

# Recurrent Neural Networks (RNN), Long Short Time Memory Networks (LSTM)

ENSE3 / Grenoble-INP  
Parcours Numérique 1A

Florent Chatelain, Olivier.JJ.Michel

GIPSA-Lab, Grenoble-INP, Univ Grenoble Alpes.

2019-2020

## Additional course material, and document sources for this lecture

Youtube MIT online courses :

- ▶ on NN : <https://www.youtube.com/watch?v=njKP3FqW3Sk>, by Alexander Amini
- ▶ on RNNs and LSTM : [https://www.youtube.com/watch?v=\\_h66BW-xNgk](https://www.youtube.com/watch?v=_h66BW-xNgk) by Ava Soleimani
- ▶ From Perceptrons to LSTM :  
<https://www.youtube.com/watch?v=yKGm4yLuTkU> by Caio Benatti Moretti

Most material and figures in the slides below are from (excellent) Colah's blog :  
<https://colah.github.io/posts/2015-08-Understanding-LSTMs/>

📖 The reference book on Deep learning :

I.J.Goodfellow, Y.Bengio, A.Courville : "Deep Learning", MIT Press 2016  
(<http://www.deeplearningbook.org>)

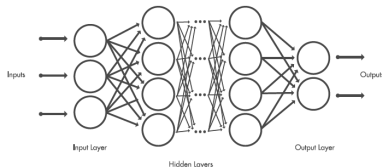
## Motivation

Remind the main features of a feed forward Neural Network

- ▶ Inputs are assumed iid
- ▶ Learning is performed via the Back propagation of gradient, to minimize a Loss function

But

- ▶ The meaning of a word in a sentence depends on both
  1. its own meaning, signification
  2. previous words, from a recent past in the sentence
  3. context, possibly defined way back in the past
- ▶ The value of a time series
  1. rarely fully uncorrelated from its predecessor (except for white noise...)
  2. may depend on an underlying process with many characteristic time of dependencies



## RNN :

Recurrent Neural network precisely address this problem :

🔗 How can a system gets a memory about previous encountered events ?

Solutions proposed in the 1990's (Jeffrey Elman (90)), as being "Simple Recurrent Neural Network" :

- ▶ Introduce some **Neuron Internal State (NIS)**  $h_t$ , depending on the current input, and past NIS :

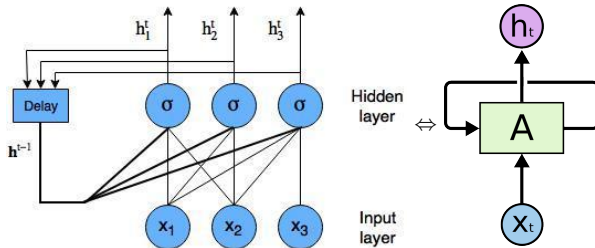
$$\begin{cases} h_t = \sigma(W_{hx}x_t + W_{hh}h_{t-1} + b_h) \\ y_t = f_o(W_{yh}h_t + b_y) \end{cases}$$

- ▶ This introduces a loop (like in many system from control theory) in the learning neuron, allowing the information to persist

**Notations** :  $\sigma$ =sigmoid function ;  $h_t$ =NIS ;  $x_t$ =input vector at  $t$ ,  $x_t \in \mathbb{R}^d$  ;  $y_t$ =output vector,  $y_t \in \mathbb{R}^{d'}$  ;  $W_{kl}$  is a weight matrix describing the connecting edges between ' $l$ ' ; ' $k$ ',  $b_{(\cdot)}$  stand for the biases, and  $f_o$  is the output function relating  $y_t$  to  $h_t$ .

- └ RNN definition
  - └ Internal state

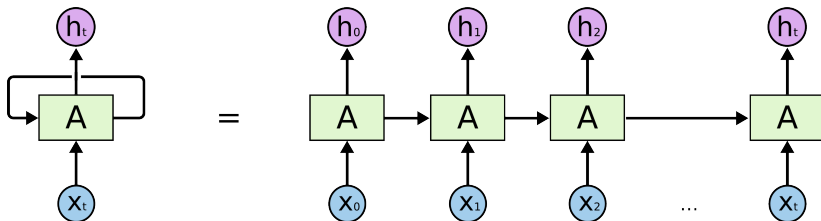
## RNN cell :



## Nota Bene :

- ▶ the output  $y_t$  is not represented here.  $y_t = f_o(W_{yh}h_t + b_y)$  may take different forms, depending on the learning task (linear operator for regression/regression or e.g. Softmax for classification)
- ▶ On this graph ; at each time  $t$ ,  $x_t \in \mathbb{R}^3$ ,  $h_t \in \mathbb{R}^3$
- ▶ Each neuron (or cell) is a MLP with a single hidden layer
- ▶ The loop is applied on the hidden layer

An alternate representation of Elman's equations, accounting for time evolution is :



The RNN can be thought of as a Deep Neural Network, with **all layers having identical weight matrices and bias**, or as multiple copies of the same network, each passing a message ( $h_{n-1}$ ) to its successor.

- ▶ The unrolled structure highlights the relation of RNNs with **lists** or **sequences**
- ▶ RNNs were recently extremely successful for speech recognition, translation, time series forecasting, .... **BUT with a slightly different functional bloc 'A'...**

## Vanishing gradient problem

Consider the unrolled structure for e.g. 3 time iterations. Learning the parameters of 'A' using GBP algorithm leads to equation that look (roughly) like this

$$\frac{\partial L}{\partial U} = \frac{\partial L_3}{\partial out_3} \frac{\partial out_3}{\partial h_3} \frac{\partial h_3}{\partial h_2} \frac{\partial h_2}{\partial h_1} \frac{\partial h_1}{\partial h_0} \frac{\partial h_0}{\partial U}$$

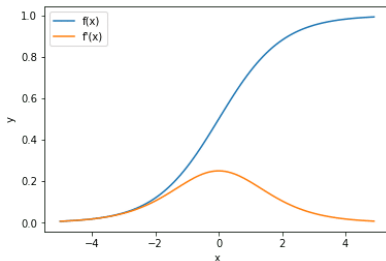
-  $U$  is any coefficient from the weight matrices or bias coeff.

where -  $out_{(.)}$  is e.g. the desired output at third iteration

-  $L$  stand for the loss function.

📺 In this product of gradients, each involves to evaluate the gradient of a sigmoid function : if the input values are such that a term in the product is very weak (which occurs if the output is close to either 0 or 1), the product becomes very small.

(Sigmoid gradient as a function of input value →).



## Vanishing gradient problem, cont'd

- ▶ The **gradient** will **become basically 0** when dealing with (too) **many time steps**, as all terms are between 0 and 1 (the same occurs by replacing the sigmoid by the tanh function).
- ▶ The **weights will not adjust** to take into account values that occurred too early in the sequence (remind the structure of the GBP algorithm)
- ▶ The **network cannot learn** relationships separated by significant period of time
- ▶ One may think about using RELU function instead. This does not solve the problem, as the gradient may as well explode....

A solution was found by Hochreiter and Schmidhuber in 1997, now very widely used :

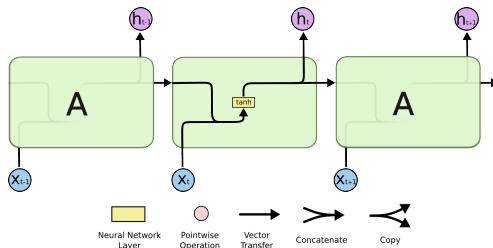
## Long Short Time Memory RNN : LSTM networks



# LSTM network... or simply 'LSTM'

## LSTM

- Keeps the form of a **repeating chain of similar modules**, as in simple RNN
- Keeps the principle of creating a **cell (neuron) internal state**, that is **transferred to the next module**
- The unrolled RNN structure will be represented as follows, where 'A' is simply containing concatenation, sigmoid (or tanh) and copy functions :

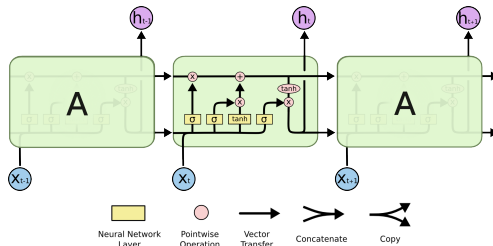


- Note that a repeating cell in a standard RNN is basically a MLP with a single hidden layer

## LSTM cont'd

### LSTM

- Instead of a single hidden layer MLP, a LSTM will count four (4) layer interacting in a special way. This increase in complexity has the unique purpose of avoiding vanishing gradient problems!

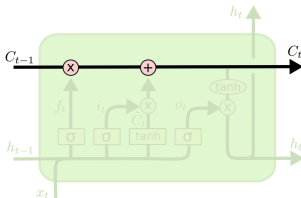


Note that each line in the diagram above carries an entire vector.

- Each of the (yellow) boxes represents a learned neural network (as a multiple output perceptron)

## LSTM as an information conveyor belt

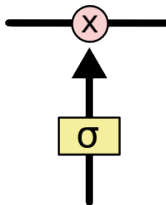
The **key information transfer** between modules (remind that in the unrolled representation, a module is associated to a time step) is represented by the upper horizontal line running through the cells :



- ▶ The first operator on the information conveyor, is a **point wise multiplication** : this will allow **to cancel** (remove via a multiplication y a very small value) **irrelevant information**.
- ▶ The second operator is a **poinwise addition** : this allows **to add information** to the cell state
- ▶ This operations are regulated by structures called **GATES**

## Gates

- **Gates** regulate how information is either cancelled or let through. Gates composed out of sigmoid neural net layer (whose outputs component values are between 0 and 1).

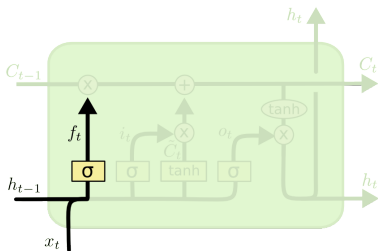


- A gate output component value of 0 means "stop this information"
- A gate output component value of 1 means "let this information through"

## Step by step LSTM

### Forget gate

The purpose of the forget gate is to decide what information must be thrown away from the cell state, by considering both the new input  $x_t$  and the previous time internal state  $h_{t-1}$ :



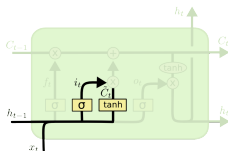
$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

## Step by step LSTM

### Input gate

The input gate decides which information must be stored in the cell state. This is a three stages operation :

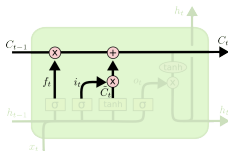
- ▶ First decide which values should be updated
- ▶ Next, a 'tanh' layer creates a new candidate  $\tilde{C}_t$  that could be added to the state



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

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

- ▶ Third, combine the two previous results to create the state update :

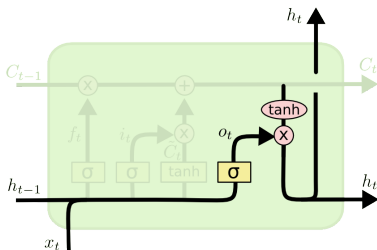


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

## Step by step LSTM

### Output gate

The output's gate role is to output a value  $h_t$  that combines both information from the current input  $x_t$ , the information state  $C_t$  and the previous hidden state  $h_{t-1}$  :



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

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

## LSTM : Summary

- ▶ A LSTM network is a **recurrent network** or equivalently a chain of **identical modules** or cells
- ▶ In comparison to simple RNN, LSTM contains **not only a hidden state but also a cell state**
- ▶ The **cell state** is transferred between modules with a **constant unit gain**, thus **avoiding vanishing gradient** problems
- ▶ The **cell state** allows **long term memory** storing (again, no vanishing gradient problem arises for  $C_t$ )
- ▶ The **hidden states** cares for **short term memory**
- ▶ The structure of each cell is based on three (3) **gates** :
  1. A **forget** gate, deciding which internal information to cancel
  2. An **input** gate, deciding which information to store in the internal cell state
  3. An **output** gate, combining last input, previous hidden state and updated cell state to provide the new hidden state



### LSTM variants

Some variants of the preceding structure may be found in the literature, among which some popular are just cited here :

- ▶ Adding "Peephole connections" : all three or only a subset of the NNs ( $\sigma$ ) also take  $C_{t-1}$  as inputs.
- ▶ Some authors proposed to couple forget and input gates : all forget info is also replaced at the input stage.
- ▶ Gated Recurrent Units (GRU) -2014- : merge input and output to make an 'update gate', merge  $C_t$  and  $h_t$

For LSTM application example, see Notebook LSTM\_RNN.jpynb.