Progetto Comunicazioni Multimediali: Predizione traiettorie pedoni

Simone Zamboni – 178641 – Ingegneria dell'informazione e Organizzazione d'Impresa

Sommario

Sommario 1
Introduzione 2
Obbiettivi 2
Modello di riferimento 3
Note sui risultati 5
Note implementative 5
Librerie 5
I dati 5
Test 6
Implementazione varianti 6
Goal 6
Social Array 8
Goal + Social Array 9
Risultati 10
Legenda tabella risultati 10
Altri parametri 11
Tabella dei risultati 12
Conclusioni 13
Sviluppi futuri 15
Riferimenti 15

Introduzione

Un software in grado di analizzare lo streaming video di una telecamera di sicurezza installata in luoghi pubblici e rilevare i comportamenti anomali potrebbe individuare subito situazioni critiche e segnalare agli agenti di sorveglianza le anomalie. Questo comporterebbe una riduzione del personale necessario per la sorveglianza e una maggiore sicurezza nei luoghi pubblici.

Per avere un software con questa funzionalità è necessario un algoritmo in grado di prevedere il comportamento "normale" delle persone, così da poter individuare gli scostamenti tra la previsione del comportamento usuale e quello che sta accadendo realmente. Se questa differenza è elevata vuol dire che è quindi in atto un comportamento anomalo. In questo progetto il comportamento di un individuo è identificato come la traiettoria che esso percorre durante la sua permanenza nel video.

La sfida è quindi quella di creare un algoritmo in grado di prevedere le traiettorie dei passanti. La difficoltà nel creare un simile modello è che le persone camminano in maniera complessa e quindi prevedere la loro futura posizione è difficoltoso.

Sono stati proposti vari modelli del comportamento dei pedoni: il modello lineare, Collision Avoidance (descritto in [1]) , Social force (descritto anch'esso in [1]) , e l'Iterative Gaussian Process (esposto in [2]) .

Un'idea innovativa in questo campo è stata quella di utilizzare le reti neurali ricorrenti di tipo LSTM ("Long Short Term Memory") che contengono al loro interno uno stato e possono così prevedere la traiettoria futura del pedone tenendo in considerazione le azioni precedenti. In questa implementazione ad ogni passante è associata una cosiddetta "cella" LSTM. Il modello che utilizza questa tecnica è descritto nella pubblicazione [3].

Questo progetto parte dalla base di conoscenza della suddetta pubblicazione e dal codice (in linguaggio Python con l'utilizzo della libreria Tensorflow) che è stato indicato come l'implementazione ufficiale della pubblicazione (reperibile al link [4]) .

Obbiettivi

L'obbiettivo del progetto è quello di modificare il modello originale per poi verificare se queste nuove varianti del modello di partenza hanno prestazioni superiori. Le modifiche verranno effettuate a partire dal codice dell'implementazione ufficiale del modello e verranno poi effettuati dei test per rilevare quale dei modelli performa meglio, utilizzando gli stessi parametri di configurazione per tutti i modelli e confrontando l'errore di posizionamento finale nel test.

Le modifiche da apportare sono le seguenti:

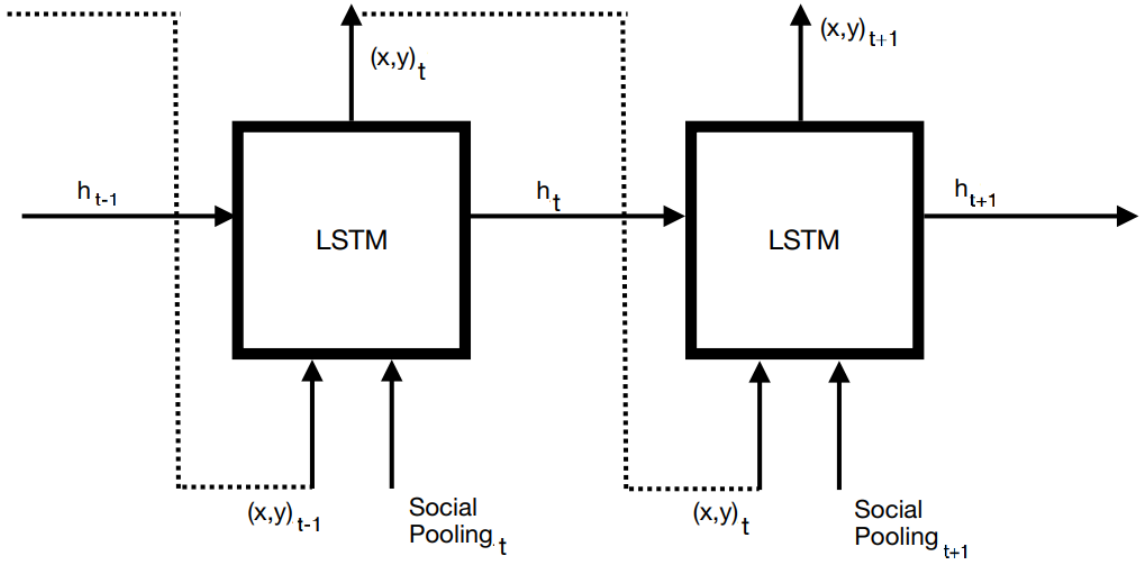
1. Aggiungere agli attuali input del modello (che sono la posizione del pedone e la griglia dei pedoni adiacenti) l'obbiettivo del pedone, definito come l'ultima posizione conosciuta del pedone nel video;
2. Sostituire la griglia dei pedoni adiacenti con un vettore contenente le posizioni dei pedoni più vicini;
3. Combinare le due modifiche precedenti in un unico modello

Modello di riferimento

Nel modello della pubblicazione [3] ogni pedone è una cella LSTM. Ogni cella ha uno stato e produce un nuovo stato e un output basandosi sugli input ricevuti e sullo stato precedente.

L'input è formato da due parti: la posizione corrente del pedone e la griglia sociale che rappresenta i pedoni adiacenti in quel momento.

Di seguito uno schema che riassume il funzionamento del modello:



Nel quale il quadrato chiamato LSTM rappresenta la cella LSTM che simula il comportamento del pedone, "h" rappresenta lo stato della cella, (x,y) la coppia di coordinate indicanti la posizione del pedone, che sono prima utilizzate come parte input della cella e che costituiscono poi l'output della cella, Social Pooling indica le informazioni riguardanti la posizione e lo stato degli altri passanti (cioè la griglia sociale) fornite in input alla cella, t-1 il momento precedente, t il momento attuale e t+1 il momento successivo.

Dopo questa introduzione generale andiamo ora più nel dettaglio ad analizzare tutte le componenti e le operazioni effettuate dal modello.

La posizione del pedone sono due coordinate in metri normalizzate tra -1 e 1 e rappresentano la posizione x e y del pedone al frame corrente. Queste coordinate vengono poi trasformate da una "embedding function" che nel codice è stata implementata con una moltiplicazione tra il vettore delle coordinate e una matrice di pesi, a cui va sommato un altro vettore (detto vettore dei "biases"). Al risultato di questa moltiplicazione va poi applicata una funzione di attivazione di tipo "rectifier". Nella pubblicazione questa operazione è descritta dalla seguente formula:

$$e_t^i = \phi(x_t^i, y_t^i; W_e)$$

Nella quale e_t^i rappresenta la parte dell'input riguardante delle coordinate del pedone i al tempo t, x_t^i la coordinata x del pedone i al tempo t, y_t^i la coordinata y del pedone i al tempo t, W_e la matrice dei pesi (e il vettore con i "biases") utilizzata per questa operazione e Φ la funzione di attivazione di tipo rectifier.

La griglia dei pedoni adiacenti è una griglia quadrata di lato N che ha profondità D, con D che rappresenta la dimensione dello stato della cella. Questa griglia viene così creata:

$$H_t^i(m,n,:) = \sum_{j \in \mathcal{N}_i} \mathbf{1}_{mn}[x_t^j - x_t^i, y_t^j - y_t^i] h_{t-1}^j$$

H_t^i rappresenta la griglia del pedone i al tempo t, m e n rappresentano la cella alla riga m-esima e alla colonna n-esima della griglia, j un pedone nel frame diverso dal pedone attuale, $1_{m,n}$ una funzione che assume solo i valori 0 e 1, e non è nulla solo se il pedone j è nella cella m,n della griglia, risultato ottenuto in base alla lunghezza del vettore distanza tra il pedone i e il pedone j, che vediamo calcolato nelle parentesi quadre, e rappresenta h^j_{t-1} lo stato del pedone j al tempo t-1.

Questa matrice appena creata di dimensioni NxNxD attraversa poi una fase del tutto analoga a quella osservata prima dalle coordinate, cioè viene moltiplicata per una matrice dei pesi, gli viene aggiunto un vettore e passa attraverso una funzione di attivazione di tipo "rectifier":

$$a_i^t = \phi(H_t^i;\ W_a)$$

Il termine a_i^t rappresenta la parte dell'input della cella ottenuto dalla griglia sociale, H_t^i la griglia sociale, W_a la matrice dei pesi (e il vettore con i "biases") di questa specifica operazione e come per le coordinate Φ rappresenta la funzione di attivazione di tipo rectifier.

Concatenando e_t^i con a_t^i si ottiene l'input del pedone i al tempo t.

Vengono inseriti come input nella cella e_t^i ed a_t^i concatenati insieme allo stato precedente della cella (h_i^{t-1}), e con i pesi propri della cella (W_l) si ottiene il nuovo stato (h_i^t):

$$h_i^t = \mathbf{LSTM}\left(h_i^{t-1}, e_i^t, a_t^i;\ W_l\right)$$

L'output della cella sono 5 parametri, che descrivono una distribuzione gaussiana multivariata a due dimensioni che rappresenta la distribuzione di probabilità della prossima posizione del pedone. Questi 5 parametri sono la media in x e la media in y (insieme indicate come μ_t^i), la deviazione standard in x e in y (indicata con σ_t^i) e il coefficiente di correlazione (ρ_t^i), di conseguenza le coordinate predette ($\hat{x}_t^i$, $\hat{y}_t^i$) si possono considerare come una distribuzione normale di parametri μ_t^i, σ_t^i e ρ_t^i:

$$(\hat{x}, \hat{y})_t^i \sim \mathcal{N}(\mu_t^i, \sigma_t^i, \rho_t^i)$$

L'obbiettivo dell'algoritmo è minimizzare la sommatoria negativa del logaritmo della probabilità che le coordinate reali siano nel punto delle coordinate previste (questa sommatoria è chiamata "Likelihood loss" e indicata con L^i):

$$L^i(W_e, W_l, W_p) = -\sum_{t=T_{obs}+1}^{T_{pred}} \log\left(\mathbb{P}(x_t^i, y_t^i | \sigma_t^i, \mu_t^i, \rho_t^i)\right)$$

Note sui risultati

Alcune note sui parametri di utilizzo dell'algoritmo utilizzato dagli scrittori della pubblicazione, parametri utilizzati per creare i risultati anche in questo progetto.

L'addestramento delle reti è avvenuto sull'80% dei frame annotati di 4 video su 5, lasciando il 20% dei 4 video come validazione, mentre il test è stato eseguito sul video non usato nella fase di addestramento. Il test prevede di lasciare l'algoritmo osservare il comportamento di un pedone per 8 frame e farne prevedere la traiettoria per i 12 successivi. I frame annotati hanno una distanza l'uno dall'altro di 0.4 secondi (quindi le annotazioni usate sono a 2.5 fps), ciò significa che l'algoritmo osserva per 3.2 secondi il comportamento di un pedone e ne prevede la traiettoria per i seguenti 4.8 .

Altro parametro molto importante è la dimensione dello stato delle celle rappresentanti i pedoni che è stato impostato a 128.

Note implementative

Librerie

Il codice originale è stato implementato con Python 2.7 e utilizza la libreria tensorflow con versione < 1.0. All'inizio del progetto è stata quindi effettuata una conversione modificando il codice per portarlo alla versione attuale di tensorflow (1.5), mentre la versione di Python è rimasta invariata. Tutti i test sono stati effettuati su una GPU Nvidia GTX 950M con installato il software CUDA 8.0 e CudNN 6.0 in un ambiente Linux (distribuzione Ubuntu).

I dati

Sono stati utilizzati gli stessi due dataset utilizzati nella pubblicazione e nell'implementazione. Il primo si chiama EHT e ha all'interno due video: hotel e univ. Il secondo si chiama UCY e ha all'interno tre video: zara1, zara2 e univ.

Tutti questi video sono stati registrati a 15 fps ed annotati ogni 0.4 secondi, cioè ogni 6 frame. Successivamente i creatori dei dataset hanno interpolato i dati per ottenere una annotazione per ogni frame e uniformare i due dataset. Sarebbe quindi possibile addestrare e testare l'algoritmo con 15 annotazioni al secondo, ma per rimanere coerenti con la pubblicazione [3] si è deciso di fornire in input al modello un frame ogni 6, utilizzando quindi una annotazione ogni 0.4 secondi come indicato dalla pubblicazione.

Per fare questo nel pre-processing dei dati, che viene effettuato ogni volta che viene avviato l'addestramento o i test, viene tenuto solo un frame annotato ogni 6. La frequenza con la quale i frame vengono tenuti è specificata dal parametro configurabile "takeOneEveryNFrames" presente

nel componente SocialDataLoader, così da rendere possibile fare delle prove con annotazioni a diversa distanza temporale tra di loro.

Test

I test sono stati eseguiti come indicato dalla pubblicazione [3]: lasciar osservare l'algoritmo per 8 frame il comportamento di un pedone e facendogli prevedere per i successivi 12 la traiettoria. I frame con annotazioni utilizzati sono distanti tra di loro 0.4 secondi.

Nell'implementazione ufficiale l'output di ogni cella che rappresenta un pedone è una normale multivariata a due dimensioni. Per ottenere da questa distribuzione le coordinate x e y predette viene utilizzata una funzione che prende le coordinate di un punto casuale di questa normale generata dalla cella. Di conseguenza ogni volta che si ripete lo stesso test con un modello già addestrato il risultato differisce (in ogni caso quasi mai più del 10% perché la distribuzione è molto stretta). Diventa quindi dispendioso in termini di tempo capire quale modello addestrato performa in maniera migliore su un video, senza contare che in un ambiente di simulazione utilizzare un simile metodo per far muovere un pedone virtuale lascerebbe molto al caso.

Alla luce di queste considerazioni si è deciso di utilizzare nei test come coordinate future predette dalla cella anziché un punto casuale della distribuzione di output della cella direttamente i valori μ_x e μ_y della distribuzione. Questo cambiamento ha influenzato la classifica dei migliori modelli: un modello che performava in un modo anche con questa modifica performa in maniera molto simile, ma è sufficiente testarlo una volta sola invece che più volte per valutare le performance, e può essere usato in maniera migliore per una simulazione.

Implementazione varianti

Goal

L'obbiettivo di questa variante del modello originale è quella di aggiungere agli input già esistenti del modello il "goal" del pedone. È stato definito come "goal" di un pedone l'ultima posizione conosciuta del pedone all'interno del video, cioè il punto nel quale il passante scompare dalla scena.

È quindi stato necessario modificare l'algoritmo di pre-processing dei dati contenuto nella classe SocialDataLoader. Questo metodo quando viene richiamato legge il file .csv delle annotazioni di ogni video e crea un array nel quale per ogni frame vi è la lista degli ID dei pedoni con la posizione x e y, e poi salva questo array in un file .pkl (nel caso venga richiamato nella fase di addestramento gli array sono due, uno contenente i dati di training e l'altro quelli per la validazione). La nuova variante del componente, oltre ad effettuare le operazioni precedentemente descritte, scorre tutti i frame di ogni video e per ogni pedone salva l'ultima posizione conosciuta, per poi memorizzare queste informazioni nel file .pkl .

In fase di training o test il file .pkl dei dati pre-processati viene elaborato per ricavare l'input di ogni cella nel metodo netx_batch() della classe SocialDataLoader. In questa nuova variante è stato modificato quel metodo in modo che per ogni pedone venga ampliato il vettore delle coordinate in modo che al suo interno siano presenti, oltre alle coordinate della posizione attuale del passante, anche le coordinate del suo obbiettivo.

Questo nuovo vettore di 4 coordinate passa attraverso lo stesso processo di moltiplicazione con la matrice dei pesi, somma con il vettore dei "biases" e trasformazione da parte della funzione di attivazione, che veniva precedentemente attraversato solamente dalle coordinate posizione del pedone.

Quindi nel modello originale:

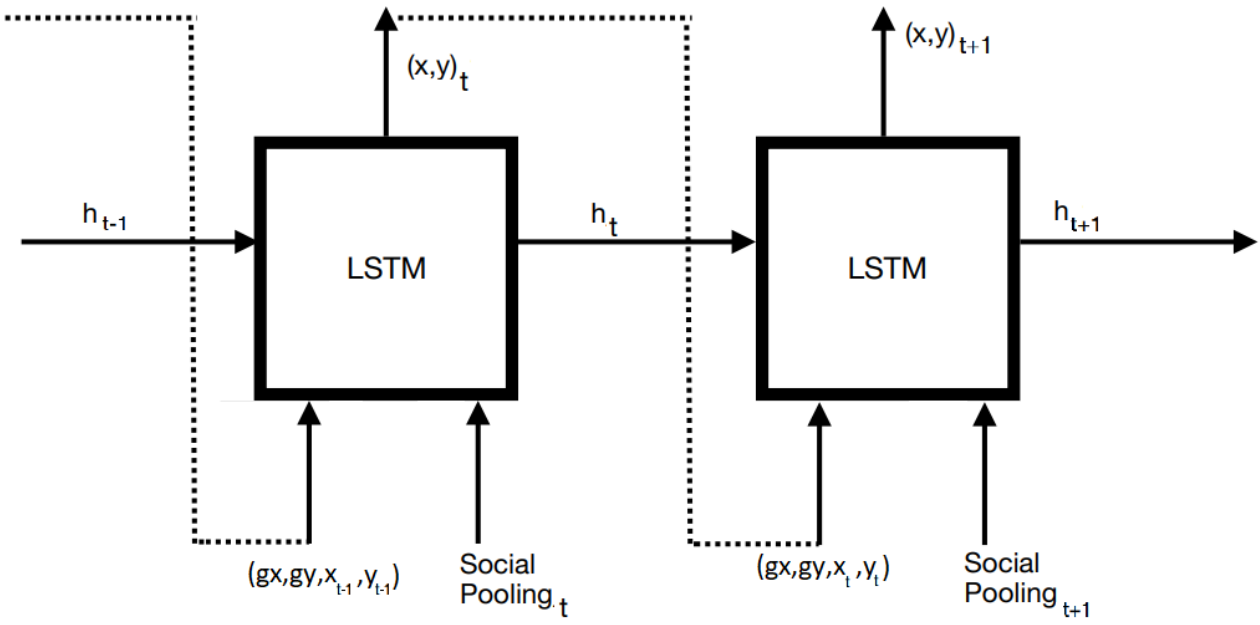
$$e_t^i = \phi(x_t^i, y_t^i; W_e)$$

Mentre in questa variante del modello (con gx^i la coordinata x dell'obbiettivo del pedone i e gy^i la coordinata y dell'obbiettivo del pedone i):

$$e_t^i = \phi(x_t^i, y_t^i, gx^i, gy^i; W_e)$$

Le righe della matrice dei pesi W_e sono quindi passate da 2 a 4 per poter essere moltiplicata con il nuovo vettore di coordinate, mentre il resto del modello è rimasto invariato.

Lo schema del modello è di conseguenza modificato, perché in input quella che prima era la coppia delle coordinate diventa un vettore di 4 numeri che corrispondo alla coppia di coordinate della posizione attuale del pedone e a un'altra coppia di coordinate corrispondente al goal del pedone:



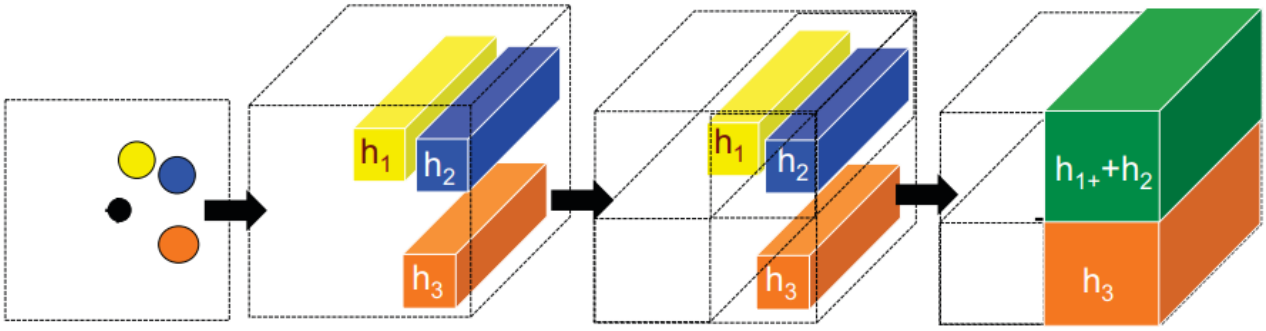
La motivazione che hanno spinto a creazione di questa variante è quella di fornire all'algoritmo informazioni aggiuntive per fare in modo possa prevedere meglio la traiettoria dei passanti. Sapendo dove un pedone scomparirà dalla scena è ragionevole pensare che si potrebbe prevedere meglio la sua traiettoria. Vi possono però essere anche alcuni svantaggi nell'aggiungere questa informazione: i pedoni si comportano in modo complesso e si possono muovere in varie direzioni nella scena, anche dalla parte opposta rispetto al loro "goal", rendendo potenzialmente questa nuova informazione aggiuntiva disorientante. Un classico esempio di questo comportamento è quando un pedone si ferma o modifica radicalmente la sua traiettoria per vedere una vetrina, esso non sta quindi andando direttamente verso il suo obbiettivo, e di conseguenza l'algoritmo potrebbe

essere disorientato dall'informazione della posizione dell'uscita di scena di quel determinato pedone.

Nella tabella dei risultati e nelle conclusioni potremo verificare se queste informazioni aggiuntive migliorano le prestazioni o disorientano l'algoritmo.

Social Array

La parte sociale di questo algoritmo è data da una griglia NxNxD centrata sul pedone che rappresenta per ogni cella della griglia la sommatoria degli stati dei pedoni presenti nel frame in quella cella. La presenza di un determinato pedone in una determinata cella è data dalla posizione di questo pedone rispetto al pedone su cui la griglia è centrata. Qui di seguito un'illustrazione dalla pubblicazione per vedere come viene creata la griglia sociale:



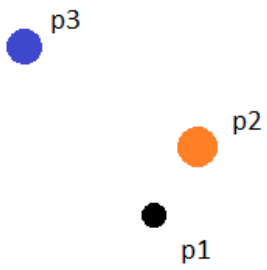
L'obbiettivo di questa variante è di sostituire questa griglia con un vettore contenente le posizioni dei pedoni vicini. È stato quindi modificato il componente che calcola la griglia per invece creare un array.

Questa nuova implementazione per ogni pedone prima calcola la distanza con tutti gli altri pedoni, poi popola l'array con le coordinate x e y degli altri pedoni in ordine dal più vicino (in posizione 0) al più lontano (in posizione N-1 nell'array). Questo array ha una dimensione fissa N*2, con N numero di pedoni vicini che si vuole considerare, numero moltiplicato per un fattore due perché per ognuno di questi pedoni viene memorizzata la coordinata x e la coordinata y.

Nel caso in cui il numero di pedoni nel frame è maggiore di N+1, nell'array di ogni pedone saranno presenti solo gli N passanti più vicini. La posizione 0 nell'array conterrà il pedone più vicino a quello di cui viene calcolato l'array, e alla posizione N-1 il pedone più lontano che però rientra negli N più vicini.

Nel caso in cui N >= numero di pedoni nel frame vuol dire che non si hanno abbastanza passanti per riempire completamente l'array sociale. In questo caso lasciare a 0 le posizioni vuote nell'array sociale crea dei problemi, perché l'algoritmo interpreterebbe questi 0 come dei pedoni fermi al centro della scena, e di conseguenza farebbe in modo stare lontano da quella posizione. Per evitare questo inconveniente e lasciare una certa flessibilità all'algoritmo, in modo che possa operare anche in scene con un numero di pedoni <= N, negli spazi altrimenti vuoti dell'array vengono ripetute le coordinate dei pedoni già nell'array, a partire da quelle del passante in posizione 0.

Di seguito un esempio:



Se N = 5 e vi sono 3 pedoni (p1,p2,p3) nel frame, assumendo dobbiamo calcolare l'array del passante p1, e il più vicino a questo pedone sia p2, l'array di p1 sarà così composto: [p2x, p2y, p3x, p3y, p2x, p2y, p3x, p3y, p2x,p2y], cioè avremo in ordine le coordinate di p2,p3,p2,p3 e p2. Questo accade perché prima l'array viene popolato con la prima posizione occupata da p2 perché è più vicino, e poi p3 perché è più lontano. Successivamente vengono riempite le 3 posizioni (corrispondenti a 6 coordinate) che sono vuote, ripetendo le coordinate dei pedoni già presenti nell'array partendo dal passante in posizione 0, fino a quando non si riempie completamente l'array.

Questo metodo permette di utilizzare l'algoritmo sia in video molto popolati che in video meno popolati, nei quali viene data molta importanza ai pochi pedoni nella scena poiché i pochi passanti presenti si ritrovano più volte nell'array.

Nel remoto caso in cui in un frame ci sia solo un pedone p1, in ogni posizione dell'array di questo passante solitario viene inserito un pedone fittizio con coordinate x: x_{p1}-2 e y: y_{p1}-2, in modo che questo passante inventato sia molto lontano (essendo tutte le coordinate comprese tra -1 e 1 se p1 è nell'angolo in alto a destra il pedone fittizio sarà appena oltre l'angolo in basso a sinistra del video) e quindi non influenzi il comportamento del pedone solitario.

Da notare che con un parametro N = 4, parametro che nell'implementazione e nella tabella dei risultati chiameremo parametro "grid_size", il modello Social_LSTM considera 16 celle (N x N) in cui ognuna può avere al suo interno più di un pedone, mentre nella variante Social Array (descritta in questo capitolo) con N = 4 il vettore è di lunghezza 8 (N*2) e quindi l'algoritmo può considerare al massimo solo 4 pedoni nelle vicinanze.

Il modello originale che utilizza la griglia può considerare posizioni vuote, ha un elemento sociale più approfondito perché considera anche gli stati dei pedoni nella griglia e non ha un limite al numero di pedoni considerati perché per ogni cella viene fatta la somma degli stati dei pedoni al suo interno. Dall'altro lato il modello ad array ha informazioni molto più precise (infatti utilizza la posizione esatta e non approssimata su una griglia) sui più importanti passanti nel frame, cioè quelli più vicini al pedone di cui si calcola l'array. Queste coordinate inoltre vengono già ordinate per distanza così da aiutare l'algoritmo a focalizzare l'attenzione ancora di più sui pedoni più vicini. La variante ad array si caratterizza quindi per una minor numero di informazioni, ma più precise ed importanti. Queste caratteristiche potrebbero dare un vantaggio maggiore a questa variante rispetto al modello originale perché la cella potrebbe imparare meglio con informazioni più chiare e coincise.

Goal + Social Array

In questo modello sono stati combinate le due varianti precedentemente descritte in un unico modello, così da combinare i benefici dei due modelli.

L'implementazione è stata una semplice unione del codice degli altri 2 modelli e non sono state apportate altre modifiche.

Risultati

Legenda tabella risultati

Nella tabella in fondo a questo capitolo vi è riportato il risultato dei test condotti sui vari modelli con diversi parametri di input.

Nella colonna “Modello” vi è descritto il modello usato: Social_LSTM per l’implementazione originale, Goal per la variante con il “goal” e Social Array per la variante che sostituisce la griglia con un array e Goal+Array per la variante che combina il “goal” e l’array al posto della griglia.

Nella colonna “Video test” è indicato su quale video è stato effettuato il test del modello. Come descritto nella pubblicazione ogni rete è stata addestrata sugli altri quattro video e poi testata su quello rimanente. I video sono 5 e sono stati enumerati come segue:

0 = dataset UCY - video zara01

1 = dataset UCY - video zara02

2 = dataset EHT - video univ

3 = dataset ETH - video hotel

4 = dataset UCY - video univ

Il numero di frame dei video considerando un frame con annotazioni ogni 0.4 secondi, sono:

0 = 867 (nel train 694 di training e 173 di validazione)

1 = 1051 (nel train 841 di training e 210 di validazione)

2 = 1433 (nel train 1147 di training e 286 di validazione)

3 = 1900 (nel train 1520 di training e 380 di validazione)

4 = 540 (nel train 432 di training e 108 di validazione)

Nel caso in cui in questa colonna sia presente la parola “Media” invece del numero del video significa che quei risultati si riferiscono alla media dei risultati di quel modello su tutti i video.

La colonna “Grid size” indica il lato della griglia sociale che è di dimensione grid_size x grid_size x lunghezza_stato. Nel caso del modello Social Array (e Goal+Array) indica quanti pedoni può contenere l’array sociale (che sarà un vettore di dimensione grid_size*2).

La colonna “N epochs” indica per quante epoch è stato fatto effettuato l’addestramento per generare i risultati ottenuti.

La colonna “Best train epoch” indica la epoch con la validation loss minore, valore presente nella colonna “Best train epoch val. loss”.

La colonna “Best test epoch” indica il modello di quale epoch è stato usato per generare l’errore minore nel test, valore riportato nella colonna “Best test epoch error”. Da notare che la pubblicazione definisce tre tipi di errore, e questi risultati si riferiscono solamente all’errore di tipo “Average displacement error”. Il valore di errore ritornato dall’algoritmo è stato moltiplicato per un

fattore 15 per poterlo denormalizzare e ottenere l'errore in metri, di conseguenza il valore nella colonna "Best test epoch error" è in metri.

Altri parametri

Altri parametri che è possibile impostare nel codice ma sono stati lasciati invariati ai valori di default sono:

- rnn_size = 128, (quanto è grande il vettore di stato delle celle RNN)
- maxNumPeds = 70 (numero massimo di pedoni in un frame gestito dall'algoritmo)
- batch_size = 16
- grad_clip = 10,
- learning_rate = 0.005,
- decay rate = 0.95,
- drop_out_keep_probability = 0.8 ,
- embedding_size = 64, (le colonne della matrice e la lunghezza del vettore dei "biases" utilizzati per la creazione di a_i^t ed e_i^t)
- lambda_param = 0.0005
- neighborhood_size = 32
- validation percentage = 0.2 (qual è la percentuale di ogni video utilizzato nella fase di addestramento destinata alla validazione)
- observation_length = 8 (nel test quanti frame osserva l'algotirmo prima di predirre)
- prediction_length = 12 (nel test quanti frame l'algoritmo deve predirre dopo aver osservato observation_length frame)
- sequence_length = 20 (questo valore indica quanto è lunga una sequenza di frame consecutivi su cui viene fatto l'addestramento; per questo parametro è stato scelto il numero 20 perché nel test l'algoritmo osserva per 8 frame e predice gli altri 12, quindi è parso opportuno effettuare l'addestramento su sequenze lunghe 8+12=20 frame)

In tutti i casi di test e per tutti i video sono state utilizzate annotazioni ogni 0.4 secondi, come indicato nella pubblicazione.

Tabella dei risultati

Qui si seguito la tabella dei risultati:

Modello	Video test	Grid size	N epochs	Best train epoch	Best train epoch val. loss	Best test epoch	Best test epoch error
Social LSTM	0	4	10	10	-54.01	10	1.362
Goal	0	4	10	6	-27.42	5	1.225
Social Array	0	8	10	8	**-57.88**	5	**1.127**
Goal+Array	0	8	10	5	-27.61	6	1.135
Social LSTM	1	4	10	9	-52.76	2	1.625
Goal	1	4	10	5	-26.64	8	1.466
Social Array	1	8	10	9	**-52.82**	2	2.112
Goal+Array	1	8	10	7	-27.83	8	**1.391**
Social LSTM	2	4	10	4	-39.02	8	1.440
Goal	2	4	10	6	-26.05	10	1.196
Social Array	2	4	10	9	-42.31	8	1.160
Social Array	2	16	10	10	**-42.66**	4	0.983
Social Array	2	64	10	8	-42.93	6	1.258
Social Array	2	128	10	6	-40.32	4	1.164
Goal+Array	2	8	10	6	-26.35	9	**0.964**
Social LSTM	3	4	10	9	**-40.06**	4	1.155
Goal	3	4	10	6	-26.11	3	1.143
Social Array	3	8	10	9	-39.06	8	**0.944**
Goal+Array	3	8	10	10	-21.7	10	0.952

Social LSTM	4	4	10	10	**-48.91**	3	**2.070**
Goal	4	4	10	8	-37.05	8	3.165
Social Array	4	8	10	8	-47.31	10	2.475
Goal+Array	4	8	10	7	-36.34	8	3.210
Social LSTM	Media	4	10	-	-46.95	-	1,530
Goal	Media	4	10	-	-28.65	-	1,640
Social Array	Media	8 (16 nel video 2)	10	-	**-47.95**	-	**1.528**
Goal+Array	Media	8	10	-	-27.96	-	1.530

Conclusioni

Osservando la tabella dei risultati possiamo effettuare delle interessanti considerazioni sui vari modelli presi in considerazione.

La variante con l'obbiettivo presenta un valore di training validation loss molto più alto rispetto al modello originale, senza che però questo influenzi il risultato sul test. Infatti il modello "goal" presenta una performance sempre migliore dell'implementazione originale (a parte nel video 4). Da queste informazioni possiamo concludere che aggiungere l'informazione dell'obbiettivo del pedone aiuta in maniera sostanziale l'algoritmo a prevedere il comportamento dei passanti in fase di test.

La variante che sostituisce alla griglia sociale l'array presenta un valore di validation loss quasi sempre migliore del modello originale, e contemporaneamente nei test performa sempre (a parte nei video 1 e 4) in maniera migliore del modello della pubblicazione [3]. Questo può indurre a pensare che gli esseri umani ragionino in maniera più simile a questo modello, rispetto al modello con la griglia, quando si tratta di tenere in considerazione le posizioni dei passanti vicini, e ciò permette quindi a questa variante di ottenere migliori risultati rispetto a quelli del modello originale.

Per quanto riguarda l'influenza del parametro che specifica il numero di pedoni presenti nell'array (nella tabella chiamato grid_size), possiamo osservare i test appositamente effettuati per questo proposito sul video 2. In questi test si è cercato di osservare come si comporta il modello Social Array con numeri molto diversi per il numero di pedoni adiacenti considerati. Con un valore di 4 l'algoritmo performa già meglio del modello di riferimento, mentre con un valore di 16 il risultato nel test migliora ancora. Questo perché nel video 2 i pedoni in un frame sono spesso meno di 16, quindi con un array capace di contenere le posizioni di altri 16 pedoni l'algoritmo ha a disposizione tutte le informazioni della scena ed è in grado di prevedere con più accuratezza il comportamento dei passanti. Con dei valori molto elevati, come 64 e 128, notiamo invece che le performance del modello peggiorano notevolmente, arrivando ad essere addirittura inferiori rispetto a quando l'algoritmo tiene in considerazione solo 4 pedoni. Questo perché in un frame del video 2 non ci sono

mai più di 40 passanti, quindi la maggior parte delle posizioni dell'array sono occupate da pedoni ripetuti, di conseguenza l'algoritmo si trova di fronte a molte informazioni principalmente inutili che lo disorientano. Possiamo quindi concludere che il numero dei pedoni considerati è un parametro molto importante ai fini delle performance dell'algoritmo e deve essere calibrato in base ai video di addestramento, di test e inoltre deve rispecchiare il numero di passanti vicini che realmente una persona prende in considerazione nel decidere i suoi spostamenti, così da simulare in maniera ancora migliore come si comporterebbe un essere umano. Nei test di tutti gli altri video è stato impostato un valore di 8 pedoni adiacenti considerati, perché ritenuto un buon compromesso rispetto alle motivazioni precedenti, anche rispetto alla densità di passanti nei video che non è molto elevata.

L'unione delle due varianti (modello Goal+Array) sembra combinare in maniera efficace i vantaggi (e svantaggi) dei due modelli. Infatti presenta come la variante Goal un valore di validation training loss molto più alto rispetto agli altri algoritmi, mentre poi nei test performa spesso meglio delle due varianti e del modello originale (a parte nei video 1 e 3 nei quali comunque sia avvicina di molto al miglior risultato e nel video 4). Possiamo quindi concludere che la modifica del modello originale con l'aggiunta del "goal" di ogni pedone e la sostituzione della griglia sociale con un array risulta migliorare le performance dell'algoritmo in quasi tutti i casi.

I risultati del video 4 differiscono in maniera sostanziale dal resto dei risultati (con un errore nel test 2-3 volte più alto rispetto agli altri video) probabilmente perché in quel determinato video vi è una densità di passanti estremamente più elevata rispetto a quelli utilizzati per l'addestramento. Nel video 4 in un frame vi possono essere fino a 65 persone contemporaneamente, mentre in quelli usati in fase di addestramento non ve ne sono più mai di 35, praticamente la metà. Di conseguenza questo video potrebbe non essere ideale da utilizzare in combinazione con gli altri perché molto diverso sotto quel punto di vista. Per completezza sono stati inseriti nella tabella dei risultati i valori dei test sul video 4, ma si hanno dei dubbi sulla loro correttezza e utilità.

Analizzando i risultati della media si potrebbe concludere che il miglior algoritmo è la variante dell'array, seguito poi dall'algoritmo originale e dalla variante Goal+Social a pari merito e per ultimo la variante con l'obbiettivo. Se però escludiamo nel calcolo della media i risultati del video 4 otteniamo una classifica completamente diversa:

1. Goal+Array errore medio in fase di test: 1.046
2. Goal errore medio in fase di test: 1.258
3. Social Array errore medio in fase di test: 1.317
4. Social_LSTM originale errore medio in fase di test: 1.396

Secondo questa nuova classifica tutte le varianti implementate migliorano le performance rispetto al modello originale, e il migliore modello è di gran lunga l'unione della variante con obbiettivo e array, che in tutti test performa (a parte sempre nel video 4) meglio degli altri o comunque arriva a una brevissima distanza dal miglior modello.

Da notare come i risultati sono stati ottenuti da modelli con un addestramento di durata 10 epoch, ma per ottenere dei risultati completamente attendibili che possano confermare con sicurezza assoluta quale algoritmo sia migliore sarebbe da utilizzare nella fase di addestramento un numero di epoch intorno a 100.

Si può quindi concludere che gli obbiettivi di questo progetto quindi stati raggiunti perché le varianti proposte sono state implementate e presentano performance superiori al modello di riferimento esposto in [3].

Sviluppi futuri

Come sviluppi futuri di questo progetto si potrebbe utilizzare il modello Goal+Array non solo per predizione ma anche per la simulazione di passanti in ambienti virtuali, nei quali ad ogni cella che simula un pedone viene data una posizione di inizio nella scena e le coordinate dell'obbiettivo finale, così queste celle LSTM continueranno ad aggiornare la loro posizione ad intervalli regolari (dando l'impressione di camminare) percorrendo traiettorie simili a quelle che sarebbero percorse da esseri umani in circostanze identiche a quelle della simulazione.

Per ottenere una simulazione ancora più accurata si potrebbe far in modo che l'algoritmo tenga conto non solo dei passanti vicini ma anche di eventuali ostacoli presenti nell'immagine (come gli edifici), inserendo queste informazioni nell'array che al momento contiene solo informazioni su altri pedoni.

Riferimenti

[1] K. Yamaguchi, A. C. Berg, L. E. Ortiz, and T. L. Berg. "Who are you with and where are you going?" In Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on, pages 1345–1352. IEEE, 2011. 2, 3, 5, 6, 7, 8

[2] P. Trautman, J. Ma, R. M. Murray, and A. Krause. "Robot navigation in dense human crowds: the case for cooperation." In Robotics and Automation (ICRA), 2013 IEEE International Conference on, pages 2153–2160. IEEE, 2013. 5

[3] A. Alahi, K. Goel, V. Ramanathan, A. Robicquet, L. Fei-Fei and S. Savarese, "Social LSTM: Human Trajectory Prediction in Crowded Spaces," 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, 2016, pp. 961-971.

[4] https://github.com/vvanirudh/social-lstm-tf