Recurrent Neural Networks (RNNs)

Michel RIVEILL
michel.riveill@univ-cotedazur.fr

Motivation

- Humans don’t start their thinking from scratch every second
  - Thoughts have persistence
- Traditional neural networks can’t characterize this phenomena
  - Ex: classify what is happening at every point in a movie
  - How a neural network can inform later events about the previous ones
- Recurrent neural networks address this issue
  - Some applications
    - POS Part of Speech Tagging
    - NER - Naming Entity Recognition
      - Same word may have a different label depending on the context.
        - Apple CEO Tim Cook eat an apple
    - Forcasting - Time-series Prediction
- How?
  - Add state to artificial neurons

What are RNNs?

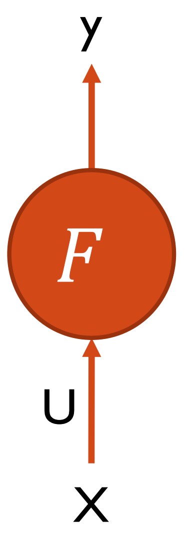
- Main idea is to make use of sequential information
- How RNN is different from neural network?
  - Vanilla neural networks (MLP) assume that all inputs and outputs are independent of each other
  - But for many tasks, that's a very bad idea
- What RNN does?
  - Perform the same task for every element of a sequence (that's what recurrent stands for)
  - Output depends on the previous computations!
- Another way of interpretation – RNNs have a "memory"
  - To store previous computations

Some applications for fun

- RNN Generated TED Talks
  - YouTube Link - https://youtu.be/-OodHtJ1saY?t=31s
  - https://medium.com/@samim/ted-rnn-machine-generated-ted-talks-3dd682b894c0
- RNN Generated Eminem rapper
  - RNN Shady - https://soundcloud.com/mrchrisjohnson/recurrent-neural-shady
- RNN Generated Music
  - Music Link - http://www.hexahedria.com/2015/08/03/composing-music-with-recurrent-neural-networks/

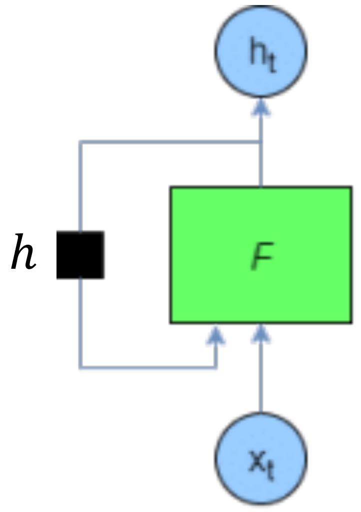
From vanilla NN to recurrent NN

- Vanilla cell
  - $y = F(U.X)$



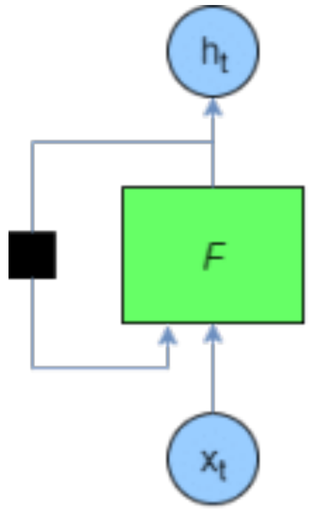
From vanilla NN to recurrent NN

- Vanilla cell
  - $y = F(U.X)$
- Recurrent cell → use 2 weights matrix
  - Add an internal variable: h
  - The output depends to the current entry and the previous internal variable:
    - $h_t = F(W.h_{t-1} + U.Xt)$
    - Could be rewritten on $h_t = F(V.[h_{t-1}, Xt])$



From vanilla NN to recurrent NN

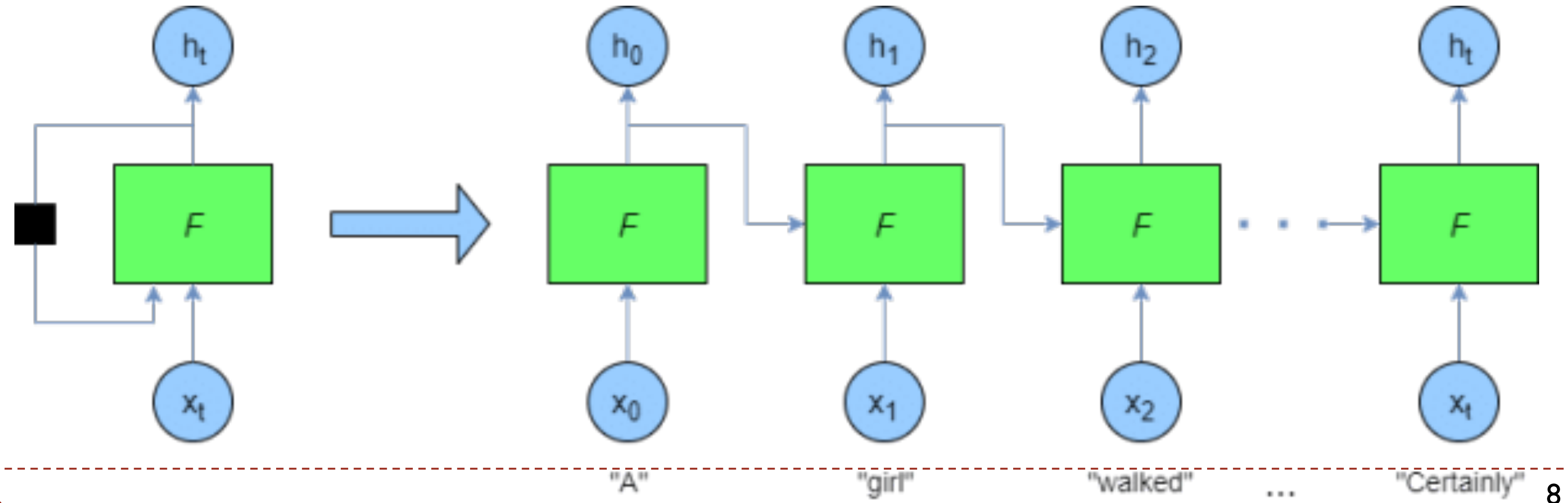
- Vanilla cell
  - $y = F(U.X)$
- Recurrent cell → use 2 weights matrix
  - $h_t = F(W.[h_{t\_1}, X_t])$
- Recurrent layer, step by step
  - At each time step
    - A new entry is being supplied
    - And a new output (h_t) is calculated using:
      - ☐ The new input X_t
      - ☐ The output of the previous step h_{t_1}



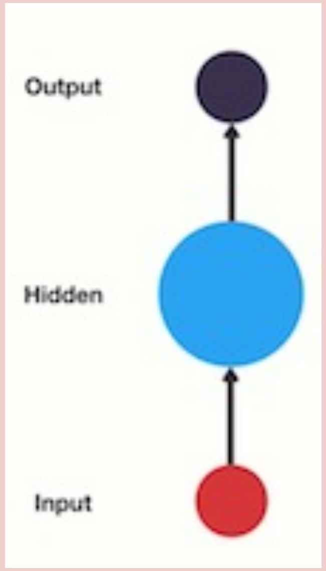
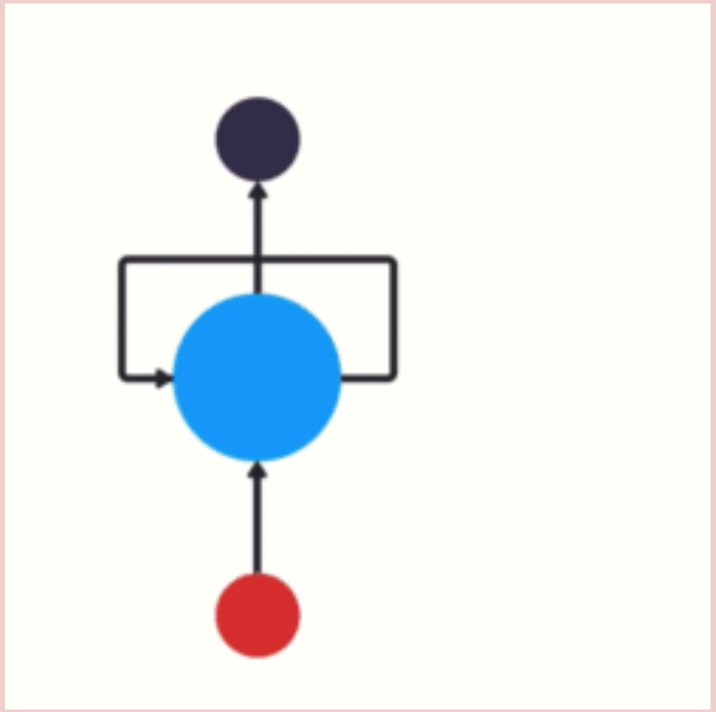
  - $h_1 = F(W.[h_0, X_1])$
  - $h_2 = F(W.h_1, X_2])$
  - $h_3 = F(W.h_2, X_3])$
  - …

From vanilla NN to recurrent NN

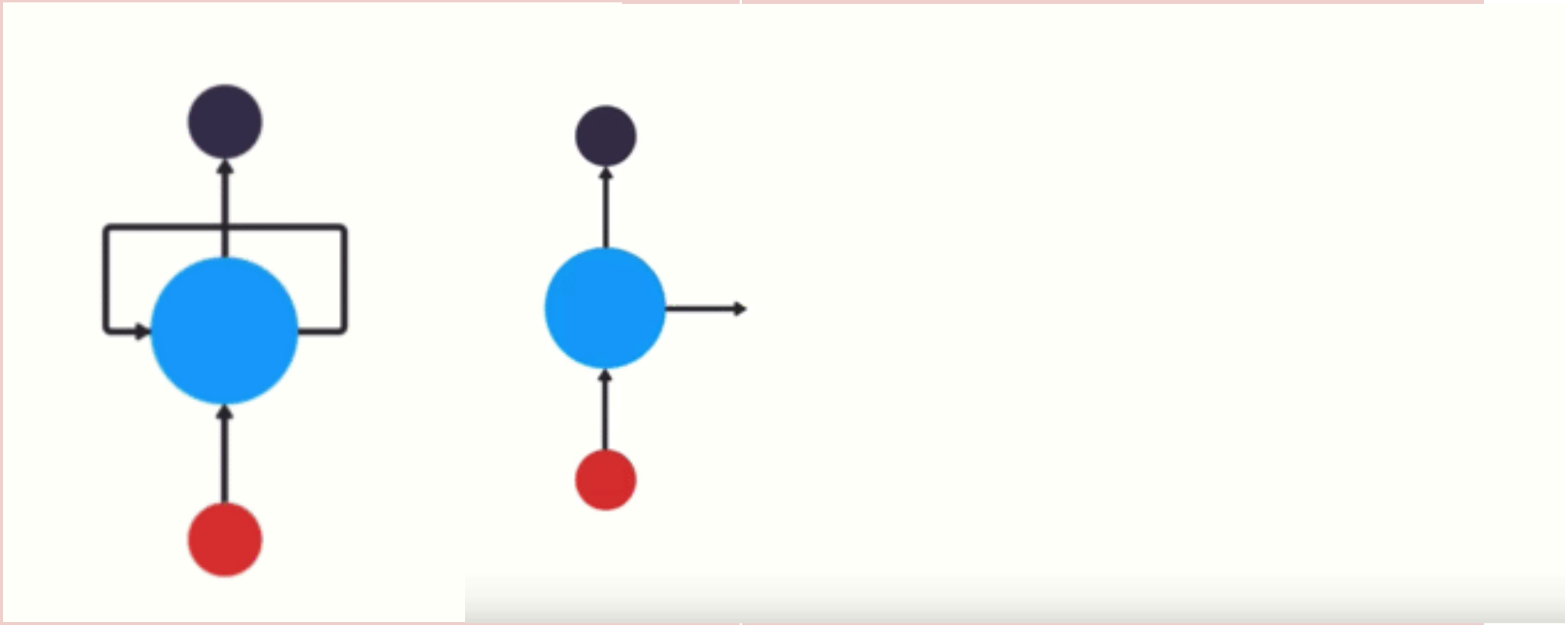
- Vanilla cell
  - $y = F(U.X)$
- Recurrent cell
  - $h_t = F(W.[h_{t-1}, X_t])$
- Recurrent neural networks are "unrolled" programmatically during training and prediction
  - All neurons share the same weight matrix



Remember

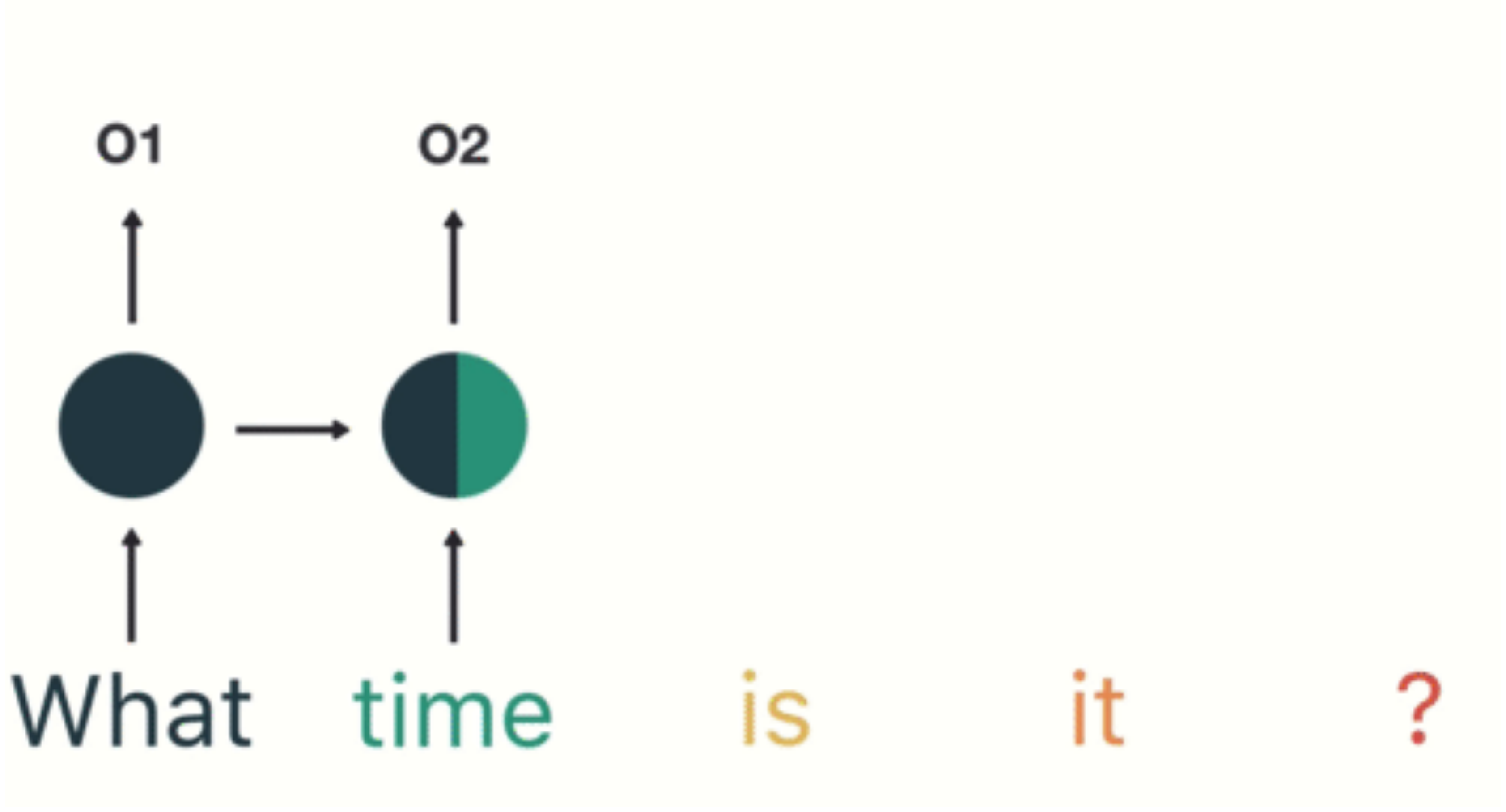
From	To
	

Remember

From	To
	

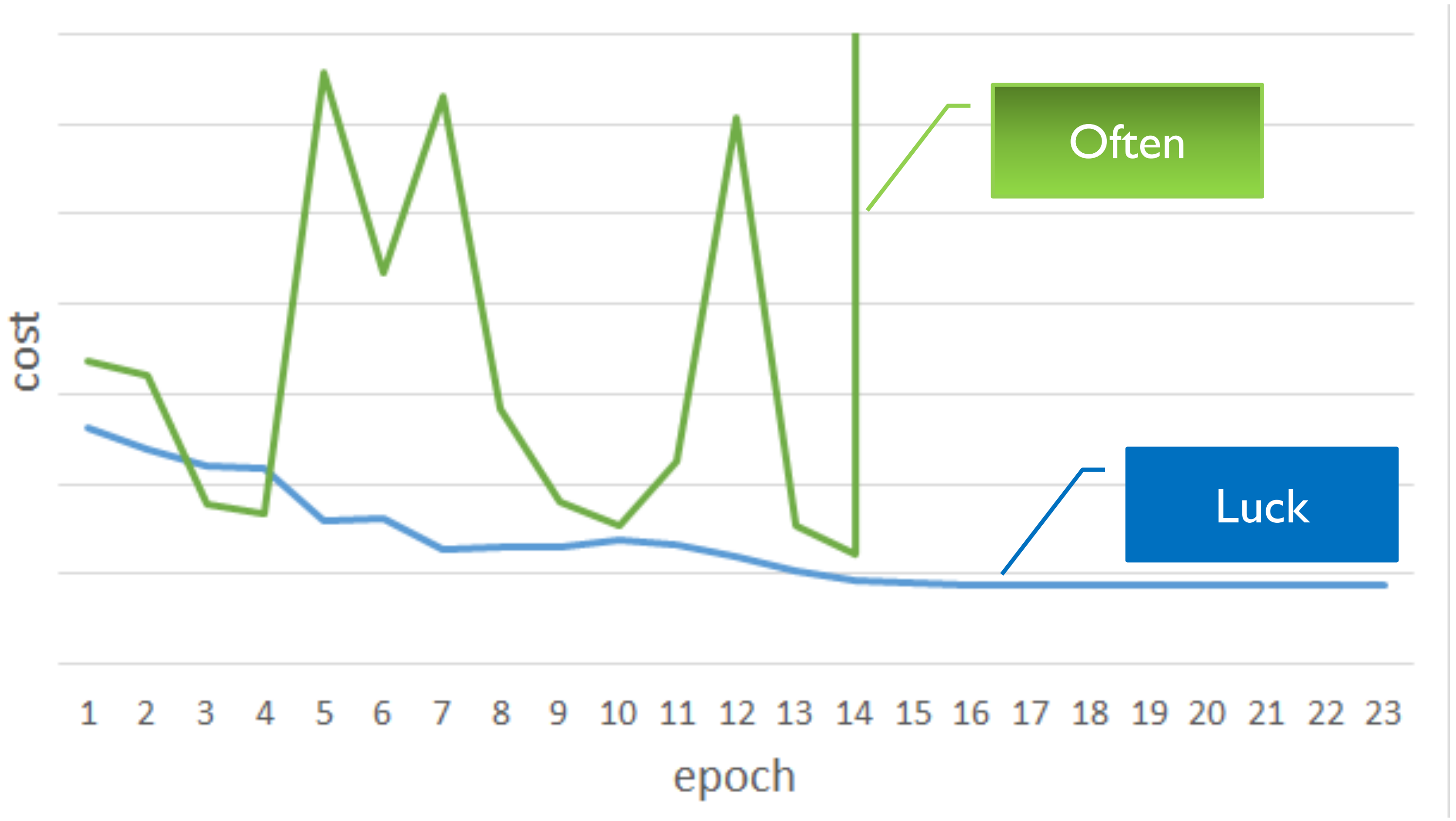
RNN in action

At each time stamp, we memorise: the previous state, a little of the state preceding the previous one, a little of the state preceding the previous one, etc.



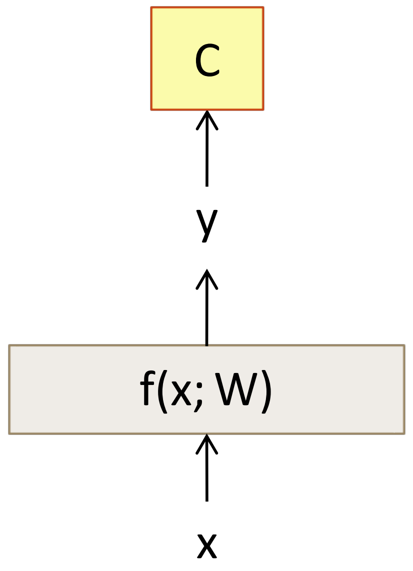
Problems with naive RNN

- RNNs do not learn easily
- Unfolding the network for learning leads to vanishing gradient problems!



Learning process
Vanishing gradient problem

BackPropagation Refresher



$$y = f(x;W)$$

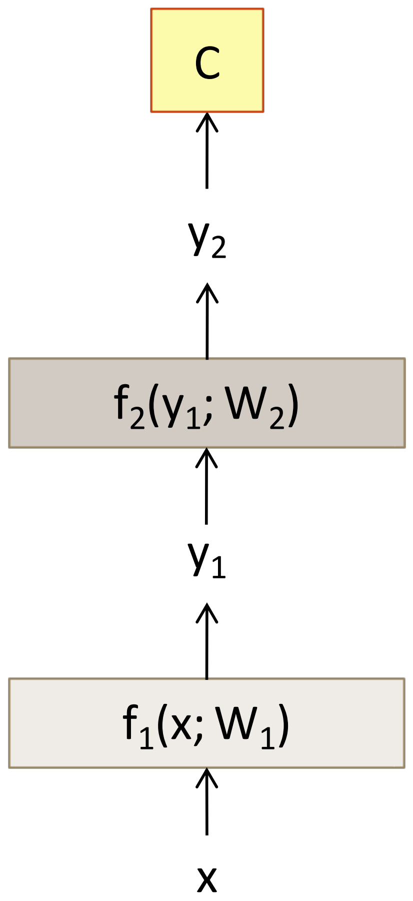
$$C = \mathrm{Loss}(y, y_{GT})$$

SGD Update

$$W \leftarrow W - \eta \frac{\partial C}{\partial W}$$

$$\frac{\partial C}{\partial W} = \left(\frac{\partial C}{\partial y}\right)\left(\frac{\partial y}{\partial W}\right)$$

Multiple Layers



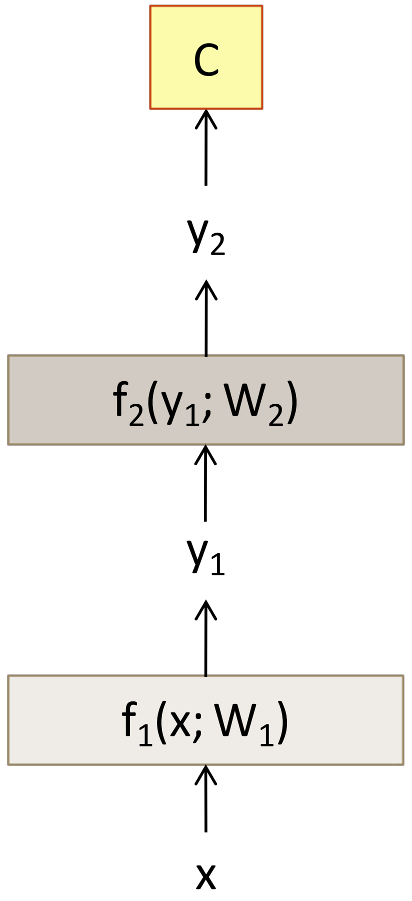
$$y_1 = f_1(x; W_1)$$
$$y_2 = f_2(y_1; W_2)$$
$$C = \text{Loss}(y_2, y_{GT})$$

SGD Update

$$W_2 \leftarrow W_2 - \eta \frac{\partial C}{\partial W_2}$$

$$W_1 \leftarrow W_1 - \eta \frac{\partial C}{\partial W_1}$$

Chain Rule for Gradient Computation



$$y_1 = f_1(x; W_1)$$

$$y_2 = f_2(y_1; W_2)$$

$$C = \text{Loss}(y_2, y_{GT})$$

$$\text{Find } \frac{\partial C}{\partial W_1}, \frac{\partial C}{\partial W_2}$$

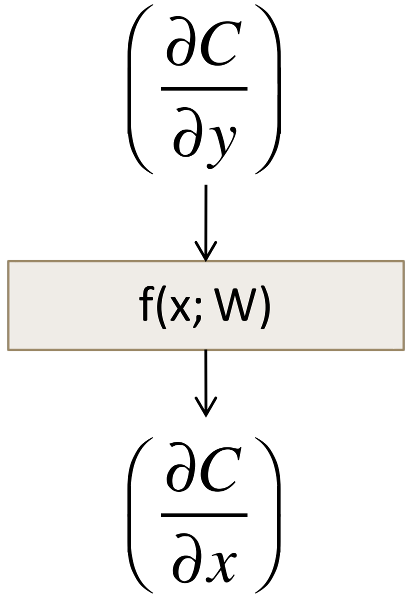
$$\frac{\partial C}{\partial W_2} = \left(\frac{\partial C}{\partial y_2}\right)\left(\frac{\partial y_2}{\partial W_2}\right)$$

$$\frac{\partial C}{\partial W_1} = \left(\frac{\partial C}{\partial y_1}\right)\left(\frac{\partial y_1}{\partial W_1}\right)$$

$$= \left(\frac{\partial C}{\partial y_2}\right)\left(\frac{\partial y_2}{\partial y_1}\right)\left(\frac{\partial y_1}{\partial W_1}\right)$$

Application of the Chain Rule

Chain Rule for Gradient Computation


$\left(\frac{\partial C}{\partial y}\right)$
f(x; W)
$\left(\frac{\partial C}{\partial x}\right)$

Given: $\left(\frac{\partial C}{\partial y}\right)$

We are interested in computing: $\left(\frac{\partial C}{\partial W}\right), \left(\frac{\partial C}{\partial x}\right)$

Intrinsic to the layer are:

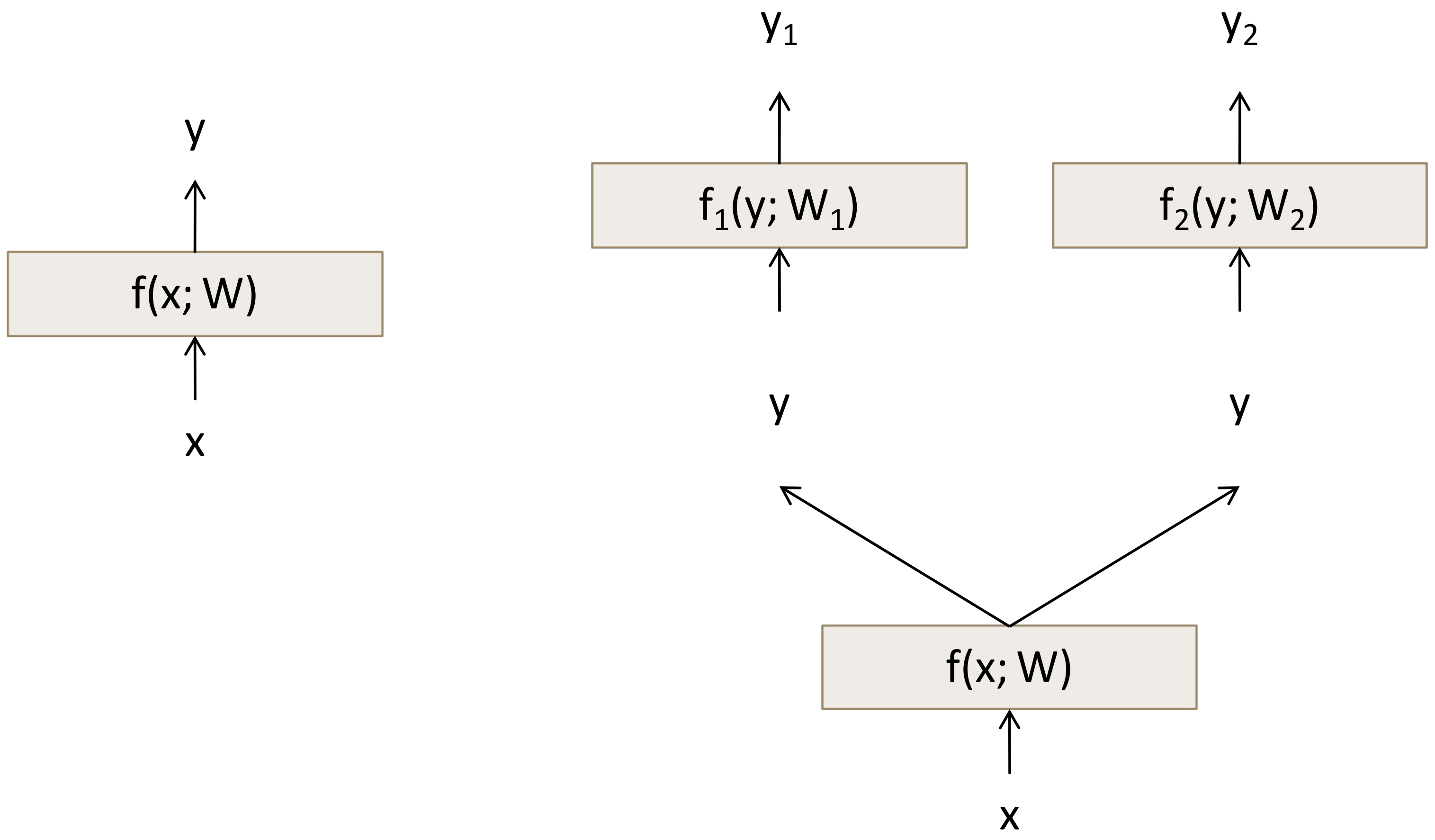
$\left(\frac{\partial y}{\partial W}\right)$ – How does output change due to params

$\left(\frac{\partial y}{\partial x}\right)$ – How does output change due to inputs

$$\left(\frac{\partial C}{\partial W}\right) = \left(\frac{\partial C}{\partial y}\right)\left(\frac{\partial y}{\partial W}\right) \qquad \left(\frac{\partial C}{\partial x}\right) = \left(\frac{\partial C}{\partial y}\right)\left(\frac{\partial y}{\partial x}\right)$$

Equations for common layers: http://arunmallya.github.io/writeups/nn/backprop.html

Extension to Computational Graphs

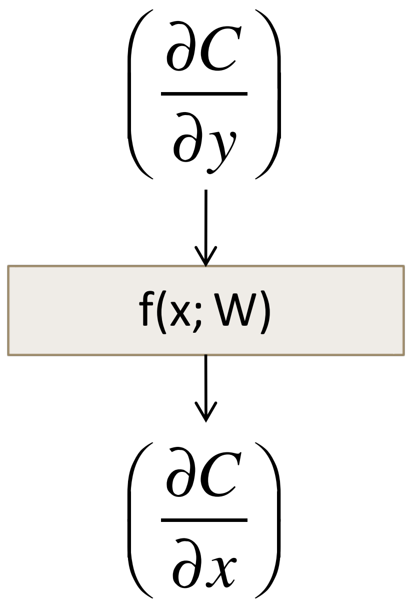


Extension to Computational Graphs

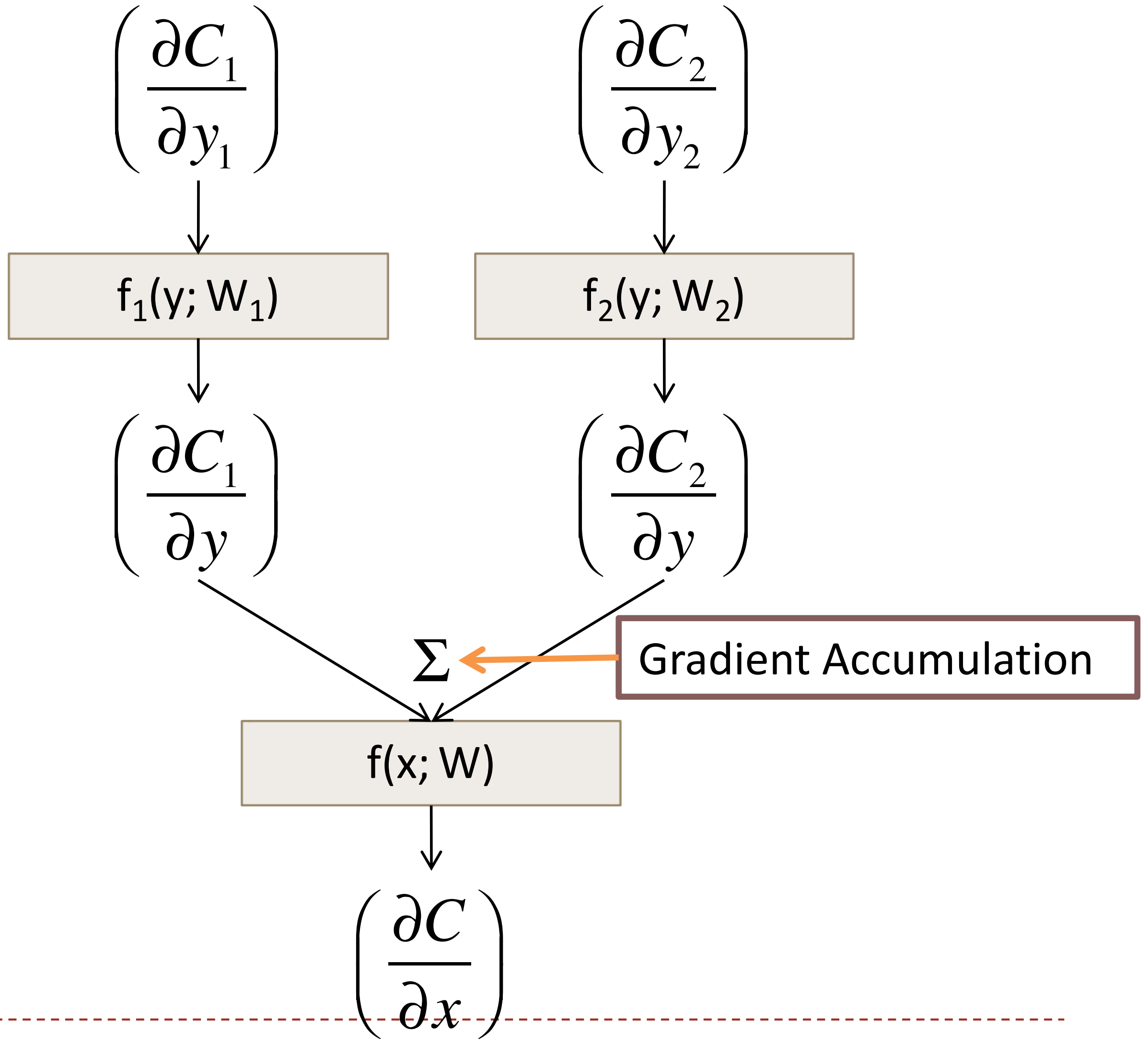
$$\begin{tikzcd}
\left(\frac{\partial C}{\partial y}\right) \arrow[d] \\
\boxed{f(x; W)} \arrow[d] \\
\left(\frac{\partial C}{\partial x}\right)
\end{tikzcd}$$

$$\begin{tikzcd}
\left(\frac{\partial C_1}{\partial y_1}\right) \arrow[d] & & \left(\frac{\partial C_2}{\partial y_2}\right) \arrow[d] \\
\boxed{f_1(y; W_1)} \arrow[d] & & \boxed{f_2(y; W_2)} \arrow[d] \\
\left(\frac{\partial C_1}{\partial y}\right) \arrow[dr] & & \left(\frac{\partial C_2}{\partial y}\right) \arrow[dl, "\Sigma"'] \\
& \boxed{f(x; W)} \arrow[d] & \\
& \left(\frac{\partial C}{\partial x}\right) &
\end{tikzcd}$$

Extension to Computational Graphs



$\left(\frac{\partial C}{\partial y}\right)$
f(x; W)
$\left(\frac{\partial C}{\partial x}\right)$



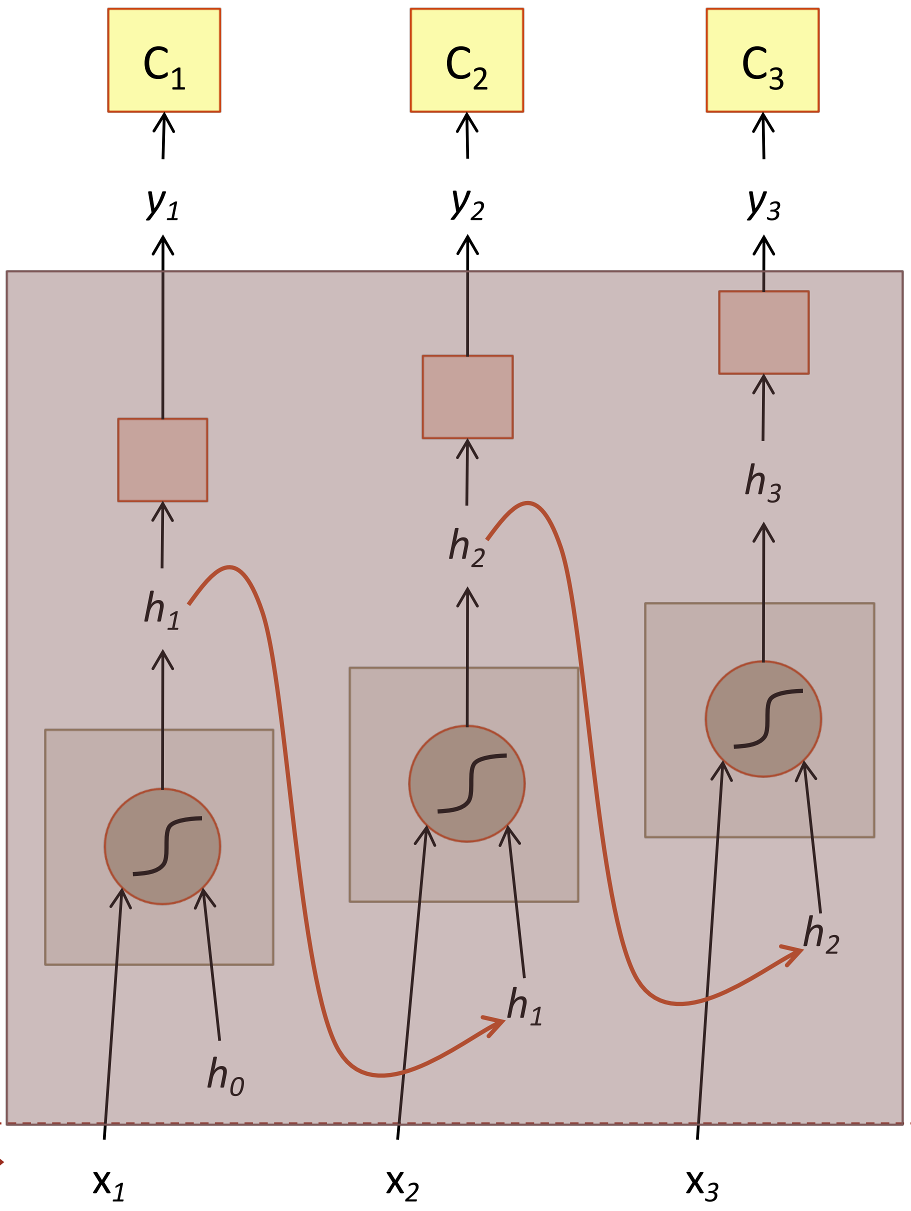
$\left(\frac{\partial C_1}{\partial y_1}\right)$
$\left(\frac{\partial C_2}{\partial y_2}\right)$
$f_1(y; W_1)$
$f_2(y; W_2)$
$\left(\frac{\partial C_1}{\partial y}\right)$
$\left(\frac{\partial C_2}{\partial y}\right)$
Σ
Gradient Accumulation
f(x; W)
$\left(\frac{\partial C}{\partial x}\right)$

BackPropagation Through Time (BPTT)

- One of the methods used to train RNNs
- The unfolded network (used during forward pass) is treated as one big feed-forward network
- This unfolded network accepts the whole time series as input

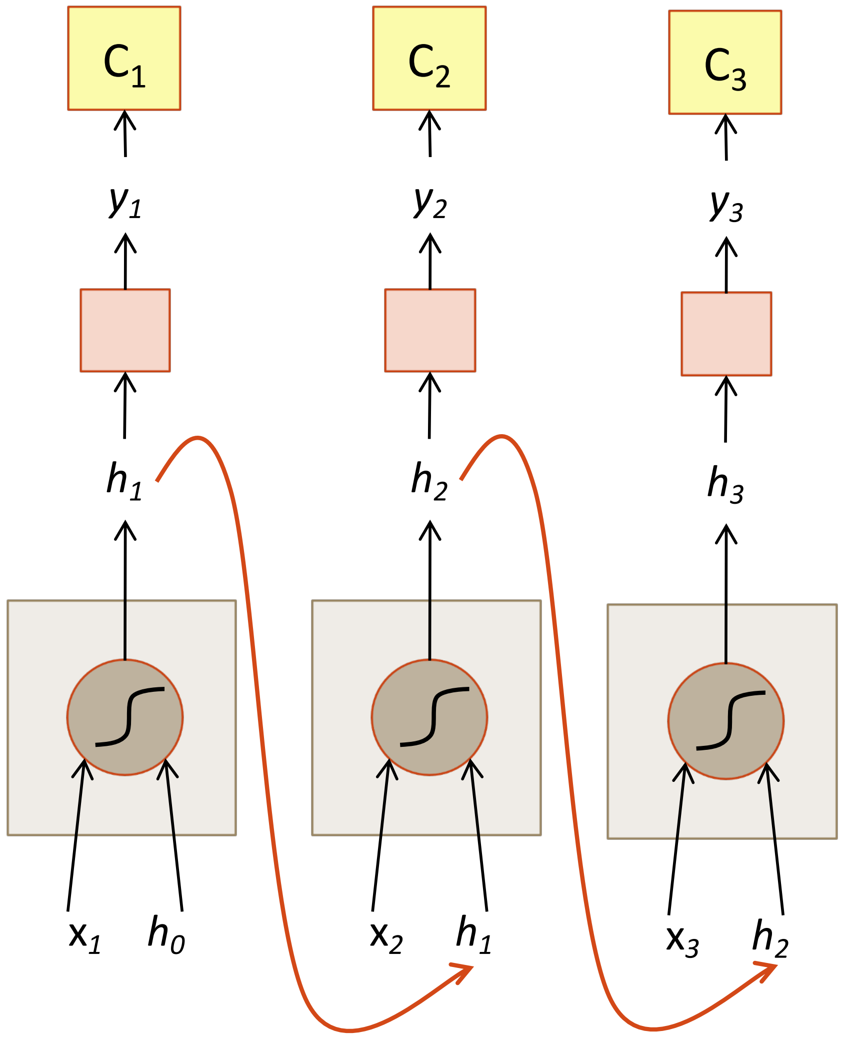
- The weight updates are computed for each copy in the unfolded network, then summed (or averaged) and then applied to the RNN weights

The Unfolded Vanilla RNN

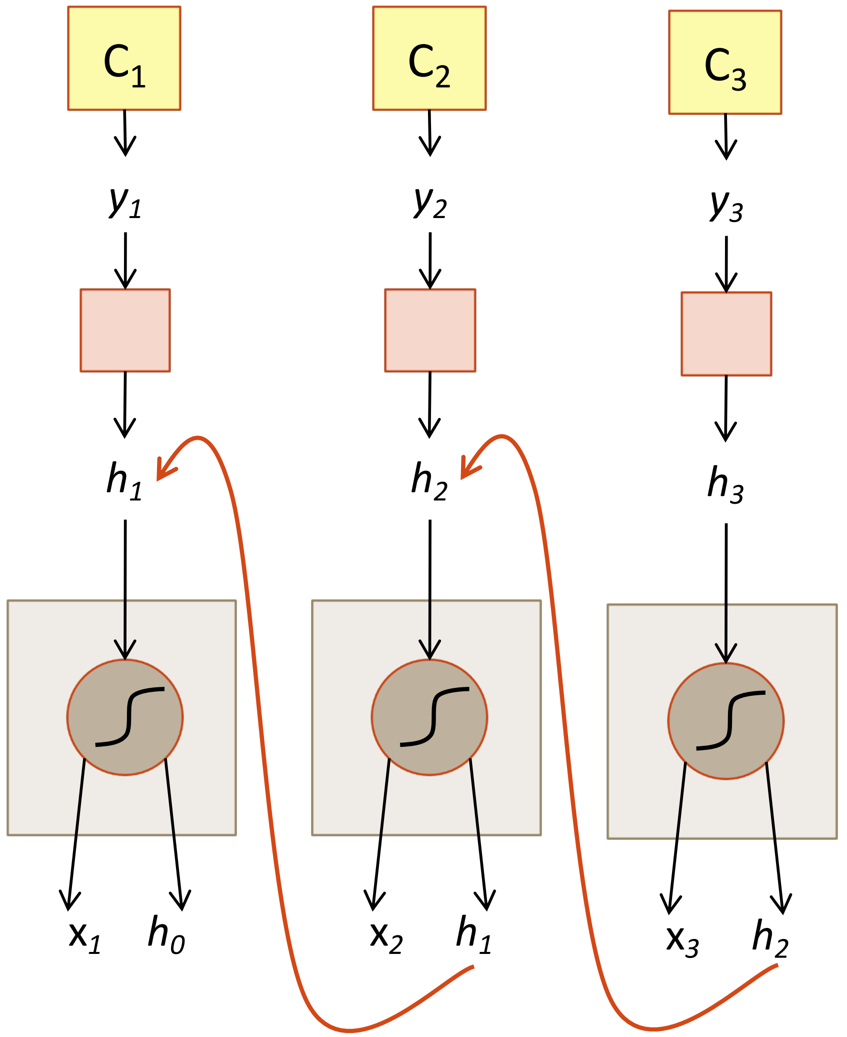


- Treat the unfolded network as one big feed-forward network!
- This big network takes in entire sequence as an input
- Compute gradients through the usual backpropagation
- Update shared weights

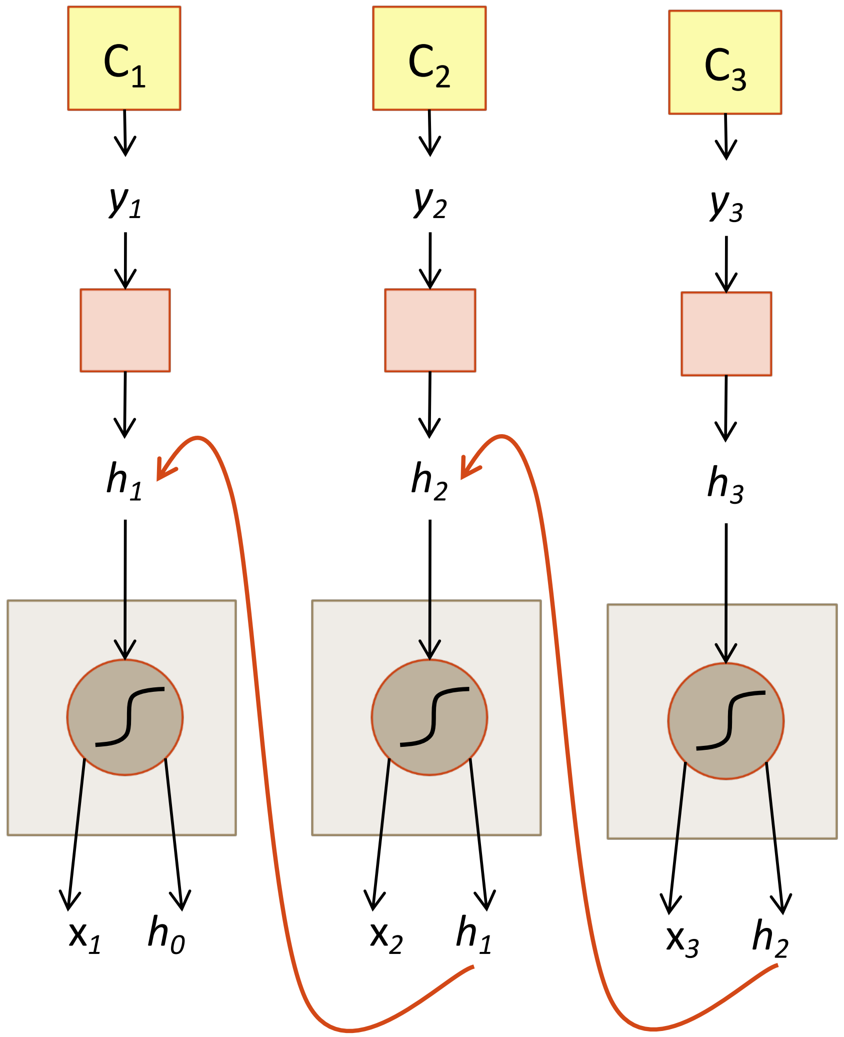
The Unfolded Vanilla RNN Forward



The Unfolded Vanilla RNN Backward



The Vanilla RNN Backward



$$h_t = \tanh W \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix}$$

$$y_t = \mathrm{F}(h_t)$$

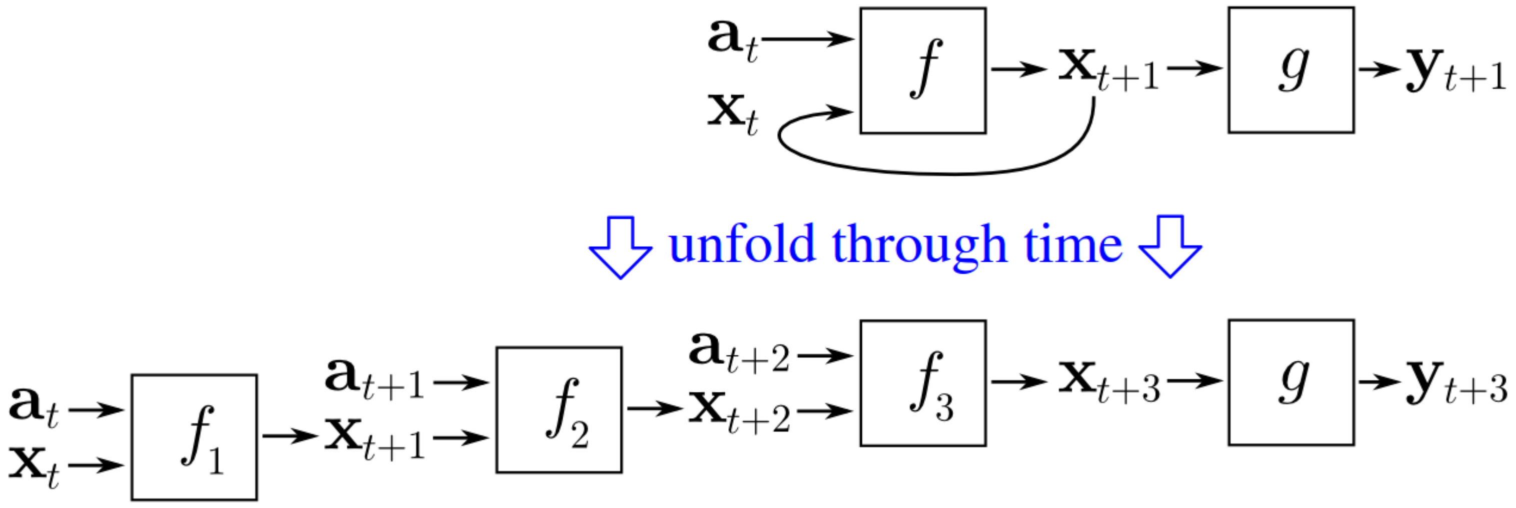
$$C_t = \mathrm{Loss}(y_t, \mathrm{GT}_t)$$

$$\frac{\partial C_t}{\partial h_1} = \left(\frac{\partial C_t}{\partial y_t}\right)\left(\frac{\partial y_t}{\partial h_1}\right)$$

$$= \left(\frac{\partial C_t}{\partial y_t}\right)\left(\frac{\partial y_t}{\partial h_t}\right)\left(\frac{\partial h_t}{\partial h_{t-1}}\right)\cdots\left(\frac{\partial h_2}{\partial h_1}\right)$$

Issues with the Vanilla RNNs

- a product of k real numbers
  - can shrink to zero : $(0,75)^n$ → n=2, 0,56 → n=10, 0,05 →n=100, 3.10^{-13}
  - or explode to infinity : $(1,25)^n$ → n=2, 1,6 → n=10, 57 →n=100, 5.10^{10}
- It's the same for a product of matrices
- Like in very deep network, Vanilla RNNs suffering from vanishing gradient



- Exploding gradients are often controlled with gradient element-wise or norm clipping

[1] On the difficulty of training recurrent neural networks, Pascanu *et al.*, 2013

The Identity Relationship

- Recall $$\frac{\partial C_t}{\partial h_1} = \left(\frac{\partial C_t}{\partial y_t}\right)\left(\frac{\partial y_t}{\partial h_1}\right)$$ $$= \left(\frac{\partial C_t}{\partial y_t}\right)\left(\frac{\partial y_t}{\partial h_t}\right)\left(\frac{\partial h_t}{\partial h_{t-1}}\right)\cdots\left(\frac{\partial h_2}{\partial h_1}\right)$$

$$h_t = \tanh W \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix}$$

$$y_t = \mathrm{F}(h_t)$$

$$C_t = \mathrm{Loss}(y_t, \mathrm{GT}_t)$$

- Suppose that instead of a matrix multiplication, we had an identity relationship between the hidden states

$$h_t = h_{t-1} + F(x_t)$$

$$\Rightarrow \left(\frac{\partial h_t}{\partial h_{t-1}}\right) = 1$$

  - The gradient does not decay as the error is propagated all the way back aka “Constant Error Flow”

The Identity Relationship

- Recall $\frac{\partial C_t}{\partial h_1} = \left(\frac{\partial C_t}{\partial y_t}\right)\left(\frac{\partial y_t}{\partial h_1}\right)$

$$= \left(\frac{\partial C_t}{\partial y_t}\right)\left(\frac{\partial y_t}{\partial h_t}\right)\left(\frac{\partial h_t}{\partial h_{t-1}}\right)\cdots\left(\frac{\partial h_2}{\partial h_1}\right)$$

$$h_t = \tanh W\begin{pmatrix} x_t \\ h_{t-1}\end{pmatrix}$$

$$y_t = \mathrm{F}(h_t)$$

$$C_t = \mathrm{Loss}(y_t, \mathrm{GT}_t)$$

- Suppose that instead of a matrix multiplication, we had an identity relationship between the hidden states

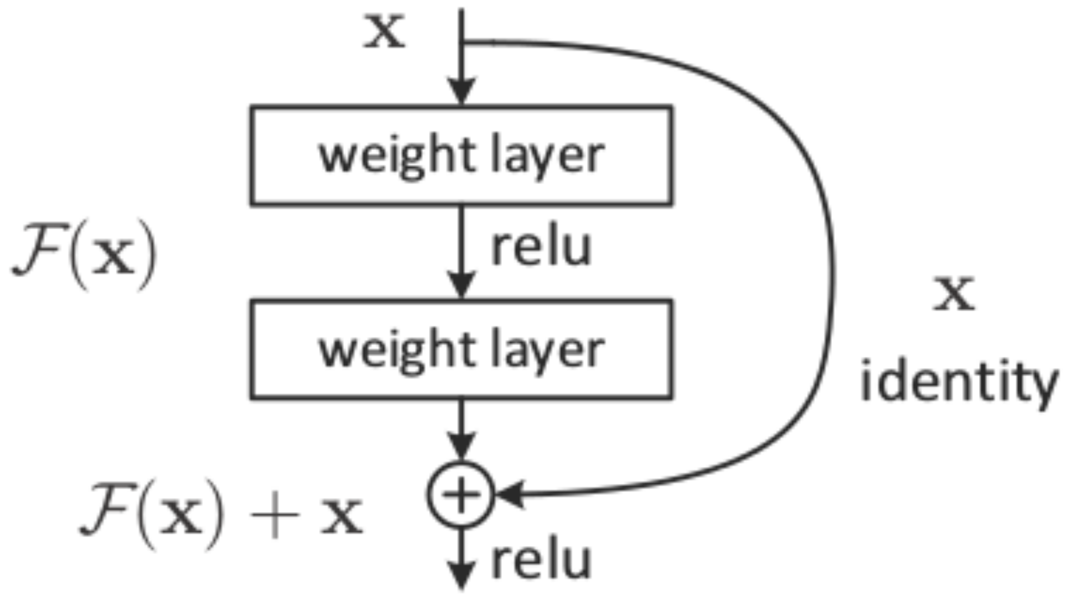
$$h_t = h_{t-1} + F(x_t)$$

Remember Resnets?

$$\Rightarrow \left(\frac{\partial h_t}{\partial h_{t-1}}\right) = 1$$

- The gradient does not decay as the error is propagated all the way back aka "Constant Error Flow"
  - Like in residual network

Residual ?


$\mathbf{x}$
weight layer
$\mathcal{F}(\mathbf{x})$
relu
weight layer
$\mathbf{x}$
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

Disclaimer

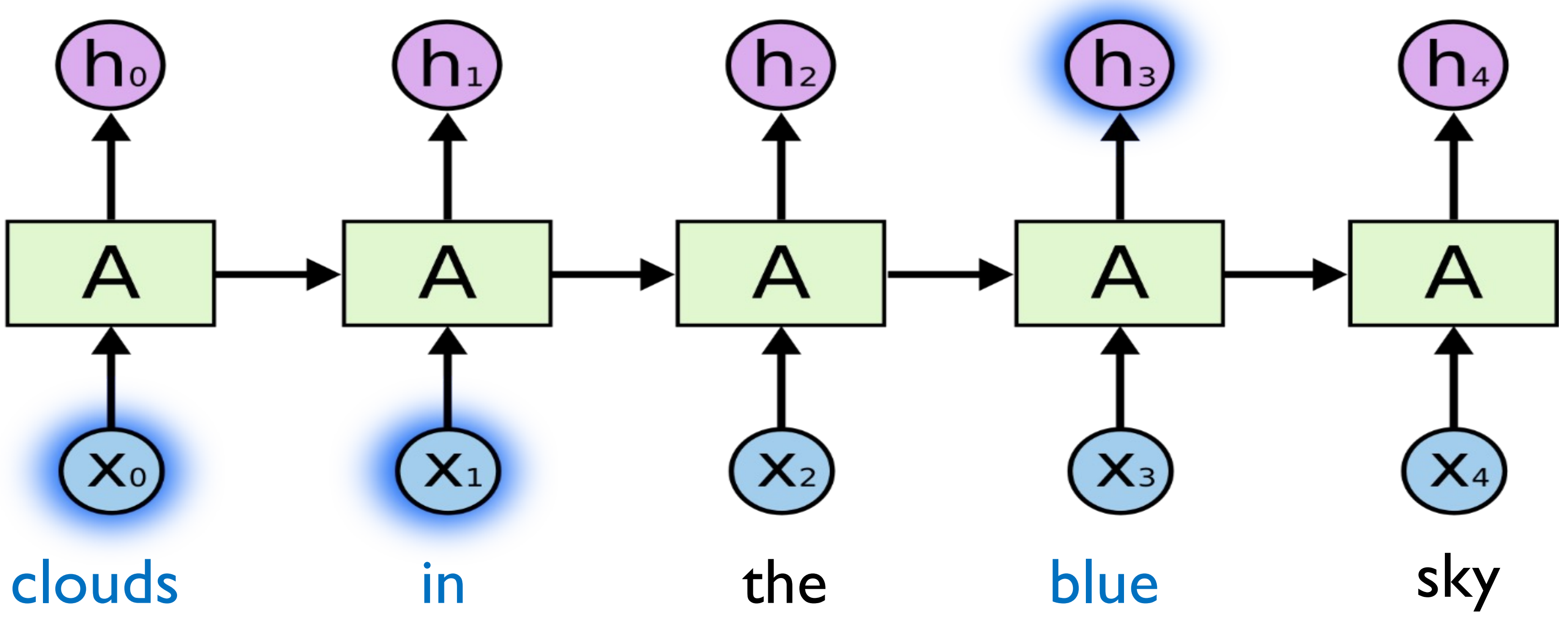
- The explanations in the previous few slides are handwavy

- For rigorous proofs and derivations, please refer to
  - On the difficulty of training recurrent neural networks, Pascanu *et al.*, 2013
  - Long Short-Term Memory, Hochreiter *et al.*, 1997
  - And other sources

- RNN: problem 1 → prevent vanishing gradient

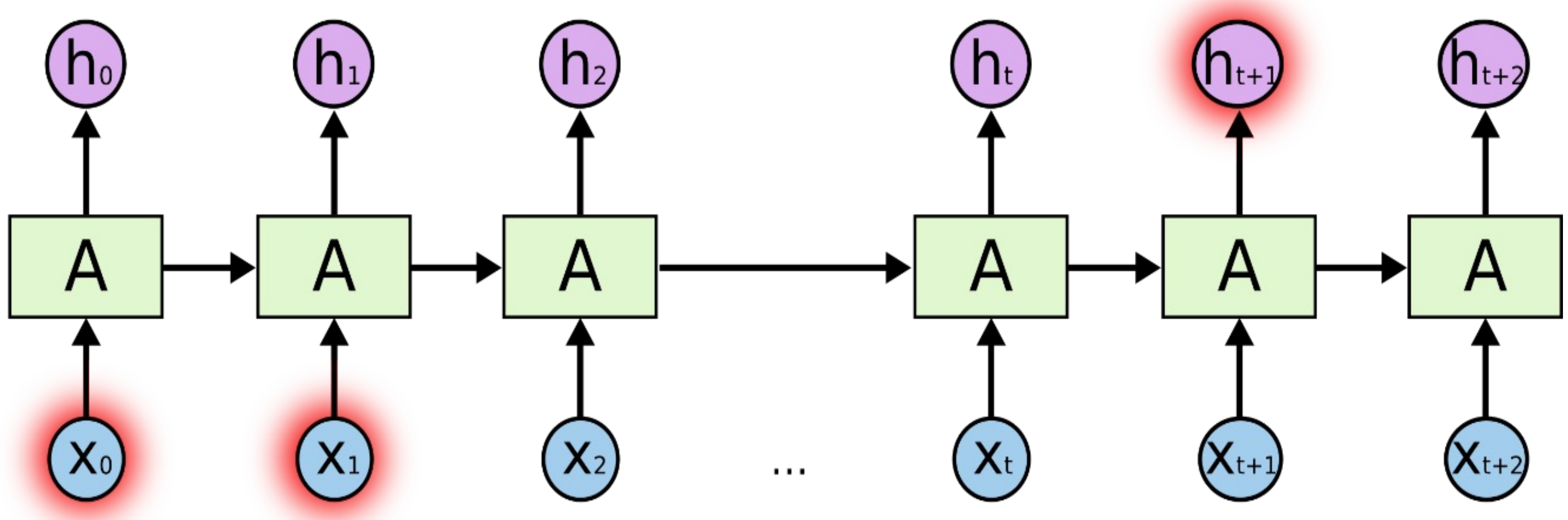
From vanilla RNN…

- The context is close to the word to be predicted
  - Few iterations separate them.
  - No problem



… to LSTM (Long Short Term Memory)

- The context is far from the word to predict
  - Many iterations separate them!
  - Pb1: Possible gradient problem
  - Pb2: Control how much of the cell's state should be retained.

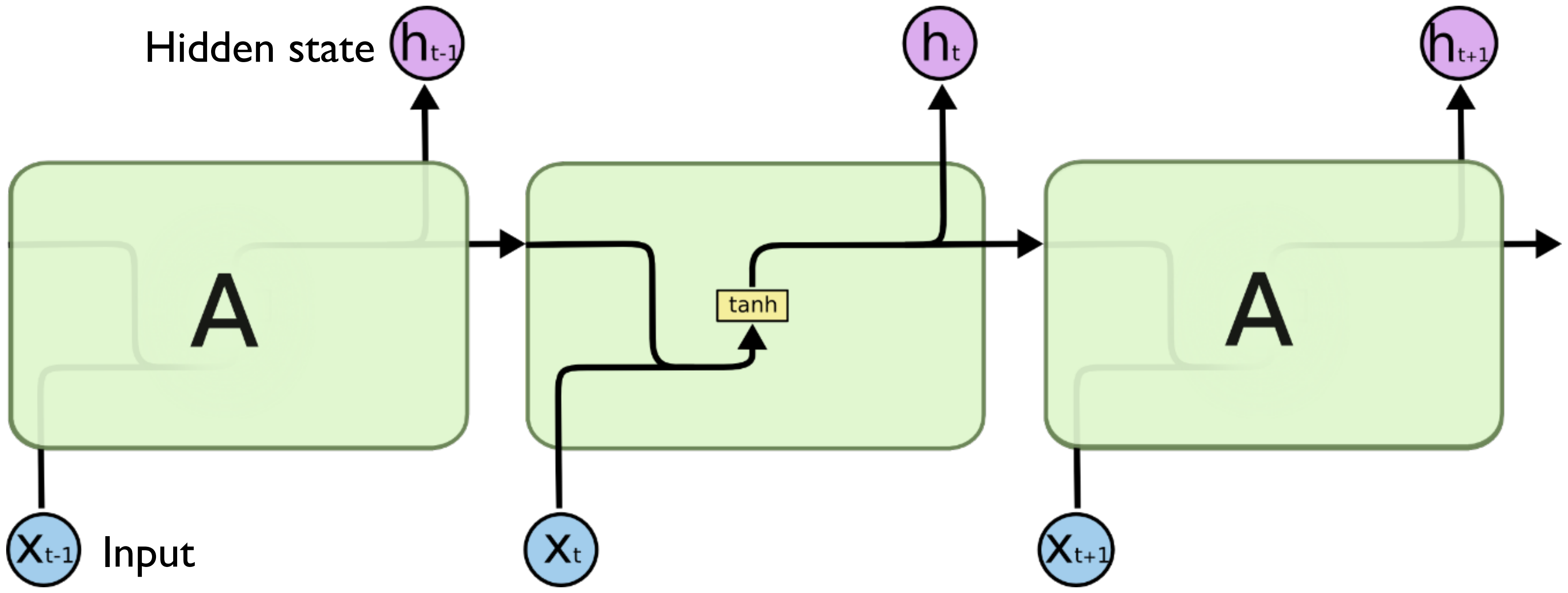


I grew up in France…

I speak French…

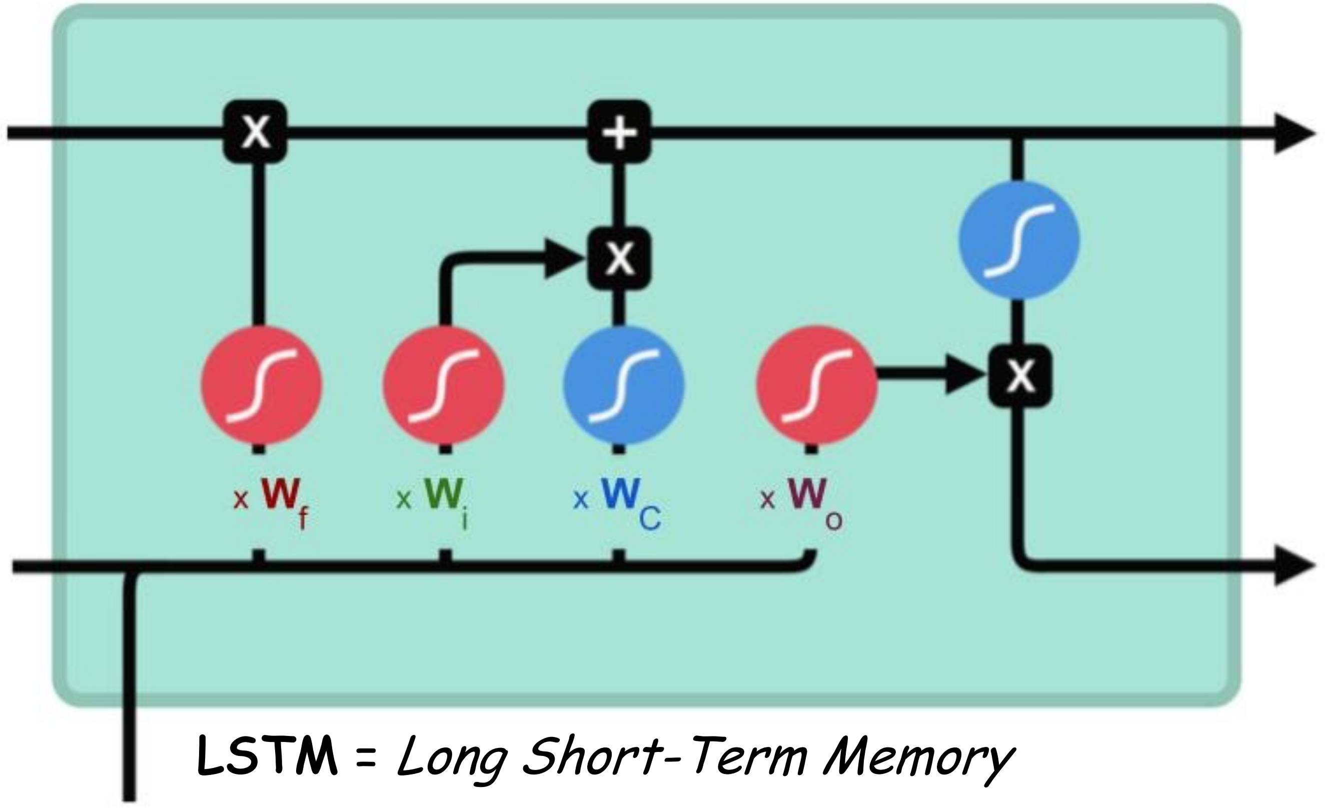
From Vanilla to LSTM Cells

- It is necessary to prevent the gradient from disappearing...
- Normally, the network memory is
  - $h_t = tanh(W \cdot [h_{t-1}, xt])$
  - Involves a single level of processing
  - Creating the risk of the evanescent gradient.



From Vanilla to LSTM cell

Pb1: Possible gradient problem
Pb2: Control how much of the cell's state should be retained.

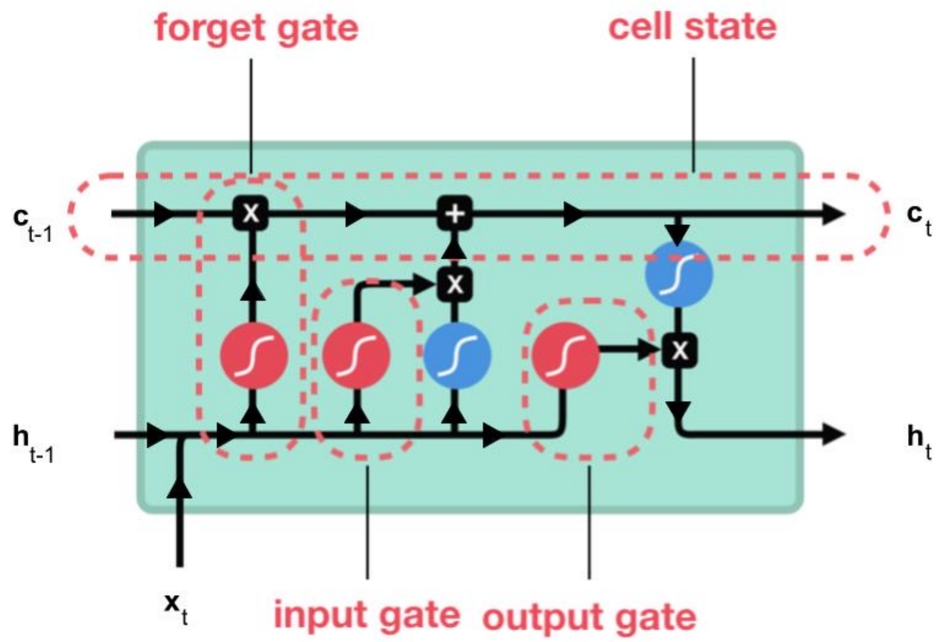


LSTM = *Long Short-Term Memory*

(crédit : image modifiée de Michaël Nguyen)

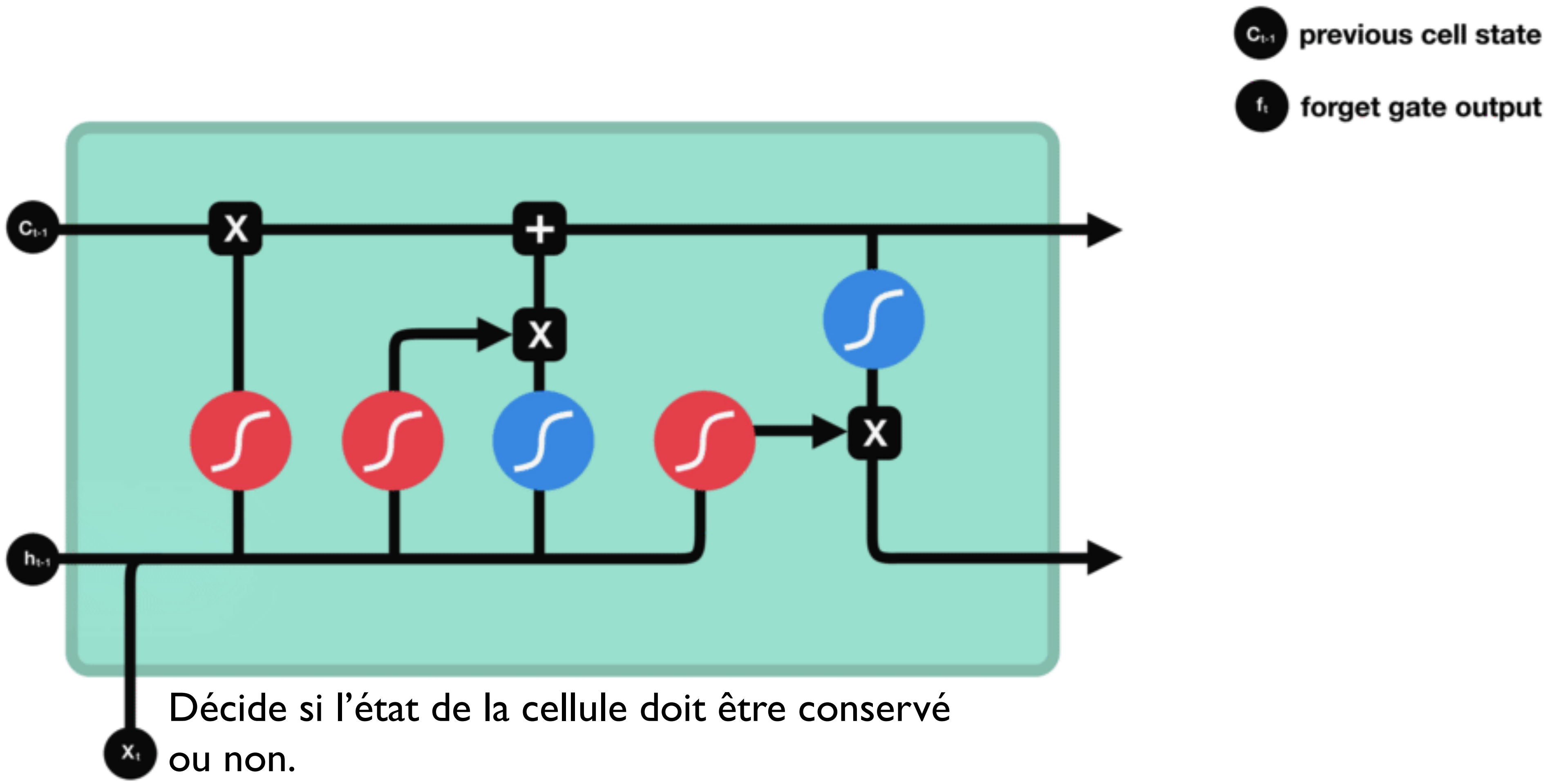
LSTM cell

- Cell composed of three "gates": these are calculation areas that regulate the flow of information (by carrying out specific actions)
  - Forget gate (porte d'oubli)
  - Input gate (porte d'entrée)
  - Output gate (porte de sortie)
- Hidden state (état caché)
- Cell state (état de la cellule)
  - Like residual



(crédit : image modifiée de Michaël Nguyen)

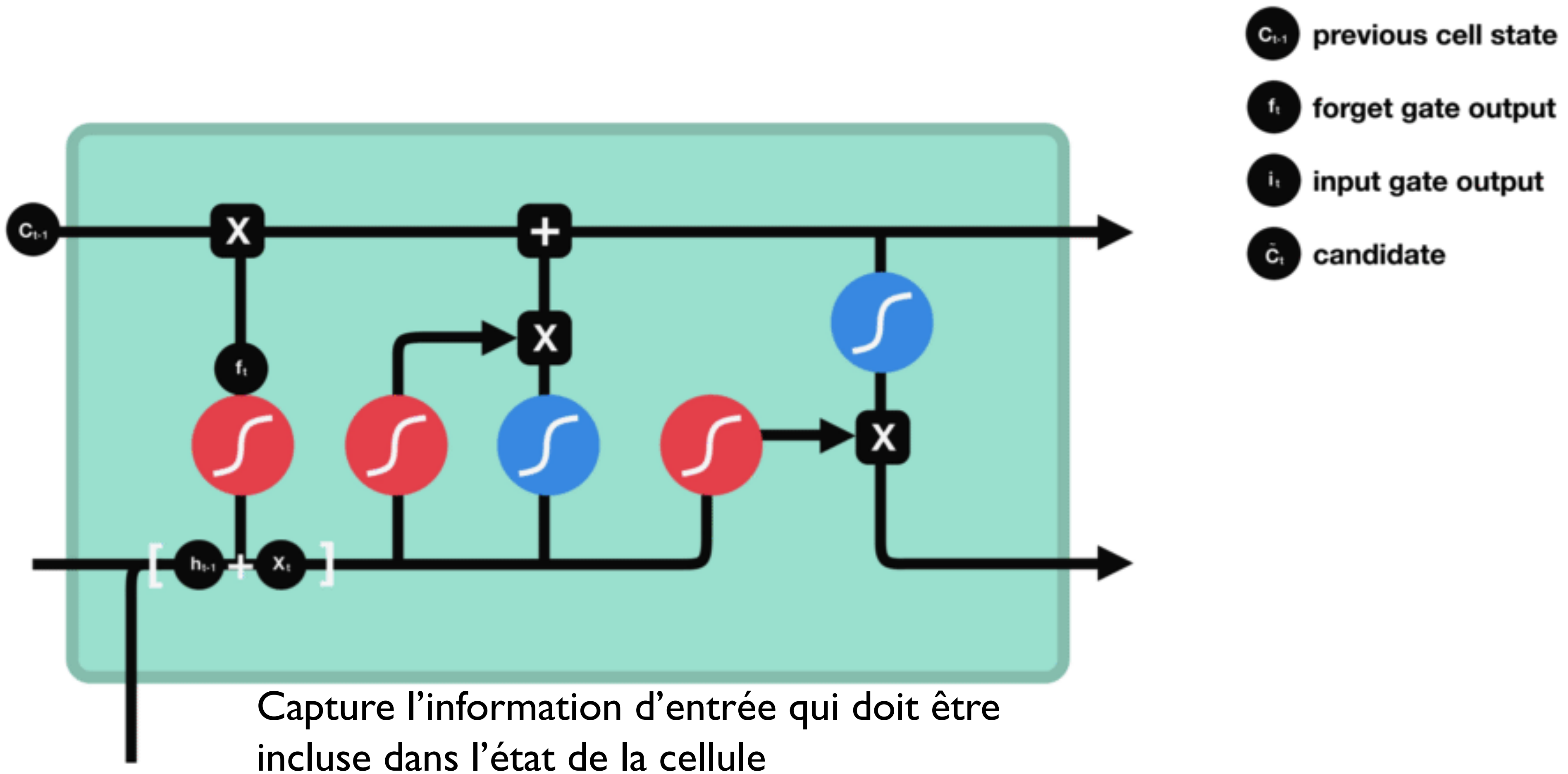
LSTM cell (porte oubli / forget get)



Décide si l'état de la cellule doit être conservé ou non.

Decides whether the state of the cell should be retained or not.

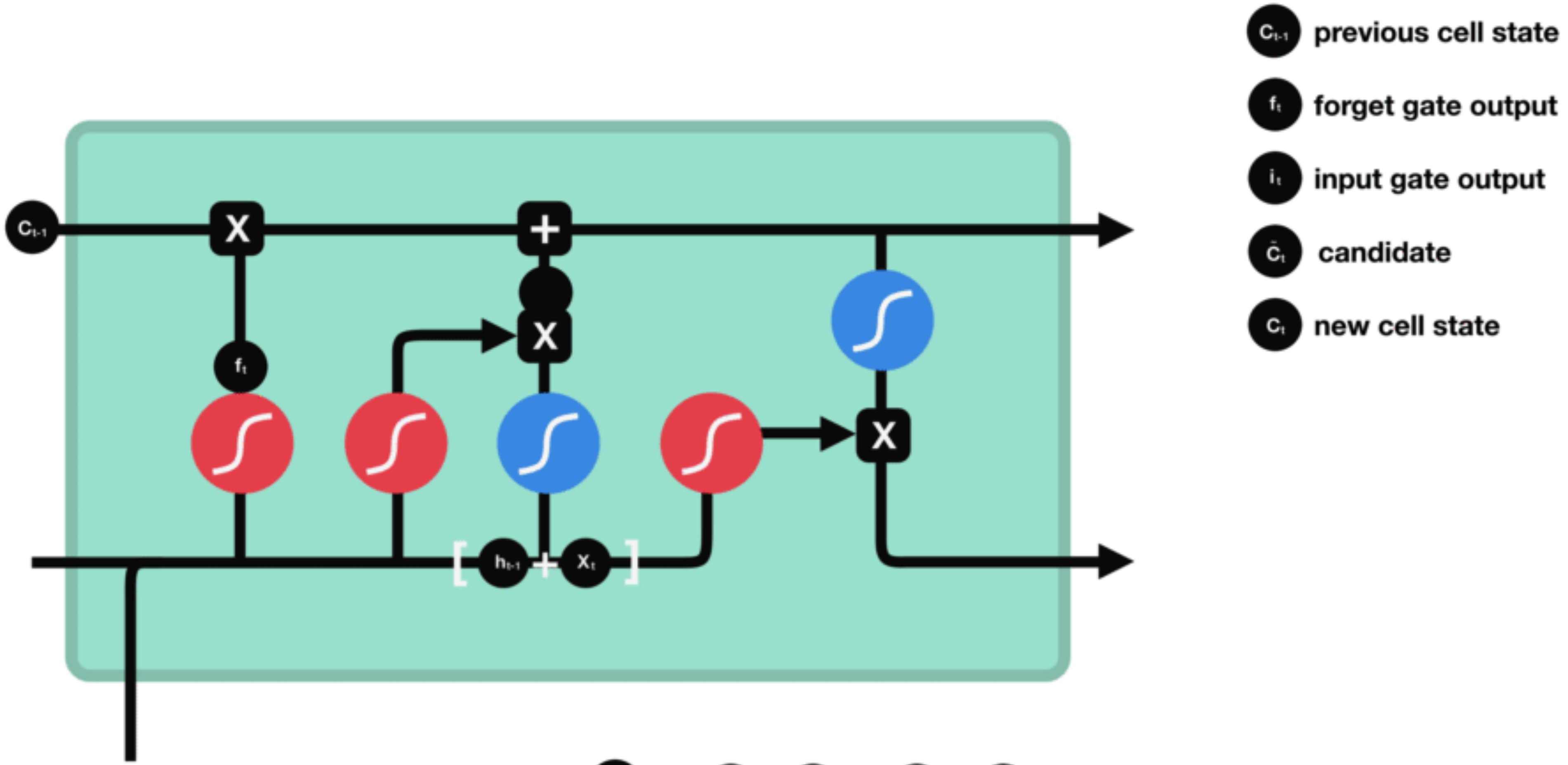
LSTM cell (porte entrée / input get)



Capture l'information d'entrée qui doit être incluse dans l'état de la cellule

Captures the input information to be included in the cell state

LSTM cell (état de la cellule / cell state)

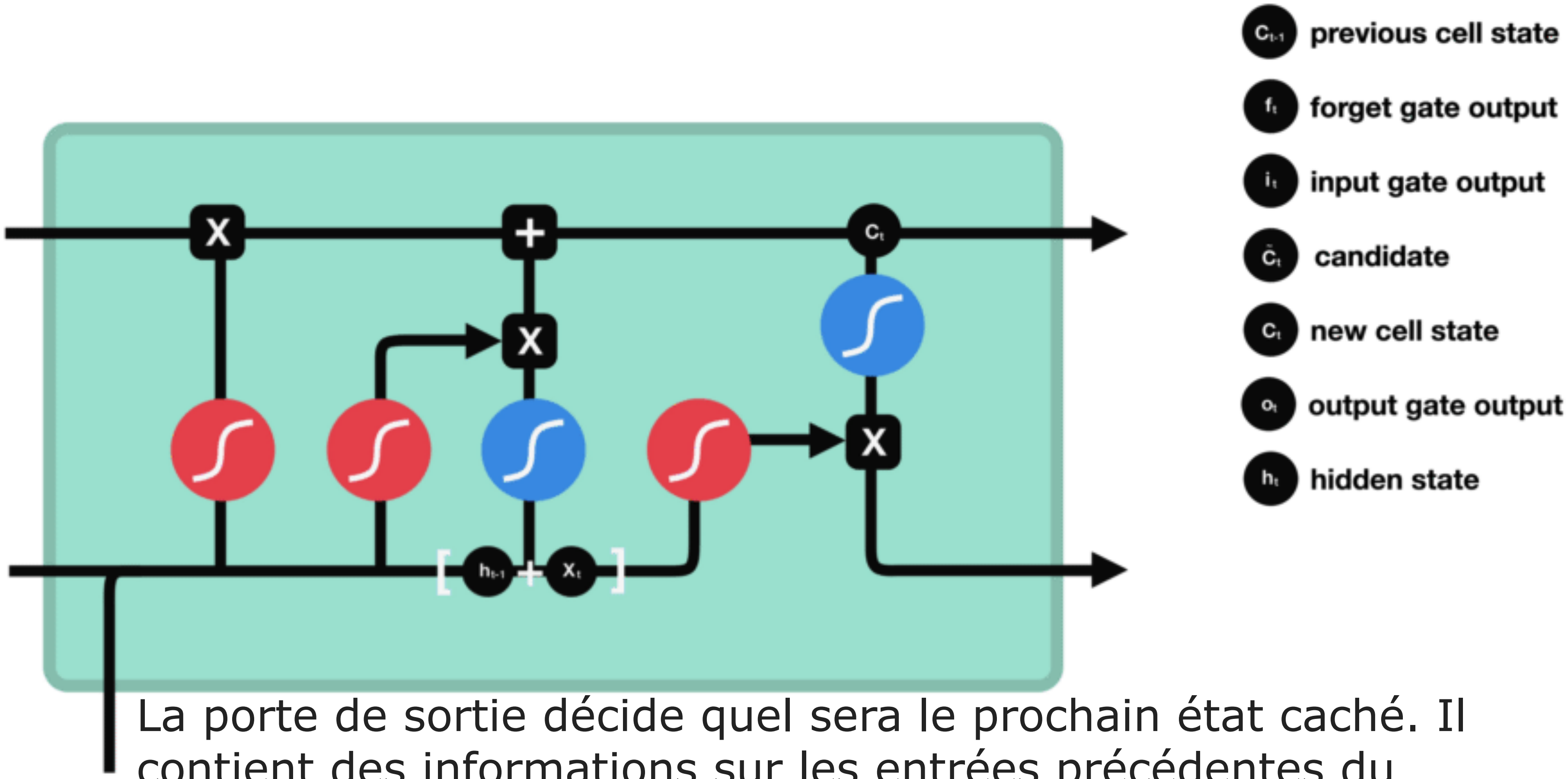


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

L’état de la cellule se calcule assez simplement à partir de la porte d’oubli et de la porte d’entrée.

The state of the cell is calculated quite simply from the forget gate and the input gate

LSTM cell (porte de sortie / output gate)

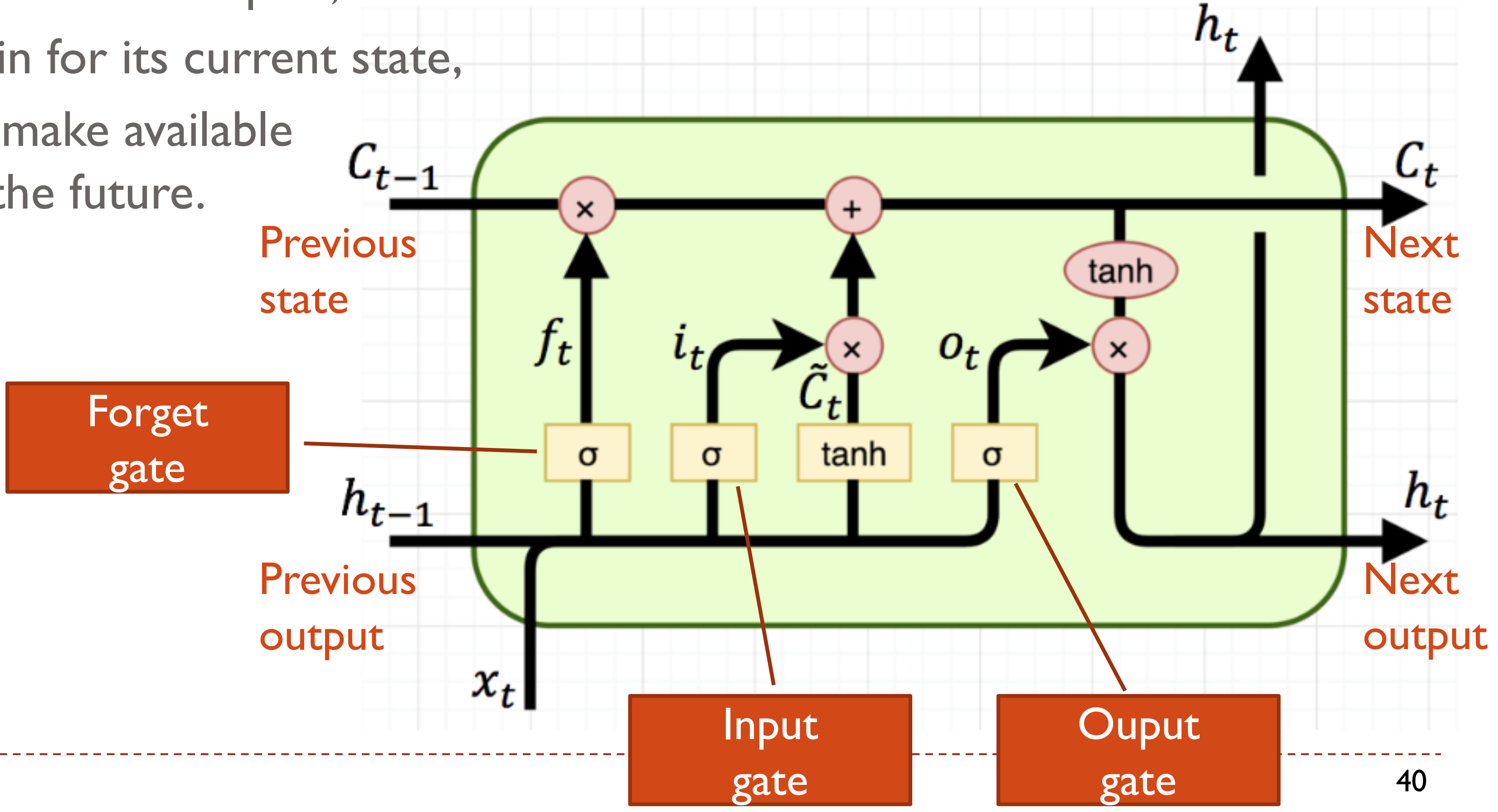


La porte de sortie décide quel sera le prochain état caché. Il contient des informations sur les entrées précédentes du réseau et sert aux prédictions.

The output gate decides what the next hidden state will be. It contains information about previous inputs to the network and is used for predictions.

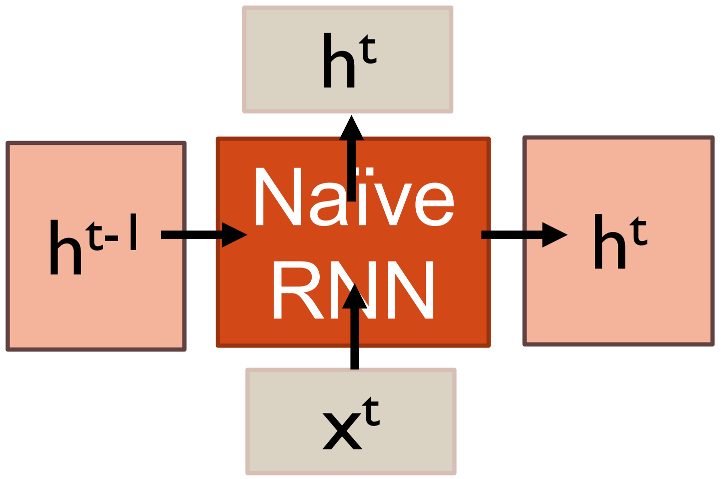
LSTM Cells - summary

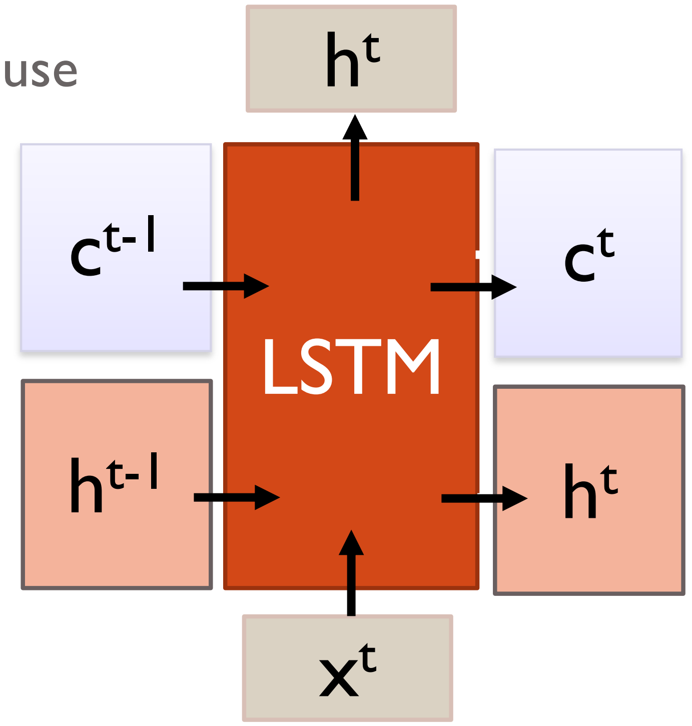
- Adds a context memory that affects the information flow and its processing (cell state).
- Three gates decide what a cell should
  - forget about its past,
  - retain for its current state,
  - and make available for the future.



Naïve RNN vs LSTM

- Naïve RNN
  - Reuse at each step the previous Output
- LSTM
  - At each step 3 gate control the use use of Input value, Cell state and previous Output



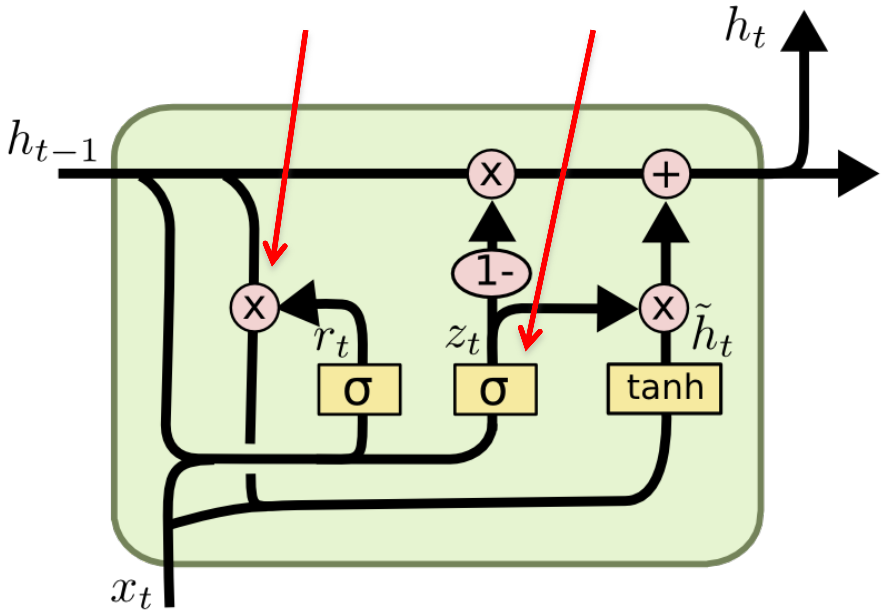


c changes slowly $\longrightarrow$ c^t is c^{t-1} added by something

h changes faster $\longrightarrow$ h^t and h^{t-1} can be very different

GRU – gated recurrent unit

- GRU = a light LSTM Cell



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

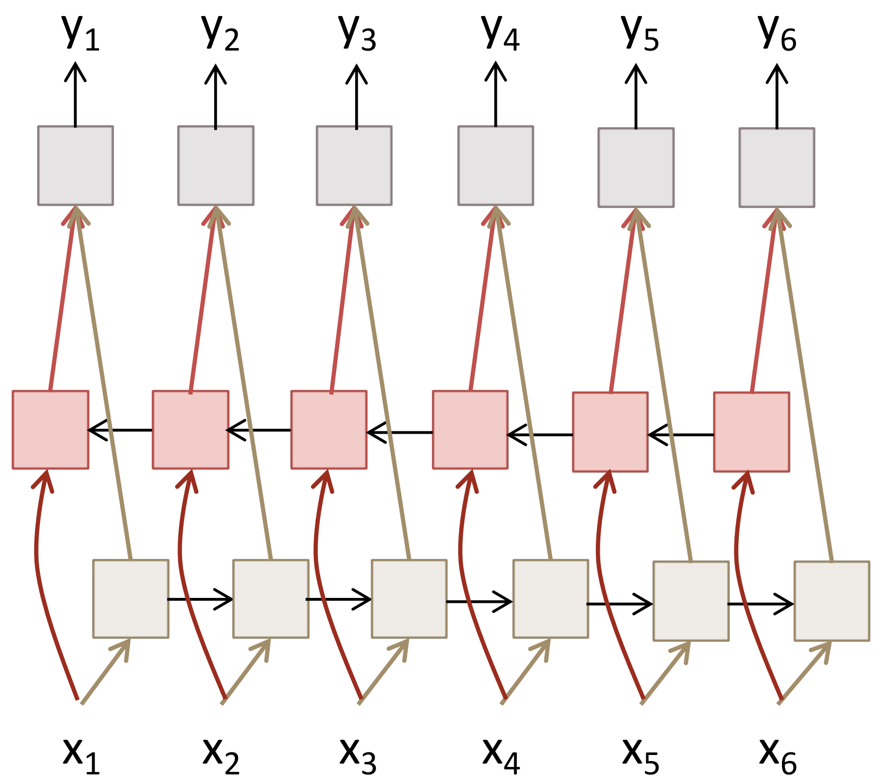
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

- It combines the forget and input into a single update gate.
- It also merges the cell state and hidden state.

→ This is simpler than LSTM.

Bi-directional RNNs

- RNNs can process the input sequence in forward and in the reverse direction



- Popular in speech recognition, could be used also with text

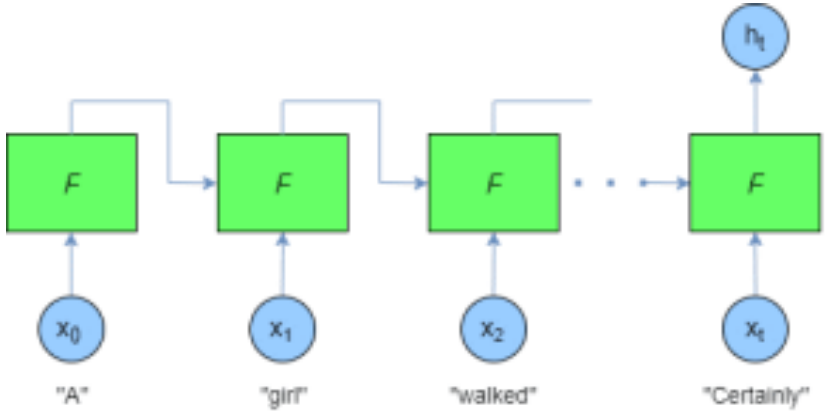
RNN cell in Keras

Keras Long Short-Term Memory Cell

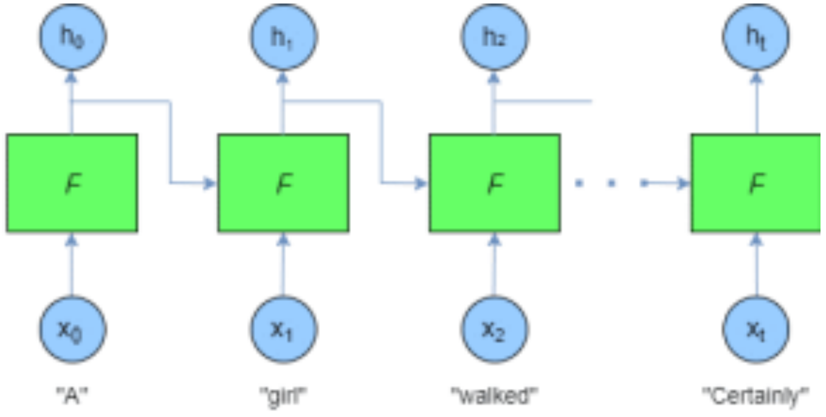
- **from** tf.keras.layers **import** LSTM

- Main params
  - **Units:** dimension of output space

  - **return_sequences:** True or False
    - If False return only the last output
    - If True return the full sequence of the output sequence
      - Output sequence = hidden state (the vocabulary change regardind documentation)
  - **return_state**: True or False
    - If True return 3 values
      - The full output sequence or only the last one (depend on return_sequences)
      - The last output sequence
      - The cell state
    - If False return nothing
  - **stateful**: True or False
    - If True, the last state for each sample at index i in a batch will be used as initial state for the sample of index i in the following batch.
    - You have to put shuffle=False in a fit method



return_sequences = False



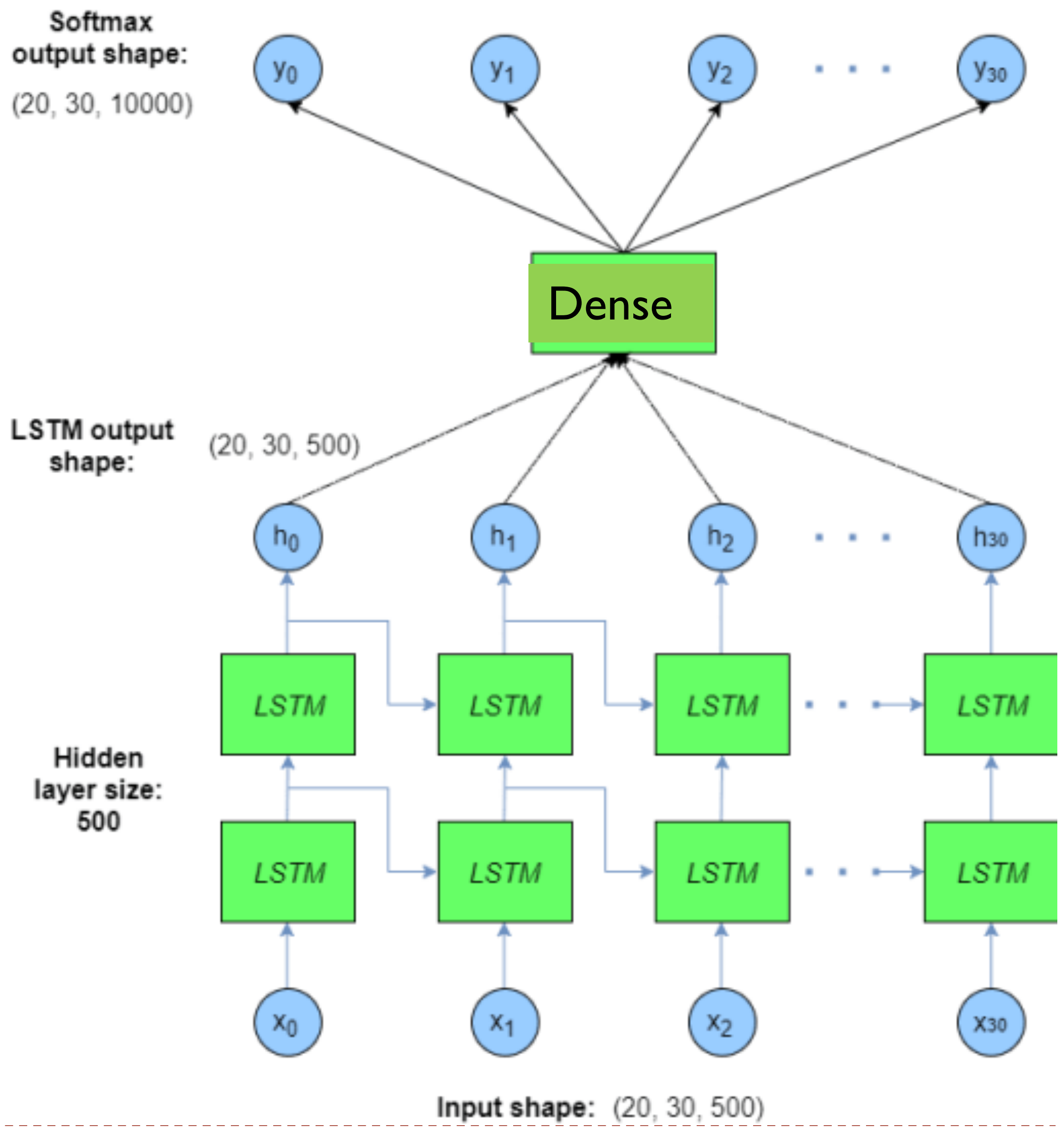
return_sequences = True

How many Dense cell ?

- X = LSTM(500, return sequence = False or True)
- Output = Dense(…)(X)

With return_sequences=False,
Dense layer is applied only once at the last cell

With return_sequences=True Dense layer is applied to every timestep



If return_sequences=True Dense layer is applied to every timestep

Keras Gated Recurrent Unit Cell

- **from** keras.layers **import** GRU

- Main params (similar to LSTM)
  - **Units:** dimension of output space

  - **return_sequences:** True or False
    - If False return only the last output
    - If True return the full sequence of the output
      - Output sequence = hidden state (the vocabulary change regardind documentation)
  - **return_state**: True or False
    - If True return 3 values
      - The full output sequence or only the last one (depend on return_sequences)
      - The last output sequence (is equivalent to the celle state for a GRU)
    - If False return nothing
  - **stateful**: True or False
    - If True, the last state for each sample at index i in a batch will be used as initial state for the sample of index i in the following batch.
    - You have to put shuffle=False in a fit method

A basic example

- inputs = Input(shape=(SEQUENCE_SIZE,))
- embedding = Embedding(VOCABULARY_SIZE,
  EMBEDDING_SIZE,
  input_length=SEQUENCE_SIZE)(inputs)
- output = LSTM(16, return_sequences=False,
  activation='relu')(embedding)
- predictions = Dense(nb_classes,
  activation='softmax')(output)

- Fit by batch
  - Model.fit(X, y, ….). ← all item have the same length
- Fit by item
  - For i in range(len(X)): ← could be different length
    - Model.fit(X[i], y[i], …)

Some use of RNN → Text Classification / Sentiment analysis

Affect a label to a text

- Classify a
  - restaurant review from Yelp!
  - movie review from IMDB

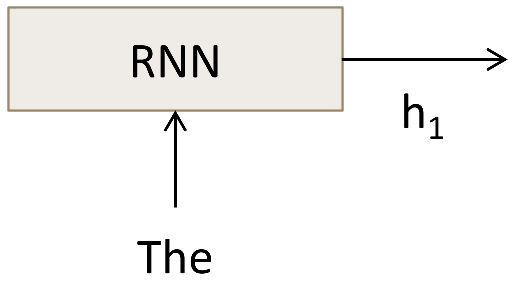
  …

  as positive or negative

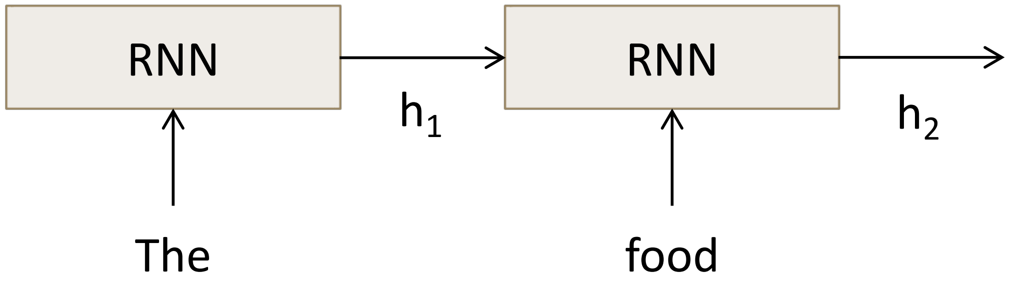
- Inputs:
  - Multiple words, one or more sentences
- Outputs:
  - Positive / Negative classification

- "The food was really good"
- "The chicken crossed the road because it was uncooked"

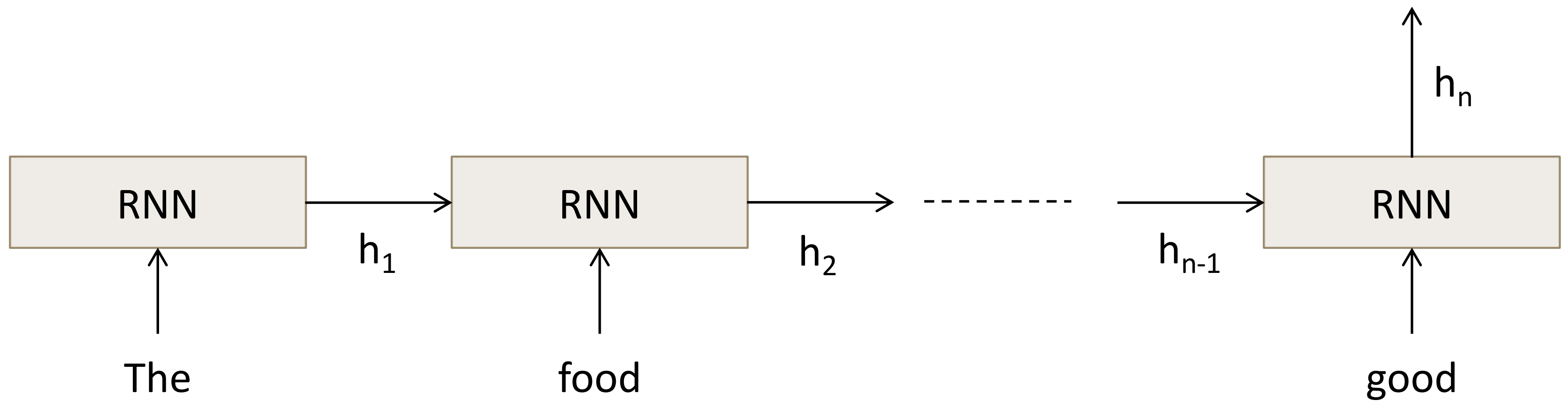
RNN for sentiment analysis



RNN for sentiment analysis

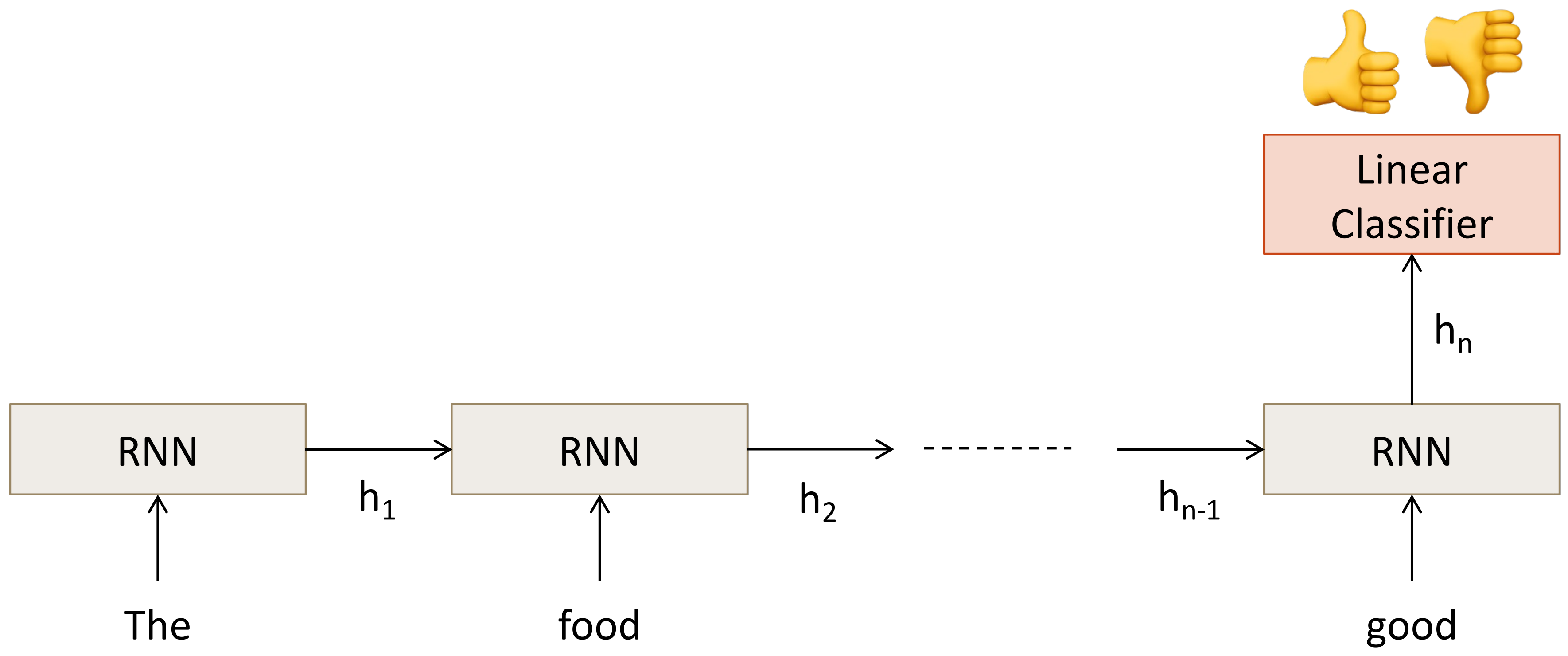


RNN for sentiment analysis



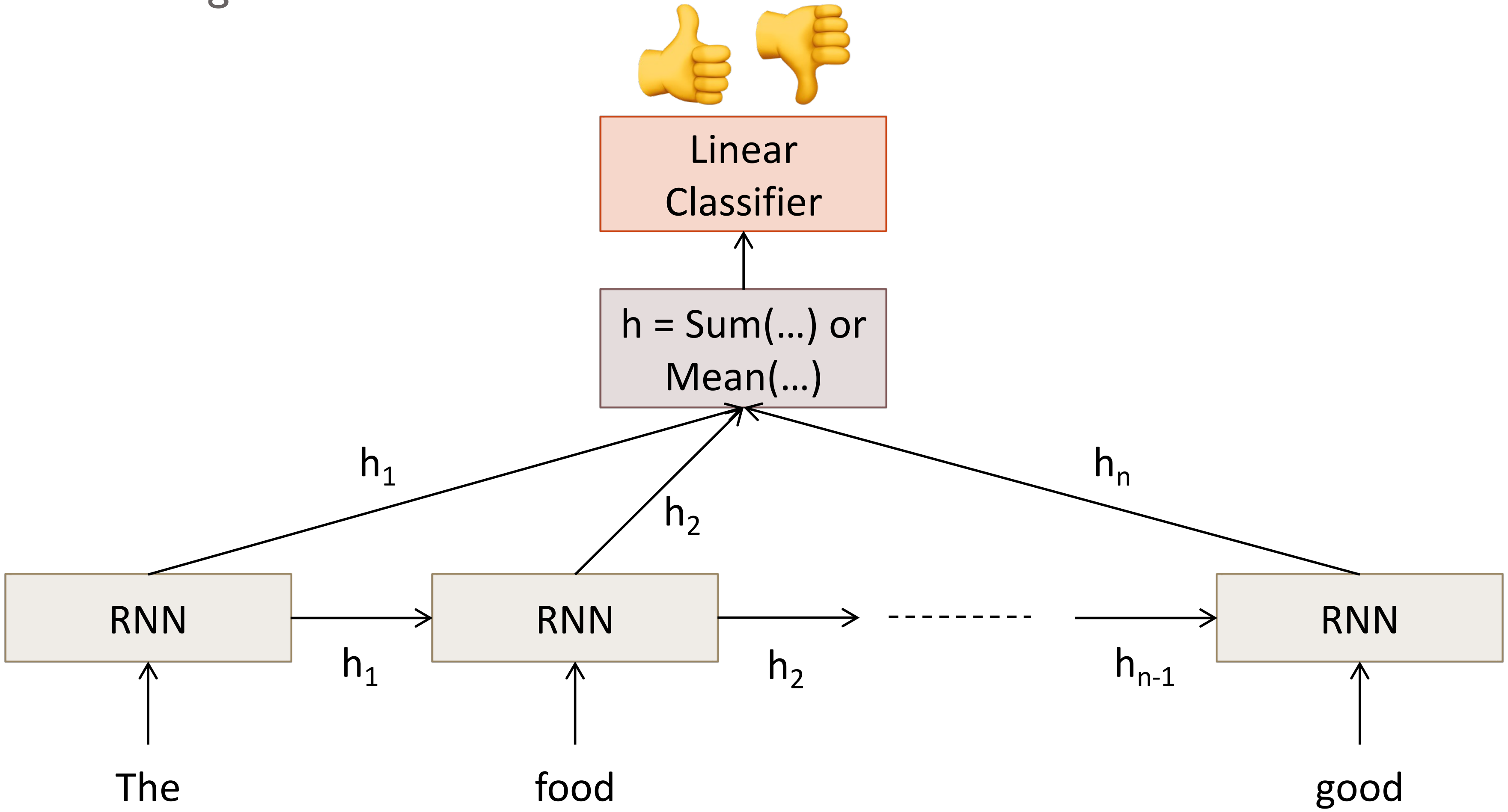
RNN for sentiment analysis - solution 1

- retrieve only the last state



RNN for sentiment analysis– solution2

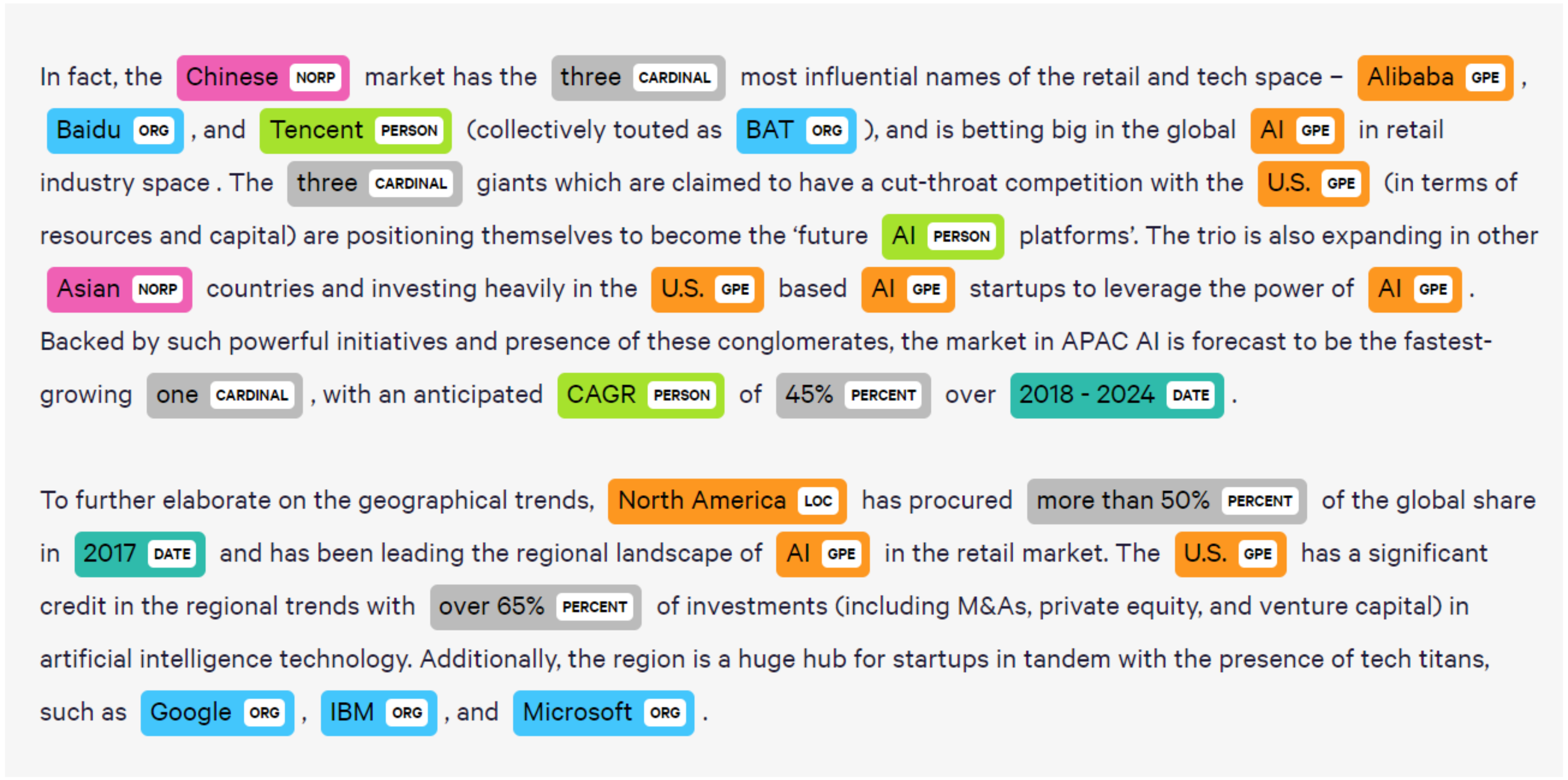
- Other possible architecture
  - Average all the internal state



http://deeplearning.net/tutorial/lstm.html

Some use of RNN
→ Named Entity Recognition / Part of Speech Tagging

- Affect a label to each word
  - **find** and **classify** names in text
    - Could bean entity : number, country, person, … (NER)
    - Could be a function : noun, verb, adverbs, … (POS)

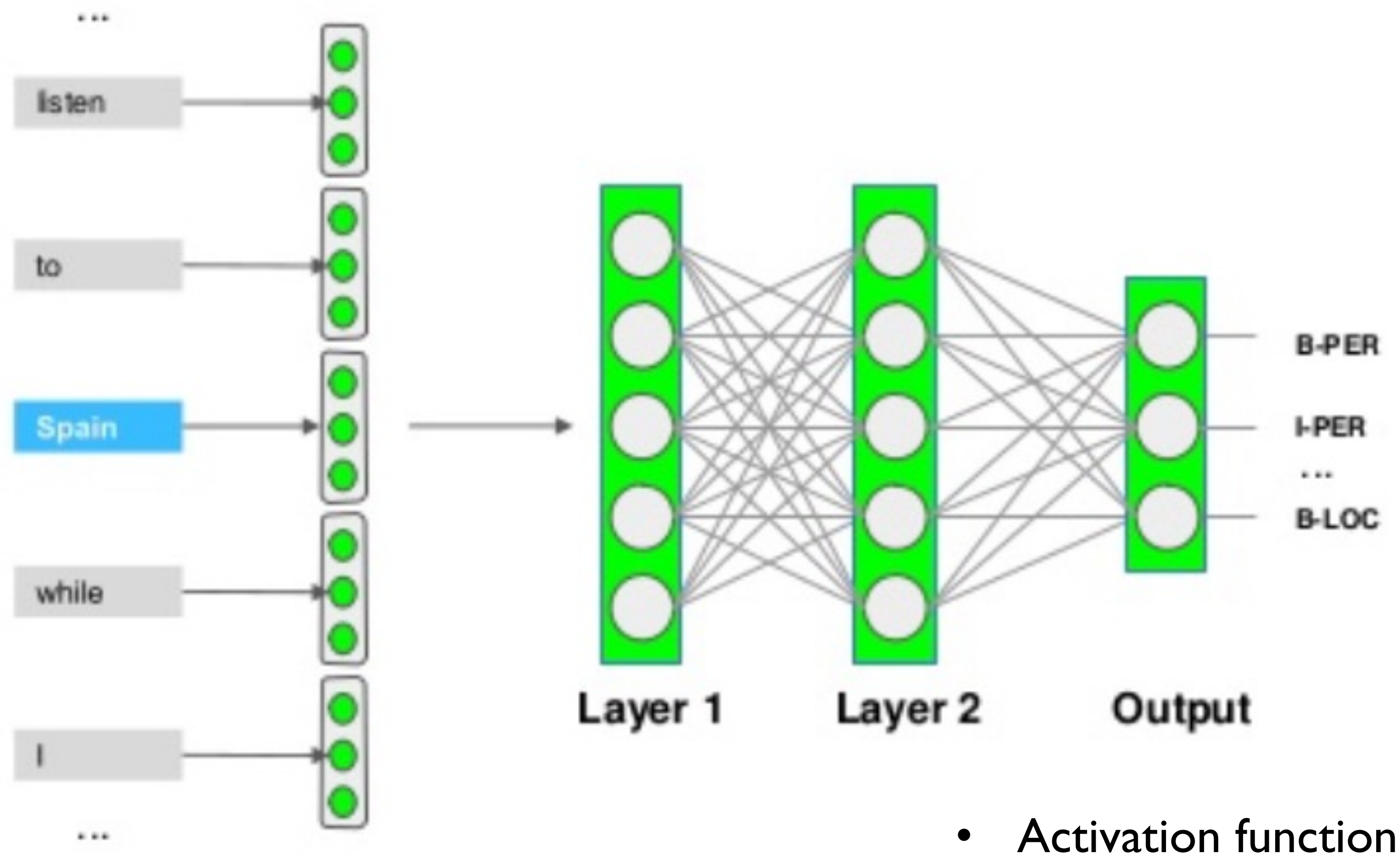

In fact, the Chinese NORP market has the three CARDINAL most influential names of the retail and tech space – Alibaba GPE , Baidu ORG , and Tencent PERSON (collectively touted as BAT ORG), and is betting big in the global AI GPE in retail industry space . The three CARDINAL giants which are claimed to have a cut-throat competition with the U.S. GPE (in terms of resources and capital) are positioning themselves to become the 'future AI PERSON platforms'. The trio is also expanding in other Asian NORP countries and investing heavily in the U.S. GPE based AI GPE startups to leverage the power of AI GPE . Backed by such powerful initiatives and presence of these conglomerates, the market in APAC AI is forecast to be the fastest-growing one CARDINAL , with an anticipated CAGR PERSON of 45% PERCENT over 2018 - 2024 DATE .

To further elaborate on the geographical trends, North America LOC has procured more than 50% PERCENT of the global share in 2017 DATE and has been leading the regional landscape of AI GPE in the retail market. The U.S. GPE has a significant credit in the regional trends with over 65% PERCENT of investments (including M&As, private equity, and venture capital) in artificial intelligence technology. Additionally, the region is a huge hub for startups in tandem with the presence of tech titans, such as Google ORG , IBM ORG , and Microsoft ORG .

Label representation – BIO tags

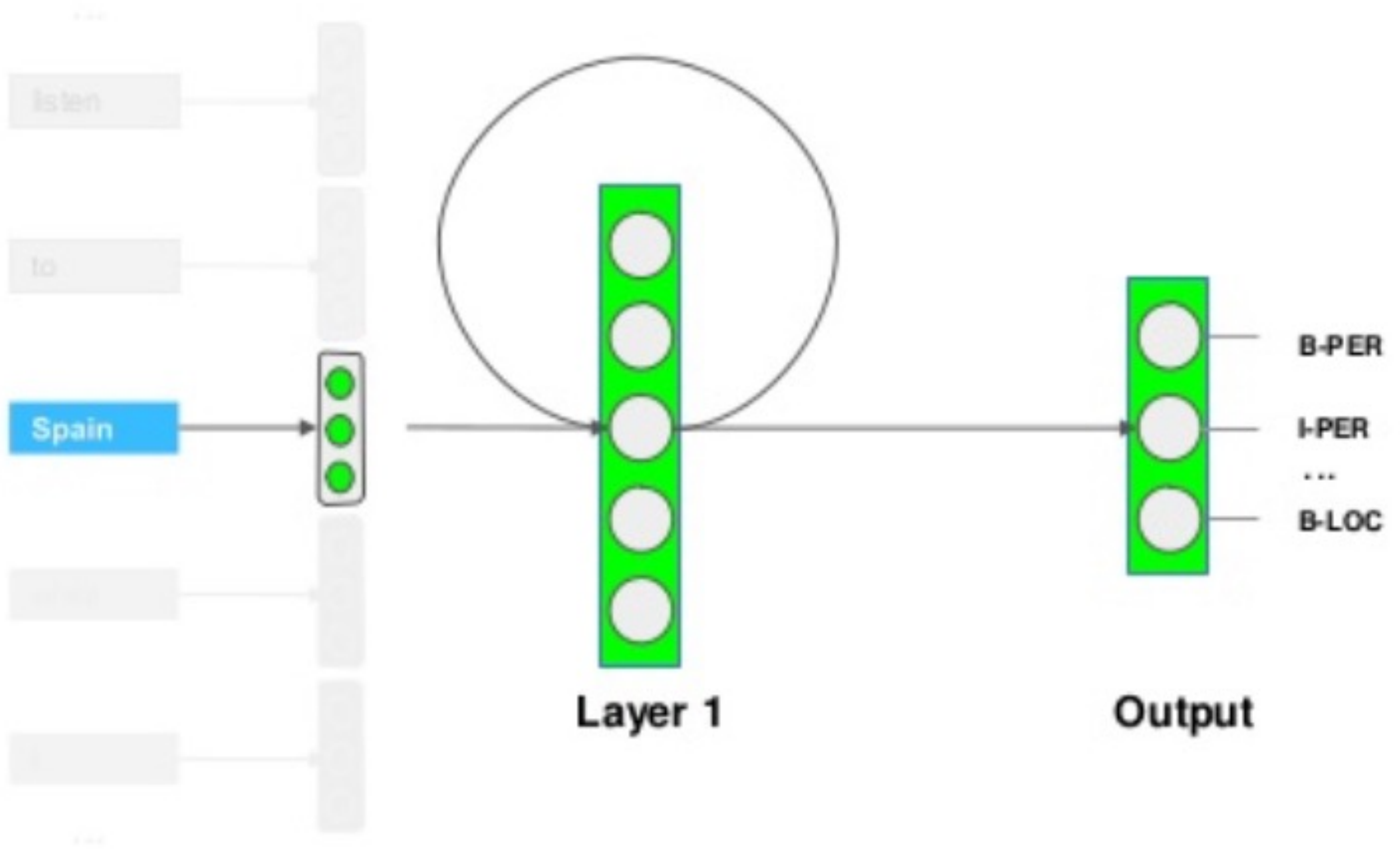
- Labels can be for words or groups of words
  - Adam Smith works for IBM , London .
- To represent this, a "BIO" representation is generally used.
  - B beginning of an entity
  - I continues the entity
  - O word outside the entity
- For example
  - [‘Adam’, ‘Smith’, ‘works’, ‘for’, ‘IBM’, ‘,’, ‘London’, ‘.’]
  - Without BIO: [PER, PER, O, O, ORG, O, GEO, O]
  - With BIO: [B_PER, I_PER, O, O, B_ORG, O, B_GEO, O]

MLP for NER

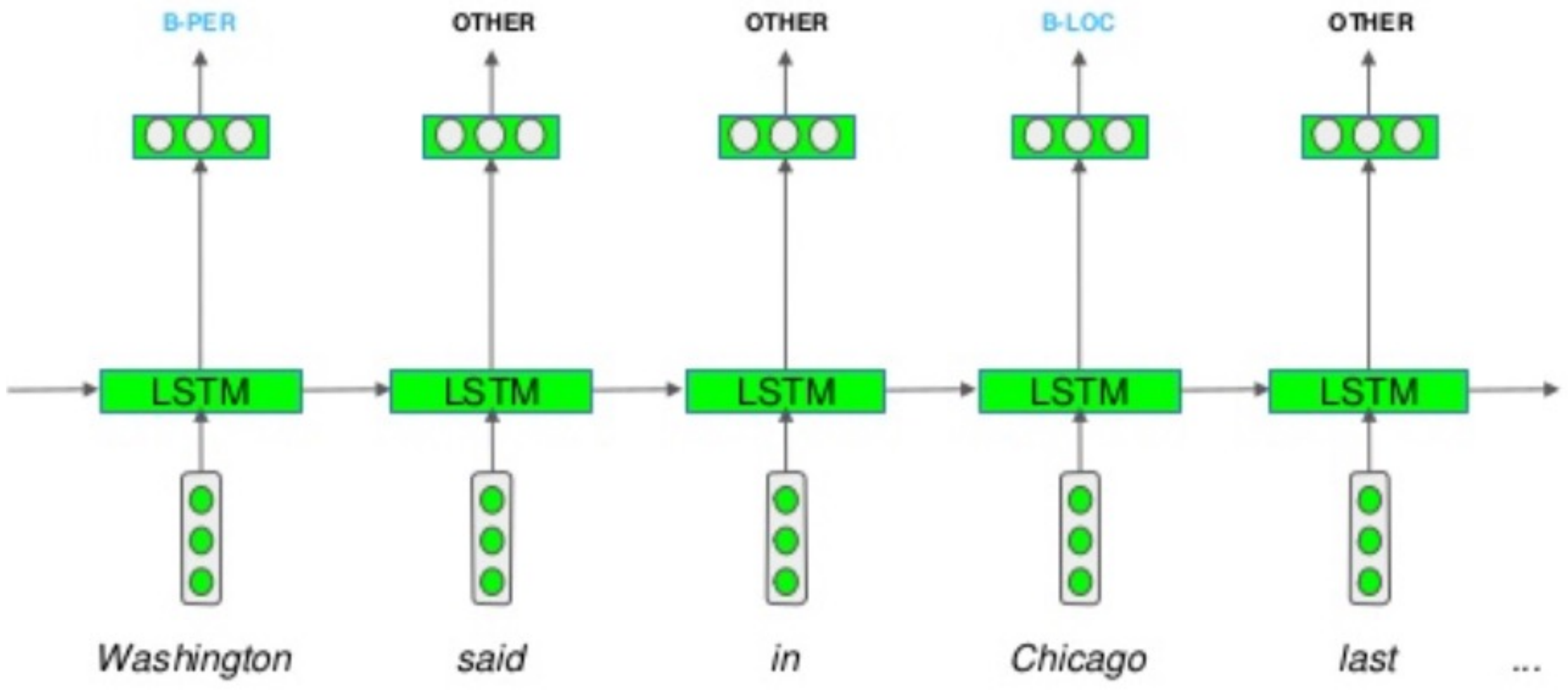


- Activation function for output: softmax
- Labels are OneHotEncoded

Recurrent neural network for NER

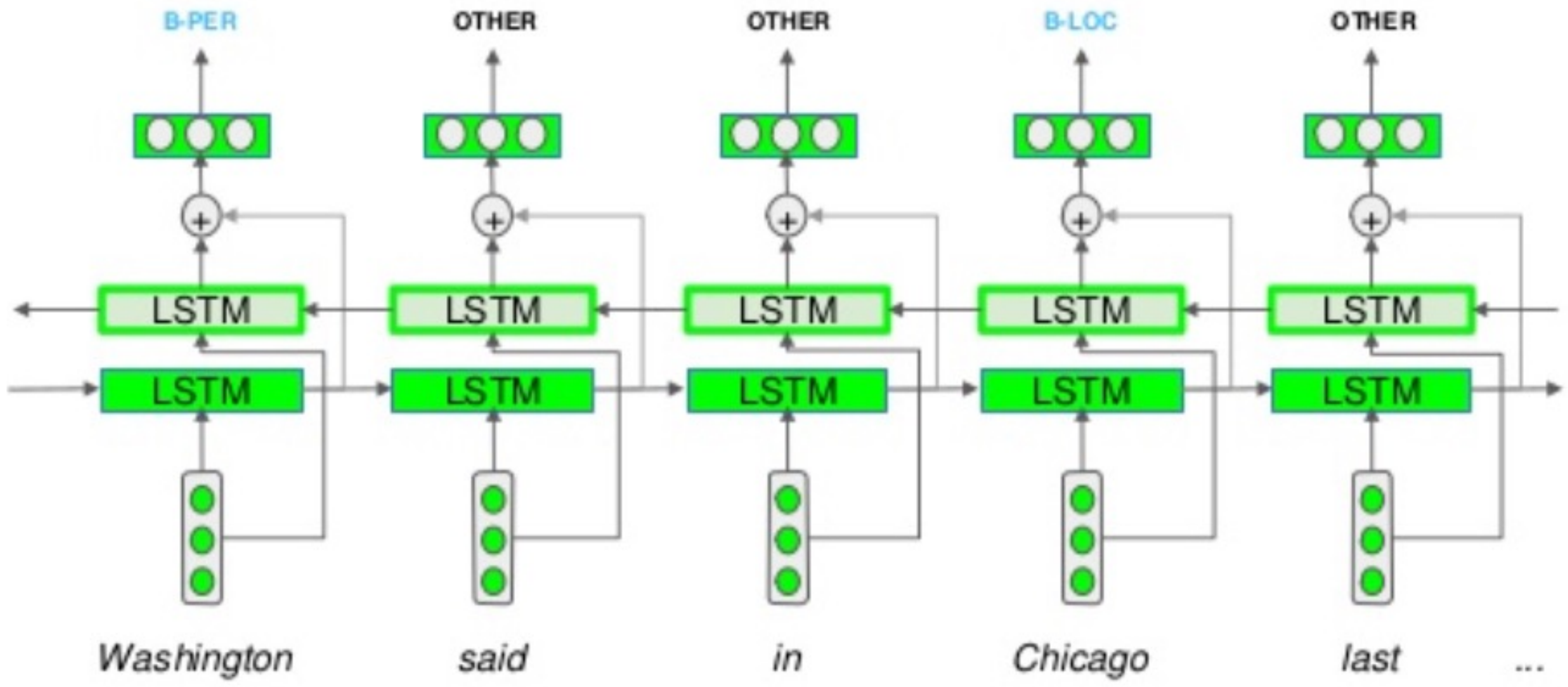


Recurrent neural network for NER (unfolded)



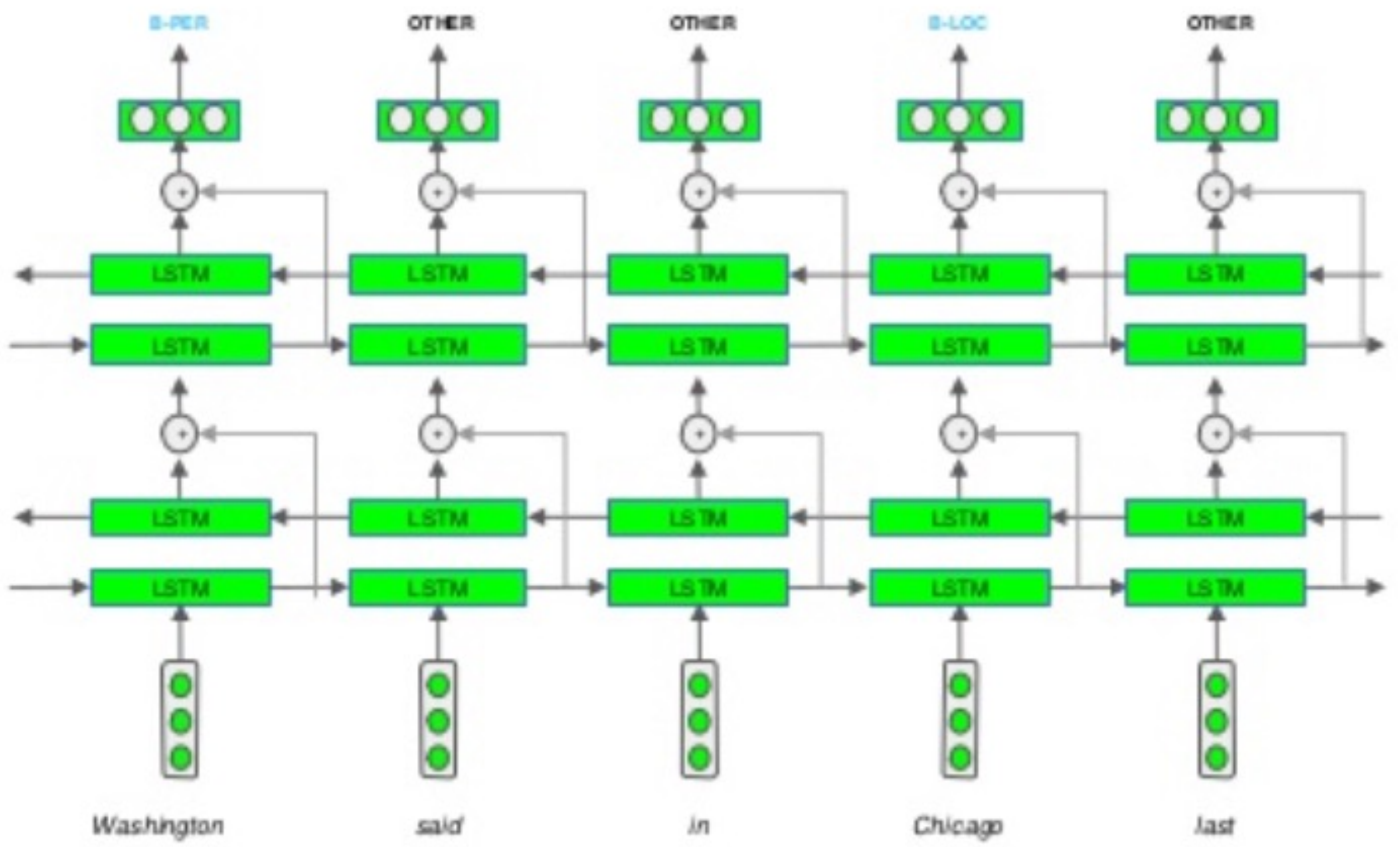
- Activation function for output: softmax
- Labels are OneHotEncoded

Bi directional recurrent neural network for NER



- Activation function for output: softmax
- Labels are OneHotEncoded

Stacked Bi-RNN



- Activation function for output: softmax
- Labels are OneHotEncoded

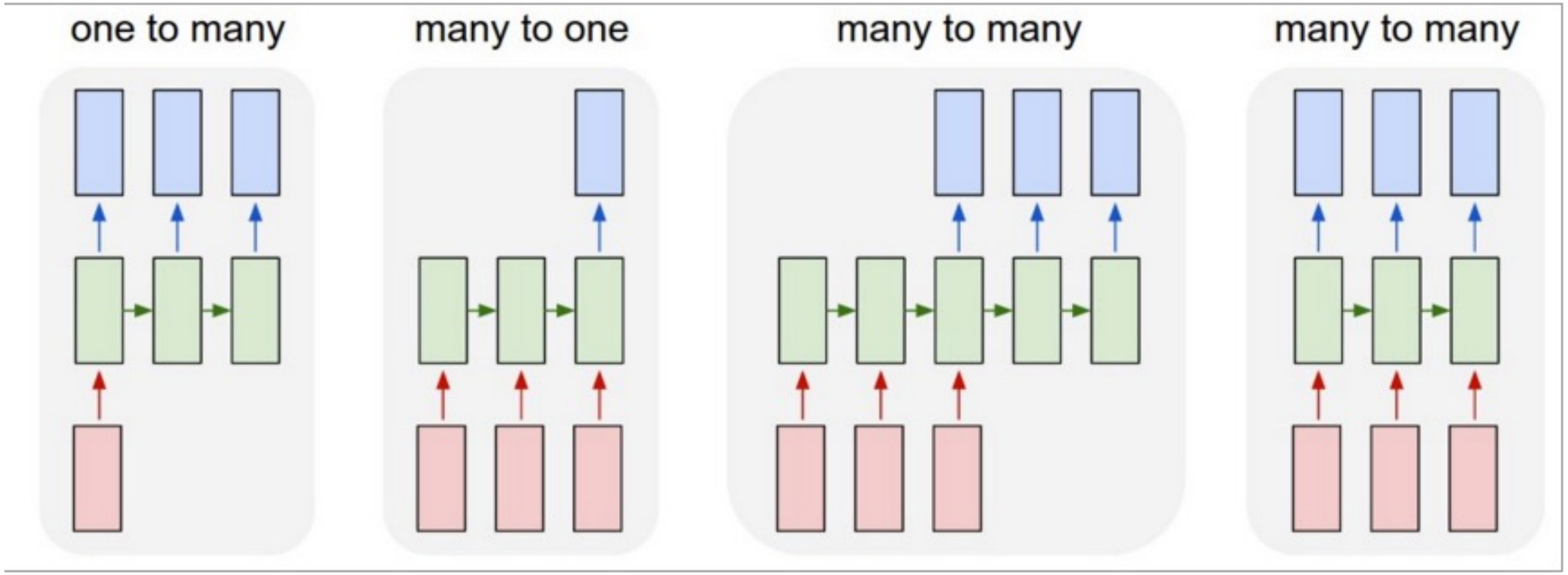
LSTMs can be used for other sequence tasks

image captioning

sequence classification

translation

named entity recognition



Conclusion

- Two main cells
  - LSTM
  - GRU

  + Bidirectionnal
- PRO
  - Inputs can be of variable size
  - Model size does not increase with input size (Weights are shared)
  - Calculations take into account previous information
- CONS
  - Long computing time
  - Difficulty to access information from the distant past
  - Impossible to take into account future information

- https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks

Other Useful Resources / References

- http://cs231n.stanford.edu/slides/winter1516_lecture10.pdf
- http://www.cs.toronto.edu/~rgrosse/csc321/lec10.pdf

- R. Pascanu, T. Mikolov, and Y. Bengio, On the difficulty of training recurrent neural networks, ICML 2013
- S. Hochreiter, and J. Schmidhuber, Long short-term memory, Neural computation, 1997 9(8), pp.1735-1780
- F.A. Gers, and J. Schmidhuber, Recurrent nets that time and count, IJCNN 2000
- K. Greff , R.K. Srivastava, J. Koutník, B.R. Steunebrink, and J. Schmidhuber, LSTM: A search space odyssey, IEEE transactions on neural networks and learning systems, 2016
- K. Cho, B. Van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, Learning phrase representations using RNN encoder-decoder for statistical machine translation, ACL 2014
- R. Jozefowicz, W. Zaremba, and I. Sutskever, An empirical exploration of recurrent network architectures, JMLR 2015