Reti ricorrenti

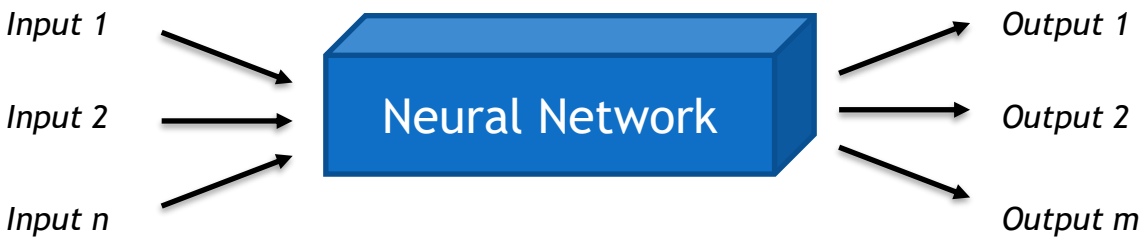
Reti ricorrenti

Premessa

Molte delle architetture di Machine Learning / Deep Learning realizzate funzionano in una modalità «one-shot»:

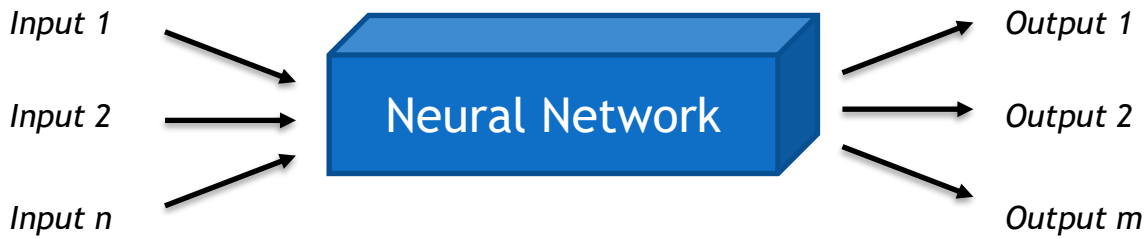
- Ricevono uno o più input.
- Forniscono uno o più output.

Il **contesto** richiesto dalla rete è solamente quello legato agli input.



Reti ricorrenti

Premessa



Queste architetture di rete «one-shot» possono essere divise in due gruppi che ne raccolgono i principali casi d'uso:

- **ANN, Artificial Neural Network** : *classificazione e regressione*.
- **CNN, Convolution Neural Network** : *computer vision*.

Reti ricorrenti

Premessa

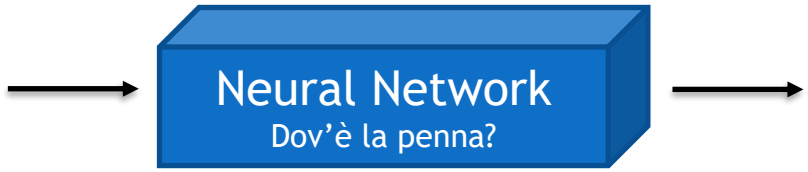
Vi sono molteplici situazioni, però, nelle quali il **contesto** che descrive l'input e dal quale ne dipenderà l'output non è «one-shot» ma:

Una sequenza di dati

Il **contesto** in questo caso:

- Permane per più di un singolo dato in input.
- Rappresenta una finestra temporale più o meno grande.
- Mette in relazione dati temporalmente distanti fra loro.

Ho comprato un tavolo. Nel tavolo appoggio molte cose. Oggi ho appoggiato una penna.



La penna è sul tavolo.

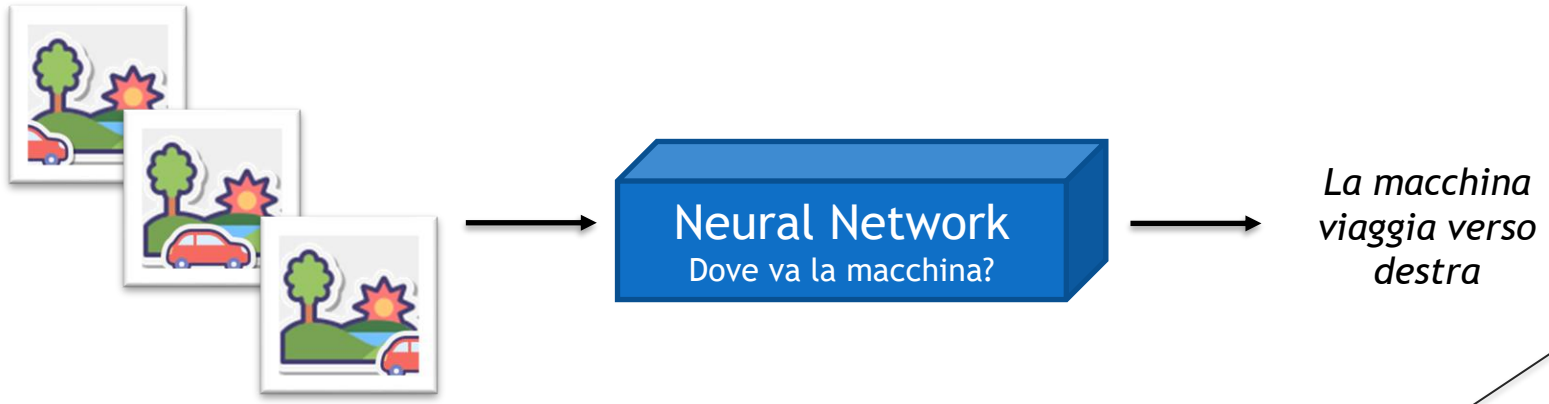
Reti ricorrenti

Premessa

Il **contesto**, per queste architetture, può assumere diverse forme e non si limita alle sole sequenze di parole.

È **contesto**:

- Una sequenza di frasi, di parole, di caratteri.
- Una sequenza di numeri.
- Una sequenza di immagini.

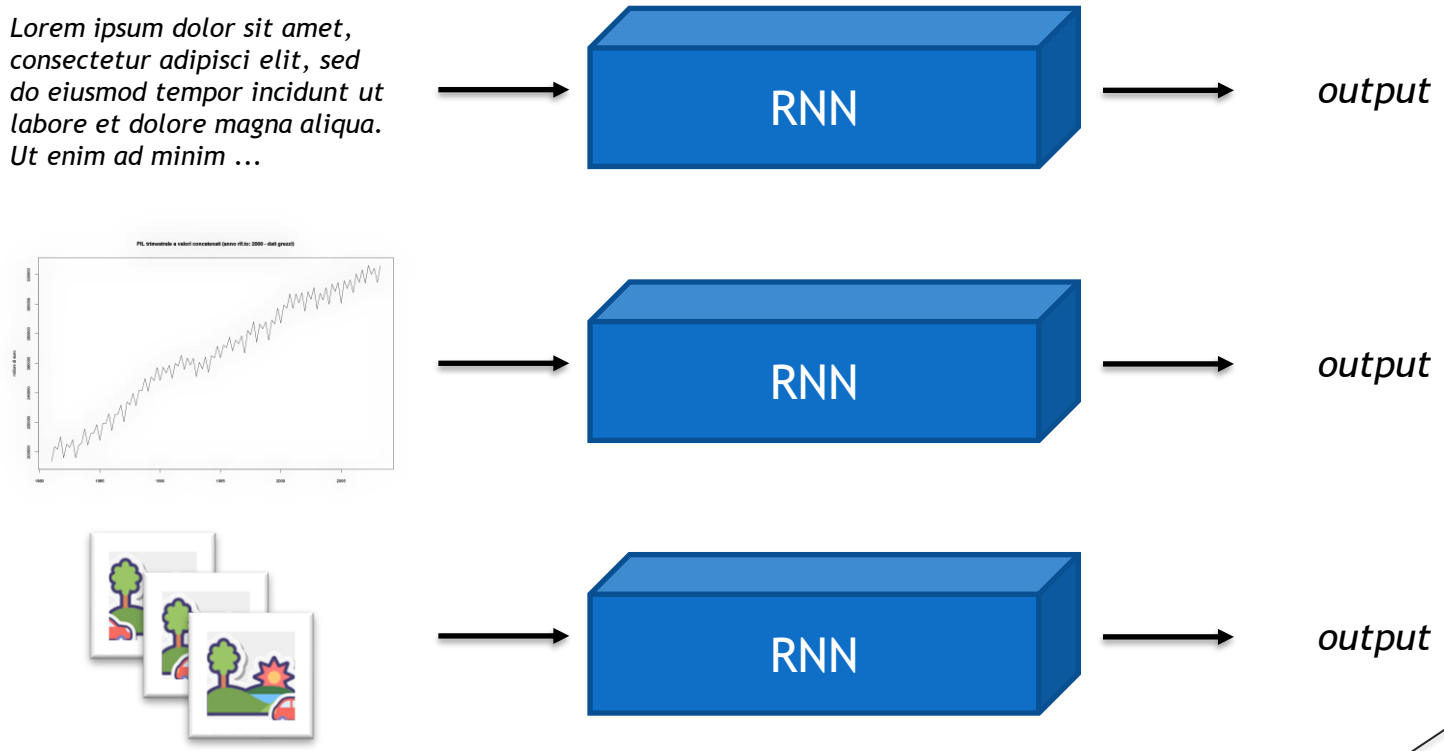


Reti ricorrenti

Premessa

È **contesto:**

Una sequenza di frasi, di parole… *Una sequenza di numeri.* *Una sequenza di immagini.*



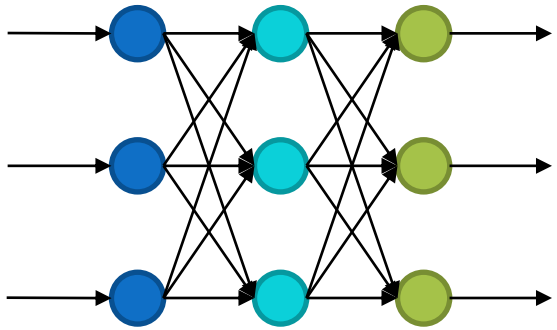
Reti ricorrenti

Definizione

Le reti ricorrenti, **Recurrent Neural Network (RNN)**, sono architetture di machine learning capaci di gestire *sequenze di dati in ingresso*.

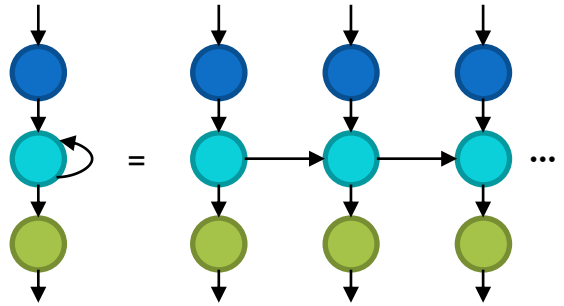
In contrapposizione con le architetture finora descritte, si distinguono quindi:

Reti feed-forward



L'input viaggia in una direzione a senso unico verso l'output

Reti recurrent



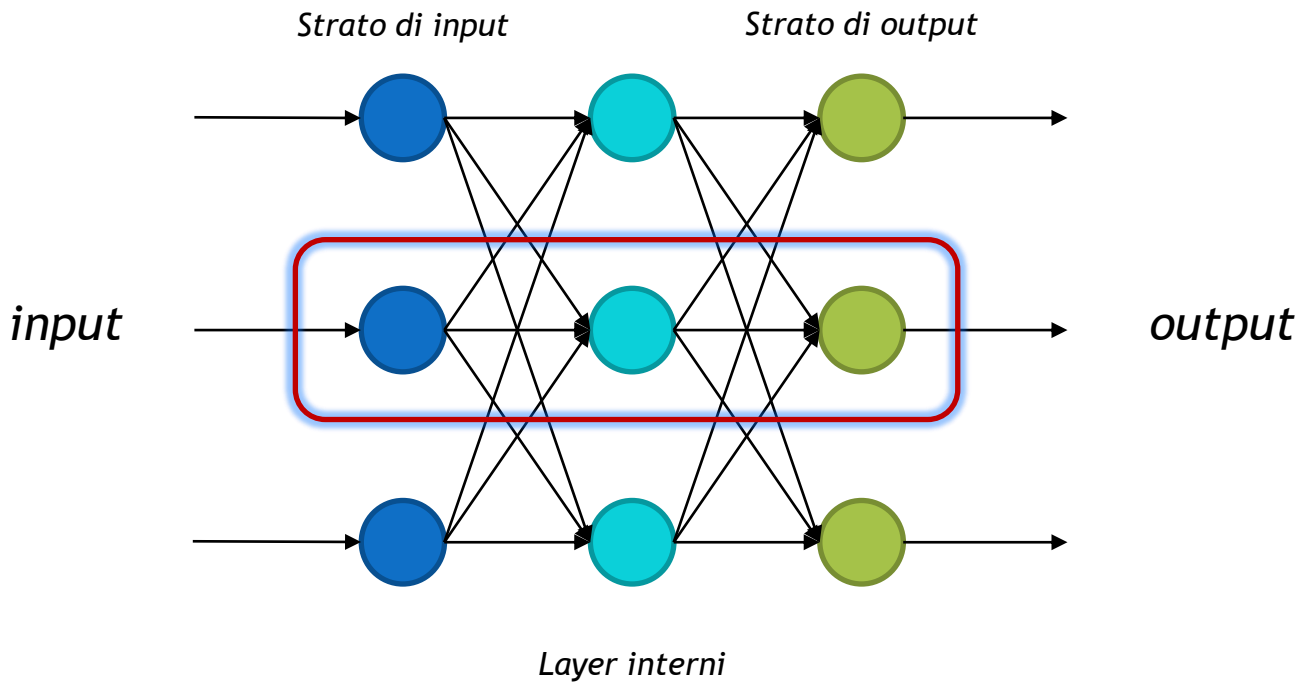
Nella rete ci sono loop dove le informazioni ritornano in circolo dai layer interni o dagli output stessi.

Reti ricorrenti

Definizione

Volendo descrivere idealmente la creazione di una rete ricorrente:

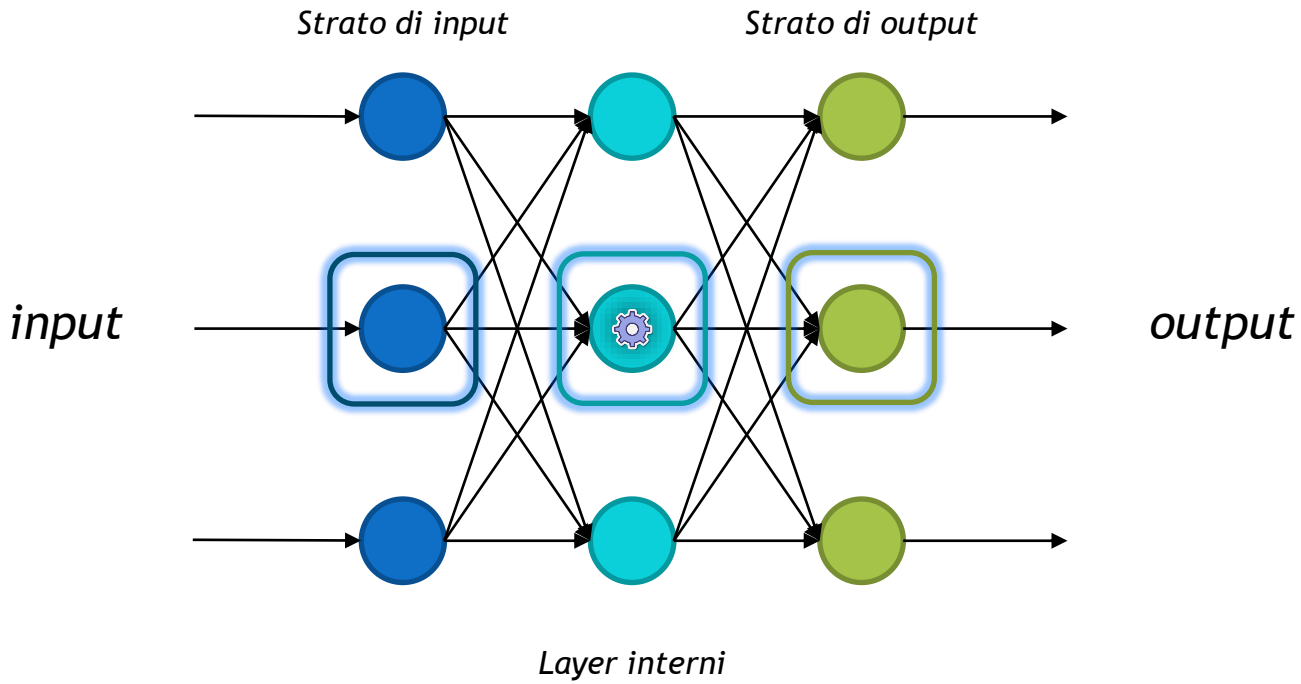
- Si potrebbe partire da una rete *feed-forward* dalla quale identifichiamo i componenti principali che servono a mappare l'input nell'output.



Reti ricorrenti

Definizione

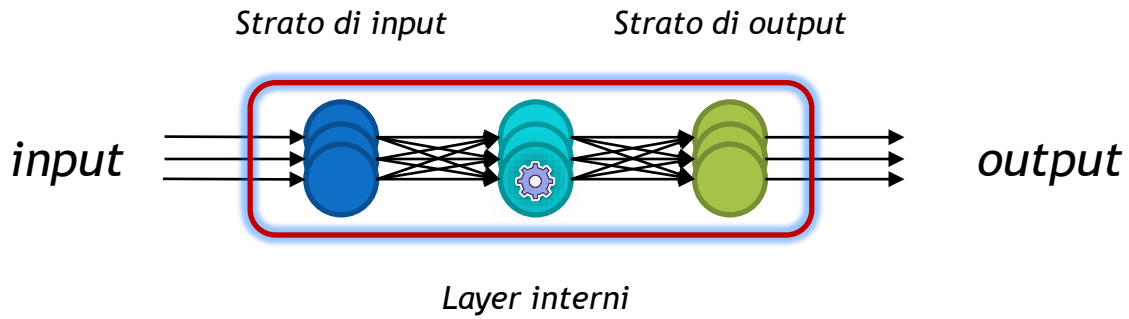
- L'*input* diventa *output* passando attraverso una serie di trasformazioni mascherate/incapsulate nel *nodo centrale*.



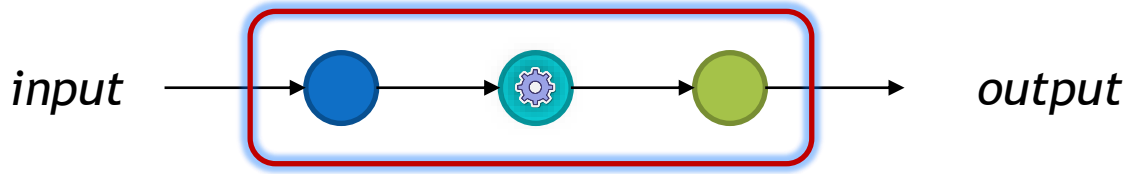
Reti ricorrenti

Definizione

- Si collassa la rete in un'unica sequenza: da input, le trasformazioni scelte generano l'output.



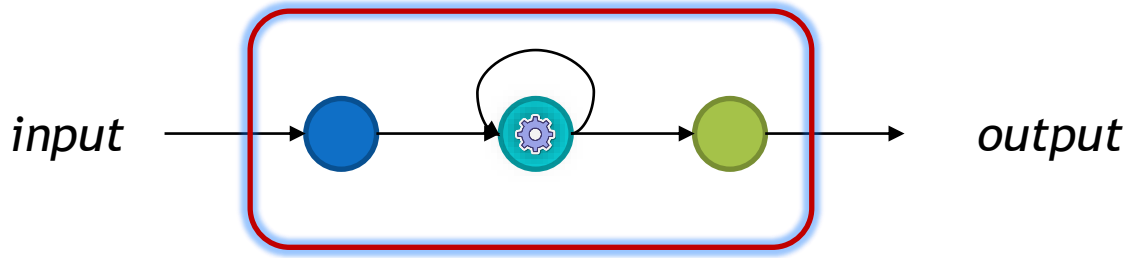
- Nella struttura risultante, la parte centrale di trasformazione è identificata come **recurrent cell**.



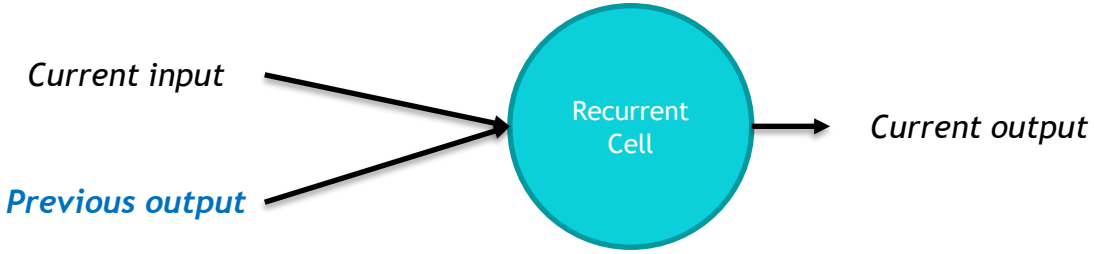
Reti ricorrenti

Definizione

- Alla **recurrent cell** si applica quindi un loop temporale.



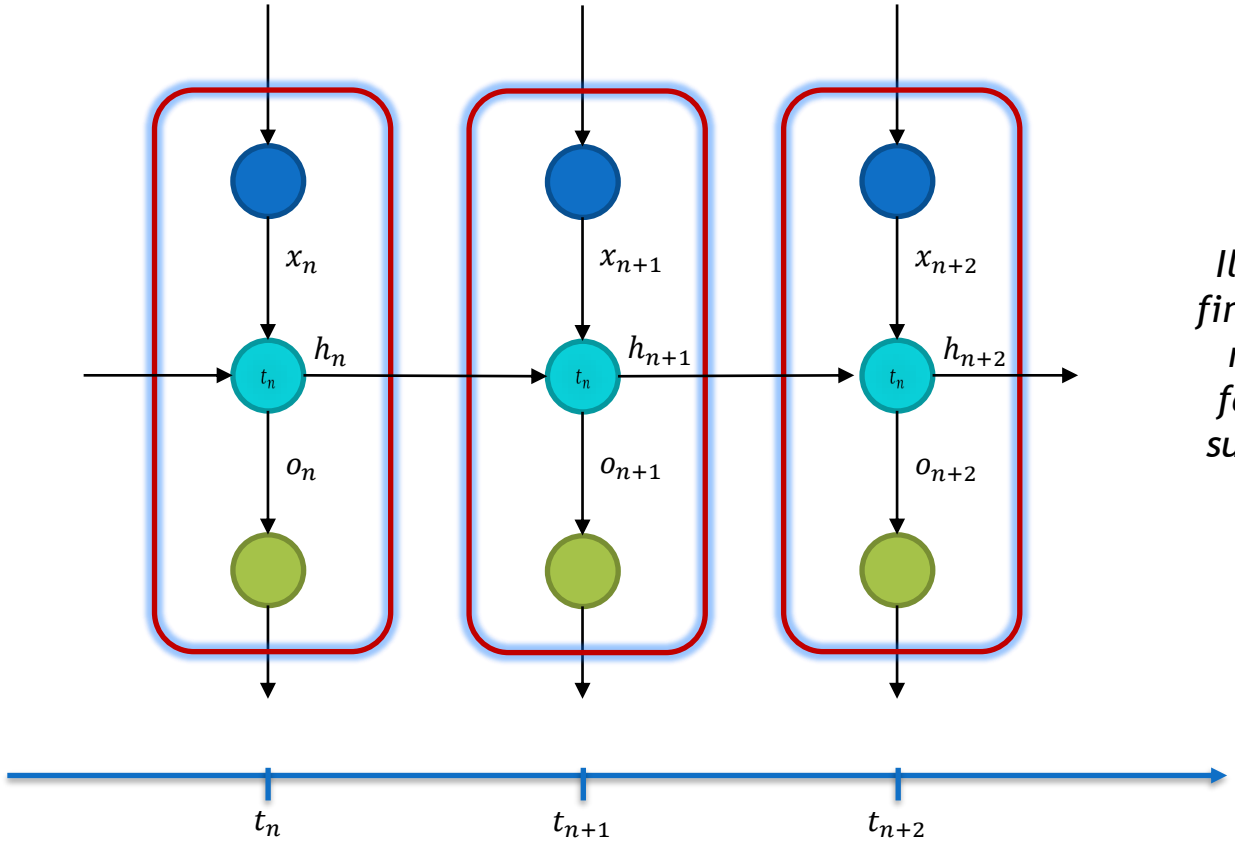
Con questo loop si indica il fatto che l'output generato dalla trasformazione **non** viaggerà solamente **avanti in maniera feed-forward** ma sarà anche reimmesso nella cella come input.



Reti ricorrenti

Definizione

- Espandendo questa struttura temporalmente, si otterrà una replica sequenziale della stessa **recurrent cell**.



Il loop temporale si rompe e finisce per mostrarsi come una rete dove l'apprendimento fatto da uno strato influisce sulla storia di apprendimento che avranno i successivi.

Reti ricorrenti

In sintesi

A contraddistinguere, quindi, le reti ricorrenti dalle classiche reti feed-forward vi sono due caratteristiche:

Architettura a loop

Le reti feed-forward possono essere rappresentate da un **grafo diretto aciclico**.

Le reti ricorrenti formano invece **grafi ciclici** dove le informazioni ritornano in circolo.

Creazione di memoria

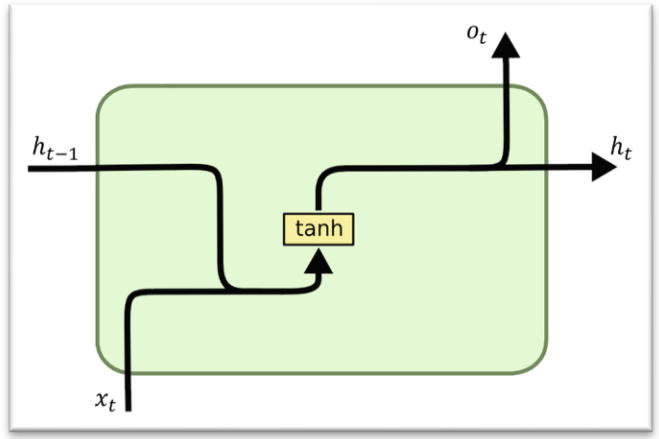
La creazione di loop crea **persistenza dei dati**.

La persistenza dei dati permette di realizzare un concetto di memoria, definendo un contesto temporale.

L'aggiornamento della rete può quindi dipendere da uno storico.

Reti ricorrenti

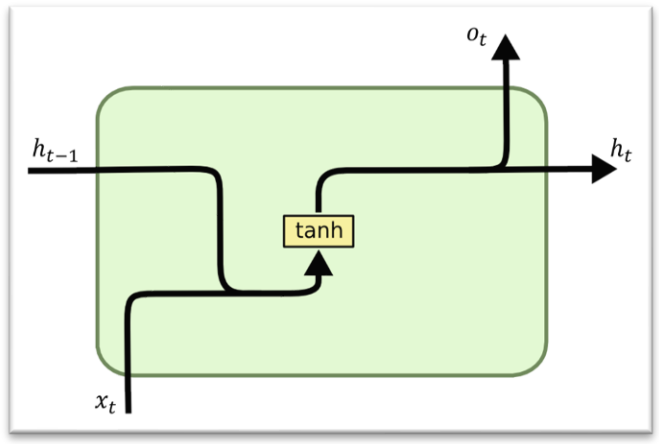
RNN cell



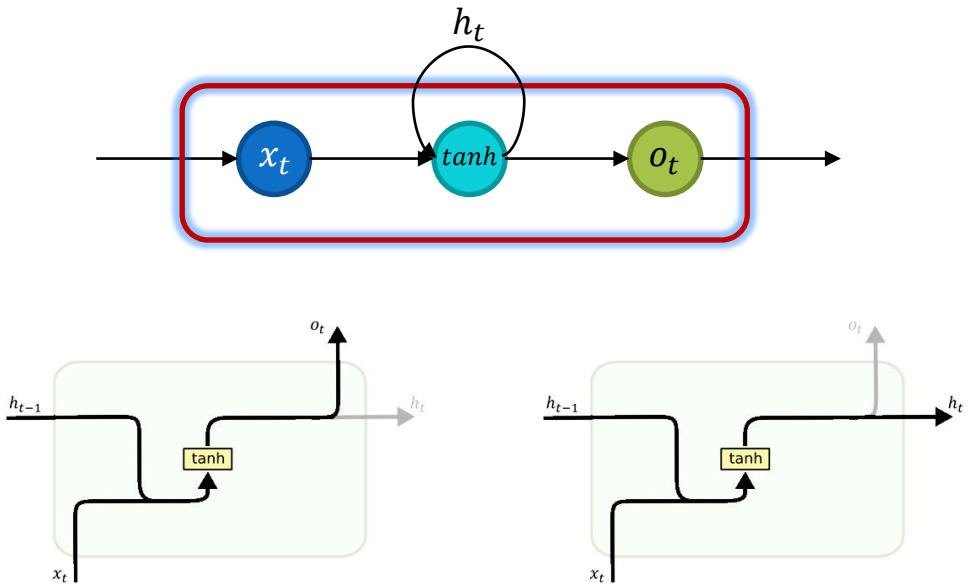
- È la struttura più semplice di cella temporale ricorrente.
- Alla cella successiva è inviata una informazione codificata sotto forma di **hidden state**.
- La memoria è codificata a partire da una combinazione fra l'input allo step corrente è l'**hidden state** della cella precedente.

Reti ricorrenti

RNN cell



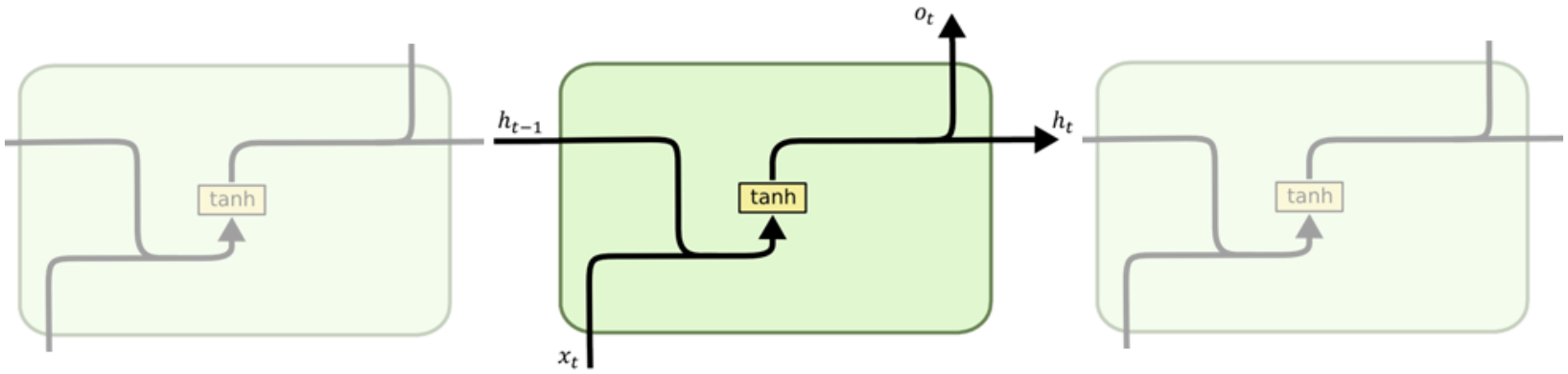
- È semplice notare il parallelo fra questa struttura e la versione astratta precedentemente descritta.



Reti ricorrenti

RNN cell: in sintesi

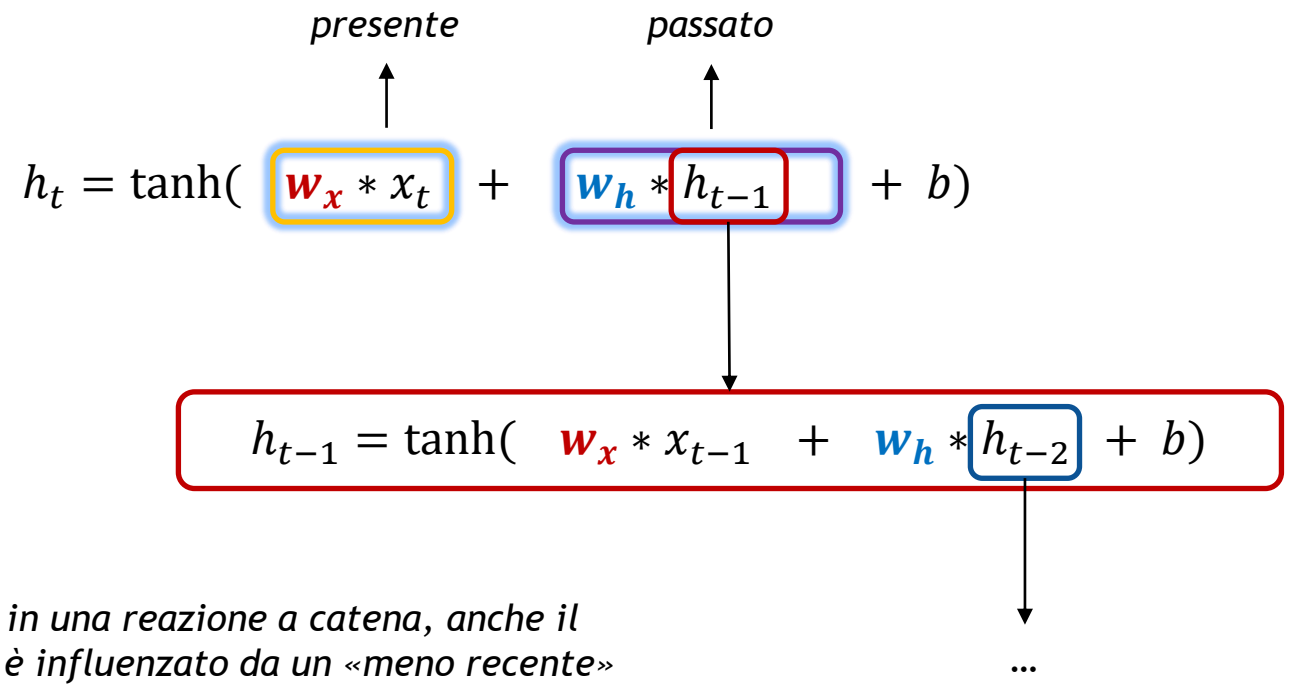
- Il loop temporale si identifica in una sequenza di repliche della stessa cella.
- Il numero di repliche rispecchia il numero di elementi della sequenza di input.
- L'output combina l'input attuale della sequenza e l'output precedente.
- L'output è input della cella successiva.



Reti ricorrenti

RNN cell: hidden state

L'**hidden state** della cella è descritto dalla seguente relazione che lega il presente con la storia passata:

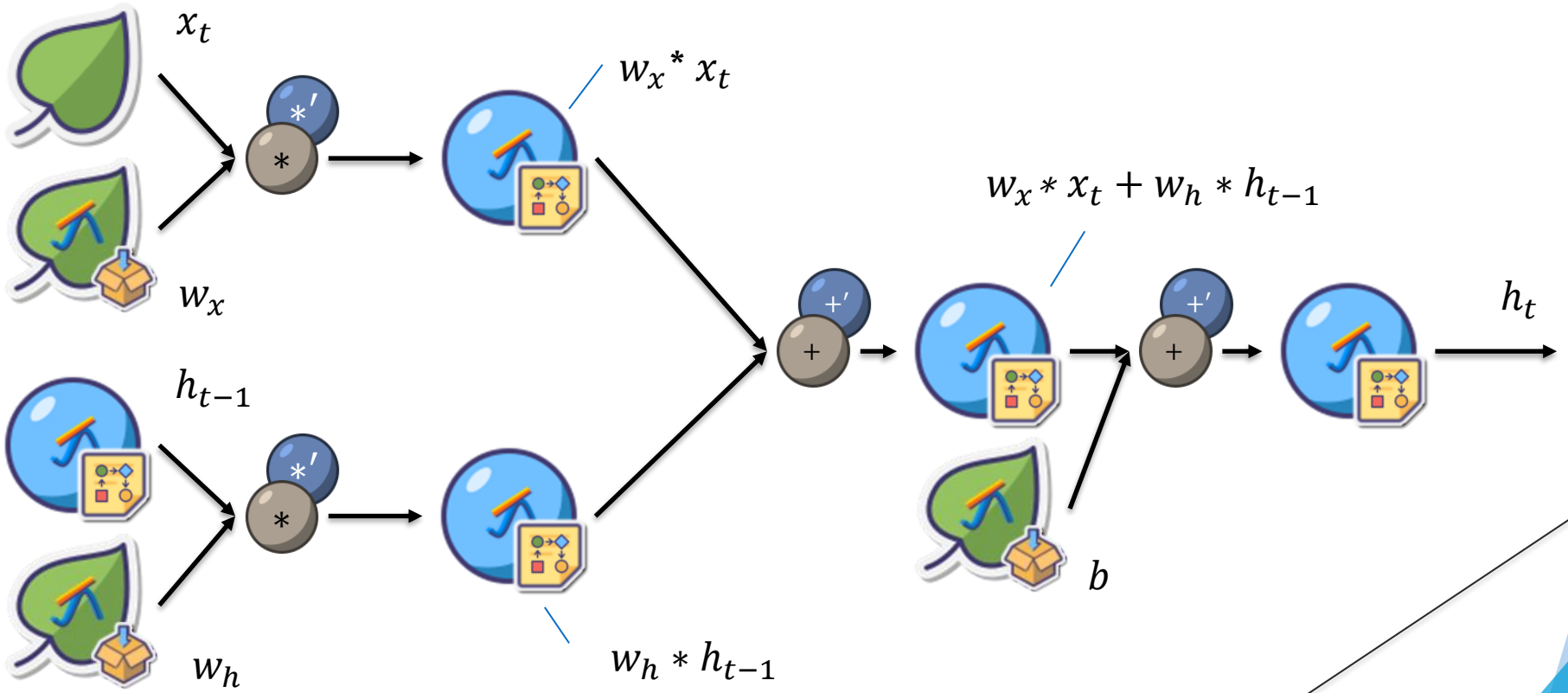


Come in una reazione a catena, anche il passato è influenzato da un «meno recente» passato.

Reti ricorrenti

RNN cell: grafo computazionale

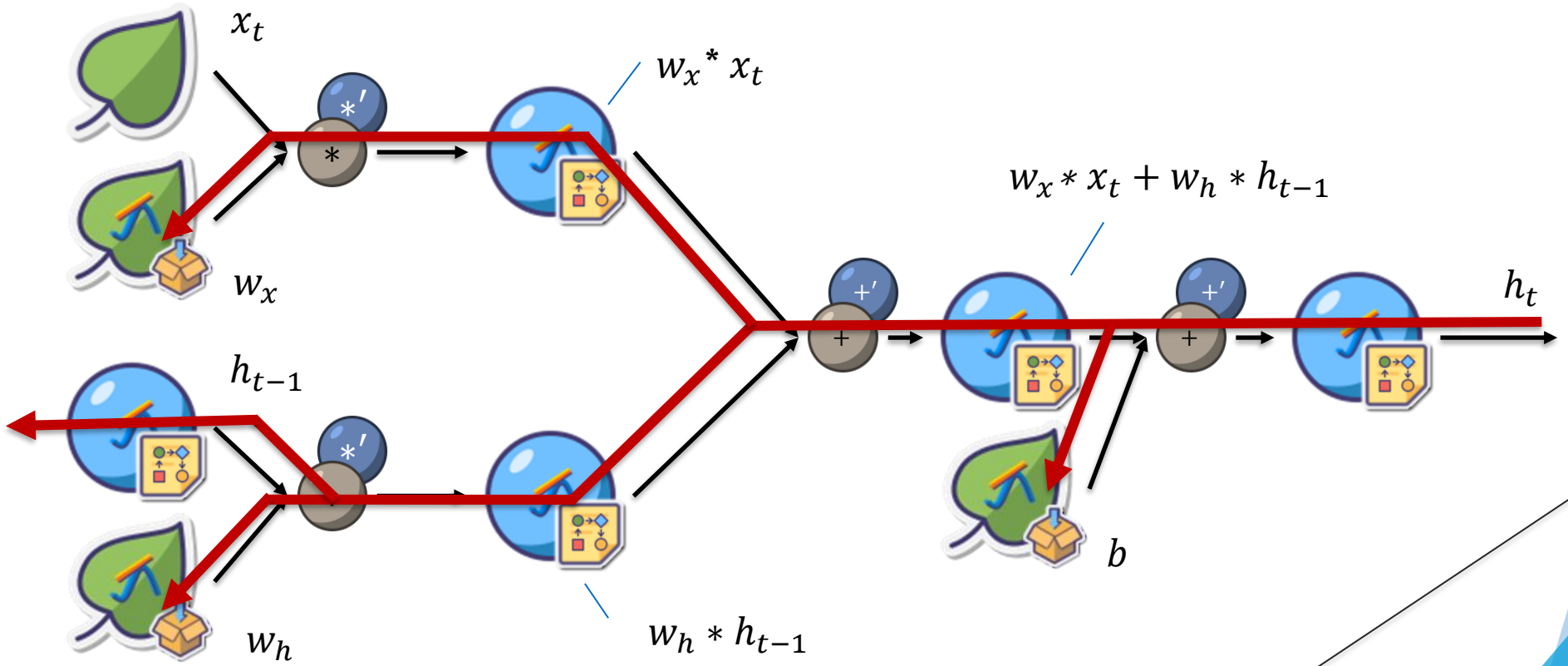
Una possibile rappresentazione del grafo computazione che si produrrebbe in PyTorch, avrebbe idealmente questa struttura:



Reti ricorrenti

RNN cell: BPTT

L'aggiornamento dei parametri in una struttura temporale come quella delle reti ricorrenti prende il nome di **Back-Propagation Through Time**, **BPTT**.



Reti ricorrenti

RNN cell: Vanishing gradient

Quello che dal grafo viene messo in luce è come esista un passaggio della back-propagation che si propaga «nel passato»: in questo caso attraverso il nodo che identifica h_{t-1}.

*La **loss** calcolata per un particolare punto temporale mostrerà il contributo **anche** delle celle temporali passate.*

Questo legame moltiplicativo che si va a creare fra i gradienti nei diversi istanti temporali potrà innescare nella rete due situazioni note:

- ***Vanishing gradient*** : moltiplicazioni di valori piccoli fino a svanire.
- ***Exploding gradient*** : moltiplicazioni di valori grandi fino a divergere.

Reti ricorrenti

RNN cell: Vanishing gradient

Si immagini di avere una catena di nodi legati fra loro da una semplice moltiplicazione con uno scalare.

Questa relazione, molto più semplice di quella fra **recurrent cell**, mette in luce come il parametro **condiviso** W possa portare, per n crescente a far esplodere o annullare $\boldsymbol{x_n}$.

$$\boldsymbol{x_n} = x_{n-1} * W = x_{n-2} * W * W$$
$$= \cdots = x_0 \boldsymbol{W^n}$$

- Per valori di W superiori a 1, $\boldsymbol{x_n}$ tenderà a divergere ad ∞.
- Per valori di W inferiori a 1, $\boldsymbol{x_n}$ tenderà a 0.

Lo stesso vale per il gradiente

Reti ricorrenti

Architetture: RNN cell

Dall'architettura di **recurrent cell** descritta, si possono quindi identificare vantaggi e svantaggi:

Vantaggi

La possibilità di accettare e gestire input di dimensione variabile.

La capacità di estrarre pattern e informazioni da serie temporali di dati.

Svantaggi

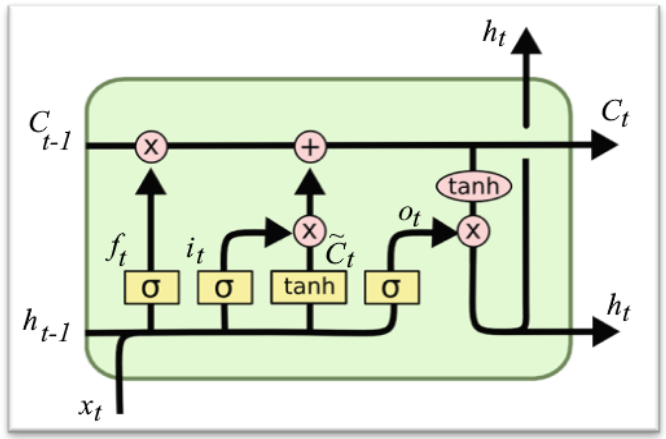
Vanishing gradient può portare ad update dei parametri poco influenti.

Exploding gradient influenza negativamente la convergenza dell'apprendimento.

RNN «ricorda» informazioni recenti ma tende a considerare meno le informazioni mano a mano che si torna indietro nella serie temporale. Presenta quindi una memoria prettamente **short-term** e non **long-term**.

Reti ricorrenti

LSTM cell



- LSTM nasce per superare i problemi quali delle semplici *rnn cell*: exploding gradient, vanishing gradient e short-memory…
- Alla cella successiva trasferisce due informazioni codificate sotto forma di **hidden state** e **cell state**.
- Ogni cella decide autonomamente come gestire la memoria tramite un sistema a **gate**.
- La miglior gestione della memoria si rispecchia in una struttura visivamente più complessa.

Reti ricorrenti

LSTM cell: Hidden e Cell State

Rispettivamente:

Hidden State

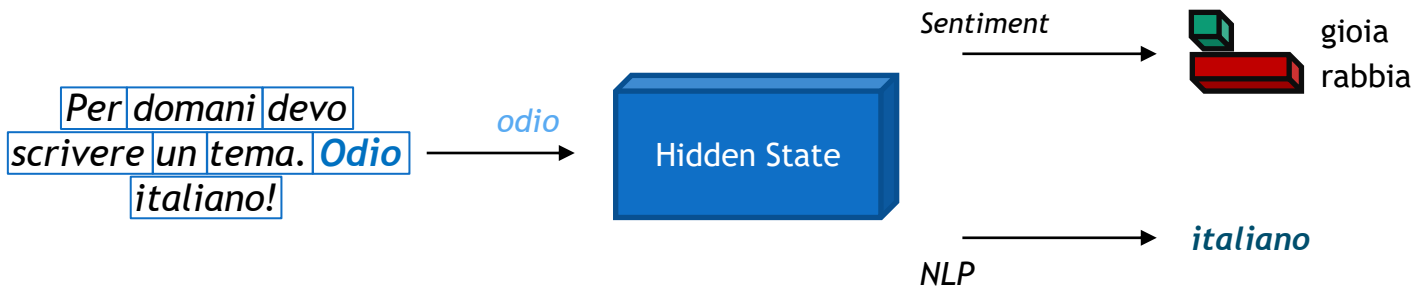
Mette in relazione l'input della sequenza corrente con il passato e crea una codifica di questa nuova informazione.

La codifica ottenuta **rispecchia l'addestramento effettuato e l'obbiettivo** da raggiungere.

Cell State

Rappresenta globalmente come mettere in relazione la storia passata con il presente.

Codifica pattern fra gli input della sequenza e identifica chi maggiormente fornisce contributo alle decisioni correnti.



Reti ricorrenti

LSTM cell: Hidden e Cell State

Rispettivamente:

Hidden State

Mette in relazione l'input della sequenza corrente con il passato e crea una codifica di questa nuova informazione.

La codifica ottenuta rispecchia l'addestramento effettuato e l'obbiettivo da raggiungere.

Cell State

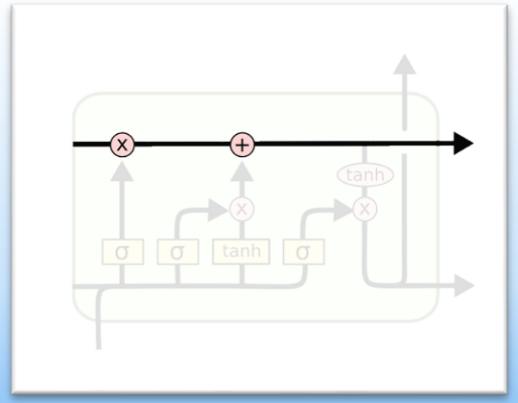
Rappresenta globalmente come mettere in relazione la storia passata con il presente.

Codifica pattern fra gli input della sequenza e identifica chi maggiormente fornisce contributo alle decisioni correnti.

Reti ricorrenti

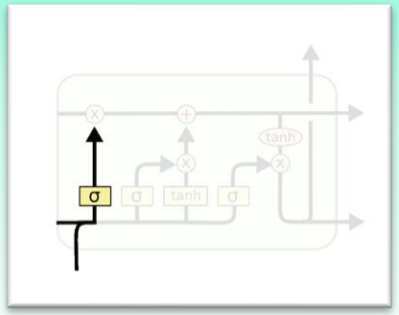
LSTM cell

In una cella LSTM si identificano in particolare:

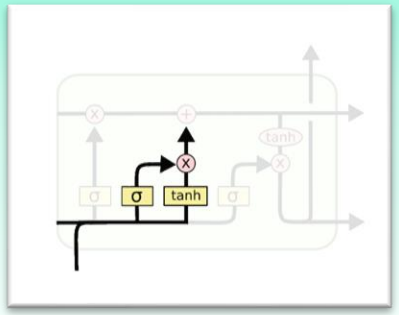


cell state

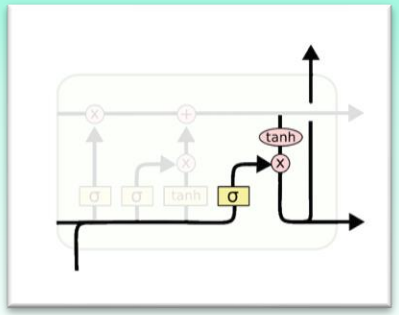
hidden state



forget gate



input gate

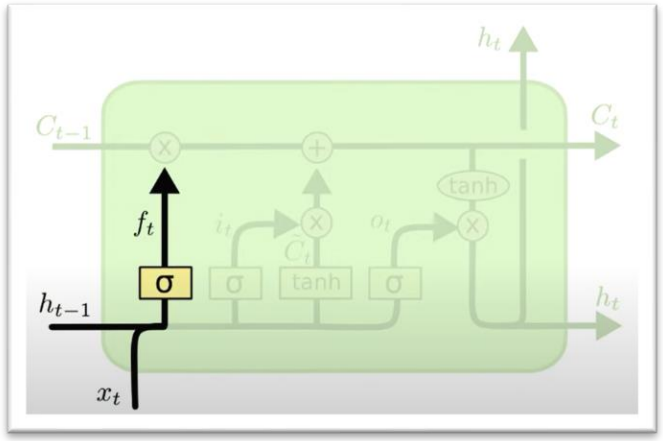


output gate

Reti ricorrenti

LSTM cell

In una cella LSTM si identificano in particolare:



$$f_t = \sigma(w_x^{(f)} x_t + w_h^{(f)} h_{t-1} + b_f)$$

forget gate

Decide se e quanto mantenere della memoria passata, identificata dal precedente cell state.

Il gate combina l'input allo step corrente con il precedente **hidden state**.

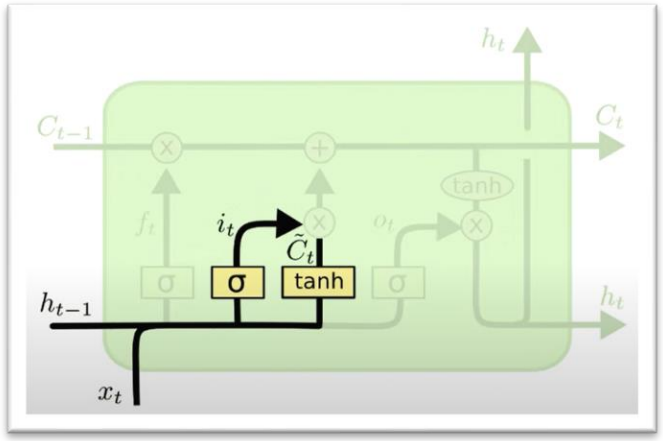
Vi è poi applicata una sigmoide e l'output viene moltiplicato al **cell state** fornito dalla cella passata.

I valori in [0, 1] descrivono al meglio il concetto di '*prendere o lasciare*' applicato al passato.

Reti ricorrenti

LSTM cell

In una cella LSTM si identificano in particolare:



$$i_t = \sigma(w_x^{(i)} x_t + w_h^{(i)} h_{t-1} + b_i)$$

$$\tilde{C}_t = tanh(w_x^{(c)} x_t + w_h^{(c)} h_{t-1} + b_c)$$

input gate

Decide se aggiornare il cell state con la storia corrente.

Il gate combina input corrente e precedente **hidden state** e vi applica una sigmoide.

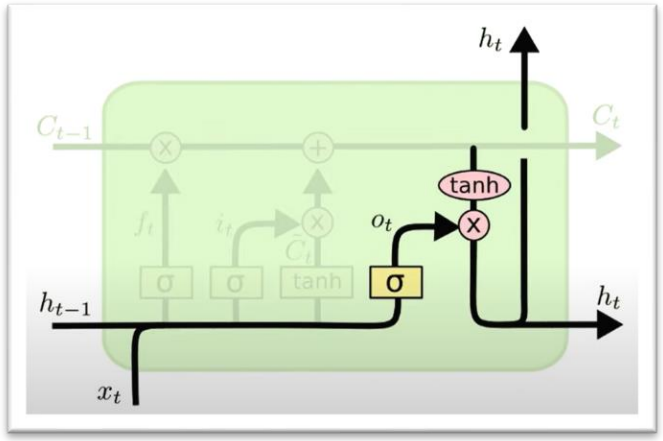
La storia corrente è combinazione fra input e precedente **hidden state**. Passa attraverso il gate dopo una normalizzazione tramite un'operazione di **tanh**.

L'output, centrato in 0 e simmetrico in [-1, 1] permette di regolarizzare la gestione dei gradienti.

Reti ricorrenti

LSTM cell

In una cella LSTM si identificano in particolare:



$$o_t = \sigma(w_x^{(o)} x_t + w_h^{(o)} h_{t-1} + b_o)$$

$$h_t = tanh(C_t)\ o_t$$

output gate

Decide il quantitativo di informazioni da esporre in uscita e alle successive celle.

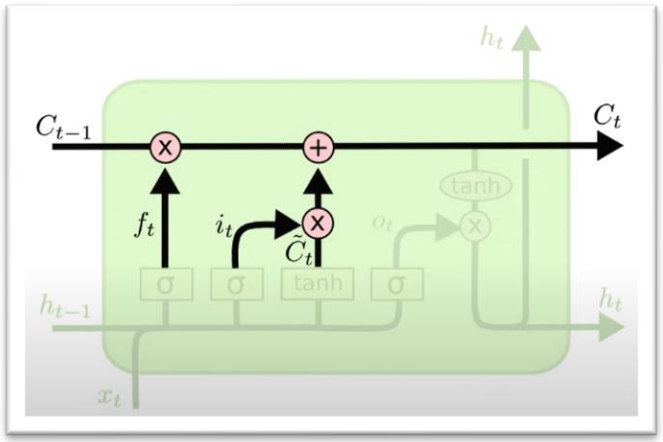
Il gate combina l'input allo step corrente e il precedente **hidden state** per applicarvi poi una sigmoide, fornendo valori in [0, 1].

Attraverso il gate viene trasferita una versione del **cell state** corrente normalizzato da un'operazione di **tanh**.

Reti ricorrenti

LSTM cell

In una cella LSTM si identificano in particolare:



$$C_t = C_{t-1} f_t + \tilde{C}_t i_t$$

cell state

Trasferisce alla cella successiva una informazione codificata della memoria.

Il **cell state** corrente è calcolato come combinazione fra precedente hidden state e input allo step attuale.

Il **cell state** corrente si combina a quello passato.

La quantità di informazione presa dalla storia passata è modulata dal *forget gate*.

La quantità di informazione presa della storia corrente è modulata dall'*input gate*.

Reti ricorrenti

LSTM cell

Dall'architettura di **LSTM cell** descritta, si possono quindi identificare vantaggi e svantaggi:

Vantaggi

La possibilità di accettare e gestire input di dimensione variabile.

La capacità di estrarre pattern e informazioni da serie temporali di dati.

Una gestione della memoria a 'lungo termine' rispetto una semplice **rnn cell**.

Svantaggi

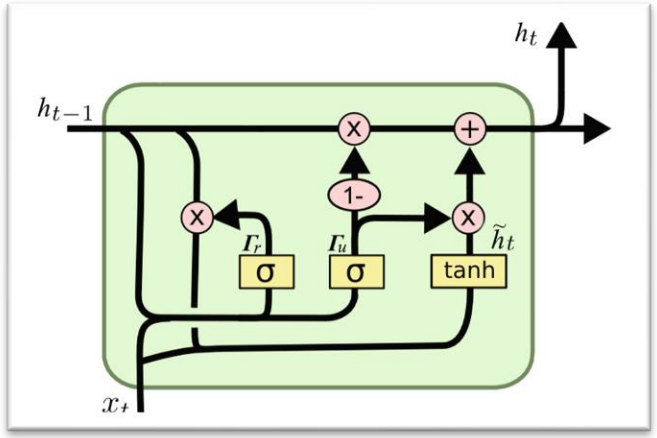
La struttura di queste celle è più complessa.

La complessità risulta in un calo di performance in termini computazionali

La complessità risulta in una possibile riduzione della velocità di addestramento.

Reti ricorrenti

GRU cell

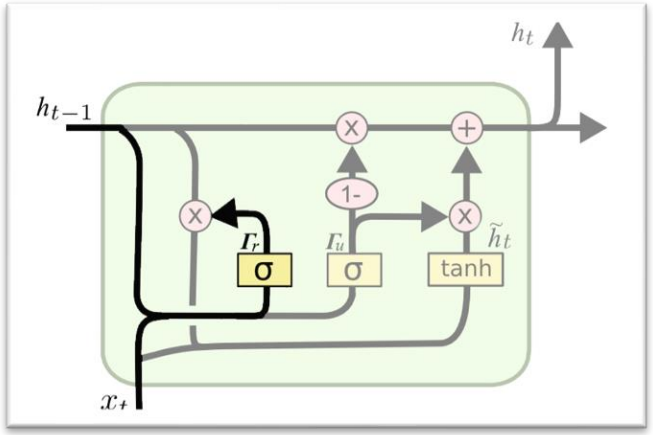


- GRU è una cella simile ad LSTM ma ne semplifica la struttura al fine di ridurne il peso computazionale e migliorare le prestazioni.
- È assente il concetto di **cell state** e l'informazione trasferita alle celle successive è codificata sotto forma di **hidden state**.
- Utilizza un meccanismo a *gate* per la gestione della memoria ma il numero di gate è inferiore rispetto ad una cella LSTM.

Reti ricorrenti

GRU cell

In una cella GRU si identificano in particolare:



$$\Gamma_r = \sigma(w_x^{(r)} x_t + w_h^{(r)} h_{t-1} + b_r)$$

reset gate

Modula la quantità di informazioni che vengono attinte dalla storia passata per rappresentare il prossimo hidden state.

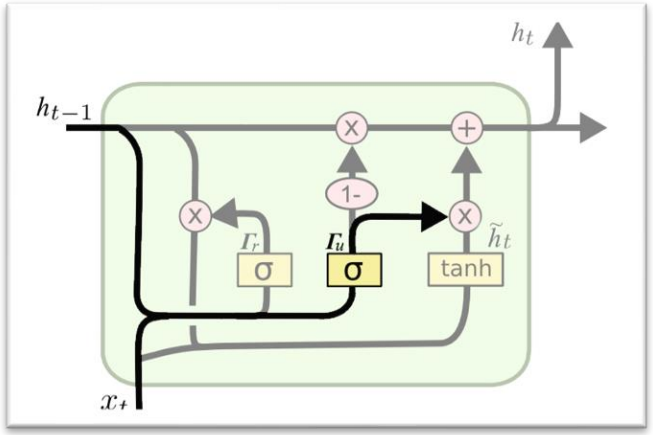
Il gate combina le informazioni dell'input corrente e del precedente **hidden state**.

Le informazioni sono codificate tramite una sigmoide che restituisce valori in [0, 1].

Reti ricorrenti

GRU cell

In una cella GRU si identificano in particolare:



$$\Gamma_u = \sigma(w_x^{(u)} x_t + w_h^{(u)} h_{t-1} + b_u)$$

update gate

Bilancia quanto il corrente hidden state dipenderà delle informazioni del precedente hidden state e del candidato hidden state .

Il gate combina le informazioni dell'input corrente e del precedente **hidden state**.

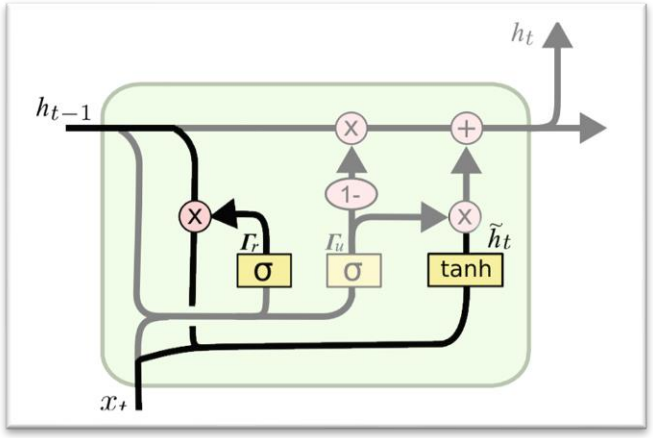
Le informazioni sono codificate tramite una sigmoide che restituisce valori in [0, 1].

Il bilanciamento avviene sfruttando il gate per sapere 'quanto scegliere' del candidato **hidden state** e 'quanto non scegliere' del precedente.

Reti ricorrenti

GRU cell

In una cella GRU si identificano in particolare:



$$\tilde{h}_t = tanh(w_x^{(\tilde{h})} x_t + w_h^{(\tilde{h})} \Gamma_r h_{t-1} + b_{\tilde{h}})$$

candidate hidden state

Rappresenta all'istante corrente l'informazione di memoria codificata nella cella.

Combina le informazioni dell'input corrente e del precedente **hidden state**.

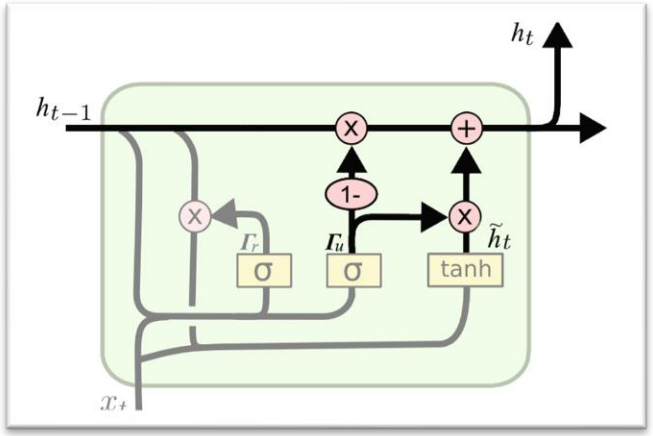
Dal precedente **hidden state** si attinge tanto quanto indicato dal *forget gate*.

Le informazioni sono codificate dall'operazione di *tanh* in [-1, 1], normalizzando e centrando i dati.

Reti ricorrenti

GRU cell

In una cella GRU si identificano in particolare:



$$h_t = \Gamma_u \tilde{h}_t + (1 - \Gamma_u) h_{t-1}$$

hidden state

Rappresenta la memoria codificata nella cella corrente e poi trasferita alla successiva celle e in uscita.

Combina le informazioni del precedente **hidden state** e del candidato.

La combinazione fra le informazioni è pesata in maniera bilanciata dal valore indicato dall'*update gate*.

Reti ricorrenti

GRU cell

Dall'architettura di **GRU cell** descritta, si possono quindi identificare vantaggi e svantaggi:

Vantaggi

In termini generali, presenta i vantaggi di una cella LSTM.

Computazionalmente più performante della cella LSTM.

Generalmente superiore in performance alle celle LSTM per sequenze ridotte.

Svantaggi

La minor complessità di queste celle riflette una riduzione della memoria a lungo termine.

Minore capacità di attingere da informazioni ed estrarre pattern e relazioni 'troppo lontane' nel passato.

Reti ricorrenti

Casi d'uso

I principali ambiti di applicazione delle reti ricorrenti possono essere ricondotti ai seguenti casi d'uso:

- *Natural language processing.*
- *Speech recognition.*
- *Speech generation.*
- *Time series analysis*
- *Image analysis.*
- *Video analysis.*
- …

Ve ne sono poi certamente altri; complice il fatto che:

Creare modelli di rete neurali è sempre più un arte nella quali si finisce per combinare in maniera modulare oggetti inizialmente pensati per differenti ambiti applicativi…

Reti ricorrenti

RNN in PyTorch

In PyTorch sono presenti e disponibili classi dedicate alla generazione delle celle o di sequenze di celle precedentemente definite.

Come per molti altri dei layer visti, il contenitore per questi elementi è il modulo *torch.nn*:

torch.nn — Recurrent Layers

Di seguito i riferimenti specifici:

- *RNN*
- *RNN Cell*
- *LSTM*
- *LSTM Cell*
- *GRU*
- *GRU Cell*

Proviamo?