Reti Neurali ricorrenti (RNN) e Attenzione

CORSO DI NATURAL LANGUAGE PROCESSING (ELABORAZIONE DEL LINGUAGGIO NATURALE)

a.a. 2025/2026

Prof. Roberto Pirrone

La natura del linguaggio

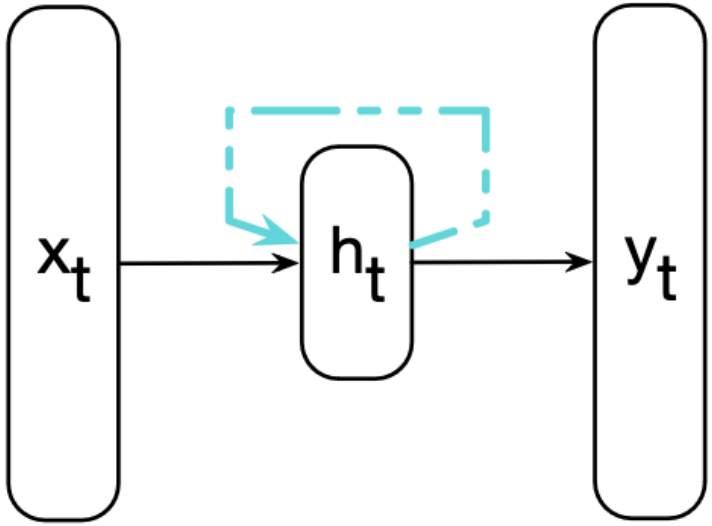
- Il linguaggio è un fenomeno intrinsecamente temporale
  - Il Linguaggio parlato
  - Il flusso della conversazione
  - Flusso di X (precedentemente Twitter)
- Abbiamo già usato Language Models che trattano sequenze, assumendo quindi questo aspetto temporale
  - *N-grammi e HMM*

La natura del linguaggio

- I ***neural language models*** usano o *sliding windows* di dimensione fissa o il *padding* dell'input fino alla frase più lunga
  - Vedremo di nuovo il padding nei Transformers
- Le *Recurrent Neural Networks* (Reti neurali Ricorrenti – RNN) trattano direttamente la natura sequenziale del linguaggio
  - Le *connessioni ricorrenti* permettono alla decisione del modello di dipendere da informazioni provenienti da centinaia di parole nel passato

Elman Network (1990)

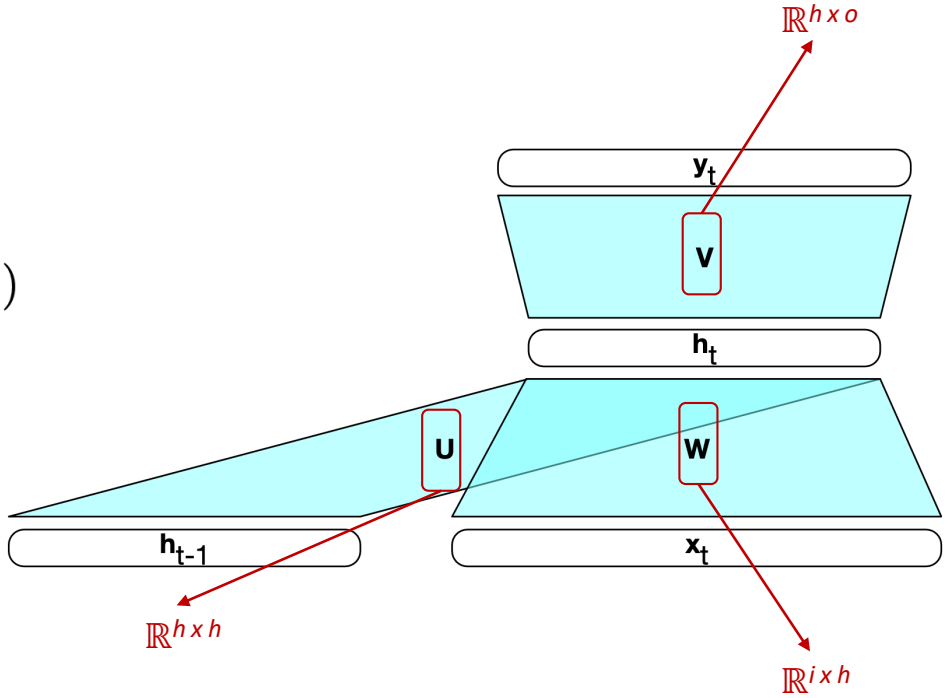
- Lo stato della rete è rappresentato al tempo **t**
- L'input dell'**hidden layer** è aumentato con *l'output al punto temporale precedente* $\boldsymbol{h_{t-1}}$



Inferenza nelle RNN

$$\mathbf{h}_t = g(\mathbf{U}\mathbf{h}_{t-1} + \mathbf{W}\mathbf{x}_t)$$
$$\mathbf{y}_t = f(\mathbf{V}\mathbf{h}_t)$$

Usiamo la softmax per la classificazione



Inferenza nelle RNN

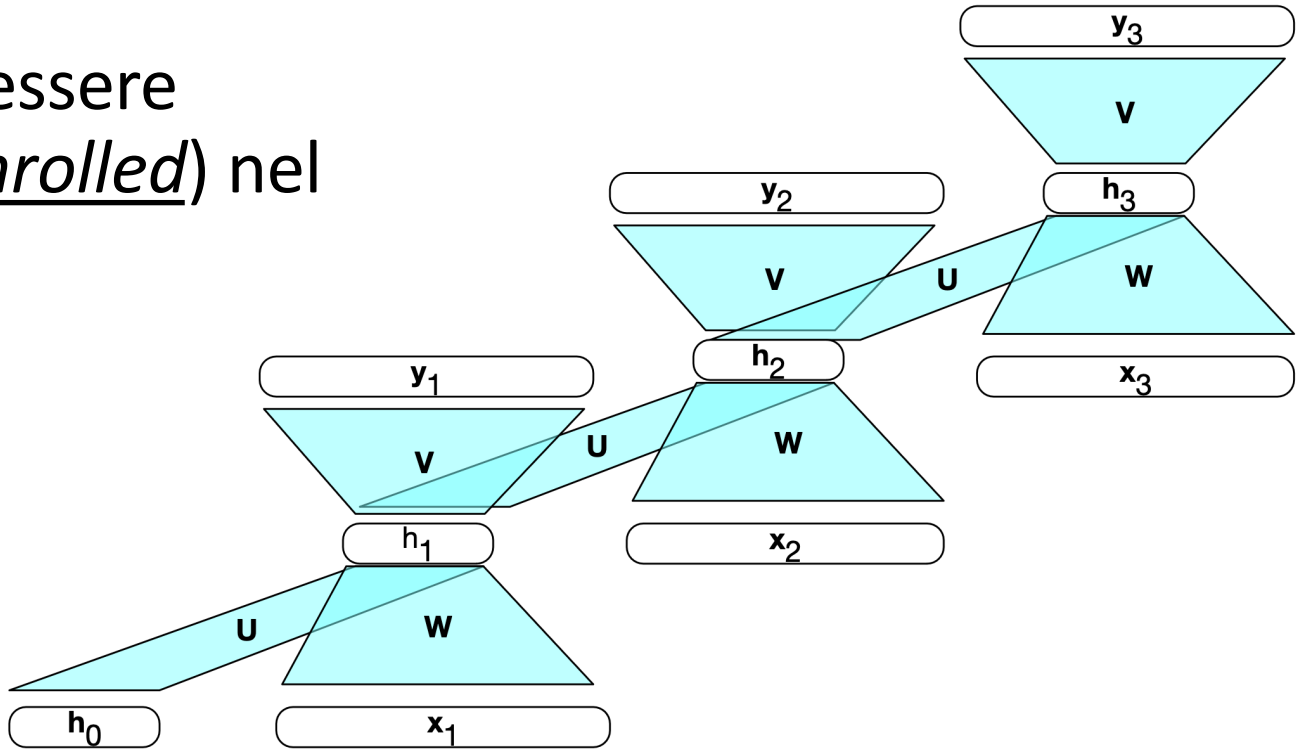
La computazione itera attraverso la sequenza di input

function FORWARDRNN($\mathbf{x}$, *network*) **returns** output sequence $\mathbf{y}$

$\mathbf{h}_0 \leftarrow 0$
for $i \leftarrow 1$ **to** LENGTH($\mathbf{x}$) **do**
  $\mathbf{h}_i \leftarrow g(\mathbf{U}\mathbf{h}_{i-1} + \mathbf{W}\mathbf{x}_i)$
  $\mathbf{y}_i \leftarrow f(\mathbf{V}\mathbf{h}_i)$
return y

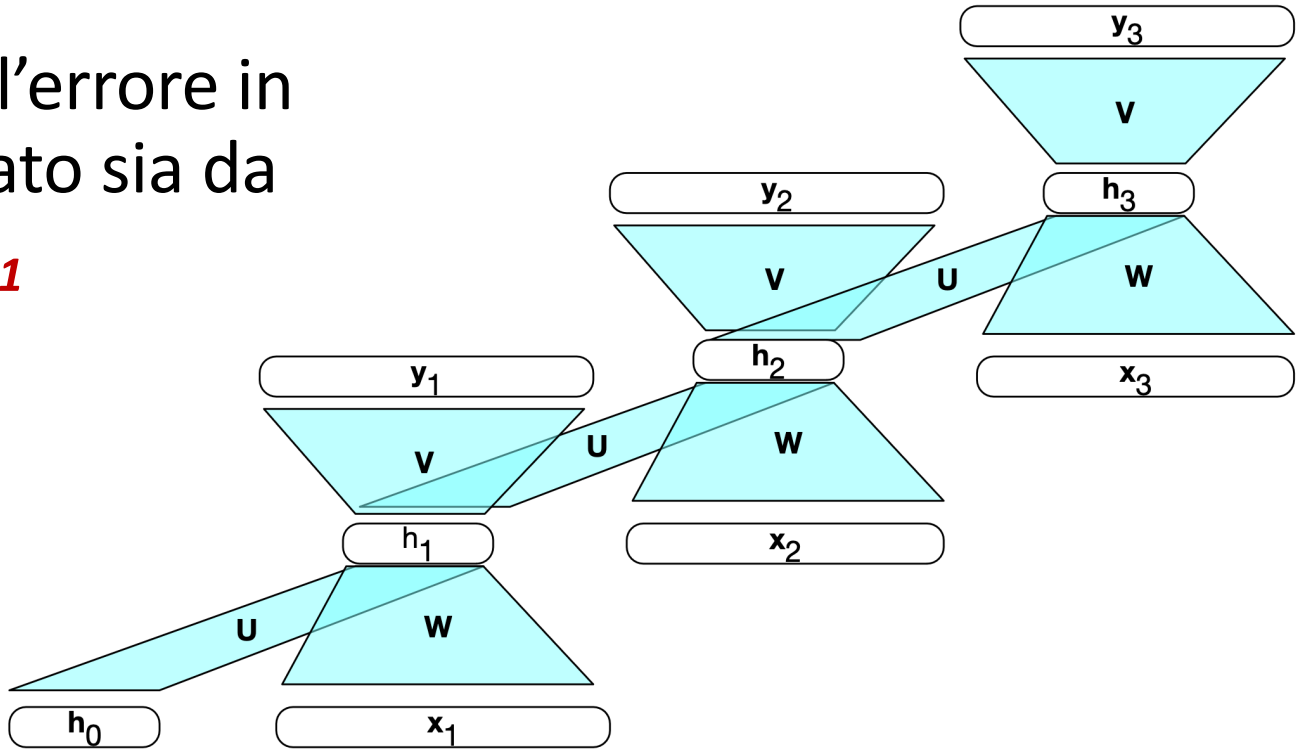
Inferenza nelle RNN

La rete può essere *srotolata* (*unrolled*) nel tempo



Training delle RNN

Il calcolo dell'errore in h_i è influenzato sia da h_{i-1} sia da h_{i+1}

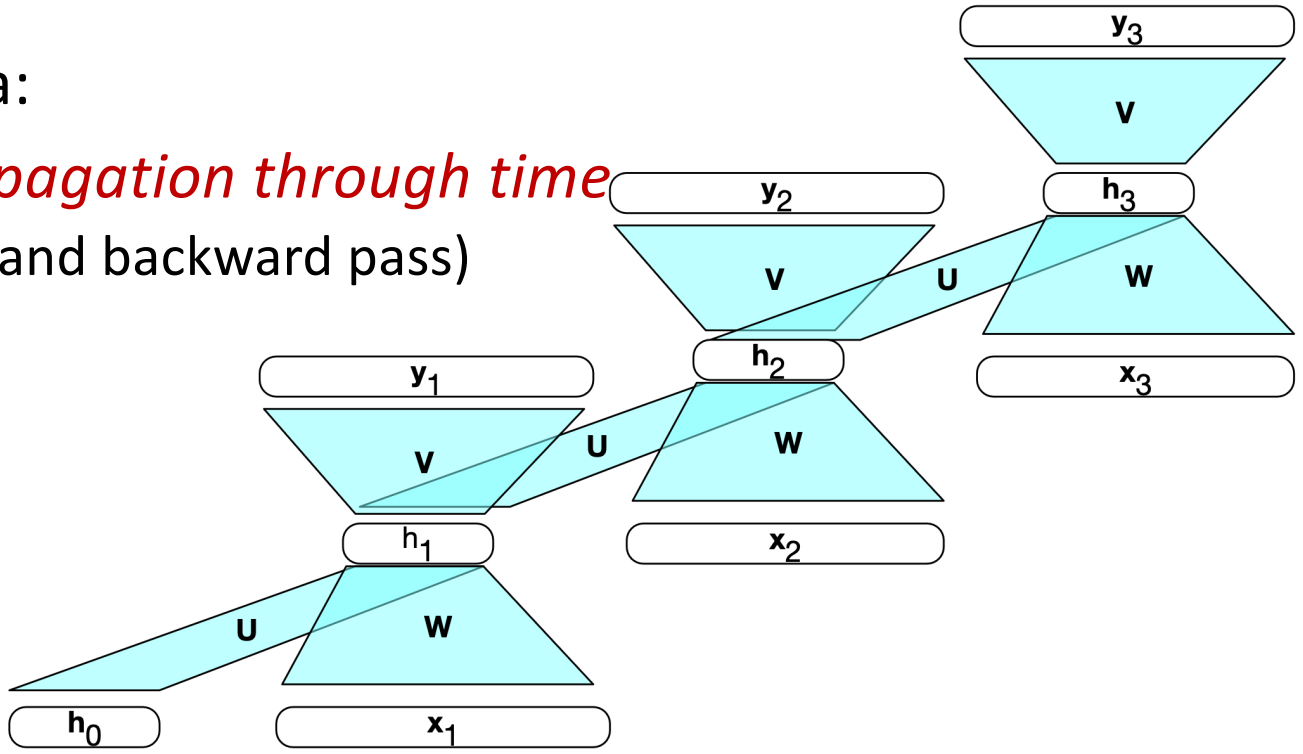


Training delle RNN

Vecchia scuola:

Backpropagation through time

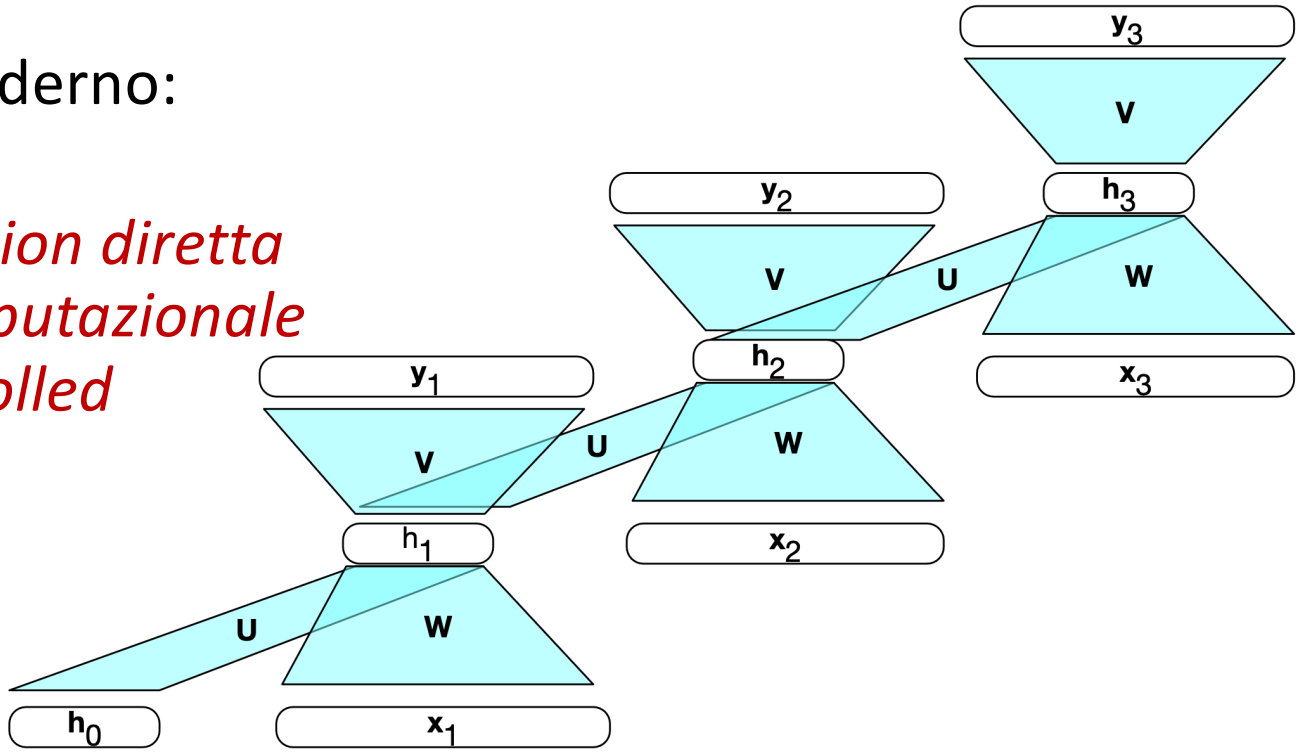
(Forward and backward pass)



Training delle RNN

Approccio moderno:

Backpropagation diretta sul grafo computazionale della rete unrolled



RNN come Language Model

Richiamo della definizione di language modeling

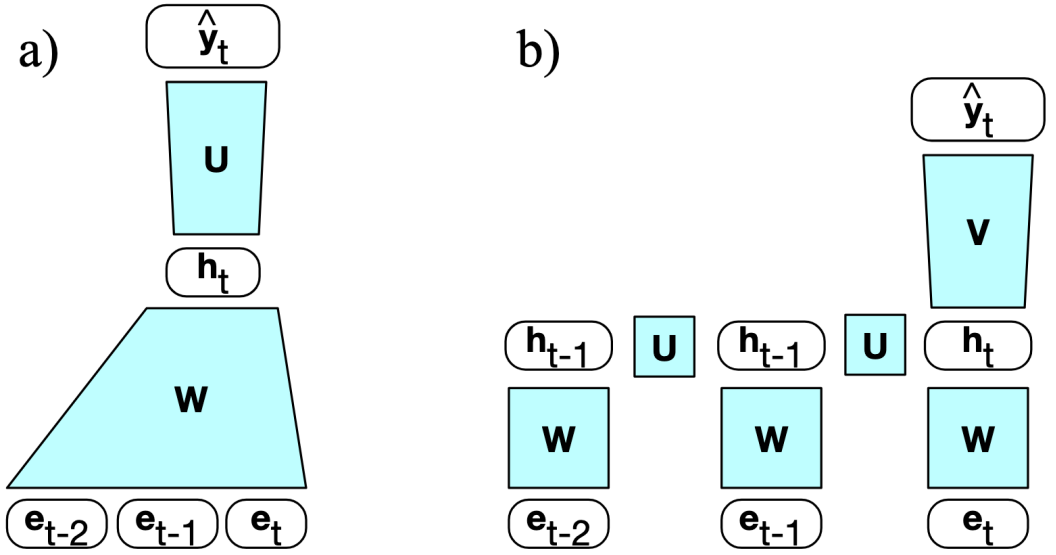
- *Language modeling:*
  - Predire quanto è probabile una parola data una sequenza di parole osservata $P(fish|Thanks\ for\ all\ the)$

- Possiamo anche predire la probabilità di un'intera sequenza

$$P(w_{1:n}) = \prod_{i=1}^{n} P(w_i|w_{<i})$$

RNN e contesto

- Le RNN non hanno problemi nel trattare un contesto illimitato
- L'hidden layer tiene conto del contesto passato



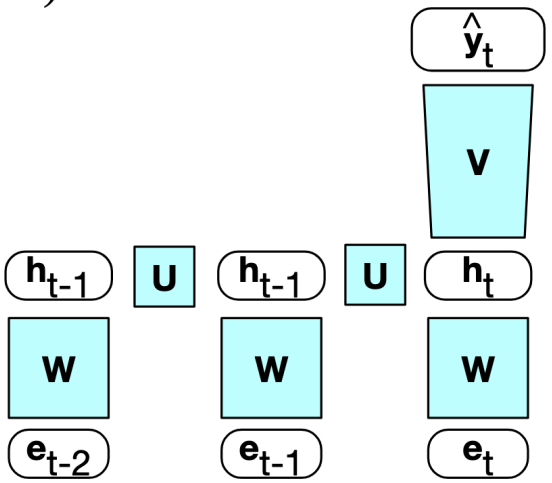
Embeddings

- Usiamo anche la matrice di **word embeddings E**

$$\mathbf{e}_t = \mathbf{E}\mathbf{x}_t$$
$$\mathbf{h}_t = g(\mathbf{U}\mathbf{h}_{t-1} + \mathbf{W}\mathbf{e}_t)$$
$$\mathbf{y}_t = \text{softmax}(\mathbf{V}\mathbf{h}_t)$$

In generale E non è addestrata e l'embedding layer è congelato (frozen), ad esempio a una rappresentazione word2vec

b)



Inferenza forward

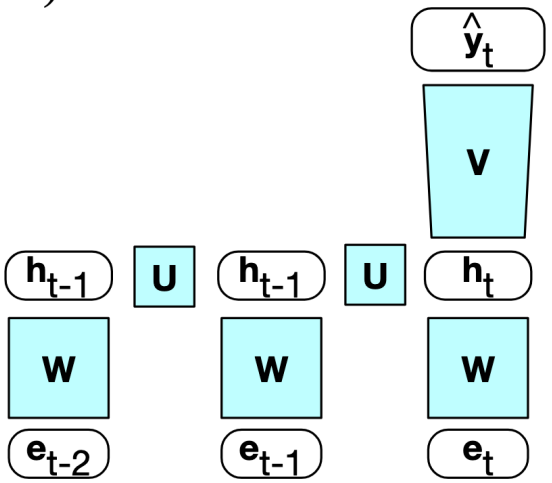
Inferenza

$$P(w_{t+1} = i | w_1, \ldots, w_t) = \mathbf{y}_t[i]$$

$$P(w_{1:n}) = \prod_{i=1}^{n} P(w_i | w_{1:i-1})$$

$$= \prod_{i=1}^{n} \mathbf{y}_i[w_i]$$

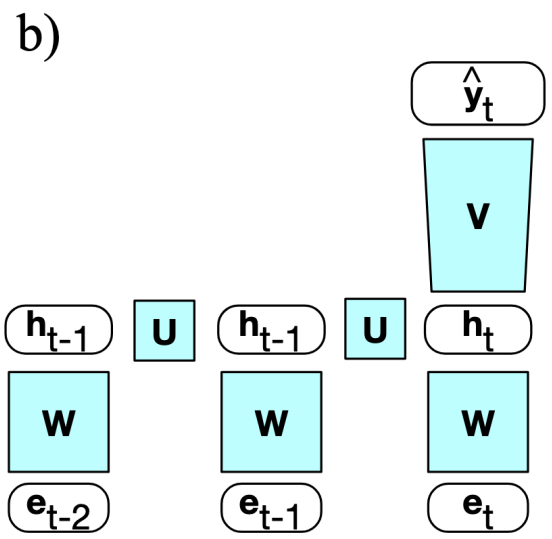
b)



Addestramento

Useremo di nuovo la *self-supervision*:

la parola corretta da predire è la successiva

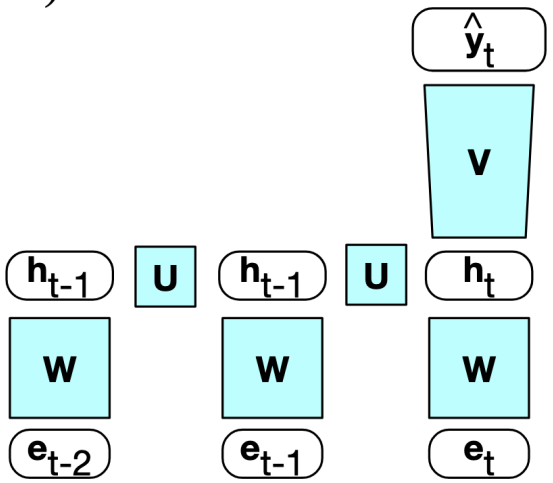


Training

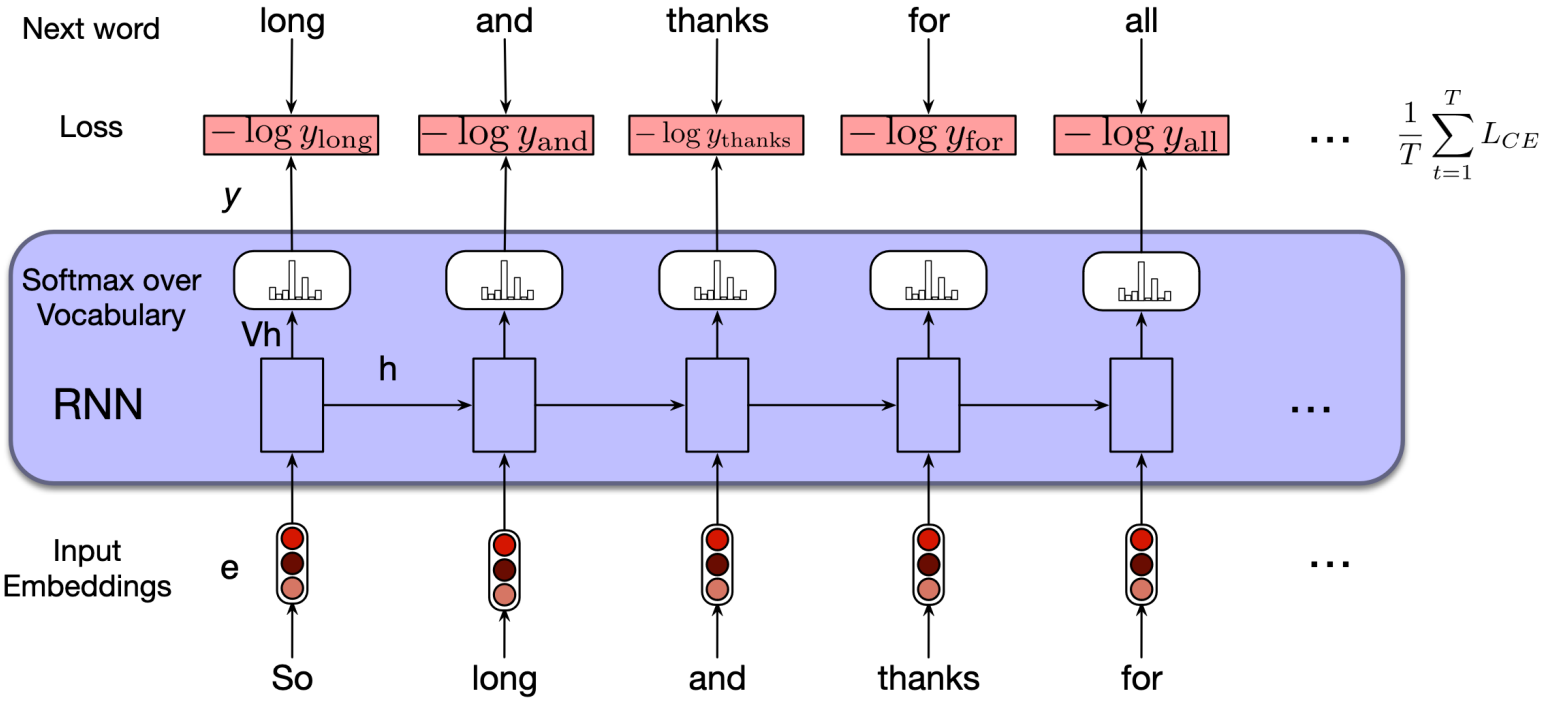
Loss: $$L_{CE} = -\sum_{w \in V} \mathbf{y}_t[w] \log \hat{\mathbf{y}}_t[w]$$

Cioè: $$L_{CE}(\hat{\mathbf{y}}_t, \mathbf{y}_t) = -\log \hat{\mathbf{y}}_t[w_{t+1}]$$

b)



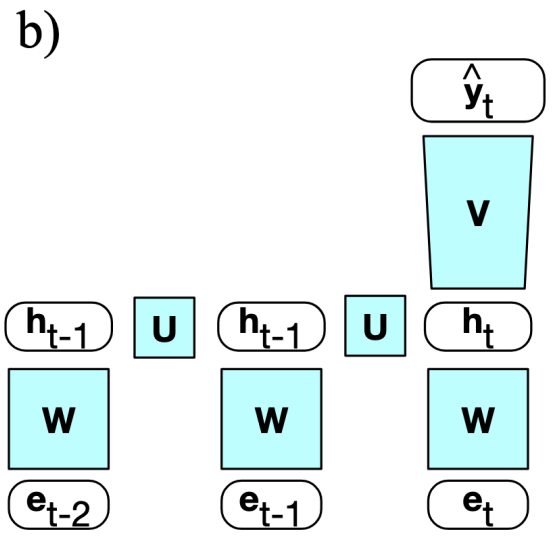
Teacher forcing



Ad ogni passo usiamo sempre la sequenza corretta per predire la parola successiva

Weight tying

- **E** è la matrice d_h x $|V|$ dei word embedding appresi
- **V** è la matrice $|V|$ x d_h degli score delle probabilità delle parole, data l'evidenza di ogni parola memorizzata in **h**
- *Sono davvero diverse??*

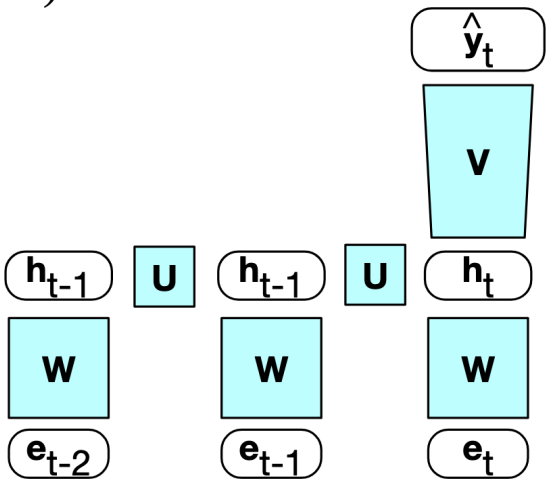


Weight tying

$$\mathbf{e}_t = \mathbf{E}\mathbf{x}_t$$
$$\mathbf{h}_t = g(\mathbf{U}\mathbf{h}_{t-1} + \mathbf{W}\mathbf{e}_t)$$
$$\mathbf{y}_t = \text{softmax}(\mathbf{V}\mathbf{h}_t)$$

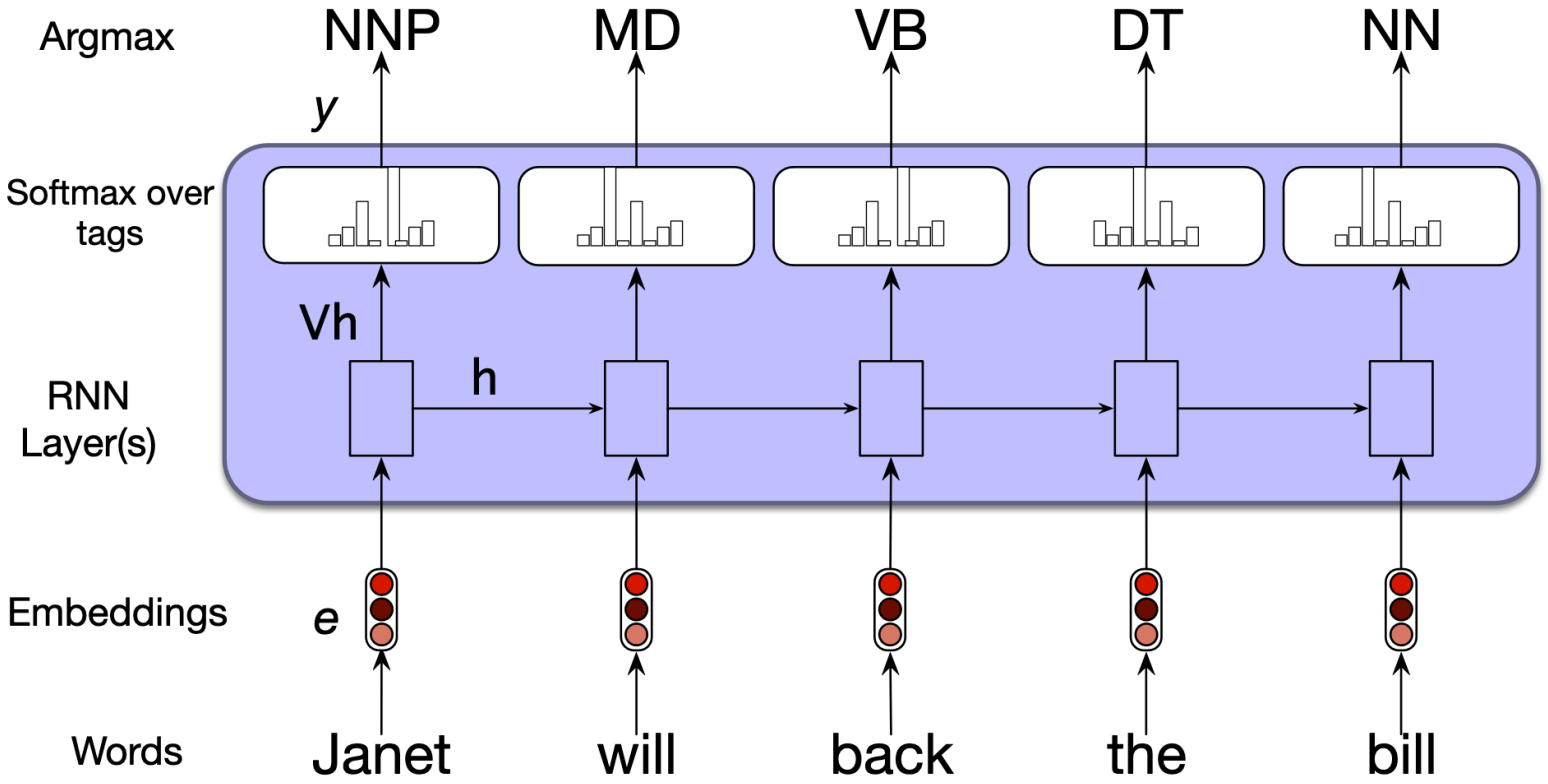
$$\mathbf{e}_t = \mathbf{E}\mathbf{x}_t$$
$$\mathbf{h}_t = g(\mathbf{U}\mathbf{h}_{t-1} + \mathbf{W}\mathbf{e}_t)$$
$$\mathbf{y}_t = \text{softmax}(\mathbf{E}^\intercal\mathbf{h}_t)$$

b)



RNN per altri task NLP

Sequence labeling



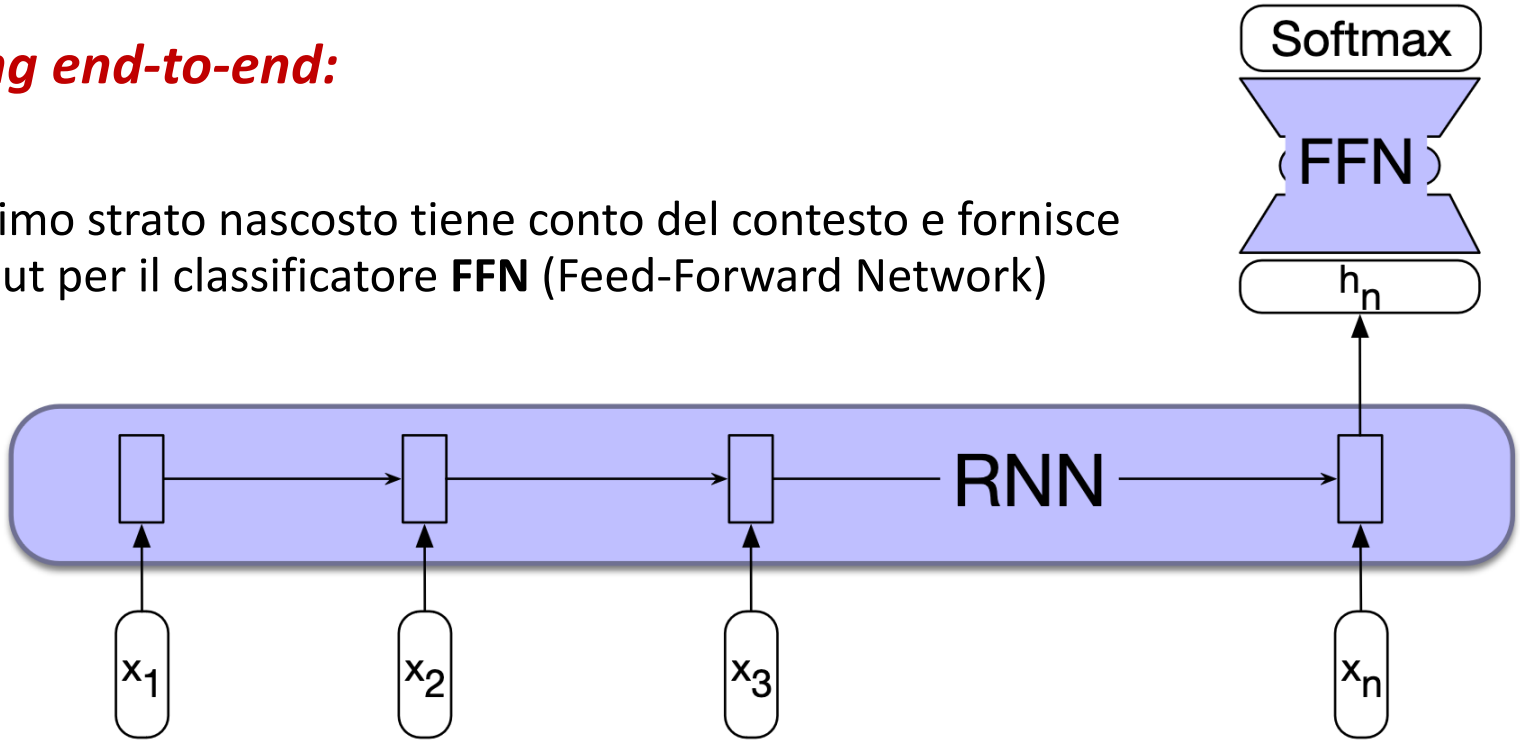
Sequence classification

- Classificare *intere sequenze* piuttosto che i token al loro interno

- Chiamata anche ***text classification***
  - Sentiment analysis
  - Rilevamento di spam
  - Rilevamento di discorsi d'odio/genere/politici
  - Classificazione di topic a livello di documento
  - …

Sequence classification

- ***Training end-to-end:***
  - L'ultimo strato nascosto tiene conto del contesto e fornisce l'input per il classificatore **FFN** (Feed-Forward Network)

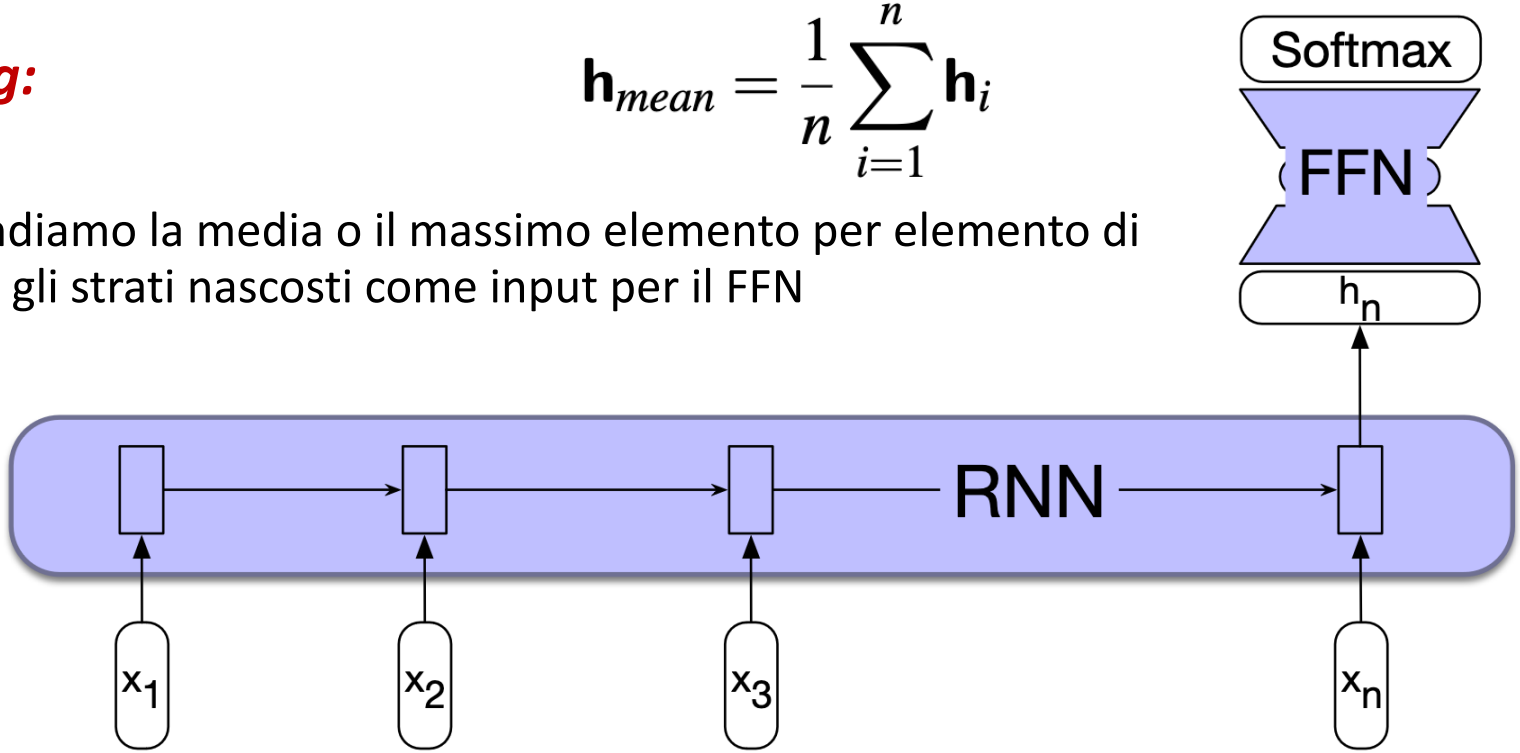


Sequence classification

- ***Pooling:***

$$\mathbf{h}_{mean} = \frac{1}{n}\sum_{i=1}^{n}\mathbf{h}_i$$

  - Prendiamo la media o il massimo elemento per elemento di tutti gli strati nascosti come input per il FFN



Generazione di testo (Text generation)

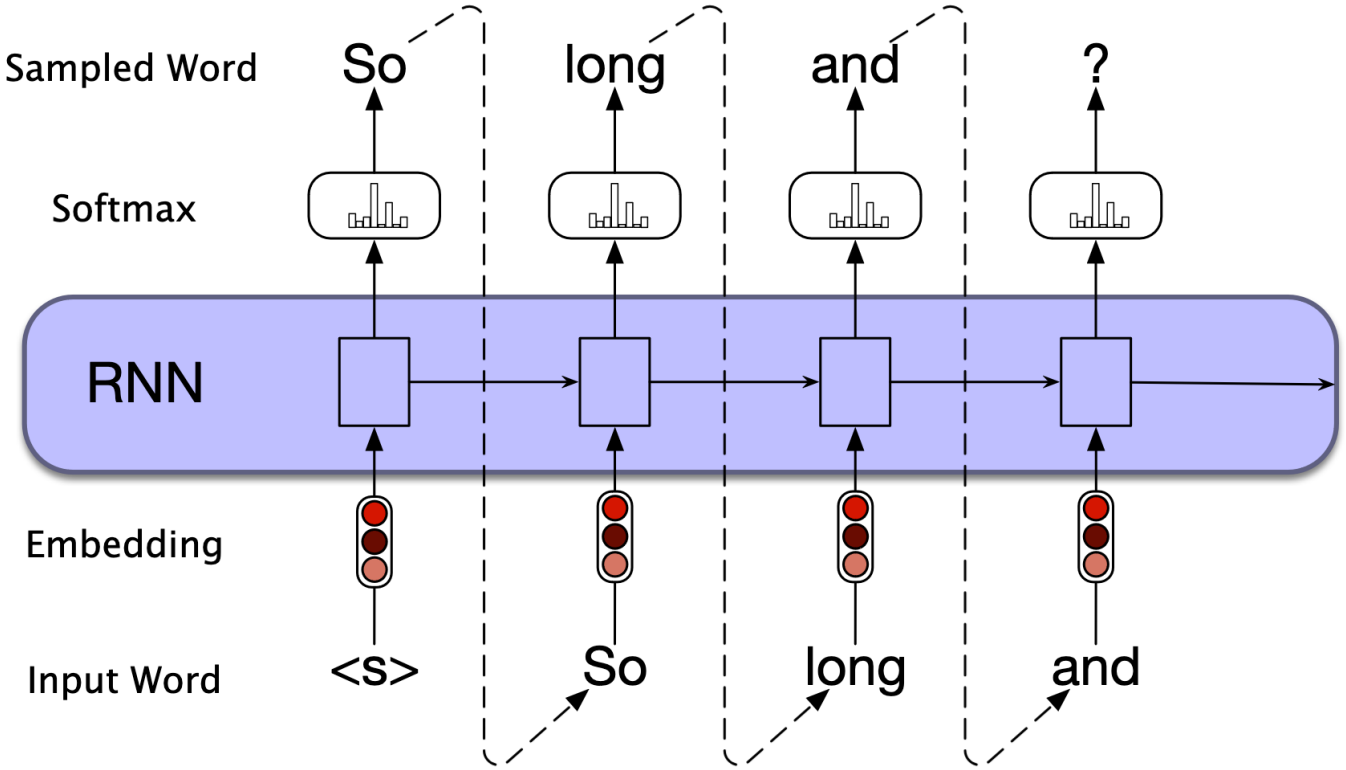
- La text generation fa parte di tutti i task in cui un sistema deve produrre testo, condizionato da altro testo
  - Question answering
  - Traduzione automatica (Machine translation)
  - Riassunto del testo (Text summarization)
  - Correzione grammaticale
  - Generazione di storie
  - Dialogo conversazionale

Text generation

- Ricordiamo lo Shannon game (campionamento)
  - Una parola è campionata in base alla sua probabilità di essere una parola d'inizio, cioè $P(w|<s>)$
  - Ogni parola nella frase è campionata condizionata alle scelte precedenti $P(w_i|w_{1:i-1})$
- Nei neural language models, il sampling è adattato all'architettura RNN

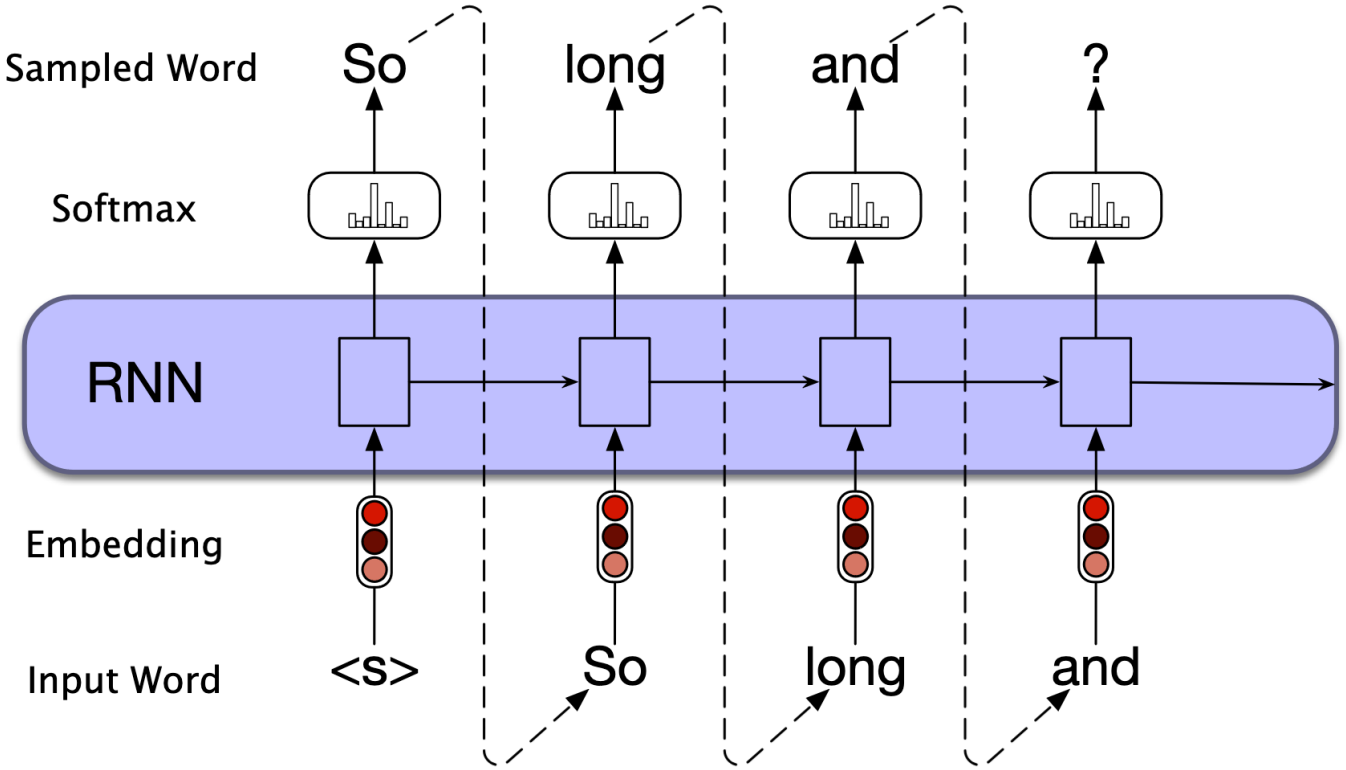
Text generation

Ogni parola è campionata come risultato della ***softmax precedente***



Text generation

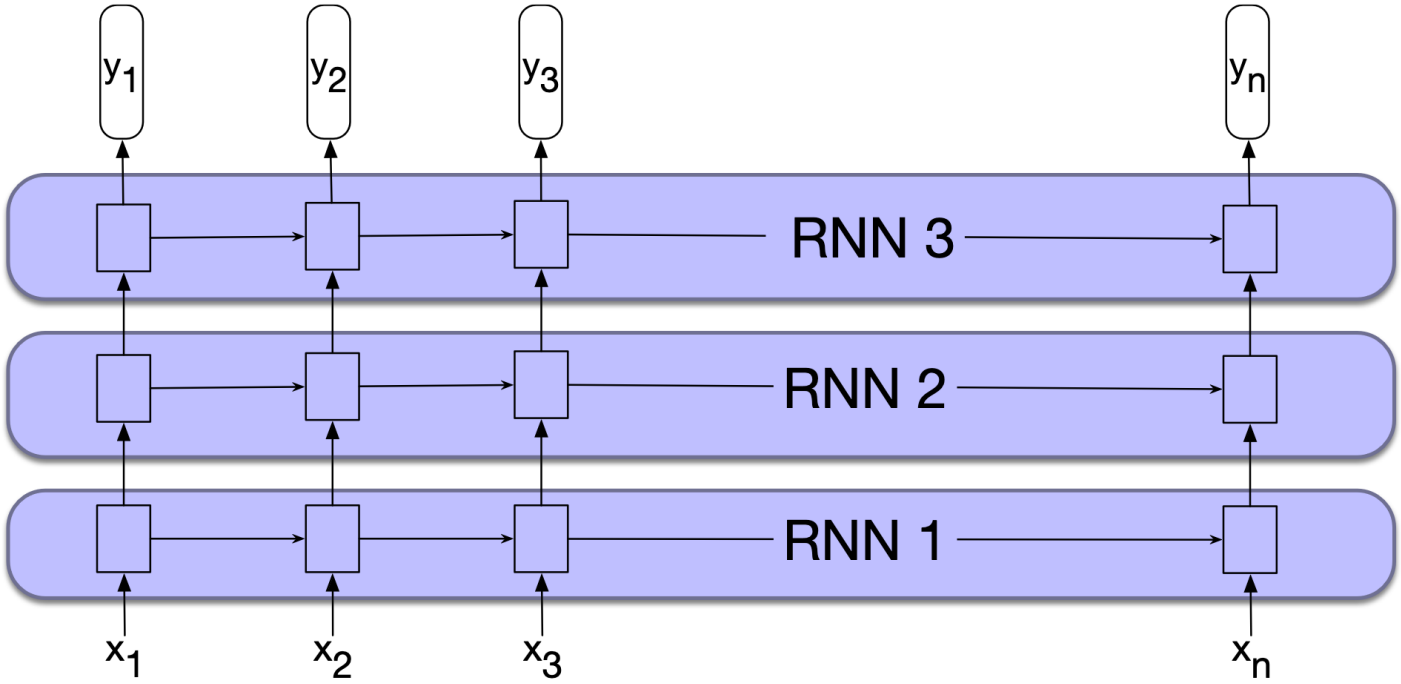
L'output campionato è dato in input all'input successivo della RNN, e così via



Architetture RNN «Impilate» (Stacked) e Bidirezionali

Stacked RNN

La sequenza di output di un layer è l'input per quello successivo



RNN Bidirezionali

- Le RNN usano informazioni provenienti dal *contesto sinistro* al passo *t*
  - Lo stato nascosto $\mathbf{h}_t^f$ al passo *t* è una funzione in avanti del contesto $\mathbf{x}_1 \ldots \mathbf{x}_t$

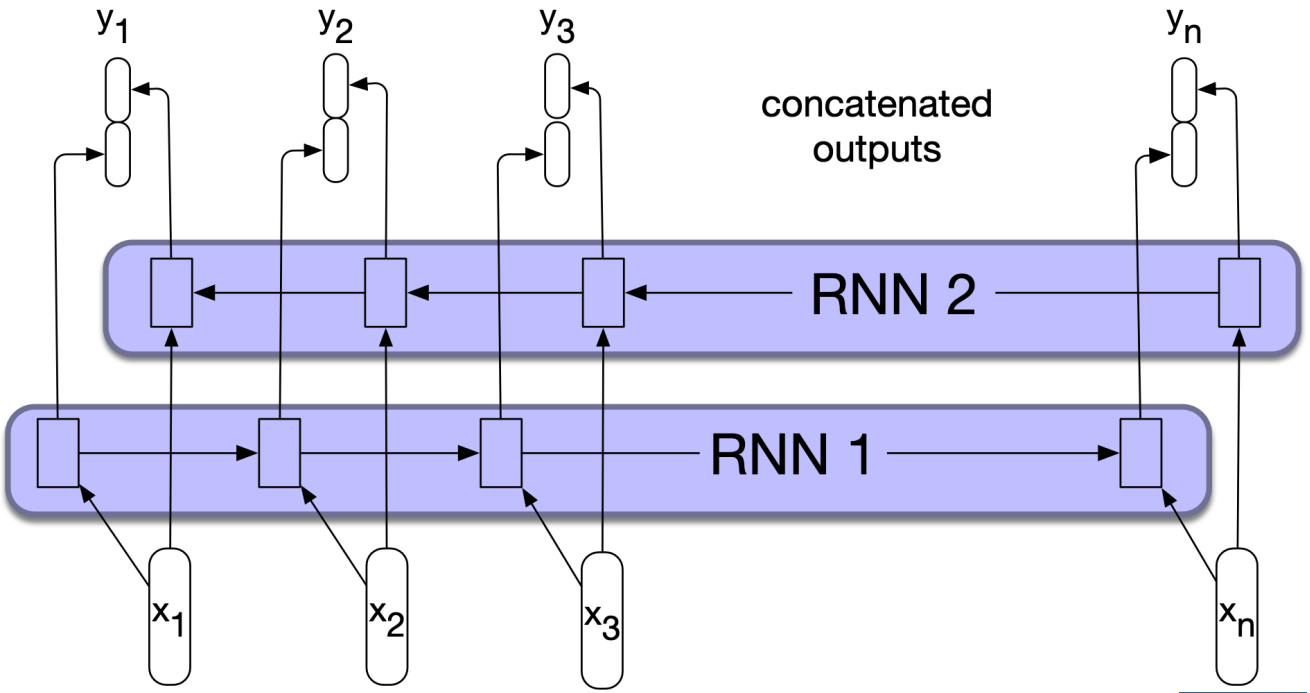
$$\mathbf{h}_t^f = \text{RNN}_{\text{forward}}(\mathbf{x}_1, \ldots, \mathbf{x}_t)$$

- Spesso è utile ottenere informazioni anche dal *contesto destro*

RNN Bidirezionali

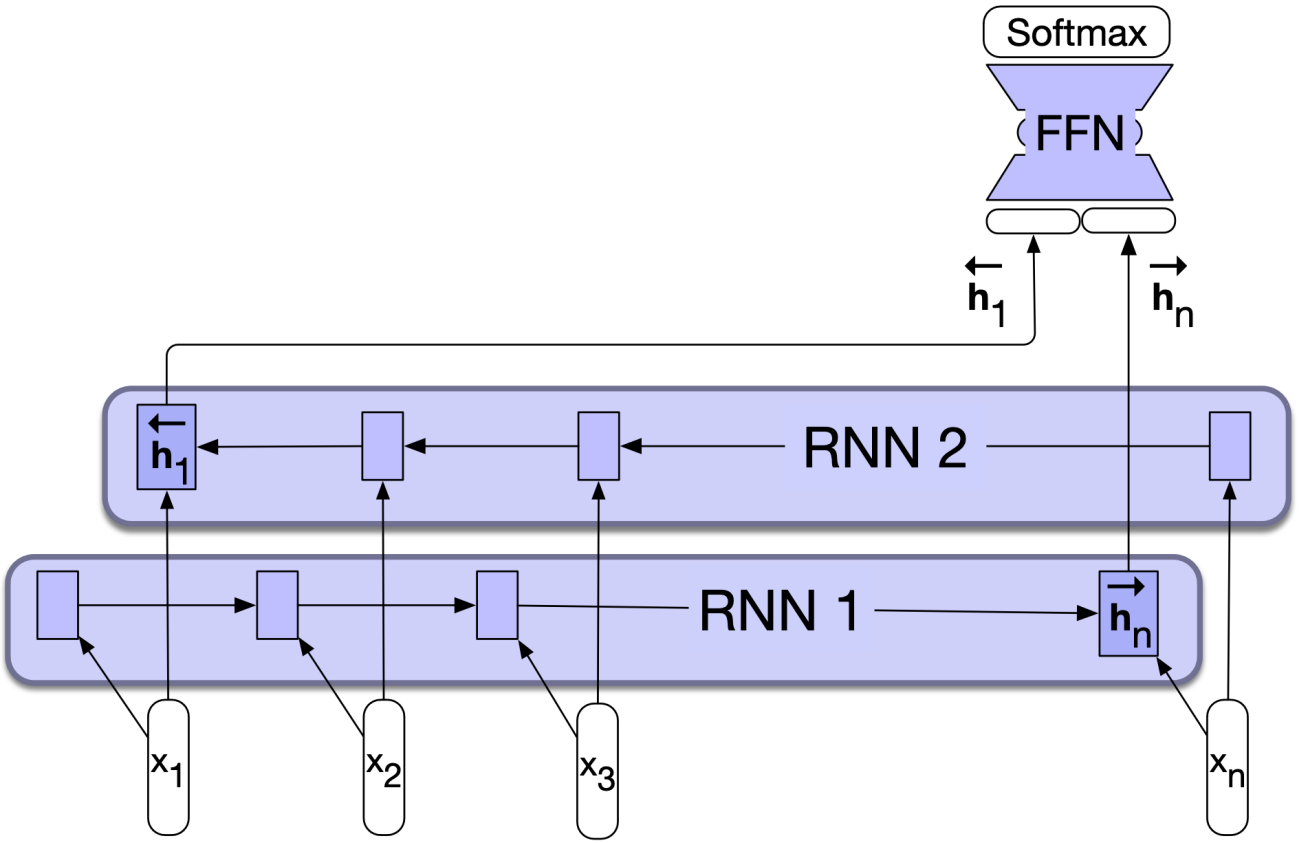
- Le RNN bidirezionali sono addestrate in modalità forward e backward
- Gli output forward e backward vengono quindi concatenati

$$\mathbf{h}_t^b = \mathrm{RNN}_{\mathrm{backward}}(\mathbf{x}_t, \ldots \mathbf{x}_n)$$



RNN Bidirezionali

Nella text classification $\mathbf{h}_1^b \oplus \mathbf{h}_n^f$ viene passato al classificatore FFN



Long Short-Term Memory (LSTM)

Informazione distante

- Le RNN non sono efficaci nel gestire dipendenze tra parole distanti

L'auto nuova che avevo comprato in concessionaria il mese scorso si è rotta

- Gli strati nascosti soffrono del problema dei *vanishing gradients* nel passaggio backward del training

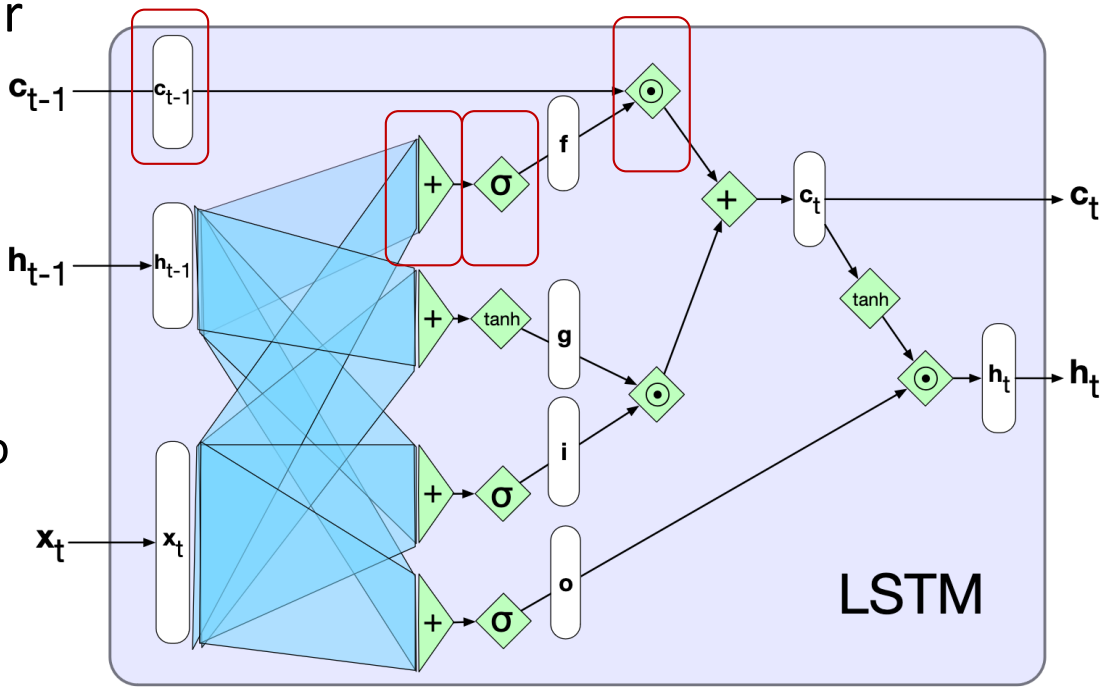
Gate

- Le LSTM dividono il problema della gestione del testo in due sottoproblemi:
  - Rimuovere informazioni non più necessarie dal contesto
  - Aggiungere informazioni che probabilmente saranno necessarie per decisioni future
- I Gates sono unità neurali progettate per imparare la gestione del contesto

Gates

- Viene aggiunto un nuovo layer di contesto $\mathbf{c}_t$

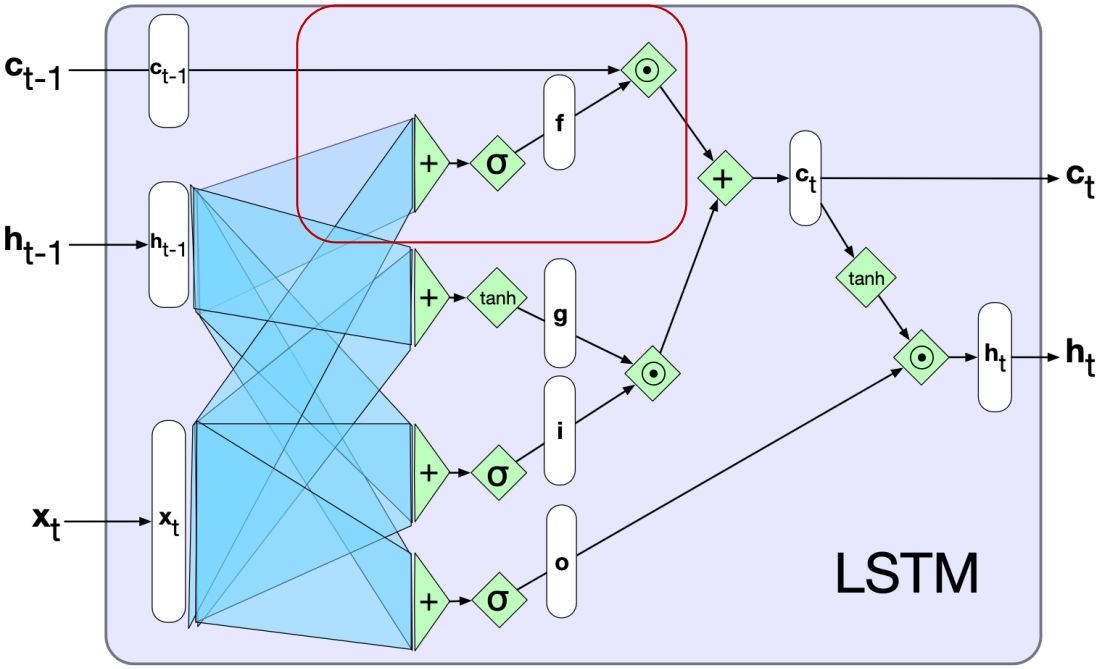
- Un *gate* è composto da:
  - Un feed-forward layer
  - Un'attivazione sigmoidale
  - Una moltiplicazione elemento per elemento con il layer da controllare



Forget Gate

- Elimina le informazioni dal contesto precedente che non sono più necessarie

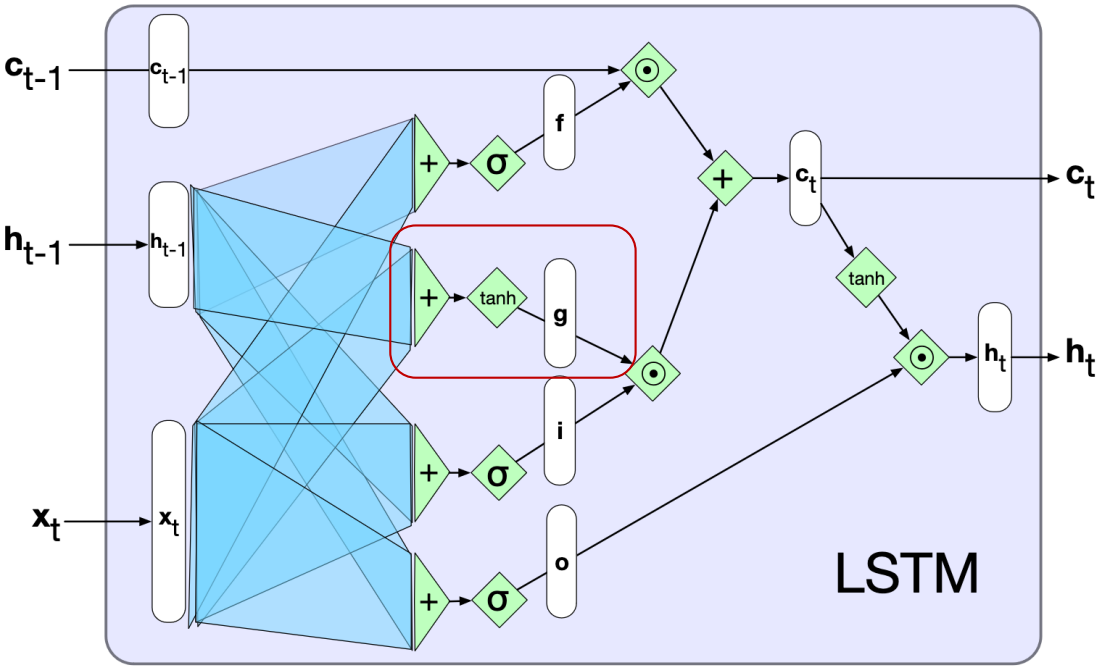
$$\mathbf{f}_t = \sigma(\mathbf{U}_f \mathbf{h}_{t-1} + \mathbf{W}_f \mathbf{x}_t)$$
$$\mathbf{k}_t = \mathbf{c}_{t-1} \odot \mathbf{f}_t$$



Estrazione di informazioni al passo *t*

- Calcola le informazioni dallo stato nascosto precedente e dagli input correnti

$$\mathbf{g}_t = \tanh(\mathbf{U}_g \mathbf{h}_{t-1} + \mathbf{W}_g \mathbf{x}_t)$$



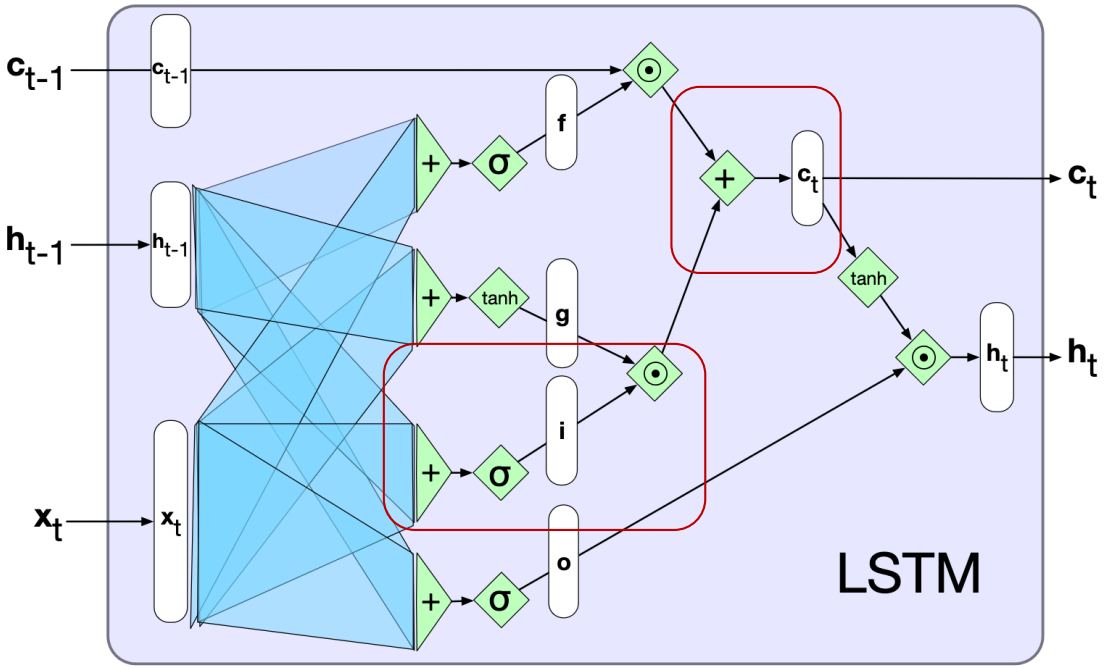
Add gate

- Seleziona le informazioni rilevanti e le aggiunge al contesto corrente

$$\mathbf{i}_t = \sigma(\mathbf{U}_i \mathbf{h}_{t-1} + \mathbf{W}_i \mathbf{x}_t)$$

$$\mathbf{j}_t = \mathbf{g}_t \odot \mathbf{i}_t$$

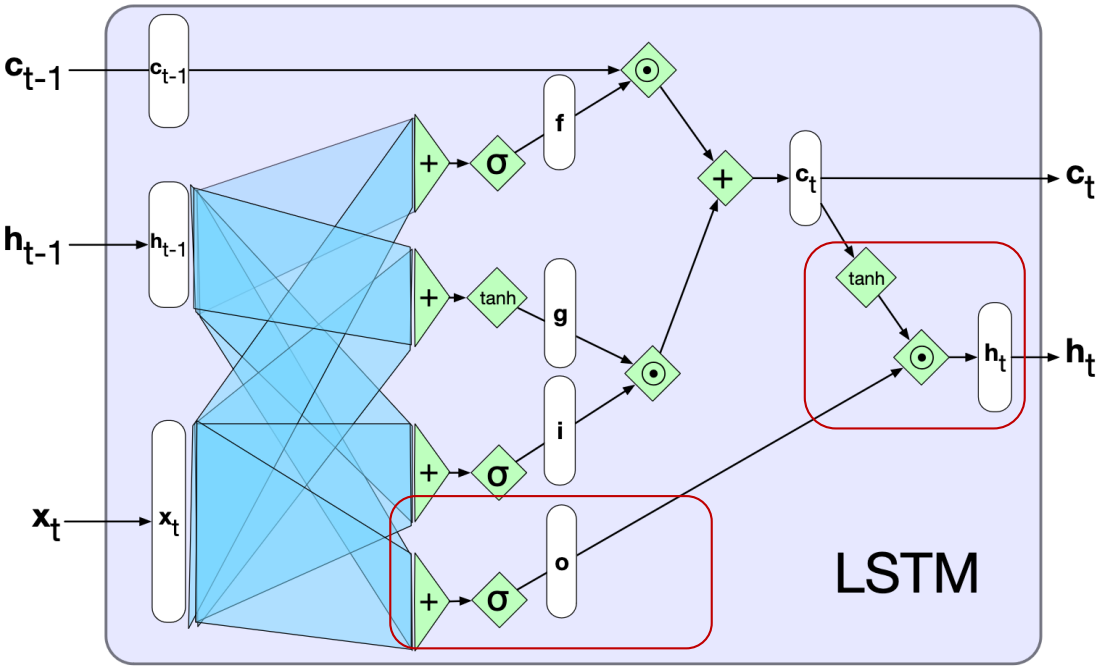
$$\mathbf{c}_t = \mathbf{j}_t + \mathbf{k}_t$$



Output gate

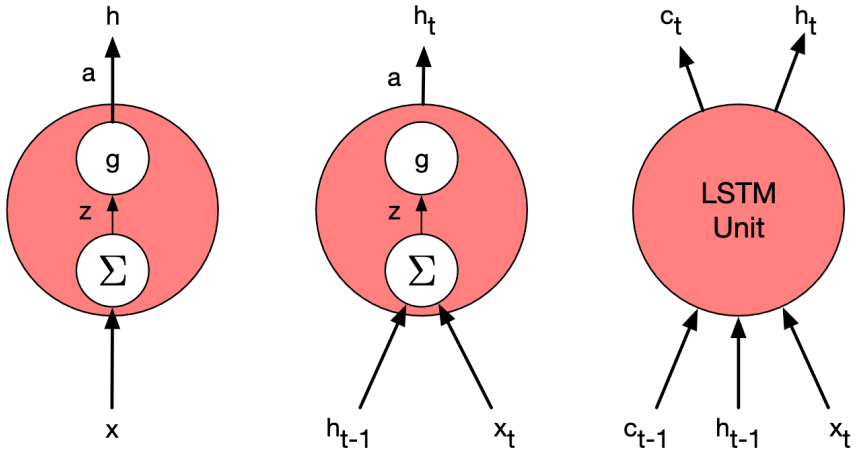
- Decide quali sono le informazioni richieste per lo stato nascosto corrente

$$\mathbf{o}_t = \sigma(\mathbf{U}_o \mathbf{h}_{t-1} + \mathbf{W}_o \mathbf{x}_t)$$
$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t)$$



Reti LSTM

- Complessità crescente da FFN a RNN e LSTM
- Il vettore di contesto è aggiunto sia come input sia come output rispetto all'unità RNN
- Reti Stacked e backpropagation attraverso il computational graph srotolato sono ancora possibili con le unità LSTM



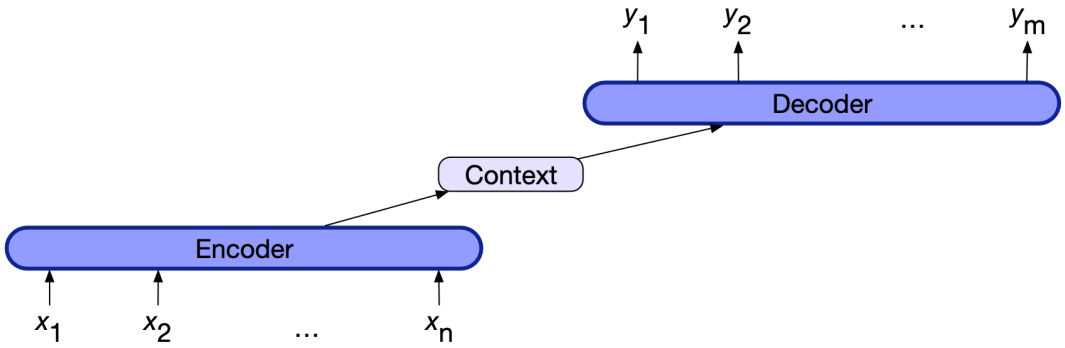
Il Modello Encoder-Decoder con RNN

Modello Encoder-decoder

- Task come la machine translation sono ancora task di sequence labeling ma…
  - La sequenza di input e output non hanno la stessa lunghezza
  - Il mapping tra coppie di token input/output può essere indiretto
    - I verbi hanno posizioni diverse in lingue diverse

Modello Encoder-decoder

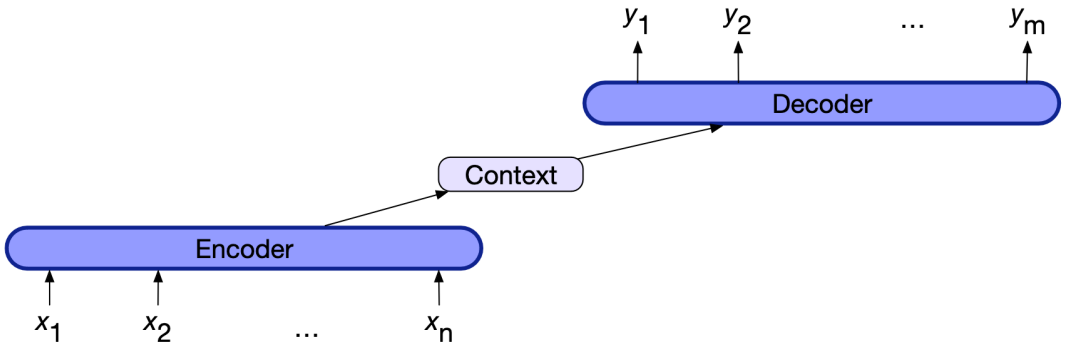
- Le reti *encoder-decoder* (note anche come *sequence-to-sequence*) affrontano questo problema



  - Encoder: CNN/LSTM/Transformer che mappa $x_{1:n} \rightarrow h_{1:n}$

Modello Encoder-decoder

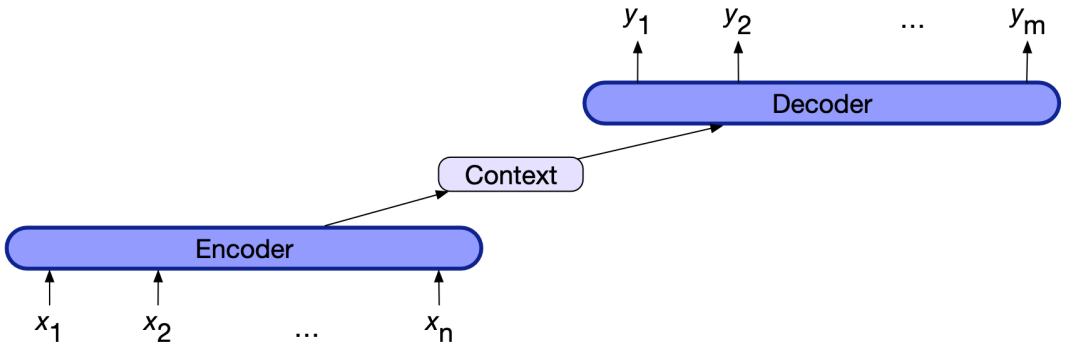
- Le reti *encoder-decoder* (note anche come *sequence-to-sequence*) affrontano questo problema



  - Vettore di contesto (Context vector):
  - $c = g(h_{1:n})$

Modello Encoder-decoder

- Le reti *encoder-decoder* (note anche come *sequence-to-sequence*) affrontano questo problema



  - Decoder: CNN/LSTM/Transformer che mappa $h_{1:m} = f(c)$, $h_{1:m} \rightarrow y_{1:m}$

Reti Encoder-decoder che usano RNN

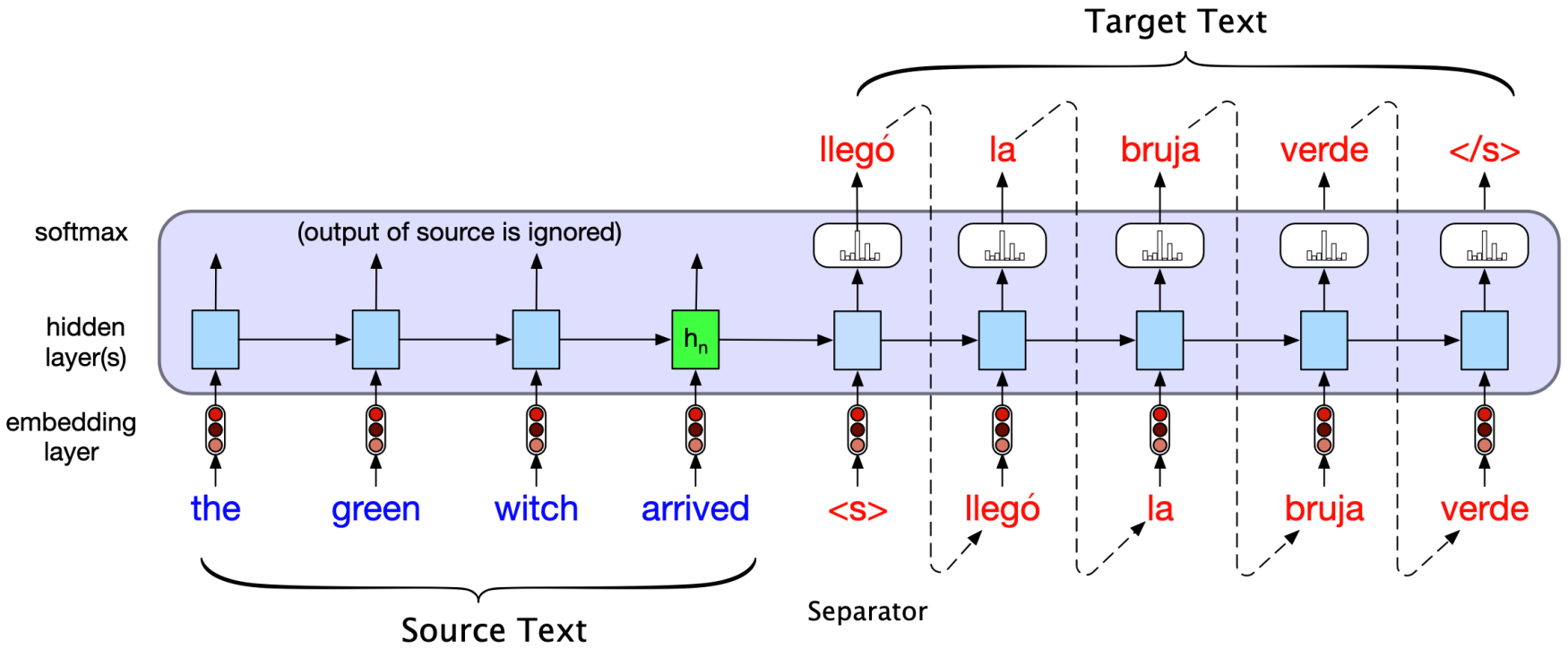
- Ricordiamo i language model con generazione autoregressiva

$$\mathbf{h}_t = g(\mathbf{h}_{t-1}, \mathbf{x}_t)$$
$$\mathbf{y}_t = f(\mathbf{h}_t)$$

  - Iniziamo con un token <s> appropriato
  - Ad ogni passo *t*, $\mathbf{x}_t \equiv \mathbf{y}_{t-1}$

- Ogni hidden state contiene informazioni sul contesto $\mathbf{x}_{1:t-1}$

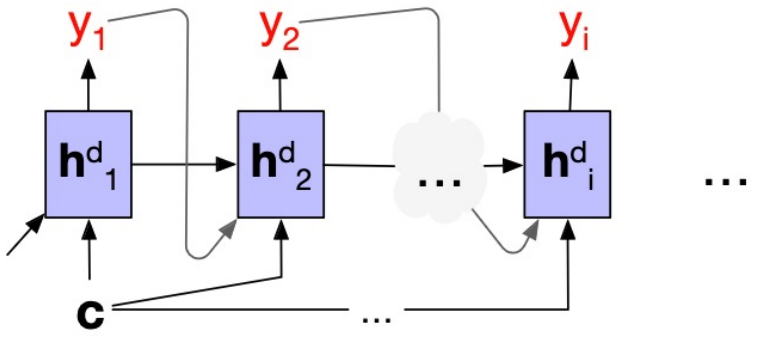
Reti Encoder-decoder che usano RNN



Inferenza

Reti Encoder-decoder che usano RNN

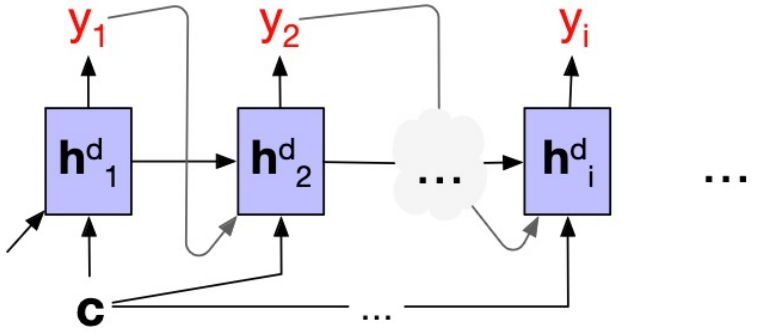
- $\mathbf{h}_n \equiv \mathbf{h}^e_n$ è il contesto **c** in questo modello
  - È passato solo al primo hidden state del decoder $\mathbf{h}^d_1$
  - Svanisce man mano che la sequenza di output viene generata
  - Idea! Passare **c** a *tutti* gli stati nascosti



Reti Encoder-decoder che usano RNN

Il word embedding per l'output campionato dalla softmax al passo precedente

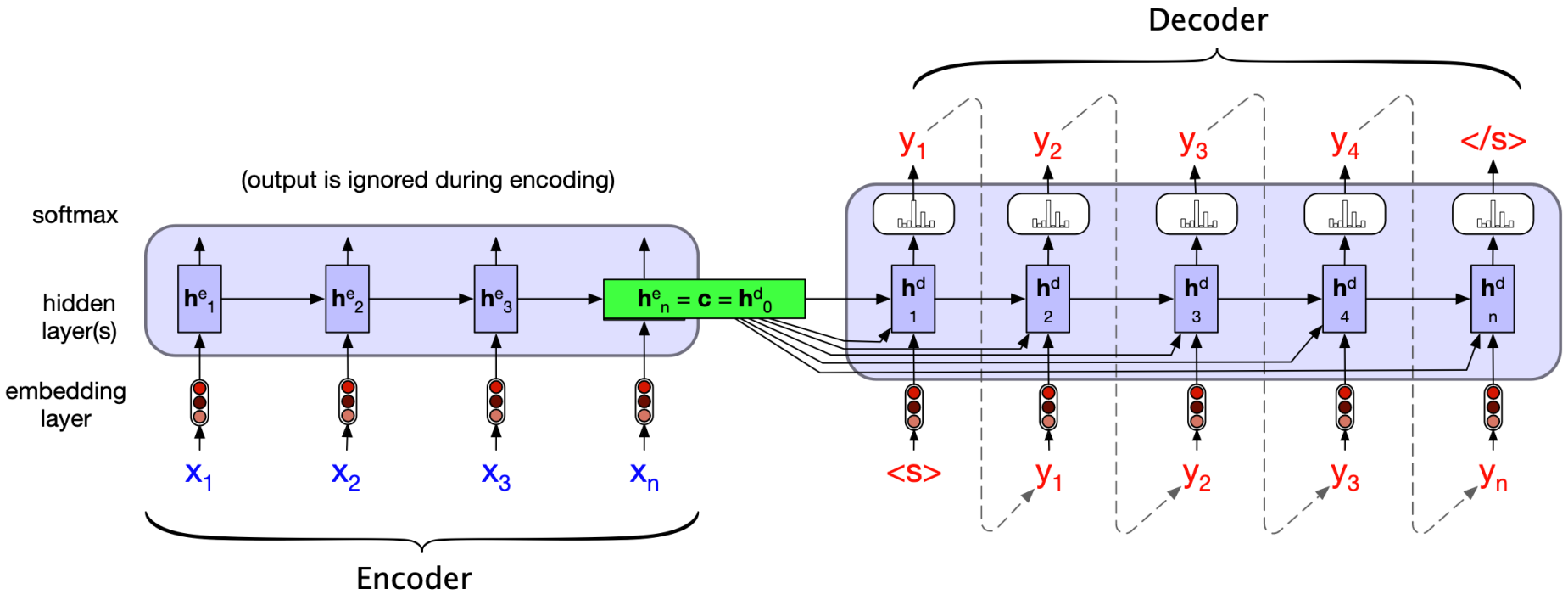
$$
\begin{aligned}
\mathbf{c} &= \mathbf{h}_n^e \\
\mathbf{h}_0^d &= \mathbf{c} \\
\mathbf{h}_t^d &= g(\hat{y}_{t-1}, \mathbf{h}_{t-1}^d, \mathbf{c}) \\
\mathbf{z}_t &= f(\mathbf{h}_t^d) \\
y_t &= \text{softmax}(\mathbf{z}_t)
\end{aligned}
$$



$$\hat{y}_t = \text{argmax}_{w \in V} P(w | x, y_1 \ldots y_{t-1})$$

Campioniamo l'embedding di uscita prendendo l'argmax sull'output della softmax

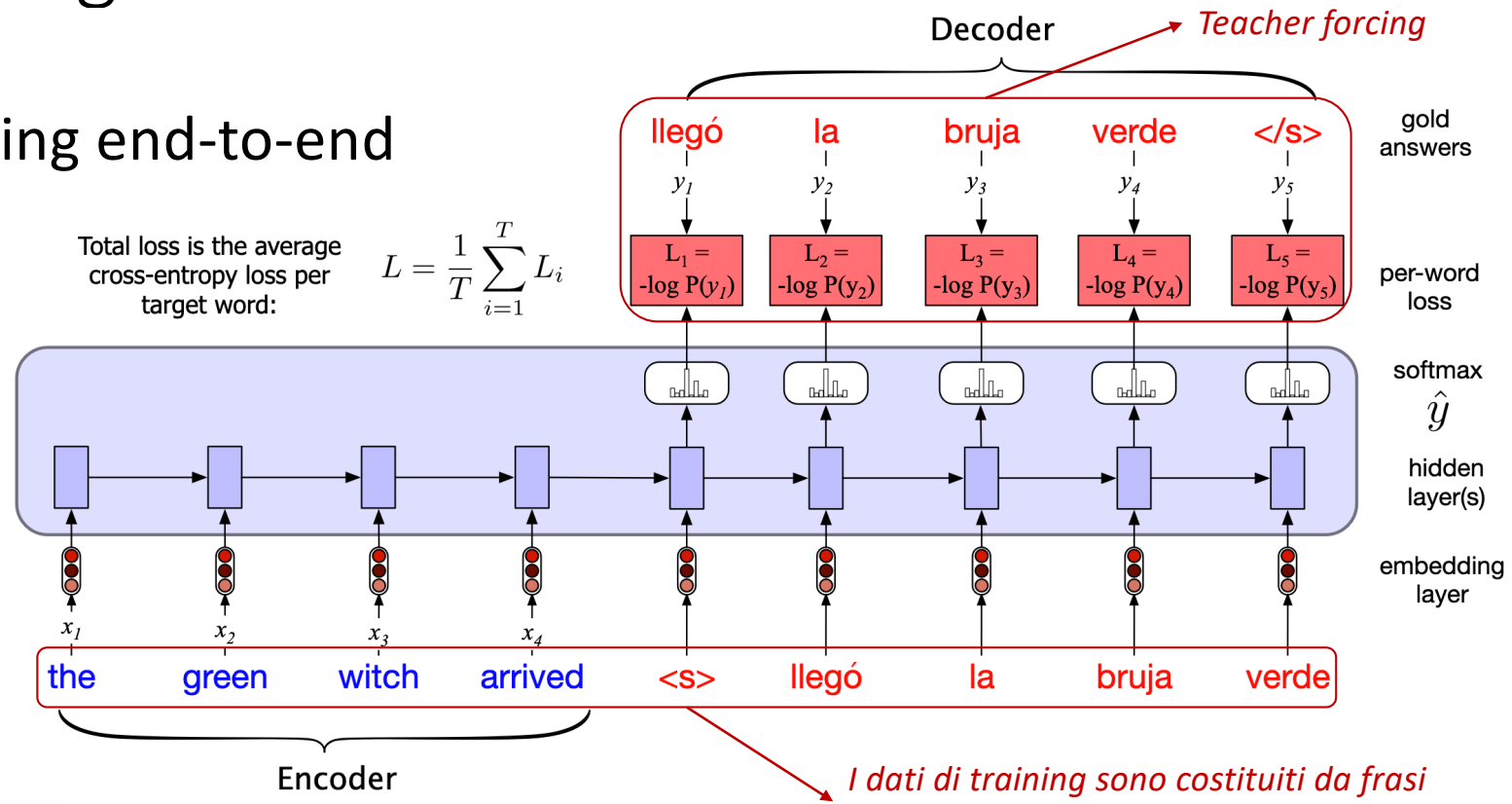
Reti Encoder-decoder che usano RNN



Inferenza

Training del modello encoder-decoder

- Training end-to-end

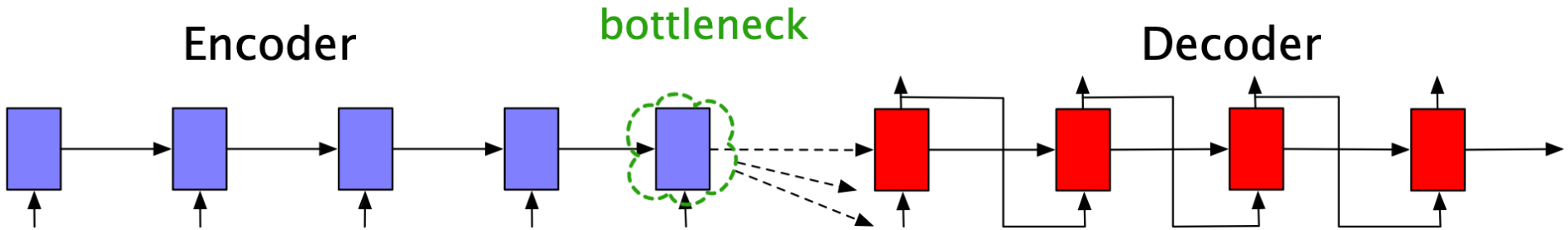


Attenzione

Il collo di bottiglia del contesto

- **c** deve rappresentare *tutte* le informazioni provenienti dal testo sorgente

- Le dipendenze a lunga distanza potrebbero non essere ben rappresentate



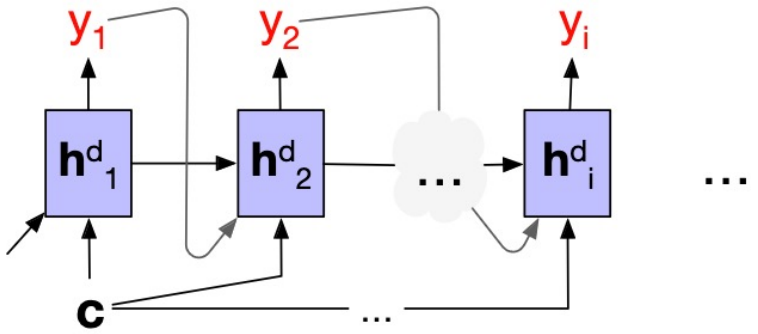
Meccanismo di Attenzione

- Volto a ottenere informazioni da tutti gli hidden states dell'encoder $\mathbf{c} = f(\mathbf{h}_1^e \ldots \mathbf{h}_n^e)$
  - Vettore di lunghezza fissa calcolato come somma pesata di tutti gli hidden states dell'encoder
  - I pesi si focalizzano su una parte specifica del testo sorgente che è rilevante per il token che il decoder sta attualmente producendo

Meccanismo di Attenzione

- Il context vector è generato di nuovo ad ogni passo di decodifica *i*

$$\mathbf{h}_i^d = g(\hat{y}_{i-1}, \mathbf{h}_{i-1}^d, \mathbf{c}_i)$$



- Iniziamo calcolando un set di scores che misurano quanto ogni hidden state dell'encoder sia rilevante per l'hidden state del decoder al passo *i-1*

$$score(\mathbf{h}_{i-1}^d, \mathbf{h}_j^e)$$

Dot-product Attention

- Lo score più semplice è il prodotto scalare

$$\text{score}(\mathbf{h}^d_{i-1}, \mathbf{h}^e_j) = \mathbf{h}^d_{i-1} \cdot \mathbf{h}^e_j$$

- Il dot product è uno scalare che riflette il grado di similarità tra i due vettori

- $\mathbf{h}^e_j$ e $\mathbf{h}^d_{i-1}$ devono avere la stessa dimensionalità

Score di attenzione score con pesi addestrabili

$$\text{score}(\mathbf{h}^d_{i-1}, \mathbf{h}^e_j) = \mathbf{h}^d_{t-1}\mathbf{W}_s\mathbf{h}^e_j$$

- I pesi $\mathbf{W}_s$ sono addestrati durante il normale training end-to-end
- La rete impara quali aspetti di similarità tra gli stati del decoder e dell'encoder sono importanti per l'applicazione corrente
- $\mathbf{h}^e_j$ e $\mathbf{h}^d_{i-1}$ possono avere dimensionalità diverse

Pesi del context vector

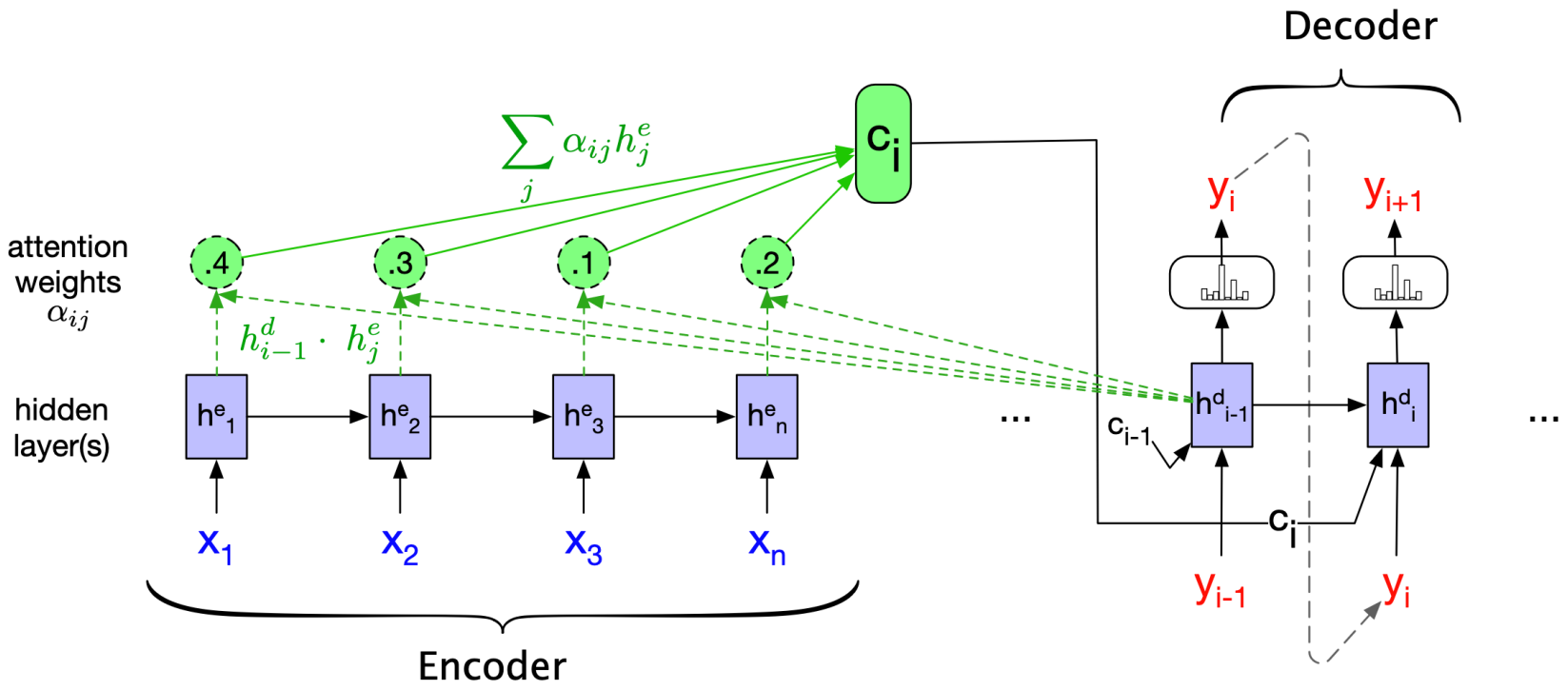
- Usiamo la softmax per normalizzare gli score e calcolare i pesi effettivi α_{ij}

$$\alpha_{ij} = \text{softmax}(\text{score}(\mathbf{h}^d_{i-1}, \mathbf{h}^e_j) \;\; \forall j \in e)$$

$$= \frac{\exp(\text{score}(\mathbf{h}^d_{i-1}, \mathbf{h}^e_j)}{\sum_k \exp(\text{score}(\mathbf{h}^d_{i-1}, \mathbf{h}^e_k))}$$

$$\mathbf{c}_i = \sum_j \alpha_{ij} \, \mathbf{h}^e_j$$

Rete Encoder-decoder con attenzione



Università degli Studi di Palermo

dipartimento di ingegneria unipa

CHILAB