

RECURRENT NEURAL NETWORK

PROBLEMI FFN

Uno dei problemi dei FFN è che abbiamo la dimensione dell'input fissata a priori. Inoltre la complessità della rete cresce linearmente col numero di inputs.

Idea: Invece di usare n input, riassumiamoli in una unica variabile h . Difatti riassumiamo il contesto in un'unica variabile:

$$\overline{P[X_i | X_{i-1} \dots X_1]} = P[X_i | h_{i-1}]$$

L'intero contesto
non solo $M-1$ parole

La variabile h è detta **latent variable**.

Da ora in avanti chiameremo h_t gli stati nascosti fino al passo (tempo) t .

L'hidden state h_t è calcolato dalla parola corrente e dallo hidden state precedente, ovvero:

$$h_t = f(X_t, h_{t-1})$$

Note: Ora dipendiamo dello stato precedente, non delle parole. Pertanto non dipendiamo più da M .

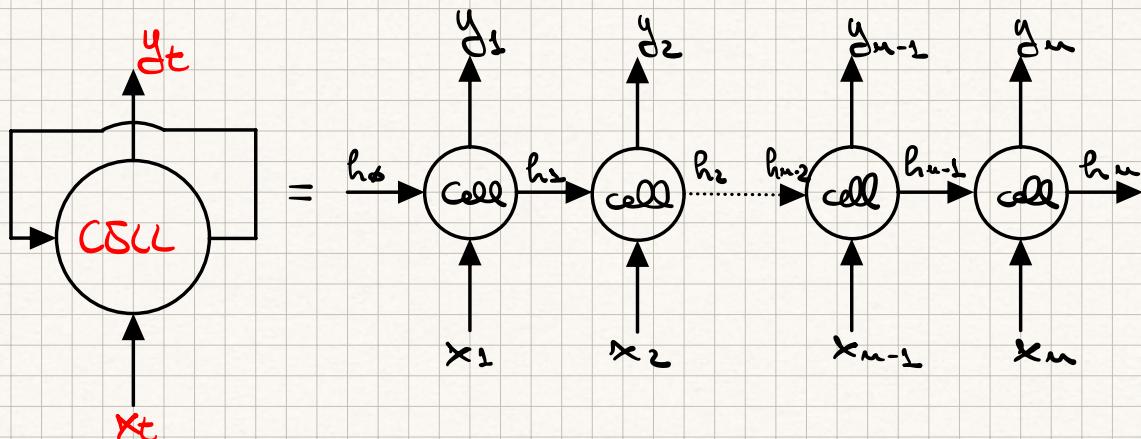
RNNs

Le Recurrent Neural Network, fanno a cosa nostra, perché sono delle reti neurali con hidden state.

L'idea principale è quello di consumare l'input, sequenzialmente e non tutto in uno (come nel FFN).

Cella

L'elemento base di una RNN è la cella. Essa prende in input lo stato precedente h_{t-1} e l'elemento corrente x_t , per dare in output y_t e h_t :

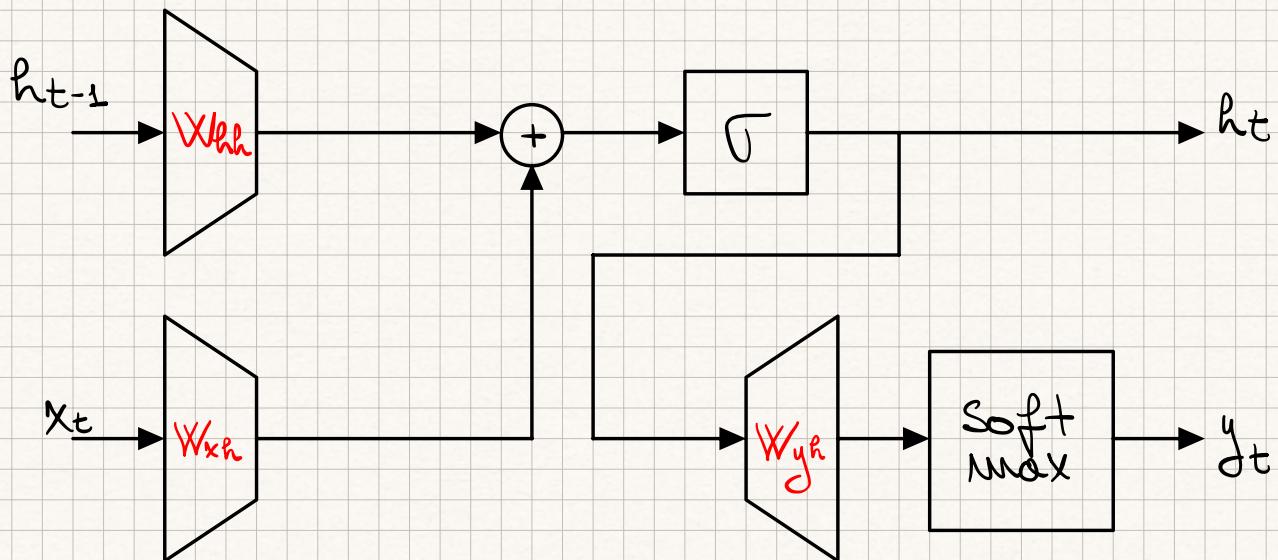


Ogni cella è concettualmente diversa, ma condividono la stessa funzione ~~f~~, che verrà imparata da una NN.
Ma dato che condividono il totale dei parametri, non dipende dalla lunghezza dell'input.

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

$$y_t = \text{softmax}_i(h_t W_{hy}) \rightarrow P(y_{t+1} | x_t, h_t)$$

SCHÉMA



Considerando h la dimensione degli hidden state e d la dimensione dei word embeddings in ingresso, abbiamo:

$$h_{t-1} \in \mathbb{R}^{1 \times h}$$

$$W_{hh} \in \mathbb{R}^{h \times h}$$

$$C \in \mathbb{R}^{V \times d}$$

$$x_t \in \mathbb{R}^{1 \times d}$$

$$W_{xh} \in \mathbb{R}^{d \times h}$$

$$\hookrightarrow \text{embedding weights}$$

$$y_t \in \mathbb{R}^{1 \times V}$$

$$W_{yh} \in \mathbb{R}^{V \times h}$$

Il numero di parametri è $O(V \cdot h + V \cdot d)$ non dipendente dal numero di parole d'ingresso. I **parametri condivisi** fra le celle sono la chiave per ridurre la complessità di ogni RNN, infatti ziasiamo W_{hh} , W_{xh} e W_{yh} per ogni cella della RNN.

Poi si deve procedere a fare l'aggiornamento dei pesi con la backpropagation.

BACK PROPAGATION THROUGH TIME (BPTT)

Iniziamo a calcolare ∇L rispetto ai suoi parametri W_{hh} e W_{xh} . Per semplicità non useremo W_{yh} perché la derivata di L rispetto a W_{yh} ($\partial L / \partial W_{yh}$) è facile per ogni cella.

Combiniamo tutti i parametri W_{hh} e W_{xh} in una unica matrice W .

Per la BPTT la loss L va calcolata per ogni step temporale, quindi fare la somma di tutti i contributi:

$$L = \sum_{t=1}^T l_t$$

Da cui:

$$\frac{\partial L}{\partial W} = \sum_{t=1}^T \frac{\partial l_t}{\partial W} = \sum_{t=1}^T \frac{\partial l_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \frac{\partial h_t}{\partial W}$$

facile da calcolare

difficile

Riscriviamo gli hidden state h_t come:

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

Da cui:

$$\frac{\partial h_t}{\partial W} = \sum_{k=1}^t \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial W}$$

$\frac{\partial h_t}{\partial h_k}$

$$\frac{\partial h_t}{\partial h_k} = \frac{\partial h_t}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial h_{t-2}} \dots = \prod_{j=k+1}^t \frac{\partial h_j}{\partial h_{j-1}}$$

Mettiamo questo risultato nel $\frac{\partial L}{\partial W}$:

$$\frac{\partial L}{\partial W} = \sum_{t=1}^T \frac{\partial y_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \sum_{k=1}^t \left[\prod_{j=k+1}^t \frac{\partial h_j}{\partial h_{j-1}} \right] \frac{\partial h_k}{\partial W}$$

Usando ora $h_t = \sigma(h_{t-1} W_{hh} + x_t W_{xh})$, calcoliamo facilmente:

$$\frac{\partial h_j}{\partial h_{j-1}} = \frac{\partial \sigma}{\partial h_{j-1}} \cdot W^\top$$

Fra le altre cose $\frac{\partial L}{\partial W}$ dipende anche da $\frac{\partial h_j}{\partial h_{j-1}}$.
Facendone la media, fissiamo degli upper bound:

$$\left\| \frac{\partial h_j}{\partial h_{j-1}} \right\| = \left\| \frac{\partial \sigma}{\partial h_{j-1}} \right\| \|W^\top\| \leq \beta_\sigma \beta_W$$

Da cui:

$$\frac{\partial h_t}{\partial h_k} \leq (\beta_\sigma \beta_W)^{t-k}$$

Questo limite superiore ci serve per calcolare $\frac{\partial L}{\partial W}$

Analizziamo i valori possibili:

1) $\beta_\theta < 1$ e $\beta_w < 1$: $(\beta_\theta \beta_w)^{t-k} \approx 0$ { VANISHING GRADIENT

2) $\beta_\theta, \beta_w \in [1, \infty)$: $(\beta_\theta \beta_w)^{t-k} \rightarrow \infty$ { EXPLODING GRADIENT

Se il gradiente scompare o esplode, non siamo in grado di allenare la rete perfettamente.

Una soluzione a questo è il GRADIENT CLIPPING, ovvero che si impongono dei limiti alla crescita (decrescita) del gradiente. Non abbiamo bisogno di andare all'infinito.

SEQ2SEQ PROBLEM - ENCODER-DECODER

Le RNN sono ottime per risolvere problemi di tipo **sequence to sequence**.

L'appuccio usato per risolvere questo problema è usare un modello di tipo **ENCODER-DECODER**:



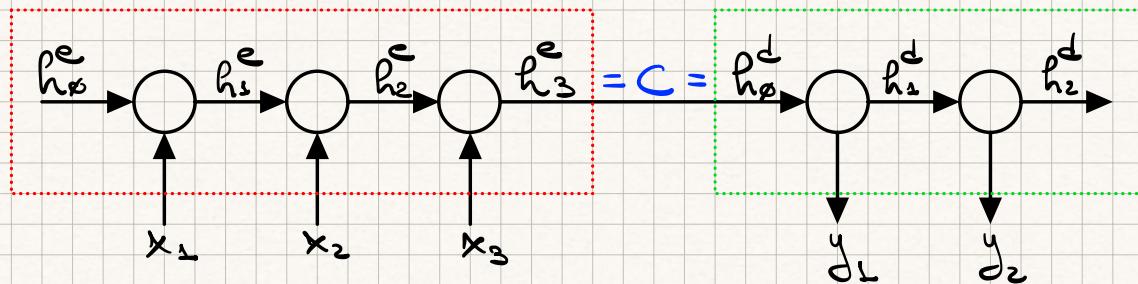
Equazioni:

$$c = f_E(x_1 \dots x_s)$$

$$y_1 \dots y_D = f_D(c)$$

Esempio:

Encoder $S=3$, Decoder $D=2$



$$\text{Encoder} \quad h_t^e = \sigma(h_{t-1}^e W_{hh} + x_t W_{hx})$$

$$\text{STATE} \quad h_0^e = C = h_0^d$$

$$\text{Decoder} \quad \begin{cases} h_t^d = \sigma(h_{t-1}^d W_{hd}) \\ y_t = \text{softmax}(h_t^d W_{hy}) \end{cases}$$

Da notare che nel encoder non ci sono equazioni di output, mentre nel decoder non ci sono equazioni di input.

DECODING STRATEGIES

Ci sono diverse strategie di decoding:

1) Greedy Search

A ogni passo si sceglie la parola più probabile come output.

Esempio:

	t_1	t_2	t_3	t_4
a	0.5	0.1	0.1	0.1
b	0.3	0.4	0.2	0.2
c	0.1	0.3	0.6	0.3
$\langle /e \rangle$	0.1	0.2	0.1	0.4

Per greedy search la sequenza più probabile è:

a - b - c - $\langle /e \rangle$

Il valore della probabilità è scelto come:

$$P[a b c \langle /e \rangle] = 0.5 \cdot 0.4 \cdot 0.6 \cdot 0.4 = 0.048$$

Attenzione: questa non è realmente la frase più probabile. Lo si nota cambiando la scelta fatta in t_2 ($t=2$) nel nostro esempio. Gli step successivi cambieranno probabilità:

	t_1	t_2	t_3	t_4
a	0.5	0.1	0.1	0.1
b	0.3	0.4	0.7	0.1
c	0.1	0.3	0.1	0.2
$\langle /e \rangle$	0.1	0.2	0.1	0.6

$$\rightarrow P[a c b \langle /e \rangle] = 0.063$$

La definizione corretta del problema del decoding per M
 è cercare una sequenza $\sigma^* \in V^*$ che, partendo da una sequenza
 iniziale $s \in V^*$, minimizza la perplexity PP o in altre
 parole massimizza la NLL:

$$\sigma^* = \underset{\sigma \in V^*}{\operatorname{argmax}} \left(\log P[\sigma | s] \right)$$

La ricerca esauriva (A tappetto) trova l'ottimo di questo
 problema (massimo decoding a posteriori (MAP)), ma
 è impraticabile perché troppo complesso e non scalabile. Ci
 sono altri modi heuristici più veloci:

2) Beam Search

A ogni step temporale valuta le top-n sequenze candidate
 e le espande tutte in tutti i modi possibili in V ,
 generando $n \cdot V$ sequenze, da cui prendere i top-n
 successivi.

3) Random Sampling

Compioniamo a caso da $P[\sigma | s]$

4) Random Sampling with Temperature

Come prima, limitando la randomicità dei campioni.

5) Top-K Sampling

Compioniamo K parole e scegliamo la più probabile
 per quella sequenza.

6) Top- p Sampling or Nucleus Sampling

Selezioniamo un valore di probabilità p e compiamo uno fra le parole più probabili con probabilità cumulativa minore o uguale a p .