::ntic master
revolucionamos la comunicación

UNIVERSIDAD COMPLUTENSE DE MADRID

Recurrent Neural Networks

Analytical Index

Recurrent Neural Networks (RNN)

Introduction to Sequence Modelling

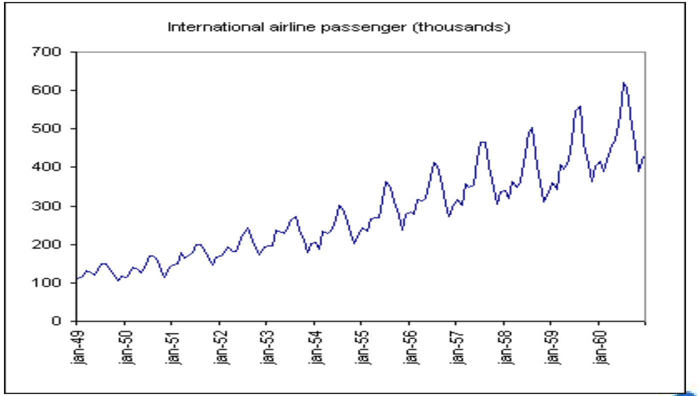
Sequences

- Most of machine learning algorithms are designed for independent, unordered data.

- Many real problems uses sequential data:
  $\mathbf{x}^{(t)} : \{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \dots, \mathbf{x}^{(T)}\}$
  - Time series, biomedical signals, audio signals, stock prices, text…
  - Does not have to be time, can be spatial measure (images), or any order measure (Recommender systems)

- The sequences are a natural way of representing reality: vision, hearing, action-reaction, words, sentences, etc.

- Don’t forget order matters!!



Sequential Processing

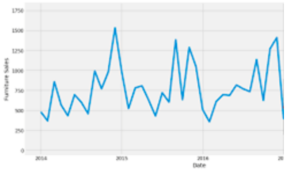
Speech recognition:

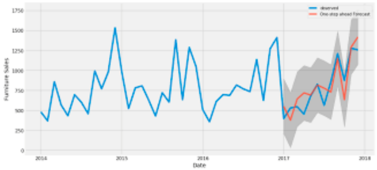
Deep learning course   D e e p l e a….

Sentiment analysis:

“Really good book…”

Time series forecasting:



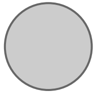


Machine translation

“Really good book…”

“Muy buen libro…”

Music generation:



Problems With Dense Networks

- Inputs and outputs can be of different lengths:
    - Can try windowing.

“I went to Greece on an end-of-term trip, …, in 2020”

- No parameter sharing:
    - Try convNets.

“I went to Greece on an end-of-term trip, …, in 2020”
“In 2020, I went to Greece on an end-of-term trip, …, ”

- Long Term dependencies (memory):

“I went to Greece on an end-of-term trip, …, in 2020”
“In 2020, I went to Greece on an end-of-term trip, …, ”

Sequence Notation

“The film is not good it’s bad”

The	Film	is	not	good	it’s	bad
$x^{(1)}$	$x^{(2)}$	$x^{(3)}$	$x^{(4)}$	$x^{(5)}$	$x^{(6)}$	$x^{(7)}$

$$Sentiment = f\left(x^{(1)}, x^{(2)}, \cdots, x^{(7)}\right)$$

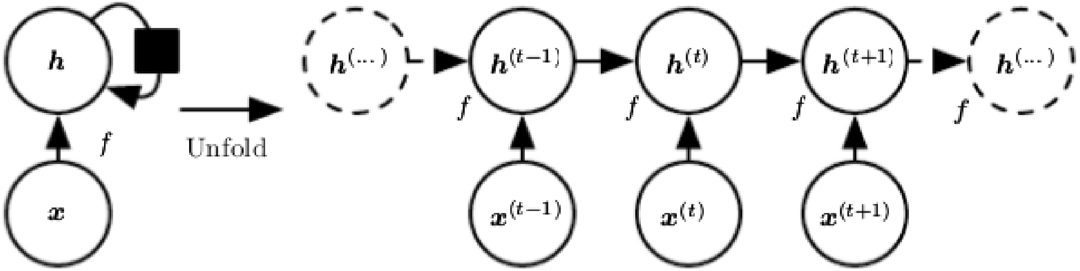
Order is important: *“The film is good it’s not bad”* (same words different order)

Recurrent relations

- Recurrent function: $x^{(t)} = f\left(x^{(t-1)}\right) = f\left(f\left(x^{(t-2)}\right)\right) = \ldots = g\left(x^{(t-1)}, \ldots, x^{(0)}\right)$

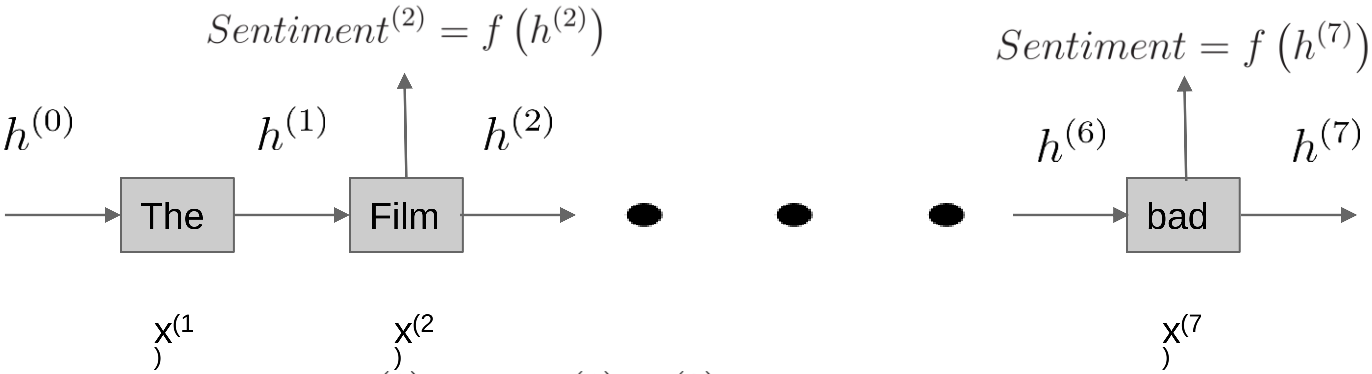
- If we want to predict future from past:
  - Repetitive structure
  - Don't need to learn g.
  - Same transition function f with the same parameters at every time step

$$\begin{aligned} \boldsymbol{h}^{(t)} &= g^{(t)}\left(\boldsymbol{x}^{(t)}, \boldsymbol{x}^{(t-1)}, \boldsymbol{x}^{(t-2)}, \ldots, \boldsymbol{x}^{(2)}, \boldsymbol{x}^{(1)}\right) \\ &= f(\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)}; \boldsymbol{\theta}) \end{aligned}$$



Recurrence Intuition

"The film is not good it's bad"



$$h^{(2)} = g(h^{(1)}, x^{(2)})$$

- The state (memory) of every step is updated with the previous state and the actual input.

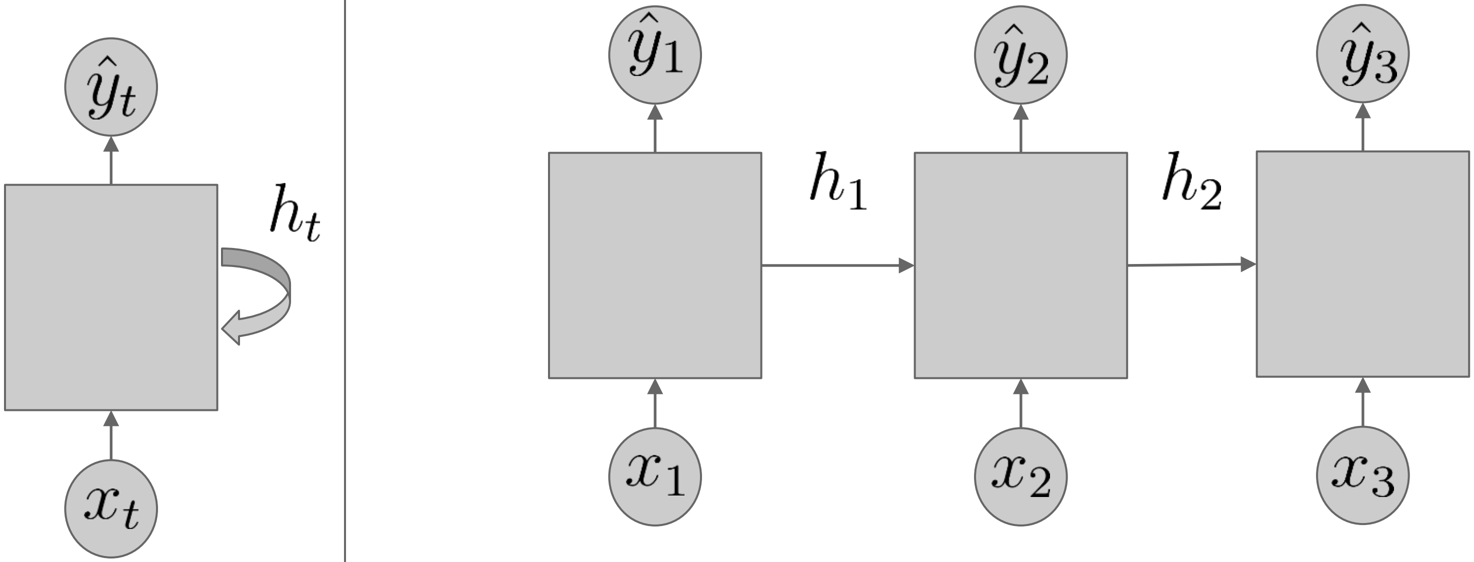
$$h^{(t)} = g(h^{(t-1)}, x^{(t)})$$

- The sentiment is a function of the state.

$$Sentiment^{(t)} = f\left(h^{(t)}\right)$$

Simple RNN

RNN Intuition



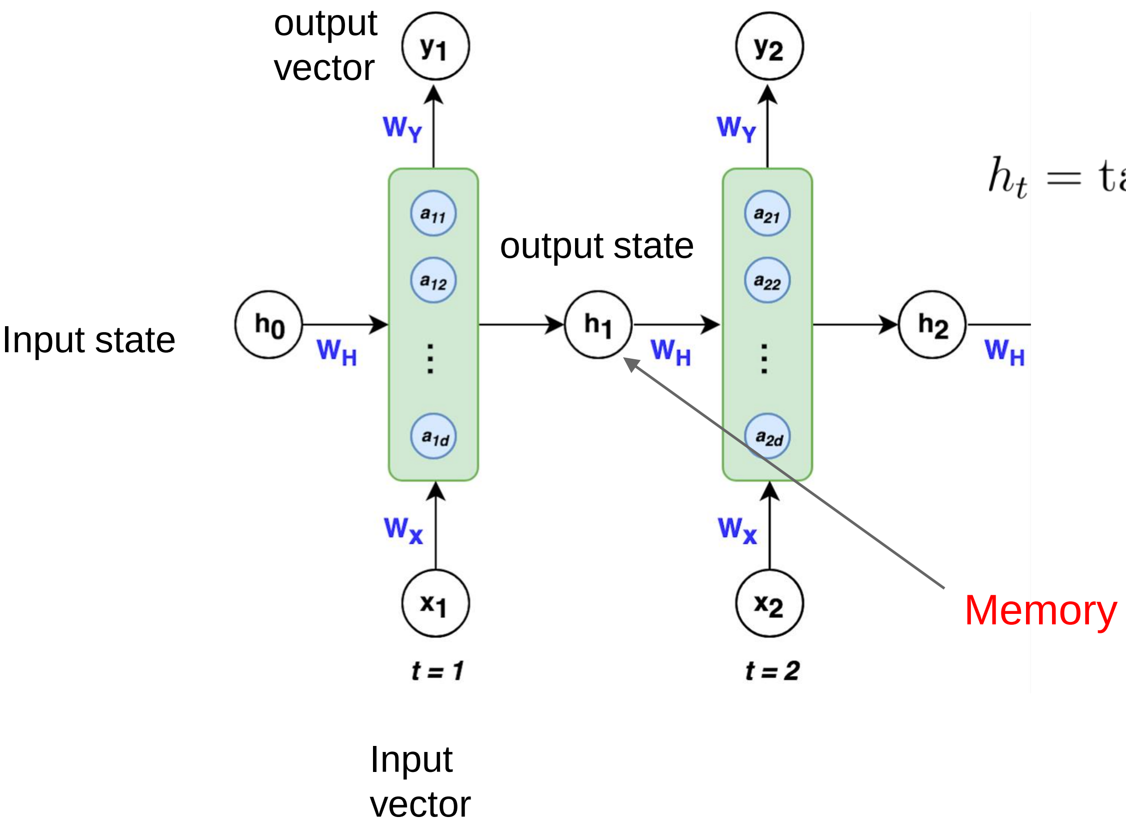
$$\hat{y}_t = f(h_{t-1}, x_t)$$

Output

Previous state

input

Simple RNN



Update hidden state:

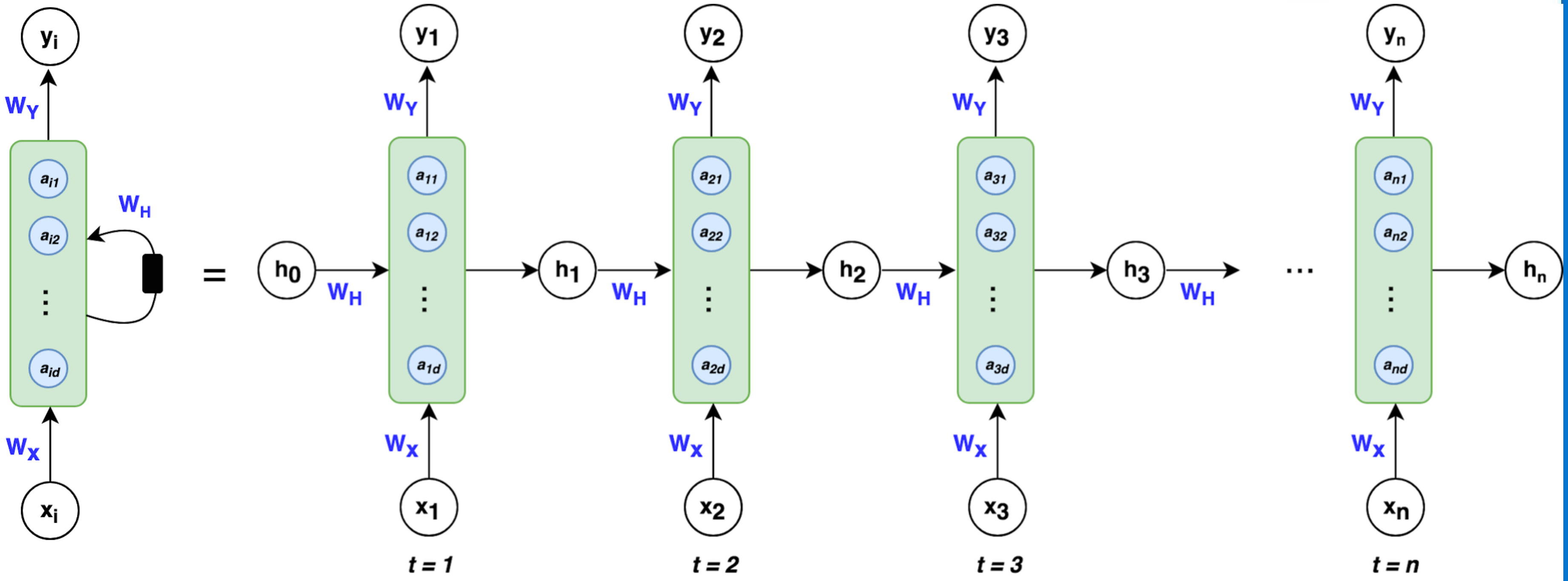
$$h_t = \tanh\left(W_{hh}h_{t-1} + W_{hx}x_t + b_h\right)$$

Output:

$$y_t = g\left(W_{yh}h_t + b_y\right)$$

Simple RNN

$$y_t = g\left(W_{yh}h_t + b_y\right)$$



$$h_t = \tanh\left(W_{hh}h_{t-1} + W_{hx}x_t + b_h\right)$$

Example: Language Model

Sentence 1: “La noche larga” => P(“La noche larga”) = 0.001

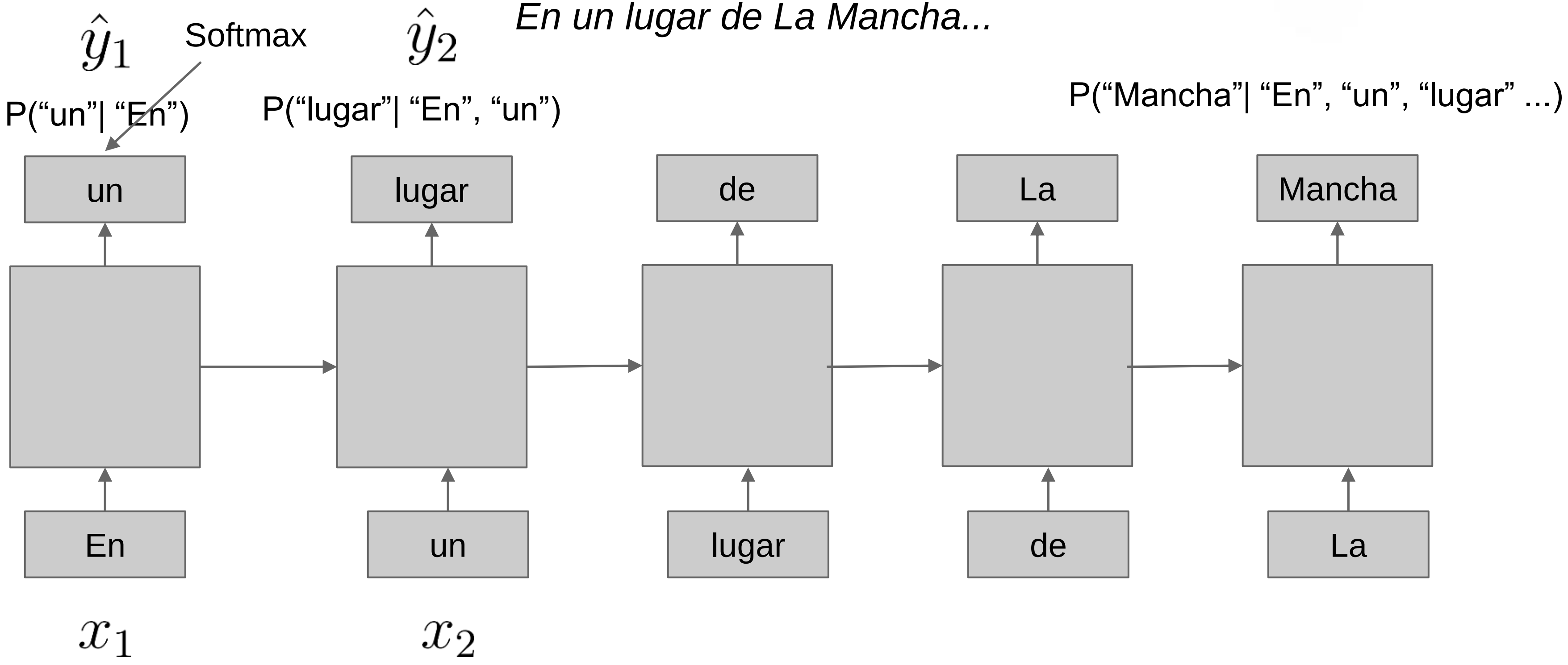
Sentence 2: “La coche larga” => P(“La coche larga”) = 0.0000000001

Model: P(“La noche larga”) = P(“La”)P(“noche” | “La”)P(“larga” | “La”, “noche”)

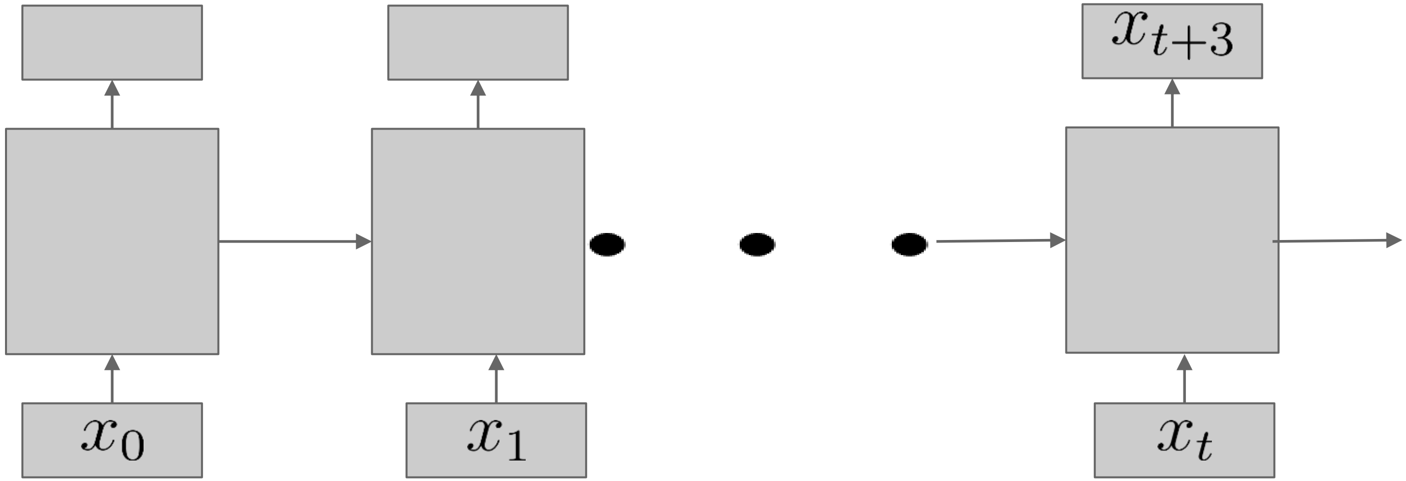
$$p(w_0, w_1, \cdots, w_n) = p(w_0)p(w_1|w_0)p(w_2|w_0, w_1)\cdots p(w_n|w_0, w_1\cdots, w_{n-1})$$

Example: Predicting Next Word

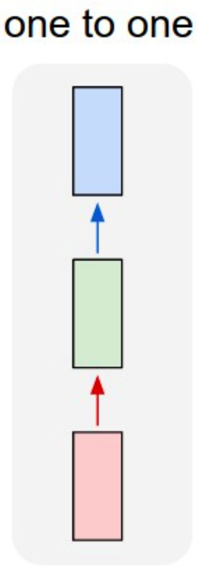
En un lugar de La Mancha...



Example: Time Series Forecasting

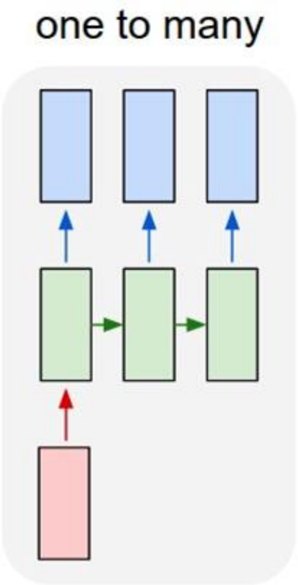


Vanilla RNN



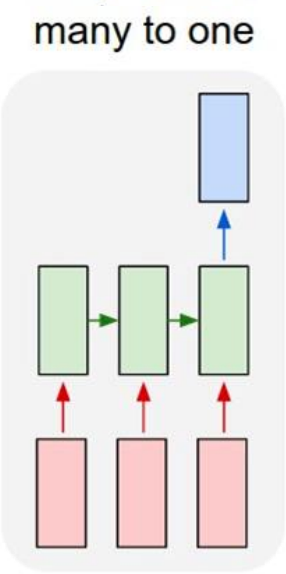
- Simple dense net

Simple RNN



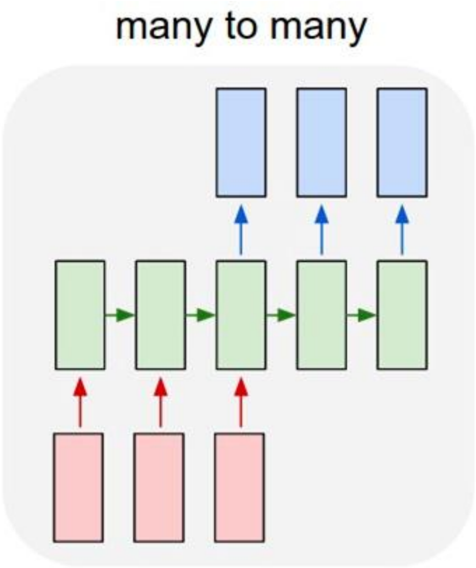
- Music generation
- Text generation
- Image captioning

Simple RNN



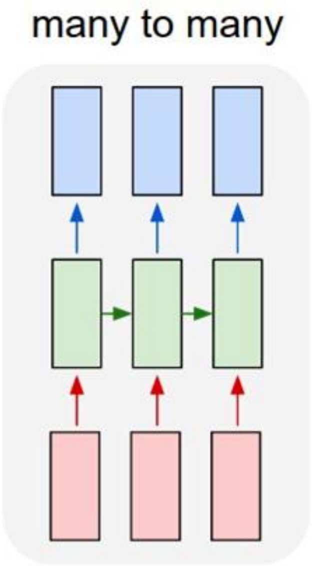
- Sentiment analysis.

Simple RNN



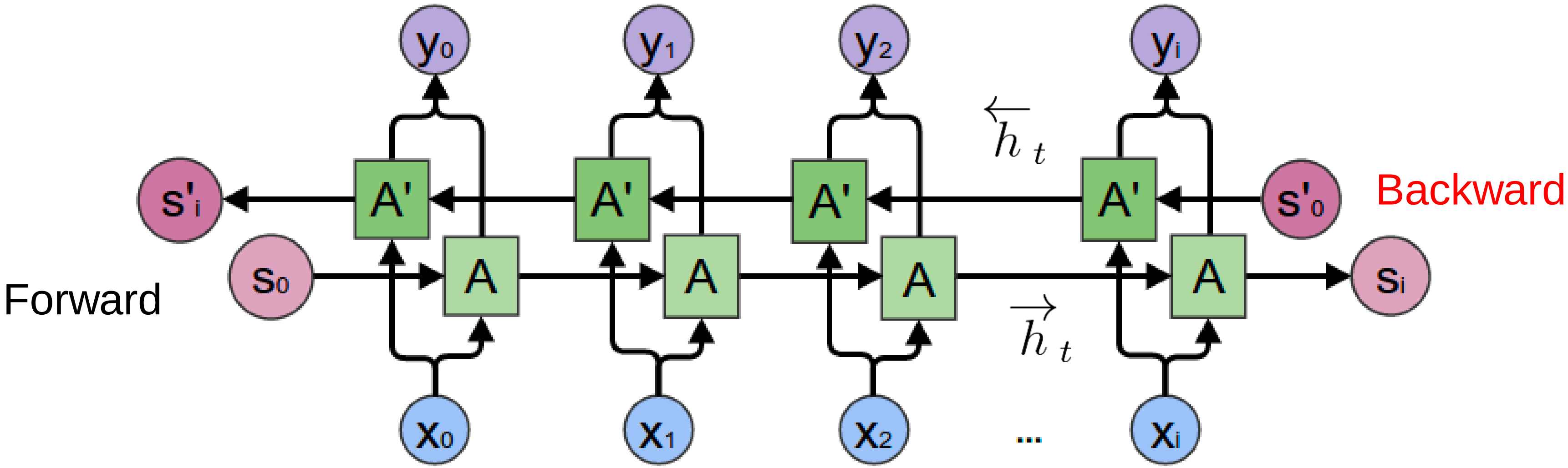
- Machine translation

Simple RNN



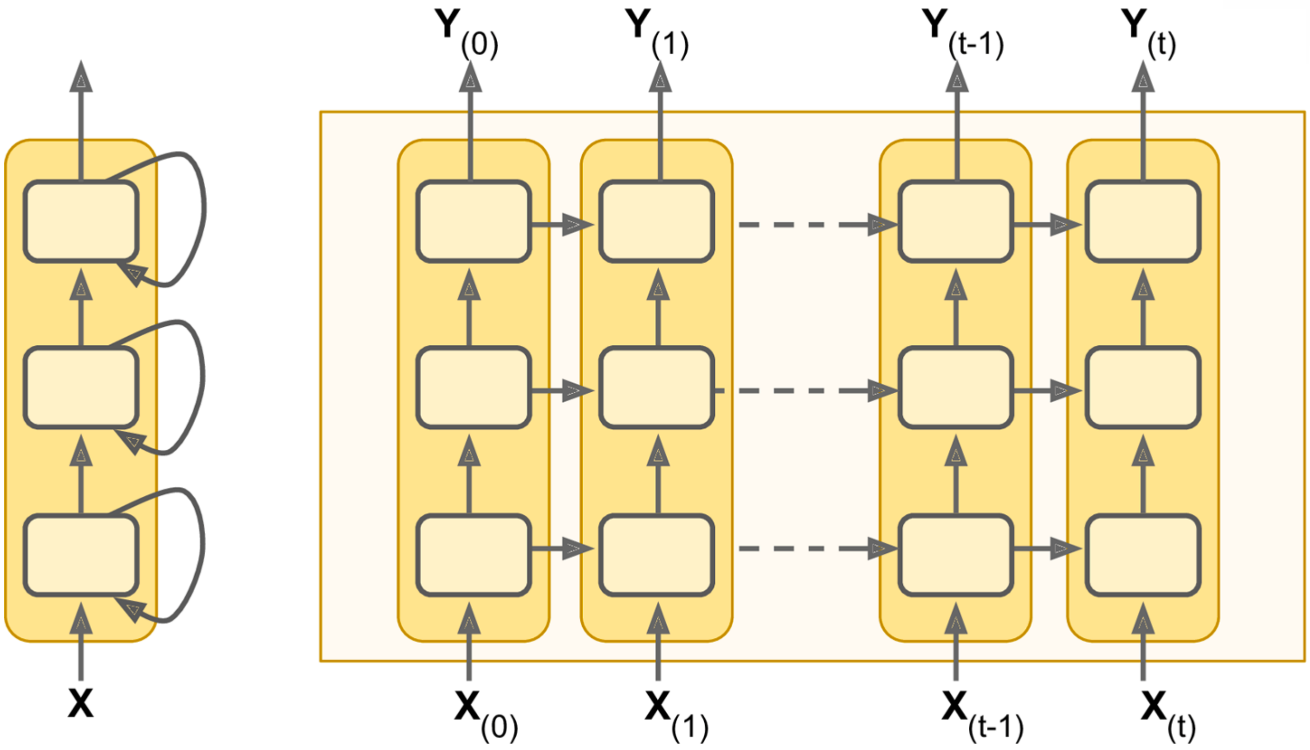
- Video classification

Bidirectional RNN



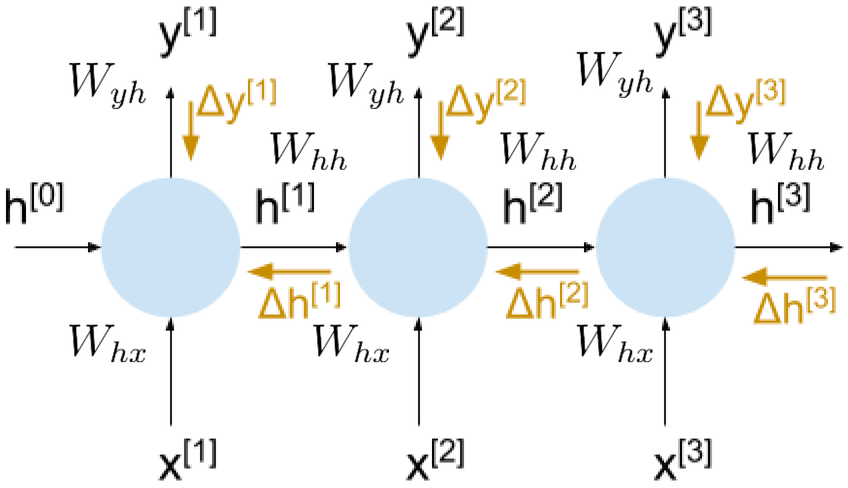
I wanted ______ to call me. Mario never called. => Him or Her ??

Deep RNN



Stack multiple recurrent layers for learning more complex functions

Backpropagation Through Time BTT

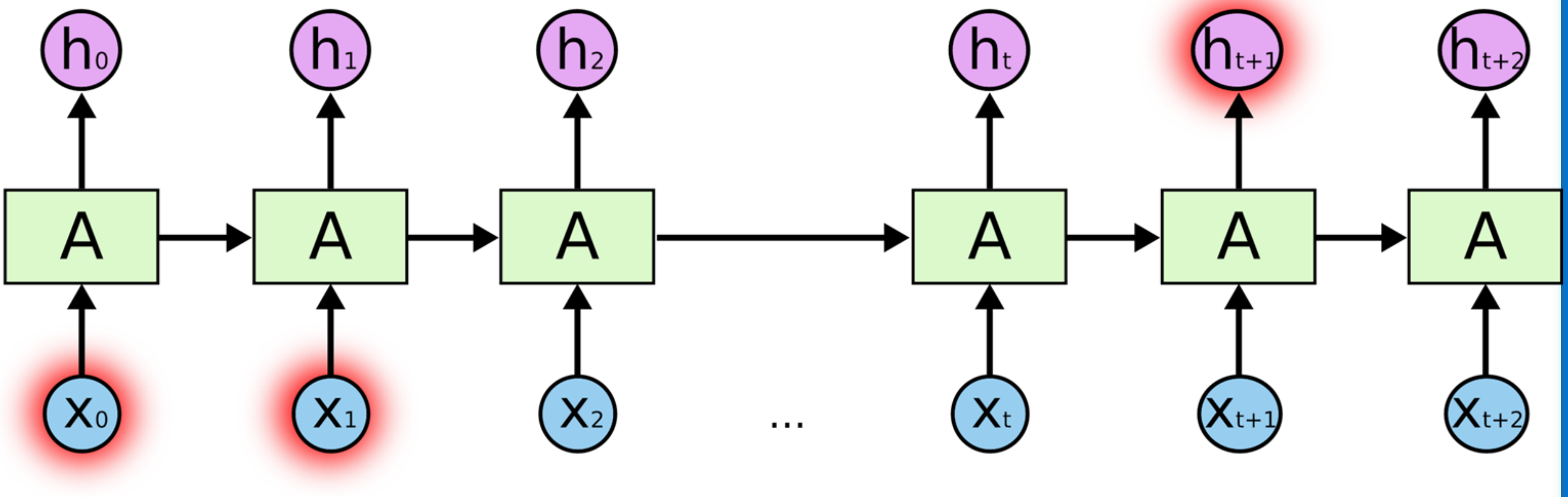


- Backpropagating the error in time involves as many recurrent derivation terms as timesteps on the net.

- Problems with gradients:
  - Vanishing
  - Exploding

Gated RNN

Simple RNN Problems: Long-Term Dependencies



http://colah.github.io/posts/2015-08-Understanding-LSTMs/

RNN Advances

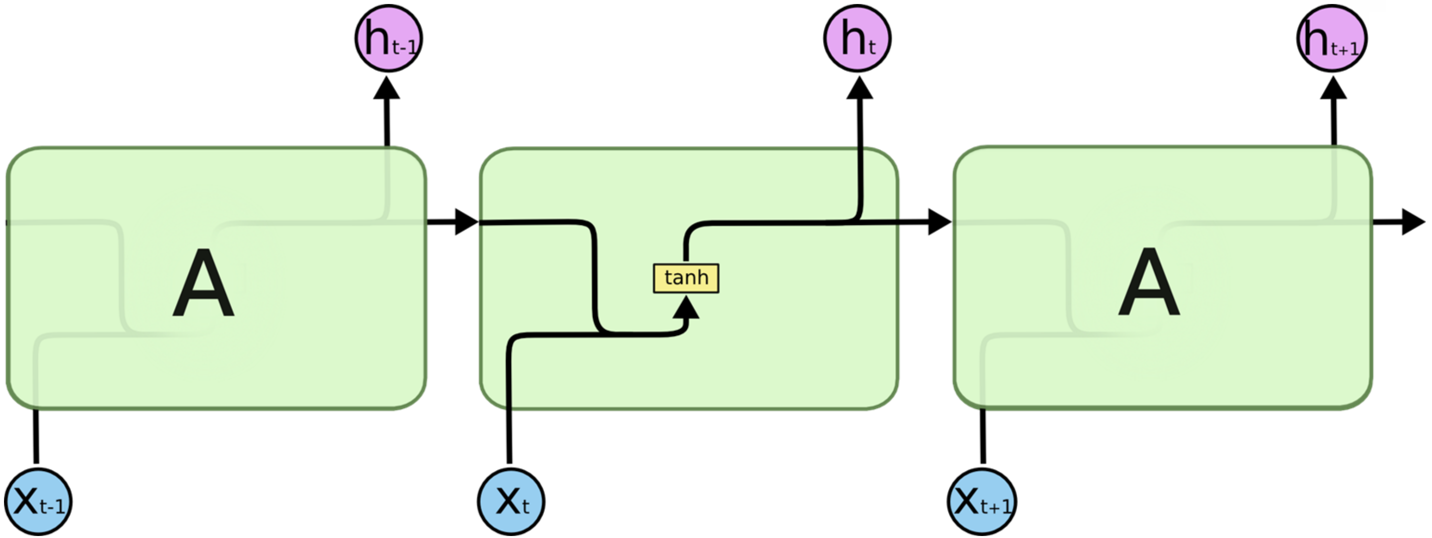
Long-Term Dependencies: Use more complex recurrent cells for control information flow.

1. GRU: Gated recurrent unit

1. LSTM: Long short-term memory

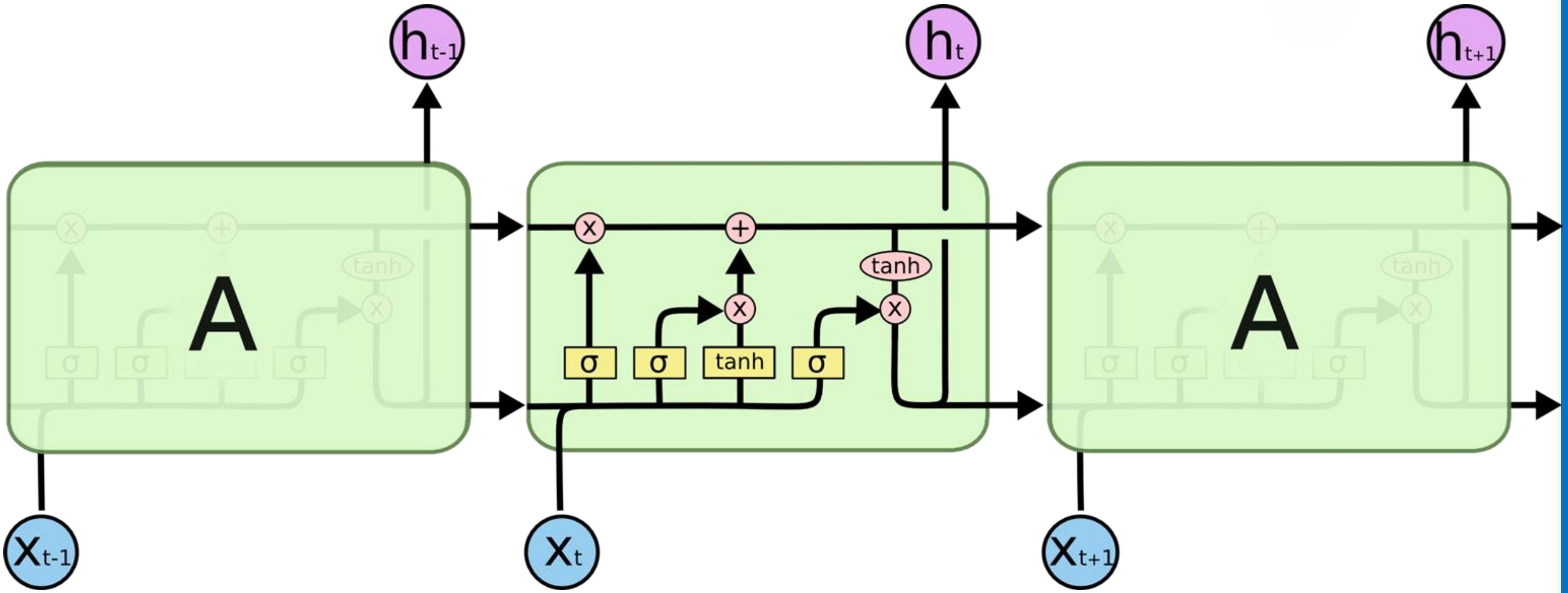
```
model1.add(layers.SimpleRNN(64)
)
model2.add(layers.GRU(64))
model3.add(layers.LSTM(64))
```

Simple RNN



http://colah.github.io/posts/2015-08-Understanding-LSTMs/

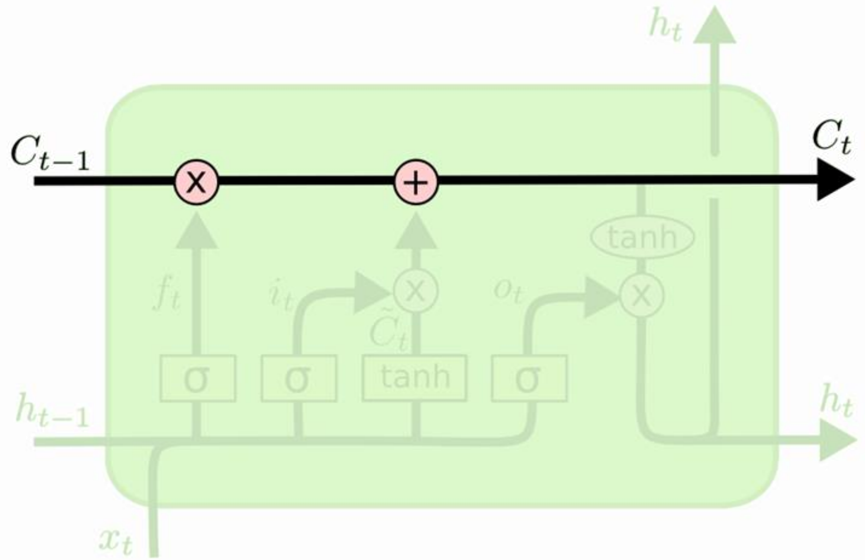
LSTM: Long Short-Term Memory



LSTMs are explicitly designed to avoid the long-term dependency problem. Remembering information for long periods of time is practically their default behavior

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

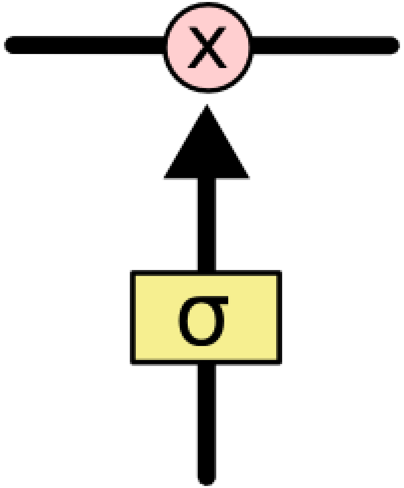
LSTM: Long Short-Term Memory



The LSTM can remove or add information to the cell state

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

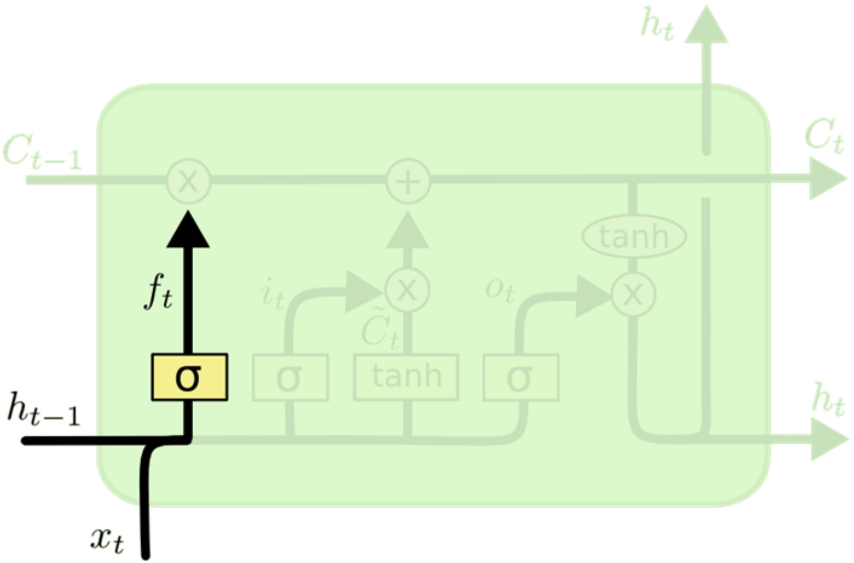
LSTM: Information Flow



- Output of a sigmoid [0,1]
- Information is added or removed with pointwise multiplication => **Gates**
- Steps:
  - 1. Forget
  - 2. Store
  - 3. Update
  - 4. Output

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

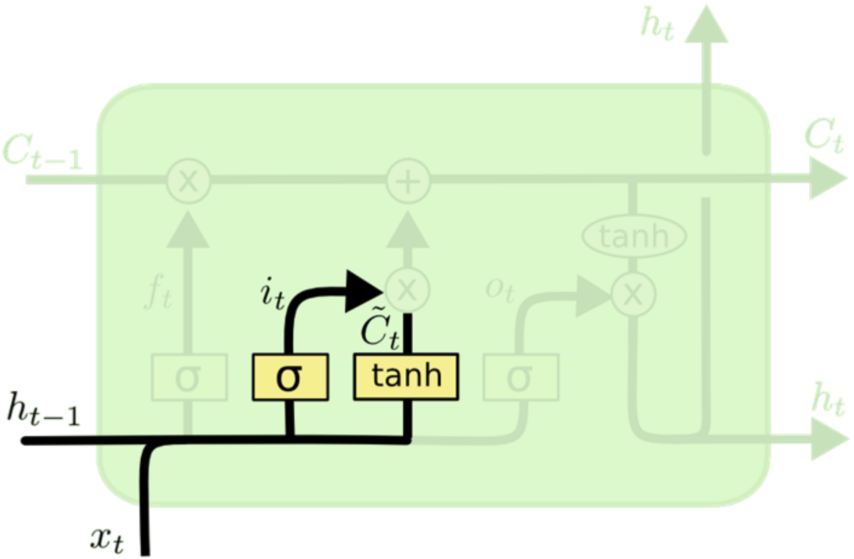
LSTM: 1. Forget Gate Layer



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

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTM: 2. Store



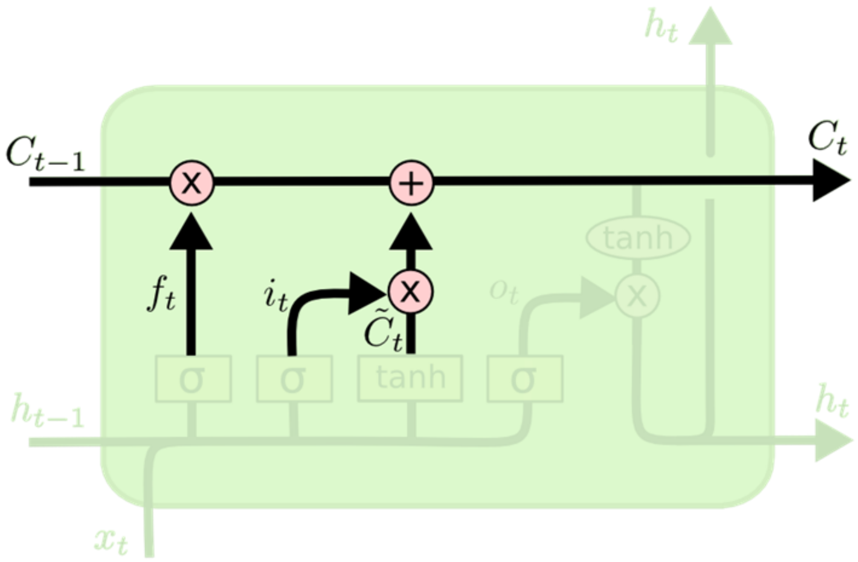
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

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

Store relevant information

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTM: 3. Update

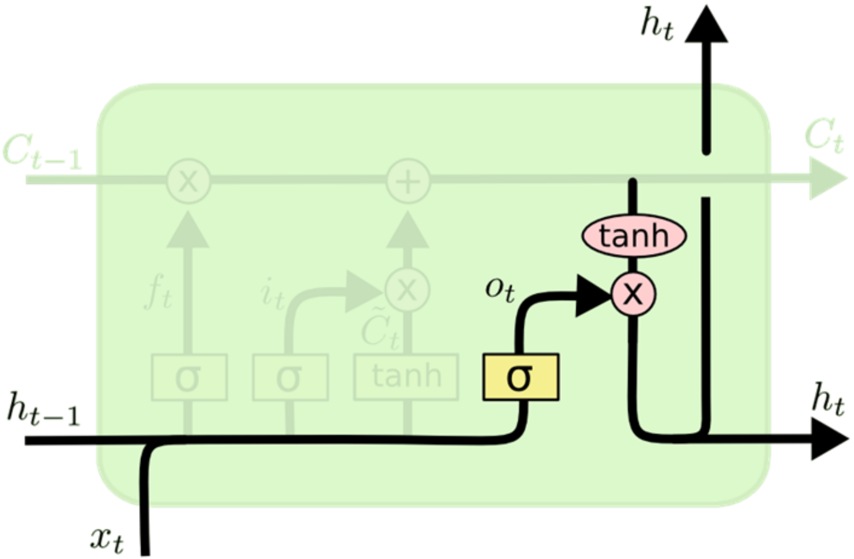


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Update cell state

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTM: 4. Output



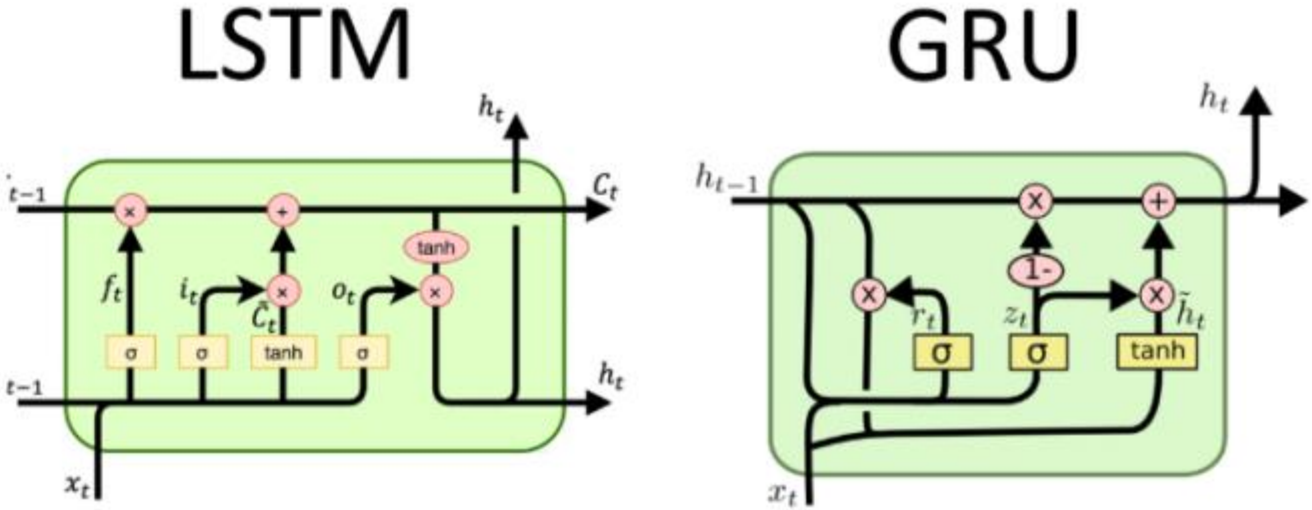
$$o_t = \sigma\left(W_o\ [h_{t-1}, x_t]\ +\ b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

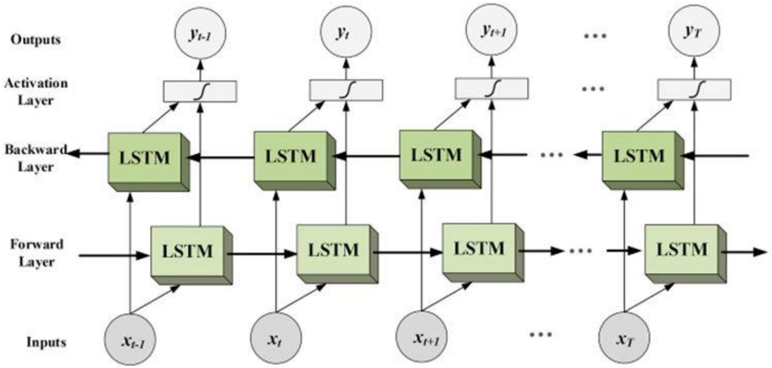
Control the relevant information

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

GRU: Gated Recurrent Unit

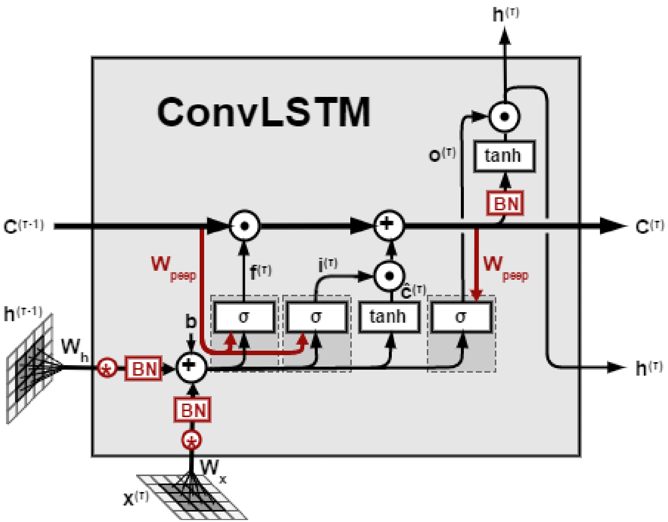


Bi-LSTM



- Video classification

Convolutions + LSTM



- Videos can be treated as a sequence of images as essentially, this is what they are. So, they are candidates to be treated sequentially.

- One possible approach is using **ConvLSTM** layers. It is a Recurrent layer, just like the LSTM, but internal matrix multiplications are exchanged with convolution operations. As a result, the data that flows through the ConvLSTM cells keeps the input dimension (3D in our case) instead of being just a 1D vector with features.

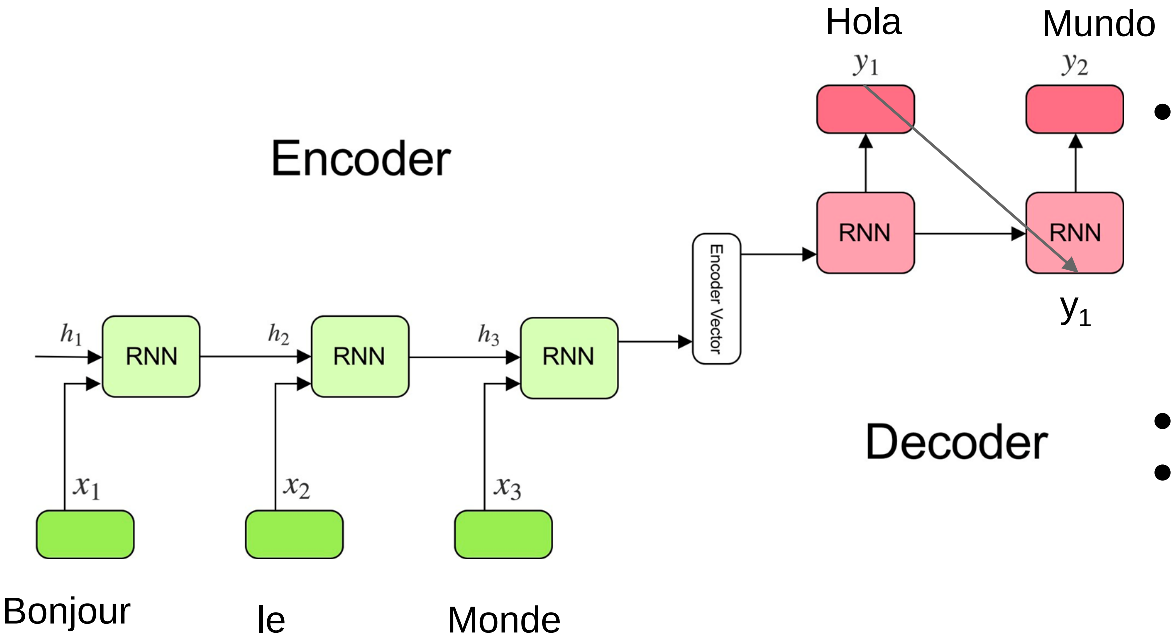
- It is very common to misunderstand ConvLSTM layers/networks with **Convolutional-LSTM** models, in which the image passes through the convolutional layers and its result is a set flattened to a 1D array with the obtained features. When repeating this process to all images in the time set, the result is a set of features over time, and this is the LSTM layer input.

Use Cases

jupyter

Sequence to Sequence Model & Attention Mechanism

Sequence to Sequence



- Two sub-models:.
  - **Encoder**: Encoding the entire sequence into a fixed length vector called a context vector.
  - **Decoder**: Reading from the context vector.
- Introduced by Google in **2014**.
- Maps variable-lengths sequences to a **fixed length** encoded vector.

Keneshloo. Neural Abstractive Text Summarization with Sequence-to-Sequence Models

Sequence to Sequence: Image Captioning

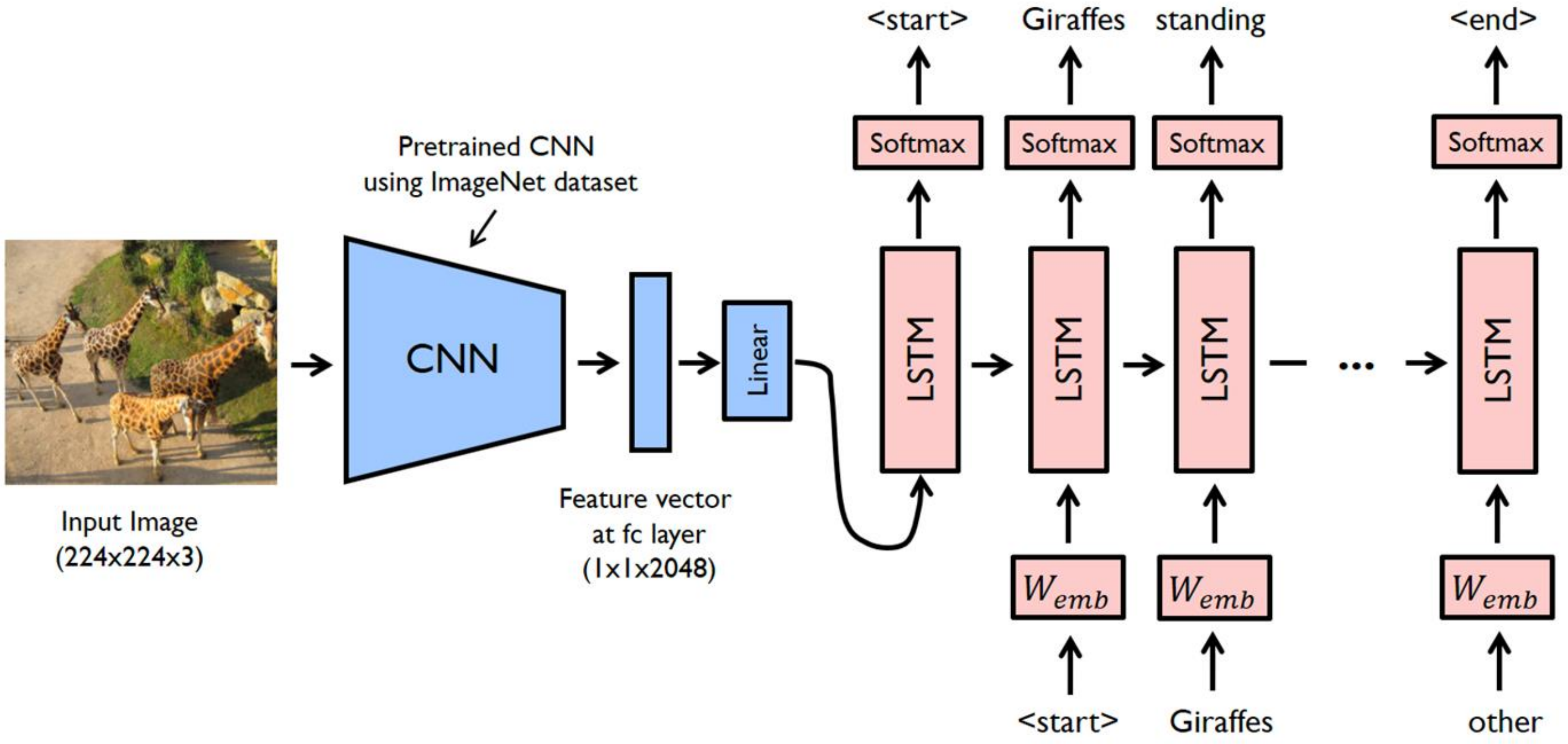
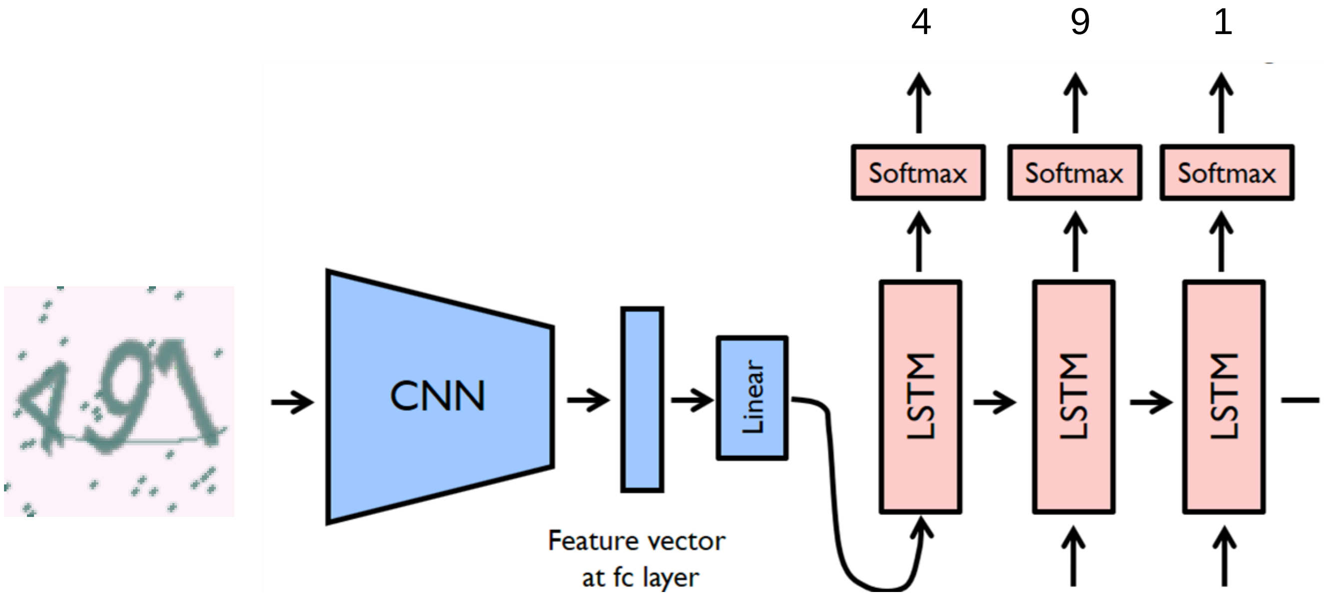
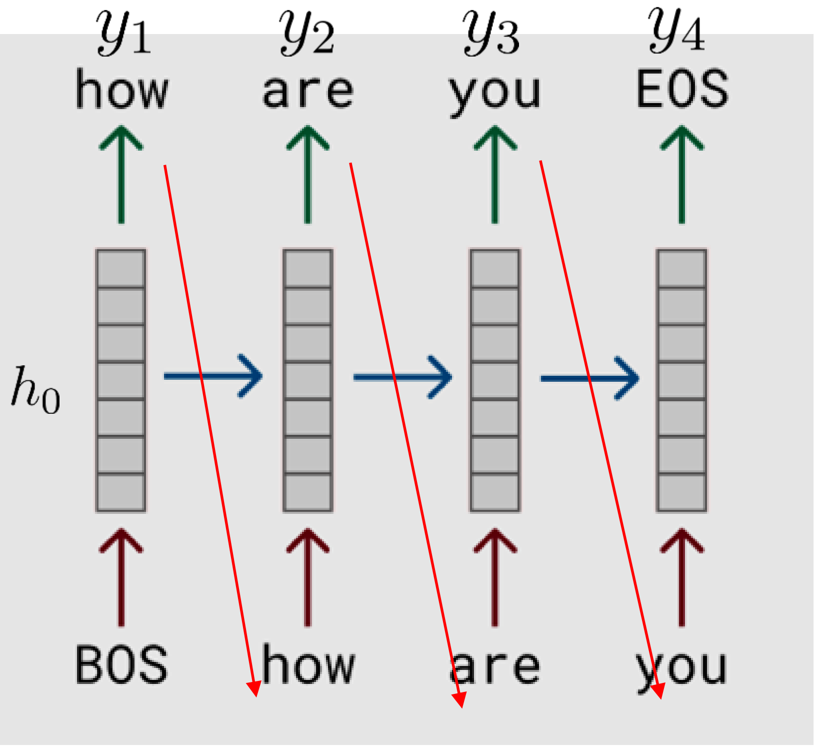


Image from www.analyticsvidhya.com

Sequence to Sequence: Captcha Hacking



Choose Output Sequence



Input: $h_0 = encoded(\text{“Qué tal estás”})$

First output: Sample from softmax

$$\hat{y}_1 = P(y_1|h_0) \rightarrow P(\text{“How”}|h_0)$$

Second output: Sample from softmax

$$\hat{y}_2 = P(y_2|h_0, y_1) \rightarrow P(\text{“are”}|\text{“}How\text{”}, h_0)$$

Third output: Sample from softmax

$$\hat{y}_3 = P(y_3|h_0, y_1, y2) \rightarrow P(\text{“you”}|\text{“}How\text{”}, \text{“}are\text{”}, h_0)$$

Fourth output: Sample from softmax

$$\hat{y}_4 = P(y_4|h_0, y_1, y_2, y_3) \rightarrow P(\text{“EOS”}|\text{“}How\text{”}, \text{“}are\text{”}, \text{“}you\text{”}, h_0)$$

Choose Output Sequence

$$y* = \text{argmin}\, P(y_1, y_2, \cdots, y_n | h_0) = P(y_1|h_0)P(y_2|y_1, h_0) \cdots P(y_n|h_0, y_1, y_2, \cdots, y_{n-1})$$

“El año pasado volví a ir a Roma con mis amigos”

“Last year I went back to Rome with my friends”

“Last year I went to Rome again with my friends”

Search Strategies

- Sampling from output softmax not always return the best output sentence.
- Greedy (choose the max probability from the softmax) doesn’t work well.
- Need other search algorithms: **Beam search**

Attention Mechanism

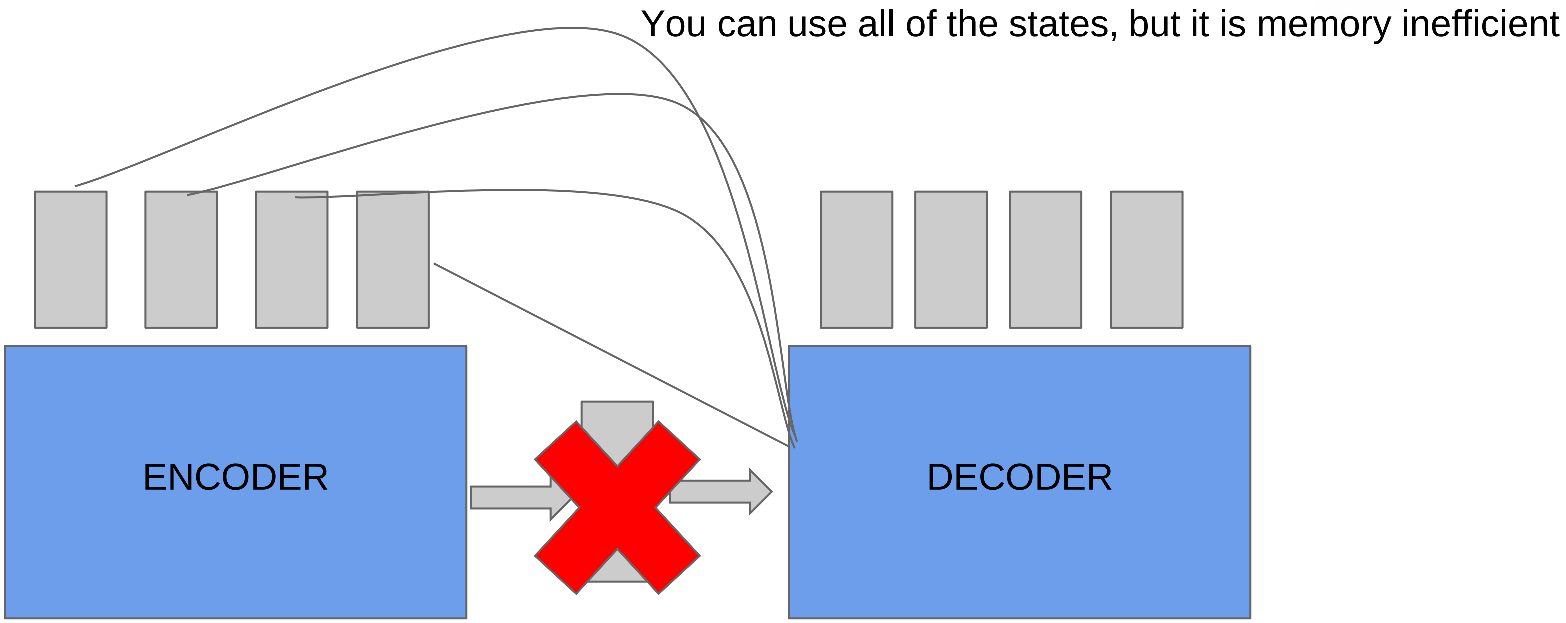
- The encoder processes the entire input sentence and encode it into a context vector. The decoder produces the words in a sentence one after another. Long-term dependency problems.

- Performance of the encoder-decoder network degrades rapidly as the length of the input sentence increases.

- **Attention Mechanism:** Selectively concentrating on a few relevant things, while ignoring others. (Also provides some **interpretability**)

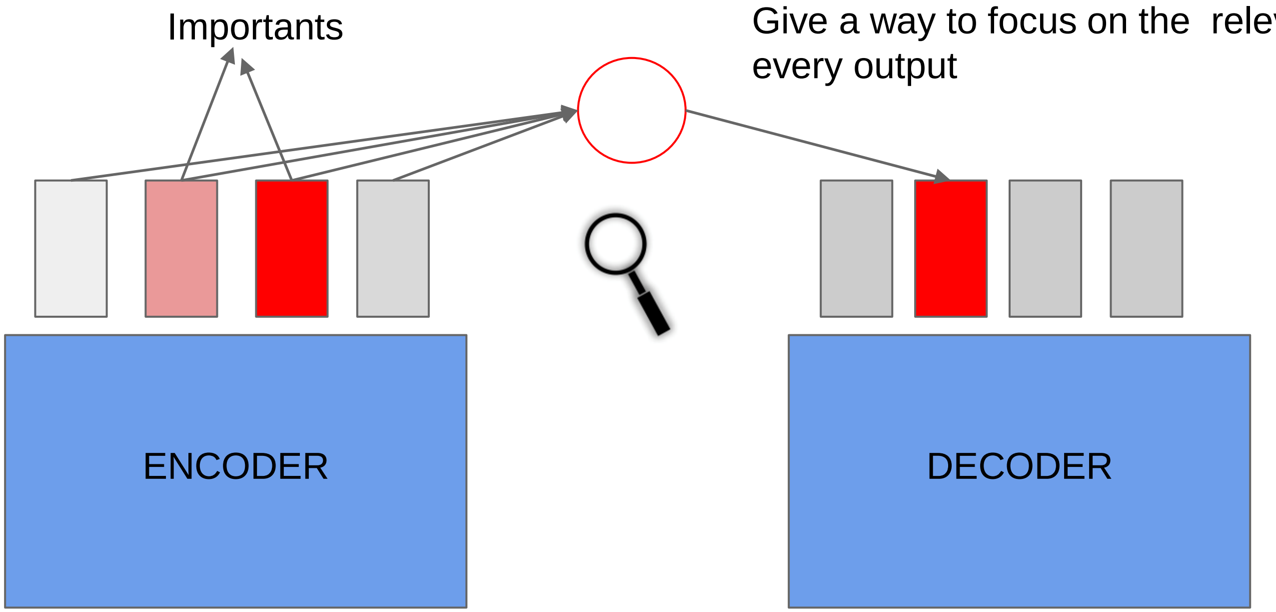
- *"Neural Machine Translation by Jointly Learning to Align and Translate" (2015)*

Attention Mechanism: Intuition



¿Cuál es tu nombre?

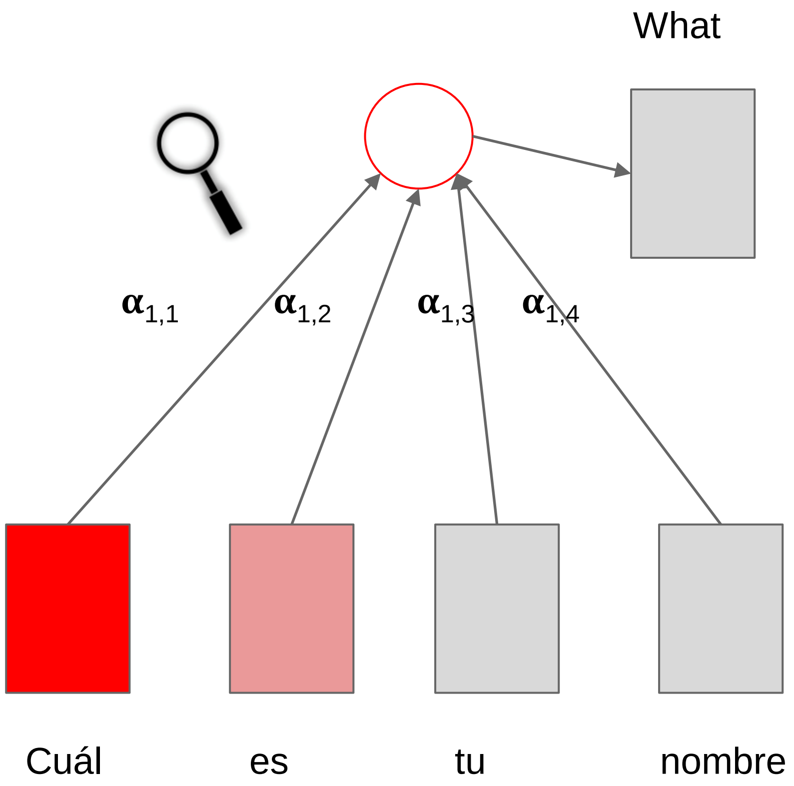
Attention Mechanism: Intuition



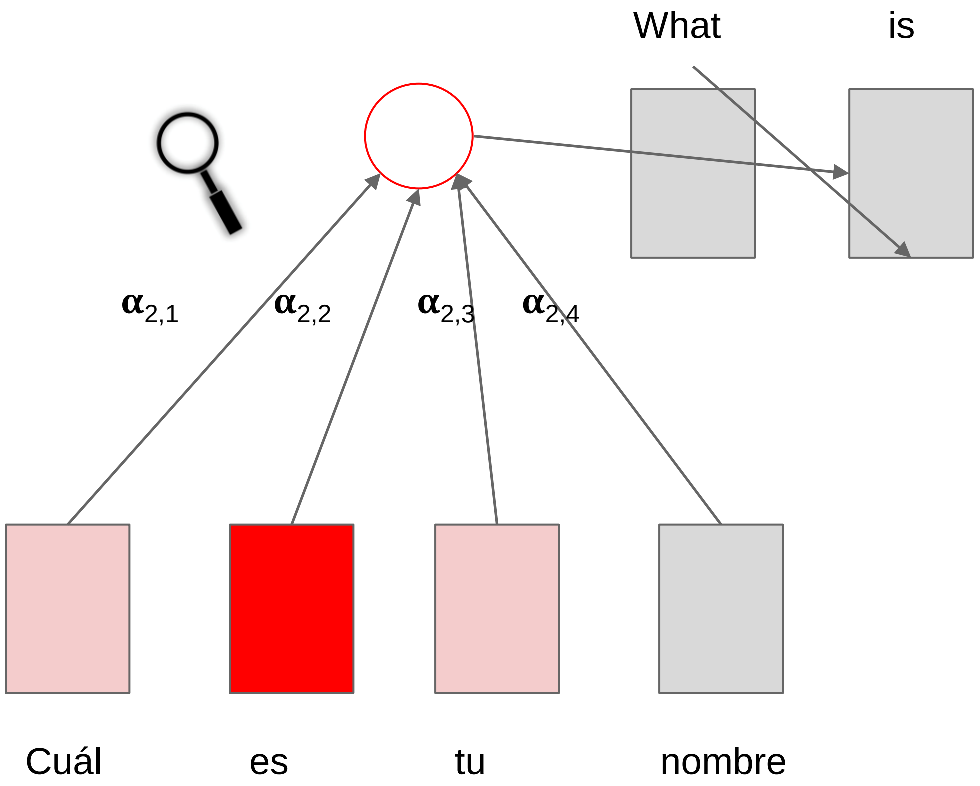
Give a way to focus on the relevant parts for every output

¿Cuál es tu nombre?

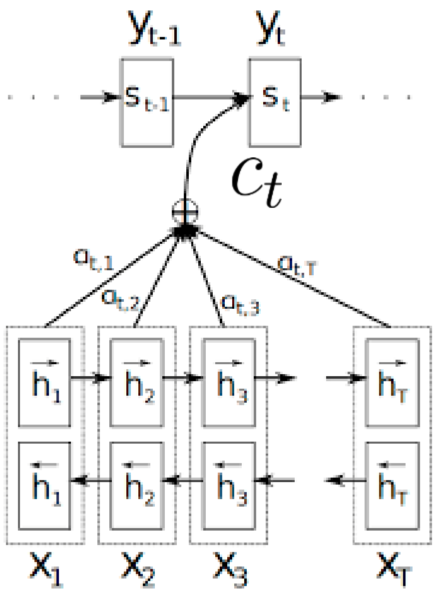
Attention Mechanism: Intuition



Attention Mechanism: Intuition



Attention Mechanism



You don't encode all the input in the last state, you use all the hidden states for every output.

$$h_t = (\overrightarrow{h}_t \overleftarrow{h}_t)$$

Focus in the relevant inputs with the context vector (input of y nn). $a_{tt'}$ weights every state h_t

$$c_t = \sum_{t'} a_{tt'} h_{t'}$$

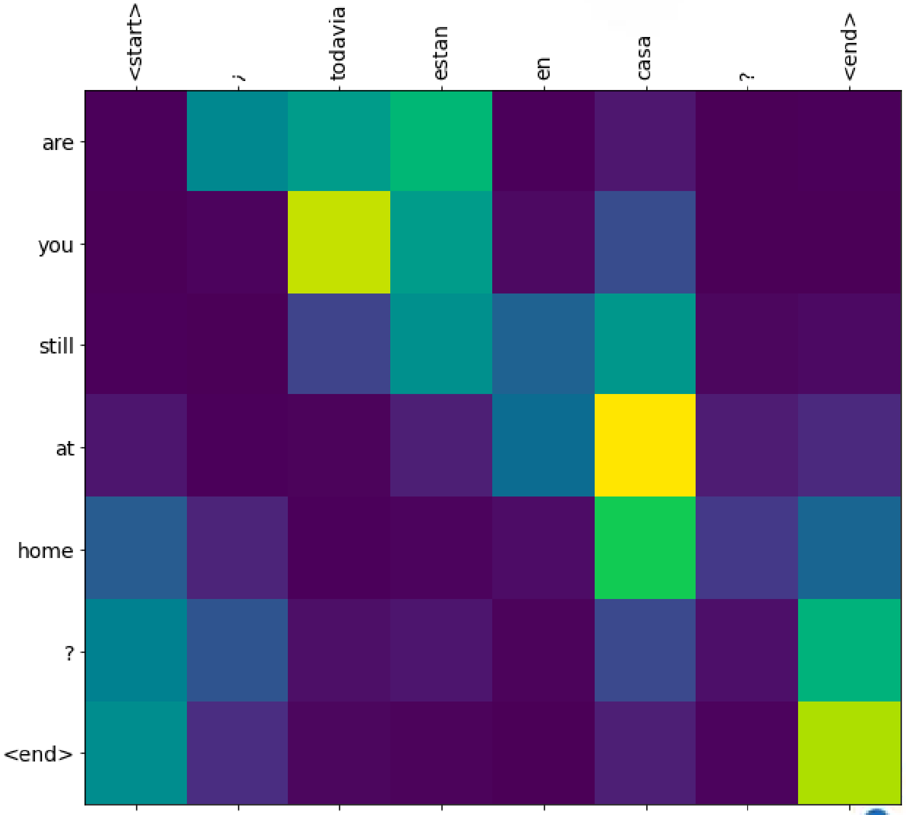
$a_{tt'}$ is the amount of attention y_t should pay to $h_{t'}$.

$$a_{tt'} = \frac{\exp(e_{tt'})}{\sum_{k=1} \exp(e_{tk})}$$

$$e_{tt'} = f(s_{t-1}, h_{t'})$$

Simple NN

Attention Mechanism



ntic master

revolucionamos la comunicación

UNIVERSIDAD COMPLUTENSE DE MADRID