Controllo Traffico con Apprendimento per Rinforzo

Applicazione dell'apprendimento per rinforzo alla gestione di un incrocio semaforico

23/02/2024

di

Andrea Falbo
a.falbo7@campus.unimib.it

Ruben Tenderini
r.tenderini@campus.unimib.it

Sotto la supervisione di
Dr. Giuseppe Vizzari

Università degli Studi di Milano-Bicocca

Contents

1 Introduzione

1.1 Definizione

Si desidera confrontare l'efficacia di diverse strategie di apprendimento per rinforzo per la gestione del traffico di una intersezione semaforizzata, identificata come BI (Big Intersection). Nello specifico, saranno esaminate quattro diverse tecniche di gestione del semaforo:

1. **Ciclo Fisso:** Le fasi del semaforo seguono un ciclo fisso, ripetendosi in modo prevedibile.
2. **Q-Learning:** L'operatività del semaforo è affidata a un agente addestrato mediante il metodo di Q-Learning.
3. **Deep Q-Learning:** L'operatività del semaforo è affidata a un agente addestrato mediante il metodo di Deep Q-Learning.
4. **Sarsa Learning:** L'operatività del semaforo è affidata a un agente addestrato mediante il metodo di Sarsa Learning.

Ogni algoritmo verrà sperimentato attraverso 2 diverse configurazioni, variando gli iperparametri in modo da esplorare la sensibilità degli algoritmi e la loro adattabilità a contesti variabili, contribuendo a identificare parametri ottimali e migliorare la generalizzazione delle prestazioni nel controllo del traffico.

L'addestramento dei modelli considererà tre diverse situazioni di traffico: traffico basso, traffico medio e traffico alto. Questa scelta è finalizzata a valutare la capacità dei modelli di adattarsi a varie dinamiche del traffico e di generalizzare efficacemente.

I modelli addestrati saranno successivamente valutati eseguendoli sia nella stessa situazione di traffico in cui sono stati addestrati, sia in situazioni di traffico non osservate durante la fase di addestramento. Tale metodologia mira a testare la capacità di generalizzazione dei modelli, evitando overfitting.

1.2 Obiettivo

L'obiettivo primario di questo studio è identificare le configurazioni ottimali per ciascun algoritmo di apprendimento. Successivamente, saranno confrontate le prestazioni di tali algoritmi attraverso diverse metriche e situazioni di traffico. Questa analisi mira a fornire una valutazione accurata e comparativa delle strategie di apprendimento, cercando di estendere le conoscenze ottenute dalla precedente indagine condotta nella 2WSI-RL. [3]

2 Apprendimento per Rinforzo

2.1 Probabilistic Planning

Nel contesto del Reinforcement Learning, l'agente affronta spesso la sfida di operare in ambienti in cui la conoscenza completa dello stato è impraticabile. Questo problema è affrontato efficacemente attraverso il Probabilistic Planning, una metodologia che incorpora l'incertezza nell'apprendimento dell'agente.

Il cuore di questa approccio risiede nella teoria delle probabilità, che consente all'agente di mantenere delle credenze probabilistiche, chiamate belief. Questi sono dinamici e variano in risposta alle nuove informazioni ottenute tramite i sensori.

Un concetto chiave in questo contesto è la Formula di Bayes, utilizzata per aggiornare i belief in base alle osservazioni sensoriali. Tale formula permette all'agente di ricalibrare le sue credenze in maniera razionale e guidata dai dati:

$$P(x|y_1 \wedge \ldots \wedge y_n) = \frac{P(y_n|x)P(x|y_1 \wedge \ldots \wedge y_{n-1})}{P(y_1 \wedge \ldots \wedge y_n)}$$

Markov Localization è una tecnica particolarmente rilevante in questo contesto. Inizialmente, l'agente parte senza conoscenza dello stato attuale. Attraverso i sensori, percepisce parte dello stato corrente, consentendo l'applicazione della probabilità condizionata e l'aggiornamento dei belief tramite la Formula di Bayes. Questo processo si svolge anche quando l'agente compie azioni, percependo il cambiamento dell'ambiente e aggiornando di conseguenza i suoi valori.

Un concetto chiave associato a Markov è il Markov Decision Process (MDP). Questo modello, caratterizzato da un insieme discreto di stati S, un insieme di azioni A, una funzione di transizione $T(s, a, s')$ e una funzione di ricompensa $R(s, a, s')$, fornisce una struttura formale per affrontare problemi decisionali in presenza di incertezza.

L'obiettivo principale in un MDP è massimizzare le ricompense, differenziandosi dai tradizionali problemi di ricerca per la sua enfasi sulla massimizzazione delle ricompense anziché sulla minimizzazione dei costi. L'approccio probabilistico adottato in Probabilistic Planning consente di modellare efficacemente l'incertezza, sviluppando strategie di pianificazione che si adattano dinamicamente alle condizioni mutevoli dell'ambiente.

2.2 Reinforcement Learning

Il Reinforcement Learning è una tecnica di apprendimento che prevede l'apprendimento di un agente attraverso l'interazione con un ambiente dinamico. L'agente interagisce con l'ambiente in modo sequenziale, compiendo azioni e ricevendo una ricompensa (*reward*).

Lo scopo dell'agente è quello di massimizzare la ricompensa cumulativa che viene fornita dall'ambiente in risposta alle sue azioni. L'RL si basa su un processo di apprendimento per esplorazione e sperimentazione, in cui l'agente deve scegliere le azioni da effettuare in modo da massimizzare la sua ricompensa. L'agente apprende dalla sua esperienza accumulando conoscenze e sviluppando strategie sempre più efficaci.

Lo scopo di un agente è quindi

$$\max \sum_{t=0}^{T} R(s_t, a_t)$$

dove T è il massimo degli istanti temporali.

Un agente che cerca di massimizzare la ricompensa cumulativa, potrebbe trovarsi in una situazione di indecisione nel caso di più sequenze di azioni la cui ricompensa totale sia uguale. Per risolvere questa indecisione si introduce il discount factor γ in modo da far scegliere all'agente la massimizzazione più veloce della

ricompensa cumulativa. La ricompensa al tempo t è data da:

$$R_t = \sum_{i=t}^{T} \gamma^{i-t} r_i$$

dove r_i è la ricompensa per la transizione dell'istante di tempo i. Questa sommatoria altro non è che la serie geometrica, e in quanto tale converge sempre a un valore finito anche per $T = \infty$.

2.3 Q-Learning

Il Q-Learning è un algoritmo specifico di Reinforcement Learning che si basa sulla costruzione di una tabella indicante il valore di ricompensa per ogni possibile stato al compimento di qualsiasi azione possibile. La procedura iterativa di costruzione della tabella coinvolge l'esplorazione dell'ambiente con azioni più o meno casuali.

Ad ogni passo, la tabella viene aggiornata con la seguente formula:

$$Q[s][a] = Q[s][a] + \alpha \left(r + \gamma \cdot \max(Q[s']) - Q[s][a]\right)$$

Dove:

- $Q(s, a)$ è la stima del valore d'azione per lo stato s e l'azione a,
- s è lo stato attuale,
- a è l'azione compiuta,
- r è la ricompensa ottenuta,
- s' è lo stato successivo,
- γ è il fattore di sconto,
- α è il tasso di apprendimento,
- $\max(Q[s'])$ restituisce la massima ricompensa ottenibile dallo stato s'.

La scelta dell'azione da compiere inizialmente è casuale fino a quando si decide di aver esplorato a sufficienza. La politica comunemente usata per questo scopo è la $\epsilon - greedy$, dove ϵ rappresenta la probabilità di compiere un'azione casuale e viene ridotto progressivamente fino a un minimo.

Il Q-Learning è una tecnica potente ed efficace, capace di apprendere strategie di azione ottimali in diverse applicazioni. Tuttavia, può essere sensibile al rumore, all'indeterminazione delle azioni e alla gestione di ambienti continui. Inoltre, richiede una grande quantità di memoria per memorizzare la tabella, soprattutto quando l'ambiente ha uno spazio di stati S e di azioni A considerevole: $(\theta(S \cdot A))$.

2.4 Deep Q-Learning

Il Deep Reinforcement Learning è una tecnica di apprendimento automatico basata su RL, ma che si pone come obiettivo di sopperire alla problematica di quest'ultimo per spazi di stati e azioni molto grandi. Per farlo si utilizzano delle reti neurali profonde per approssimare i valori della tabella senza richiederne le stesse risorse in termini di memoria.

L'approccio più semplice per l'implementazione di una rete neurale profonda come approssimatore per la tabella consiste nell'utilizzare una rete neurale profonda ad ogni passo per ottenere la ricompensa attesa e aggiornare i pesi della rete neurale tramite il metodo del gradient descent rispetto alla ricompensa effettivamente ottenuta.

2.5 SARSA Learning

Il SARSA (State-Action-Reward-State-Action) Learning è un algoritmo di apprendimento per il controllo di agenti in ambienti di apprendimento per rinforzo. L'algoritmo mantiene una funzione Q che stima il valore dell'azione in uno stato specifico. Durante l'apprendimento, l'agente esplora l'ambiente, seleziona azioni in base alla sua policy corrente e aggiorna la sua stima Q utilizzando l'equazione di aggiornamento SARSA:

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

Dove:

- $Q(s,a)$ è la stima del valore d'azione per lo stato s e l'azione a,
- α è il tasso di apprendimento,
- r è il reward ottenuto eseguendo l'azione a nello stato s,
- γ è il fattore di sconto,
- s' è lo stato successivo,
- a' è l'azione successiva selezionata dalla policy dell'agente.

Questa equazione regola la stima del valore d'azione in base al reward ottenuto, al valore d'azione successivo previsto e al tasso di apprendimento, guidando l'adattamento della policy dell'agente nel corso dell'apprendimento.

Mentre Q-Learning utilizza una strategia off-policy, apprendendo dalla massima stima del valore d'azione futura indipendentemente dall'azione effettivamente selezionata, SARSA adotta un approccio on-policy. Ciò significa che SARSA considera le azioni effettivamente scelte durante l'esplorazione, incorporando la policy corrente nella stima dei valori d'azione. Questa differenza fa sì che SARSA sia più sensibile alle politiche di esplorazione, garantendo che l'agente apprenda in base alle azioni che effettivamente compie.

3 Strumenti

3.1 SUMO

SUMO, acronimo di "Simulation of Urban MObility", rappresenta un software open-source dedicato alla simulazione dettagliata della mobilità urbana. Sviluppato presso l'Istituto di Sistemi di Trasporto del Centro Aerospaziale Tedesco (DLR), SUMO offre una piattaforma flessibile e altamente portatile. Questo strumento consente la modellazione intricata di reti stradali e varie modalità di trasporto all'interno di ambienti urbani. [4]

3.2 SUMO-RL

SUMO-RL è una risorsa significativa nel contesto della simulazione della mobilità urbana. Questo framework, sviluppato da Lucas N. Alegre, è focalizzato sull'integrazione di SUMO con algoritmi di apprendimento per rinforzo (RL). Il suo obiettivo è estendere le capacità di SUMO, consentendo agli utenti di esplorare e sviluppare strategie di controllo del traffico avanzate utilizzando approcci di RL. [1]

3.3 Stable Baselines 3

Stable Baselines 3 è una libreria di apprendimento per rinforzo (RL) in Python, sviluppata da OpenAI. Essa fornisce un set di implementazioni di algoritmi di RL stabili e affidabili, progettati per essere facilmente accessibili e utilizzabili da parte degli sviluppatori. Stable Baselines 3 è una versione migliorata e ristrutturata della sua controparte precedente, Stable Baselines. In questo progetto è stato fondamentale anche per l'integrazione di ambienti Gymnasium. [2]

3.4 2WSI-RL

2WSI-RL, acronimo di 2 Way Single Intersection for Reinforcement Learning, è uno studio sull'applicazione di apprendimento per rinforzo alla gestione di un'intersezione semaforizzata, nello specifico la 2 Way Single Intersection. [3]

3.5 DQL-TSC

DQL-TSC, acronimo di Deep Q-Learning agent for Traffic Signal Control, è un framework dove un agente che apprende per rinforzo tramite Q-Learning prova a scegliere la fase verde dell'interserzione per massimizzare l'efficienza. [5]

3.6 Python

Python è un linguaggio di programmazione ad alto livello ampiamente utilizzato nel campo dell'apprendimento automatico, dell'elaborazione dati e in molteplici ambiti scientifici. Grazie alla sua sintassi chiara e flessibilità, Python è diventato uno degli strumenti preferiti dagli sviluppatori e dai ricercatori per la scrittura di codice efficiente e comprensibile.

3.7 Matplotlib

Matplotlib è una libreria di visualizzazione dati in Python, progettata per creare grafici statici, interattivi e animazioni. Con una vasta gamma di opzioni di personalizzazione e una sintassi intuitiva, Matplotlib è uno strumento essenziale per rappresentare graficamente i risultati delle analisi dati e comunicare in modo efficace le informazioni visive.

3.8 PyTorch

PyTorch è una libreria open-source di machine learning per Python, sviluppata da Meta AI. Con un focus particolare sul calcolo su tensori e il supporto per l'apprendimento profondo, PyTorch è ampiamente utilizzato per la costruzione e l'addestramento di reti neurali. La sua architettura dinamica e la comunità attiva di

sviluppatori lo rendono uno strumento potente per l'implementazione di modelli complessi e l'esplorazione delle ultime tecnologie nel campo dell'intelligenza artificiale.

3.9 PyCharm

PyCharm è un ambiente di sviluppo integrato specificamente progettato per il linguaggio di programmazione Python. Sviluppato da JetBrains, PyCharm offre funzionalità avanzate come la refactoring automatica del codice, la navigazione intelligente, il supporto per l'analisi statica e una vasta gamma di strumenti di debugging. La sua interfaccia utente intuitiva e la facilità di integrazione con strumenti di gestione del versionamento lo rendono una scelta popolare tra gli sviluppatori Python.

3.10 Github

GitHub è una piattaforma di hosting per progetti software che utilizza il sistema di controllo del versionamento Git. Fornisce un ambiente collaborativo per lo sviluppo di software, consentendo agli sviluppatori di caricare, condividere e gestire il versionamento dei loro progetti. GitHub offre funzionalità come problemi, richieste pull, e integrazioni con strumenti di continuous integration, facilitando il lavoro in team distribuiti e la contribuzione alla comunità open-source.

4 Ambiente

4.1 Introduzione

L'ambiente in cui gli agenti sono stati inseriti è caratterizzato da un incrocio con un singolo semaforo, noto come big-intersection.net.xml in Sumo-RL. Questo ambiente definisce le corsie, i semafori, i sensi di marcia e altri aspetti. Ad ogni lato dell'incrocio sono presenti quattro corsie: una per svolte a sinistra, due per procedere diritto e una per procedere diritto o svoltare a destra.

Successivamente, questa configurazione è stata modellata in tre scenari di traffico distinti, rappresentando livelli variabili di congestione stradale: traffico basso, traffico medio e traffico alto. I tre file sono stati rinominati come big-intersection-lt.net.xml, big-intersection-mt.net.xml e big-intersection-ht.net.xml e sono presenti nella cartella big-intersection).

Il numero di veicoli per le tre situazioni di traffico è stato determinato attraverso una serie di esperimenti iterativi per identificare condizioni ottimali. In conclusione, i valori più adeguati a rappresentare le 3 situazioni si sono rivelati:

1. Traffico Basso: 50 veicoli per rotta
2. Traffico Medio: 100 veicoli per rotta
3. Traffico Alto: 150 veicoli per rotta

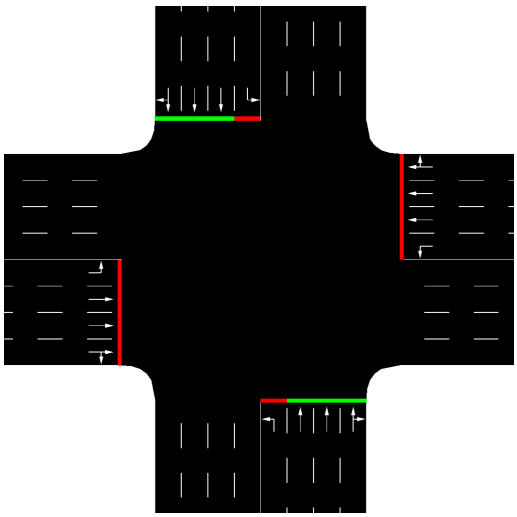

Figure 1: Rappresentazione dell'ambiente

4.2 MDP: Osservazioni, Azioni e Ricompense

Il nostro modello, basato sul processo decisionale di Markov, rimarca le definizioni fornite da Sumo-RL:

L'osservazione predefinita per ciascun agente dei segnali stradali è un vettore:

obs = [phase_one_hot, min_green, lane_1_density, ..., lane_n_density, lane_1_queue, ..., lane_n_queue]

dove:

- *phase_one_hot* è un vettore che codifica l'attuale fase verde attiva
- *min_green* è un valore booleano che indica se sono già passati almeno *min_green* secondi durante l'esecuzione

- *lane_i_density* è il numero di veicoli in arrivo nella corsia i diviso la capacità totale di tale corsia
- *lane_i_queue* è il numero di veicoli accodati in arrivo nella corsia i diviso la capacità totale di tale corsia

Lo spazio delle azioni è discreto. Ogni *delta_time* secondi, ciascun agente semaforico ha la possibilità di selezionare la configurazione successiva della fase verde. Ogni volta che una fase termina, la successiva sarà preceduta da una fase gialla che dura *yellow_time* secondi. Nel nostro caso avremo quindi $|A| = 4$ azioni discrete.

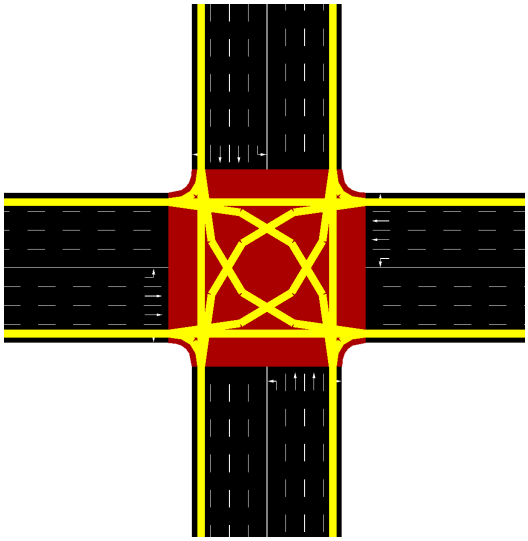

Figure 2: Rappresentazione delle azioni

La funzione di ricompensa, denominata Differential Waiting Time, è definita come il cambiamento cumulativo dei tempi di attesa dei veicoli: $r_t = D_{\alpha_t} - D_{\alpha_{t+1}}$

Questa misura riflette quanto il tempo di attesa, in risposta a un'azione, sia migliorato o peggiorato. L'obiettivo è spingere l'agente a compiere azioni che portino a una diminuzione del tempo totale di attesa. Se questo diminuisce nello step successivo, la differenza sarà positiva, indicando un risultato favorevole.

4.3 Configurazioni

L'ambiente SUMO è stato configurato seguendo i parametri definiti in 2WSI-RL:

- Durata totale della simulazione: 100,000 secondi
- Intervallo tra le azioni dell'agente: 10 secondi
- Durata della fase gialla: 4 secondi
- Durata minima per una fase verde: 10 secondi
- Durata massima per una fase verde: 50 secondi

Per valutare le prestazioni degli agenti, sono state selezionate quattro metriche di sistema fornite da Sumo-RL:

- *system_total_stopped*: rappresenta il numero totale di veicoli fermi nel passo attuale.
- *system_total_waiting_time*: indica la somma del tempo in secondi di simulazione in cui i veicoli sono stati fermi dall'ultima volta.

- *system_mean_waiting_time*: rappresenta la media aritmetica di tutti i tempi di attesa dei veicoli.
- *system_mean_speed*: indica la media aritmetica delle velocità dei veicoli.

5 Test

5.1 Training: Traffico Basso

ezezez

5.1.1 Test: Traffico Basso

ezezez

5.1.2 Test: Traffico Medio

ezezez

5.1.3 Test: Traffico Alto

ezezez

5.2 Training: Traffico Medio

ezezez

5.2.1 Test: Traffico Basso

ezezez

5.2.2 Test: Traffico Medio

ezezez

5.2.3 Test: Traffico Alto

ezezez

5.3 Training: Traffico Alto

ezezez

5.3.1 Test: Traffico Basso

ezezez

5.3.2 Test: Traffico Medio

ezezez

5.3.3 Test: Traffico Alto

ezezez

6 Conclusioni

ezez

References

[1] Lucas N. Alegre. SUMO-RL. `https://github.com/LucasAlegre/sumo-rl`, 2019.

[2] Adam Gleave Anssi Kanervisto Maximilian Ernestus Noah Dormann Antonin Raffin, Ashley Hill. Stable-Baselines3: Reliable Reinforcement Learning Implementations. `http://jmlr.org/papers/v22/20-1364.html`, 2021.

[3] Riccardo Chimisso. 2WSI-RL : 2 Way Single Intersection with Reinforcement Learning. `https://github.com/rChimisso/2WSI-RL`, 2022.

[4] DRL. Eclipse SUMO - Simulation of Urban MObility. `https://github.com/eclipse-sumo/sumo`, 2023.

[5] Andrea Vidali. Deep Q-Learning Agent for Traffic Signal Control. `https://github.com/AndreaVidali/Deep-QLearning-Agent-for-Traffic-Signal-Control`, 2021.