Introduction to Deep Learning

Convolution (part 2) and (gated) recurrent Network

Alexandre Allauzen

ESPCI PARIS | PSL

Dauphine UNIVERSITÉ PARIS | PSL

MILES Machine Intelligence and Learning Systems

02/03/20

Outline

Outline

Sequence of discrete symbols classification

Movie review classification

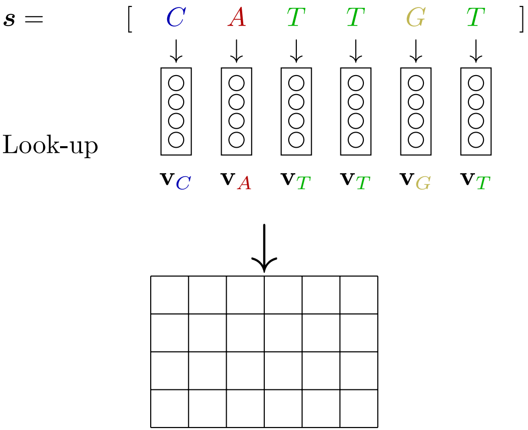
my wonderful friend took me to see this movie for our anniversary. it was terrible.	$: \; \boldsymbol{x} \in \mathbb{R}^D \longrightarrow c \in \{0,1\}$

Enhancer Identification in DNA Sequences

A T C G A T C G ... G T A A T C G	$: \; \boldsymbol{x} \in \mathbb{R}^D \longrightarrow c \in \{0,1\}$

- $\mathcal{D} = (\boldsymbol{x}_{(i)}, c_{(i)})_{i=1}^{N}$
- The input is a sequence $\rightarrow$ how to build $\boldsymbol{x}$?
- A sequence of discrete symbols $\in \mathcal{V}$
- Symbols interact with each other, the neighborhood is important

Embedding of discrete symbols



Convolution 1D

Extract a frame, or a window, and apply a “filter”

The input sequence of $L = 6$ vectors in $\mathbb{R}^D$, $D = 4$

$x_{1,4}$	$x_{2,4}$	$x_{3,4}$	$x_{4,4}$	$x_{5,4}$	$x_{6,4}$
$x_{1,3}$	$x_{2,3}$	$x_{3,3}$	$x_{4,3}$	$x_{5,3}$	$x_{6,3}$
$x_{1,2}$	$x_{2,2}$	$x_{3,2}$	$x_{4,2}$	$x_{5,2}$	$x_{6,2}$
$x_{1,1}$	$x_{2,1}$	$x_{3,1}$	$x_{4,1}$	$x_{5,1}$	$x_{6,1}$

The filter: kernel size of $ks = 2$

$w_{1,4}$	$w_{2,4}$
$w_{1,3}$	$w_{2,3}$
$w_{1,2}$	$w_{2,2}$
$w_{1,1}$	$w_{2,1}$

The output value (output channel)

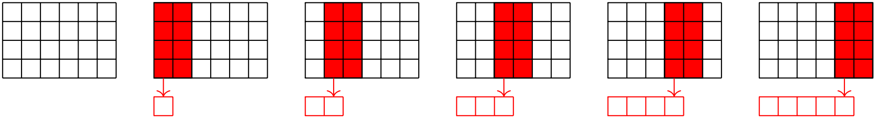
$$\text{At time } t = 1,\ h_1 = \sum_{i,j} w_{i,j} \times x_{i,j}$$

Convolution 1D

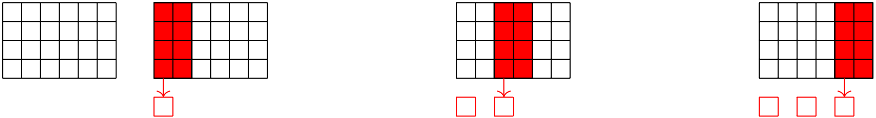
Stride, or a sliding window

The input is a matrix $(L = 6, d = 4)$, one filter of kernel size $= 2$:

With a stride $= 1$



With a stride $= 2$



Convolution 1D

With 2 output channels

$L = 6,\ D = 4$

$x_{1,4}$	$x_{2,4}$	$x_{3,4}$	$x_{4,4}$	$x_{5,4}$	$x_{6,4}$
$x_{1,3}$	$x_{2,3}$	$x_{3,3}$	$x_{4,3}$	$x_{5,3}$	$x_{6,3}$
$x_{1,2}$	$x_{2,2}$	$x_{3,2}$	$x_{4,2}$	$x_{5,2}$	$x_{6,2}$
$x_{1,1}$	$x_{2,1}$	$x_{3,1}$	$x_{4,1}$	$x_{5,1}$	$x_{6,1}$

Filters:kernel size of $ks = 2$

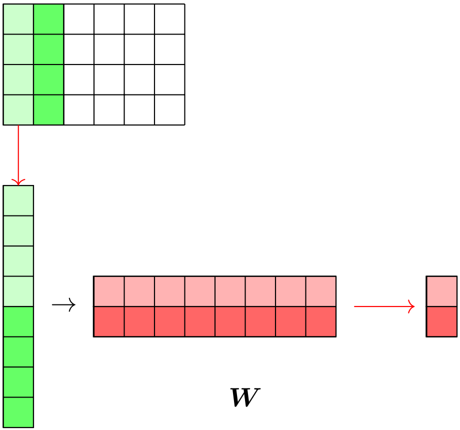
$w^{(2)}_{1,4}$	$w^{(2)}_{2,4}$
$w^{(2)}_{1,3}$	$w^{(2)}_{2,3}$
$w^{(2)}_{1,2}$	$w^{(2)}_{2,2}$
$w^{(2)}_{1,1}$	$w^{(2)}_{2,1}$

The output value (output channel)

$$h_{1,1} = \sum_{i,j} w^{(1)}_{i,j} \times x_{i,j}$$

$$h_{2,1} = \sum_{i,j} w^{(2)}_{i,j} \times x_{i,j}$$

Another wiew for two output channels



- Two filters applied to the same frame (or window)
- Each filter generates one feature

$\rightarrow$ a vector of two values, two features $(h_{1,1}, h_{2,1})$

- $h_{c,t}$ is the feature extracted for the channel c at time t.
- $\boldsymbol{W}$ gathers the parameters of the filters in one matrix
- The parameters $\boldsymbol{W}$ are learnt

Exercise

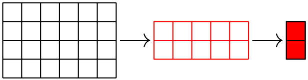
Stride

- With an input of length L, a kernel size of ks and a stride $= 1$, what is the output dimension ?
- And with a stride $= 2$, what is the output dimension ?
- With a stride of s ?

Side effect

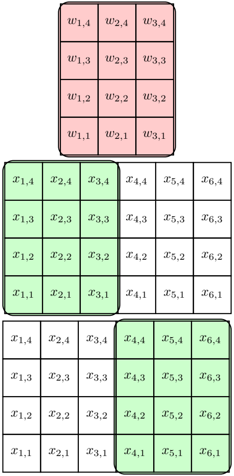
- The first (and last) time steps are not processed as the others. How to correct this aspects ?
- How to ensure the same length in output (assuming a stride of 1) ?
- How to ensure that every inputs are “seen” equally ?

Pooling over time



- Mean pooling
- Max pooling, and k-max pooling

Exercise



The convolution filter generates h_1 and h_2, and the pooling operation: $o = f(h_1, h_2)$. During training the loss gradient should be back-propagated:

$$\frac{\partial l}{\partial \boldsymbol{\theta}} = \frac{\partial l}{\partial o} \times \frac{\partial o}{\partial h_i} \times \frac{\partial h_i}{\partial \boldsymbol{\theta}}$$

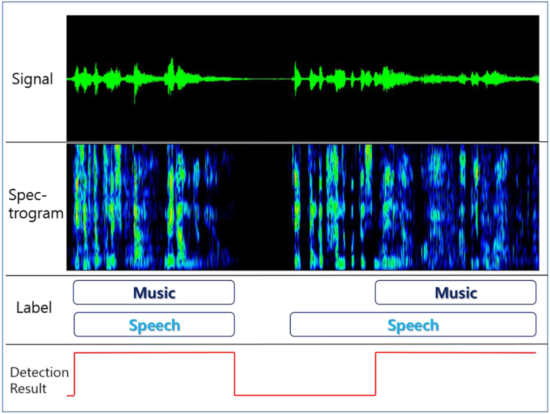
With mean-pooling:

1. Write f and $\frac{\partial l}{\partial h_i}$ given $\frac{\partial l}{\partial o}$
2. With 2 output channels, at time 1 $\rightarrow (h_{1,1}, h_{1,2})$ and at time 2 $\rightarrow (h_{2,1}, h_{2,2})$, and the pooling $\rightarrow (o_1, o_2)$. Compute the back-prop. gradient.
3. For an input of length L, how many updates for one filter ?

Reconsider the questions for the max-pooling !

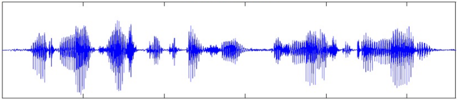
Audio classification / segmentation

(Jang et al.2019)



- Classification at each time step
- But the context is crucial (for the input and the output) !
- **Spectrogram ?**

Interlude: the input spectrogram



↓

Segmentation in frames (Hamming window)

↓

F.F.T

↓

Mel Filters

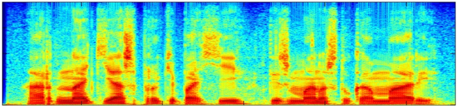
↓

Log

↓

D.C.T

↓



Human activity Recognition

The data

10k samples of fixed length (128 points at 50kHz) [a]:

- x, y, and z accelerometer data (linear acceleration)
- and the three gyroscopic data (angular velocity)

The classes are: Walking, Upstairs, Downstairs, Sitting, Standing, Laying.

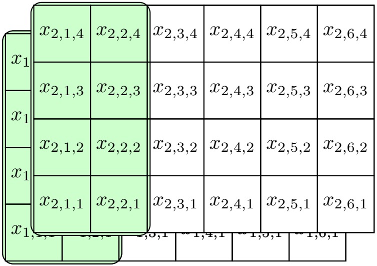
[a] https://archive.ics.uci.edu/ml/datasets/human+activity+recognition+using+smartphones

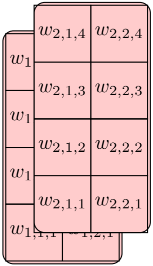
→ Two different input channels (accelerometer and gyroscopic).

→ How to define a convolution filter for two input channels ?

→ The sequence is long but of fixed size, the overall max-pooling is maybe not the best option.

Convolution 1D with 2 input channels





The output value (output channel)

$$\text{At time } t = 1,\ h_1 = \sum_{k,i,j} w_{k,i,j} \times x_{k,i,j}$$

We can have multiple input and output channels.

Convolution 1D and max-pooling in pytorch

Convolution 1D

torch.nn.Conv1d(

- in_channels,
- out_channels,
- kernel_size,
- stride=1,
- padding=0,
- dilation=1, ...

)

Max-Pooling 1D

torch.nn.MaxPool1d(

- kernel_size,
- stride=None,
- padding=0,
- dilation=1,
- return_indices=False,
- ceil_mode=False

)

Outline

Image classification

An image = (2D array of values) × (number of channels)

2D array

The spatial structure:

- A 2D real space
- With distance

Channels

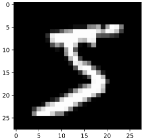
- For image : R,G,B
- In fluid mechanics: pression, velocity, ...
- In general: different measures on the same spatial domain

Sources

Many figures and examples are inspired by, or extracted from the course of the Stanford course of Fei-Fei Li.

http://cs231n.stanford.edu/

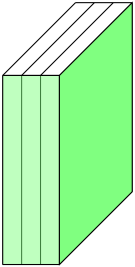
Image classification



$\rightarrow$

$\boldsymbol{x} \in \mathbb{R}^{784} \longrightarrow c \in \{0, 1, 2, \ldots, 9\}$

Another example with a color image (3 channels) of 32×32 pixels

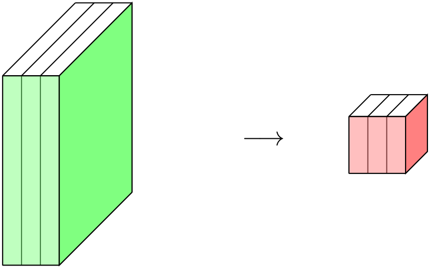


$\rightarrow$

$\boldsymbol{x} \in \mathbb{R}^{3072} \longrightarrow c \in \{0, 1, 2, \ldots, 9\}$

Convolution in 2D

The basics



$3 \times 32 \times 32$ image

$3 \times 5 \times 5$ filter

Convolution of the filter with the image:

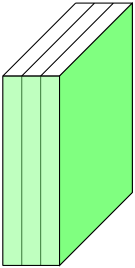
- Sliding the filter along the two axis
- Computing the “dot product” at each step

→ preserve the spatial structure along the channels

→ each step extract a “local” and spatial feature

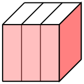
Convolution in 2D

With a single output channel



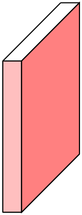
$3 \times 32 \times 32$ image

$\longrightarrow$



$3 \times 5 \times 5$ filter

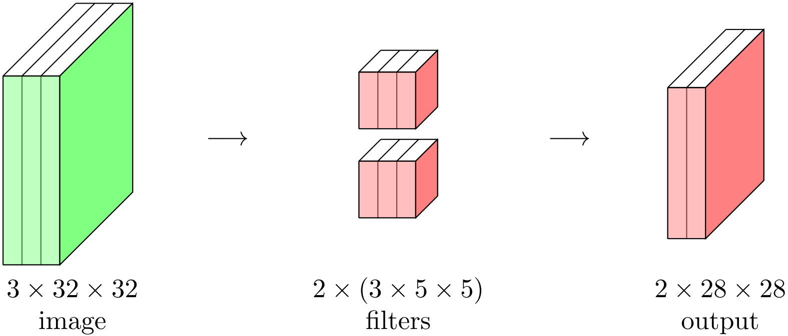
$\longrightarrow$



$1 \times 28 \times 28$ output

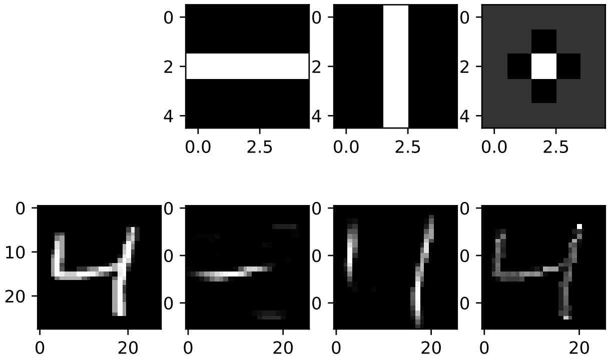
Convolution in 2D

Add a second output channel



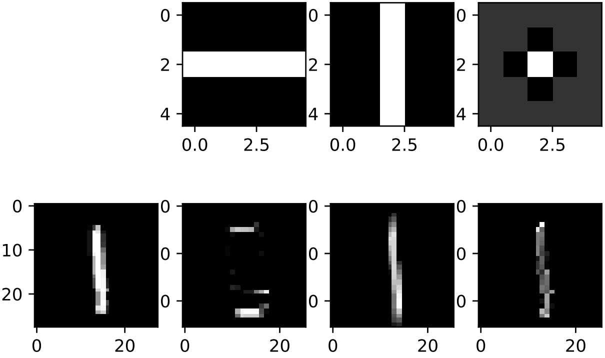
Motivation for convolution

Extract "low level" features



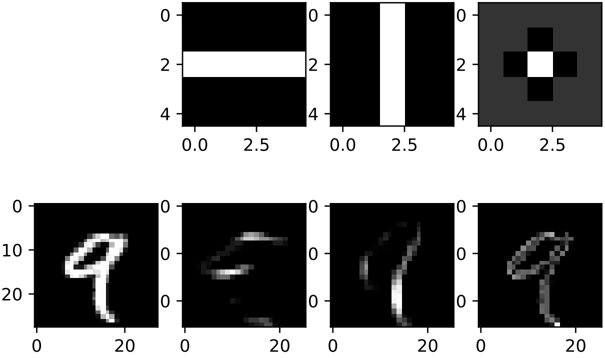
Motivation for convolution

Extract “low level” features



Motivation for convolution

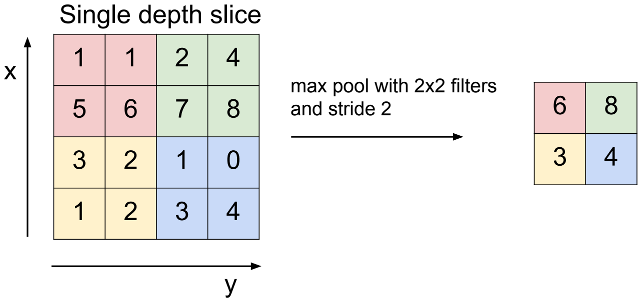
Extract "low level" features



Max-pooling or Downsampling in 2D

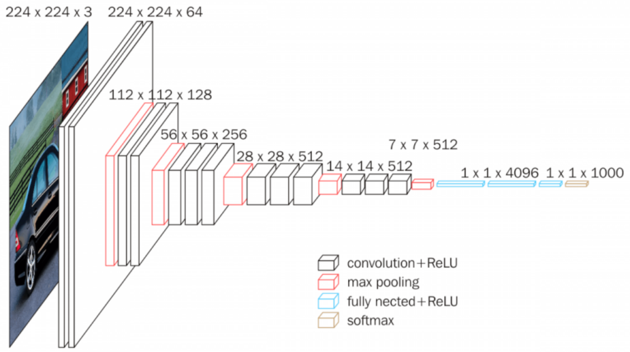
The goal

- Convolution extracts local features (followed by non-linearity)
- Max-pooling acts as a selection, compression, or contraction operator
- The back-propagation promote feature saliency for each channel



Architecture of deep convolution NNet for image processing

VGGNet (Simonyan and Zisserman2015)



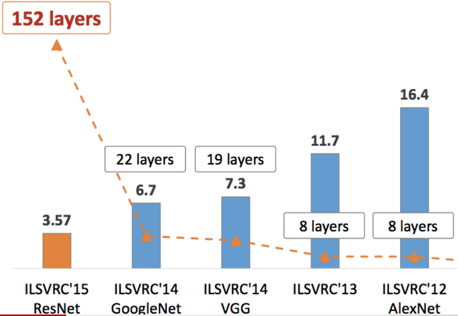
Conv. layers with kernel size of 3×3, stride and padding, followed by pooling layers (max on 2×2 with stride 2).

Architecture of deep convolution NNet for image processing

A summary

The basic block

- A convolution layer with a ReLU activation followed by a max-pooling layer
- An alternative is two convolution layers in a Residual block (ResNet (He et al.2015))



Outline

Sequence processing

Sequence generation model

$$P(w_1^L) = P(w_1, w_2, ...w_L) = \prod_{i=1}^{L} P(w_i|w_1^{i-1}), \quad \forall i, w_i \in \mathcal{V}$$

with the n**-gram assumption** (convolution):

$$P(w_1^L) = \prod_{i=1}^{L} P(w_i|w_{i-n+1}^{i-1}), \quad \forall i, w_i \in \mathcal{V},$$

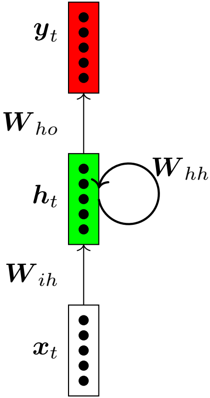
in the **recurrent** way

$$P(w_i|w_1^{i-1})$$

Sequence representation

$$\boxed{\begin{array}{c} \text{A T C G A T C G} \\ \cdots \text{ G T A A T C G} \end{array}} \; : \; \boldsymbol{x} \in \mathbb{R}^D \longrightarrow c \in \{0, 1\}$$

Recurrent Cell



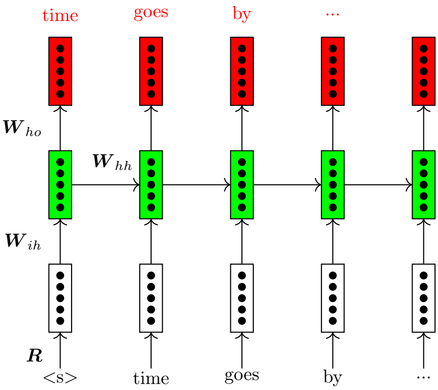
A dynamic system, at time t:

- maintains a hidden representation, the internal state: $\boldsymbol{h}_t$
- Updated with the observation of $\boldsymbol{x}_t$ and the previous state $\boldsymbol{h}_{t-1}$
- The prediction $\boldsymbol{y}_t$ depends on the internal state ($\boldsymbol{h}_t$)
- For a language model, $\boldsymbol{x}_t$ comes from word embeddings

The same parameter set is shared across time steps

Recurrent network sequence model

Unfolding the structure: a deep-network



At each step t

- Read the word $w_t \rightarrow \boldsymbol{x}_t$ from $\boldsymbol{R}$
- Update the hidden state $\boldsymbol{h}_t = f(\boldsymbol{W}_{ih}\boldsymbol{x}_t + \boldsymbol{W_{hh}h_{t-1}})$
- The word at $t+1$ can be predicted from $\boldsymbol{h}_t$:

$$\boldsymbol{y}_t = g(\boldsymbol{W}_{ho}\boldsymbol{h}_t)$$

- g is the softmax function over the vocabulary

Training recurrent language model

Training algorithm

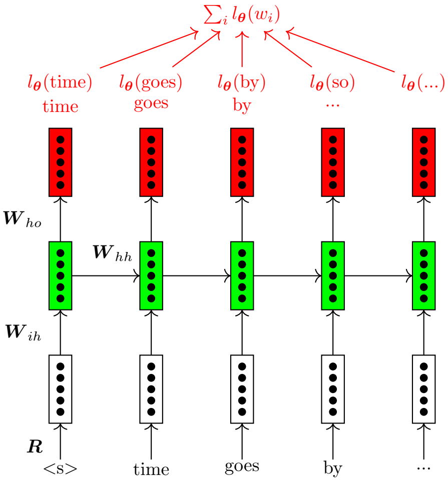
Back-Propagation through time or BPTT
(Rumelhart et al.1986; Mikolov et al.2011):

- for each step t
  - compute the loss gradient
  - Back-Propagation through the unfolded structure

Inference

- Cannot be easily integrated to conventional approaches (ASR, SMT, ...)
- A powerful device for end-to-end system

Training recurrent language model - 2



Mini-batching for RNN

Mini-batching makes things much faster!

Mini-batch

- Add a dimension to your input example $\boldsymbol{x}$
- Forward propagation of the whole mini-batch at a time
- Compute the loss and back-propagation

But

- mini-batching in RNNs is harder than in feed-forward networks
- Each word depends on the previous word
- Sequences are of various length

Batching / Padding / Masking

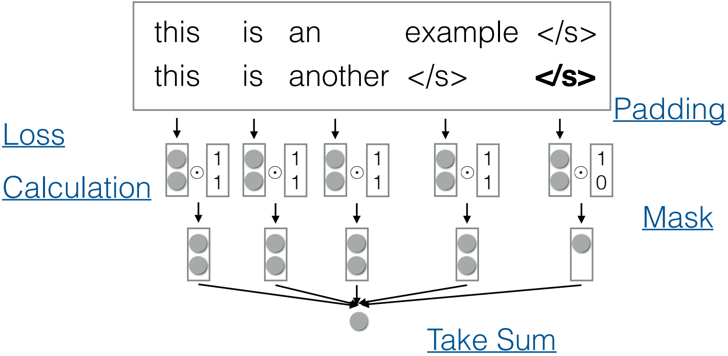
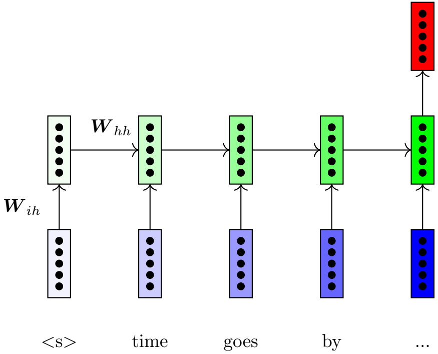


Figure from G. Neubig

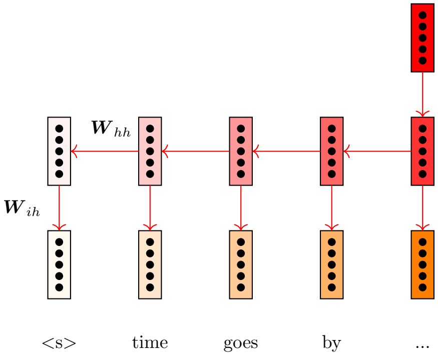
Short term memory



During inference :

- With the distance, the influence of observations reduces
- The memory is limited
- No way to keep/skip information

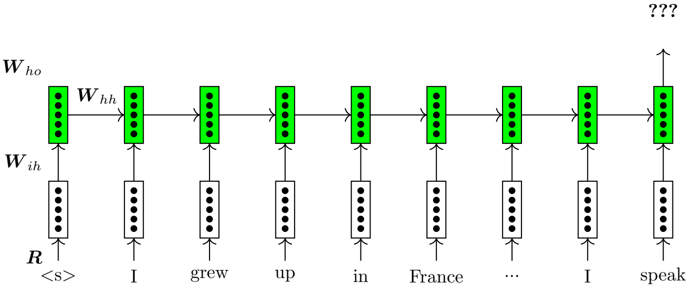
Vanishing gradient issue



During training:

- The gradient diminishes at each backward step
- No long term propagation of the gradient

The Problem of Long-Term Dependencies



- Recent observations hide the older ones (Bengio et al.1994)
- The vanishing (exploding) gradient is a real issue (Pascanu et al.2013)

Solutions

Improved optimization

- Gradient clipping (Pascanu et al.2013)
- Hessian-Free optimzation (Martens and Sutskever2012) or natural gradient (Desjardins et al.2013; Ollivier2015)

Modified unit

A recurrent network should be able to mitigate the observations *vs* its internal state:

- LSTM or Long-Short-Term-Memory cell (Hochreiter and Schmidhuber1997; Graves and Schmidhuber2009)
- Gated Recurrent Unit or GRU (Cho et al.2014)

Gradient clipping

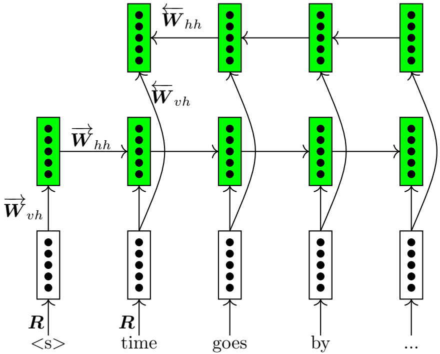
A simple and efficient trick

Given a threshold γ, before each update:

- Compute the norm of the gradient (at each time step) : $||\nabla_{\boldsymbol{\theta}}||$
- If $||\nabla_{\boldsymbol{\theta}}|| > \gamma$:

$$\nabla_{\boldsymbol{\theta}} \leftarrow \frac{\gamma}{||\nabla_{\boldsymbol{\theta}}||}\nabla_{\boldsymbol{\theta}}$$

Sentence encoder: the bi-recurrent solution



A each step t, from left to right

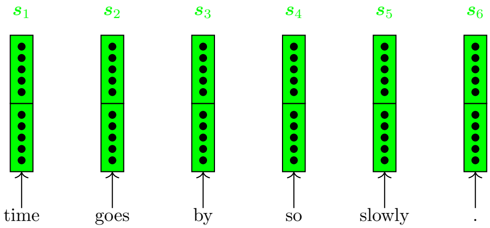
- $w_t \rightarrow \boldsymbol{x}_t$
- $\overrightarrow{\boldsymbol{h}}_t = f(\overrightarrow{\boldsymbol{W}}_{vh}\boldsymbol{x}_t + \overrightarrow{\boldsymbol{W}}_{hh}\overrightarrow{\boldsymbol{h}}_{t-1})$

A each step t, from right to left

- $w_t \rightarrow \boldsymbol{x}_t$
- $\overleftarrow{\boldsymbol{h}}_t = f(\overleftarrow{\boldsymbol{W}}_{vh}\boldsymbol{x}_t + \overleftarrow{\boldsymbol{W}}_{hh}\overleftarrow{\boldsymbol{h}}_{t-1})$

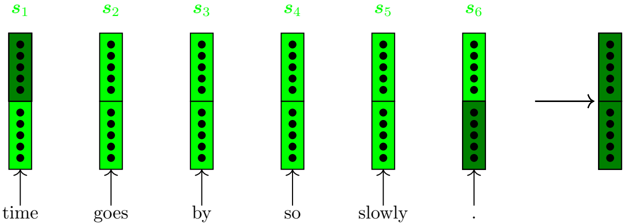
$[\overrightarrow{\boldsymbol{h}}_t; \overleftarrow{\boldsymbol{h}}_t]$ **: contextualized representation of** w_t

Sequence representation



- A bi-recurrent encoder
- Each observation is associated to a vector: a contextualised representation s_t

Sequence representation



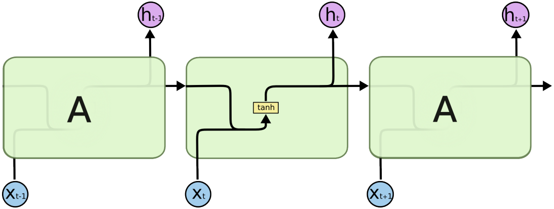
- A bi-recurrent encoder
- Each observation is associated to a vector: a contextualised representation s_t

Outline

Introduction to LSTM

The standard recurrent cell

- A recurrent cell is neural network layer
- Conveyor belt : the hidden state

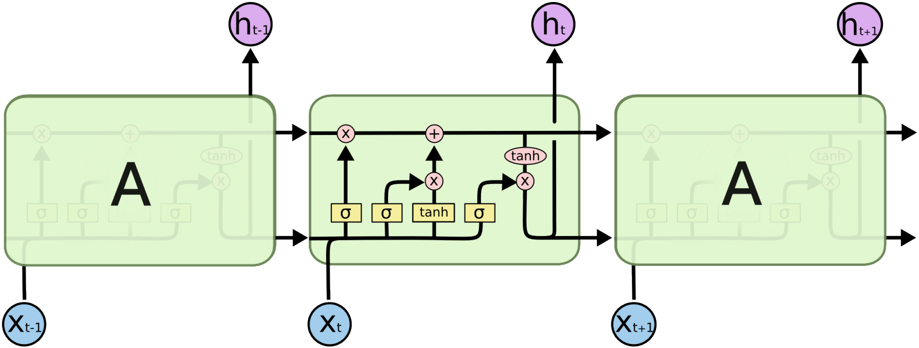


Lines carry a vectors

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

Introduction to LSTM

The LSTM cell



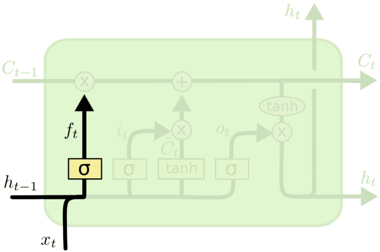
- LSTM introduces a second channel:
  the cell state
- The cell is now four neural layers, interacting in a very special way
- It acts as a memory
- Gates control the memory

Roadmap of inference

1. Memory organization
   1. what should be kept ?
   2. what shoud be updated ?
2. Update the cell state
3. Filter the state to provide the "hidden" state

LSTM : Control flow - 1

What should be forgotten from the previous cell state ?



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

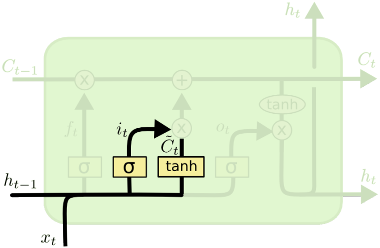
Action

The sigmoid (forget gate) answers for each component:

- 1: to keep it,
- 0 to forget it, or
- a value in-between to mitigate its influence

LSTM : Control flow - 2

What should be taken into account ?



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

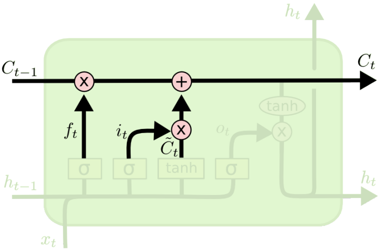
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

Actions

- Create the update $\tilde{C}_t$ of the cell state
- and its contribution i_t (the input gate with a sigmoid activation)

LSTM : Control flow - 3

Write the new state



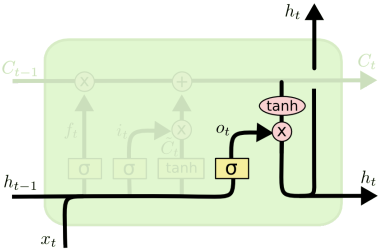
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Actions

- Merge the old cell state modified by the forget gate
- with the new input

LSTM : Control flow - 4

Write the new hidden state



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

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

Actions

- Decide what parts of the (filtered) cell state to output o_t
- Compute the hidden state

LSTM summary

A special kind of recurrent architecture

- keep or skip information in memory
- to reset or mitigate the long-term memory

Consequences: an efficient model of sequences

- Overcome the vanishing gradient issue
- Very promising results in generation

State of the art

- Stacked LSTM and Bi-recurrent encoder
- Variants: Gated recurrent units (GRU) (Cho et al.2014) or a more complicated one (Gers and Schmidhuber2000)

Y. Bengio, P. Simard, and P. Frasconi.
1994.
Learning long-term dependencies with gradient descent is difficult.
Trans. Neur. Netw., 5(2):157–166, March.

Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio.
2014.
Learning phrase representations using rnn encoder–decoder for statistical machine translation.
In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar, October. Association for Computational Linguistics.

Guillaume Desjardins, Razvan Pascanu, Aaron Courville, and Yoshua Bengio.
2013.
Metric-free natural gradient for joint-training of boltzmann machines.
In *International Conference on Learning Representations (ICLR)*.

F.A. Gers and J. Schmidhuber.
2000.
Recurrent nets that time and count.
In *Neural Networks, 2000. IJCNN 2000, Proceedings of the IEEE-INNS-ENNS International Joint Conference on*, volume 3, pages 189–194 vol.3.

Alex Graves and Juergen Schmidhuber.
2009.
Offline handwriting recognition with multidimensional recurrent neural networks.
In D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, editors, *Advances in Neural Information Processing Systems 21*, pages 545–552. Curran Associates, Inc.

Klaus Greff, Rupesh Kumar Srivastava, Jan Koutník, Bas R Steunebrink, and Jürgen Schmidhuber.
2015.
Lstm: A search space odyssey.
arXiv preprint arXiv:1503.04069.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun.
2015.
Deep residual learning for image recognition.

Sepp Hochreiter and Jürgen Schmidhuber.
1997.
Long short-term memory.
Neural Comput., 9(8):1735–1780, November.

Byeong-Yong Jang, Woon-Haeng Heo, Jung-Hyun Kim, and Oh-Wook Kwon.
2019.
Music detection from broadcast contents using convolutional neural networks with a mel-scale kernel.

EURASIP Journal on Audio, Speech, and Music Processing, 2019(1):11, Jun.

Rafal Józefowicz, Wojciech Zaremba, and Ilya Sutskever.
2015.
An empirical exploration of recurrent network architectures.
In *Proceedings of the International Conference of Machine Learning (ICML)*, pages 2342–2350.

James Martens and Ilya Sutskever.
2012.
Training deep and recurrent networks with hessian-free optimization.
In Grégoire Montavon, Genevieve B. Orr, and Klaus-Robert Müller, editors, *Neural Networks: Tricks of the Trade - Second Edition*, volume 7700 of *Lecture Notes in Computer Science*, pages 479–535. Springer.

Tomas Mikolov, Stefan Kombrink, Lukas Burget, Jan Cernocký, and Sanjeev Khudanpur.
2011.
Extensions of recurrent neural network language model.
In *Proceedings of the IEEE international conference on Acoustics, speech, and signal processing (ICASSP)*, pages 5528–5531.

Yann Ollivier.
2015.
Riemannian metrics for neural networks i: feedforward networks.
Information and Inference.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio.
2013.
On the difficulty of training recurrent neural networks.
In *Proceedings of the 30th International Conference on Machine Learning, ICML 2013, Atlanta, GA, USA, 16-21 June 2013*, volume 28 of *JMLR Proceedings*, pages 1310–1318. JMLR.org.

D. E. Rumelhart, G. E. Hinton, and R. J. Williams.
1986.
Parallel distributed processing: explorations in the microstructure of cognition, vol. 1.
chapter Learning internal representations by error propagation, pages 318–362. MIT Press, Cambridge, MA, USA.

Karen Simonyan and Andrew Zisserman.
2015.
Very deep convolutional networks for large-scale image recognition.
In *International Conference on Learning Representations*.