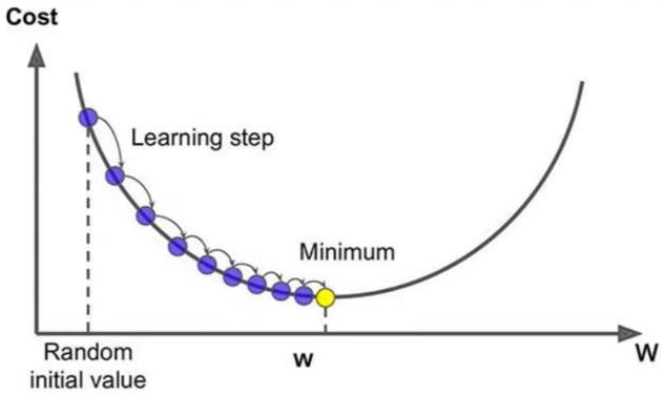

Figura 2.4: Funzione costo in funzione del generico peso w

2.4 Deep Q-Learning

Per risolvere i problemi legati al *Q-Learning* tabellare è stato introdotto l'algoritmo di *Deep Q-Learning* o *Deep Q-Network* (DQN), che accoppia il *Q-Learning* con l'utilizzo di reti neurali artificiali in modo tale da operare in spazi continui, trovando in questo modo maggiore applicabilità in problemi reali come la gestione energetica dei veicoli ibridi elettrificati. In questo algoritmo l'apprendimento non consiste nell'aggiornare una tabella ma nel variare opportunatamente i pesi della rete neurale attraverso il processo di *backpropagation*, al fine di poter predire correttamente i *Q-values* di ogni possibile azione dato uno stato. Nel DQN l'agente inizialmente stima in maniera arbitrale i valori di Q ed esplora l'ambiente utilizzando una $\varepsilon - greedy$ policy, successivamente durante i vari episodi di *training* si addestra a predire i *Q-values* in maniera sempre più accurata modificando i pesi sulla base della *loss function:*

$$L_t = \left(E[r_t + \gamma \max_{a'} Q(s_{t+1}, a')] - Q(s_t, a_t) \right)^2$$

L_t quindi rappresenta l'errore commesso e l'obbiettivo è quello di far tendere questa differenza a zero in modo tale che la previsione si avvicini il più possibile al risultato desiderato, in particolare $E[r_t + \gamma \max_{a'} Q(s_{t+1}, a')]$ rappresenta l'*expected return* target mentre $Q(s_t, a_t)$ è il valore predetto dalla rete. Minimizzando questa differenza l'agente è in grado di prendere delle decisioni in termini di azioni, che portano alla massimizzazione del *Discounted Return* sulla base di una funzione di *reward* precedentemente definita.

2.4.1 Architettura DQN

Gli elementi principali di un'architettura DQN sono: *rete Q*, *rete Q target* e *Experience replay*. In particolare la *rete Q* prevede il *Q-value* di una determinata azione *a* a partire dallo stato *s* in un generico istante di tempo *t*, la rete target invece prevede il valore Q target, dato dalla ricompensa immediata a cui viene sommato il miglior valore di Q tra tutte le azioni che possono essere intraprese dallo stato s_{t+1}. In un algoritmo di DQN i pesi di una rete Q vengono aggiornati in ogni istante

temporale, perciò affinché il valore previsto si avvicini sempre di più a quello desiderato diventa di fondamentale importanza introdurre una seconda rete neurale (*rete Q target*), i cui pesi non vengono aggiornati ad ogni istante temporale ma con una certa frequenza durante gli episodi di *training*, chiamata *target update frequence.* Se così non fosse i valori Q target non rimarrebbero stabili e la rete Q non farebbe altro che inseguire un bersaglio in movimento rendendo impossibile la convergenza. Perciò l'utilizzo di una rete target consente di avere un allenamento più stabile e dopo un prefissato intervallo temporale i pesi appresi dalla *rete Q* vengono copiati all'interno della *rete Q target*. Assume un ruolo rilevante anche l'*Experience replay*, all'interno della quale vengono salvate un determinato numero di esperienze, dove ogni esperienza è una tupla (s_t, a_t, r_t, s_{t+1}). In particolare durante l'addestramento in ogni istante temporale vengono estratte in maniera randomica N tuple, in modo tale da consentire alla rete di apprendere dei pesi che si generalizzano bene con tutti gli scenari che l'agente dovrà gestire. Perciò l'introduzione dell'*Experience replay* aiuta ad attenuare il rumore e ad ottenere un allenamento più stabile.

2.4.2 Flusso di lavoro

Il DQN viene addestrato su più passaggi temporali in molti episodi ed esegue una sequenza di operazioni in ogni fase temporale.

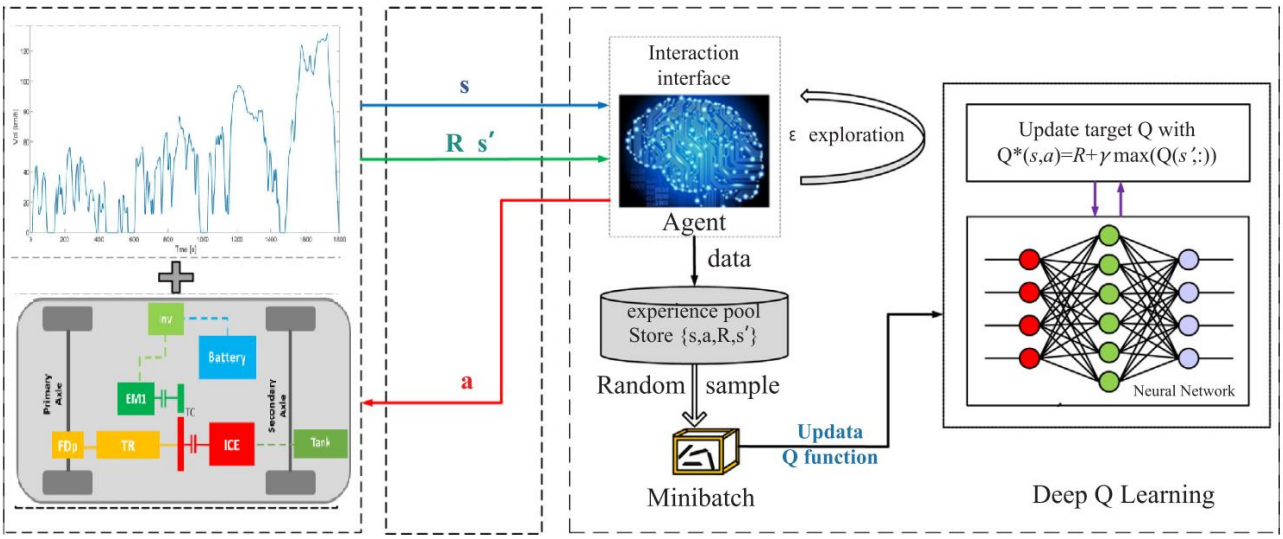

Figura 2.5: Diagramma flusso di lavoro DQN per EMS di un HEV

In una fase iniziale si ha la generazione dei dati di addestramento senza alcun aggiornamento dei pesi della rete, in particolare l'agente interagendo con l'ambiente produce delle tuple $(s_t, a_t, r_t, s_{t+1}, done)$ che vengono accumulate all'interno del buffer di esperienze (*Experience replay*):

- s_t rappresenta lo stato del sistema all'istante di tempo *t*
- a_t rappresenta l'azione compiuta all'istante di tempo *t* dallo stato s_t
- r_t rappresenta la ricompensa ottenuta dall'agente all'istante di tempo *t* compiendo l'azione a_t
- s_{t+1} rappresenta lo stato del sistema al tempo *t + 1* a seguito dell'azione a_t

- $done$ è una variabile booleana, in particolare se $done = 1$ si ha la terminazione dell'episodio corrente. Tale situazione può verificarsi in tre casi, nello specifico se lo stato s_t è uno stato terminale, ossia è l'ultimo stato di un episodio, se l'azione compiuta dall'agente è *unfisible,* oppure se lo stato di carica della batteria esce al di fuori del range prefissato prima del lancio della simulazione.

Dopo aver accumulato un quantitativo sufficiente di esperienze ha inizio l'addestramento della rete, ossia in ogni *time-step* viene prelevato un *batch* di campioni in maniera randomica dal *buffer*, che entrano in input alla rete Q e alla rete target e vengono utilizzati per calcolare la *loss function* L_t. Gli errori calcolati dalla *loss function* saranno propagati all'indietro nella rete Q mediante un passo *backword* (*backpropagation*), seguendo la logica di discesa del gradiente. Tutte queste operazioni vengono eseguite in ogni fase temporale, mentre la rete target viene addestrata solo con una certa frequenza (*target update frequence*), andando a copiare i pesi della rete Q all'interno della rete target.

3. Modellazione del veicolo e del ciclo guida

In questo capitolo viene presentato il modello del simulatore che si compone del modello del veicolo e di tutti i suoi sottocomponenti (modello del motore termico, della macchina elettrica, della batteria, della trasmissione), i quali sono stati sviluppati in ambiente Matlab ed utilizzati per il lavoro di tesi. Infine vengono presentati anche i cicli guida utilizzati per simulare il comportamento del veicolo stesso su tipici scenari di guida reale.

3.1 Architettura veicolo

Per il nostro caso studio è stata considerato una *passenger car* ibrida con architettura parallela p2, che si compone di una sola macchina elettrica (EM1) e di un motore termico (ICE) collegati tra loro tramite un accoppiamento tra ruote dentate (TC). Grazie alla presenza di due frizioni è possibile disaccoppiare uno dei due motori, in modo tale da lavorare in modalità puro elettrico o puro termico prima della trasmissione (TR) e il differenziale posto sull'assale primario (FDp). Il flusso di potenza scorre dalla batteria (*Battery*) alla macchina elettrica in fase di trazione, tramite un inverter (Inv), o viceversa nel caso di frenata rigenerativa, facendo funzionare la macchina elettrica come generatore. In un'architettura p2 tutta la potenza, sia quella proveniente dal motore elettrico che quella proveniente dal motore termico, viene trasferita sull'assale frontale.

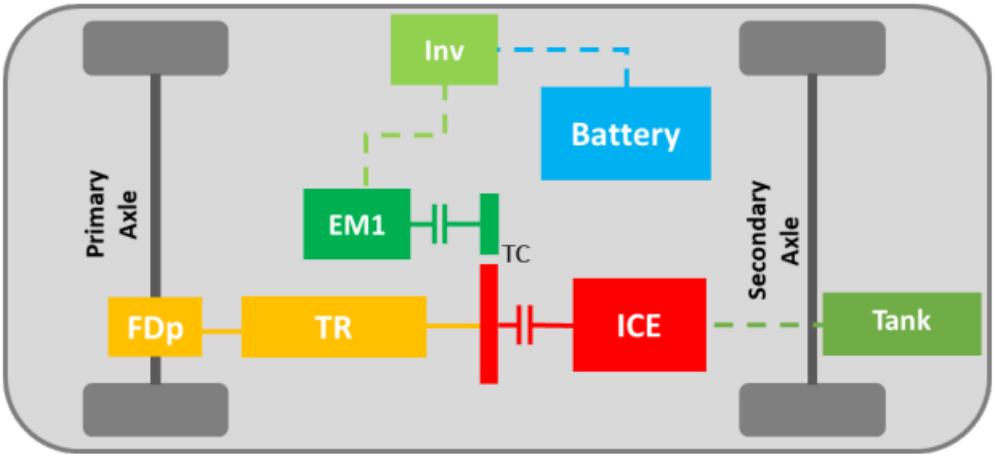

Figura 3.1: Schema architettura p2

3.1.1 *Powerflow*

Tutte le architetture ibride devono soddisfare la coppia richiesta dal guidatore utilizzando il motore elettrico e quello termico in maniera esclusiva o in contemporanea. Il modo in cui questo avviene è definito dal *powerflow,* ossia dalle possibili distribuzioni della potenza. Il flusso di potenza è descritto da uno o due parametri che teoricamente possono variare su un intervallo continuo; tuttavia devono essere discretizzati per poter eseguire delle simulazioni di *energy-management* per HEV. In altre parole deve essere specificato un insieme finito di flussi di potenza tramite α. Per un'architettura p2 è possibile definire 4 categorie di *powerflow*:

- PE (*pure electric*): il motore elettrico fornisce da solo la potenza necessaria per muovere il veicolo, mentre il motore termico è spento. La frenata rigenerativa è inclusa in questa categoria.
- PT (*pure thermal*): il motore termico fornisce da solo la potenza necessaria per muovere il veicolo e il motore elettrico è spento.
- PS (*power-split*): sia il motore termico che quello elettrico forniscono la coppia necessaria per la propulsione del veicolo.
- BC (*battery-charging*): il motore termico è acceso e la macchina elettrica funge da generatore. ICE fornisce più della potenza necessaria per muovere il veicolo, la potenza in eccesso viene convertita dalla macchina elettrica in energia elettrica e immagazzinata nella batteria.

Il flusso di potenza può essere identificato con α, che definisce la quota parte di potenza fornita dalla macchina elettrica:

$$\alpha = \frac{P_{EMp}}{P_{req}}$$

Dove P_{EMp} è la potenza fornita dalla macchina elettrica (o assorbita se negativa) e P_{req} è la potenza complessiva richiesta per la trazione del veicolo.

Powerflows		$\boldsymbol{\alpha}$
PE	EM1	1
PT	ICE	0
PS	EM1 + ICE	$0 < \alpha < 1$
BC	EM1 + ICE	$\alpha < 0$

Tabella 3.1: *Powerflows*

3.2 Modello del veicolo

La potenza necessaria per la trazione del veicolo è considerata come somma delle potenze resistenti al moto, dovute alle seguenti forze: resistenza al rotolamento, resistenza aerodinamica, resistenza dovuta alla pendenza della strada e resistenza legata alla forza di inerzia. Successivamente vengono riportate le equazioni utilizzate per il calcolo della potenza totale di trazione P_{tot}:

$$P_{tot} = P_{rot} + P_{aero} + P_{rs} + P_{inerzia}$$

$$P_{rot} = K_{rot} m\, g\, cos(\beta) V$$

$$P_{aero} = \frac{1}{2}\rho_{aria} C_x A_f V^3$$

$$P_{rs} = m\, g \sin(\beta)\, V$$

$$P_{inerzia} = m \frac{\Delta V_v}{\Delta t} V$$

Dove:

- K_{rot} è il coefficiente di resistenza al rotolamento
- m è la massa totale del veicolo [kg]
- β è la pendenza della strada [rad]
- V è la velocità del veicolo [m/s]
- ρ_{aria} è la densità dell'aria
- C_x è il coefficiente di resistenza aerodinamica
- A_f è l'area frontale del veicolo

Infine, la potenza complessiva richiesta per l'avanzamento del veicolo è ottenuta sommando alla P_{tot} la potenza dissipata a causa dell'inerzia dei diversi componenti del veicolo.
Sulla base di tali potenze viene definita la condizione di trazione del veicolo: considerando un istante di tempo t e il passo di discretizzazione temporale Δt, il veicolo viene considerato in trazione nell'intervallo di tempo compreso tra $t - \Delta t$ e t se la somma delle potenze all'istante di tempo $t - \Delta t$ e all'istante t è maggiore di zero.
Di seguito vengono riportati i principali parametri del veicolo:

Parametri veicolo	Valore	Unità di misura
Peso a vuoto	750	kg
Carico massimo veicolo	300	kg
Diametro del pneumatico	0,6014	m
Numero ruote veicolo	4	-
Inerzia della ruota	1,05	kg·m^2
Resistenza al rotolamento	0,0879	N/kg
Coefficiente di resistenza aero	0,27	-
Area frontale	2.19	m^2
Distribuzione freno anteriore/posteriore	0,75	-
Potenza media richiesta da ausiliari ICE	0	W
Potenza media richiesta da ausiliari batteria	0	W
Consumo medio energia batteria	0,5	kWh/km

Tabella 3.2: Parametri veicolo

3.2.1 Modello motore a combustione interna

Le prestazioni del motore sono modellate utilizzando dei dati derivati sperimentalmente. La portata massica del consumo di combustibile è ottenuta interpolando una mappa 2D, che è funzione della potenza meccanica e della velocità del motore:

$$\dot{m}_{FC} = \dot{m}_{FC}(P_{ICE}, \omega_{ICE})$$

I dati sperimentali permettono di determinare la caratteristica velocità-coppia del motore, quindi forniscono informazioni riguardo la velocità massima e minima ed i limiti legati alla potenza. In

particolare è stato utilizzato un motore ad accensione comandata ed i parametri principali che caratterizzano il motore vengono riportati nella Tabella 3.3:

Principali parametri	Valore	Unità di misura
Cilindrata	1	[L]
Coefficiente d'inerzia	0,14	$[kg \cdot m^2]$
Velocità minima	1000	[rpm]
Velocita massima	6250	[rpm]
Densità combustibile	0,78	$[kg/L]$
Potere calorifico inferiore combustibile	43,4	[MJ/kg]

Tabella 3.3: Caratteristiche ICE

3.2.2 Modello macchina elettrica ed inverter

Il modello della macchina elettrica simula la conversione di potenza dalla forma meccanica a quella elettrica e viceversa, considerando le perdite di energia tramite mappe di efficienza, che sono funzione della potenza e velocità di rotazione della macchina elettrica. Le mappe relative alla conversione della potenza elettrica-meccanica e meccanica-elettrica sono precalcolate per velocizzare la simulazione.

$$P_{EM,e} = \eta_{EM,m2e}(P_{EM,m}, \omega_{EM}) \cdot P_{EM,m}$$

$$P_{EM,m} = \eta_{EM,e2m}(P_{EM,e}, \omega_{EM}) \cdot P_{EM,e}$$

I dati sperimentali forniscono inoltre informazioni riguardo il limite di potenza inferiore (modalità generatore) e superiore (modalità motore), e sulla velocità massima di rotazione. Dell'inverter è stato fornito solamente la sua efficienza, pari a 0,95.

3.2.3 Modello batteria

In questo studio sono state adottate delle batterie agli ioni-litio, la cui struttura è costituita da celle/unità/moduli come visibile in Figura 3.2:

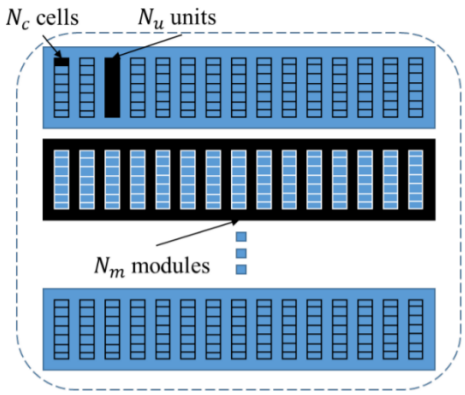

Figura 3.2 : Modello batteria

In questa sezione viene illustrata la procedura di dimensionamento della batteria. Il tool utilizzato calcola la potenza massima richiesta alla batteria, data la potenza massima della macchina elettrica ed il rapporto Potenza/Energia (PE). Il tool è in grado di determinare il numero di celle e come collegarle, partendo dalla tensione nominale della batteria e da una cella di riferimento le cui specifiche sono fornite in input.
I dati specificati in input sono la tensione nominale della cella V_c, la capacità nominale della cella C_c, il numero di celle in un'unità N_c, la tensione nominale di riferimento della batteria V_{nom}, la massa della batteria $M_{batt,ref}$ ed il numero totale di celle della batteria $N_{tot,ref}$.
L'energia della batteria E_{batt} è ottenuta dividendo la potenza di picco della macchina elettrica per il rapporto PE:

$$E_{batt} = \frac{P_{EM,max}}{PE}$$

Infine vengono specificati in input C-rate massimi della batteria in carica e scarica $C_{charhe,max}$ e $C_{dis,max}$. Il tool quindi ridimensiona la batteria ridefinendo il numero di unità che costituiscono un modulo e il numero di moduli stessi.

Nel dettaglio vengono implementate le seguenti equazioni:

Capacità unitaria	$C_u = C_c$
Capacità batteria	$C_{batt} = \frac{E_{batt}}{V_{nom}}$
Numero di unità in parallelo	$N_{pu} = \frac{C_{batt}}{C_u}$
Tensione unitaria	$V_u = V_c \cdot N_c$
Tensione del modulo	$V_m = V_u$
Numero di moduli in serie	$N_{sm} = \frac{V_{nom}}{V_m}$

Il modello della batteria si basa sul modello di resistenza interna. In questo modello la tensione di circuito aperto della batteria e la sua resistenza interna sono descritti in funzione dello stato di carica (SOC). Inoltre possono essere calcolate altre informazioni utili relative alla batteria, come la potenza massima della batteria in carica e in scarica, e la massima corrente a partire dal massimo C-rate. Infine sono stabiliti dei range massimi e minimi del SOC.

3.2.4 Modello trasmissione (gearbox)

Anche in questo caso sono stati utilizzati dei dati sperimentali, ed in particolare sono stati considerati sei rapporti di trasmissione possibili, l'efficienza della trasmissione in corrispondenza del rapporto di trasmissione innestato, il momento d'inerzia e la massa del cambio.
Di seguito vengono riportati i principali parametri del cambio:

Numero di marce	1	2	3	4	5	6
Rapporto di trasmissione	4,17	2,13	1,32	0,95	0,75	0,62
efficienza	0,945	0,956	0,969	0,965	0,954	0,953

Tabella 3.4: Parametri gearbox

3.3 Cicli guida

A partire dal 1° settembre 2017 in Europa i dati relativi al consumo di carburante per le nuove automobili immesse nel mercato sono stati determinati sulla base del ciclo WLTP (*Wolrdwide harmonized Light-Duty vehicles Test Procedure*), che sostituisce il NEDC (*New European Driving Cycle*).
L'algoritmo di DQN è stato inizialmente addestrato sul ciclo WLTP, il quale è suddiviso in 4 fasi che si differenziano non tanto per le accelerazioni ma per le velocità raggiunte: *low*, *medium*, *high*, *extra high*. Le fasi *low* e *medium* sono rappresentative delle condizioni urbane mentre le altre due sono rappresentative dell'extra urbano. In questo caso la durata del ciclo è di 1800 s e non 1180 s come nel NEDC, quindi è stata incrementata la distanza percorsa durante il ciclo. Vedendo il profilo di velocità è possibile notare che si tratta di un ciclo guida reale caratterizzato da accelerazioni molto rapide e quindi molto più severo rispetto al NEDC e le fasi di *Constant driving* e *Stop* si riducono notevolmente. Lo shift da NEDC a WLPT si prefigge l'obbiettivo di ridurre il gap presente tra le reali condizioni di guida e quelle dei test omologativi, dovuto alle innumerevoli semplificazioni che esistono nel voler creare un ciclo che sia esemplificativo della realtà.

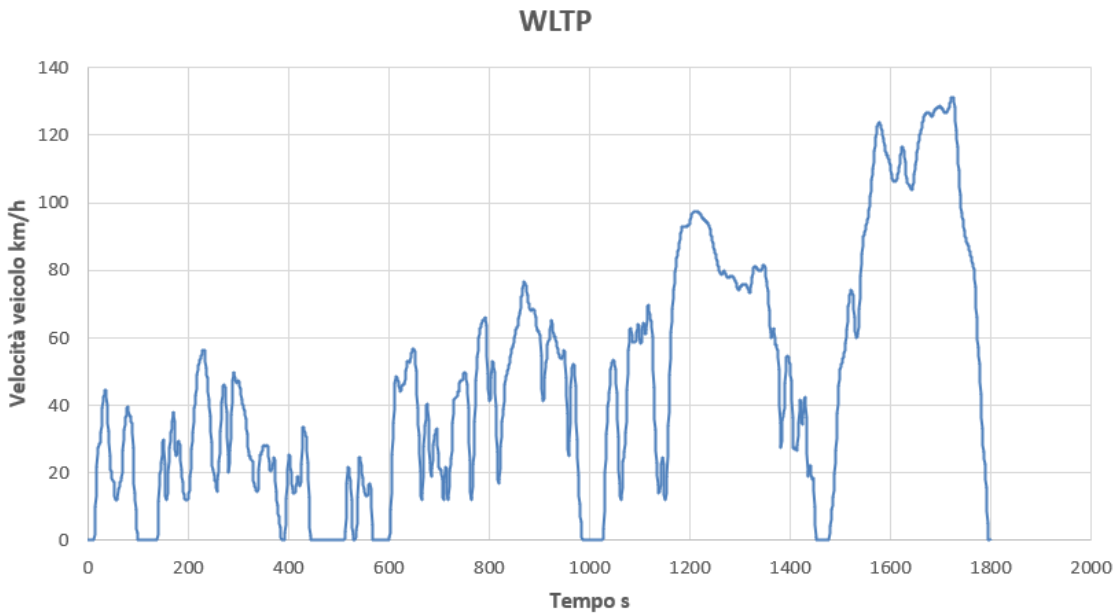

Figura 3.3 : ciclo WLTP

Di seguito vengono riportati i principali parametri del ciclo:

Fase	Durata	Durata stop	Distanza	Durata stop in %	Velocità max	Vel. media w/o stop	Vel. media w/ stop	Acc. min	Acc. max
	s	s	m		Km/s	Km/s	Km/s	m/s^2	m/s^2
Low	586	156	3095	26.5%	56.5	25.7	18.9	-1.47	1.47
Middle	433	48	4756	11.1 %	76.6	44.5	39.5	-1.49	1.57
High	455	31	7158	6.8%	97.4	60.8	56.6	-1.49	1.58
Extra-high	323	7	8254	2.2%	131.3	94.0	92.0	-1.21	1.03
Totale	1800	242	23262						

Tabella 3.5: Caratteristiche ciclo WLTP

Successivamente l'agente DQN è stato testato anche su dei cicli guida reali denominati clust4, clust7, clust11, clust12, al fine di verificarne l'adattabilità al variare delle condizioni di guida. Questi sono dei cicli più brevi rispetto al WLTP e sono caratterizzati da velocità massime inferiori. I cicli 4 e 7 sono più lunghi rispetto all'11 e al 12, i quali però sono più complicati da gestire in quanti presentano delle accelerazioni più spiccate.

Di seguito vengono riportate le traiettorie di velocità in funzione del tempo dei cicli clust:

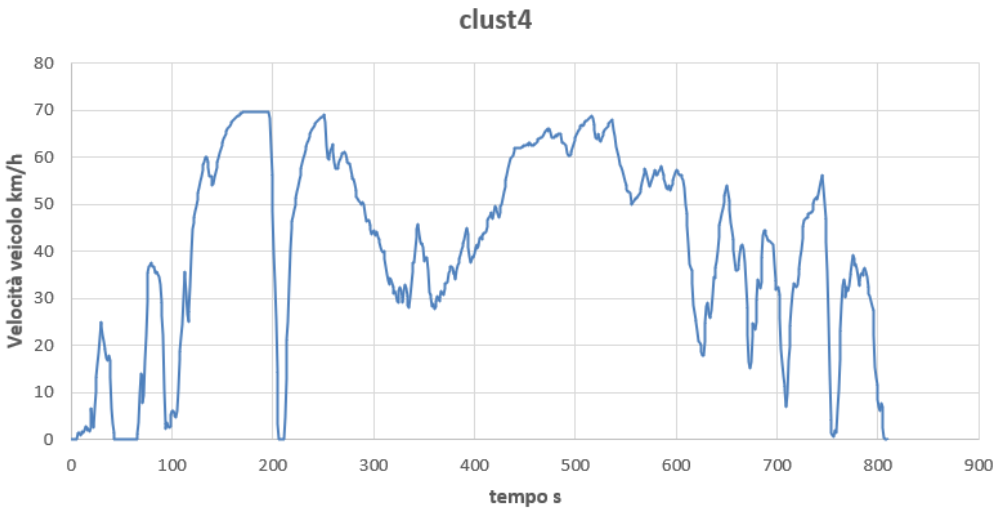

Figura 3.4: Ciclo clust4

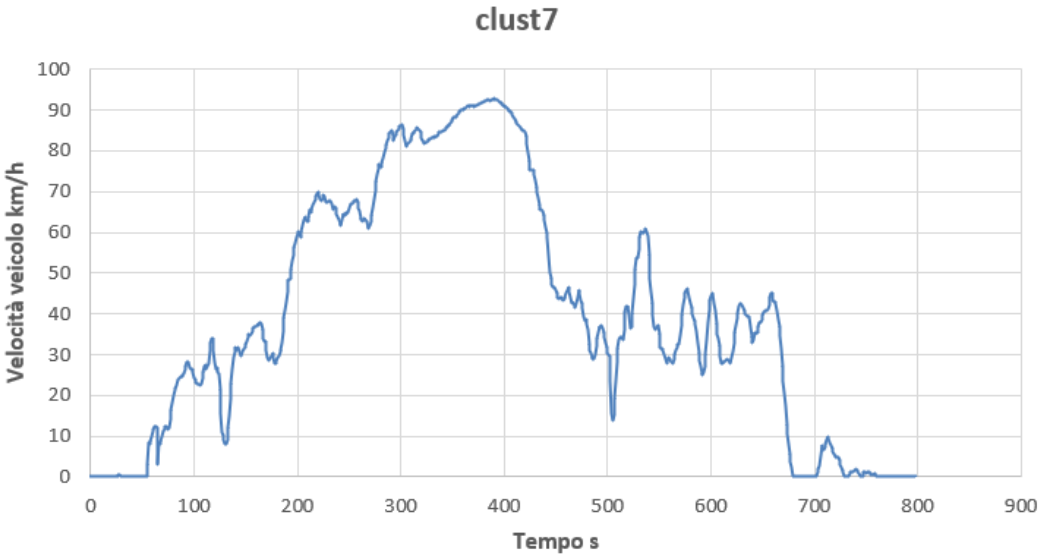

Figura 3.5: Ciclo clust7

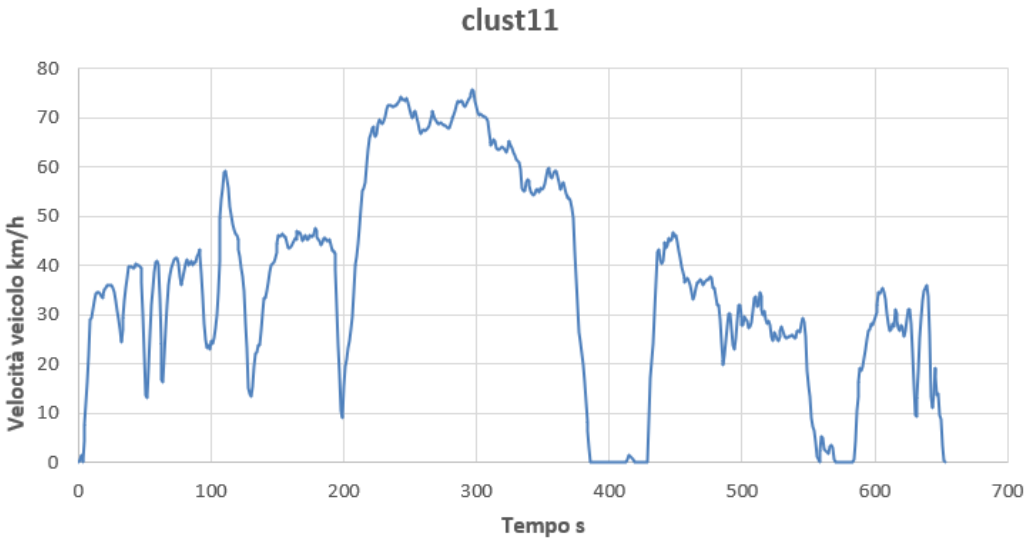

Figura 3.6: Ciclo clust11

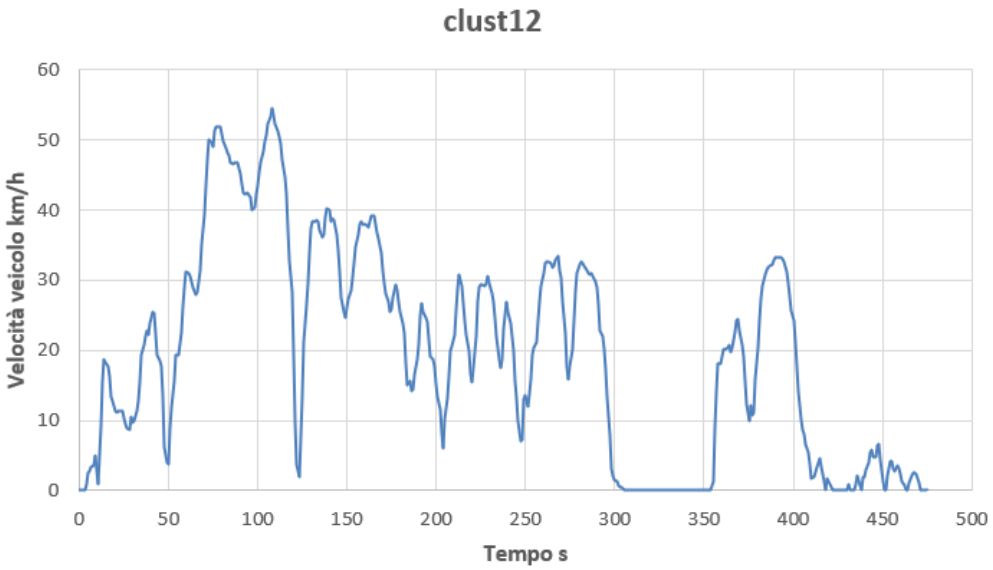

Figura 3.7: Ciclo clust12

4. Documentazione Software

4.1 Architettura Software

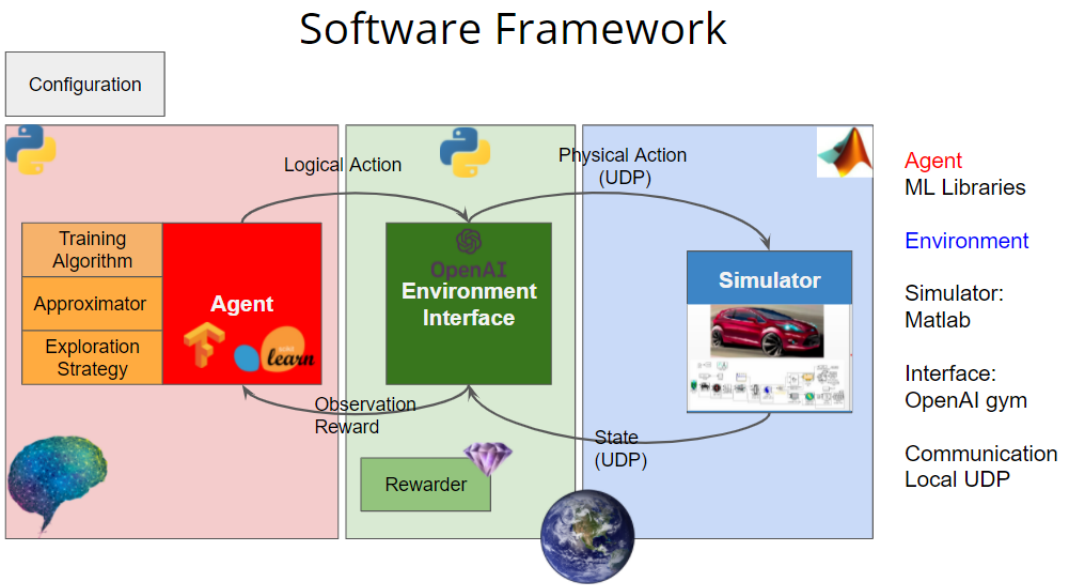

Figura 4.1: struttura del software

Lo strumento utilizzato in questo lavoro di tesi si compone di un *Simulator*, il quale è implementato su Matlab e rappresenta il modello del veicolo descritto nel capitolo precedente, ossia può essere visto come la parte fisica dell'*environment*. Il *Simulator* a sua volta comunica con l'interfaccia dell'*environment* (*Environment Interface*) e con l'*Agent*, i quali sono invece implementati su Python. Ad ogni step fisico compiuto dal *Simulator* corrisponde uno step logico, ossia lo step in cui è coinvolto l'agente e quindi la componente decisionale dell'apprendimento del RL. Lo step fisico si completa quando la simulazione avanza di un secondo e viene prodotto uno stato fisico, questo a sua volta viene inviato all' *Environment Interface* (*environment* logico) mediante protocollo UDP. L'*Environment Interface* è la parte di interfaccia e si occupa di trasformare tutte le componenti di comunicazione tra l'agente e la simulazione fisica, in particolare in questa fase ho la trasformazione da stato fisico a stato logico e quest'ultimo a sua volta viene esposto mediante un'osservazione, cioè la parte dello stato logico che l'agente può effettivamente vedere. Inoltre in funzione dello stato logico viene calcolato il *reward*, che è la ricompensa all'istante logico che l'agente riceve. L'agente a sua volta ha un'interfaccia logica in cui però si possono distinguere tre sezioni principali: l'algoritmo di *training* (*Training Algorithm*), uno o più approssimatori (*Approximator*) come ad esempio una rete neurale artificiale, ed infine la strategia di esplorazione (*Exploration strategy*), che è la componente che gestisce la *policy*. Dopo essere passati attraverso la strategia di esplorazione si ottiene un'azione logica, che è un valore discreto, e a sua volta viene tradotta in un'azione fisica sul simulatore, come l'inserimento di una marcia oppure la determinazione di un *powerflow*. Infine, tramite la parte di configurazione (*Configuration*) è possibile andare a configurare tutte le varie componenti che intervengono nell'addestramento.

4.2 Descrizione dell'algoritmo di DQN

L'agente viene addestrato su più passaggi temporali in molti episodi ed esegue una sequenza di operazioni ad ogni *time step* della simulazione. Il ciclo di addestramento di un agente DQN è suddiviso in episodi di *training* e di *testing*. Gli episodi di test vengono eseguiti ad intervalli predefiniti per monitorare l'andamento dell'apprendimento in condizioni di assenza di componente di esplorazione, quindi per verificare se la *policy* appresa senza esplorazione sia sufficiente per risolvere il problema che si sta affrontando.
Un parametro importante della NN è il *leraning starts* che determina la parte iniziale dell'addestramento, cioè il numero di step prima che rete inizi ad aggiornarsi. Il *learning starts* va a dialogare con un *minibatch* caratterizzato da una determinata taglia (*batch size*), quindi si inizia ad addestrare la rete Q nel momento in cui si può approssimativamente assumere che i campioni all'interno del *minibatch* siano indipendenti fra loro. La taglia del *replay memory buffer*, che contiene un numero limitato di transizioni $(s_t, a_t, r_t, s_{t+1}, done)$, va configurata in modo tale che l'agente dimentichi le informazioni che non sono più utili. Un altro parametro rilevante per le performance dell'agente è il *target update frequence*, ossia la frequenza con cui i pesi della rete Q vengono copiati all'interno della rete Q target. Tale parametro deve essere calibrato in modo tale da garantire una buona stabilità del *training* dell'agente.

Di seguito si riporta uno schema descrittivo dell'algoritmo di *Deep Q Network* :

Inizializzo Replay memory buffer D con capacità N
Inizializzo minibatch con capacità n
Inizializzo M episodi di training
Inizializzo leraning starts
Inizializzo target update frequence = C
Stabilisco missione di guida lunga T
Inizializzo rete Q con pesi random θ
Inizializzo rete target Q con pesi θ'= θ
for episodio = 1: M
  for t = 1: T
    seleziona con probabilità ε un azione random a_t
    altrimenti seleziona $a_t = arg \max_{a' \in A} Q(s_t, a')$
    Osserva il reward r_t e il nuovo stato s_{t+1}
    Salva la transizione $(s_t, a_t, r_t, s_{t+1}, done_t)$ nel buffer D
    if t > learning starts && t > n
      preleva n transizioni $(s_j, a_j, r_j, s_{j+1}, done_j)$ dal minibatch in maniera randomica
      for j = 1: n
        imposto $Q_{target,j} = r_j$ se $done_j$ = 1, altrimenti $Q_{target,j} = r_j + \gamma \max_{a'} Q(s_{j+1}, a'; \theta')$
        addestro la rete Q minimizzando $L_j = \left(Q_{target,j} - Q(s_j, a_j; \theta)\right)^2$
      end for
    end if
  Ogni C time steps impongo $\theta = \theta'$, quindi copio i pesi della rete Q nella rete target
  end for
end for

4.2.1 Definizione delle variabili di controllo

Sulla base di un determinato stato l'agente può intraprendere due differenti tipologie di azioni (variabili di controllo), in particolare può intervenire sul *powerflow* (PF) α oppure sulla marcia da inserire, caratterizzata dal proprio rapporto di trasmissione. Nello strumento utilizzato entrambe le variabili sono considerate discrete, nonostante α potrebbe variare nel continuo tra -1 e 1. Nel dettaglio in questo lavoro di tesi sono state considerate 6 marce N_{GB} e 7 possibili *powerflow* N_{PF}, ossia 7 possibili distribuzioni di potenza tra i due motori. Perciò complessivamente il numero delle azioni N_{AZ} (livelli delle variabili di controllo) che possono essere intraprese dall'agente è dato da tutte le possibili combinazioni tra PF e marcia innestata :

$$N_{AZ} = N_{GB} \cdot N_{PF}$$

Precisamente sono stati considerati i seguenti possibili valori di α:

- $\alpha = 1$: puro elettrico
- $\alpha = 0$: puro termico
- $\alpha = 0.25, \alpha = 0.5, \alpha = 0.75$: modalità *power-split*
- $\alpha = -0.5, \alpha = -1$: modalità *battery-charging*

Inoltre all'interno del codice sono state inserite due variabili booleane, denominate *feas* e *new feas*, le quali forniscono delle informazioni sullo spazio delle azioni, quindi le azioni che sono disponibili all'agente dopo aver osservato le osservazioni *obs* e *new obs*. Tramite la variabile di *feasibility*, ossia di fattibilità, permettiamo quindi all'agente di eseguire un sottoinsieme di tutte le azioni disponibili, in funzione di quella che sappiamo essere la conseguenza delle stesse azioni. Il tool quindi verifica che l'azione presa sia fisicamente possibile, tenendo conto della velocità massima raggiungibile dai vari componenti, della potenza massima erogata o assorbita da ICE e EM e della massima potenza che la batteria è in grado di gestire, sia in fase di alimentazione che in fase di scarica, sulla base del suo dimensionamento e del suo valore massimo di *C-rate*.

4.2.2 Definizione degli Stati

Come analizzato precedentemente l'algoritmo di DQN permette di operare in spazi continui, superando i limiti del QL, il quale è caratterizzato da un *observation space* discreto.
Di seguito vengono riportati i possibili stati selezionabili dall'utente:

- SOC
- SOC_high
- SOC_low
- feas
- FC
- roadVel
- time
- roadPower
- next_roadVel
- next_deltaVel
- next_roadPower

La selezione degli stati è di cruciale importanza, in quanto questi devono fornire all'agente le informazioni necessarie per prendere l'azione corretta, ossia quella che mi massimizza il *Discounted return*. Solitamente l'aggiunta di più stati, se correttamente selezionati, mi permette di caratterizzare meglio l'*environment* con conseguente miglioramento delle performance.

4.2.3 Definizione funzione di *reward*

La definizione di un'adeguata funzione di *reward* rappresenta un punto cruciale per il perseguimento dell'obbiettivo prefissato, ossia il controllo simultaneo del FC e del SOC. Nel dettaglio si cerca di minimizzare il consumo di carburante durante la missione di guida selezionata, controllando al contempo lo stato di carica della batteria all'interno del range [0.55, 0.65]. La funzione di *reward* utilizzata prevede una conversione dell'energia erogata dalla batteria in consumo di combustibile equivalente FC_{eq} , in modo tale da poter essere confrontato con il consumo di carburante FC. Di seguito viene riportata la formulazione del *reward* puntuale, nel caso in cui *done* = 0

$$r(s,a) = \begin{cases} b + c \cdot FC & se\ \Delta SOC > 0 \\ b + c \cdot FC + d \cdot FC_{eq} & se\ \Delta SOC < 0 \end{cases}$$

Dove:

- *b,c,d* sono tre coefficienti scelti dall'utente. In particolare *b* rappresenta il valore della ricompensa massima ricevuta dall'agente, mentre *c* e *d* sono rispettivamente i pesi del FC e del SOC puntuali, ottenuti dopo aver intrapreso l'azione *a* a partire dallo stato s.
- $\Delta SOC = SOC - SOC_{rif}$, dove $SOC_{rif} = 0.6$
- $FC_{eq} = \frac{-\Delta SOC \cdot E_{batt}}{H_i}$, dove $E_{batt} = 21913\ kj$ e $H_i = 43400\ kj/kg$

Nel caso in cui *done* = 1, ossia viene selezione un'azione *unfisible* oppure lo stato di carica fuoriesce dal range prefissato, l'agente riceve un *reward* molto negativo:

$$r(s,a) = \begin{cases} -100 & se\ azione\ unfisible \\ -100 & se\ SOC > SOC_{high} \\ b + c \cdot FC + d \cdot FC_{eq} & se\ SOC < SOC_{low} \end{cases}$$

Dove:

- $SOC_{high} = 0.65$
- $SOC_{low} = 0.55$

La ricompensa massima *b* è stata posta pari a 10, mentre i coefficienti *c* e *d* sono stati calibrati in modo tale da premiare maggiormente la riduzione di FC.

4.2.4 Definizione strategia di esplorazione

Per bilanciare correttamente il rapporto tra *exploration* e *exploitation* è stata utilizzata una $\varepsilon - greedy$ policy per la determinazione dell'azione. Il parametro ε è collegato direttamente all'esplorazione, ossia più elevato è il suo valore maggiore sarà l'esplorazione e viceversa. Nel training dell'agente è stato utilizzato un ε che decresce linearmente all'aumentare del numero di episodi, in modo tale da esplorare l'environment nelle fasi iniziali dell'apprendimento e successivamente sfruttare le informazioni ricavate dall'esplorazione, selezionando l'azione che mi massimizza l'*action value function*. Prima del lancio della simulazione l'utente stabilisce ε_{start}, ossia il valore di partenza, ε_{finish}, cioè il valore minimo ed infine $\varepsilon_{decay\ episode}$, che indica la frazione di cui ε decresce all'avanzare di ogni episodio. Tali parametri devono essere opportunatamente calibrati in modo tale da permettere all'agente di individuare la *policy* ottimale, ossia quella che mi massimizza il *Discounted return*. Negli episodi di *testing* invece vi è un'assenza di componente di esplorazione, quindi tutte le decisioni prese dall'agente in quel determinato episodio di *testing* porteranno alla massimizzazione della *Q function*, sulla base di quanto esplorato fino ad allora, con l'obbiettivo di monitore l'addestramento dell'agente.

5. Esperimenti su ciclo WLTP

5.1 Configurazioni di partenza

All'interno di questo capitolo vengono ripercorsi i passaggi salienti che hanno portato ad una soluzione sub-ottimale del problema. Le prestazioni di un algoritmo di RL dipendono fortemente dalla scelta di tutti i suoi parametri di allenamento, perciò è stato condotto uno studio approfondito sull'influenza di ciascun fattore sulle performance dell'agente. Come analizzato precedentemente l'obbiettivo è quello di ottimizzare la gestione energetica tra ICE e EM, controllando simultaneamente sia il FC che il SOC grazie all'utilizzo di un algoritmo di DQN. In particolare sono stati valutati gli effetti dei principali iperparametri della rete, sono stati definiti gli stati del sistema, vi è stato un adeguato studio della funzione di *reward* e della strategia di esplorazione con l'obbiettivo di ottenere un *reward FC-oriented* e un corretto rapporto tra esplorazione e sfruttamento. Come studio iniziale l'agente DQN è stato allenato sul ciclo WLTP su 1000 episodi. I parametri *learning rate*, *target update frequence* e *replay memory size* in prima istanza sono stati impostati in modo tale da essere proporzionali alla lunghezza del ciclo guida:

$$learning\ starts = 20 \cdot lunghezza\ ciclo\ guida\ (s)$$

$$target\ update\ frequence = 3 \cdot learning\ starts$$

$$replay\ memory\ size = 50 \cdot lunghezza\ ciclo\ guida\ (s)$$

E' stata utilizzata una rete neurale con due strati intermedi (*hidden layers*), ciascuno dei quali costituito da 64 neuroni, e come funzione di attivazione è stata scelta la funzione ReLU. Tale funzione di attivazione è la più utilizzata in letteratura, in quanto garantisce computazioni efficienti e semplici e migliora la capacità di evitare la saturazione dei pesi in allenamento. Il *discounted factor* è stato posto pari a 0.99, in modo tale da dare importanza alla ricompensa immediata senza trascurare quelle future, il parametro di esplorazione ε decresce linearmente a partire da 0.8 e giunge al suo valore finale di 0.05 in corrispondenza del 150 esimo episodio, lasciando comunque una piccola componente di esplorazione per poter migliorare la *policy*. Infine in questa prima fase sono stati scelti come stati del sistema il SOC e il FC, ed è stato adottato un reward che premia maggiormente la riduzione del ΔSOC. A parità di pesi c e d un incremento del ΔSOC impatta maggiormente sulla diminuzione della ricompensa ottenuta, in quanto i valori del SOC sono più elevati di quelli del consumo istantaneo di carburante.

Nella Tabella 5.1 vengono riportate le configurazioni iniziali per lo studio del problema:

Agente DQN	*Funzione di attivazione*	ReLU
	Taglia del minibatch	32
	Discounted factor γ	0.99
	Numero di neuroni	64
	Numero di layer	2
	Learning starts	36000
	Target update frequence	108000
	Replay memory size	720000
	Learning rate lr	0.001
Strategia di esplorazione	ε_{start}	0.8
	ε_{finish}	0.05
	$\varepsilon_{decay\ episode}$	0.005
Reward	*b*	10
	c	-1000
	d	-1000
Stati	*SOC*	
	FC	
Training	*Ciclo guida*	wltp
	Numero episodi	1000

Tabella 5.1 Configurazioni iniziali

A partire da queste configurazioni si è agito progressivamente su quasi tutti i parametri per migliorare le performance dell'agente. I plot di ciascuna simulazione presentati nei paragrafi successivi hanno l'intento di mostrare l'effetto della modifica di un determinato parametro, sulla base delle configurazioni ed i risultati della prova ad essa precedente.

Di seguito vengono riportati i risultati ottenuti con queste configurazioni iniziali:

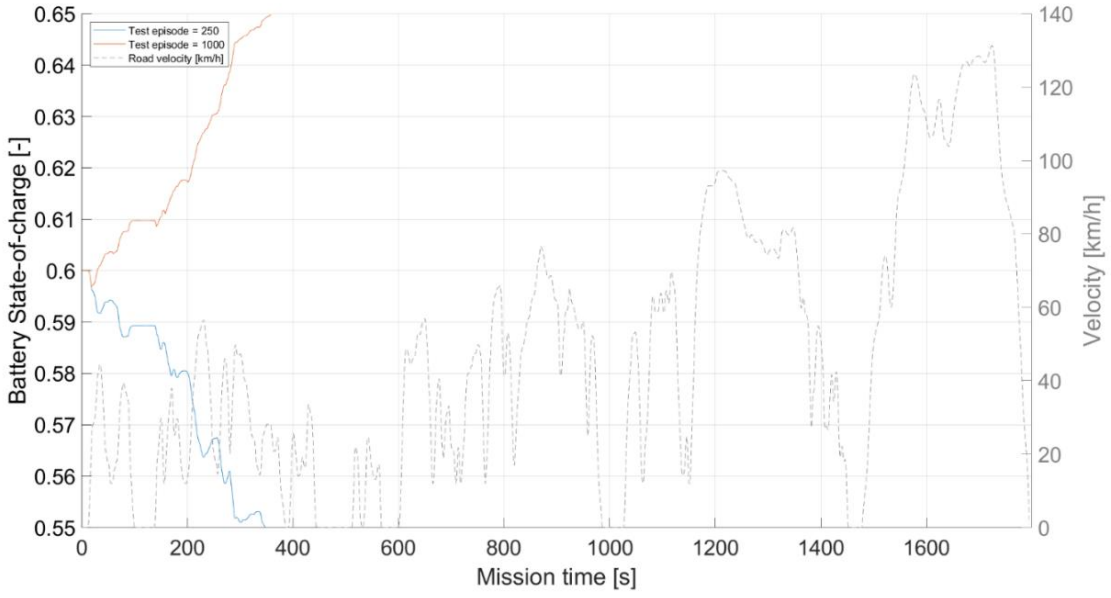

Figura 5.1: SOC episodi di test-Simulazione 1

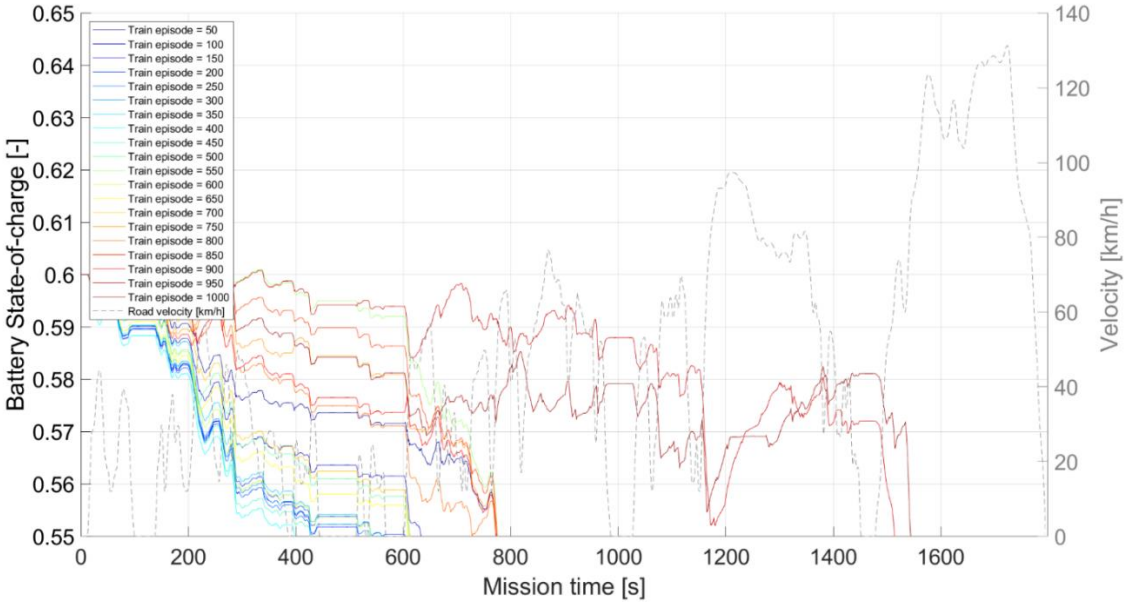

Figura 5.2: SOC episodi di train-Simulazione 1

Analizzando il SOC trend negli episodi di testing e di training è evidente come l'agente non sia riuscito ad individuare delle corrette traiettorie di controllo entro i 1000 episodi di allenamento, in quanto le azioni intraprese hanno portato lo stato di carica della batteria oltre i limiti prefissati, per cui si ha la terminazione anzitempo di ogni singolo episodio.

5.1.1 Effetto del *learning starts* e del *target update frequence*

Al fine di velocizzare l'apprendimento si è agito inizialmente sul *learning starts* e sul *target update frequence,* in modo tale da ottenere un approccio di tipo *charge-sustaining* del SOC entro i 1000 episodi. Abbassando il *learning starts* di fatto iniziamo ad addestrare prima la rete neurale, cercando sempre di accumulare un quantitativo sufficiente di esperienze e che siano indipendenti fra di loro. In questo caso, come analizzato precedentemente, ci viene in aiuto l'*Experience replay* dal quale vengono estratte in maniera totalmente randomica 32 tuple per ogni time step, permettendoci di ridurre le correlazioni tra i campioni di addestramento e quindi l'apprendimento viene stabilizzato. Anche la *target update frequence* è stata ridotta di circa due ordini di grandezza, nonostante un aggiornamento meno frequente della rete target solitamente comporti un processo di apprendimento più stabile, ma risulta importante anche conciliare l'apprendimento di un agente con dei tempi di calcolo non troppo eccessivi.
Di seguito vengono riportate le modifiche che sono state apportate alle configurazioni di Tabella 5.1 ed i risultati ottenuti in termini di traiettorie di *SOC*, *FC* e *train loss*:

- $learning\ starts: 2000$
- $target\ update\ frequence: 6000$

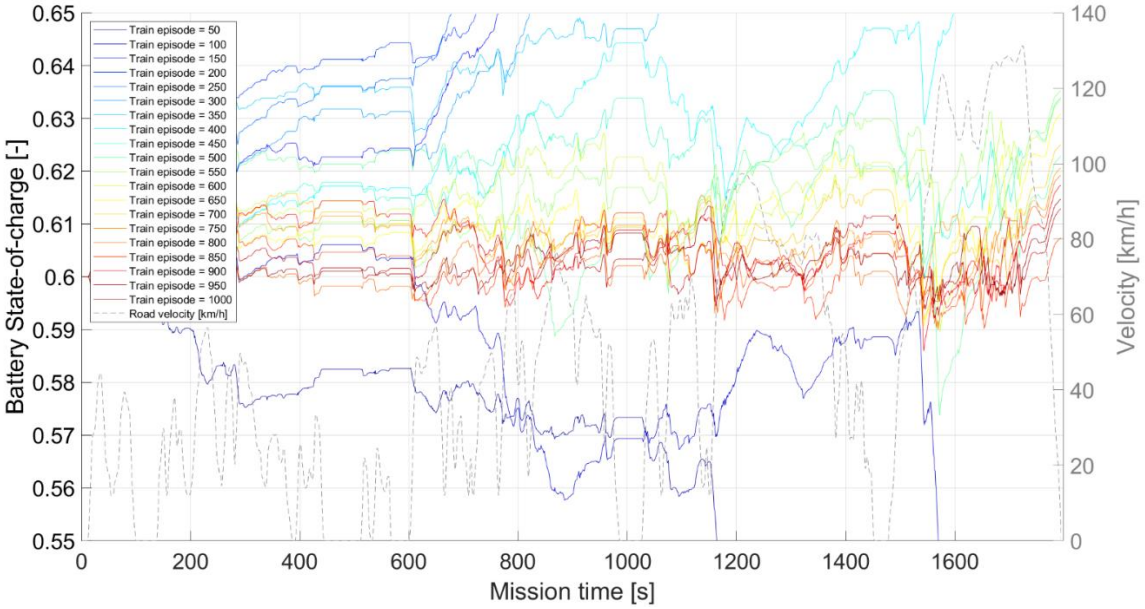

Figura 5.3: SOC episodi di train-Simulazione 2

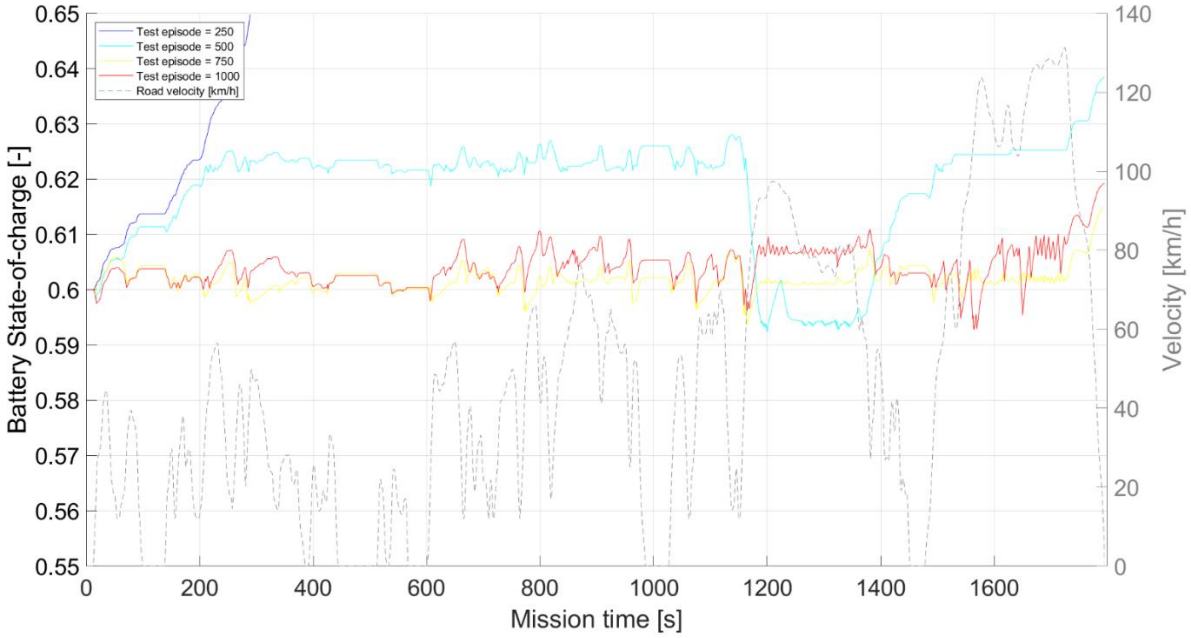

Figura 5.4: SOC episodi di test-Simulazione 2

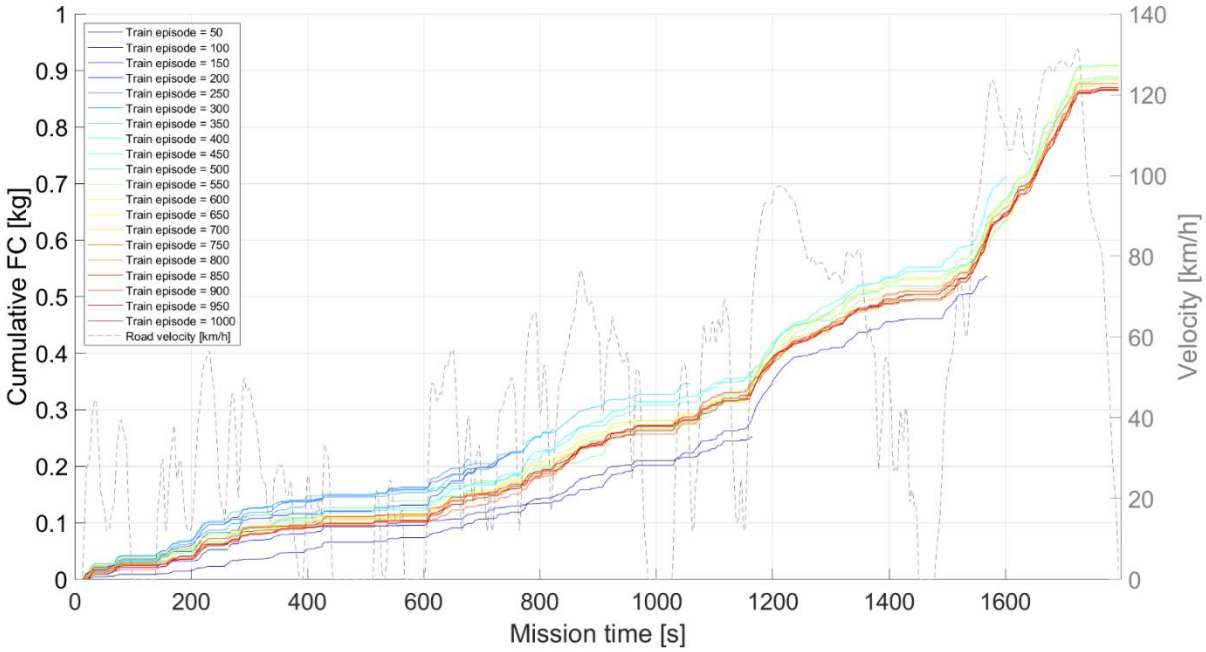

Figura 5.5 Cumulative FC episodi di train-Simulazione 2

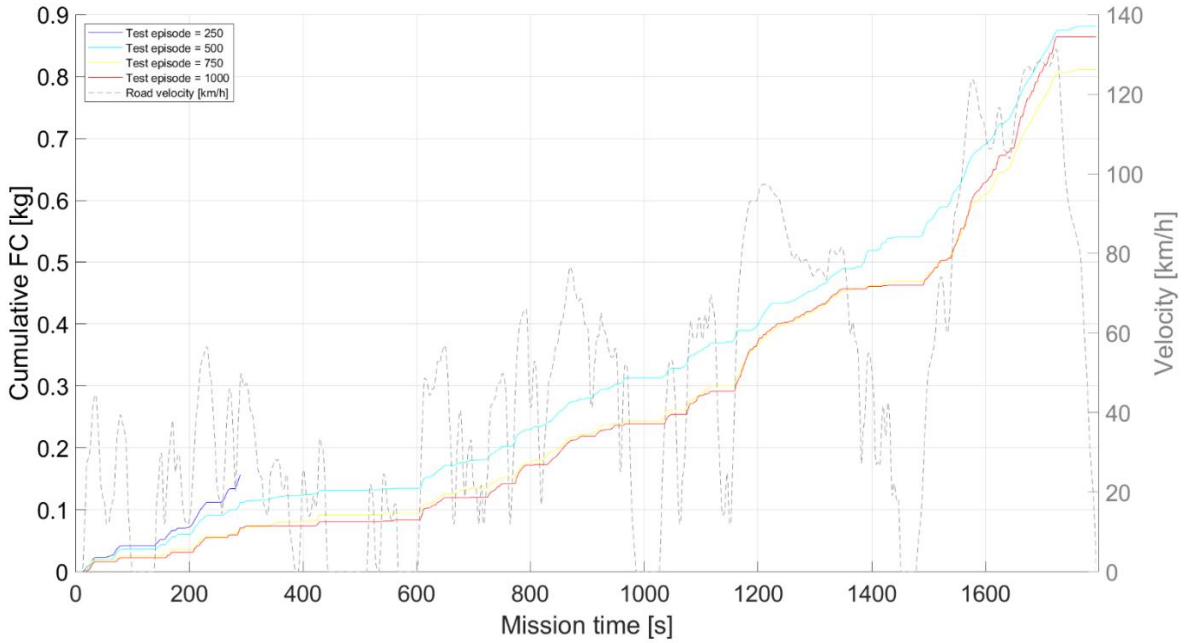

Figura 5.6: Cumulative FC episodi di test-Simulazione 2

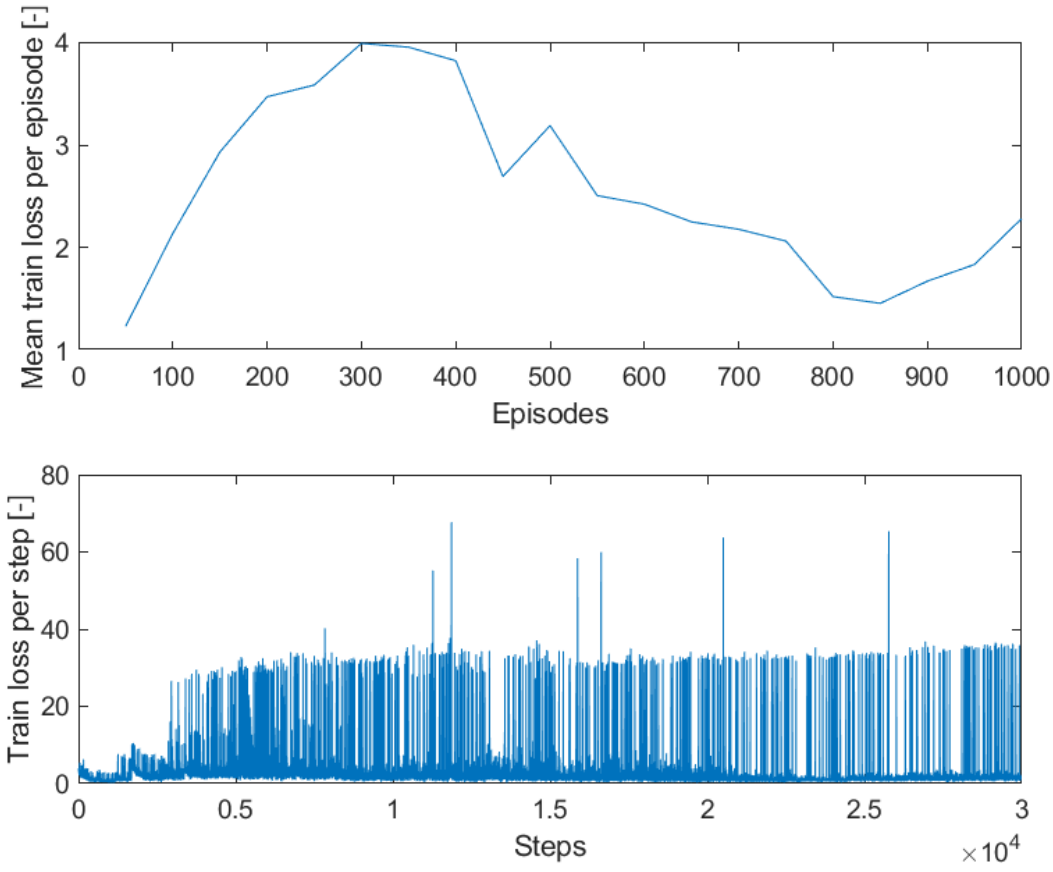

Figura 5.7 Train loss-Simulazione 2

Le modifiche apportate velocizzano l'addestramento pur mantenendo una buona stabilità dell'agente durante gli episodi di *training*, permettendo di raggiungere i risultati fisici desiderati. La *train loss* mostra un andamento prima crescente e poi decrescente, segno del fatto che l'agente stia riuscendo a prevedere i corretti valori di Q, minimizzando l'errore L_t.

5.1.2 Introduzione di un *reward FC-oriented*

Successivamente si è agito sulla funzione di *reward*, in particolare sui coefficienti c e d, in modo tale da premiare maggiormente la riduzione di FC.
Nelle simulazioni precedenti era stato utilizzato $c = -1000$ (riferito al FC) e $d = -1000$ (riferito al SOC), in questo caso invece si è andati progressivamente ad aumentare il coefficiente c riducendo in contemporanea d, con l'obbiettivo di privilegiare la riduzione di combustibile durante la missione guida, lavorando comunque in modalità *charge-sustaining.*
Inoltre, sono stati aggiunti più Stati in input alla rete con lo scopo di fornire maggiori informazioni sull'*environnement* e di conseguenza migliorare le prestazioni dell'agente. L'aggiunta di più stati rende più complesso l'*environment* da esplorare perciò si è deciso di aumentare l'esplorazione, in modo tale da ottenere delle stime più accurate delle *action value functions*. In particolare il parametro di esplorazione decresce da un valore iniziale di 0.8 fino a raggiungere il suo valore minimo in corrispondenza del 75% della missione di guida.

Di seguito vengono mostrate le modifiche apportare rispetto alla simulazione del paragrafo precedente ed i plot di *SOC*, *FC*, *train loss* e *Q-values* per gli episodi di *testing* e di *training*:

- $Stati: SOC, roadVel\ e\ next\_roadPower$
- $Reward: b = -3000, d = -200$
- Strategia di esplorazione: $\varepsilon_{start} = 0.8,\ \varepsilon_{finish} = 0.05, \varepsilon_{decay\ episode} = 0.001$

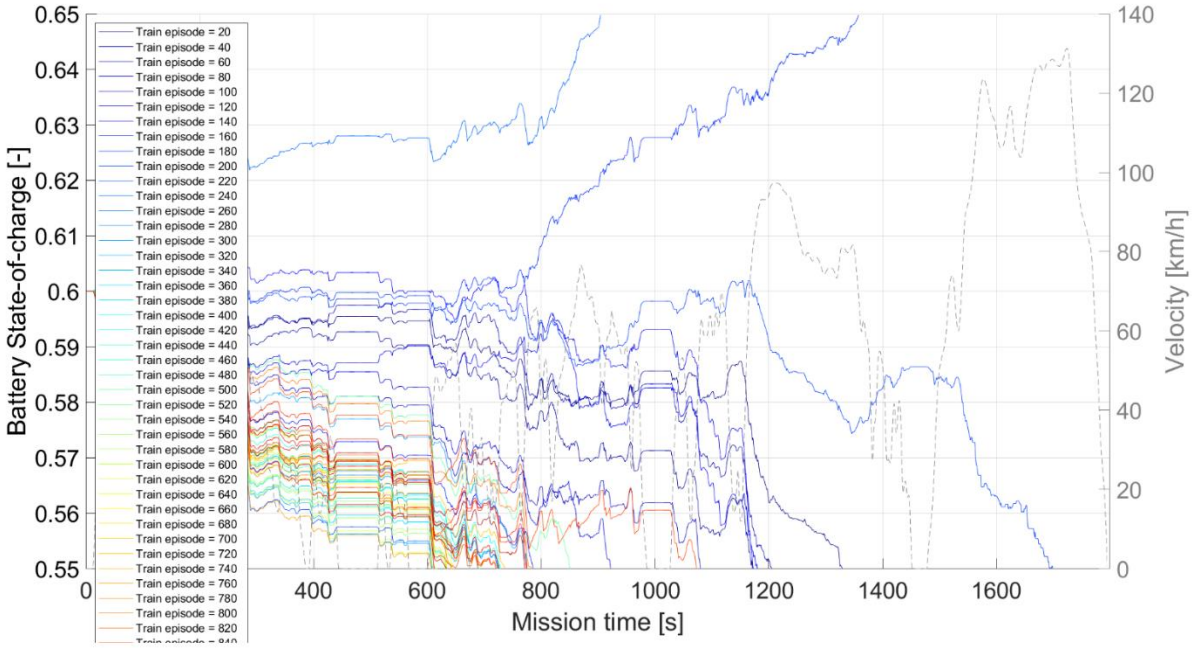

Figura 5.8: SOC episodi di train-Simulazione 3

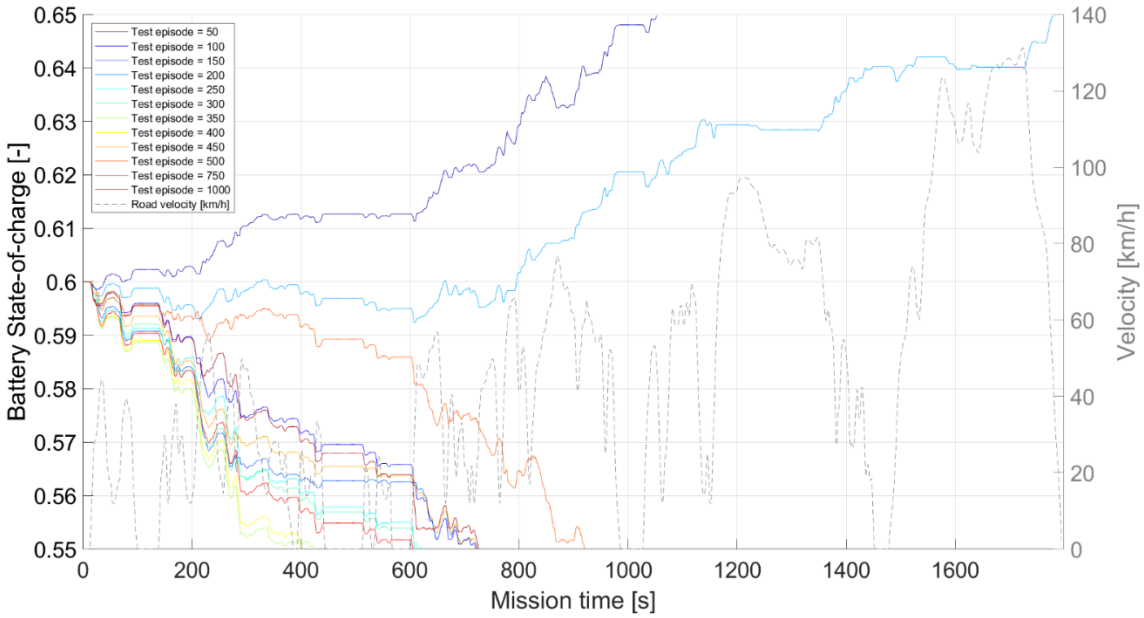

Figura 5.9: SOC episodi di test-Simulazione 3

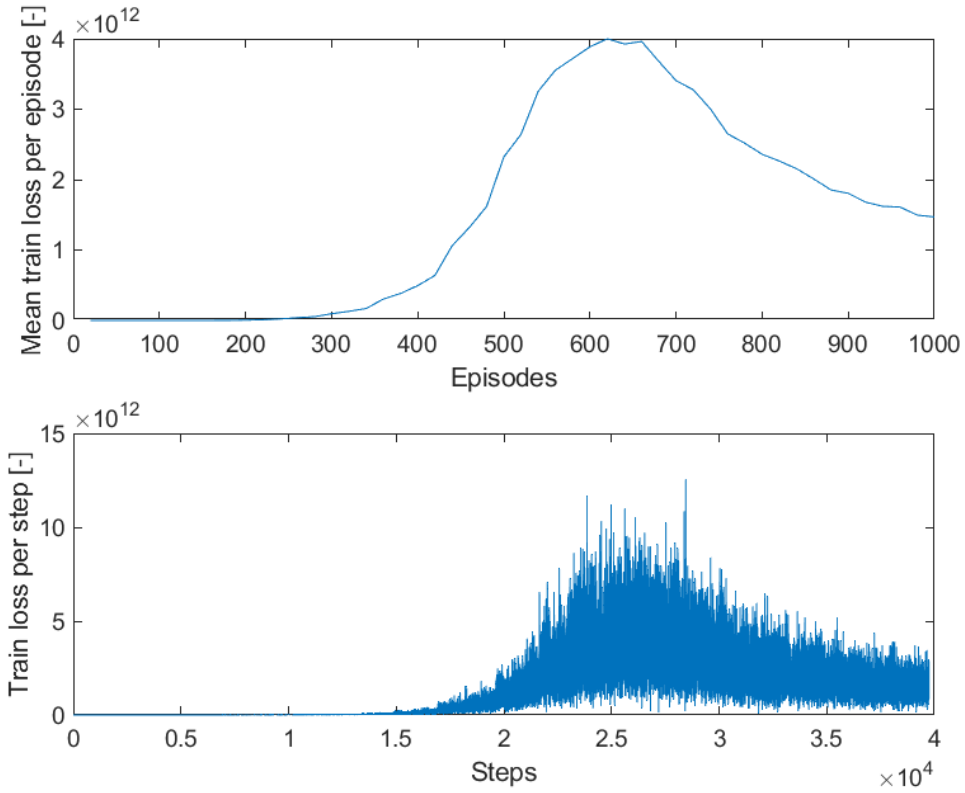

Figura 5.10: Train loss-Simulazione 3

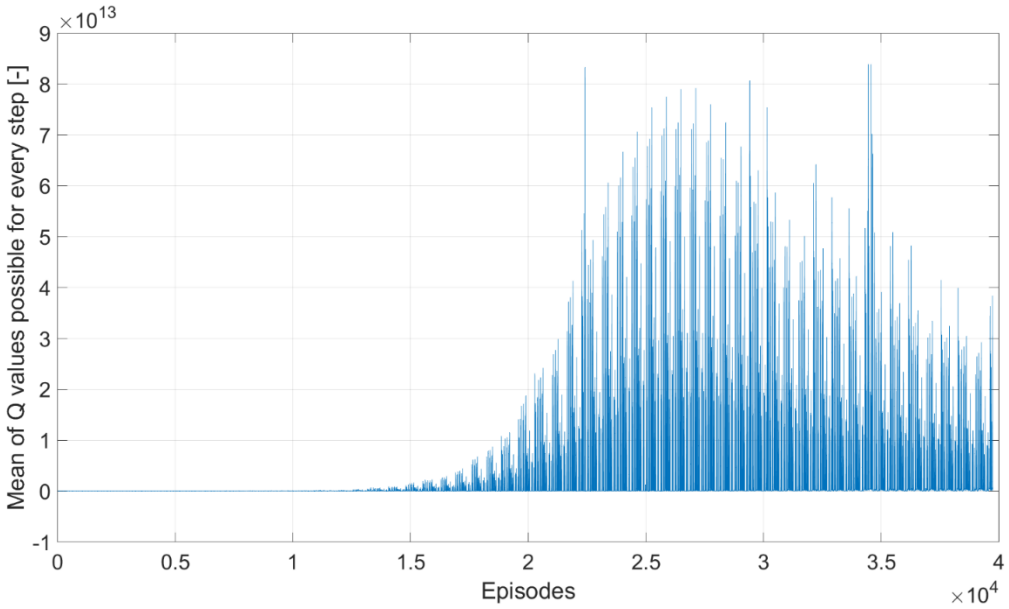

Figura 5.11: Q-values-Simulazione 3

L'introduzione di più Stati in concomitanza con la definizione di un *reward* più complesso da gestire porta alla divergenza dei *Q-values*, che salgono oltre il loro massimo valore teorico. Tale fenomeno può essere causato dall'esplosione del gradiente durante il processo di *backpropagation*, perciò il modello risulta essere instabile e incapace di un apprendimento efficace. La derivata dell'errore viene propagata a ritroso per aggiornare i pesi. I pesi a loro volta vengono aggiornati utilizzando una frazione

dell'errore retro propagato, controllato dalla velocità di apprendimento. E' possibile che gli aggiornamenti dei pesi siano così grandi che i pesi stessi superano o riducono la loro precisione numerica. In pratica i pesi possono assumere dei valori tendenti all'infinito, rendendo la rete inutilizzabile.

5.1.3 Normalizzazione degli stati

Poiché gli stati in input alla rete neurale presentano delle scale molto diverse fra loro si è pensato di eseguire una normalizzazione degli stati stessi. Sulla base delle caratteristiche del ciclo guida sono stati definiti i valori massimi e minimi di ciascuno stato, in modo tale che i loro valori siano compresi tra 0 e 1. In particolare questa tecnica è utilizzata per migliorare la velocità, le performance e la stabilità di una rete neurale.

Di seguito vengono riportati i risultati ottenuti adottando la normalizzazione:

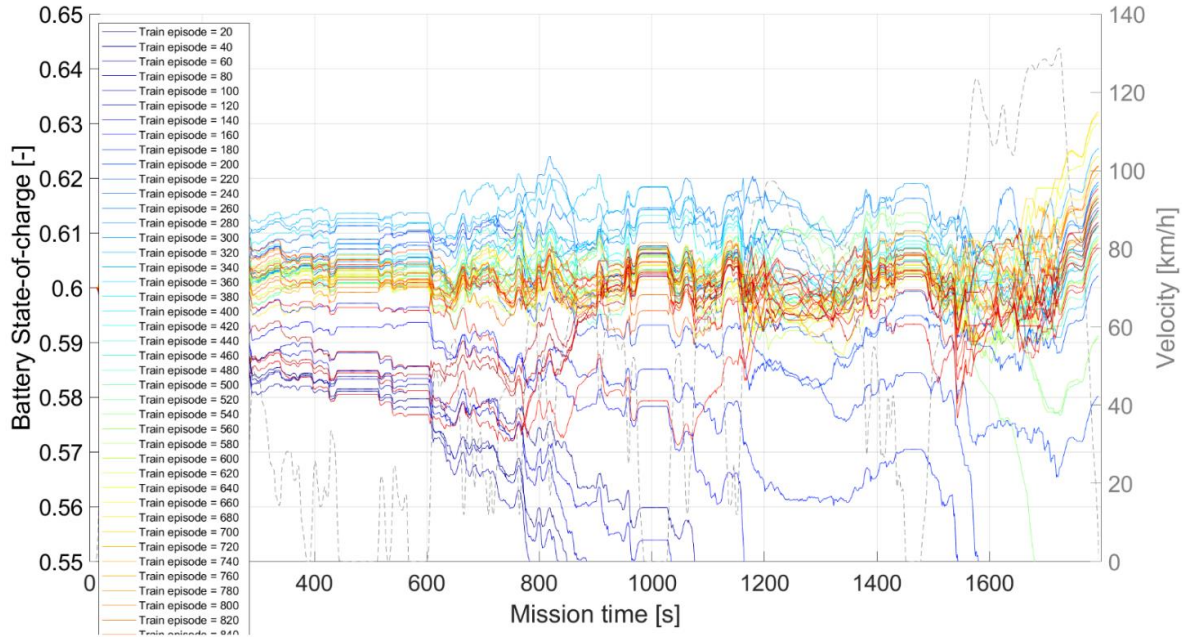

Figura 5.12: SOC episodi di train-Simulazione 4

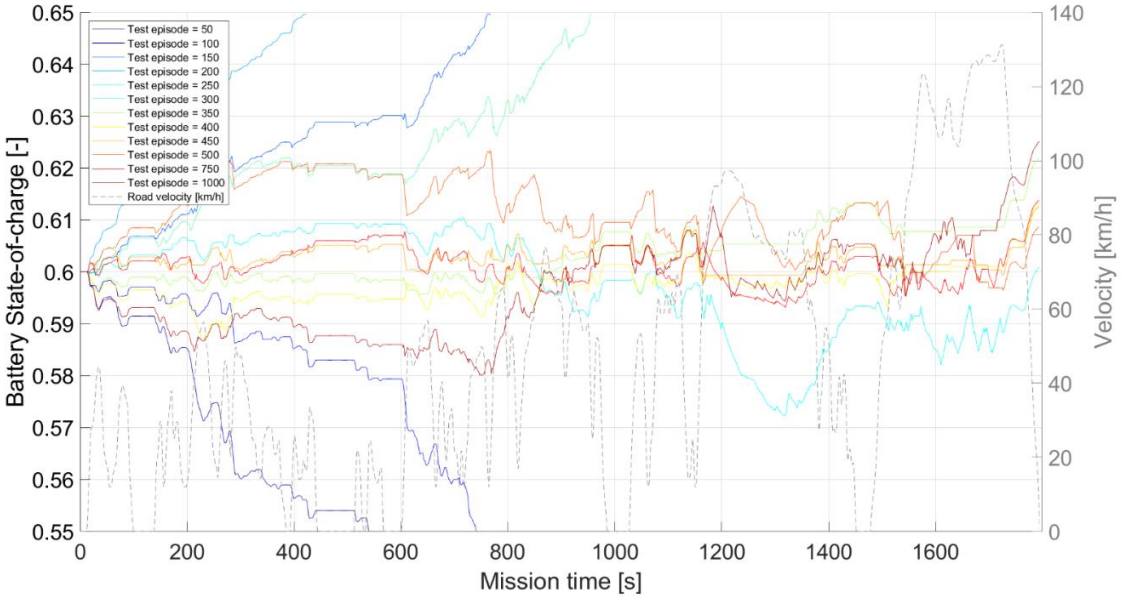

Figura 5.13: SOC episodi di test-Simulazione 4

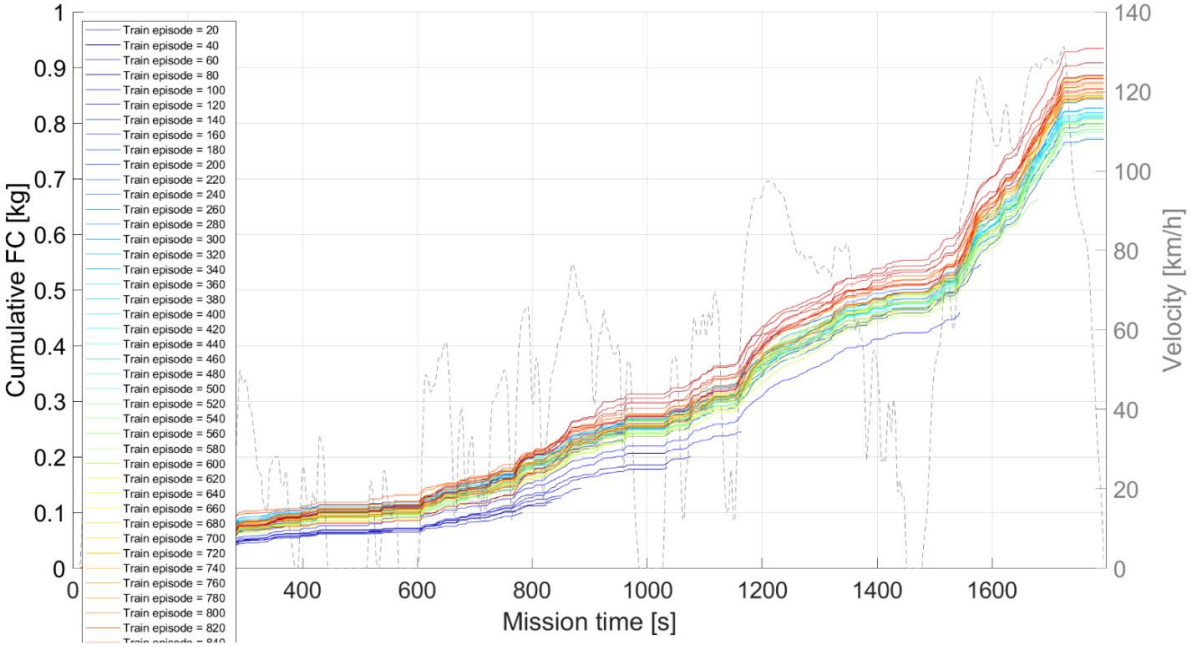

Figura 5.14: Cumulative FC episodi di train-Simulazione 4

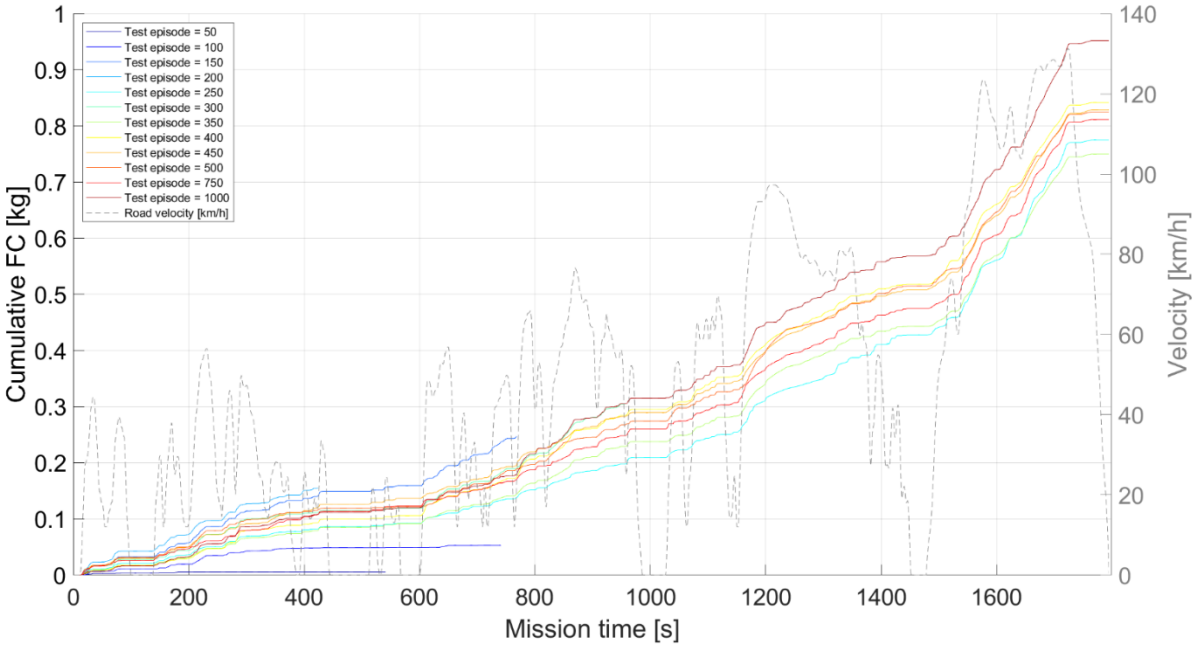

Figura 5.15: Cumulative FC episodi di test-Simulazione 4

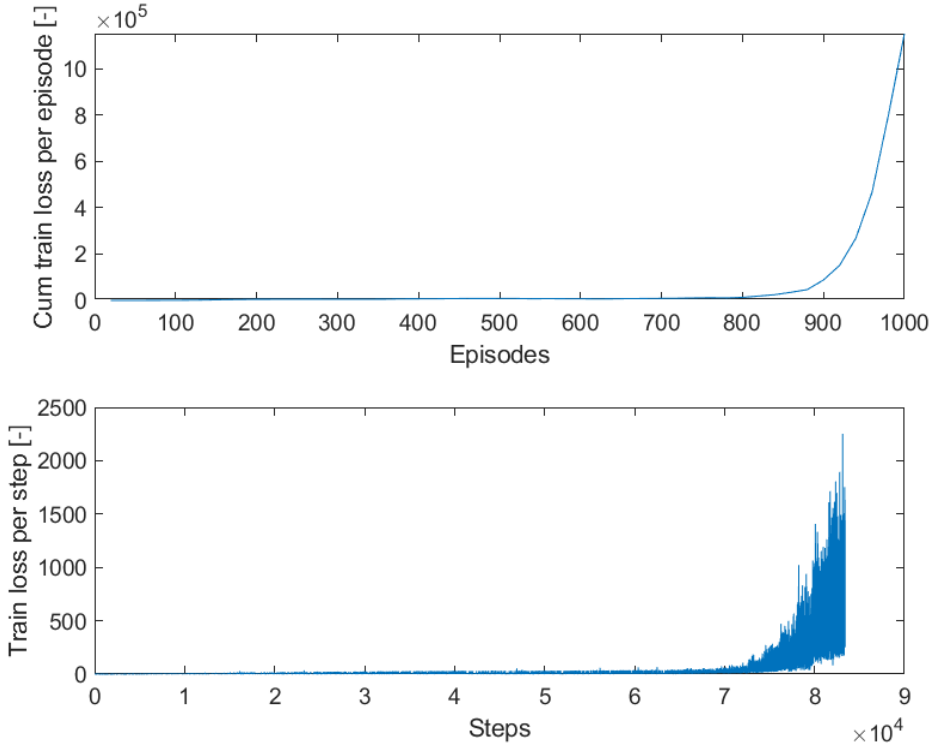

Figura 5.16: Train loss-Simulazione 4

La normalizzazione degli stati incide pesantemente sulle prestazioni dell'agente ma non riesce a risolvere la divergenza dei *Q-values* e della *train loss*. Analizzando le traiettorie di *SOC e Cumulative FC* si può notare come la *policy* individuata dall'agente non sia ottimale, in quanto continua ad essere preferito un mantenimento dello stato di carica a scapito di un eccessivo aumento del consumo di carburante. Quindi, sulla base della funzione di *reward* definita precedentemente, all'avanzare degli

episodi di *training* si ha un peggioramento delle performance dell'agente con conseguente diminuzione del *Discounted return*. L'esplosione del gradiente provoca la divergenza dei *Q-values* rendendo il processo di addestramento instabile, generando di fatto dei risultati fisici che non sono in linea con la funzione di *reward* utilizzata.

5.1.4 Effetto del *learning rate*

Una errata configurazione del *learning rate* ed in particolare un suo valore troppo alto si traduce in grandi aggiornamenti dei pesi della rete, con conseguente divergenza dei *Q-values*. Per ottenere una convergenza del modello è stato adottato un *learning rate* più piccolo di un ordine di grandezza. Poiché tale parametro controlla la velocità di apprendimento, una sua riduzione permette di apprendere un set di pesi ottimali provocando al contempo una dilatazione dei tempi di allenamento, perciò in questa circostanza si è deciso di aumentare il numero di episodi su cui l'agente viene addestrato. E' stata apportata anche una modifica alla strategia di esplorazione, ossia è stato imposto $\varepsilon_{start} = 1$ in modo tale da avere completa esplorazione nelle fasi iniziali dell'addestramento, e per far si che tale parametro decresca fino al suo valore limite sempre in corrispondenza del 75% della missione di guida. Dovendo affrontare un problema di gestione dell'energia tra EM e ICE ed avendo scelto come variabili di controllo il PF e la marcia, sono stati individuati come stati ottimali: *SOC* (stato di carica della batteria), *roadVel (velocità veicolo al time step t), next_roadVel (velocità veicolo al time step t+1).* Assolutamente fondamentale è il SOC, questo infatti non può andare oltre certi limiti imposti per garantire sia il corretto funzionamento della batteria che il completamento del ciclo guida. Inoltre, la velocità di avanzamento al *time step t* e *t+1* aggiungono delle informazioni riguardo l'accelerazione e la potenza necessaria alla trazione del veicolo.
Di seguito vengono mostrate le modifiche apportare rispetto alla prova del paragrafo precedente ed i risultati ottenuti dal lancio della simulazione:

- $Learning\ rate = 0.0002$
- Numero episodi: 1300
- $Strategia\ di\ esplorazione$: : $\varepsilon_{start} = 0.8,\ \varepsilon_{finish} = 0.05, \varepsilon_{decay\ episode} = 0.001$
- $Stati: SOC, roadVel, next\_roadVel$

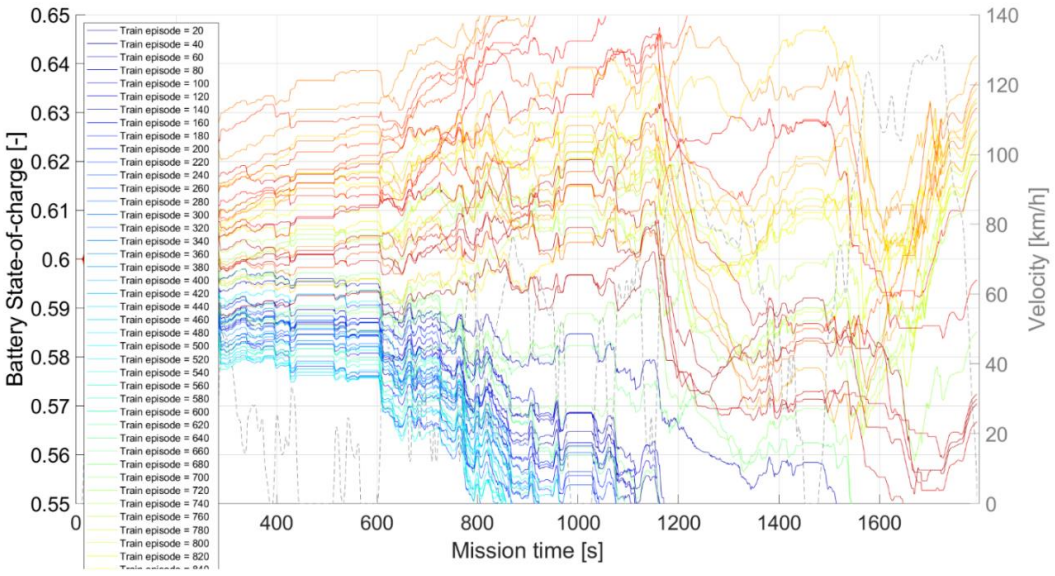

Figura 5.17: SOC episodi di train-Simulazione 5

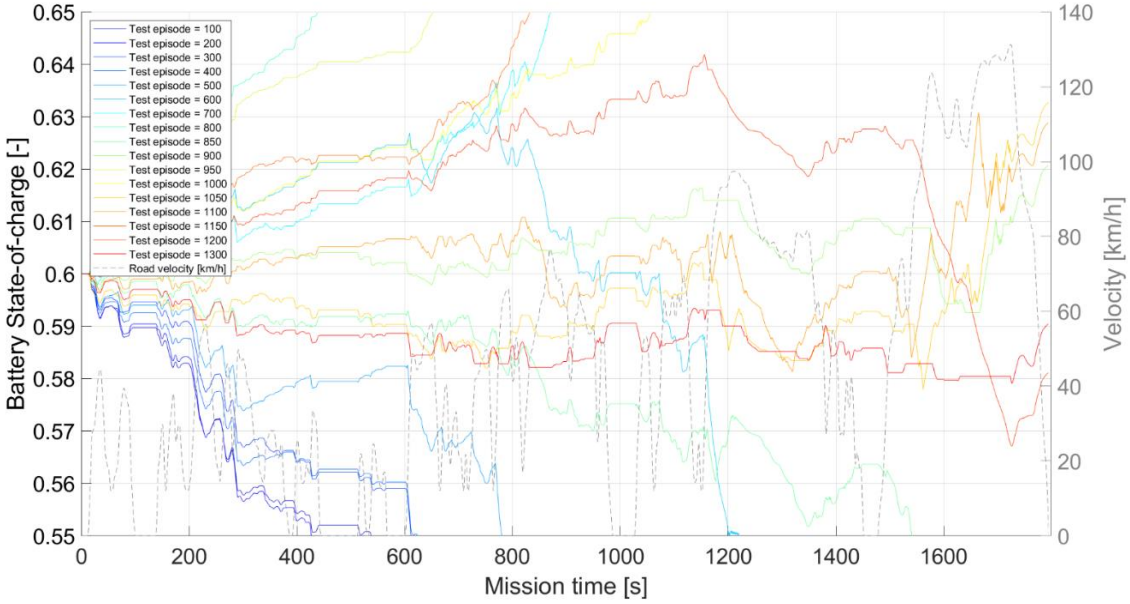

Figura 5.18: SOC episodi di test-Simulazione 5

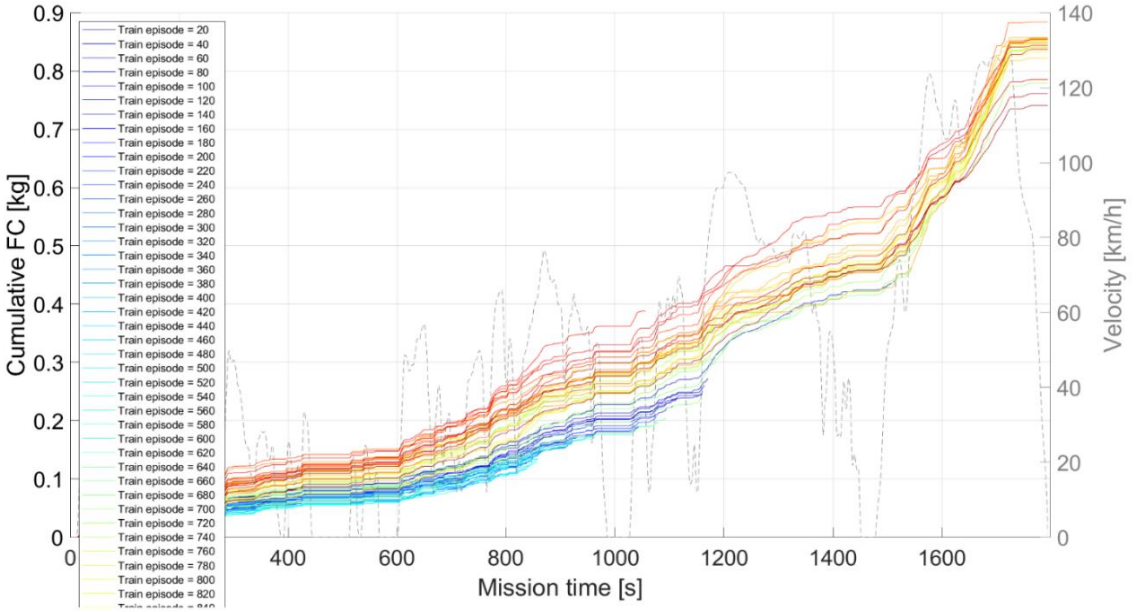

Figura 5.19: Cumulative FC episodi di train-Simulazione 5

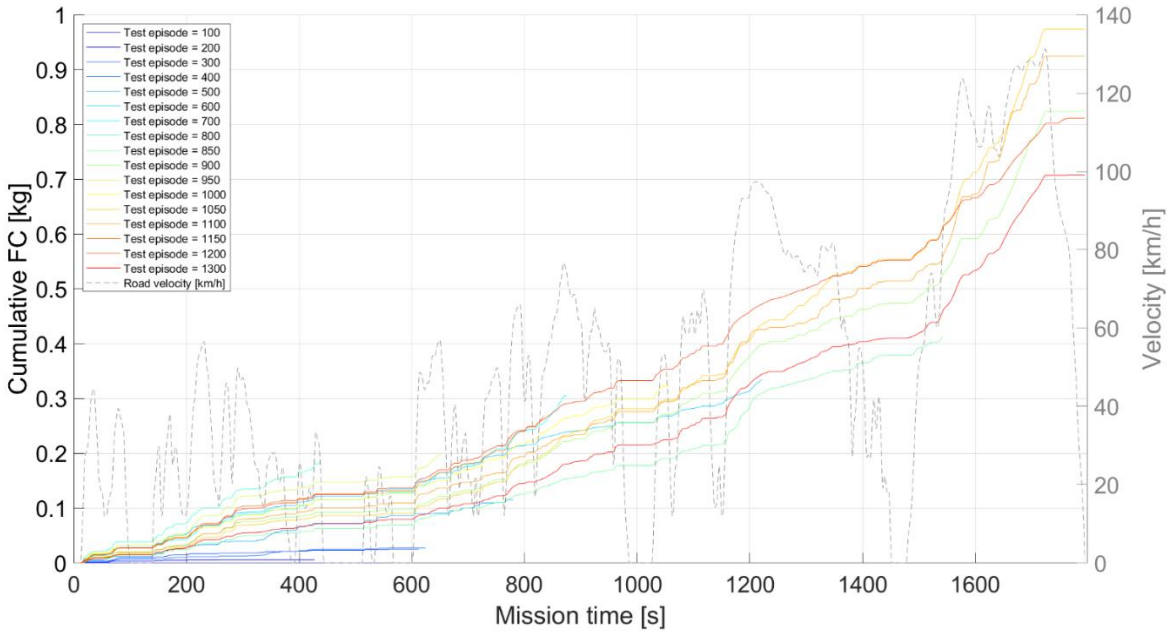

Figura 5.20: Cumulative FC episodi di test-Simulazione 5

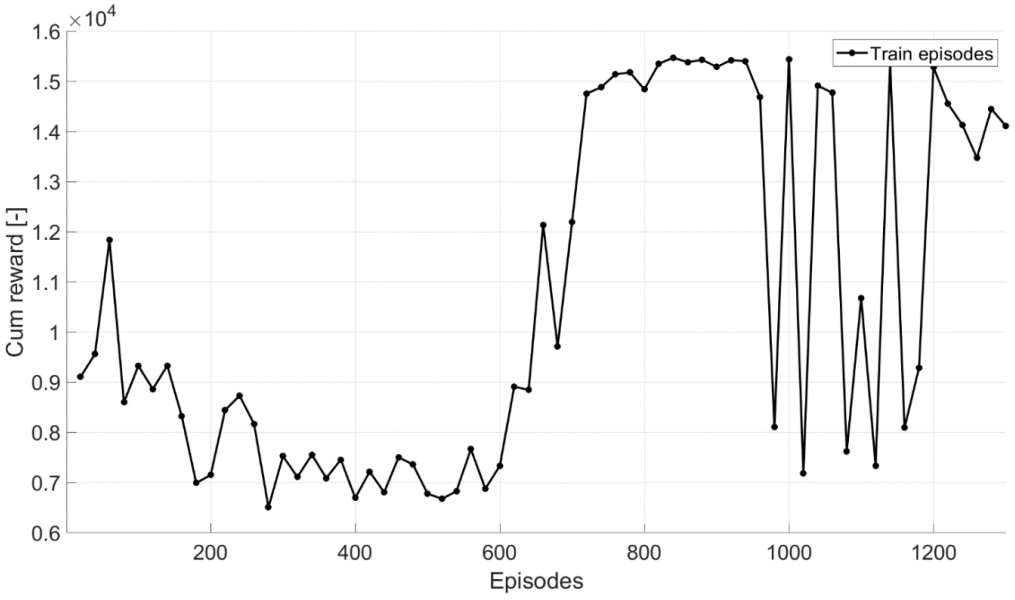

Figura 5.21:Cumulative reward episodi di train-Simulazione 5

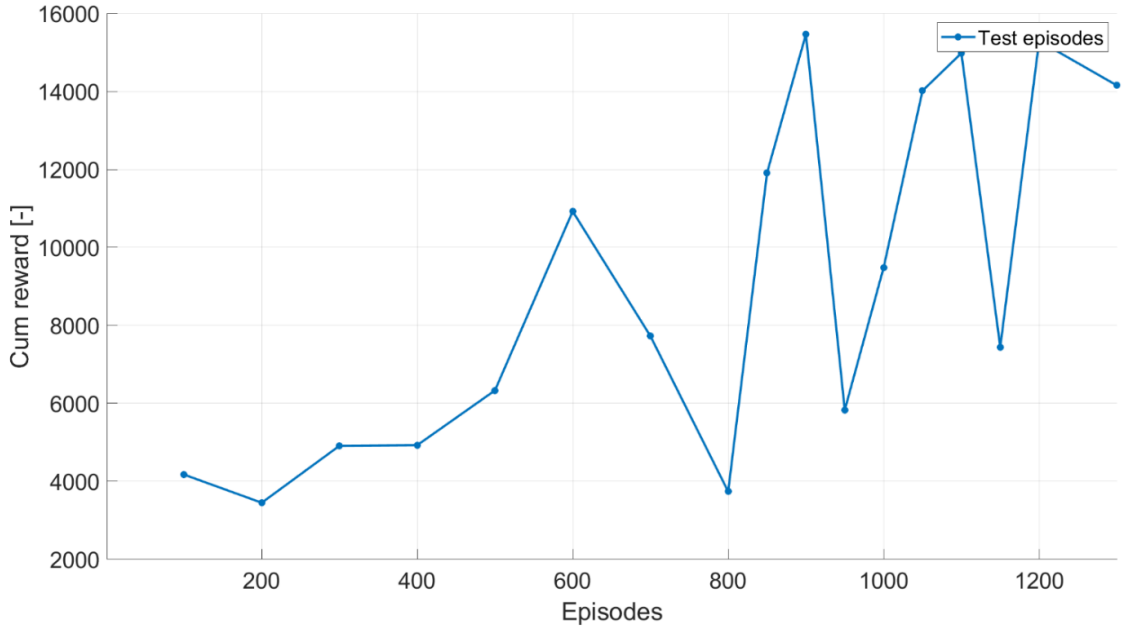

Figura 5.22:Cumulative reward episodi di test-Simulazione 5

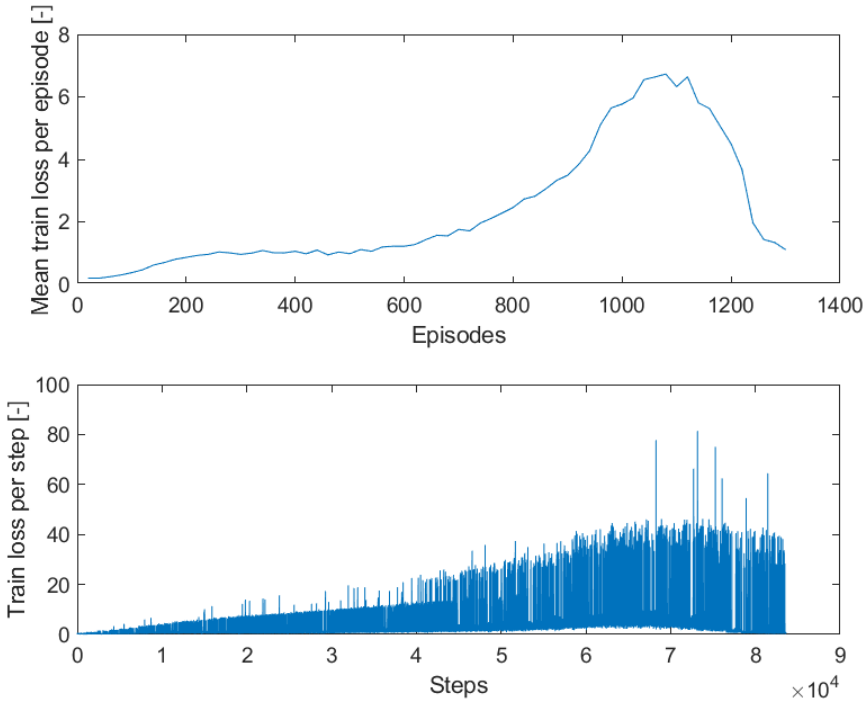

Figura 5.23 : Train loss-Simulazione 5

La riduzione del *learning rate* migliora notevolmente le performance dell'agente, difatti tale modifica scongiura la divergenza dei *Q-values* e conseguentemente quella della *train loss*. L'agente riesce ad individuare delle ottime strategie di controllo, minimizzando il FC e controllando al contempo il SOC. Nonostante il modello giunga a convergenza è doveroso notare come però la fase di addestramento sia piuttosto instabile e confusionaria, in quanto il *Cumuative reward* risulta essere particolarmente oscillante tra un episodio e quello successivo, specie nelle fasi finali dell'allenamento.

5.1.5 Effetto del *Replay memory size*

La dimensione del *buffer*, all'interno del quale andiamo ad immagazzinare tutte le esperienze, deve essere configurato in modo che l'agente dimentichi le informazioni non più necessarie, quindi deve essere garantito un aggiornamento del *buffer* stesso con una certa frequenza durante il *training*. L'aggiornamento è di tipo ciclico, ossia ogni qual volta che deve essere aggiunta una nuova esperienza ma il *buffer* ha raggiunto la sua capacità massima, l'esperienza più vecchia viene rimossa per fa spazio a quella nuova. Successivamente dal *buffer* viene estratto un *minibatch* in maniera randomica, su cui la rete si addestra, rimuovendo così le correlazioni nella sequenza di osservazioni e riducendo la varianza degli aggiornamenti.
In questa sezione sono stati analizzati gli effetti della riduzione del *Replay memory size* sulle performance dell'agente.

- $Replay\ memory\ size = 10000$

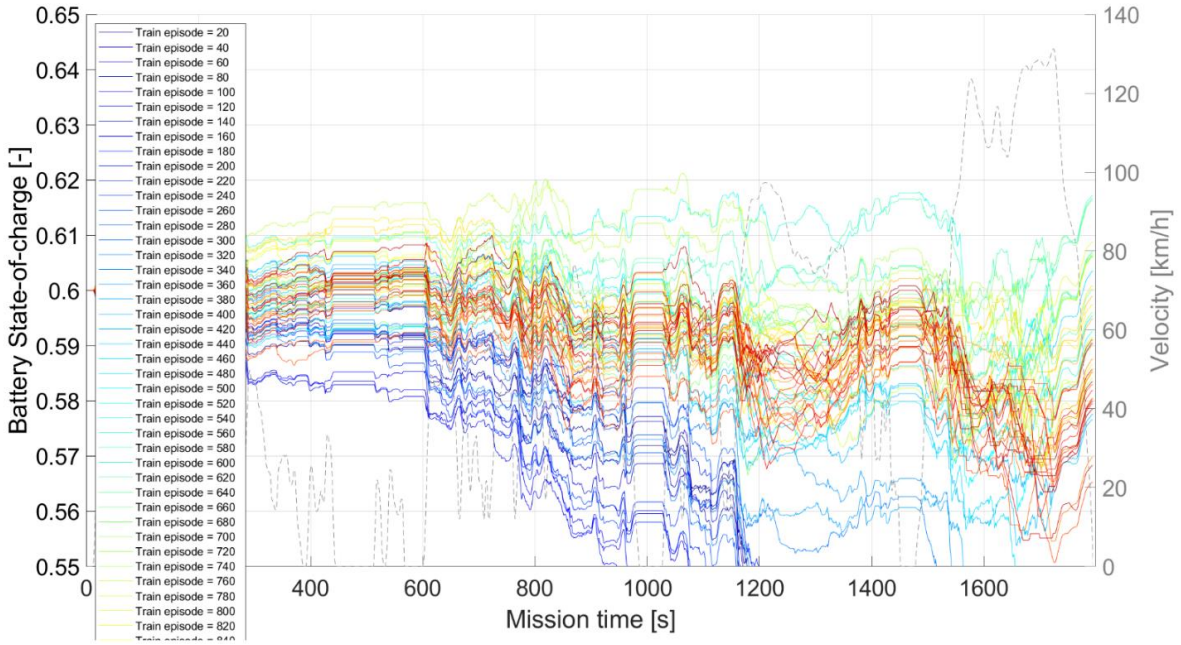

Figura 5.24: SOC episodi di train-Simulazione 6

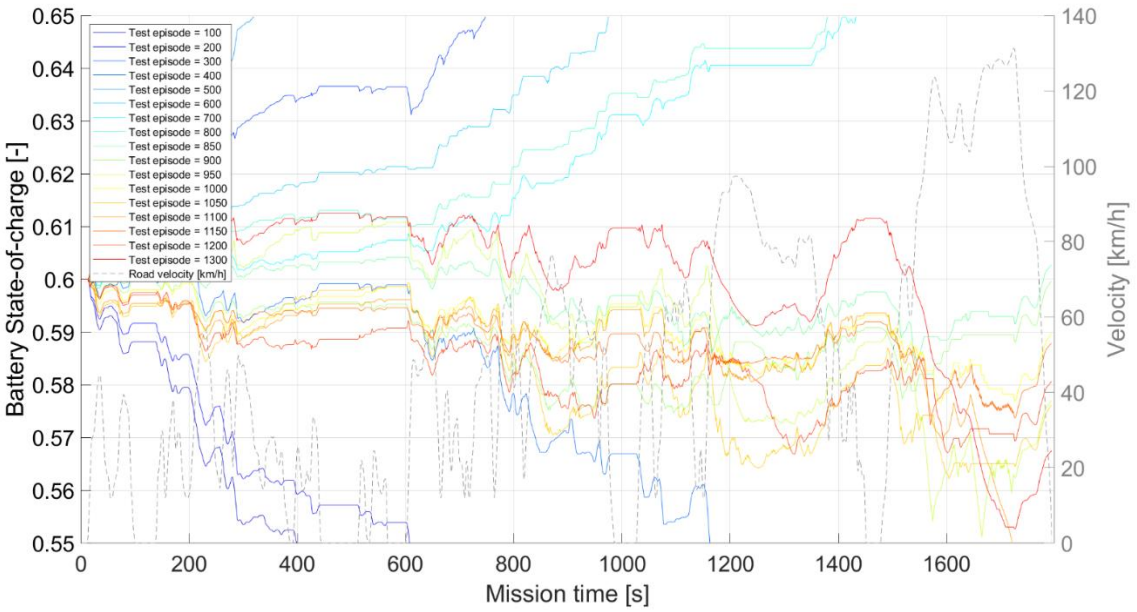

Figura 5.25: SOC episodi di test-Simulazione 6

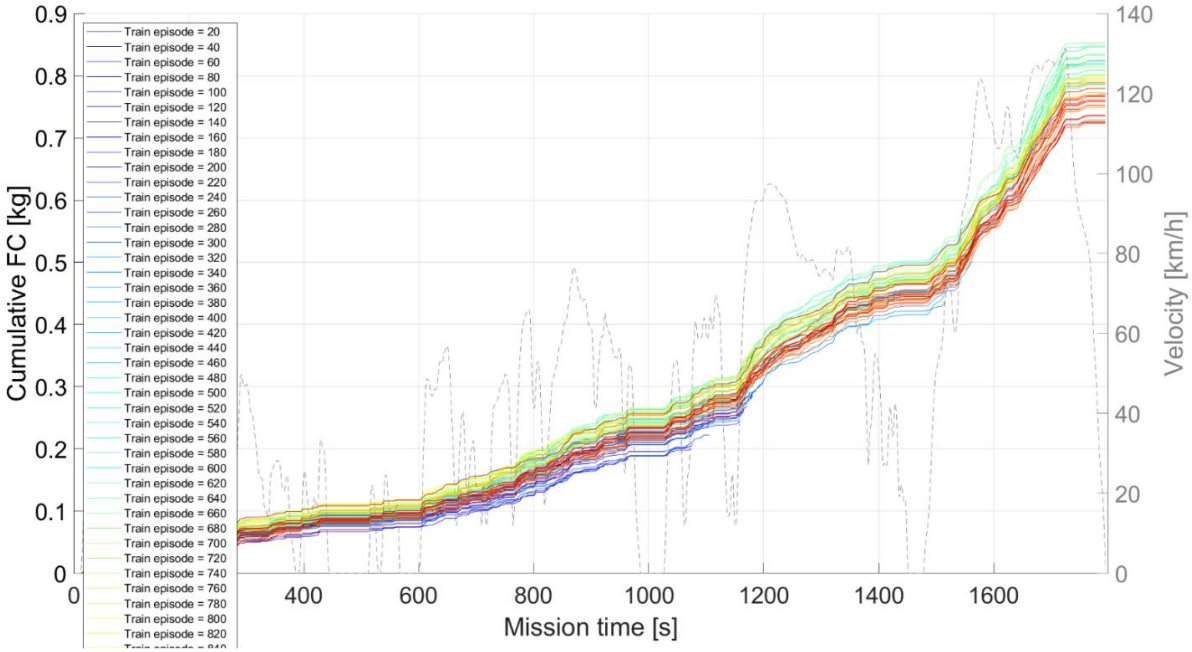

Figura 5.26: Cumulative FC episodi di train-Simulazione 6

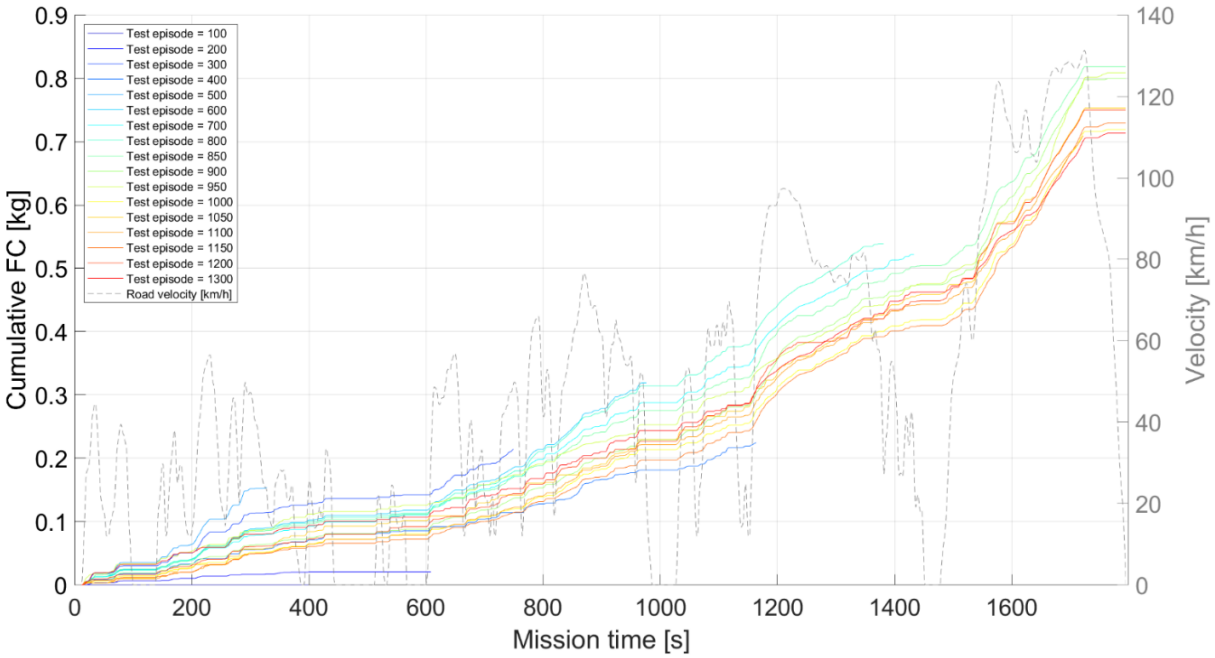

Figura 5.27: Cumulative FC episodi di test-Simulazione 6

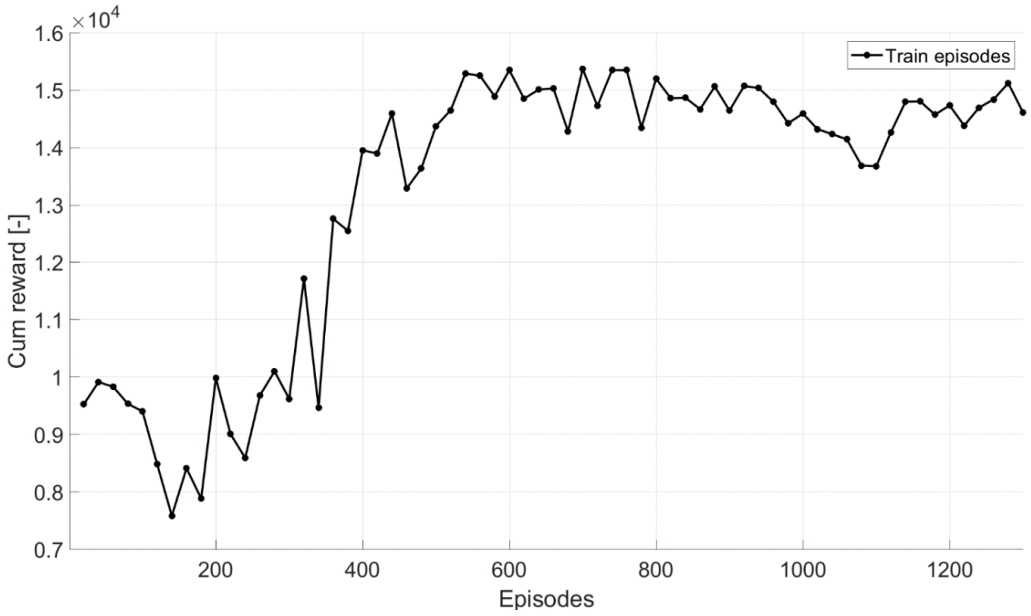

Figura 5.28: Cumulative reward episodi di train-Simulazione 6

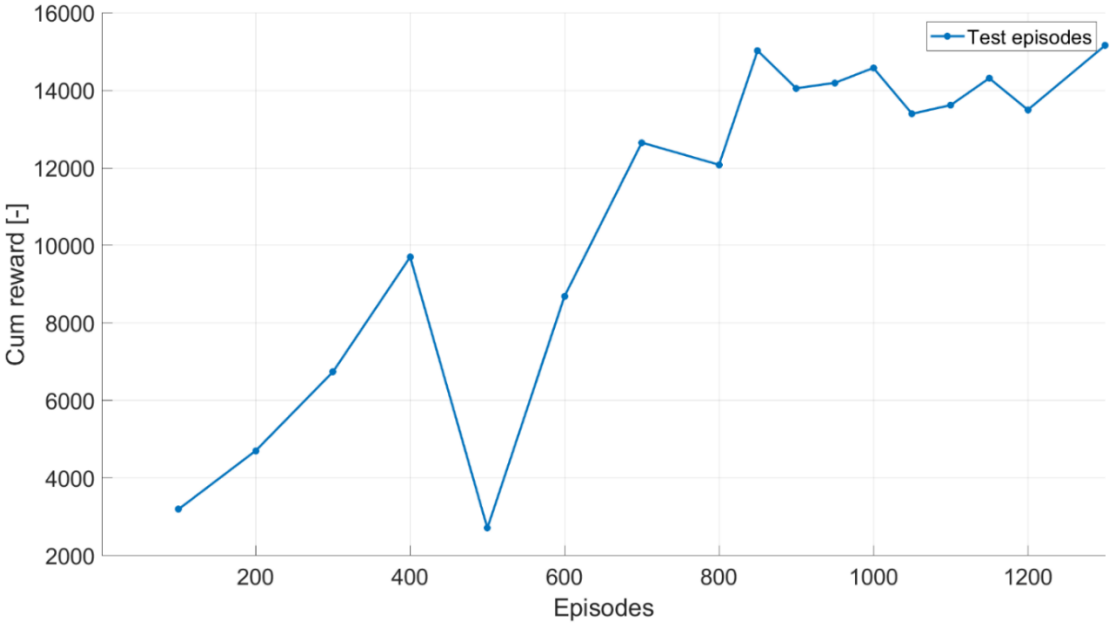

Figura 5.29: Cumulative reward episodi di test-Simulazione 6

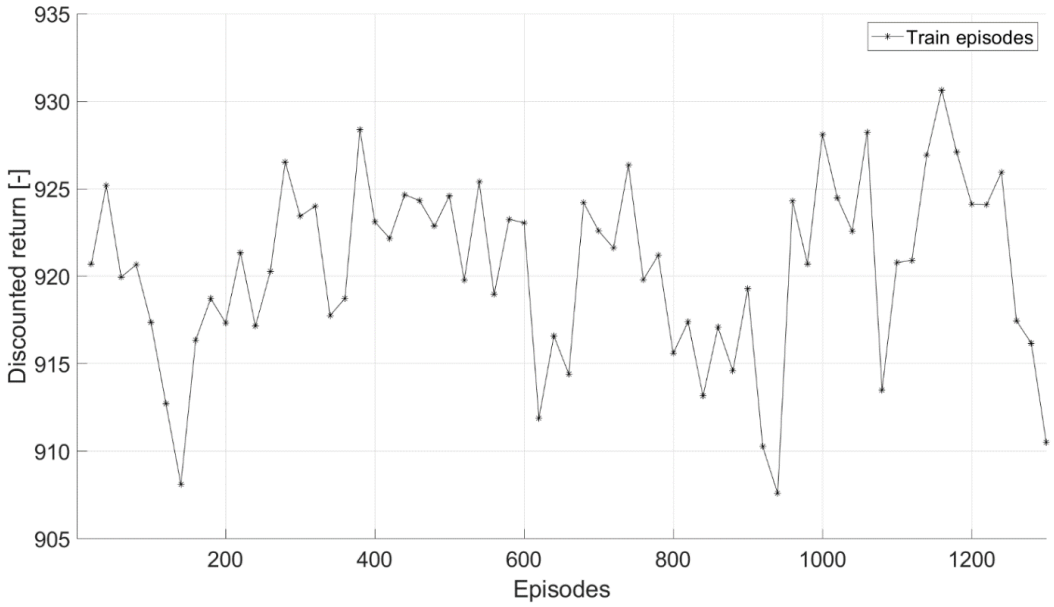

Figura 5.30: Discounted return episodi di train-Simulazione 6

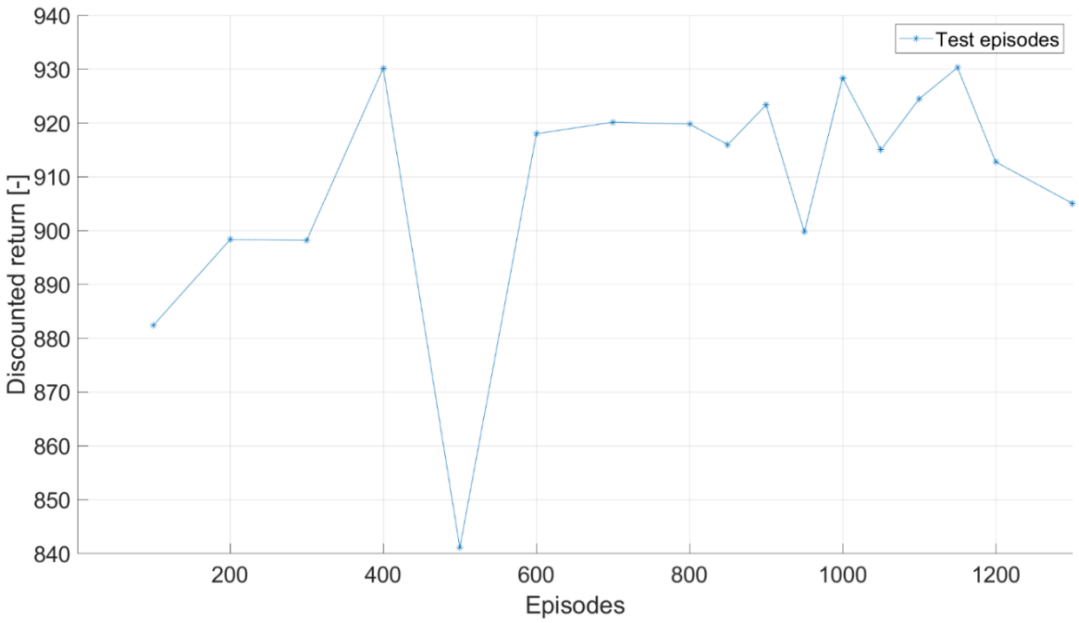

Figura 5.31: Discounted return episodi di test-Simulazione 6

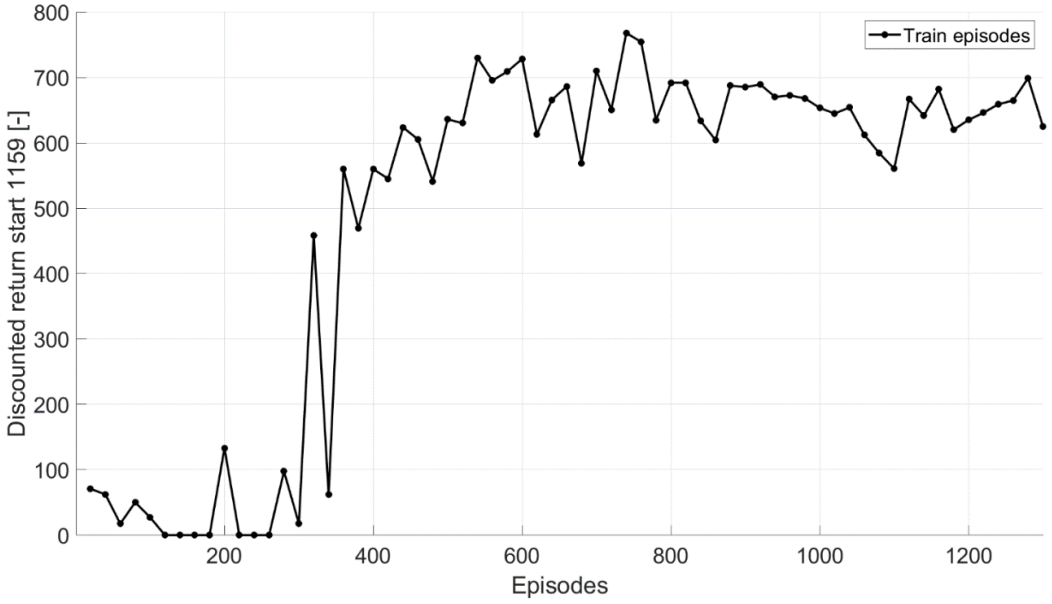

Figura 5.32: Discounted return start 1159 episodi di train-Simulazione 6

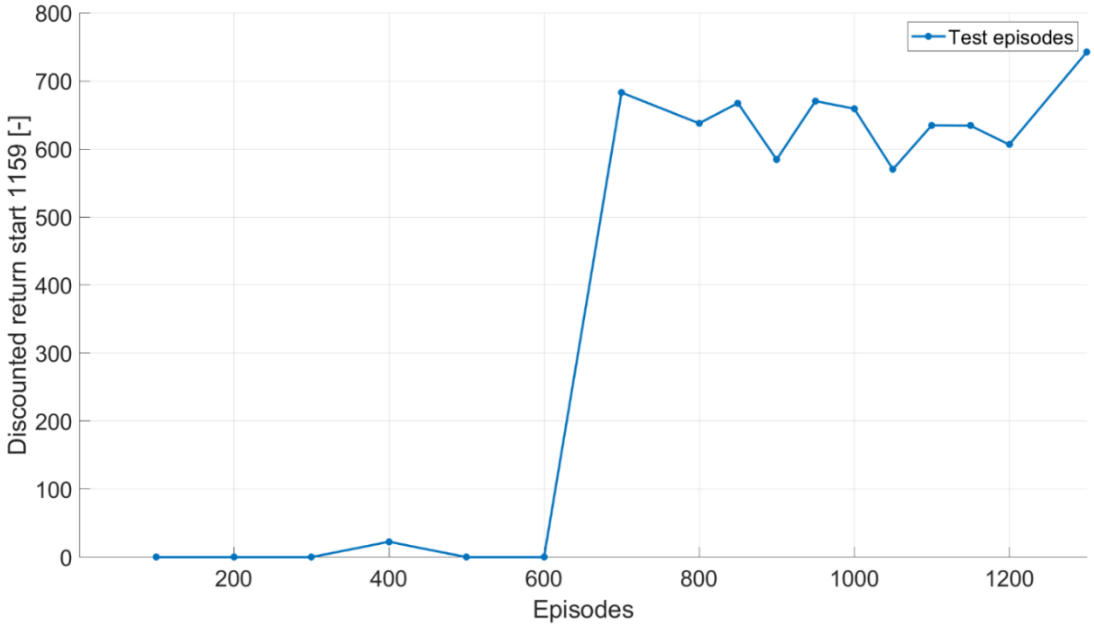

Figura 5.33: Discounted return start 1159 episodi di test-Simulazione 6

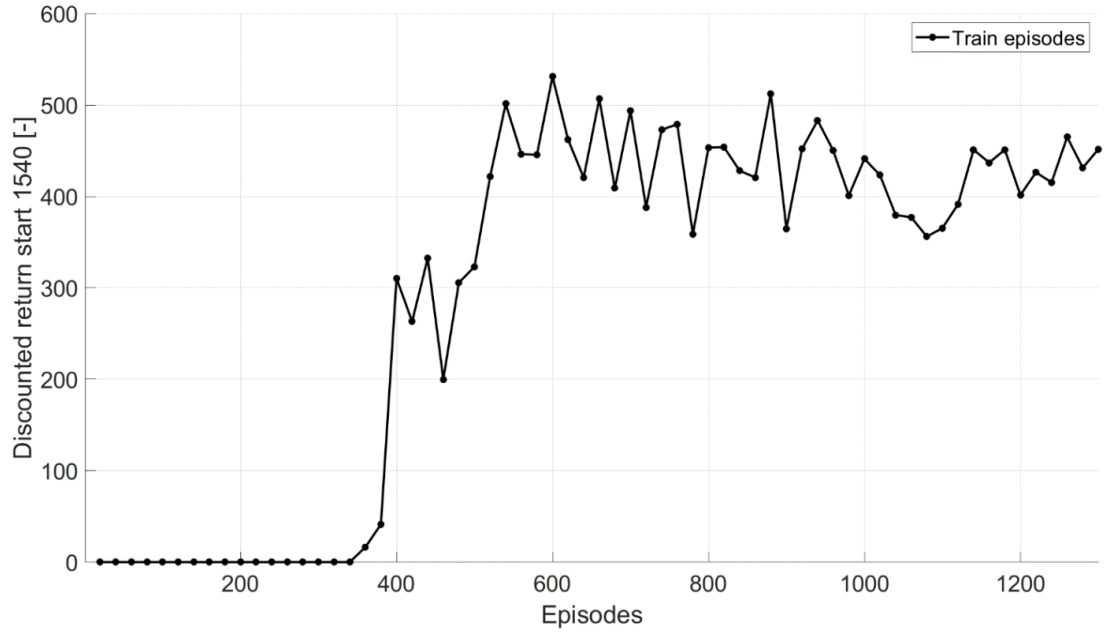

Figura 5.34: Discounted return start 1540 episodi di train-Simulazione 6

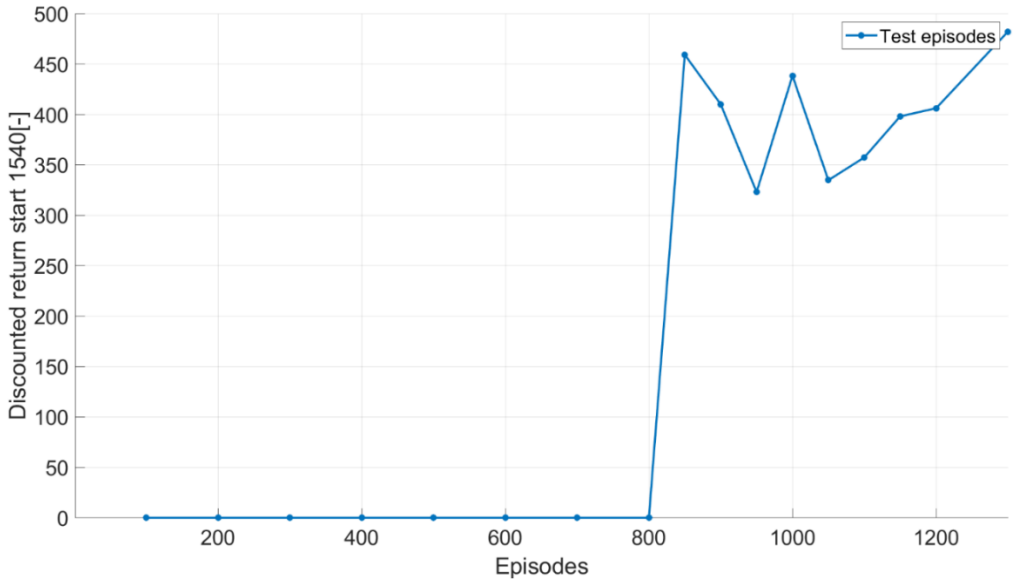

Figura 5.35: Discounted return start 1540 episodi di test-Simulazione 6

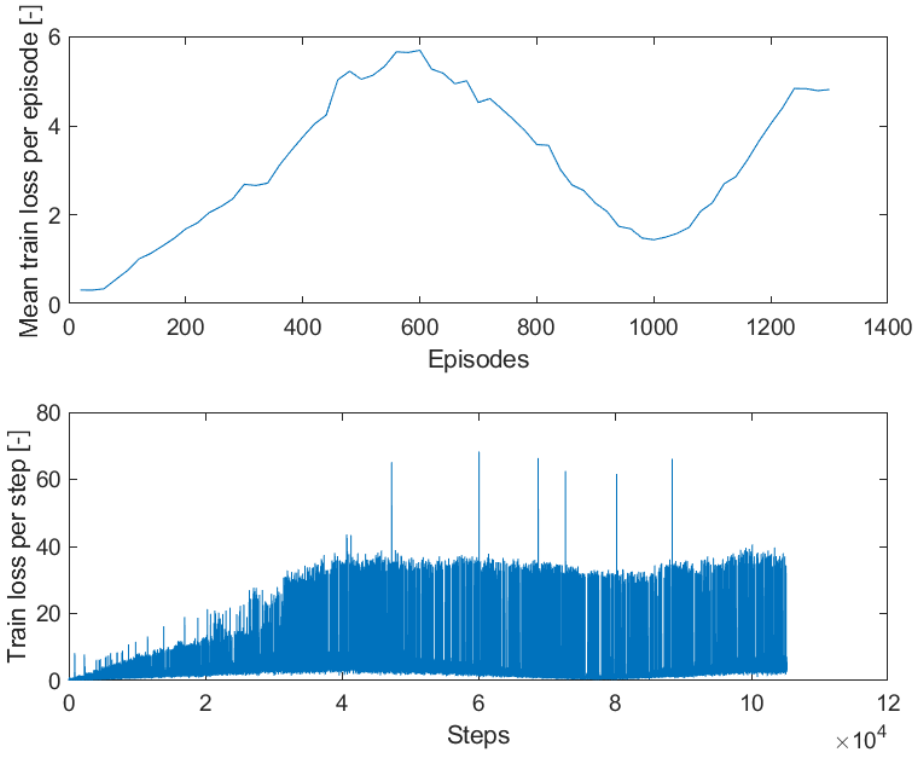

Figura 5.36: Train loss-Simulazione 6

In questa simulazione è stato utilizzato un buffer di circa due ordini di grandezza inferiori rispetto a quello dell'esperimento presente al paragrafo precedente, in questo modo si ha un ricircolo più frequente delle esperienze accumulate. Tale modifica quindi stabilizza l'allenamento dell'agente migliorando anche i risultati fisici ottenuti. In Figura 5.32 viene riportato l'andamento della *loss* media per ogni singolo episodio di *training*. L'agente riesce a ridurre la *train loss* ma in corrispondenza del millesimo episodio si assiste ad un secondo innalzamento. Paradossalmente

nonostante vi sia un aumento della *loss* le performance dell'agente continuano a migliorare, in quanto si ha un'ulteriore minimizzazione dei consumi ed un aumento del *Cumulative reward* negli episodi finali. Poiché la *train loss* è calcolata come il quadrato della differenza tra il valore Q previsto ed il valore Q target, anche una minima differenza può portare all'innalzamento di tale valore, quindi quando l'agente giunge in un intorno molto vicino al risultato sperato esso non riesce più a modificare la propria *policy* e ad azzerare l'errore, continuando a muoversi nell'intorno del minimo globale. Perciò tale esperimento risulta essere completamente coerente dal punto di vista fisico, meno da quello matematico. Per superare tale problema potrebbe essere implementata una tecnica di *Early stopping*, ossia di arresto anticipato, una volta che la *loss* incomincia a risalire per un certo numero di episodi. Nelle Figure 5.31-5.34 si è analizzato l'andamento del *Discounted return* per gli episodi di *testing* e di *training* a partire da due punti critici del WLTP (1159 s-1540 s), in corrispondenza dei quali si hanno delle importanti accelerazioni, quindi avendo definito una *reward function FC-oriented* risulta essere più complesso per l'agente gestire questi tratti, in quanto necessita di quasi tutta la banda di SOC a disposizione per andare in puro elettrico. Difatti il *Discounted retun* calcolato a partire da questi punti è inferiore rispetto a quello calcolato a partire dall'inizio della simulazione. Inoltre, sono state lanciate delle simulazioni anche con un *Replay memory size* intermedio di 100000 e un *batch size* di 64 e 128. Tali modifiche non hanno apportato dei miglioramenti apprezzabili sulla performace dell'agente, appesantendo al contempo dal punto di vista computazionale l'esperimento. Di seguito vengono riportati i principali risultati ottenuti con agente DQN su ciclo WLTP e le configurazioni utilizzate:

Consumo di carburante in kg	0.71
Consumo di carburante in km/l	25.6
SOC iniziale	0.6
SOC finale	0.5675
Cumulative reward	15159

Tabella 5.2: Risultati su ciclo WLTP con agente DQN

Agente DQN	*Funzione di attivazione*	ReLU
	Taglia del minibatch	32
	Discounted factor γ	0.99
	Numero di neuroni	64
	Numero di layer	2
	Learning starts	2000
	Target update frequence	6000
	Replay memory size	10000
	Learning rate lr	0.0002
Strategia di esplorazione	ε_{start}	1
	ε_{finish}	0.05
	$\varepsilon_{decay\ episode}$	0.001
Reward	*b*	10
	c	-3000
	d	-200
Stati	*SOC*	
	roadVel	
	next_roadVel	
Training	*Ciclo guida*	wltp
	Numero episodi	1300

Tabella 5.3: Configurazioni finali

6. Simulazioni sui cicli clust

Dopo aver trovato una soluzione sub-ottimale al nostro problema, l'agente DQN è stato testato anche su altri cicli guida, ossia i cicli clust, i quali non sono dei cicli omologativi ma sono molto utili per fare delle analisi in ambito di simulazione. In particolare sono state lanciate delle prove sui cicli clust4, clust7, clust11, clust12 utilizzando le configurazioni presenti in Tabella 5.3, con l'obbiettivo di valutare le performance dell'agente al variare delle condizioni di guida. In Tabella 6.1 vengono riportate le principali caratteristiche dei cicli clust, le quali vengono messe a confronto con il WLTP.

Ciclo	Durata	Vel. max	Acc. Max	Acc min.	Potenza media di trazione
	s	km/h	m/s^2	m/s^2	kw
WLTP	1800	131,3	1,75	-1,5	10
Clust4	810	70	2,1	-3,2	7
Clust7	799	93	1,5	-2,5	7,3
Clust11	653	76	2,5	-3,5	7,3
Clust12	475	55	2,2	-2,6	6

Tabella 6.1: Caratteristiche cicli clust e WLTP

6.1 Clust4

Il ciclo clust 4 ha una lunghezza complessiva di 810 s ed una velocità massima di 70 km/h. Inoltre, è caratterizzato da un'accelerazione massima e minima rispettivamente di 2,1 m/s^2 e -3,2 m/s^2. Sommariamente però tale ciclo risulta essere meno gravoso del WLTP, in quanto presenta una potenza media di trazione inferiore e di conseguenza anche l'FC medio sarà più basso.

Di seguito vengono mostrati i risultati ottenuti:

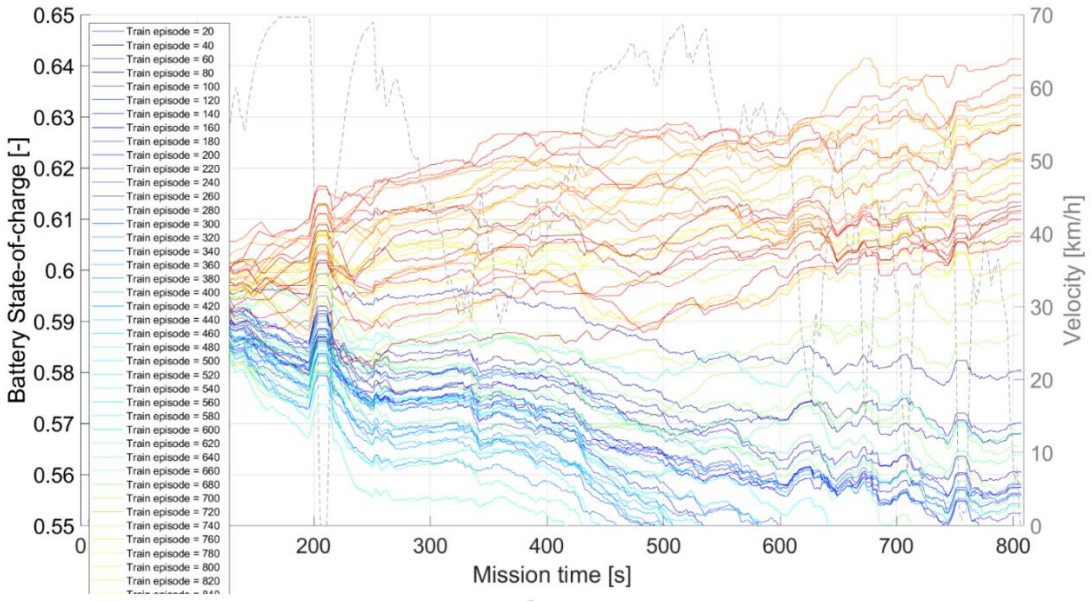

Figura 6.1: SOC episodi di train-clust4

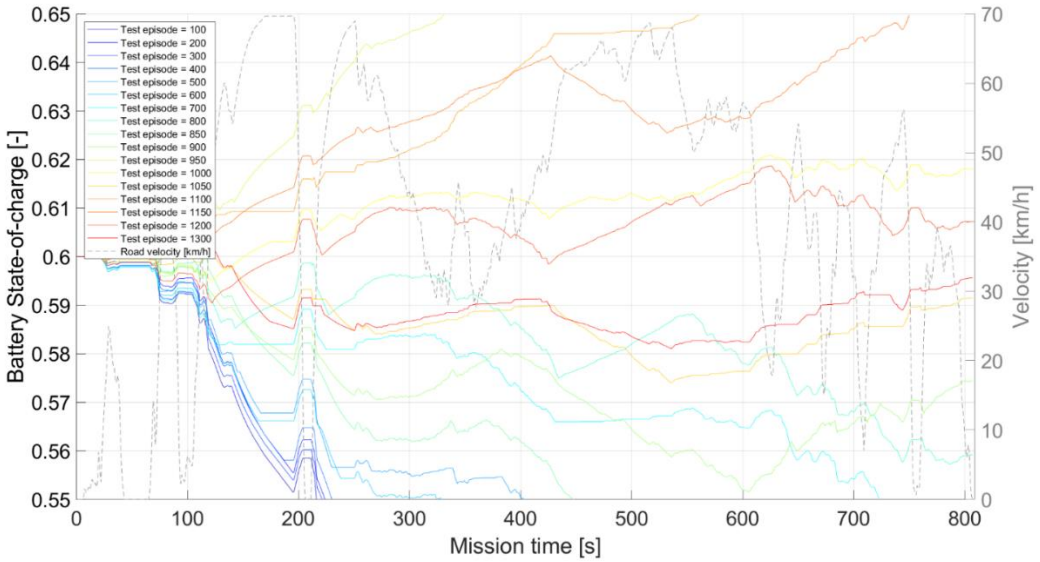

Figura 6.2: SOC episodi di test-clust4

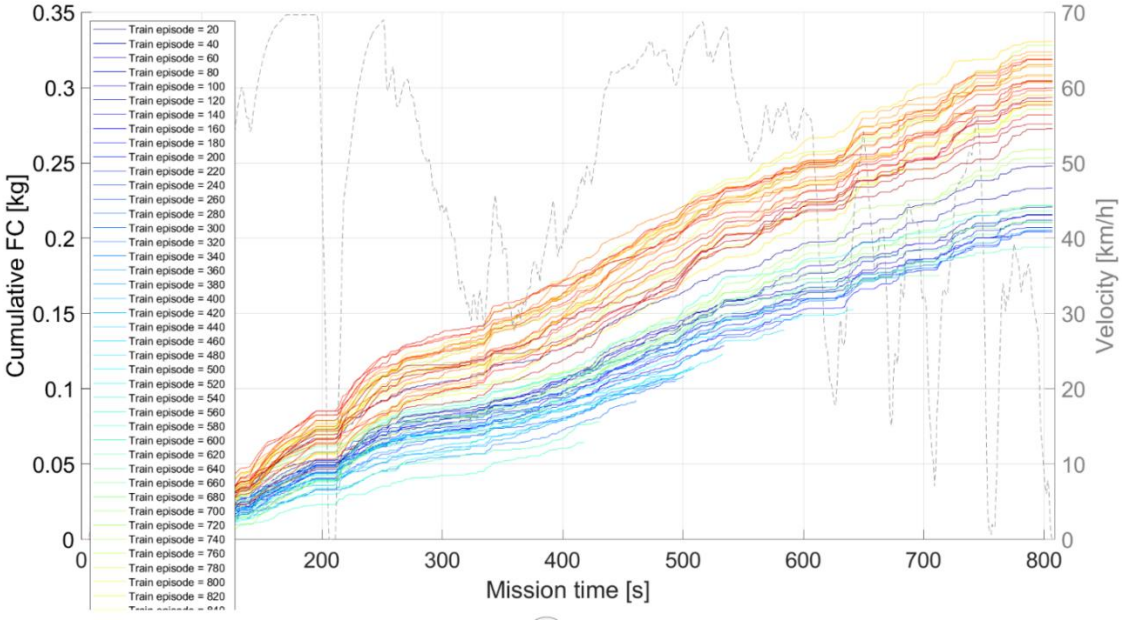

Figura 6.3: Cumulative FC episodi di train-clust4

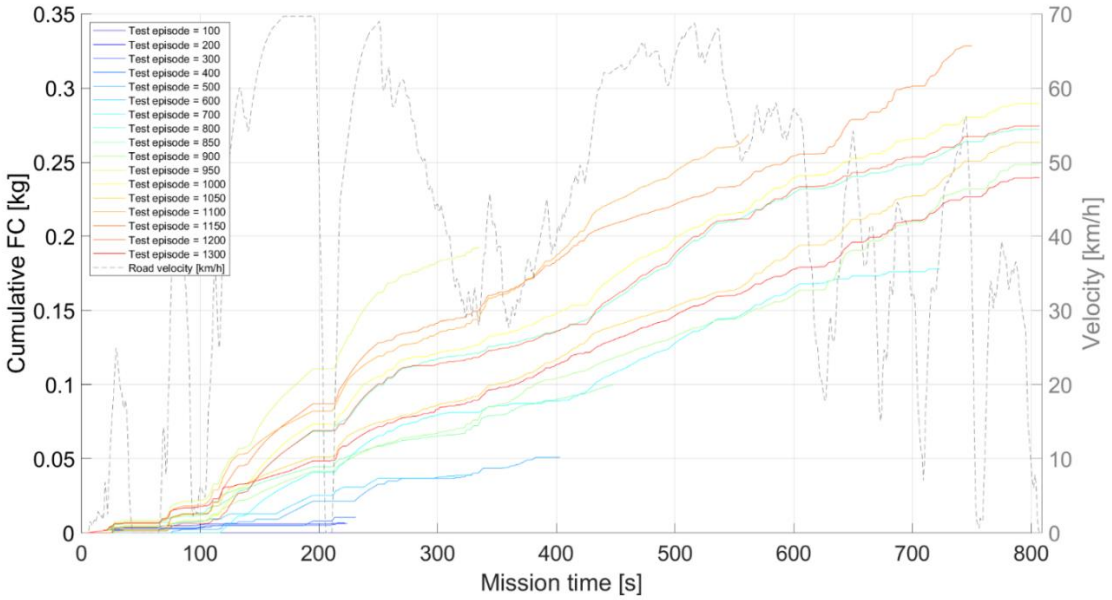

Figura 6.4: Cumulative FC episodi di test-clust4

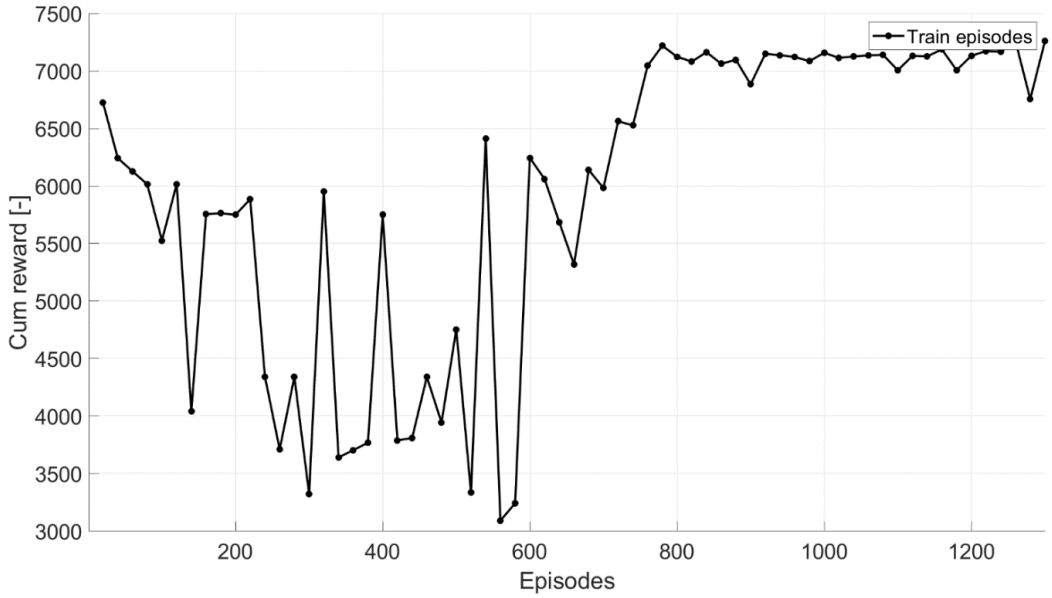

Figura 6.5: Discounted return episodi di train-clust4

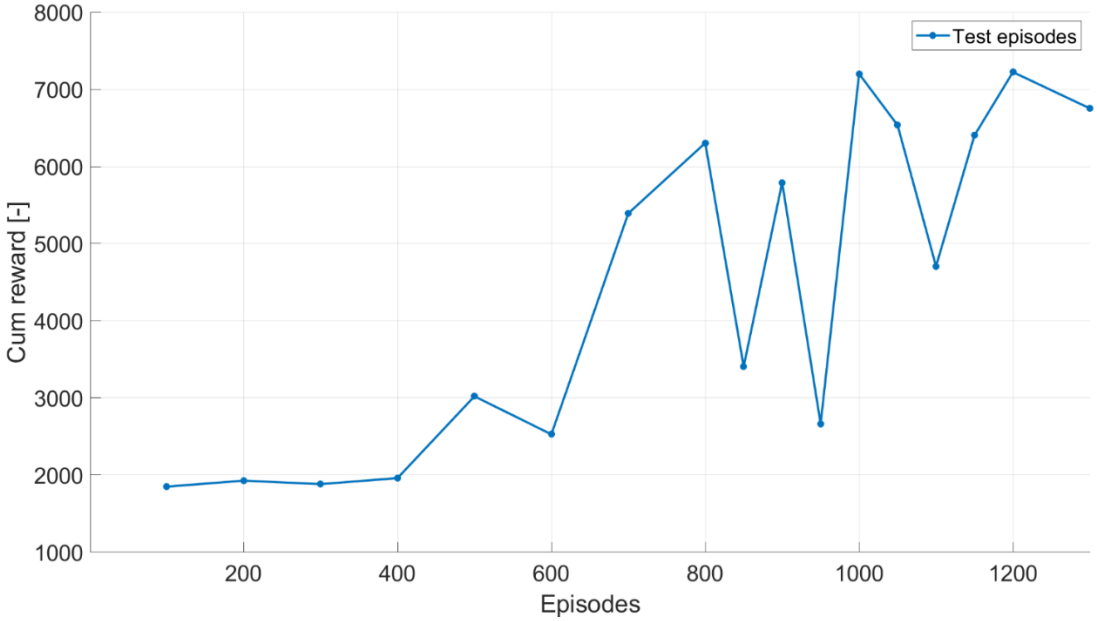

Figura 6.6: Discounted return episodi di test-clust4

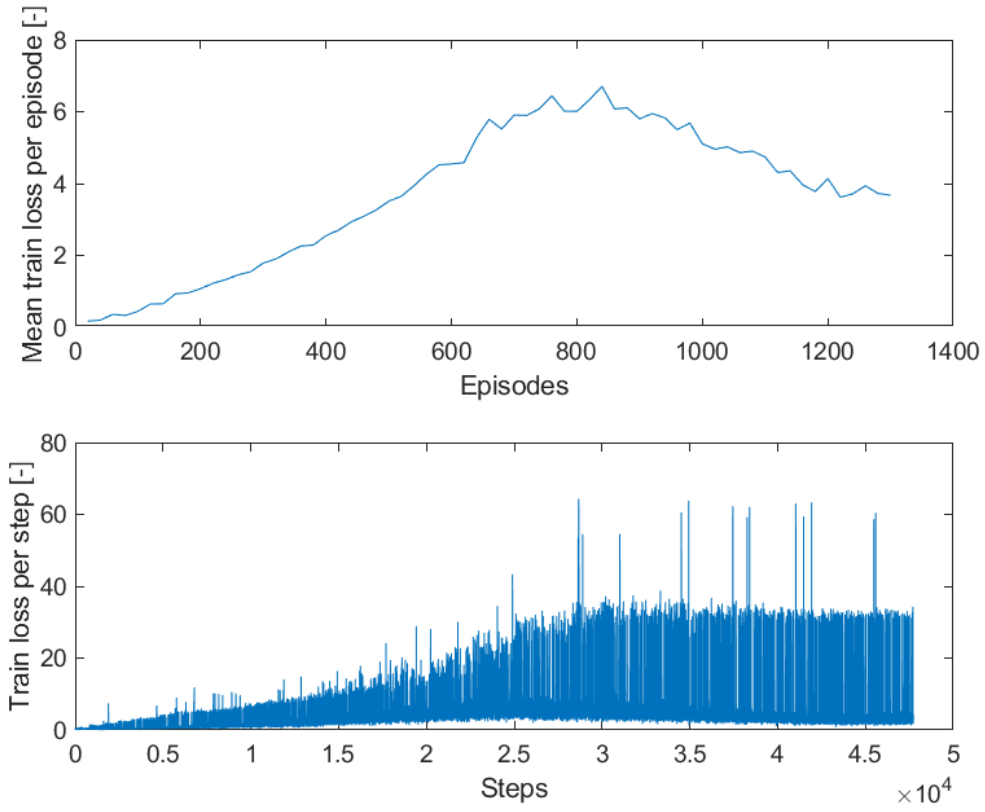

Figura 6.7: Train loss-clust4

Consumo di carburante in kg	0.2397
SOC iniziale	0.6
SOC finale	0.5956
Cumulative reward	6753

Tabella 6.2: Risultati ciclo clust4 con agente DQN

Durante l'allenamento l'agente riesce ad individuare una *policy* che massimizza la somma delle ricompense ottenute e permette di far decrescere la *train loss*. E' chiaro però come in questa circostanza il consumo di combustibile impatti di meno rispetto al ΔSOC sulla funzione di *reward*, infatti al crescere del SOC e conseguente aumento di FC si ha un aumento della ricompensa ottenuta. Perciò la funzione di *reward* non risulta essere così *FC-oriented* come lo era sul ciclo WLTP. Quindi otteniamo delle buone performance dell'agente e dei buoni risultati fisici sulla base della funzione di *reward* definita e delle caratteristiche del ciclo guida.

6.2 Clust7

Il ciclo clust7 ha una lunghezza complessiva di 799 s ed una velocità massima di 93 km/h. E' caratterizzato da un'accelerazione massima e minima rispettivamente di 1,5 m/s^2 e -2,5 m/s^2. In questo caso la potenza media di trazione è maggiore rispetto al clust4, ma sempre inferiore al WLTP.

Di seguito vengono riportati i risultati ottenuti:

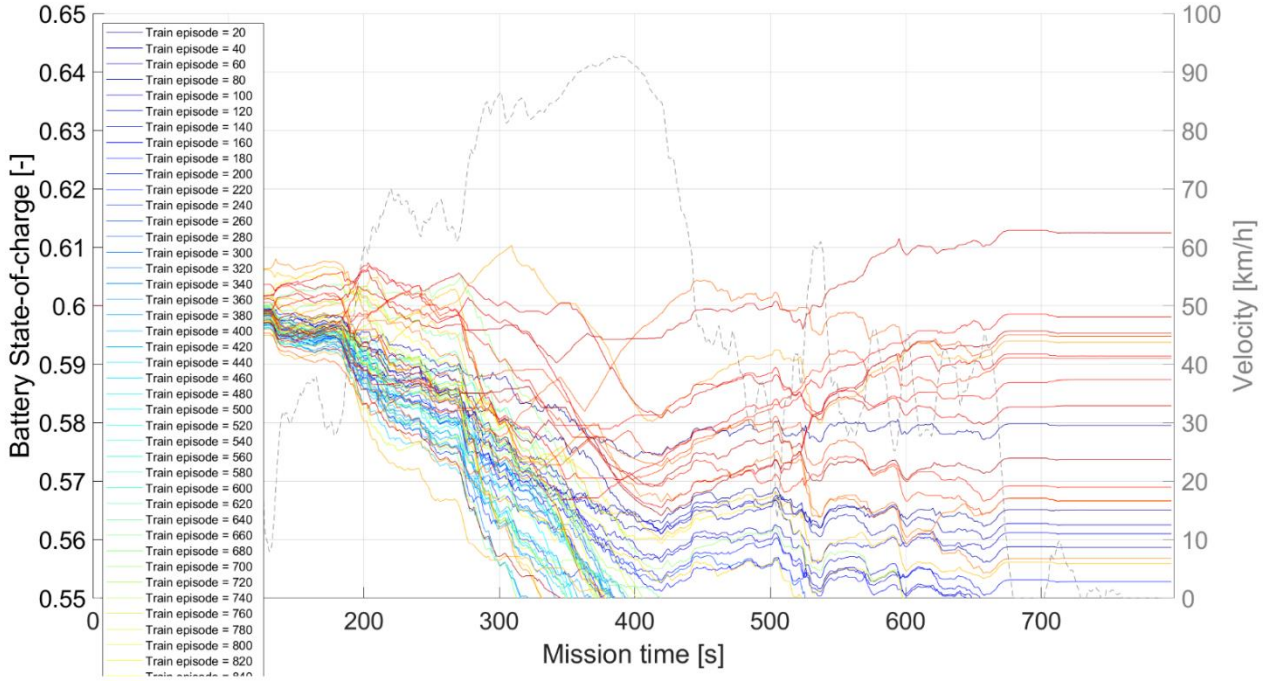

Figura 6.8: SOC episodi di train-clust7

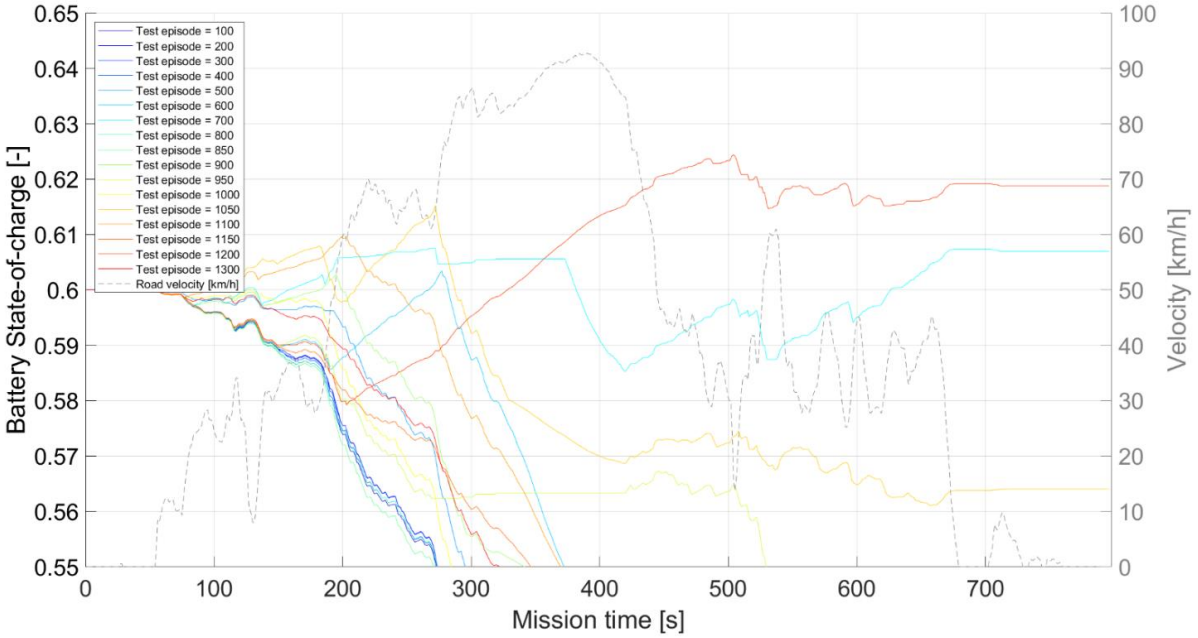

Figura 6.9: SOC episodi di test-clust7

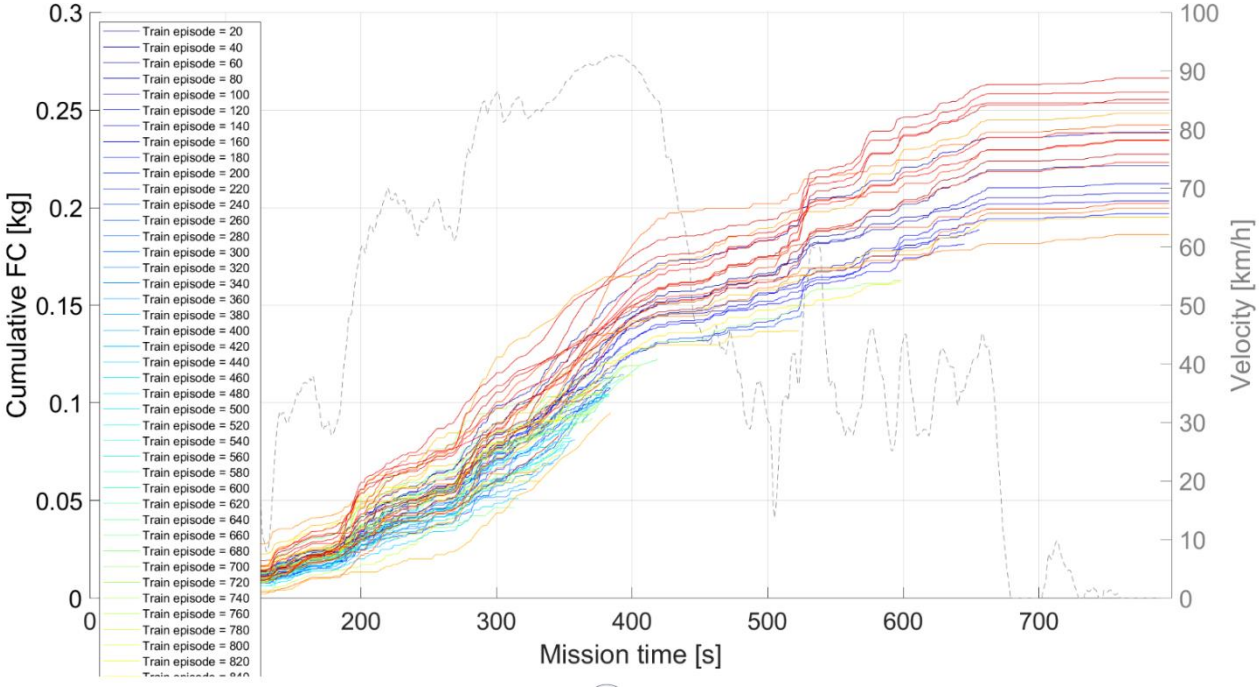

Figura 6.10: Cumulative FC episodi di train-clust7

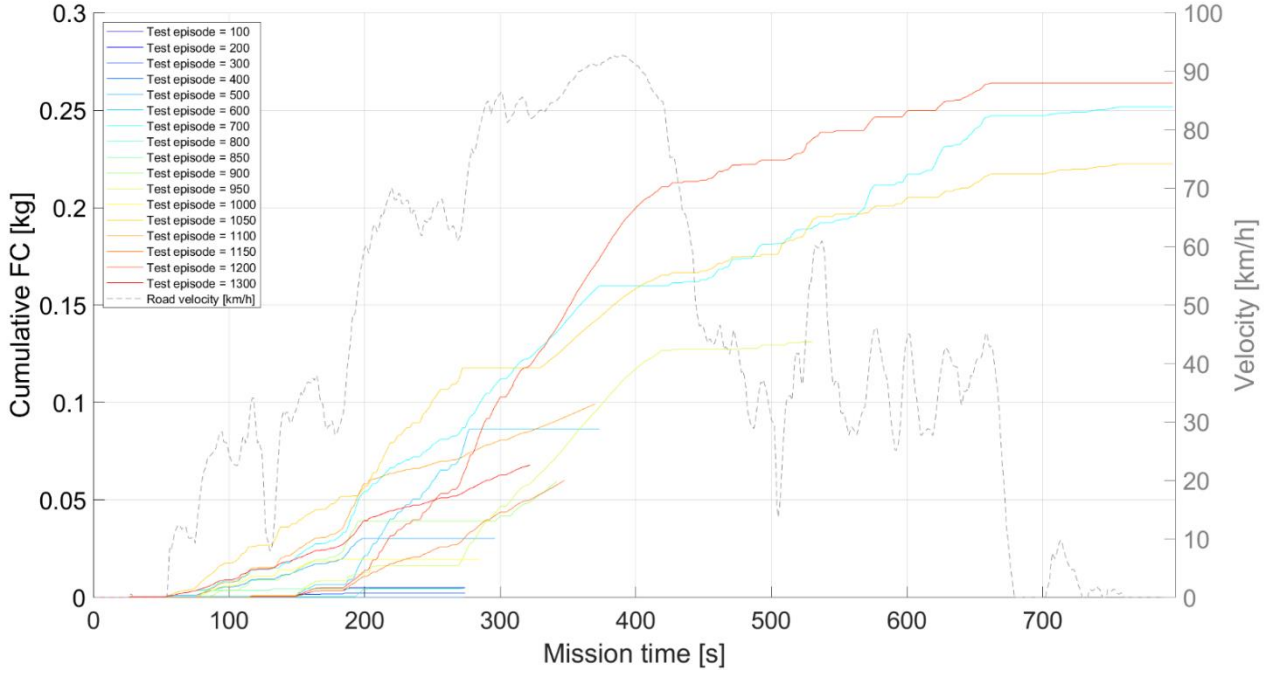

Figura 6.11: Cumulative FC episodi di test-clust7

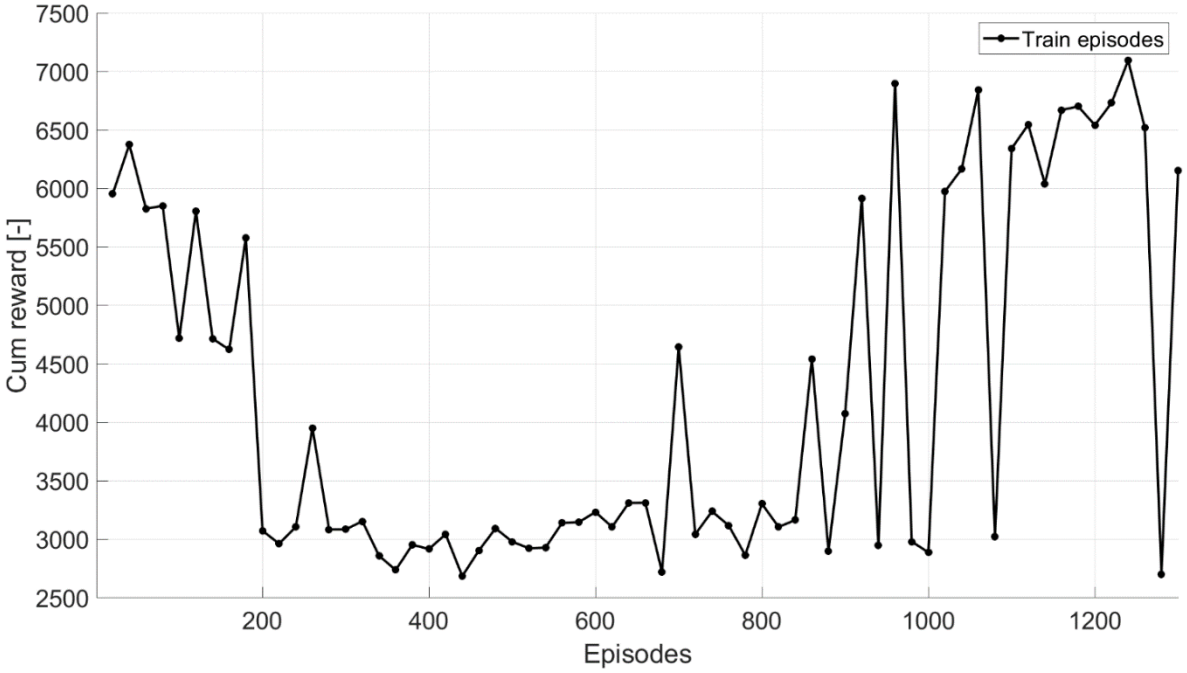

Figura 6.12: Discounted return episodi di train-clust7

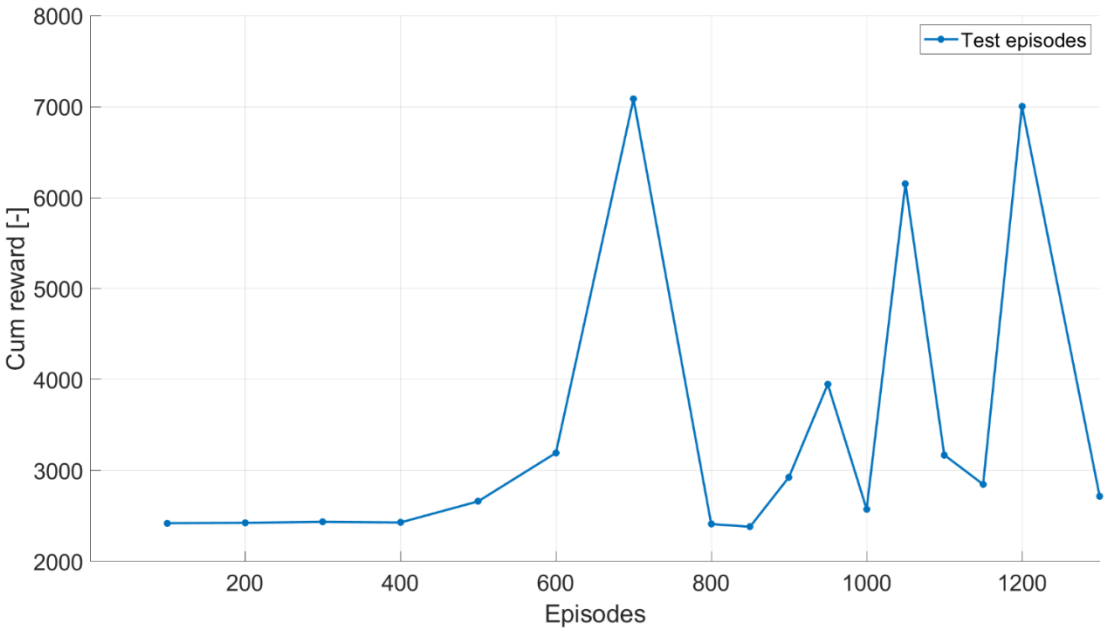

Figura 6.13: Discounted return episodi di test-clust7

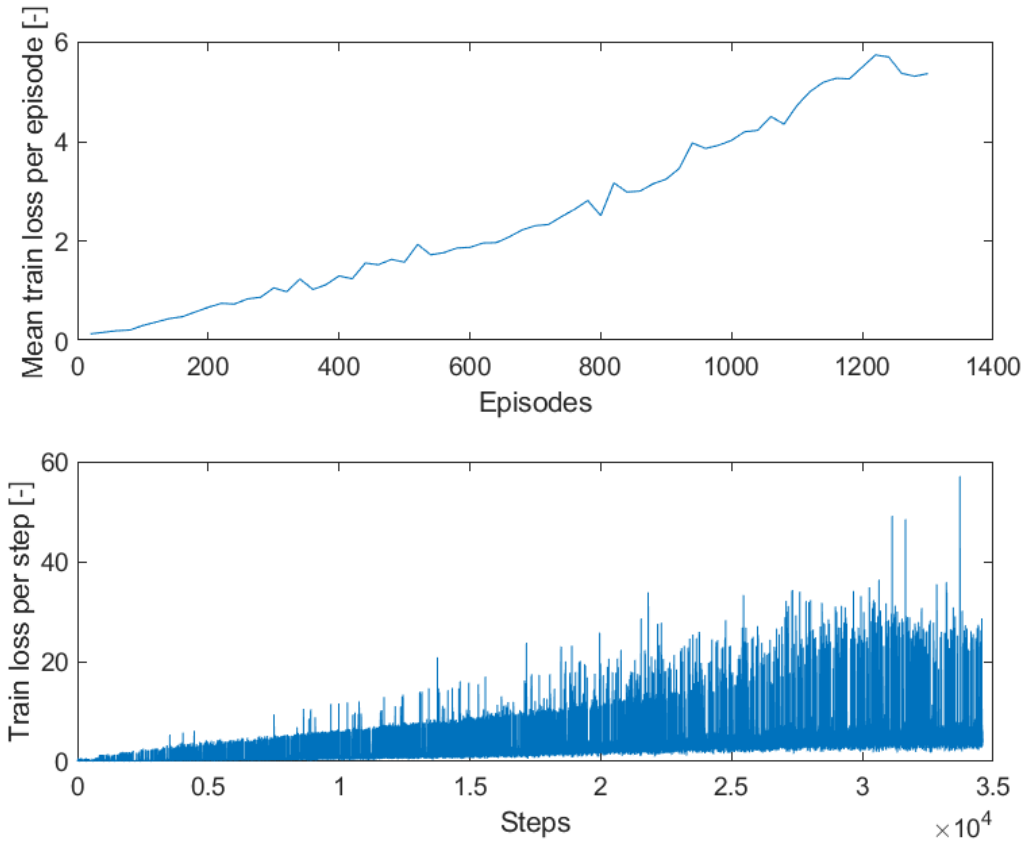

Figura 6.14: Train loss-clust7

In questa circostanza non vengono mostrati i risultati numerici di SOC ed FC poiché l'ultimo episodio di test si conclude in anticipo. L'agente utilizzato appare piuttosto instabile, difatti il *Cumulative reward* mostra un andamento altalenante, in quanto non tutti gli episodi giungono a conclusione. Perciò in questa circostanza i parametri di *training* utilizzati non sembrano generalizzarsi bene al variare delle condizioni di guida. Inoltre, non viene individuata una *policy* ottimale che riduca l'errore. Anche in questo caso l'agente predilige un innalzamento del SOC a scapito di consumi di combustibile più elevati, in quanto il FC non impatta in maniera importante sulla funzione di *reward*. La strategia di controllo che massimizza il *Cumulative reward* non è quella che minimizza i consumi, come avveniva sul WLTP, proprio perché in questa circostanza la potenza media di trazione risulta essere inferiore.

6.3 Clust11

Il ciclo clust11 è più breve rispetto ai cicli precedenti ma più complesso da gestire per l'agente. In particolare ha una durata di 653 s e si caratterizza di un punto che fa da attrattore, ossia in corrispondenza dei 100 s della missione di guida si ha un'accelerazione molto spiccata, e quindi utilizzando una *reward function FC-oriented* l'agente potrebbe aver bisogno di tutta la banda di SOC a disposizione per poter utilizzare il motore elettrico.

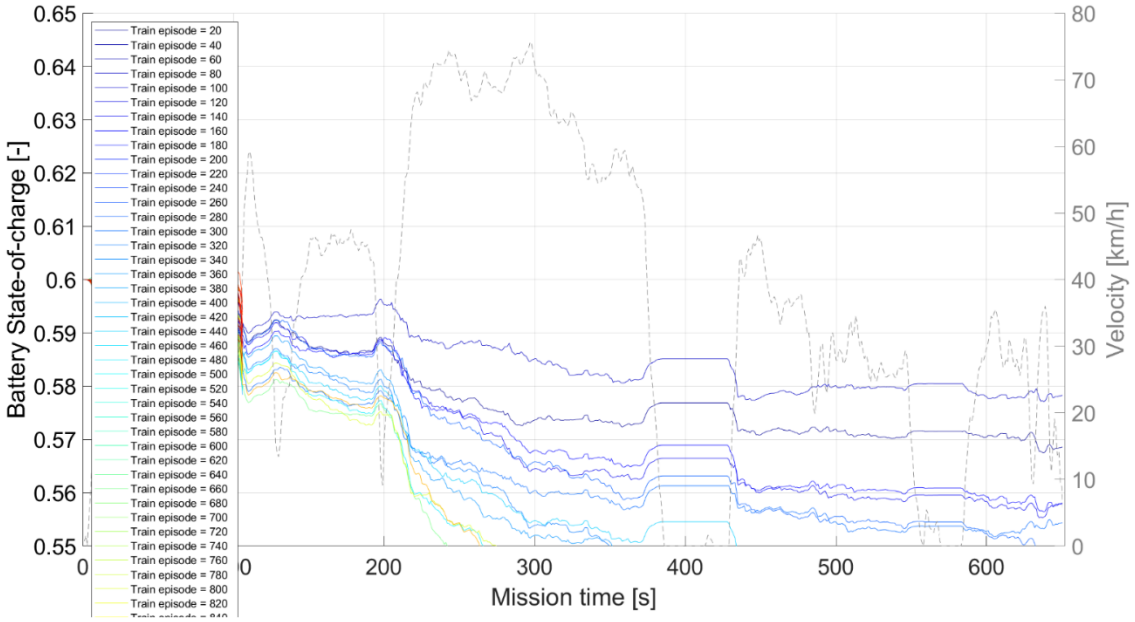

Figura 6.15: SOC episodi di train-clust11

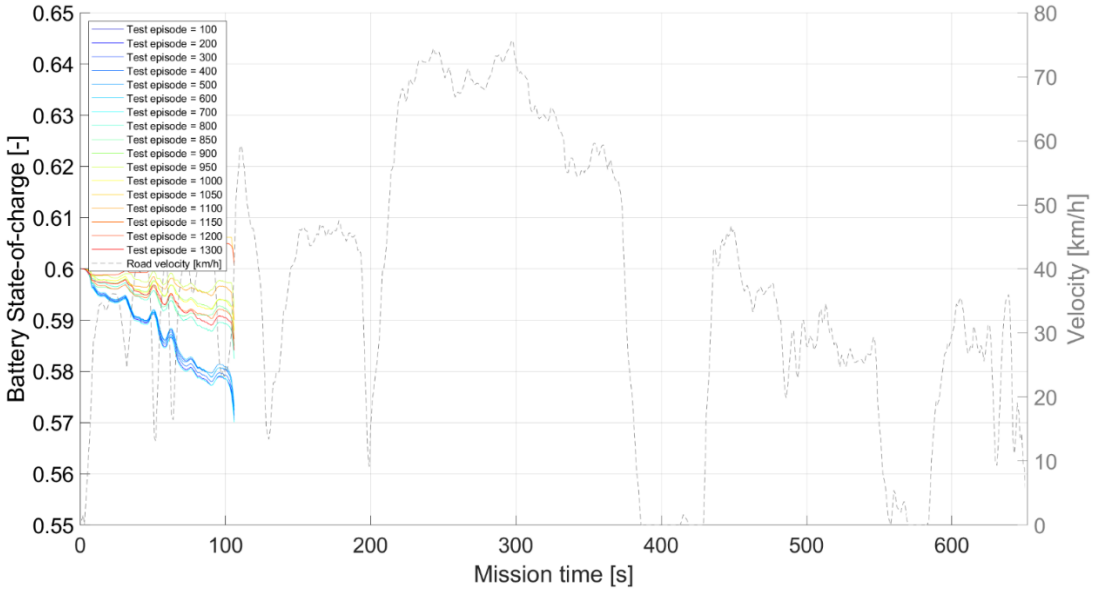

Figura 6.16: SOC episodi di test-clust11

Analizzando i risultati di Figura 6.15-6.16 è possibile notare come gran parte degli episodi di *training* e di *testing* non giungano a conclusione. In corrispondenza del punto attrattore viene selezionata un'azione *unfisible*, provocando così la terminazione dell'episodio. In particolare l'agente giunge in corrispondenza di uno stato s_t dal quale non è possibile intraprendere un'azione fisica fattibile, quindi non riesce a battere la *reward function FC-oriented* precedentemente definita.

6.4 Clust12

Il ciclo clust12 ha una lunghezza complessiva di 475 s ed un picco di velocità di 55 km/h. Tale ciclo risulta essere più aggressivo in termini di accelerazioni rispetto ai cicli clust4 e clust7, ma è caratterizzato da una potenza di trazione media inferiore.

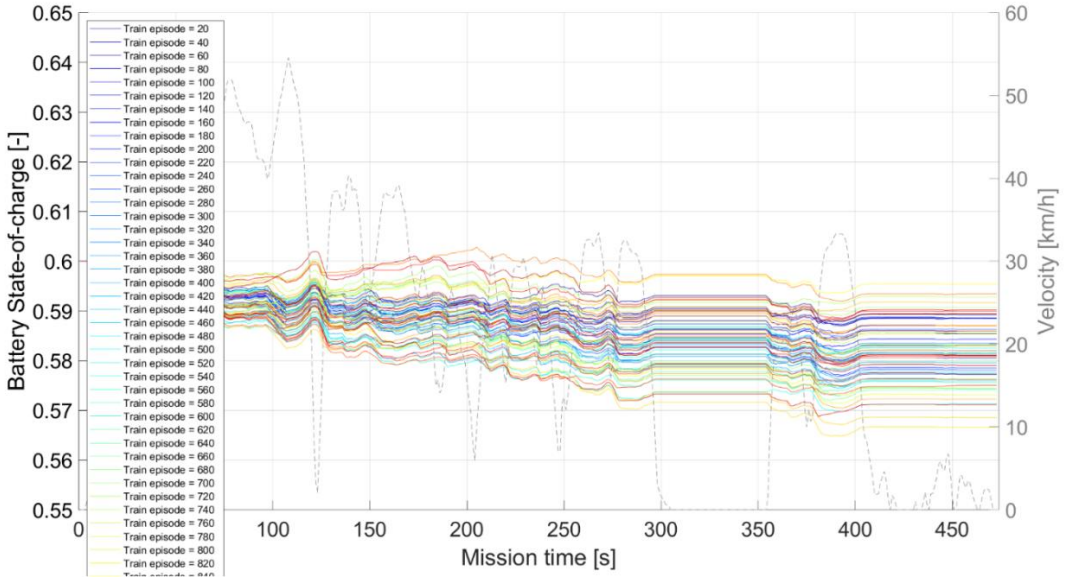

Figura 6.17: SOC episodi di train-clust12

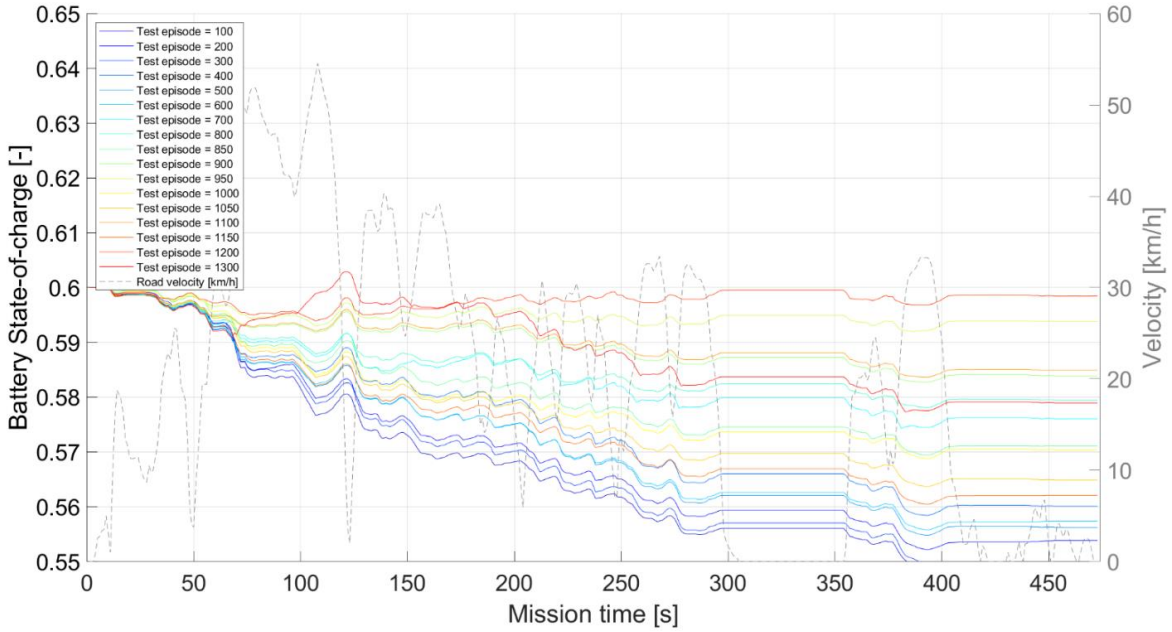

Figura 6.18: SOC episodi di test-clust12

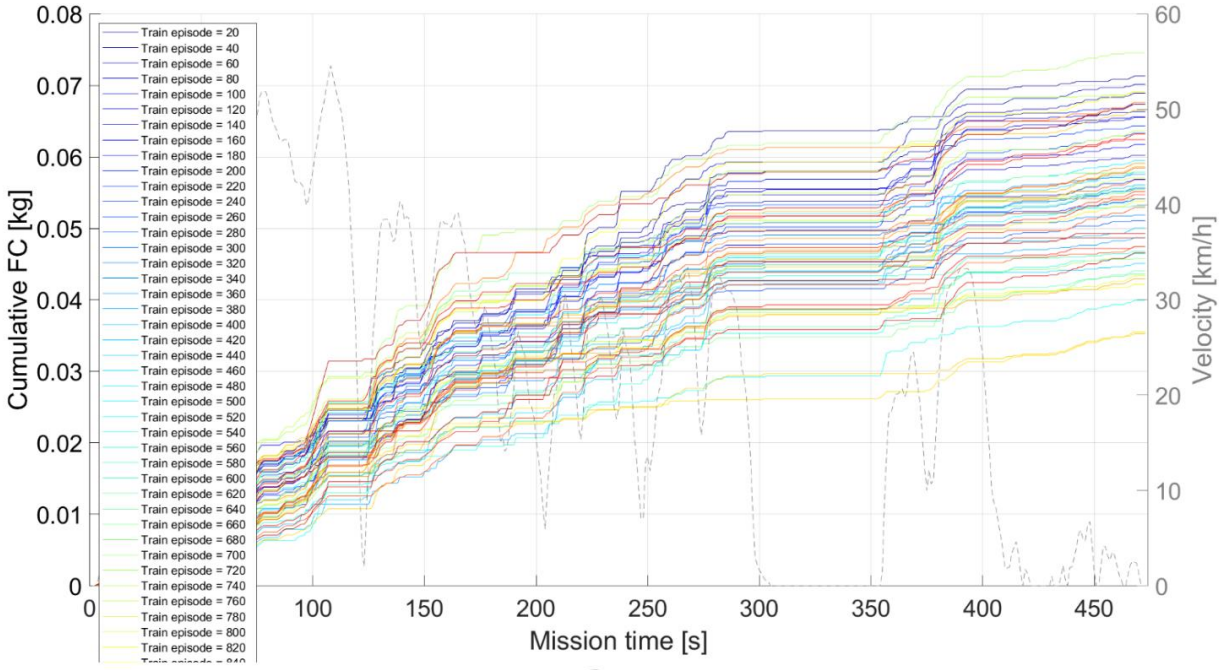

Figura 6.19: Cumulative FC episodi di train-clust12

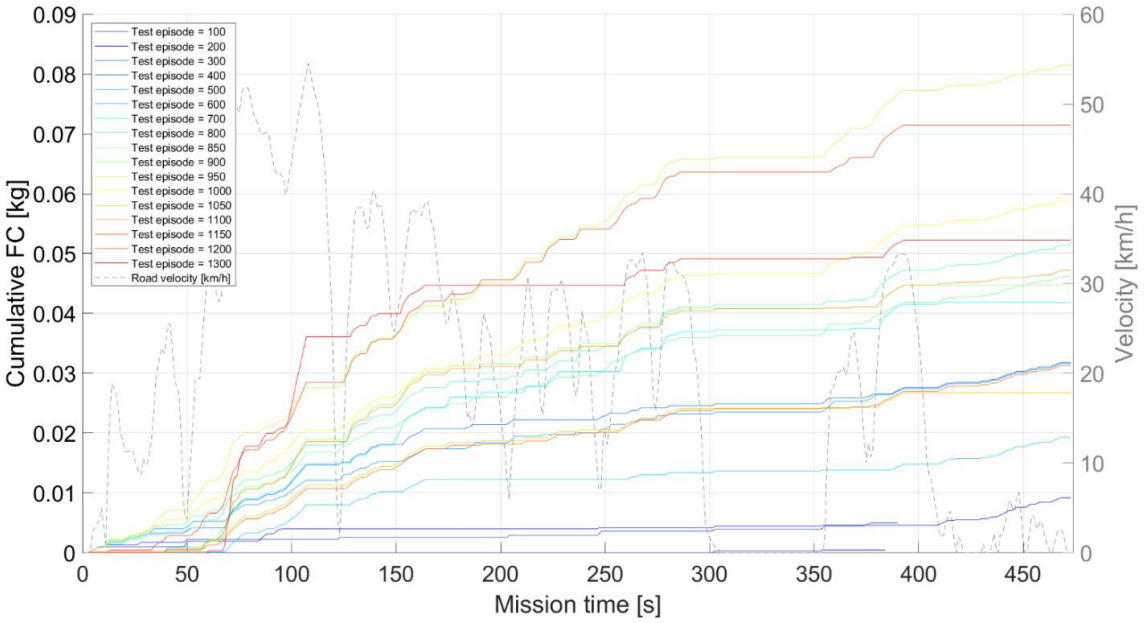

Figura 6.20: Cumulative FC episodi di test-clust12

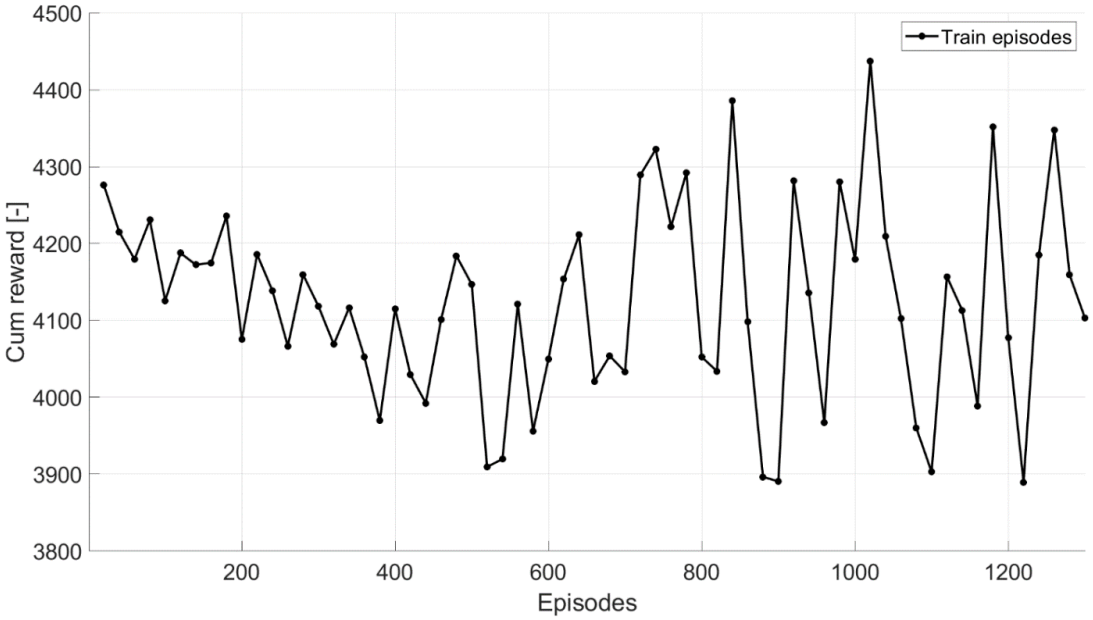

Figura 6.21: Discounted return episodi di train-clust12

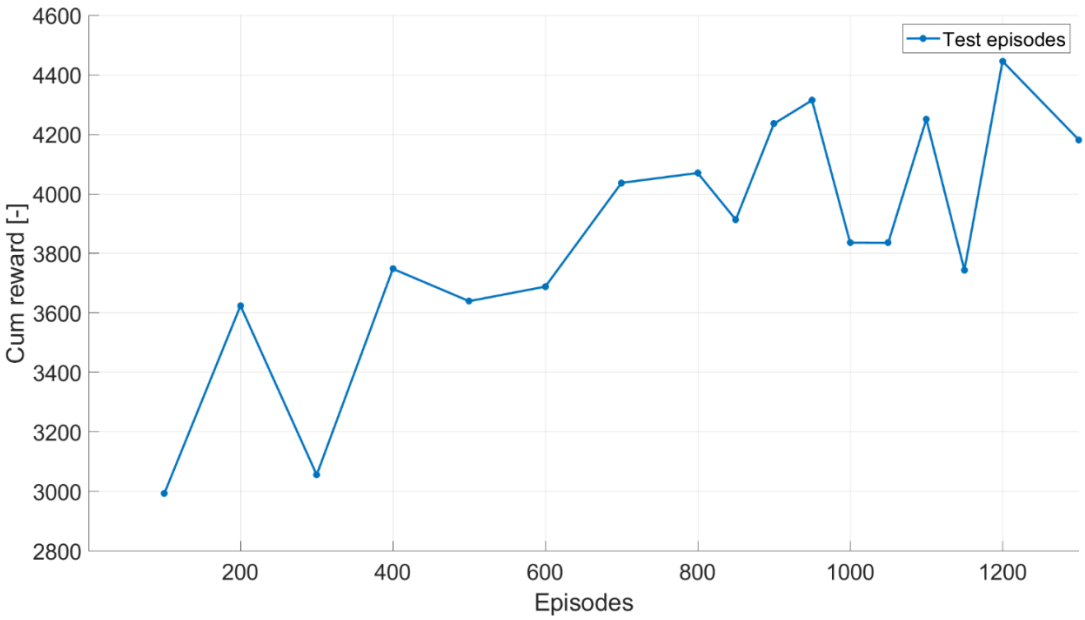

Figura 6.22: Discounted return episodi di test-clust12

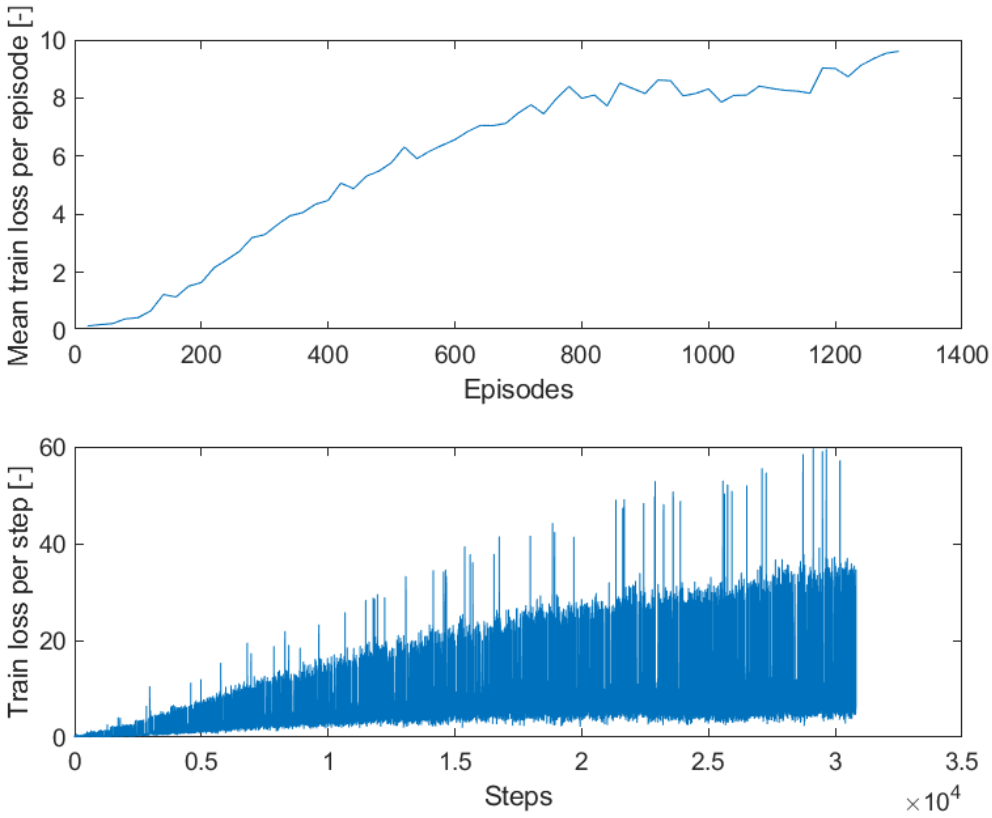

Figura 6.23: Train loss-clust12

Consumo di carburante in kg	0.05
SOC iniziale	0.6
SOC finale	0.5789
Cumulative reward	1547

Tabella 6.3: Risultati su ciclo clust12 con agente DQN

Anche in questo caso il consumo di combustibile ha un peso inferiore rispetto allo stato di carica della batteria sulla *reward function*, in quanto l'agente predilige una minimizzazione del ΔSOC con conseguente aumento del Cumulative FC. La strategia di controllo individuata porta alla massimizzazione delle ricompense, anche se si ha un andamento crescente della *train loss.* Sommariamente l'allenamento appare piuttosto stabile, in quanto l'oscillazione del *Cumulative reward* tra i vari episodi risulta essere abbastanza contenuta.

7. Conclusioni

In questo lavoro di tesi è stato condotto uno studio parametrico di un algoritmo di *Deep Q-Network* in un'applicazione di gestione energetica di un veicolo ibrido elettrico, poiché l'efficacia del *Reinforcement Learning* dipende in gran parte dalla selezione ottimizzata dei vari parametri di *training*. Sono state valutate le influenze dei principali parametri sulle performance di un agente DQN per lo sviluppo dell'EMS, in particolare: *learning starts*, *target update frequence*, *replay memory size*, *batch size*, *learning rate*, scelta degli Stati e loro normalizzazione, *reward function*, esplorazione e sfruttamento. L'aggiunta di più Stati ha permesso di caratterizzare meglio l'ambiente, garantendo quindi delle migliori performace dell'agente in termini di minimizzazione dei consumi di combustibile, tramite l'introduzione di una *reward function FC-oriented*. Una corretta riduzione del *learning starts* e del *target upadate frequence* ha permesso di incrementare la velocità di addestramento, mentre un'opportuna calibrazione del *replay memory size* ha garantito un corretto aggiornamento delle esperienze accumulate, in modo tale da regolarizzare il *training* dell'agente. Una riduzione del *learning rate* ha prodotto un leggero allungamento dei tempi di allenamento ma ha permesso di individuare un set di pesi sub-ottimali per la risoluzione del problema, scongiurando l'esplosione del gradiente e conseguente divergenza dei *Q-values* e della *train loss*. Per individuare un corretto *trade-off* tra esplorazione e sfruttamento è stata utilizzata una ε-greedy *policy*. Tuttavia, senza un'adeguata esplorazione non è possibile individuare una strategia di controllo ottimale, in quanto l'algoritmo rimarrà intrappolato in massimi locali. Sono stati valutati gli effetti del parametro di esplorazione ε, facendolo decrescere fino al suo valore minimo in corrispondenza del 10%, 50% e 75% degli episodi complessivi di addestramento. Al crescere dell'esplorazione si è assistito ad un miglioramento delle prestazioni e quindi ad una minimizzazione dei consumi di combustibile. Infine è stato valutato l'effetto del *batch size*, ossia il numero di esperienze su cui andiamo ad addestrare la rete neurale ad ogni *time step*. In particolare un aumento della taglia del *batch* non ha prodotto dei miglioramenti apprezzabili, appesantendo però la simulazione dal punto di vista computazionale. Una corretta calibrazione dei principali parametri di *training* su ciclo WLTP ha permesso di ottenere dei buoni risultati dal punto di vista fisico, mentre dal punto di vista matematico rimane aperta la questione sulla crescita della *train loss*.

Dopo aver ottenuto una soluzione parametrica sub-ottimale al problema sul ciclo WLTP è stata valutata la robustezza dell'agente DQN, ossia sono state valutate le sue performance al variare della missione di guida (cicli clust). Cambiando l'*environment* l'agente ha mostrato una scarsa adattabilità, perciò le configurazioni utilizzate sul WLTP non si estendono sempre correttamente al variare delle condizioni di guida. In particolare la *reward function* non sembra generalizzarsi bene al variare delle caratteristiche del ciclo guida. Poiché i cicli clust risultano essere meno gravosi del WLTP, ossia la potenza media di trazione è inferiore, il consumo di combustibile sembra impattare meno sulla funzione di *reward*, perciò l'agente predilige un maggior controllo e contenimento del SOC a scapito di maggiori consumi di carburante.

Come proseguimento naturale del lavoro potrebbe essere condotto uno studio approfondito della *loss function*, in modo tale che essa giunga a convergenza, permettendo quindi di migliore la robustezza dell'agente DQN e le sue performance. Inoltre, potrebbe essere implementata una *reward function* che si generalizzi meglio al variare della missione di guida, ottenendo così una strategia di ottimizzazione che si possa adattare al variare dell'*environment*.

Tale studio ha lo scopo di evidenziare le possibili potenzialità di questo strumento per risolvere un problema di EMS per un HEV. Questa analisi parametrica può essere ritenuta valida anche su veicoli ibridi elettrici con diverse specifiche e diverse architetture. Ovviamente i risparmi di carburante non saranno gli stessi ottenuti in questa trattazione, tuttavia si dovrebbe riscontrare un'analogia di comportamento al variare dei parametri presi in esame.

Bibliografia:

[1] H. Tan, H. Zhang, J. Peng, Z. Jiang, and Y. Wu, “Energy management of hybrid electric bus based on deep reinforcement learning in continuous state and action space,” Energy Convers. Manag., vol. 195, no. January, pp. 548–560, 2019, doi: 10.1016/j.enconman.2019.05.038

[2] Y. Wu, H. Tan, J. Peng, H. Zhang, and H. He, “Deep reinforcement learning of energy management with continuous control strategy and traffic information for a series-parallel plugin hybrid electric bus,” Appl. Energy, vol. 247, no. March, pp. 454–466, 2019, doi: 10.1016/j.apenergy.2019.04.021

[3] J. Wu, H. He, J. Peng, Y. Li, and Z. Li, “Continuous reinforcement learning of energy management with deep Q network for a power split hybrid electric bus,” Appl. Energy, vol. 222, no. January, pp. 799–811, 2018, doi: 10.1016/j.apenergy.2018.03.104

[4] X. Han, H. He, J. Wu, J. Peng, and Y. Li, “Energy management based on reinforcement learning with double deep Q-learning for a hybrid electric tracked vehicle,” Appl. Energy, vol. 254, no. August, p. 113708, 2019, doi: 10.1016/j.apenergy.2019.113708

[5] B. Xu, D. Rathod, D. Zhang, A. Yebi, X. Zhang, X. Li, Z. Filipi, “Parametric study on reinforcement learning optimized energy management strategy for a hybrid electric vehicle,” Appl. Energy, vol. 259, no. August 2019, p. 114200, 2020, doi: 10.1016/j.apenergy.2019.114200

[6] Z. Chen, H. Hu, Y. Wu, Y. Zhang, G. Li, and Y. Liu, “Stochastic model predictive control for energy management of power-split plug-in hybrid electric vehicles based on reinforcement learning,” Energy, vol. 211, p. 118931, 2020, doi: 10.1016/j.energy.2020.118931

[7] Y. Hu, W. Li, K. Xu, T. Zahid, F. Qin, and C. Li, “Energy management strategy for a hybrid electric vehicle based on deep reinforcement learning,” Appl. Sci., vol. 8, no. 2, 2018, doi: 10.3390/app8020187

[8] X. Wang, C. Wu, J. Xue, Z. Chen, “A Method of Personalized Driving Decision for Smart Car Based on Deep Reinforcement Learning”

[9] R. Lian, J. Peng, Y. Wu, H. Tan, and H. Zhang, “Rule-interposing deep reinforcement learning based energy management strategy for power-split hybrid electric vehicle,” Energy, vol. 197, p. 117297, 2020, doi: 10.1016/j.energy.2020.117297

[10] Teng Liu, Xiaosong Hu, Senior Member, IEEE, Shengbo Eben Li, and Dongpu Cao, “ Reinforcement Learning Optimized Look-Ahead Energy Management of a Parallel Hybrid Electric Vehicle”

[11] H. Lee, C. Song, N. Kim, “Comparative Analysis of Energy Management Strategies for HEV: Dynamic Programming and Reinforcement Learning”

[12] T. Liu, Y. Zou, D. Liu and F. Sun, “Reinforcement Learning–Based Energy Management Strategy for a Hybrid Electric Tracked Vehicle”

[13] Z. Kong, Y. Zou, Teng Liu, “Implementation of real-time energy management strategy based on reinforcement learning for hybrid electric vehicles and simulation validation”

[14] E. Spessa, “Controllo delle emissioni inquinanti”. Politecnico di Torino

[15] Matteo Acquarone, https://webthesis.biblio.polito.it/19497/1/tesi.pdf

[16] Michele Buzzoni, https://amslaurea.unibo.it/15539/1/tesiBuzzoniMichele.pdf

[17] Marco Conciatori, https://amslaurea.unibo.it/19132/1/Tesi%20Upload.pdf

[18] Diego pergolini, https://amslaurea.unibo.it/19415/1/Tesi_Diego_Pergolini.pdf

[19] Rete neurale artificiale,https://it.wikipedia.org/wiki/Rete_neurale_artificiale

[20] Rete neurale artificiale, https://www.spindox.it/it/blog/ml1-reti-neurali-demistificate/#

[21] Deep Q-Network, https://ichi.pro/it/apprendimento-per-rinforzo-spiegato-visivamente-parte-5-deep-q-networks-passo-dopo-passo-99312916332363
[22] A. Mastropietro, L.Sorrentino, “Reinforcement Learning for Hybrid Electric Vehicles”
[23] Denerg-Politecnico di Torino, “HEVbox User Manual”
[24] Commissione europea, https://ec.europa.eu/info/policies/climate-action_it