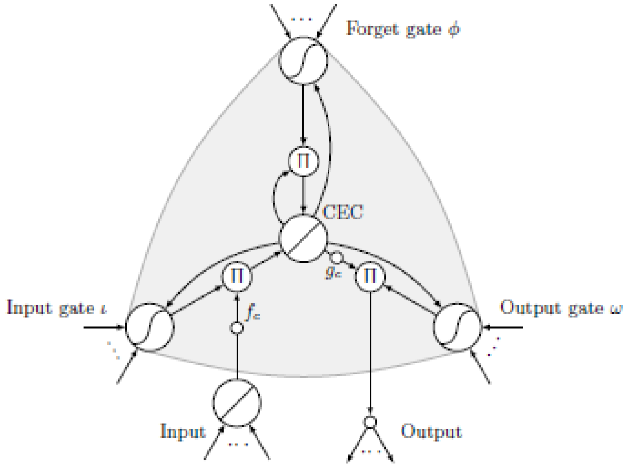
EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

FACULTY OF
SCIENCE

Institute of Computer Science
Chair of Cognitive Modeling



Advanced Neural Networks

3. Long Short Term Memory RNNs (i.e. LSTMs)

Martin V. Butz
Tutoren / Co-Dozenten: Sebastian Otte & Fabian Schrodt

RNNs versus LSTMs

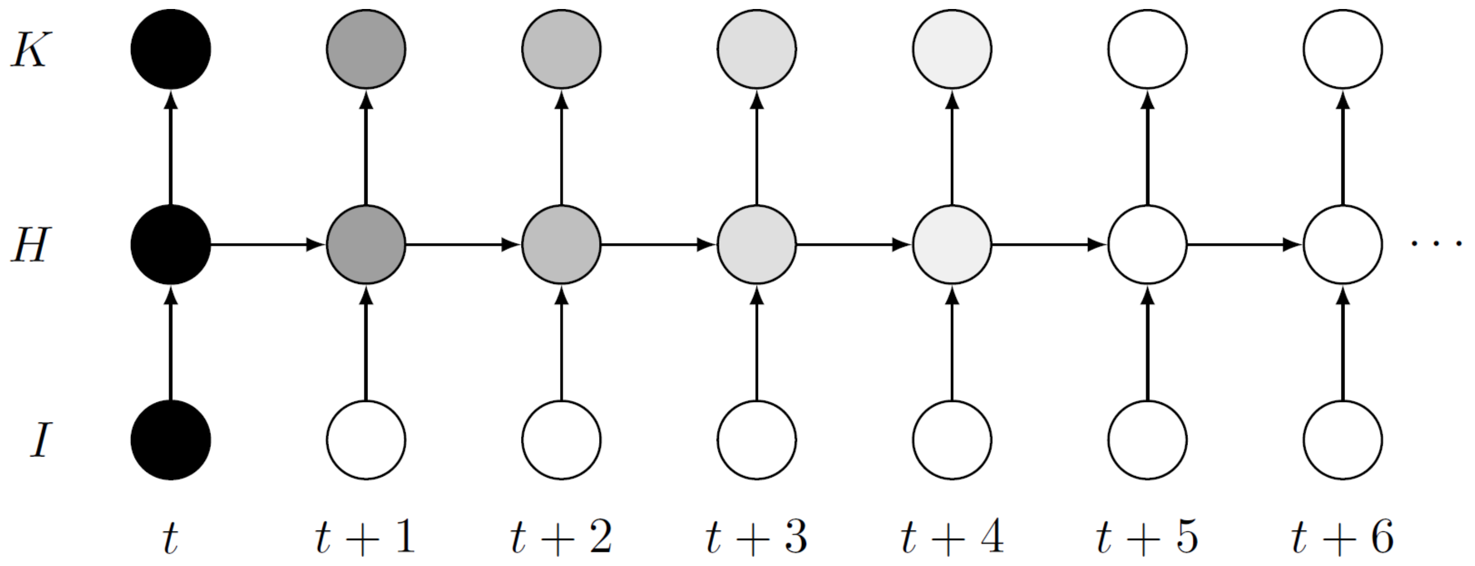
- Recurrent Neural Networks are great when
  - Learning to reproduce time series
  - Classifying dynamic data
  - Extracting systematic patterns out of time series.
- RNNs are Turing Complete (!) - over a compact interval.
  - ➢ That is, they can map between arbitrary sequences!
- However, standard RNNs have **problems** when
  - there are “deep” dependencies in the data – those where past inputs influence outputs in the farther distant future.
- This problem is referred to as the **“vanishing gradient”** problem.
- What are examples of such problems:
  - Grammar Problems:
    - e.g.: ($a^n b^n c^n$) where n is unknown
    - In general, any kind of counting and respective reproduction problem.
  - Also “memorized context problems” (as I characterize them) are hard… e.g.:
    - Push a switch, go somewhere else, benefit from having pushed the switch.
    - Notice a signal, proceed, use the memorized signal for further decision making.
  - That is, any kind of problem where a **long short-term memory** is useful, that is, a short-term memory that persists sufficiently long over time.

The Vanishing Gradient Problem

- The Vanishing Gradient Problem:
  When RNNs are trained with backpropagation, the error gradient from error signals in the future quickly vanishes when projected into the past.
  - It **vanishes exponentially** with respect to the weights over time.

- Consequence:
  - The time-horizon affected by an error is very small (T < 10).
  - Input events that are temporally widely spaced cannot be combined effectively.
  - Local error signals always overrule further backpropagated error signals.

- Note: The same problem also generally holds for deep feed-forward networks.

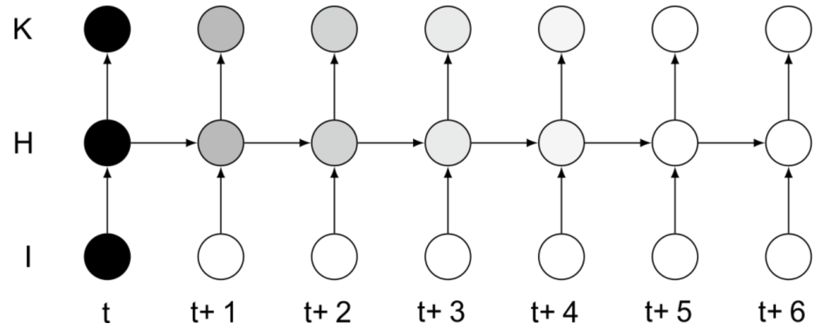


Vanishing Gradient Directions

Forward Direction

- Information from the past can only influence few steps into the future.
  - Past is quickly forgotten, i.e. information is “washed-out”.
  - Past information cannot be acquired / recalled several time-steps ahead.

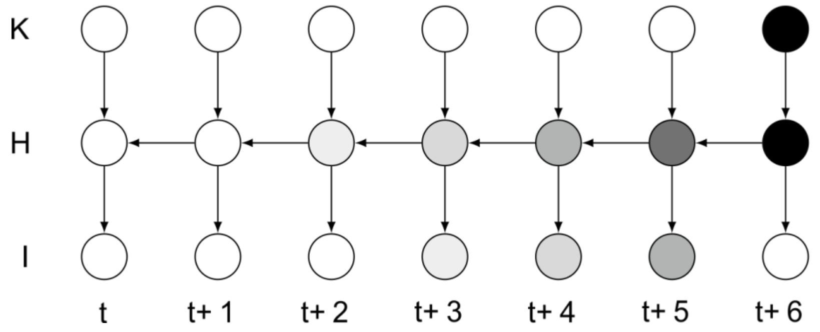
➢Memory is a quickly diffusing short-term memory.



Backward Direction

- Errors from the future can only influence the recent past.
  - Future errors have limited adaptation influence.
  - Errors do not affect the more distant past.

➢Network does not learn to minimize further-reaching (in time) errors.



Strategies to Overcome the V.G. Problem

- Extending the input layer feeding in several time steps at once.
  - Downside: Quick growth of the network!
  - ➢ Can help, but is limited to the number of time steps (and few beyond that) considered.
- Time-delayed recurrent neural networks
  - Recurrences that affect the network state after several time steps e.g. 1, 5, 10 etc…
  - ➢ Can help but not very significantly.
- Pre-compressing the input sequence temporally.
  - E.g. using temporal kernels
  - Works for the types of temporal compressions considered.
  - Rather limited because strong dependence on types of compressions considered.
- Adding multiplicative memory units → LSTMs!

LSTMs

- Proposed in:
  Hochreiter, S. & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation, 9*, 1735-1780.

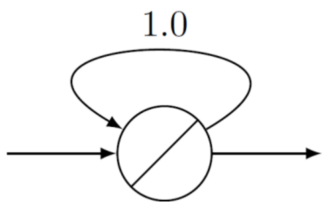
- LSTM overcomes the Vanishing Gradient Problem introducing differentiable “memory cells”.

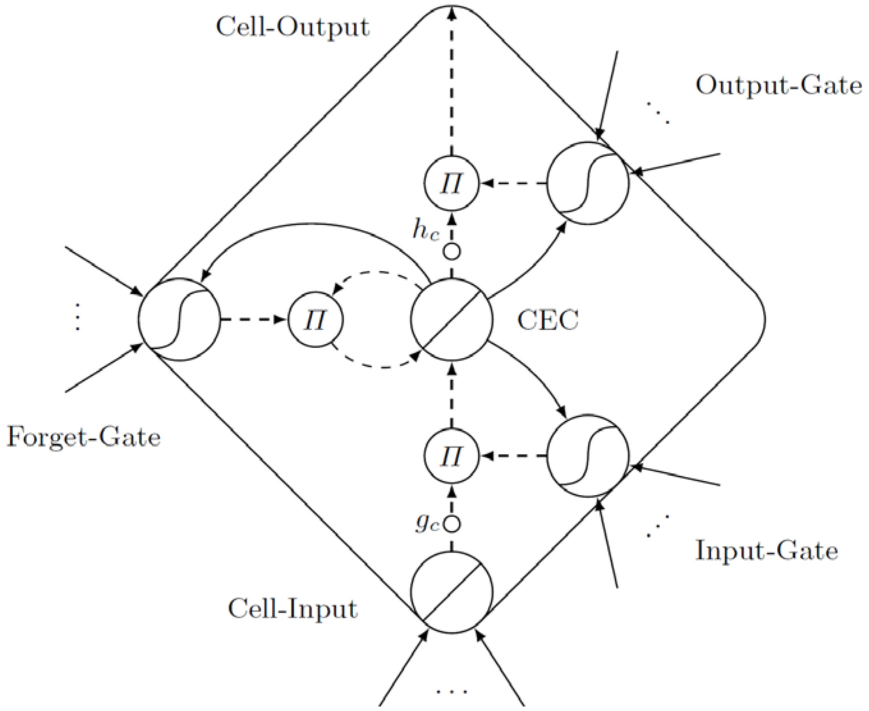
- Fundamental ideas:
  - Use a self-recurrent linear cell with constant weight one (called the Constant Error Carousel, CEC).
  - Gate the CEC with multiplicative gates yielding a memory cell.
  - Combine and integrate several of such memory cells into a complete LSTM with additional input and output layers.

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

A Memory Cell

- Constant Error Carousel:
  - Enables to loop the error back in time arbitrarily long (in principle).
  - Note however that numerical instabilities need to be avoided during such computations.
- Multiplicative gates “protect” the CEC from outer influences.
  - The gates essentially control the inflow and outflow of information to and from the CEC, respectively…
    - Gates closed: Memory is maintained without disruption.
    - Input gate open: Memory is modified / set to new input accordingly.
    - Output gate open: Memory state is read-out and influences further processing / output generation.
    - Forget gate open: Memory state may be forgotten.
  - Thus, gates essentially enable read, write, and reset of the memory cell.





Dotted connections: Constant with weight=1

Development of LSTMs over the last two decades

- Hochreiter, S. & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation, 9*, 1735-1780.
  - Originally trained with truncated gradient (only errors within CECs flow back in time).

- Gers, F. A., Schmidhuber, J. & Cummins, F. (2000). Learning to Forget: Continual Prediction with LSTM. *Neural Computation, 12*, 2451-2471.
  - Introduction of the forget gate, which modulates the memory cell state multiplicatively.

- Gers, F. A., Schraudolph, N. & Schmidhuber, J. (2002). Learning precise timing with LSTM recurrent networks. *Journal of Machine Learning Research, 3*, 115-143.
  - Trainable connections from the internal state to the gates.
  - Peephole connections (trainable weights; error does no affect internal memory state).

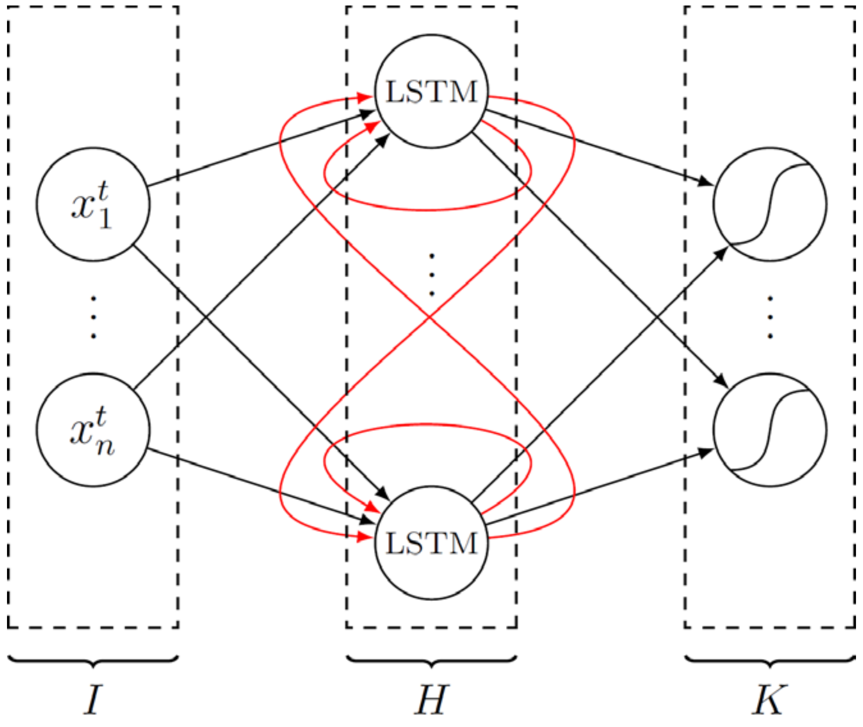
- Graves, A. & Schmidhuber, J. (2005). Framewise Phoneme Classification with Bidirectional LSTM and Other Neural Network Architectures. *Neural Networks, 18*, 602-610.
  - Computing the exact gradient with BPTT.

- Otte, S., Liwicki, M. & Zell, A. (2014). Dynamic Cortex Memory: Enhancing Recurrent Neural Networks for Gradient-Based Sequence Learning. In Stefan Wermter et al (Eds.), *Artificial Neural Networks and Machine Learning - ICANN 2014*, LNCS 8681, pages 1-8.
  - Gate-communication infrastructure.

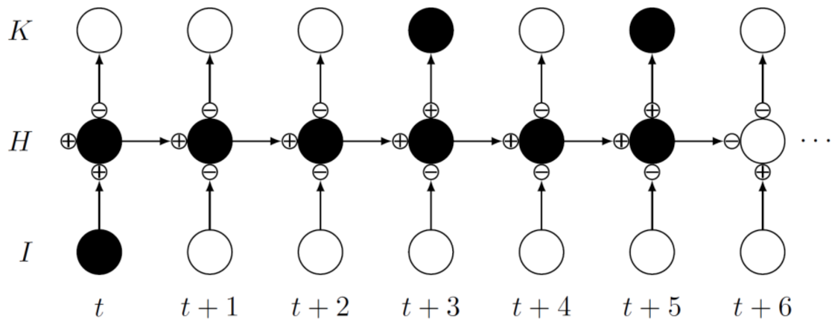
EBERHARD KARLS UNIVERSITÄT TÜBINGEN

LSTMs and the No-Longer Vanishing Gradient

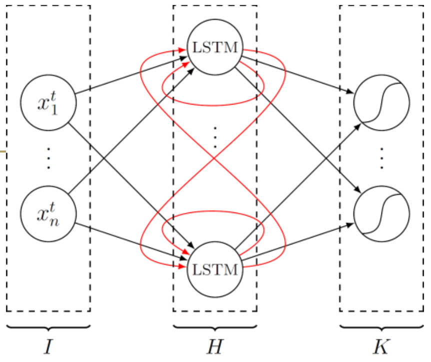
Sketch of an LSTM-RNN



Gradient does NOT vanish any-longer



Notes on Notation…



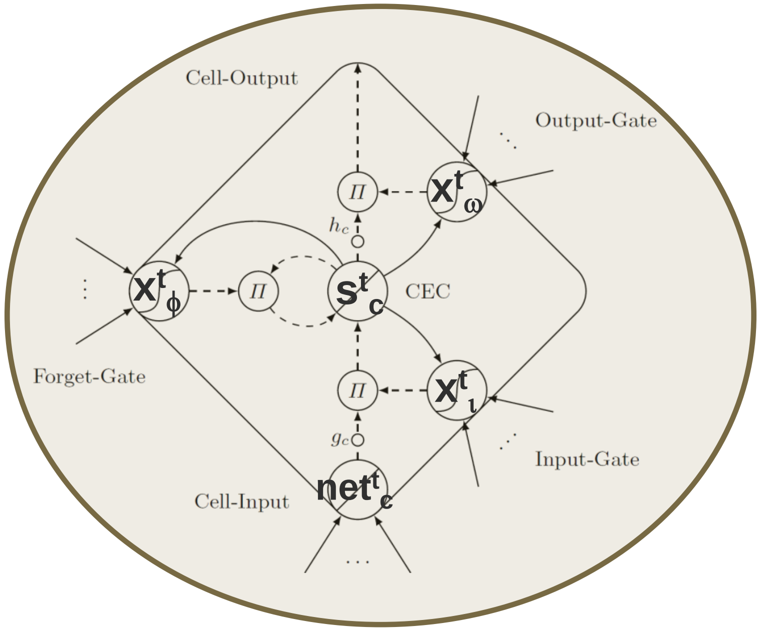
- LSTM Network consists of
  - Input layer I
  - Hidden layer H (containing multiple units including memory cell(s))
  - A memory cell with D CECs (indexed by c).
  - An output layer K

- A neuron j in the LSTM:
  - Activation function: φ_j
  - Current net activation: net^t_j
  - Current output activity: x^t_j

- A CEC c:
  - Input squashing function: g_c
  - Output squashing function: h_c
  - Current internal state: s^t_c

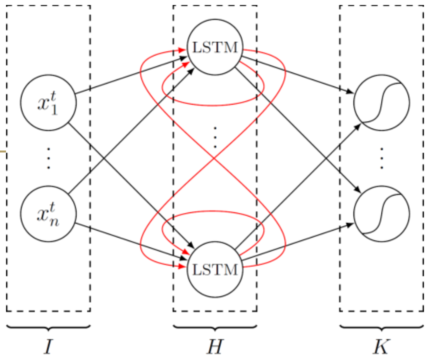
- At the beginning:
  - All x^t_j and s^t_c are set to zero at time t=0.

- Time is indicated by the superscript t.



EBERHARD KARLS UNIVERSITÄT TÜBINGEN

Forward Pass Computation



- Input Gate x^t_ι:

$$net^t_\iota = \sum_{i\in I} w_{i\iota} x^t_i + \sum_{h\in H} w_{h\iota} x^{t-1}_h + \sum_{c\in D} w_{c\iota} s^{t-1}_c$$

$$x^t_\iota = \varphi_\iota(net^t_\iota)$$

- Forget Gate x^t_ϕ:

$$net^t_\phi = \sum_{i\in I} w_{i\phi} x^t_i + \sum_{h\in H} w_{h\phi} x^{t-1}_h + \sum_{c\in D} w_{c\phi} s^{t-1}_c$$

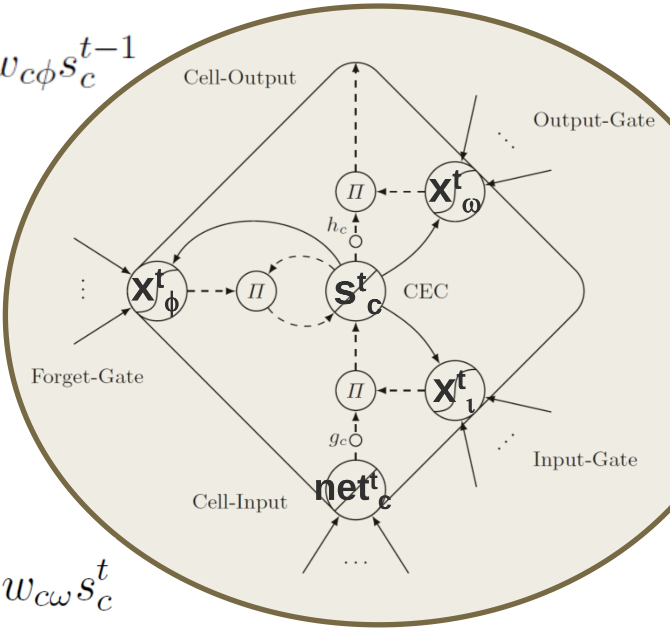
$$x^t_\phi = \varphi_\phi(net^t_\phi)$$

- Cell Input net^t_c:

$$net^t_c = \sum_{i\in I} w_{ic} x^t_i + \sum_{h\in H} w_{hc} x^{t-1}_h$$

- Cell State s^t_c:

$$s^t_c = x^t_\iota g_c(net^t_c) + x^t_\phi s^{t-1}_c$$



- Output Gate x^t_ω:

$$net^t_\omega = \sum_{i\in I} w_{i\omega} x^t_i + \sum_{h\in H} w_{h\omega} x^{t-1}_h + \sum_{c\in D} w_{c\omega} s^t_c$$

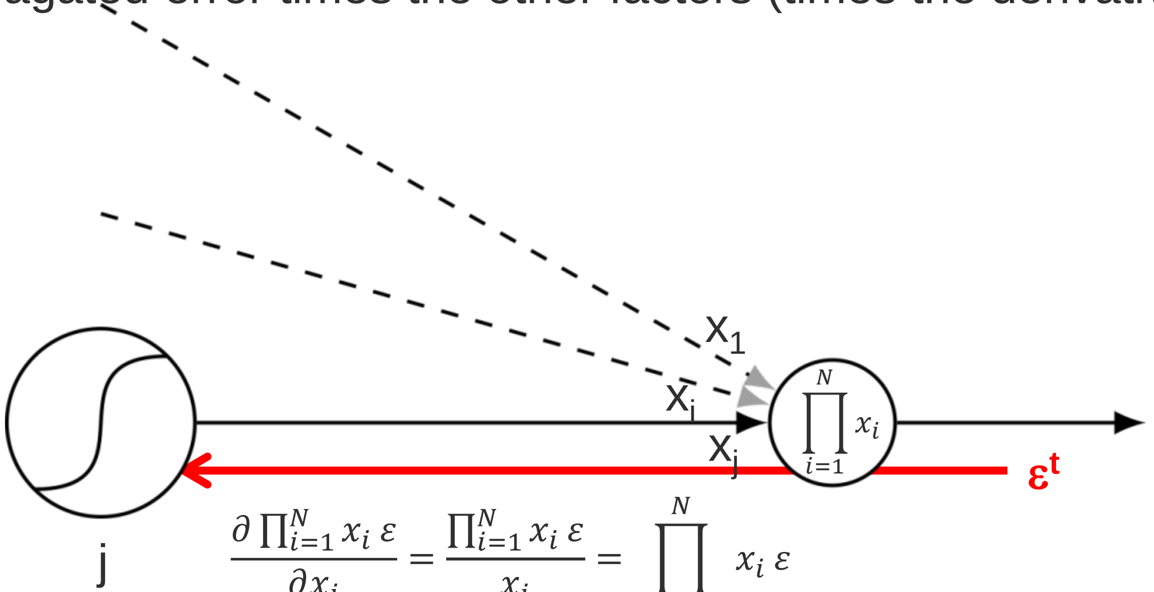
$$x^t_\omega = \varphi_\omega(net^t_\omega)$$

- Cell Output x^t_c:

$$x^t_c = x^t_\omega h_c(s^t_c)$$

Backward Pass Through Multiplicative Units

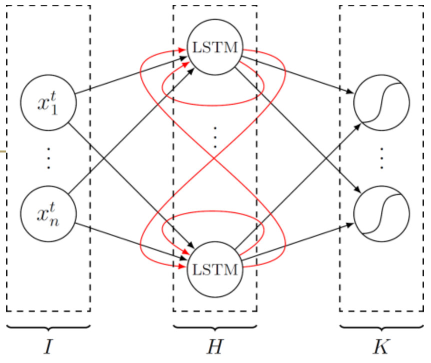
- When the error is back-propagated through multiplicative units, it is “split-up” into the different sub-units.
- Seeing that the partial derivatives are always with respect to one of the inputs into a multiplicative neuron (unit), the other units are constant factors.
  - Remember: When taking the derivative, constants are just constants !
  - Example: f(x) = ax; f’(x) = a .
- Thus: The derivative with respect to one of multiplicative factors j is the backpropagated error times the other factors (times the derivative of factor j):



$$\frac{\partial \prod_{i=1}^{N} x_i\, \varepsilon}{\partial x_j} = \frac{\prod_{i=1}^{N} x_i\, \varepsilon}{x_j} = \prod_{i=1, i\neq j}^{N} x_i\, \varepsilon$$

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

Backward Pass Computation



- Cell Output x^t_c:

$$\epsilon_c^t = \sum_{k\in K} w_{ck}\delta_k^t + \sum_{h\in H} w_{ch}\delta_h^{t+1}$$

- Output Gate x^t_ω:

$$\delta_\omega^t = \varphi'_\omega(net_\omega^t)\left[\sum_{c\in D} h_c(s_c^t)\epsilon_c^t\right]$$

- Cell State s^t_c:

$$\zeta_c^t = x_\omega^t h'_c(s_c^t)\epsilon_c^t + x_\phi^{t+1}\zeta_c^{t+1} + w_{c\iota}\delta_\iota^{t+1} + w_{c\phi}\delta_\phi^{t+1} + w_{c\omega}\delta_\omega^t$$

- Cell Input net^t_c:

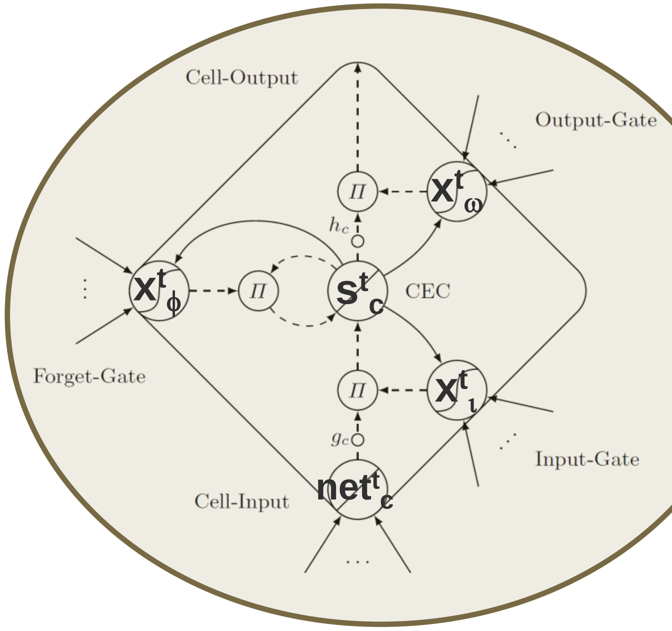
$$\delta_c^t = x_\iota^t g'_c(net_c^t)\zeta_c^t$$

- Forget Gate x^t_ϕ:

$$\delta_\phi^t = \varphi'_\phi(net_\phi^t)\left[\sum_{c\in D} s_c^{t-1}\zeta_c^t\right]$$

- Input Gate x^t_ι:

$$\delta_\iota^t = \varphi'_\iota(net_\iota^t)\left[\sum_{c\in D} g_c(net_c^t)\zeta_c^t\right]$$



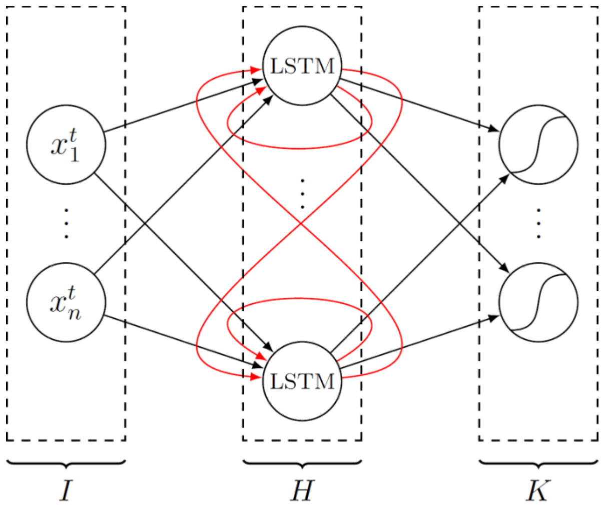
<u>Helpful definitions:</u>

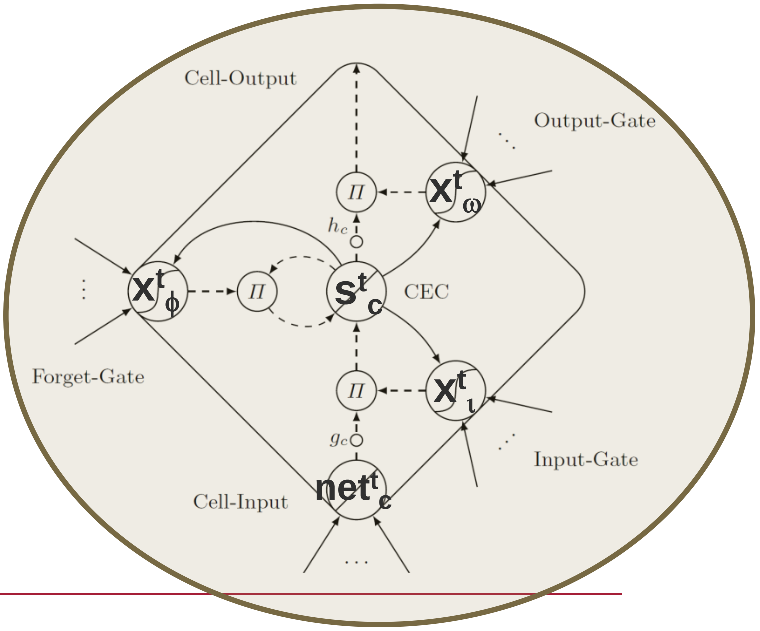
Output gradient: $\epsilon_c^t =_{\text{def}} \frac{\partial E}{\partial x_c^t}$

State gradient: $\zeta_c^t =_{\text{def}} \frac{\partial E}{\partial s_c^t}$

LSTM – Further Notes

- Forward and Backward pass should be calculated in the indicated order.
- Activation function of gates are typically standard sigmoids (thus open or closed mostly).
- Activation function of squashing functions g and h are typically hyperbolic tangents.





EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Example: Learning a Context-Sensitive Grammar

- Typical benchmark for LSTMs:
  Context-free and context-sensitive grammar
- These are typically hard to learn with standard RNNs.

- Example: Context-sensitive grammar $\{a^n b^n c^n\}$

- LSTM with
  - 4 input units representing a, b, c, and start symbol;
  - 4 output units representing again a, b, c, and terminal symbol;
  - 3 hidden memory blocks.

- Network is supposed to learn to predict the next symbol(s) allowed at each time step.

- For example with n=3:

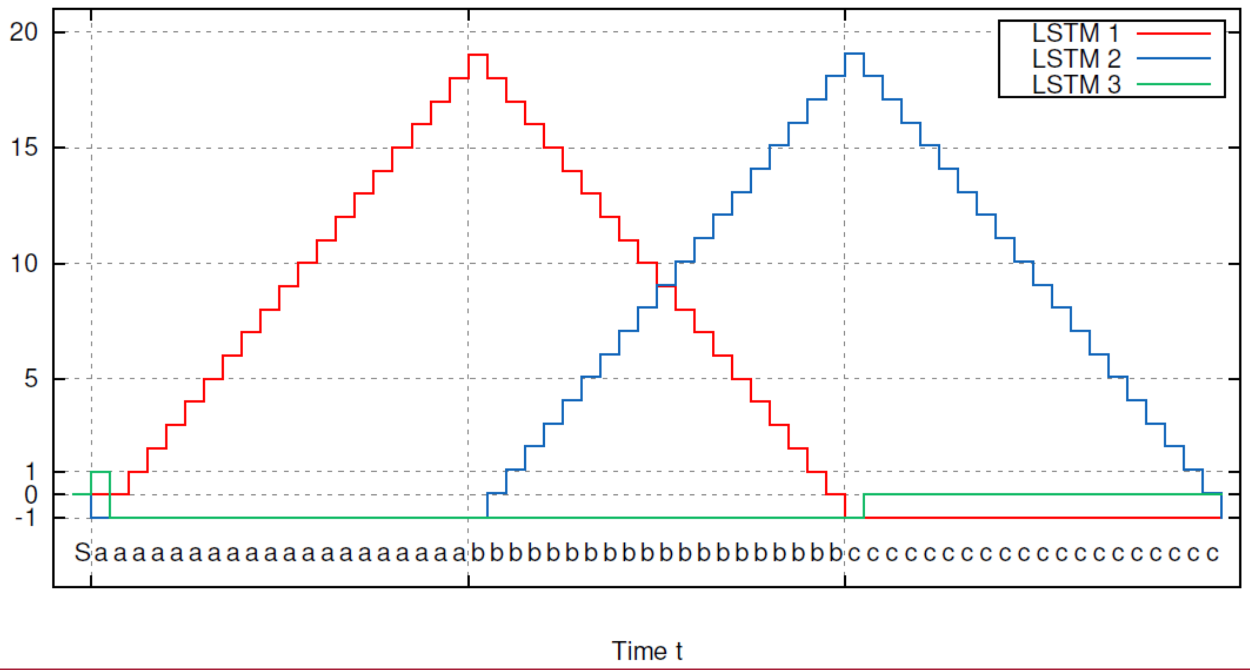
- Input:	S	a	a	a	b	b	b	c	c	c
- Output:	#,a	a,b	a,b	a,b	b	b	c	c	c	#

Example (cont.): Learning a Context-Sensitive Grammar

- For example with n=3:

- Input:	S	a	a	a	b	b	b	c	c	c	
- Output:		#,a	a,b	a,b	a,b	b	b	c	c	c	#

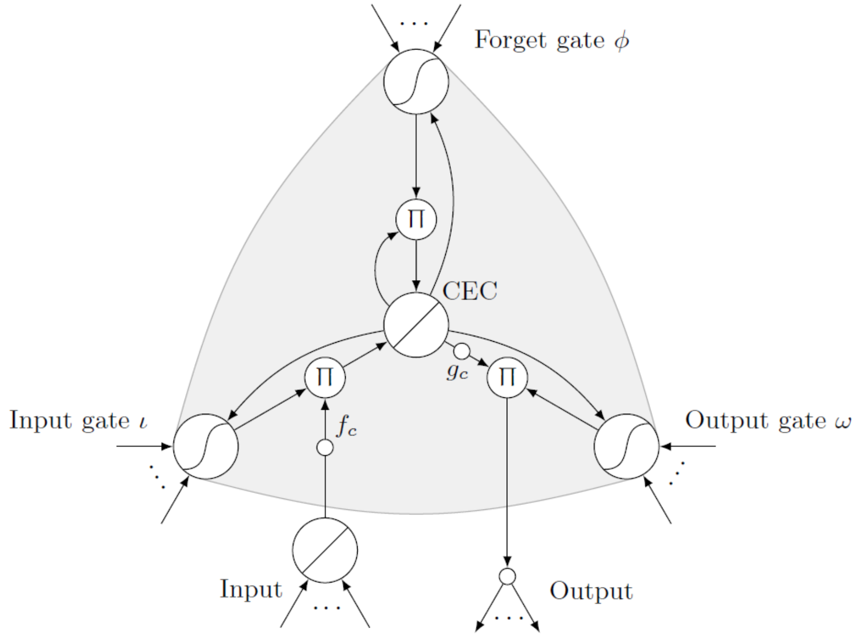
- Network was trained with various examples {a^{n}b^{n}c^n} with $n \in \{0,1, \dots 5\}$
- The resulting images show the activation of the memory cells (LSTM1, LSTM2, LSTM3) in the three LSTM memory blocks.
  - ➢ The LSTM network has learned a counting-up and counting-down strategy to learn the context-sensitive grammar.



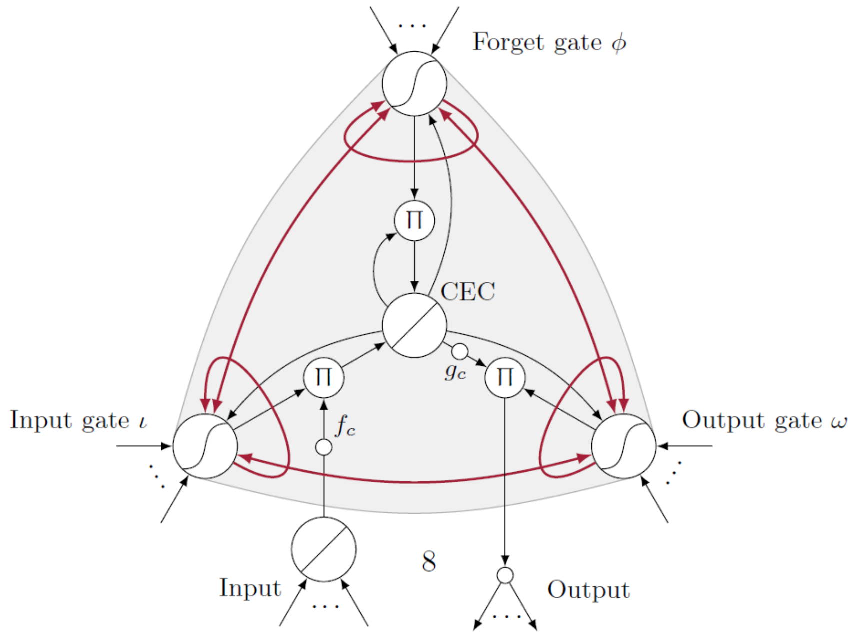
EBERHARD KARLS UNIVERSITÄT TÜBINGEN

Recent Development: Dynamic cortex memory

Slightly different visualization of LSTM memory block

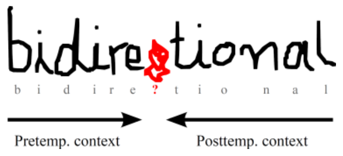


"Dynamic Cortex Memory"



Additional interactions between the gates enable even better coordination of the memory.

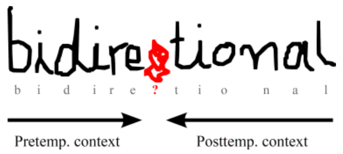
Bidirectional RNN Architectures



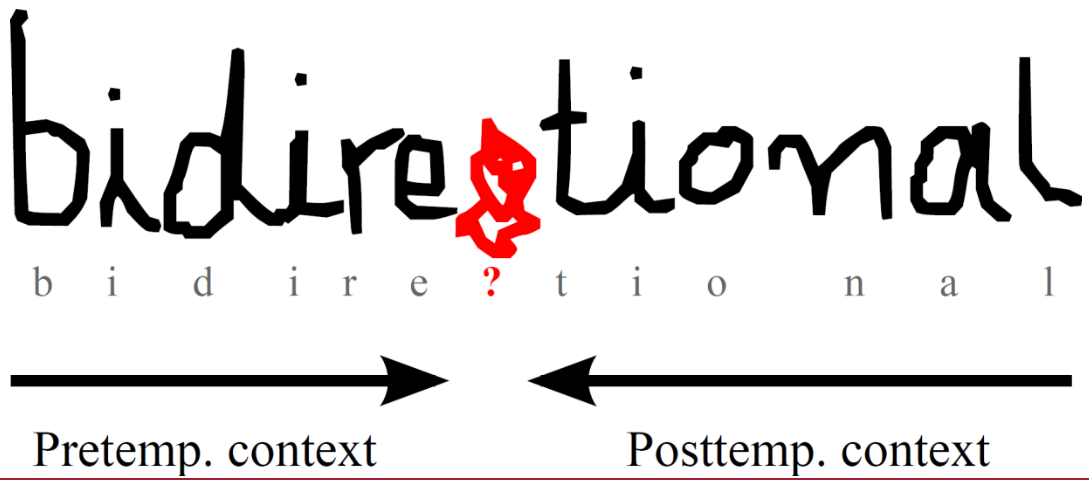
- Classical recurrent NN architectures are characterizable as **unidirectional**.
  - They perform in a temporal-causal way.
  - Input sequences are presented in a temporal, forward direction.
  - The input-history is accumulated in a network's past context (via the recurrences).
- In many time series problems, when the task is to classify the time series, however, both directions forward AND backward may contain useful information.
- In particular, an ambiguous state may be disambiguated by information from the future!
- Thus, the **future context** AND the **past context** may be used when processing time series data.
- Note: Obviously this works only when the full time series data is available at the time of computation.
- To make BOTH contexts available, **bidirectional RNNs (BRNNs)** have been introduced.

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

BRNN Architectures: A Second Hidden Layer



- In BRNNs a second hidden layer (or hidden cluster of neurons) is introduced that
  - also connects input to output
  - does NOT connect with the other hidden layer (or hidden cluster of neurons).
- This hidden layer is again self-recurrent and is used to process the time series in reverse order.
- Thus, the second hidden layer processes the post-temporal context (while the "normal" hidden layer processes the pre-temporal context).

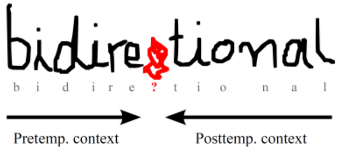


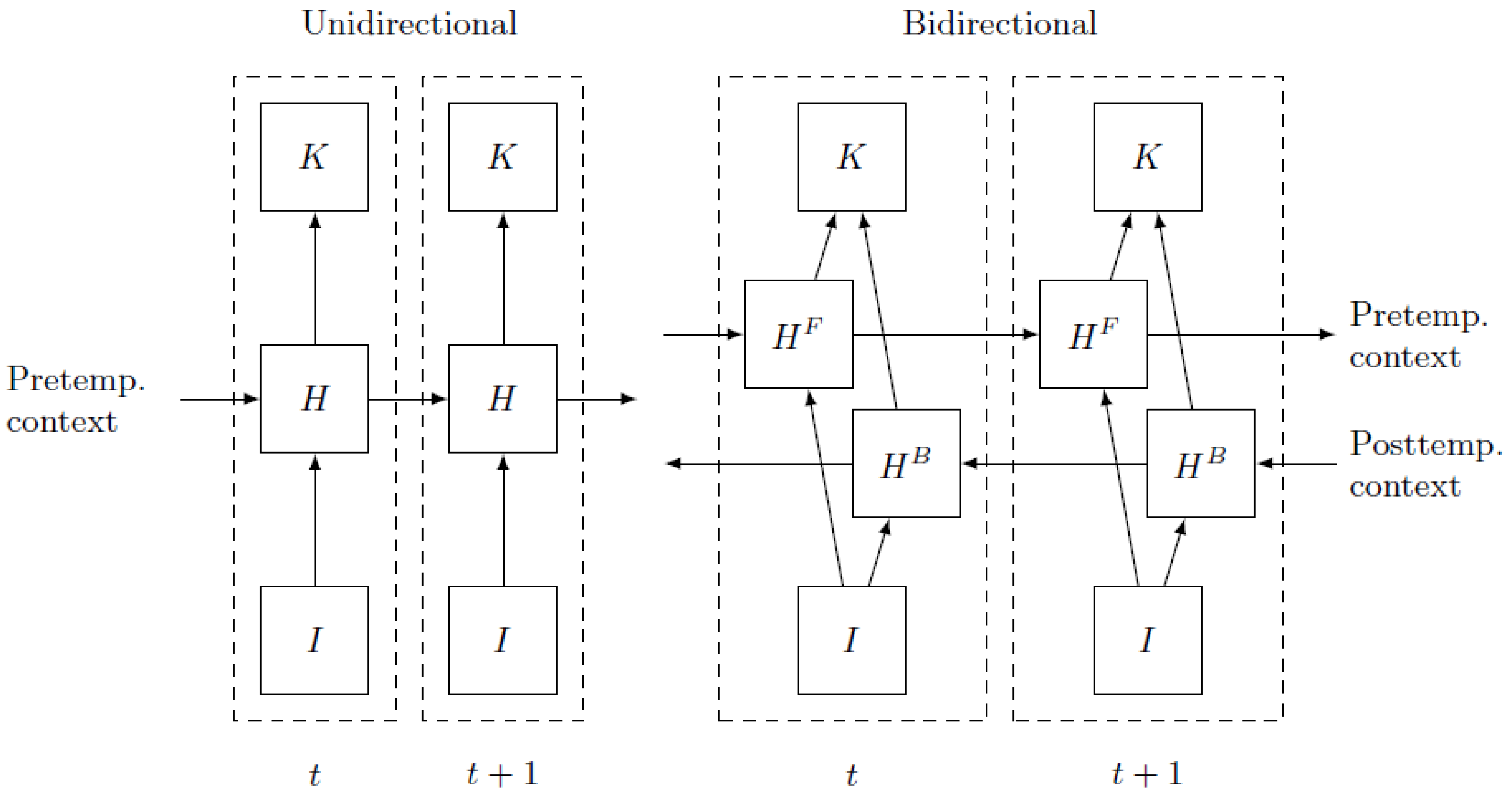
BRNN Architectures:
A Second Hidden Layer

- As a result, processing of a time series in a BRNN becomes slightly more complicated, consisting of three steps:
  1. Do the normal forward pass into the first hidden layer and store its states over time.
  2. Do an inverse pass processing the time series backwards, processing the inputs and passing them on to the second hidden layer.
  3. Compute the output layer by summing up the inputs from the two hidden layers for each respective time step.

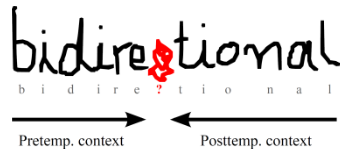
- To do the gradient computation, everything works as before, only that for the second hidden layer the error is processed inversely in the temporal direction.

Uni- versus Bidirectional



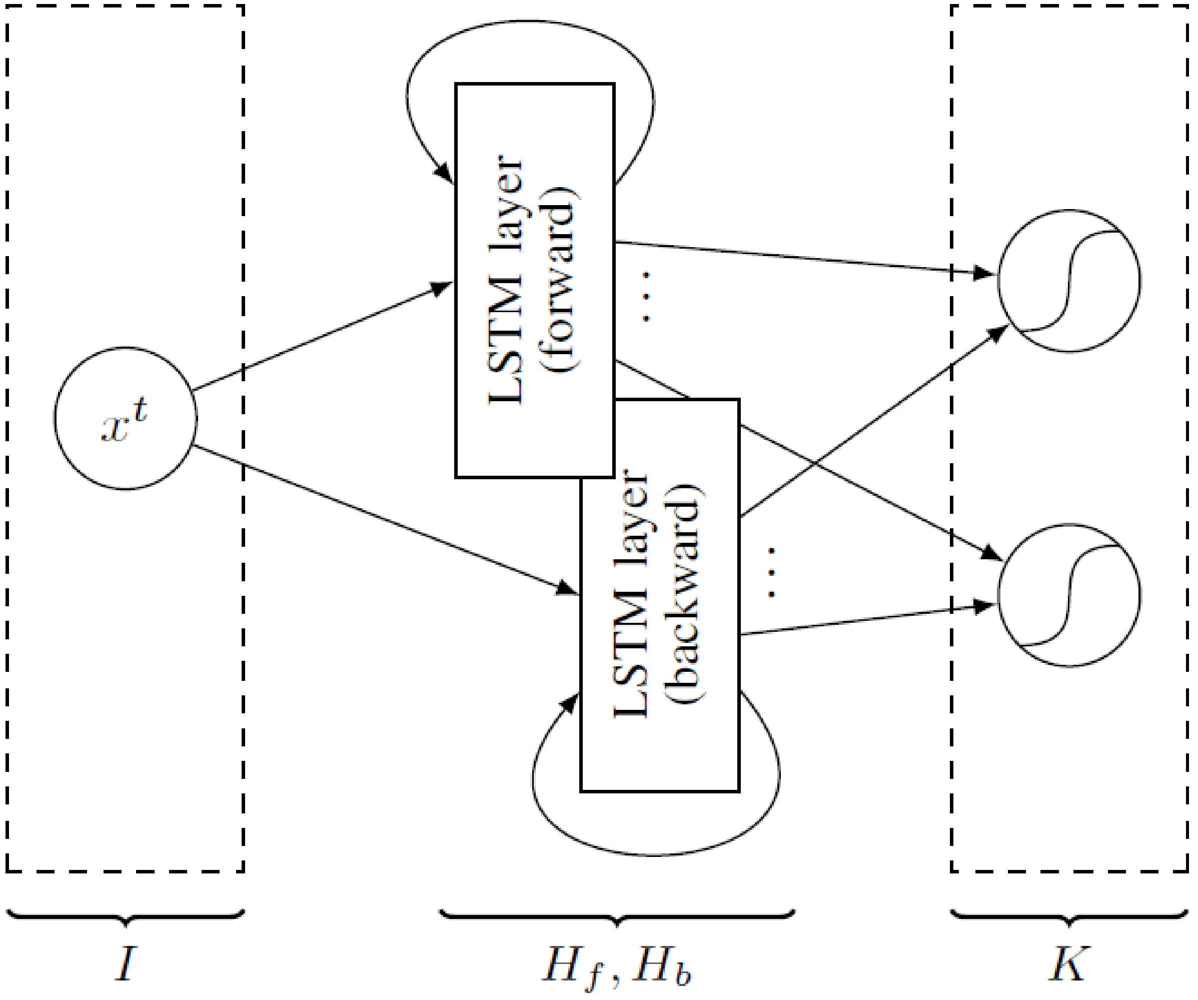


- In BRNNs, there are a forward and a backwards hidden layer (of hidden cluster of neurons).
- Note that both hidden clusters project to the current output, but the backwards hidden layer projects its activity recurrently into the past, while the forward hidden layer projects its activity recurrently into the future.

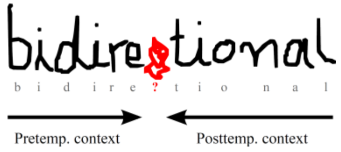


Output Computation

To compute the output, the states from both hidden layers are once again combined by the normal summation (resulting in the net activation in each output unit).

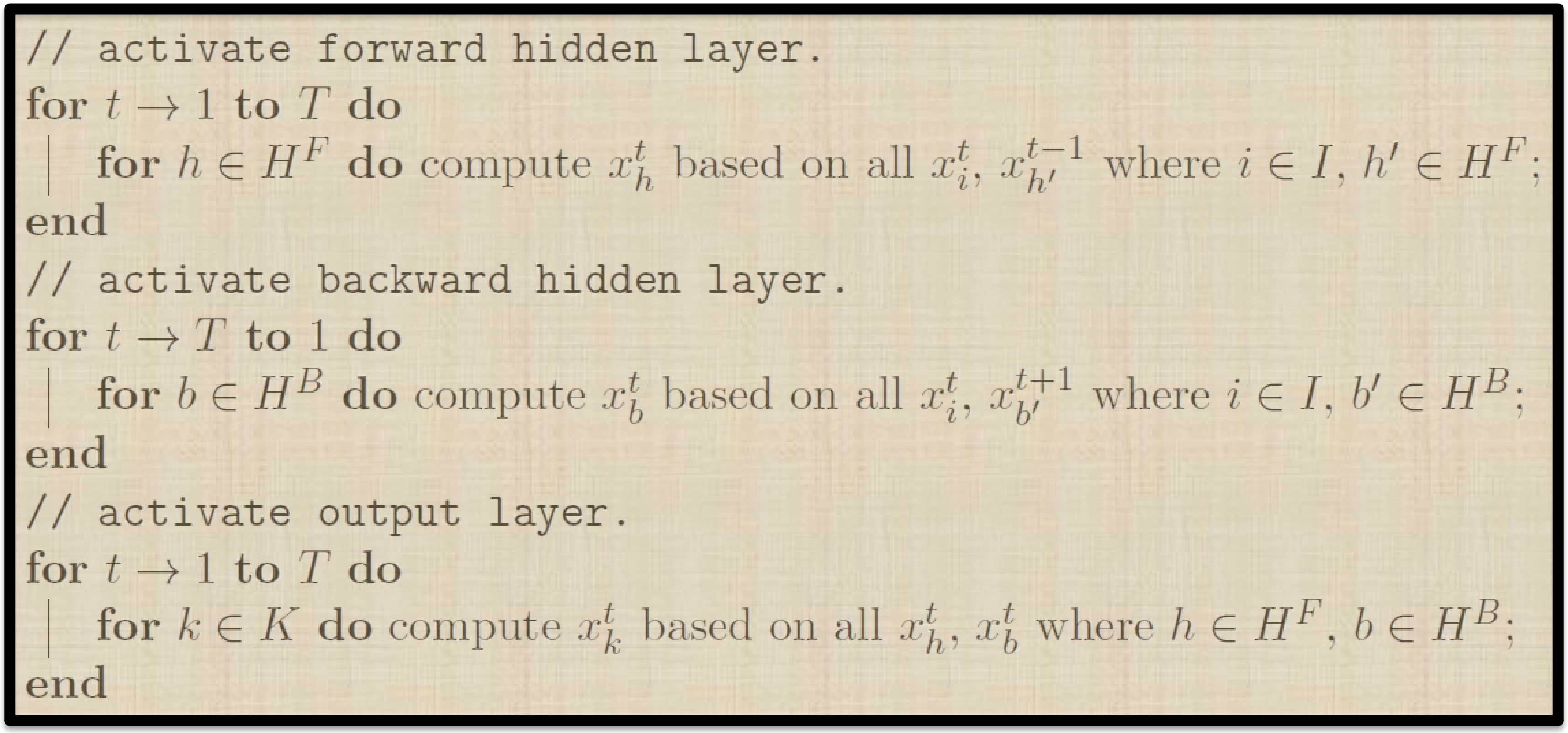


Computing the Activations in BRNNs



- Given a time series with
  - input values $x^1,\ldots,x^T$
  - corresponding output values $z^1,\ldots,z^T$

Execute the following algorithm:

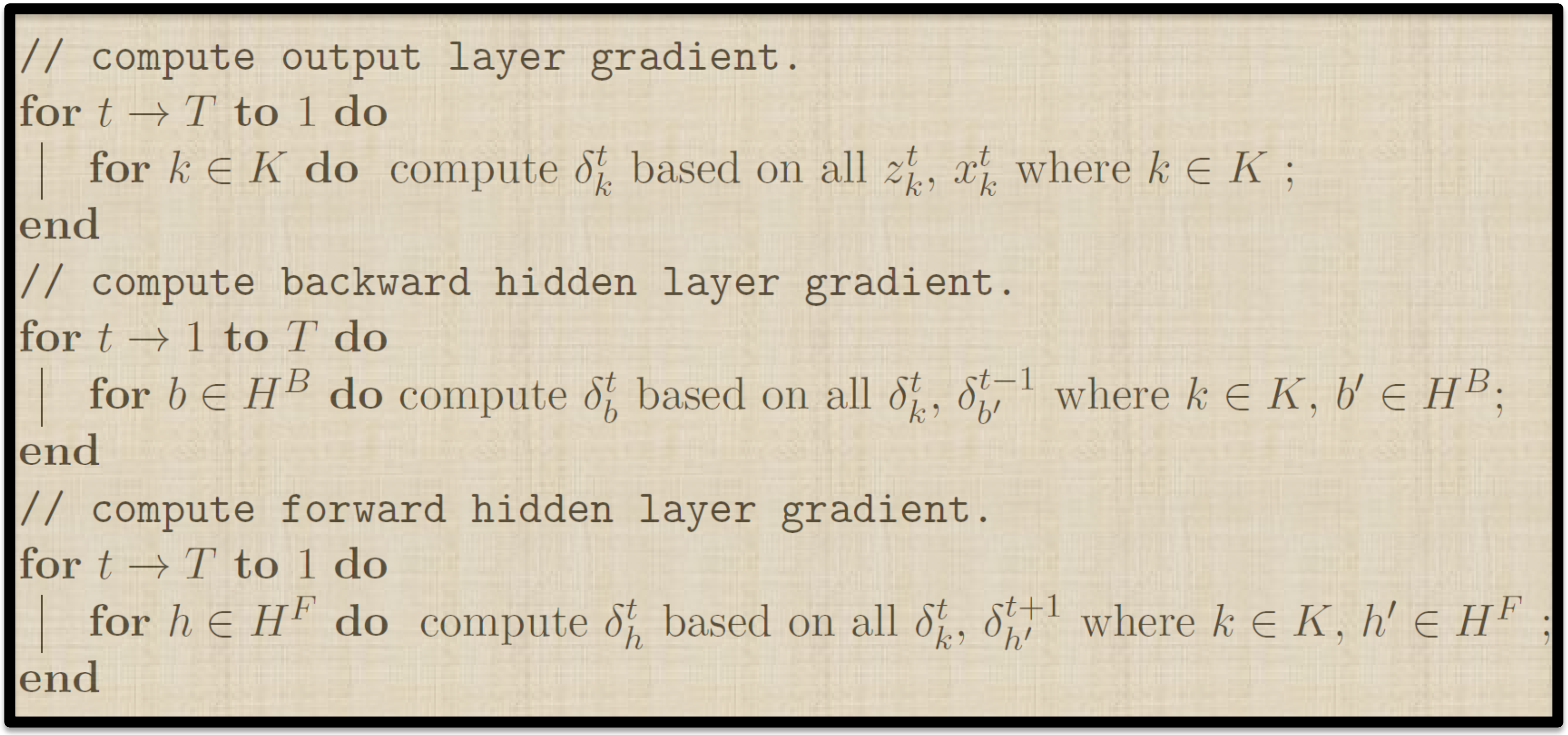


```
// activate forward hidden layer.
for $t \to 1$ to $T$ do
    for $h \in H^F$ do compute $x_h^t$ based on all $x_i^t$, $x_{h'}^{t-1}$ where $i \in I$, $h' \in H^F$;
end
// activate backward hidden layer.
for $t \to T$ to $1$ do
    for $b \in H^B$ do compute $x_b^t$ based on all $x_i^t$, $x_{b'}^{t+1}$ where $i \in I$, $b' \in H^B$;
end
// activate output layer.
for $t \to 1$ to $T$ do
    for $k \in K$ do compute $x_k^t$ based on all $x_h^t$, $x_b^t$ where $h \in H^F$, $b \in H^B$;
end
```

Computing the Gradients in BRNNs

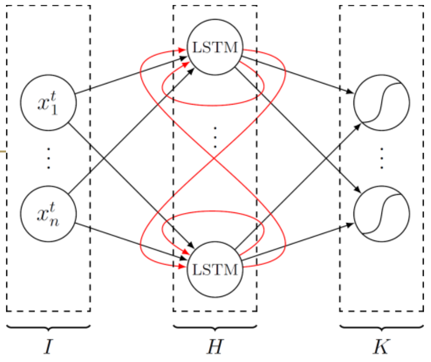
- Given a time series with
  - input values $x^1,\dots,x^T$
  - corresponding output values $z^1,\dots,z^T$

Execute the following algorithm:

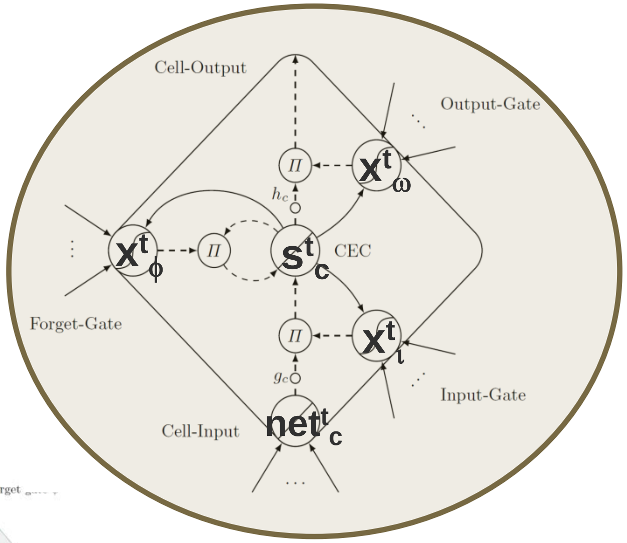


```
// compute output layer gradient.
for $t \to T$ to 1 do
    for $k \in K$ do compute $\delta_k^t$ based on all $z_k^t$, $x_k^t$ where $k \in K$ ;
end
// compute backward hidden layer gradient.
for $t \to 1$ to $T$ do
    for $b \in H^B$ do compute $\delta_b^t$ based on all $\delta_k^t$, $\delta_{b'}^{t-1}$ where $k \in K$, $b' \in H^B$;
end
// compute forward hidden layer gradient.
for $t \to T$ to 1 do
    for $h \in H^F$ do compute $\delta_h^t$ based on all $\delta_k^t$, $\delta_{h'}^{t+1}$ where $k \in K$, $h' \in H^F$ ;
end
```

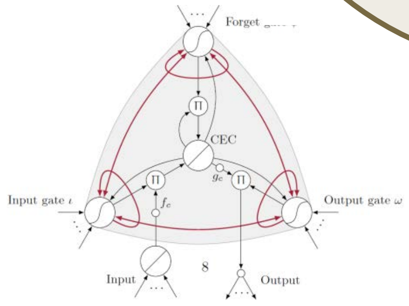
LSTMs – Summary



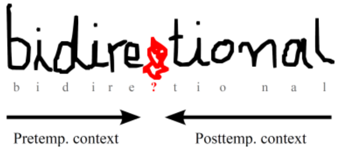
- Normal RNNs suffer from the Vanishing Gradient Problem
- LSTMs solve this problem by providing memory cells.
- Memory cells have
  - One or several self-recurrent linear cells with constant weight one (called the Constant Error Carousel, CEC)
  - Gates, which protect the CEC neurons:
    - Output gate, Input gate, Forget gate.
- Learning works once again with BPTT.



- Enhancements of LSTMs include:
  - Dynamic Cortex Memory
  - Bidirectional RNNs (also applicable with "normal" RNNs).



Forget
Π
CEC
g_c
Π
Π
f_c
Input gate ι
Output gate ω
Input
8
Output



bidire?tional
b i d i r e ? t i o n a l
Pretemp. context
Posttemp. context