Recurrent Neural Networks and Attention

Recurrent Neural Networks (RNN)

The nature of language

Language is an inherently temporal phenomenon

- Spoken language
- Flow of conversation
- X (formerly Twitter) stream

We already used language models that deal with sequences thus assuming this temporal apsect

- N-grams and HMM

The nature of language

Neural language models use either fixed size *sliding windows* or input *padding to the longest sentence*

- We will see padding again in Transformers

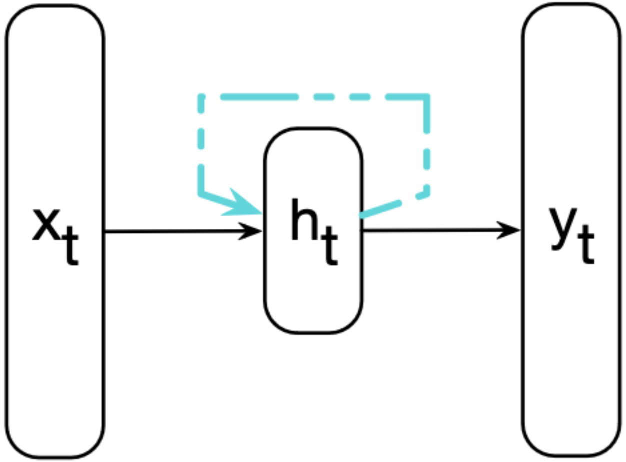
RNN deal directly with the sequential nature of language

- *Recurrent connections* allow the model's decision to depend on information from hundreds of words in the past

Elman Network (1990)

The state of the network is represented at time *t*

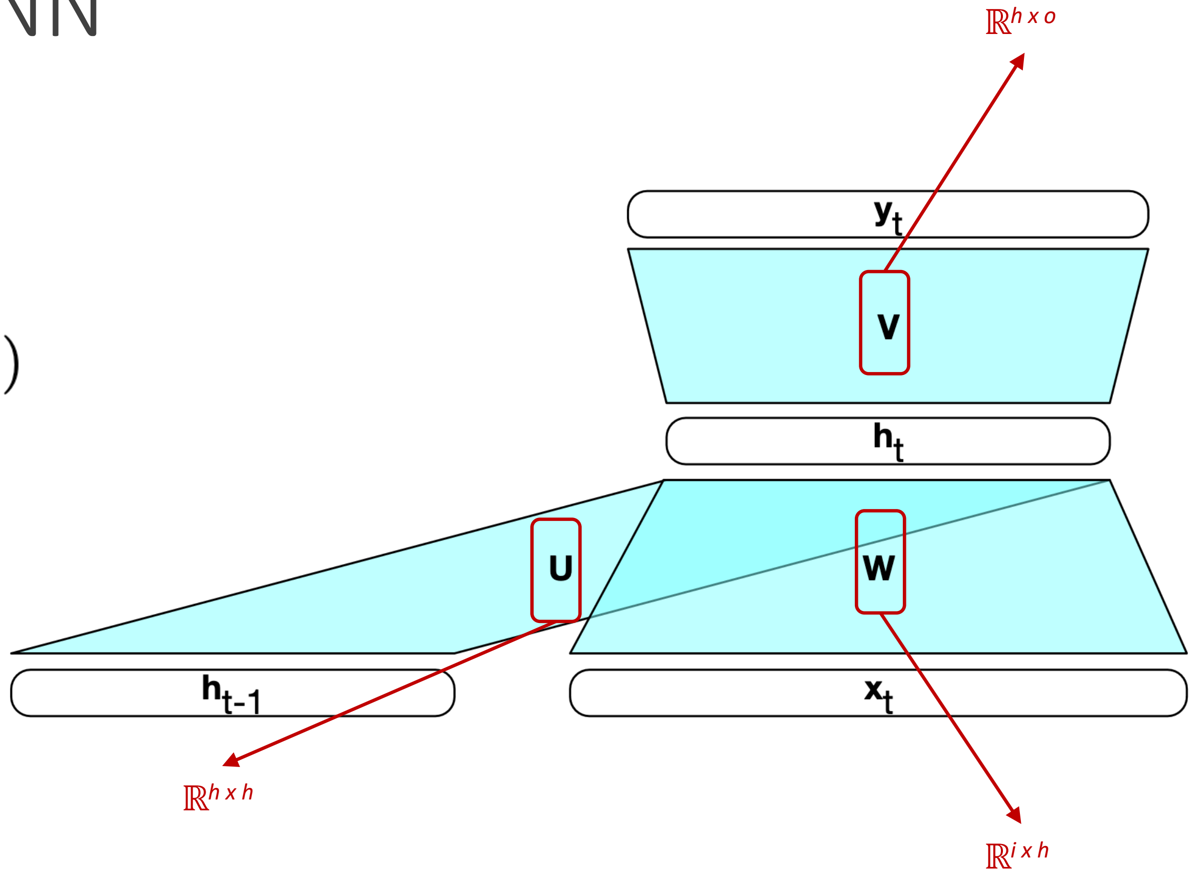
The input of the hidden layer is augmented with the ***output at the preceeding time point*** h_{t-1}



Inference in RNN

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

We use softmax for classification



Inference in RNN

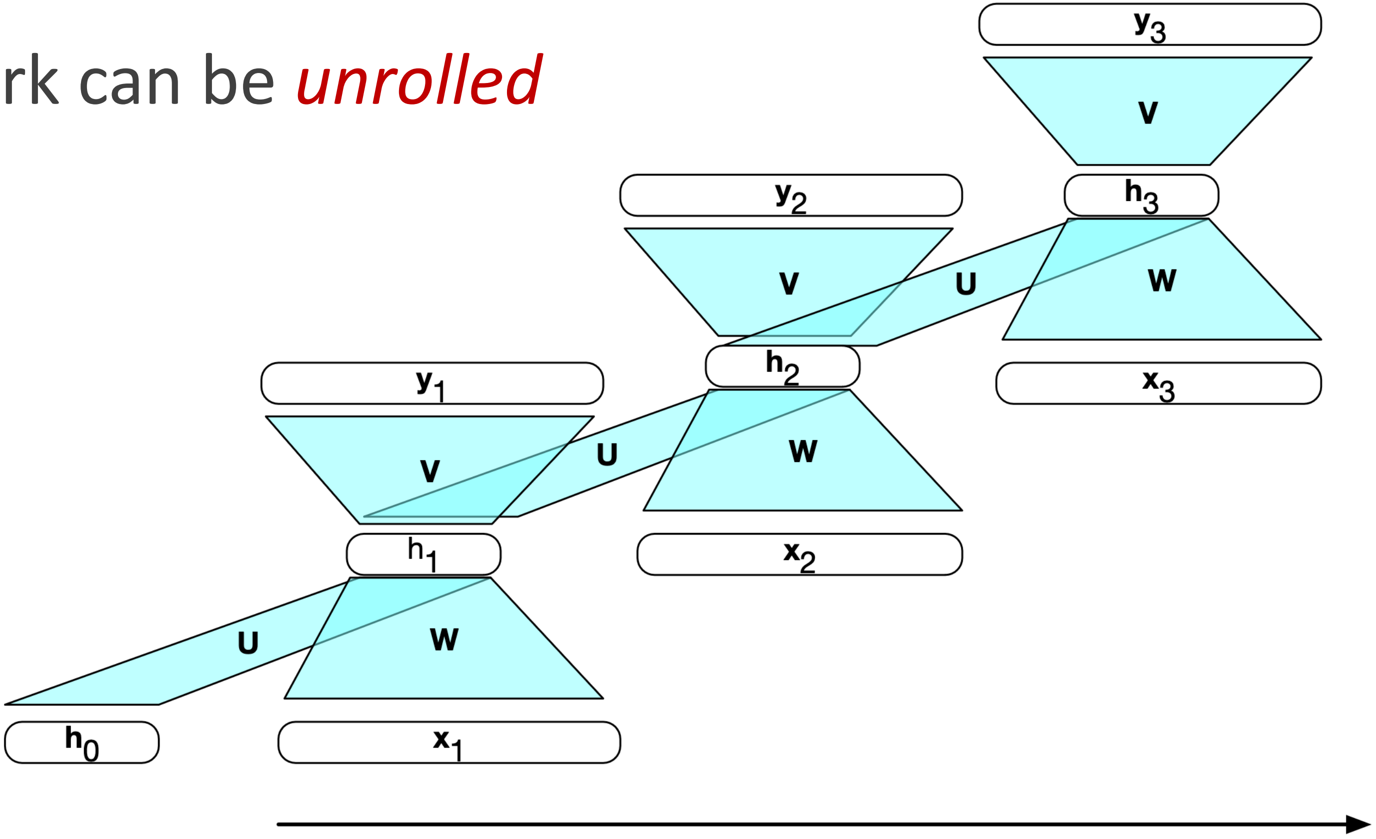
Computation iterates through the input sequence

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

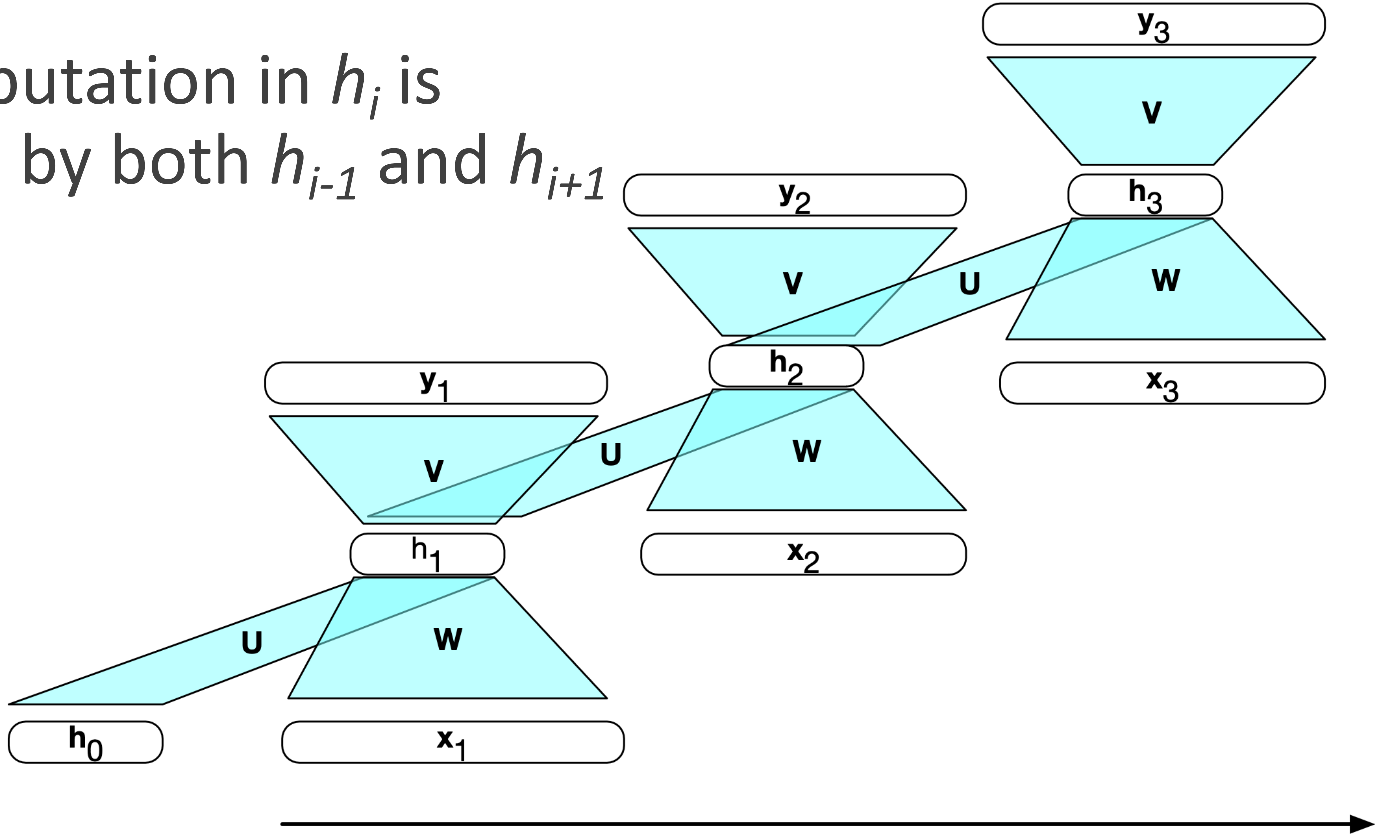
Inference in RNN

The network can be *unrolled* over time



Training RNNs

Error computation in h_i is influenced by both h_{i-1} and h_{i+1}

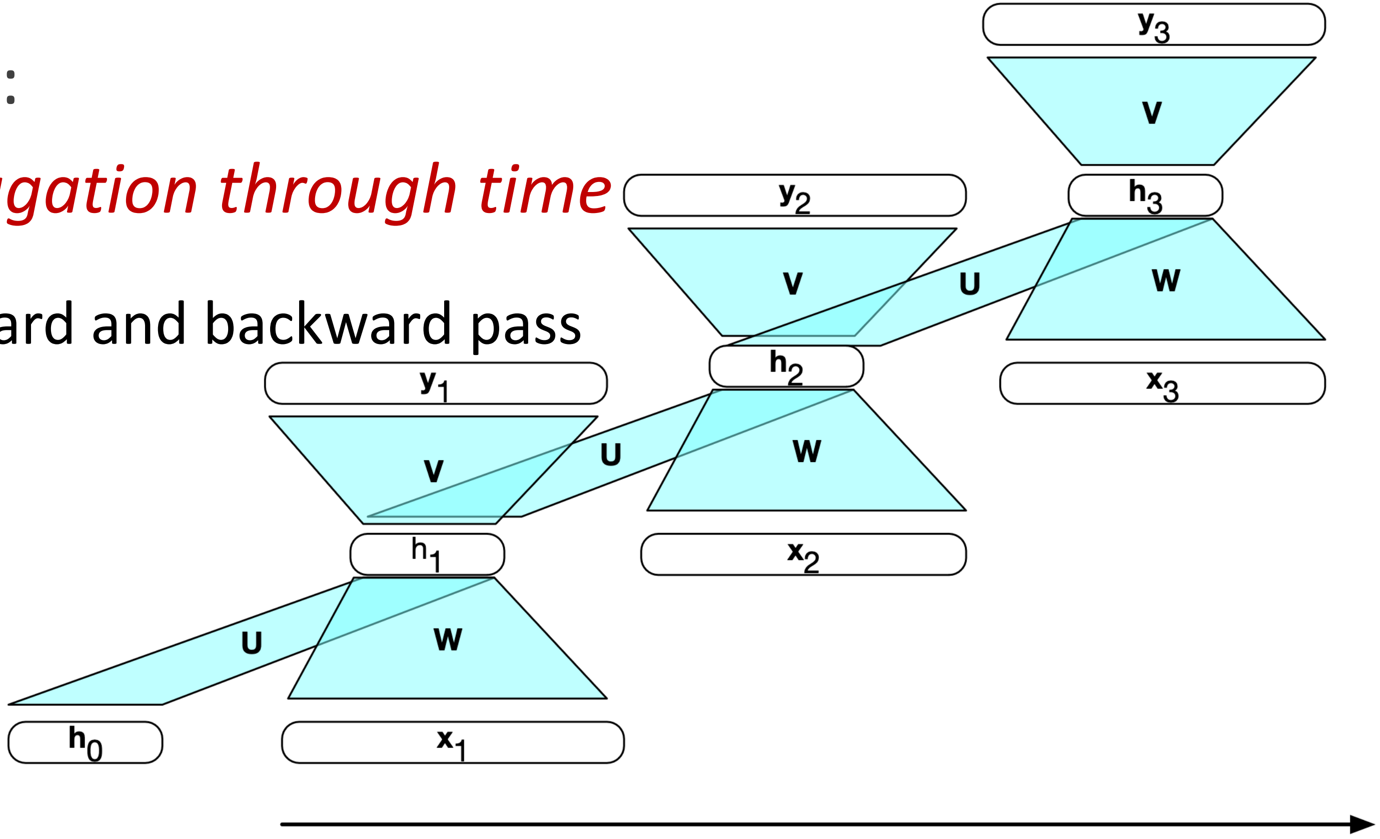


Training RNNs

Old school:

Backpropagation through time

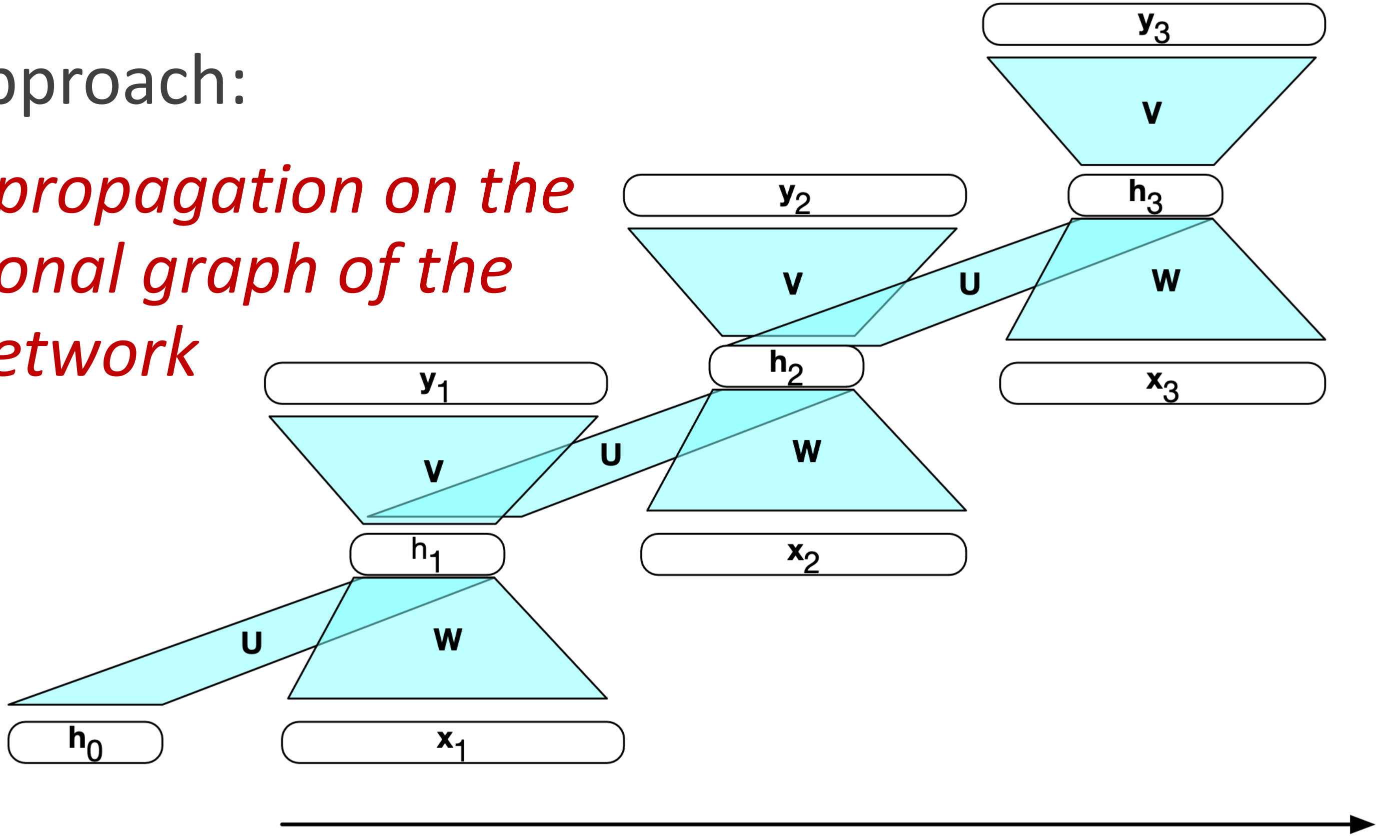
- Forward and backward pass



Training RNNs

Modern approach:

Plain Backpropagation on the computational graph of the unrolled network



Recurrent Neural Networks and Attention

RNNs as language models

Recall language modeling definition

Language modeling:

predicting how probable is a word given an observed word sequence $P(fish|Thanks\ for\ all\ the)$

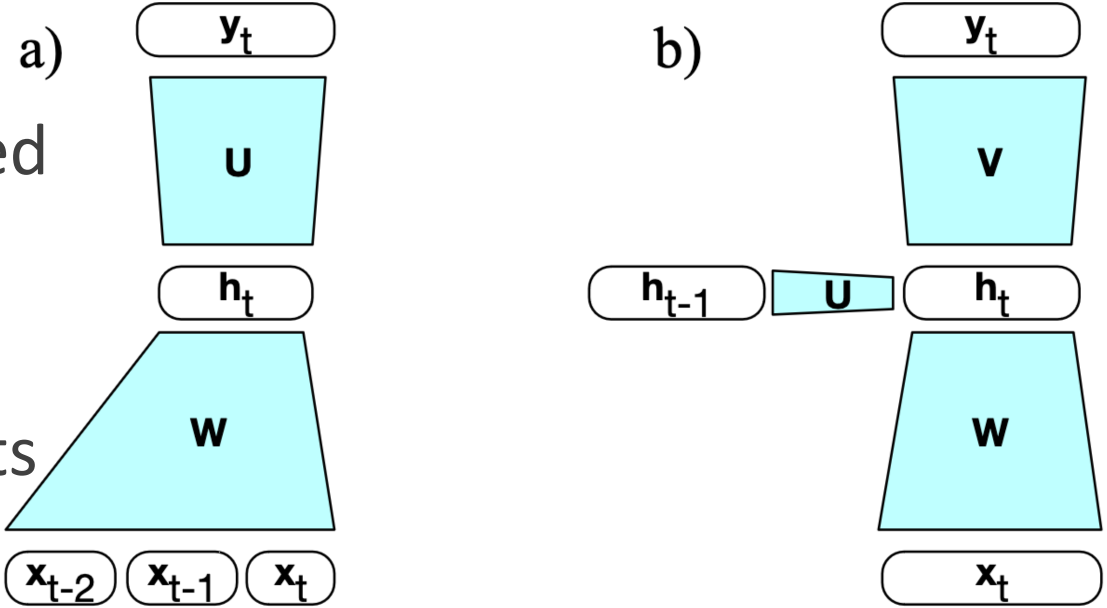
We can also predict the probability of an entire sequence

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

RNNs as and context

RNN do not have problems when dealing with unlimited context

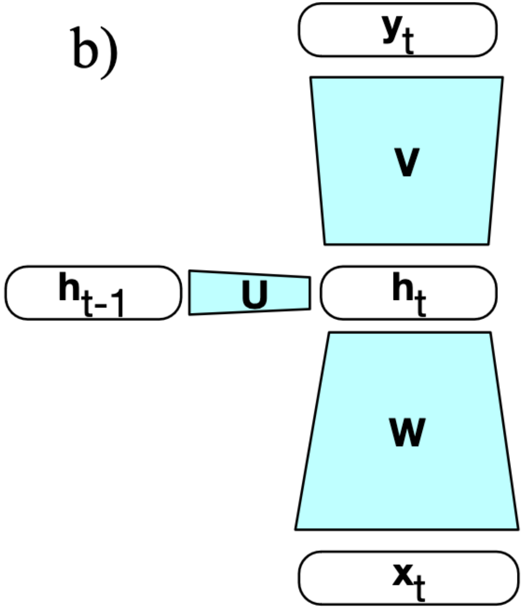
Hidden layer accounts for past context



Embeddings

We can use also the word embeddings matrix **E**

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



*In general **E** is not trained, and the layer embedding is frozen i.e. to a* word2vec *representation*

Forward inference

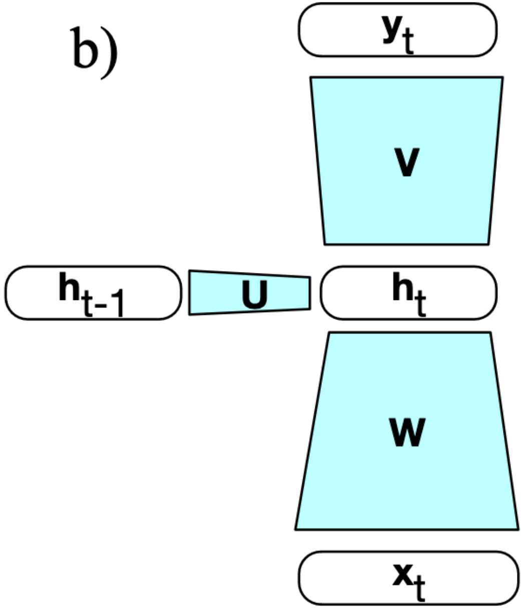
Inference

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

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

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

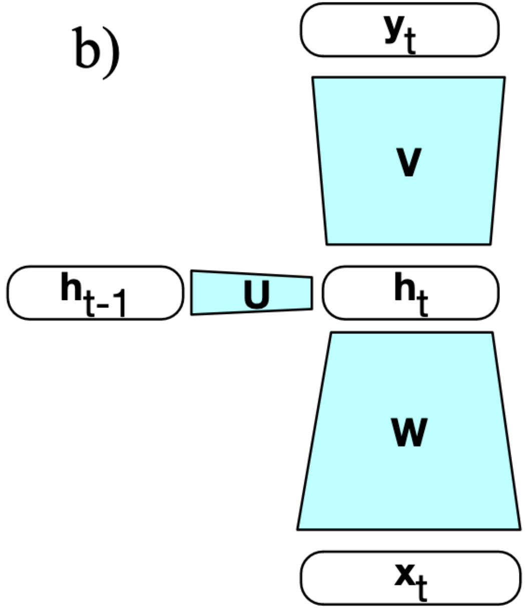
b)



Training

We will use again *self-supervision*:

the correct word to be predicted is the next one

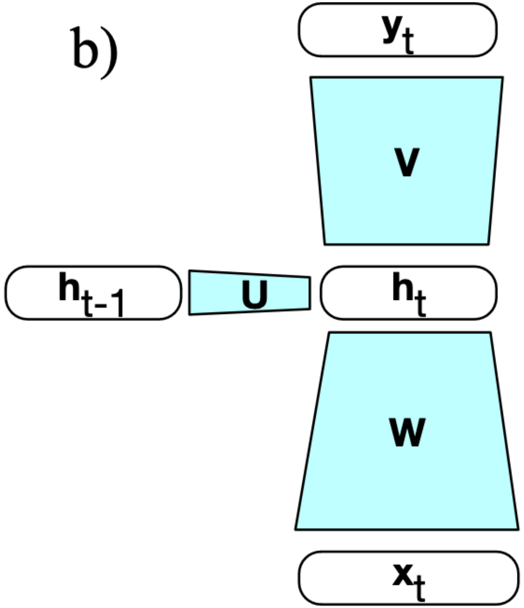


Training

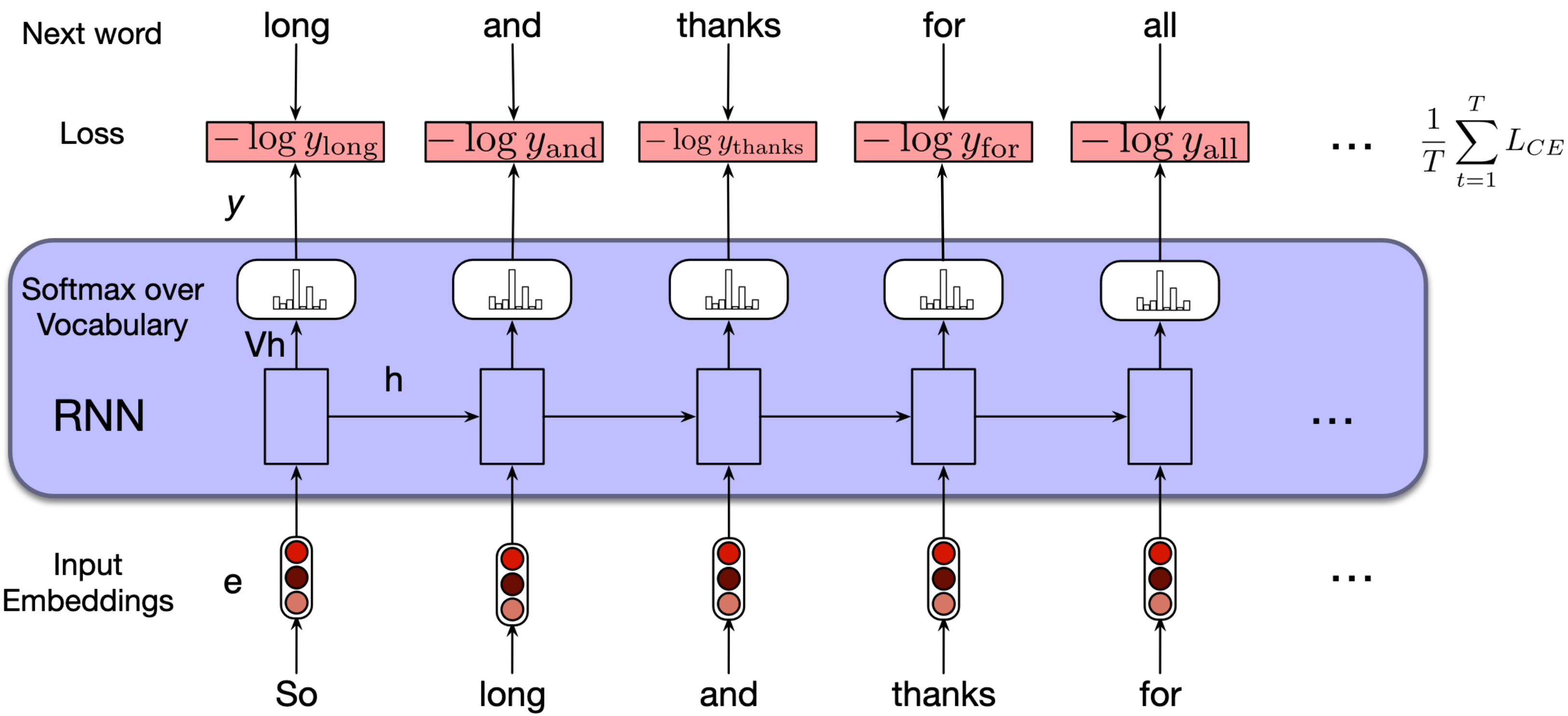
Loss: $$L_{CE} = -\sum_{w \in V} \mathbf{y}_t[w] \log \hat{\mathbf{y}}_t[w]$$

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

b)



Teacher forcing

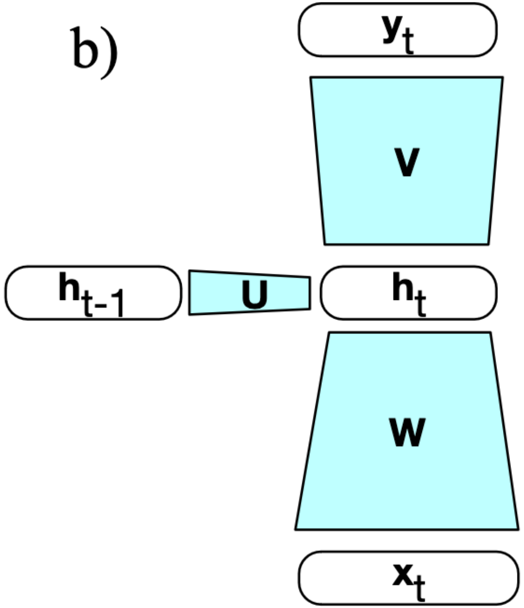


At each step we use always the correct sequence to predict next word

Weight tying

- **E** is the matrix d_h x $|V|$ of the learned word embeddings
- **V** is the matrix $|V|$ x d_h of the scores of the word probabilities given the evidence of each word stored in **h**
- *Are they actually different??*

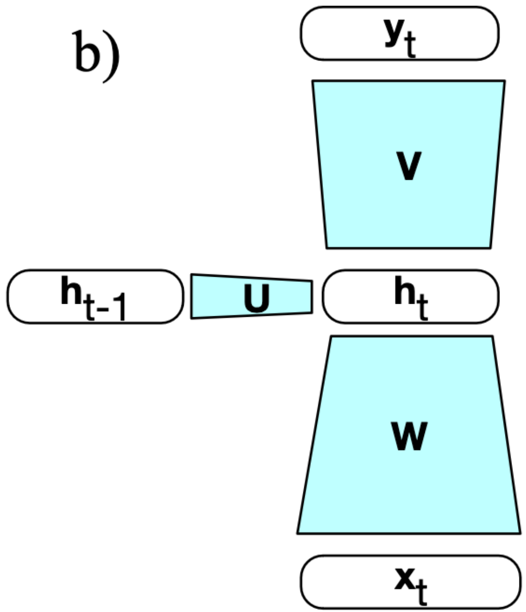
b)



Weight tying

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

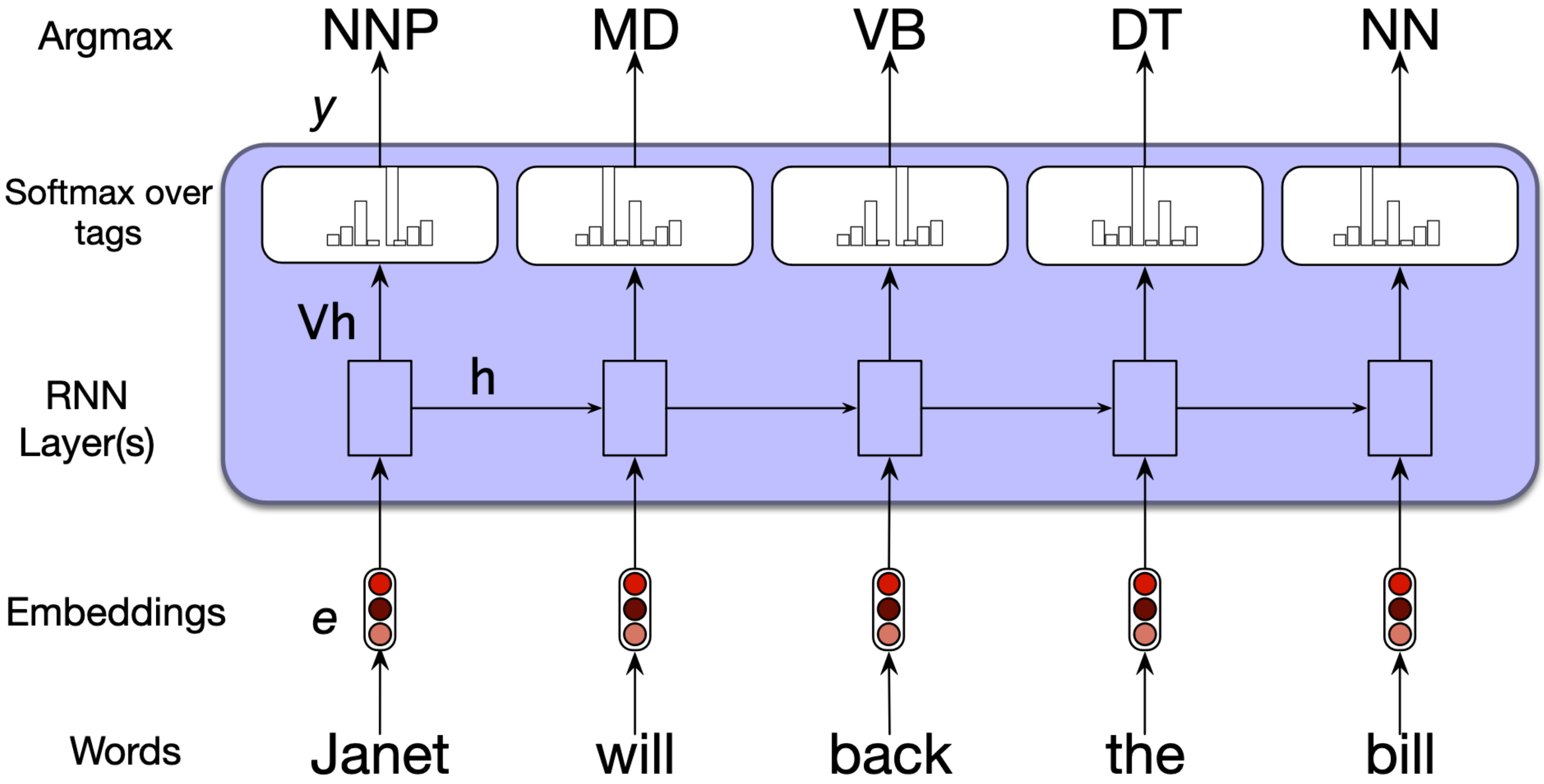
$$\mathbf{e}_t = \mathbf{E}\mathbf{x}_t$$
$$\mathbf{h}_t = g(\mathbf{U}\mathbf{h}_{t-1} + \mathbf{W}\mathbf{e}_t)$$
$$\mathbf{y}_t = \text{softmax}(\mathbf{E}^\intercal\mathbf{h}_t)$$



Recurrent Neural Networks and Attention

RNNs for other NLP tasks

Sequence labeling



Sequence classification

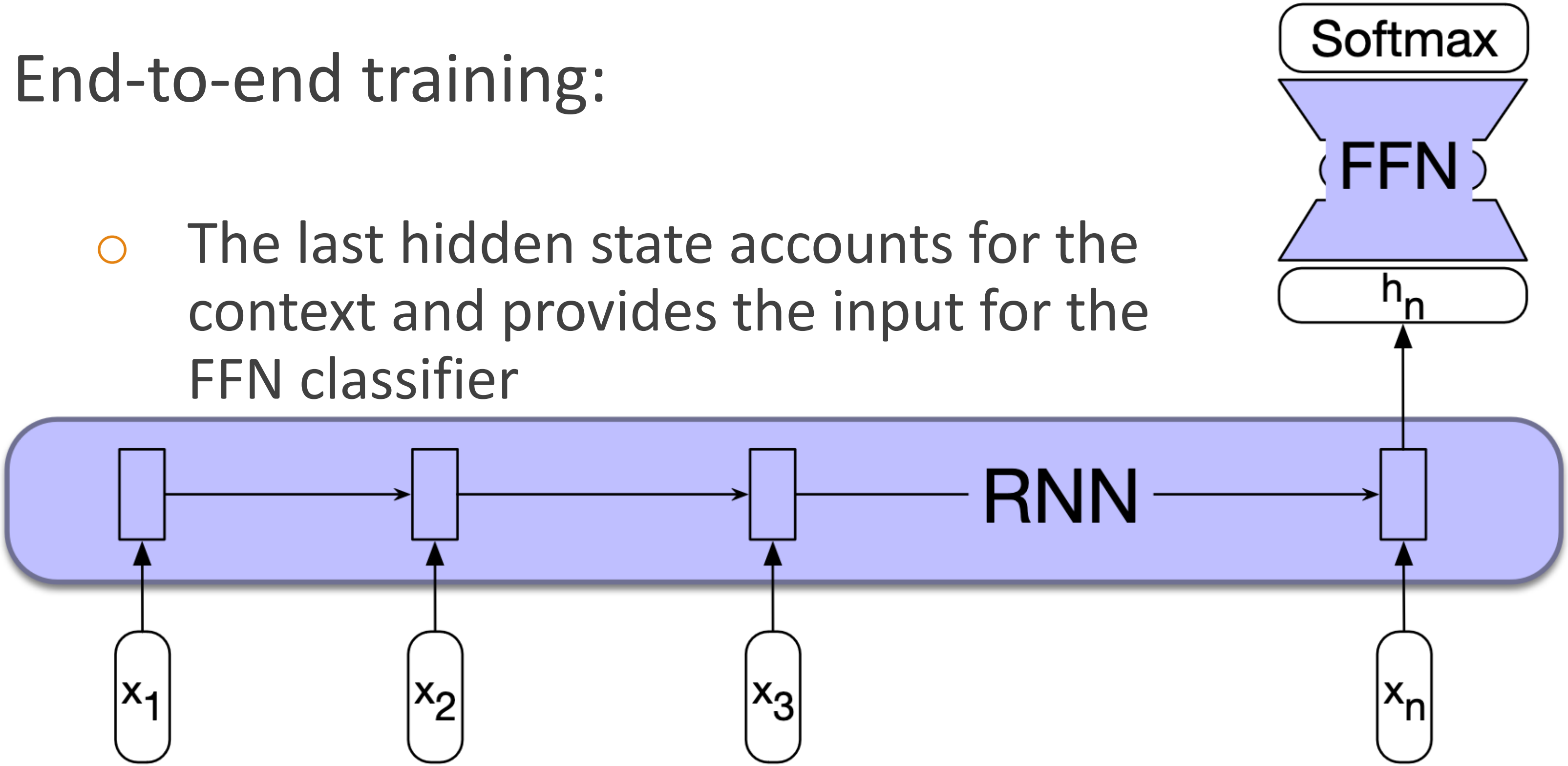
Classifying entire sequences rather than the tokens within them

- Also called *text classification*
  - Sentiment analysis
  - Spam detection
  - Hate/Gender/Political speech detection
  - Document-level topic classification
  - …

Sequence classification

End-to-end training:

- The last hidden state accounts for the context and provides the input for the FFN classifier

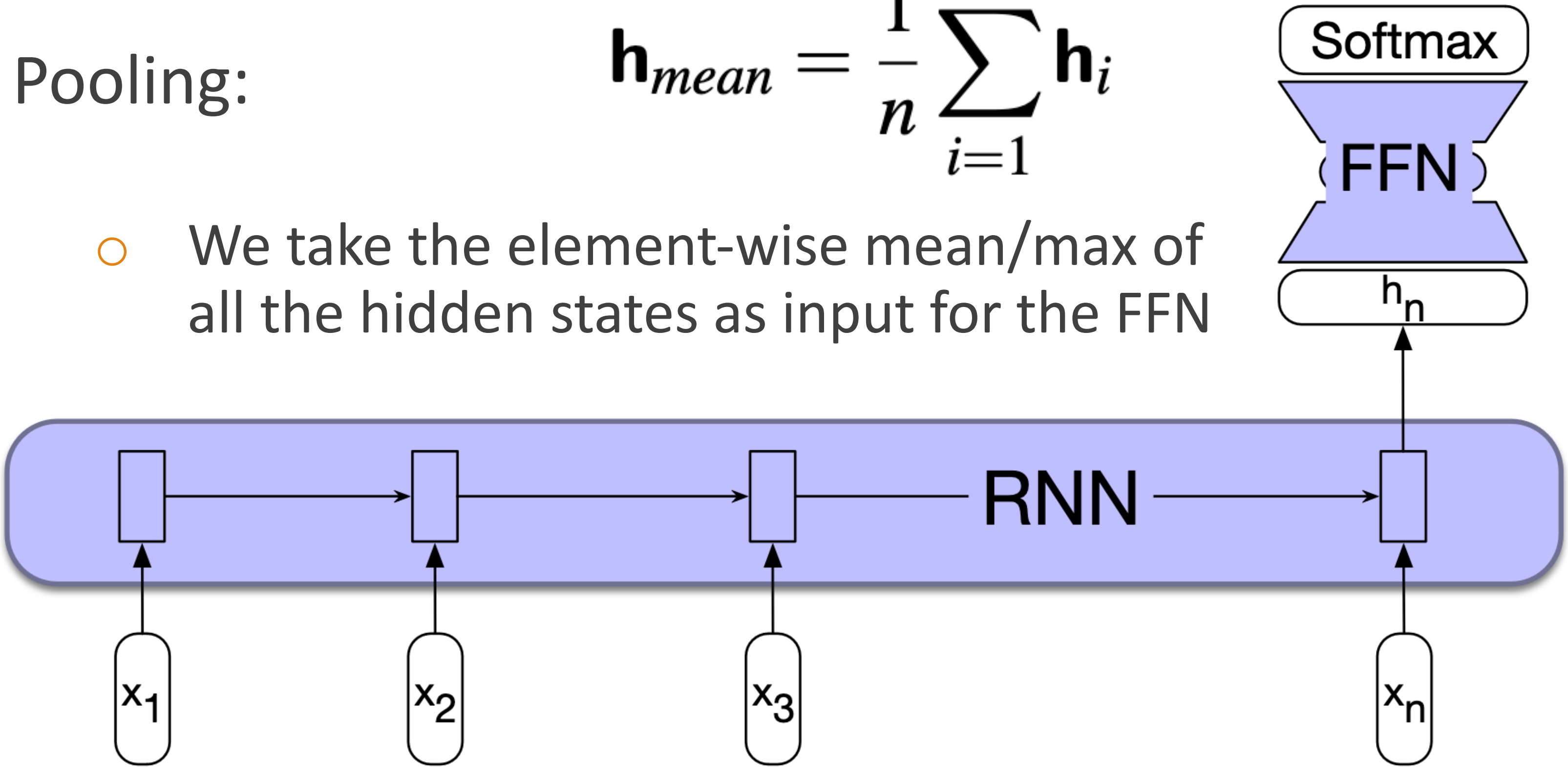


Sequence classification

Pooling:

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

- We take the element-wise mean/max of all the hidden states as input for the FFN



Text generation

Text generation is part of all the tasks where a system needs to produce text, conditioned on some other text

- Question answering
- Machine translation
- Text summarization
- Grammar correction
- Story generation
- Conversational dialogue

Text generation

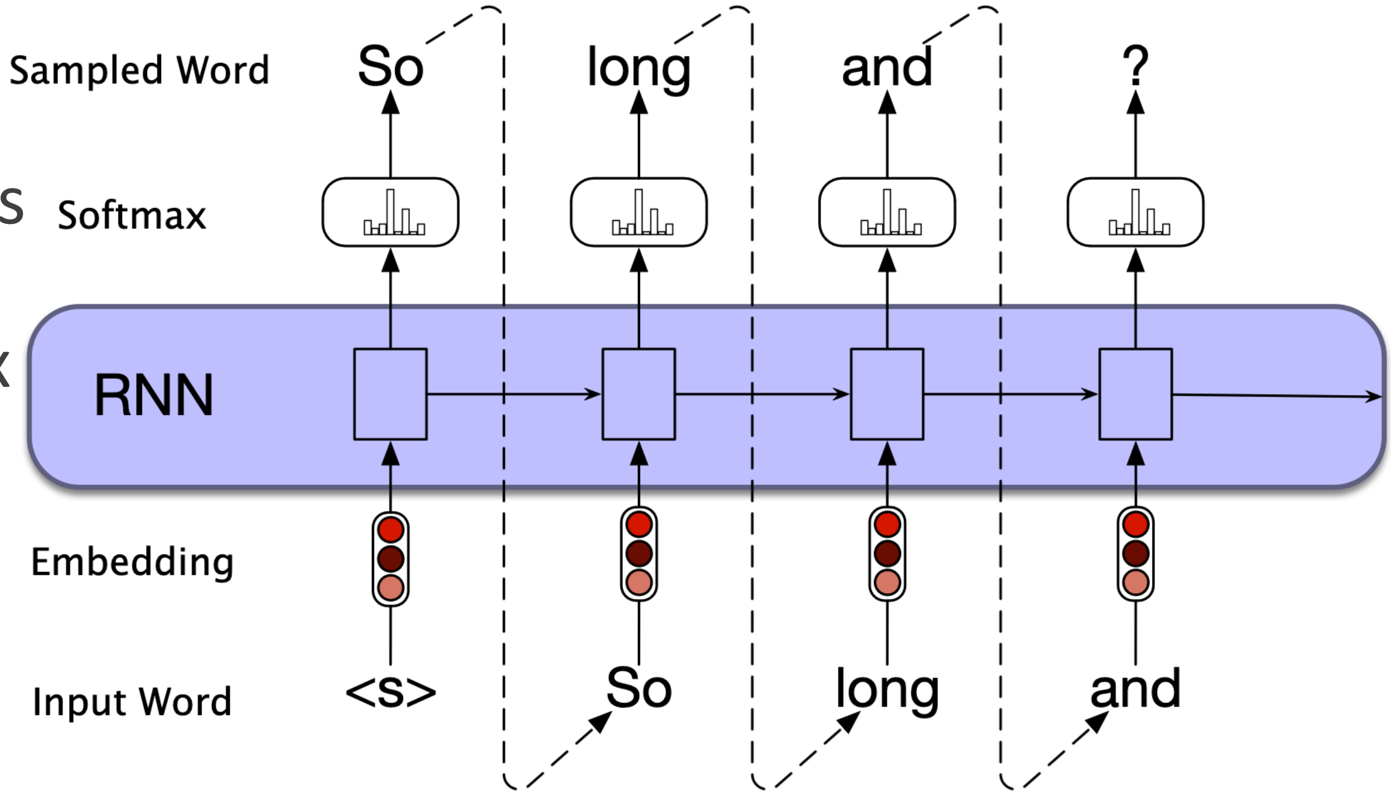
Recall the Shannon game

- A word is sampled based on its probability of being a start word i.e. $P(w|\text{<s>})$
- Each word in the sentence is sampled conditioned on the previous choiches $P(w_i|\ w_{1:i-1})$

In neural language models, sampling is adapted to the RNN architecture

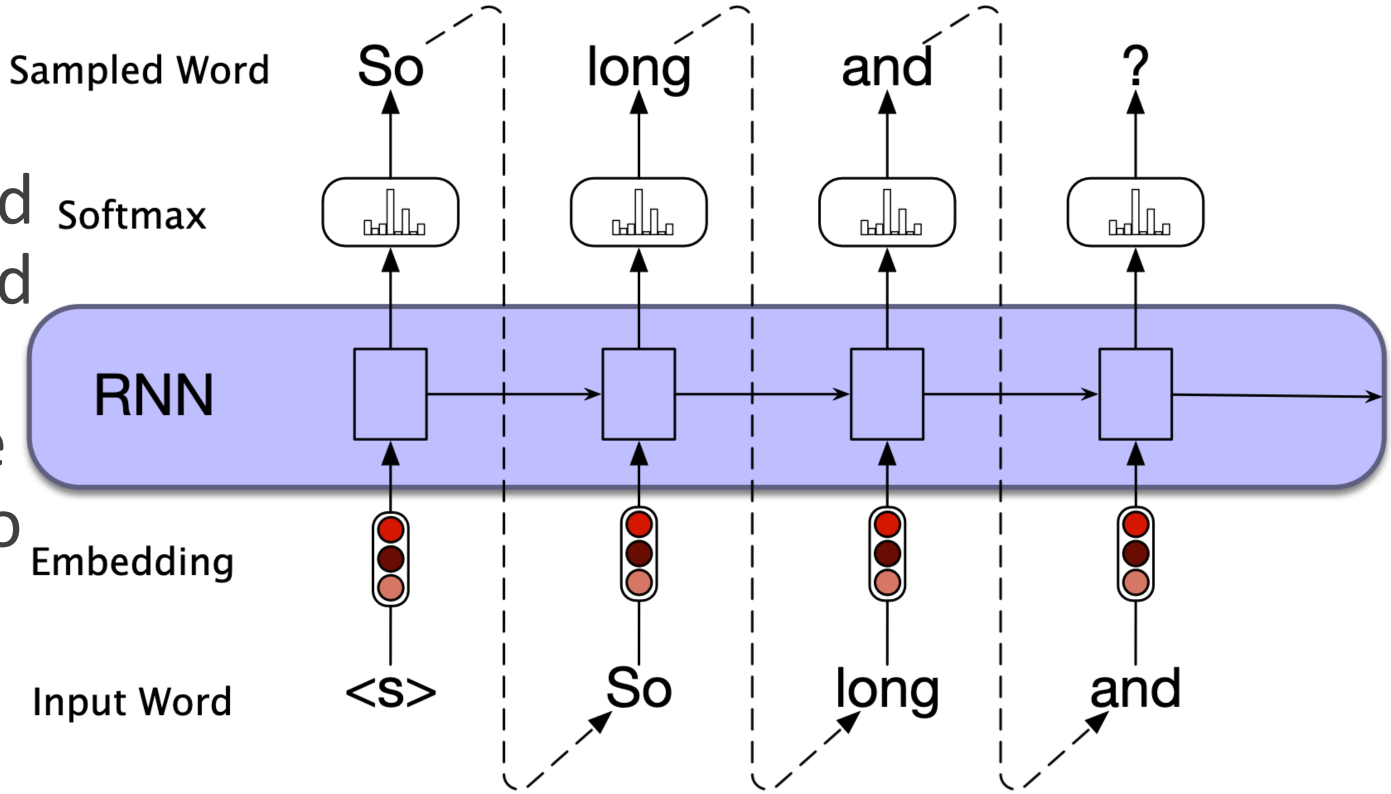
Text generation

Each word is sampled as the softmax outcome from the previous one



Text generation

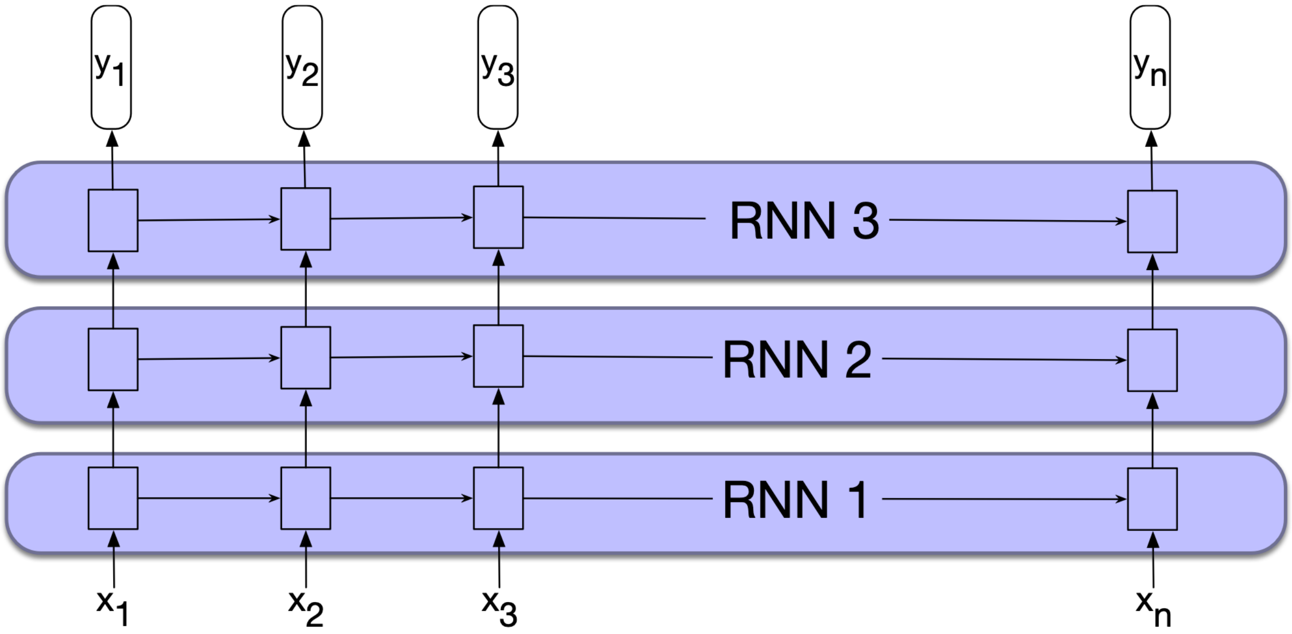
The sampled output is fed to the next input of the RNN, and so on



Stacked and Bidirectional RNN Architectures

Stacked RNNs

The output sequence of a layer is the input for the next one



Bidirectional RNNs

RNNs use information coming from *left context* at step *t*

- The hidden state $\mathbf{h}_t$ at step t is a forward function of the context $\mathbf{x}_1 \dots \mathbf{x}_t$

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

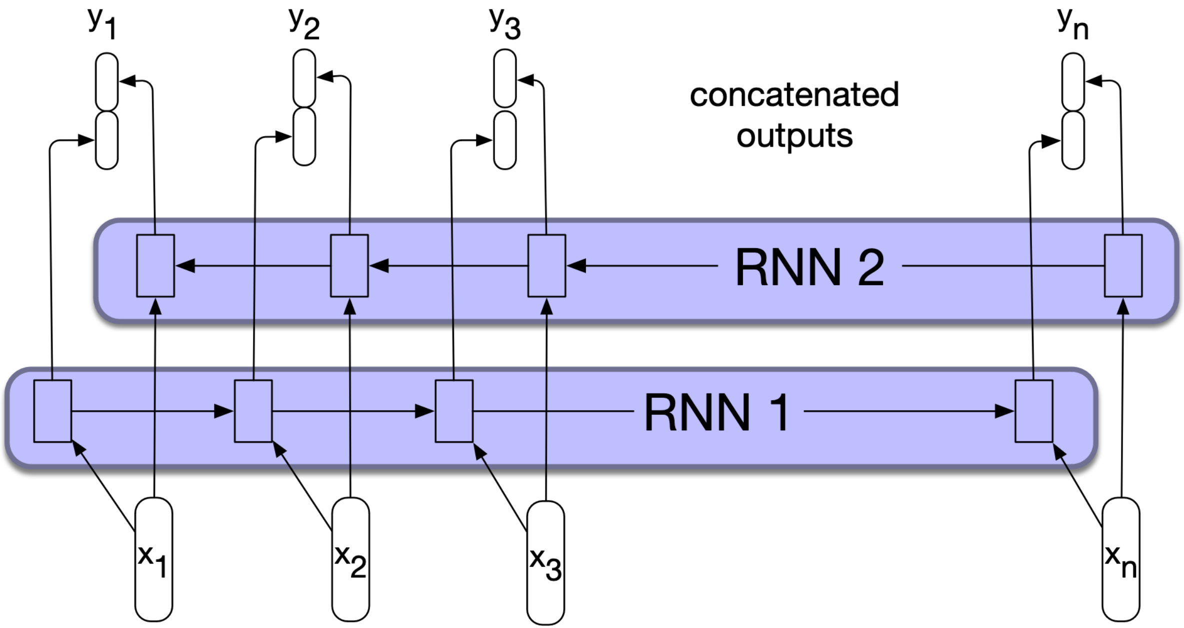
Often it is useful obtain information also from the *right context*

Bidirectional RNNs

Bidirectional RNNs are trained in forward and backward mode

Forward and backward output are then concatenated

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

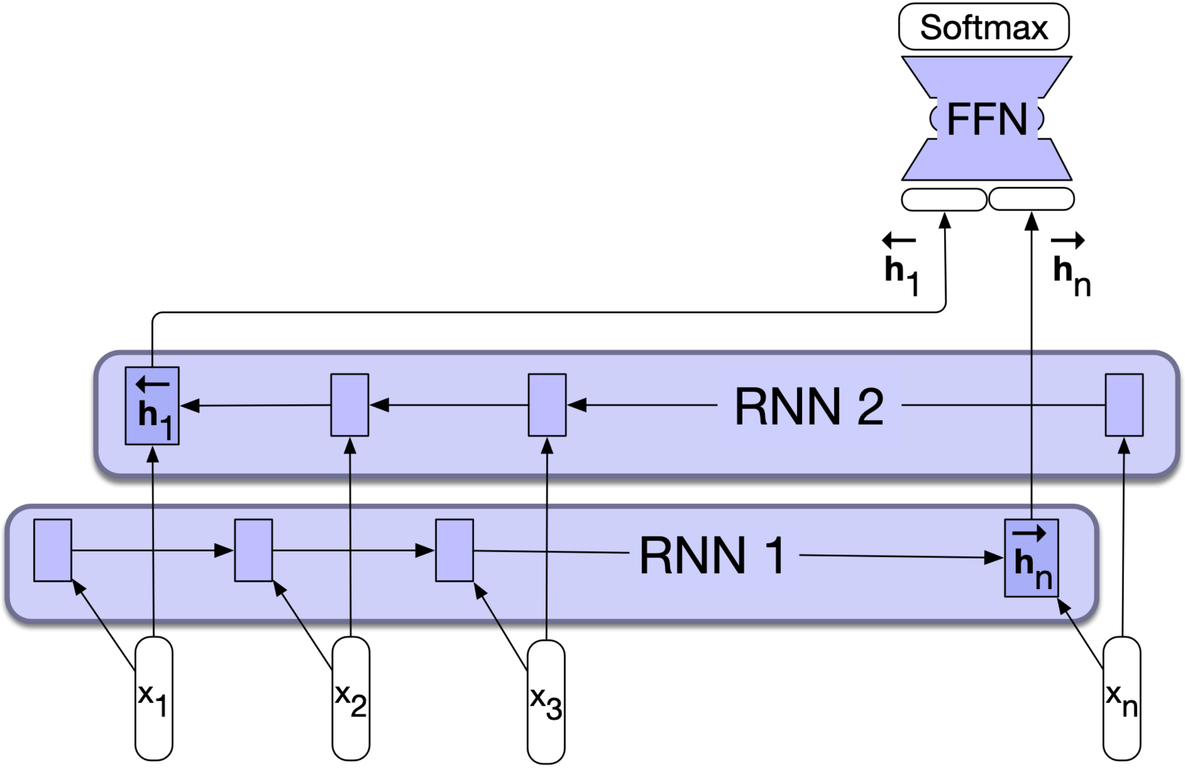


Bidirectional RNNs

In text classification

$\mathbf{h}_1{}^b \oplus \mathbf{h}_n{}^f$

is passed to the FFN classifier



Recurrent Neural Networks and Attention

The Long Short-Term Memory (LSTM)

Distant information

RNNs are not good in dealing with distant words dependencies

The flights the airline was cancelling were full

Hidden layers suffer from *vanishing gradients* in the backward pass of training

Gates

LSTMs divide the text management problem into two subproblems:

- Removing information no longer needed from the context
- Adding information likely to be needed for later decision making

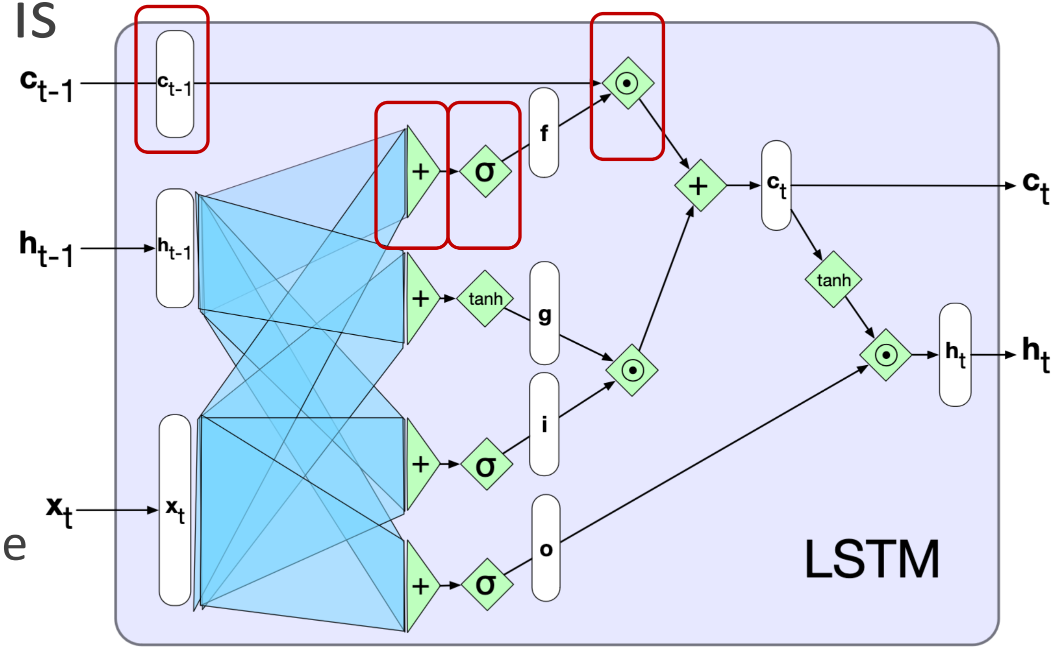
Gates are neural units designed to learn context management

Gates

A new context layer $\mathbf{c}_t$ is added

A *gate* is made by:

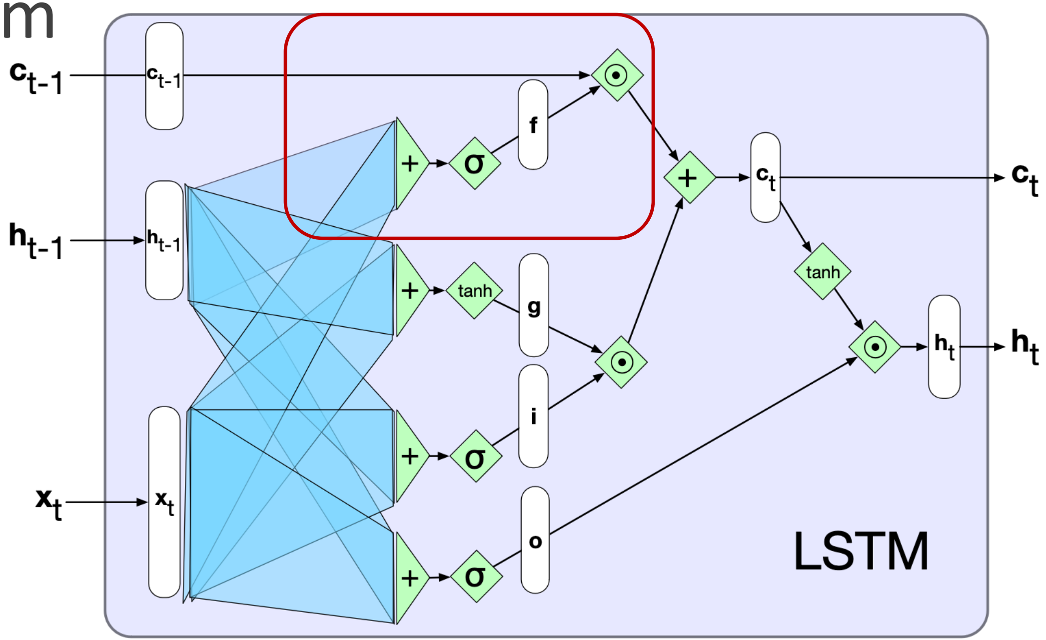
- A feed-forward layer
- A sigmoid activation
- A point-wise multiplication with the layer to be gated



Forget Gate

Delete information from the previous context that is no longer needed

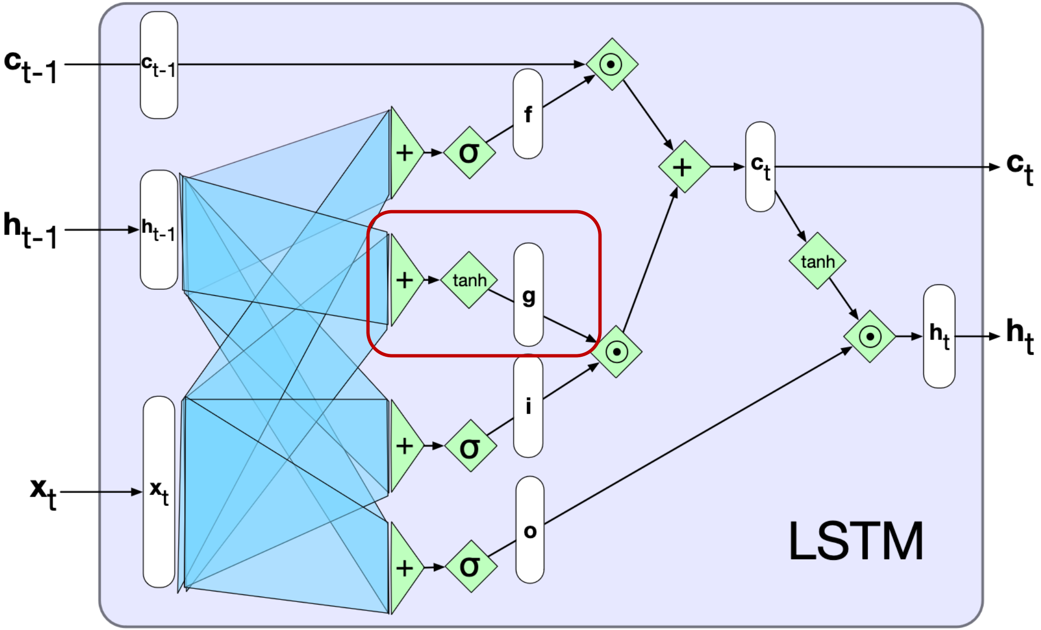
$$\mathbf{f}_t = \sigma(\mathbf{U}_f \mathbf{h}_{t-1} + \mathbf{W}_f \mathbf{x}_t)$$
$$\mathbf{k}_t = \mathbf{c}_{t-1} \odot \mathbf{f}_t$$



Extract information at step *t*

Compute information from the previous hidden state and current inputs

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



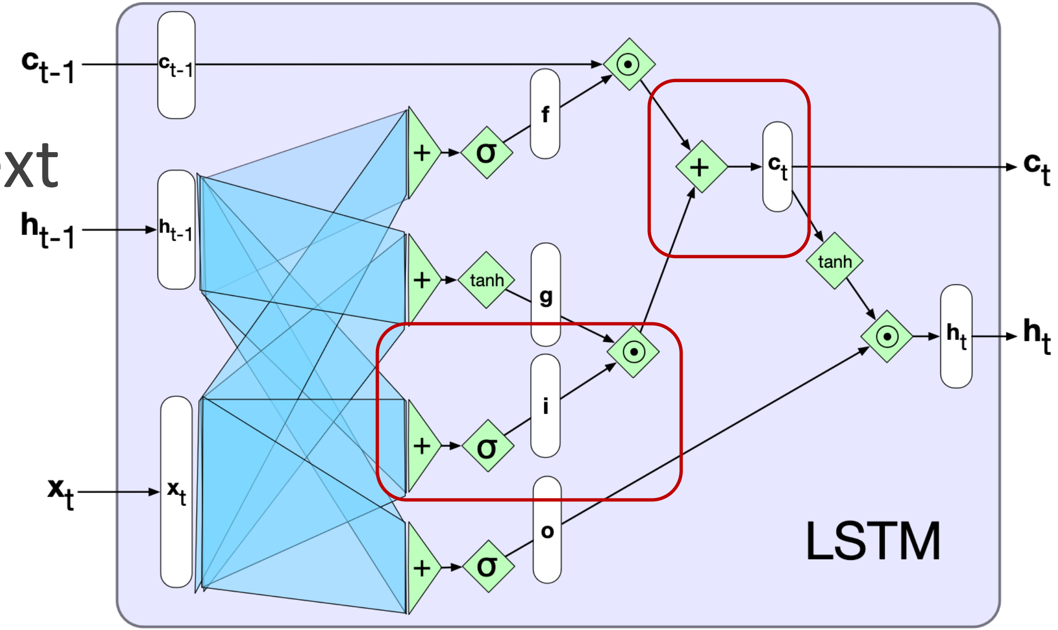
Add gate

Select the relevant information, and add it to the current context

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

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

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

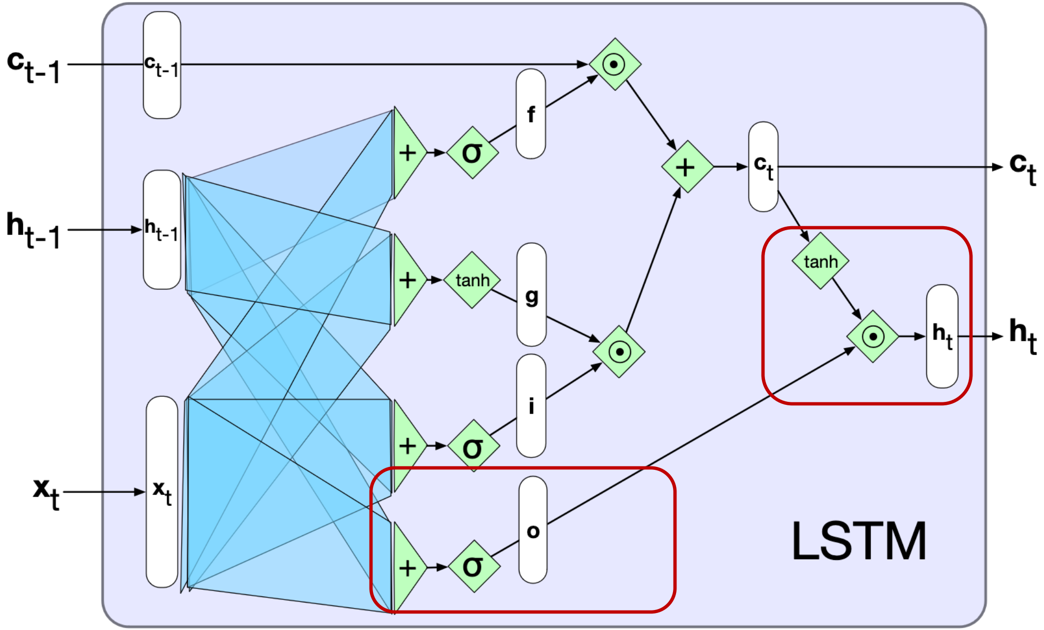


Output gate

Decide what is the required information for the current hidden state

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

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t)$$

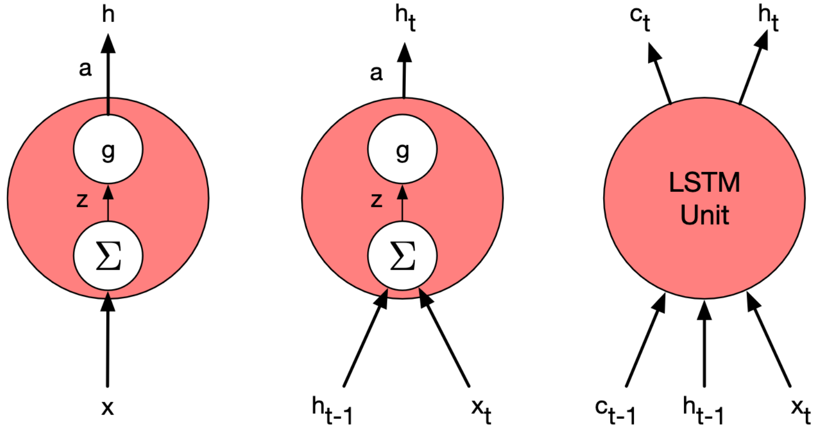


LSTM networks

Increasing complexity from FFN to RNN and LSTM

The context vector is added as both input and output w.r.t. the RNN unit

Stacked networks and plain backpropagation through unrolled computational graph are still possible with LSTM units



Recurrent Neural Networks and Attention

The Encoder-Decoder Model with RNNs

Encoder-decoder model

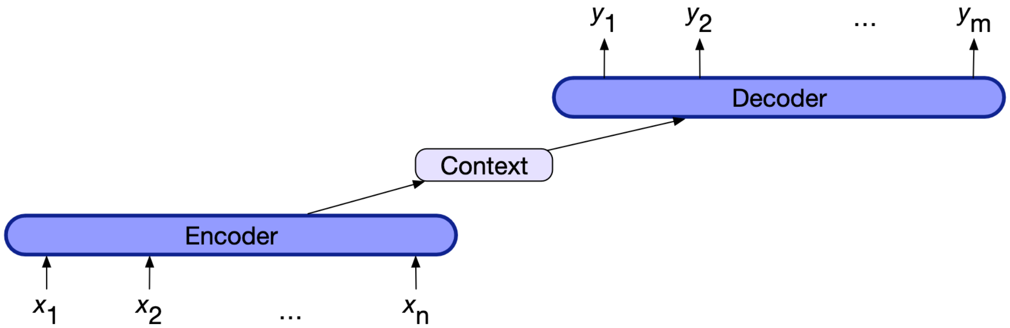
Tasks like machine translation are still sequence labeling tasks but

- Input and output sequence do not have the same length
- Mapping between input/output token pairs can be very indirect
  - Verbs have different positions in different languages

Encoder-decoder model

The encoder-decoder (a.k.a. sequence-to-sequence) networks cope with this problem

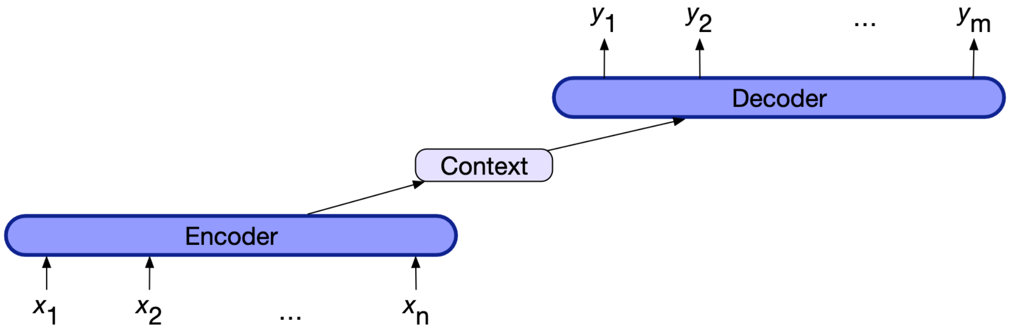
- Encoder: CNN/LSTM/Transformer mapping $x_{1:n} \rightarrow h_{1:n}$



Encoder-decoder model

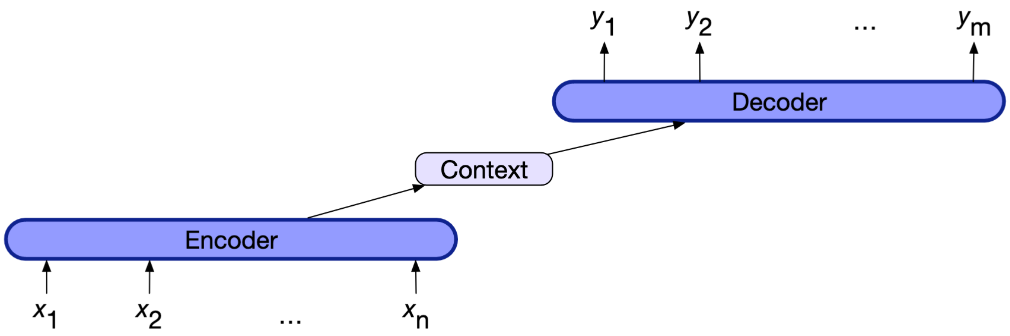
The encoder-decoder (a.k.a. sequence-to-sequence) networks cope with this problem

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



Encoder-decoder model

The encoder-decoder (a.k.a. sequence-to-sequence) networks cope with this problem



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

Encoder-decoder networks using RNNs

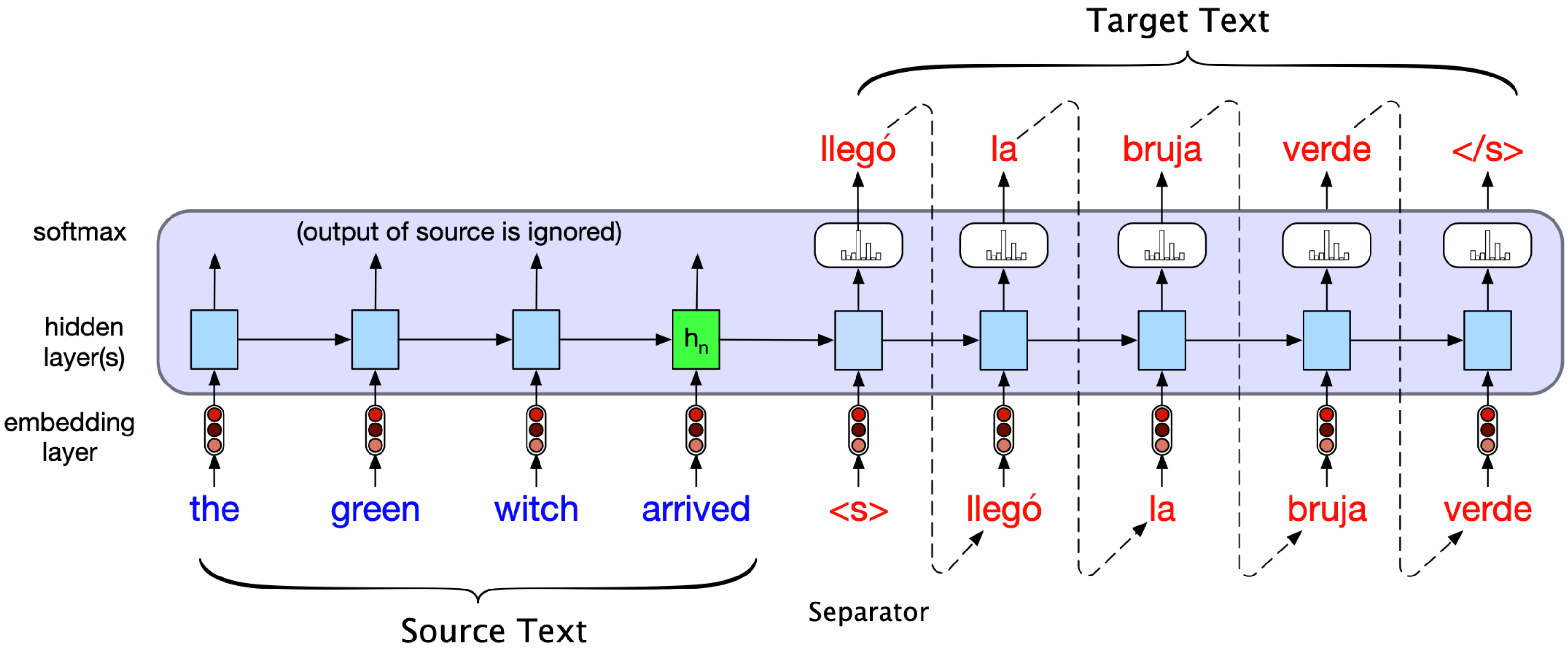
Recall language modeling with autoregressive generation

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

- We start with a suitable <s> token
- At each step t, $\mathbf{x}_t \equiv \mathbf{y}_{t-1}$

Each Hidden state contains information about the $\mathbf{x}_{1:t-1}$ context

Encoder-decoder networks using RNNs

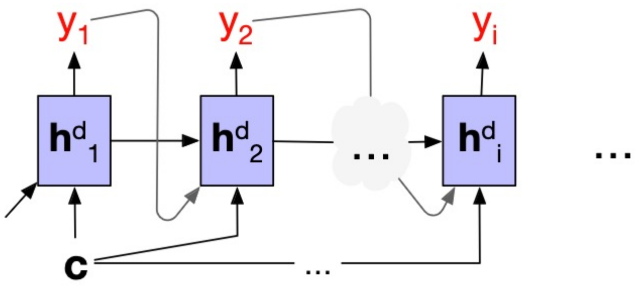


Inference

Encoder-decoder networks using RNNs

$\mathbf{h}_n \equiv \mathbf{h}^e_n$ is the context $\mathbf{c}$ in this model

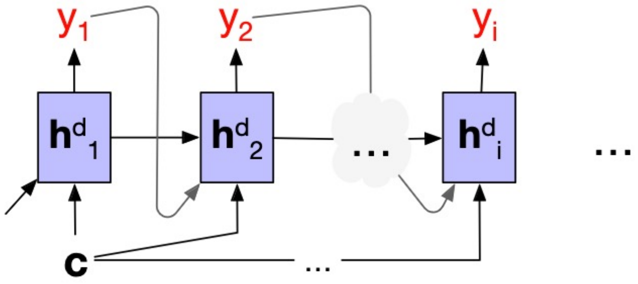
- It is passed *only* to the first decoder's hidden state $\mathbf{h}^d_1$
- It wanes as the output sequence is being generated
- Idea! Pass **c** to *all* the hidden states



Encoder-decoder networks using RNNs

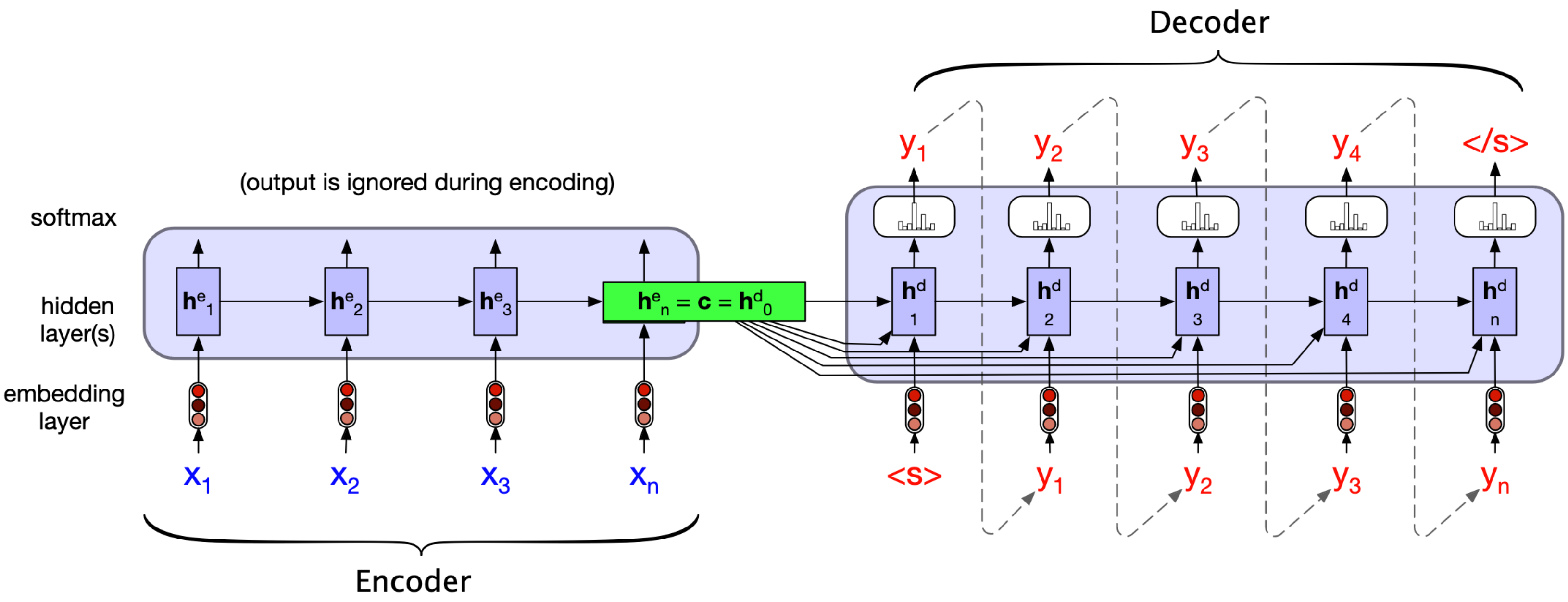
$$\mathbf{c} = \mathbf{h}_n^e$$
$$\mathbf{h}_0^d = \mathbf{c}$$
$$\mathbf{h}_t^d = g(\boxed{\hat{y}_{t-1}}, \mathbf{h}_{t-1}^d, \mathbf{c})$$
$$\mathbf{z}_t = f(\mathbf{h}_t^d)$$
$$y_t = \text{softmax}(\mathbf{z}_t)$$
$$\hat{y}_t = \text{argmax}_{w \in V} P(w|x, y_1 \ldots y_{t-1})$$

The word embedding for the output sampled from the softmax at the previous step



We sample the output embedding taking the argmax *over the softmax output*

Encoder-decoder networks using RNNs



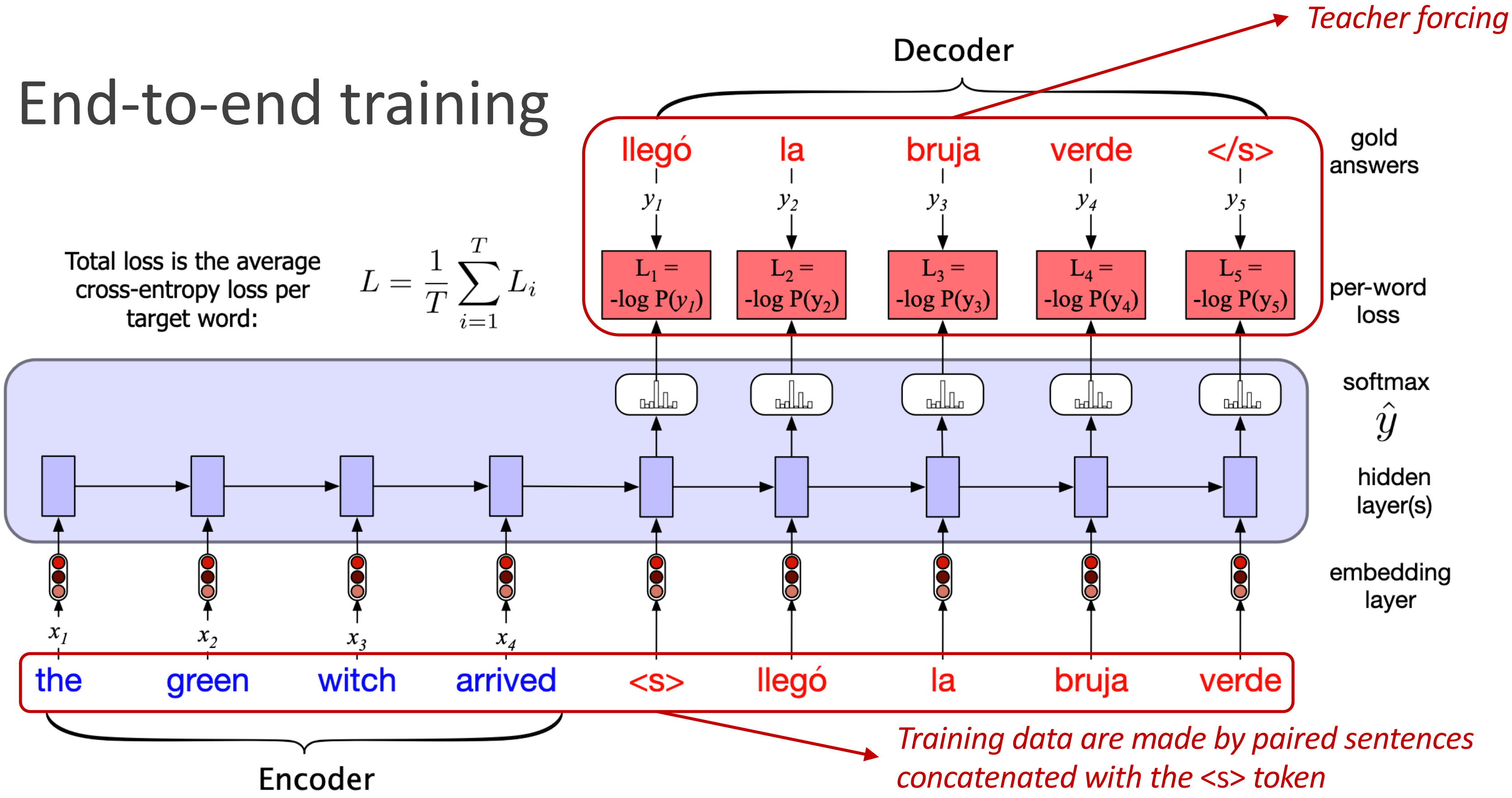
Inference

Training the encoder-decoder model

End-to-end training

Total loss is the average cross-entropy loss per target word:

$$L = \frac{1}{T}\sum_{i=1}^{T} L_i$$



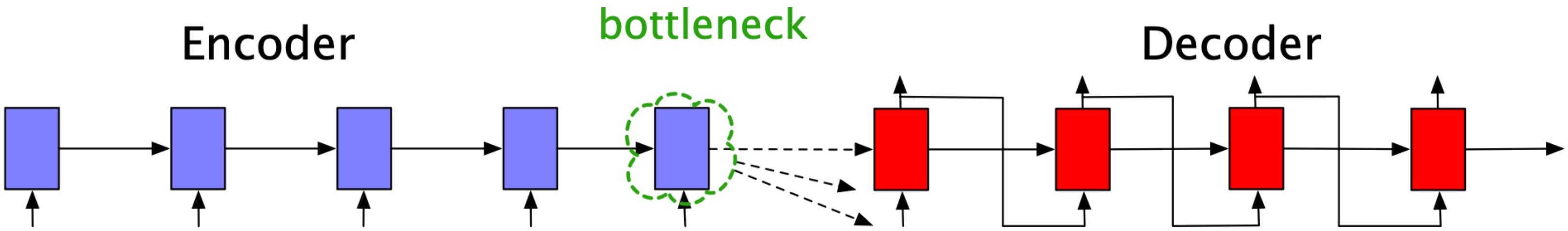
Recurrent Neural Networks and Attention

Attention

The context bottleneck

c has to represent *all* the information coming from the source text

Long distance dependencies may be not well represented



Attention mechanism

Getting information from all the hidden states of the encoder $\mathbf{c} = f(\mathbf{h}_1^e \ldots \mathbf{h}_n^e)$

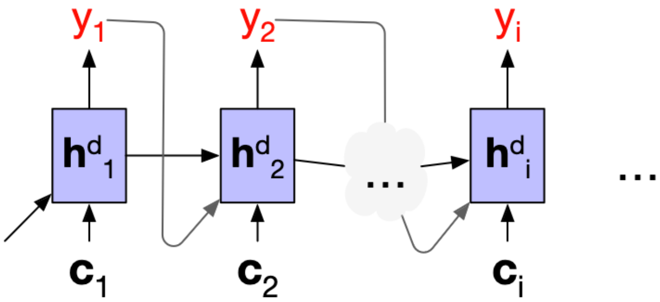
- Fixed-length vector computed as a weighted sum of all the encoder hidden states

- Weights focus on (*attend to*) a particular part of the source text that is relevant for the token the decoder is currently producing

Attention mechanism

The context vector is generated anew with each decoding step *i*

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



We start computing a set of scores measuring how relevant each encoder hidden state is for the decoder hidden state at step *i-1*

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

Dot-product attention

The simplest score is the dot product

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

- Dot product is a scalar that reflects the degree of similarity between the two vectors
- $\mathbf{h}^e{}_j$ and $\mathbf{h}^d{}_{i-1}$ must have the same dimensionality

Attention score with trainable weights

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

- Weights $\mathbf{W}_s$ are trained during normal end-to-end training
- The network learns which aspects of similarity between the decoder and encoder states are important to the current application
- $\mathbf{h}^e{}_j$ and $\mathbf{h}^d{}_{i-1}$ can have different dimensionality

Weights of the context vector

We use the softmax to normalize the scores, and computing the actual weights α_{ij}

$$\begin{aligned}
\boldsymbol{\alpha}_{ij} &= \text{softmax}(\text{score}(\mathbf{h}^d_{i-1}, \mathbf{h}^e_j) \ \forall j \in e) \\
&= \frac{\exp(\text{score}(\mathbf{h}^d_{i-1}, \mathbf{h}^e_j)}{\sum_k \exp(\text{score}(\mathbf{h}^d_{i-1}, \mathbf{h}^e_k))} \\
\mathbf{c}_i &= \sum_j \alpha_{ij}\, \mathbf{h}^e_j
\end{aligned}$$

Encoder-decoder network with attention