RNN Convoluzionale: un modello migliorativo per estrarre features da dati sequenziali

1. Introduzione

I livelli convoluzionali tradizionali estraggono features da patches di dati applicando non linearità su una funzione affine dell'input. Il modello che si propone è un miglioramento del processo di features extraction per il caso di dati sequenziali, inserendo patches di dati in una rete neurale ricorrente e usando gli outputs o gli stati nascosti delle unità ricorrenti per effettuare l'estrazione delle features.
In questo modo, si valorizza il fatto che una window, contenente pochi frames di dati sequenziali, è se stessa una struttura e questa struttura aggiuntiva potrebbe incapsulare informazioni valutabili.
Inoltre, in questo modo, si consentono più passaggi di calcolo nel processo di features extraction, che è potenzialmente vantaggioso perché può risultare che una funzione affine seguita da non linearità restituisca delle features che siano troppo semplici.

Quando si applica un livello convoluzionale su dei dati, questo livello estrae features dalle patches locali e, spesso, è seguito da un meccanismo di pooling per raggruppare i valori delle features sulle patches vicine. Le caratteristiche estratte possono essere l'input del livello successivo in una rete neurale, possibilmente un altro livello convoluzionale o un classificatore. I modelli che utilizzano livelli convoluzionali per l'estrazione di funzionalità dai dati grezzi possono superare i modelli che utilizzano features realizzate a mano e ottenere risultati all'avanguardia.
Le reti neurali ricorrenti (RNN), sono modelli per il processamento dei dati sequenziali che sono, spesso, di varia lunghezza come testo o suono. Le Long-Short Term Memory networks (LSTM) sono un tipo speciale di reti neurali ricorrenti che utilizzano un meccanismo di gating per una migliore modellazione delle dipendenze a lungo termine nei dati. Questi modelli sono stati utilizzati con successo per lo speech recognition e la machine translation.
Quando il dato è una sequenza di frames e di varia lunghezza, un modello che combina entrambi, strati convoluzionali e strati ricorrenti, può essere costruito nel seguente modo: i livelli convoluzionali vengono utilizzati per estrarre le features dalle patches di dati che, nel caso in cui si andrà ad applicare questa tipologia di modello, sono windows composte da pochi frames consecutivi nella sequenza degli audio. Queste features, estratte dalle windows, sono un'altra sequenza temporale, che può essere l'input di un altro strato convoluzionale, o, infine, l'input di un livello ricorrente che modella le relazioni temporali nei dati. Un ulteriore vantaggio di questo metodo è l'utilizzo di un meccanismo di pooling dopo lo strato convoluzionale può accorciare la sequenza di input al livello ricorrente, in modo che i livelli ricorrenti avranno necessità di modellare le dipendenze temporali su un numero più piccolo di frames.
Una possibile limitazione degli strati convoluzionali tradizionali è che usano una non linearità applicata su funzioni affini per estrarre features dai dati. In particolare, una feature estratta viene calcolata moltiplicando per elemento per elemento una patch di dati per una matrice pesata, sommando il prodotto su tutti gli elementi della matrice, aggiungendo uno scalare di bias e applicando una funzione non lineare. Teoricamente, la funzione che mappa una patch di dati su un valore di feature scalare può essere di complessità arbitraria ed è plausibile che per ottenere una migliore rappresentazione dei dati di input, possano essere utilizzate funzioni non lineari più complicate (cioè con più livelli di calcolo). Infatti, ad esempio, un semplice stacking di più livelli convoluzionali con kernel size maggiore di 1×1 non risolve questo problema, perché un livello successivo mescolerà gli outputs del livello precedente per posizioni diverse e non mostrerà il comportamento potenzialmente desiderato di estrazione di features più complicate da una patch di dati usando solo questa stessa patch.
Nel caso di dati sequenziali, window di pochi data frames consecutivi hanno una proprietà aggiuntiva: ogni window è se stessa una piccola sequenza composta da pochi frames. Questa proprietà aggiuntiva può essere potenzialmente sfruttata per estrarre features migliori dalle windows. Viene introdotto il

Modello CRNN (Convolutional Recurrent Neural Network) che alimenta ogni window, frame per frame, in un livello ricorrente e usa gli outputs e gli stati nascosti delle unità ricorrenti in ciascun frame per estrarre le features dalle windows sequenziali. Così facendo, il contributo principale è che si può, potenzialmente, ottenere features migliori rispetto ai livelli convoluzionali standard, poiché vengono usate informazioni aggiuntive su ciascuna window (le informazioni temporali) e le features sono create dalle differenti windows utilizzando una funzione più complicata rispetto ai tradizionali strati convoluzionali (più passaggi di calcolo).

2. LSTM Convoluzionale

a. Estrazione delle features locali

Si descrive la struttura generale per un livello che estrae (raggruppate) features temporalmente locali da un set di dati di sequenze. Usando questa terminologia, possono essere descritti sia i tradizionali strati convoluzionali che il metodo, qui esposto, di Convolutional Recurrent Neural Network, (CRNN). Si supponga un dataset etichettato in cui ogni esempio nel dataset è una sequenza di un numero variabile di frame e tutti i frame hanno lo stesso numero fisso di features. Si denota *x* un esempio dal set di dati, *l* il numero di frames nella sequenza *x* e *k* il numero di funzioni in ogni frame in ogni esempio nel set di dati, quindi *x* è di dimensione $k \times l$. Si noti che per semplicità si va ad assumere che ogni frame contenga un vettore di feature unidimensionali di lunghezza *k*, ma questo può essere facilmente generalizzato al caso di dati multidimensionali in ogni frame (come i dati video) appiattendo questi dati in una dimensione. Dalla sequenza *x* creiamo una sequenza di windows, ciascuna composta da *r1* frame consecutivi. Quindi, ogni window è di dimensioni $k \times r1$, e si prende uno spostamento di *r2* frames tra il punto iniziale di due finestre consecutive. Successivamente, applichiamo una funzione *f* di estrazione delle features su ciascuna finestra per ottenere un insieme di *n* features che descrivono ciascuna finestra, il che significa che il vettore delle caratteristiche di dimensione *n* che descrive una window *w* è *f (w).* Applicando *f* su ciascuna finestra si ottiene un'altra sequenza *x* in cui ogni frame è rappresentato con *n* caratteristiche. Infine, per il pooling, si creano window di dimensione $n \times p1$ dalla sequenza *x* allo stesso modo in cui si sono create windows dalla sequenza *x*, con uno spostamento di *p2* frames tra i punti di partenza di due window consecutive, e si esegue il max-pooling tra frame (per ciascuna feature in modo separato) in ciascuna window per trasformare windows di dimensione $n \times p1$ in un vettore di dimensione $n \times 1$. Dopo aver applicato il max-pooling a ciascuna finestra, la sequenza risultante contiene elementi temporalmente locali (raggruppati) della sequenza *x* estratti dalla funzione *f* e possono essere inseriti nel livello successivo della rete o in un classificatore.

b. Long Short Term Memory networks and Bidirectional Long Short Term Memory networks

Una semplice RNN prende una sequenza (*x1, ..., xt*) e produce una sequenza (*h1, ..., ht*) di stati nascoti e una sequenza (*y1, ..., y2*) di outputs nel seguente modo:

$$h_t = \sigma(W_{xh}x_t + W_{hh}h_{t-1} + b_h) \qquad (1)$$

$$y_t = W_{hy}h_t + b_y, \qquad (2)$$

Dove $\boldsymbol{\sigma}$ è la funzione sigmoide logistica, Wxh, Whh, Why sono matrici pesate e bh, by sono biases. Come detto in precedenza, le reti LSTM sono un tipo speciale di RNN che usano un meccanismo di gating per consentire una migliore modellazione a lungo termine delle dipendenze nei dati.

La versione della LSTM che si propone qui, viene implementata rimpiazzando l'Equazione 1 con i seguenti passi:

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i) \quad (3)$$

$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f) \quad (4)$$

$$c_t = f_t c_{t-1} + i_t tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \quad (5)$$

$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_t + b_o) \quad (6)$$

$$h_t = o_t tanh(c_t), \quad (7)$$

Dove $\boldsymbol{\sigma}$ è la funzione sigmoide logistica, *i, f, o* sono, rispettivamente, i vettori di attivazione dei gate per input, livello hidden e output e *c, h* sono vettori degli stati cell (lo stato cell è un po' come un nastro trasportatore, scorre direttamente lungo l'intera catena, con solo alcune interazioni lineari minori e, quindi, è molto facile che le informazioni fluiscano inalterate) e hidden.
Poiché la standard LSTM elabora l'input solo in una direzione, viene proposto un miglioramento di questo modello, vale a dire Bidirezionale LSTM (BLSTM), in cui l'input viene elaborato sia nell'ordine standard che nell'ordine inverso, permettendo di combinare informazioni future e passate in ogni passo temporale. Un livello BLSTM viene composto da due livelli LSTM, processando l'input separatamente per produrre $\vec{\boldsymbol{h}}$, $\vec{\boldsymbol{c}}$ stati hidden e cell di una LSTM processando l'input nell'ordine standard, e, $\overleftarrow{\boldsymbol{h}}$, $\overleftarrow{\boldsymbol{c}}$ stati hidden e cell di una LSTM processando l'input nell'ordine inverso. Entrambi, $\vec{\boldsymbol{h}}$ e $\overleftarrow{\boldsymbol{h}}$, sono, quindi, combinati nel seguente modo:

$$y_t = W_{\vec{h}y}\vec{h}_t + W_{\overleftarrow{h}y}\overleftarrow{h}_t + b_y, \quad (8)$$

in modo da produrre la sequenza di output del livello di BLSTM. Da notare che è possibile usare gli stati di cell invece che quelli hidden di due livelli LSTM in un livello BLSTM, nel seguente modo:

$$y_t = W_{\vec{c}y}\vec{c}_t + W_{\overleftarrow{c}y}\overleftarrow{c}_t + b_y, \quad (9)$$

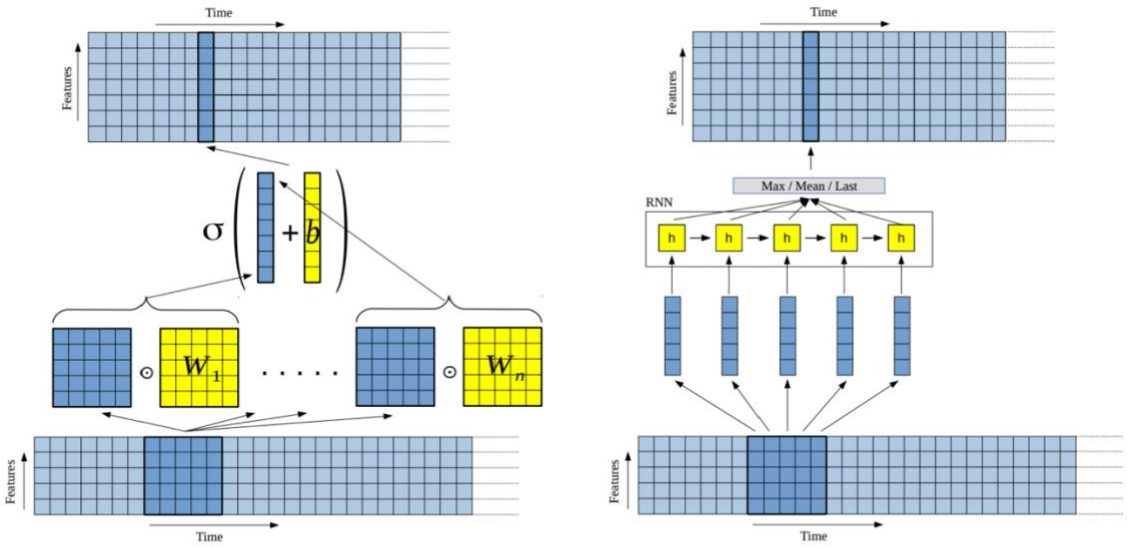
in modo da produrre la sequenza di output del livello di BLSTM.

c. Livelli Convoluzionali e Livelli Ricorrenti Convoluzionali

I livelli convoluzionali tradizionali (con il max_pooling) possono essere descritti come fatto precedentemente (nel paragrafo 2a), settando la funzione f per l'estrazione di *n* features da una patch di dati *w*:

$$f(w) = (\sigma(\mathrm{sum}(W_1 \odot w) + b_1), \ldots, \sigma(\mathrm{sum}(W_n \odot w) + b_n)),$$

Dove *W1, ..., Wn* sono matrici pesate, *b1, ..., bn* sono biases, è una moltiplicazione elemento per elemento, e *sum* è un operatore che somma tutti gli elementi di una matrice. Questo da origine a n features differenti computate per ogni window usando le n matrici pesate e biases. I valori di una specifica feature sulle windows vengono tradizionalmente chiamate 'feature map'. La Figura 1a descrive la procedura di estrazione delle feature in un livello convoluzionale standard.



(a) Extraction of features from a time window using a standard convolutional layer. For each feature the time window is multiplied element-wise by a different matrix. This product is summed and added to a bias term and then a non-linearity is applied. Here ⊙ denotes an element-wise multiplication followed by summation.

(b) Extraction of features from a time window using a CRNN layer. First, the time window is fed frame by frame into a recurrent layer. Then, the hidden states of the recurrent layer along the different frames of the window are used to compute the extracted features, by applying a max/mean operator or simply by taking the vector of hidden states of the last time frame in the window. Note that it is possible to extract features from the window using the outputs of the recurrent layer in each time step instead of the hidden states.

Fig. 1. Extraction of features from a time window by a traditional convolutional layer (a) and a CRNN layer (b).

Di seguito vengono esposti due Convolutional Recurrent Neural Network (CRNN), in particolare due casi di questa architettura, Convolutional Long Short Term Memory (CLSTM) e Convolutional Bidirectional Long Short Term Memory (CBLSTM), descrivendo la funzione di feature extraction *f(w)* che, sostanzialmente, corrisponde a questi livelli proposti.
Un livello CRNN ha come principale vantaggio che ogni window *w* di size *k* x *r1* può anche essere interpretata come una sequenza temporale breve di *r1* frames, ognuno caratterizzato da una size *k* x 1. In questo modo, si può caratterizzare una window w come una sequenza di stati hidden (h1, …, hr1), con ogni elemento di size n x 1, dove n è il numero di unità nel livello RNN. Infine, con lo scopo di creare il vettore di feature di lunghezza n, che rappresenta una window w, si può settare f(w) ad essere la media o il massimo sui vettori della sequenza (h1, …, hr1) (computando la media o il massimo per ogni feature separatamente), o, semplicemente, settando f(w) per essere l'ultimo elemento della sequenza. La figura 1b descrive il processo di feature extraction in un livello CRNN. Come caso speciale di CRNN, come detto in precedenza, si ha CLSTM e CBLSTM, dove il livello RNN è, rispettivamente, un LSTM e un BLSTM.
Da notare che si usa lo stesso livello RNN per computare f(w) per tutte le windows w, nel senso che, come in un livello convoluzionale, vengono usati gli stessi valori di parametri del modello per estrarre features locali da tutte le windows. In più, si nota che la RNN potrebbe produrre due sequenze addizionali, una di outputs (y1, …, yr1) e una di stati cell (c1, …, cr1) (nel caso di CLSTM o BLSTM) che può essere usata al posto della sequenza di stati hidden per computare le features rappresentanti una window. Nel caso si usa una sequenza di outputs, le dimensioni hidden del livello ricorrente sono libere da vincoli.

Questo lavoro fa riferimento al seguente paper: https://arxiv.org/pdf/1602.05875v3.pdf.