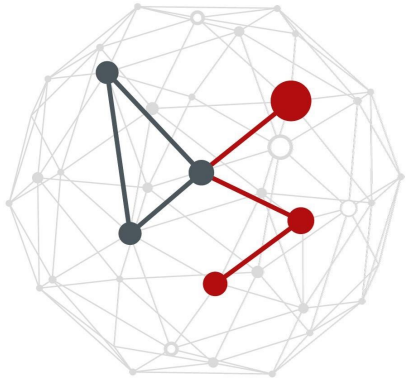
MiME
Master's degree ICT Internet Multimedia Engineering

LEARNING FOR SEQUENTIAL DATA: TOOLS AND APPLICATIONS

Michele Rossi
michele.rossi@dei.unipd.it

Dept. of Information Engineering
University of Padova, IT



DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

DIPARTIMENTO MATEMATICA

UNIVERSITÀ DEGLI STUDI DI PADOVA

Outline

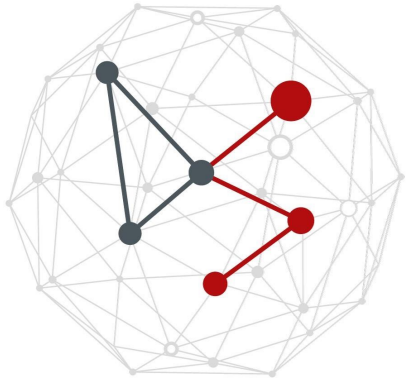
- In this lesson
  - We are interested in time correlated data sequences
- Data points (samples)
  - Are generated one at a time
  - One data point can be: a real number, a vector, an image..
  - Are time correlated
  - Sequences can have a different length – important!
  - Are sequentially fed to an algorithm
  - To capture some key features of the data
- Learning objectives
  - Correlated seqs: capture temporal evolution

So far

- We have been dealing with sequences
  - But the points were i.i.d. sampled
  - Physical phenomena are often time correlated
  - *What about correlated data?*
- Need richer architectures
  - **From 1998-2009:** Hidden Markov Models
  - **Nowadays:** Recurrent Neural Networks (RNN)

MIME
Master's degree ICT Internet Multimedia Engineering

RECURRENT NEURAL NETWORKS (RNN)



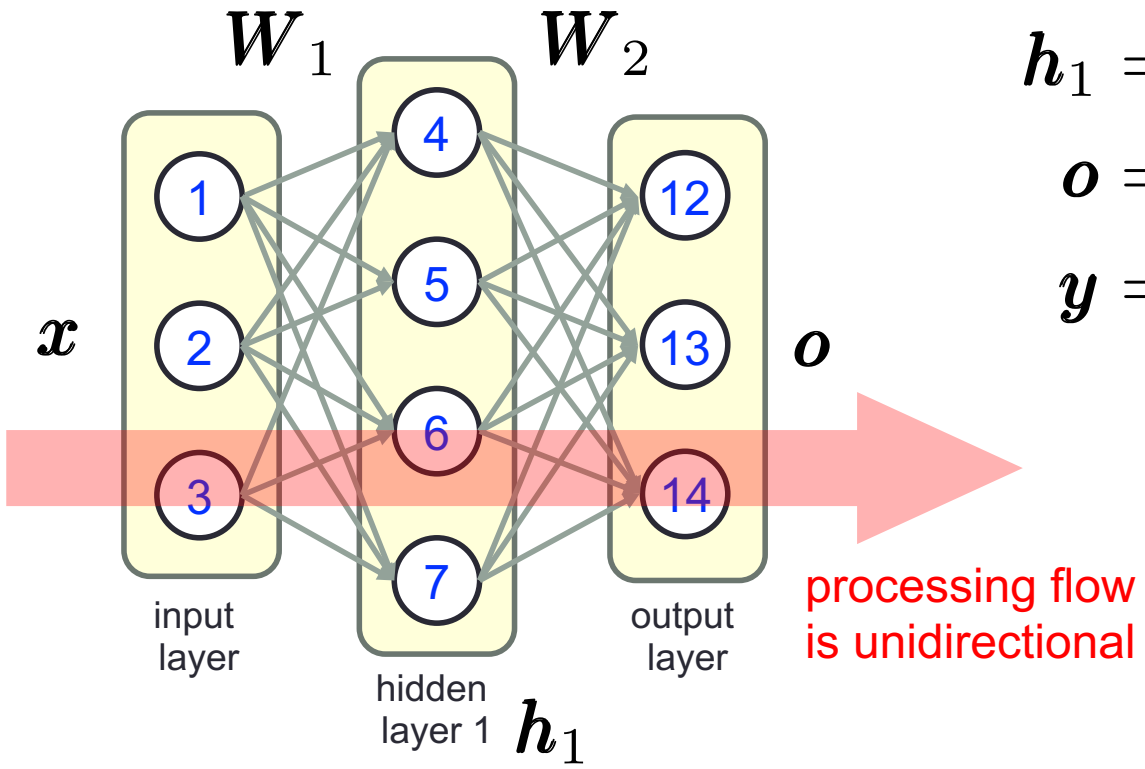
DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

DIPARTIMENTO
MATEMATICA

UNIVERSITÀ
DEGLI STUDI
DI PADOVA

FFNN recap

- Feed Forward Neural Networks (FFNNs)
  - Stack one layer on top of another
  - Output of last layer is a function of the input
  - Can be decomposed into layer-wise transforms



$$h_1 = \sigma(W_1 x + b_1)$$
$$o = \sigma(W_2 h_1 + b_2)$$
$$y = \mathrm{softmax}(o)$$

Quick recap - softmax

- Often used as the output function of a FFNN
- When the objective is to estimate probabilities for a number of classes
- If the output vector from the previous layer is

$$\boldsymbol{o} = (o_1, \ldots, o_K)$$

- The softmax output vector is

$$y_j = \text{softmax}(\boldsymbol{o})_j = \frac{e^{o_j}}{\sum_{i=1}^{K} e^{o_i}}$$

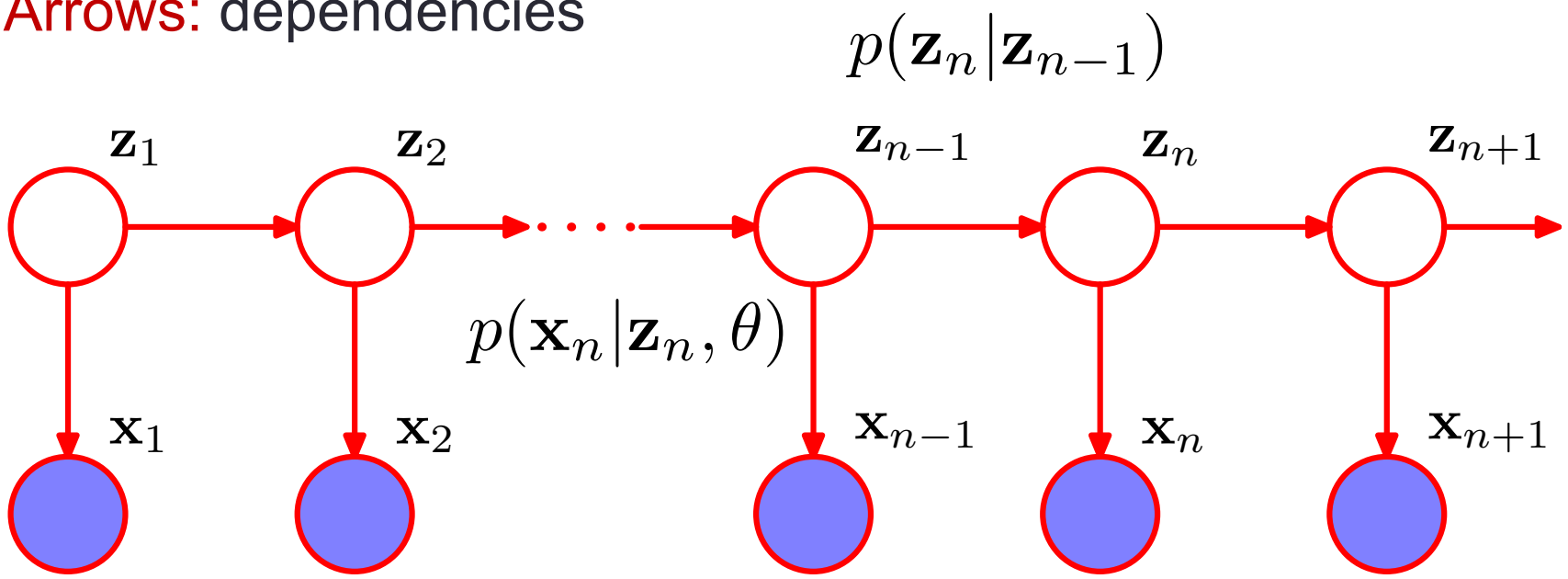
$$y_j \in (0, 1)\,, \sum_{j=1}^{K} y_j = 1$$

FFNN limitations

- FFNNs
  - rely on the assumption of *independence* among the training and test examples
  - after each example (data point) is processed, the entire state of the network *is lost*
- Classical approaches
  - use a sliding window approach

Traditional Approach: Hidden Markov Model

- Graphical model
- States: white filled circles (not accessible)
- Emissions or observations: blue filled circles
- Arrows: dependencies



HMM formalism

- Let us consider N observations

$$\boldsymbol{X} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N\} \quad \text{observations}$$

$$\boldsymbol{Z} = \{\boldsymbol{z}_1, \boldsymbol{z}_2, \ldots, \boldsymbol{z}_N\} \quad \text{latent variables}$$

$$\boldsymbol{\theta} = \{\boldsymbol{\pi}, \boldsymbol{A}, \boldsymbol{\phi}\} \quad \text{HMM parameters}$$

- An HMM model is represented by the tuple:

$$\mathcal{M} = \{\boldsymbol{X}, \boldsymbol{Z}, \boldsymbol{\pi}, \boldsymbol{A}, \boldsymbol{\phi}\}$$

What's wrong with HMM?

- Efficient inference with HMM
  - S = number of states
  - Uses the Viterbi algorithm (exploits Bellman eq.)
  - Time complexity $O(|S|^2)$
  - Space (memory) complexity $|S|^2$
- Complexity
  - Gets way too large for increasing no. of states
  - Memory is limited to one-step
  - Can be increased at the cost of a growth in the number of states (exponential)

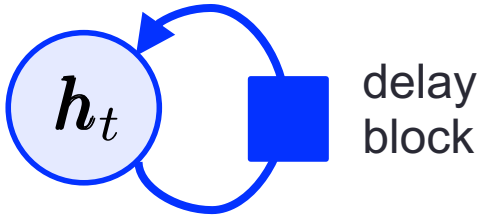
What's wrong with HMM?

- Efficient inference with HMM
  - S = number of states
  - Uses the Viterbi algorithm
  - Time complexity $O(|S|^2)$
  - Space (memory) complexity $|S|^2$
- Complexity
  - Gets *way too large* for increasing no. of states
  - Memory is *limited* to one-step
    - can be increased at the cost of a increasing the number of states (exponential growth)

RNN – recursion

- RNN
  - Add cycles to FFNNs
  - Output is a function of **1)** input, **2)** internal state
  - This is achieved using a recursive formulation
- Recursive function, no input

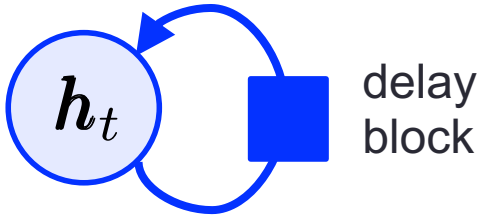
$$\boldsymbol{h}_t = f(\boldsymbol{h}_{t-1}; \boldsymbol{\theta})$$



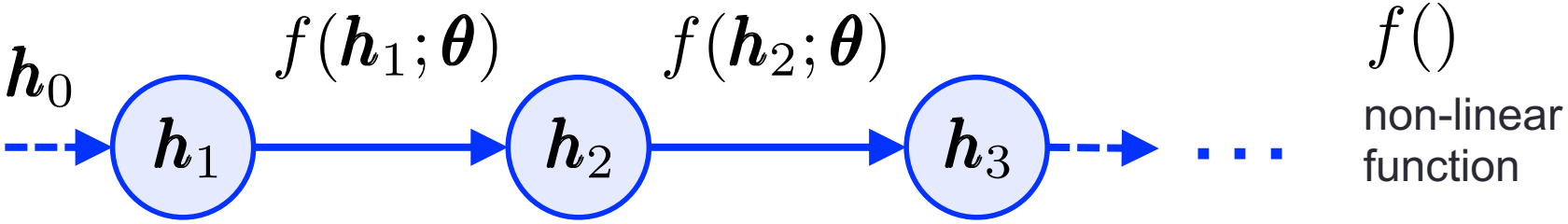
RNN – unfolding

- RNN
  - Add cycles to FFNNs
  - Output is a function of **1)** input, **2)** internal state
  - This is achieved using a recursive formulation
- Recursive function, no input

$$\boldsymbol{h}_t = f(\boldsymbol{h}_{t-1}; \boldsymbol{\theta})$$



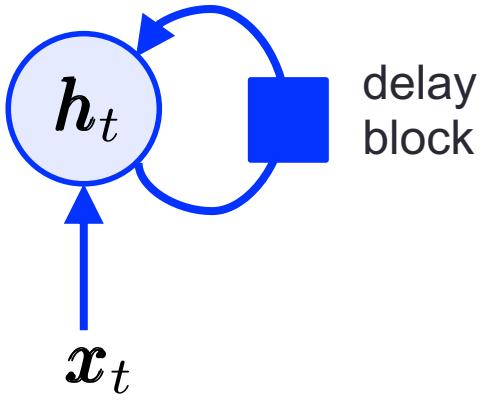
- Unfolding
  - Leads to a Directed Acyclic Graph (DAG) structure
  - More preferable for training RNN weights



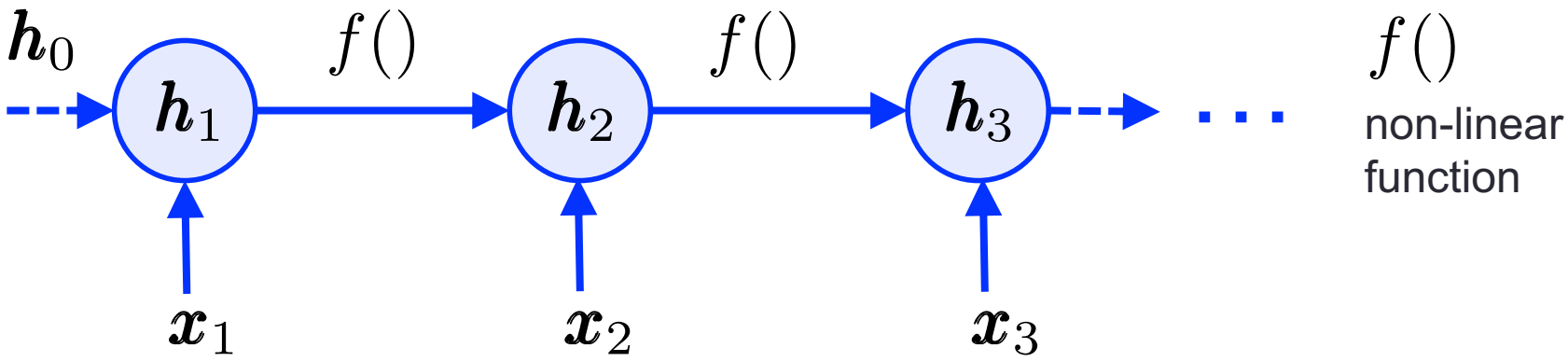
RNN with external input

- RNN
  - The recursion can be made more complex
- Recursive function, with added input

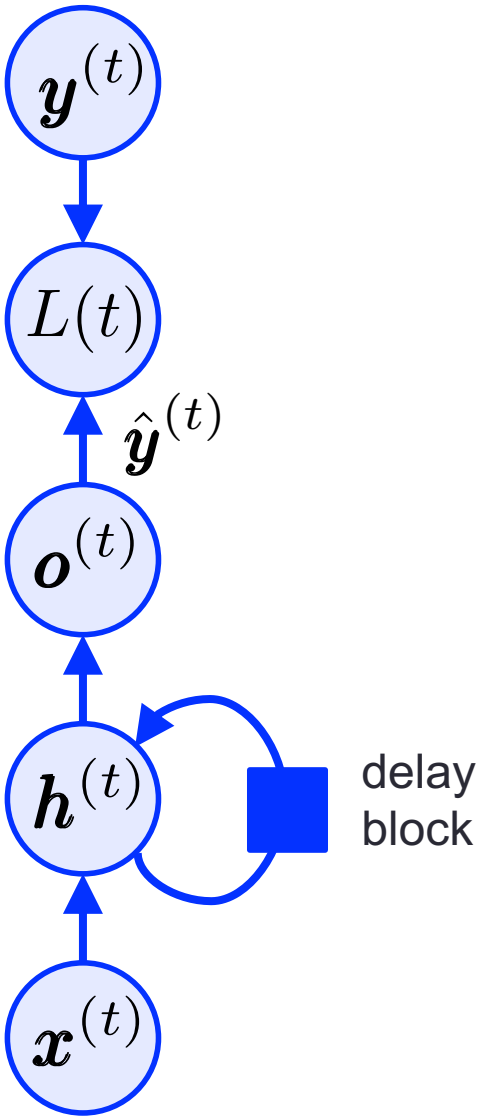
$$\boldsymbol{h}_t = f(\boldsymbol{h}_{t-1}, \boldsymbol{x}_t; \boldsymbol{\theta})$$



- Unfolding
  - Lead to a DAG



A full-fledged RNN



Target output (label)

$$\boldsymbol{y}^{(t)}$$

Loss (error) function

$$L(t) = \mathcal{L}(\hat{\boldsymbol{y}}^{(t)}, \boldsymbol{y}^{(t)})$$

Output (e.g., softmax for classification)

$$\boldsymbol{o}^{(t)} \Rightarrow \hat{\boldsymbol{y}}^{(t)} = \mathrm{softmax}(\boldsymbol{o}^{(t)})$$

Internal state (non-linearity is applied element-wise)

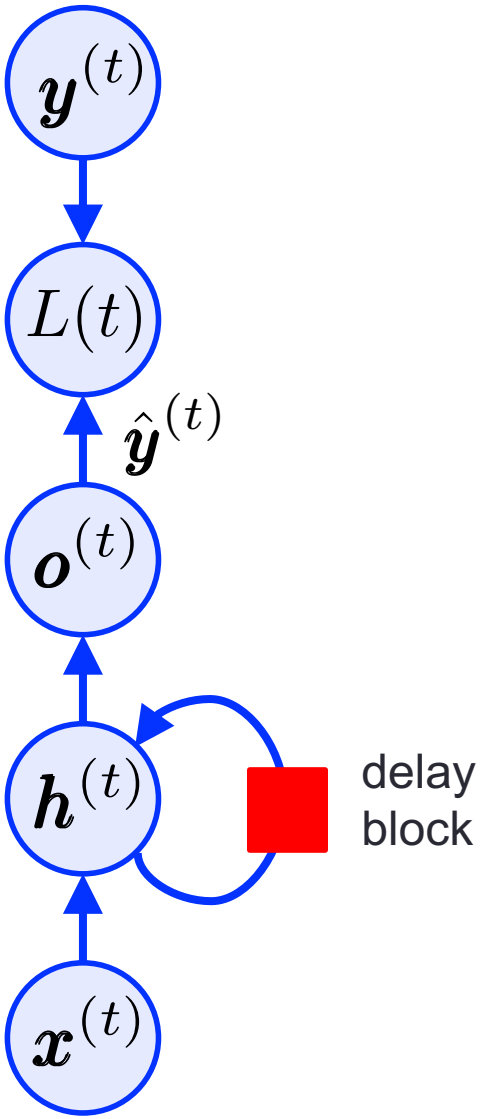
$$\boldsymbol{h}^{(t)} = \tanh\left(\boldsymbol{a}^{(t)}\right)$$

Linear activation

$$\boldsymbol{a}^{(t)} = \boldsymbol{U}\boldsymbol{x}^{(t)} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{b}$$

Input at time t (e.g., 1-hot vector, or vector of real numbers)

A full-fledged RNN



Target output (label)

$$y^{(t)}$$

Loss (error) function

$$L(t) = \mathcal{L}(\hat{y}^{(t)}, y^{(t)})$$

Output

$$o^{(t)} \Rightarrow \hat{y}^{(t)} = \mathrm{softmax}(o^{(t)})$$

Internal state

$$h^{(t)} = \tanh\left(a^{(t)}\right)$$

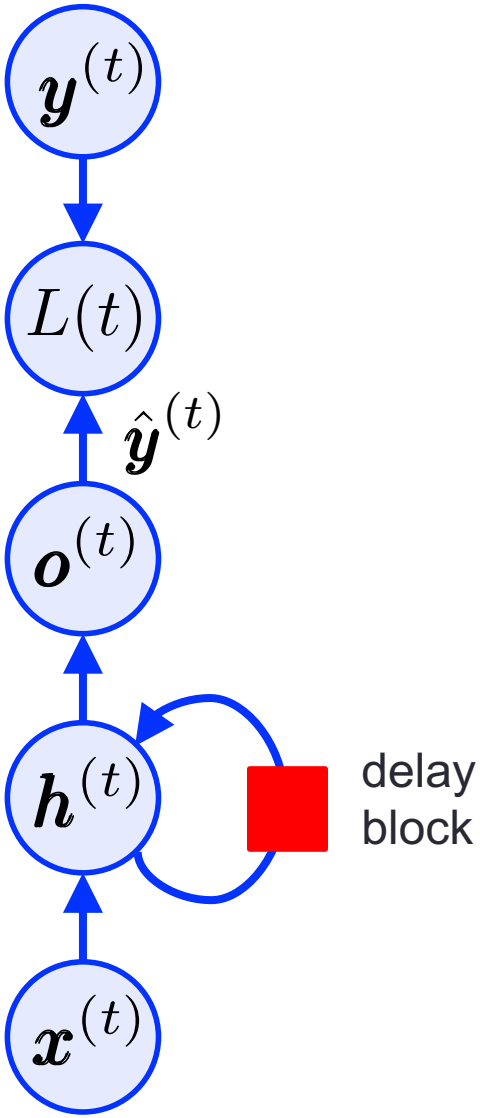
Activation

$$a^{(t)} = Ux^{(t)} + Wh^{(t-1)} + b$$

recurrence

Input at time t

A full-fledged RNN



Overall

$$\boldsymbol{a}^{(t)} = \boldsymbol{U}\boldsymbol{x}^{(t)} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{b}$$

linear activations

$$\boldsymbol{h}^{(t)} = \tanh\left(\boldsymbol{a}^{(t)}\right)$$ element-wise

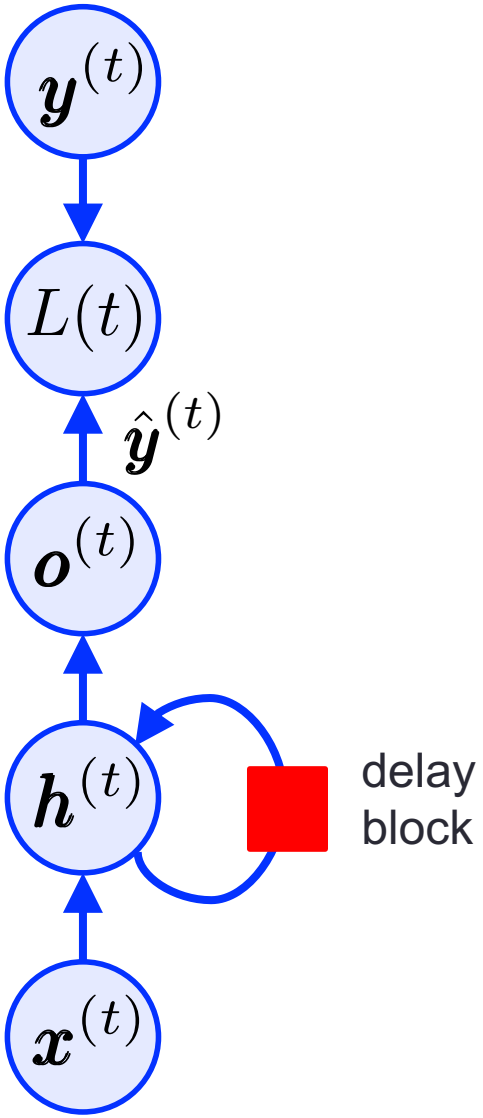
$$\boldsymbol{o}^{(t)} = \boldsymbol{V}\boldsymbol{h}^{(t)} + \boldsymbol{c}$$ dense layer

$$\hat{\boldsymbol{y}}^{(t)} = \mathrm{softmax}\left(\boldsymbol{o}^{(t)}\right),$$

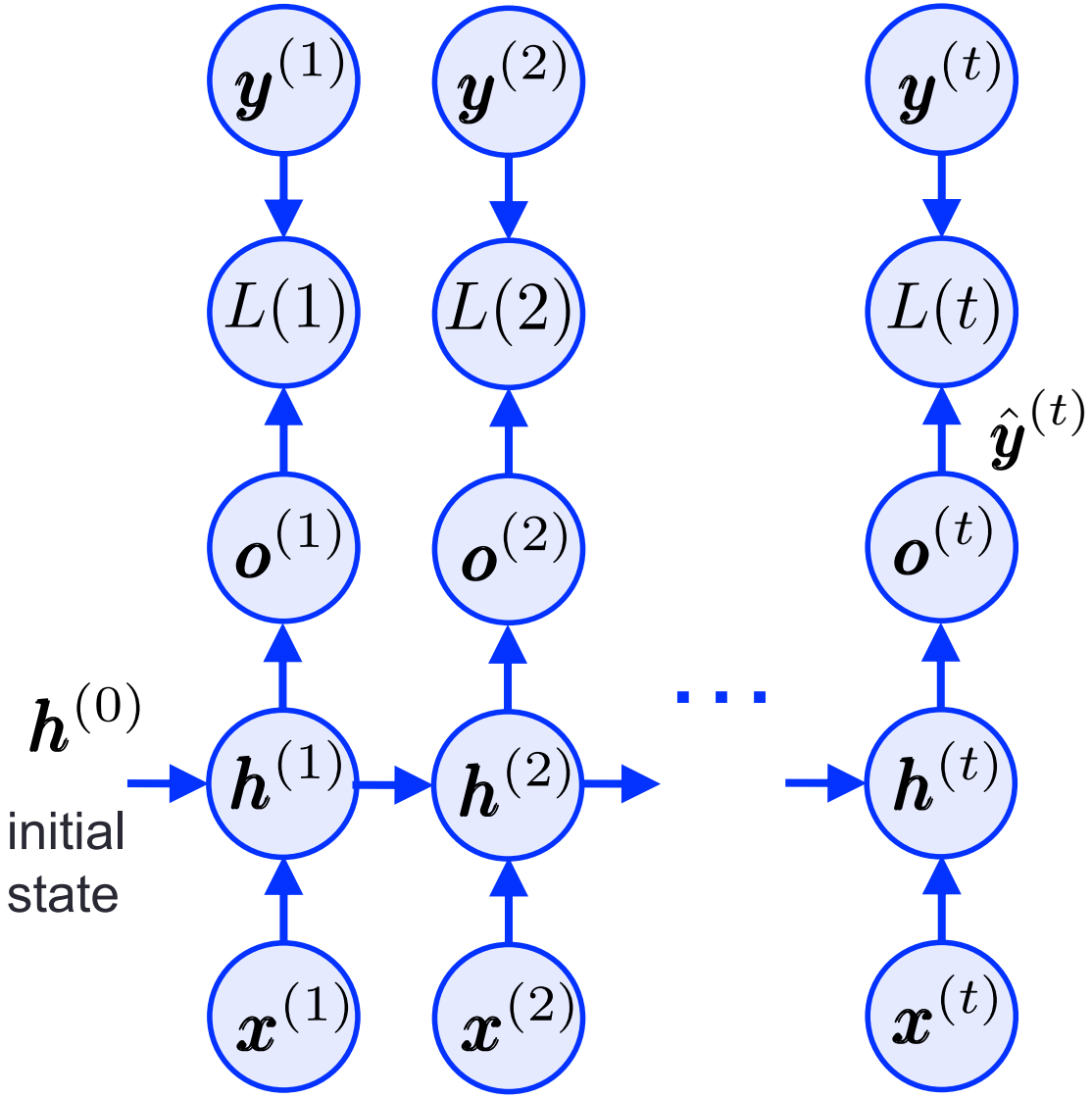
RNN hyperparameters (to be learned)

$$\boldsymbol{\theta} = \{\boldsymbol{U}, \boldsymbol{W}, \boldsymbol{V}, \boldsymbol{b}, \boldsymbol{c}\}$$

RNN unfolding



unfolded graph is a standard DAG

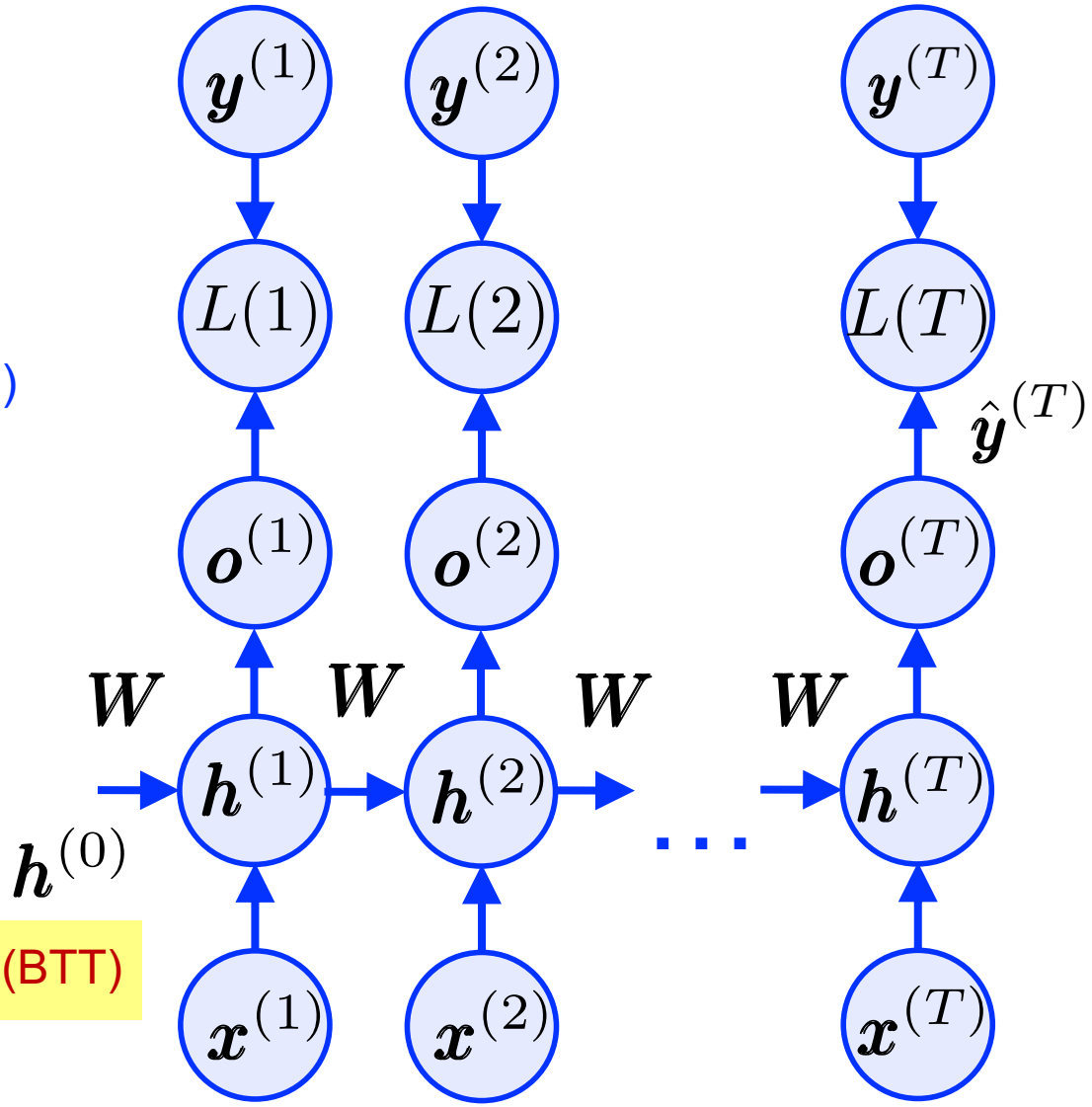


Dependencies

unfolded graph from t=1,…,T is a standard DAG

- Consider T time steps
  - t = 1,2,…,T
- Output at time T depends on
  - the initial state at time 0
  - all inputs in 1,2,...,T
- Shared weights (as for CNNs)
  - the same matrix **W**
  - is used for t=1,2, ...,T
  - **W** is constant
  - and has to be learned
- Error gradient
  - Can be propagated
    - from output to input
    - from time T down to 1

Backpropagation Through Time (BTT)



Input, output and loss – classification

- Target vector $\boldsymbol{y}^{(t)}$ at each t
  - For a classification problem, use, e.g., *1-hot vectors*
    - K elements (number of classes)
    - only one element is 1 all other elements are 0
    - the non-zero element indicates the input class
- Output $\boldsymbol{o}^{(t)}$ is also a vector
  - Same size K, real numbers
- Loss function L(t) and output
  - For classification: softmax output + cross-entropy loss

$$\hat{\boldsymbol{y}}^{(t)} = \mathrm{softmax}(\boldsymbol{o}^{(t)})$$ Softmax (from real to prob. distrib.)

$$L_{\mathrm{class}}(t) = -\sum_{k=1}^{K} y_k^{(t)} \log(\hat{y}_k^{(t)})$$ cross-entropy loss

Input, output and loss – regression

- Target vector $\boldsymbol{y}^{(t)}$ at each t
  - For a regression problem, is a vector of real numbers
    - K elements
- Output is also a vector
  - Vector of real numbers
  - Same size K
- Loss function L(t) and output
  - For regression: linear output + Mean Square Error (MSE)

$$\hat{\boldsymbol{y}}^{(t)} = \boldsymbol{o}^{(t)} \quad \text{linear}$$

$$L_{\text{regr}}(t) = \frac{1}{K}\sum_{k=1}^{K}(y_k^{(t)} - \hat{y}_k^{(t)})^2 \quad \text{MSE}$$

Definitions

- Given a matrix **X** (mxn) and a scalar function y

$$\nabla_{\boldsymbol{X}} y = \begin{bmatrix} \frac{\partial y}{\partial x_{11}} & \frac{\partial y}{\partial x_{12}} & \cdots & \frac{\partial y}{\partial x_{1n}} \\ \frac{\partial y}{\partial x_{21}} & \frac{\partial y}{\partial x_{22}} & \cdots & \frac{\partial y}{\partial x_{2n}} \\ \vdots & \cdots & \ddots & \vdots \\ \frac{\partial y}{\partial x_{m1}} & \frac{\partial y}{\partial x_{m2}} & \cdots & \frac{\partial y}{\partial x_{mn}} \end{bmatrix}$$

matrix of partial derivatives

- Given a vector **x** (nx1) and a scalar function y

$$\nabla_{\boldsymbol{x}} y = \begin{bmatrix} \frac{\partial y}{\partial x_1} & \frac{\partial y}{\partial x_2} & \cdots & \frac{\partial y}{\partial x_n} \end{bmatrix}^\top$$

the gradient (as column vector)

Definitions

- Given two vectors **x** and **y** (same size, nx1), we define

$$\frac{\partial \boldsymbol{x}}{\partial \boldsymbol{y}} = \left[\frac{\partial x_i}{\partial y_j}\right] = \begin{bmatrix} \frac{\partial x_1}{\partial y_1} & \frac{\partial x_1}{\partial y_2} & \cdots & \frac{\partial x_1}{\partial y_n} \\ \frac{\partial x_2}{\partial y_2} & \frac{\partial x_2}{\partial y_2} & \cdots & \frac{\partial x_2}{\partial y_n} \\ \vdots & \ddots & \ddots & \vdots \\ \frac{\partial x_n}{\partial y_1} & \frac{\partial x_n}{\partial y_2} & \cdots & \frac{\partial x_n}{\partial y_n} \end{bmatrix}$$

Backpropagation Through Time (BTT)

- Is a standard backpropagation algo
  - As the one commonly used for FFNN (including CNNs)
  - Gets a special name due to its importance in RNNs
- The main steps are
  1. Compute the total error from t=1,..,T as: $L \triangleq \sum_{t=1}^{T} L(t)$
  2. Unfold network from time 1 (start) to T (end) and find derivatives of total error with respect to *all network parameters* (weights, biases):

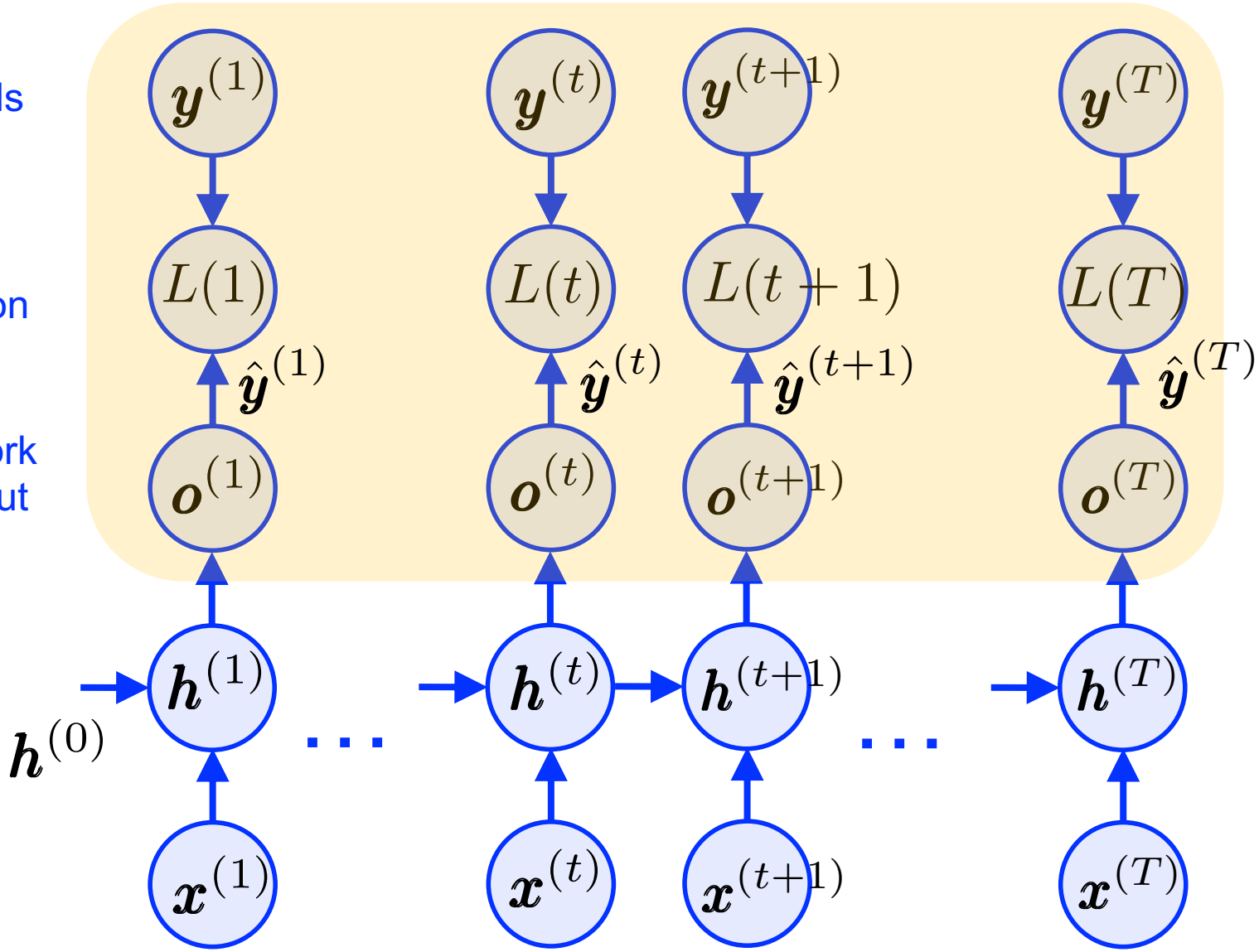
$$\frac{\partial L}{\partial \boldsymbol{V}}, \frac{\partial L}{\partial \boldsymbol{W}}, \frac{\partial L}{\partial \boldsymbol{U}}, \nabla_{\boldsymbol{b}}(L), \nabla_{\boldsymbol{c}}(L)$$

  3. Gradient descent (p is *any* network parameter)

$$p(t+1) = p(t) - \eta \frac{\partial L}{\partial p}$$

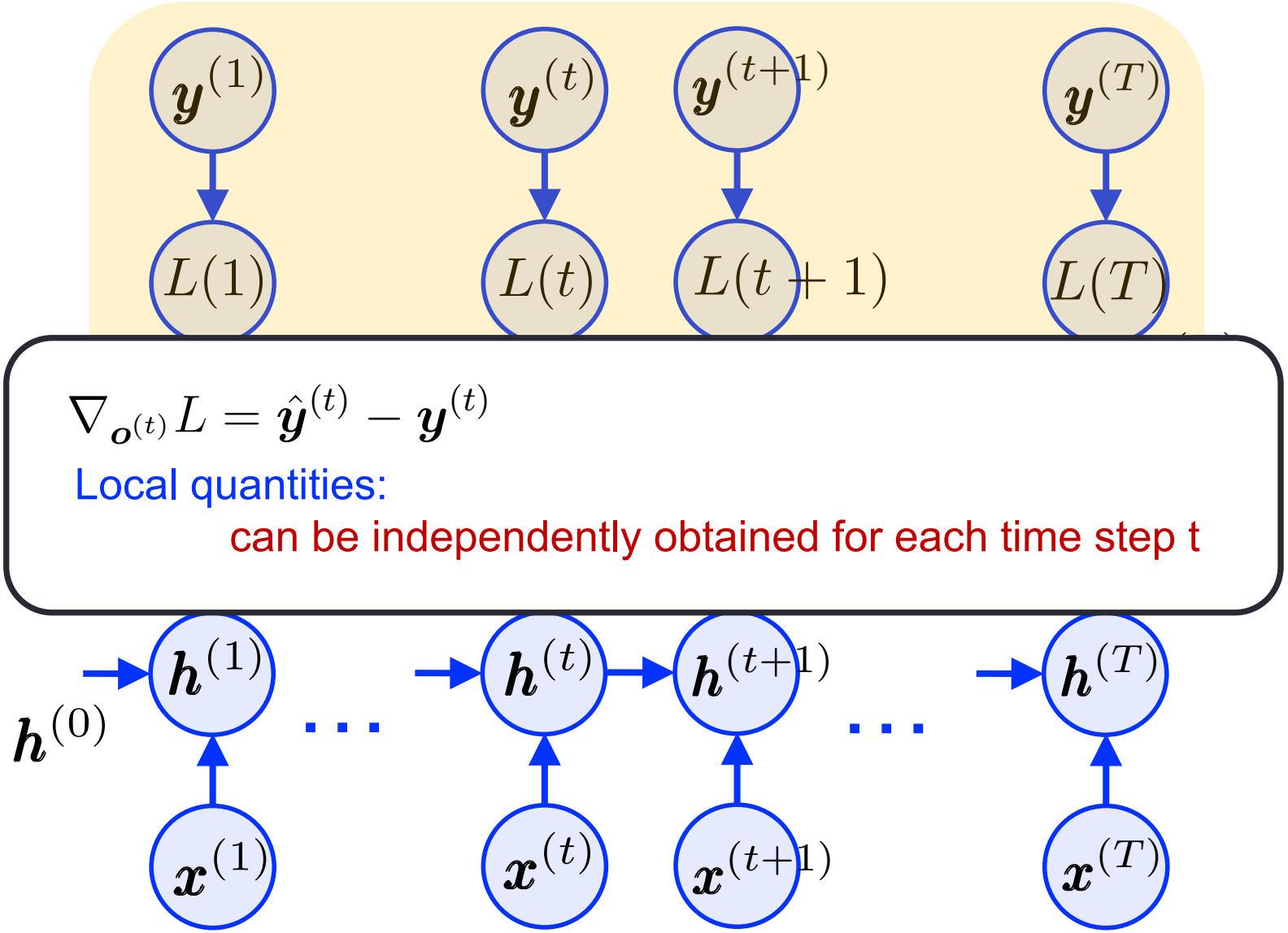
BTT – unfolded graph

$$\nabla_{\boldsymbol{o}^{(t)}} L = \hat{\boldsymbol{y}}^{(t)} - \boldsymbol{y}^{(t)}$$



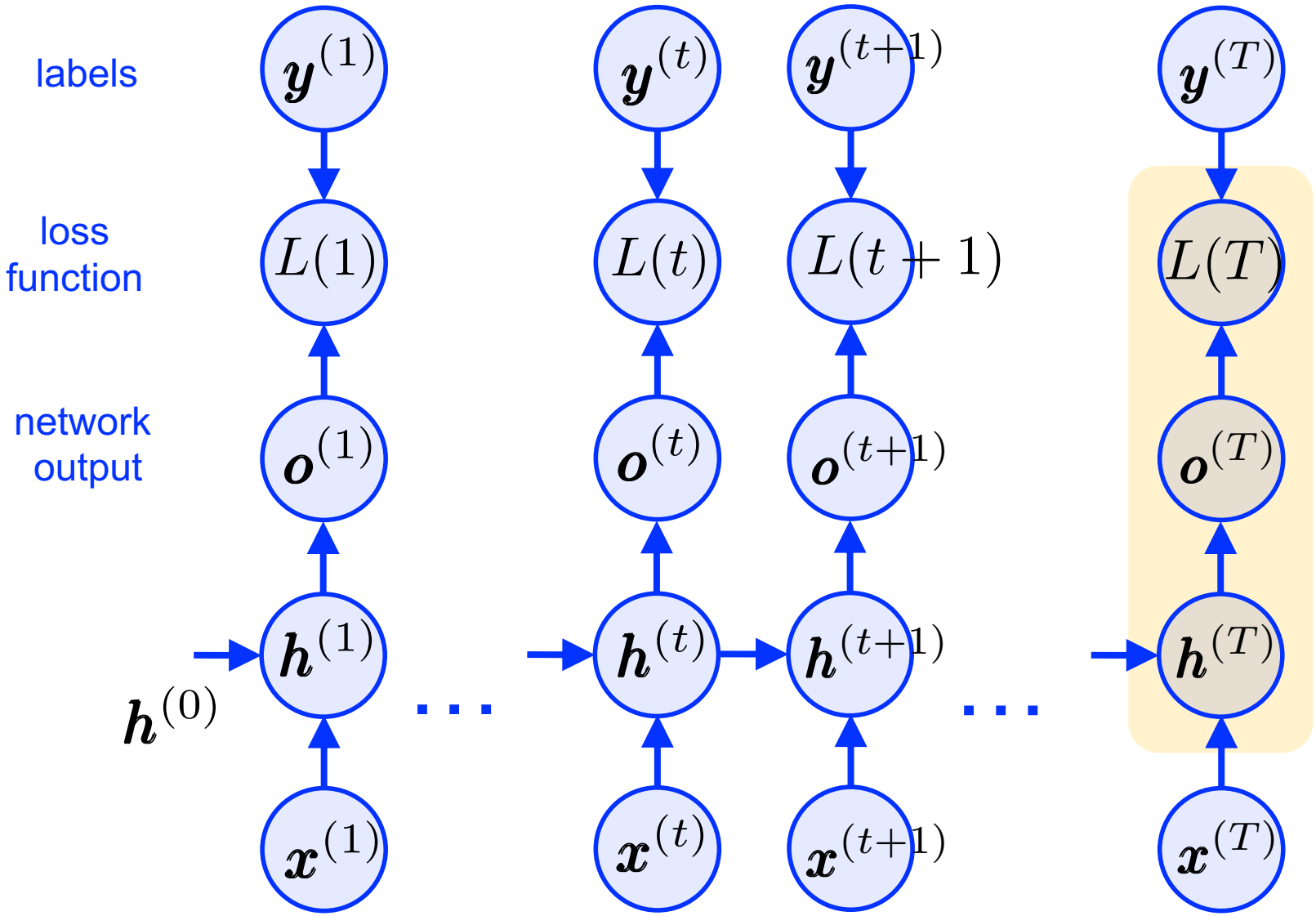
BTT – unfolded graph

$$\nabla_{\boldsymbol{o}^{(t)}} L = \hat{\boldsymbol{y}}^{(t)} - \boldsymbol{y}^{(t)}$$



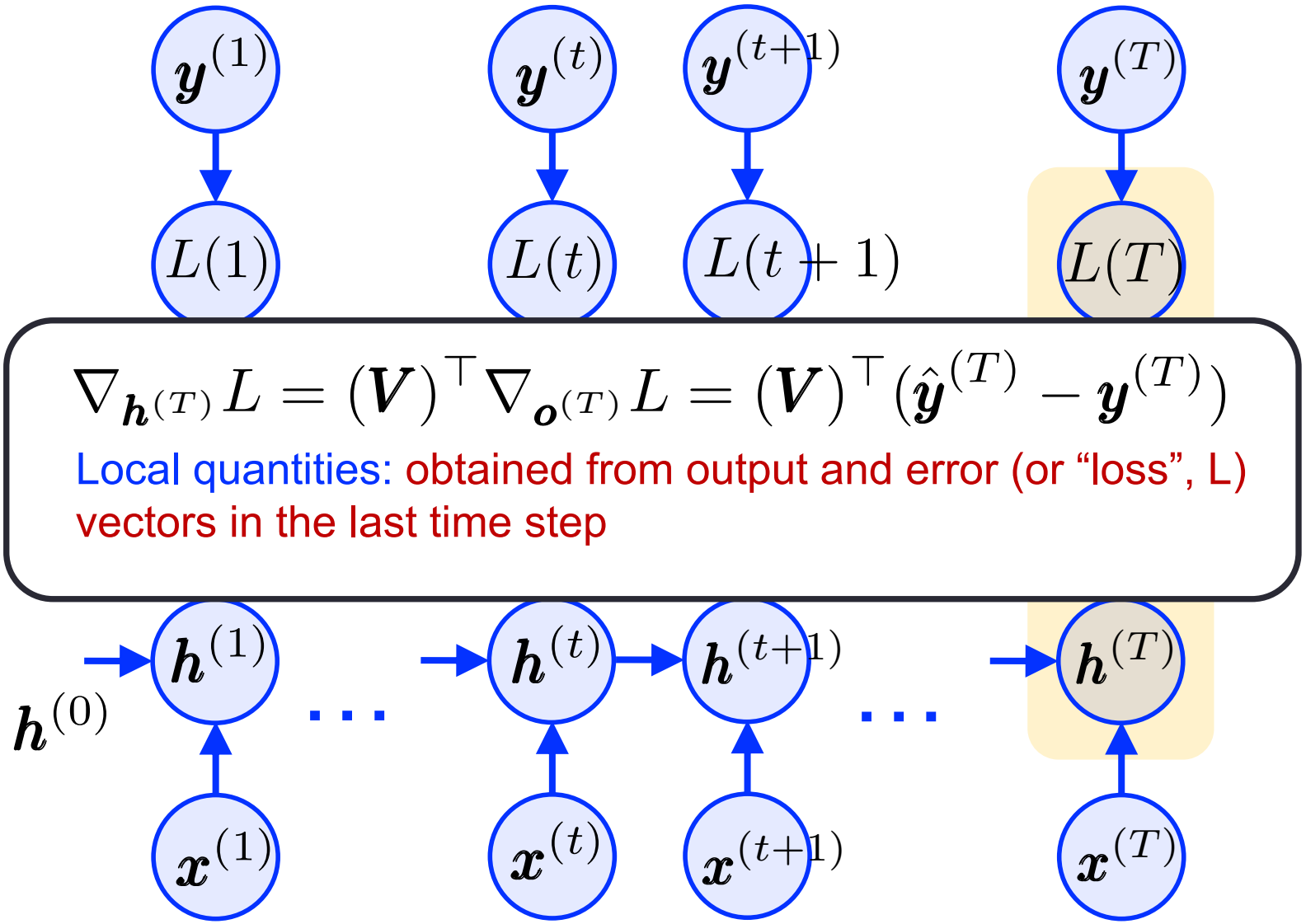
BTT – time T

$$\nabla_{\boldsymbol{h}^{(T)}} L = (\boldsymbol{V})^{\top} \nabla_{\boldsymbol{o}^{(T)}} L$$



BTT – time T

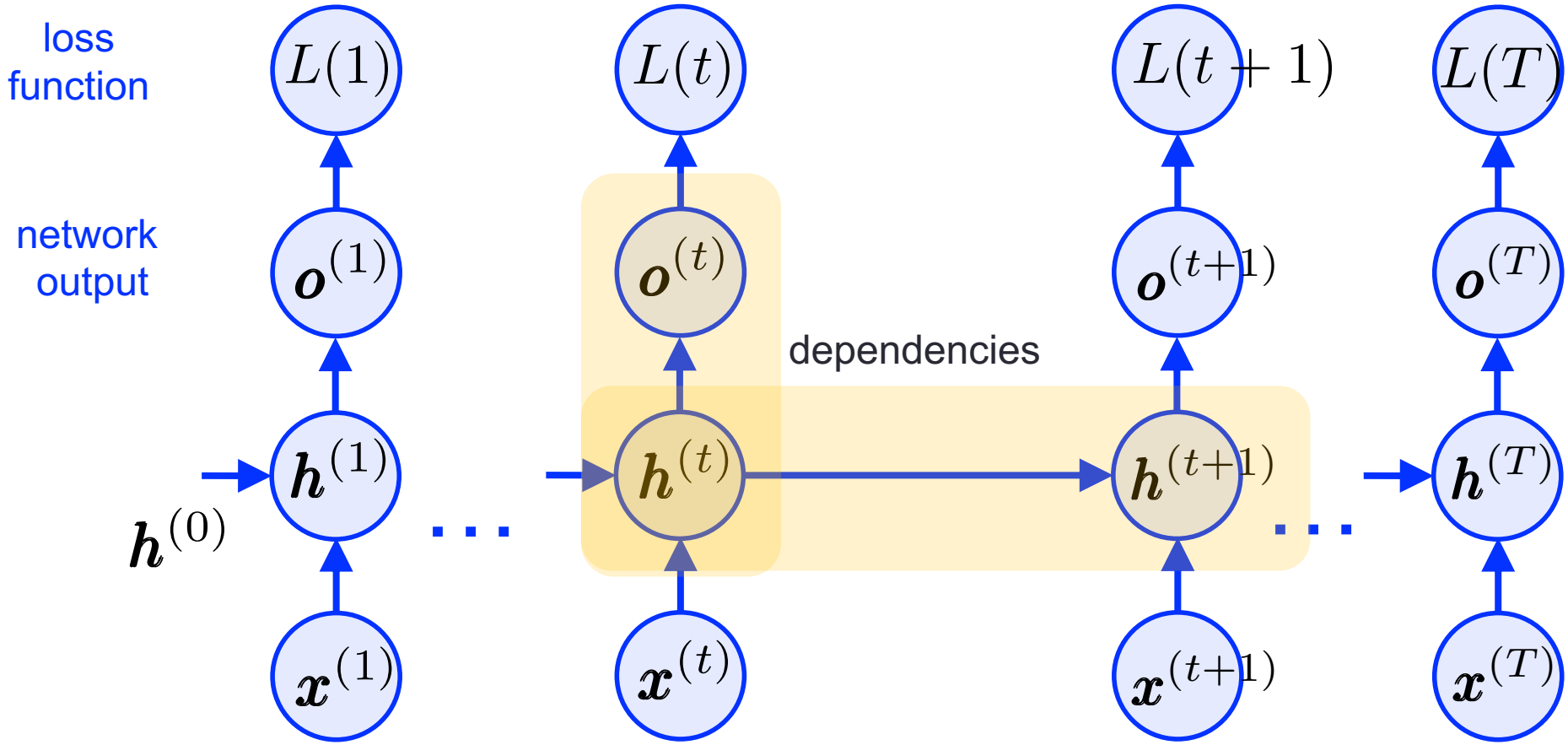
$$\nabla_{\boldsymbol{h}^{(T)}} L = (\boldsymbol{V})^\top \nabla_{\boldsymbol{o}^{(T)}} L$$



BTT – time t < T

From the chain rule of calculus

$$\frac{\partial L}{\partial h_i^{(t)}} = \sum_j \frac{\partial L}{\partial h_j^{(t+1)}} \frac{\partial h_j^{(t+1)}}{\partial h_i^{(t)}} + \sum_j \frac{\partial L}{\partial o_j^{(t)}} \frac{\partial o_j^{(t)}}{\partial h_i^{(t)}}$$

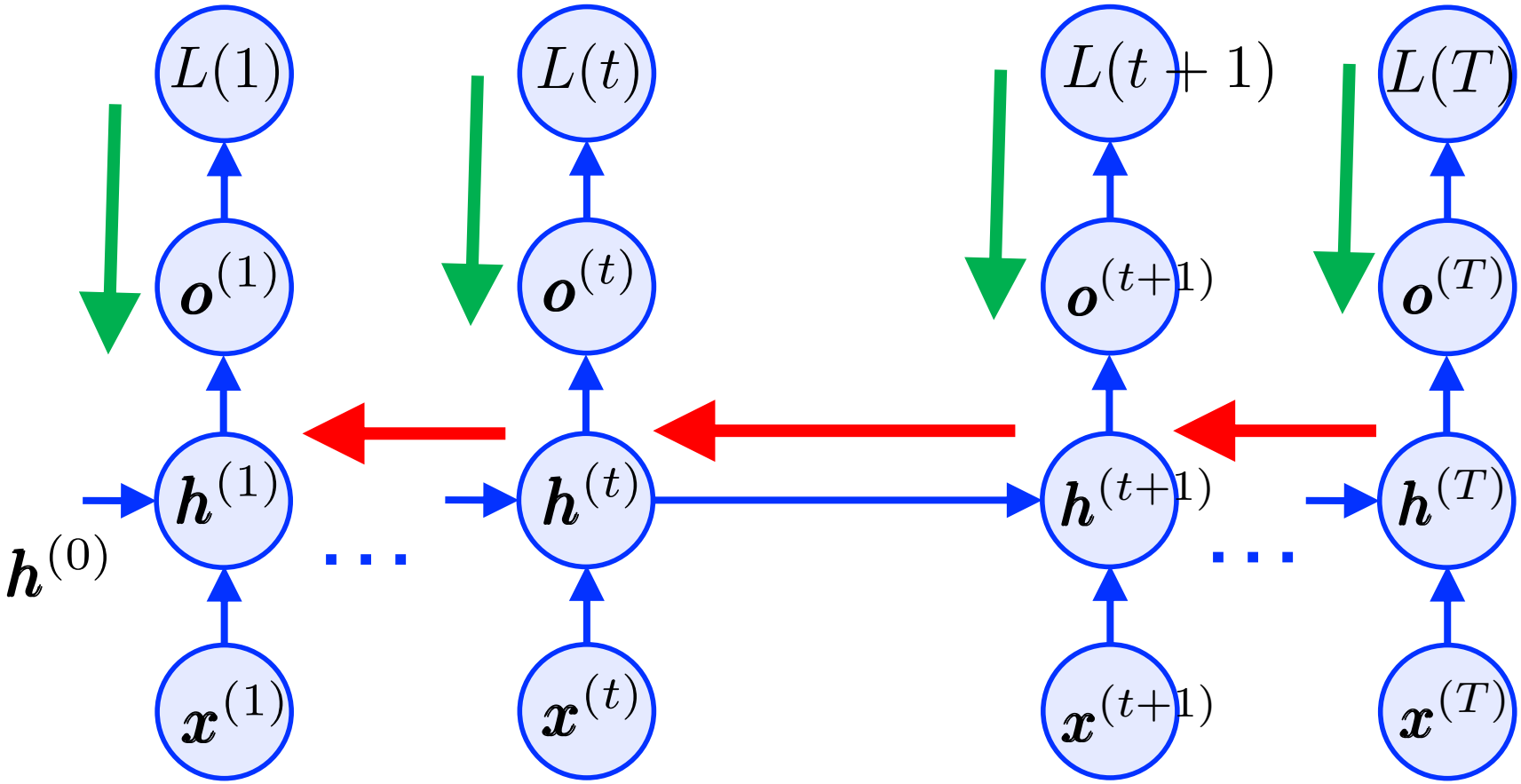


BTT – time t < T

gradient of $\mathbf{h}^{(t)}$ is computed recursively

$$\nabla_{\boldsymbol{h}^{(t)}} L = \boldsymbol{W}^\top \operatorname{diag}\left(1 - \left(\boldsymbol{h}^{(t+1)}\right)^2\right)(\nabla_{\boldsymbol{h}^{(t+1)}} L) + \boldsymbol{V}^\top(\nabla_{\boldsymbol{o}^{(t)}} L)$$

in compact form — recursion — local term



BTT – once we have the gradients wrt $\mathbf{h}^{(t)}$ and $\mathbf{o}^{(t)}$

The derivatives wrt the network params are [Rossi22]

$$\nabla_{\boldsymbol{V}} L = \sum_{t=1}^{T} (\nabla_{\boldsymbol{o}^{(t)}} L)(\boldsymbol{h}^{(t)})^{\top}$$

$$\nabla_{\boldsymbol{c}} L = \sum_{t=1}^{T} \nabla_{\boldsymbol{o}^{(t)}} L$$

$$\nabla_{\boldsymbol{W}} L = \sum_{t=1}^{T} \mathrm{diag}\left(1 - \left(\boldsymbol{h}^{(t)}\right)^2\right) (\nabla_{\boldsymbol{h}^{(t)}} L)(\boldsymbol{h}^{(t-1)})^{\top}$$

$$\nabla_{\boldsymbol{U}} L = \sum_{t=1}^{T} \mathrm{diag}\left(1 - \left(\boldsymbol{h}^{(t)}\right)^2\right) (\nabla_{\boldsymbol{h}^{(t)}} L)(\boldsymbol{x}^{(t)})^{\top}$$

$$\nabla_{\boldsymbol{b}} L = \sum_{t=1}^{T} \mathrm{diag}\left(1 - \left(\boldsymbol{h}^{(t)}\right)^2\right) (\nabla_{\boldsymbol{h}^{(t)}} L)$$

Let's have a closer look at BTT

- Let the RNN state **h**(t) be a column vector with H elements, as

$$\boldsymbol{h}^{(t)} = \begin{bmatrix} h_0^{(t)} & h_1^{(t)} & \cdots & h_H^{(t)} \end{bmatrix}^\top$$

- Applying the chain rule on the unfolded graph, we get

$$\frac{\partial L}{\partial h_i^{(t)}} = \sum_j \frac{\partial L}{\partial h_j^{(t+1)}} \frac{\partial h_j^{(t+1)}}{\partial h_i^{(t)}} + \sum_j \frac{\partial L}{\partial o_j^{(t)}} \frac{\partial o_j^{(t)}}{\partial h_i^{(t)}} \quad (1)$$

- In vector form

our focus (recursion)

$$\nabla_{\boldsymbol{h}^{(t)}} L = \left(\frac{\partial \boldsymbol{h}^{(t+1)}}{\partial \boldsymbol{h}^{(t)}} \right)^\top (\nabla_{\boldsymbol{h}^{(t+1)}} L) + \left(\frac{\partial \boldsymbol{o}^{(t)}}{\partial \boldsymbol{h}^{(t)}} \right)^\top (\nabla_{\boldsymbol{o}^{(t)}} L)$$

Let's have a closer look at BTT

- Let:

$$\boldsymbol{h}^{(t)} = [h_1^{(t)}\ h_2^{(t)} \cdots h_H^{(t)}]^\top$$

$$h_i^{(t+1)} = \tanh(a_i^{(t+1)})$$

$$a_i^{(t+1)} = (\boldsymbol{W}\boldsymbol{h}^{(t)} + \boldsymbol{U}\boldsymbol{x}^{(t+1)} + \boldsymbol{b})_i$$

- From these relations, it descends

$$\begin{cases} \dfrac{\partial h_i^{(t+1)}}{\partial a_k^{(t+1)}} = \begin{cases} 1 - (h_i^{(t+1)})^2 & i = k \\ 0 & i \neq k \end{cases} & (2) \\ \dfrac{\partial a_i^{(t+1)}}{\partial h_j^{(t)}} = w_{ij} & (3) \end{cases}$$

Let's have a closer look at BTT

Using (2) and (3) into Eq. (1)

non zero terms for i=k (see Eq. (2))

$$\frac{\partial h_i^{(t+1)}}{\partial h_j^{(t)}} = \sum_k \frac{\partial h_i^{(t+1)}}{\partial a_k^{(t+1)}} \frac{\partial a_k^{(t+1)}}{\partial h_j^{(t)}} = (1 - (h_i^{(t+1)})^2) \frac{\partial a_i^{(t+1)}}{\partial h_j^{(t)}} = (1 - (h_i^{(t+1)})^2) w_{ij}$$

Rewriting this equation in vector form

$$\frac{\partial \boldsymbol{h}^{(t+1)}}{\partial \boldsymbol{a}^{(t+1)}} = \frac{\partial \tanh(\boldsymbol{a}^{(t+1)})}{\partial \boldsymbol{a}^{(t+1)}} = \mathrm{diag}(1 - (\boldsymbol{h}^{(t+1)})^2)$$

$$\frac{\partial \boldsymbol{h}^{(t+1)}}{\partial \boldsymbol{h}^{(t)}} = \mathrm{diag}(1 - (\boldsymbol{h}^{(t+1)})^2)\boldsymbol{W} \qquad (4)$$

Let's have a closer look at BTT

$$\nabla_{\boldsymbol{h}^{(t)}} L = \left(\frac{\partial \boldsymbol{h}^{(t+1)}}{\partial \boldsymbol{h}^{(t)}}\right)^{\top} \nabla_{\boldsymbol{h}^{(t+1)}} L + \left(\frac{\partial \boldsymbol{o}^{(t)}}{\partial \boldsymbol{h}^{(t)}}\right)^{\top} (\nabla_{\boldsymbol{o}^{(t)}} L) =$$

$$= \boldsymbol{W}^{\top} \mathrm{diag}(1 - (\boldsymbol{h}^{(t+1)})^2) \nabla_{\boldsymbol{h}^{(t+1)}} L + \boldsymbol{V}^{\top} (\hat{\boldsymbol{y}}^{(t)} - \boldsymbol{y}^{(t)})$$

using (4)

(5)

- Developing the (forward) recursion

$$\nabla_{\boldsymbol{h}^{(t)}} L = \sum_{k=t+1}^{T} f(k) (\boldsymbol{V}^{\top} (\hat{\boldsymbol{y}}^{(k)} - \boldsymbol{y}^{(k)})) + \boldsymbol{V}^{\top} (\hat{\boldsymbol{y}}^{(t)} - \boldsymbol{y}^{(t)})$$

(6)

$$f(k) \triangleq \prod_{s=t+1}^{k} \boldsymbol{W}^{\top} \mathrm{diag}(1 - (\boldsymbol{h}^{(s)})^2)$$

Vanishing gradients

- The error gradient from time step k (from time t+1 to T)
  - is back-propagated to time t
  - weighted by function f(k)
  - f(k) may have a detrimental effect

gradient from vertical branch at time k>t

gradient from vertical branch at time t

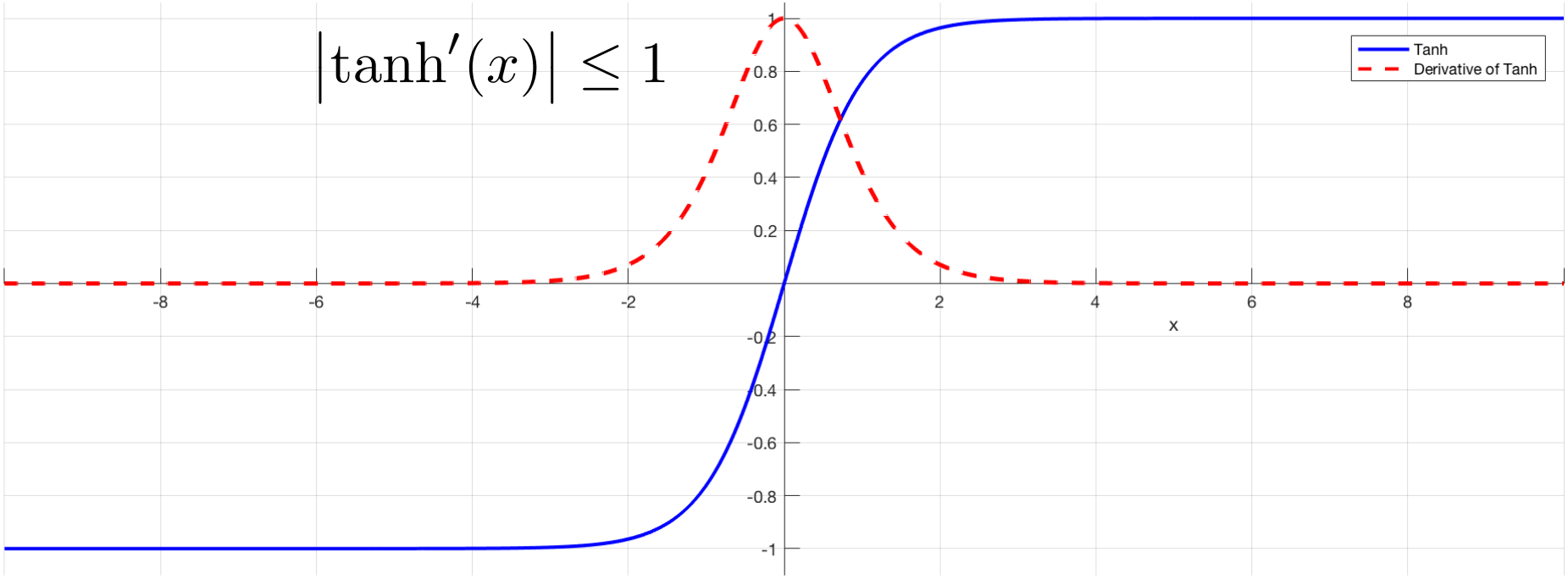
$$\nabla_{\boldsymbol{h}^{(t)}} L = \sum_{k=t+1}^{T} f(k)(\boldsymbol{V}^\top(\hat{\boldsymbol{y}}^{(k)} - \boldsymbol{y}^{(k)})) + \boldsymbol{V}^\top(\hat{\boldsymbol{y}}^{(t)} - \boldsymbol{y}^{(t)})$$

$$f(k) \triangleq \prod_{s=t+1}^{k} \boldsymbol{W}^\top \mathrm{diag}(1 - (\boldsymbol{h}^{(s)})^2)$$

$$\tanh(x) = \frac{\sinh(x)}{\cosh(x)} = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

$$\tanh'(x) = 1 - \left(\frac{e^x - e^{-x}}{e^x + e^{-x}}\right)^2 = 1 - (\tanh(x))^2$$

$$\left|\tanh'(x)\right| \leq 1$$



Vanishing gradients

- Let now compute

$$\|f(k)\| = \left\| \prod_{s=t+1}^{k} \boldsymbol{W}^\top \mathrm{diag}(1 - (\boldsymbol{h}^{(s)})^2) \right\| \leq$$

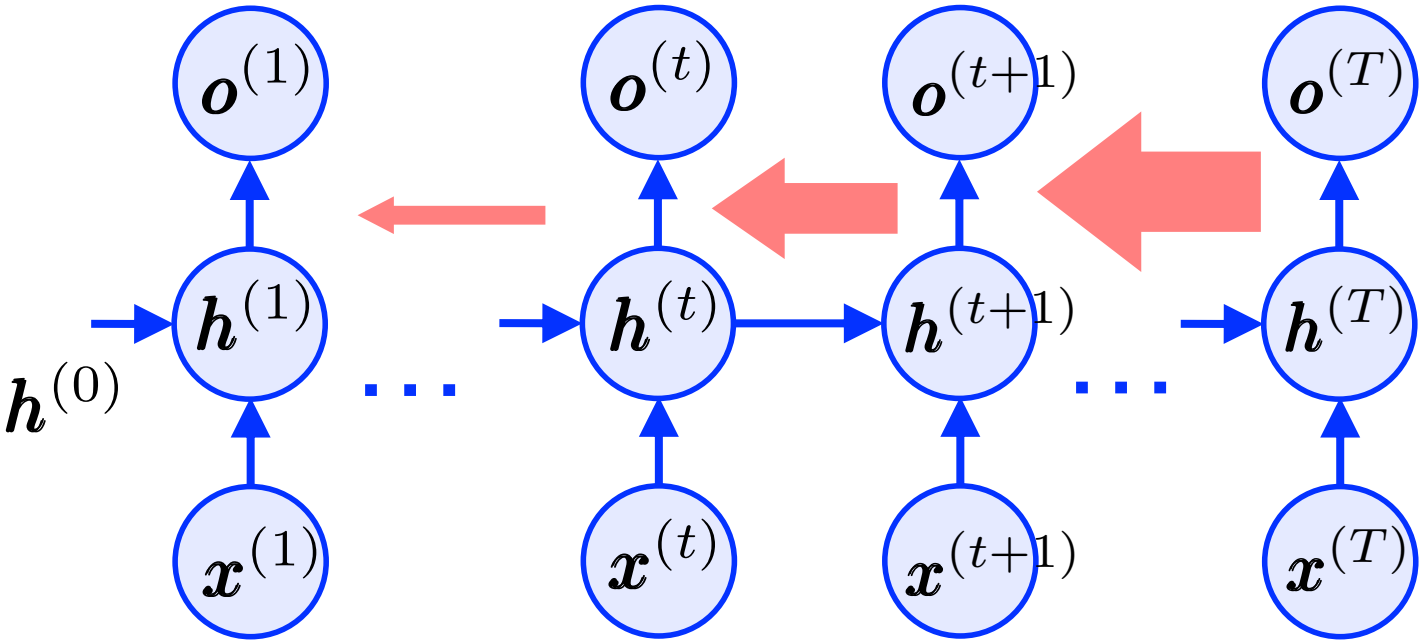
$$\leq \prod_{s=t+1}^{k} \|\boldsymbol{W}^\top\| \|\mathrm{diag}(1 - (\boldsymbol{h}^{(s)})^2)\| \leq \sigma_{\max}^{k-t}$$

- where $\sigma_{\max} = \sqrt{\lambda_{\max}(\boldsymbol{W}^\top \boldsymbol{W})}$
  - is the largest singular value of **W**
  - is referred to as the spectral norm of matrix **W**

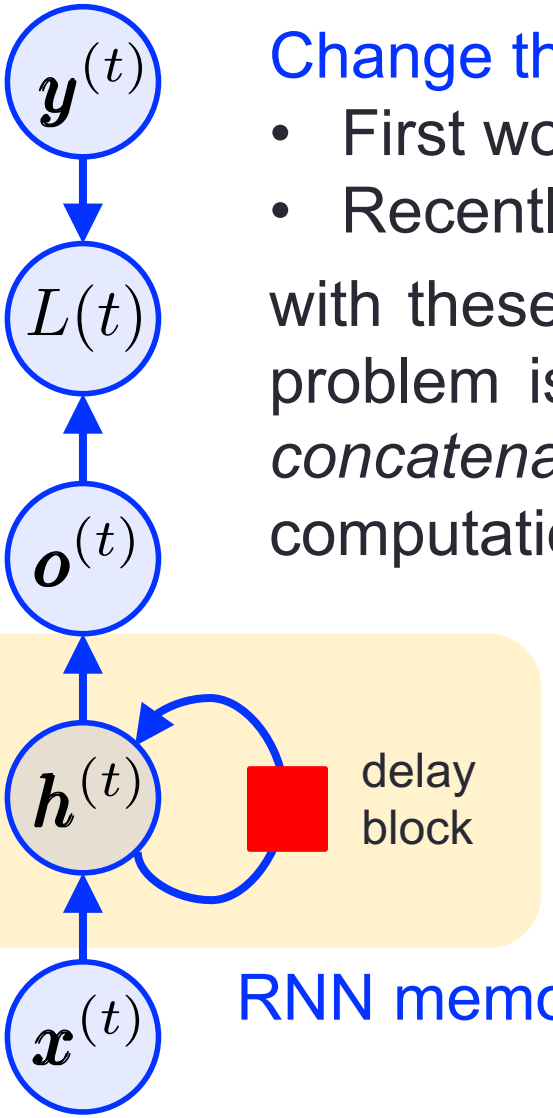
Vanishing gradients

$$\|f(k)\| \leq \prod_{s=t+1}^{k} \|\boldsymbol{W}^T\| \|\text{diag}(1 - (\boldsymbol{h}^{(s)})^2)\| \leq \sigma_{\max}^{k-t}$$

- If $\sigma_{\max} < 1$ gradients will vanish for sufficiently large k-t…



Solution – new memory cells



RNN memory cell

Change the structure of the RNN memory cell

- First work LSTM [Hochreiter97]
- Recently Gated Recurrent Units (GRU) [Cho14]

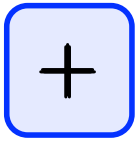
with these new cell structures the gradient vanishing problem is solved: tanh does no longer appear *in a concatenated (recursive) fashion* in the gradient computation

GRU cells in four steps…

Definitions

hyperbolic tangent (applied element-wise)

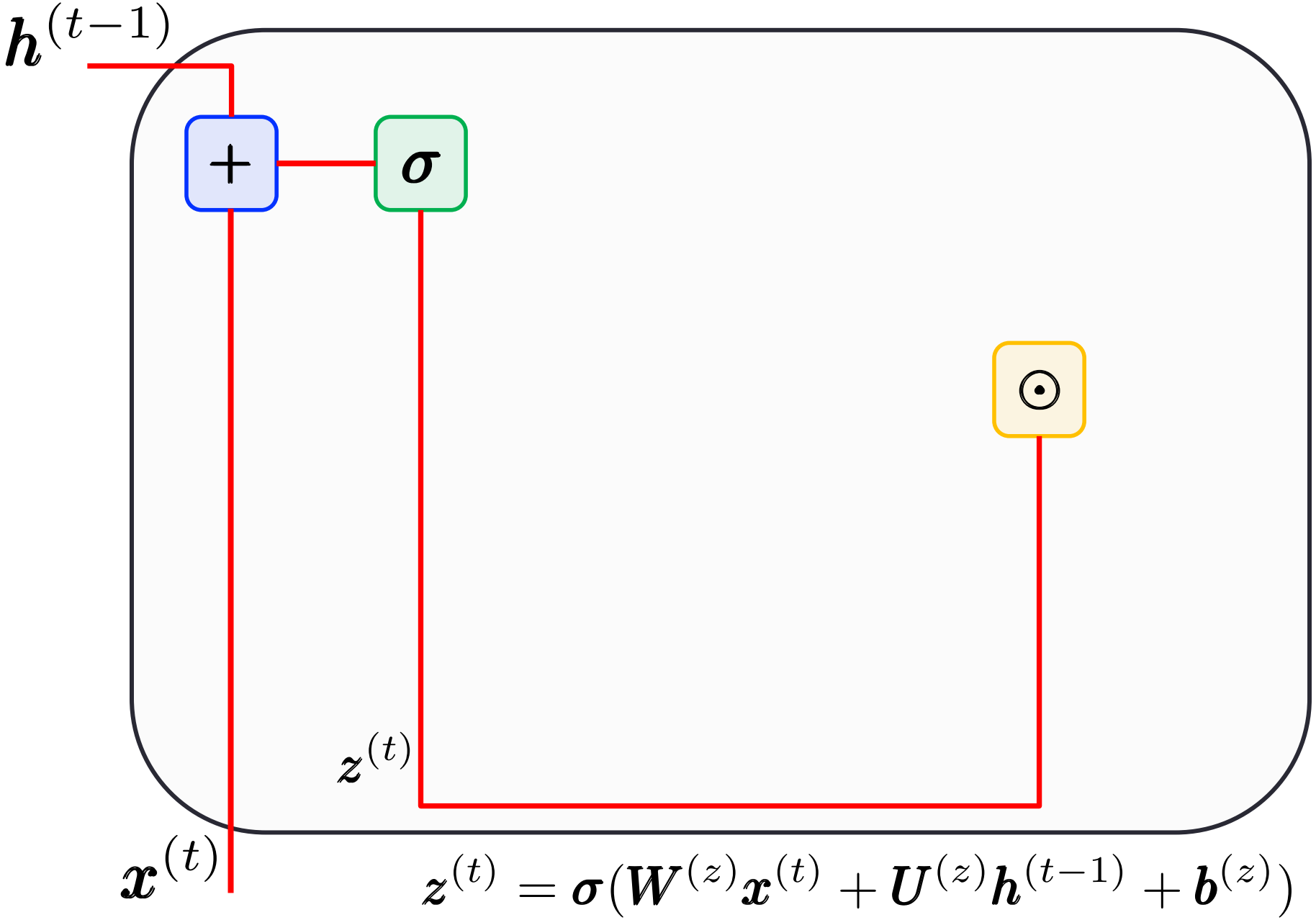
sigmoid nonlinearity (applied element-wise)



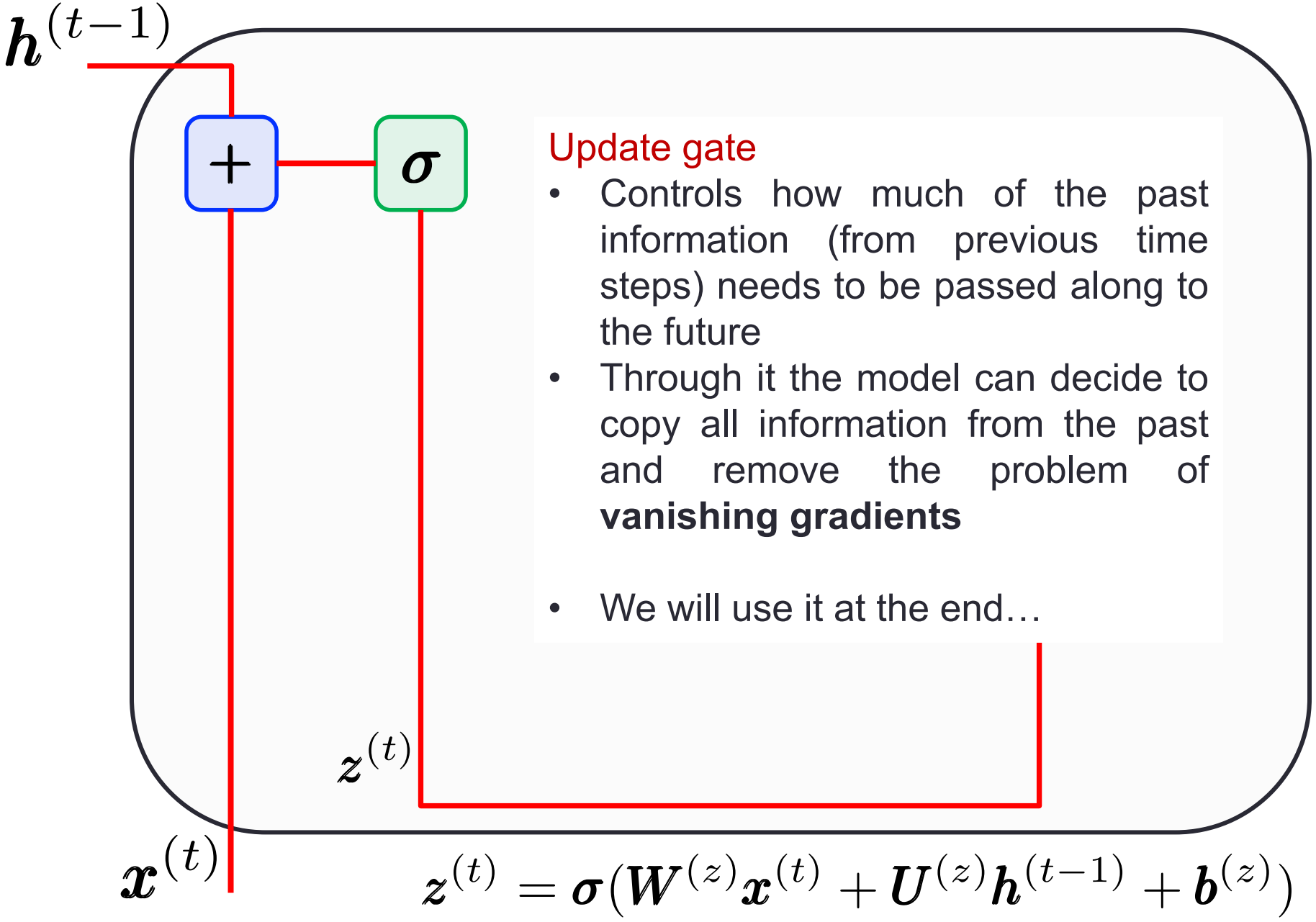
vector addition

element-wise product between vectors
(also known as Hadamard product)

Step 1: update gate



Step 1: update gate



Update gate

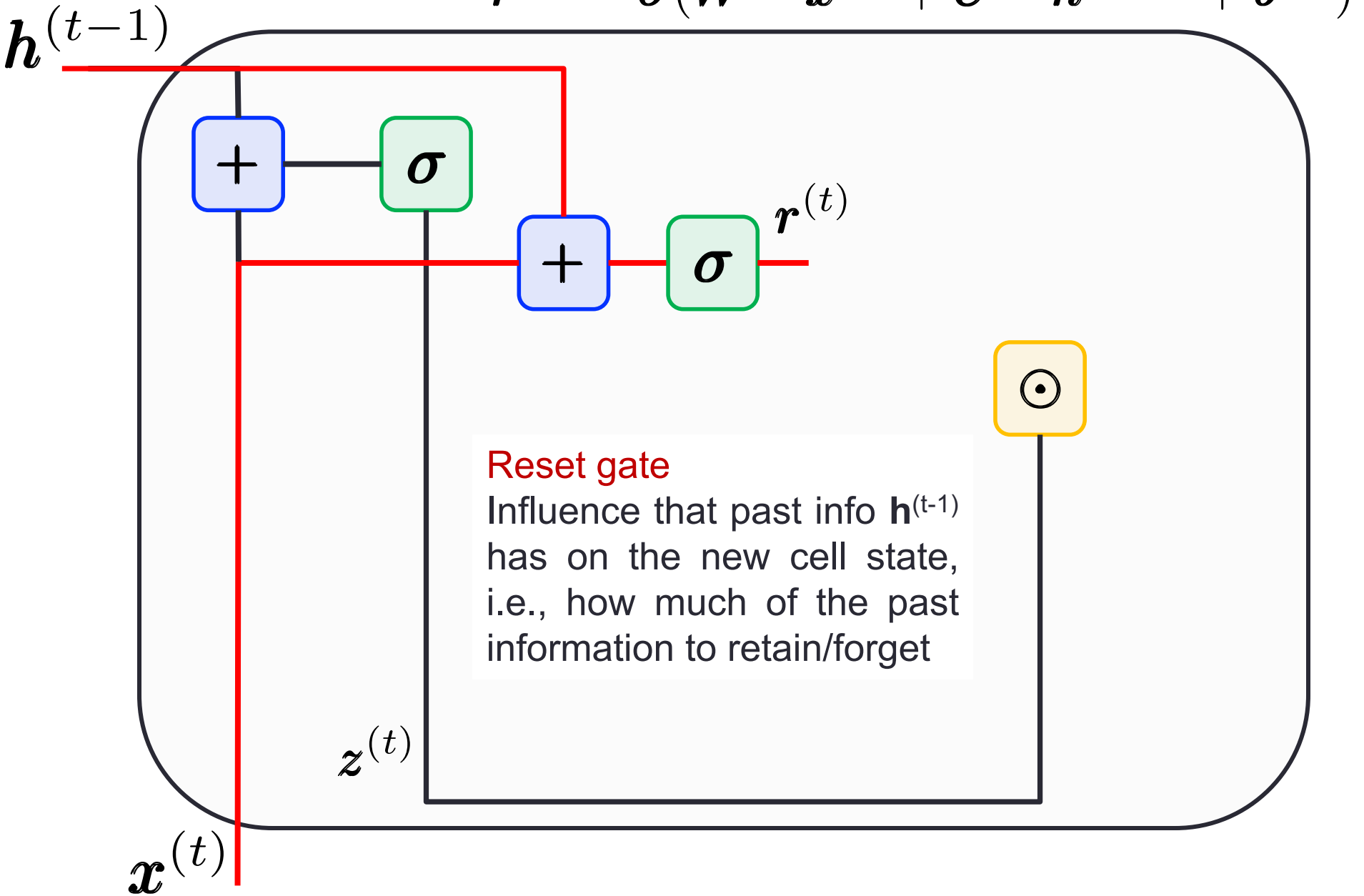
- Controls how much of the past information (from previous time steps) needs to be passed along to the future
- Through it the model can decide to copy all information from the past and remove the problem of **vanishing gradients**

- We will use it at the end…

$$z^{(t)} = \sigma(W^{(z)}x^{(t)} + U^{(z)}h^{(t-1)} + b^{(z)})$$

Step 2: reset gate

$$\boldsymbol{r}^{(t)} = \boldsymbol{\sigma}(\boldsymbol{W}^{(r)}\boldsymbol{x}^{(t)} + \boldsymbol{U}^{(r)}\boldsymbol{h}^{(t-1)} + \boldsymbol{b}^{(r)})$$

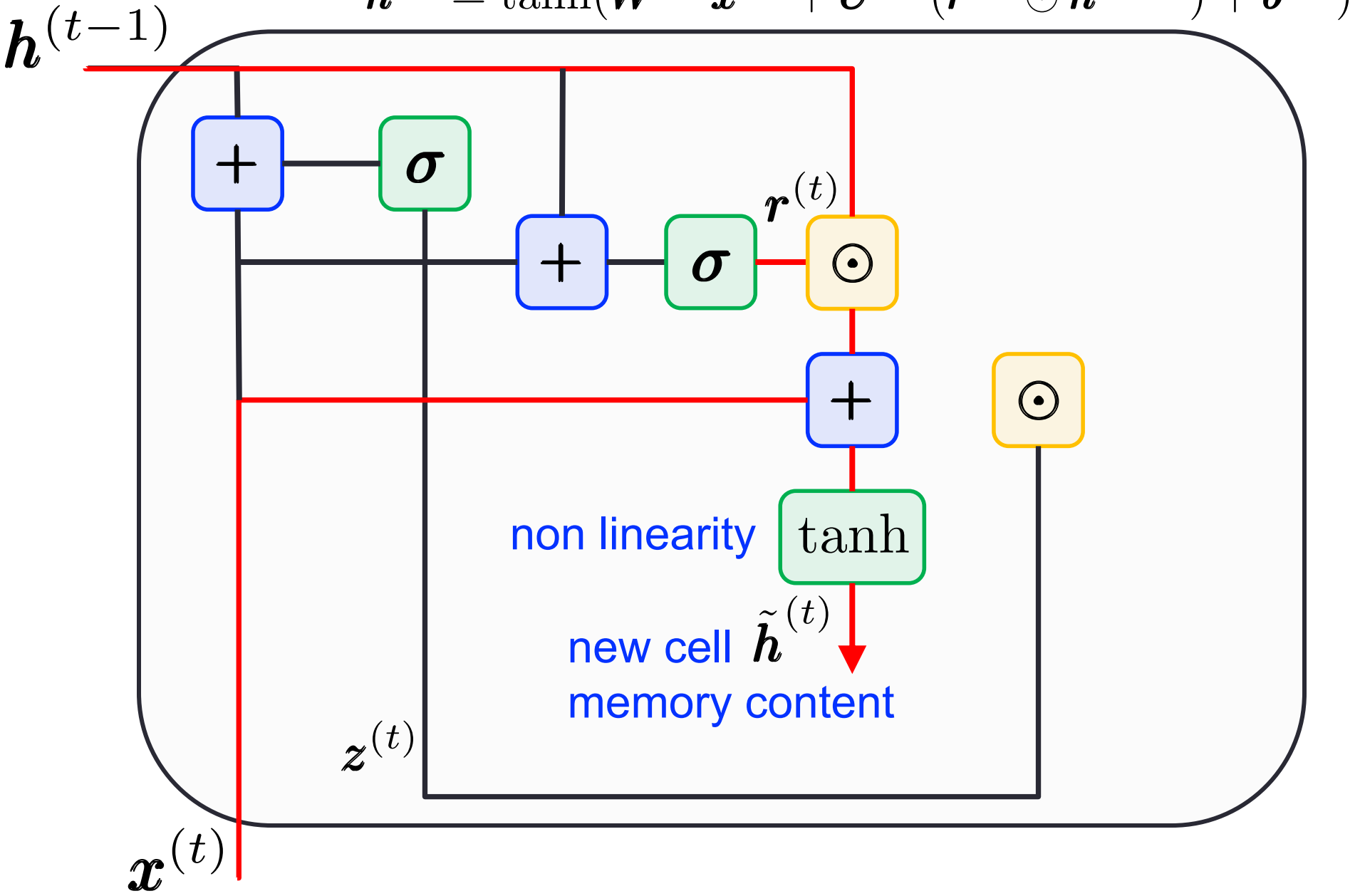


Reset gate

Influence that past info $\mathbf{h}^{(t-1)}$ has on the new cell state, i.e., how much of the past information to retain/forget

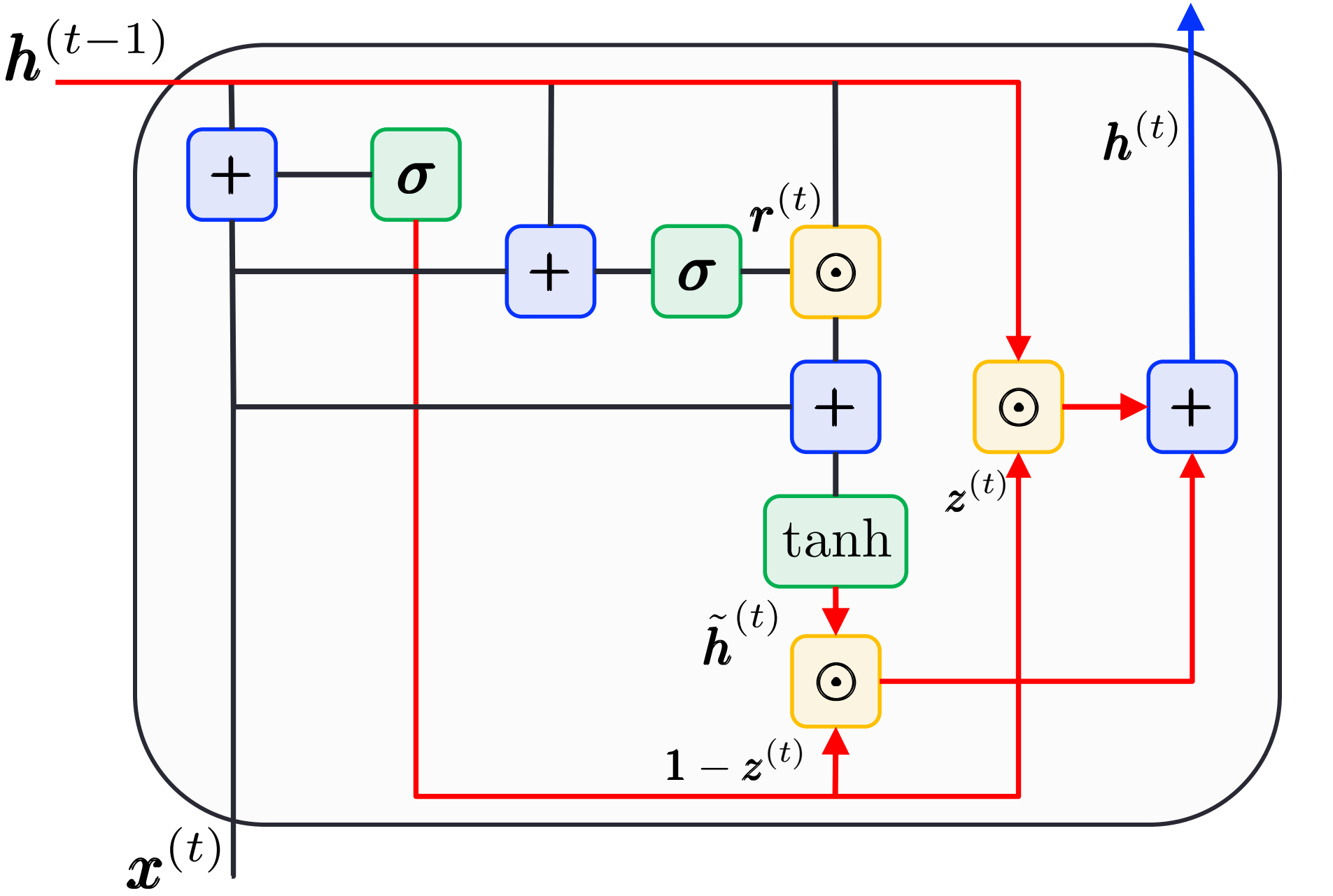
Step 3: current memory content

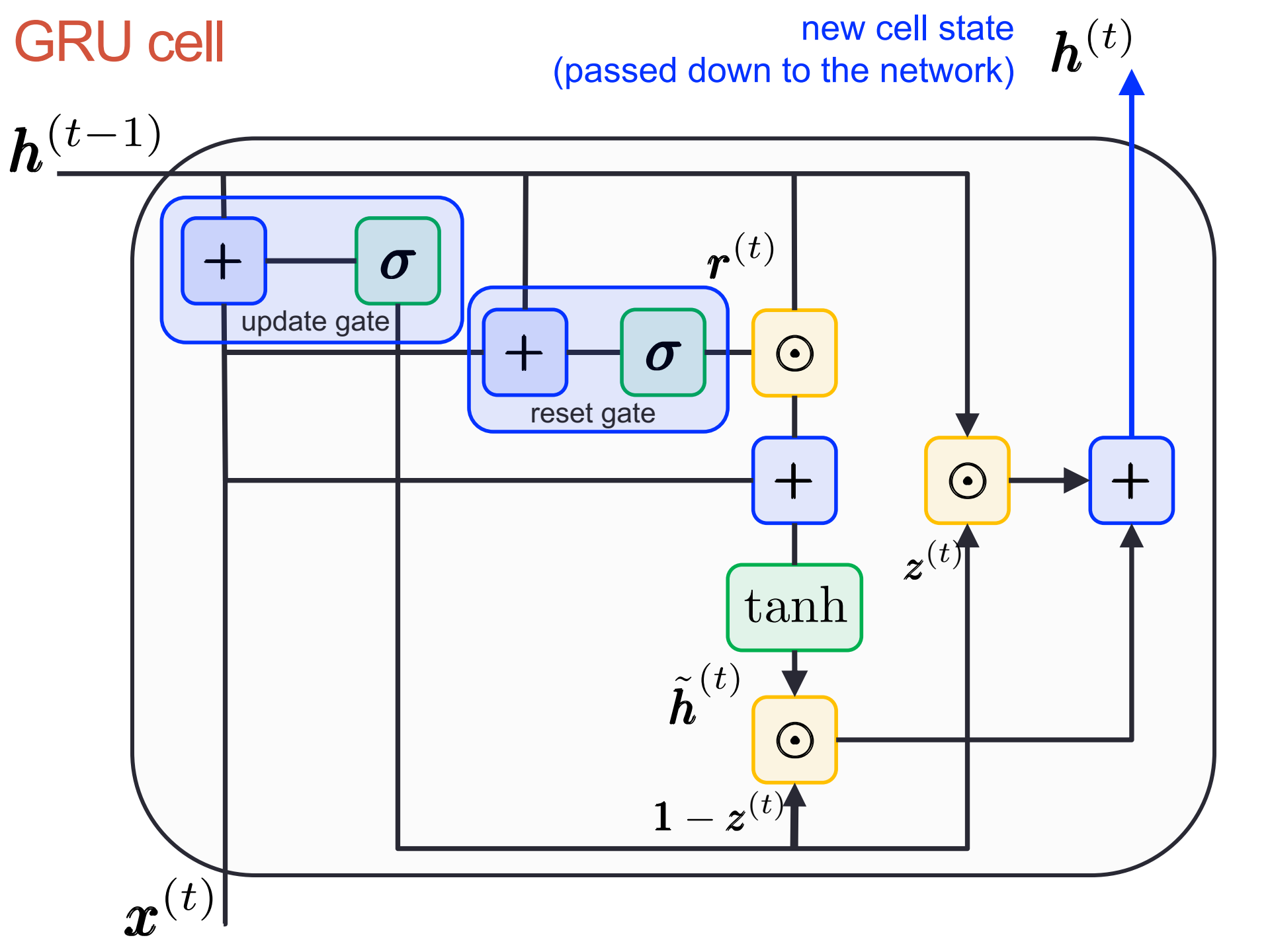
$$\tilde{\boldsymbol{h}}^{(t)} = \tanh(\boldsymbol{W}^{(h)}\boldsymbol{x}^{(t)} + \boldsymbol{U}^{(h)}(\boldsymbol{r}^{(t)} \odot \boldsymbol{h}^{(t-1)}) + \boldsymbol{b}^{(h)})$$



Step 4: final state

$$\boldsymbol{h}^{(t)} = \boldsymbol{z}^{(t)} \odot \boldsymbol{h}^{(t-1)} + (1 - \boldsymbol{z}^{(t)}) \odot \tilde{\boldsymbol{h}}^{(t)}$$




GRU cell
new cell state
(passed down to the network)
$h^{(t)}$
$h^{(t-1)}$
σ
update gate
$r^{(t)}$
σ
reset gate
tanh
$z^{(t)}$
$\tilde{h}^{(t)}$
$1 - z^{(t)}$
$x^{(t)}$

GRU cell "fully gated"

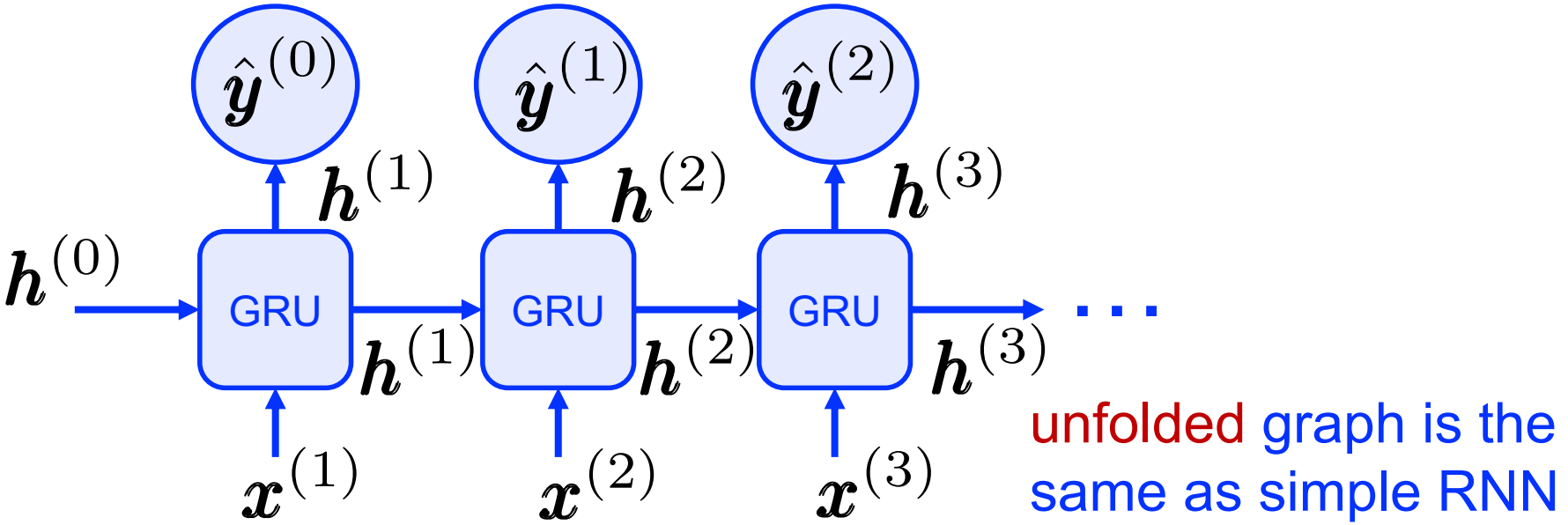
$$\boldsymbol{z}^{(t)} = \sigma(\boldsymbol{W}^{(z)}\boldsymbol{x}^{(t)} + \boldsymbol{U}^{(z)}\boldsymbol{h}^{(t-1)} + \boldsymbol{b}^{(z)})$$

$$\boldsymbol{r}^{(t)} = \sigma(\boldsymbol{W}^{(r)}\boldsymbol{x}^{(t)} + \boldsymbol{U}^{(r)}\boldsymbol{h}^{(t-1)} + \boldsymbol{b}^{(r)})$$

$$\tilde{\boldsymbol{h}}^{(t)} = \tanh(\boldsymbol{W}^{(h)}\boldsymbol{x}^{(t)} + \boldsymbol{U}^{(h)}(\boldsymbol{r}^{(t)} \odot \boldsymbol{h}^{(t-1)}) + \boldsymbol{b}^{(h)})$$

$$\boldsymbol{h}^{(t)} = \boldsymbol{z}^{(t)} \odot \boldsymbol{h}^{(t-1)} + (1 - \boldsymbol{z}^{(t)})\tilde{\boldsymbol{h}}^{(t)}$$

fully gated unit



unfolded graph is the same as simple RNN

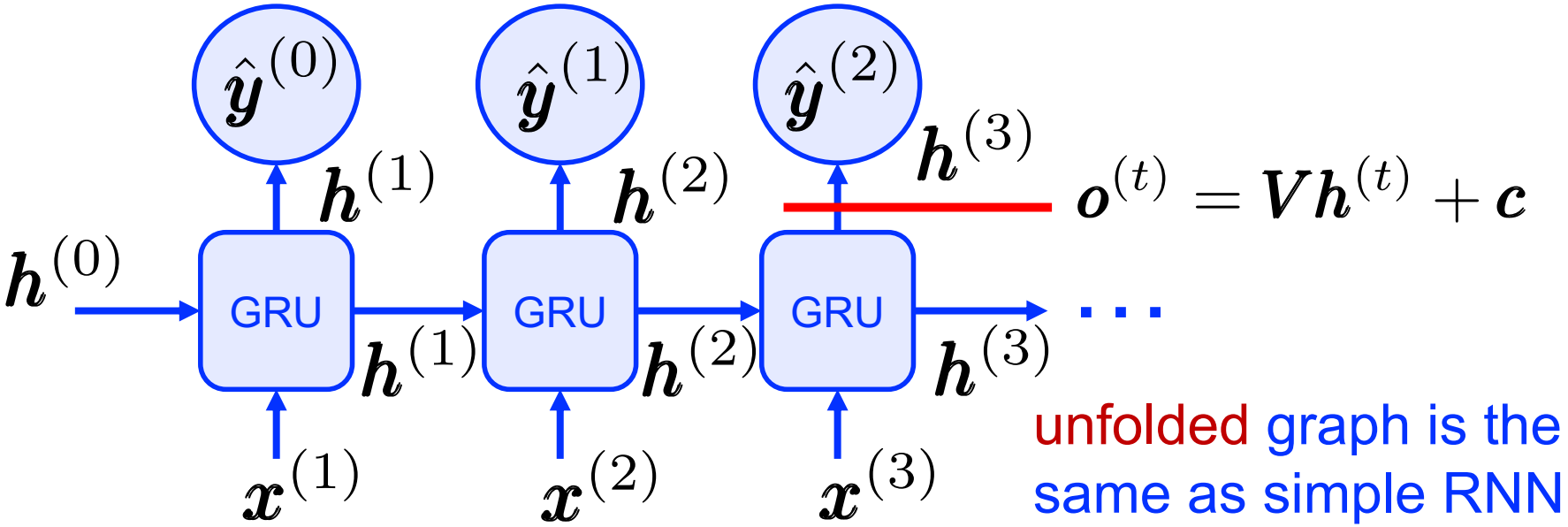
GRU cell "fully gated"

$$\boldsymbol{z}^{(t)} = \sigma(\boldsymbol{W}^{(z)}\boldsymbol{x}^{(t)} + \boldsymbol{U}^{(z)}\boldsymbol{h}^{(t-1)} + \boldsymbol{b}^{(z)})$$

$$\boldsymbol{r}^{(t)} = \sigma(\boldsymbol{W}^{(r)}\boldsymbol{x}^{(t)} + \boldsymbol{U}^{(r)}\boldsymbol{h}^{(t-1)} + \boldsymbol{b}^{(r)})$$

$$\tilde{\boldsymbol{h}}^{(t)} = \tanh(\boldsymbol{W}^{(h)}\boldsymbol{x}^{(t)} + \boldsymbol{U}^{(h)}(\boldsymbol{r}^{(t)} \odot \boldsymbol{h}^{(t-1)}) + \boldsymbol{b}^{(h)})$$

$$\boldsymbol{h}^{(t)} = \boldsymbol{z}^{(t)} \odot \boldsymbol{h}^{(t-1)} + (1 - \boldsymbol{z}^{(t)})\tilde{\boldsymbol{h}}^{(t)}$$



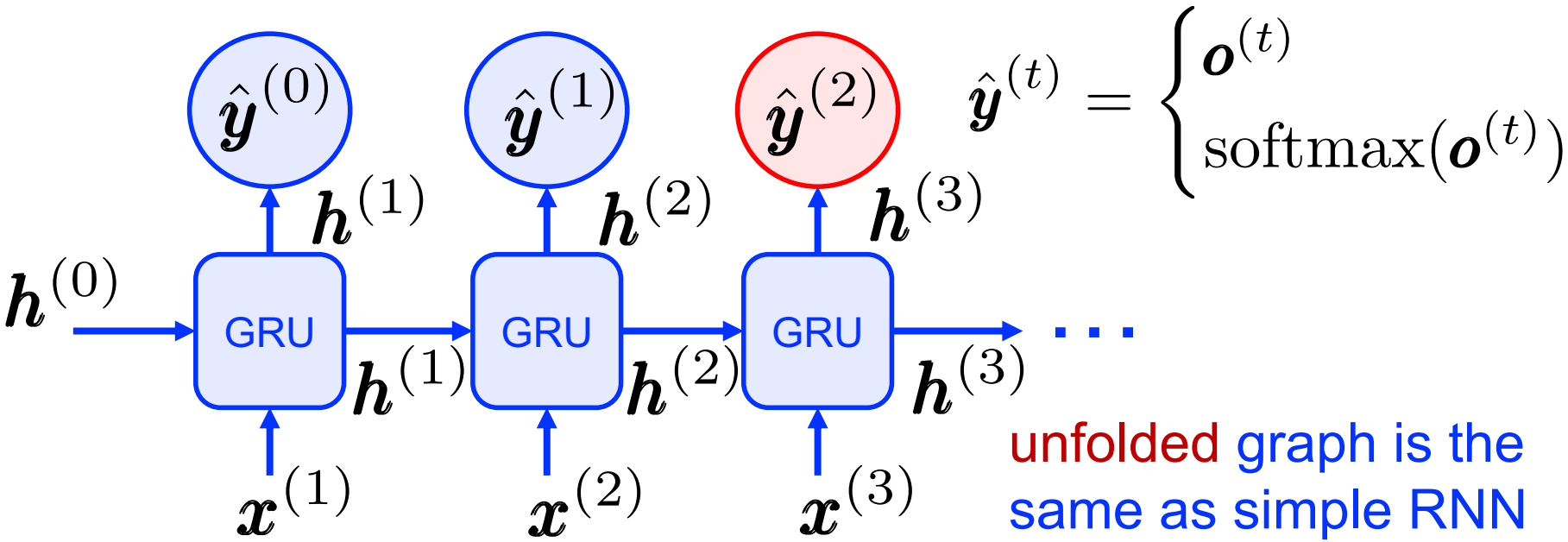
GRU cell "fully gated"

$$\boldsymbol{z}^{(t)} = \sigma(\boldsymbol{W}^{(z)}\boldsymbol{x}^{(t)} + \boldsymbol{U}^{(z)}\boldsymbol{h}^{(t-1)} + \boldsymbol{b}^{(z)})$$

$$\boldsymbol{r}^{(t)} = \sigma(\boldsymbol{W}^{(r)}\boldsymbol{x}^{(t)} + \boldsymbol{U}^{(r)}\boldsymbol{h}^{(t-1)} + \boldsymbol{b}^{(r)})$$

$$\tilde{\boldsymbol{h}}^{(t)} = \tanh(\boldsymbol{W}^{(h)}\boldsymbol{x}^{(t)} + \boldsymbol{U}^{(h)}(\boldsymbol{r}^{(t)} \odot \boldsymbol{h}^{(t-1)}) + \boldsymbol{b}^{(h)})$$

$$\boldsymbol{h}^{(t)} = \boldsymbol{z}^{(t)} \odot \boldsymbol{h}^{(t-1)} + (1 - \boldsymbol{z}^{(t)})\tilde{\boldsymbol{h}}^{(t)}$$



GRU "minimally gated unit"

$$
\begin{cases}
\boldsymbol{f}^{(t)} = \sigma_g(\boldsymbol{W}^{(f)}\boldsymbol{x}^{(t)} + \boldsymbol{U}^{(f)}\boldsymbol{h}^{(t-1)} + \boldsymbol{b}^{(f)}) \\
\tilde{\boldsymbol{h}}^{(t)} = \sigma_h(\boldsymbol{W}^{(h)}\boldsymbol{x}^{(t)} + \boldsymbol{U}^{(h)}(\boldsymbol{f}^{(t)} \odot \boldsymbol{h}^{(t-1)}) + \boldsymbol{b}^{(h)}) \\
\boldsymbol{h}^{(t)} = \boldsymbol{f}^{(t)} \odot \boldsymbol{h}^{(t-1)} + (1 - \boldsymbol{f}^{(t)}) \odot \tilde{\boldsymbol{h}}^{(t)}
\end{cases}
$$

$\sigma_g(\cdot)$: sigmoid

$\sigma_h(\cdot)$: tanh

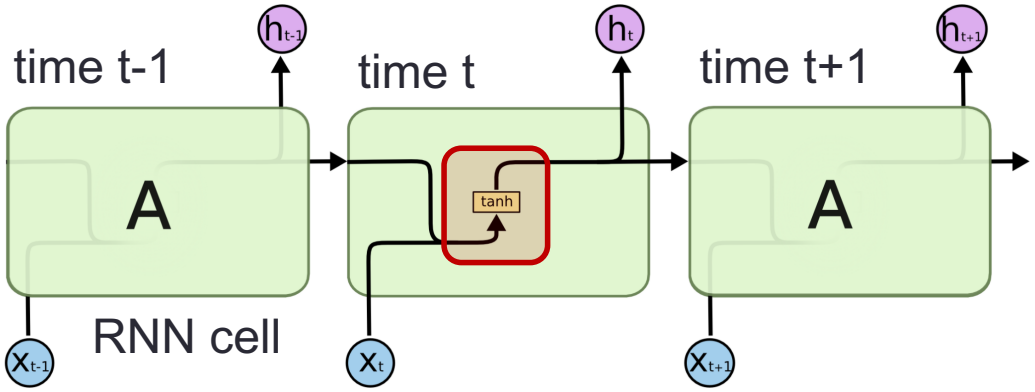
a single gate is used to control

- internal state (2nd equation)
- how much of the past info to let flow (3rd equation)

Wrap up

- Standard RNNs

Problems with *back propagation* of gradient due to the **tanh**



- LSTM/GRU-based RNNs

Gradient can now be back propagated and the RNN cell params can be used to control to which extent

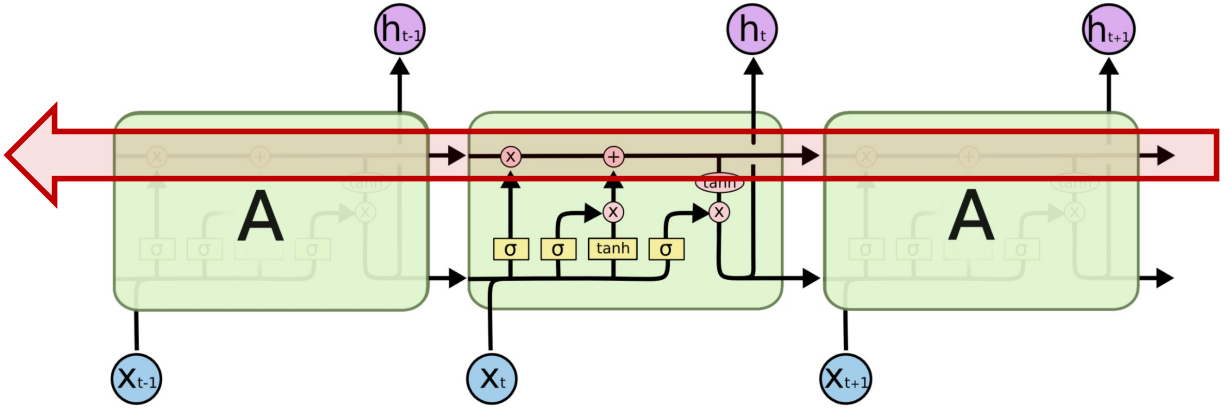
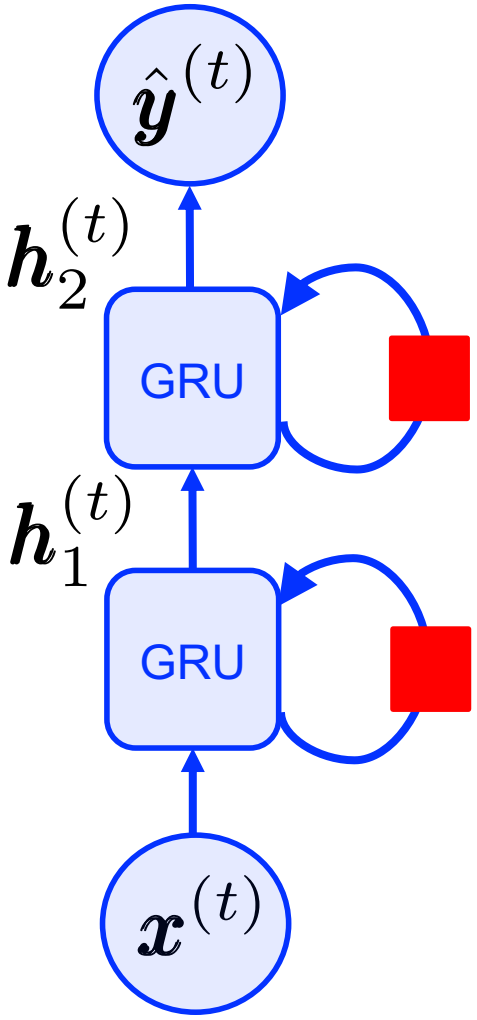


Figure: Christopher Olah, "Understanding LSTM Networks," 2022.

Deep RNN

- Obtained by just
  - Stacking multiple GRU cells
  - BTT still used to train it
- Multiple GRU layers
  - Usually lead to better results
  - More abstract (powerful) features
  - Right number of layers has to be tuned
    - Grid search for hyperparameters



Bi-directional RNN (1/2)

- Simple but powerful concept [Schuster97]
  - Obtained by combining two *independent* RNNs
- Input sequence
  - Fed in *normal* time 1,2,…,T order to *forward* network
  - Fed in *reverse* time T,T-1,…,1 order to *backward* network
- Output sequence
  - Sum of output from forward & backward RNN

[Schuster97] M. Schuster, K. K. Paliwal, "Bidirectional recurrent neural networks," *IEEE Transactions on Signal Processing*, Vol. 45, No. 11, 1997.

Bi-directional RNN (2/2)

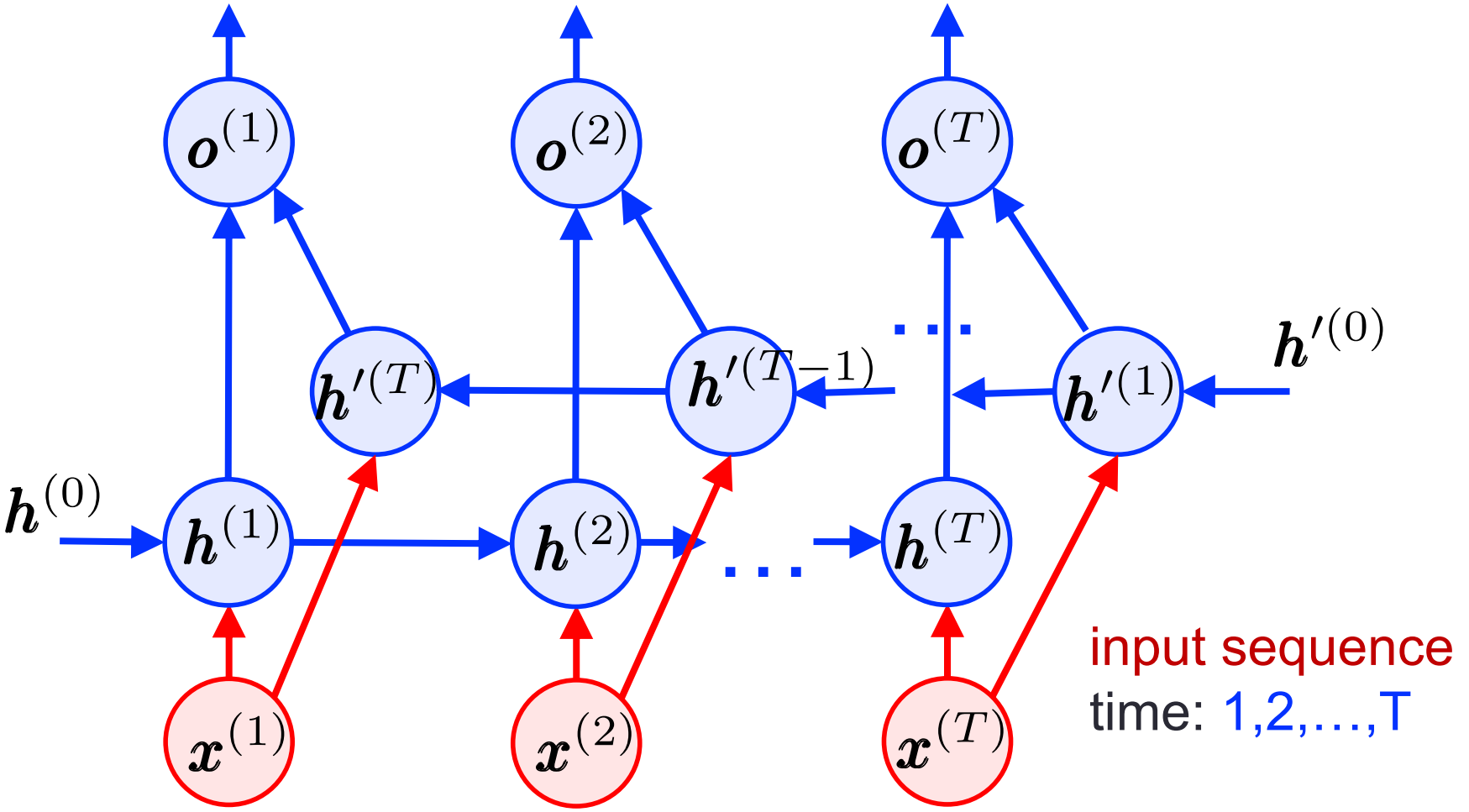
- Advantage
  - Connect present output to *past* and *future* samples
- Example from speech

  "I am really angry as I have been working out at the gym today"

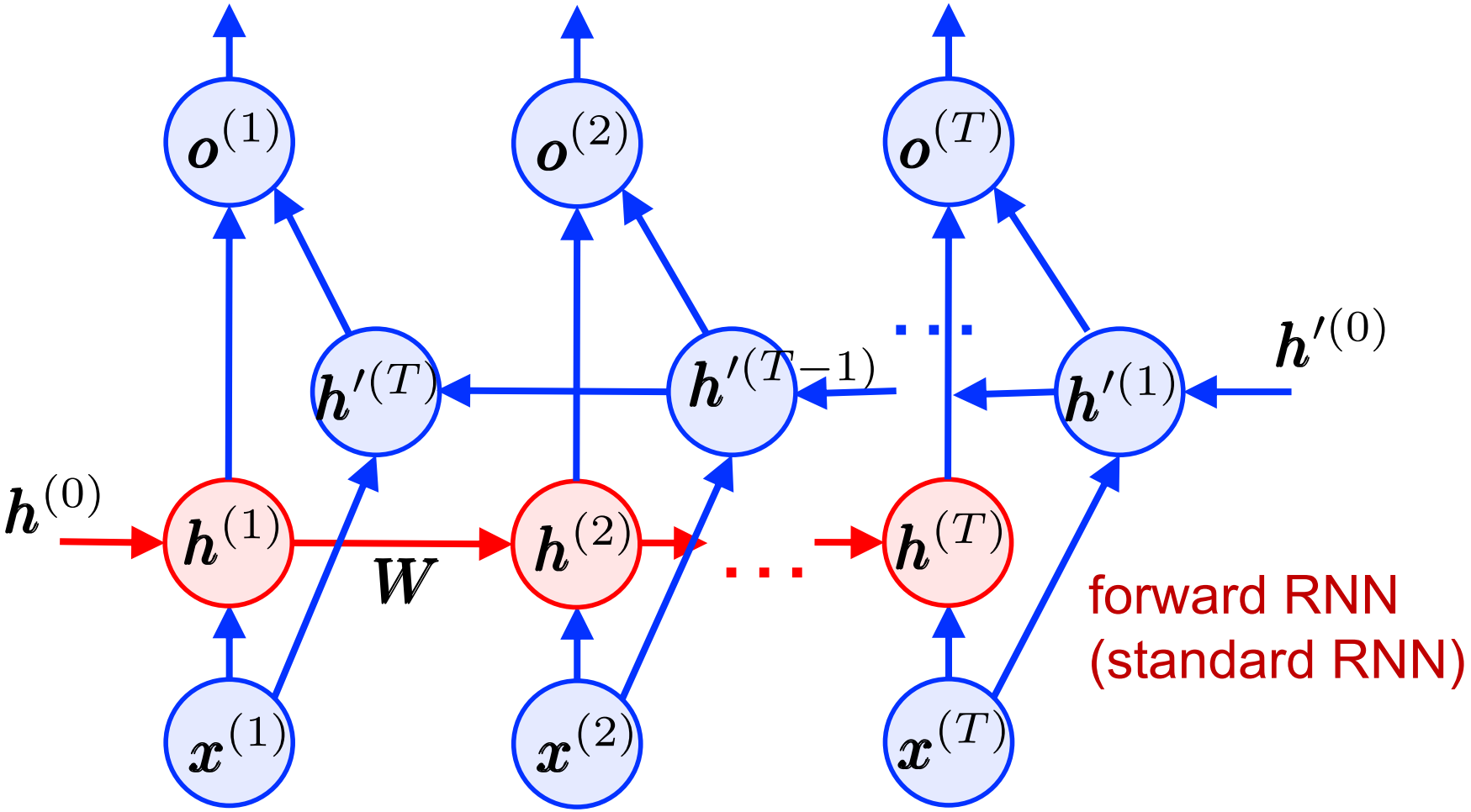
  - Once you have inputted "I am really angry": you are unable to say anything about the correctness of "angry"
  - But if you look forward, through the whole sequence "as I have been working out at the gym today"

    → you realize that angry is probably wrong

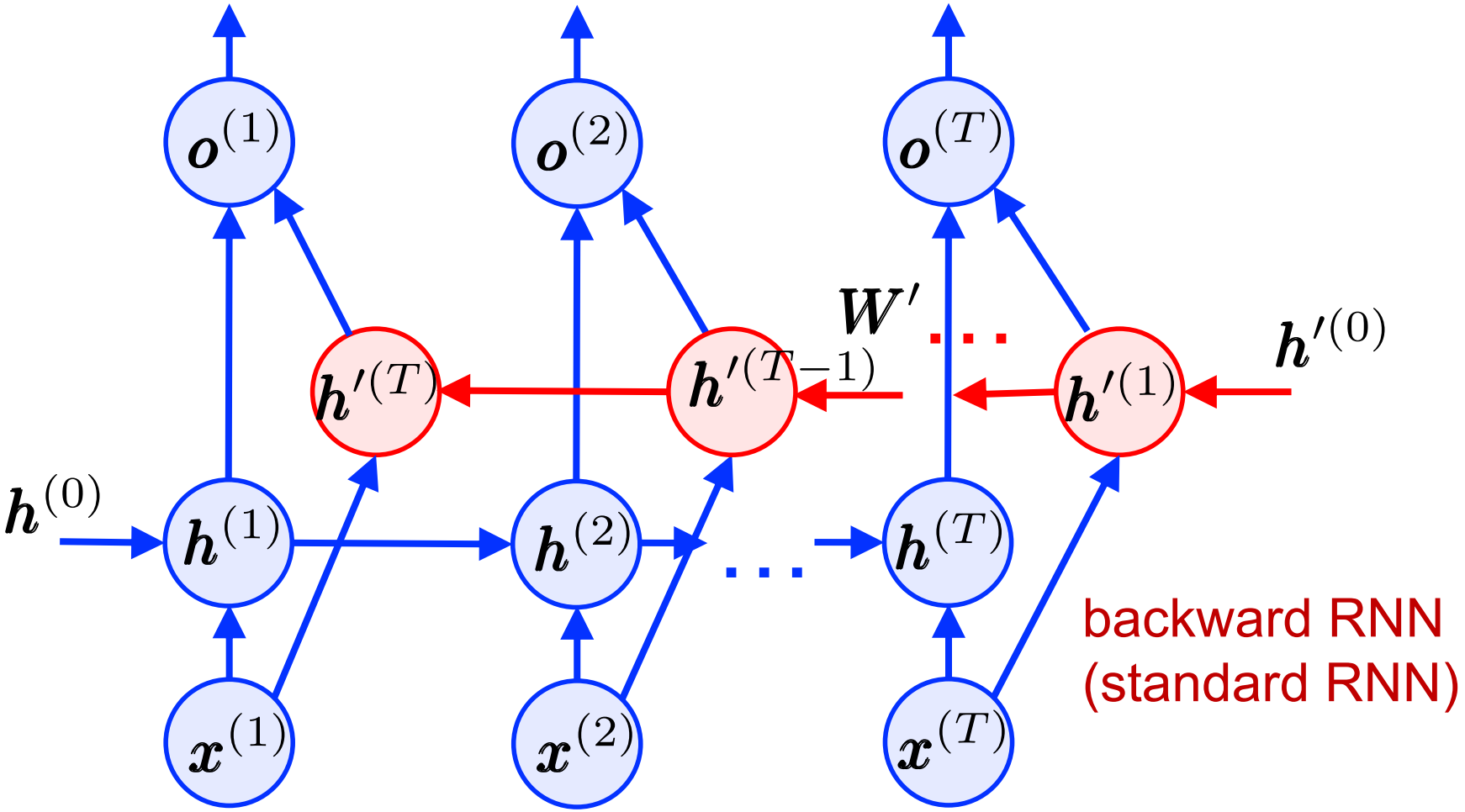
Bi-directional RNN (unfolded)



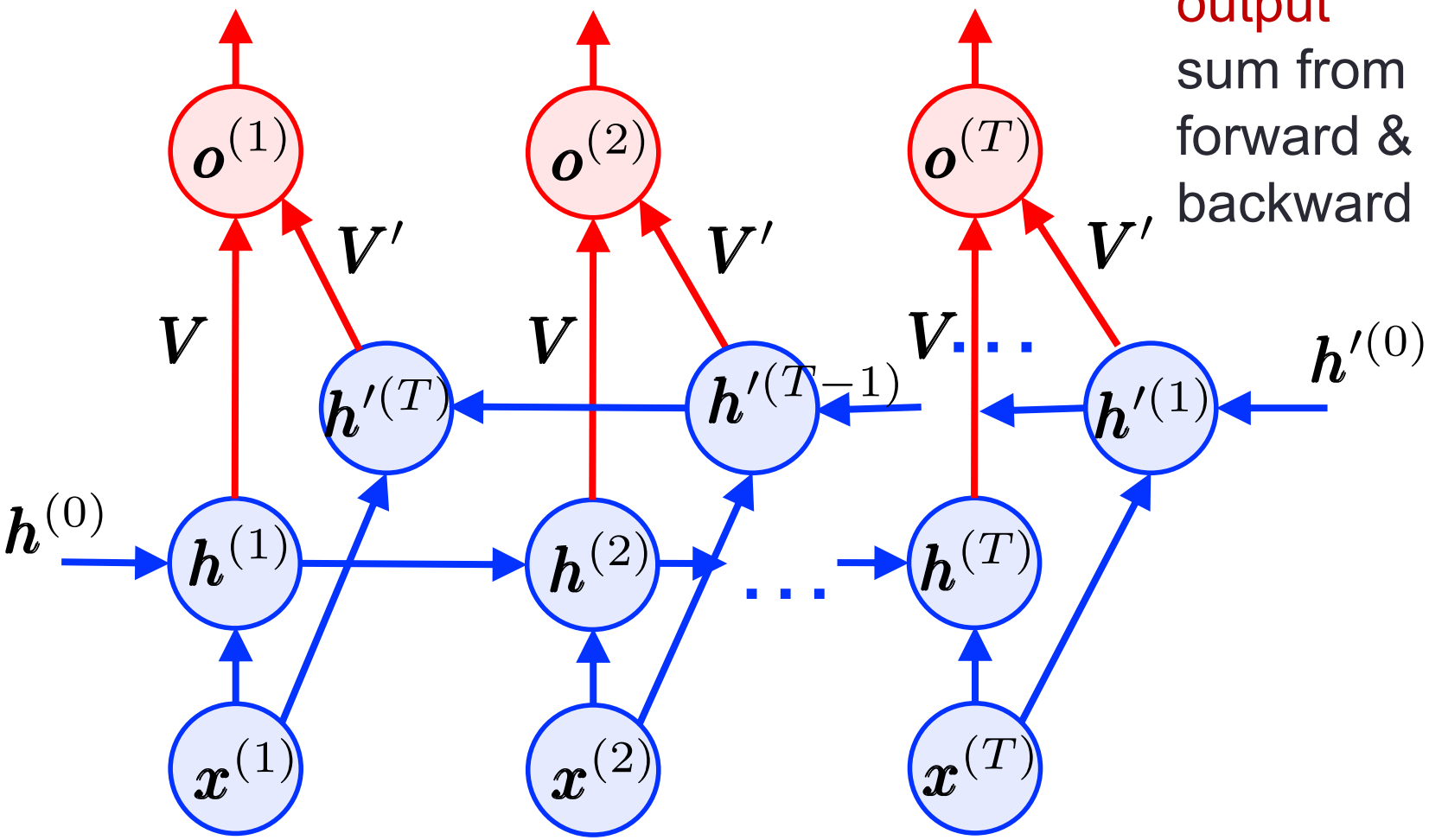
Bi-directional RNN (unfolded)



Bi-directional RNN (unfolded)



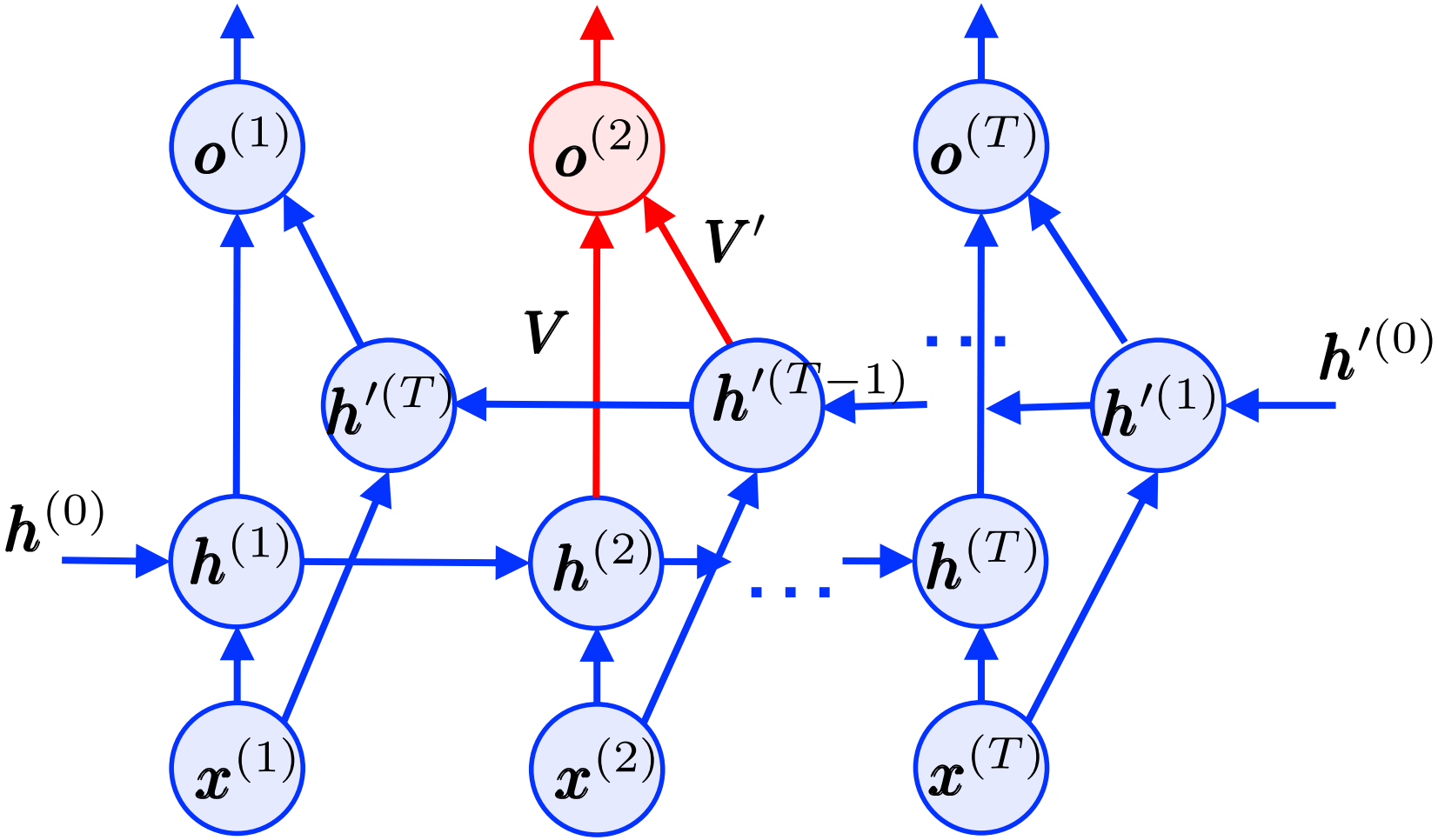
Bi-directional RNN (unfolded)



output at generic time i

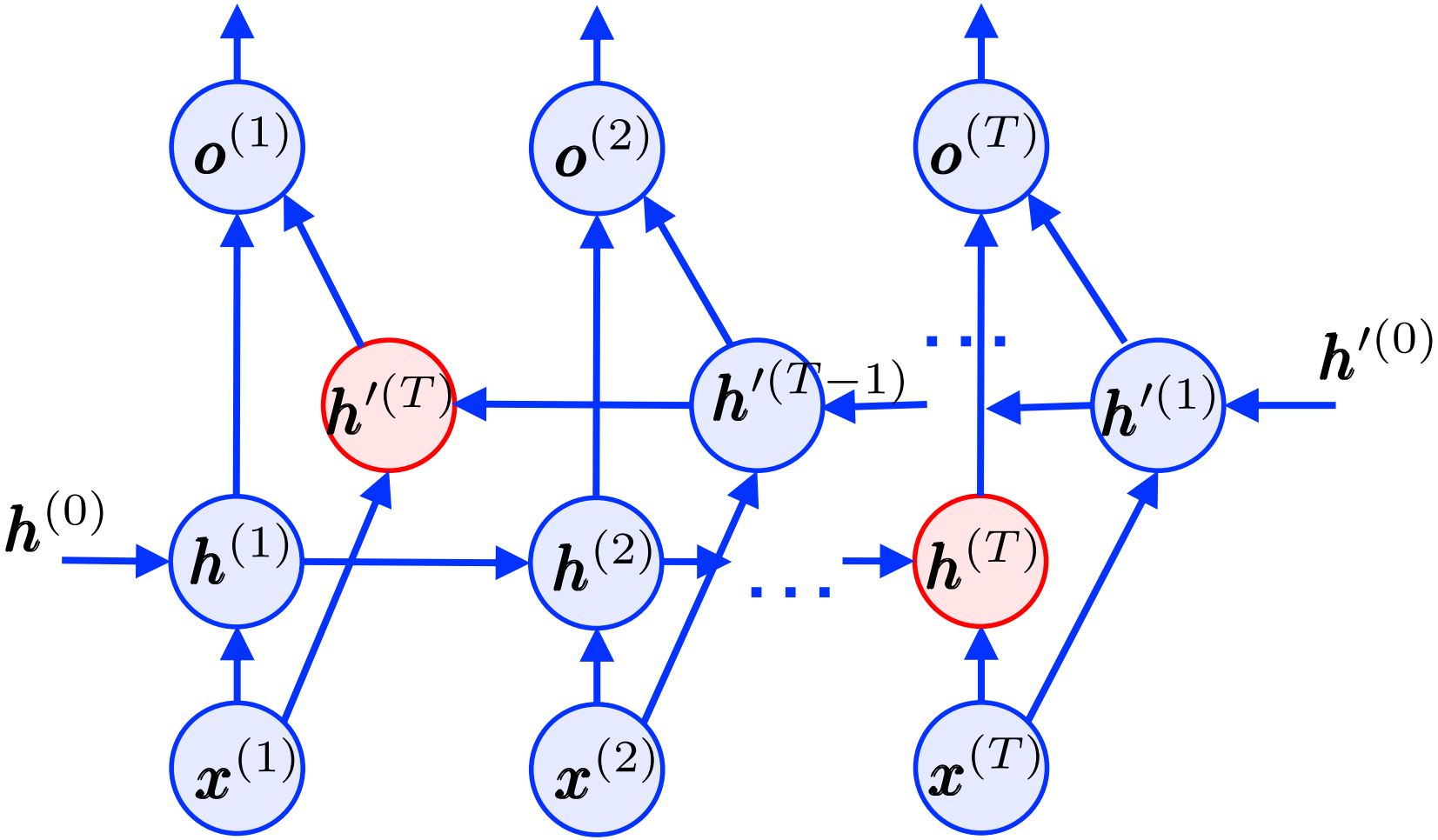
it is obtained by feeding the forward RNN with samples 1,2,…,i and the backward one with samples T,T-1,...,i

$$\boldsymbol{o}^{(i)} = \boldsymbol{V}\boldsymbol{h}^{(i)} + \boldsymbol{V}'\boldsymbol{h}'^{(T-i+1)} + \boldsymbol{c}$$



RNN final state

concatenation of **h**(T) and **h**'(T)

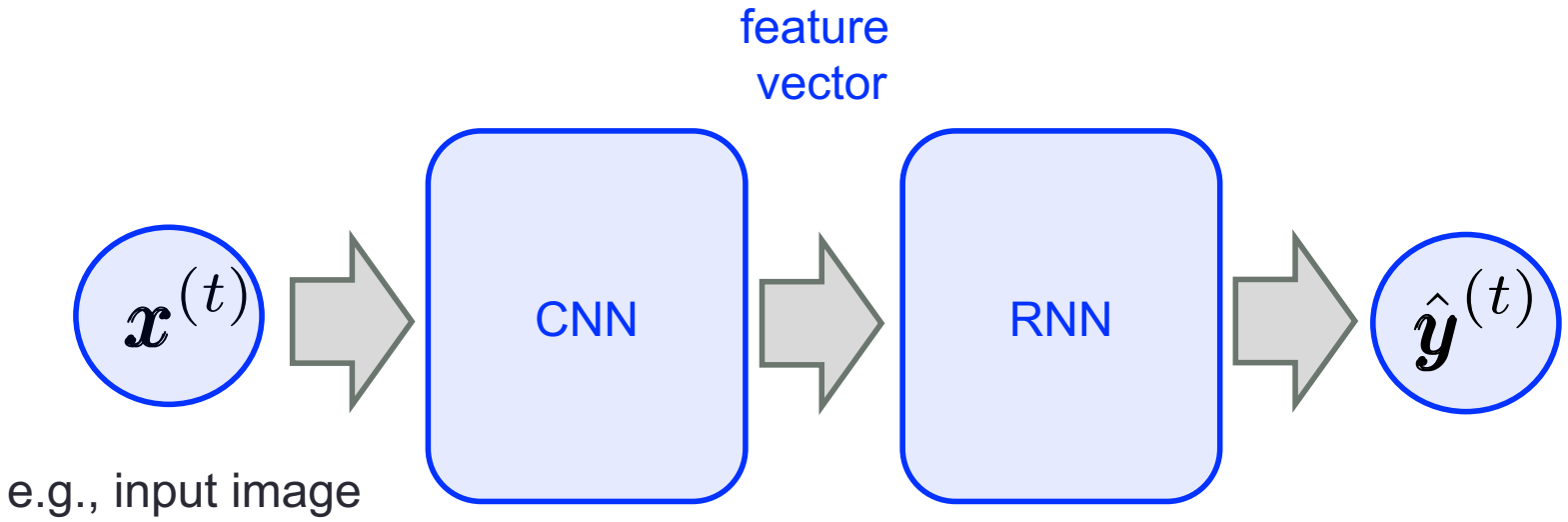


Use of RNNs

- Different alternatives exist
  - Vector-to-sequence
    - Take as input a (feature) vector representation (e.g., representing an image) and output a sequence (e.g., the image description)
  - Sequence-to-vector
    - Take as input a sequence (e.g., a sentence or a time series) and output a feature vector, *summarizing* the sequence → note that this allows to classify sequences using a following unsupervised clustering algo
  - Sequence-to-sequence – "Seq-to-Seq" (most popular)
    - Take as input a sequence and outputs another sequence (e.g., *language translation*) → it can be trained in a supervised manner (a target sequence is available for each input one) or as an autoencoder (the target sequence corresponds to the input one) → the code can be used as a compact representation of the sequence (sentence, time series, speech data, etc.)

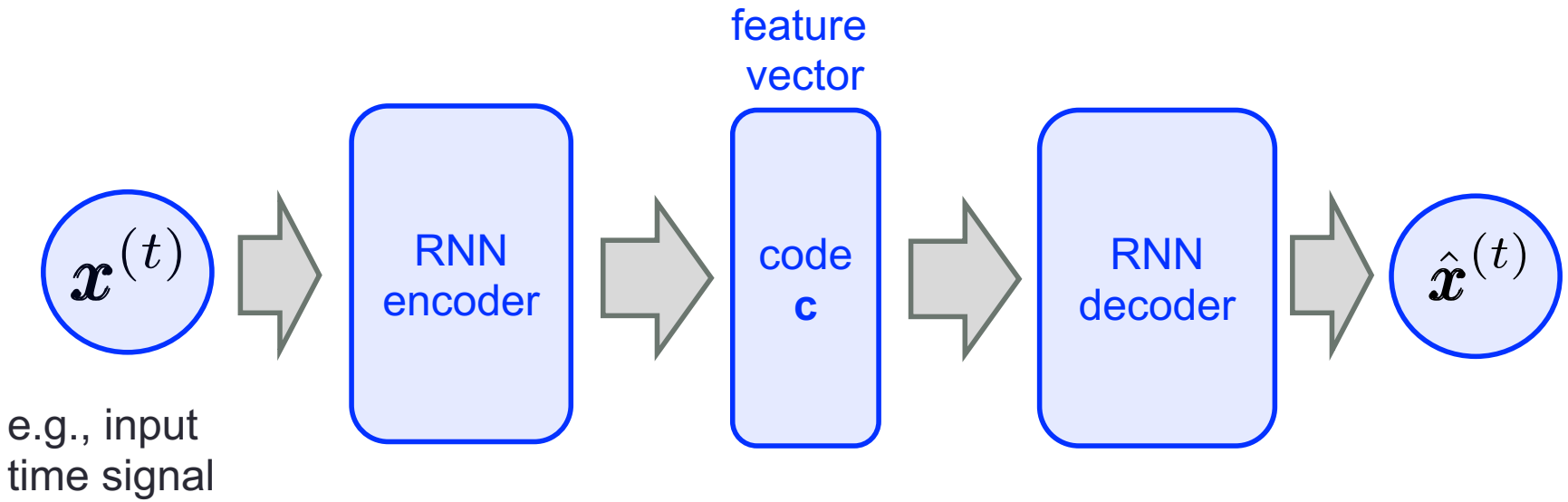
Combined CNN/RNN architectures

- Often times
  - CNN used as feature extractor
  - Feature vector is inputted into a subsequent RNN
  - Useful when input has a complex structure
    - e.g., 2D or 3D volumes



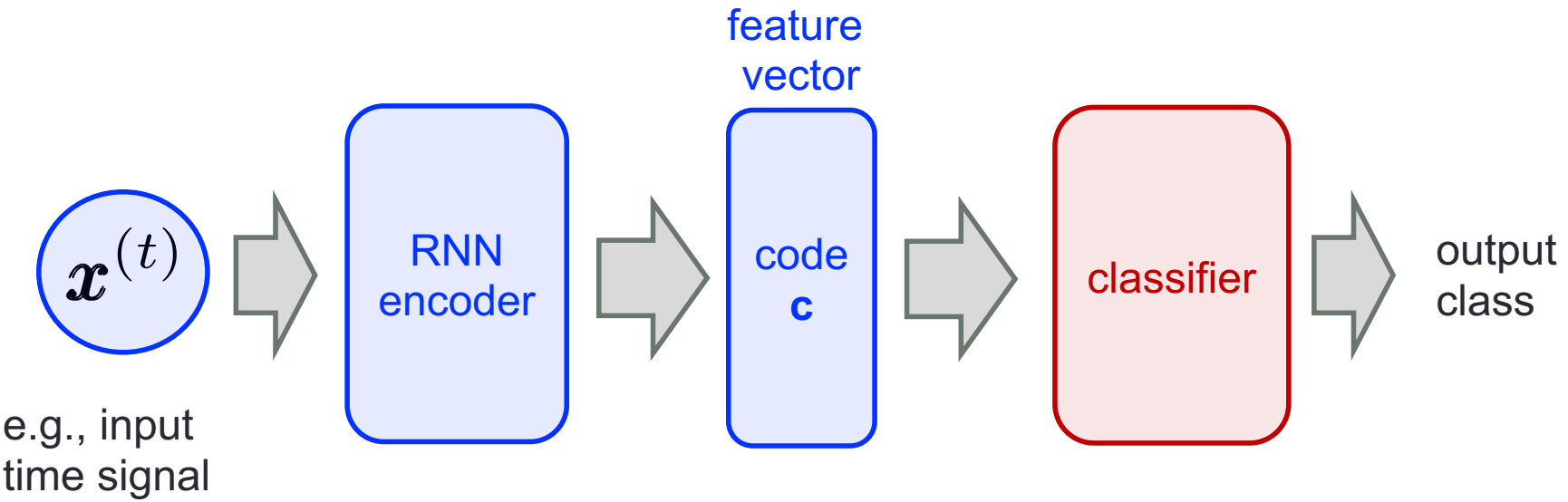
RNN-AE as feature extractor (1/2)

- RNN-AE
  - Seq-2-Seq RNN trained as an autoencoder
  - The code **c** contains the final RNN encoder state
  - When training is complete
    - **c** is a compact representation of the input sequence



RNN-AE as feature extractor (2/2)

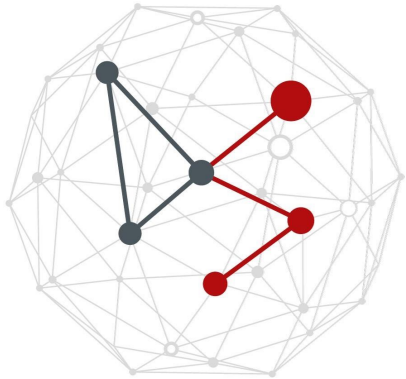
- Advantages
  - At runtime
    - Get rid of the RNN decoder
    - Use the RNN encoder as a feature extractor for (correlated) sequences
  - To keep in mind
    - Input sequences may be of different length
    - But the code is a vector of fixed size!!!



MIME

Master's degree ICT Internet Multimedia Engineering

APPLICATION: UNSUPERVISED REPRESENTATION LEARNING OF AUDIO WAVEFORMS

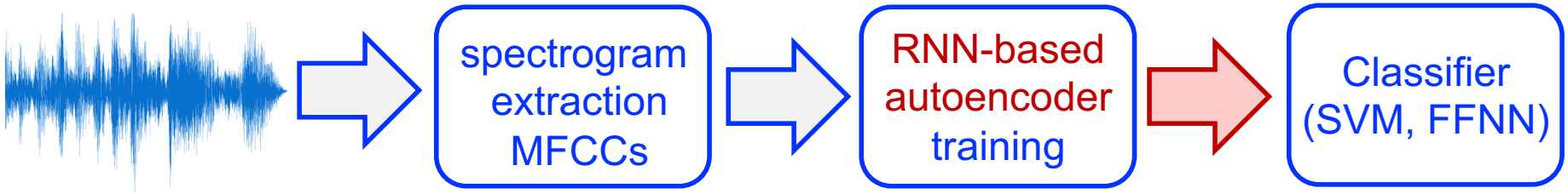


DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

DIPARTIMENTO MATEMATICA

UNIVERSITÀ DEGLI STUDI DI PADOVA

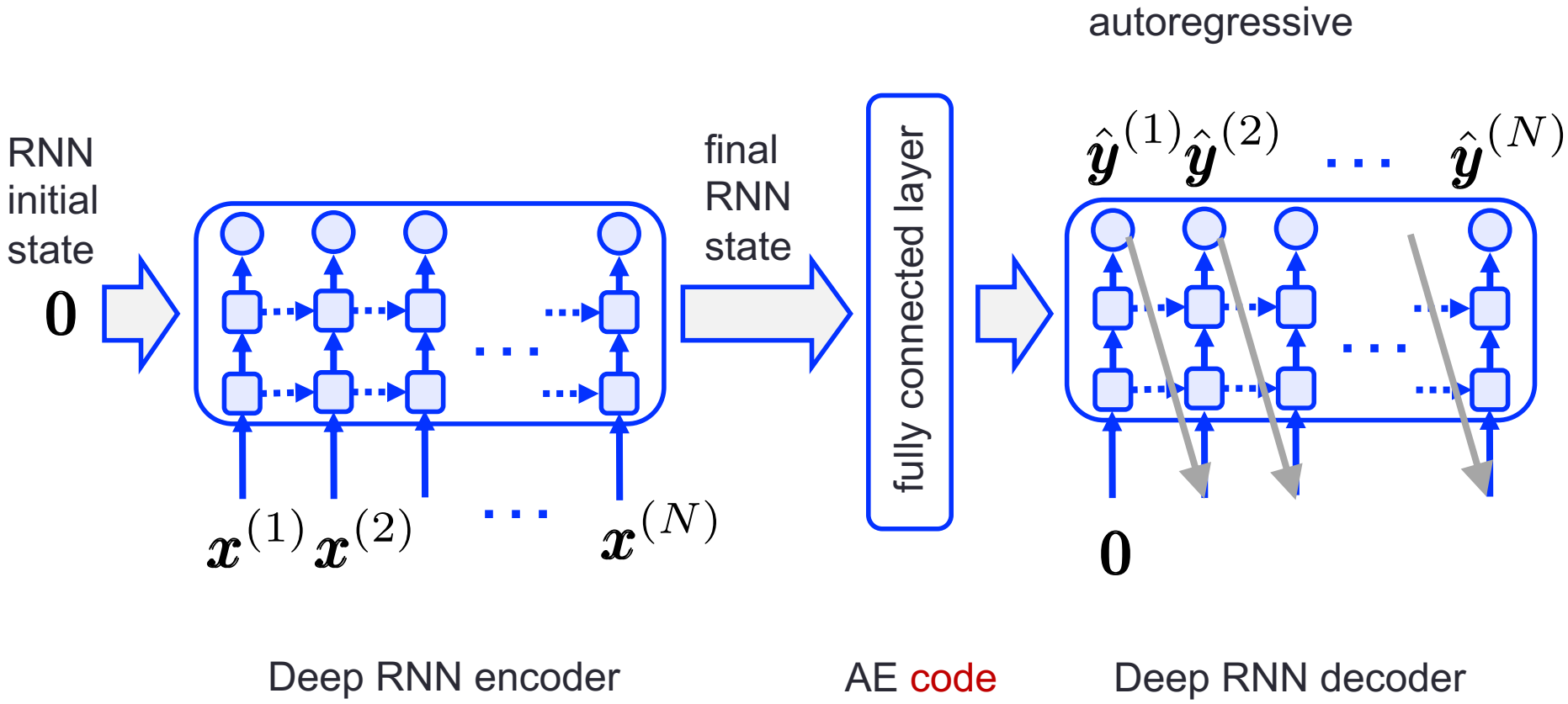
Unsupervised representation of audio



Signal processing pipeline from [Amiriparian17]

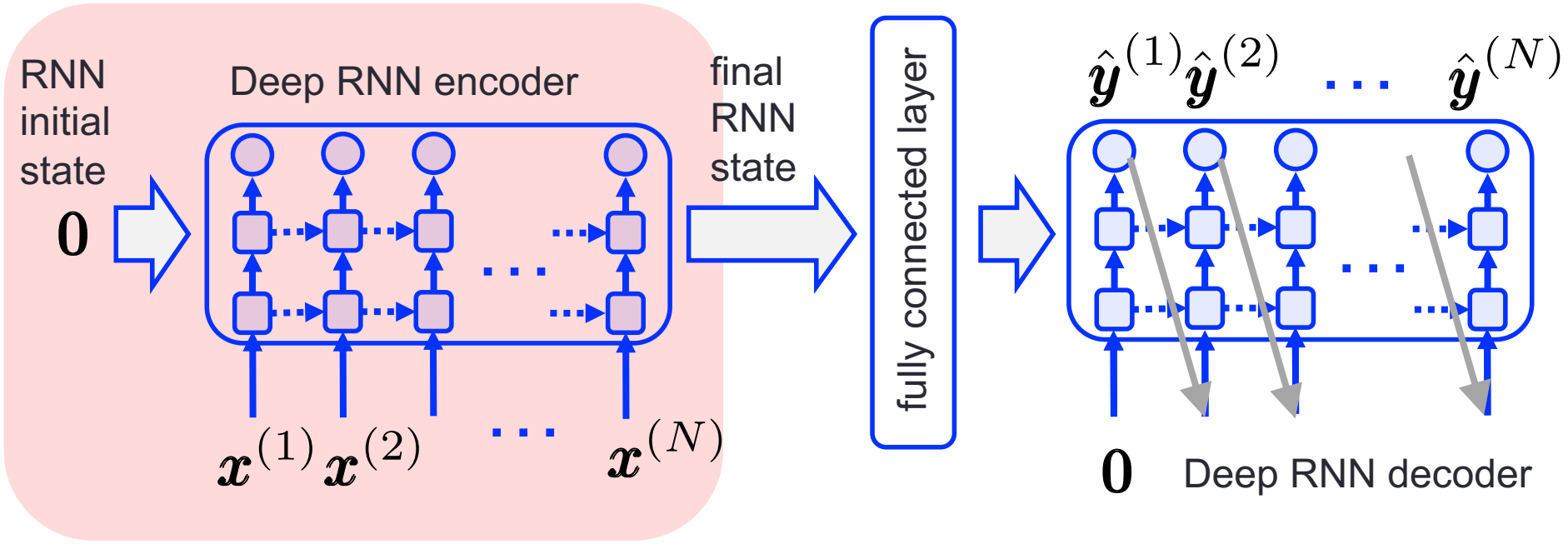
[Amiriparian17] S. Amiriparian, M. Freitag, N. Cummins, B. Schuller, “Sequence to Sequence Autoencoder for Unsupervised Representation Learning from Audio,” *Detection and Classification of Acoustic Scenes and Events (DCASE)*, Munich, Germany, November 2017.

Deep RNN – AE (1/5)



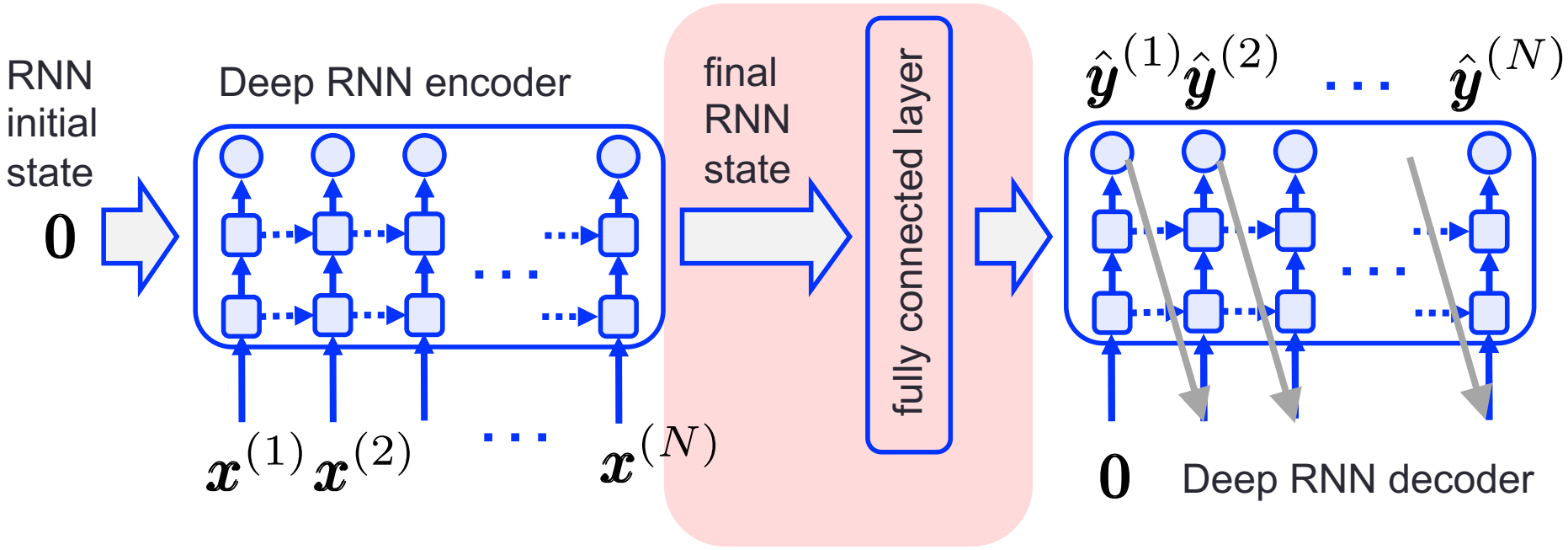
Training objective is: $\hat{\boldsymbol{y}}^{(i)} \simeq \boldsymbol{x}^{(i)}$, $i = 1, 2, \ldots, N$

Deep RNN – AE (2/5)



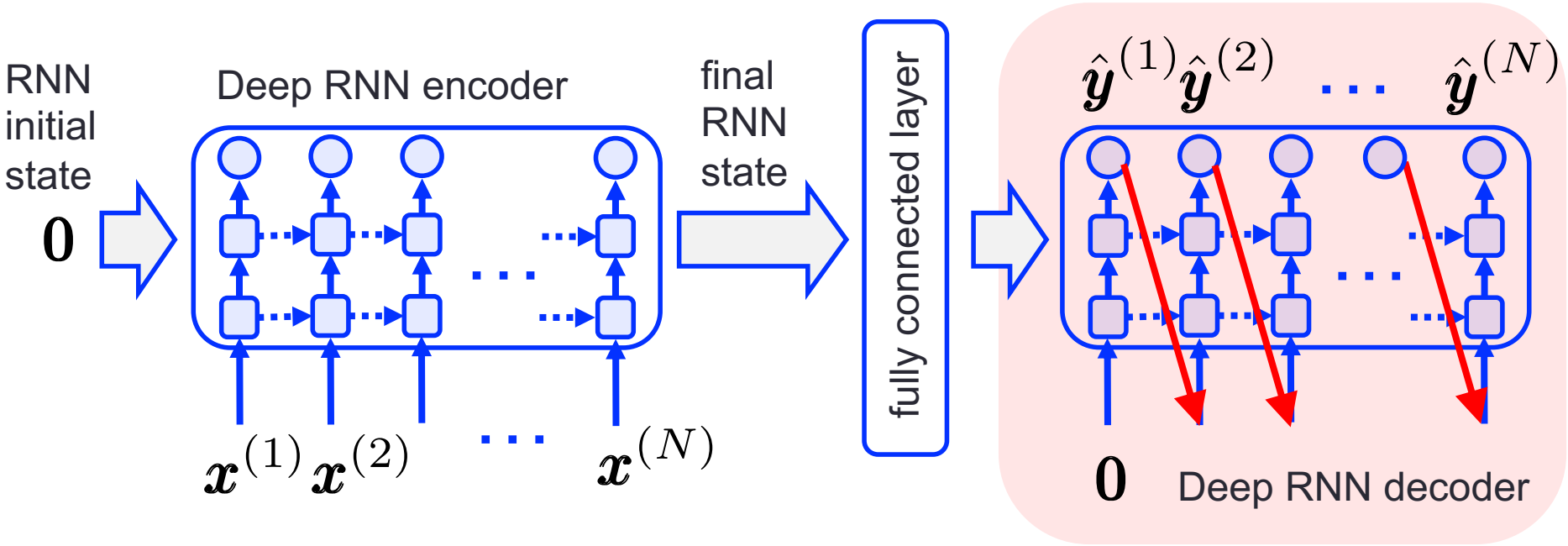
- RNN encoder
  - RNN state → zero vector
  - input vector sequence has N time steps n=1,2,…,N
  - input one feature vector at a time from time 1 to time N

Deep RNN – AE (3/5)



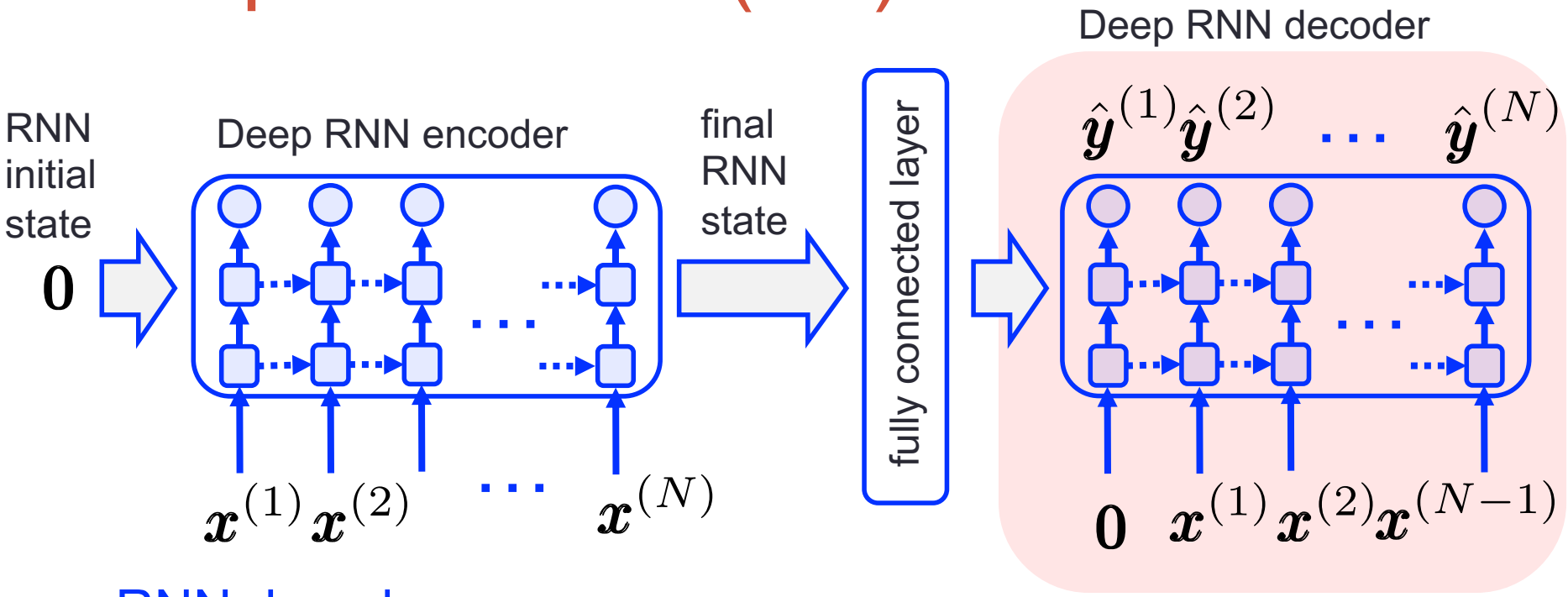
- After N time steps
  - final RNN state "flattened" into a vector (the "code")
  - code inputted into a fully connected layer
    - with tanh activation functions

Deep RNN – AE (4/5)



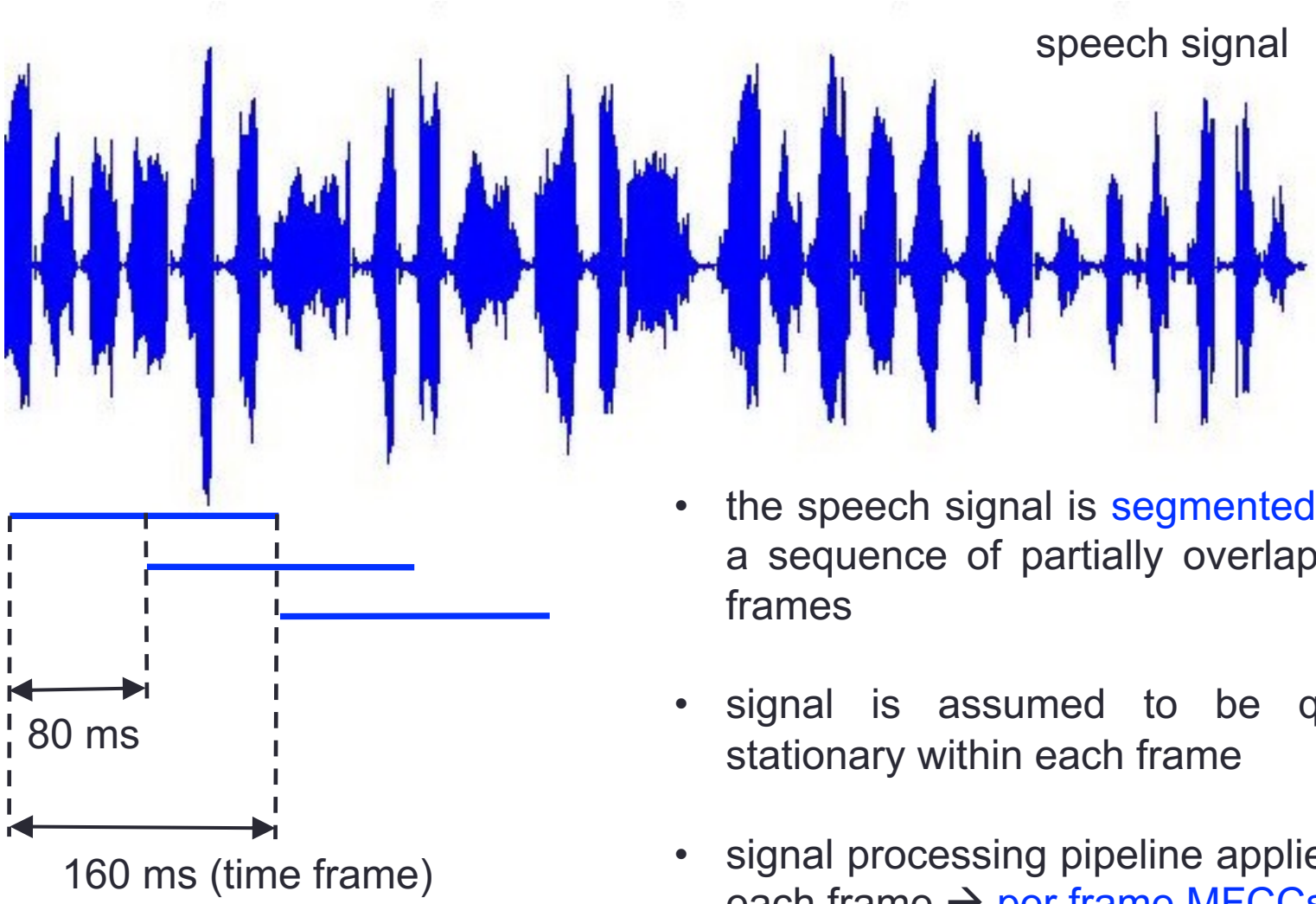
- RNN decoder
  - initial decoder state = AE code
  - decoder is a predictor of the input sequence
  - output from time t-1 used as input for time t
    - to predict output at time t

Deep RNN – AE (5/5)



- RNN decoder
  - Trick: the *shifted* input sequence is used
    - in place of the one outputted by the decoder
    - In [Amiriparian17] it is proven that
      - this leads to faster training
      - performance degradation is negligible

Preprocessing - spectrograms



- the speech signal is segmented into a sequence of partially overlapping frames
- signal is assumed to be quasi stationary within each frame
- signal processing pipeline applied to each frame → per frame MFCCs

Preprocessing - spectrograms

- Mel Frequency Cepstral Coefficients (MFCCs)
  - First step in any speech recognition system
  - MFCCs are audio features
- Shape of the vocal tract
  - Is captured by the envelope of the *short-term power spectrum* of the audio signal
  - MFCCs accurately represent this envelope [Davis1980]

[Davis1980] S. Davis, P. Mermelstein, "Comparison of parametric representations for monosyllabic word recognition in continuously spoken sequences," *IEEE Transactions on Acoustic, Speech and Signal Processing*, Vol. 28, No. 4, 1980.

Dataset – DCASE challenge

- Audio samples from 15 acoustic scenes [Mesaros2017]
  - Recorded at distinct geographic locations
- For each location
  - Between 3 and 5 minutes of audio are recorded
  - And split into 10 seconds long audio excerpts
- Final dataset for training
  - 4,680 instances
  - 312 instances per class
- Test set
  - 1,620 instances (with label not provided for the challenge)

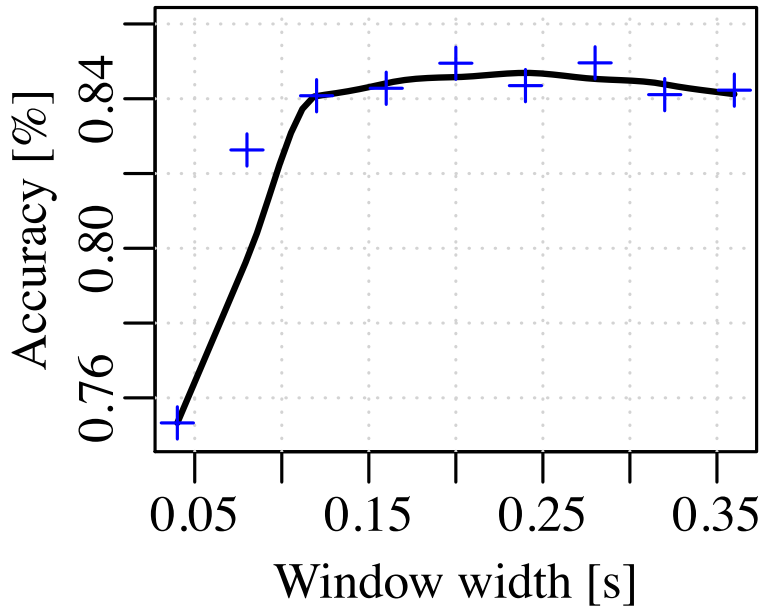
[Mesaros2017] A. Mesaros, T. Heittola, A. Diment, B. Elizalde, A. Shah, E. Vincent, B. Raj, and T. Virtanen, "DCASE 2017 Challenge Setup: Tasks, Datasets and Baseline System," in Detection and Classification of Acoustic Scenes and Events (DCASE2017), Munich, Germany, Nov 2017.

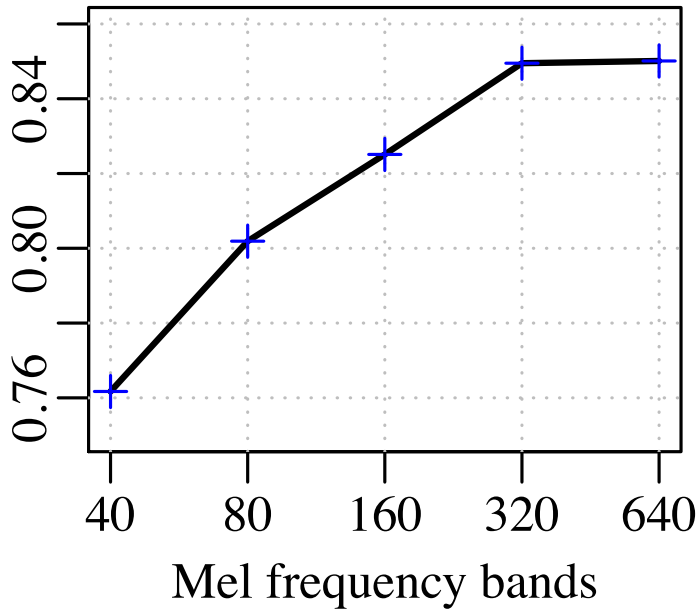
Computation of MFCCs

- Windows of 160 ms each
- Overlap among windows is 80 ms (half of the window)

1. Segment the signal into short frames
2. For each frame: calculate its power spectrum
   - uses DFT with Hann windows
3. Apply a *log-scaled* Mel filterbank to the power spectrum
   - compute energy within each sub-band – take its log
   - Feature vector: 320 log-Mel energies per audio frame
4. Normalize each filterbank output within [-1,1]
   - since the output of the Seq-2-Seq AE are in [-1,1]

Classification accuracy



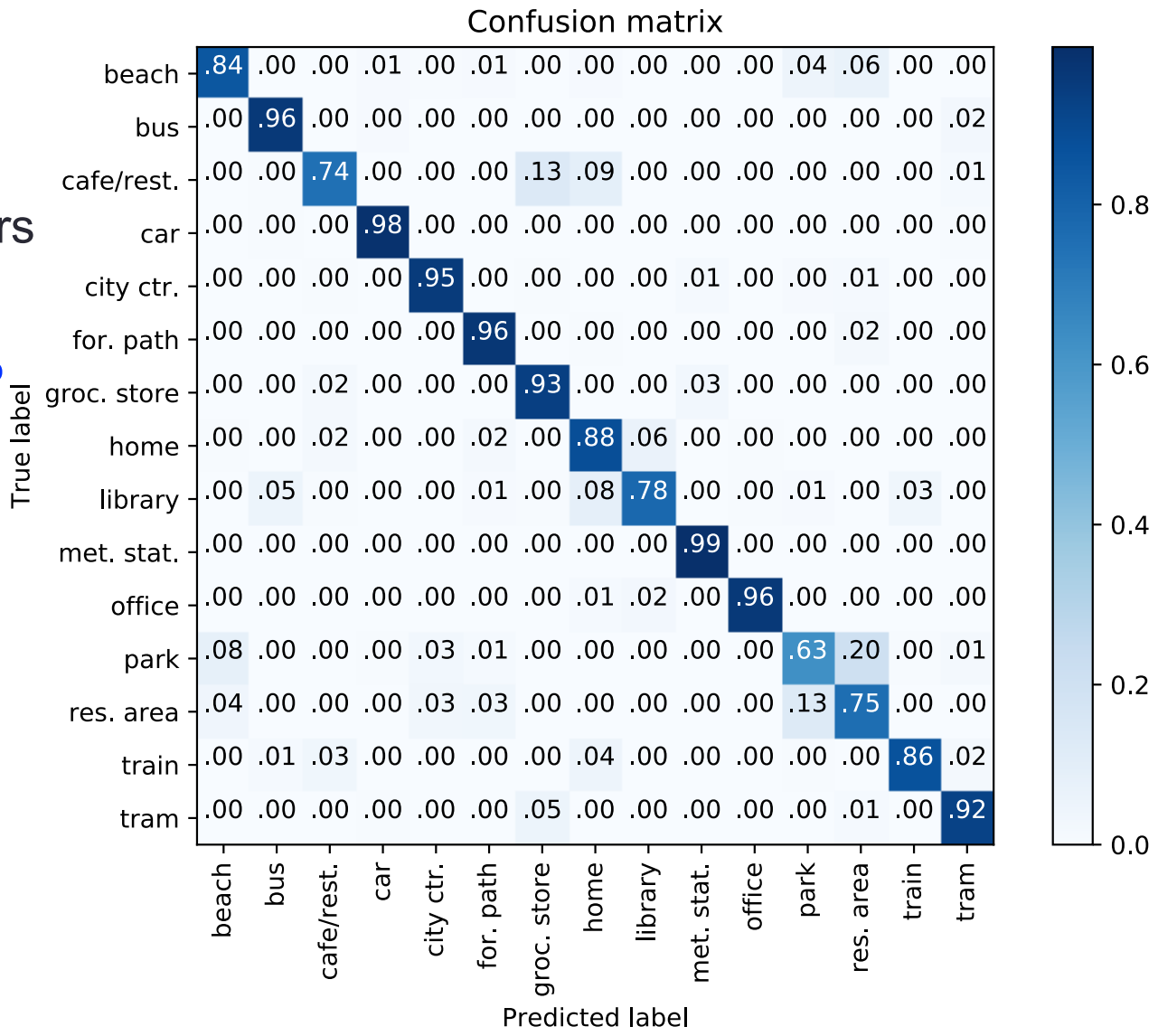


Grid search for preprocessing parameters

- Length of spectrogram windows (audio frames)
- Amount of overlap between frames
- Number of Mel frequency bands

Confusion matrix

- Confusion matrix
  - best AE found
  - 2 stacked RNN layers
  - 256 neurons x layer
  - mean accuracy 88%



Bibliography (1/3)

Deep Learning

[LeCun15] Yann LeCun, Yoshua Bengio, Gaoffrey Hinton, “Deep Learning,” Nature, 2015.

[Goodfellow16] I. Goodfellow, Y. Bengio, A. Courville, “Deep Learning,” The MIT Press, 2017.

RNN – LSTM neural networks

[Hochreiter97] S. Hochreiter, J. Schmidhuber, “Long short-term memory,” *Neural Computation*, 9(8):1735–1780.

[Schuster97] Mike Schuster, Kuldip K. Paliwal. "Bidirectional recurrent neural networks," *IEEE Transactions on Signal Processing*, Vol. 45, No. 11, 1997.

[Rossi22] M. Rossi, “Instructor’s Notes on Recurrent Neural Networks,” *Tech. Report*, University of Padova, 2022.

Bibliography (2/3)

Seq-to-Seq autoencoders

[Sutskever14] I. Sutskever, O. Vinyals, Q. V. Le, “Sequence to sequence learning with neural networks,” International Conference on Neural Information Processing Systems (NIPS), Montreal, Canada, 2014.

[Cho14] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, Y. Bengio, “Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation,” Conference on Empirical Methods in Natural Language Processing (EMNLP), Doha, Qatar, 2014. (introduced GRU cells for RNNs)

Bibliography – applications (3/3)

Audio - Seq-2-Seq autoencoder

[Davis1980] S. Davis, P. Mermelstein, “Comparison of parametric representations for monosyllabic word recognition in continuously spoken sequences,” *IEEE Transactions on Acoustic, Speech and Signal Processing*, Vol. 28, No. 4, 1980.

[Amiriparian17] S. Amiriparian, M. Freitag, N. Cummins, B. Schuller, “Sequence to Sequence Autoencoder for Unsupervised Representation Learning from Audio,” Detection and Classification of Acoustic Scenes and Events (DCASE), Munich, Germany, November 2017.

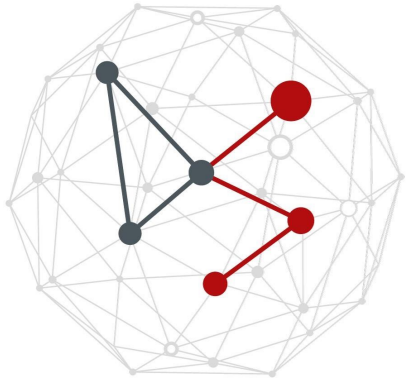
Software: https://github.com/auDeep/auDeep

MiME
Master's degree ICT Internet Multimedia Engineering

LEARNING FOR SEQUENTIAL DATA: TOOLS AND APPLICATIONS

Michele Rossi
michele.rossi@dei.unipd.it

Dept. of Information Engineering
University of Padova, IT



DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

DIPARTIMENTO MATEMATICA

UNIVERSITÀ DEGLI STUDI DI PADOVA