Lez16_RetiRicorrenti

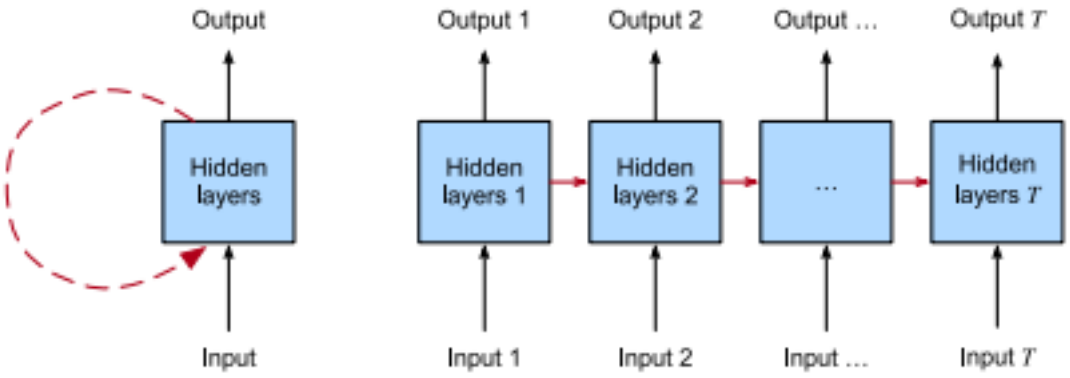
November 28, 2023

0.1 Sequenze di dati

Un esempio è un video, cioè sequenza di frame; o un audio, sequenza di segnali. Un caso generico sono le *serie temporali*, come l'andamento di una borsa. La sequenza di dati può anche essere solo in output, ad esempio, partendo da una immagine, fornire una descrizione. Infine, possiamo avere sequenza di dati sia in input, sia in output. Ne è un esempio la traduzione di un testo.

0.2 Reti neurali ricorrenti

L'informazione può presentare un ciclo, quindi non andiamo per forza sempre avanti.



Possiamo vederla come rete *feed forward srotolata*, in quanto possiamo avere un input in più, fornito dalla ricorrenza su sè stesso.

Queste reti funzionano bene con *sequenze*, ma non è una associazione $1-1$. Esistono reti convoluzionali usate per questi scopi, semplicemente si aspetta l'intero input dall'inizio. Un modello di rete neurale, chiamato *Transformers*, si è dimostrato migliore di molte reti ricorrenti.

Il problema che affrontiamo con le reti ricorrenti è, data una sequenza x_1,...x_{t-1}, ottenere $P(x_t|x_{t-1}, x_{t-2}, ..., x_1)$.

Una rete neurale dovrebbe immagazzinare tutte le osservazioni, e poi fare una predizione. L'output dipenderebbe da tutti, quindi abbiamo una complessità elevata!

Ciò che si fa è introdurre un **hidden state** (stato nascosto) h_{t-1} per catturare la sequenza di informazioni *fino* a $t-1$.

$P(x_t|x_{t-1}, x_{t-2}, ..., x_1) \approx P(x_t|h_{t-1})$, con : $h_t = f(x_t, h_{t-1})$

Esempio

Sequenza numeri di interi, per sapere il prossimo valore intero, basta memorizzare in h il valore precedente, e poi lo incrementiamo di 1. Qui la funzione $f(x)$ è facile, sommiamo 1, ma in altri casi non è detto sia così facile!

0.3 Costruzione di rete neurale ricorrente

In realtà non memorizziamo nulla, bensì la rete ad ogni istante si passa un valore prodotto all'istante precedente a supporto della predizione.

NB: prima parlavamo di *hidden layers*, nascosto perchè interno alla rete. L'output della rete non viene direttamente da questi layers. L'*hidden state* è qualche stato interno, non direttamente visibile come output della rete.

Nelle reti neurali ricorrenti abbiamo entrambi.

Supponiamo che, al tempo t, si estragga un valore dalla sequenza di input x_t. Ciò che viene prodotto come output dell'*hidden layer* è:

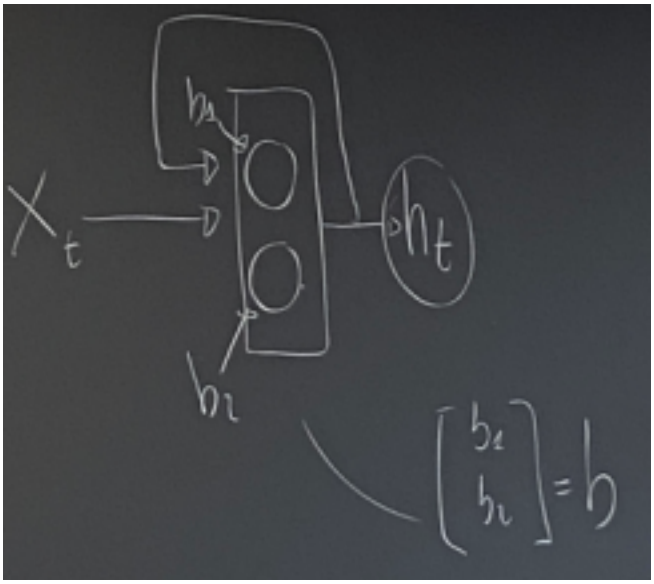
$h_t = \phi(W_x x_t + W_h h_{t-1} + b)$

Nelle reti precedenti, avevamo:

$h_t = \phi(W \cdot x + b)$

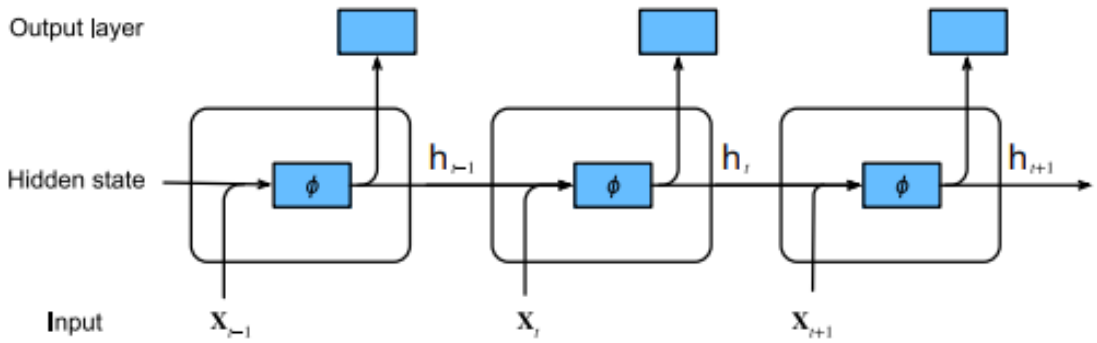
La differenza è che ora ci sono connessioni che tornano indietro, l'input che torna indietro è l'hidden state calcolato all'istante precedente. Quindi h_t deriva da h_{t-1} Anche qui si moltiplica per una matrice, ovvero W_h. Il calcolo è comunque analogo. Alle matrici posso associare nomi totalmente diversi.

Prendiamo un livello, che riceve in input un vettore x, di cardinalità pari al numero di feature, e tale livello produce h_t, con la possibilità di reintroduzione di input. Se ho due neuroni (ogni neurone ha coefficiente di bias), l'uscita dipenderà in numero da questo numero di neuroni. L'uscita del livello è l'hidden state. Lo scopo di tale livello, è cercare di catturare informazioni dall'hidden state, o comunque capire cosa rappresenta tale sequenza.



Perchè parliamo di *ricorrenza*? Perchè c'è dipendenza dall'hidden state dello stato precedente.

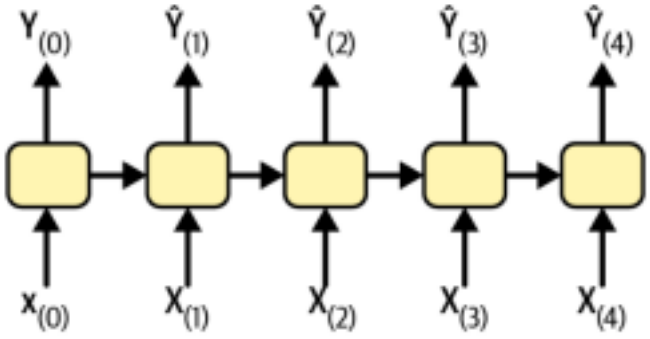
Per queste reti, $\phi()$ è *tanh* o *sigmoid*, quasi mai la *relu*.



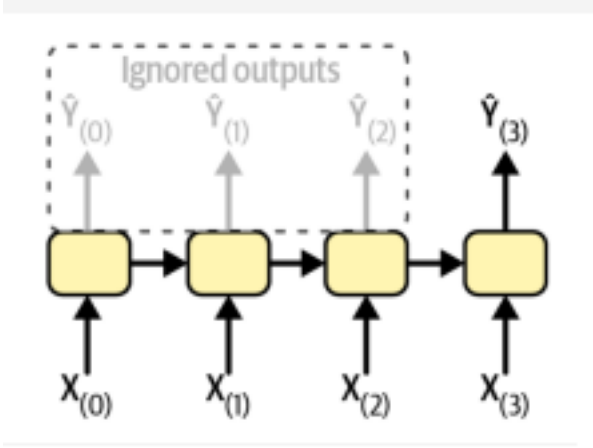
Ad ogni istante temporale, prendiamo un elemento della sequenza, in uscita avremo un'altra sequenza di output. Qui assumiamo la produzione di un nuovo *hidden state*, che riportiamo in input. Potrei però definire un'altra matrice di pesi, con funzionalità di output. Cioè potrei diversificare ciò che va in input da ciò che va in output.

0.4 Casi da analizzare

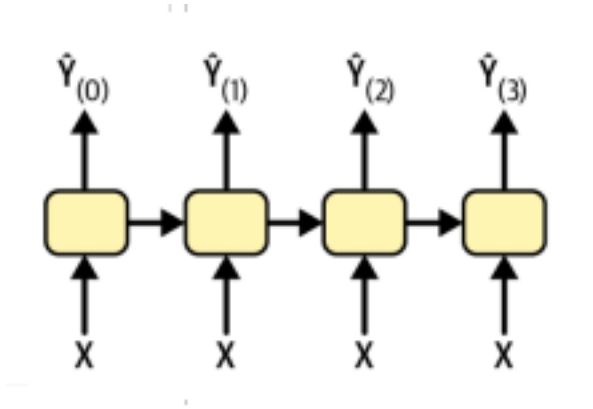
- **Sequence to sequence**: ne è un esempio l'analisi delle temperature, magari ora per ora. Entra sequenza di misure, esce sequenza di previsioni.



- **Sequence to vector**: ne è un esempio il dare una canzone e ottenerne il nome. Oppure, dato un disegno fatto sul tablet, identifiare l'oggetto. Potremmo affrontare il problema anche in modo *convoluzionale*, poichè è una immagine. Però, nel training, dovrei fornire immagini simili, e con disegni non è tanto facile. Con reti ricorrenti, potrei sfruttare i *movimenti* fatti per creare il disegno.



- **Vector to sequence**: ne è un esempio l'image captioning, come la descrizione di una immagine. Oppure fornisco delle note musicali, per formare una sequenza musicale. Fornisco lo stesso input X, ottengo *output diversi* che formerà una sequenza coerente. Funziona perchè c'è l'*hidden state*, quindi internamente, a parità di input, possiamo osservare cambiamenti.



0.5 Addestramento

Nelle reti *feed-forward* avevamo le due propagazioni.

- La *forward propagation* è facile, basta srotolare la rete nel tempo.
- La *backward propagation* è insidiosa, ed infatti si usa una variante detta **backpropagation through time**, in cui oltre agli archi, consideriamo anche il tempo. La complicazione è che il gradiente, rispetto ad ogni parametro, deve essere sommato su tutte le occorrenze del parametro.

0.6 Backpropagation through time BPTT

Ci troviamo nel seguente caso, dove facciamo qualche semplificazione:

- $h_t = W_x x_t + W_h h_{t-1}$
- $y_t = W_y h_t$
- Loss Function $L = \frac{1}{T} \sum_{t=1}^{T} \ell(t_t, y_t)$

Per ogni istante t, è facile calcolare: $\frac{\partial L}{\partial y_t} = \frac{1}{T}\frac{\partial \ell(t_t, y_t)}{\partial y_t}$

Navighiamo all'indietro sul grafo:

Ad ogni istante temporale, il termine della loss dipende dai pesi di W_y, necessari per computare h_t, quindi, per calcolare il gradiente W_y:

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}_y} = \sum_{t=1}^{T} \frac{\partial \mathcal{L}}{\partial \boldsymbol{y}_t}\frac{\partial \boldsymbol{y}_t}{\partial \boldsymbol{W}_y} = \frac{1}{T}\sum_{t=1}^{T} \frac{\partial \ell(\boldsymbol{t}_t, \boldsymbol{y}_t)}{\partial \boldsymbol{y}_t}\boldsymbol{h}_t^\top$$

Quindi abbiamo gradiente rispetto W_y, ora lo vogliamo rispetto h_t. Facciamolo solo per l'ultimo istante temporale T.

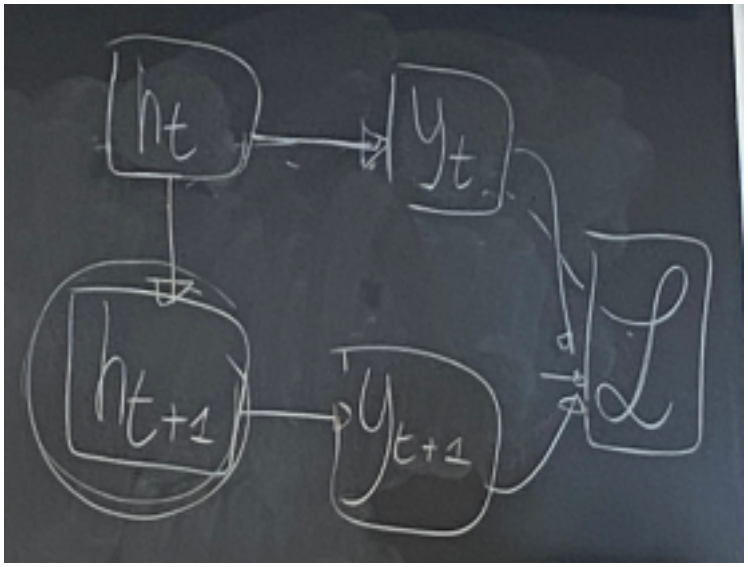
$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{h}_T} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{y}_T}\frac{\partial \boldsymbol{y}_T}{\partial \boldsymbol{h}_T} = \boldsymbol{W}_y^\top \frac{\partial \mathcal{L}}{\partial \boldsymbol{y}_T}$$

Ora, lo abbiamo fatto solo per un istante temporale, quello finale. Per uno generico, in cui $t < T$?

Le cose sono più complicate: La figura di riferimento, nel caso **base** è:

Da *hidden state* ho uscita, che sarà termine di funzione di *loss*.

Ora, questo h_t viene **anche** usato per h_{t+1} e quindi y_{t+1}, che contribuisce anch'egli. Abbiamo due percorsi per la *loss*. Questo si ripeterà iterativamente, il che porterà ad una sommatoria.



Se applichiamo la chain rule, abbiamo quindi due elementi da considerare, e quindi da sommare.

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{h}_t} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{y}_t}\frac{\partial \boldsymbol{y}_t}{\partial \boldsymbol{h}_t} + \frac{\partial \mathcal{L}}{\partial \boldsymbol{h}_{t+1}}\frac{\partial \boldsymbol{h}_{t+1}}{\partial \boldsymbol{h}_t}$$

Il primo termine sappiamo calcolarlo, il secondo termine (rispetto h_{t+1}) lo lasciamo in standby, perchè ciò che faremo è svilupparlo fino all'origine, e quindi otteniamo una sommatoria di questi elementi.

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial \boldsymbol{h}_t} &= \frac{\partial \mathcal{L}}{\partial \boldsymbol{y}_t}\frac{\partial \boldsymbol{y}_t}{\partial \boldsymbol{h}_t} + \frac{\partial \mathcal{L}}{\partial \boldsymbol{h}_{t+1}}\frac{\partial \boldsymbol{h}_{t+1}}{\partial \boldsymbol{h}_t} = \boldsymbol{W}_y^\top \frac{\partial \mathcal{L}}{\partial \boldsymbol{y}_t} + \boldsymbol{W}_h^\top \frac{\partial \mathcal{L}}{\partial \boldsymbol{h}_{t+1}} = \\ &= \boldsymbol{W}_y^\top \frac{\partial \mathcal{L}}{\partial \boldsymbol{y}_t} + \boldsymbol{W}_h^\top \left(\boldsymbol{W}_y^\top \frac{\partial \mathcal{L}}{\partial \boldsymbol{y}_{t+1}} + \boldsymbol{W}_h^\top \frac{\partial \mathcal{L}}{\partial \boldsymbol{h}_{t+2}} \right) = \\ &= \sum_{i=0}^{T-t} \left(\boldsymbol{W}_h^\top \right)^i \boldsymbol{W}_y^\top \frac{\partial \mathcal{L}}{\partial \boldsymbol{y}_{t+i}}\end{aligned}$$

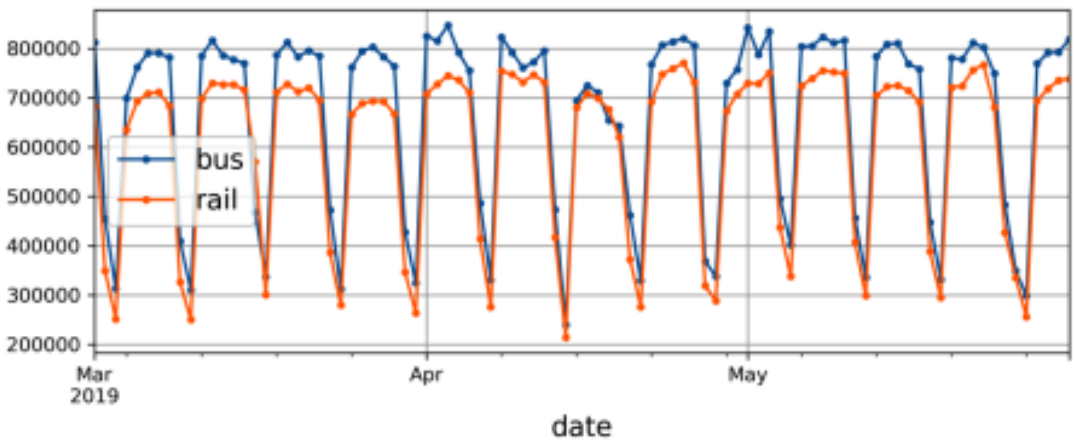
Questa è una sequenza di calcoli per il gradiente di L rispetto a h_t. Si può notare che per sequenze lunghe è necessario calcolare potenze elevate di W_h (gli autovalori minori di 1 possono annullarsi e quelli maggiori di 1 possono divergere!). In particolare:

- Se i valori *esplodono*, si usa una *soglia*, se eccede, non lo faccio crescere ulteriormente.

A parte queste difficoltà, possiamo finalmente trovare:

0.7 Esempio

Usiamo i gradienti come sempre, SDG o Adam ad esempio. Vogliamo predire il numero di passeggeri su un autobus il giorno successivo. Ovviamente la prima fase è sempre l'analisi dei dati, in cui vediamo che l'andamento si ripete.



Un primo approccio è **Naive Forecasting**, quindi mi baso sui valori delle settimane precedenti. L'errore è minore del 10%. Cercheremo di migliorare questi risultati. In particolare, faremo confronti anche con il modello **Sarima**, usato per il *time series forecasting*.

0.7.1 Come creo training set per questo tipo di problema?

Possiamo prendere 3 anni di misure, darle alla rete e predire il valore successivo. Però come uso questi dati? Alla rete servono tanti esempi. Potrei *spezzare* il dataset, ad esempio prendendo finestre di osservazioni ogni 2 mesi, e prendere un valore target per valutare l'accuratezza della rete. Per ogni finestra, si crea un input x_i, e poi lo confronto col valore vero presente nella riga successiva del dataset t_1 (quindi cosa accade il giorno successivo ai due mesi). Questo lo faccio dall'inizio alla fine del dataset, producendo tanti input e tanti valori target.

Non dobbiamo fare noi questo lavoro. Creo un dataset iniziale con

`raw_dataset = tf.data.Dataset.range(6)`, con 6 pari al numero delle finestre.

Poi, con `raw_dataset.window(4,shift=1,drop_remaining=true)` creo finestre di dimensione 4, che scorre di 1 elemento alla volta.

Questo sarà alla base del nostro notebook. Alcuni passi nel notebook: Definiamo `fit_and_evaluate` da usare su reti diverse per l'addestramento e la valutazione, con `early_stopping`. Il modello passato viene compilato, con `loss=Huber`. In keras, per definire una rete ricorrente uso una rete sequenziale con livello nascosto `SimpleRNN` e livello `dense` di una unità. Dopo 100 epoche, otteniamo un errore quadratico medio del 6%.