Traffic Control With AI

Capaldo Gennaro, Senatore Carmela Pia

Gennaio 2025

Abstract

Questo progetto analizza e confronta l'efficacia di tre algoritmi di Reinforcement Learning (RL): Deep Q-Learning, Q-Learning e SARSA, applicati alla gestione dei semafori per decongestionare il traffico in un'intersezione stradale simulata. Gli algoritmi sono stati valutati fra di loro, con il semaforo a fasi e infine con il semaforo che cambiava il suo stato in base a quanti veicoli erano in attesa su una direzione. La simulazione è stata realizzata utilizzando una mappa di un incrocio reperita online con due intersezioni, scritta in Python e opportunamente modificata per adattarsi agli esperimenti. I risultati mostrano che il Deep Q-Learning è l'approccio più efficace, garantendo un miglioramento nel tempo medio di attesa dei veicoli rispetto agli altri metodi implementati.

1 Metodologia

1.1 Descrizione dell'ambiente

Per valutare le prestazioni degli algoritmi di Reinforcement Learning (RL), è stato utilizzato un ambiente simulato rappresentante un'intersezione stradale urbana. La mappa rappresenta un incrocio a quattro vie in cui vengono generati i veicoli. Nel sistema di simulazione i veicoli sono modellati come degli oggetti dinamici che interagiscono con l'ambiente circostante e reagiscono alle condizioni del traffico e del meteo. Ogni veicolo è caratterizzato da proprietà fisiche quali la loro dimensione (standard), velocità, accelerazione e decelerazione (dinamiche) che definiscono il suo comportamento. La velocità massima è fissata a 16.6 m/s, l'accelerazione pari a 1.44 m/s e una decelerazione massima di 4.61 m/s. Questi parametri sono utilizzati per calcolare il movimento e l'interazione con altri veicoli presenti in strada. I veicoli seguono un percorso predefinito rappresentato dalla sequenza di strade, ma il loro comportamento è dinamico. Le auto, infatti, tengono conto della distanza di sicurezza rispetto al veicolo che la precede e regolano l'accelerazione di conseguenza. Inoltre, quando un veicolo è fermo, viene monitorato il suo tempo di attesa totale, aggiornato durante ogni arresto. Un altro aspetto importante è l'adattabilità dei veicoli alle condizioni meteorologiche. In caso di pioggia, neve o nebbia, i veicoli riducono la velocità massima e aumentano la distanza di sicurezza, riflettendo le condizioni reali di

guida in situazioni climatiche avverse. Ad esempio, in condizioni di neve, la velocità massima viene ridotta della metà e la distanza di sicurezza aumenta a 15 metri per garantire una maggiore sicurezza. Il flusso di veicoli è regolato da 4 semafori situati vicino all'intersezione. Ogni semaforo è associato ad un insieme di strade, suddivise in direzioni alternate (verticale ed orizzontale). Il comportamento dei semafori viene regolato da un ciclo predefinito che specifica lo stato (verde, giallo e rosso) di ciascuna direzione. Il ciclo dei semafori è espresso tramite una lista di tuple booleane, in cui ogni elemento della lista definisce quali direzioni sono verdi in quel momento. Ad esempio, se la direzione verticale è verde, anche la strada opposta nella stessa direzione lo sarà, mentre le strade perpendicolari saranno rosse. Questo schema garantisce un'alternanza regolare tra le direzioni, consentendo ai veicoli di attraversare l'incrocio senza collisioni. Oltre al controllo del flusso, i semafori influenzano anche il comportamento dei veicoli nelle vicinanze. I veicoli rallentano quando si avvicinano a un semaforo in base alla distanza di rallentamento definita e a un fattore di rallentamento. Se un veicolo si trova entro la distanza di arresto e il semaforo è rosso, il veicolo si ferma completamente. Tuttavia se il veicolo si trova entro la distanza di arresto e in quel momento il semaforo cambia il suo stato (passare da verde a rosso), il veicolo potrebbe attraversare con il rosso effettuando una collisione.

2 Metriche di valutazione

2.1 Tempo medio di attesa

L'obiettivo principale di questo progetto è stato quello di simulare uno scenario di traffico e cercare di capire quale fosse la strategia più adatta per decongestionare più velocemente le code di auto presenti sulle strade valutando diverse alternative. A tal proposito è stata scelta come metrica di valutazione dominante il tempo medio di attesa dei veicoli. Ridurre il tempo medio di attesa significa che il traffico è più fluido, i semafori sono meglio sincronizzati, e i veicoli affrontano meno ritardi. Ogni veicolo registra durante il suo percorso il tempo in cui è rimasto fermo fino alla sua uscita dall'intersezione. La funzione che calcola il tempo di attesa medio dell'episodio è divisa in due parti: 1. I veicoli che hanno terminato il loro percorso (usciti dall'intersezione) contribuiscono al calcolo con il loro tempo di attesa totale registrato durante il viaggio. 2. I veicoli attualmente presenti sulla mappa contribuiscono al calcolo con i loro tempi di attesa accumulati fino al momento attuale. Il risultato finale è la somma delle due componenti che definisce il tempo medio di attesa globale. E stata scelta questa metrica in quanto l'episodio potrebbe non terminare con l'uscita di tutti i veicoli generati dall'intersezione, ma potrebbe terminare anche con una collisione. In questo caso i veicoli presenti sulla mappa (non ancora usciti dall'intersezione) non contribuirebbero al tempo di attesa globale e di conseguenza si avrebbe una visione parziale dello stato del traffico.

2.2 Semaforo a fasi

Si implementa un semaforo classico a fasi con un ciclo fisso. Questo tipo di controllo dei semafori è utilizzato come baseline per confrontare le prestazioni degli algoritmi di reinforcement learning in termini di tempo medio di attesa globale. Questa logica replica il comportamento dei semafori tradizionali che alternano rosso e verde in modo predeterminato, senza adattarsi alle condizioni di traffico. Tale algoritmo implementa una variabile che controlla la durata del ciclo del semaforo, ovvero il tempo che intercorre prima che il semaforo cambi stato. Quando il tempo di un determinato stato del semaforo supera la soglia predefinita, il semaforo cambia il suo stato. Ripete questo ciclo finchè non vengono fatti passare tutti i veicoli generati.

2.3 Coda più lunga

Un'altra baseline di confronto riguarda la strategia di gestione del traffico basata sulla lunghezza delle code presenti sulle strade. Questo approccio rappresenta una baseline più sofisticata rispetto al semaforo a fasi, poiché cerca di ottimizzare il traffico tenendo conto del numero di veicoli presenti nelle diverse direzioni. Il funzionamento parte da un controllo del tempo trascorso dall'ultimo cambio di stato del semaforo. Si utilizza un intervallo di tempo minimo predefinito, indicato come t (ad esempio 30 secondi), che serve a evitare che il semaforo cambi troppo frequentemente. Se non è trascorso questo tempo, il semaforo non cambia stato. Questo meccanismo impedisce che i veicoli siano soggetti a cambi repentini, garantendo un minimo di stabilità. Una volta verificato che il tempo minimo è trascorso, il sistema analizza la lunghezza delle code nelle due direzioni principali dell'incrocio. La decisione si basa su un semplice criterio: se la direzione che ha attualmente il semaforo rosso ha una coda più lunga rispetto all'altra, allora il semaforo cambia stato, dando il verde alla direzione con il maggior numero di veicoli. Questo processo si applica anche in senso opposto: se la direzione che ha attualmente il semaforo verde ha una coda più corta rispetto alla direzione opposta, il semaforo viene commutato per dare priorità a quest'ultima.

3 Implementazione degli algoritmi di RL

3.1 Definizione dello stato

La definizione dello stato è cruciale per modellare l'interazione tra l'agente e l'ambiente. Lo stato, infatti, rappresenta l'informazione necessaria che l'agente utilizza per poter prendere decisioni ottimali. A tal proposito è stato definito lo stato come una tupla che contiene i seguenti elementi:

- Stato del segnale del traffico: Rappresenta la fase corrente del ciclo semaforico (verde per una direzione e rosso per l'altra);
- Numero di veicoli presenti sulla direzione verticale;

- Numero di veicoli presenti sulla direzione orizzontale;
- Indicazione di incrocio vuoto: un valore booleano che indica se l'incrocio è libero da veicoli;
- Condizioni meteorologiche: una rappresentazione del meteo corrente che include opzioni come "Soleggiato", "Pioggia", "Neve", "Nebbia".

3.2 Penalità e Ricompense

Nel modello implementato, le penalità e le ricompense sono state definite per guidare l'agente verso comportamenti ottimali nella gestione del traffico.

3.2.1 Penalità

Il modello considera penalità basate sul tempo medio di attesa dei veicoli come indicatore dell'efficienza del sistema. Per ogni ciclo di simulazione, il tempo medio di attesa dei veicoli presenti sulla mappa viene calcolato e utilizzato per assegnare una penalità proporzionale. Questo approccio incoraggia l'agente a ridurre i tempi di fermo, promuovendo una gestione dinamica ed efficace delle code. È stato scelto di includere nel calcolo esclusivamente il tempo medio di attesa dei veicoli ancora presenti sulla mappa, ignorando quelli che hanno già lasciato l'intersezione. Questa decisione consente di fornire un feedback diretto all'algoritmo rispetto alla situazione attuale, evitando di penalizzare eccessivamente l'agente per azioni sbagliate compiute in precedenza (ad esempio, veicoli che hanno aspettato troppo prima di uscire dall'intersezione).

3.2.2 Ricompense

Oltre al tempo medio di attesa, la funzione di ricompensa del modello tiene conto del cambiamento nel flusso di traffico, misurato attraverso il numero di veicoli presenti sulle strade orizzontale e verticale. Ogni episodio parte da una configurazione iniziale di traffico che evolve nel tempo in base alle azioni intraprese dal modello, e la ricompensa viene assegnata in funzione della variazione nel numero di veicoli.

Il sistema monitora il numero di veicoli sulle due strade principali in ogni stato. La funzione di ricompensa si basa sulla differenza tra il numero di veicoli sulle strade in entrata dopo l'azione precedente e il numero di veicoli in entrata dopo l'azione corrente. Le ricompense sono calcolate come segue:

- Aumento del numero di veicoli: La ricompensa è negativa, poiché indica un peggioramento della congestione.
- Diminuzione del numero di veicoli: La ricompensa è positiva, premiando l'agente per aver migliorato la fluidità del traffico.
- Assenza di cambiamenti significativi: La ricompensa tende a zero, segnalando una stabilizzazione del traffico.

Questa metodologia permette al modello di adattarsi in modo flessibile, premiando azioni che riducono la congestione in tempo reale e incentivando la gestione dinamica del traffico.

4 Q-Learning

Il **Q-Learning** è un algoritmo di *apprendimento per rinforzo* (*Reinforcement Learning*), utilizzato per risolvere problemi di decisione sequenziale modellati come *Processi di Decisione di Markov* (MDP). Questo metodo consente a un agente di apprendere una politica ottimale che massimizza la ricompensa cumulativa attesa a lungo termine, esplorando l'ambiente e interagendo con esso in maniera iterativa.

Grazie a questa interazione continua con l'ambiente, l'agente sperimenta varie situazioni definite come **stati**. Mentre si trova in uno stato, l'agente può scegliere tra una serie di azioni consentite, ciascuna delle quali può portare a una ricompensa (o penalità) diversa. Nel tempo, attraverso il processo di apprendimento, l'agente impara a massimizzare queste ricompense per comportarsi in modo ottimale in qualsiasi stato in cui si trovi.

4.1 Obiettivo

Il **Q-Learning** è un algoritmo *off-policy*, il che significa che l'aggiornamento dei valori Q non dipende direttamente dalle azioni intraprese dall'agente durante l'esplorazione. In altre parole, si assume che l'agente esegua sempre l'azione migliore possibile, anche se durante l'addestramento può agire in modo esplorativo. Questo approccio consente all'algoritmo di concentrarsi maggiormente sullo sfruttamento di strategie ottimali a lungo termine, senza legarsi alla policy usata durante la fase di esplorazione.

L'obiettivo principale del Q-Learning è determinare una funzione di valore $Q(s, a)$, che rappresenta il valore atteso dell'azione a, presa in uno stato s, considerando tutte le ricompense future che l'agente può ricevere. Matematicamente, il valore della funzione è definito come:

$$Q(s,a) = E\left[\sum_{t=0}^{\infty} \gamma^t r_{t+1} \,\middle|\, s_t = s, a_t = a\right],$$

dove:

- s è lo stato corrente dell'ambiente,
- a è l'azione scelta dall'agente nello stato s,
- r_t è la ricompensa istantanea ricevuta al tempo t,
- $\gamma \in [0, 1]$ è il *fattore di sconto*, che riduce il peso delle ricompense future per favorire quelle più immediate.

Il termine γ gioca un ruolo fondamentale: un valore γ vicino a zero rende l'agente "miope," concentrandosi quasi esclusivamente sulle ricompense immediate. Al contrario, un valore γ vicino a 1 incoraggia l'agente a prendere decisioni che ottimizzano le ricompense a lungo termine.

4.2 La Q-Table

La funzione $Q(s, a)$ è generalmente rappresentata tramite una struttura dati chiamata **Q-Table**. Si tratta di una tabella bidimensionale in cui:

- Le righe corrispondono agli **stati** dell'ambiente (s),
- Le colonne rappresentano le **azioni** possibili (a).

Ogni elemento della Q-Table, indicato come $Q(s, a)$, contiene il valore atteso dell'azione a nello stato s. Durante il processo di apprendimento, i valori nella Q-Table vengono aggiornati iterativamente secondo la seguente formula:

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left[r + \gamma \max_{a'} Q(s', a') - Q(s, a) \right],$$

dove:

- $\alpha \in (0, 1]$ è il *tasso di apprendimento*, che determina quanto peso dare alla nuova informazione rispetto al valore esistente,
- r è la ricompensa immediata ricevuta dopo aver eseguito l'azione a nello stato s,
- $\max_{a'} Q(s', a')$ è il valore massimo atteso per il prossimo stato s',
- $Q(s, a)$ è il valore corrente dell'azione a nello stato s.

All'inizio dell'addestramento, la Q-Table è inizializzata con valori arbitrari (spesso zero). L'agente quindi esplora l'ambiente aggiornando gradualmente i valori nella tabella man mano che raccoglie informazioni sulle ricompense associate alle diverse azioni.

Una volta completato l'addestramento, la Q-Table contiene una stima dei valori ottimali $Q^*(s, a)$ per ogni coppia stato-azione. La politica ottimale π^* può quindi essere derivata selezionando, in ogni stato, l'azione con il valore $Q(s, a)$ massimo:

$$\pi^*(s) = \arg\max_a Q(s, a).$$

La Q-Table è semplice da implementare ed efficace per ambienti con un numero limitato di stati e azioni. Tuttavia, il suo utilizzo diventa impraticabile per ambienti con spazi di stato molto grandi o continui, a causa dell'esplosione combinatoria delle dimensioni della tabella.

4.3 Esplorazione ed esploitazione

Un aspetto cruciale del Q-Learning è il bilanciamento tra **esplorazione** ed **esploitazione**:

- **Esplorazione**: l'agente esegue azioni casuali per acquisire nuove informazioni sull'ambiente e aggiornare la funzione $Q(s, a)$ con dati non ancora conosciuti.
- **Esploitazione**: l'agente esegue l'azione con il valore $Q(s, a)$ massimo noto per massimizzare immediatamente la ricompensa attesa.

Questo bilanciamento è spesso implementato attraverso una strategia ϵ-*greedy*, in cui:

- Con probabilità ϵ, l'agente sceglie un'azione casuale (esplorazione),
- Con probabilità $1 - \epsilon$, l'agente seleziona l'azione ottimale conosciuta (esploitazione).

Il valore di ϵ può essere ridotto gradualmente nel corso dell'addestramento, per favorire l'esplorazione iniziale e lo sfruttamento delle conoscenze apprese nelle fasi successive.

4.4 Algoritmo Q-Learning

L'algoritmo completo può essere sintetizzato come segue:

Algorithm 1 Q-Learning

Inizializza $Q(S, A)$ arbitrariamente per tutti gli stati S e azioni A
for ogni episodio **do**
    Inizializza lo stato iniziale S
    while S non è uno stato terminale **do**
        Seleziona un'azione A da S utilizzando una politica π (es. ε-greedy)
        Esegui l'azione A, osserva la ricompensa R e il nuovo stato S'
        Aggiorna $Q(S, A)$ utilizzando la formula:

$$Q(S, A) \leftarrow Q(S, A) + \alpha \left[R + \gamma \max_{a'} Q(S', a') - Q(S, A) \right]$$

        Aggiorna lo stato corrente $S \leftarrow S'$
    end while
end for

4.5 Convergenza

L'algoritmo di Q-Learning è garantito per convergere ai valori ottimali $Q^*(s, a)$, a patto che:

- Ogni coppia stato-azione (s, a) venga visitata infinite volte,
- Il tasso di apprendimento α decresca gradualmente nel tempo, rispettando $\sum_t \alpha_t = \infty$ e $\sum_t \alpha_t^2 < \infty$,
- L'ambiente segua la proprietà di Markov.

Una volta raggiunta la convergenza, la funzione $Q^*(s, a)$ definisce una politica ottimale $\pi^*(s)$, che massimizza la ricompensa attesa.

5 Implementazione Q-Learning

5.1 Descrizione del Modello

Il modello utilizza un agente di *Q-Learning* implementato attraverso una tabella Q dinamica. L'agente interagisce con un ambiente simulato, rappresentato dalla classe `ExMeteo`, per apprendere una politica ottimale che minimizzi i tempi di attesa medi e renda scorrevole e fluido il traffico accumulato. Gli iper-parametri utilizzati sono:

- **Tasso di apprendimento (α)**: 0.3
- **Tasso di esplorazione (ϵ)**: 0.1
- **Fattore di sconto (γ)**: 0.9

5.2 Inizializzazione

L'ambiente `ExMeteo` è stato inizializzato per simulare l'intersezione stradale contententi i due semafori. Le azioni che possono essere utilizzate dai semafori sono:

- 0 : Verde per la direzione orizzontale e rosso per quella verticale;
- 1 : Rosso per la direzione orizzontale e verde per quella verticale.

5.3 Fase di Training

L'agente è stato addestrato per un totale di 300.000 episodi (*n_train_episodes*), durante i quali:

1. L'agente ha interagito con l'ambiente, scegliendo azioni o casuali o azioni con il valore Q più alto.
2. La Q-table è stata aggiornata in base alla ricompensa ricevuta e al massimo valore Q futuro.
3. Il tempo medio di attesa e il punteggio complessivo sono stati registrati per ogni episodio.

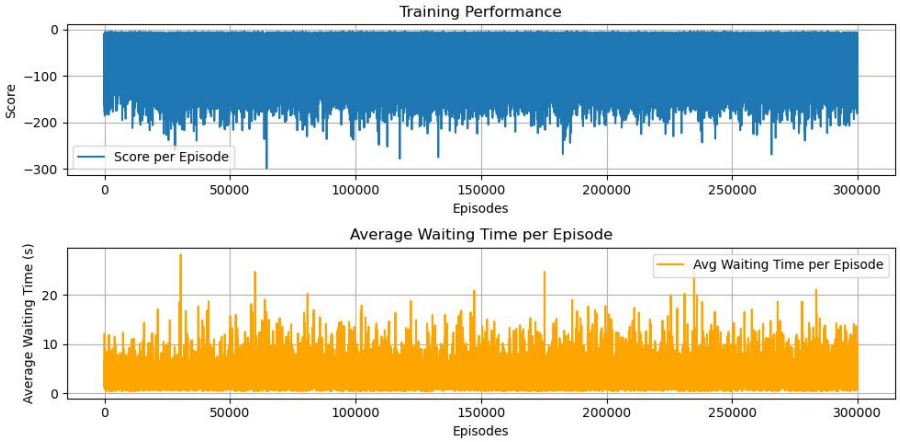

Figure 1: Addestramento Q-Learning su 300.000 episodi

Alla fine dell'addestramento, la Q-table e il modello addestrato sono stati salvati rispettivamente in un file di testo e in un file `.pkl` per utilizzi futuri. Il grafico mostra le performance del modello durante il training. Sulla parte superiore del grafico troviamo il punteggio accumulato dall'agente durante gli episodi. Come possiamo notare dal grafico si nota una certa variabilità nei punteggi che variano in un range compreso fra 0 e -300. Il secondo grafico invece mostra il tempo medio di attesa per singolo episodio. Il tempo medio di attesa durante la fase iniziale del training mostra delle oscillazioni significative raggiungendo dei picchi molto alti. Durante il training, con l'avanzare degli episodi, tende a stabilizzarsi attorno a valori più bassi (intorno al 10), anche se rimangono delle fluttuazioni sporadiche.

5.4 Validazione

Il modello addestrato è stato validato successivamente su un numero prestabilito di episodi in cui non è stato eseguito nessun aggionramento della Q-Table. Sono stati registrati, inoltre, i tempi medi di attesa e il numero di collisioni per episodio. Durante la fase di validazione dell'algoritmo di Q-Learning, i grafici mostrano l'andamento delle performance dell'agente in termini di punteggio e tempo medio di attesa per episodio. Il primo grafico, relativo al punteggio per episodio, evidenzia che l'agente riesce generalmente a ottenere risultati vicini a zero, suggerendo che ha acquisito una buona politica durante il training. Tuttavia, si notano occasionalmente episodi con punteggi nettamente più bassi, segno che l'agente potrebbe affrontare situazioni complesse o occasionali errori di esplorazione. Nonostante questi episodi anomali, la performance globale appare stabile e consistente. Il secondo grafico, che riporta il tempo medio di attesa, mostra un'evoluzione positiva. Il tempo di attesa si mantiene generalmente basso, oscillando attorno a valori compresi tra 2 e 6 secondi. Anche qui

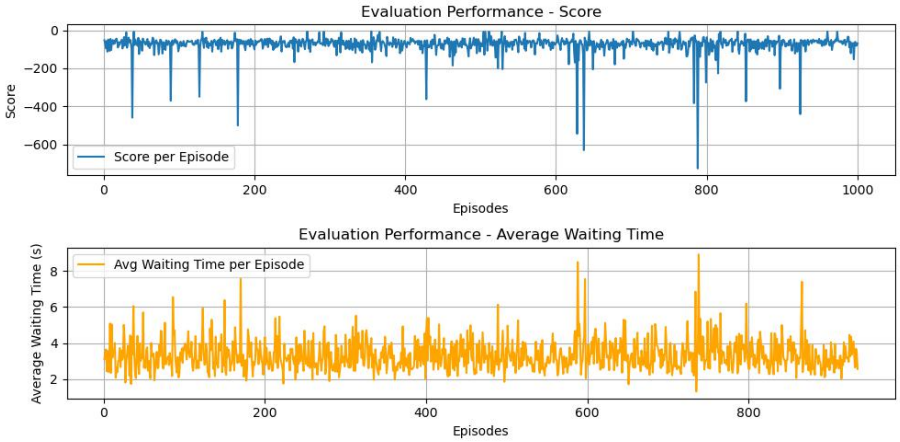

Figure 2: Validazione Q-Learning su 1.000 episodi

si osservano fluttuazioni, ma queste diminuiscono nel corso degli episodi, indicando che l'agente è capace di mantenere un flusso di traffico più fluido rispetto all'inizio.

6 SARSA

SARSA (*State-Action-Reward-State-Action*) è un algoritmo on-policy, il che significa che l'aggiornamento dei valori Q si basa sulle azioni effettivamente scelte dall'agente durante l'esplorazione. In questo caso l'agente aggiorna i valori Q considerando sia lo stato corrente che l'azione che sceglierà successivamente, in base alla sua policy attuale. Il suo nome deriva dalla sequenza di stati e azioni utilizzata per aggiornare la funzione Q, ovvero:

Stato (S), Azione (A), Ricompensa (R), Nuovo Stato (S'), Nuova Azione (A').

L'aggiornamento della funzione Q è governato dalla seguente formula:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t) \right],$$

dove:

- $Q(S_t, A_t)$ rappresenta il valore atteso dello stato S_t e dell'azione A_t,
- $\alpha \in (0, 1]$ è il tasso di apprendimento, che controlla la velocità di aggiornamento dei valori Q,
- R_{t+1} è la ricompensa ricevuta dopo aver eseguito A_t in S_t,

- $\gamma \in [0, 1]$ è il fattore di sconto, che determina quanto peso viene dato alle ricompense future,
- $Q(S_{t+1}, A_{t+1})$ è il valore atteso dello stato successivo S_{t+1} e dell'azione A_{t+1}, scelta secondo una politica on-policy.

6.1 Differenze rispetto al Q-Learning

A differenza del Q-Learning, che è un algoritmo off-policy, SARSA aggiorna i valori Q considerando sia lo stato e l'azione corrente sia la politica seguita dall'agente per selezionare A_{t+1}. Questo rende SARSA più "cauto" rispetto al Q-Learning, poiché tiene conto delle decisioni reali prese dall'agente, invece di fare ipotesi su un comportamento ottimale ideale.

6.2 Pseudocodice

L'algoritmo SARSA può essere descritto come segue:

Algorithm 2 SARSA (State-Action-Reward-State-Action)

Inizializza $Q(S, A)$ arbitrariamente per tutti gli stati S e azioni A
for ogni episodio **do**
    Inizializza lo stato iniziale S
    Seleziona un'azione A da S secondo una politica π (es. ε-greedy)
    while S non è uno stato terminale **do**
        Esegui l'azione A, osserva la ricompensa R e il nuovo stato S'
        Seleziona la nuova azione A' da S' secondo la politica π
        Aggiorna $Q(S, A)$ utilizzando la formula:

$$Q(S, A) \leftarrow Q(S, A) + \alpha \left[R + \gamma Q(S', A') - Q(S, A)\right]$$

        Aggiorna $S \leftarrow S'$ e $A \leftarrow A'$
    end while
end for

7 Implementazione SARSA

7.1 Descrizione del modello

L'algoritmo SARSA (*State-Action-Reward-State-Action*) utilizza una politica ϵ-greedy per il bilanciamento tra esplorazione e sfruttamento. Gli iperparametri utilizzati nell'implementazione sono i seguenti:

- **Tasso di apprendimento (α):** 0.3
- **Tasso di esplorazione (ϵ):** 0.1
- **Fattore di sconto (γ):** 0.9

- **Azioni disponibili:** $A = \{0, 1\}$,

I valori $Q(s, a)$ vengono inizializzati a zero e aggiornati secondo la seguente regola:

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

dove s è lo stato corrente, a l'azione eseguita, r il premio ricevuto, s' il nuovo stato, e a' l'azione successiva scelta dalla politica.

7.2 Training

Per l'addestramento dell'agente SARSA, sono stati utilizzati 300,000 episodi. Durante il training, l'agente ha iterativamente aggiornato i valori $Q(s, a)$ in base alle interazioni con l'ambiente.

Il processo di training è stato monitorato mediante i seguenti indicatori:

- **Punteggio per episodio:** rappresenta la somma dei premi accumulati.
- **Tempo medio di attesa:** indica l'efficacia del controllo semaforico.

In Figura 3 sono mostrati i risultati relativi al punteggio medio e al tempo medio di attesa durante il training su 300.000 episodi. I punteggi oscillano principalmente in un range compreso fra 0 e -4.000, con dei picchi occasionali di punteggi molto negativi. Non si osserva una chiara tendenza di miglioramento, suggerendo che l'algoritmo potrebbe non aver trovato una strategia stabile o efficace nella gestione del traffico. Per quanto riguarda il tempo medio di attesa durante il training, anche qui viene mantenuta una certa instabilità con tempi che si aggirano principalmente fra gli 0 e 15 secondi, con picchi occasionali sopra i 20.

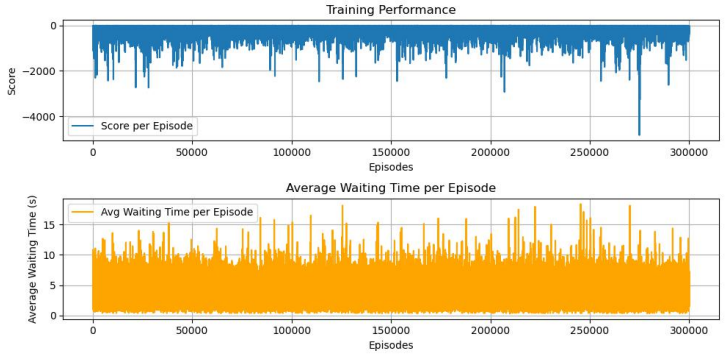

Figure 3: Andamento del punteggio e del tempo medio di attesa durante il training dell'agente SARSA.

7.3 Validazione

Nella fase di validazione, l'agente è stato testato su un totale di 1.000 episodi, senza aggiornare i valori $Q(s, a)$. I principali parametri di valutazione utilizzati sono stati:

- **Tempo medio di attesa:** tempo medio impiegato dai veicoli per attraversare l'intersezione.
- **Numero di collisioni:** indicatore della sicurezza dell'agente.

I risultati ottenuti sono illustrati nella Figura 4. Per quanto riguarda il punteggio calcolato per episodio, si nota un range di valori piuttosto ampio che va da 0 a 20.000. Alcuni picchi raggiungono valori estremamente bassi indicando delle situazioni in cui l'algoritmo ha performato molto male. Il grafico inferiore mostra invece il tempo medio di attesa dei veicoli. Il tempo oscilla visibilmente tra 0 e 40 secondi con dei valori che variano sensibilmente da episodio a episodio. La presenza di numerosi picchi indica una variabilità elevata nell'efficienza dei semafori gestiti dall'algoritmo.

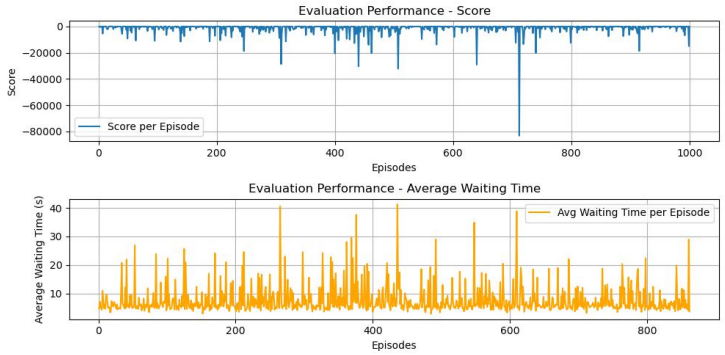

Figure 4: Andamento del punteggio e del tempo medio di attesa durante la validazione dell'agente SARSA.

8 Deep Q-Learning (DQL)

Il Deep Q-Learning (DQL) è un'estensione del classico Q-Learning che utilizza una rete neurale profonda (Deep Neural Network, DNN) per approssimare la funzione Q, consentendo di affrontare problemi con spazi di stato di dimensioni molto elevate. Il DQN combina il Q-Learning con le reti neurali per stimare i Q-value, che rappresentano il valore atteso di intraprendere una certa azione in uno stato specifico. Questo approccio si è dimostrato particolarmente efficace nei contesti in cui lo spazio degli stati è troppo grande per essere rappresentato esplicitamente. La rete neurale generalizza l'apprendimento, mappando gli stati alle azioni ottimali senza dover esplorare ogni combinazione possibile, e apprende

una rappresentazione degli stati che aiuta nella selezione delle azioni ottimali anche in ambienti complessi.

Un'innovazione chiave del DQN è l'uso di una rete target separata per stabilizzare l'addestramento. Inoltre, l'algoritmo implementa un processo chiamato replay esperienziale, che consiste nel memorizzare le esperienze passate (sotto forma di stati, azioni, ricompense e stati successivi) in un buffer e riutilizzarle per aggiornare i pesi della rete. Questo meccanismo riduce la correlazione tra i dati, migliorando la convergenza e permettendo di usare esperienze eterogenee per addestrare il modello.

8.1 Obiettivo del Deep Q-Learning

L'obiettivo principale del DQL è approssimare la funzione Q ottimale $Q^*(s, a)$, che rappresenta il valore atteso della somma scontata delle ricompense future eseguendo un'azione a in uno stato s e seguendo la politica ottimale:

$$Q^*(s, a) = E\left[r_t + \gamma \max_{a'} Q^*(s_{t+1}, a') \mid s_t = s, a_t = a\right], \tag{1}$$

dove r_t è la ricompensa immediata, $\gamma \in [0, 1]$ è il fattore di sconto, e s_{t+1} è lo stato successivo.

8.2 Struttura del Deep Q-Learning

A differenza del Q-Learning tradizionale, che memorizza una Q-table esplicita per ogni stato e azione, nel Deep Q-Learning la funzione Q non viene rappresentata da una tabella. Invece, viene approssimata tramite una rete neurale profonda (DNN). La rete neurale prende come input uno stato s e fornisce come output i valori Q per tutte le azioni possibili in quello stato, senza la necessità di memorizzare una tabella di dimensioni potenzialmente molto grandi. In questo modo, la rete neurale generalizza l'apprendimento e consente all'agente di selezionare azioni ottimali anche in ambienti con spazi di stato e azione complessi o continui.

Questa differenza è fondamentale, poiché permette al Deep Q-Learning di affrontare ambienti con spazi di stato e azione molto grandi o continui, dove sarebbe impraticabile o impossibile costruire una tabella Q esplicita. La rete neurale, quindi, impara una rappresentazione degli stati che permette di selezionare azioni ottimali senza dover esplorare ogni combinazione possibile.

L'algoritmo del DQL può essere suddiviso nei seguenti passaggi:

Algorithm 3 Deep Q-Learning

Inizializza i pesi della rete neurale $Q(s, a; \theta)$
Imposta i parametri ϵ, γ, α, ϵ_{decay} e ϵ_{min}
for ogni episodio **do**
    $s \leftarrow$ Stato iniziale dell'ambiente
    while non terminale **do**
        Esegui $a \leftarrow epsilon - greedypolicy(s)$
        Osserva r, s' dall'ambiente dopo a
        Calcola Q_{target}: $r + \gamma \cdot \max_{a'} Q(s', a'; \theta)$
        Calcola la perdita L: $(Q_{target} - Q(s, a; \theta))^2$
        Aggiorna θ utilizzando $\nabla_\theta L$
        $s \leftarrow s'$
    end while
    Aggiorna $\epsilon \leftarrow \max(\epsilon \cdot \epsilon_{decay}, \epsilon_{min})$
end for

8.3 Componenti chiave del Deep Q-Learning

1. **Rete neurale** $Q(s, a; \theta)$: La rete neurale è utilizzata per approssimare la funzione Q. I pesi della rete neurale (denotati con θ) vengono inizializzati e aggiornati durante l'addestramento. La rete neurale approssima poi i valori di Q per ogni stato e azione, consentendo all'agente di prendere decisioni ottimali in spazi di stato e azione complessi.

2. **Politica ϵ-greedy**:

   La politica ϵ-greedy bilancia l'esplorazione (provare azioni nuove) e l'exploitation (scegliere l'azione che massimizza il valore di Q). Con probabilità ϵ, l'agente esegue un'azione casuale, altrimenti seleziona l'azione con il valore di Q più alto. Durante l'apprendimento, ϵ diminuisce progressivamente con il tempo, riducendo l'esplorazione e aumentando l'exploitation. Questo consente all'agente di esplorare inizialmente ampiamente e successivamente sfruttare meglio le conoscenze acquisite.

3. **Funzione di Perdita**:

   La funzione di perdita misura l'errore tra il valore di Q attuale e il valore target Q_{target}. Il valore Q_{target} è calcolato usando l'equazione di Bellman:

   $$Q_{target} = r + \gamma \cdot \max_{a'} Q(s', a'; \theta)$$

   Minimizzare la funzione di perdita è l'obiettivo dell'ottimizzazione della rete neurale, affinché la rete apprenda a prevedere i valori di Q corretti.

4. **Aggiornamento dei Pesi della Rete Neurale**: I pesi della rete neurale vengono aggiornati minimizzando la funzione di perdita, tramite il metodo di backpropagation.

9 Implementazione Deep Q-Learning

9.1 Struttura dell'Agente

L'agente `DQNAgent` rappresenta l'entità responsabile dell'apprendimento tramite l'utilizzo del Deep Q-Learning. La classe che rappresenta l'agente ha le seguenti caratteristiche:

- **Input e Output:** L'agente riceve in input uno stato rappresentato come vettore numerico (`state`), mentre l'output consiste nell'azione ottimale da eseguire.
- **Rete Neurale:** Una rete neurale feed-forward è utilizzata per approssimare la funzione Q. La rete è composta da:
  - Un livello di input con dimensione pari alla lunghezza dello stato.
  - Due livelli nascosti con 256 neuroni ciascuno e funzione di attivazione ReLU.
  - Un livello di output con dimensione pari al numero di azioni possibili.
- **Funzione di Perdita:** La funzione `MSELoss` (Mean Squared Error) misura la differenza tra il valore Q predetto e il valore target.
- **Ottimizzatore:** L'algoritmo `Adam` è utilizzato per aggiornare i pesi della rete neurale.
- **Epsilon-Greedy:** La strategia di esplorazione `epsilon-greedy` consente di bilanciare esplorazione ed exploit durante l'apprendimento.

9.2 Addestramento del Modello

L'addestramento viene eseguito tramite il metodo `train_deep_q_learning`, che esegue i seguenti passi:

1. **Inizializzazione:** L'ambiente `ExMeteo` viene resettato per ottenere lo stato iniziale.
2. **Episodi di Addestramento:** Per ciascun episodio:
   (a) Determinare l'azione ottimale tramite la politica epsilon-greedy.
   (b) Eseguire l'azione e osservare il nuovo stato, la ricompensa e la condizione di fine episodio.
   (c) Calcolare il valore target Q utilizzando l'equazione di Bellman:

$$Q_{target} = r + \gamma \cdot \max_{a'} Q(s', a') \cdot (1 - done) \tag{2}$$

   (d) Calcolare la perdita rispetto al valore predetto e aggiornare i pesi della rete neurale tramite backpropagation.

3. **Decadimento di Epsilon:** Aggiornare il valore di `epsilon` per ridurre progressivamente l'esplorazione.

Nella figura è mostrato il risultato dell'addestramento ottenuto con l'algoritmo Deep Q-Learning (DQL), effettuato su un totale di 300.000 episodi. Per quanto riguarda il punteggio, il comportamento del DQL risulta abbastanza instabile, con un range che varia principalmente tra 0 e -500. Tuttavia, si osservano anche alcuni picchi significativi che arrivano fino a -2000 punti. Nella parte inferiore, è visibile il grafico relativo al tempo medio di attesa. Durante l'addestramento, il tempo non sembra diminuire significativamente, mantenendosi in una fascia compresa tra 0 e 10 secondi.

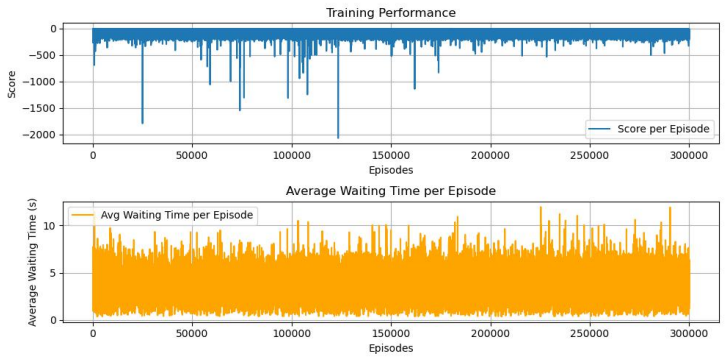

Figure 5: Andamento del punteggio e del tempo medio di attesa durante il training dell'agente DQL.

9.3 Validazione del Modello

Dopo l'addestramento, il modello viene validato su episodi di test tramite il metodo `validate_model`. I pesi della rete salvati vengono caricati per valutare la performance dell'agente su episodi in cui l'esplorazione è disattivata. Vengono analizzati il tempo medio di attesa per episodio e il numero di collisioni.

Di seguito è presentato il grafico relativo alla validazione del modello. Il punteggio appare molto più stabile rispetto a quello osservato durante l'addestramento, oscillando all'interno di un intervallo ridotto, compreso tra 0 e un massimo di -200. Per quanto riguarda la metrica del tempo medio di attesa per episodio, essa risulta piuttosto stabile, con valori che variano tra 2 e 3.5 secondi per episodio.

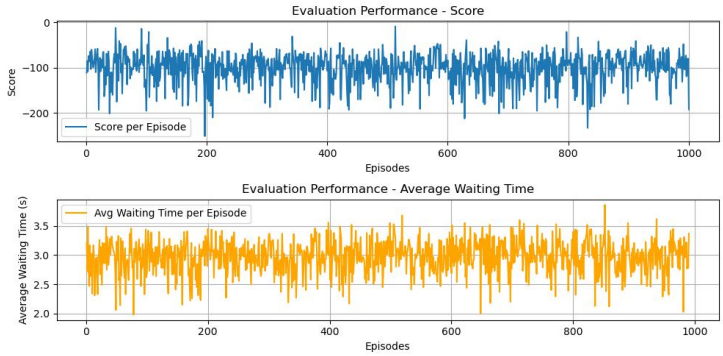

Figure 6: Andamento del punteggio e del tempo medio di attesa durante la validazione dell'agente DQL.

10 Confronti

Tutte le simulazioni effettuate comprendono un numero totale di episodi osservati pari a 1.000. I risultati sono riportati nella seguente tabella:

Modello	Tempo Medio	Collisioni
Fixed Cycle	3.56	0.05
Longest Queue First	3.54	0.05
Deep Q-Learning	2.99	0.01
Q-Learning	3.29	0.04
SARSA	7.80	0.13

Table 1: Confronti

10.1 Analisi dei Risultati

- **Fixed Cycle** e **Longest Queue First**: Questi due modelli mostrano prestazioni simili, con un tempo medio rispettivamente di 3.56 e 3.54 secondi e un numero di collisioni di 0.05. Sebbene i tempi siano moderati, il numero di collisioni rimane basso, segnalando una certa affidabilità ma con margini di miglioramento per l'efficienza.

- **Deep Q-Learning**: Questo modello spicca per le migliori prestazioni complessive, con il tempo medio più basso (2.99 secondi) e un numero molto contenuto di collisioni (0.01). Tali risultati evidenziano una netta superiorità in termini di efficienza e sicurezza rispetto agli altri modelli.

- **Q-Learning**: Si posiziona come un'opzione intermedia, con un tempo medio di 3.29 secondi e un numero di collisioni pari a 0.04. Rispetto a

Deep Q-Learning, ha prestazioni leggermente inferiori, ma supera sia il modello Fixed Cycle che Longest Queue First.

- **SARSA**: Questo modello è quello meno performante. Con un tempo medio di 7.80 secondi e un numero di collisioni pari a 0.13, dimostra di essere meno efficace sia nell'ottimizzazione del traffico sia nella riduzione degli incidenti rispetto a tutti gli altri modelli analizzati.

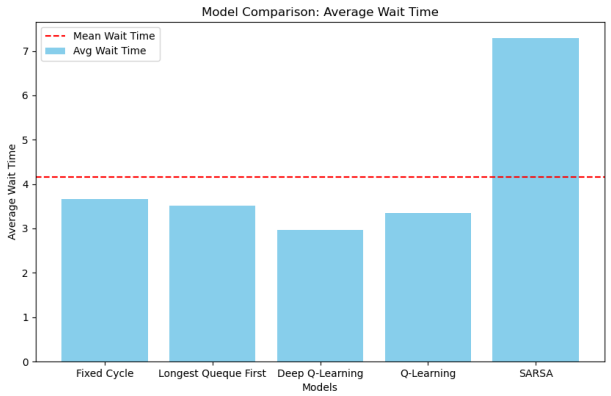

Figure 7: Risultati finali

10.2 Conclusioni

Il confronto evidenzia che il **Deep Q-Learning** è il modello più promettente, garantendo tempi di attesa ridotti e un basso tasso di collisioni. Al contrario, il **SARSA** mostra prestazioni significativamente peggiori, risultando meno adeguato per l'ottimizzazione del traffico. I modelli tradizionali come **Fixed Cycle** e **Longest Queue First** offrono una stabilità accettabile ma non riescono a competere con l'efficienza dei metodi di apprendimento più avanzati.