

# OEAW AI SUMMER SCHOOL

## DEEP LEARNING IV

### Recurrent Neural Networks



Michael Widrich, Johannes Brandstetter  
Institute for Machine Learning

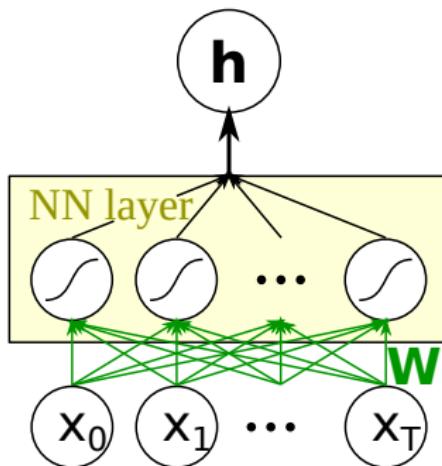
# Lecture IV: Recurrent Neural Networks



# **Basics of Recurrent Neural Networks (RNNs)**

# Limitations of Feedforward Networks

- Size of input vector  $x$  is fixed
- Temporal relations of elements in **sequence of inputs** lost
  - Restricted work-around: Windowing via convolution
- Can learn mathematical functions but not **Turing Complete**



# Recurrent Neural Networks (RNNs) (1)

- Assume a sample is a sequence of length  $T$  with  $D$  features at each timestep  $t$ .
  - Each sample represented by matrix  $\mathbf{X}$  of shape  $T \times D$
  - $T$  may vary between samples but  $D$  is constant

# Recurrent Neural Networks (RNNs) (1)

- Assume a sample is a sequence of length  $T$  with  $D$  features at each timestep  $t$ .
  - Each sample represented by matrix  $\mathbf{X}$  of shape  $T \times D$
  - $T$  may vary between samples but  $D$  is constant
- To address the mentioned limitations of feedforward networks, our network needs to:
  1. be able to handle variable sequence lengths  $T$ ,
  2. remember previous inputs

## Recurrent Neural Networks (RNNs) (2)

- Solution: Recurrent Neural Networks (RNNs)

## Recurrent Neural Networks (RNNs) (2)

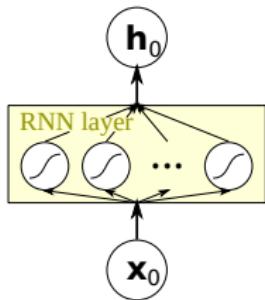
### ■ Solution: Recurrent Neural Networks (RNNs)

- Feed input sequence  $X$  timestep by timestep into network  
(= vector  $x_t$  with  $D$  features at each timestep  $t$ )
- Add previous output  $h_{t-1}$  (=hidden state) to current input  $x_t$   
→  $x_t$  and  $h_{t-1}$  are inputs to compute new  $h_t$ :  $h_t = f(x_t, h_{t-1})$

# Recurrent Neural Networks (RNNs) (2)

## ■ Solution: Recurrent Neural Networks (RNNs)

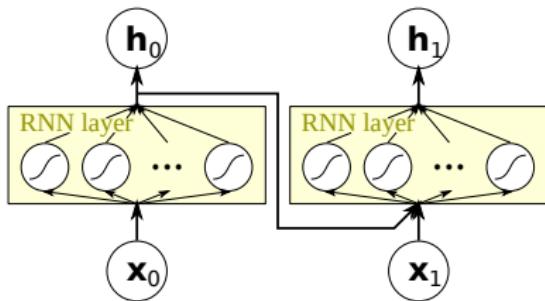
- Feed input sequence  $\mathbf{X}$  timestep by timestep into network  
(= vector  $\mathbf{x}_t$  with  $D$  features at each timestep  $t$ )
- Add previous output  $\mathbf{h}_{t-1}$  (=hidden state) to current input  $\mathbf{x}_t$   
→  $\mathbf{x}_t$  and  $\mathbf{h}_{t-1}$  are inputs to compute new  $\mathbf{h}_t$ :  $\mathbf{h}_t = f(\mathbf{x}_t, \mathbf{h}_{t-1})$



# Recurrent Neural Networks (RNNs) (2)

## ■ Solution: Recurrent Neural Networks (RNNs)

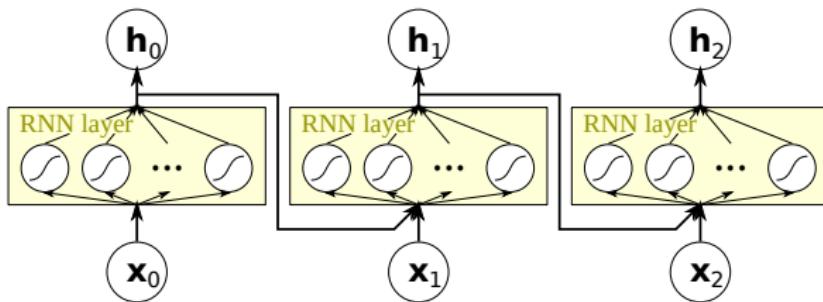
- Feed input sequence  $X$  timestep by timestep into network  
(= vector  $x_t$  with  $D$  features at each timestep  $t$ )
- Add previous output  $h_{t-1}$  (=hidden state) to current input  $x_t$   
→  $x_t$  and  $h_{t-1}$  are inputs to compute new  $h_t$ :  $h_t = f(x_t, h_{t-1})$



# Recurrent Neural Networks (RNNs) (2)

## ■ Solution: Recurrent Neural Networks (RNNs)

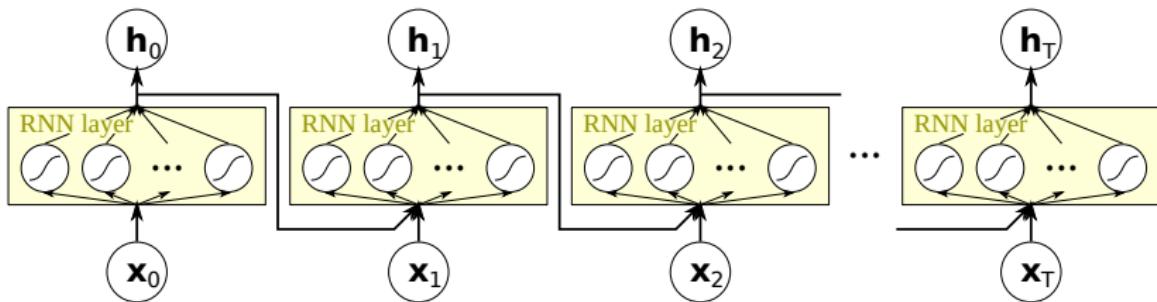
- Feed input sequence  $X$  timestep by timestep into network  
(= vector  $x_t$  with  $D$  features at each timestep  $t$ )
- Add previous output  $h_{t-1}$  (=hidden state) to current input  $x_t$   
→  $x_t$  and  $h_{t-1}$  are inputs to compute new  $h_t$ :  $h_t = f(x_t, h_{t-1})$



# Recurrent Neural Networks (RNNs) (2)

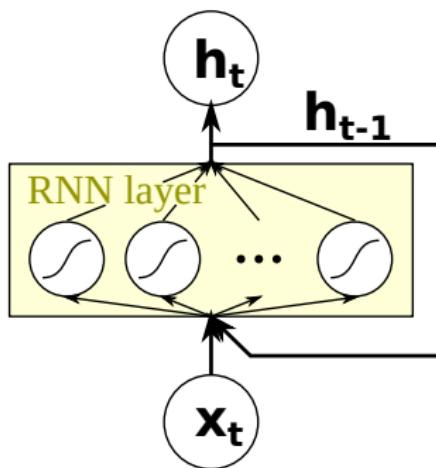
## ■ Solution: Recurrent Neural Networks (RNNs)

- Feed input sequence  $X$  timestep by timestep into network  
(= vector  $x_t$  with  $D$  features at each timestep  $t$ )
- Add previous output  $h_{t-1}$  (=hidden state) to current input  $x_t$   
→  $x_t$  and  $h_{t-1}$  are inputs to compute new  $h_t$ :  $h_t = f(x_t, h_{t-1})$



## Recurrent Neural Networks (RNNs) (2)

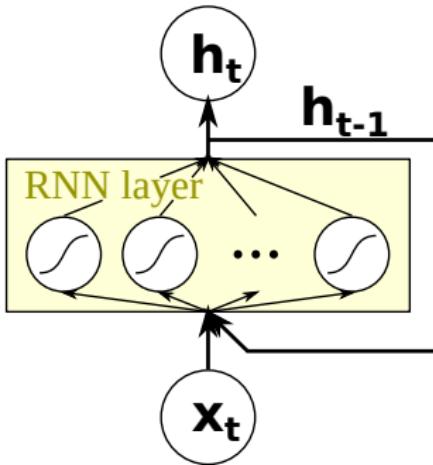
- Solution: Recurrent Neural Networks (RNNs)
  - Feed input sequence  $X$  timestep by timestep into network  
(= vector  $x_t$  with  $D$  features at each timestep  $t$ )
  - Add previous output  $h_{t-1}$  (=hidden state) to current input  $x_t$   
→  $x_t$  and  $h_{t-1}$  are inputs to compute new  $h_t$ :  $h_t = f(x_t, h_{t-1})$



## Recurrent Neural Networks (RNNs) (3)

- Typically only single layers recursively connected
- Layer weight matrix  $\mathbf{W}$  reused (=shared) for all timesteps
- Computation of  $\mathbf{h}_t$  similar to feedforward networks:

$$\mathbf{h}_t = \mathbf{W}^T \cdot \begin{pmatrix} \mathbf{x}_t \\ \mathbf{h}_{t-1} \end{pmatrix} + \mathbf{b}$$



## Power of RNNs

- RNNs are in essence a state-space model:  $y_t = f(x, y_{t-1})$
- RNNs are used for sequence classification, sequence generation, control systems, meta learning, ...

## Power of RNNs

- RNNs are in essence a state-space model:  $y_t = f(x, y_{t-1})$
- RNNs are used for sequence classification, sequence generation, control systems, meta learning, ...
- RNNs are theoretically **Turing Complete**
  - For each program in a turing complete programming language, you could find an RNN that executes the code correctly

# Power of RNNs

- RNNs are in essence a state-space model:  $y_t = f(x, y_{t-1})$
- RNNs are used for sequence classification, sequence generation, control systems, meta learning, ...
- RNNs are theoretically **Turing Complete**
  - For each program in a turing complete programming language, you could find an RNN that executes the code correctly
  - Like the Universal Approximation Theorem, this is mostly useless in practice
  - The hard problem is not “what can we represent?” but finding a good representation

# Power of RNNs

- RNNs are in essence a state-space model:  $y_t = f(x, y_{t-1})$
- RNNs are used for sequence classification, sequence generation, control systems, meta learning, ...
- RNNs are theoretically **Turing Complete**
  - For each program in a turing complete programming language, you could find an RNN that executes the code correctly
  - Like the Universal Approximation Theorem, this is mostly useless in practice
  - The hard problem is not “what can we represent?” but finding a good representation

# Sequence Learning Settings

# Sequence Learning Settings (1)

Alex Graves distinguishes 3 types of classification tasks:

**Sequence Classification:** 1 label per sequence

**Segment Classification:** 1 label per part of sequence

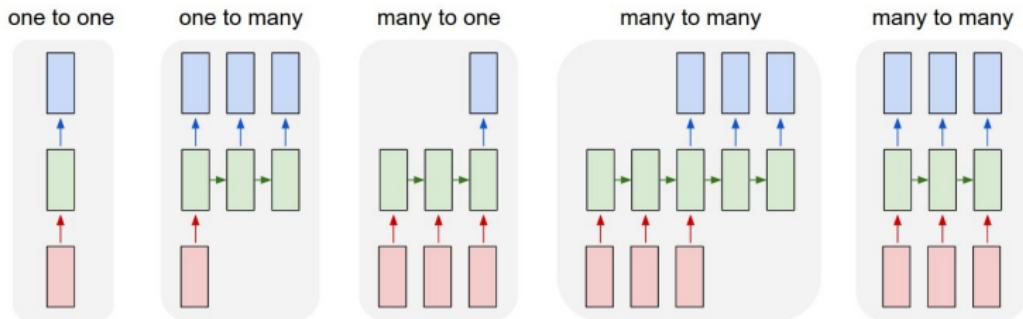
**Temporal Classification:** sequence of labels per sequence

*Foreign Minister.* → FOREIGN MINISTER.

→ THE SOUND OF

[Supervised Sequence Labelling with Recurrent Neural Networks, A. Graves, 2012]

## Sequence Learning Settings (2)

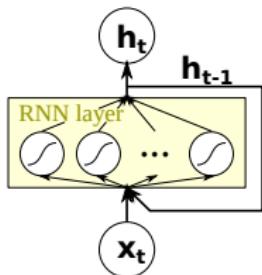


[The Unreasonable Effectiveness of Recurrent Neural Networks, A. Karpathy, 2015]

# RNN Training

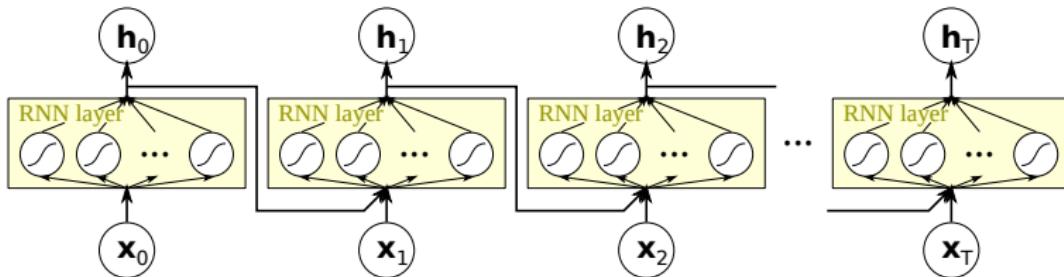
# Unrolling an RNN

- RNN can be viewed as feed forward network with shared weights = **unrolled** over time



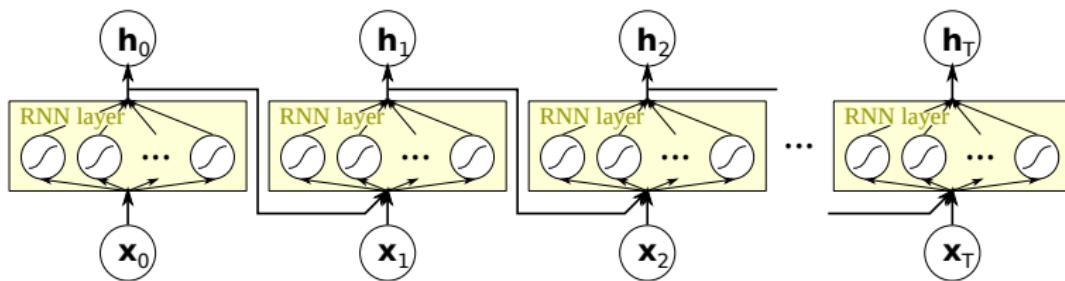
# Unrolling an RNN

- RNN can be viewed as feed forward network with shared weights = **unrolled** over time



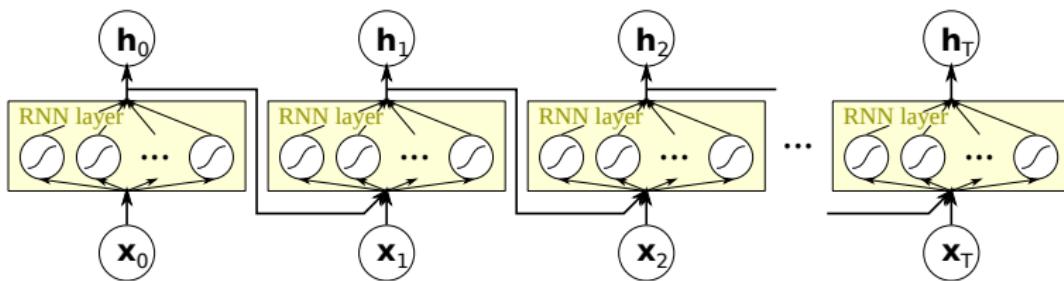
# Back-Propagation Through Time (BPTT)

- Most common way to train RNNs: Back-Propagation Through Time (BPTT)



# Back-Propagation Through Time (BPTT)

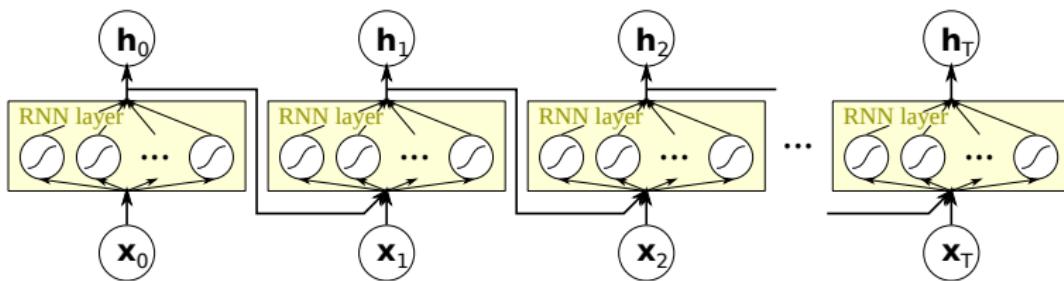
- Most common way to train RNNs: Back-Propagation Through Time (BPTT)



- Complexity:  $\mathcal{O}(N^2T)$ 
  - $N$ : number of hidden units
  - $T$ : Length of sequence

# Back-Propagation Through Time (BPTT)

- Most common way to train RNNs: Back-Propagation Through Time (BPTT)



- Complexity:  $\mathcal{O}(N^2T)$ 
  - $N$ : number of hidden units
  - $T$ : Length of sequence
- Truncated BPTT: only unfold  $n$  timesteps into the past

# Real-Time Recurrent Learning (RTRL)

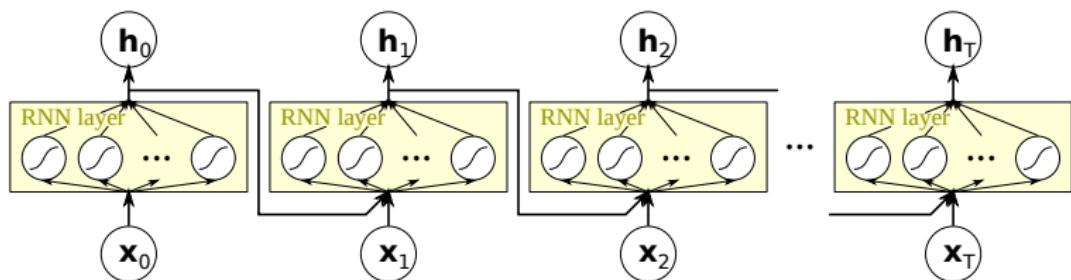
- Alternative to BPTT
- Computes all gradient information during forward pass
- Complexity  $\mathcal{O}(N^4) \Rightarrow$  Independent of sequence length
- Very rarely used today

## Vanishing Gradients Problem

- BPTT generates very deep networks ( $T \cong$  depth)  
→ Vanishing or Exploding Gradients (Hochreiter, 1991)

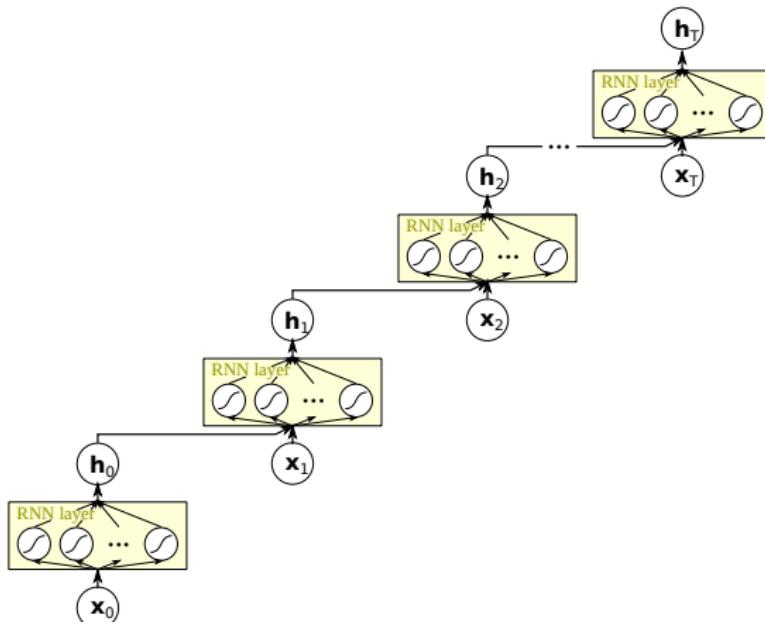
# Vanishing Gradients Problem

- BPTT generates very deep networks ( $T \cong \text{depth}$ )  
→ Vanishing or Exploding Gradients (Hochreiter, 1991)



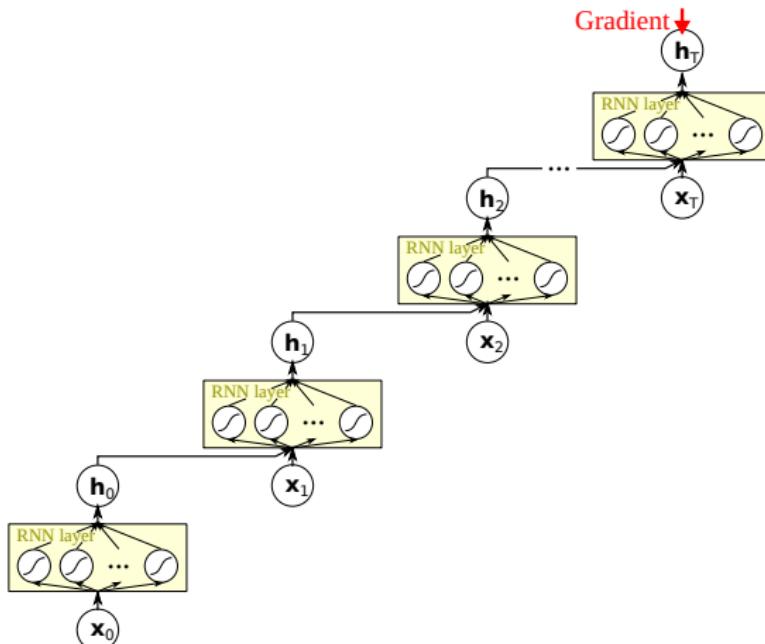
# Vanishing Gradients Problem

- BPTT generates very deep networks ( $T \cong \text{depth}$ )  
→ Vanishing or Exploding Gradients (Hochreiter, 1991)



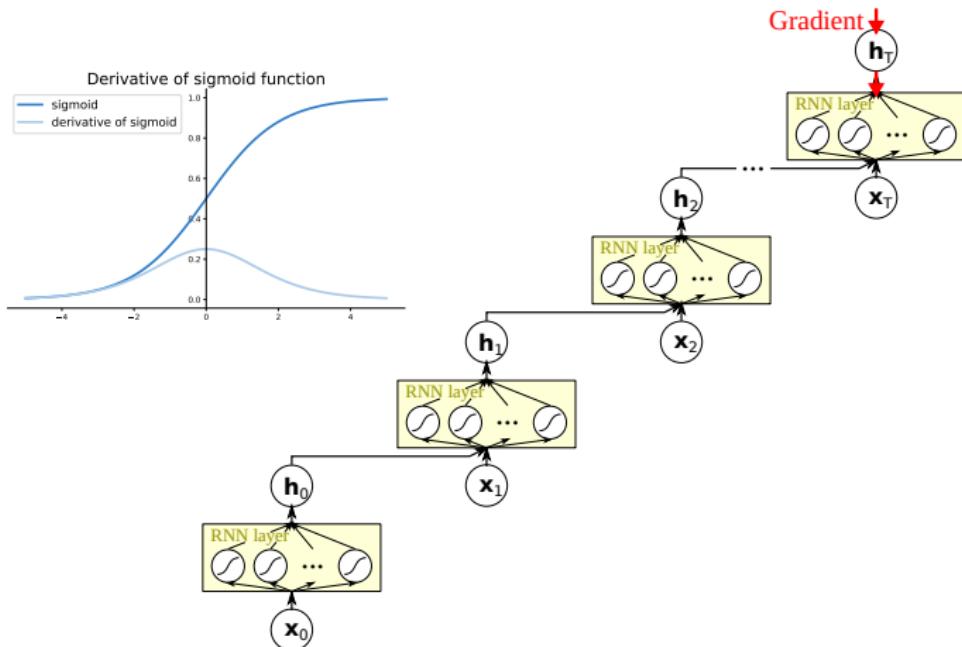
# Vanishing Gradients Problem

- BPTT generates very deep networks ( $T \cong \text{depth}$ )  
→ Vanishing or Exploding Gradients (Hochreiter, 1991)



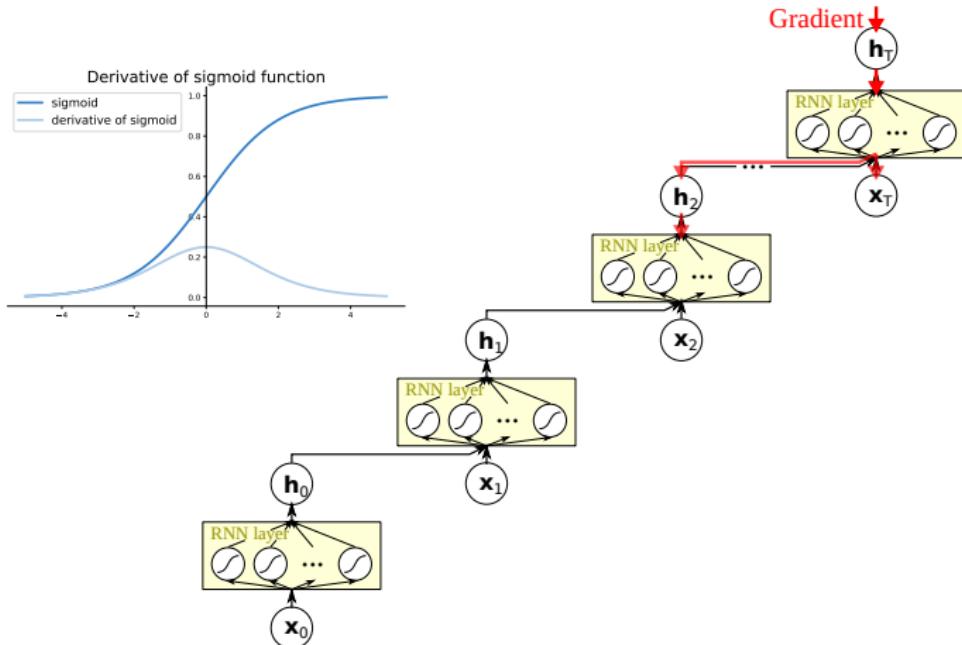
# Vanishing Gradients Problem

- BPTT generates very deep networks ( $T \cong \text{depth}$ )  
→ Vanishing or Exploding Gradients (Hochreiter, 1991)



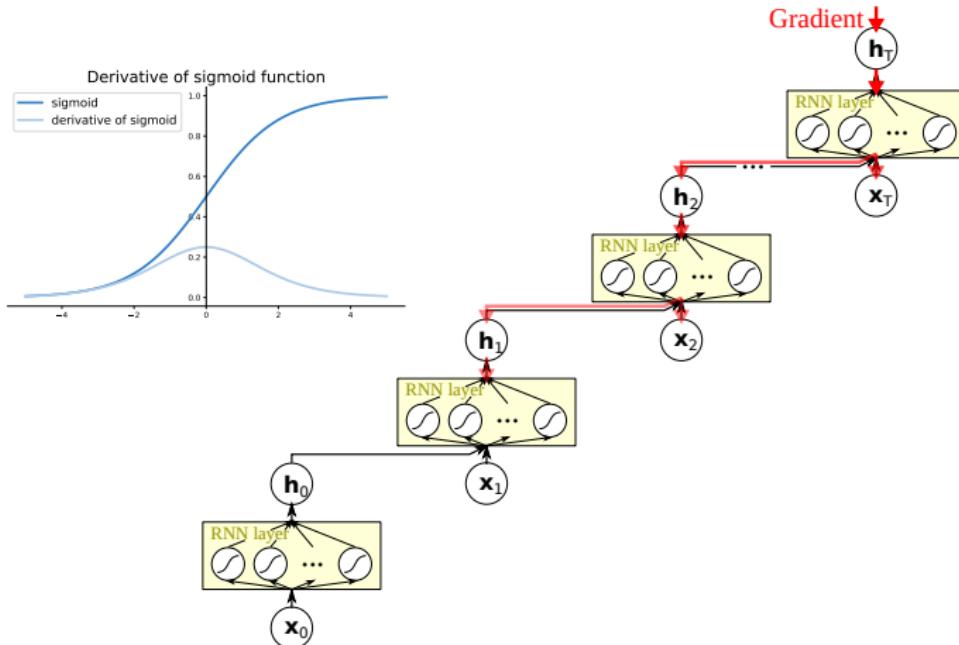
# Vanishing Gradients Problem

- BPTT generates very deep networks ( $T \cong \text{depth}$ )  
→ Vanishing or Exploding Gradients (Hochreiter, 1991)



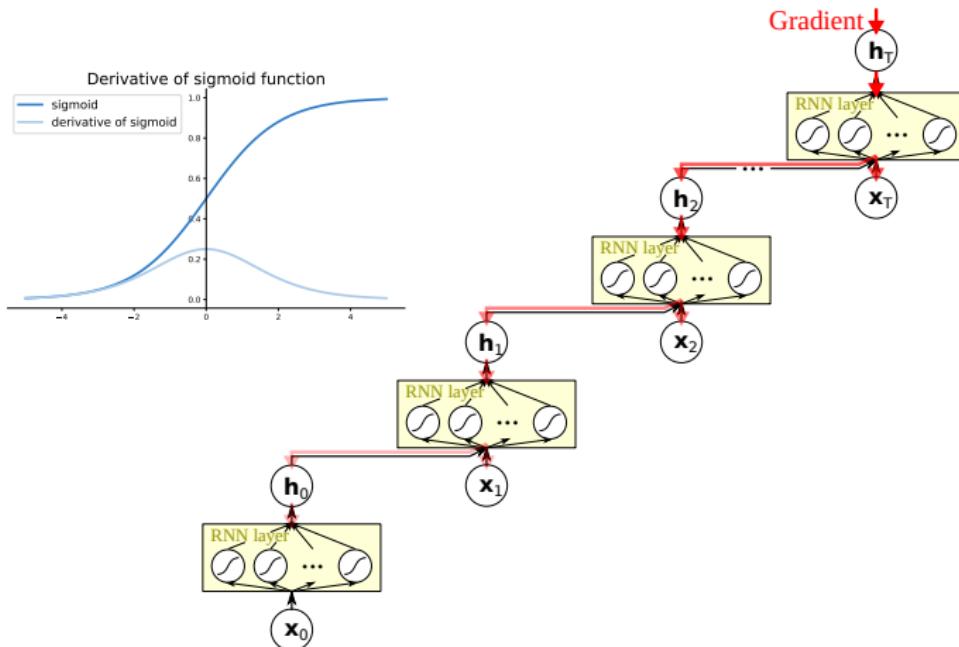
# Vanishing Gradients Problem

- BPTT generates very deep networks ( $T \cong \text{depth}$ )  
→ Vanishing or Exploding Gradients (Hochreiter, 1991)



# Vanishing Gradients Problem

- BPTT generates very deep networks ( $T \cong \text{depth}$ )  
→ Vanishing or Exploding Gradients (Hochreiter, 1991)



## Vanishing Gradients Problem - Consequences

- RNNs tend to forget events that happened a long time ago
- Learning **long-term dependencies** depends on the recurrent weights
  - If  $|f'| < 1$ , we will forget things over time
  - If  $|f'| > 1$ , our system is unstable
  - we would need  $|f'| = 1$

# **Long Short-Term Memory (LSTM)**

# Long Short-Term Memory (LSTM)

- **Idea:** Store information indefinitely but be selective about what to store

# Long Short-Term Memory (LSTM)

- **Idea:** Store information indefinitely but be selective about what to store
- **Solution:**
  1. Integrator
    - add up information over time
    - store information indefinitely
    - identity function, no vanishing gradients

# Long Short-Term Memory (LSTM)

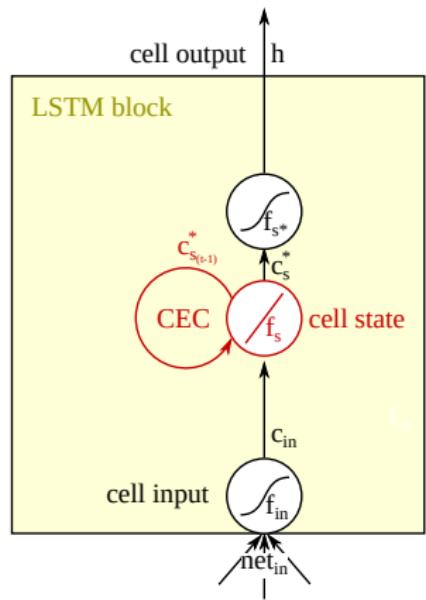
- **Idea:** Store information indefinitely but be selective about what to store
- **Solution:**
  1. Integrator
    - add up information over time
    - store information indefinitely
    - identity function, no vanishing gradients
  2. Gates
    - hidden units
    - activations multiplied with input
    - write-access (remembering)
    - read-access (communicating)
    - reset (forgetting)

# Long Short-Term Memory (LSTM)

- **Idea:** Store information indefinitely but be selective about what to store
- **Solution:**
  1. Integrator
    - add up information over time
    - store information indefinitely
    - identity function, no vanishing gradients
  2. Gates
    - hidden units
    - activations multiplied with input
    - write-access (remembering)
    - read-access (communicating)
    - reset (forgetting)
- This system is called Long Short-Term Memory (LSTM)  
(Hochreiter & Schmidhuber, 1997)

# Long Short-Term Memory (LSTM)

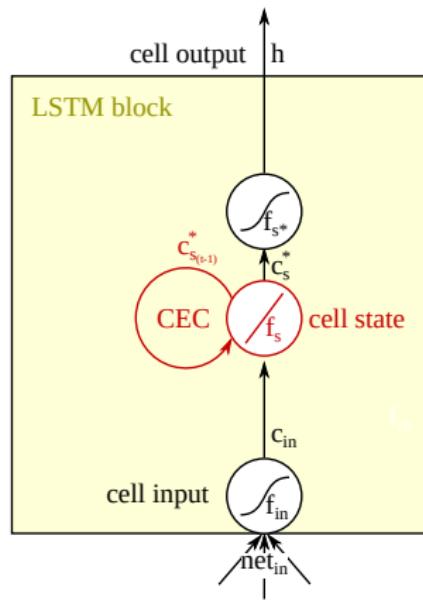
## Constant Error Carousel (CEC)



# Long Short-Term Memory (LSTM)

## Constant Error Carousel (CEC)

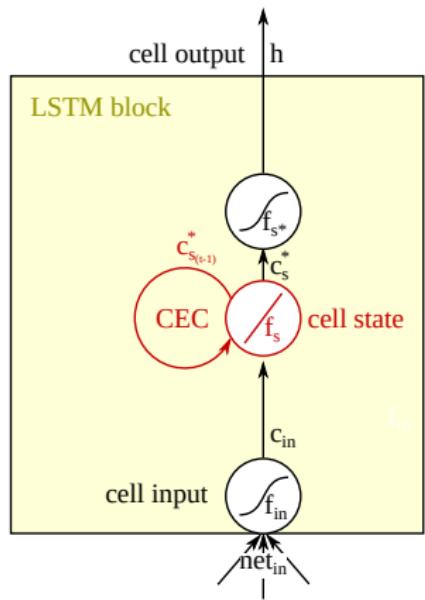
- New information  $net_{in}$  is squashed to scalar  $c_{in}$  via function  $f_{in}$



# Long Short-Term Memory (LSTM)

## Constant Error Carousel (CEC)

- New information  $net_{in}$  is squashed to scalar  $c_{in}$  via function  $f_{in}$
- New cell state:  
 $c_{st}^* = c_{s(t-1)}^* + c_{int}$   
⇒ simple integrator, no vanishing gradients!



# Long Short-Term Memory (LSTM)

## Constant Error Carousel (CEC)

- New information  $net_{in}$  is squashed to scalar  $c_{in}$  via function  $f_{in}$

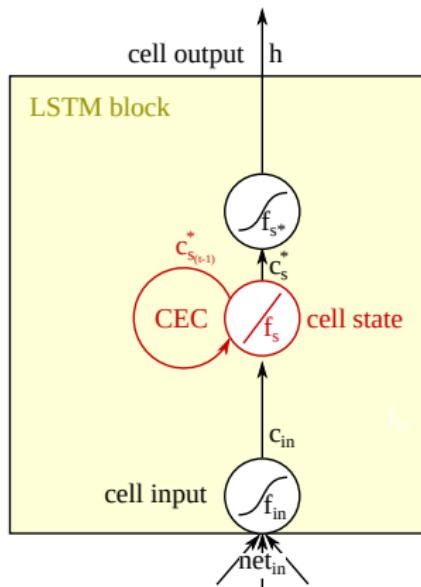
- New cell state:

$$c_{s_t}^* = c_{s_{(t-1)}}^* + c_{int}$$

⇒ simple integrator, no vanishing gradients!

- $f_{in}$ : e.g.  $\tanh$

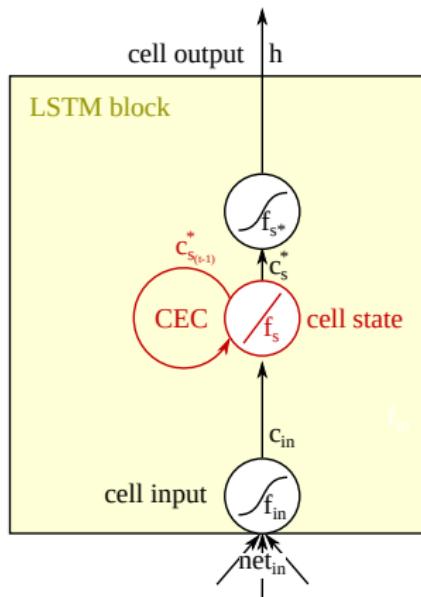
- $f_{s^*}$ : e.g.  $\tanh$ , linear



# Long Short-Term Memory (LSTM)

## More terminology

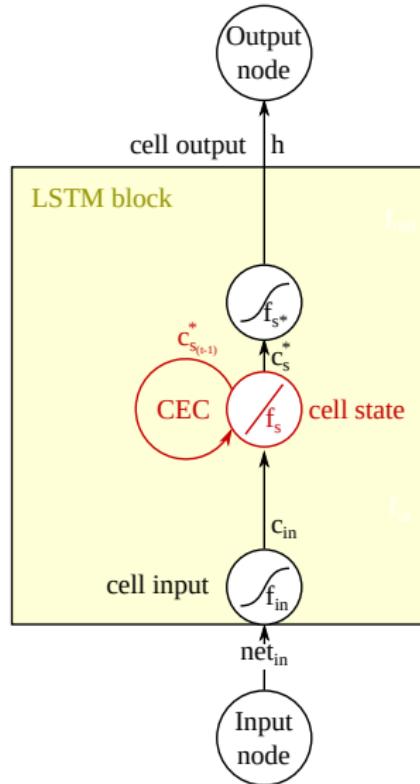
- CEC and gates constitute LSTM block or LSTM unit
- Cell output  $h$  (hidden state) is output of LSTM block
- Multiple LSTM blocks in one layer are referred to as LSTM layer



# Long Short-Term Memory (LSTM)

## More terminology

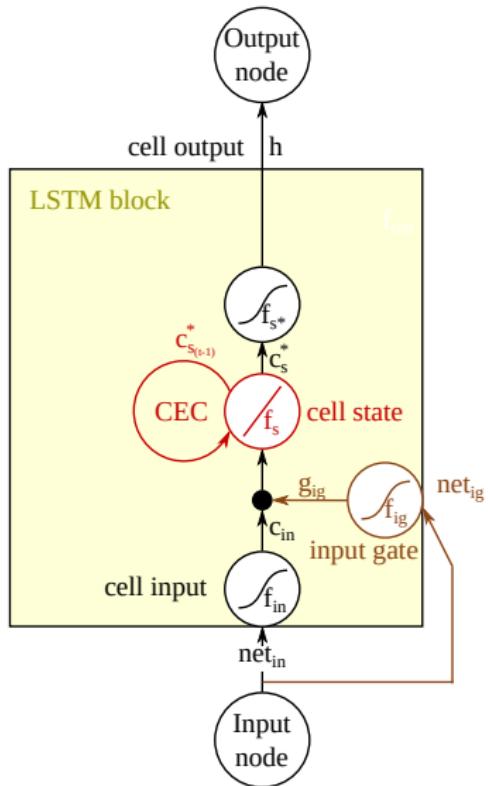
- CEC and gates constitute LSTM block or LSTM unit
- Cell output  $h$  (hidden state) is output of LSTM block
- Multiple LSTM blocks in one layer are referred to as LSTM layer



# Long Short-Term Memory (LSTM)

## Input gate

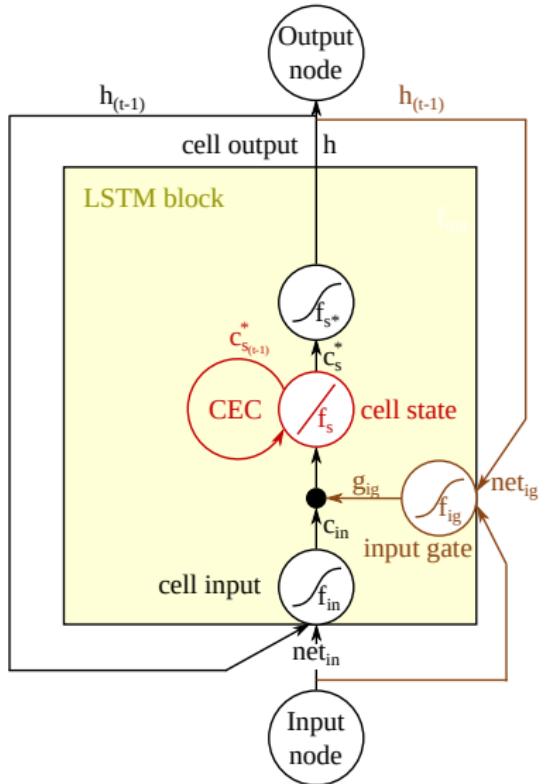
- Input gate serves as gating/attention mechanism
- $c_{in}$  is multiplied by input gate activation  $g_{ig}$  before entering CEC
- $f_{ig}$ : e.g. sigmoid



# Long Short-Term Memory (LSTM)

## Recurrent hidden state

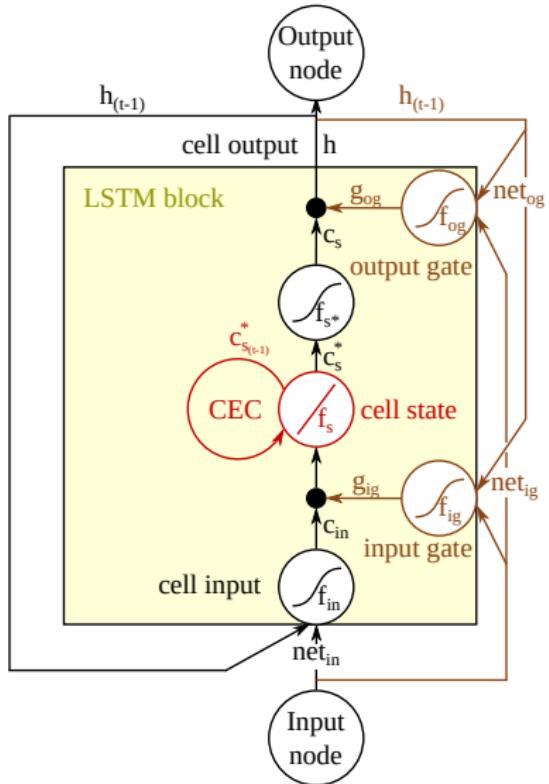
- Input gate and cell input may receive old hidden state  $h_{(t-1)}$  as recurrent input
- In an LSTM layer, the hidden states of all LSTM blocks are the recurrent input per block
- But: fully connected LSTM might not always be the best way to go!



# Long Short-Term Memory (LSTM)

## Output gate

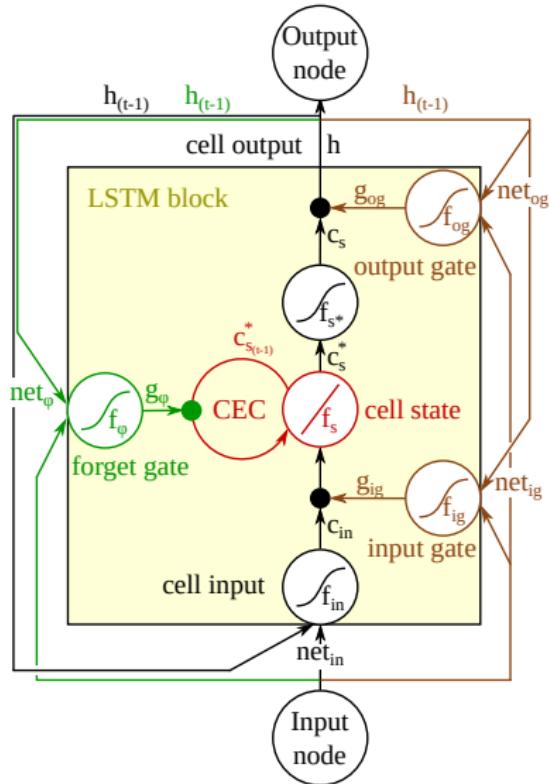
- Output gate mechanism analogous to input gate
- Output gate controls if cell state  $c_s$  is visible to rest of network
- $f_{og}$ : e.g. sigmoid



# Long Short-Term Memory (LSTM)

## Forget gate

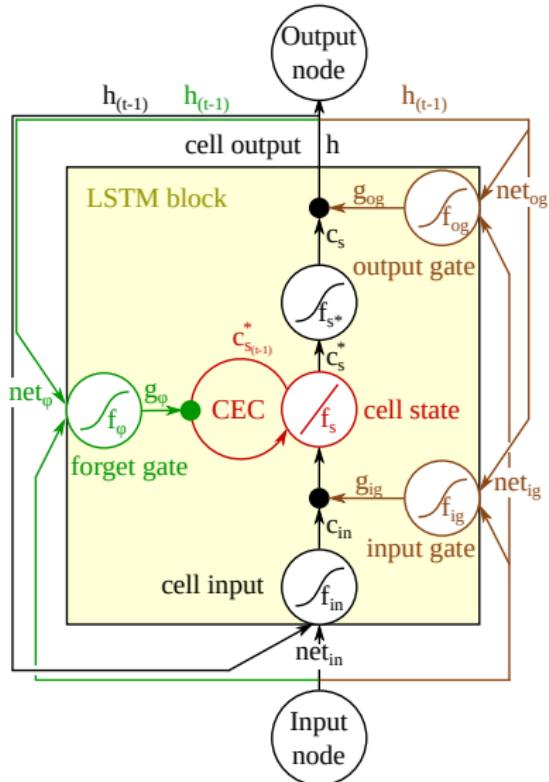
- Forget gate mechanism analogous to other gates
- Can reset or decrease CEC content
- $f_\varphi$ : e.g. sigmoid



# Long Short-Term Memory (LSTM)

## Forget gate

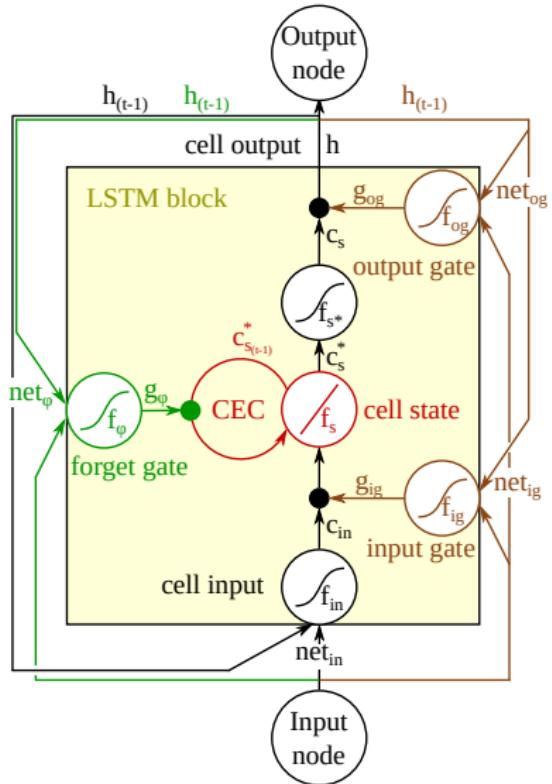
- Forget gate mechanism analogous to other gates
  - Can reset or decrease CEC content
  - $f_\varphi$ : e.g. sigmoid
- ⇒ Problem: this re-introduces vanishing gradients! Only use if necessary!



# Long Short-Term Memory (LSTM)

## Learning behavior

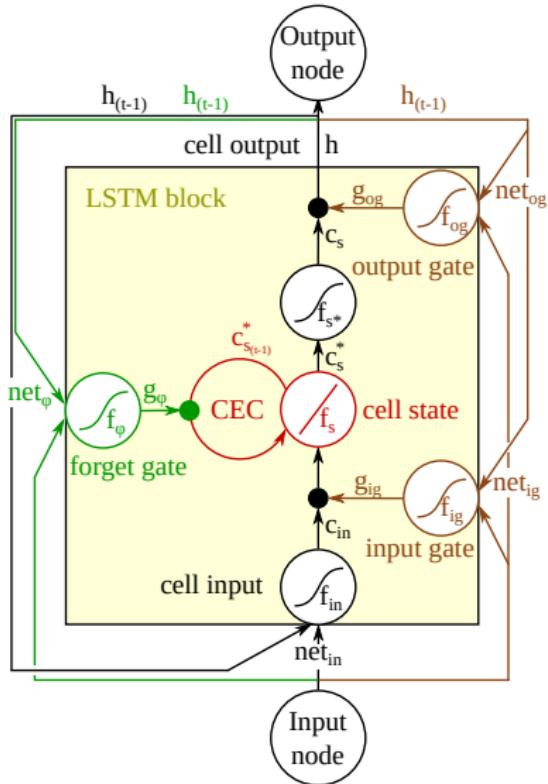
- LSTM core (CEC) is an integrator
- Gates introduce complex dynamics
- LSTM blocks (de)activate and complement each other dynamically



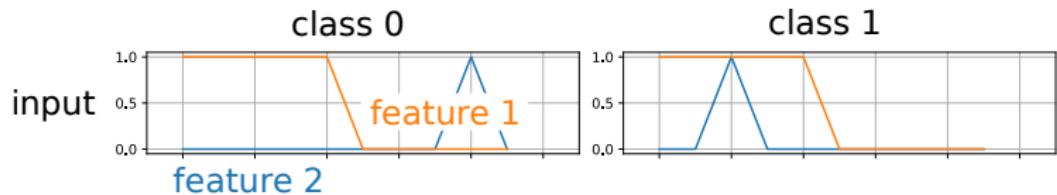
# Long Short-Term Memory (LSTM)

## Tricks of the trade

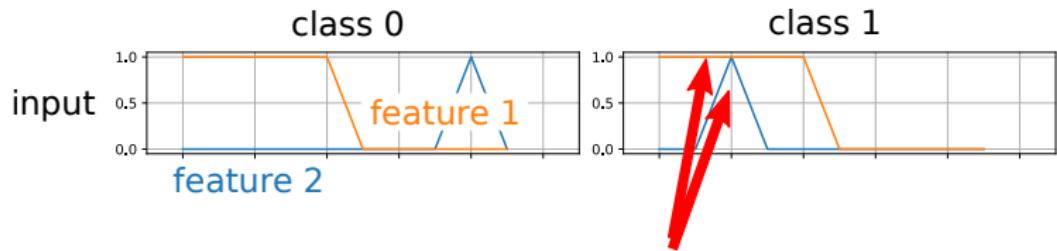
- Plot your LSTM cell- and hidden states
- Fully connected LSTM not always needed
- Negative input gate bias helps for long sequences
- Use forget gate only if necessary



# LSTM example: Task description

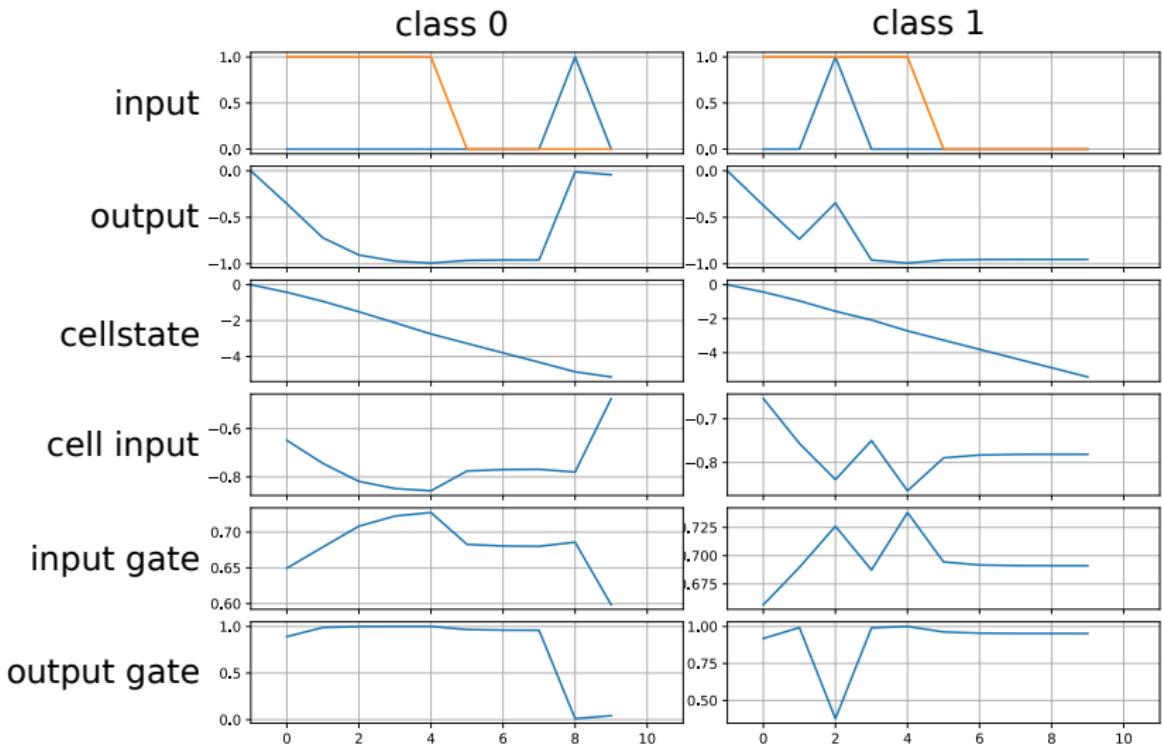


# LSTM example: Task description

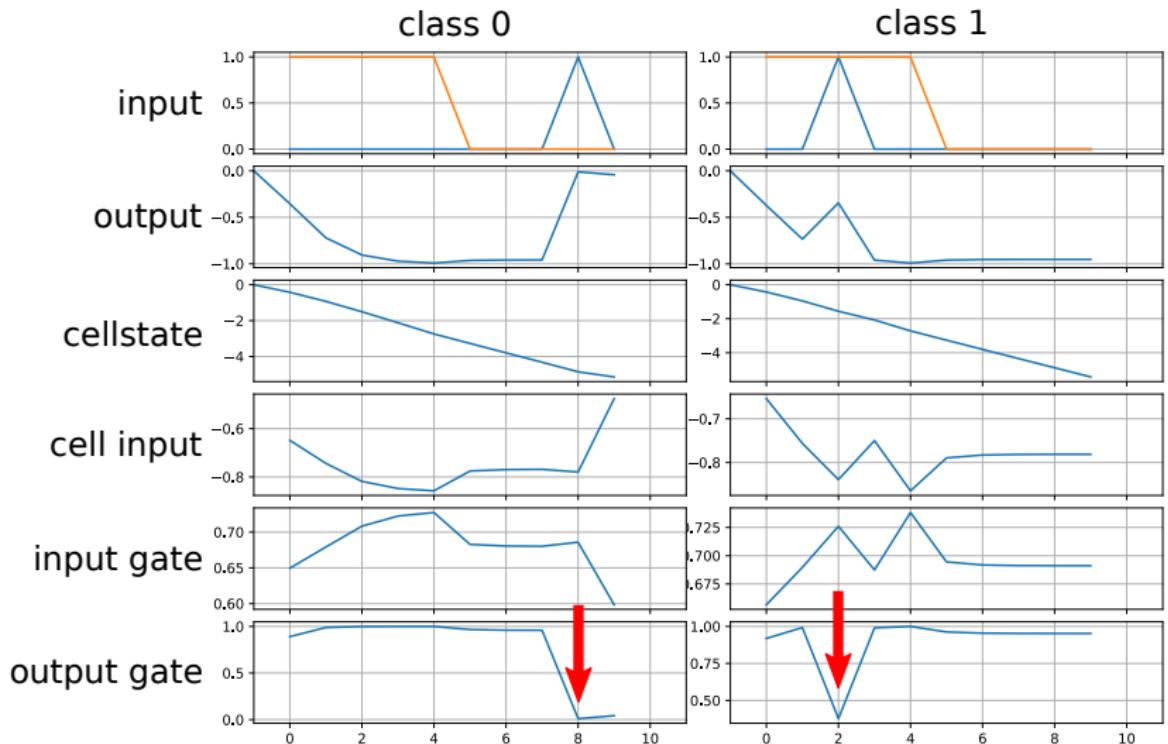


class 1: feature 1 and feature 2 active at same time

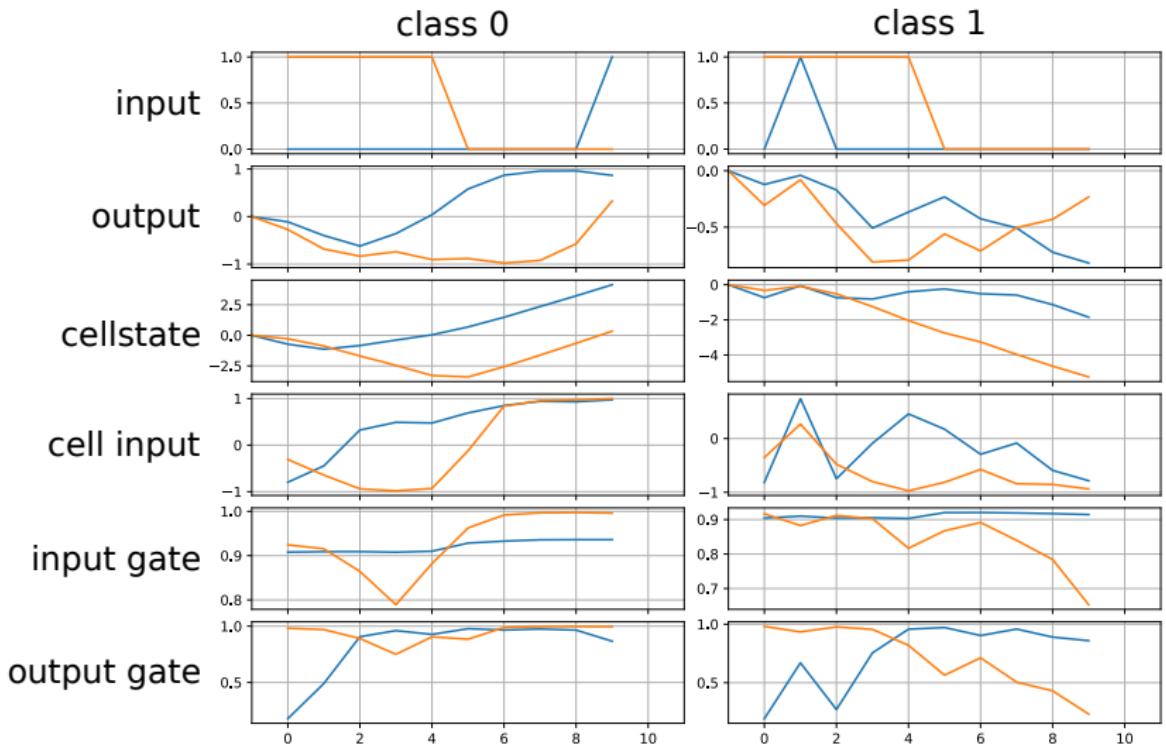
# LSTM example: 1 LSTM (fully connected)



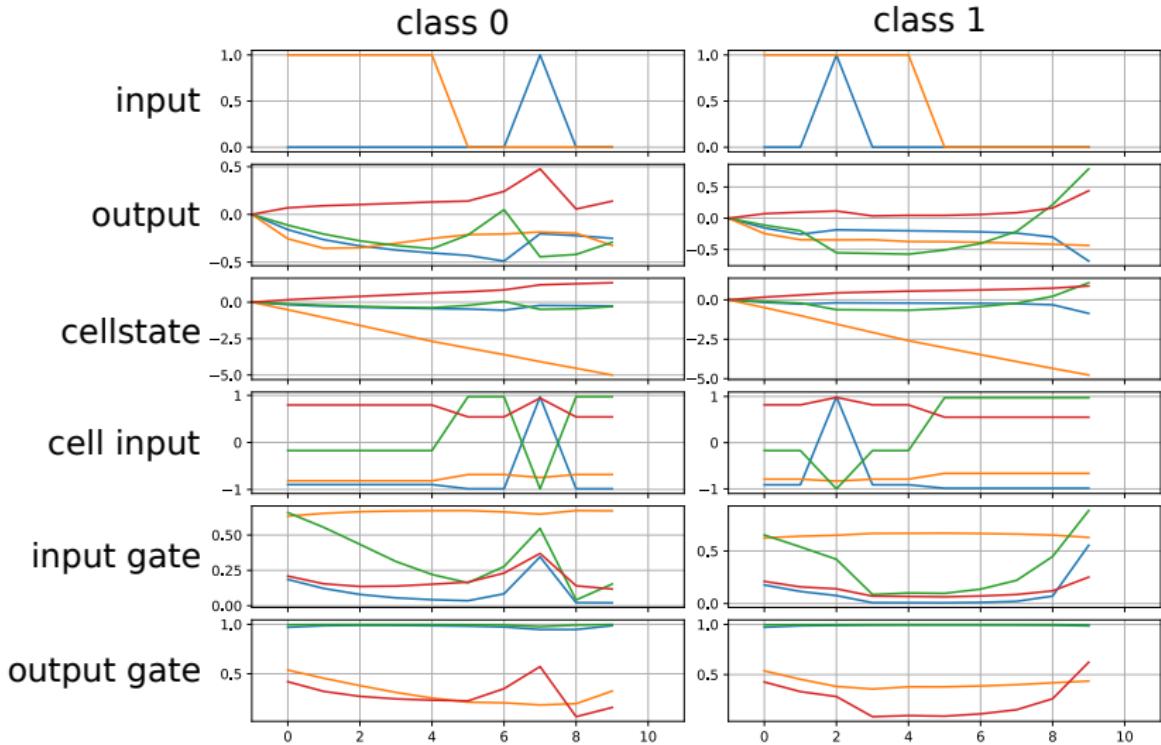
# LSTM example: 1 LSTM (fully connected)



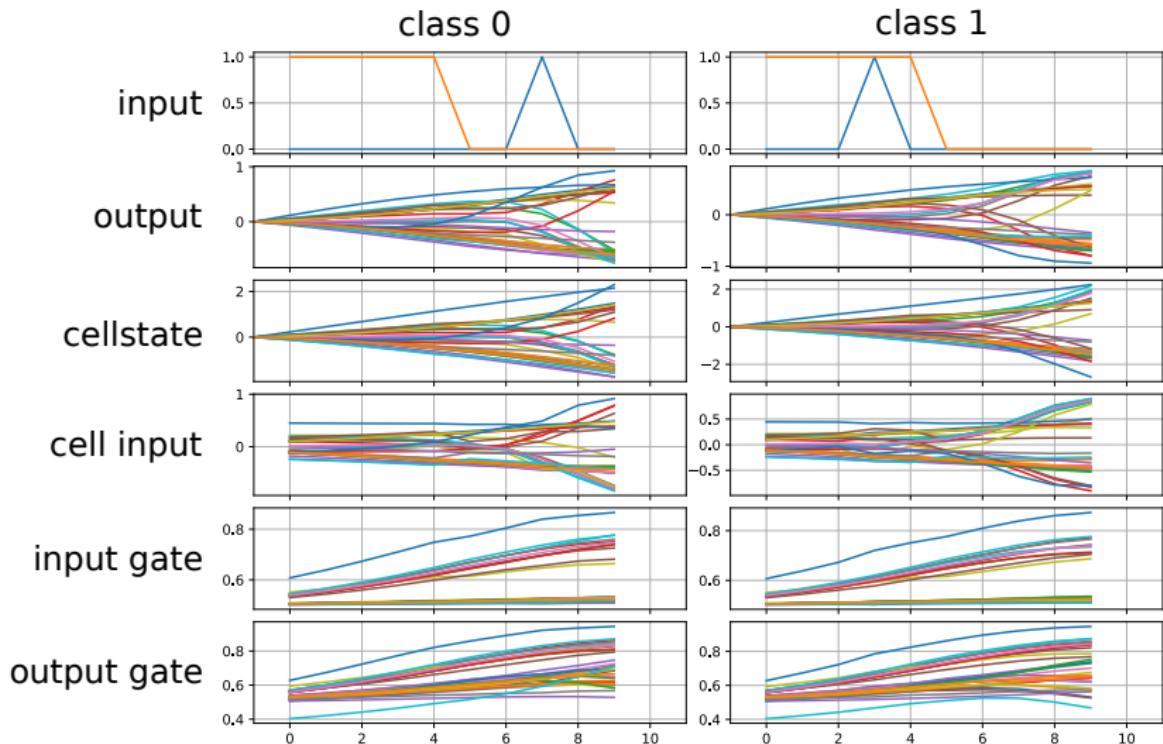
# LSTM example: 2 LSTM (fully connected)



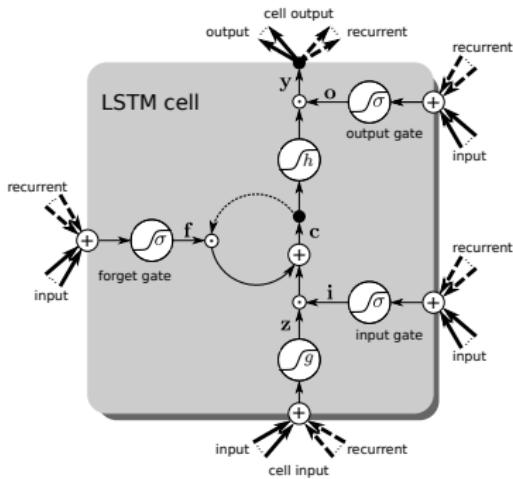
# LSTM example: 4 LSTM (not fully connected)



# LSTM example: 32 LSTM (fully connected)



# LSTM Formulas



## Legend

- feedforward data flow
- ↔ recurrent data flow
- feedforward weights
- ↔ recurrent weights
- branching point
- multiplication
- ⊕ sum over all inputs
- ( $\sigma$ ) gate activation function (usually sigmoid)
- ( $\circ$ ) input activation function (usually tanh or sigmoid)
- ( $h$ ) output activation function (usually tanh or sigmoid)

$$z^t = g(W_z x^t + R_z y^{t-1} + b_z)$$

$$i^t = \sigma(W_i x^t + R_i y^{t-1} + b_i)$$

$$f^t = \sigma(W_f x^t + R_f y^{t-1} + b_f)$$

$$c^t = i^t \odot z^t + f^t \odot c^{t-1}$$

$$o^t = \sigma(W_o x^t + R_o y^{t-1} + b_o)$$

$$y^t = o^t \odot h(c^t)$$

cell input

input gate

forget gate

cell state

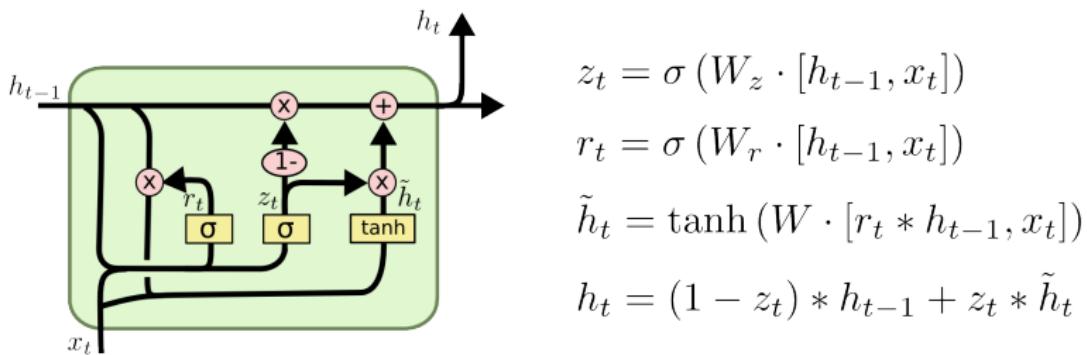
output gate

cell output

# Gated Recurrent Units (GRUs)

# Gated Recurrent Units (GRUs)

- Reduced LSTM with merged gates (Cho et al, 2014)
- Suffers from Vanishing Gradients (always forgets)
- Less parameters, easier to use, lower complexity



[Image source: <http://colah.github.io/posts/2015-08-Understanding-LSTMs/>]

# LSTM Applications

# LSTM Applications

- LSTM can effectively learn long-term dependencies
- One of the most-used models today
- State of the Art in many applications
  - Speech/Text generation and recognition
  - Amino acid sequence classification
  - Time-Series classification/generation
  - ...



# Handwriting Generation

from his travels it might have been

from his travels - it might have been

more of national temperament

[Generating Sequences With Recurrent Neural Networks, A. Graves, arxiv 2013]

Online interactive example: <https://www.cs.toronto.edu/~graves/handwriting.html>

# Source Code Generation

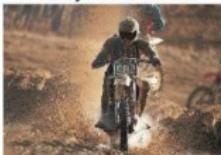
```
static void do_command(struct seq_file *m, void *v)
{
    int column = 32 << (cmd[2] & 0x80);
    if (state)
        cmd = (int)(int_state ^ (in_8(&ch->ch_flags) & Cmd) ? 2 : 1);
    else
        seq = 1;
    for (i = 0; i < 16; i++) {
        if (k & (1 << 1))
            pipe = (in_use & UMXTHREAD_UNCCA) +
                ((count & 0x00000000fffffff8) & 0x000000f) << 8;
        if (count == 0)
            sub(pid, ppc_md.kexec_handle, 0x20000000);
        pipe_set_bytes(i, 0);
    }
    /* Free our user pages pointer to place camera if all dash */
    subsystem_info = &of_changes[PAGE_SIZE];
    rek_controls(offset, idx, &soffset);
    /* Now we want to deliberately put it to device */
    control_check_polarity(&context, val, 0);
    for (i = 0; i < COUNTER; i++)
        seq_puts(s, "policy ");
}
```

[The Unreasonable Effectiveness of Recurrent Neural Networks, A. Karpathy, 2015]

Many more examples: <http://karpathy.github.io/2015/05/21/rnn-effectiveness/>

# Image Captioning

A person riding a motorcycle on a dirt road.



Two dogs play in the grass.



A skateboarder does a trick on a ramp.



A dog is jumping to catch a frisbee.



A group of young people playing a game of frisbee.



Two hockey players are fighting over the puck.



A little girl in a pink hat is blowing bubbles.



A refrigerator filled with lots of food and drinks.



A herd of elephants walking across a dry grass field.



A close up of a cat laying on a couch.



A red motorcycle parked on the side of the road.



A yellow school bus parked in a parking lot.



Describes without errors

Describes with minor errors

Somewhat related to the image

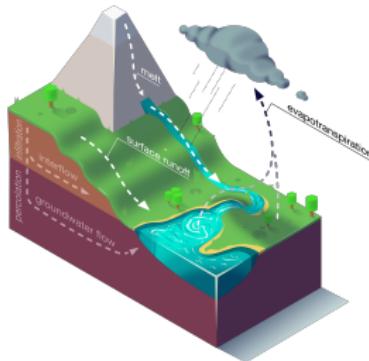
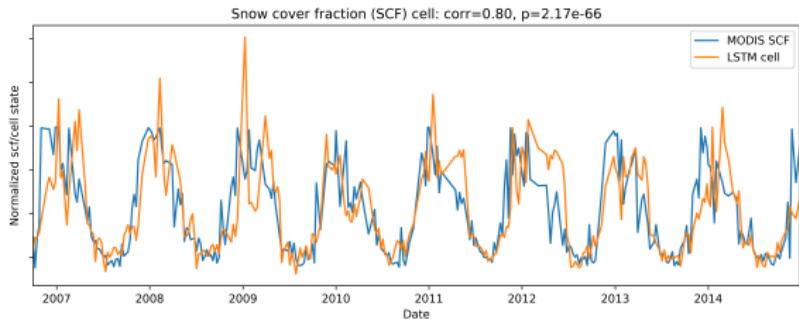
Unrelated to the image

# Language Translation

Type	Sentence
<b>Our model</b>	Ulrich UNK , membre du conseil d' administration du constructeur automobile Audi , affirme qu' il s' agit d' une pratique courante depuis des années pour que les téléphones portables puissent être collectés avant les réunions du conseil d' administration afin qu' ils ne soient pas utilisés comme appareils d' écoute à distance .
<b>Truth</b>	Ulrich Hackenberg , membre du conseil d' administration du constructeur automobile Audi , déclare que la collecte des téléphones portables avant les réunions du conseil , afin qu' ils ne puissent pas être utilisés comme appareils d' écoute à distance , est une pratique courante depuis des années .
<b>Our model</b>	" Les téléphones cellulaires , qui sont vraiment une question , non seulement parce qu' ils pourraient potentiellement causer des interférences avec les appareils de navigation , mais nous savons , selon la FCC , qu' ils pourraient interférer avec les tours de téléphone cellulaire lorsqu' ils sont dans l' air " , dit UNK .
<b>Truth</b>	" Les téléphones portables sont véritablement un problème , non seulement parce qu' ils pourraient éventuellement créer des interférences avec les instruments de navigation , mais parce que nous savons , d' après la FCC , qu' ils pourraient perturber les antennes-relais de téléphonie mobile s' ils sont utilisés à bord " , a déclaré Rosenker .
<b>Our model</b>	Avec la crémation , il y a un " sentiment de violence contre le corps d' un être cher " , qui sera " réduit à une pile de cendres " en très peu de temps au lieu d' un processus de décomposition " qui accompagnera les étapes du deuil " .
<b>Truth</b>	Il y a , avec la crémation , " une violence faite au corps aimé " , qui va être " réduit à un tas de cendres " en très peu de temps , et non après un processus de décomposition , qui " accompagnerait les phases du deuil " .

Table 3: A few examples of long translations produced by the LSTM alongside the ground truth translations. The reader can verify that the translations are sensible using Google translate.

# Hydrology Forecasts



[Sequence to Sequence Learning with Neural Networks, Kratzert & Herrnegger & Klotz & Hochreiter & Klambauer]

# Summary

# Summary

- Recurrent Neural Networks (RNNs):
  - Can handle sequence data of variable length
  - Turing Complete
  - Vanishing Gradients problem
  
- Long Short-Term Memory (LSTM):
  - Integrator with gating mechanisms
  - Solve Vanishing Gradients problem
  - State of the Art for sequence-like data