



CORPORATE EDUCATION

CdP-Insurance Data Management

Introduction to NLP

Milano, 03.07.2020

Mauricio Soto -mauricioabel.soto@polimi.it



POLITECNICO DI MILANO
GRADUATE SCHOOL
OF BUSINESS

FT
Executive Education
Ranking 2018

FT
European Business Schools
Ranking 2017

EFMD
EQUIS
ACCREDITED

ASSOCIATION
AMBA
ACCREDITED

EFMD
EOCCS
CERTIFIED

Overview

- ▶ Main Objective: Introduction to problems and techniques used in NLP.
 - ▶ Data preparation for text
 - ▶ Sentiment Analysis
 - ▶ Topic Modeling
 - ▶ Word embedding: Word2vec and Glove
 - ▶ *RNN - Long Short Term Memory networks (LSTM)

Some important questions

- ▶ What is the meaning of a word?
- ▶ How we can represent it?
- ▶ Which are the limits/problems of a representation?

Text Cleaning

- ▶ Convert to lower case
- ▶ Remove punctuation
- ▶ Remove numerical values
- ▶ Typos
- ▶ Remove special characters ([@?])
- ▶ Remove stop words (the, it, etc)
- ▶ Remove special description words ([chorus], [fade], [applause])
- ▶ Tokenize text (O'Neill → [o] [neill], [o'neill]?; aren't → [arent], [are][nt]?)
- ▶ Create bi-grams or tri-grams ([United Kingdom] vs [United][Kingdom])
- ▶ Normalization:
 - ▶ Stemming (car, cars, car's, cars' → car;)
 - ▶ Lemmatization (am, are, is → be)

Text Representation

1. Corpus: a collection of text
2. Document-Term Matrix: word counts in matrix format
3. TF-IDF: Term Frequency - Inverse Document Frequency

$$\text{TF-IDF} = f_{t,d} \times \text{idf}(t, D)$$

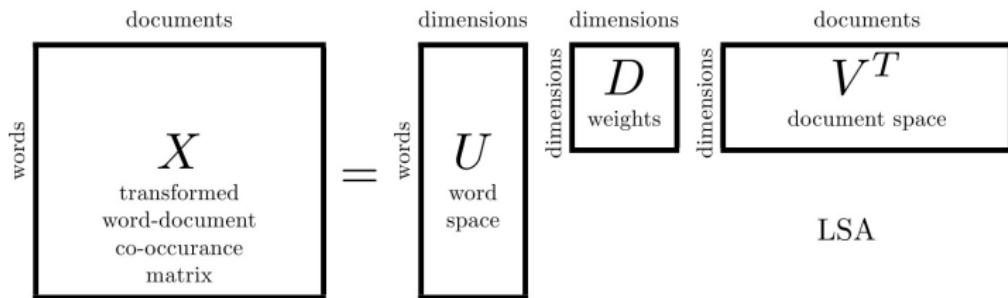
where

- ▶ $tf(t)$: the number of times that term t occurs in document d .
- ▶ $\text{idf}(t, D) = \log \frac{|D|}{|\{d \in D : t \in d\}| + 1}$

Sentiment Analysis

- ▶ TextBlob Module: Linguistic labeled the sentiment of words.
<https://github.com/sloria/TextBlob/blob/eb08c120d364e908646731d60b4e4c6c1712ff63/textblob/en/en-sentiment.xml>
- ▶ Sentiment Labels: Each word is labeled in terms of
 - ▶ Polarity: negative(-1) or positive(+1)
 - ▶ Subjectivity: subjective(0) or fact(+1)
- ▶ Sentiment of words can vary based on where it is in a sentence.
 - ▶ Negation multiplies the polarity by -0.5

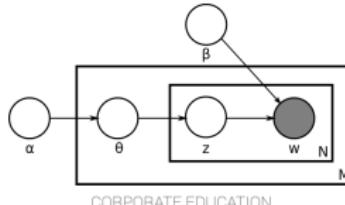
Topic Modeling - Latent Semantic Analysis (LSA)



- ▶ **Singular Value Decomposition (SVD) of the Document-Term Matrix**

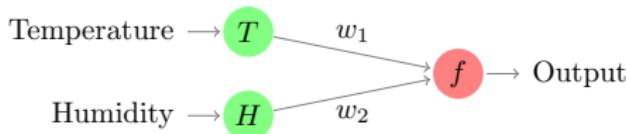
Topic Modeling - Latent Dirichlet Allocation (LDA)

- ▶ **Documents are probabilistic distribution over topics:** let say that a document is $p_i\%$ of topic i .
- ▶ **Topics are probabilistic distribution over words:** given a topic chosen according to the distribution of the document, we generate a word according to the topic distribution
- ▶ Random initialisation: assign each word to a random topic
- ▶ Update each word by considering
 - ▶ proportion of words in the document of topic
 - ▶ proportion of topics in all documents for the word
- ▶ We sample until a “reasonable” result



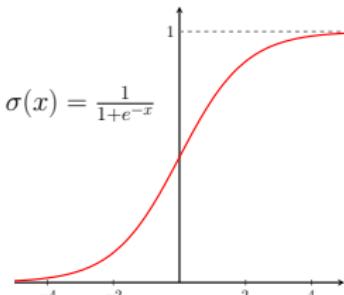
Neural Networks: Multi-Layer Perceptron

Temp. [C]	20	31	15	18	21
Humidity [%]	40	36	23	45	30
Prob. Rain	0.70	0.52	0.55	0.73	0.60



$$f(T, H, w_1, w_2) = \underbrace{\sigma}_{\text{activation function}}(w_1 \cdot T + w_2 \cdot H)$$

$$\min_{w_1, w_2} \sum_{i=1}^5 [P_i(\text{rain}) - f(T_i, H_i, w_1, w_2)]^2$$

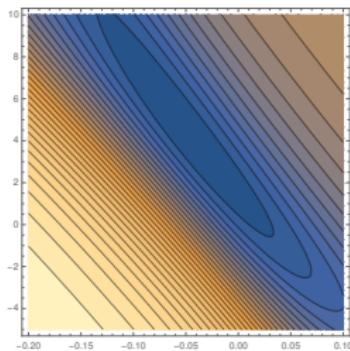


We can use the Likelihood as loss function.

Neural Networks: Multi-Layer Perceptron

$$\max_{w_1, w_2} \sum_{i=1}^5 [P_i(\text{rain}) - f(T_i, H_i, w_1, w_2)]^2$$

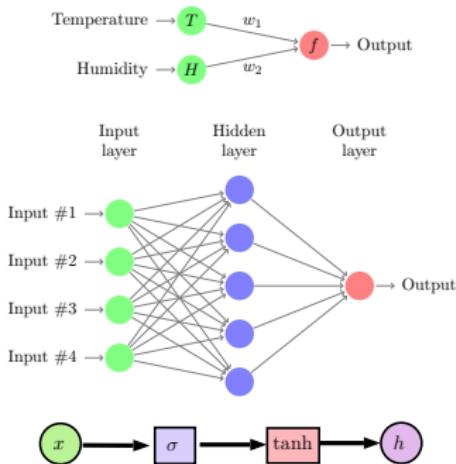
$$= \max \left[0.7 - 1/(1 + e^{-(w_1 \cdot 20 + w_2 \cdot 0.4)}) \right]^2 + \left[0.52 - 1/(1 + e^{-(31 \cdot w_1 + w_2 \cdot 0.36)}) \right]^2$$



$$(w_1^*, w_2^*) = (-0.044, 4.147)$$

Temp. [C]	20	31	15	18
Humidity [%]	40	36	23	45
Prob. Rain	0.70	0.52	0.55	0.73
Predicted	0.70	0.56	0.58	0.75
Error	0.0	-0.04	-0.03	-0.02

Neural Networks: Multi-Layer Perceptron



Main Parameters

- ▶ `hidden_layer_sizes`: (n_1, n_2, \dots, n_L)
- ▶ `activation`: identity, logistic, tanh, relu
- ▶ Resolution algorithm parameters: `solver`, `tol`, `batch_size`, `learning_rate`, `max_iter`.

Word Embedding

Distributed Representations of Words (a.k.a. word embeddings) are geometric representation of words/entities learned from the data/corpus in such a way that semantically related words are often close to each other.

In practice we attempt to embed entities onto a low-dimensional metric space in which similar words are placed close

Distributional hypothesis

“You shall know a word by the company it keeps”
(Firth, 1957)

Similar words tend to occur in similar contexts, therefore a word can be represented based on the co-occurrence across the data in the same context (word window).

Word2Vec Input

Source Text

The quick brown fox jumps over the lazy dog. →

The quick brown fox jumps over the lazy dog. →

The quick brown fox jumps over the lazy dog. →

The quick brown fox jumps over the lazy dog. →

Training Samples

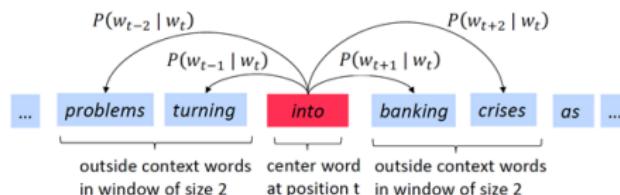
(the, quick)
(the, brown)

(quick, the)
(quick, brown)
(quick, fox)

(brown, the)
(brown, quick)
(brown, fox)
(brown, jumps)

(fox, quick)
(fox, brown)
(fox, jumps)
(fox, over)

1



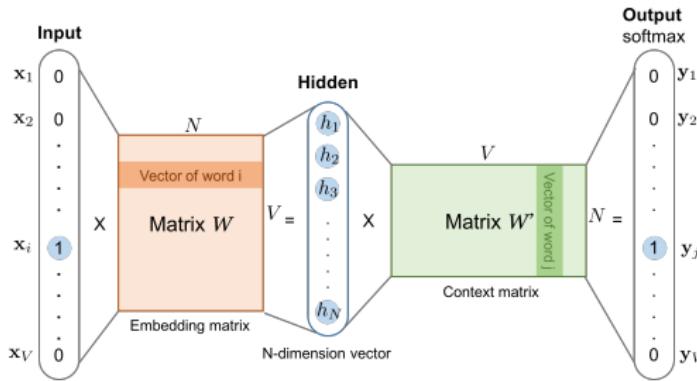
2

¹ <http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/>

² <http://web.stanford.edu/class/cs224n/>

Word2Vec Skip-Gram

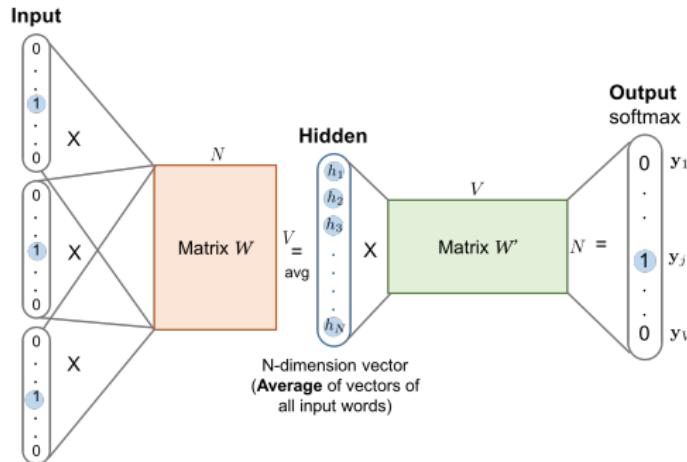
A single-layer architecture based on the inner product between two word vectors.



$$\underbrace{e_i^\top \times W_{N \times d}}_h \times W_{d \times N} \xrightarrow{\text{softmax}} \mathbb{P}(\text{word}_j | \text{word}_i)$$

We maximize $\frac{1}{T} \sum_{t=1}^T \sum_{-c \leq k \leq c, k \neq 0} \log(p(w_{i+k} | w_i))$ where $p(o|c) = \exp(v_o^\top v_c) / \sum_{w=1}^V \exp(u_w^\top v_c)$

Word2Vec CBOW



$$h = \frac{1}{|\text{window}|} \sum_{i=1}^{|\text{window}|} e_i^\top \times W_{N \times d}$$

<https://lilianweng.github.io/lil-log/2017/10/15/learning-word-embedding.html>

Cosine Similarity

Similarity distance measure as the cosine similarity:

$$\cos - sim(a, b) = \frac{v(a)^\top v(b)}{\|v(a)\| \|v(b)\|}$$

Glove

- ▶ We use the co-occurrence matrix for the entire corpus.
- ▶ As in SVD we try to encode a matrix into some “principal components”
- ▶ As in word2vec the probability of the context word given the central word is proportional to the dot product of vector representations.

counts	I	like	enjoy	deep	learning	NLP	flying	.
I	0	2	1	0	0	0	0	0
like	2	0	0	1	0	1	0	0
enjoy	1	0	0	0	0	0	1	0
deep	0	1	0	0	1	0	0	0
learning	0	0	0	1	0	0	0	1
NLP	0	1	0	0	0	0	0	1
flying	0	0	1	0	0	0	0	1
.	0	0	0	0	1	1	1	0

Example corpus:

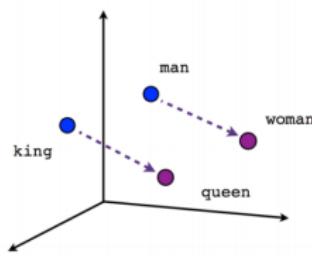
- I like deep learning.
- I like NLP.
- I enjoy flying.

$$J(\theta) = \frac{1}{2} \sum_{i,j=1}^W f(P_{ij})(u_i^T v_j - \log P_{ij})^2$$

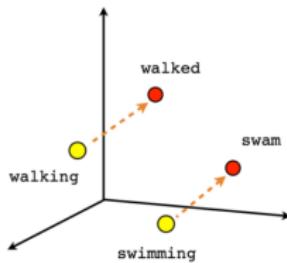
Analogical Reasoning

The city of Rome is in relation with the country Italy in the same way as the city of Paris is in relation with the country France.

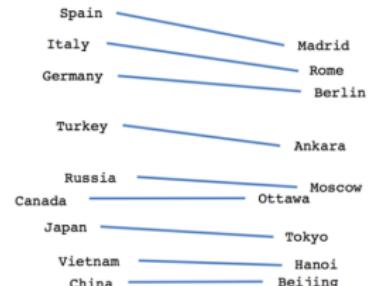
The propositional analogy task: find an $?x$ such that $\text{Rome} : \text{Italy} = ?x : \text{France}$



Male-Female



Verb tense



Country-Capital

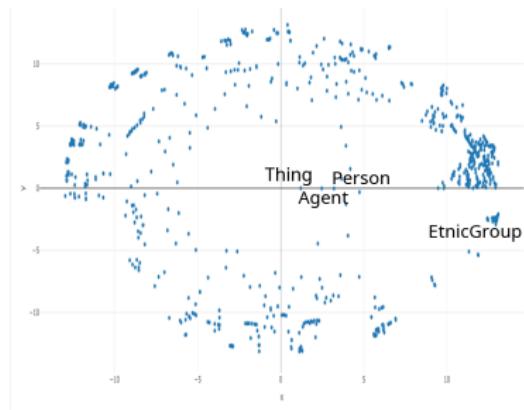
$$v(\text{Italy}) - v(\text{Rome}) \sim v(\text{France}) - v(\text{Paris})$$

Implementations

- ▶ Neural Network: Word2Vec [Mikolov+, 2013], ELMO [Peters+, 2018]
<https://code.google.com/archive/p/word2vec>
- ▶ GloVe [Pennington+, 2014]. Matrix Factorization/Neural Network. <https://nlp.stanford.edu/projects/glove/>

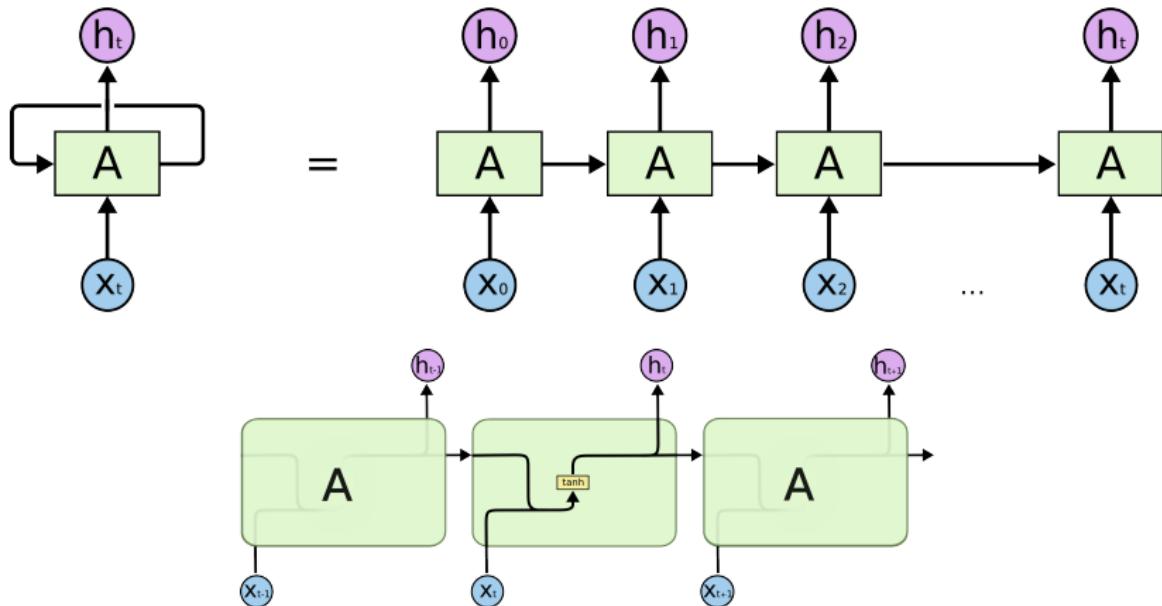
Hyperbolic embeddings

Hyperbolic embedding for hierarchical relations (as ontology types)
3



RNN: Recurrent Neural Networks

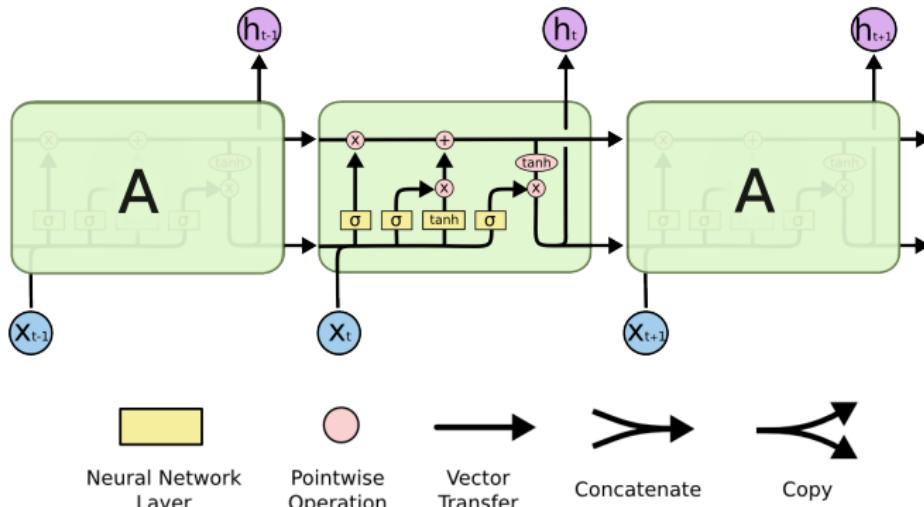
Given a sequence (of words): $x = x_1 x_2 \cdots x_t$



<https://colah.github.io/posts/2015-08-Understanding-LSTMs/>

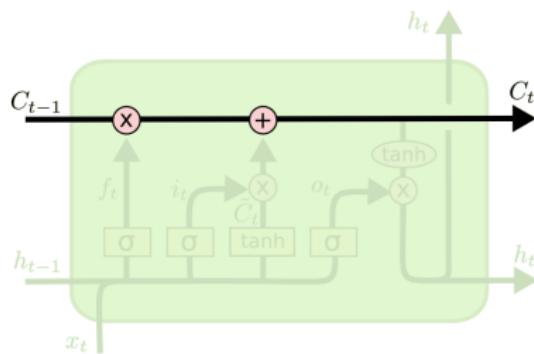
LSTM: Long Short Term Memory networks

1. We keep a cell state across the sequence C_t
 2. After each step t we:
 - ▶ forget something: f_t
 - ▶ include something : i_t
 - ▶ update the cell state: C_t
 - ▶ output something to the next step: h_t



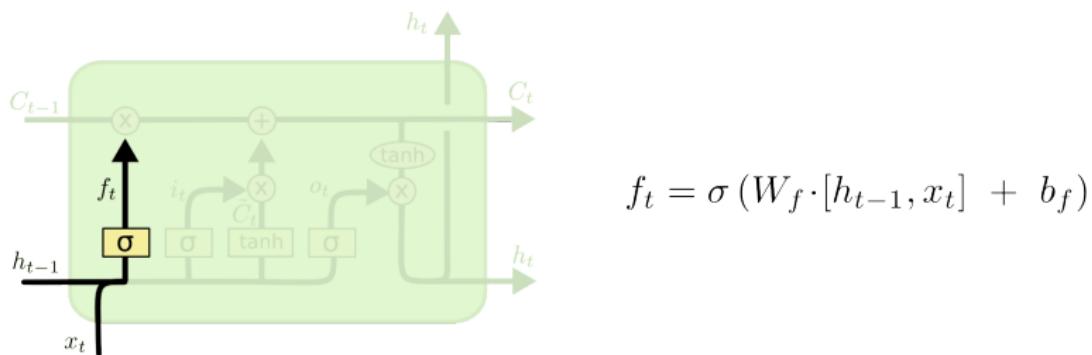
LSTM: Keep Global state

1. We keep a cell state across the sequence C_t
2. After each step t we:
 - ▶ forget something: f_t
 - ▶ include something : i_t
 - ▶ update the cell state: C_t
 - ▶ output something to the next step: h_t



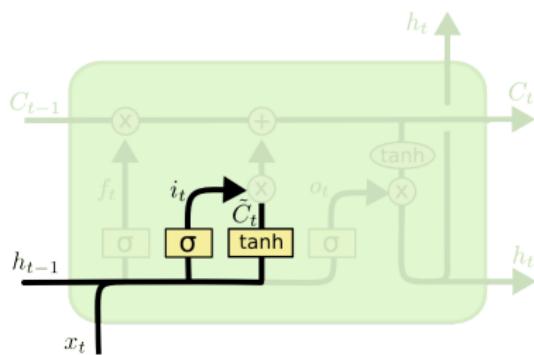
LSTM: forget gate state

1. We keep a cell state across the sequence C_t
2. After each step t we:
 - ▶ **forget something:** f_t
 - ▶ include something : i_t
 - ▶ update the cell state: C_t
 - ▶ output something to the next step: h_t



LSTM: input gate state

1. We keep a cell state across the sequence C_t
2. After each step t we:
 - ▶ forget something: f_t
 - ▶ **include something** : i_t
 - ▶ update the cell state: C_t
 - ▶ output something to the next step: h_t

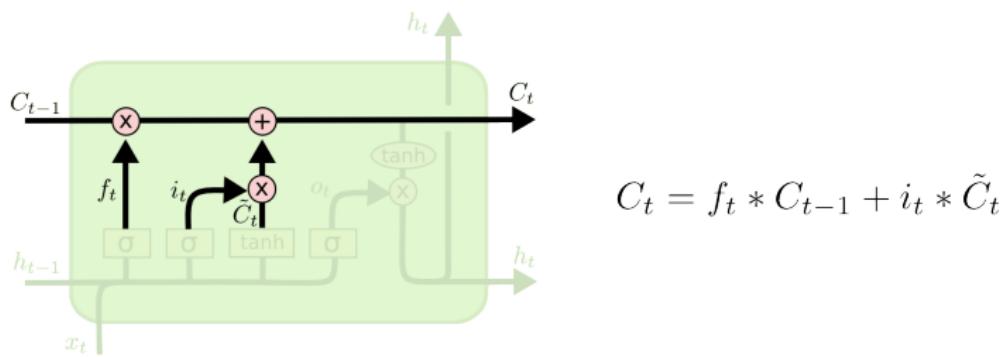


$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

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

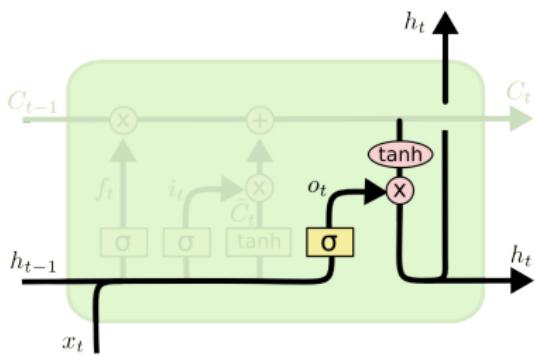
LSTM: update cell state

1. We keep a cell state across the sequence C_t
2. After each step t we:
 - ▶ forget something: f_t
 - ▶ include something : i_t
 - ▶ **update the cell state:** C_t
 - ▶ output something to the next step: h_t



LSTM: cell output

1. We keep a cell state across the sequence C_t
2. After each step t we:
 - ▶ forget something: f_t
 - ▶ include something : i_t
 - ▶ update the cell state: C_t
 - ▶ **output something to the next step:** h_t



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

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

Generating text

1. From the text, we create a training set form by couples $([x_1, \dots, x_t], y_t)$ where:
 - ▶ $[x_1, \dots, x_t]$ is a sequence of t elements (letters, words)
 - ▶ y_t is the element to be predicted
2. From a seed sequence we sequentially generate the text
consider as input the last sequence.