Deep-Learning:

Recurrent Neural Networks (RNN)

Pr. Fabien MOUTARDE
Center for Robotics
MINES ParisTech
PSL Université Paris

Fabien.Moutarde@mines-paristech.fr
http://people.mines-paristech.fr/fabien.moutarde

Acknowledgements

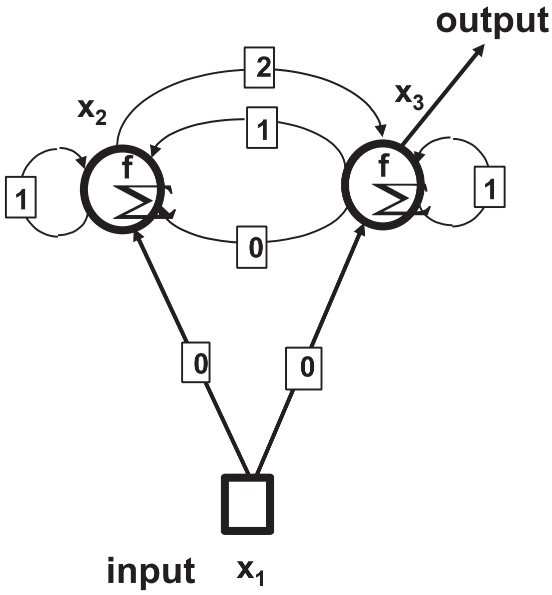
During preparation of these slides, I got inspiration and borrowed some slide content from several sources, in particular:

- **Fei-Fei Li + J.Johnson + S.Yeung: slides on "*Recurrent Neural Networks*" from the** "*Convolutional Neural Networks for Visual Recognition*" **course at Stanford**
  http://cs231n.stanford.edu/slides/2019/cs231n_2019_lecture10.pdf
- **Yingyu Liang: slides on "*Recurrent Neural Networks*" from the** "*Deep Learning Basics*" **course at Princeton**
  https://www.cs.princeton.edu/courses/archive/spring16/cos495/slides/DL_lecture9_RNN.pdf
- **Arun Mallya: slides "*Introduction to RNNs*" from the** "Trends in Deep Learning and Recognition" **course of Svetlana LAZEBNIK at University of Illinois at Urbana-Champaign**
  http://slazebni.cs.illinois.edu/spring17/lec02_rnn.pdf
- **Tingwu Wang: slides on "*Recurrent Neural Network*" for a course at University of Toronto**
  https://www.cs.toronto.edu/%7Etingwuwang/rnn_tutorial.pdf
- **Christopher Olah: online tutorial "*Understanding LSTM Networks*"**
  https://colah.github.io/posts/2015-08-Understanding-LSTMs/

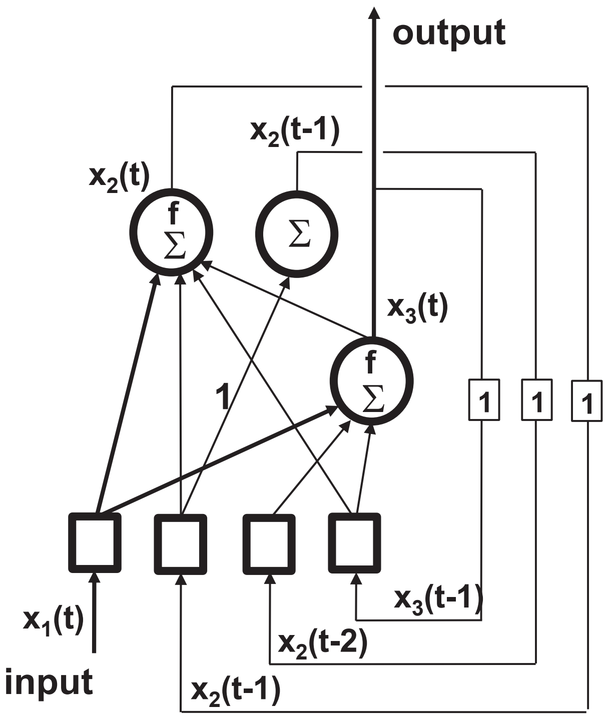
Outline

- **Standard Recurrent Neural Networks**
- Training RNN: BackPropagation Through Time
- LSTM and GRU
- Applications of RNNs

Recurrent Neural Networks (RNN)

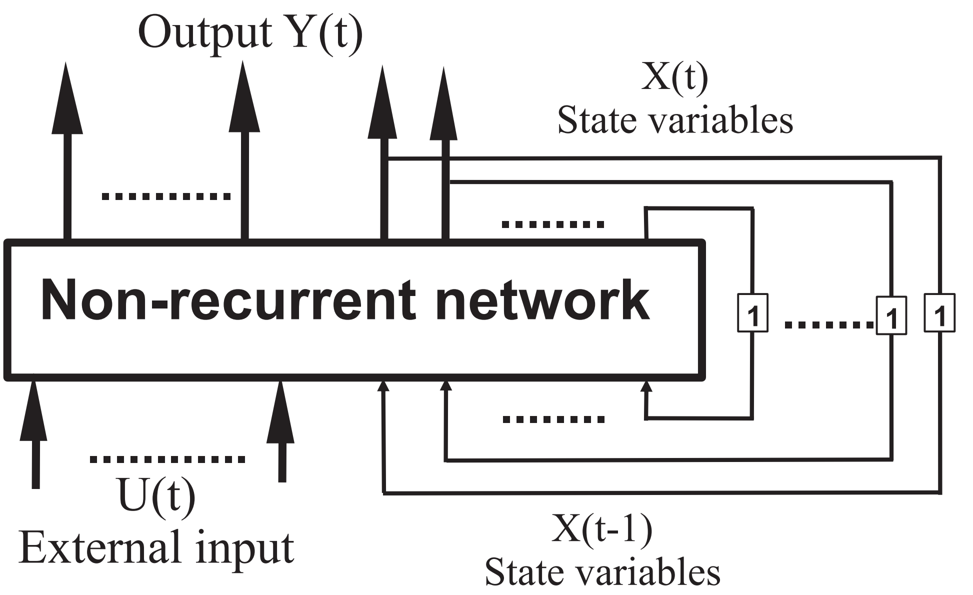


Time-delay for each connection

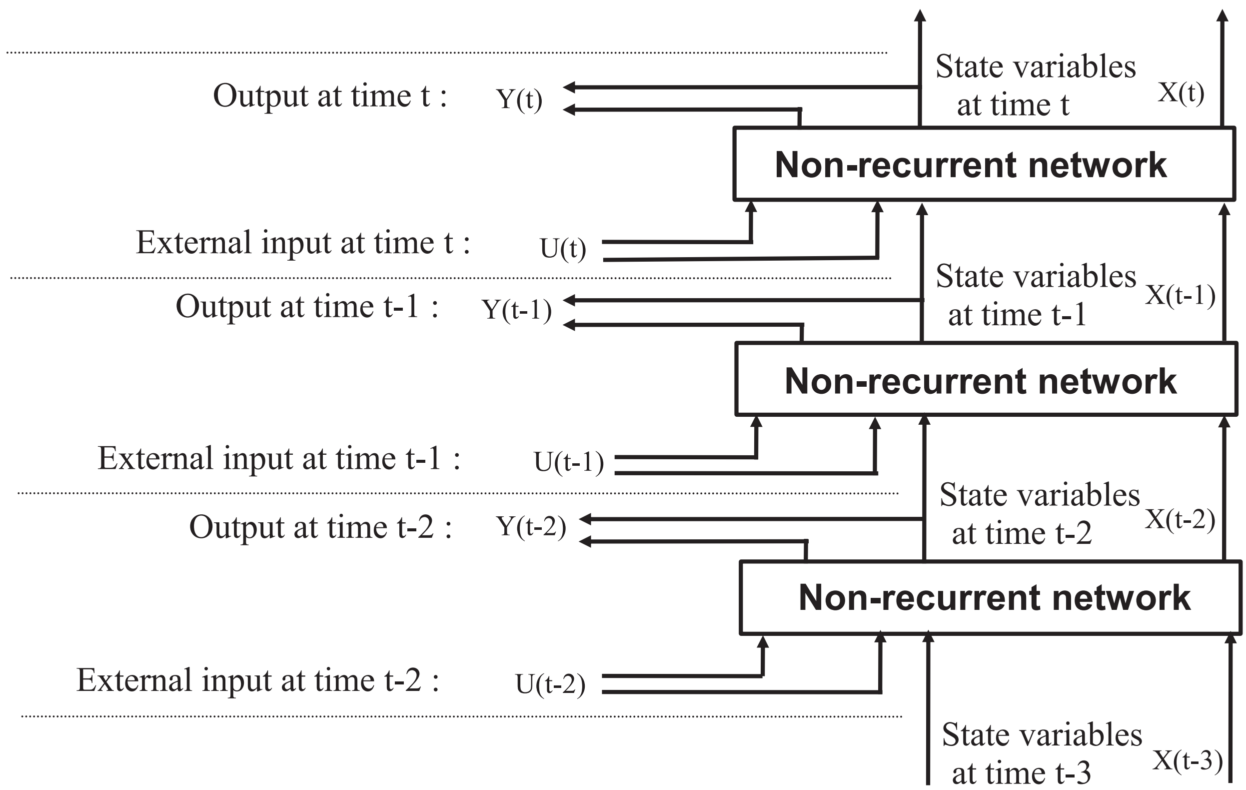


Equivalent form

Canonical form of RNN

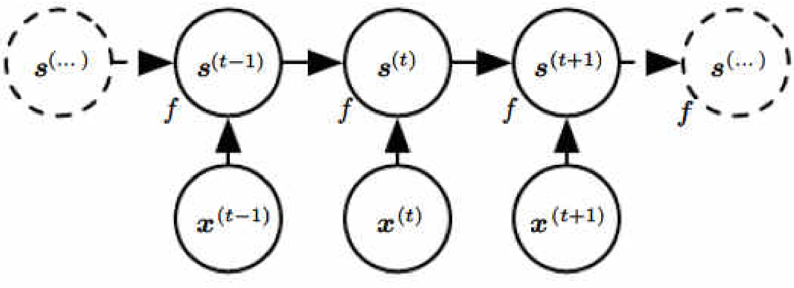


Time unfolding of RNN

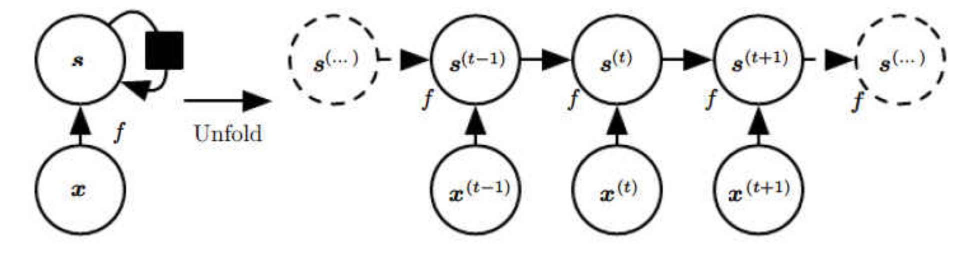


Dynamic systems & RNN

$$s^{(t+1)} = f\left(s^{(t)}, x^{(t+1)}\right)$$



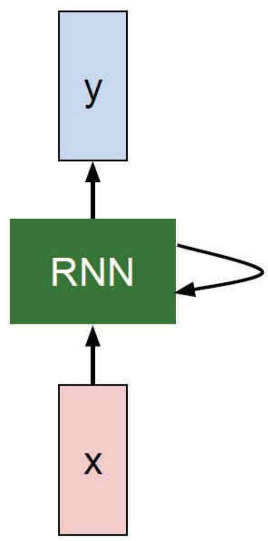
If using a Neural Net for f, this is EXACTLY a RNN!



Figures from *Deep Learning*, Goodfellow, Bengio and Courville

Standard ("vanilla") RNN

State vector s ←→ vector h of hidden neurons



$$\boxed{h_t} = \boxed{f_W}\left(\boxed{h_{t-1}}, \boxed{x_t}\right)$$

new state

old state

input vector at some time step

some function with parameters W

$$h_t = \tanh(W_{hh} h_{t-1} + W_{xh} x_t)$$

$$y_t = W_{hy} h_t$$

ou $y_t = \mathrm{softMax}(W_{hy} h_t)$

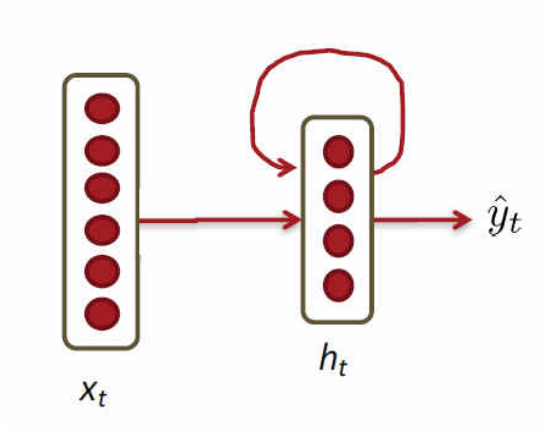
Advantages of RNN

The ***hidden state*** s of the RNN builds a kind of lossy summary of the past

RNN totally adapted to processing SEQUENTIAL data (same computation formula applied at each time step, but modulated by the evolving "memory" contained in state s)

Universality of RNNs: any function computable by a Turing Machine can be computed by a finite-size RNN (Siegelmann and Sontag, 1995)

RNN hyper-parameters

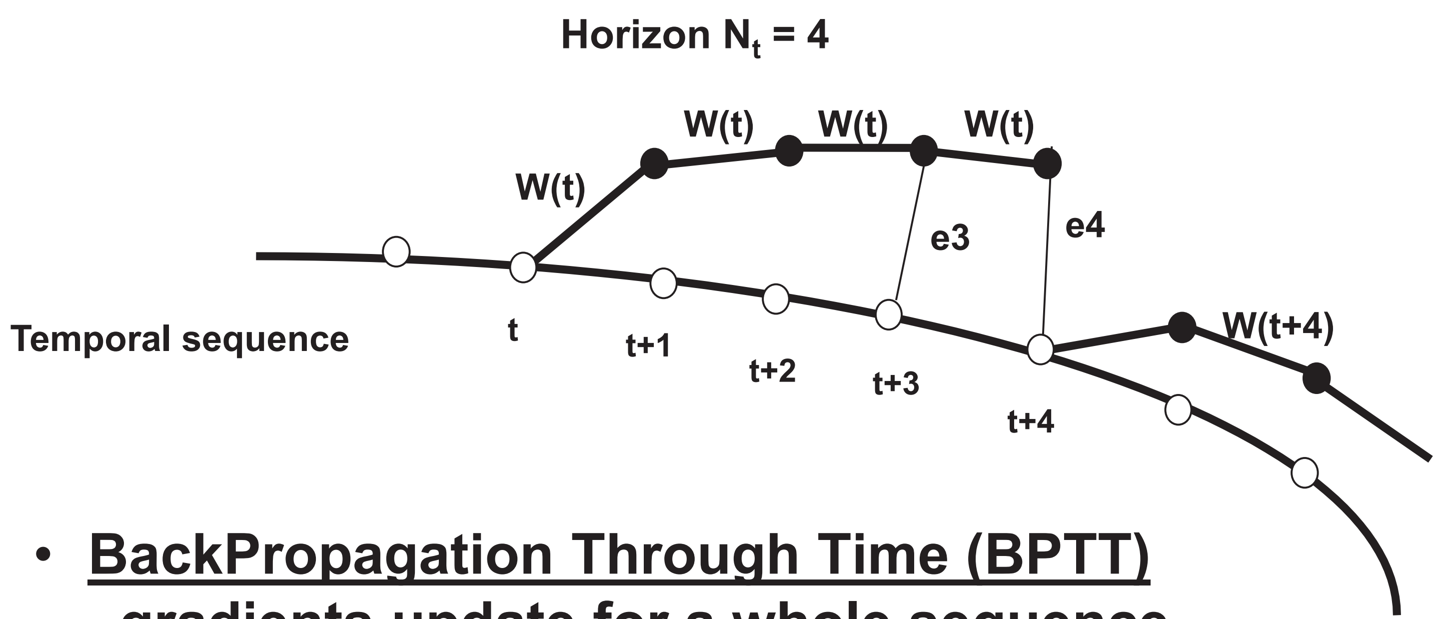


- As for MLP, main hyperparameter =
  size of hidden layer (=size of vector h)

Outline

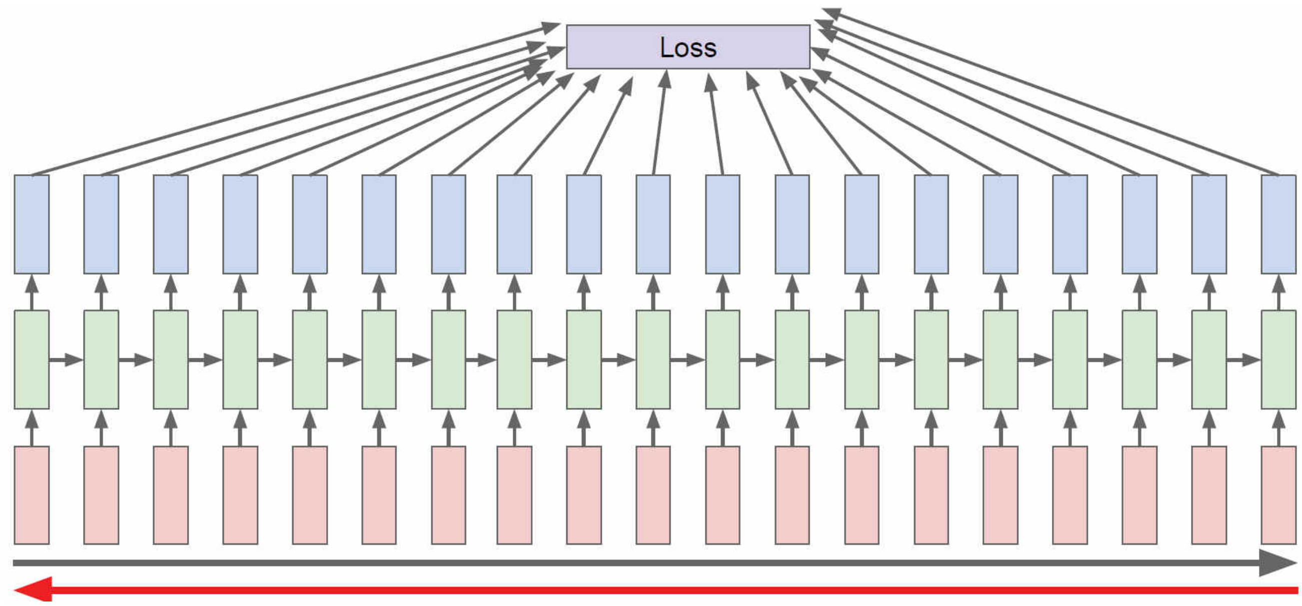
- Standard Recurrent Neural Networks
- **Training RNN: BackPropagation Through Time**
- LSTM and GRU
- Applications of RNNs

RNN training



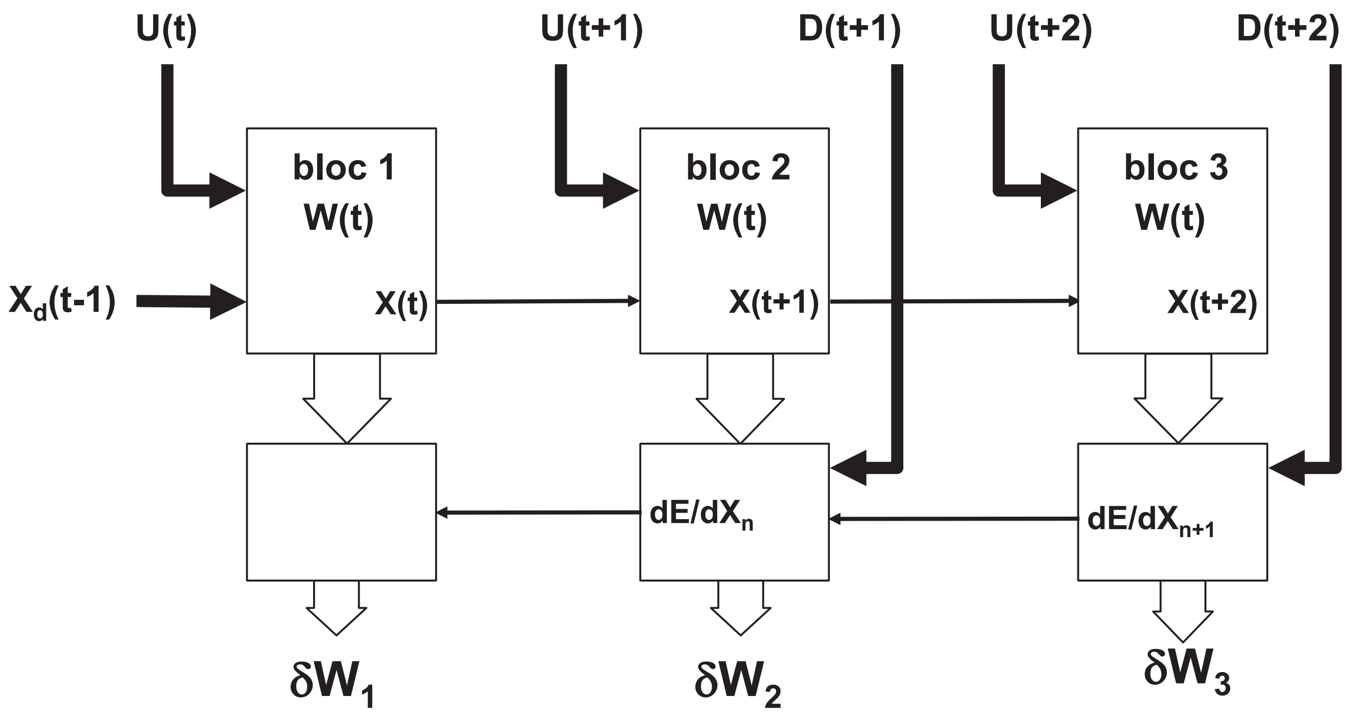
- <u>BackPropagation Through Time (BPTT)</u>
  gradients update for a whole sequence
- or Real Time Recurrent Learning (RTRL)
  gradients update for each frame in a sequence

BackPropagation THROUGH TIME (BPTT)



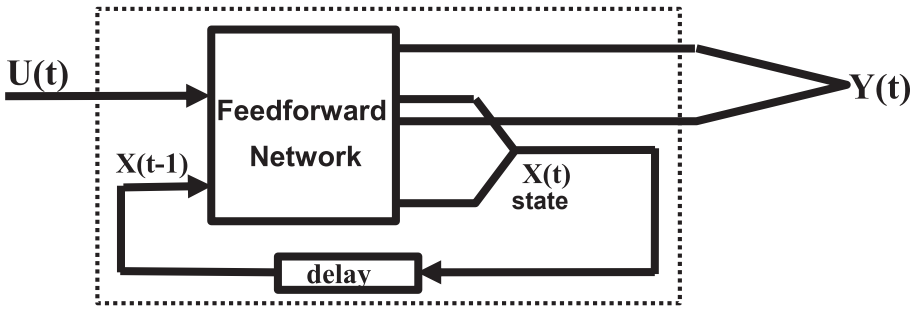
- **Forward through entire sequence to compute SUM of losses at ALL (or part of) time steps**
- **Then backprop through ENTIRE sequence to compute gradients**

BPTT computation principle



$$\delta W = \delta W_1 + \delta W_2 + \delta W_3$$

BPTT algorithm



W(t+N$_t$) = W(t) - λ grad$_W$(E) avec E = $\sum_\tau (Y_\tau - D_\tau)^2$

$$\frac{\partial E}{\partial W} = \sum_{t=1}^{T} \frac{\partial E_t}{\partial W}$$ **and** $$\forall t, \frac{\partial E_t}{\partial W} = \frac{\partial E_t}{\partial Y_t}\frac{\partial Y_t}{\partial X_{t-1}}\frac{\partial X_{t-1}}{\partial W}$$ **(chain rule)**

$$\frac{\partial X_t}{\partial W} = \sum_{k=1}^{t-1} \frac{\partial X_t}{\partial X_{t-k}}\frac{\partial X_{t-k}}{\partial W} \qquad \frac{\partial X_t}{\partial X_{t-k}} = \prod_{j=1}^{t} \boxed{\frac{\partial X_j}{\partial X_{j-1}}}$$

Jacobian matrix of the Feedforward net

Vanishing/exploding gradient problem

- **If eigenvalues of Jacobian matrix >1, then <u>gradients tend to EXPLODE</u>**
  ➔ Learning will never converge.
- **Conversely, if eigenvalues of Jacobian matrix <1, then <u>gradients tend to VANISH</u>**
  ➔ Error signals can only affect small time lags
  ➔ short-term memory.

➔Possible solutions for exploding gradient: *CLIPPING* trick

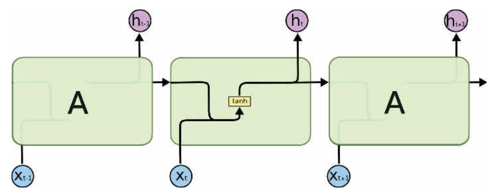
➔ Possible solutions for vanishing gradient:
- **use *ReLU* instead of tanh**
- ***change what is inside the RNN!***

Outline

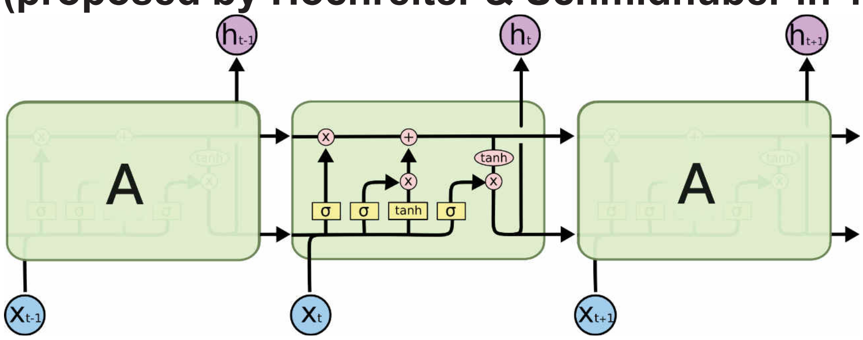
- Standard Recurrent Neural Networks
- Training RNN: BackPropagation Through Time
- **LSTM and GRU**
- Applications of RNNs

Long Short-Term Memory (LSTM)

Problem of *standard* RNNs = no actual LONG-TERM memory



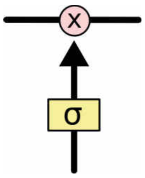
LSTM = RNN variant for solving this issue
(proposed by Hochreiter & Schmidhuber in 1997)



[Figures from https://colah.github.io/posts/2015-08-Understanding-LSTMs/]

- **Key idea = use "gates" that modulate respective influences of input and memory**

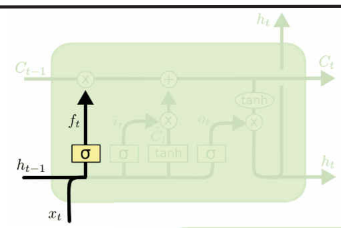
LSTM gates



Gate = ***pointwise*** multiplication by σ in]0;1[
➔ modulate between "*let nothing through*"
and "*let everything through*"

- **FORGET gate**

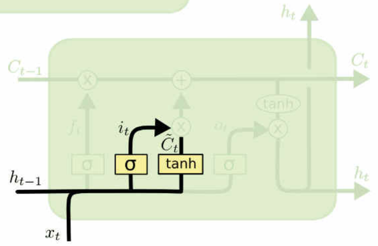
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$



- **INPUT gate**

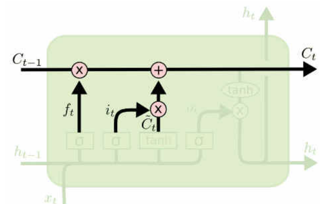
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$

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



➔ next state = mix between pure memory or pure new

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



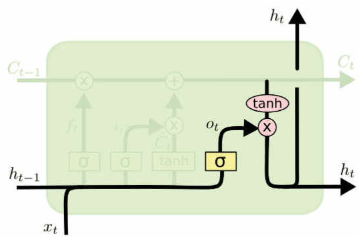
[Figures from https://colah.github.io/posts/2015-08-Understanding-LSTMs/]

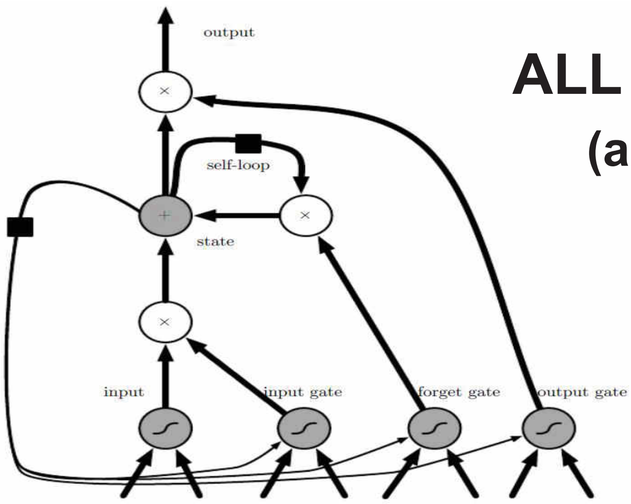
LSTM summary

- **OUTPUT gate**

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

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

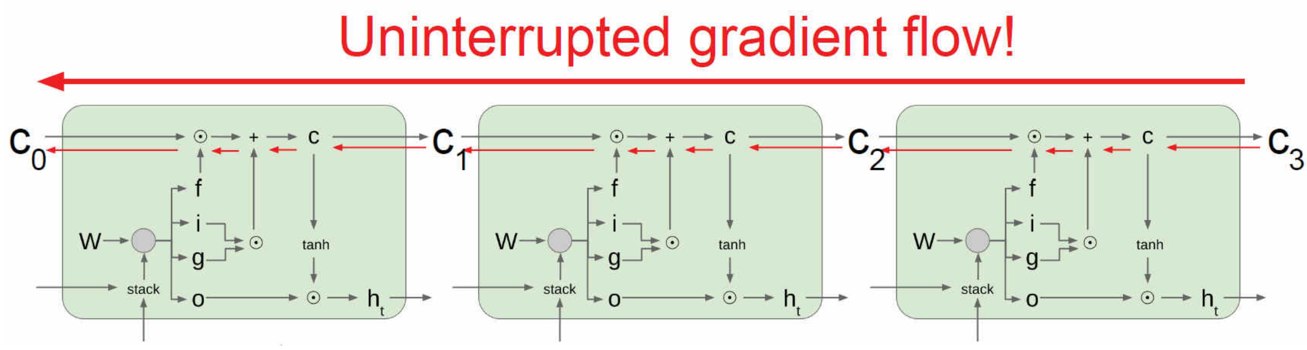




ALL weigths W_f, W_i, W_c and W_o (and biases) are LEARNT

[Figure from Deep Learning book by I. Goodfellow, Y. Bengio & A. Courville]

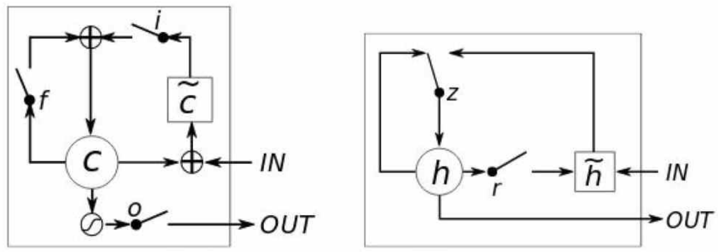
Why LSTM avoids vanishing gradients?



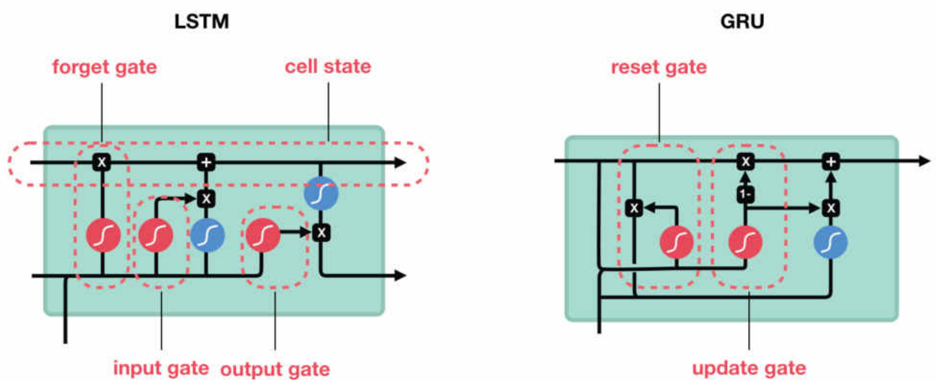
Gated Recurrent Unit (GRU)

Simplified variant of LSTM, with only 2 gates: a RESET gate & an UPDATE gate

(proposed by Cho, et al. in 2014)



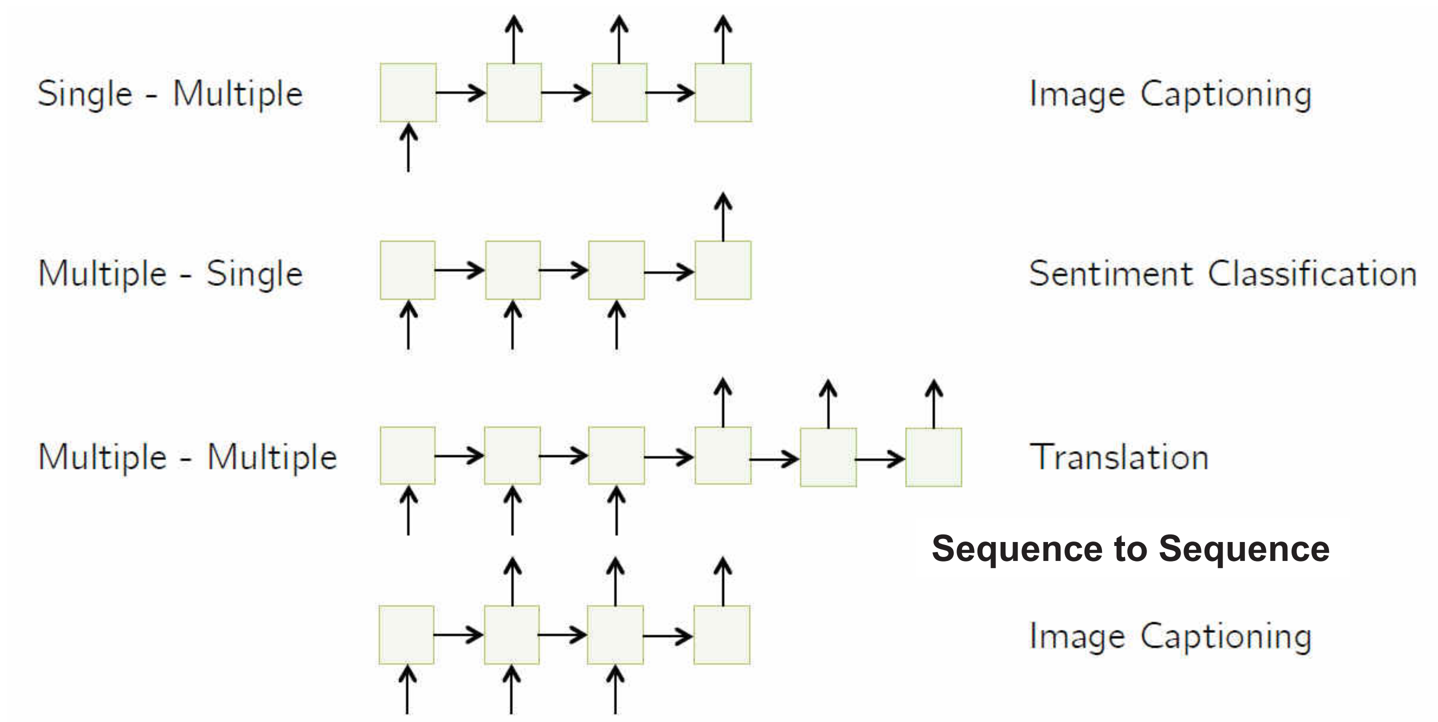
(a) Long Short-Term Memory (b) Gated Recurrent Unit



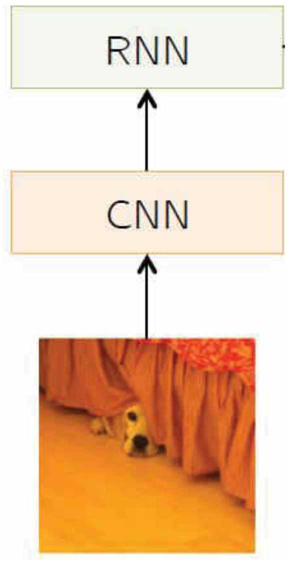
Outline

- Standard Recurrent Neural Networks
- Training RNN: BackPropagation Through Time
- LSTM and GRU
- **Applications of RNNs**

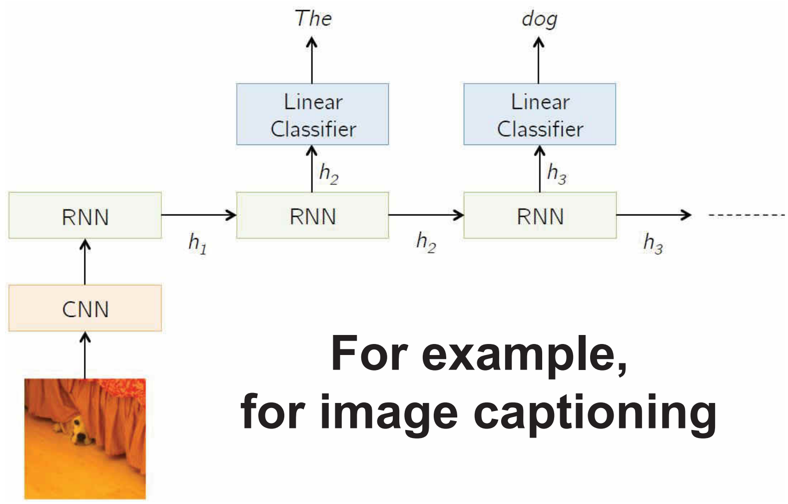
Typical usages of RNNs



Combining RNN with CNN

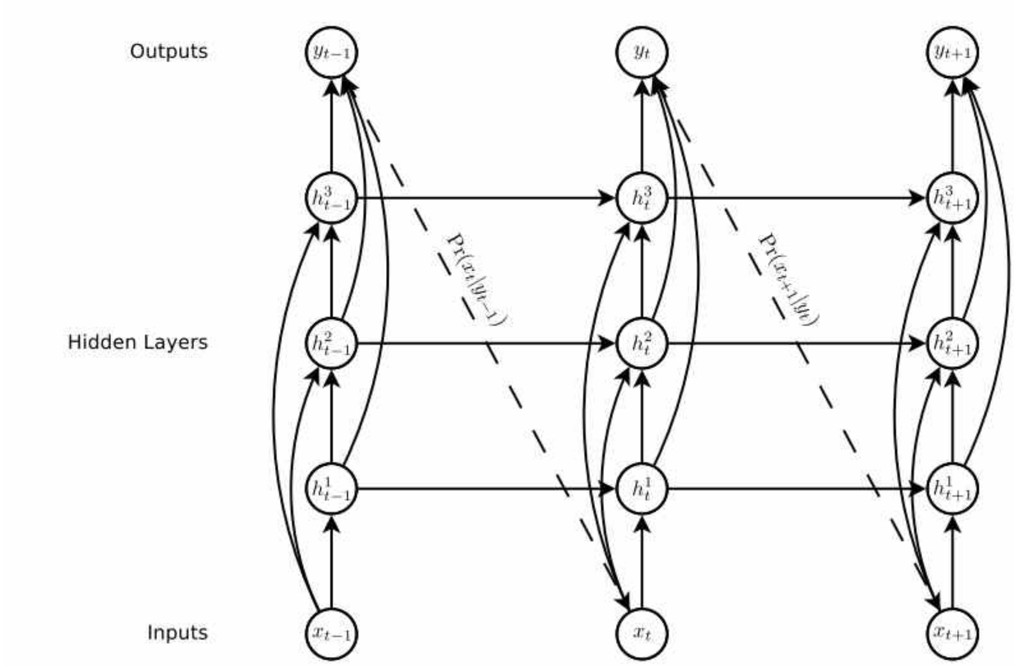


Input into RNN the features from last convolutional layer



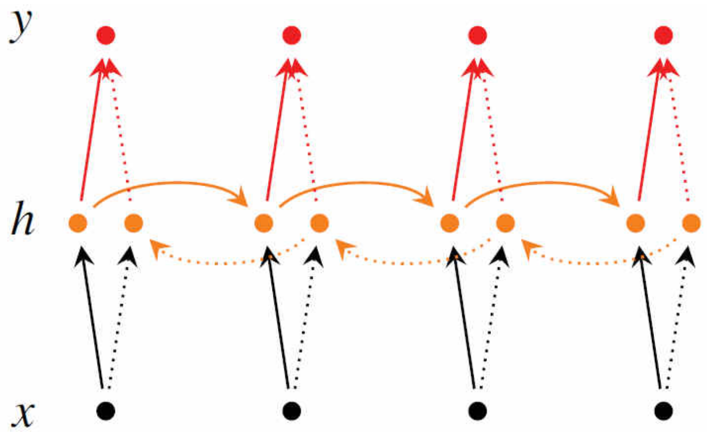
For example, for image captioning

Deep RNNs



Several RNNs stacked (like layers in MLP)

Bi-directional RNNs



$$\overrightarrow{h}_t = f(\overrightarrow{W}x_t + \overrightarrow{V}\overrightarrow{h}_{t-1} + \overrightarrow{b})$$

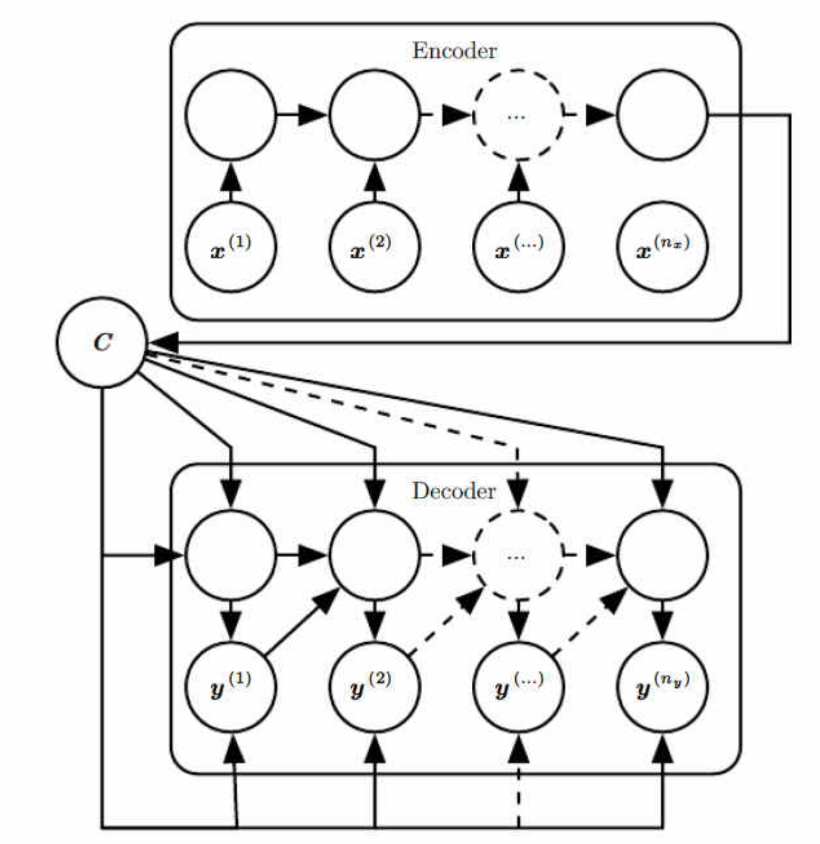
$$\overleftarrow{h}_t = f(\overleftarrow{W}x_t + \overleftarrow{V}\overleftarrow{h}_{t+1} + \overleftarrow{b})$$

$$y_t = g(U[\overrightarrow{h}_t;\overleftarrow{h}_t] + c)$$

$h = [\overrightarrow{h};\overleftarrow{h}]$ now represents (summarizes) the past and future around a single token.

(e.g. for offline classification of sequence of words)

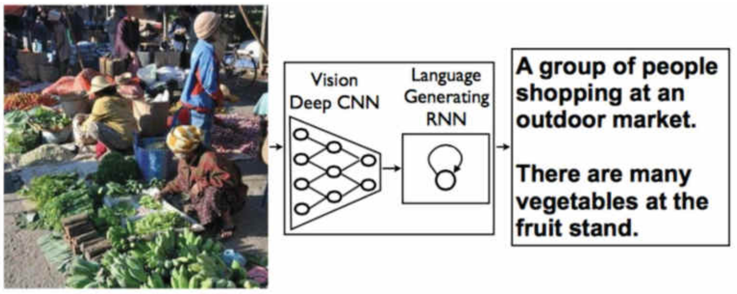
Encoder-decoder RNN



Applications of RNN/LSTM

<u>Wherever data is intrinsicly SEQUENTIAL</u>

- Speech recognition
- ***<u>Natural Language Processing (NLP)</u>***
  - Machine-Translation
  - Image caption generator



- <u>Gesture recognition</u>
- Music generation
- *Potentially any kind of time-series!!*

Summary and perspectives on Recurrent Neural Networks

- For <u>SEQUENTIAL</u> data
  (speech, text, …, gestures, …)
- Impressive results in
  Natural Language Processing (in particular Automated Real-Time Translation)
- Training of standard RNNs can be tricky
  (vanishing gradient…)
- LSTM / GRU now more used than standard RNNs

Any QUESTIONS ?