Università degli Studi della Calabria

Dipartimento di Ingegneria Informatica, Modellistica, Elettronica e Sistemistica

---

Corso di Laurea Magistrale in Ingegneria Informatica

Tesi di Laurea

Un approccio basato su LSTM per la previsione delle traiettorie dei veicoli in ambito Smart City

Relatore

Prof. Giancarlo Fortino
Prof. Noel Crespi
Dr. Roberto Minerva
Ing. Claudio Savaglio

Candidato

Fabrizio Mangione
Matr. 235188

Anno Accademico 2022-2023

Ai miei genitori per il sostegno ricevuto in tutti i momenti.
A me stesso per la dedizione, l'impegno e la costanza.

"Sic parvis magna"

Sommario

Il crescente fenomeno dell'urbanizzazione ha reso necessario lo sviluppo di sistemi di trasporto intelligenti per migliorare l'efficienza, la sicurezza e la sostenibilità della mobilità urbana. In questo contesto, la previsione accurata della traiettoria dei veicoli gioca un ruolo fondamentale, consentendo una migliore gestione del traffico. Questo lavoro di Tesi Magistrale esplora l'applicazione delle reti neurali ricorrenti, in particolare le Long Short-Term Memory (LSTM), nel contesto urbano delle Smart City, per la previsione della traiettoria dei veicoli. In particolare, sfruttando i dati storici relativi alle posizioni dei veicoli (provenienti da dispositivi collegati alla rete quali, ad esempio, telecamere IP), sono stati sviluppati due modelli di Machine Learning (ML) basati su un'architettura di tipo Encoder-Decoder LSTM. I modelli, che differiscono tra loro per la tipologia dell'input, incorporano le caratteristiche spaziotemporali dei dati e producono in output le previsioni delle posizioni dei veicoli, espresse in coordinate (x, y), consentendone il calcolo della traiettoria. I modelli sono stati addestrati, validati e testati utilizzando un dataset reale, inerente le informazioni sul traffico di un incrocio stradale situato nella città di Tokyo, in Giappone. I risultati ottenuti nella fase di inferenza dimostrano come le reti neurali ricorrenti, in particolare le LSTM, siano in grado di catturare complessi modelli di traffico e di adattarsi a condizioni altamente dinamiche tipiche di un contesto urbano, risultando quindi elementi fondamentali per la gestione delle mobilità all'interno delle Smart City attuali e future.

Indice

Capitolo 1

Introduzione

Nell'era moderna, caratterizzata dall'incessante crescita demografica e dall'innovazione tecnologica, le città di tutto il mondo stanno adattando le proprie infrastrutture al modello dettato dalle Smart City. Questo nuovo paradigma di sviluppo urbano fa leva sulle tecnologie dell'informazione e della comunicazione (ICT) e, in modo particolare, sull'Internet of Things (IoT) [1] per creare ambienti urbani più efficienti, sostenibili e abitabili, potenziando la qualità della vita dei residenti, ottimizzando l'utilizzo delle risorse e rafforzando la protezione. L'interconnessione di dispositivi, infrastrutture e servizi attraverso l'IoT è infatti al centro dello sviluppo delle Smart City. Sensori intelligenti, dispositivi mobili e sistemi di comunicazione avanzata lavorano in sinergia per raccogliere informazioni in tempo reale e consentire una pianificazione urbana più reattiva.

In questo contesto, l'integrazione degli Autonomous Vehicle (AV) può portare a una maggiore efficienza del traffico, una riduzione delle emissioni inquinanti e una diminuzione degli incidenti stradali. In particolare, la capacità di predire con precisione la traiettoria dei veicoli [2] [3] [4], non solo contribuisce a garantire una mobilità più fluida e sicura, ma rappresenta anche un elemento fondamentale per una migliore pianificazione delle risorse urbane e delle infrastrutture. Gli AV, pertanto, sono destinati a trasformare radicalmente la mobilità dei residenti delle Smart City e giocano un ruolo fondamentale nell'ambito degli Intelligent transport systems (ITS). Attraverso la condivisione di informazioni cruciali tra i veicoli e l'infrastruttura circostante, è possibile anticipare e prevedere il comportamento dei veicoli, impattando quindi sulla:

1. **Gestione del traffico**: Un'accurata previsione delle traiettorie può ottimizzare il flusso del traffico. Gli AV possono coordinare le loro azioni, come l'ingresso o l'uscita da una strada, in modo da minimizzare le congestioni e migliorare l'efficienza globale del sistema di trasporto.

2. **Sicurezza stradale**: La previsione delle traiettorie consente di rilevare in anticipo potenziali collisioni o situazioni pericolose e, in alcuni casi, evitarle senza l'intervento umano.

La ricerca nel campo della predizione della traiettoria dei veicoli nell'ambito degli ITS e degli AV affronta sfide multidisciplinari ed eterogenee, proprie della natura del problema, complessa, incerta e dinamica. In particolare, il contesto urbano è ricco di interazioni multi modali che richiedono quindi capacità di comprendere, comunicare e rispondere a segnali provenienti da diverse unità sensoriali e fonti, al fine di facilitare una comunicazione efficace e una coesistenza sicura con gli altri utenti della strada. A rendere ulteriormente complessa l'attività di previsione della traiettoria dei veicoli si aggiunge anche la necessità di algoritmi efficienti in grado di gestire rapidamente grandi volumi di dati a bassa latenza. Infine, l'accurata predizione della traiettoria dei veicoli si basa in larga misura su serie di dati di alta qualità e diversificati, la cui acquisizione, soprattutto in contesti dinamici come l'ambiente urbano, rappresenta una criticità non trascurabile.

1.1 Obiettivi e Organizzazione

Questo lavoro di Tesi è il risultato di un periodo di mobilità all'interno del progetto Erasmus, e della collaborazione tra il laboratorio del Professore Noel Crespi (DICE Lab, Telecom SudParis, Parigi - Francia) e il laboratorio del Professore Giancarlo Fortino (SPEME Lab, Unical, Rende - Italia) e si concentra su tre obiettivi fondamentali: il miglioramento della sicurezza stradale, l'ottimizzazione dei flussi veicolari e la pianificazione efficiente dei percorsi degli AV. In questa direzione, è stato esplorato l'utilizzo delle Recurrent Neural Networks (RNN), nello specifico le Long-Short-Term-Memory (LSTM), per la predizione della traiettoria dei veicoli nel contesto urbano delle Smart City. Le RNN sono un tipo di rete neurale artificiale alimentate da dati sequenziali o serie temporali. Questi algoritmi sono comunemente utilizzati per problemi ordinali o temporali, come la traduzione linguistica, l'elaborazione del linguaggio naturale (NLP) e il riconoscimento vocale. Come le feed-forward networks o le reti convoluzionali, anche le RNN sono caratterizzate da una fase di apprendimento, ma differentemente dalle reti feed-forward e dalle reti convoluzionali, le RNN possiedono una memoria interna che gli consente di mantenere traccia dell'influenza che l'input ha sull'output. Inoltre, mentre le Deep Neural networks assumono che input ed output siano tra loro indipendenti, l'output delle RNN dipende strettamente dai precedenti elementi della sequenza di input.
Inoltre, questo lavoro di tesi ha sviluppato un approccio innovativo per risolvere la complessa sfida dell'acquisizione dei dati. Considerata la difficoltà nell'individuare un dataset open source relativo al contesto di riferimento, si è deciso di applicare un algoritmo di rilevazione di oggetti, nello specifico YOLOv8, a video inerenti a un incrocio stradale caratterizzato da un elevato volume traffico: l'uso di questa tecnica ha portato alla creazione di un dataset reale e rappresentativo del contesto preso in esame. Questa strategia ha consentito di superare le limitazioni legate alla disponibilità di dataset preesistenti e di ottenere dati di alta qualità per lo sviluppo di modelli di deep learning.

Il lavoro di tesi è strutturato come segue. Nel **secondo capitolo**, verrà esaminato lo stato dell'arte relativo al problema della predizione della traiettoria dei veicoli, con un'analisi delle principali soluzioni e delle sfide attuali. Inoltre, verranno esplorati i concetti teorici alla base delle tecnologie utilizzate nella creazione dei modelli di Machine Learning. Nel **terzo capitolo**, saranno illustrati i framework, le tecnologie e i software utilizzati nello sviluppo dei suddetti modelli. Nel **quarto capitolo**, si procederà con una dettagliata descrizione dei modelli di Machine Learning sviluppati e delle scelte implementative adottate. Il **quinto capitolo** sarà dedicato a un'analisi completa dei risultati ottenuti dalla validazione e dall'inferenza dei modelli. Infine, nel **sesto capitolo**, verranno esposte le conclusioni e le direzioni future per lo sviluppo del sistema.

Capitolo 2

Stato dell'arte

Le **Reti Neurali Ricorrenti** (RNN) si sono imposte come un pilastro fondamentale per l'analisi e la previsione di sequenze temporali di dati. Tra queste, le **Long Short-Term Memory** (LSTM) affrontano le sfide delle tradizionali RNNs e aprono la strada a nuovi orizzonti nell'elaborazione delle sequenze temporali. Questo capitolo si propone di esplorare l'attuale stato dell'arte relativo alle RNN, concentrando l'attenzione sulle LSTM e sul loro ruolo cruciale nel miglioramento della mobilità urbana. Saranno quindi analizzati approcci recenti, classi speciali di architetture e metodologie avanzate di predizione della traiettoria dei veicoli.

2.1 Reti neurali ricorrenti - RNNs

Le **RNNs** sono un tipo di rete neurale artificiale alimentate mediante dati sequenziali o serie temporali [5]. Questi algoritmi di **deep learning** sono comunemente usati per problemi di tipo ordinale o temporale, come la traduzione linguistica, e i vari temi legati al **natural language processing** (NLP) [6]; sono integrati in applicazioni popolari come l'assistente vocale di Apple, e in Google Translate.

Le RNNs si distinguono all'interno del panorama delle reti neurali, per la loro capacità di memorizzare le informazioni, relative al processamento degli input a tempo t_j, per influenzare l'input e l'output a tempo t_i, con $j << i$, mentre le tradizionali reti neurali presuppongono che input e output siano tra loro indipendenti. Ad ogni passo temporale t_i, detto anche frame, i neuroni, ovvero gli elementi costitutivi delle RNNs, ricevono in input i dati e l'output prodotto al passo temporale precedente, rispettivamente: x_t e y_{t-1}. Dal momento che non esiste alcun neurone per $t < 0$, y_{-1} è generalmente inizializzato a 0. Il funzionamento di questa rete è mostrato dalla figura 2.1.
Ogni neurone ricorrente è definito da due pesi: W_x per x_t e W_y per y_{t-1}. Questi vettori sono utilizzati per calcolare l'output tramite l'equazione seguente:

$$y_t = \theta(W_x^t \cdot x_t + W_y^t \cdot y_{t-1} + b)$$

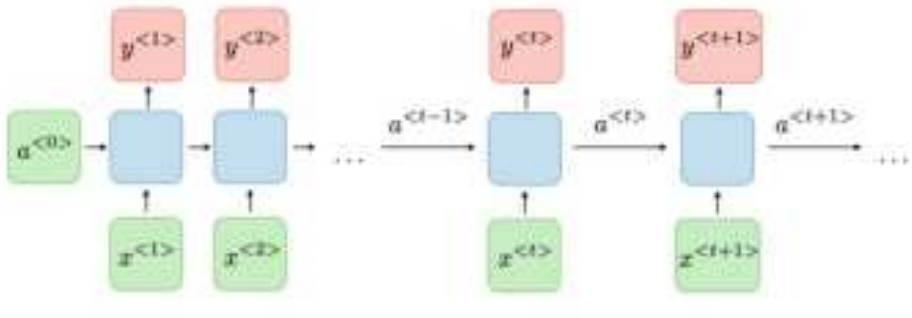


Figura 2.1: *Architettura di una RNN*

Le RNNs unidirezionali, ovvero il tipo di rete sin'ora presentato, non sono in grado di catturare l'influenza che gli eventi futuri hanno sull'output. Per tale motivo, sono state sviluppate le **RNNs bidirezionali** [7], che rappresentano un'importante innovazione nel campo dell'apprendimento automatico, con implicazioni significative per l'analisi e la comprensione di dati sequenziali complessi, come il linguaggio naturale o le serie temporali. Il processamento bidirezionale permette alle reti di apprendere le relazioni tra gli elementi passati e futuri di una sequenza e gli attuali input e output. Anche se le reti bidirezionali sono efficaci nella risoluzione di intricati problemi, esse presentano una complessità computazionale elevata, che porta a un aumento dei tempi di addestramento rispetto alle RNNs unidirezionali. Inoltre, poiché vengono considerati sia gli elementi passati che futuri per la previsione dell'output attuale, ciò può portare le reti bidirezionali a inclinarsi verso la memorizzazione di dettagli specifici dei dati di addestramento, con il conseguente aumento del rischio di overfitting. Una ulteriore estensione delle RNNs è rappresentata dalle **Stacked RNNs**, ovvero RNN sovrapposte. In una RNN sovrapposta, ogni strato della rete è composto da unità ricorrenti, spesso implementate utilizzando celle LSTM o GRU per gestire meglio le dipendenze a lungo termine nei dati sequenziali. Gli strati sono impilati uno sopra l'altro, creando una pila di RNN. Le stacked RNNs presentano numerosi vantaggi che fanno capo a un aumento delle capacità di apprendimento della rete, tuttavia la complessità dell'architettura allunga i tempi di addestramento ed espone la rete a problemi di overfitting e ai fenomeni di esplosione del gradiente e vanishing gradient che saranno approfonditi successivamente. Un ulteriore aspetto distintivo delle RNNs consiste nella condivisione dei parametri lungo ciascun livello della architettura della rete. A differenza delle reti feed-forward, in cui ciascun nodo possiede un proprio peso, le RNNs adottano un approccio in cui i medesimi parametri di peso sono condivisi all'interno di ogni strato della rete. Le RNNs sfruttano l'algoritmo di **backpropagation nel tempo** (Backpropagation through time, BPTT) per determinare i gradienti, che differisce dalla backpropagation tradizionale, in quanto è specifico per le sequenze temporali di dati. Proprio come nella normale backpropagation, c'è un primo passaggio in avanti attraverso lo srotolamento della rete (forward

pass). Quindi, viene valutata la sequenza di output utilizzando una funzione di costo $C(Y_0, Y_1, ..., YT)$. Successivamente, i gradienti della funzione di costo sono propagati all'indietro. Infine vengono aggiornati i parametri del modello utilizzando i gradienti calcolati durante BPTT. Si noti che i gradienti scorrono all'indietro attraverso tutti gli output utilizzati dalla funzione di costo. Ad esempio, nella figura 2.2, la funzione di costo viene calcolata utilizzando gli ultimi tre risultati della rete $(Y(2), Y(3), Y(4))$ quindi i gradienti fluiscono attraverso questi tre output ma non attraverso $Y(0), Y(1)$.

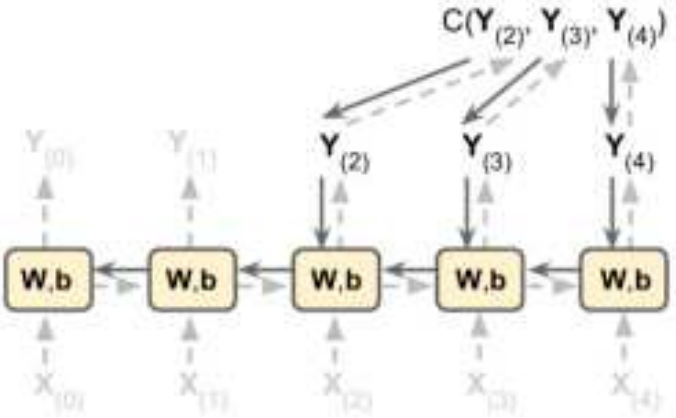


Figura 2.2: *BPTT*

Durante questo processo di backpropagation, le RNNs tendono a riscontrare due tipi di problemi, noti come: **Vanishing Gradient** ed **Esplosione dei gradienti**. Il **vanishing Gradient problem**, o svanizione del gradiente, è quel fenomeno che, a causa della prossimità del valore del gradiente a zero, rende difficile l'addestramento delle RNNs limitando la loro capacità di catturare dipendenze temporali a lungo raggio. Il fenomeno dell'**esplosione dei gradienti** si verifica invece, quando i gradienti sono troppo grandi. Ciò può causare problemi di stabilità durante l'addestramento e portare a oscillazioni e instabilità nei pesi della rete.

Nel panorama delle RNNs emergono differenti architetture, ciascuna delle quali progettata per adattarsi a specifici contesti applicativi. Un esempio è rappresentato dalle reti di tipo **Many-To-One** (come illustrato nella figura 2.3), che trovano ampio impiego nell'ambito della sentiment analysis. Questa architettura è stata ottimizzata per affrontare con successo problemi che coinvolgono l'analisi e la comprensione delle emozioni e delle opinioni espresse in testi, consentendo un'efficace categorizzazione dei contenuti in base alle sfumature emotive.

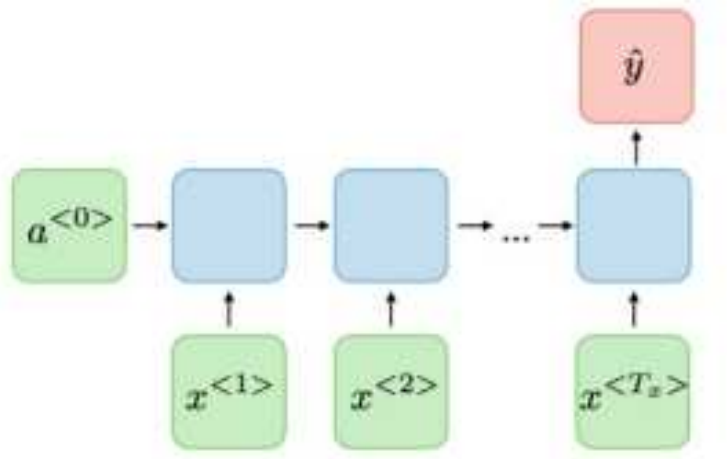


Figura 2.3: *Architettura di una RNN Many-To-One*

Sebbene esistano una moltitudine di architetture RNN, questo lavoro si concentra su quelle di tipo **Many-To-Many** (illustrata nella figura 2.4), ovvero l'architettura implementata nello sviluppo dei modelli di predizione della traiettoria di ciascun veicolo.

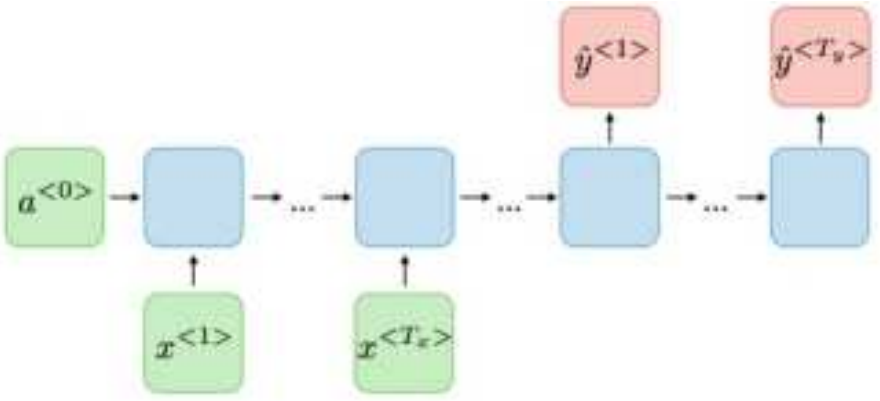


Figura 2.4: *Architettura di una RNN Many-To-Many*

2.2 Long Short Term Memory - LSTM

Le **LSTMs** sono un tipo speciale di RNN capace di catturare le dipendenze tipo long-term tra i dati [8] [9] [10].
Tutte le RNNs sono caratterizzate da una catena di moduli ripetuti detti unità. Ognuno di questi, è una rete neurale di tipo feedforward come evidenziato dalla figura 2.5. Nelle RNNs, le unità hanno una struttura molto semplice, in quanto costituite da un singolo layer *tanh*, mentre le unità LSTM sono costituite da quattro layer feed-forward che interagiscono tra loro come mostrato dalla figura 2.6.

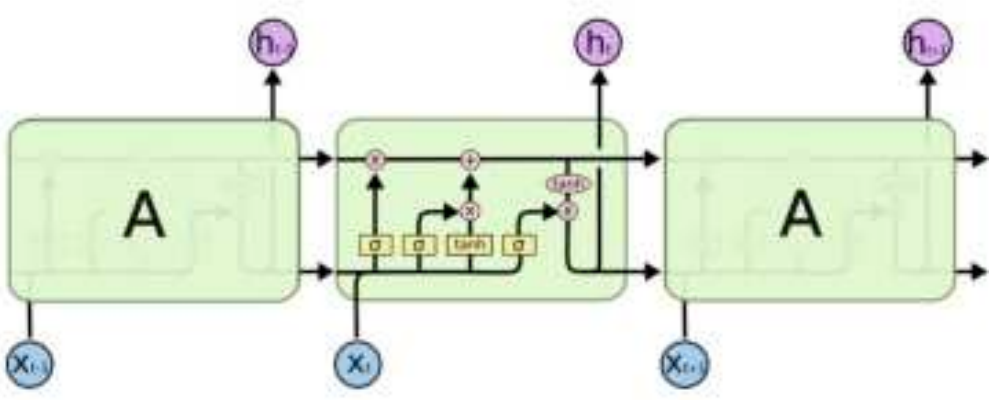


Figura 2.5: *Struttura a catena delle LSTM*

Nelle unità LSTM, tre delle quattro reti di tipo feed-forward svolgono un ruolo cruciale nella selezione e nella gestione delle informazioni. Queste sono il **forget gate**, **input gate** e **output gate**. Ciascuno di questi tre gate è responsabile di specifiche operazioni di gestione della memoria all'interno delle celle LSTM, che possono essere riassunte come segue:

1. Il **forget gate** è utilizzato per eliminare o scartare informazioni dalla memoria delle celle LSTM.

2. L'**input gate** è responsabile della memorizzazione di nuove informazioni nella memoria delle celle LSTM.

3. L'**output gate** regola l'accesso e l'uso delle informazioni presenti nella memoria delle celle LSTM.

Questi elementi di controllo sono comunemente denominati **gate controllers**, poiché producono output nell'intervallo tra 0 e 1 svolgendo un ruolo cruciale nel regolare il flusso delle informazioni all'interno delle unità LSTM. L'output di ciascun gate determina quanto sia importante conservare o scartare specifiche informazioni. In aggiunta a questi gate, esiste anche una quarta rete denominata **candidate memory**. Questa rete ha la responsabilità di generare un vettore contenente l'informazione candidata per

la memorizzazione. Tale vettore rappresenta l'informazione potenziale che potrebbe essere conservata nella memoria delle celle LSTM. Ogni unità LSTM riceve tre vettori in ingresso, come illustrato nella figura 2.6. Due di questi vettori, noti come **cell state** ($c_{t-1} \in R^h$) e **hidden state** ($h_{t-1} \in (-1,1)^h$), sono generati all'istante temporale t_{-1}. Questi due vettori provengono dall'uscita dell'unità LSTM precedente.

Il terzo vettore, $x_t \in R^d$, rappresenta il vettore di input che viene fornito alla rete all'istante temporale corrente. È importante notare che d si riferisce al numero di feature dell'input, mentre h si riferisce al numero di unità nascoste (hidden units).

Le unità LSTM regolano attraverso i gates il flusso interno delle informazioni e trasformano costantemente i valori dei vettori cell state e dell'hidden state.

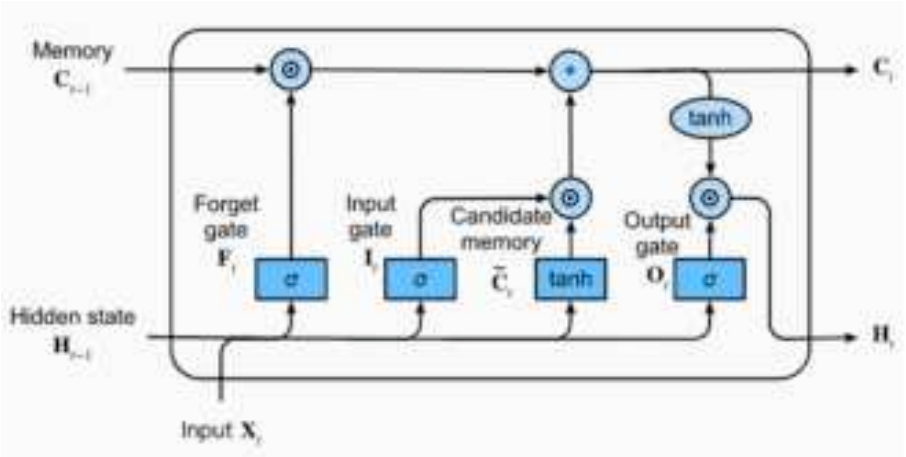


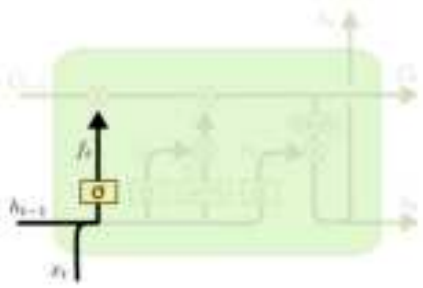
Figura 2.6: *Architettura unità LSTM*

2.2.1 Hidden state e Cell state

L'**hidden state** rappresenta la comprensione della sequenza di input fino all'istante temporale corrente. Esso, racchiude le informazioni ritenute rilevanti dall'LSTM per la predizione dell'output. Il **cell state**, rappresenta invece le informazioni che la rete LSTM ha appreso e che ha deciso di memorizzare e propagare nel tempo. In sintesi, il cell state agisce come una memoria interna che conserva le informazioni nel tempo e ha la capacità di catturare le dipendenze a lungo termine nei dati sequenziali. L'hidden state invece, è un output che rappresenta la comprensione dei dati da parte dell'LSTM fino all'istante temporale corrente ed è utilizzato per generare previsioni o inviare informazioni ad altre unità della rete. Questi due vettori consentono alle LSTM di modellare ed elaborare efficacemente dati sequenziali con meccanismi di memoria a lungo termine.

2.2.2 Forget gate

La prima attività di un'unità LSTM è il calcolo del **forget gate**. Tale gate decide, sulla base delle informazioni x_t e h_t il quantitativo di informazione da eliminare da c_{t-1}.



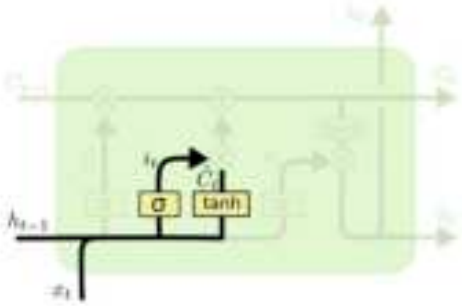
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Figura 2.7: *Forget gate LSTM*

2.2.3 Input gate e Candidate memory

Dopo aver filtrato e rimosso una parte dell'informazione ricevuta in input (c_{t-1}) il passo successivo coinvolge l'acquisizione di nuove informazioni. Questa fase è gestita da due reti neurali specifiche: la **candidate memory** e **l'input gate**. Quest'ultime, ricevono in ingresso sia x_t che h_{t-1}, i quali vengono concatenati al fine di formare un unico vettore consistente.

La **candidate memory** ($\tilde{C}_t$) è incaricata della generazione di un vettore candidato, ovvero un vettore contenente informazioni che potrebbero essere selezionate per essere memorizzate in c_t. L'**input gate** (i_t) ha il compito di generare un vettore selettore che verrà moltiplicato elemento per elemento con la candidate memory. Il risultato di questa operazione sarà sommato a c_t. In questo modo, vengono aggiunte nuove informazioni al cell state (c_t) (Figura 2.8).



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

Figura 2.8: *Calcolo dei vettori $\tilde{C}_t$ e i_t*

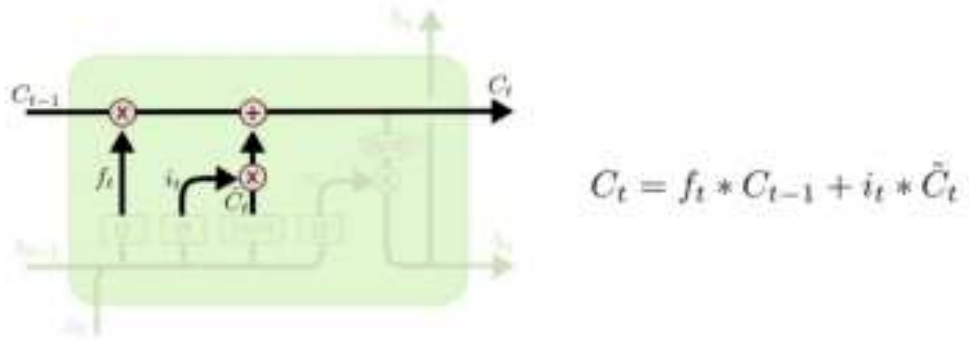


Figura 2.9: *Aggiunta di $\tilde{C}_t$ a c_t*

2.2.4 Output Gate

L'**output gate** determina il valore dell'hidden state emesso dall'unità LSTM, nell'istante t, e ricevuto dall'unità LSTM nell'istante successivo $t+1$. Anche la generazione dell'output è eseguita mediante il prodotto tra il vettore selettore e un vettore candidato. In tal caso, però, il vettore candidato non è generato da una rete neurale, ma è ottenuto semplicemente utilizzando la funzione *tanh* su c_t. Questo passaggio normalizza c_t in un intervallo $(-1, 1)$.

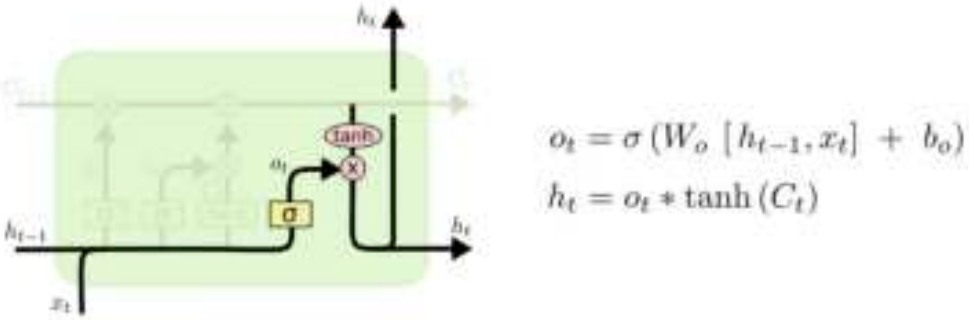


Figura 2.10: *Output gate LSTM*

2.3 Encoder-Decoder LSTM

I modelli **seq2seq** (sequence to sequence) sono una classe speciale di architetture di RNNs tipicamente utilizzate per risolvere problemi i cui obiettivi sono tradurre o trasformare una sequenza di dati di input in una sequenza di dati di output. Questo tipo di modelli si applica a una vasta gamma di domini, ed è utilizzato in molte applicazioni del campo dell'elaborazione del linguaggio naturale e dell'elaborazione delle sequenze di dati. I seq2seq models, sono noti anche con il nome **Encoder-Decoder**, in quanto costituiti da due componenti: l'encoder e il decoder.

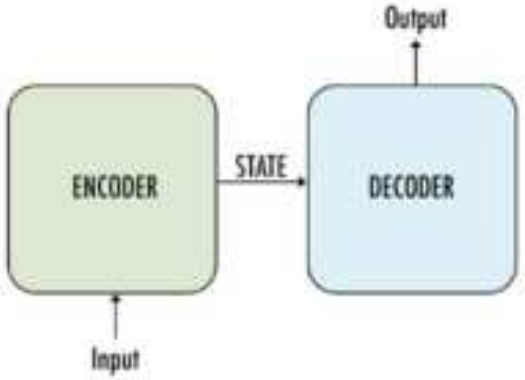


Figura 2.11: *Architettura modelli Seq2seq*

Tipicamente sia l'encoder che il decoder sono reti neuruali di tipo LSTM, ma talvolta possono essere implementati come GRU, che verranno approfondite nel paragrafo successivo. Indipendentemente dall'implementazione specifica, l'encoder riceve in ingresso la sequenza di dati di input e la converte in una rappresentazione vettoriale di dimensione fissa chiamata vettore latente o context vector, che cattura le informazioni chiave dalla sequenza di input. Il decoder, invece, riceve in ingresso il context vector proveniente dall'encoder, e lo utilizza per generare una sequenza di output passo dopo passo.

L'implementazione delle componenti encoder e decoder mediante LSTM consente di gestire sequenze di dati di lunghezza variabile in modo efficace ed efficiente. Le LSTM infatti, sono in grado di catturare le dipendenze a lungo termine all'interno delle sequenze di dati, il che è fondamentale per i problemi seq2seq. Nello specifico, encoder e decoder LSTM si decompongono in una sequenza di unità LSTM come mostrato dalla figura 2.12.

Ogni unità dell'encoder riceve in input x_i, insieme all'hidden state h_{t-1} e al cell state c_{t-1} provenienti dall'unità LSTM precedente. È importante notare che i vettori h e c sono inizializzati come vettori nulli a t_0. Una volta calcolati i vettori h_t e c_t che rappresentano gli output dell'ultima unità LSTM dell'encoder, essi vengono utilizzati per l'inizializzazione dei corrispondenti vettori relativi alla prima unità LSTM del decoder.

Il decoder, similmente all'encoder, è composto da una sequenza di unità LSTM, ognuna delle quali, ricevendo in ingresso l'hidden state e il cell state dalla cella precedente, oltre a un vettore denominato Y, predice l'output $i-esimo$. È importante notare che il decoder può essere implementato utilizzando quattro diverse tecniche, che differiscono per la natura del vettore Y.

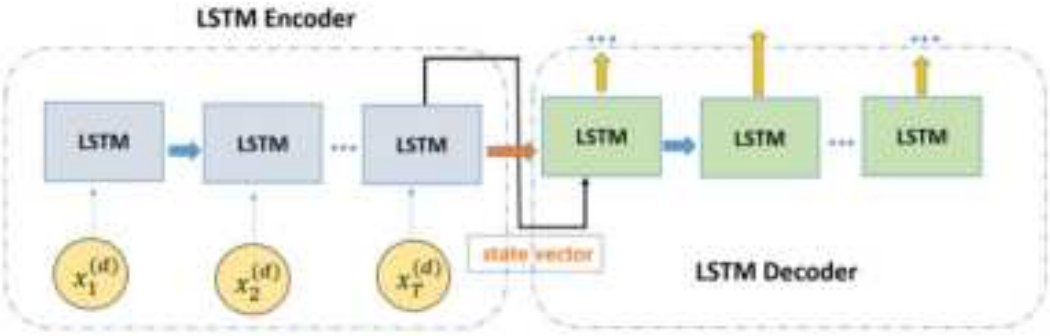


Figura 2.12: *Encoder LSTM*

Recursive decoder

La prima tecnica, chiamata **Recursive** (Ricorsiva), si contraddistingue per l'implementazione di un decoder LSTM in grado di generare l'output senza ricevere alcuna informazione sulla sequenza target (l'output reale). In questa configurazione, il vettore Y è costituito dalla predizione effettuata dall'unità LSTM precedente, e nel caso della prima unità LSTM del decoder esso corrisponde all'input al tempo t, ovvero x_t.

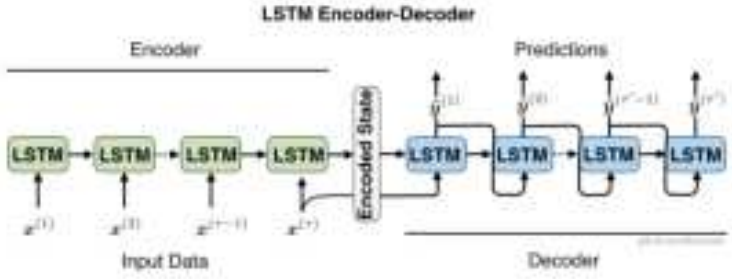


Figura 2.13: *Recursive decoder LSTM*

Teacher Forcing decoder

La seconda tecnica prende il nome di **Teacher Forcing**. In questa implementazione, il decoder riceve la sequenza target come dato di input. Ciò aiuta il modello ad apprendere più velocemente, tuttavia può portare a problemi di overfitting.

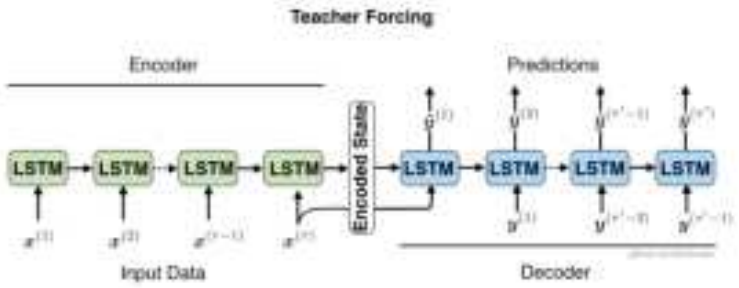


Figura 2.14: *Teacher Forcing decoder LSTM*

Le ultime due tecniche rappresentano delle rielaborazioni della tecnica ricorsiva e del teacher forcing, e possono essere considerate come approcci ibridi che combinano elementi delle tecniche precedenti in modo creativo e flessibile per adattarsi alle specifiche esigenze del problema o dell'applicazione.

Mixed teacher forcing decoder

Nel **Mixed Teacher Forcing** il decoder riceve in input il vettore target solo per alcuni passi temporali e nei rimanenti è alimento con la predizione effettuata al passo precedente. Questo permette di beneficiare dell'informazione reale e allo stesso tempo di far sviluppare al modello una propria autonomia nella generazione dell'output.

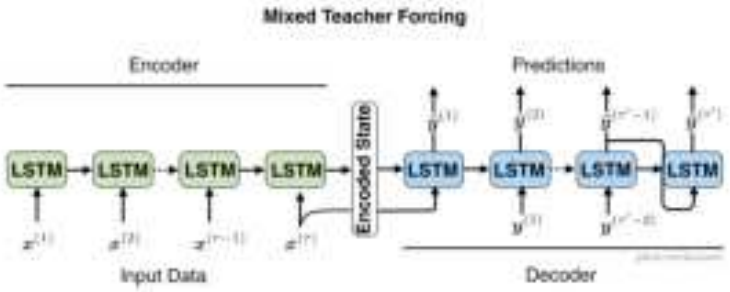


Figura 2.15: *Mixed Teacher Forcing decoder LSTM*

Dynamic teacher forcing decoder

L'ultima tecnica prende il nome di **Dynamic Teacher Forcing**. Con questa implementazione si varia il tasso di utilizzo del Teacher Forcing durante l'addestramento, in base al comportamento del modello. Tipicamente in fase di avvio si favorisce il teacher forcing, per instradare correttamente il modello, mentre per le fasi successive si preferisce utilizzare la tecnica ricorsiva.

È importante notare che, sebbene siano disponibili quattro differenti tecniche implementative del decoder, queste vengono utilizzate esclusivamente nella fase di addestramento del modello. Durante la fase di inferenza, il decoder è implementato utilizzando la tecnica ricorsiva. Questo significa che, quando il modello è utilizzato per fare predizioni in situazioni reali, viene adottata la tecnica ricorsiva per generare le sequenze di output desiderate.

2.4 Attention Mechanisms

Proposto nel 2014 [11], obiettivo principale dell'**attention mechanisms** è permettere ai modelli con architettura Encoder-Decoder di identificare gli elementi più significativi nella sequenza di input al fine di migliorare la predizione degli elementi nella sequenza di output. La figura 2.16, mostra come il meccanismo di attenzione possa essere implementato in una classica architettura Encoder-Decoder. Differentemente dall'approccio tradizionale, a ogni istante di tempo, il decoder riceve in ingresso sia l'hidden state, proveniente dalla precedente cella del decoder (s_{t-1}), che gli output generati da ogni cella dell'encoder. Conseguentemente il decoder, mediante una somma pesata degli input pervenuti, determina l'elemento o gli elementi che più influenzeranno la predizione dell'output all'istante di tempo $i - esimo$. I pesi α, responsabili del procedimento, sono generati da una rete neurale di tipo feed-forward chiamata **alignment model** che viene addestrata congiuntamente al resto del modello Encoder-Decoder.

Questa rete, illustrata nella parte destra della figura 2.16, è caratterizzata da uno strato denso distribuito nel tempo caratterizzato da un singolo neurone che riceve in ingresso gli output dell'encoder, concatenati con il precedente hidden state del decoder. In una rete neurale, uno strato denso è profondamente connesso allo strato, o livello precedente. Ciò significa che ogni neurone dello strato denso è collegato a ciascun neurone del livello precedente. Dunque, uno strato denso distribuito nel tempo, è equivalente a uno strato denso applicato in modo indipendente a ogni istante temporale.

La rete, produce uno score (o energia) per ogni output dell'encoder, il cui valore misura l'allineamento di ogni uscita con il precedente hidden state del decoder. Infine, al vettore contenete gli score è applicata la funzione softmax.

Questo particolare meccanismo di attenzione è chiamato attenzione di Bahdanau. Nello specifico, le componenti principali del meccanismo di attenzione di Bahdanau sono:

1. s_{t-1}, **hidden state** a tempo $t-1$ del decoder.
2. h_t, **hidden state** a tempo t dell'encoder.
3. $\alpha_{t,i}$, **peso** assegnato a h_t.
4. $e_{t,i}$, **attention score** generato dal modello di allineamento.

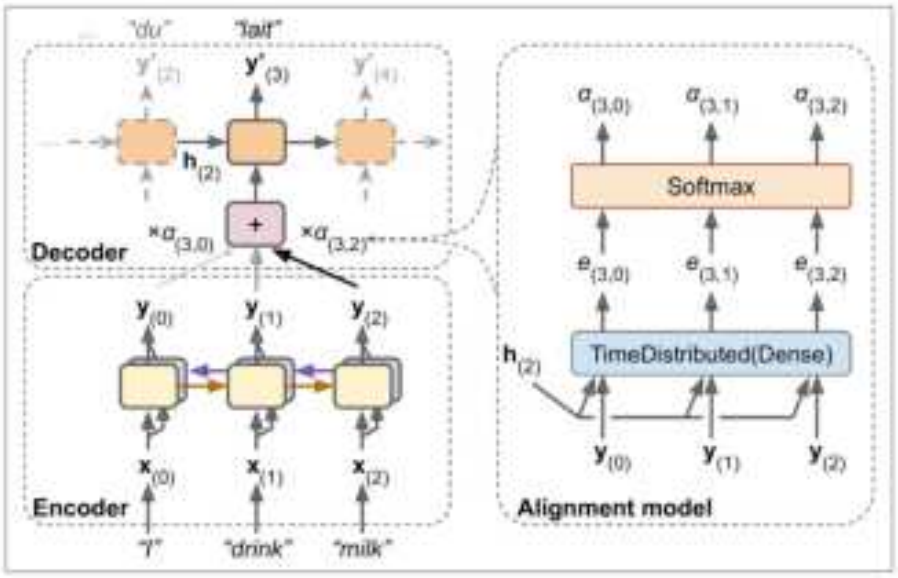


Figura 2.16: *Attention mechanism*

La funzione implementata dal modello di allineamento combina s_{t-1} con h_t mediante un'operazione additiva. Per tale ragione il meccanismo è noto anche con il nome di **additive attention** (attenzione additiva). Tale funzione può essere implementata in due modi differenti: applicando una matrice di pesi a vettori concatenati (1), oppure applicando separatamente ai vettori s_{t-1} e h_i i relativi pesi ed eseguire la concatenazione successivamente (2).

1. $a(s_{t-1}, h_i) = V^T * tanh(W[h_i; s_{t-1}])$
2. $a(s_{t-1}, h_i) = V^T * tanh(W_1 * h_i; W_2 * s_{t-1}])$

dove V^T è un vettore di pesi.

Una volta calcolati gli attention score, si procede applicando la funzione softmax, come rappresentato dalla seguente equazione: $\alpha_{t,i} = softmax(e_{t,i})$.

Questa operazione è fondamentale in quanto normalizza i valori degli attention score in un intervallo compreso tra 0 e 1. Di conseguenza, i pesi ottenuti dall'output della funzione softmax possono essere interpretati come valori di probabilità. Ciascun valore di probabilità (o peso) riflette l'importanza relativa di h_i e s_{t-1} nel processo di generazione dello stato successivo s_t e dell'uscita successiva y_t.
In altre parole, utilizzando la funzione softmax, si attribuisce un peso specifico a ciascun elemento dell'input in modo da riflettere la sua importanza nella generazione dell'output desiderato.

Minh-Thang Luong et al [12] hanno introdotto l'uso del prodotto scalare, per valutare la similarità tra l'output dell'encoder e il precedente hidden state del decoder, nel contesto del meccanismo di attenzione. Questa scelta è stata motivata da diverse ragioni. Innanzitutto, il prodotto scalare è una metrica computazionalmente efficiente per calcolare la similarità tra coppie di vettori. La sua implementazione richiede meno risorse computazionali rispetto ad altre misure di similarità, il che la rende adatta per l'elaborazione in tempo reale e per il training su grandi dataset. Inoltre, il prodotto scalare è una metrica di similarità sufficientemente robusta per molti scenari. Tale tecnica prende il nome di Luong's attention o talvolta **multiplicatative attention** (attenzione moltiplicativa). L'output del prodotto scalare definisce uno score, al quale viene successivamente applicata la funzione softmax.

2.5 Gated Recurrent Units - GRU

Le GRU [13] sono una forma semplificata delle LSTM, in quanto caratterizzate da un numero inferiore di parametri. Tuttavia, sebbene abbiano un'architettura più snella, condividono con le LSTM l'abilità di gestire efficientemente le dipendenze a lungo termine nei dati sequenziali.

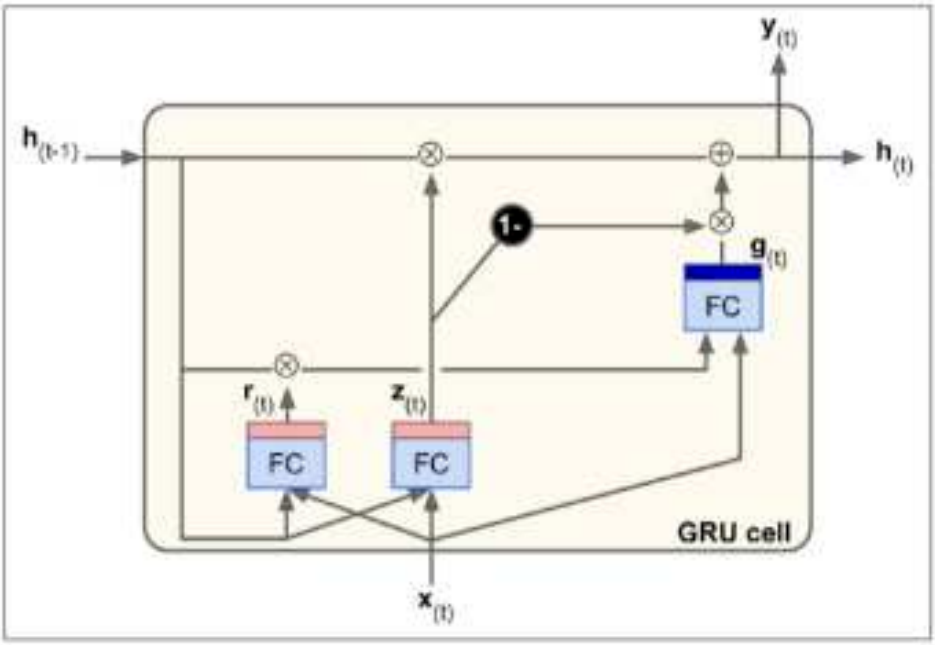


Figura 2.17: *GRU*

Di seguito le semplificazioni introdotte dalle GRUs:

1. I due vettori di stato sono combinati in un singolo vettore: h_t.
2. In una rete GRU, un singolo gate, noto come z_t, è responsabile delle operazioni svolte dal **forget gate** e dall'**input gate**. Quando il gate z_t restituisce il valore 1, significa che il forget gate genera il valore 1 mentre l'input gate genera il valore 0. Inversamente, se il gate z_t dovesse emettere il valore 0, avverrebbe l'esatto opposto. In altre parole, ogni volta che è necessario memorizzare un elemento, la memoria viene prima resettata.
3. È fondamentale notare che nelle reti LSTM non esiste un **output gate** specifico. Tuttavia, c'è un gate di controllo denominato r_t che regola la propagazione dello stato precedente, ottenendo la concatenazione dello stato della cella con lo stato nascosto.

2.6 Metodi di predizione della traiettoria dei veicoli

Nel contesto della mobilità moderna, dove veicoli autonomi e sistemi di assistenza alla guida stanno diventando sempre più comuni, la capacità di prevedere con precisione il comportamento futuro dei veicoli è di fondamentale importanza.

2.6.1 Formulazione del problema e classificazione dei metodi di predizione della traiettoria

Formulazione del problema

I sistemi di predizione della traiettoria dei veicoli sono descritti da modelli che, sfruttando le informazioni passate degli stessi (comunemente denominate "stati"), sono in grado di predire le posizioni future dei veicoli [14]. Indichiamo con $X = \{p^1, p^2, ..., p^{th}\}$ l'insieme degli stati passati del veicolo target, ovvero il veicolo di cui si vuole predire la traiettoria. Nella maggior parte dei modelli di predizione della traiettoria, gli stati consistono in tuple che rappresentano le coordinate (x, y) relative alle posizioni passate del veicolo. In alcuni casi, oltre alle posizioni passate, gli stati possono includere grandezze cinematiche come la velocità e l'accelerazione del veicolo a tempo t. Inoltre, sia $Y = \{p^{th+1}, p^{th+2}, ..., p^{th+n}\}$ l'output del problema. Alcuni modelli sono in grado di generare Y in modo diretto e altri mediante la definizione di risultati intermedi.

Classificazione dei metodi

Come evidenziato dalla figura 2.18, esistono 4 classi di metodi risolutivi:

1. **Physic based methods**. Essi descrivono il movimento dei veicoli mediante modelli cinematici e dinamici. Tali metodi sono molto flessibili in quanto applicabili a una varietà di scenari con costi computazionali ridotti e senza la necessità di una fase di training. Tuttavia, essendo i modelli incapaci di descrivere le interazioni tra i veicoli, i Physic based methods sono limitati all'applicazione in scenari statici con previsione a breve termine. Dunque, grazie alla loro semplicità e rapidità di risposta, questi metodi possono essere facilmente utilizzati in applicazioni come l'analisi del rischio di collisione.

2. **Classic Machine learning based methods**. Rispetto ai physic based method, tali metodi hanno una accuracy relativamente elevata. Inoltre, sono applicabili a contesti long-term, ma con un costo computazionale più elevato. La maggior parte di questi metodi sono di tipo "maneuver-based", cioè si basano sulla previsione della traiettoria del veicolo tenendo conto delle manovre passate del veicolo. Tuttavia, è importante notare che le manovre dei veicoli possono essere estremamente varie e diversificate, e questo rende la capacità di generalizzazione di tali modelli piuttosto limitata. Ciò significa che i modelli maneuver-based potrebbero non essere in grado di gestire efficacemente situazioni in cui le manovre del veicolo non rientrano nei pattern precedentemente appresi. Un'applicazione concreta in cui questi metodi vengono utilizzati è lo studio del cambio di corsia.

3. **Deep learning based methods**. I metodi tradizionali di predizione della traiettoria degli AV sono adatti solo per scene semplici e previsioni a breve termine. I deep learning based methods sono in grado di effettuare previsioni accurate su un orizzonte temporale più ampio. Tuttavia, è importante sottolineare che tali metodologie richiedono una vasta quantità di dati per l'addestramento, e i costi computazionali e i tempi di elaborazione aumentano in modo significativo all'aumentare della complessità della modellazione. Un vantaggio notevole di questi metodi è la loro capacità di poter generare traiettorie multimodali, tenendo conto della diversità delle manovre dei veicoli. Questo significa che possono prevedere una gamma di possibili traiettorie anziché limitarsi a una singola previsione, il che è particolarmente utile in scenari in cui i comportamenti dei veicoli possono variare ampiamente. Tuttavia, è importante sottolineare che, nonostante il loro potenziale, l'accuratezza delle previsioni rimane legata all'accesso a dati di alta qualità e all'adeguato addestramento dei modelli. Numerosi studi hanno adottato con successo approcci di deep learning per la risoluzione del problema [15] [16]. Ad esempio Deo e Trivedi [4], usano una Convolutional social pooling network combinata con reti LSTM per la predizione della traiettoria dei veicoli nel contesto autostradale. Kim et al [3] propongono invece, una rete LSTM in grado di predire le traiettorie dei veicoli usando una occupancy grid per caratterizzare l'ambiente circostante ai veicoli. Mentre Lin et al [2] propongono un modello LSTM con modulo di attenzione spazio temporale. Tale modello non solo raggiunge risultati comparabili allo stato dell'arte, ma è in grado di spiegare anche l'influenza delle traiettorie storiche e delle posizioni dei veicoli, nella predizione della traiettoria del veicolo target.

4. **Reinforcement learning based methods**. Tali metodi generano traiettorie con una maggiore precisione rispetto ai metodi di deep learning, in un dominio temporale più ampio. I reinforcement learning based methods possono evolvere continuamente attraverso l'apprendimento e adattarsi ad ambienti complessi e a scenari di tipo long-term. Tuttavia, la maggior parte di questi metodi sono tipicamente computazionalmente costosi e richiedono lunghi tempi di addestramento. Nelle applicazioni reali per gli AV, i metodi di previsione della traiettoria basati sull'reinforcement learning sono maggiormente applicati alla pianificazione della traiettoria.

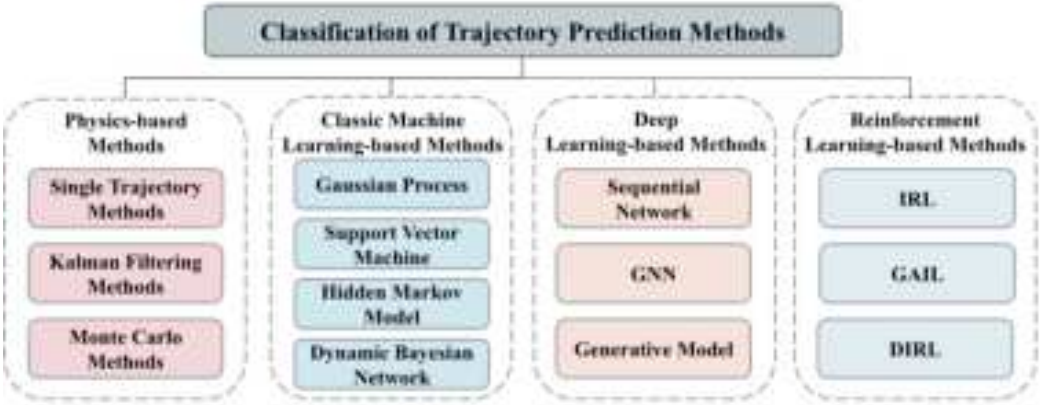


Figura 2.18: *Tassonomia dei metodi di predizione delle traiettorie dei veicoli*

È possibile classificare i sistemi di predizione delle traiettorie dei veicoli in base a diversi fattori, tra cui il tipo di informazioni contenute negli stati e il tipo di output generato dai modelli. In particolare, è possibile distinguere in base all'input: sistemi basati su dati cinematici (**Physic related factors**), sistemi basati su dati relativi al traffico e alle caratteristiche stradali come il numero di corsie, presenza o assenza di impianti semaforici, attraversamenti pedoanali, etc (**Road related factors**). Infine sistemi basati su stati rappresentanti le norme sociali e le interdipendenze tra i veicoli (**Interaction related factors**). L'integrazione di queste diverse categorie di informazioni può migliorare la capacità di previsione delle traiettorie dei veicoli e contribuire a una guida autonoma più sicura ed efficiente. Tuttavia, richiede una modellazione complessa e l'addestramento dei modelli su dati diversificati per ottenere previsioni accurate in una vasta gamma di scenari stradali.

Per quanto concerne l'output, esso può essere unimodale o multi modale e inoltre, alcuni sistemi sono in grado di restituire anche l'intenzione dei veicoli nel eseguire una determinata manovra piuttosto che un altra. È dunque possibile classificare i metodi in tre classi distinte:

1. **Unimodal trajectory**, dove l'output è la traiettoria futura del veicolo target.
2. **Multimodal trajectory**, dove l'output è un insieme di possibili traiettorie. Ogni traiettoria è caratterizzata da una probabilità.
3. **Intention**, ovvero metodi che producono intenzioni di comportamento a supporto della previsione. Le intenzioni possono essere parte dell'output o dati intermedi dai quali generare le traiettorie (modelli indiretti).

Figura 2.19: *Fattori di input ed output per il problema di predizione delle traiettorie dei veicoli.*

2.6.2 Vehicle Trajectory Prediction Using LSTMs With Spatial–Temporal Attention Mechanisms

Overview

É proposto un modello di deep learning, basato su LSTM e moduli di attenzione spazio temporale, di tipo unimodale e interaction related factor based. STA-LSTM, oltre ad avere delle ottime prestazioni, è in grado di spiegare accuratamente l'influenza delle traiettorie storiche del veicolo target, e dei vicini mediante la definizione di pesi d'attenzione. Questo modello, denominato STA-LSTM, non solo offre prestazioni eccellenti ma è anche in grado di spiegare in modo accurato come le traiettorie storiche del veicolo target e dei veicoli circostanti influenzino il processo decisionale. Questa spiegazione è resa possibile dalla definizione di pesi d'attenzione, noti come "attention weights". Viene quindi condotta un'analisi dettagliata dei pesi del meccanismo di attenzione, appresi in diversi scenari autostradali, tenendo conto delle varie tipologie di veicoli e dei fattori ambientali variabili.

Dataset

L'addestramento del modello LSTM è stato effettuato impiegando il dataset NGSI, abbreviazione di Next Generation Simulation (NGSIM) [17]. Questo dataset contiene la traiettoria dei veicoli che hanno attraversato due diverse tipologie di strade americane: la U.S. Highway 101 (US-101) e l'Interstate 80 (I-80). I dati relativi all'US-101 e all'I-80 sono stati registrati a una frequenza di 10 hz per un periodo complessivo di 45 minuti. Questo intervallo temporale è stato poi suddiviso in tre sottoinsiemi di 15 minuti, ciascuno dei quali è stato registrato in un diverso istante temporale. Questa suddivisione ha portato alla creazione di sei dataset distinti.
Per scopi di addestramento e valutazione, tali dataset sono stati suddivisi ulteriormente in tre insiemi separati: il set di addestramento (Training), il set di validazione (Validation) e il set di test (Test). La proporzione utilizzata per questa divisione è stata del 70% per il training set, del 10% per il validation set e del 20% per il test set.

Metodologia

Gli autori, seguendo l'approccio proposto da Deo e Trivedi [4], eseguono una discretizzazione dello spazio attorno al veicolo target. Tale operazione è svolta mediante la definizione di una griglia di dimensione 3×13 centrata nel veicolo target. Le righe della matrice rappresentano rispettivamente: la corsia sinistra, la corsia centrale e la corsia destra della strada di riferimento (US-101 o I-80 a seconda del dataset). Le colonne rappresentano invece, le celle della griglia ognuna delle quali ha una larghezza di 4.6 metri. All'interno di tale struttura, vengono individuati e identificati i veicoli che si trovano nelle vicinanze del veicolo target. Ogni veicolo vicino viene associato a una cella specifica della griglia in base alla posizione del suo paraurti anteriore. Questo metodo consente di determinare in modo accurato la posizione dei veicoli circostanti al veicolo target.

Il modello riceve in input le informazioni di tutti i veicoli che si trovano all'interno della griglia, fino all'istante temporale t. Durante il processo di previsione della traiettoria, il modello calcola i pesi d'attenzione (attention weights) sia a livello temporale che spaziale. In particolare, i pesi d'attenzione temporale (temporal attention weights) sono utilizzati per valutare l'importanza delle traiettorie passate, del veicolo target e dei veicoli circostanti, nella previsione. D'altra parte, i pesi d'attenzione spaziale (spatial attention weights) vengono utilizzati per analizzare come la disposizione spaziale dei veicoli circostanti influenzi la predizione della traiettoria del veicolo target. Nello specifico, il modello riceve in ingresso un vettore rappresentante le 15 posizioni passate del veicolo target con un time-step di 0.2 secondi al fine di predire un vettore contenete le 5 posizioni future che verranno occupate dal veicolo target con il medesimo time-step. L'obiettivo è la minimizzazione della seguente funzione:

$$\min \frac{1}{N_{train}} * \sum_{i=1}^{N_{train}} \sum_{j=1}^{H} (X_j^i - X_j^i)^2$$

Temporal-Level Attention

I Temporal-Level Attention weights associati al veicolo target v, $A_t^v = \{\alpha_{t-T+1}^v, ...\alpha_j^v, \alpha_t^v\}$ sono calcolati come segue:

$$A_t^v = softmax(tanh(W_\alpha * S_t^v)), A_t^v \in R^{1*T}, W_\alpha \in R^{1*d}$$

dove W_α sono ottenuti da una rete neurale feed-forward. Gli hidden states s e i pesi α sono combinati al fine di derivare il tensore associato alla cella del veicolo target v:

$$H_t^v = S_t^v * (A_t^v)^T = \sum_{j=t-T+1}^{t} \alpha_j^v * h_t^v \in R^{d*1}$$

I valori delle celle tensoriali sono utilizzati per calcolare i pesi di attenzione a livello spaziale e prevedere la traiettoria del veicolo.

Spatial-Level Attention

Si deriva un vettore contenente i tensori associati alle celle della griglia,
$G_t = \{G_t^1, ..., G_t^n, ..., G_t^N\}, G_t \in R^{d*N}, G_t^n \in R^{d*1}$, con N pari a 39 ($3 \times 13$). G_t^n assume la seguente forma:

$$G_t^n = \begin{cases} H_t^v, & \text{if il veicolo v è localizzato nella cella n} \\ 0 \in R^{d*1}, & \text{else} \end{cases}$$

Gli spatial level attention weights, associati ai veicoli a tempo t ($B_t = \{\beta_t^1, ...\beta_t^2, \beta_t^N\}$) sono calcolati come segue:

$$B_t = softmax(tanh(W_\beta * G_t)), W_\beta \in R^{1*N}$$

Dove, W_β è un vettore di pesi ottenuti da una rete feed-forward, come visto precedentemente nel caso dell'attenzione temporale. Infine, sono combinate le informazioni del veicolo target e dei vicini come segue:

$$V_t = G_t * (B_t)^T = \sum_{n=1}^{N} \beta_t^n * G_t^n$$

Inoltre, una rete di tipo feed-forward viene successivamente alimentata con V_t, per predire la traiettoria del veicolo target. L'output di questa rete consiste nella definizione di un vettore di coordinate, rappresentanti le posizioni che verranno occupate dal veicolo. La figura sottostante riporta il funzionamento del modello.

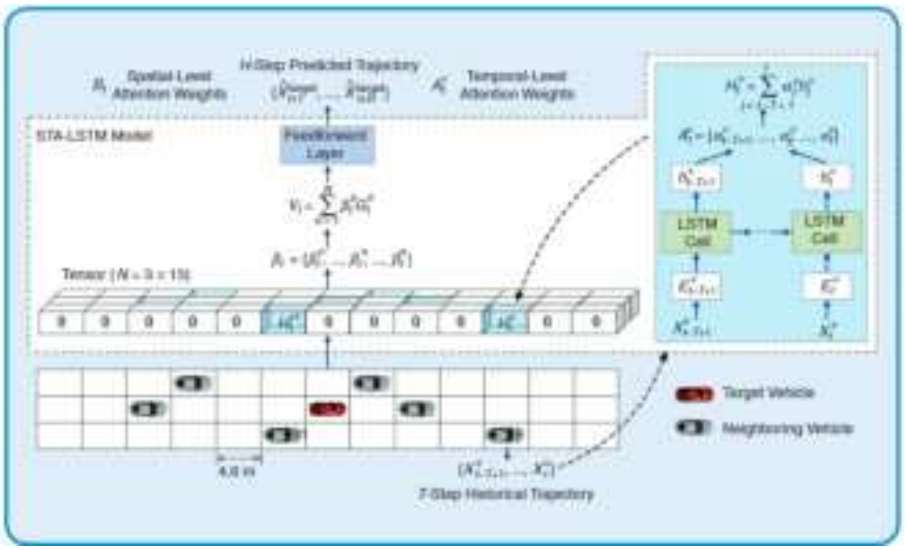


Figura 2.20: *Modello STA-LSTM.*

Analisi di Temporal Attention Weights

Dall'analisi del meccanimo di attenzione temporale, emerge che i pesi relativi a istanti temporali precedenti a $t-5$ hanno una influenza trascurabile nella predizione della traiettoria dei veicoli. Il peso massimo è invece assunto di dati relativi all'istante temporale t. Ciò suggerisce che la traiettoria futura del veicolo target è principalmente influenzata dai dati più recenti.

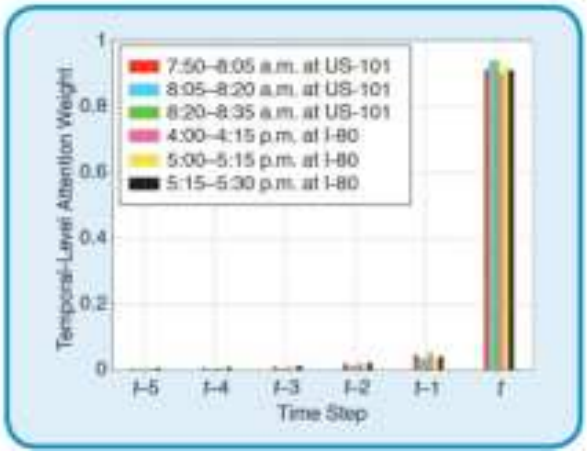


Figura 2.21: *Temporal attention weights.*

Analisi degli Spatial Attention Weights

Dall'analisi dei pesi relativi al meccanismo di attenzione spaziale, per classe di veicolo, emerge che la cella del veicolo target ha il peso maggiore. Inoltre, si osserva che tale valore è più alto per i veicoli pesanti. Questo risultato, combinato con la precedente analisi dell'attenzione a livello temporale, rivela che la traiettoria futura del veicolo target dipende in larga misura stato corrente. L'influenza maggiore di un autocarro può essere dovuta al fatto che questi veicoli necessitano di un tempo più lungo per reagire a stimoli esterni (veicoli vicini). Dunque, per tale motivo la loro posizione gioca un ruolo fondamentale nella previsione della traiettoria. In tutti i casi le celle antecedenti la cella contenente il veicolo target assumono valori nulli, a dimostrazione dell'influenza trascurabile che queste posizioni hanno nella predizione della traiettoria del veicolo target. Per le auto i pesi con maggior valore si trovano all'interno delle celle (Current, 2), (Current, 3), e (Current, 4). Invece, per i veicoli pesanti in (Current, 3), (Current, 4), e (Current, 5).

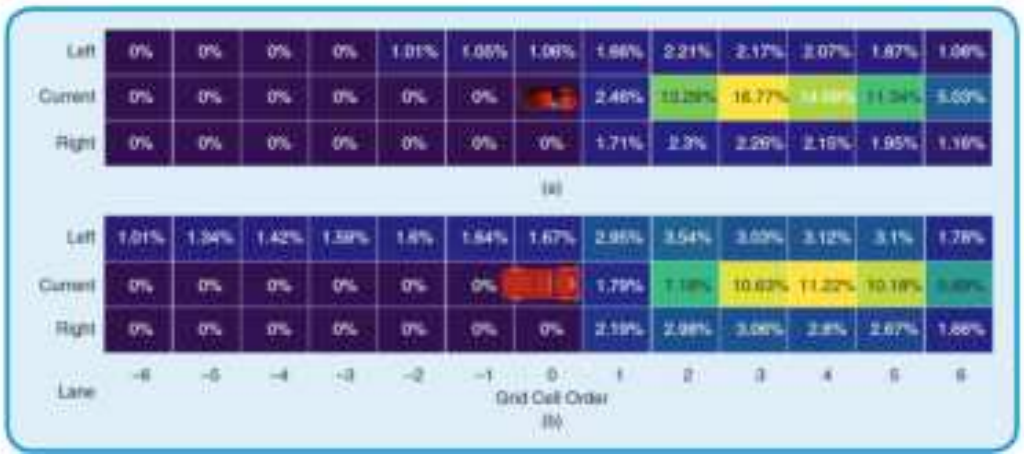


Figura 2.22: *Spatial attention weights per classe di veicolo.*

Dall'analisi dei pesi relativi al meccanismo di attenzione spaziale, per densità di traffico, emerge che nel caso di strade poco congestionate (meno di 7 veicoli vicini) la posizione del veicolo target è caratterizzata dal peso maggiore. Con l'aumentare della congestione, il peso dell'attenzione della cella del veicolo target diminuisce, indicando che i veicoli vicini hanno maggiore influenza sul veicolo in un ambiente congestionato.

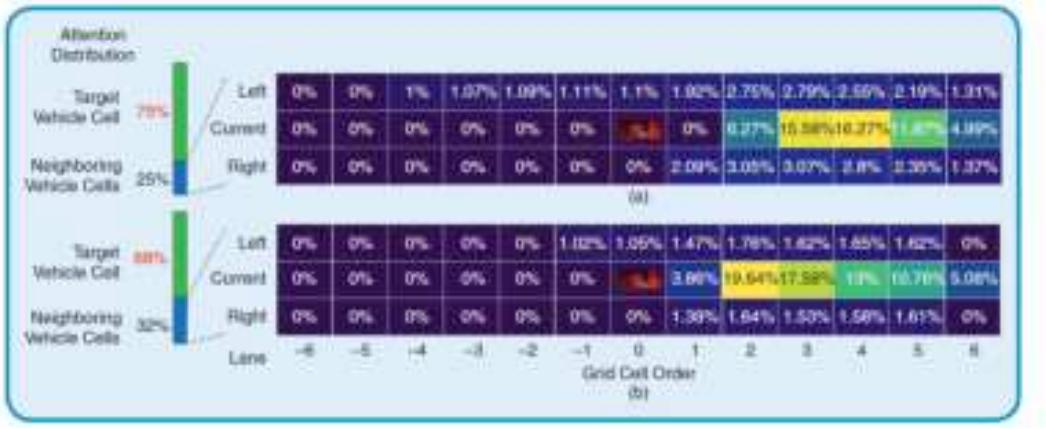


Figura 2.23: *Spatial attention weights per densità di traffico.*

Capitolo 3

Tecnologie utilizzate

In questo capitolo, saranno presentate le tecnologie e i framework utilizzati nella modellazione di reti neurali ricorrenti basate su un'architettura encoder-decoder LSTM per la predizione delle traiettorie dei veicoli. Gli argomenti trattati in questo capitolo includono: l'introduzione al framework **YOLOv8** (You Only Look Once version 8) per il rilevamento di oggetti in tempo reale; una presentazione mirata del linguaggio di programmazione **Python** nell'ambito del deep learning e dell'analisi dei dati, nonché la sua versatilità in riferimento alla prototipazione rapida e lo sviluppo di modelli di machine learning; l'introduzione a **PyTorch**; la presentazione di **Google Colab** come piattaforma di cloud computing per lo sviluppo di modelli di deep learning.

3.1 YOLO

YOLO, acronimo di "You Only Look Once," è una delle tecnologie più rivoluzionarie nell'ambito della visione artificiale e dell'elaborazione delle immagini. Creato da Joseph Redmon e Santosh Divvala nel 2016 [18], YOLO segna un significativo progresso nel campo della rilevazione degli oggetti in tempo reale, grazie a un'approccio innovativo. A differenza dei modelli convenzionali, YOLO tratta la rilevazione degli oggetti come un problema di regressione. L'idea fondamentale è quella di sviluppare una singola rete

neurale in grado di prevedere sia le coordinate dei bounding box relativi agli oggetti presenti nelle immagini che le probabilità associate alle classi di tali oggetti. L'architettura di YOLO si basa su una serie di differenti tecniche e metodologie, ognuna delle quali svolge un ruolo cruciale per il corretto funzionamento del modello. Le componenti chiave di YOLO sono rappresentate da:

- **Convolutional Neural Networks**: YOLO impiega una CNN come backbone, per il processamento delle immagini. I layer convoluzionali sono responsabili dell'estrazione delle feature dalle immagini. L'architettura di YOLO è simile a quella di GoogleNet [19]. Come evidenziato dall'immagine di seguito, l'architettura ha complessivamente 24 layer convoluzionali, quattro layer di max-pooling e due layer completamente connessi. L'architettura funziona come segue:
  - Ridimensiona l'immagine di ingresso in un'immagine 448x448 prima dell'applicazione del layer convoluzionale.
  - Viene prima applicata una convoluzione 1x1 per ridurre il numero di canali, seguita da una convoluzione 3x3 per generare un output cuboidale.
  - La funzione di attivazione è la ReLU, tranne che per lo strato finale, che utilizza una funzione di attivazione lineare.
  - Alcune tecniche aggiuntive, come la batch normalization e il dropout, regolarizzano il modello ed evitano l'overfitting.

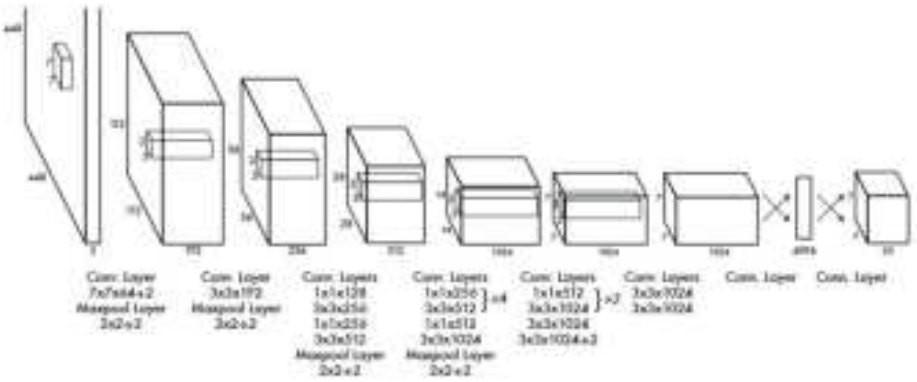


Figura 3.1: *Architettura del modello.*

- **Grid-based detection**: YOLO divide l'immagine di input in una griglia di celle.
- **Anchor Boxes**: YOLO determina i riquadri di delimitazione che corrispondono ai rettangoli che evidenziano tutti gli oggetti dell'immagine. Si possono avere tanti riquadri di delimitazione quanti sono gli oggetti presenti in una data immagine.

- **Objectness Score**: Per ogni anchor box presente in ogni cella della griglia, YOLO predice un punteggio di objectness. Ovvero, la probabilità che ci sia un oggetto all'interno di quella anchor box.
- **Class probabilities**: YOLO predice le probabilità di classe per ognuna delle anchor box delle celle della griglia, specificando la probabilità che l'oggetto appartenga a diverse classi (ad esempio, "auto", "cane", "albero", ecc.).
- **Intersection over Union (IoU)**: Nella maggior parte dei casi, un singolo oggetto in un'immagine può essere predetto da più anchor boxes, sebbene alcune di queste siano irrilevanti. L'obiettivo dell'IOU quello di scartare box con valori inferiori ad una threshold fissata dall'utente.
- **Non Maximum Suppression (NMS)**: L'impostazione di una soglia per l'IOU non è sempre sufficiente, poichè un oggetto potrebbe essere caratterizzato da più caselle con valori di IOU oltre la soglia e il mantenimento di tutte queste caselle potrebbe genere del rumore. In questo caso è possibile utilizzare NMS per mantenere solo le caselle con il punteggio di probabilità di rilevamento più elevato.
- **Loss Function**: YOLO utilizza una funzione di perdita specifica che combina gli errori nella previsione dell'oggetto, nella localizzazione del rettangolo di selezione e nella previsione della classe.

3.2 Python

Python è un linguaggio interpretato, di alto livello e multiparadigma, che ha acquisito un'incredibile popolarità nel corso degli anni, diventando uno strumento fondamentale per professionisti di molteplici settori. Python si distingue per la sua sintassi chiara e leggibile, che favorisce l'analisi del codice e la semplicità concettuale. Questa caratteristica rende Python un linguaggio di programmazione altamente accessibile. Uno dei tratti distintivi di Python è la sua vasta libreria standard, che offre una ricca raccolta di moduli e funzioni pronte all'uso per una vasta gamma di compiti, da operazioni di base come l'interazione con il sistema operativo fino alla gestione avanzata di dati e calcoli scientifici. Questa estensibilità e flessibilità rendono Python un linguaggio adatto per applicazioni in vari campi, tra cui sviluppo web, automazione, elaborazione di testi, intelligenza artificiale, apprendimento automatico, e analisi dei dati.

3.3 Pytorch

PyTorch [20] è un framework open-source per il deep learning, noto per la sua flessibilità e facilità d'uso. Ciò è dovuto in parte alla sua compatibilità con il popolare linguaggio di programmazione di alto livello Python. PyTorch è un framework completo per la costruzione di modelli di deep learning, comunementi usati in applicazioni come il riconoscimento delle immagini e l'elaborazione del linguaggio naturale, inoltre PyTorch si distingue per l'eccellente supporto delle GPU. Il framework combina le efficienti e flessibili librerie di backend, accelerate dalle GPU di Torch, con un frontend Python intuitivo.

3.4 Google Colab

Google Colab, abbreviazione di "Google Colaboratory," è una piattaforma di cloud computing offerta da Google che mette a disposizione degli utenti una potente infrastruttura di calcolo basata su cloud. Questa piattaforma è particolarmente adatta per lo sviluppo, l'esecuzione e la condivisione di progetti di machine learning, analisi dati e programmazione in generale. Google Colab fornisce accesso pay-per-use a GPU (Graphics Processing Units) e TPU (Tensor Processing Units), che consentono di addestrare modelli di machine learning in modo significativamente più veloce rispetto a una CPU tradizionale. Inoltre, Colab è preconfigurato con una vasta gamma di librerie e framework popolari per il machine learning, come TensorFlow, PyTorch, scikit-learn e molti altri.

Capitolo 4

Modelli di predizione della traiettoria dei veicoli

In questo capitolo, verrà fornita un'approfondita analisi dei due modelli di deep learning, basati sull'architettura encoder-decoder LSTM, sviluppati per risolvere il complesso problema della previsione della traiettoria dei veicoli nel contesto urbano. Saranno esaminate le architetture dei modelli, comprensive della descrizione del funzionamento dell'encoder-decoder LSTM, utilizzate per catturare le intricate relazioni spazio-temporali nei dati. Saranno inoltre discussi i dettagli relativi all'addestramento, la scelta delle funzioni di loss e le strategie di ottimizzazione utilizzate per massimizzare la capacità predittiva dei modelli.

4.1 Metodologia

L'implementazione del sistema è stata guidata da una metodologia articolata in diverse fasi. Inizialmente, si è proceduto alla raccolta e preparazione dei dati. Successivamente, è stato creato un dataset da cui sono stati estratti i dati per l'addestramento, la validazione e l'inferenza. Infine, si è proceduto alla progettazione delle reti neurali. Queste reti sono state sviluppate seguendo i principi dell'architettura encoder-decoder, in cui l'encoder ha il compito di acquisire informazioni rilevanti dalle sequenze temporali di input, mentre il decoder genera le previsioni relative alle future coordinate dei veicoli. Durante il processo di addestramento, sono state applicate tecniche di regolarizzazione e ottimizzazione, con un monitoraggio costante delle metriche di valutazione per garantire la capacità di generalizzazione e un apprendimento efficace dei modelli.

Sono stati sviluppati due modelli LSTM distinti, rispettivamente **HIST** e **NBRS**, che differiscono l'un l'altro per l'input utilizzato. Esso è indicato come $X = \{x_{t-5}, x_{t-4}, ..., x_t\}$ con $|X| = 6$.

Il modello **HIST** è caratterizzato da un vettore contenente le precedenti cinque coordinate del veicolo, più le coordinate relative alla posizione corrente. Il modello **NBRS** si basa invece, sulle coordinate dei veicoli classificati come vicini più prossimi del veicolo target, nei precedenti cinque istanti temporali e nell'istante temporale corrente.

Al fine di ottenere un dataset completo e coerente riguardante le condizioni del traffico, è stato utilizzato YOLOv8 per l'analisi di video YouTube relativi alle condizioni del traffico in prossimità di un incrocio stradale. Questo approccio, ha consentito di estrapolare dati dettagliati e affidabili, frame per frame, riguardo la posizione dei veicoli all'interno del contesto di riferimento. In particolare, l'output prodotto dall'inferenza di YOLOv8, su ciascun video relativo al contesto di riferimento, ha prodotto le seguenti informazioni:

- **frame_id**: id del frame.
- **id**: id del veicolo rilevato. Tale id identifica univocamente un veicolo nel corso dell'inferenza sull'intero video. Per cui, qualora lo stesso veicolo dovesse essere rilevato in frame successivi al frame dov'è avvenuta la prima identificazione, esso manterrà il medesimo id.
- **bbox_left**
- **bbox_top**
- **bbox_w**
- **bbox_h**
- **class_id**: tipologia del veicolo.
- **conf**: accuratezza relativa alla predizione effettuata sulla classe del veicolo.

Gli output **bbox_left,bbox_top,bbox_w e bbox_h** rappresentano le coordinate caratterizzanti i bounding box degli oggetti rilevati all'interno di un'immagine, come mostrato dai riquadri colorati nella figura 4.2. Queste coordinate sono cruciali poiché forniscono le informazioni necessarie per localizzare e identificare con precisione gli oggetti. In YOLO, ogni bounding box è rappresentato da quattro valori dove:

- bbox_left e bbox_top indicano rispettivamente la distanza del box dal margine sinistro, e dal margine superiore dell'immagine.
- bbox_w e bbox_h rappresentano invece la larghezza e l'altezza dei bounding box.

Le coordinate sono ottenute rispetto agli assi cartesiani disposti come evidenziato nella figura 4.1. In questa rappresentazione, l'asse delle ordinate (y) si estende verticalmente verso il basso dalla parte superiore dell'immagine, mentre l'asse delle ascisse (x) si estende orizzontalmente dalla sinistra verso destra dell'immagine stessa.

Figura 4.1: *Incrocio stradale di riferimento.*

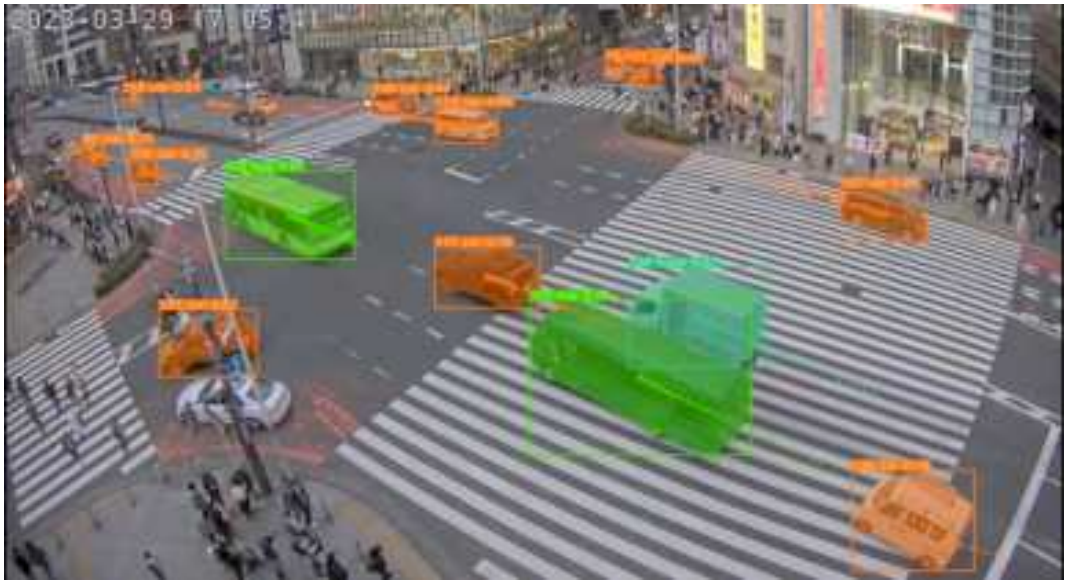


Figura 4.2: *Applicazione di YOLOv8 al video relativo all'incrocio e relativi assi cartesiani.*

Al fine di ridurre i tempi di inferenza di YOLOv8, si è deciso di diminuire il frame rate da 25fps a 10fps. Considerato che il video originale è a 30fps, questa riduzione ha comportato la definizione di un time-step pari a $0.1s$, ciò significa che la distanza temporale tra coppie di rilevamenti adiacenti (frames) è pari a $0.1s$.

Per ottenere un dataset significativo e completo, è stato necessario applicare YOLOv8 a una serie di video relativi al medesimo time-slot. L'obiettivo era quello di catturare il maggior numero possibile di rilevazioni, al fine di creare un dataset consistente e massimizzare la probabilità di identificazione dei veicoli. Per garantire la coerenza dei dati e massimizzare l'efficacia delle rilevazioni, le analisi sono state svolte sull'orario di punta, compreso tra le 12:00 am e le 14:00 pm (ora locale). Questo intervallo temporale è noto per essere caratterizzato da un elevato volume di traffico stradale, il che lo rende particolarmente rilevante per l'obiettivo prefissato. L'approccio di analizzare video relativi allo stesso time-slot ha contribuito a creare un dataset omogeneo e rappresentativo delle dinamiche del traffico durante l'orario di punta, migliorando così la qualità e l'utilità delle informazioni ottenute.

Questo dataset fornisce una base solida per ulteriori analisi, volte a comprendere meglio il flusso del traffico e a sviluppare soluzioni efficaci per la gestione e l'ottimizzazione delle risorse stradali. L'intero lavoro relativo all'applicazione, alla gestione e allo sviluppo delle tecniche legate a YOLOv8 è stato effettuato dal collega Stefano Perna, per il suo progetto di tesi.

Infine, sono state sviluppate le reti neurali. Queste, ricevendo in ingresso l'input X, restituiscono in output un vettore $Y = \{(x_{t+1}, y_{t+1}), (x_{t+2}, y_{t+2}), (x_{t+3}, y_{t+3})\}$ contenente le coordinate del veicolo target nei prossimi tre istanti temporali. Attraverso l'unione dei punti, cioè le coordinate predette, viene generato un segmento rappresentante la predizione della traiettoria del veicolo.

4.2 Preprocessing dei dati e modellazione del dataset

L'utilizzo di YOLO ha generato un insieme di dataset, corrispondente al numero di video su cui è stata condotta l'inferenza, contenenti informazioni altamente rilevanti ma, inizialmente, piuttosto disaggregate. Questo significa che, sebbene fossero disponibili dati relativi ai veicoli, le informazioni inerenti alle coordinate spaziali (x, y) dei target e dei vicini, non erano presenti nei dataset grezzi. L'attività di preprocessing è stata articolata in diverse fasi. Inizialmente sono state ricavate le informazioni relative alle coordinate spaziali. Nello specifico, avendo a disposizione i dati bbox_left e bbox_top, sono state ricavate le coordinate, corrispondenti al centro del veicolo, come evidenziato dal listing 4.1 e dalla figura 4.3.

```
1 df['x'] = df['bbox_left'] + (df['bbox_w'])/2
2 df['y'] = df['bbox_top'] + (df['bbox_h'])/2
```

Listing 4.1: *Calcolo coordinate*

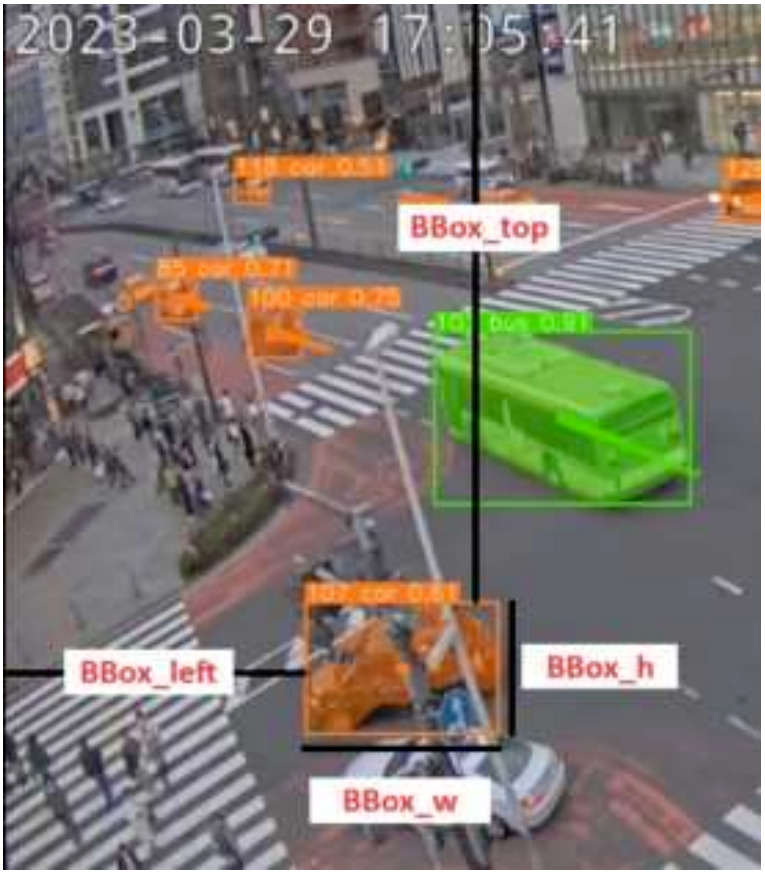


Figura 4.3: *Output YOLOv8*

Successivamente, sono state effettuate operazioni di filtraggio delle voci del dataset. Nello specifico, in riferimento all'attributo **class_id**, ovvero la classe di identificazione del veicolo, sono state mantenute solo le voci corrispondenti alle categorie pertinenti, come auto, moto, bus e autocarri, eliminando quelle non riconducibili a tali categorie. Questo fenomeno è dovuto al fatto che YOLOv8 è in grado di riconoscere un grande quantitativo di elementi non limitandosi, dunque alla sola identificazione dei veicoli.

In un passaggio successivo, è stata condotta un'analisi per identificare gli 8-nearest neighbors (8-nn) di ciascun veicolo rilevato da YOLOv8. È importante notare che non sempre è stato possibile individuare gli 8-nn per tutti i veicoli e in ogni istante temporale. In questi casi, sono state applicate tecniche di padding. Gli 8-nn rappresentano gli otto veicoli più vicini al veicolo target rispetto alla distanza euclidea, come illustrato dal listing 4.2. Questa metrica è stata utilizzata per analizzare la vicinanza spaziale tra i veicoli, consentendo di comprendere meglio le loro interazioni.

```
def find8NN(x_t,y_t,neighbors):
  ret = []
  for element in neighbors:
    v_id = element[0]
    x_v = element[1]
    y_v = element[2]
    d = math.sqrt((x_t - x_v)**2 + (y_t - y_v)**2)
    d = round(d, 2)
    ret.append([d,v_id,x_v,y_v]) #[nn_id ; nn_x ; nn_y; distance]
  ret = (sorted(ret, key = lambda x: x[0]))
  for element in ret:
    element.pop(0)
    element.pop(0)
  return ret
```

Listing 4.2: *8-Nearest neighbors*

A seguito del complesso processo di preprocessing eseguito su ciascun dataset, sono stati ottenuti dati ben strutturati e pronti per le analisi successive. Ogni dataset ha subito le opportune trasformazioni e integrazioni per garantire la coerenza e la qualità dei dati. Una tappa cruciale è stata quella di unire questi dataset per creare un dataset unico, completo e consolidato. Questa operazione di fusione dei dati è stata resa possibile grazie all'attenzione dedicata alla standardizzazione delle variabili e alla coerenza delle informazioni in ogni dataset. L'obiettivo era creare un insieme di dati coeso che riflettesse la varietà delle situazioni e delle fonti originali, senza compromettere la qualità degli stessi. A seguito di tali operazioni, si è ottenuto un dataset composto da un totale di 160457 entry, caratterizzato dai seguenti attributi:

- **vehicle_id**: questo attributo rappresenta un identificativo associato a ciascun veicolo target rilevato da YOLO. L'identificativo consente di tenere traccia di ogni veicolo in modo univoco all'interno del dataset.
- **HIST**: l'attributo, è un vettore che contiene un insieme di informazioni storiche riguardanti il veicolo target. Nello specifico, le coordinate delle cinque posizioni passate del target e quelle attuali. Questo attributo fornisce una visione completa del comportamento passato e corrente del veicolo target.
- **NBRS**: questo attributo è un vettore che raccoglie le coordinate degli 8 vicini più prossimi del veicolo target in ciascun istante temporale all'interno di un intervallo specifico. Questo permette di analizzare le interazioni spaziali e temporali tra il veicolo target e i suoi vicini più prossimi.
- **fut**: l'attributo è un vettore di tre elementi, il quale rappresenta le prossime tre coordinate del veicolo target.

Successivamente, il dataset è stato suddiviso in tre componenti distinte: training set, validation set e test set. La suddivisione è stata effettuata seguendo una proporzione bilanciata. Nello specifico, al training set è stata garantita una size del 70% della size del dataset inziale, mentre al validation set e al test set una size pari al 15% della dimensione del dataset originale (70%, 15%, 15%). Si tratta di un rapporto di divisione comunemente utilizzato, soprattutto quando si dispone di dataset di dimensioni moderate. Tale suddivisione, fornisce un buon equilibrio tra i dati di addestramento per l'apprendimento del modello, i dati di validazione per la regolazione degli iperparametri e i dati relativi al test set per la valutazione finale del modello. Il miglior rapporto di suddivisione di un dataset in training, validation e test set dipende da vari fattori, tra cui le dimensioni del dataset, la complessità del modello di apprendimento automatico e gli obiettivi specifici del progetto.

4.3 Sviluppo dei modelli

4.3.1 Modello HIST

Il modello di machine learning basato su LSTM, noto come **HIST**, è un modello senza vincoli che si focalizza esclusivamente sulle informazioni spaziali dei veicoli al fine di predire la loro traiettoria. Questo modello non prende in considerazione altre dati o informazioni, come la tipologia dei veicoli, la loro velocità, la loro accelerazione, il volume traffico etc. Dunque è possibile classificare HIST come un modello di deep learning unimodale e di tipo interaction related factors. Questo approccio, può essere utile in situazioni in cui, a causa dell'assenza di altre informazioni, si dispone solo di dati spaziali. È cruciale evidenziare che il modello in questione presenta limiti intrinseci a causa della mancata considerazione di altre informazioni come le grandezze cinematiche. Di conseguenza, la sua precisione potrebbe essere compromessa in scenari complessi, dove l'impiego di altre informazioni e dati risulta essere fondamentale per la corretta esecuzione del task di predizione della traiettoria dei veicoli. Tuttavia, è fondamentale sottolineare che lo sviluppo di un modello di questa tipologia è stato dettato dalla mancanza di un dataset che comprendesse informazioni supplementari, oltre a quelle spaziali dei veicoli.

L'obiettivo del modello è quello di predire le coordinate spaziali, relative ai prossimi tre istanti temporali, a partire dalle quali è generato il segmento rappresentante la traiettoria predetta. Ciò avviene, minimizzando una funzione di loss quale l'RMSE. Nello specifico il modello è addestrato e validato con l'obiettivo di minimizzare la seguente funzione:

$$\text{loss} = \frac{\left(\text{loss}t_1 + \text{loss}t_2 + \text{loss}_{t_3}\right)}{3}$$

Dove la generica $loss_t$, ovvero la loss relativa alla predizione della $t - esima$ posizione, è:

$$\sqrt{\frac{\left(\sum_{i=1}^{\text{batch size}}((x_i - \tilde{x_i})^2 + (y_i - \tilde{y_i})^2)\right)}{\text{batch size}}}$$

Poiché PyTorch non fornisce una versione nativa dell'RMSE, è stato definito un criterio basato sull'MSE (Mean Squared Error). Durante la fase di addestramento del modello, l'RMSE viene calcolato applicando la radice quadrata all'output ottenuto tramite l'MSE.

```
# Target lists
train_loss_hist = np.full(trainEpochs, np.nan)
val_loss_hist = np.full(trainEpochs, np.nan)
# Loss
criterion_mse = torch.nn.MSELoss(reduction = 'sum')
```

Listing 4.3: *Definzione funzione di loss e varibili di loss*

```
l1 = torch.sqrt(criterion_mse(t1,t1_p)/128)
l2 = torch.sqrt(criterion_mse(t2,t2_p)/128)
l3 = torch.sqrt(criterion_mse(t3,t3_p)/128)
```

Listing 4.4: *RMSE*

Parametri

Il modello è caratterizzato dai seguenti parametri:

```
trainEpochs = int(sys.argv[2])
batch_size = int(sys.argv[3])

args = {}
args['use_cuda'] = True
args['encoder_size'] = 128 #lstm encoder hidden state size
args['decoder_size'] = 128 #lstm decoder hidden state size
args['in_length'] = 6 #input length
args['out_length'] = 3 #output length
args['batch_size'] = batch_size
```

```
11 args['n_layers'] = 1
12 args['input_embedding_size'] = 64 #input dimension for lstm encoder
```

Listing 4.5: *Parametri modello HIST*

L'**encoder_size** e il **decoder_size** rappresentano le dimensioni dei vettori relativi agli hidden states dell'encoder e del decoder. Essendo che l'hidden state del decoder è inizializzato con l'hidden state dell'encoder, è necessario che i due vettori abbiano la medesima dimensione. Le variabili **in_length** e **out_lenght** rappresentano rispettivamente le dimensioni dei vettori di input e di output. La **batch_size** indica invece, la dimensione dei batch. Infine con **input_embedding_size** è indicata la dimensione dello spazio di embedding per l'input.

Sono stati condotti diversi test per determinare la configurazione ottimale della variabile **trainEpochs**, come verrà illustrato nel prossimo capitolo. Sulla base dei risultati ottenuti si è deciso di fissare il valore a **150**.

L'elaborazione in batch è una tecnica di processamento dei dati in gruppi, piuttosto che singolarmente o in modo continuo. Tale tecnica è spesso utilizzata nell'apprendimento automatico al fine di ottimizzare i parametri durante la fase di addestramento. Uno dei principali vantaggi derivanti dall'utilizzo dei batch è il miglioramento dell'efficienza e della scalabilità dell'elaborazione dei dati. Infatti, elaborando i dati in batch è possibile ridurre il numero di operazioni di input/output (I/O), l'utilizzo della memoria e l'overhead di rete. Inoltre, utilizzando i batch, è possibile introdurre un po' di casualità e di rumore nei dati, limitando il rischio di overfitting, e migliorando la capacità di generalizzazione del modello. Tuttavia, l'elaborazione in batch presenta anche alcuni svantaggi di cui è bene essere consapevoli. Uno degli svantaggi principali è che può aumentare la latenza e la complessità dell'elaborazione dei dati. Infatti, qualora vengano utilizzati i batch, è necessario attendere il termine dell'elaborazione dell'intero batch prima di passare a quello successivo, il che può ritardare il feedback e i risultati. Questo può essere un problema se si necessita di un'elaborazione dei dati in tempo reale o in streaming, come nel caso dell'apprendimento online, del rilevamento delle anomalie o dei sistemi di raccomandazione. Pertanto, la scelta della dimensione dei batch è un compromesso tra diversi fattori, come l'efficienza, l'accuratezza, la latenza e la complessità. Non esiste una soluzione unica, poiché le diverse dimensioni dei batch possono avere effetti diversi su modelli diversi. Generalmente, batch più piccoli possono portare a una convergenza più rapida, a una migliore generalizzazione e a una minore latenza, ma possono anche aumentare la varianza, il rumore e la complessità dell'elaborazione dei dati. Batch più grandi, invece, possono portare a una maggiore efficienza, stabilità e scalabilità, ma possono anche aumentare il bias, l'overfitting e la latenza dell'elaborazione dei dati. Si è deciso di fissare la batch size a 128. Questa scelta è stata effettuata per ottimizzare il processo di addestramento del modello, garantendo un equilibrio tra l'efficienza computazionale e la capacità del modello di apprendere dai dati.

```
trainingSet = CustomDataset('trainingSet.csv')
validationSet = CustomDataset('validationSet.csv')

trDataloader = DataLoader(
  trainingSet,
  batch_size=batch_size,
  shuffle=True,
  num_workers=2,
  collate_fn=trainingSet.collate_fn
  )

valDataloader = DataLoader(
  validationSet,
  batch_size=batch_size,
  shuffle=False,
  num_workers=2,
  collate_fn=validationSet.collate_fn
  )
print("End data loaders")
```

Listing 4.6: *Dataloader*

Come evidenziato dal Listing 4.6, su entrambi i dataset di addestramento e validazione sono applicate tecniche di batching. Nello specifico è stata utilizzata la funzione **Dataloader** messa a disposizione da PyTorch. Tra i vari paramenti della funzione, i più rilevanti sono: **shuffle**, **num_workers** e **collate_fn**, i quali svolgono rispettivamente i compiti di:

- shuffling dei dati;
- stabilire il numero di sottoprocessi per la fase di data loading;
- stabilire con quali criteri sono sviluppati i tensori, ovvero gli array multidimensionali.

Lo **shuffling** dei dati è una pratica comune nell'apprendimento automatico. Essa serve a garantire che il modello non apprenda pattern indesiderati relativi all'ordine di proccessamento dei dati. Ciò è particolarmente importante quando si lavora con delle serie temporali, dove l'ordine dei dati può avere un impatto significativo sulle prestazioni del modello. Si noti che lo shuflfing dei dati è eseguito solo durante la fase di addestramento, mentre durante la fase di validazione non è applicato. Lo shuffling dei dati di validazione introdurrebbe una casualità che non è rappresentativa degli scenari reali in cui i dati non vengono rimescolati.

La **collate function** è una funzione definita dall'utente che viene utilizzata per elaborare e organizzare i singoli campioni di dati (ad esempio, tensori o altre strutture di dati) caricati da un dataset in batch quando si utilizza un DataLoader. Questa funzione è particolarmente utile quando i campioni di dati hanno forme o strutture diverse ed è necessario standardizzarli in batch uniformi per l'elaborazione da parte della rete neurale.

Ottimizzazioni

Il modello è stato sottoposto a un processo di addestramento e validazione completo, finalizzato a garantire la sua abilità di apprendimento e generalizzazione. Questo complesso processo ha coinvolto l'utilizzo di epoche e il trattamento dei dati in batch, entrambi elementi fondamentali per il corretto allenamento di una rete neurale. Un'epoca rappresenta una singola iterazione attraverso l'intero dataset di addestramento. Durante ogni epoca, il modello è stato esposto a tutti i campioni di addestramento, organizzati in batch di dimensione 128. Questo approccio ha permesso al modello di apprendere dai dati in modo efficiente, sfruttando al meglio le capacità di elaborazione parallela delle moderne unità di elaborazione grafica (GPU). Dopo ciascuna epoca di addestramento, il modello è stato attentamente valutato utilizzando un dataset di validazione appositamente sviluppato. Questo dataset contiene campioni su cui il modello non è stato precedentemente addestrato. L'obiettivo principale della fase di validazione è stato quello di misurare le performance del modello su dati sconosciuti, mettendo in risalto la sua capacità di generalizzazione. In particolare, dalla validazione del modello, è stato possibile identificare il numero ottimale di epoche per prevenire l'overfitting, assicurando che il modello mantenga una capacità di generalizzazione adeguata e non si adatti eccessivamente ai dati di addestramento.

Le tecniche di ottimizzazione rivestono un ruolo fondamentale nell'addestramento delle reti neurali profonde. Considerando la complessità di addestrare una rete di grandi dimensioni, una delle chiavi per migliorare l'efficienza e la velocità di questo processo risiede nell'uso di ottimizzatori altamente efficienti. Gli ottimizzatori sono responsabili dell'aggiornamento dei pesi della rete in modo da minimizzare la funzione di loss durante il processo di apprendimento. PyTorch offre differenti algoritmi di ottimizzazione, ognuno dei quali presenta pro e contro. Nel caso specifico è stato utilizzato l'algoritmo Adam [21], che rappresenta un'estensione dell'algoritmo stochastic gradient descent (SGD). L'idea alla base dell'algoritmo di ottimizzazione Adam è quella di regolare il learning rate di ogni parametro in virtù delle informazioni provenienti dai gradienti calcolati in precedenza. Questo aiuta l'optimizer a convergere più velocemente e con maggiore precisione rispetto ai metodi con learning rate fisso come l'SGD. L'algoritmo Adam, combina i vantaggi di altri due algoritmi di ottimizzazione, **Momento** [22] e **RMSProp**.
Consideriamo una situazione in cui una palla da bowling si muove lungo una discesa su una superficie liscia: inizialmente, la palla ha una velocità ridotta, ma con il passare del tempo accumula rapidamente un notevole slancio, fino a raggiungere la sua velocità massima. Questo concetto elementare costituisce il fondamento dell'algoritmo di ottimizzazione del momento. Ad ogni iterazione, l'algoritmo **Momento**, sottrae il gradiente corrente dal vettore del momento m, e aggiorna i pesi rispetto quest'ultimo. Questo sottolinea che il ruolo del gradiente è quello di accelerare il processo, non di determinare direttamente la velocità. Per simulare una sorta di meccanismo di attrito, e prevenire che il momento cresca eccessivamente, l'algoritmo introduce un nuovo iperparametro β, che deve essere impostato tra 0 (alto attrito) e 1 (nessun attrito). Un valore tipico di β è 0.1.

Algorithm 1 Algoritmo di ottimizzazione momento

$m \leftarrow \beta m - \eta \nabla_\theta J(\theta)$ ▷ Update momentum
$\theta \leftarrow \theta + m$ ▷ Update weights using momentum

L'**algoritmo della discesa del gradiente** (GD) seleziona, a ogni iterazione, la direzione di massima riduzione, che potrebbe non essere direttamente orientata verso l'ottimo globale. Sarebbe vantaggioso se l'algoritmo potesse correggere la sua direzione in modo da orientarsi in anticipo, verso l'ottimo globale. L'algoritmo **AdaGrad** applica questa correzione riducendo il vettore del gradiente lungo le direzioni più ripide.

Algorithm 2 Algoritmo di ottimizzazione AdaGrad

$s \leftarrow s + \nabla_\theta J(\theta) \otimes \nabla_\theta J(\theta)$ ▷ Update accumulated squared gradient
$\theta \leftarrow \theta - \eta \nabla_\theta J(\theta) \oslash \sqrt{s + \epsilon}$ ▷ Update model parameters with AdaGrad

AdaGrad tende a diminuire la velocità troppo rapidamente e potrebbe non convergere all'ottimo globale. Per affrontare questa sfida, l'algoritmo RMSProp adotta un diverso approccio. Differentemente da AdaGrad, **RMSProp** accumula solo i gradienti delle iterazioni più recenti, grazie all'introduzione di un apposito iperparametro di attenuazione.

Algorithm 3 Algoritmo di ottimizzazione RMSProp

$s \leftarrow \beta s + (1 - \beta)(\nabla_\theta J(\theta) \otimes \nabla_\theta J(\theta))$ ▷ Update squared gradients
$\theta \leftarrow \theta - \eta \nabla_\theta J(\theta) \oslash \sqrt{s + \epsilon}$ ▷ Update weights

L'algoritmo **Adam**, abbreviativo di "Adaptive Moment Estimation," combina le idee alla base degli algoritmi di ottimizzazione del **Momento** e **RMSProp**.

Di seguito, viene riportato il codice relativo alla definizione dell'ottimizzatore PyTorch basato sull'algoritmo Adam.

```
optimizer = torch.optim.Adam(net_hist.parameters())
optimizer.zero_grad()
loss_hist.backward()
a = torch.nn.utils.clip_grad_norm_(net_hist.parameters(), 10)
optimizer.step()
```

Listing 4.7: *Definizione optimizer*

Come evidenziato, l'ottimizzatore è inizializzato con i parametri della rete, ossia i pesi che devono essere ottimizzati. Successivamente, vengono azzerati i gradienti precedentemente calcolati. Questo passaggio assicura che la determinazione dei nuovi gradienti sia basata esclusivamente sull'attuale batch di dati. Nel passaggio successivo, è invece calcolato il gradiente della funzione di loss mediante il processo di backpropagation. Si

noti che, al fine di gestire eventuali fenomeni di esplosione del gradiente, si è utilizzata la tecnica di clipping. Esistono due differenti tecniche di clipping: **by value** o **by norm**. L'idea alla base del **clipping by value** è molto semplice. Essa si basa sulla definizione di un upper bound, e può essere così riassunta:

$$grad\_value = \begin{cases} upper\_bound, & \text{if } grad\_value > upper\_bound \\ grad\_value, & \text{else} \end{cases}$$

L'idea alla base del **clipping by norm** è simile a quella vista precedentemente. La differenza è che i valori dei gradienti sono modificati moltiplicando il vettore unitario dei gradienti con la soglia, come riportato dallo pseudo codice sottostante:

Algorithm 4 Gradient Clipping pseudo codice

$g \leftarrow \frac{\partial C}{\partial W}$ ▷ Calculate the gradient
if $\|g\| \geq$ threshold **then**
  $g \leftarrow \frac{\text{threshold} \cdot g}{\|g\|}$ ▷ Clip the gradient
end if

Infine, con la funzione **.step()**, sono aggiornati i pesi della rete neurale.

Livelli

Il modello è costituito da quattro distinti strati, ciascuno dei quali svolge un ruolo cruciale nella previsione della traiettoria dei veicoli, come riportato dal codice sottostante.

```
# Input embedding layer
self.ip_emb = torch.nn.Linear(2,self.input_embedding_size)
# Encoder LSTM
self.enc_lstm = torch.nn.LSTM(self.input_embedding_size,self.encoder_size
    ,self.n_layers)
# Decoder LSTM
self.dec_lstm = torch.nn.LSTM(self.input_embedding_size,self.encoder_size
    ,self.n_layers)
# Output layer
self.op = torch.nn.Linear(self.dec_lstm_size,2)
```

Listing 4.8: *Layers*

Input embedding layer Il processo di embedding è una tecnica di rappresentazione di dati che trasforma oggetti, come parole o altri elementi, in vettori di numeri reali in uno spazio multidimensionale. Questo processo è utilizzato per catturare le relazioni semantiche tra gli oggetti e ridurre la dimensionalità dei dati, consentendo ai modelli di machine learning di lavorare in modo più efficiente su tali rappresentazioni. Tuttavia, va notato che le tecniche di embedding non sono limitate alla riduzione della dimensionalità, ma possono anche essere utilizzate per espandere la dimensionalità dei dati quando è

necessario migliorare la capacità di un modello nell'identificare relazioni intricate tra le caratteristiche dei dati. Un esempio di quanto detto è ciò che si verifica quando si lavora con valori numerici reali e si desidera catturare relazioni non lineari tra le variabili. In queste situazioni, le tecniche di embedding possono essere impiegate per trasformare tali variabili da uno spazio tipicamente bidimensionale a uno spazio di dimensione superiore. Questo consente al modello di apprendere relazioni più complesse all'interno dei dati. Un'applicazione concreta di quanto appena discusso è riscontrabile nel modello **HIST** il quale, essendo alimentato da coordinate spaziali e quindi dati rappresentati in R^2, adopera l'embedding con l'obiettivo di proiettare tali dati in uno spazio dimensionale più ampio, ossia R^{64}.

Encoder LSTM Il layer Encoder LSTM, ha come obiettivo la definizione di un vettore latente o context vector, che catturi le informazioni chiave dalla sequenza di input. Il layer è inizializzato con le informazioni relative alla dimensione dell'input e al numero di feature nell'hidden state, rispettivamente: 64 e 128. Come mostrato dal codice sottostante l'encoder è alimentato da dati sui quali sono state applicate le tecniche di embedding secondo quanto detto precedentemente. Inoltre, al fine di prevenire il fenomeno di overfitting, è stata utilizzata la funzione **Dropout** [23] messa a disposizione da PyTorch. Questo algoritmo, durante ogni passo di addestramento, assegna a ciascun neurone (compresi quelli di input, ma esclusi quelli di output), una probabilità p di essere temporaneamente disattivato, cioè di essere completamente ignorato durante la fase di addestramento, sebbene possa tornare a essere attivo nel passo successivo.

```
1 self.dropout = nn.Dropout(0.5)
2 hist_out,(hidden_hist,cell_hist) = self.enc_lstm(self.dropout(self.ip_emb
    (hist)))
```

Listing 4.9: *"Forward function" Encoder LSTM*

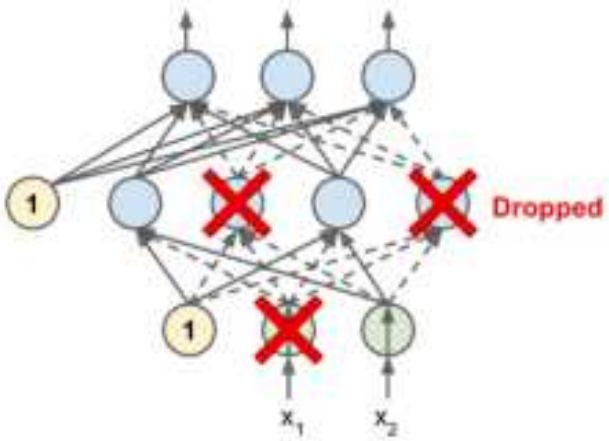


Figura 4.4: *Dropout.*

Decoder LSTM Come evidenziato dal codice sottostante l'hidden e il cell state del decoder sono inizializzati rispettivamente con l'hidden e il cell state dell'encoder. Inoltre, indipendentemente dalla specifica implementazione del decoder, l'input di quest'ultimo è inizializzato con l'input dell'encoder al tempo t, sul quale vengono poi applicate le funzioni di embedding e dropout, come precedentemente osservato.

```
outputs = torch.zeros(self.out_length, fut.shape[1], 2)
decoder_input = self.dropout(self.ip_emb(hist[-1, :, :].unsqueeze(0)))
decoder_hidden = hidden_hist
decoder_cell = cell_hist
```

Listing 4.10: *Inizializzazione Decoder LSTM*

Come detto in precedenza il decoder è stato implementato utilizzando quattro differenti tecniche, quali:

- Recursive;
- Teacher forcing;
- Mixed teacher forcing;
- Dynamic teacher forcing.

Nel **recursive** decoder, l'output generato al passo temporale precedente viene utilizzato come input per generare l'output successivo. Questo approccio è completamente auto-regressivo, poiché ogni passo temporale dipende dal risultato del passo precedente. Può essere efficace per la generazione di sequenze, ma è suscettibile a errori accumulativi e può essere lento a convergere.

```
if type_of_prediction == 'recursive':
    for t in range(self.out_length):
        decoder_output, (decoder_hidden, decoder_cell) = self.dec_lstm(
    decoder_input,(decoder_hidden,decoder_cell))
        fut_pred = self.op(decoder_output)
        outputs[t] = fut_pred
        decoder_input = self.dropout(self.ip_emb(fut_pred))
```

Listing 4.11: *Recursive decoder decoder LSTM*

Nel **teacher forcing**, la ground truth dell'iterazione precedente viene utilizzata come input per l'iterazione successiva, anziché l'output generato come visto nell'approccio ricorsivo. Questa tecnica rende l'addestramento più stabile, poiché l'errore di predizione non si accumula durante la generazione. Tuttavia tale tecnica limita la capacità di generalizzazione del modello causando del potenziale overfitting.

```
elif type_of_prediction == 'teacher_forcing':
    for t in range(self.out_length):
```

```
        decoder_output, (decoder_hidden, decoder_cell) = self.dec_lstm(
    decoder_input,(decoder_hidden,decoder_cell))
        fut_pred = self.op(decoder_output)
        outputs[t] = fut_pred
        decoder_input = self.dropout((self.ip_emb(fut[t, :, :].unsqueeze
    (0))))
```

Listing 4.12: *Teacher forcing Decoder LSTM*

Il **mixed teacher forcing** combina il teacher forcing con i vantaggi della generazione autonoma relativi all'approccio ricorsivo. L'obiettivo di questa tecnica è quello di beneficiare della ground truth, e allo stesso tempo di far sviluppare al modello una propria autonomia nella generazione dell'output. Questo avviene confrontando una soglia predefinita con un valore casuale, come mostrato dal listing 4.13.
Infine, il **dynamic teacher forcing** è una variazione del **mixed teacher forcing** in cui la probabilità di utilizzare la ground truth o l'output generato, come input del decoder è dinamicamente determinata durante l'addestramento.

```
else:
    for t in range(self.out_length):
        decoder_output, (decoder_hidden, decoder_cell) = self.dec_lstm(
    decoder_input,(decoder_hidden,decoder_cell))
        fut_pred = self.op(decoder_output)
        outputs[t] = fut_pred
        if (random.random() < teacher_forcing_ratio):
            decoder_input = self.dropout((self.ip_emb(fut[t, :, :].
    unsqueeze(0))))
        else:
          decoder_input = self.dropout(self.ip_emb(fut_pred))
```

Listing 4.13: *Mixed e Dynamic teacher Decoder LSTM*

```
if(sys.argv[1]=="dynamic_teacher_forcing" and teacher_forcing_ratio > 0):
    teacher_forcing_ratio = teacher_forcing_ratio - 0.02
```

Listing 4.14: *Dynamic teacher Decoder LSTM*

Output layer Considerato che l'output di ciascuna cella LSTM del decoder è un tensore di coordinate in R^{128}, si è resa necessaria l'implementazione di una rete neurale feed-forward in grado di restituire un'output in R^{2}.

```
# Output layer
self.op = torch.nn.Linear(self.dec_lstm_size,2)
```

Listing 4.15: *Feed-forward net*

4.3.2 Modello NBRS

Il modello di machine learning basato su LSTM, noto come **NBRS**, è un modello senza vincoli che si focalizza esclusivamente sulle informazioni spaziali dei veicoli vicini più prossimi al veicolo target. Questo modello, come il suo gemello HIST con il quale condivide la classificazione (modello di deep learning unimodale e di tipo interaction related factors), non prende in considerazione altri dati o fonti di informazioni, come la tipologia dei veicoli, la loro velocità, la loro accelerazione, il volume traffico etc. Dunque, similmente ad HIST, il modello in questione presenta limiti intrinseci, tuttavia si sottolinea nuovamente che lo sviluppo di un modello di questa tipologia, è stato dettato dalla mancanza di un dataset che comprendesse informazioni supplementari, oltre a quelle spaziali dei veicoli.

Parametri

```
trainEpochs = int(sys.argv[2])
batch_size = int(sys.argv[3])

args = {}
args['use_cuda'] = True
args['encoder_size'] = 128
args['decoder_size'] = 128
args['in_length'] = 24
args['out_length'] = 3
args['batch_size'] = batch_size
args['n_layers'] = 1
args['input_embedding_size'] = 64
```

Listing 4.16: *Parametri modello NBRS*

I parametri del modello NBRS sono gli stessi del modello HIST, a meno del parametro **in_length**, in quanto la tipologia degli input dei due modelli è differente. Nel processo di pre-elaborazione dei dati, è stata effettuata una scelta significativa riguardo ai vicini più prossimi del veicolo target. Sebbene inizialmente fossero stati individuati gli otto veicoli più vicini al veicolo target, in ogni istante temporale, nel processo di addestramento sono stati considerati solo i quattro vicini più prossimi. Questa scelta è stata guidata da diverse considerazioni. Innanzitutto, ridurre il numero di vicini riduce la probabilità di utilizzare valori di riempimento (padding) per completare sequenze con un numero di vicini inferiori a quello prestabilito. Questo è vantaggioso in quanto evita che il modello apprenda informazioni poco significative o indesiderate causate da dati di riempimento. Inoltre, concentrarsi sui quattro vicini più prossimi può migliorare l'efficacia del modello nel catturare relazioni significative e influenti nei dati, poiché si presta particolare attenzione agli eventi più rilevanti nell'ambito dell'applicazione specifica.

Sono stati condotti diversi test per determinare la configurazione ottimale della variabile **trainEpochs**, come verrà illustrato nel prossimo capitolo. Sulla base dei risultati ottenuti si è deciso di fissare il valore di **trainEpochs** a **60** nel caso di modelli con

decoder di tipo recursive e dynamic teacher forcing, **30** nel caso di modelli con decoder di tipo mixed teacher forcing e **40** nel caso di modelli con decoder di tipo teacher forcing.

Ottimizzazioni

Il processo di addestramento e validazione del modello NBRS è stato condotto seguendo una metodologia simile a quella utilizzata per il modello HIST. Questo approccio è stato suddiviso in epoche, durante le quali i dati sono stati elaborati in batch di dimensione 128. L'uso di epoche e batch è una pratica essenziale nell'addestramento di reti neurali profonde, in quanto consente di sfruttare al meglio le capacità di elaborazione parallela delle moderne unità di elaborazione grafica (GPU).

Un aspetto fondamentale relativo all'addestramento del modello è stato l'utilizzo di algoritmi di ottimizzazione, i quali rivestono un ruolo cruciale all'interno del panorama delle reti neurali profonde. Nel caso specifico del modello NBRS, è stato utilizzato l'ottimizzatore Adam, che rappresenta un notevole progresso rispetto all'algoritmo stochastic gradient descent (SGD). La sua caratteristica distintiva sta nella capacità di adattare dinamicamente il tasso di apprendimento per ciascun parametro, basandosi sulle informazioni dei gradienti calcolati nelle iterazioni precedenti. Questa approfondita comprensione del comportamento dei gradienti, durante l'addestramento, consente all'ottimizzatore di convergere in modo più rapido rispetto ai metodi che adottano un tasso di apprendimento statico, come l'SGD.

Per affrontare eventuali problemi legati all'esplosione del gradiente durante l'addestramento del modello NBRS, è stata adottata la tecnica di clipping. Questa tecnica consiste nel limitare i gradienti a un valore massimo prefissato. In questo modo, si evita che i gradienti diventino troppo grandi e destabilizzino il processo di ottimizzazione.

L'obiettivo di questo insieme di ottimizzazioni era di garantire che il modello NBRS potesse apprendere in modo efficiente dai dati di addestramento, evitando così potenziali problemi legati all'overfitting e assicurando una maggiore stabilità durante il processo di ottimizzazione.

Livelli

Il modello, similmente ad HIST, si compone di quattro strati distinti, ognuno dei quali riveste un ruolo fondamentale nella predizione della traiettoria dei veicoli, come evidenziato nel codice sottostante.

```
# Define network weights
self.lin = torch.nn.Linear(self.encoder_size, 1)

# Input embedding layer
self.ip_emb = torch.nn.Linear(2,self.input_embedding_size)

```

```
7  # Encoder LSTM
8  self.enc_lstm = torch.nn.LSTM(self.input_embedding_size,self.encoder_size
       ,self.n_layers)
9
10 # Dropout
11 self.dropout = nn.Dropout(0.5)
12
13 # Decoder LSTM
14 self.dec_lstm = torch.nn.LSTM(self.input_embedding_size,self.encoder_size
       ,self.n_layers)
15
16 # Output layer
17 self.op = torch.nn.Linear(self.decoder_size,2)
```

Listing 4.17: *Livelli modello NBRS*

Capitolo 5

Risultati

In questo capitolo, saranno esposti i risultati derivati dagli esperimenti condotti sui modelli HIST e NBRS. I modelli in questione hanno caratteristiche distinte: HIST è basato sulle traiettorie storiche del veicolo target, mentre NBRS si basa sulle posizioni dei veicoli più prossimi al target. Nello specifico, saranno analizzate le learning curves dei modelli, al fine di identificare l'adeguato numero di epoche per l'addestramento, e i risultati relativi alla predizione delle coordinate e delle traiettorie dei veicoli.

5.1 Modello HIST

5.1.1 Epoche

Le **learning curves** (curve di apprendimento) sono uno strumento fondamentale nell'ambito dei modelli di machine learning. Servono a monitorare e valutare le prestazioni di un modello durante il processo di addestramento e validazione. Queste curve forniscono informazioni preziose per comprendere come il modello stia apprendendo dai dati e per identificare eventuali problemi di underfitting o overfitting.
L'**underfitting** si verifica quando un modello non è in grado di apprendere dal dataset di addestramento, e di conseguenza non riesce a ridurre l'errore in modo sufficiente sul training set. L'**overfitting**, d'altra parte, si verifica quando un modello apprende e si adatta troppo bene ai dati di addestramento, compresi i dettagli insignificanti o il rumore statistico presenti nel dataset, e di conseguenza, ha difficoltà a generalizzare correttamente su nuovi dati. Questa mancanza di generalizzazione si traduce in un aumento dell'errore di generalizzazione, che misura quanto bene il modello si comporta su dati non visti durante l'addestramento.

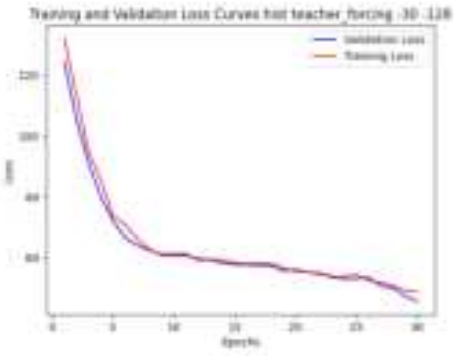
Le **curve di training** rappresentano l'andamento della loss del modello, sui dati di addestramento, al variare del numero di epoche o del tempo. Nello specifico la loss di training misura quanto bene il modello si adatta ai dati di addestramento.
Le **curve di validazione**, al contrario, illustrano la dinamica della loss su dati che sono

completamente estranei al modello. La loss di validazione rappresenta un indicatore fondamentale delle performance del modello in situazioni in cui non ha mai avuto esperienza diretta. La sua utilità principale consiste nell'analizzare la capacità del modello di estendere le sue conoscenze per effettuare previsioni accurate su dati nuovi. La forma e l'andamento delle curve di apprendimento sono utilizzate per valutare il comportamento di un modello di machine learning e per suggerire le possibili modifiche atte a migliorare il processo di apprendimento e le prestazioni complessive.

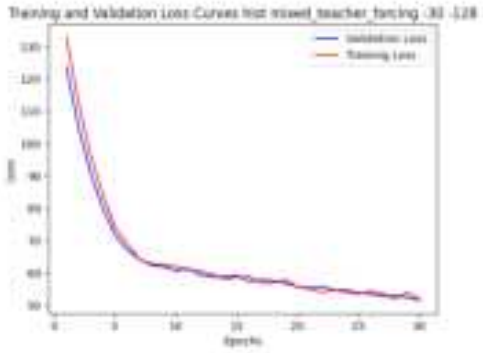
In questa sezione, saranno presentati i risultati ottenuti dall'analisi delle curve di apprendimento. Questa analisi è suddivisa in tre fasi, ognuna delle quali si concentra su un diverso numero di epoche di addestramento, rispettivamente 30, 60 e 150. Ogni fase riporta i risultati relativi a quattro differenti implementazioni del modello HIST. Tali modelli differiscono tra loro per il decoder, ovvero la tecnica impiegata per implementare questa componente della rete neurale ricorrente.

Analisi a 30 epoche Nella fase iniziale di questa analisi, sono prese in esame le curve di perdita relative ai modelli addestrati per sole 30 epoche. Questa fase è fondamentale per ottenere una panoramica dettagliata delle prestazioni iniziali dei modelli e comprendere come essi stiano apprendendo dai dati. Come è possibile osservare dai grafici sottostanti, i modelli presentano un interessante comportamento durante le prime fasi dell'addestramento. Le curve di validazione sono costantemente posizionate leggermente al di sotto delle rispettive curve di training fino alla quinta epoca. Questo pattern suggerisce un addestramento iniziale caratterizzato da una buona capacità di generalizzazione. In altre parole, i modelli stanno apprendendo dai dati di addestramento senza eccessiva specializzazione. Tuttavia, la vera complessità si manifesta successivamente alla quinta epoca, in quanto si assiste a un fenomeno caratterizzato da picchi di salita e di discesa da parte di entrambe le curve. Ciò nonostante, è importante sottolineare che questo fenomeno non si traduce in un allontanamento significativo tra le curve di validazione e quelle di training. Dunque, sebbene i modelli stiano affinando i dettagli nelle epoche successive, non si osservano segni evidenti di overfitting. Questo comportamento può essere interpretato come un segnale di un addestramento robusto, in cui i modelli stanno perfezionando le loro capacità di generalizzazione senza cadere nell'errore di adattarsi eccessivamente ai dati di training. Tuttavia, rimane importante continuare a monitorare attentamente le curve di loss, all'aumentare del numero di epoche con cui il modello è addestrato, per garantire che questo comportamento positivo perduri.

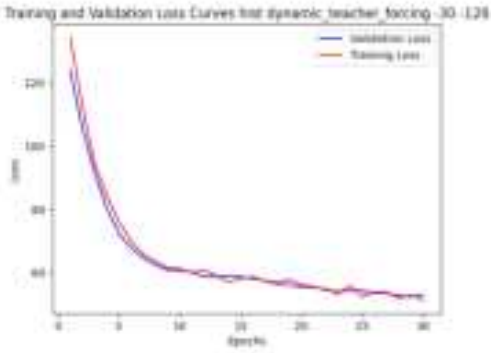
(a) Recursive decoder a 30 epoche



(b) Teacher forcing decoder a 30 epoche



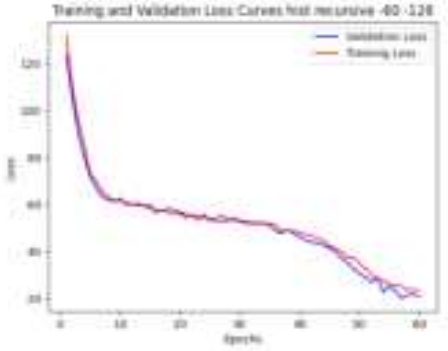
(c) Mixed teacher forcing decoder a 30 epoche



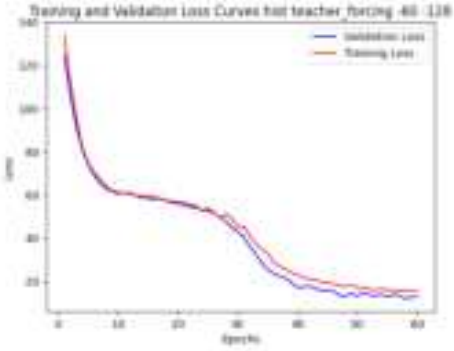
(d) Dynamic teacher forcing decoder a 30 epoche

Analisi a 60 epoche Nella seconda fase, sono approfondite le analisi delle curve di loss relative ai modelli addestrati con 60 epoche. Questa prolungata fase di addestramento ha consentito di analizzare con maggiore attenzione le dinamiche di apprendimento dei modelli e di valutare se vi fossero miglioramenti o peggioramenti rispetto alla fase precedente, durante la quale i modelli erano addestrati con sole 30 epoche. Ciò che emerge dall'analisi delle suddette curve rafforza quanto è stato precedentemente osservato. Sebbene possano verificarsi picchi di salita e di discesa, non emergono chiari segni di un comportamento monotono crescente relativo alle curve di validation. Tale andamento, rappresenterebbe un chiaro indicatore di overfitting. Inoltre, come si può chiaramente osservare dai grafici, dopo una fase iniziale di stabilizzazione in cui la loss RMSE relativa all'addestramento e alla validazione si mantiene costante tra i valori 50 e 60, si verifica una progressiva riduzione di tali valori. Le loss, infatti, convergono a valori vicini a 20. In sintesi, i risultati dell'analisi delle curve di loss nella fase di addestramento a 60 epoche confermano la robustezza del modello rispetto all'overfitting. La stabilità e la vicinanza delle curve di validation rispetto a quelle di training suggeriscono che il modello continua a generalizzare in modo efficace, nonostante il prolungato periodo di addestramento. Inoltre, l'aumento del numero di epoche di addestramento ha consentito

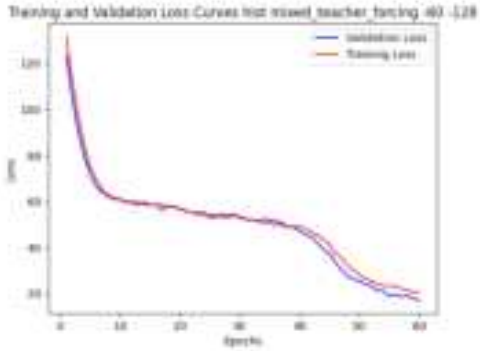
di ridurre significativamente i valori della loss.



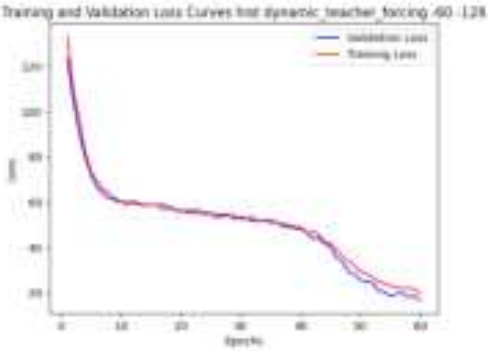
(a) *Recursive decoder a 60 epoche*



(b) *Teacher forcing decoder a 60 epoche*



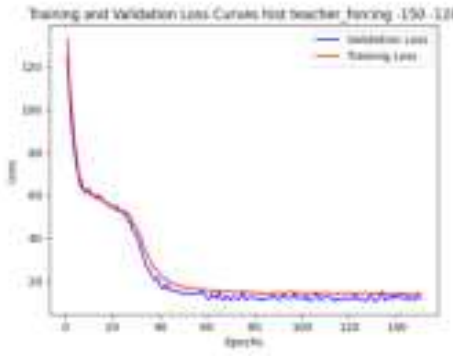
(c) *Mixed teacher forcing decoder a 60 epoche*



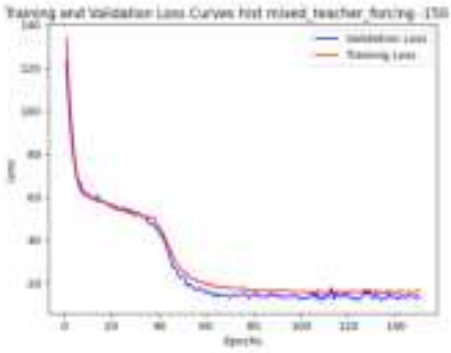
(d) *Dynamic teacher forcing decoder a 60 epoche*

Analisi a 150 epoche Infine, sono riportati le analisi dei modelli allenati con 150 epoche. Questa fase finale restituisce una visione completa delle prestazioni dei modelli dopo un addestramento esteso, consentendo di valutare se l'aumento del numero di epoche per la fase di addestramento e la conseguente validazione, possa portare a un miglioramento significativo delle prestazioni o se i modelli abbiano raggiunto una sorta di plateau. I grafici mostrano che le curve di validazione tendono a stabilizzarsi a livelli inferiori rispetto alle curve di addestramento, sebbene la differenza tra i valori di perdita delle due curve non sia molto significativa. Questo comportamento è riscontrabile in tutti e quattro i modelli considerati, e pertanto non si osservano evidenze di overfitting. In definitiva, si può concludere che il modello ha raggiunto una fase di stabilità o plateau delle prestazioni.

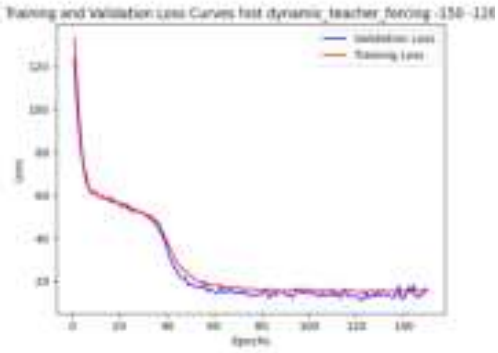
(a) *Recursive decoder a 150 epoche*



(b) *Teacher forcing decoder a 150 epoche*



(c) *Mixed teacher forcing decoder a 150 epoche*



(d) *Dynamic teacher forcing decoder a 150 epoche*

Dunque, si conclude che il numero di epoche adeguato all'addestramento dei modelli è **150**. Questo valore, che mantiene la sua efficacia indipendentemente dall'implementazione specifica del decoder, rappresenta un ottimo equilibrio tra prestazioni ottimali e complessità computazionale. Di seguito è riportata una tabella rappresentante il tempo necessario all'addestramento e validazione dei modelli, al variare delle implementazioni del decoder e del numero di epoche utilizzate. L'addestramento, così come la validazione e l'inferenza è stato condotto mediante l'utilizzo delle risorse HW messe a disposizione da Google Colab. Nello specifico, i dati si riferiscono all'utilizzo della GPU V100 ad elevata RAM.

Epoche	Tempo di addestramento
30	25 minuti
60	50 minuti
150	2 ore

Tabella 5.1: *Tempi di addestramento e validazione*

5.1.2 Predizione delle coordinate dei veicoli

Durante l'addestramento, la validazione, e l'inferenza del modello, è stata utilizzata la metrica RMSE (Root Mean Square Error) per valutare le prestazioni e riconoscere eventuali anomalie. Questa scelta è stata motivata dalla necessità di ottenere una misura sensibile agli errori più significativi nel processo di addestramento, consentendo una migliore identificazione di eventuali deviazioni significative tra le previsioni del modello e i dati reali. L'equazione della loss, nel contesto specifico, può essere interpretata come una misura della distanza euclidea tra due punti in uno spazio bidimensionale. Questa equazione, rappresentata come segue,

$$\sqrt{\frac{\left(\sum_{i=1}^{\text{batch size}}((x_i-\tilde{x_i})^2+(y_i-\tilde{y_i})^2)\right)}{\text{batch size}}}$$

calcola la radice quadrata della media delle distanze euclidee tra i punti di un batch e le rispettive previsioni. Questa operazione è coerente con la definizione della distanza euclidea tra punti nel piano x-y. Quanto più grande è il risultato, tanto maggiore è la discrepanza tra le previsioni e le coordinate reali, indicando una maggiore distanza euclidea tra i punti nel piano x-y. Pertanto, questa equazione è una metrica utile per valutare la precisione di un modello nel prevedere le coordinate di punti in uno spazio bidimensionale.

Di seguito, sono presentate le tabelle contenenti i valori di loss, per ciascuna tipologia di decoder. Queste tabelle forniscono un'analisi dettagliata delle prestazioni dei modelli durante la fase di inferenza. Ogni tabella riporta i valori della loss RMSE associata alle coordinate predette dal modello, al variare del numero di epoche con cui il modello è stato addestrato. Questi valori forniscono un'indicazione chiara delle prestazioni del modello corrispondenti a diverse tipologie di addestramento. Inoltre, ciascuna tabella presenta un valore medio della loss, calcolato come la media dei tre valori associati alle coordinate predette. Questo valore medio rappresenta una misura complessiva delle prestazioni del modello per una specifica epoca di addestramento. Le tabelle non si limitano a fornire soltanto una misura generale delle prestazioni, infatti sono incluse le misure di RMSE calcolate separatamente sulle sole coordinate x e sulle sole coordinate y per ciascun time-stamp. Questa suddivisione mira a fornire una misura dell'errore relativa alla distanza frontale e laterale, consentendoci di valutare in dettaglio la precisione dei modelli.

Loss modello HIST con recursive decoder

Come si può chiaramente notare dai risultati riportati nella tabella 5.1, c'è un evidente miglioramento delle prestazioni del modello quando si aumenta il numero di epoche di addestramento.

Inoltre, si osserva che l'accuratezza delle previsioni del modello decresce all'aumentare del numero di coordinate predette. Relativamente all'analisi della RMSE calcolata sulle componenti x e y al variare del time stamp, si osserva che all'aumentare del numero di epoche il modello è in grado di restituire delle predizioni più realistiche, e che l'errore sulle componenti x è mediamente maggiore del corrispondente calcolato sulle componenti y.

Epoche	(x_{t+1}, y_{t+1})	(x_{t+2}, y_{t+2})	(x_{t+3}, y_{t+3})	Media
30	454.01	431.78	420.07	435.29
60	201.89	184.43	190.08	192.13
150	124.33	131.23	147.49	134.35

Tabella 5.2: *Loss modello HIST con Recursive decoder*

Epoche	RMSE x t1	RMSE y t1	RMSE x t2	RMSE y t2	RMSE x t3	RMSE y t3
30	385.19	240.07	363.54	232.69	352.09	228.82
60	153.48	130.72	144.20	114.41	152.98	112.11
150	107.04	62.86	115.50	61.94	131.55	66.28

Tabella 5.3: *Loss su x e y HIST con Recursive decoder*

Loss modello HIST con teacher forcing decoder

Come chiaramente evidenziato dai risultati riportati nella tabella 5.3, è possibile notare un significativo miglioramento delle prestazioni del modello all'aumentare del numero di epoche utilizzate per la fase di addestramento. Inoltre, è interessante notare che il modello con decoder di tipo teacher forcing presenta valori di loss costantemente più bassi rispetto alle altre implementazioni. Questo suggerisce che l'approccio teacher forcing guida il modello a generare previsioni più accurate, contribuendo così a ridurre la sua incertezza e migliorare le prestazioni complessive. Dunque il costante utilizzo della ground truth, durante l'addestramento, favorisce la produzione di previsioni più accurate da parte del modello, contribuendo significativamente alla riduzione dell'errore di previsione. Questi risultati sono estremamente promettenti e indicano che l'ottimizzazione delle epoche di addestramento e l'implementazione del decoder mediante teacher forcing, sono fattori chiave per ottenere le migliori prestazioni dal modello.
Tale comportamento si riflette anche sull'applicazione della RMSE sulle componenti x e y. Anche in tal caso si osserva che all'aumentare del numero di epoche il modello è in grado di restituire delle predizioni più realistiche, e che l'errore sulle componenti x è mediamente maggiore del corrispondente calcolato sulle componenti y.

Epoche	(x_{t+1}, y_{t+1})	(x_{t+2}, y_{t+2})	(x_{t+3}, y_{t+3})	Media
30	422.67	435.51	427.41	428.53
60	127.69	147.33	160.47	145.16
150	108.89	130.39	147.04	128.77

Tabella 5.4: *Loss modello HIST con Teacher forcing decoder*

Epoche	RMSE x t1	RMSE y t2	RMSE x t2	RMSE y t3	RMSE x t3	RMSE y t3
30	351.49	234.48	366.11	235.61	359.79	230.43
60	108.40	66.99	127.92	72.67	141.15	75.80
150	94.97	52.79	115.95	59.21	131.78	64.72

Tabella 5.5: *Loss su x e y HIST con Teacher forcing decoder*

Loss modello HIST con mixed e dynamic teacher forcing decoder

Sulla base dei risultati evidenziati dalle tabelle sottostanti, emerge chiaramente che l'aumento del numero di epoche di addestramento ha un impatto positivo anche sulla riduzione dei valori di loss del modello mixed e del modello dynamic teacher forcing. Tuttavia, è importante notare che, nonostante questa progressiva diminuzione, i valori di loss, ottenuti da entrambe le implementazioni, rimangono inferiori rispetto a quelli dell'approccio teacher forcing. Inoltre, rapportando tali valori a quelli ottenuti dal modello con decoder ricorsivo, si nota che i modelli mixed e dynamic teacher forcing sono caratterizzati da una riduzione dei valori di loss (come si evince dalle medie), in particolare quando si passa da 30 a 60 epoche, maggiore della corrispondente riduzione ottenuta dal modello con decoder ricorsivo. Tale risultato può essere spiegato dal fatto che i modelli mixed e dynamic, favoriscono l'applicazione della tecnica teacher forcing nelle prime iterazioni rispetto all'approccio ricorsivo. Ciò è causa di una più rapida convergenza verso risultati ottimali.

Epoche	(x_{t+1}, y_{t+1})	(x_{t+2}, y_{t+2})	(x_{t+3}, y_{t+3})	Media
30	459.42	445.64	433.11	446.06
60	179.75	172.17	179.03	176.98
150	123.35	136.32	149.08	136.25

Tabella 5.6: *Loss modello HIST con Mixed teacher forcing decoder*

Epoche	RMSE x t1	RMSE y t2	RMSE x t2	RMSE y t3	RMSE x t3	RMSE y t3
30	386.96	247.38	375.11	240.33	364.00	234.41
60	142.54	109.00	138.67	101.46	147.97	100.04
150	107.85	59.48	120.15	64.00	133.14	66.65

Tabella 5.7: *Loss su x e y HIST con Mixed teacher forcing decoder*

Epoche	(x_{t+1}, y_{t+1})	(x_{t+2}, y_{t+2})	(x_{t+3}, y_{t+3})	Media
30	465.14	444.79	433.09	447.67
60	173.68	174.69	183.01	177.13
150	126.82	140.49	155.02	140.78

Tabella 5.8: *Loss modello HIST con Dynamic teacher forcing decoder*

Epoche	RMSE x t1	RMSE y t2	RMSE x t2	RMSE y t3	RMSE x t3	RMSE y t3
30	393.69	247.46	375.70	237.82	366.17	230.94
60	134.21	109.72	140.68	103.05	151.04	102.67
150	110.08	62.57	123.51	66.56	137.75	70.62

Tabella 5.9: *Loss su x e y HIST con Dynamic teacher forcing decoder*

In definitiva, questi risultati indicano chiaramente che l'ottimizzazione del numero di epoche di addestramento influisce positivamente sulla riduzione dei valori di loss, l'accuratezza nella previsione delle coordinate dei veicoli decresce all'aumentare del time-stamp e che l'errore sulla componente x è maggiore dell'errore sulla componente y. Inoltre, si ha il decoder di tipologia teacher forcing assicura un miglioramento generale delle prestazioni.

5.1.3 Predizione della traiettoria dei veicoli

Per valutare la precisione delle previsioni relative alla traiettoria di un veicolo, è stato implementato un meccanismo che confronta l'andamento delle curve ottenute unendo i punti nel piano corrispondenti alle coordinate reali, con l'andamento delle curve ottenute unendo i punti nel piano corrispondenti alle coordinate predette dal modello. Questo meccanismo aiuta a determinare quanto le previsioni del modello siano coerenti con i dati reali e se riescano a seguire in modo adeguato l'andamento della traiettoria.

Di seguito è riportato il codice Python che esegue questa verifica, incrementando un contatore ogni volta che viene rilevato un caso in cui l'andamento della curva prevista corrisponde a quello della curva reale:

```
def check (x,y,x_real,y_real):
  not_respected = 0
  for i in range(0,len(x)-1):
    if((x[i] < x[i+1]) and (x_real[i] > x_real[i+1])):
      not_respected += 1
    if((x[i] > x[i+1]) and (x_real[i] < x_real[i+1])):
      not_respected += 1
    if((y[i] < y[i+1]) and (y_real[i] > y_real[i+1])):
      not_respected += 1
    if((y[i] > y[i+1]) and (y_real[i] < y_real[i+1])):
      not_respected += 1
  if(not_respected > 0):
    return False
  return True

...

if dummy_check(x_pred,y_pred,x_real,y_real):
    respected += 1
    x_r.append(x_pred)
    y_r.append(y_pred)
    x_r_r.append(x_real)
    y_r_r.append(y_real)
```

Listing 5.1: *Livelli modello NBRS*

In questo codice vengono confrontate le coordinate predette (x, y), con le coordinate reali (x_{real}, y_{real}), punto per punto. Se in uno qualsiasi dei casi l'andamento previsto non fosse coerente a quello reale, il contatore not_respected verrebbe incrementato e si restituirebbe false, altrimenti true.

Questo metodo fornisce una misura di quanto il modello sia in grado di predire la traiettoria del veicolo in base alle coordinate fornite e può essere utilizzato come misura di accuratezza relativa alle previsioni. Di seguito sono riportati i risultati dei test svolti. Questi, rappresentano, al variare del numero di epoche d'addestramento e per ognuno dei modelli sviluppati, il numero di traiettorie correttamente predette.

La figura 5.4 è un esempio rappresentante una traiettoria predetta il cui andamento è conforme a quello della traiettoria reale.

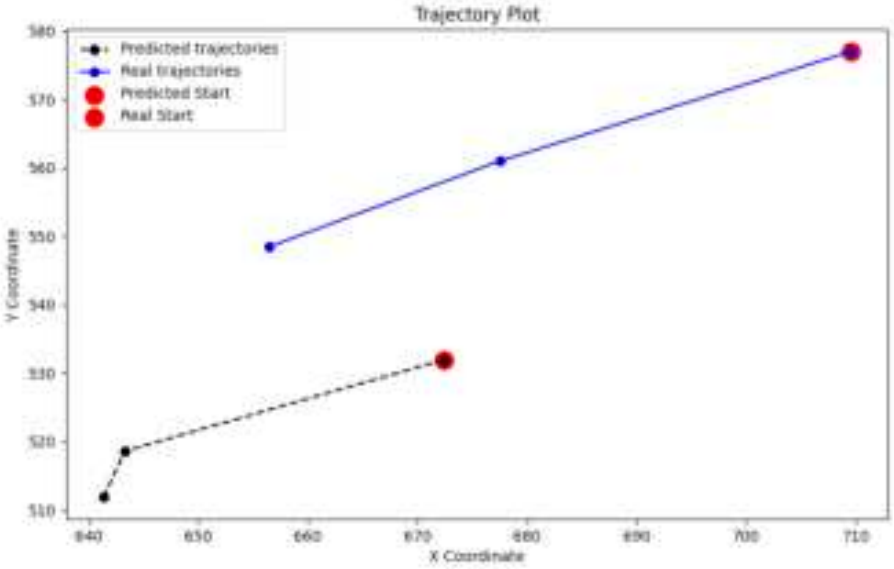


Figura 5.4: *Esempio di predizione della traiettoria*

Epoche	Predizioni corrette	Predizioni errate
30	1953	22116
60	2426	21643
150	2470	21599

Tabella 5.10: *Loss modello HIST con recursive decoder*

Epoche	Predizioni corrette	Predizioni errate
30	3229	20840
60	2727	21342
150	2915	21154

Tabella 5.11: *Loss modello HIST con teacher forcing decoder*

Epoche	Predizioni corrette	Predizioni errate
30	2275	21794
60	2285	21784
150	2899	21170

Tabella 5.12: *Loss modello HIST con mixed teacher forcing decoder*

Epoche	Predizioni corrette	Predizioni errate
30	2220	21849
60	2476	21593
150	3093	20976

Tabella 5.13: *Loss modello HIST con dynamic teacher forcing decoder*

Dai test condotti, emerge chiaramente che l'aumento del numero di epoche ha un impatto sulle prestazioni del modello nella predizione delle traiettorie. Ciò dimostra la rilevanza critica dell'addestramento a lungo termine per ottenere previsioni più accurate. Inoltre, si osserva che il modello con decoder di tipo teacher forcing, addestrato con il minor numero di epoche (30), risulta essere capace di predire con successo il maggior numero di traiettorie. In generale i risultati mostrano un tasso di accuracy relativamente basso, sebbene i risultati relativi alla predizione delle coordinate dei veicoli siano promettenti. La causa di tali risultati, potrebbe risiedere in un meccanismo di verifica troppo stringete. A tal punto si è deciso di ripetere i test, considerando come andamenti coerenti, quelli caratterizzati da al più una dissimilarità. Di seguito sono riportati i risultati:

Epoche	Predizioni corrette	Predizioni errate
30	7302	16767
60	6740	17329
150	8439	15630

Tabella 5.14: *Loss modello HIST con recursive decoder*

Epoche	Predizioni corrette	Predizioni errate
30	9127	14932
60	8491	15578
150	9481	14588

Tabella 5.15: *Loss modello HIST con teacher forcing decoder*

Epoche	Predizioni corrette	Predizioni errate
30	7739	16330
60	7057	17012
150	9079	14990

Tabella 5.16: *Loss modello HIST con mixed teacher forcing decoder*

Epoche	Predizioni corrette	Predizioni errate
30	7282	16787
60	7492	16577
150	9973	14096

Tabella 5.17: *Loss modello HIST con dynamic teacher forcing decoder*

Come dimostrato dai risultati ottenuti, si osserva un notevole miglioramento nelle previsioni. In termini percentuali, l'accuratezza aumenta significativamente, passando dal 13% al 41%. Tali valori sono ottenuti prendendo come modello di riferimento quello caratterizzato dal maggior numero di traiettorie correttamente predette. Nello specifico, il modello con decoder di tipologia teacher forcing (13% - 3229 traiettorie correttamente predette a 30 epoche) e il modello con decoder di tipo dynamic teacher forcing (41% - 9973 traiettorie correttamente predette a 150 epoche).

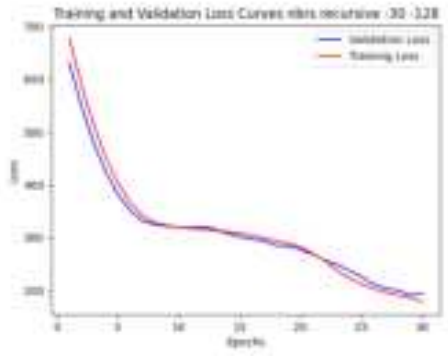
5.2 Modello NBRS

5.2.1 Epoche

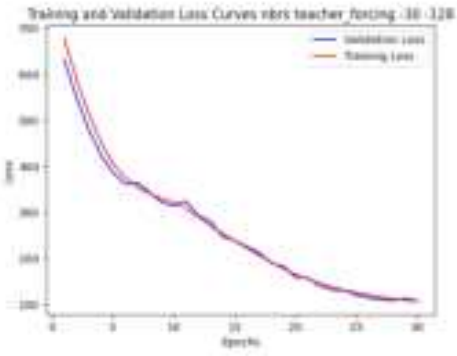
L'approccio utilizzato per l'analisi delle learning curves del modello NBRS è il medesimo utilizzato per il modello HIST. Pertanto l'analisi è suddivisa in tre fasi, ognuna delle quali presenta i risultati relativi all'addestramento dei modelli su un differente numero di epoche. In ogni fase sono analizzate le learning curves relative alle quattro tipologie di decoder.

Analisi a 30 epoche

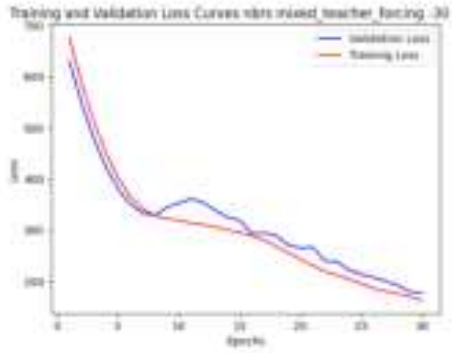
Come si osserva dai grafici sottostanti, i modelli presentano un interessante comportamento durante le prime fasi dell'addestramento. Le curve di validazione relative ai modelli con decoder ricorsivo e di tipo mixed teacher forcing, sono costantemente posizionate leggermente al di sotto delle rispettive curve di training fino alla settima epoca. Questo pattern suggerisce un addestramento iniziale caratterizzato da una buona capacità di generalizzazione. In altre parole, i modelli stanno apprendendo dai dati di addestramento senza eccessiva specializzazione o overfitting. Tuttavia, i modelli con decoder di tipo teacher forcing e dynamic teacher forcing, sono caratterizzati da un innalzamento delle curve di validation a partire dalla quinta epoca, anticipando il comportamento degli altri modelli. Tale tendenza, non rappresenta un chiaro segno di overfitting. Infatti la divergenza tra le curve di validazione e quelle di training è trascurabile. Le curve mostrano una forma simile, contraddistinta da una natura non uniforme, che si manifesta con una serie di picchi, per poi stabilizzarsi. Questo andamento è particolarmente evidente nelle implementazioni con teacher forcing e le relative varianti come mixed e dynamic teacher forcing. Tuttavia, differentemente dai modelli HIST allenati sul medesimo numero di epoche, si notano valori di loss RMSE decisamente superiori.



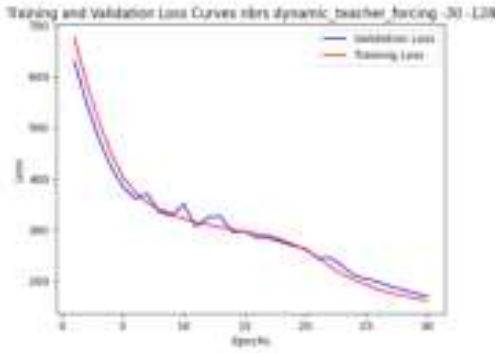
(a) Recursive decoder a 30 epoche



(b) Teacher forcing decoder a 30 epoche



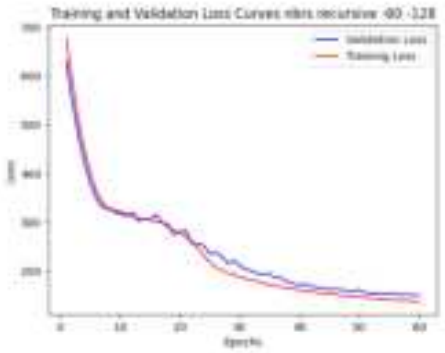
(c) Mixed teacher forcing decoder a 30 epoche



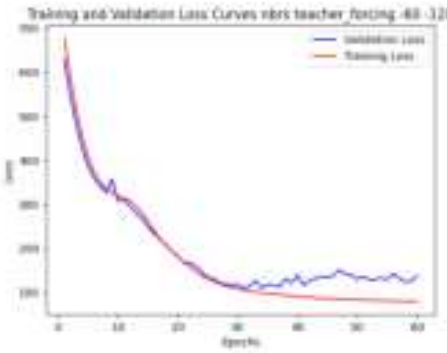
(d) Dynamic teacher forcing decoder a 30 epoche

Analisi a 60 epoche

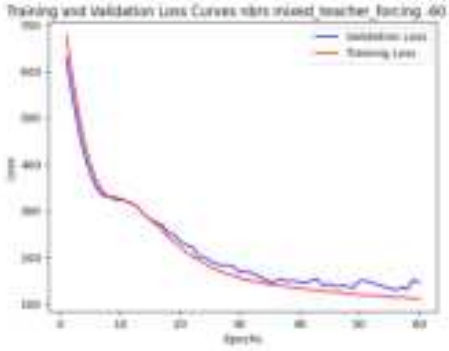
L'analisi delle curve, relative ai modelli NBRS allenati con 60 epoche, rivela degli interessanti risultati. In primo luogo, è cruciale evidenziare una differenza significativa rispetto ai risultati delle analisi dei modelli HIST addestrati con 60 epoche. Nei modelli NBRS, il fenomeno della risalita delle curve di validation è un segnale di overfitting. In particolare, tale fenomeno è più pronunciato nel modello con decoder di tipo teacher forcing. Come si può notare dal grafico B, a partire dalla trentesima epoca, la curva di validazione assume un andamento crescente, confermando la presenza di overfitting. Nei restanti modelli, pur osservando una tendenza di risalita delle curve di validazione, non si osserva una distanza significativa tra queste curve e quelle di training. Nel caso del modello con decoder di tipo dynamic teacher forcing, la curva di validazione mostra una lieve tendenza monotona crescente, ma è necessario aumentare il numero di epoche per trarre delle conclusioni. Questi risultati sottolineano l'importanza di monitorare attentamente le curve di validazione durante l'addestramento del modello e di adottare misure appropriate per mitigare l'overfitting, specialmente nelle implementazioni che mostrano una risalita più marcata delle curve di validazione.



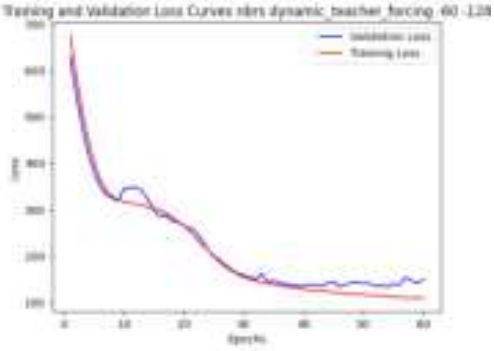
(a) Recursive decoder a 30 epoche



(b) Teacher forcing decoder a 30 epoche



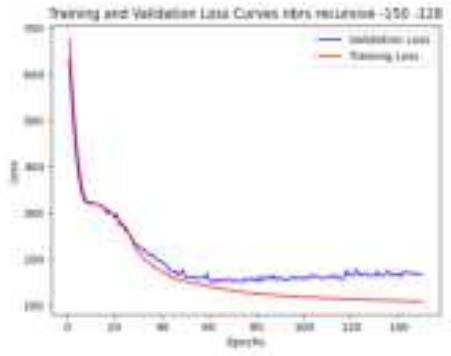
(c) Mixed teacher forcing decoder a 30 epoche



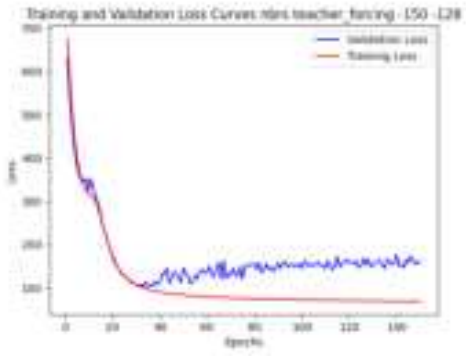
(d) Dynamic teacher forcing decoder a 30 epoche

Analisi a 150 epoche

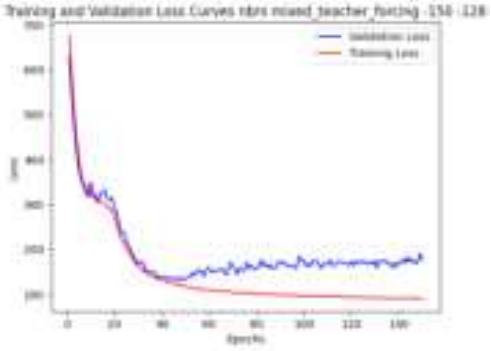
Questa fase conclusiva fornisce una panoramica dettagliata delle performance dei modelli dopo un prolungato periodo di addestramento. Le analisi rivelano che le curve di validazione, a prescindere dall'implementazione specifica del decoder, mostrano un andamento monotono crescente indicando chiaramente la presenza di overfitting. In altre parole, i risultati individuano il numero massimo di epoche con cui i modelli possono essere allenati senza incorrere in overfitting.



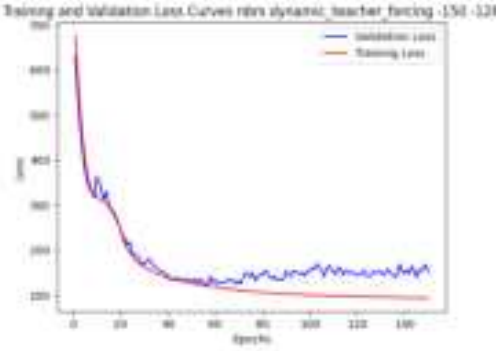
(a) Recursive decoder a 30 epoche



(b) Teacher forcing decoder a 30 epoche



(c) Mixed teacher forcing decoder a 30 epoche



(d) Dynamic teacher forcing decoder a 30 epoche

Pertanto, come emerge dall'analisi dei grafici, il numero ideale di epoche per l'addestramento dei modelli con decoder di tipologia ricorsiva e dynamic teacher forcing è **60**. Tuttavia, quando si lavora con modelli caratterizzati da un decoder di tipo teacher forcing o mixed teacher forcing, è consigliabile ridurre il numero di epoche rispettivamente a **30** e **40**. Tuttavia, è importante notare che i modelli NRBS presentano dei valori di loss RMSE più alti dei corrispondenti modelli HIST. Dunque è prevedibile che i risultati relativi alla predizione delle coordinate e alla previsione della traiettoria dei veicoli, ottenuti dall'inferenza dei modelli NBRS, siano peggiori di quelli ottenuti dall'inferenza dei modelli HIST.

5.2.2 Predizione delle coordinate dei veicoli

Differentemente dalle analisi relative al modello HIST, in cui si assistito a un graduale miglioramento delle prestazioni dei modelli, all'aumentare del numero di epoche, nel modello NBRS si assiste invece, a un netto peggioramento. Questo fenomeno è dovuto principalmente all'overfitting che affligge i modelli già a partire da un numero di epoche di addestramento relativamente basso, ovvero 60. Inoltre, emerge chiaramente la limitata capacità del modello NBRS nel generare coordinate che si avvicinino in modo significativo alle coordinate reali. Questa inefficienza nell'approssimazione, può essere dovuta a vari fattori tra cui la complessità del modello e la mancanza di dati di addestramento rappresentativi per affinare la sua capacità predittiva. Considerati gli alti valori di loss RMSE, si è ritenuto poco significativa l'analisi delle loss sulle componenti x e y.

Loss modello NBRS con recursive decoder

Epoche	(x_{t+1}, y_{t+1})	(x_{t+2}, y_{t+2})	(x_{t+3}, y_{t+3})	Media
30	556.70	545.34	537.91	546.65
60	619.63	605.14	590.14	604.97
150	637.67	628.87	617.10	627.88

Tabella 5.18: *Loss modello NBRS con Recursive decoder*

Loss modello NBRS con teacher forcing decoder

Epoche	(x_{t+1}, y_{t+1})	(x_{t+2}, y_{t+2})	(x_{t+3}, y_{t+3})	Media
30	602.32	606.21	598.31	602.28
60	636.88	637.50	631.39	635.26
150	630.89	622.13	612.76	621.92

Tabella 5.19: *Loss modello NBRS con Teacher forcing decoder*

Loss modello NBRS con Mixed teacher forcing decoder

Epoche	(x_{t+1}, y_{t+1})	(x_{t+2}, y_{t+2})	(x_{t+3}, y_{t+3})	Media
30	562.20	576.45	578.60	572.42
60	592.77	585.88	577.47	585.37
150	626.38	620.34	616.88	621.20

Tabella 5.20: *Loss modello NBRS con Mixed teacher forcing decoder*

Loss modello NBRS con Dynamic teacher forcing decoder

Epoche	(x_{t+1}, y_{t+1})	(x_{t+2}, y_{t+2})	(x_{t+3}, y_{t+3})	Media
30	574.42	579.53	576.33	576.76
60	621.96	624.77	618.60	621.78
150	623.37	614.79	605.80	614.65

Tabella 5.21: *Loss modello NBRS con Dynamic teacher forcing decoder*

5.2.3 Predizione della traiettoria dei veicoli

Epoche	Predizioni corrette	Predizioni errate
30	3424	20645
60	3278	20791
150	3593	20476

Tabella 5.22: *Loss modello NBRS con recursive decoder*

Epoche	Predizioni corrette	Predizioni errate
30	2396	21673
60	2618	21451
150	2610	21459

Tabella 5.23: *Loss modello NBRS con teacher forcing decoder*

Epoche	Predizioni corrette	Predizioni errate
30	2461	21608
60	2554	21515
150	2611	21458

Tabella 5.24: *Loss modello NBRS con mixed teacher forcing decoder*

Epoche	Predizioni corrette	Predizioni errate
30	3318	20751
60	2085	21984
150	2065	22004

Tabella 5.25: *Loss modello NBRS con dynamic teacher forcing decoder*

È evidente che il modello NBRS ha mostrato una notevole discrepanza nelle prestazioni rispetto al modello HIST. A causa di fenomeni di overfitting e della sua limitata capacità di acquisire informazioni rilevanti, il modello NBRS ha ottenuto un tasso di accuratezza estremamente modesto, raggiungendo solamente il 17%. Questo risultato mette in luce la necessità di una valutazione critica e di ulteriori ricerche volte a migliorare le performance di questo specifico modello, oltre ad affrontare le problematiche di overfitting e di scarsa generalizzazione che al momento ne limitano le prestazioni. Si noti che a causa degli alti valori di loss RMSE, ottenuti nella fase di predizione delle coordinate dei veicoli, si è deciso di utilizzare la prima metodologia per l'identificazione delle traiettorie correttamente predette. L'utilizzo della seconda tecnica, che rilassa i vincoli del problema, risulterebbe essere poco significativa considerati gli alti valori di loss RMSE.

Capitolo 6

Conclusioni e possibili sviluppi

In questo lavoro di Tesi è stato definito un approccio innovativo per la previsione della traiettoria dei veicoli nel contesto delle Smart City mediante modelli di machine learning basati su LSTM. Tali modelli, alimentati con dati ottenuti dall'applicazione di YOLOv8 su video di monitoraggio del traffico stradale, predicono le coordinante dei veicoli attraverso le quali è possibile definire dei segmenti rappresentativi delle traiettorie predette. Nonostante le limitate informazioni disponibili e le necessarie semplificazioni effettuate, i risultati ottenuti risultano significativi e lasciano diversi spunti per futuri studi in questo campo.

Il modello HIST, basato sulle informazioni storiche del veicolo sotto esame, effettua un'ottima predizione delle coordinate mentre la predizione della traiettoria presenta ancora margini di miglioramento. Attualmente, infatti, l'accuratezza è calcolata mediante una metrica basata sul numero di traiettorie definite *respected*, ovvero il numero di segmenti, ottenuti dall'unione dei punti corrispondenti alle coordinate predette, che hanno un andamento coerente a quello dei segmenti reali, ottenuti mediante l'unione dei punti corrispondenti alle coordinate reali dei veicoli. Tale meccanismo, non tiene in considerazione il tasso d'errore, ovvero la distanza tra la traiettoria predetta e quella reale. In futuro, si potrebbe definire una versione alternativa, meno stringente, che basandosi sulla differenza tra la coordinata precedente e successiva di un veicolo, sia predette che reali, e su un valore di soglia valuti la bontà della traiettoria predetta. Inoltre, i modelli implementati si basano esclusivamente sui dati relativi alle posizioni dei veicoli e dunque non tengono minimamente in considerazione altre informazioni, come le grandezze cinematiche o le informazioni inerenti al contesto specifico (impianti semaforici, intensità del traffico, condizioni atmosferiche, ecc.), che potrebbero significativamente influenzare la predizione delle coordinate e della traiettoria di ciascun veicolo. Tuttavia, l'implementazione dei suddetti modelli è stata dettata dall'assenza di un dataset conforme al problema di riferimento, e l'applicazione dell'algoritmo di rilevazione di oggetti in tempo reale ha consentito di determinare esclusivamente le informazioni relative alla posizione dei veicoli. Inoltre, a differenza della maggior parte degli studi che si concentrano sul-

l'applicazione delle RNN, e in particolare delle LSTM, per la previsione delle traiettorie dei veicoli in un contesto autostradale o simile, l'approccio presentato ha l'obiettivo di determinare le traiettorie dei veicoli in un contesto urbano. Tale contesto presenta notevoli differenze rispetto a quello comunemente affrontato negli studi, infatti l'ambiente urbano si distingue in modo significativo dal contesto autostradale. A causa dell'elevata variabilità delle condizioni, delle interazioni e degli ostacoli presenti in un ambiente urbano, questo scenario risulta essere molto più complesso da affrontare. Mentre in un contesto autostradale un modello deve apprendere un numero limitato di possibili traiettorie, nel contesto urbano i modelli devono affrontare una gamma molto più ampia di possibilità e adattarsi a situazioni molto più variabili. Nello specifico, prendendo come riferimento il caso di studio analizzato in questo lavoro di Tesi, ovvero l'incrocio stradale dal quale sono stati estratti i dati del traffico, ogni veicolo ha la possibilità di eseguire almeno tre differenti manovre corrispondenti a tre differenti traiettorie possibili. Inoltre, a differenza della maggior parte degli studi, il dataset comprende informazioni su veicoli che si muovono in differenti direzioni. Questo rappresenta una sfida nell'addestramento dei modelli, poiché essi sono chiamati ad apprendere da un vasto insieme di veicoli, ciascuno dei quali potrebbe muoversi in un modo totalmente differente dagli altri. In un contesto urbano, questa varietà di direzioni è la norma, mentre in un contesto autostradale i modelli sono addestrati su un dataset costituito da veicoli che viaggiano tutti nella stessa direzione. Pertanto, in un contesto autostradale, i veicoli che dovessero muoversi in un modo non conforme alle direzioni predefinite, verrebbero considerati anomalie. Il ridotto numero di traiettorie possibili favorisce un miglior apprendimento dei modelli e di conseguenza dei risultati più significativi. Un possibile miglioramento potrebbe essere rappresentato dalla discretizzazione dello spazio di riferimento (incrocio stradale), mediante la definizione di una griglia caratterizzata da celle aventi la lunghezza media dei veicoli e l'ampiezza delle corsie stradali. In tal modo, i modelli potrebbero essere utilizzati per la predizione della posizione dei veicoli all'interno della griglia, anziché prevedere le loro coordinate. Questo approccio garantirebbe certamente una migliore explanability dei modelli. Inoltre, considerando i risultati poco soddisfacenti del modello NBRS, il quale si basa esclusivamente sui dati dei vicini più prossimi del veicolo target, potrebbe essere interessante sviluppare un terzo modello di machine learning che sfrutti i punti di forza dei modelli HIST e NBRS. In aggiunta, sarebbe opportuno esplorare l'applicazione delle tecniche basate su Attention-LSTM al fine di consentire al nuovo modello, basato su un vasto insieme di dati, di individuare quelli che hanno un impatto significativo nella predizione dell'output. L'applicazione delle tecniche Attention-based, migliorerebbe inoltre l'explanability dei modelli, e potrebbero essere utilizzare per svolgere delle analisi inerenti all'influenza che le traiettorie storiche del veicolo target e la posizione dei veicoli vicini, hanno nella predizione della traiettoria del target.

Bibliografia

[1] Giancarlo Fortino et al. «Internet of things as system of systems: A review of methodologies, frameworks, platforms, and tools». In: *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 51.1 (2020), pp. 223–236.

[2] Lei Lin et al. «Vehicle Trajectory Prediction Using LSTMs With Spatial–Temporal Attention Mechanisms». In: *IEEE Intelligent Transportation Systems Magazine* 14.2 (2022), pp. 197–208. DOI: 10.1109/MITS.2021.3049404.

[3] ByeoungDo Kim et al. *Probabilistic Vehicle Trajectory Prediction over Occupancy Grid Map via Recurrent Neural Network*. 2017. arXiv: 1704.07049 [cs.LG].

[4] Nachiket Deo e Mohan M. Trivedi. *Convolutional Social Pooling for Vehicle Trajectory Prediction*. 2018. arXiv: 1805.06771 [cs.CV].

[5] *Reti neurali riccorenti*. https://www.ibm.com/it-it/topics/recurrent-neural-networks.

[6] Richard Sproat e Navdeep Jaitly. *RNN Approaches to Text Normalization: A Challenge*. 2017. arXiv: 1611.00068 [cs.CL].

[7] M. Schuster e K.K. Paliwal. «Bidirectional recurrent neural networks». In: *IEEE Transactions on Signal Processing* 45.11 (1997), pp. 2673–2681. DOI: 10.1109/78.650093.

[8] Jurgen Schmidhuber Sepp Hochreiter. «Long Short-Term Memory». In: (1997).

[9] Haşim Sak, Andrew Senior e Françoise Beaufays. *Long Short-Term Memory Based Recurrent Neural Network Architectures for Large Vocabulary Speech Recognition*. 2014. arXiv: 1402.1128 [cs.NE].

[10] Wojciech Zaremba, Ilya Sutskever e Oriol Vinyals. *Recurrent Neural Network Regularization*. 2015. arXiv: 1409.2329 [cs.NE].

[11] Dzmitry Bahdanau, Kyunghyun Cho e Yoshua Bengio. *Neural Machine Translation by Jointly Learning to Align and Translate*. 2016. arXiv: 1409.0473 [cs.CL].

[12] Minh-Thang Luong, Hieu Pham e Christopher D. Manning. *Effective Approaches to Attention-based Neural Machine Translation*. 2015. arXiv: 1508.04025 [cs.CL].

[13] Kyunghyun Cho et al. *Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation*. 2014. arXiv: 1406.1078 [cs.CL].

[14] Yanjun Huang et al. «A Survey on Trajectory-Prediction Methods for Autonomous Driving». In: *IEEE Transactions on Intelligent Vehicles* 7.3 (2022), pp. 652–674. DOI: 10.1109/TIV.2022.3167103.

[15] Xiao Wang et al. «Capturing Car-Following Behaviors by Deep Learning». In: *IEEE Transactions on Intelligent Transportation Systems* 19.3 (2018), pp. 910–920. DOI: 10.1109/TITS.2017.2706963.

[16] Mofan Zhou, Xiaobo Qu e Xiaopeng Li. «A recurrent neural network based microscopic car following model to predict traffic oscillation». In: *Transportation Research Part C: Emerging Technologies* 84 (2017), pp. 245–264. ISSN: 0968-090X. DOI: https://doi.org/10.1016/j.trc.2017.08.027. URL: https://www.sciencedirect.com/science/article/pii/S0968090X17302371.

[17] *Next generation simulation (NGSIM). Federal Highway Administration*. https://ops.fhwa.dot.gov/trafficanalysistools/ngsim.htm.

[18] Joseph Redmon et al. *You Only Look Once: Unified, Real-Time Object Detection*. 2016. arXiv: 1506.02640 [cs.CV].

[19] Christian Szegedy et al. *Going Deeper with Convolutions*. 2014. arXiv: 1409.4842 [cs.CV].

[20] Adam Paszke et al. *PyTorch: An Imperative Style, High-Performance Deep Learning Library*. 2019. arXiv: 1912.01703 [cs.LG].

[21] Diederik P. Kingma e Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG].

[22] Boris Polyak. «Some methods of speeding up the convergence of iteration methods». In: *Ussr Computational Mathematics and Mathematical Physics* 4 (dic. 1964), pp. 1–17. DOI: 10.1016/0041-5553(64)90137-5.

[23] Geoffrey E. Hinton et al. *Improving neural networks by preventing co-adaptation of feature detectors*. 2012. arXiv: 1207.0580 [cs.NE].

Ringraziamenti

Cari,
sono giunto alla fine di questo percorso e non posso fare a meno di sentire un profondo senso di orgoglio. È stato un viaggio intenso ed elettrizzante, pieno di sfide, successi, e momenti indimenticabili che hanno plasmato chi sono oggi. In questi cinque anni, ho imparato molto e ho avuto la fortuna di incontrare persone straordinarie.

Vorrei esprimere la mia profonda gratitudine a tutti coloro che hanno fatto parte di questa avventura. Innanzitutto, ai miei genitori che mi hanno sostenuto in ogni momento. A tutti i miei amici, nessuno escluso, con i quali ho condiviso momenti indelebili, grandi risate e bicchieri di vino. Ai miei relatori, nonché mentori, i Professori Giancarlo Fortino, Roberto Minerva, Noel Crespi e l'Ing. Claudio Savaglio, che mi hanno ispirato e guidato.

Mi auguro che questo sia solo l'inizio di un radioso futuro. **Signa Inferre!**