

METODI DI INTELLIGENZA ARTIFICIALE E MACHINE LEARNING PER LA FISICA

S. Giagu - AA 2022/2023

Lezione 13 - 17.5.2023



SAPIENZA
UNIVERSITÀ DI ROMA

RECURRENT NEURAL NETWORKS

- Abbiamo visto come le ConvNet siano DNN ottimizzate per l'identificazione di correlazioni spaziali tra le feature presenti in strutture di dati organizzate in mesh fissate e simmetriche quali le immagini fotografiche
- molti problemi nel DL sono invece caratterizzati da input che hanno come caratteristica principale le **correlazioni di tipo temporale** tra le feature presenti nell'input:
 - analisi di testo scritto, analisi del linguaggio, sentiment analysis, riconoscimento di musica, dinamica di oggetti soggetti a forze (es. una reazione tra particelle elementari che evolve nel tempo), etc...
 - serie temporali: analisi di dati finanziari di borsa, predizioni su domanda/offerta di merci, sensori (ad esempio medici), analisi di segnali temporali misurati in interferometri gravitazionali, etc...
- Le reti neurali ricorrenti (RNN) rappresentano un'architettura DNN ottimizzata per processare sequenze: cioè ottimizzata per tenere conto in qualche modo durante il processamento dell'informazione in input non solo l'input corrente (come in una CNN) ma anche una parte degli input precedenti



esempio: i risultati dell'analisi delle parole/lettere di un testo analizzate negli step precedenti e non solo quella analizzata nello step corrente

MODELLI PER SEQUENZE

- tipico problema che si vuole risolvere: predire la successiva parola data una sequenza di parole che la precedono:

Questa mattina ho indossato i miei scarponi per fare un trekking

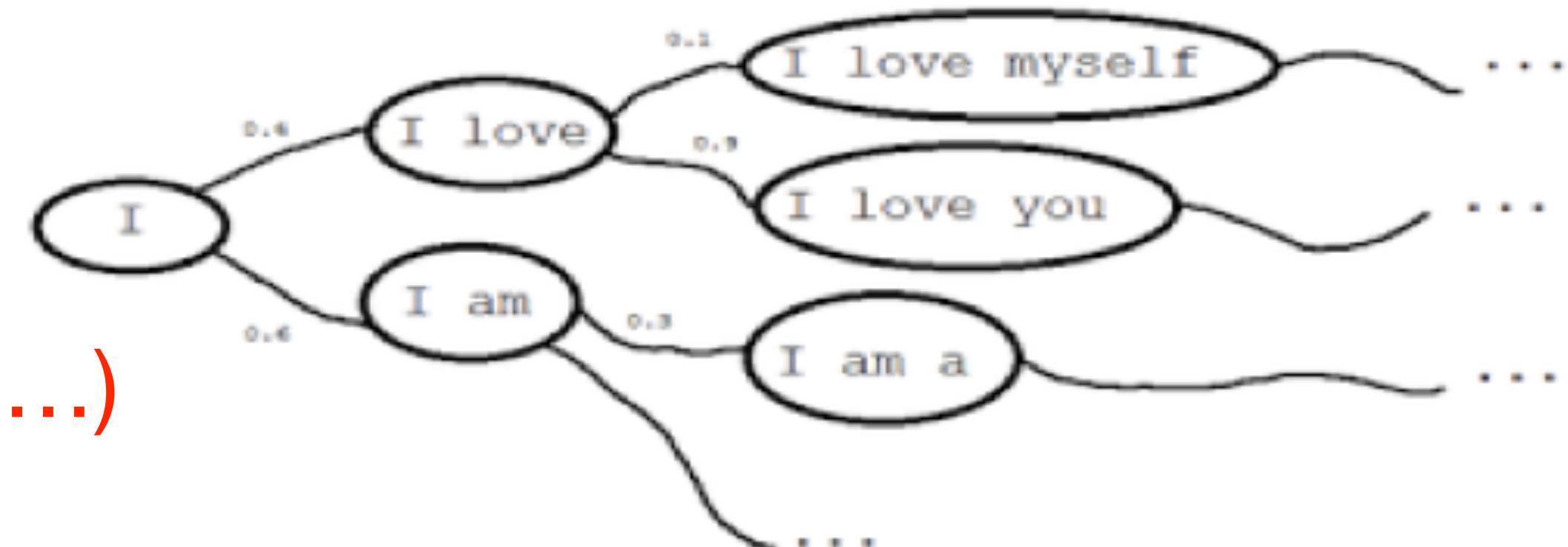


sequenza data

parola da predire

- per risolvere il problema dobbiamo costruire un modello di linguaggio che approssimi nel modo migliore la distribuzione reale:

- distribuzione reale: $P(\text{text}) = P(w_0, w_1, \dots, w_n) = P(w_0) \cdot P(w_1|w_0) \cdot P(w_2|w_1w_0) \cdot \dots \cdot P(w_n|\dots)$



- USO DI FIXED WINDOWS:
 - una CNN con filtri convoluzionali 1D è in grado di analizzare sequenze non troppo lunghe: la sequenza temporale di informazioni è passata come vettore piatto su cui scorre un filtro Conv1D, catturando le correlazioni tra elementi che rientrano nella dimensione del kernel

Questa mattina ho indossato i miei scarponi per fare un trekking

- ogni parola viene convertita in un indice numerico: per esempio come vettore one-hot
[1000010000100001]
- le sequenze di 0/1 vengono analizzate con filtri Conv1D con pesi ottimizzati rispetto al **target** (*trekking*)
- problema: dipendenza da elementi lontani nella sequenza:
 - “Io sono cresciuto in Italia ... bla bla ... lo parlo fluentemente italiano”

individuare correttamente la parola richiede spesso informazioni lontane nel vettore
che codifica l’informazione che il filtro Conv1D non riesce a catturare ...

- possibile soluzione: analizziamo l'intera sequenza di parole:

Questa mattina ho indossato i miei scarponi per fare un trekking

[10000000001000000000100000000...] ← nota: dimensione one-hot vector = spazio di tutte le parole!

- **problema 1:** è necessario trovare metodi “smart” per il **word embedding** (e.g. per convertire parole in vettori)
- **problema 2:** come approssimiamo la distribuzione reale del modello di linguaggio $P(\text{text})$?

- l’approccio più semplice possibile:

- **bag of words model:** ogni parola indipendente dalle altre, scordiamoci di grammatica e ordine delle parole nella frase:

$$P(\text{text}) = P(w_0, w_1, \dots, w_n) \sim P(w_0) \cdot P(w_1) \cdot \dots \cdot P(w_n)$$

- in pratica il riconoscimento del testo viene ad essere basato solo sulla frequenza relativa di ogni parola nel vocabolario di parole (spesso specifico del contesto per il quale si sta sviluppando il modello)

- limite fondamentale dei modelli bag of words ...
... i modi in cui si presentano le parole sono importanti!

*Il risultato è **positivo**, assolutamente non **disprezzabile***

VS

*Il cibo è **disprezzabile**, assolutamente non **positivo***

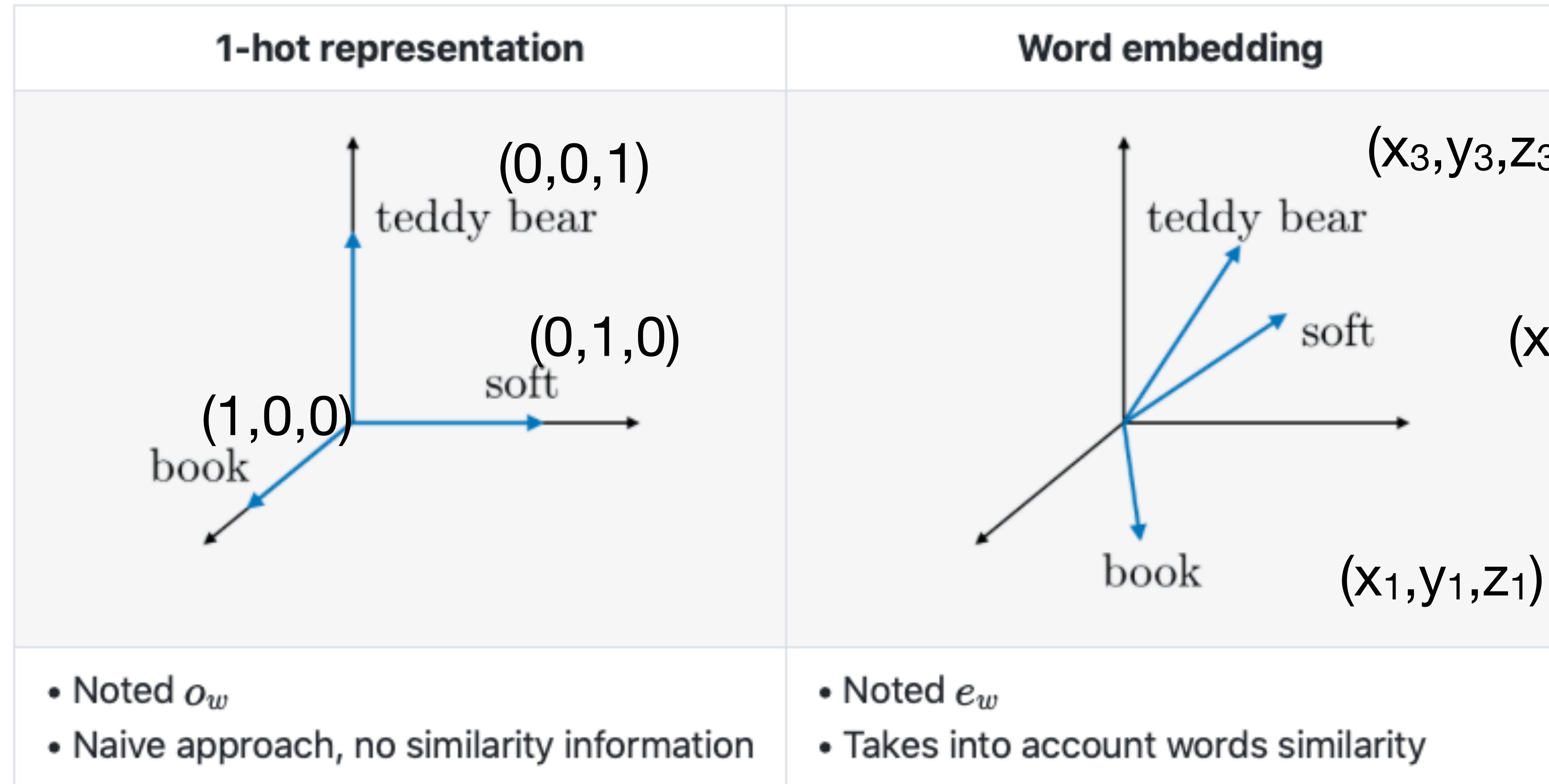
- per evitare il problema possiamo migliorare il nostro modello di linguaggio:
 - markov model:
$$P(\text{text}) = P(w_0, w_1, \dots, w_n) \sim P(w_0) \cdot P(w_1|w_0) \cdot P(w_2|w_1) \cdot \dots \cdot P(w_n|w_{n-1})$$
 - le probabilità dipendono solo dallo stato effettivo (la parola che precede una data parola)
 - oppure un'approssimazione migliore: **Recurrent Neural Networks**

$$P(\text{text}) = P(w_0, w_1, \dots, w_n) \sim P(w_0) \cdot P(w_1|w_0) \cdot P(w_2|w_1, w_0) \cdot \dots \cdot P(w_n|w_0, w_1, \dots, w_{n-1})$$

WORD EMBEDDING (I.E. APPRENDERE RAPPRESENTAZIONI DI PAROLE)

per essere capito da un NN un testo deve essere vettorizzato + rappresentato in modo efficace
due tecniche principali tipicamente utilizzate:

esempio:
dizionario
costituito da
3-parole



semplice,
diverge con la dimensione del dizionario

più complesso, più potente,
scale bene con la dimensione del dizionario

WORD EMBEDDING

Immaginiamo le frasi:

- 1) *Have a good day*
- 2) *Have a great day*

dicono ovviamente la stessa cosa perché *good* e *great* hanno in questo contesto significato simile. Tuttavia se codificate come one-hot vector **hanno rappresentazioni diverse (ortogonali nello spazio dell'embedding)**:

- 1) [1,0,0,0,0] [0,1,0,0,0] [0,0,1,0,0] [0,0,0,0,1]
- 2) [1,0,0,0,0] [0,1,0,0,0] [0,0,0,1,0] [0,0,0,0,1]

in questo spazio 5D quindi *great* e *good* sono diverse quanto lo sono *have* e *day* ...

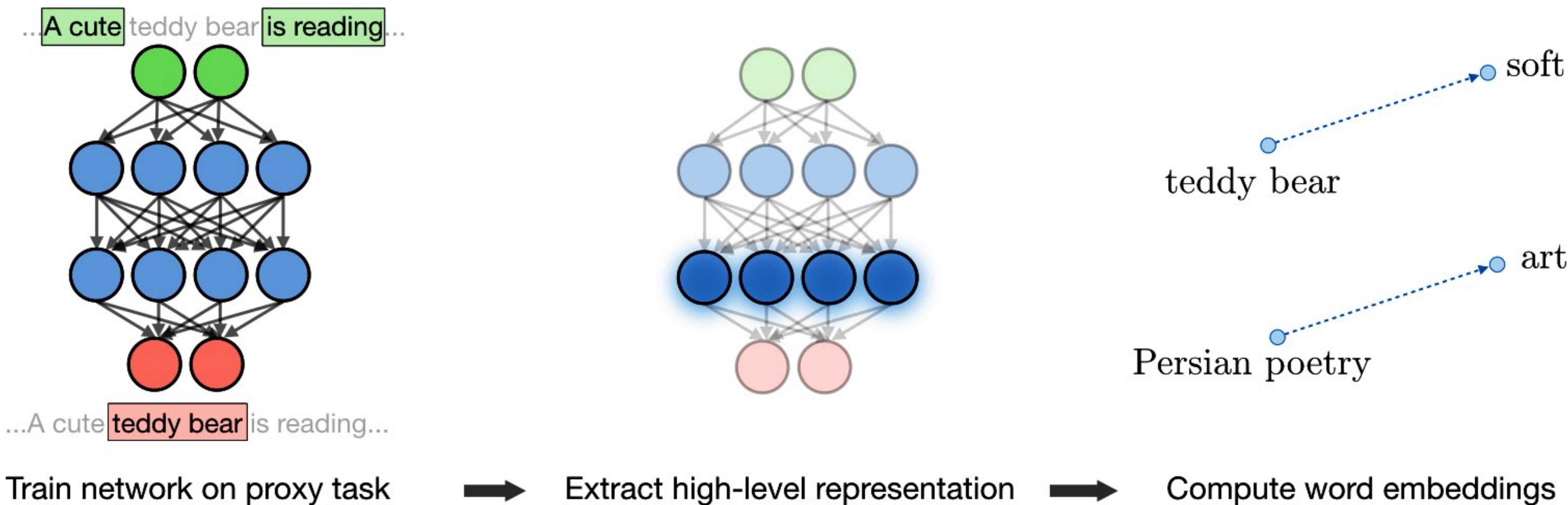
il word embedding (Y. Bengio 2001) sviluppa l'idea di associare in modo automatico rappresentazioni più efficaci dei vettori che rappresentano le parole nelle sequenze di testo da analizzare. Viene appreso o in modo naïve per esempio basandosi sulla frequenza delle parole, oppure in modo più sofisticato assumendo che le parole con contesti simili occupino posizioni “spaziali” simili

WORD EMBEDDING: WORD2VEC METHOD

Word2Vec (Mikolov (Google) 2013) è una delle tecniche più popolari di word embedding

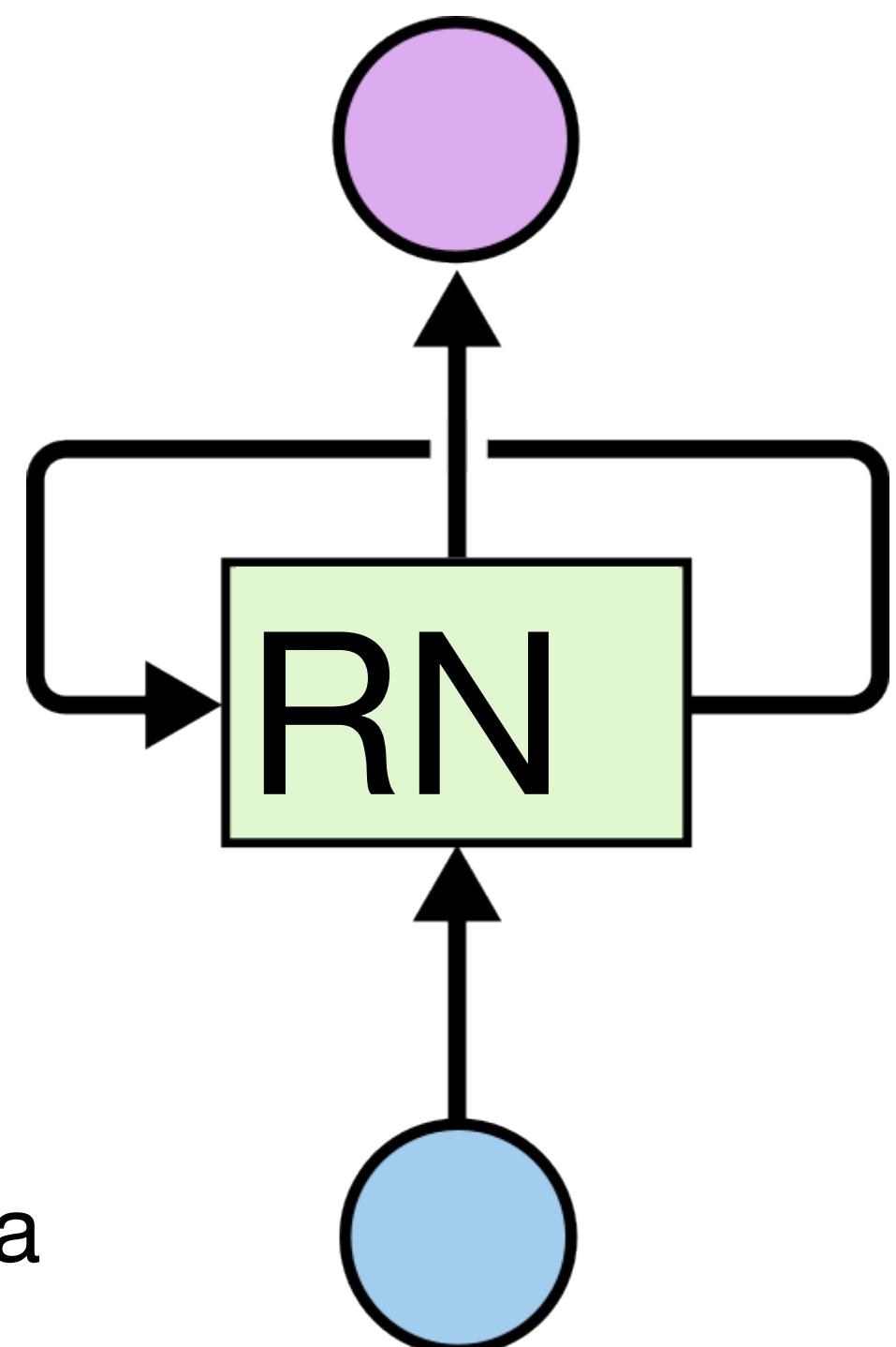
si allena una DNN ad associare un vettore di numeri reali ad ogni parola in modo tale che parole con significati simili o correlate in qualche modo siano associate a vettori “simili”

- Common Bag Of Word model: prende una frase in input e cerca di predire la parola che corrisponde a tale contesto
 - le rappresentazioni di alto livello vengono utilizzate come embedding per la parola di target
- Skip-gram model: simile ma lavora al contrario parte dalla parola target e cerca di predire le probabilità che sia associata alle altre N parole della frase analizzata



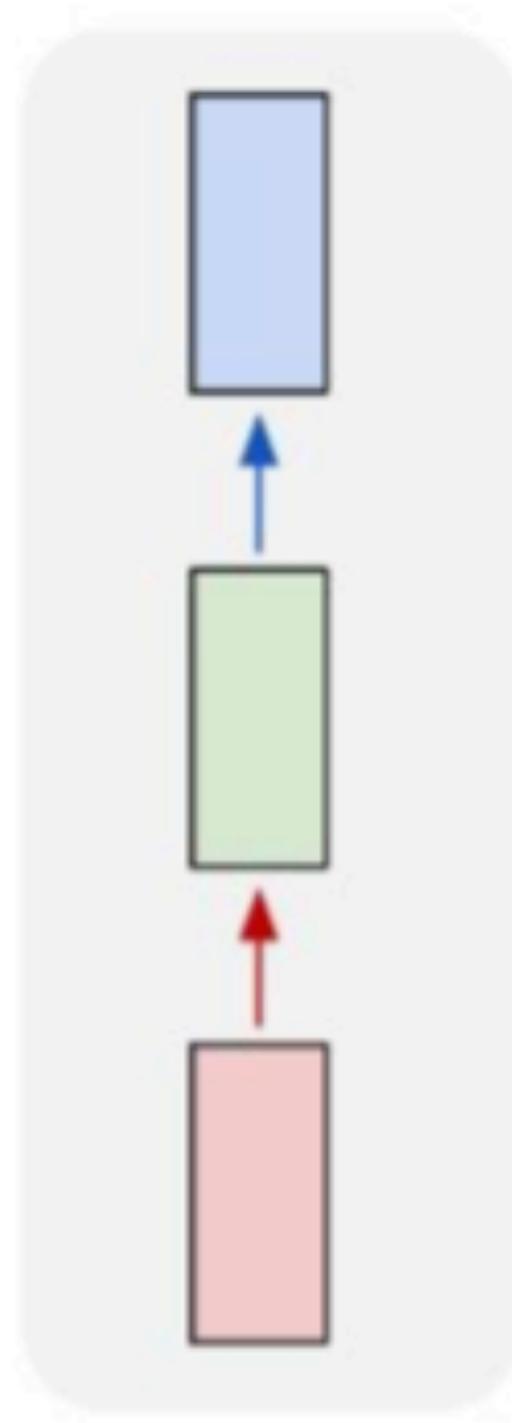
RNN: ARCHITETTURE PER SEQUENZE

- RNN sono specifiche architecture ottimizzate per identificare correlazioni a lungo-termine in sequenze ordinate di informazioni di lunghezza variabile
- tipica task per una RNN: data una sequenza di feature (testo, muscia, un segnale temporale, una lista di particelle prodotta in un decadimento, ...), predire uno più target (la successiva parola in una frase, il tempo atmosferico nelle successive 24h, il sapore di un jet aerobico in un esperimento hep, ...)
- una RNN deve poter:
 - ricevere input di lunghezza variabile
 - riuscire a mantenere traccia delle dipendenze tra elementi distanti della sequenza
 - riuscire a mantenere informazioni sull'ordine (past features) tra gli elementi della sequenza
 - avere parametri (pesi) condivisi, in modo che le correlazioni identificate tra elementi possano essere trasferite in diversi punti della sequenza



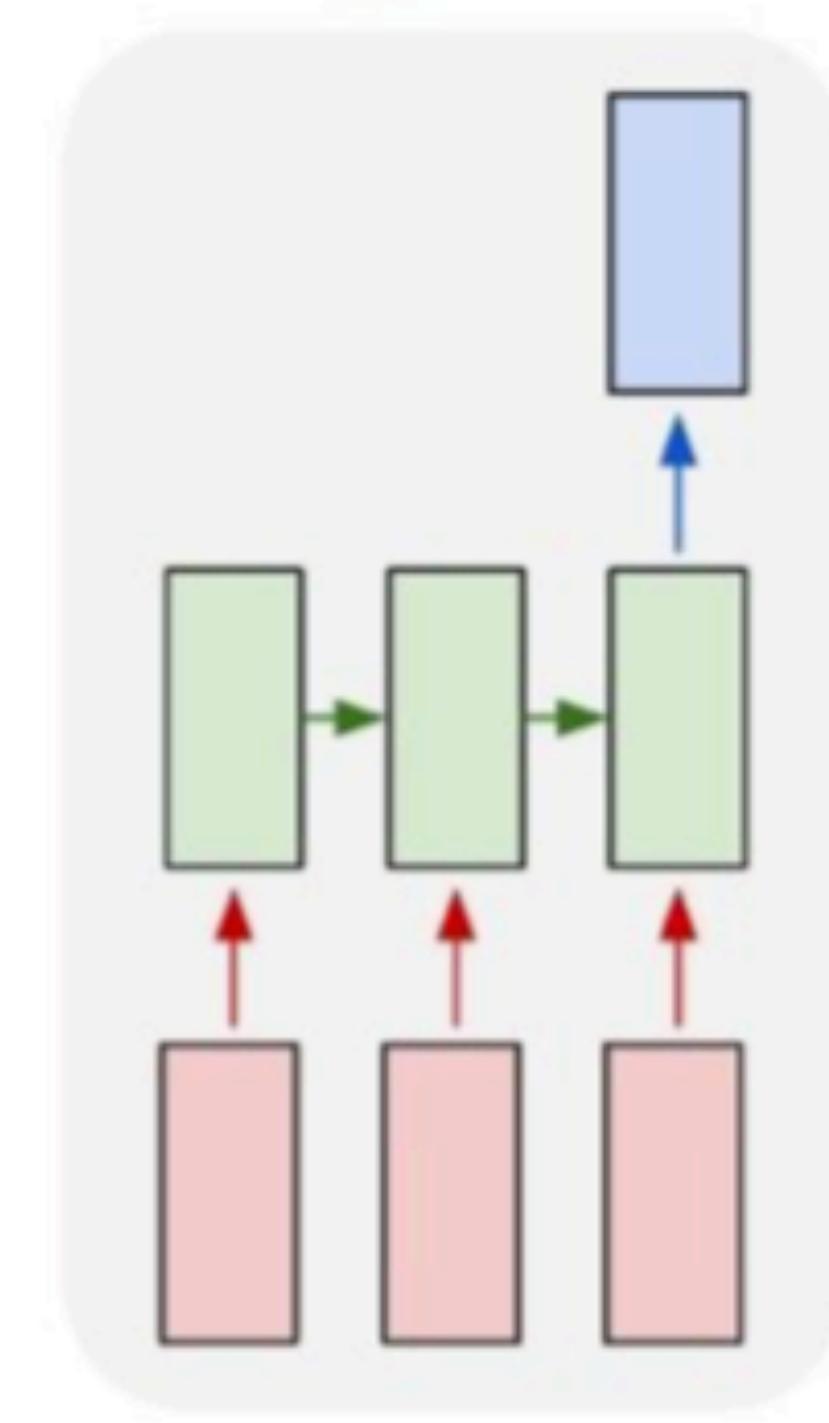
DIVERSE TOPOLOGIE DI RNN SONO UTILIZZATE PER ANALIZZARE DIFFERENTI TIPI DI SEQUENZE

one to one



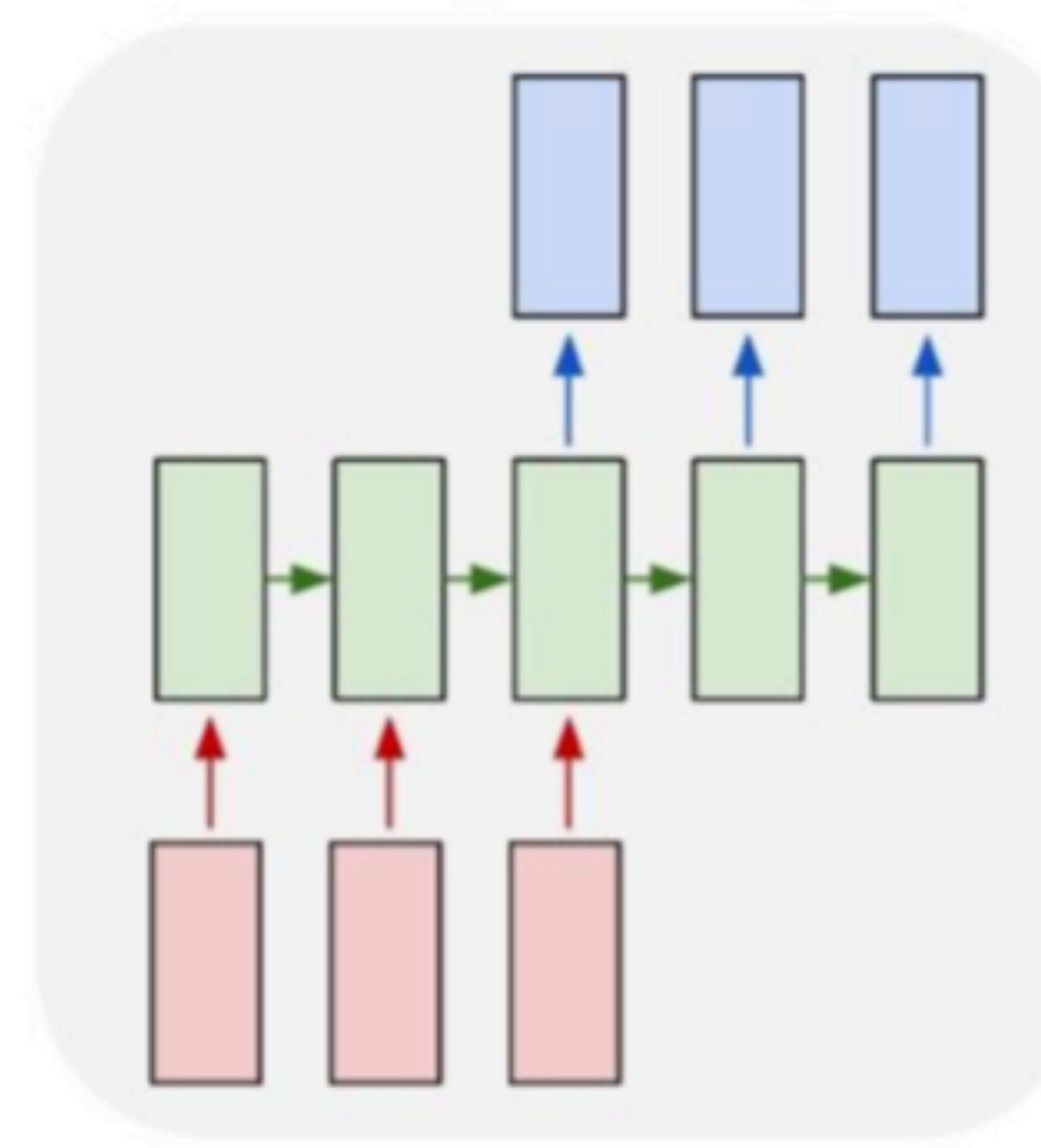
DNN/CNN
convenzionale
image→class

many to one



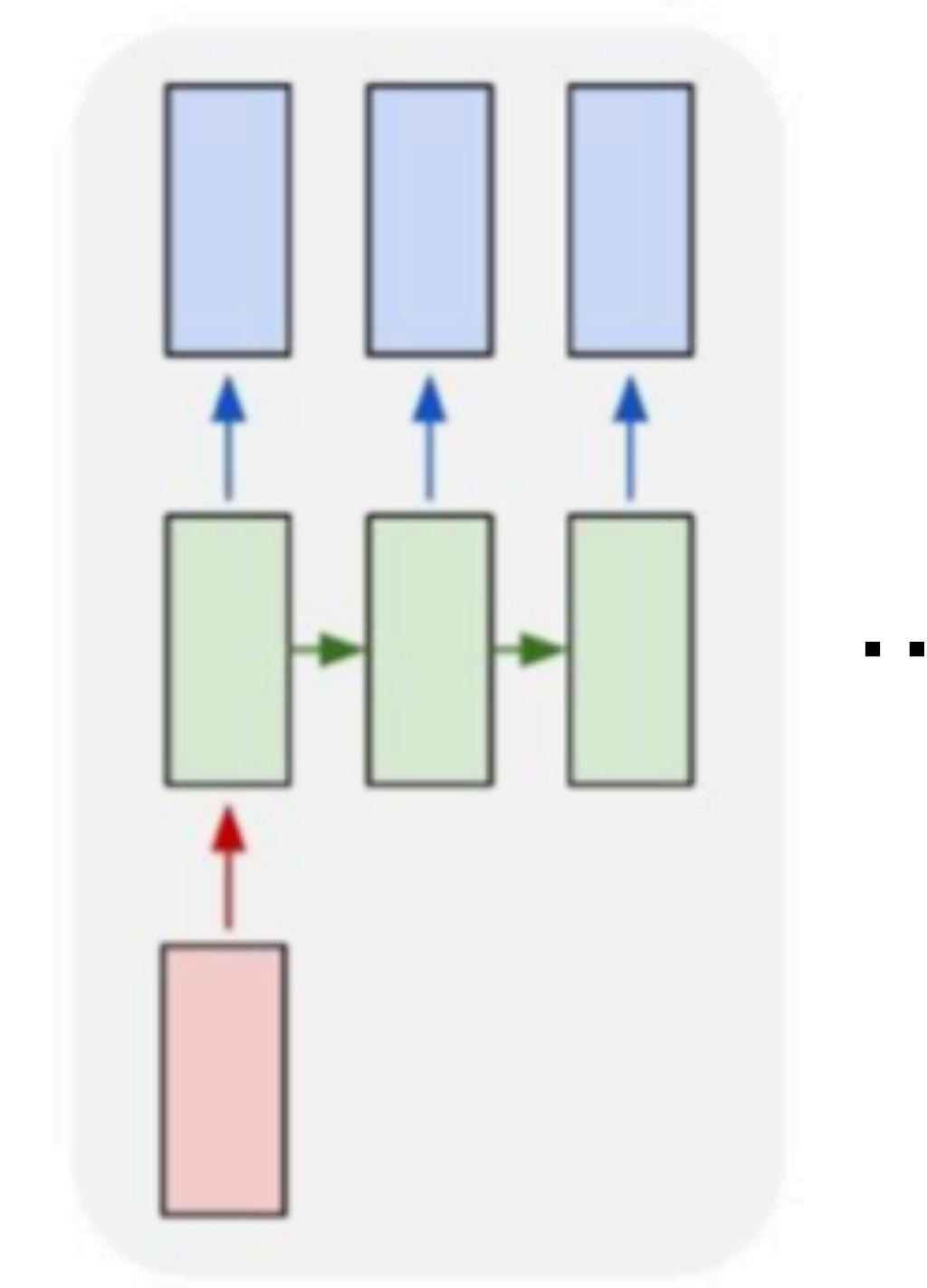
prende una sequenza e
genera un output singolo
ex. sentiment analysis
sequence of words → sentiment
ex. time series forecasting
sequence of obs. → future values

many to many



prende una sequenza e
genera un'altra sequenza
ex. traduzione automatica
ex. generazione di musica ...

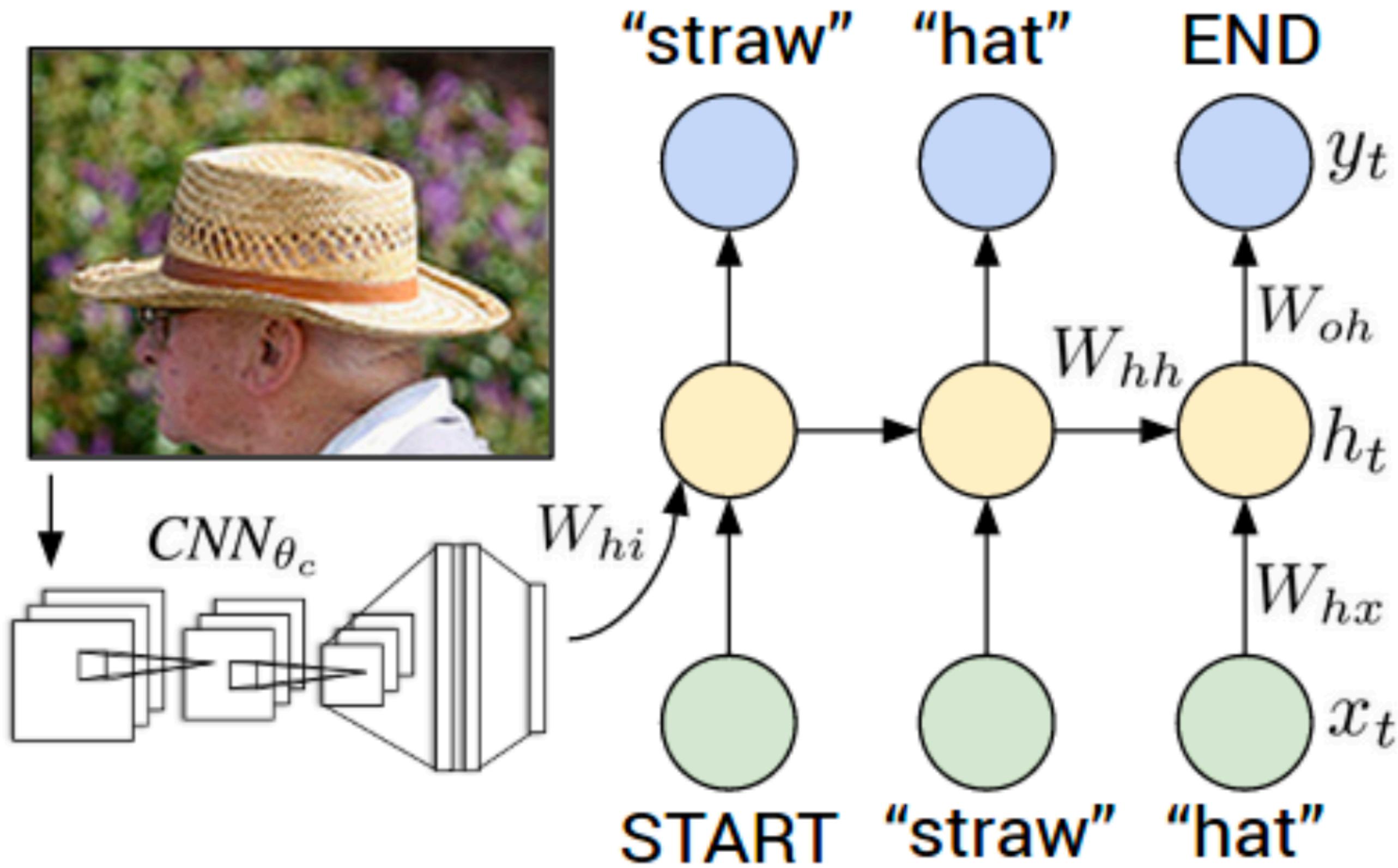
one to many



prende un input singolo single
e genera una sequenza
ex. image captioning
image → sequence of words

ESEMPIO USO RNN: IMAGING CAPTIONING

ConvNet + RNN

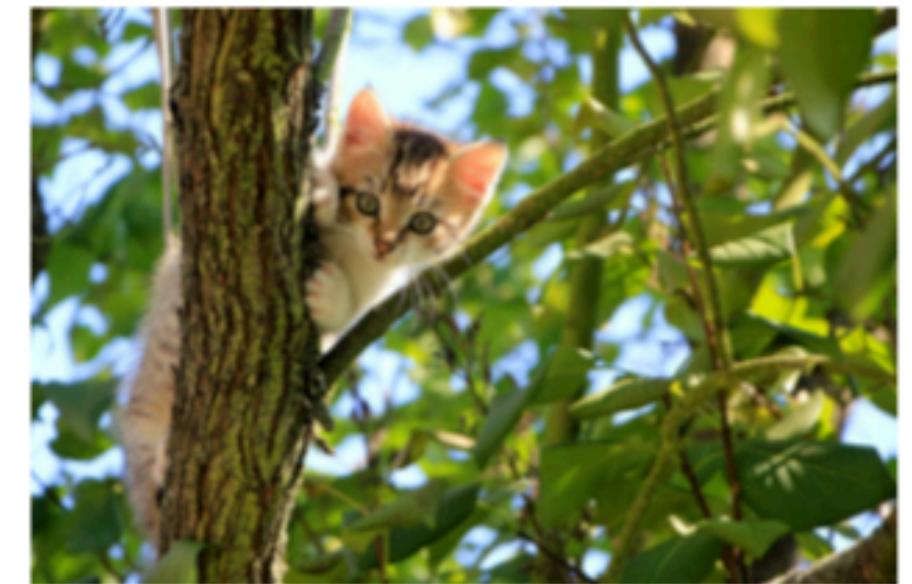


real example: captions generated with neuraltalk2
<https://github.com/karpathy/neuraltalk2>
paper

CNN with attention + RNN



A cat sitting on a suitcase on the floor



A cat is sitting on a tree branch



Two people walking on the beach with surfboards



A tennis player in action on the court



A woman is throwing a frisbee in a park.

ALTRI ESEMPI...

THE SONNETS

by William Shakespeare

From fairest creatures we desire increase,
That thereby beauty's rose might never die,
But as the riper should by time decease,
His tender heir might bear his memory:
But thou, contracted to thine own bright eyes,
Feed'st thy light's flame with self-substantial fuel,
Making a famine where abundance lies,
Thyself thy foe, to thy sweet self too cruel:
Thou that art now the world's fresh ornament,
And only herald to the gaudy spring,
Within thine own bud buried thy content,
And tender churl mak'st waste in niggarding:
Pity the world, or else this gluton be,
To eat the world's due, by the grave and thee.

When forty winters shall besiege thy brow,
And dig deep trenches in thy beauty's field,
Thy youth's proud livery so gazed on now,
Will be a tatter'd weed of small worth held:
Then being asked, where all thy beauty lies,
Where all the treasure of thy lusty days;
To say, within thine own deep sunken eyes,
Were an all-eating shame, and thrifless praise.
How much more praise deserved thy beauty's use,
If thou couldst answer 'This fair child of mine
Shall sum my count, and make my old excuse,'
Proving his beauty by succession thine!
This were to be new made when thou art old,
And see thy blood warm when thou feel'st it cold.

text
generation

RNN

tyntd-iafhatawiaoahrdemot lytdws e ,tfti, astai f ogoh eoase rrranbyne 'nhthnee e
plia tkldrgd t o idoe ns,smtt h ne etie h,hregtrs nigtike,aoaenns lng

train more

"Tmont thithey" fomesscerliund
Keushey. Thom here
sheulke, anmerenith ol sivh I lalterthend Bleipile shuw fil on aseterlome
coaniogennc Phe lism thond hon at. MeiDimorotion in ther thize."

train more

Aftair fall unsuch that the hall for Prince Velzonski's that me of
her hearly, and behs to so arwage fiving were to it beloge, pavu say falling misfort
how, and Gogition is so overelical and ofter.

train more

"Why do what that day," replied Natasha, and wishing to himself the fact the
princess, Princess Mary was easier, fed in had oftened him.
Pierre aking his soul came to the packs and drove up his father-in-law women.

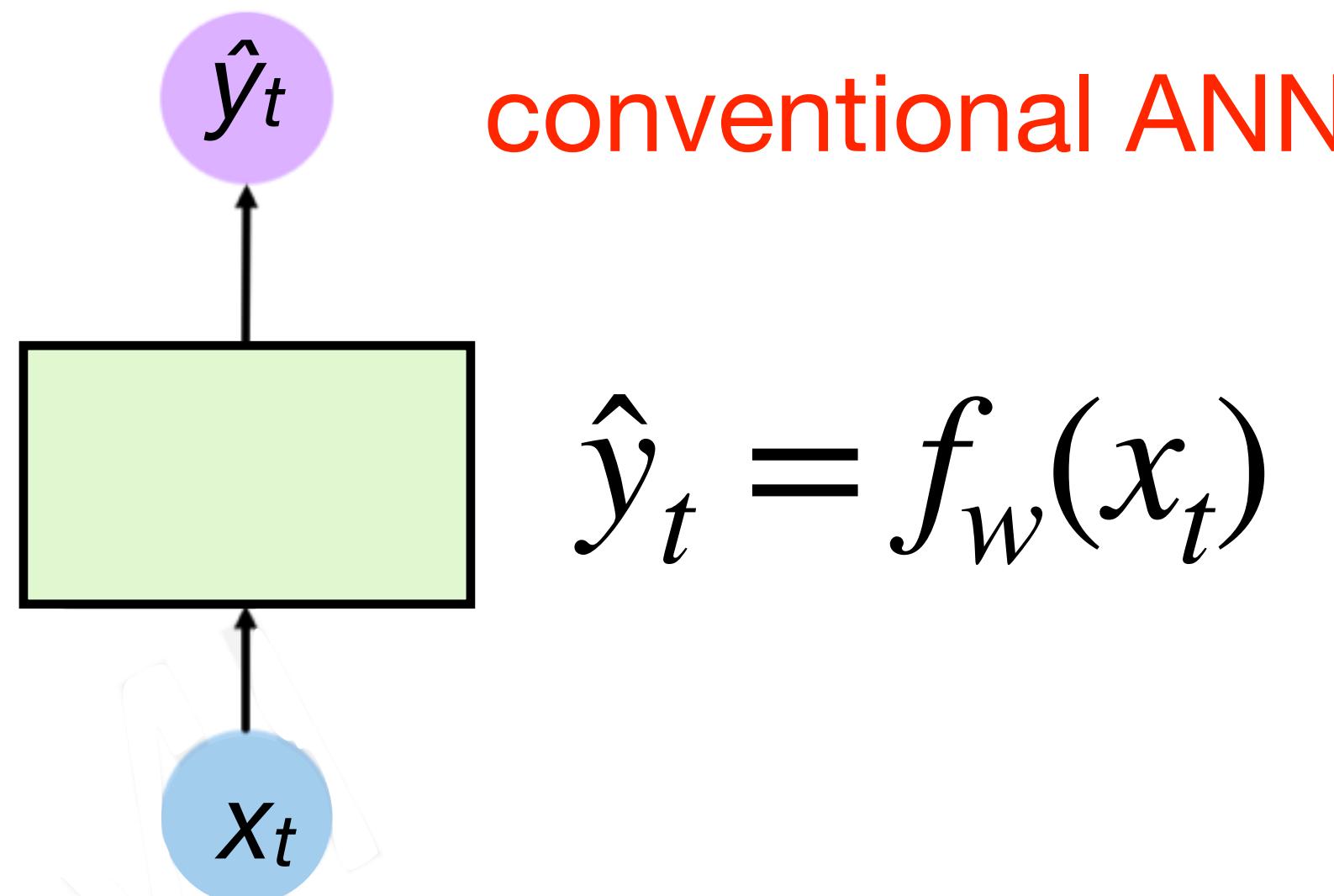
code
generation

RNN

static void do_command(struct seq_file *m, void *v)
{
 int column = 32 << (cmd[2] & 0x80);
 if (state)
 cmd = (int)(int_state ^ (in_8(&ch->ch_flags) & Cmd) ? 2 : 1);
 else
 seq = 1;
 for (i = 0; i < 16; i++) {
 if (k & (1 << 1))
 pipe = (in_use & UMXTHREAD_UNCCA) +
 ((count & 0x00000000fffff8) & 0x0000000f) << 8;
 if (count == 0)
 sub(pid, ppc_md.kexec_handle, 0x20000000);
 pipe_set_bytes(i, 0);
 }
 /* Free our user pages pointer to place camera if all dash */
 subsystem_info = &of_changes[PAGE_SIZE];
 rek_controls(offset, idx, &soffset);
 /* Now we want to deliberately put it to device */
 control_check_polarity(&context, val, 0);
 for (i = 0; i < COUNTER; i++)
 seq_puts(s, "policy ");
}

MODULO BASE DI UNA RNN: CELL

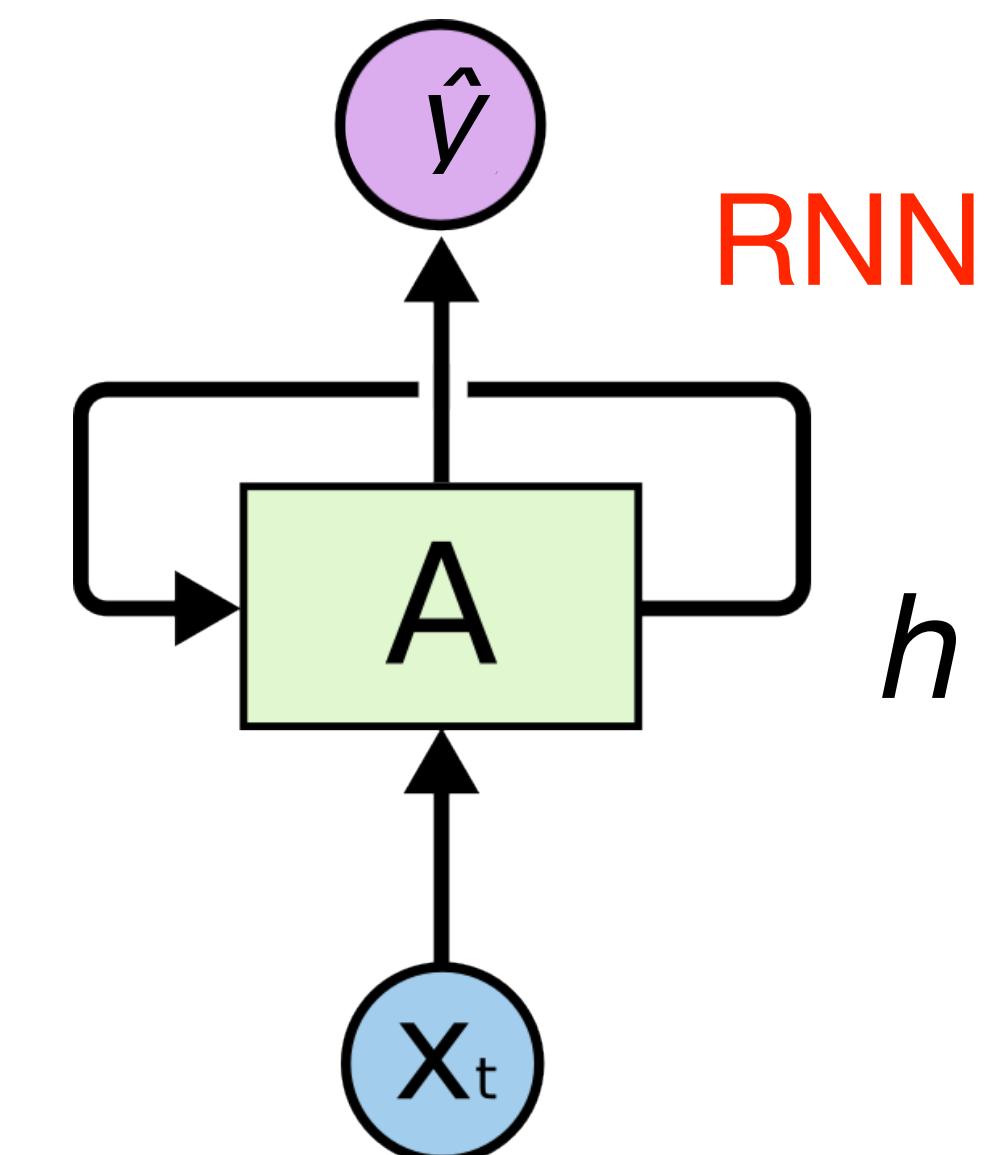
- una RNN processa l'input tramite un loop (connessione ricorrente) che permette la persistenza dell'informazione durante l'intera fase di processamento della sequenza
- **cell**: è definito da un set di operazioni eseguite ad ogni time step: A è una NN che analizza l'elemento $t \in [0, n]$ della sequenza di input: x_t , e produce l'output h_t (**hidden state o context vector**). h_t viene passato allo stesso NN durante il processamento del successivo elemento della sequenza nello step $t+1$



old-state

input allo step t

$$h_t = f_w(h_{t-1}, x_t)$$

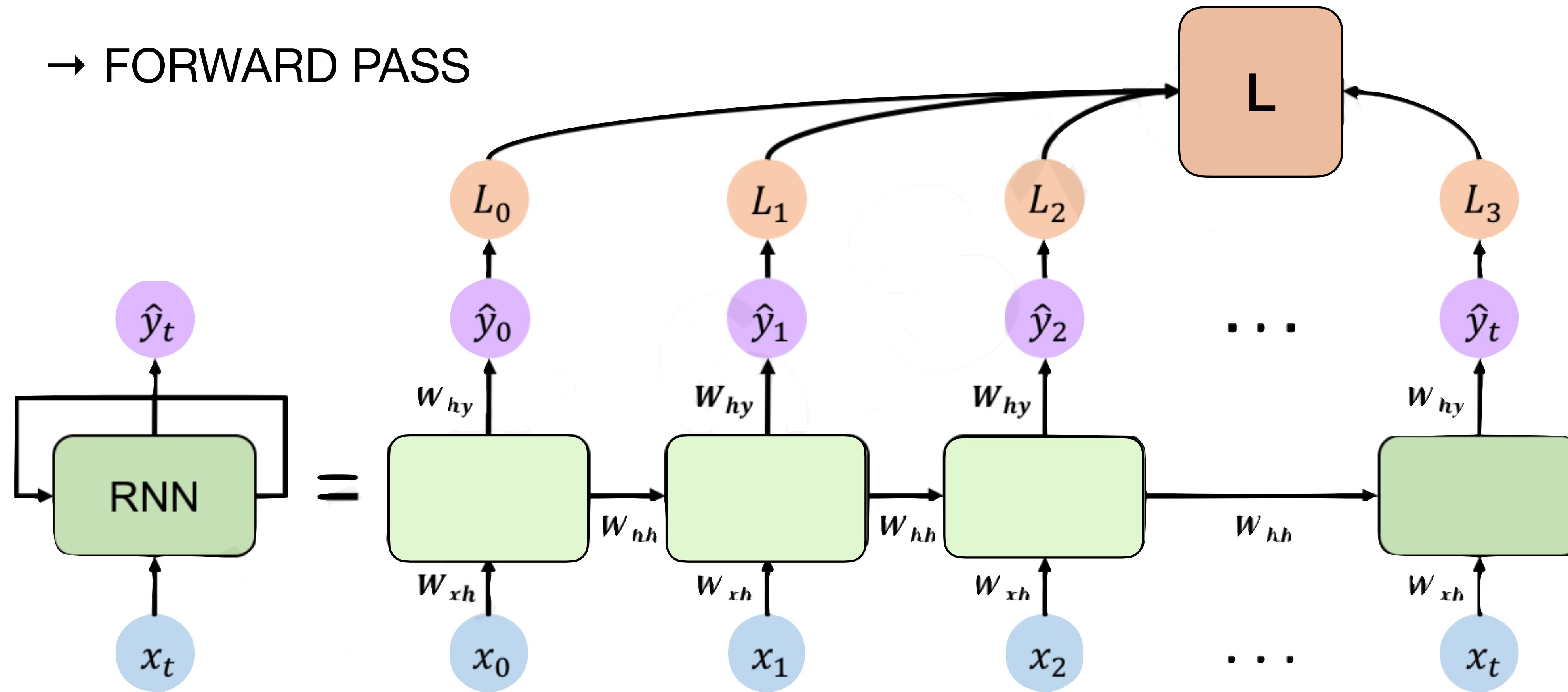


- **Punto fondamentale**: la stessa funzione f_w (i.e. lo stesso NN con lo stesso set di pesi w) viene utilizzata per processare ogni elemento t della sequenza ...

RNN COME GRAFO COMPUTAZIONALE TEMPORALE

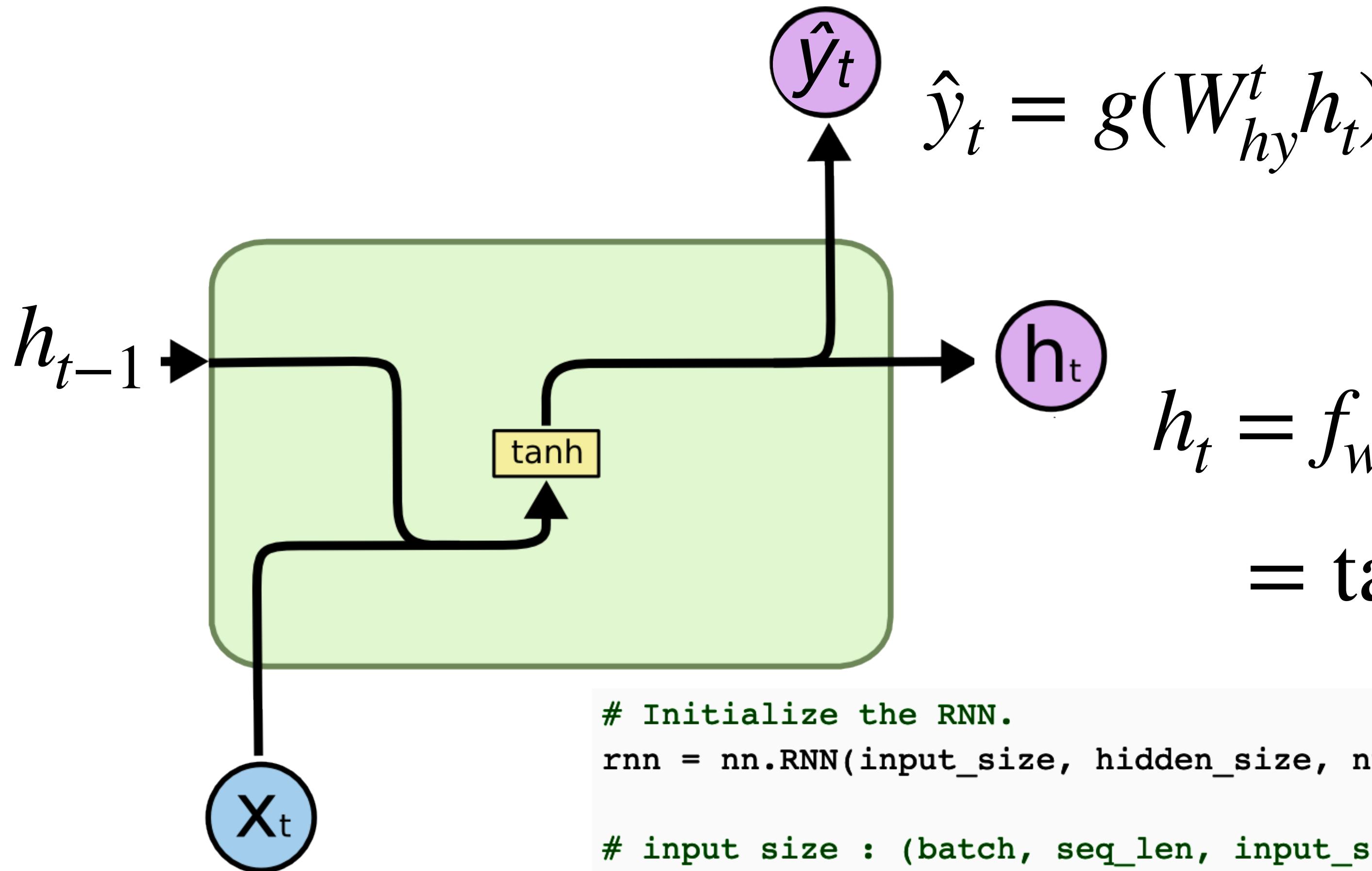
- La RNN può essere pensata come una serie di copie multiple della stessa rete neurale convenzionale, ognuna che passa un messaggio al suo successore ...

→ FORWARD PASS



passare il messaggio è ciò che permette alla RNN di identificare le correlazioni a lunga distanza 15

IMPLEMENTAZIONE VANILLA DI UNA RNN



$$\hat{y}_t = g(W_{hy}^t h_t)$$

ex. sequence-labeling:

y_t task: classificare ogni input x_t della sequenza

$g = \text{softmax} + \text{loss} = \text{accuracy media sulla sequenza}$

$$h_t = f_w(h_{t-1}, x_t) =$$

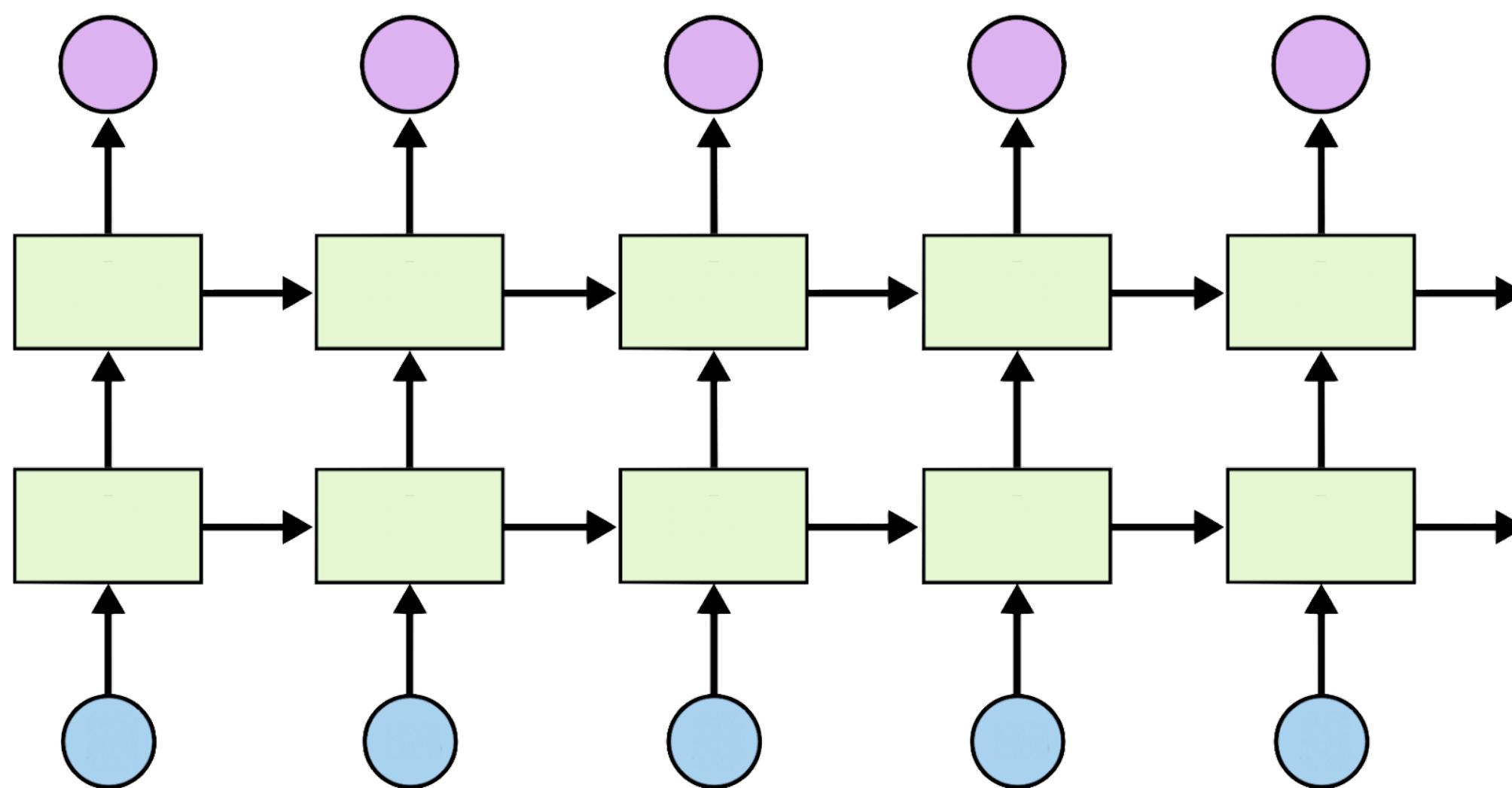
$$= \tanh(W_{hh}^t h_{t-1} + W_{xh}^t x_t)$$

```
# Initialize the RNN.  
rnn = nn.RNN(input_size, hidden_size, num_layers = 1, nonlinearity='tanh', batch_first=True)  
  
# input size : (batch, seq_len, input_size)  
  
out, h_n = rnn(inputs)  
  
# num_directions = 1 (= 2 for bidirectional RNNs)  
# out shape = (batch, seq_len, num_directions * hidden_size)  
# h_n shape = (num_layers * num_directions, batch, hidden_size)
```

VARIAZIONI

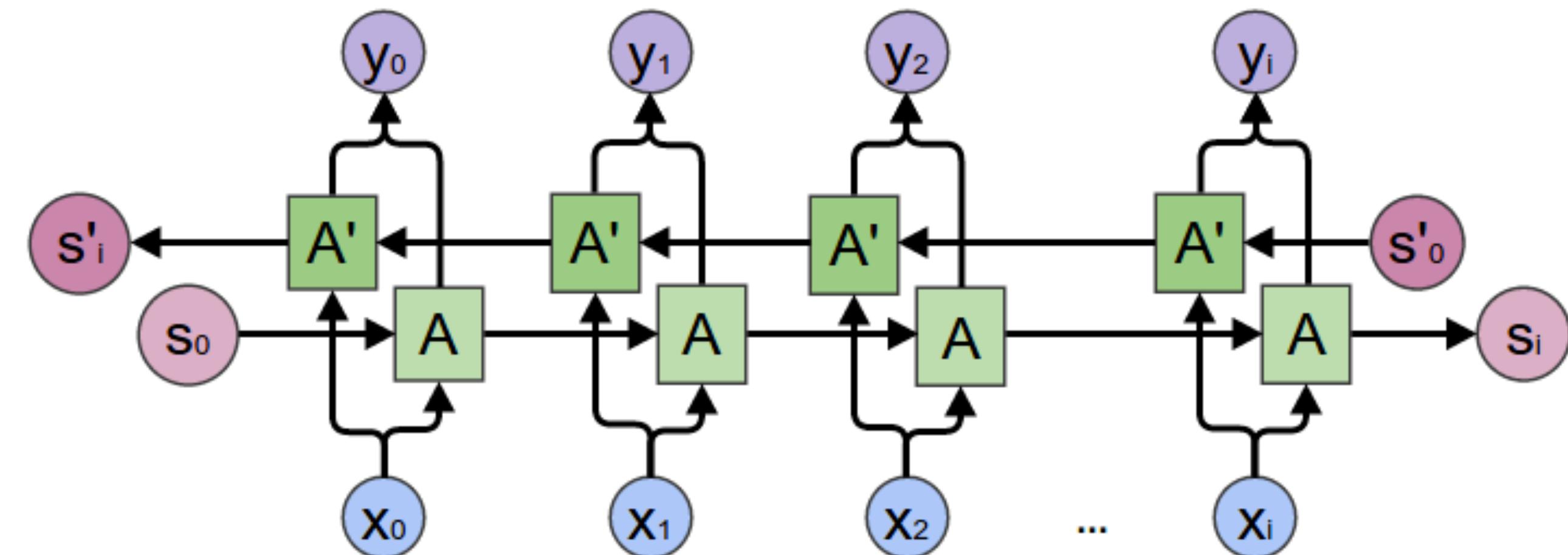
Stacked RNNs

permette rappresentazioni più complesse
catturando correlazioni a different scale



```
# Initialize the RNN.  
rnn = nn.RNN(input_size, hidden_size, num_layers = 2,  
              nonlinearity='tanh', batch_first=True)
```

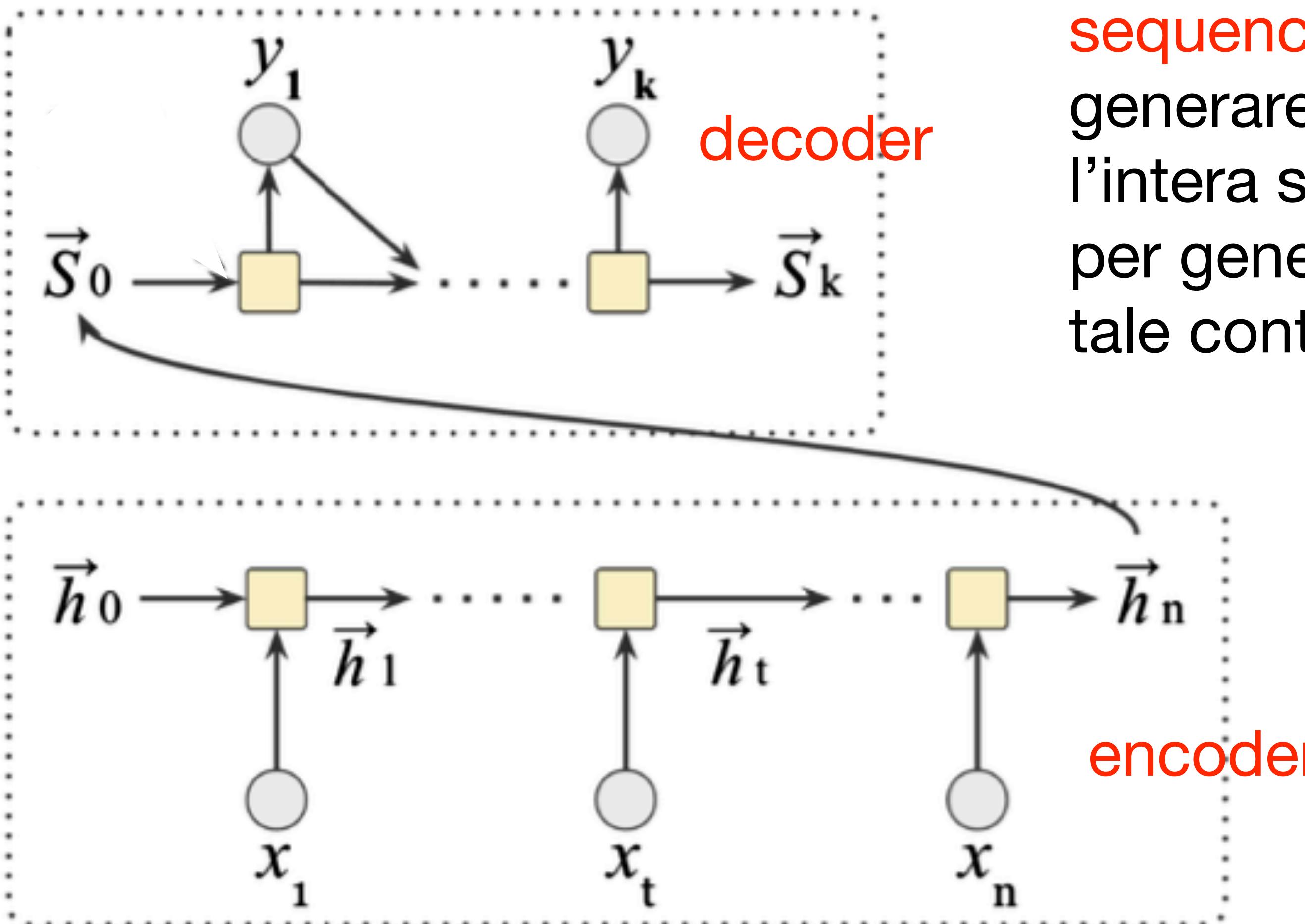
Bidirectional RNNs
input è processato sia nell'ordine
temporale normale che invertito
permette alla BRNN di guardare anche
a context vector futuri



```
# Initialize the RNN.  
rnn = nn.RNN(input_size, hidden_size, num_layers = 1,  
              nonlinearity='tanh',  
              bidirectional = True, batch_first=True)
```

VARIAZIONI

Encoder-Decoder RNNs



sequence2sequence: usa una RNN per generare un context vector che rappresenti l'intera sequenza di input, e una RNN separata per generare una nuova sequenza a partire da tale context vector

usata spesso nella traduzione automatica

BACKPROPAGATIN THROUGH TIME

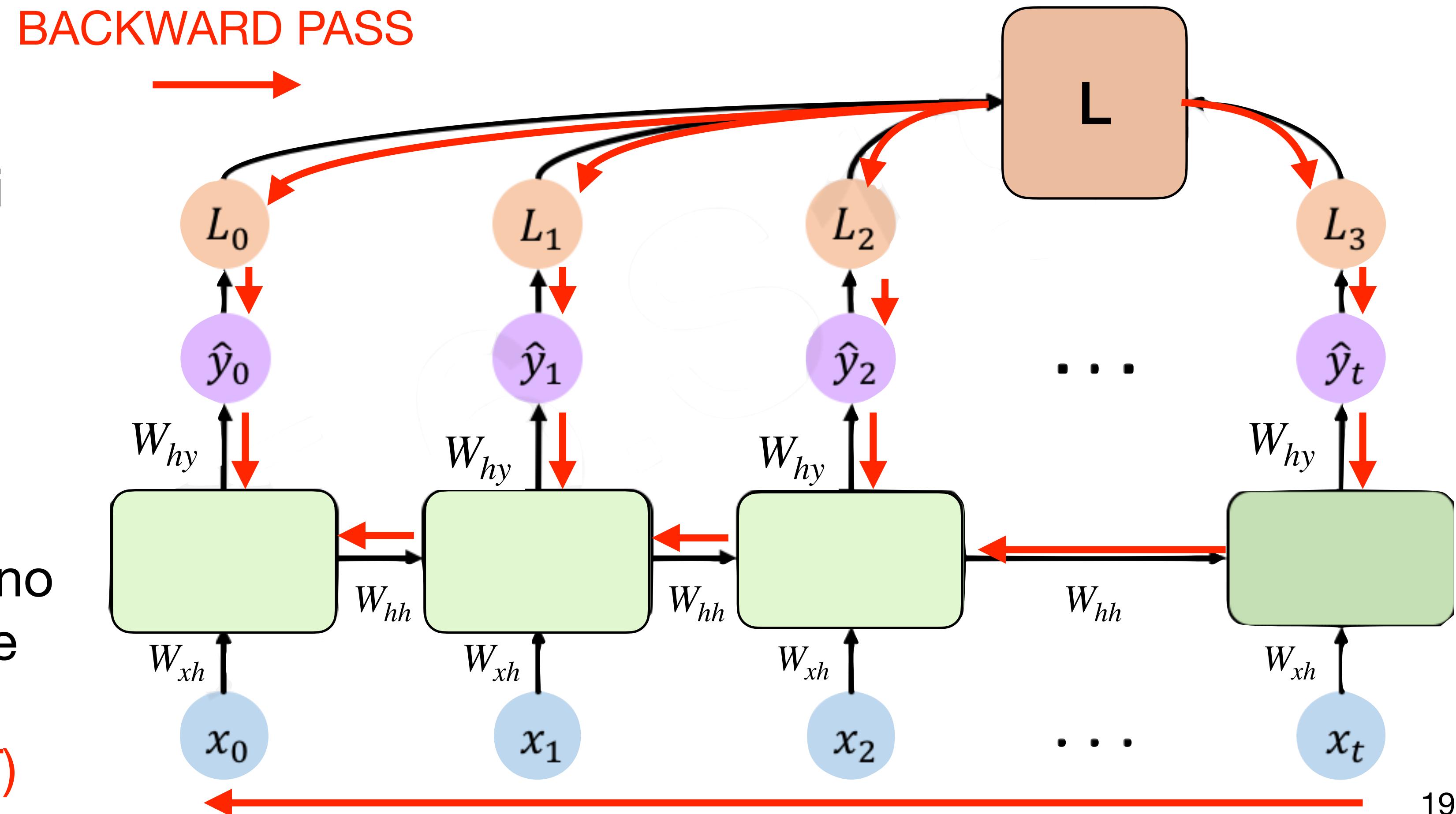
BPTT è l'applicazione della backprop ad una RNN

Concettualmente funziona eseguendo l'unrolling di tutti i timestep dell'input

al timestep T, la derivata della loss L rispetto alla matrice dei pesi W è data da:

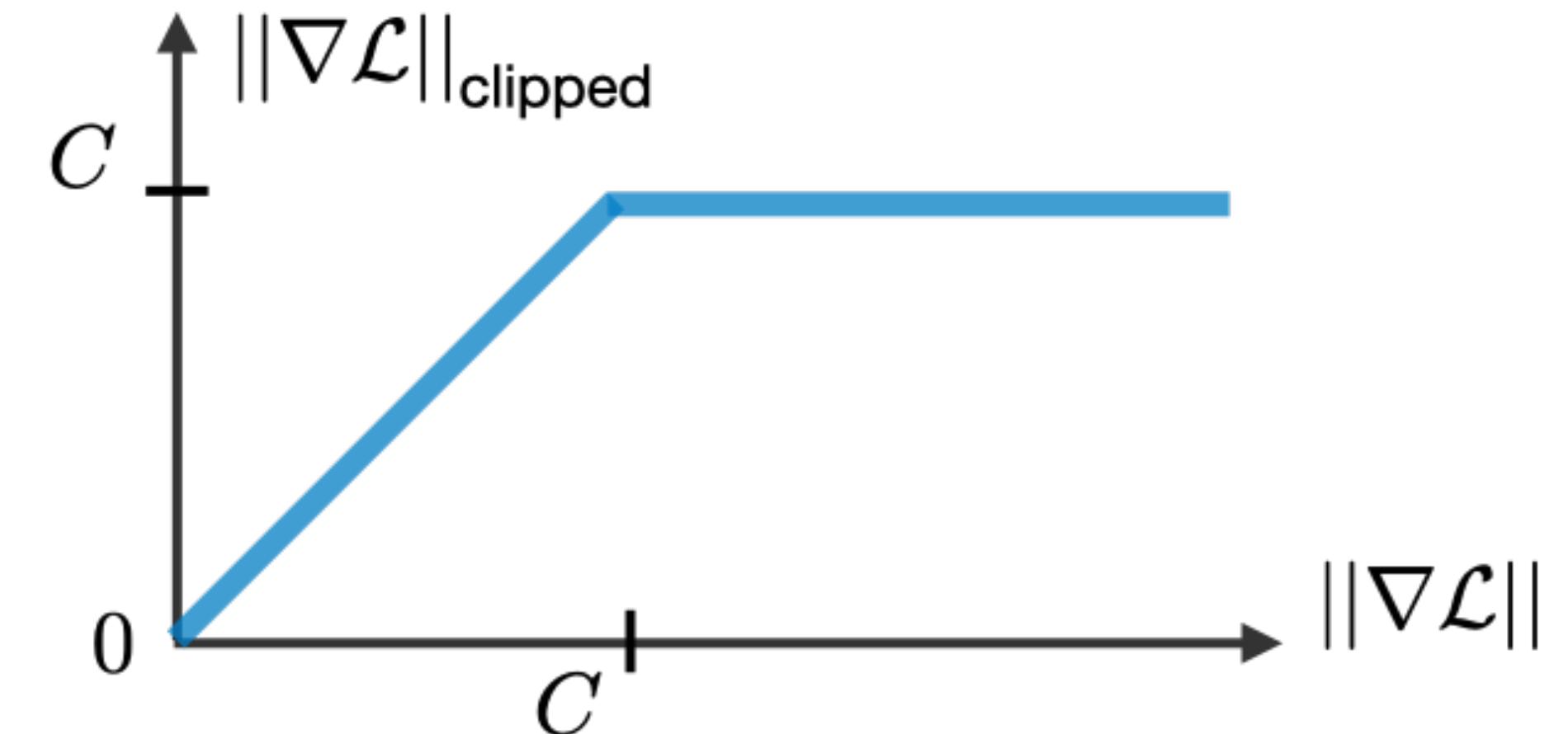
$$\frac{\partial L^{(T)}}{\partial W} = \sum_{t=1}^T \frac{\partial L^{(T)}}{\partial W} \Big|_{(t)}$$

molto costosa! Le RNN risultano molto più lente da addestrare delle CNN
(mitigato via truncated-BPTT)



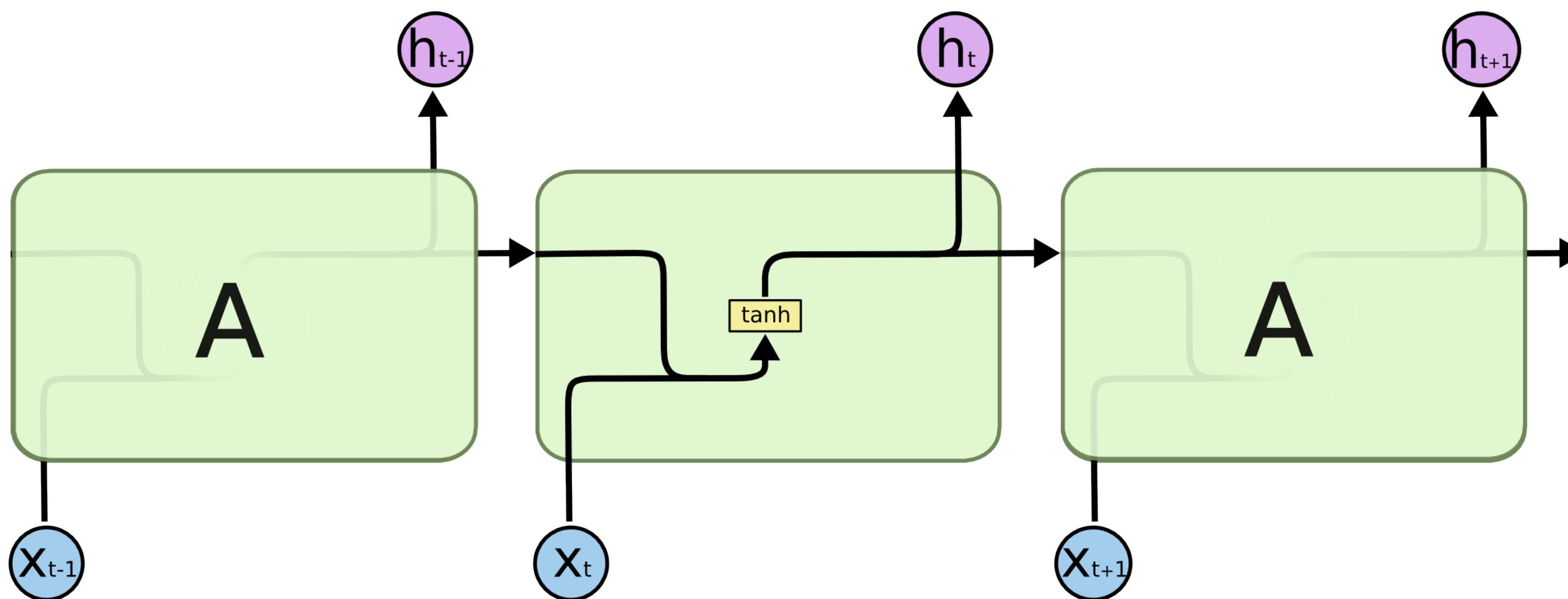
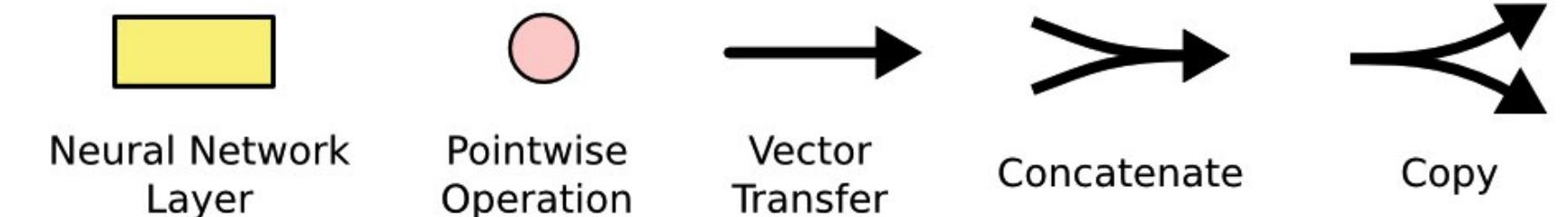
PROBLEMI CON LE VANILLA RNNs: LONG TERM DEPENDENCIES

- BPTT fa sì che la RNN sia molto sensibile ai fenomeni del vanishing e del exploding gradient. Lunghe sequenze implicano un gran numero di moltiplicazioni di gradienti, producendo risultati sempre più piccoli ($\text{grad} < 1$) o sempre più grandi ($\text{grad} > 1$)
- attivazioni non limitate come **ReLU** risolvono il problema ($\text{grad} < 1$) ma non quello ($\text{grad} > 1$) (**exploding gradients**)
- mitigazione del problema: **gradient clipping**
 - si limita il massimo valore del gradiente durante la back-propagation
 - **ma riduce le prestazioni del modello**



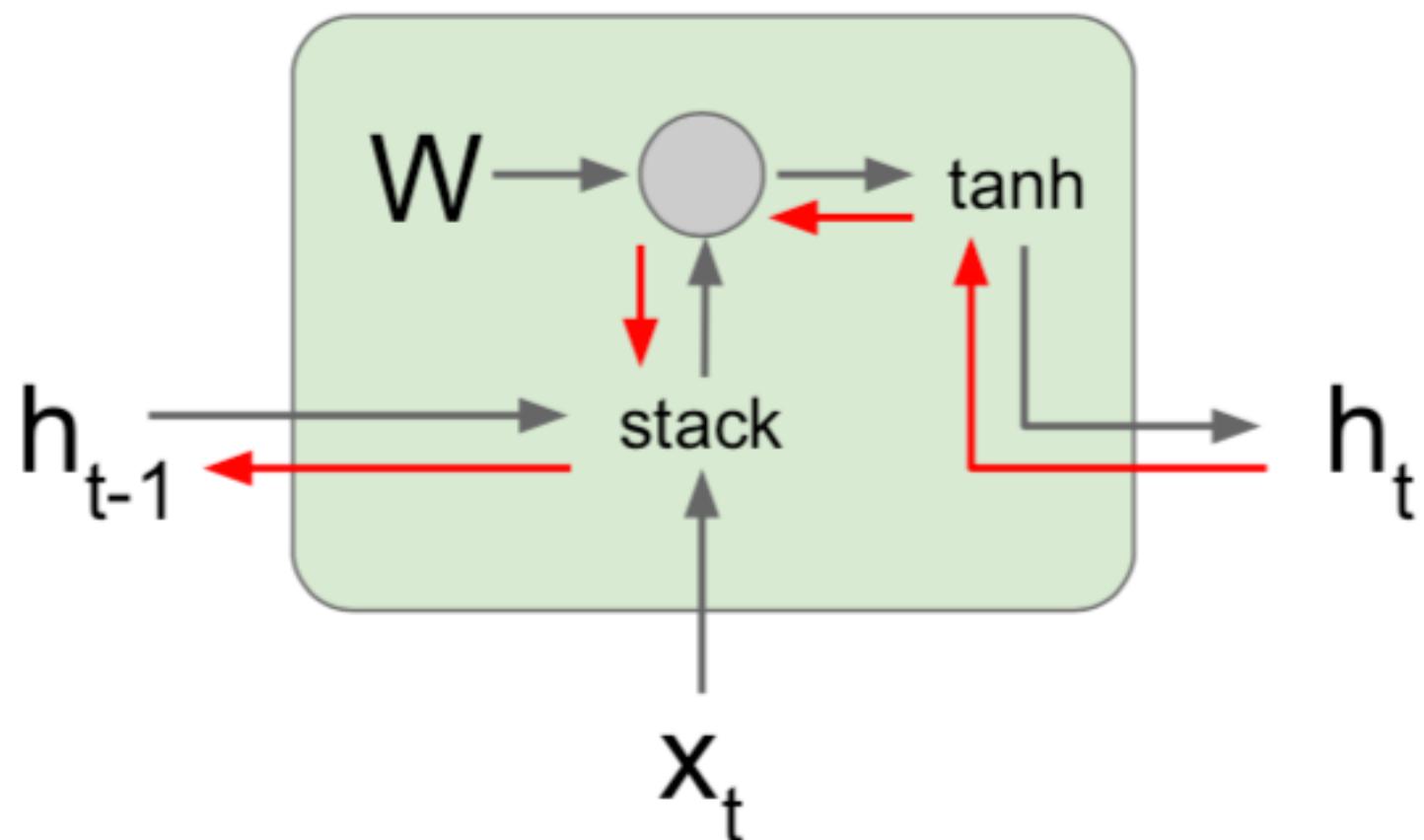
- sigmoid e tanh sono bounded e quindi non hanno problemi con l'esplosione del gradiente, ma come sappiamo soffrono del problema opposto(vanishing gradients)
- soluzione (parziali) al problema:
 - usare sigmoid e tanh, ma:
 - inizializzando i pesi a matrici identità per aiutare ad evitare che si riducano a zero durante il training
 - usando gated cell e tecniche di memorizzazione per controllare in modo selettivo il flusso di informazione durante il processamento della RNN, implementando una sorta di data-flow parallelo che ad ogni step t metta a disposizione di ogni layer i context vector prodotti negli step precedenti senza essere affatto dalla diluizione del gradiente

RNN VANISHING GRADIENT

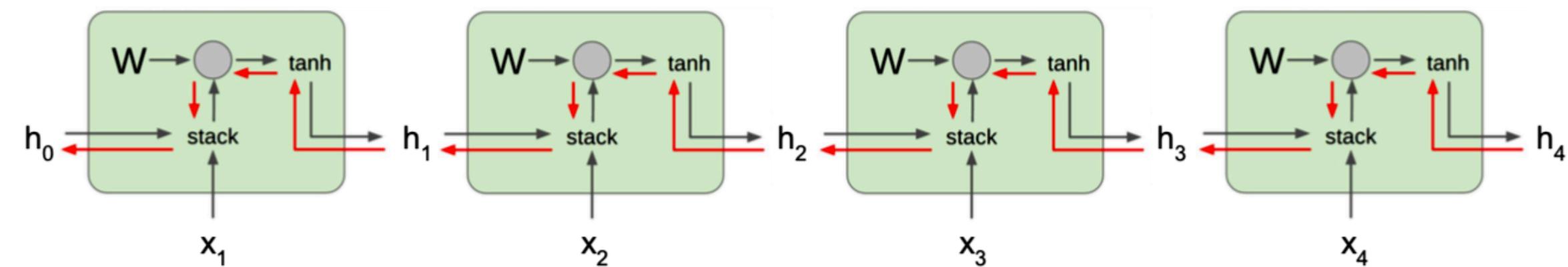


RNN: modulo elementare
1 semplice layer

gradient flow durante la back-propagation



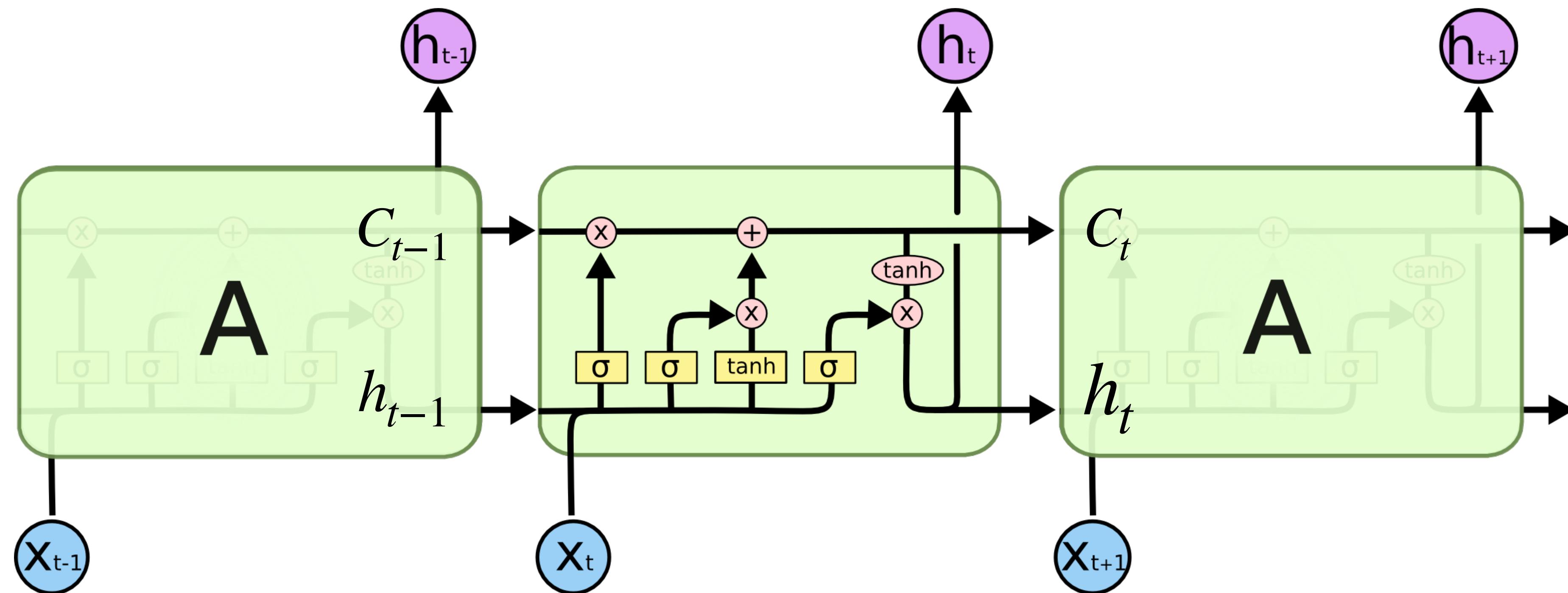
$$\begin{aligned}
 h_t &= \tanh(W_{hh}h_{t-1} + W_{xh}x_t) \\
 &= \tanh \left(\begin{pmatrix} W_{hh} & W_{hx} \end{pmatrix} \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} \right) \\
 &= \tanh \left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} \right)
 \end{aligned}$$



il calcolo del gradiente di h_0
coinvolge prodotti multipli di
 W e del gradiente di tanh (i.e.
prodotti di numeri piccoli)

LSTM: LONG SHORT TERM MEMORY RNN

le reti **LSTM** (Hochreiter, '97) risolvono il problema con un “trucco software”: invece di avere un singolo layer neurale, ne ha quattro, che interagiscono in modo tale da implementare una sorta di data-flow parallelo che ad ogni step t rende disponibili i dati precedenti ad ogni layer della rete

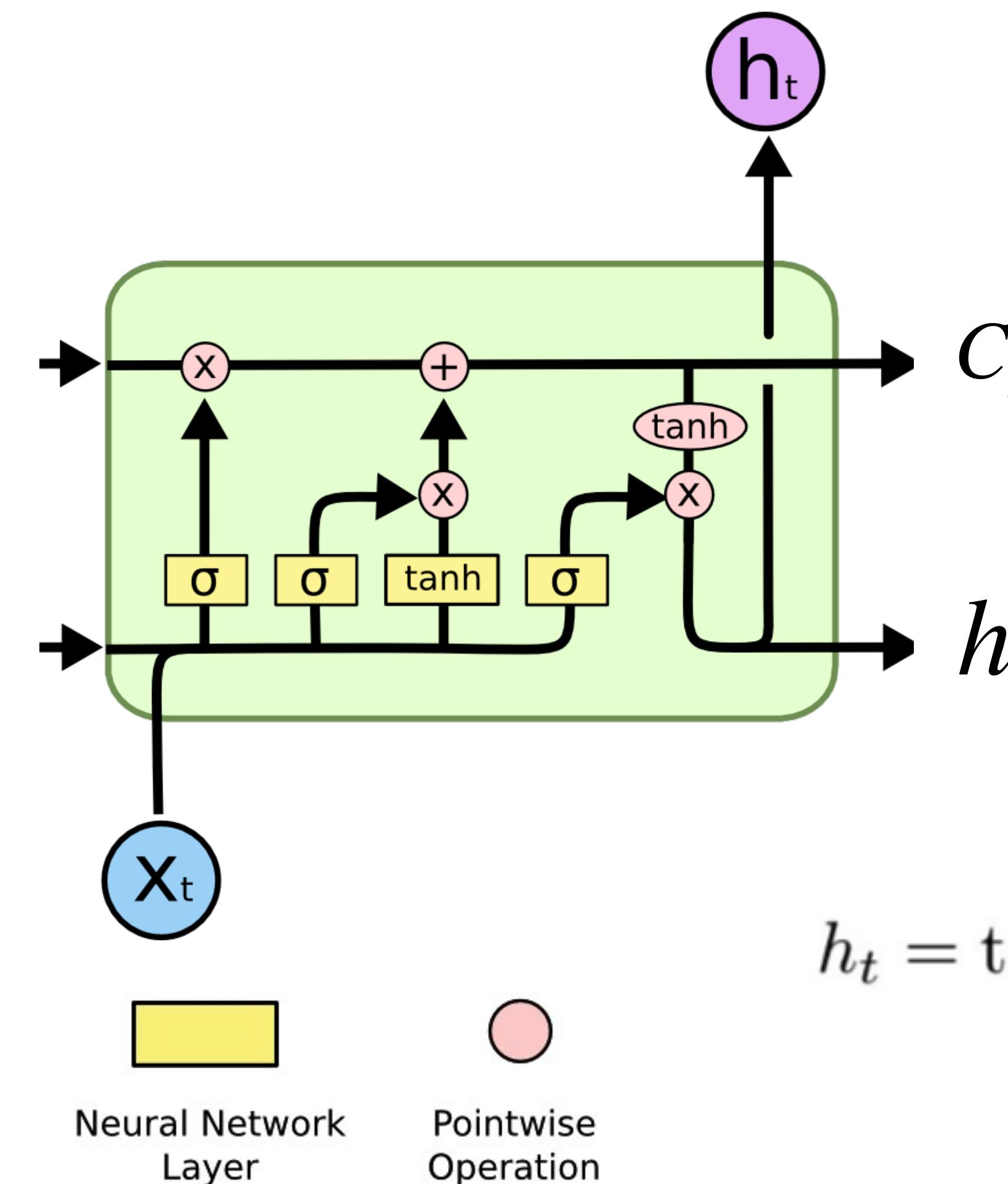


elemento chiave: **cell-state C_t**

è una unità di memoria (“conveyor belt”) alla quale è possibile aggiungere o sottrarre informazioni usando strutture dette “gate”

- il **cell state conserva memoria long-term**
- l'**hidden state conserva memoria short-term**

GATED LAYERS

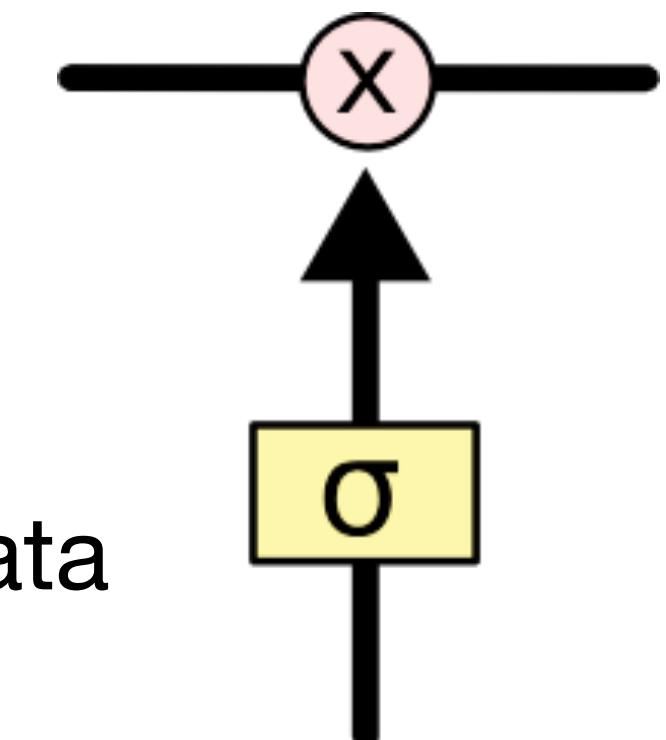


gate: NN-layer con attivazione sigmoide e una moltiplicazione point-wise

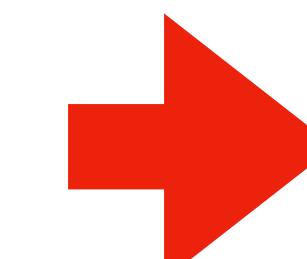
output $\in [0,1]$: permette di regolare quanta parte dell'informazione sulla cell-state debba essere lasciata passare

ogni LSTM ha 3 gate:

- f: forget gate (controlla cancellazione dal cell-state)
- i: input gate (controlla la scrittura sul cell-state)
- o: output gate (controlla l'output su h_t)



$$h_t = \tanh \left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} \right)$$



$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

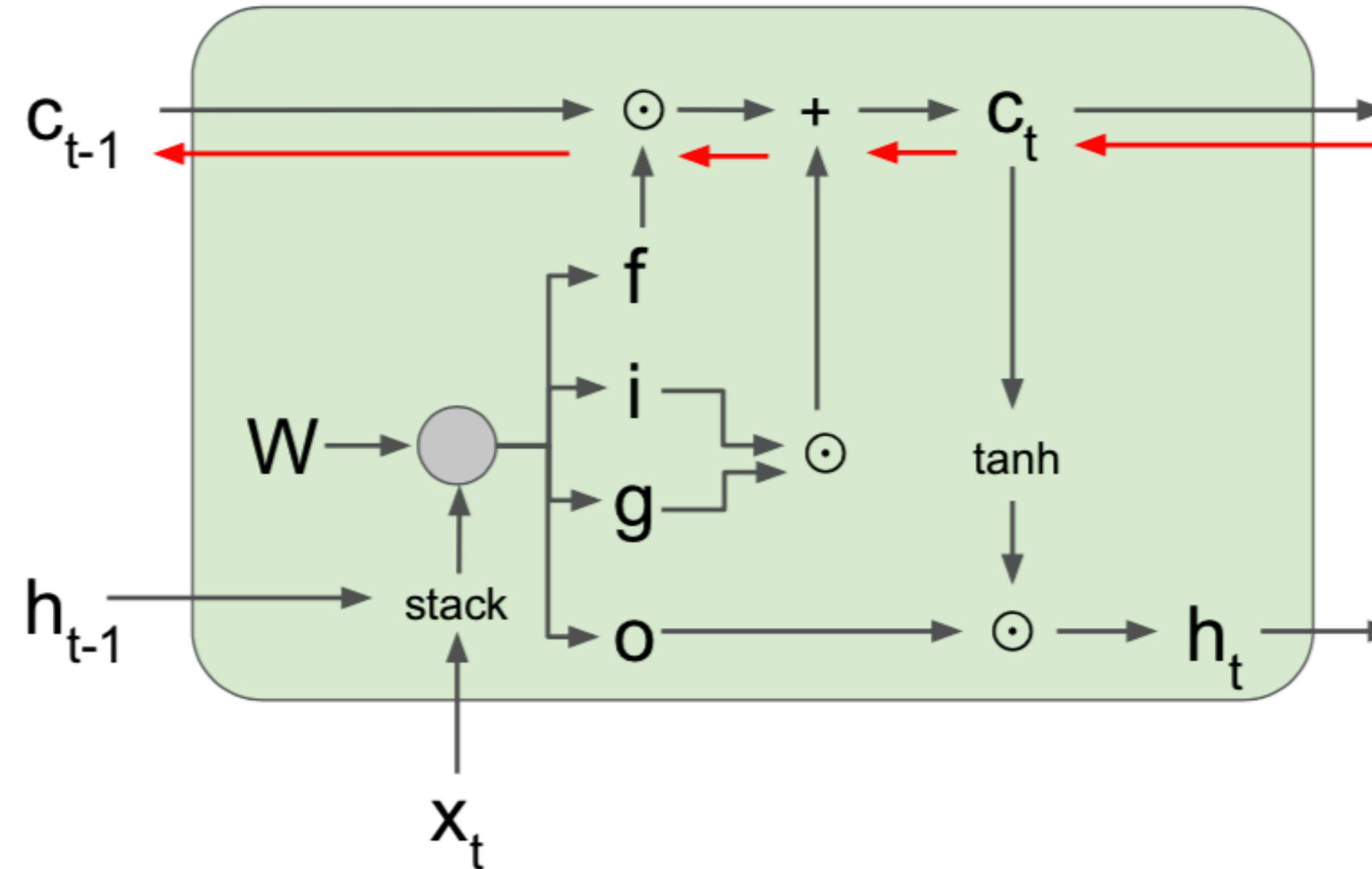
$\odot \doteq$ Hadamard product

$$(A \odot B)_{ij} = A_{ij}B_{ij}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

LSTM: GRADIENT FLOW

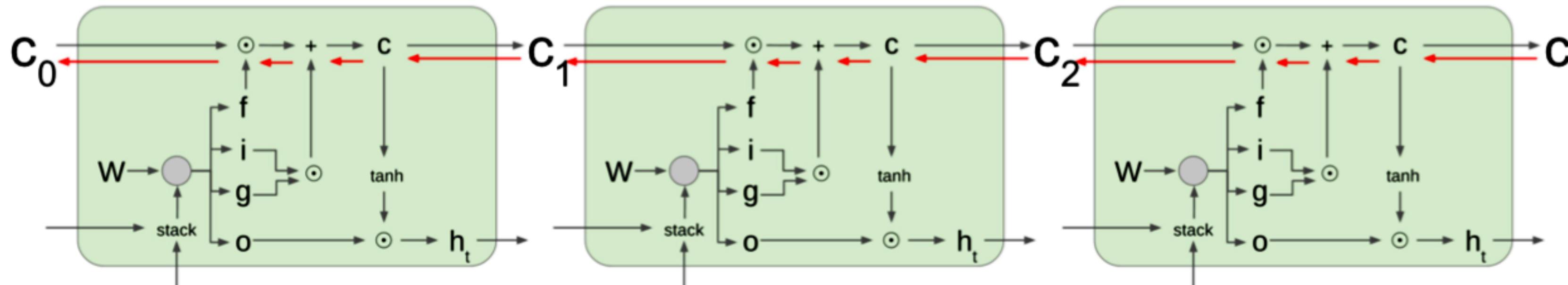


$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

si può dimostrare come in una LSTM la backprop da $C_t \rightarrow C_{t-1}$ richieda solo moltiplicazioni element-wise per f e non moltiplicazioni per W e tanh ...



si genera un flusso di gradiente continuo come in una CNN tipo ResNET

STEP1: FORGET GATE

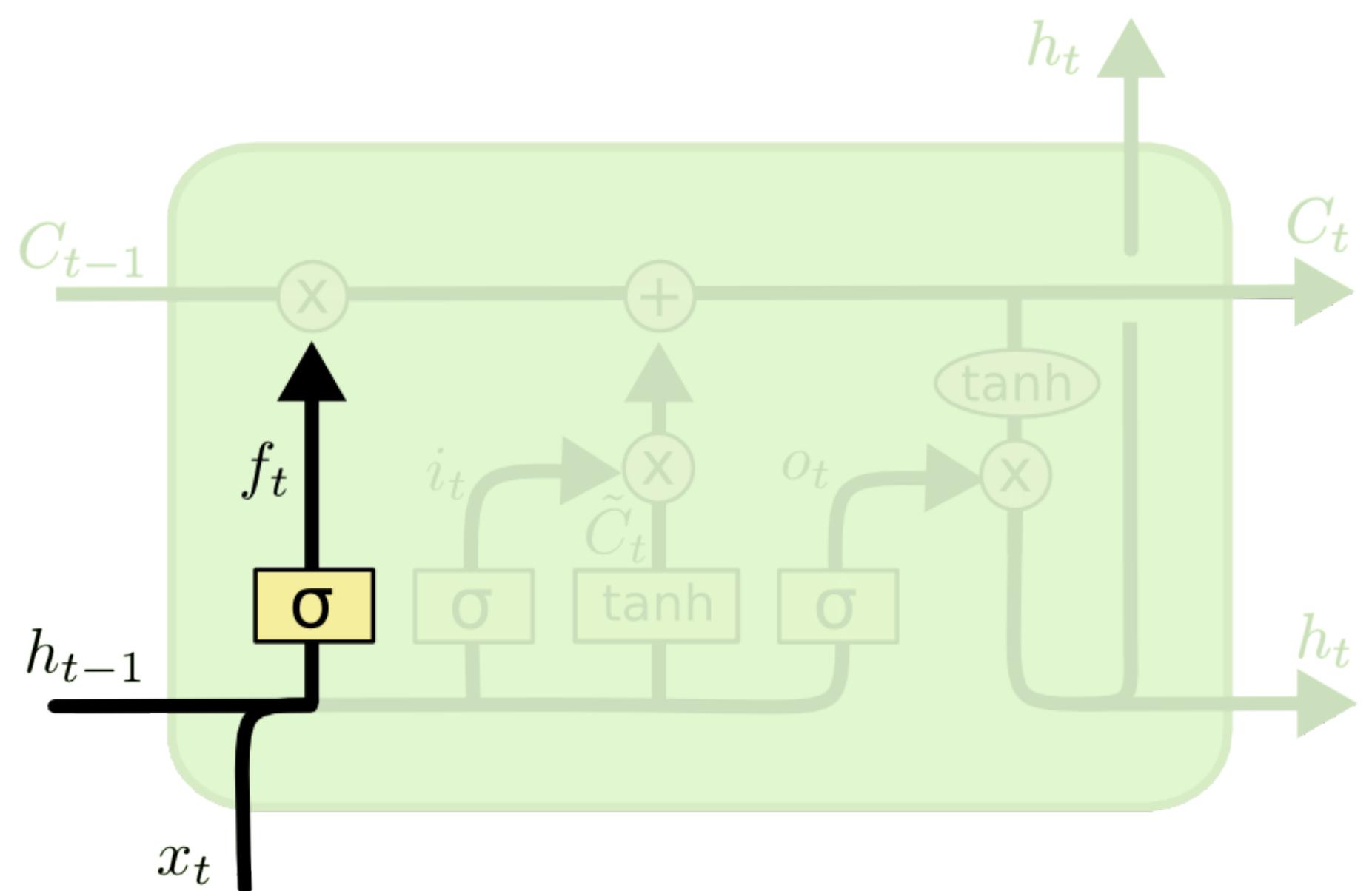
Decide quale parte dell'informazione precedente debba essere rimossa dal cell-state

guarda a h_{t-1} e a x_t e produce un numero in $[0,1]$ per ogni valore del cell-state C_{t-1}

- 0: elimina completamente l'informazione precedente
- 1: mantiene completamente l'informazione precedente

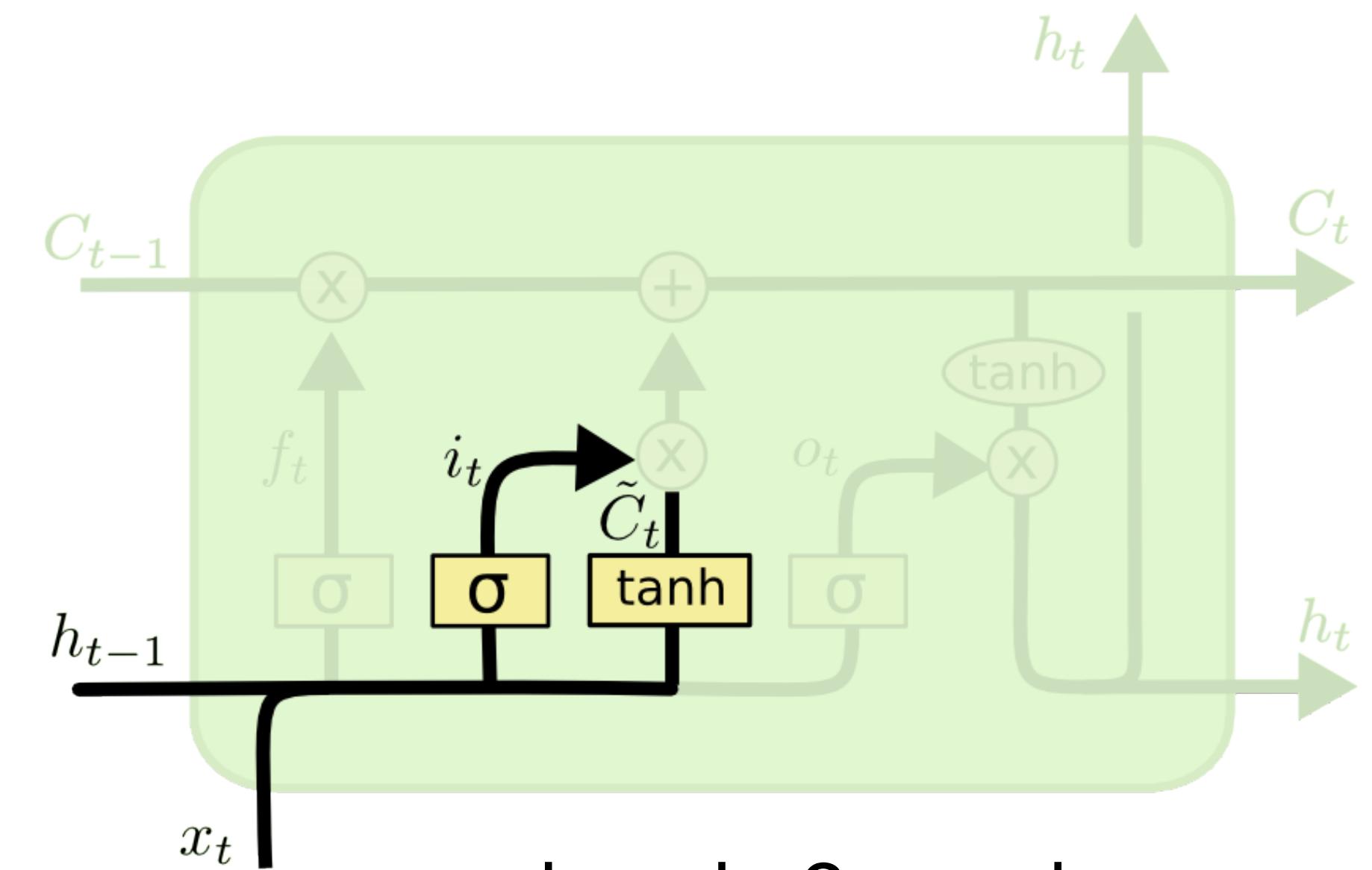
$$f_t = \sigma (W_f \cdot [h_{t-1}, x_t] + b_f)$$

weights



esempio: C_{t-1} potrebbe contenere il genere del soggetto corrente nella frase analizzata, quindi una informazione utile per predire il pronome appropriato. Se la rete riconosce nell'input x_t un nuovo soggetto, molto probabilmente vorrà dimenticare il genere associato al precedente soggetto ...

STEP 2: INPUT GATE



decide quale nuova informazione attaccare al cell-state

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

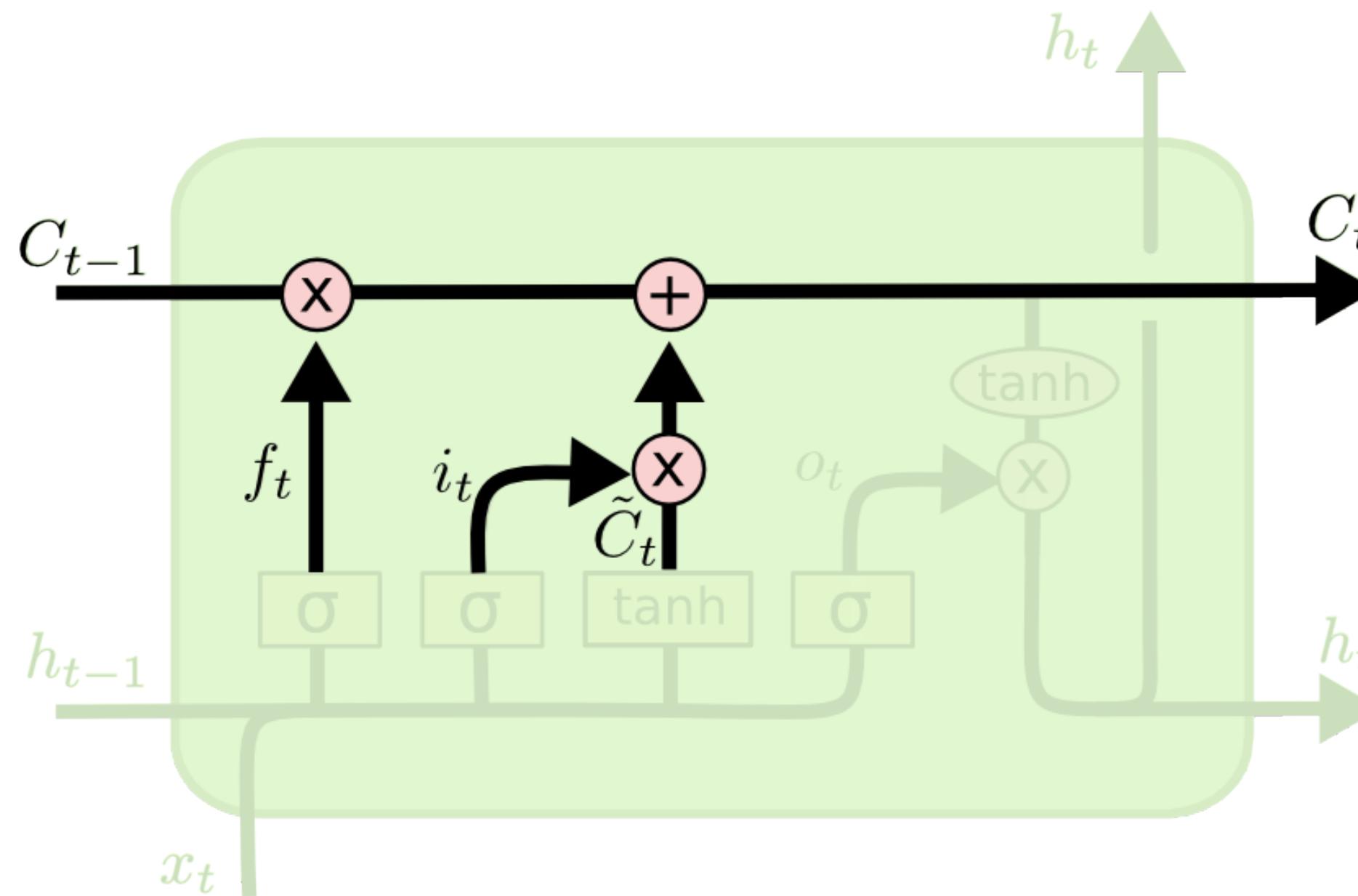
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

agisce in 2 passi:

- un σ -layer “input gate layer” decide quale valore aggiornare
- un layer tanh crea un nuovo vettore di candidati C_t che potenzialmente potrebbe essere aggiunto al cell-state

esempio: l’algoritmo vuole aggiungere il genere del nuovo soggetto sul cell-state per rimpiazzare il precedente ...

STEP 3: CELL-STATE UPDATE



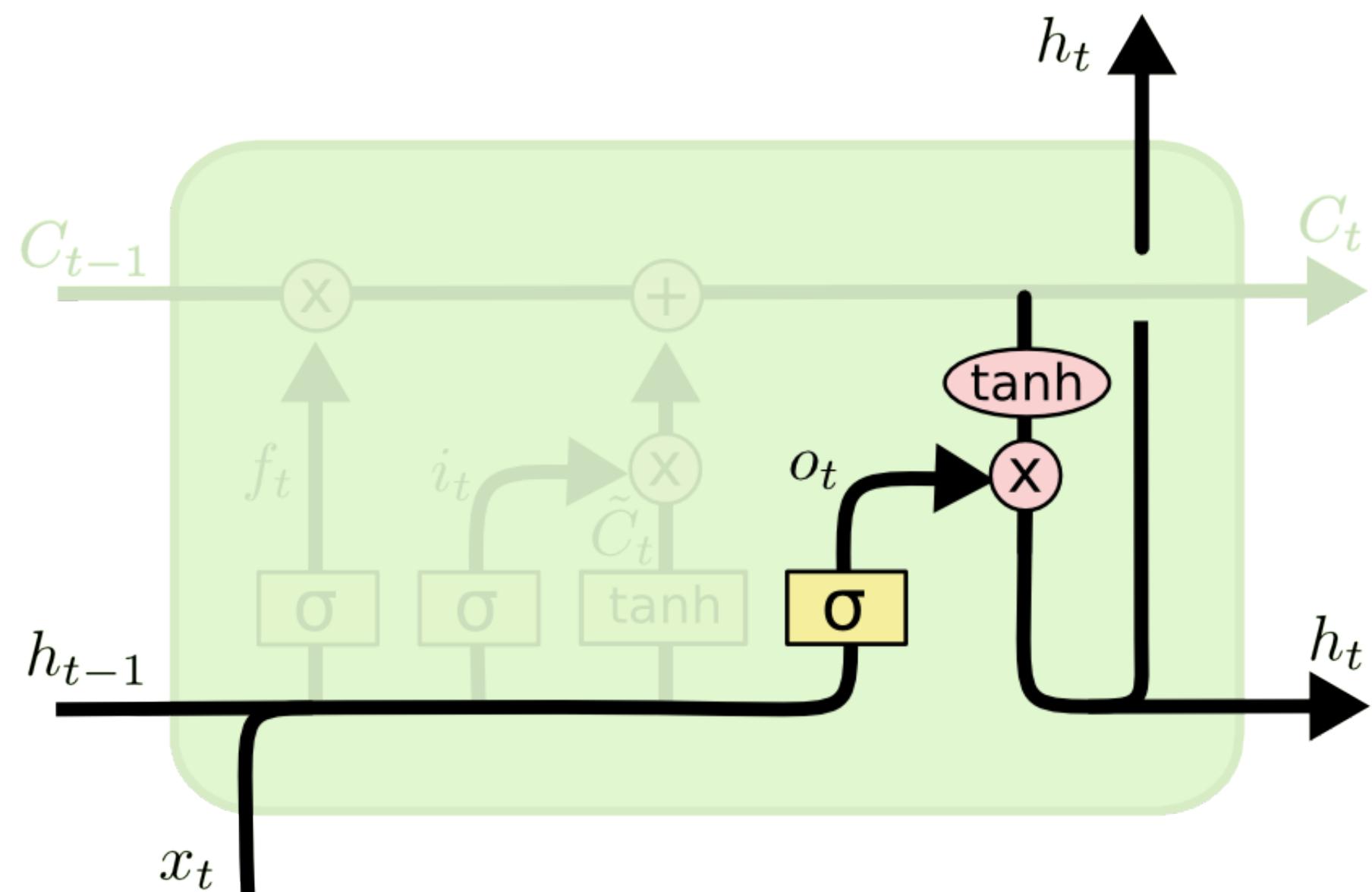
update $C_{t-1} \rightarrow C_t$

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

- lo stato precedente viene moltiplicato per f_t
- e viene aggiunto a $i_t \cdot \tilde{C}_t$

esempio: questo è lo step in cui l'algoritmo realmente elimina l'informazione di genere del precedente soggetto e la rimpiazza con la nuova ...

STEP 4: OUTPUT GATE



prepara ed emette h_t

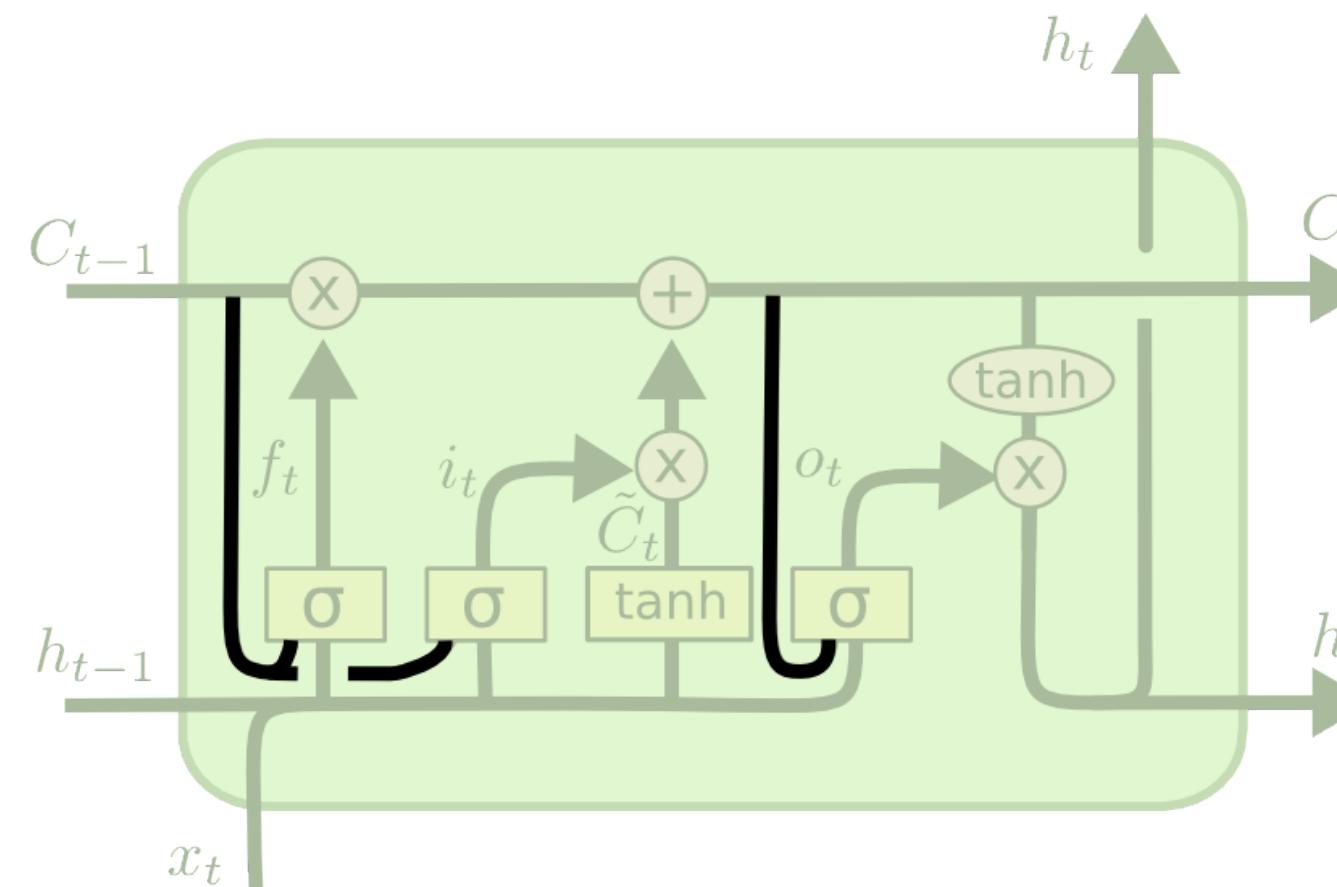
$$o_t = \sigma (W_o [h_{t-1}, x_t] + b_o)$$
$$h_t = o_t * \tanh (C_t)$$

è un gate basato su una versione filtrata del cell-state: prima un σ -layer decide quale parte del cell-state scrivere in output, poi il cell-state viene valutato tramite una tanh (compresso in $(-1,1)$) e moltiplicato per l'output del gate

esempio: poiché la rete ha identificato un soggetto, potrebbe voler produrre in uscita informazioni utili per poi associare il soggetto ad una forma verbale (per esempio se il soggetto è singolare o plurale), nel caso in cui la successiva parola nella sequenza fosse una forma verbale ...

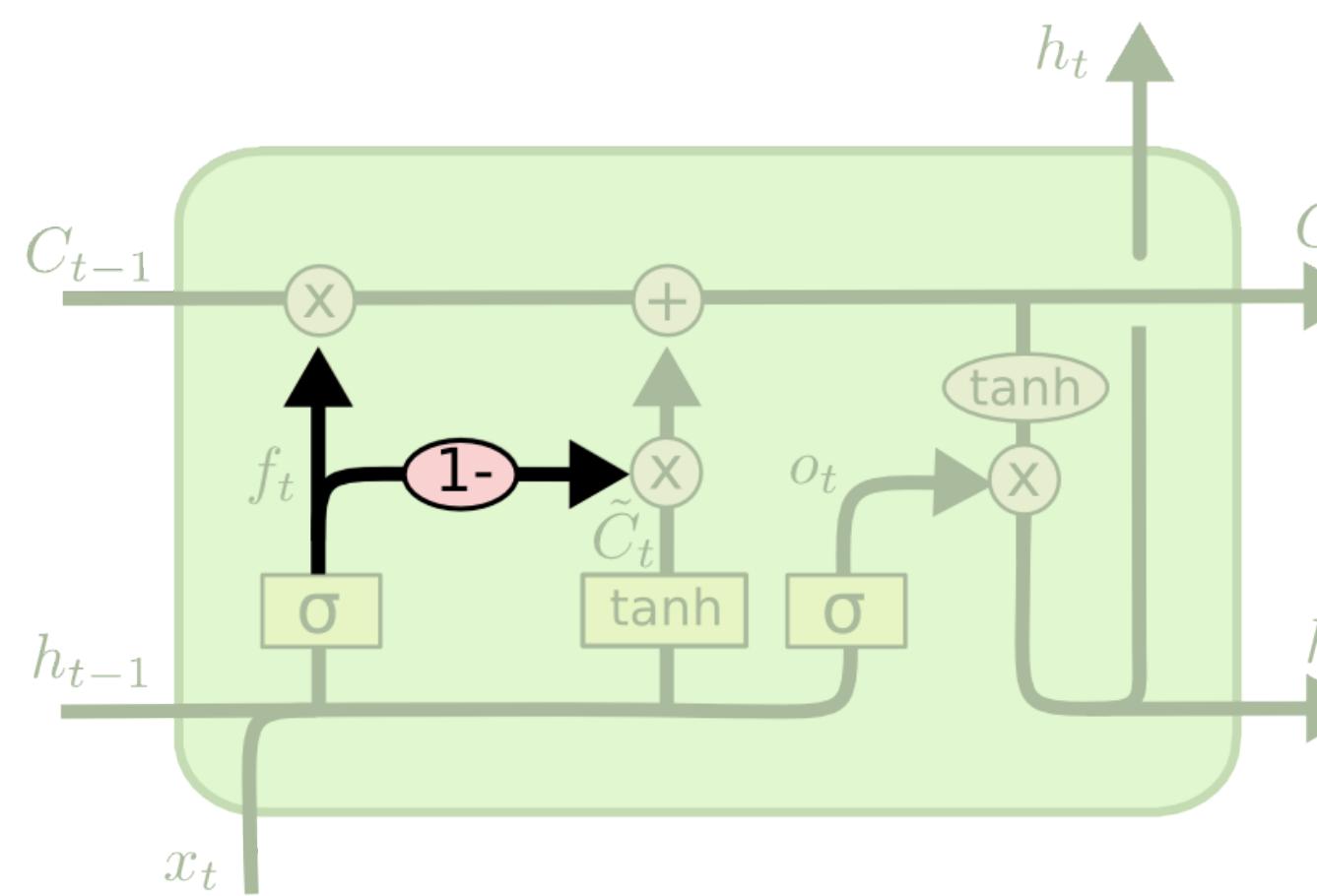
LSTM: VARIAZIONI ...

quasi ogni pubblicazione che usa LSTM ha implementato una versione leggermente differente dell'algoritmo per giustificare l'originalità ...



$$f_t = \sigma(W_f \cdot [C_{t-1}, h_{t-1}, x_t] + b_f)$$
$$i_t = \sigma(W_i \cdot [C_{t-1}, h_{t-1}, x_t] + b_i)$$
$$o_t = \sigma(W_o \cdot [C_t, h_{t-1}, x_t] + b_o)$$

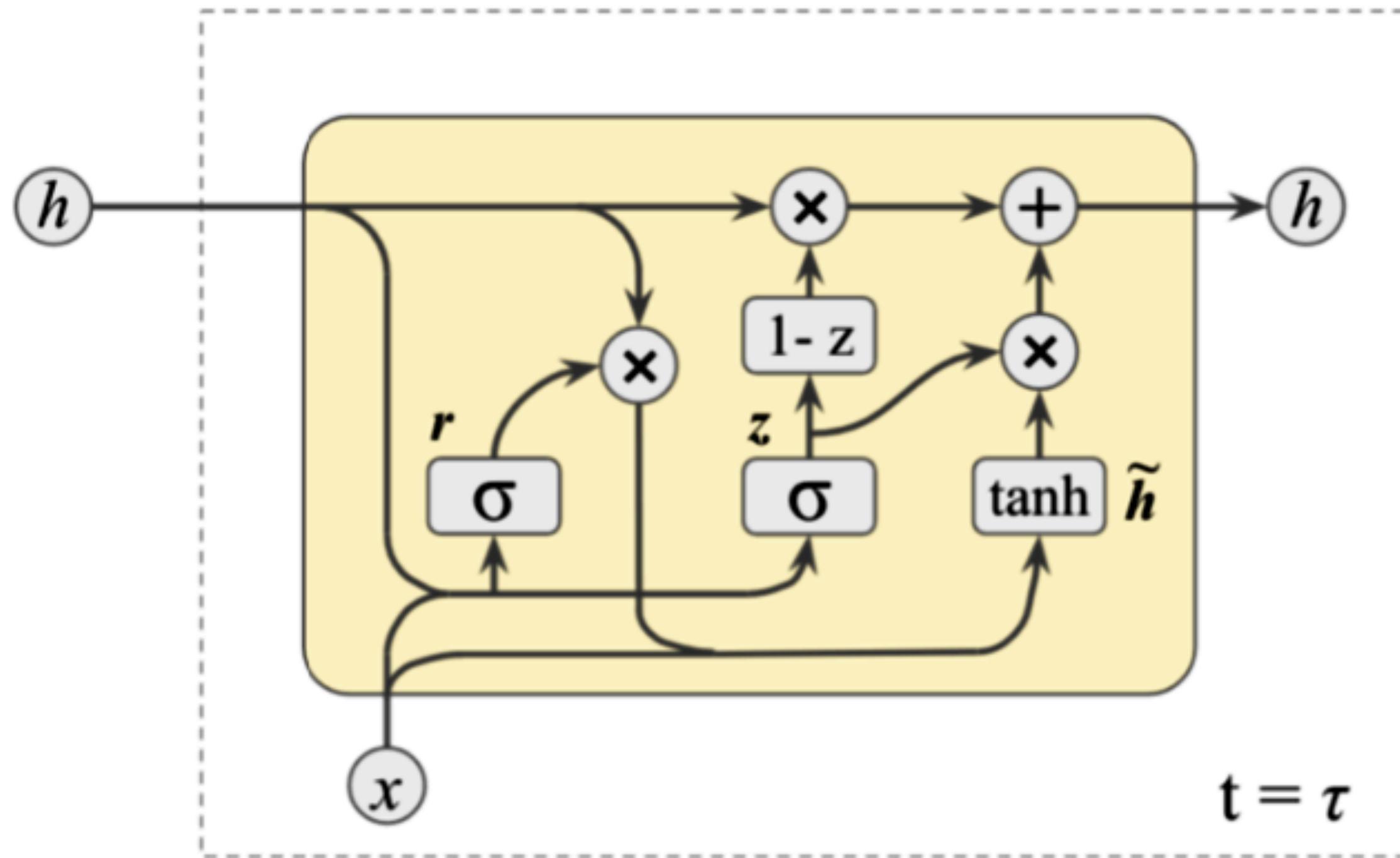
LSTM con “peephole”: i gate layers
possono vedere il cell-state



$$C_t = f_t * C_{t-1} + (1 - f_t) * \tilde{C}_t$$

LSTM con gate forget e input accoppiati

LSTM VARIAZIONI: GRU



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

with

$$r_t = \sigma(W^r x_t + V^r h_{t-1} + b^r)$$

$$z_t = \sigma(W^z x_t + V^z h_{t-1} + b^z)$$

$$\tilde{h}_t = \tanh\left(W^h x_t + V^h (r_t \odot h_{t-1}) + b^h\right)$$

GRU (Gate Recurrent Unit): combina i vettori cell state e hidden state, e combina 3 gate in due per semplificare il modello e il numero di parametri

Richiede minor potenza di calcolo ma mantiene prestazioni simili alla LSTM quando si analizzano lunghe sequenze (è uno dei layer RNN più utilizzati)

UN SEMPLICE CLASSIFICATORE LSTM IN PYTORCH

```
#Simple LSTM Classifier model

class MyLSTM(nn.Module):
    def __init__(self, vocab_size, embed_dim, rnn_hidden_size, fc_hidden_size):
        def __init__(self, input_size, hidden_size, fc_hiddn_size):
            super().__init__()
            self.rnn = nn.LSTM(input_size,
                               hidden_size,
                               num_layers=1,
                               batch_first=True)

            self.fc1 = nn.Linear(rnn_hidden_size, fc_hidden_size)
            self.relu = nn.ReLU()
            self.fc2 = nn.Linear(fc_hidden_size, 1)
            self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        out, (hidden, cell) = self.rnn(x)

        # pass to the dense layer only the last hidden state
        out = hidden[-1, :, :]
        out = self.fc1(out)
        out = self.relu(out)
        out = self.fc2(out)
        out = self.sigmoid(out)

        return out
```

N = batch size
 L = sequence length
 D = 2 if bidirectional=True otherwise
 H_{in} = input_size
 H_{out} = hidden_size

- **input_size** – The number of expected features in the input x
- **hidden_size** – The number of features in the hidden state h
- **num_layers** – Number of recurrent layers. E.g., setting `num_layers=2` would mean stacking two RNNs together to form a *stacked RNN*, with the second RNN taking in outputs of the first RNN and computing the final results. Default: 1
- **batch_first** – If `True`, then the input and output tensors are provided as $(batch, seq, feature)$ instead of $(seq, batch, feature)$. Note that this does not apply to hidden or cell states. See the Inputs/Outputs sections below for details. Default: `False`
- **input**: tensor of shape (L, H_{in}) for unbatched input, (L, N, H_{in}) when `batch_first=False` or (N, L, H_{in}) when `batch_first=True` containing the features of the input sequence. The input can also be a packed variable length sequence. See `torch.nn.utils.rnn.pack_padded_sequence()` or `torch.nn.utils.rnn.pack_sequence()` for details.
- **h_0**: tensor of shape $(D * num_layers, H_{out})$ for unbatched input or $(D * num_layers, N, H_{out})$ containing the initial hidden state for the input sequence batch. Defaults to zeros if not provided.
- **output**: tensor of shape $(L, D * H_{out})$ for unbatched input, $(L, N, D * H_{out})$ when `batch_first=False` or $(N, L, D * H_{out})$ when `batch_first=True` containing the output features (h_t) from the last layer of the RNN, for each t . If a `torch.nn.utils.rnn.PackedSequence` has been given as the input, the output will also be a packed sequence.
- **h_n**: tensor of shape $(D * num_layers, H_{out})$ for unbatched input or $(D * num_layers, N, H_{out})$ containing the final hidden state for each element in the batch.