LEARNING FOR SEQUENTIAL DATA: TOOLS AND APPLICATIONS

Michele Rossi

rossi@dei.unipd.it

Dept. of Information Engineering

University of Padova, IT

Outline

- Why? What? How?
- Unsupervised learning for i.i.d. seqs
  - Autoencoders
  - Application example
- Unsupervised learning for time correlated seqs
  - Recurrent neural networks
  - Application example
- Bibliography

Why? What? How?

- What we are going to see today
  - We are interested in data sequences
- Data points (samples)
  - Are generated one at a time
  - Can be either i.i.d. or time correlated
  - Are sequentially fed to an algorithm
  - To capture some key features of the data
- Learning objectives
  - i.i.d. seqs: the data probability density function (pdf)
  - Correlated seqs: capture temporal evolution

Unsupervised learning

- We are interested in
  - Unsupervised learning algorithms
  - That automatically extract useful structure from data
- Useful to what?
  - To **(i)** compress, **(ii)** classify, **(ii)** predict (or interpolate)

"We expect unsupervised learning to become *far more important in the longer term*. Human and animal learning is largely unsupervised: we discover the structure of the world by observing it, not by being told the name of every object." [LeCun15]

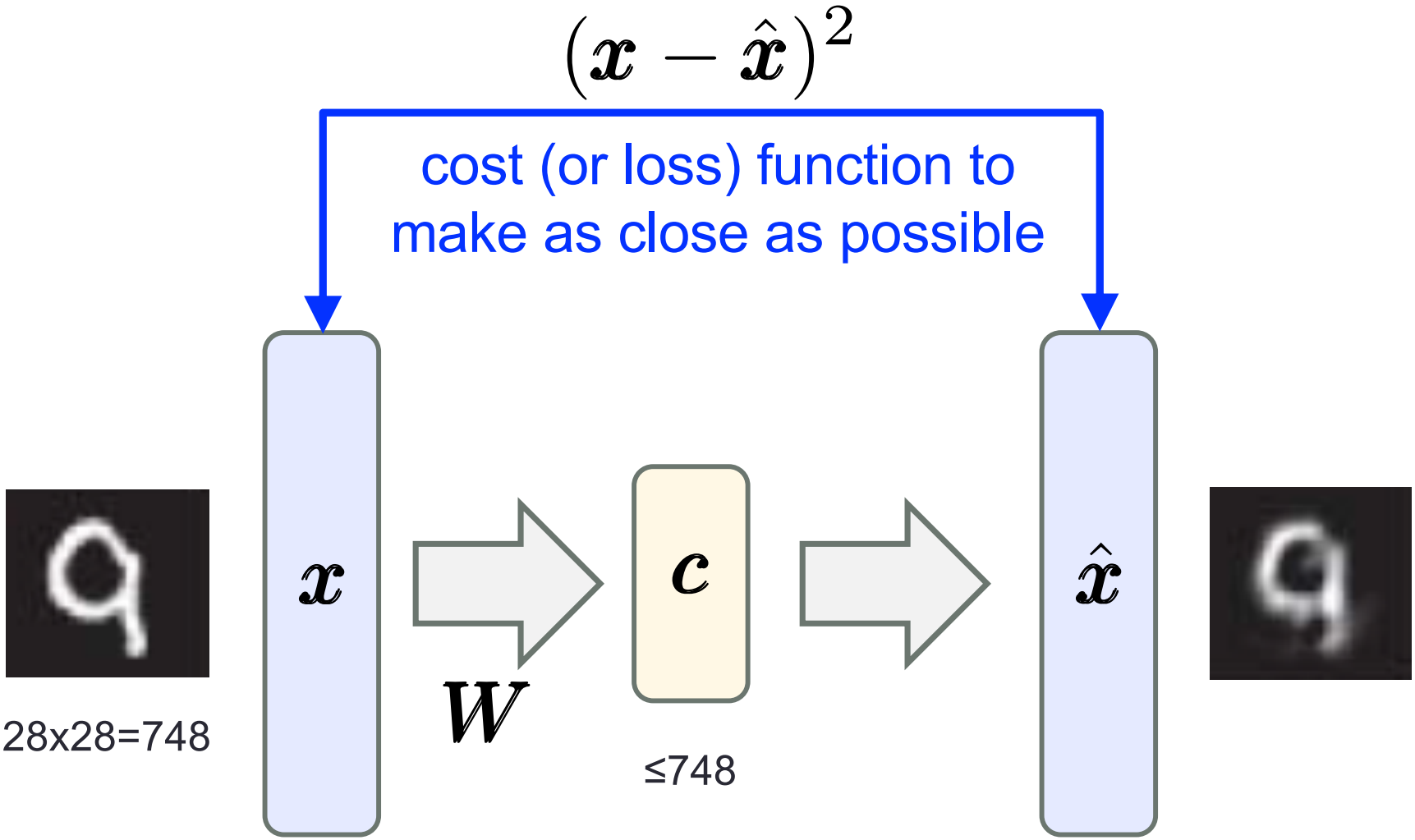
[LeCun15] Yann LeCun, Yoshua Bengio, Gaoffrey Hinton, "Deep Learning," *Nature*, 2015.

AUTOENCODERS

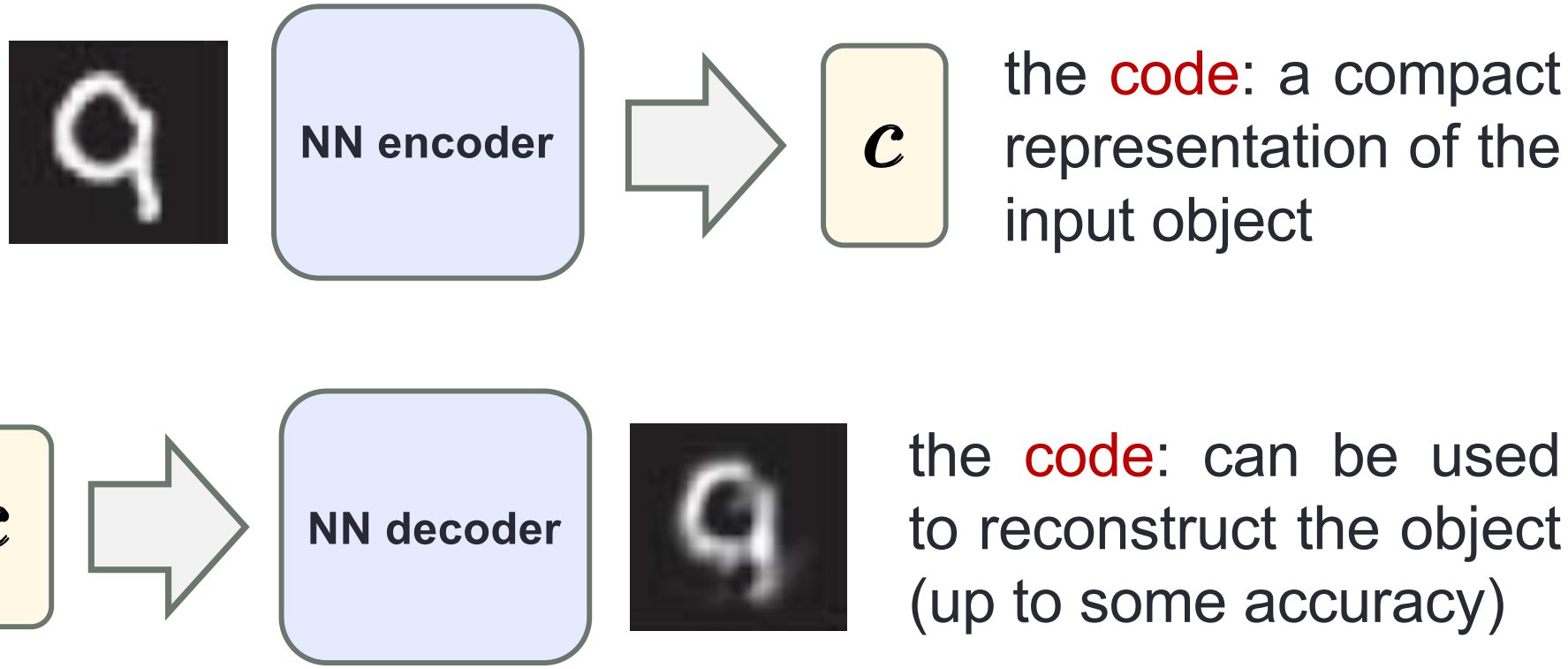
Principal Component Analysis (PCA)

- PCA
  - Popular signal processing technique
  - Used for dimensionality reduction (feature extraction)
- Transform basis
  - Linear transform (matrix multiplication)
  - Computed from the (sample) data covariance matrix
- Output vector space
  - Output points are linearly independent
  - Obtained projecting input data along
    - its principal directions (max. variance)
    - assumption high variance = interesting structure

Principal Component Analysis (PCA)

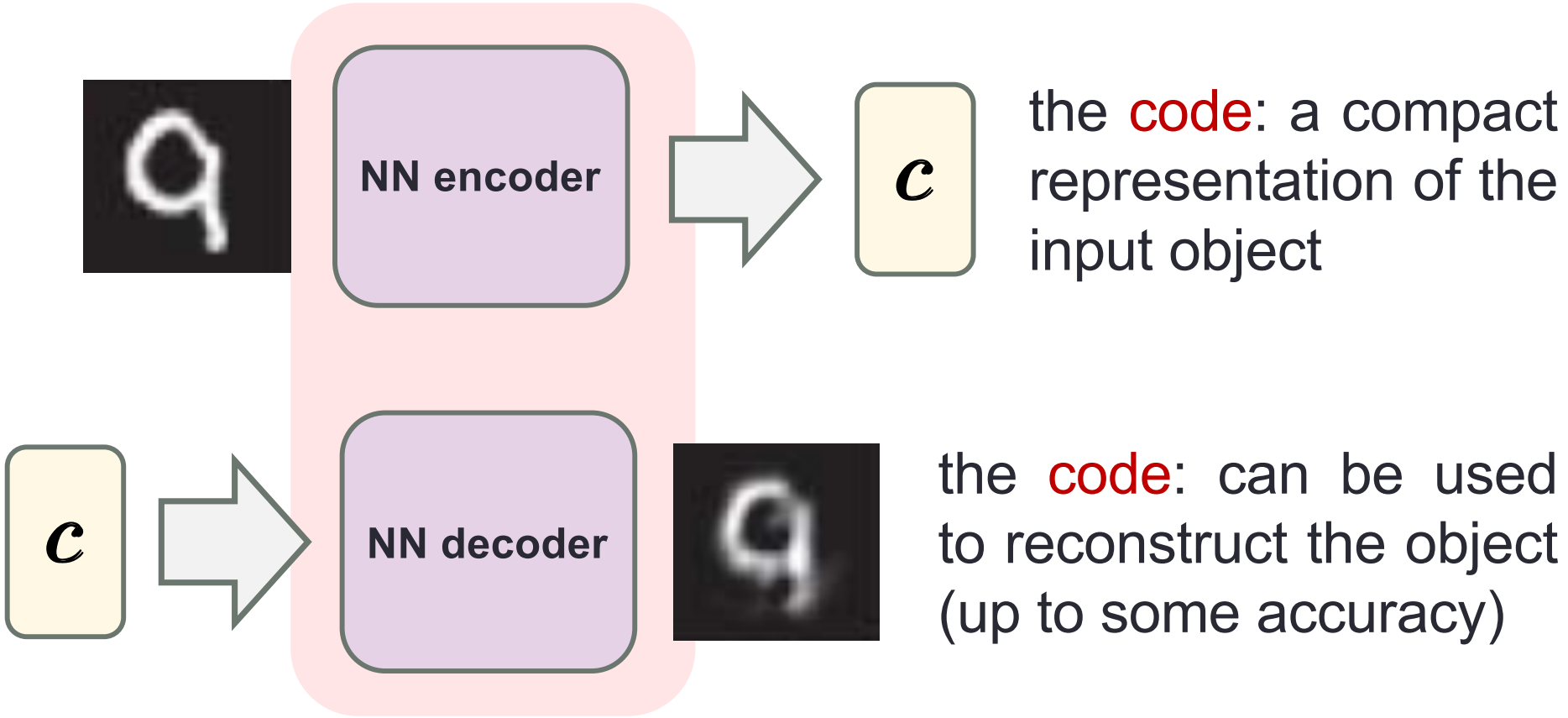


Autoencoder [Hinton06]



[Hinton06] G. E. Hinton and R. R. Salakhutdinov, “Reducing the Dimensionality of Data with Neural Networks,” Science, Vol. 313, July 2006.

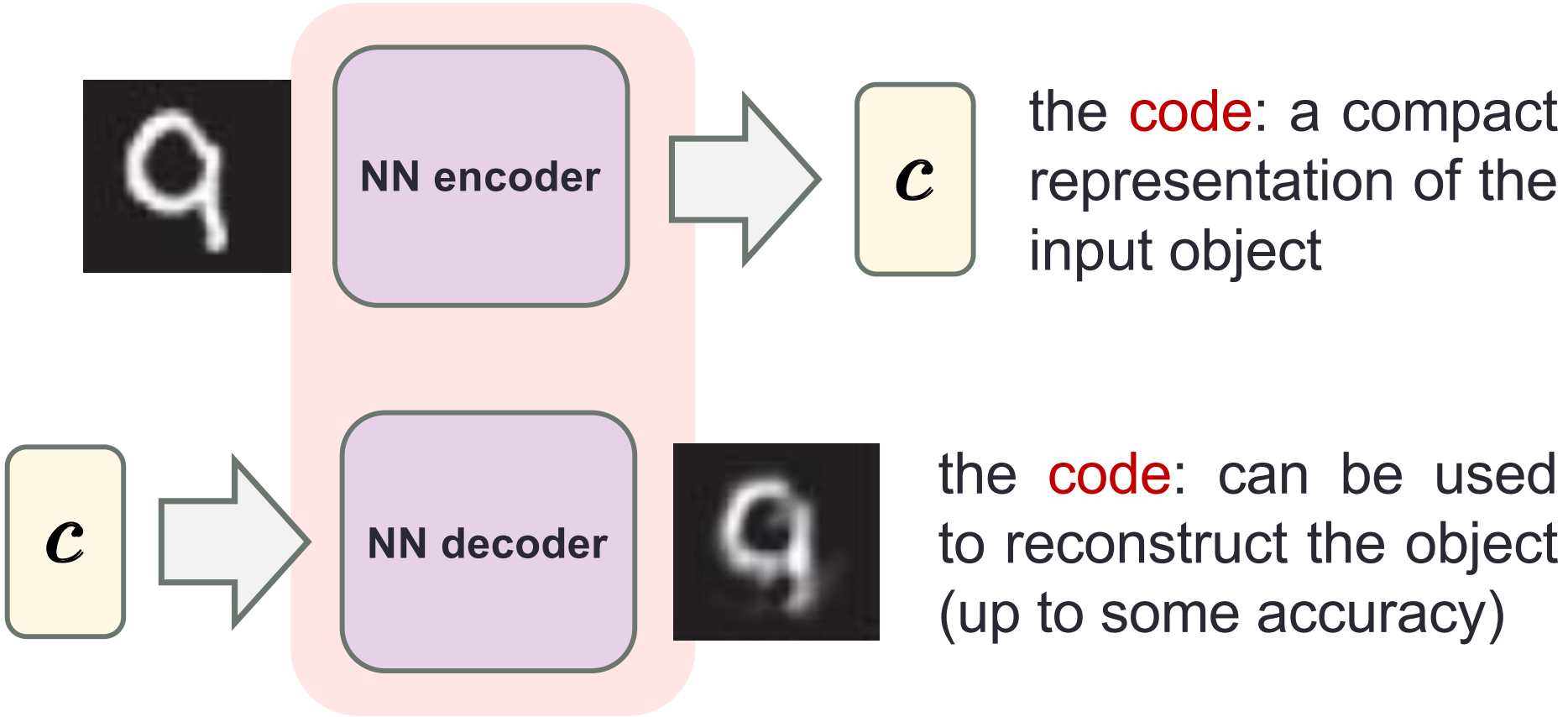
Autoencoder



Key point 1: encoder and decoder

- are jointly trained

Autoencoder

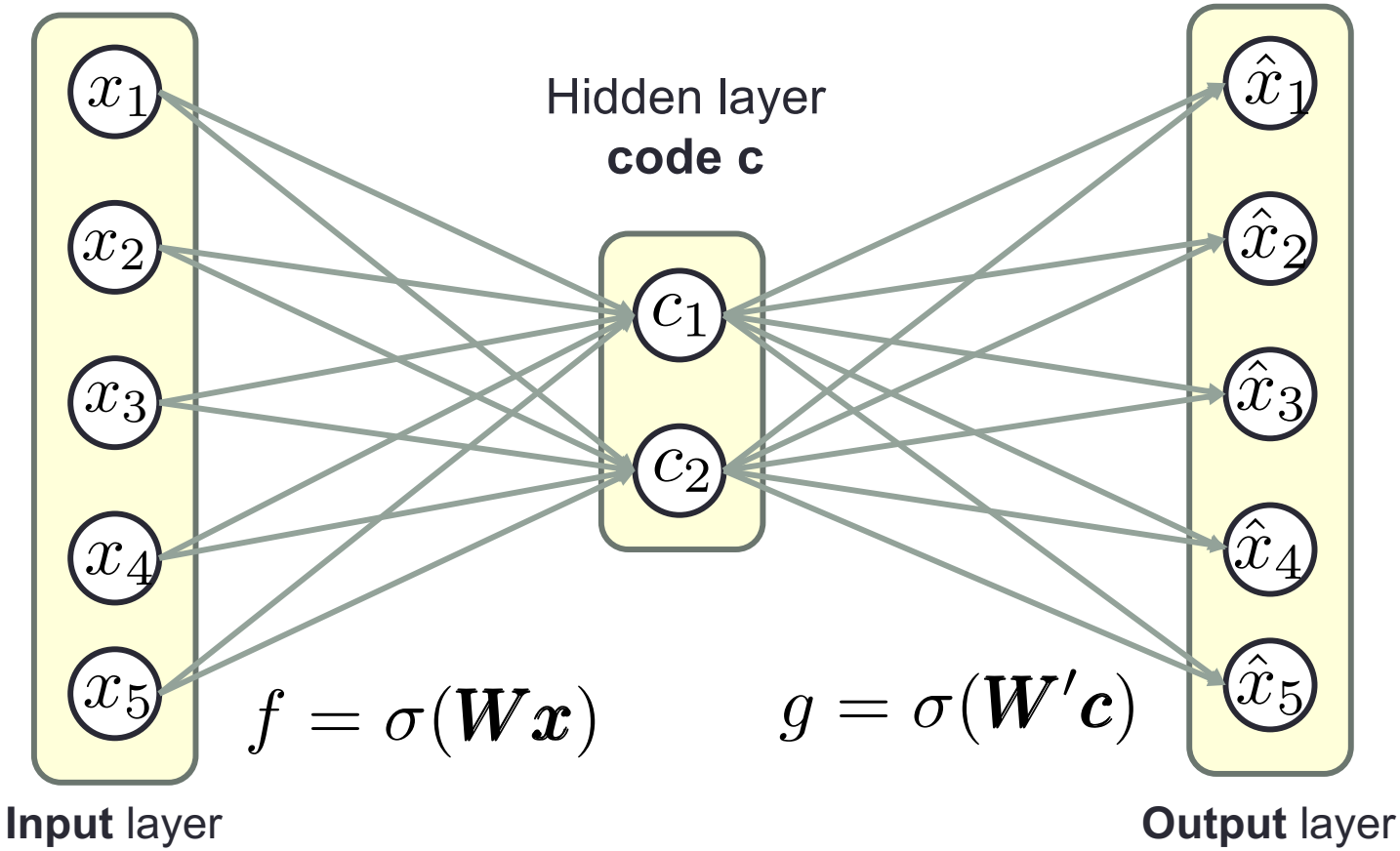


Key point 2: encoder and decoder

- are non linear (usually through sigmoid or tanh activations)

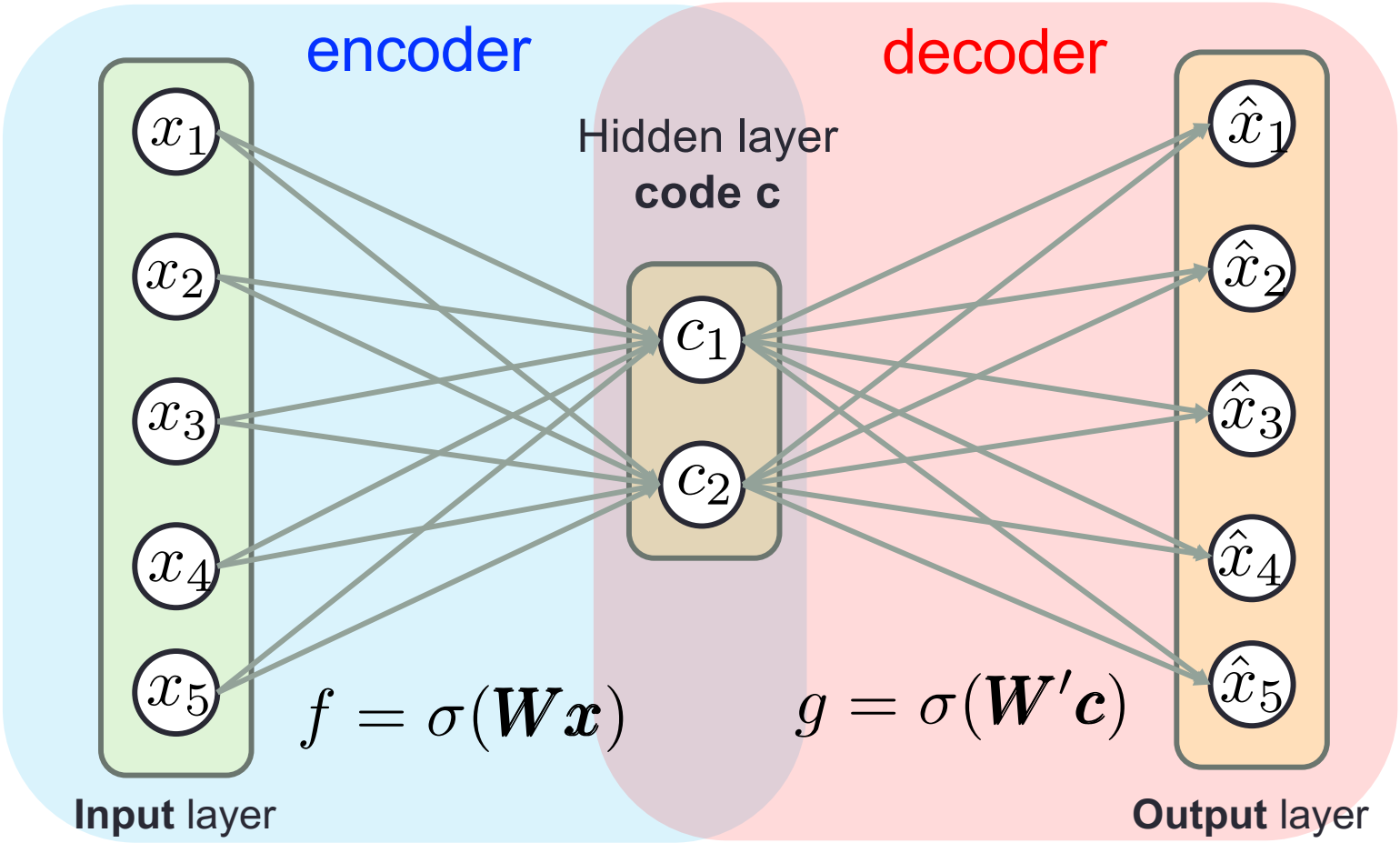
Autoencoder through FFNNs

- Feed Forward Neural Networks (FFNN)
  - Implement functions, do not have internal states (memory cells)
  - Artificial neurons organized in layers, signals flow from input (left) to output (right)
  - Structure is fully connected (i.e., dense)



Autoencoder through FFNNs

- Encoder-decoder FFNN architecture
  - Intended to reproduce the input



Autoencoder caveats

- If after training $\boldsymbol{x} = \hat{\boldsymbol{x}}$ everywhere
  - This is useless and must be avoided !!!
- Autoencoders (AE) should be trained such that
  - They are unable to learn to copy perfectly
  - They only copy approximately
  - And only copy input that resembles input data
- In this way AE
  - Must prioritize which aspects of the input should be copied
  - Usually learn useful properties of the data

Example: linear multiplication by identity matrix copies perfectly every input vector but is useless

AE training

- Unsupervised
  - Label (target output) is the input data itself
  - Are trained using gradient descent as any FFNN
- Standard gradient descent techniques
  - Batch-mode gradient descent: all data points considered to compute the true error derivative wrt the FFNN weights
  - Stochastic gradient descent (SGD): 1) one input vector at a time is fed to the FFNN, 2) gradient computed solely based on it, 3) network weights are updated using gradient descent of this pointwise gradient, 4) reiterate for all points
  - Mini-batches: between batch-mode and SGD

Learning strategies

- Objective
  - Prevent the AE from *just copying* the data
- Solutions
  - Limit the AE approximation *capacity*
- Popular strategies
  - Undercomplete AEs
  - Sparse AEs
  - Denoising AEs

Undercomplete AE

- Input $\boldsymbol{x}$
- Output $\hat{\boldsymbol{x}} = g(f(\boldsymbol{x}))$
- Loss (error) $\mathcal{L}(\boldsymbol{x}, g(f(\boldsymbol{x})))$
- Under completeness
  - Code dimension << input dimension
  - Forces the AE to capture the most salient data features
- Special case
  - Linear encoder/decoder and quadratic loss: the AE learns to span the same subspace as PCA (they are equivalent)

Sparse AE

- Input $\boldsymbol{x}$
- Output $\hat{\boldsymbol{x}} = g(f(\boldsymbol{x}))$
- Loss (error) $\mathcal{L}(\boldsymbol{x}, g(f(\boldsymbol{x})))$
- Training criterion $\mathcal{L}(\boldsymbol{x}, g(f(\boldsymbol{x}))) + \Omega(\boldsymbol{c})$

Sparsity penalty:

- It is a regularizer term
- Expresses a preference over functions
- For instance (Laplacian prior), we have:

$$\Omega(\boldsymbol{c}) = \lambda \sum_i |c_i|$$

Laplacian prior (1/2)

- Idea
  - See the sparse AE framework as approximating ML training of a generative model that has latent variables (code **c**)
  - Explicit joint distribution (*input* **x** and *latent* variable **c**)

$$p_{\text{model}}(\boldsymbol{x}, \boldsymbol{c}) = p_{\text{model}}(\boldsymbol{c}) p_{\text{model}}(\boldsymbol{x}|\boldsymbol{c})$$

$$\log p_{\text{model}}(\boldsymbol{x}, \boldsymbol{c}) = \log p_{\text{model}}(\boldsymbol{c}) + \log p_{\text{model}}(\boldsymbol{x}|\boldsymbol{c})$$

- Laplacian prior over c_i (assume c_i i.i.d.)

$$p_{\text{model}}(c_i) = \frac{\lambda}{2} e^{-\lambda c_i}$$

Laplacian prior (2/2)

- Laplacian prior over c_i

$$p_{\text{model}}(c_i) = \frac{\lambda}{2} e^{-\lambda c_i}$$

- As a training cost, we take –log of the prior

$$-\log p_{\text{model}}(\boldsymbol{c}) = \sum_i \left(\lambda |c_i| - \log \frac{\lambda}{2} \right) = \Omega(\boldsymbol{c}) + \text{constant}$$

  - Minimizing the cost: amounts to maximizing the prior pdf
  - Other priors are possible: lead to different penalties
  - This show why the features learned by an AE are useful: they describe the latent variables that explain the input

Denoising AE (1/3)

- Rather than constrain the representation (the code)
  - Train the AE for a more challenging task:
  - cleaning partially corrupted input (denoising)
- From [Vincent10]:

"a good representation is one that can be obtained robustly ***from a corrupted input*** *and that will be* ***useful for recovering the corresponding clean input****..."*

[Vincent10] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, P.A. Manzagol, "Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion," Journal of Machine Learning Research, 2010.

Denoising AE (1/3)

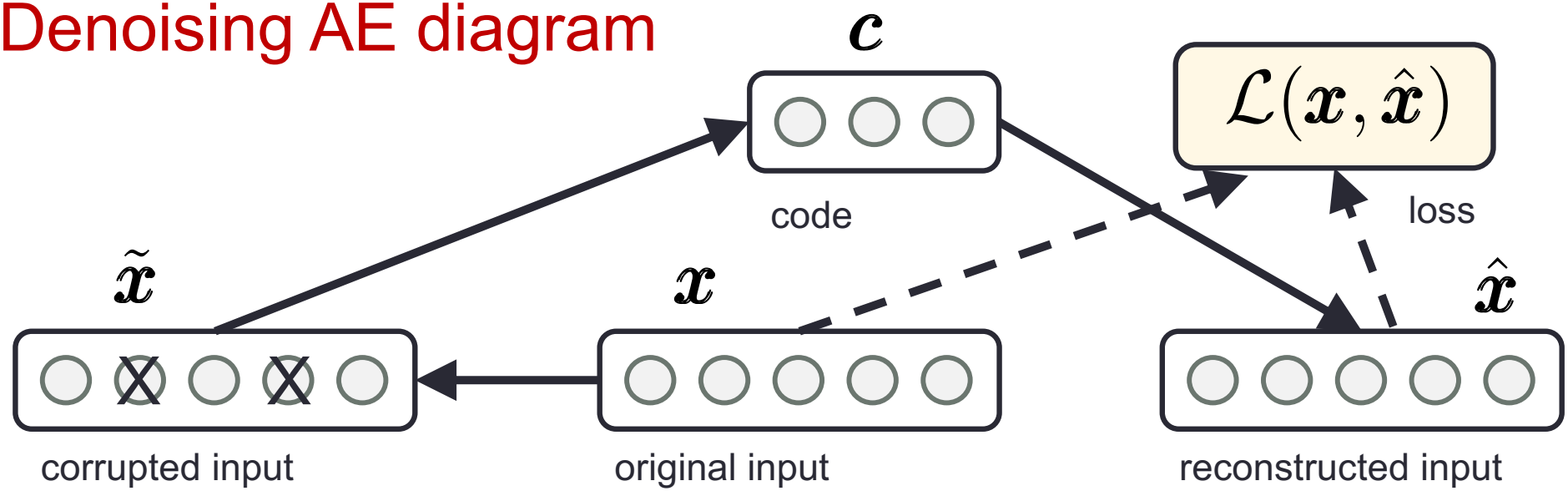
- Rather than constrain the representation (the code)
  - Train the AE for a more challenging task:
  - cleaning partially corrupted input (denoising)
- From [Vincent10]:

"...our goal is not the task of denoising per se. Rather, ***denoising is advocated and investigated as a training criterion for learning to extract useful features*** *that will constitute better higher level representations..."*

[Vincent10] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, P.A. Manzagol, "Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion," Journal of Machine Learning Research, 2010.

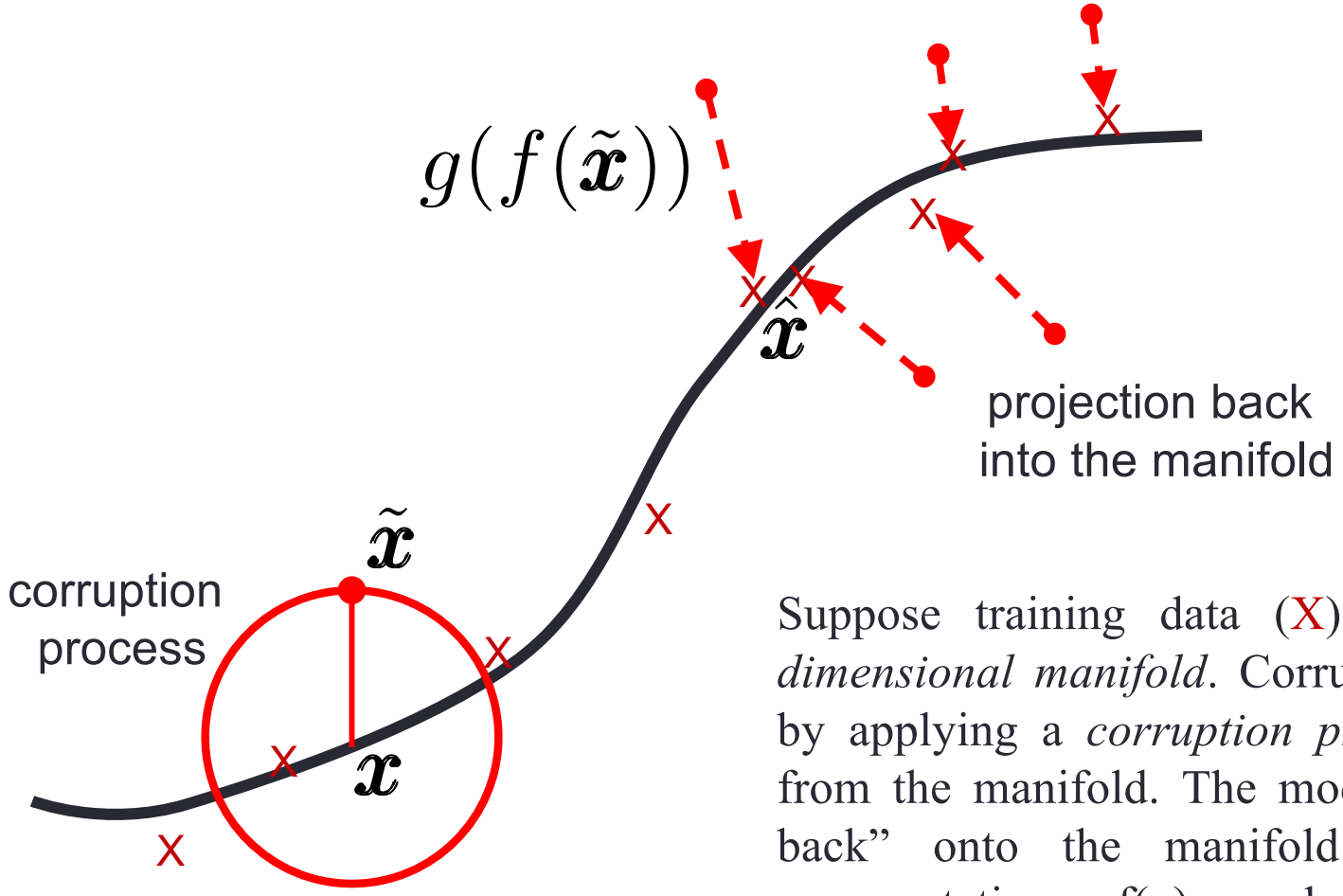
Denoising AE (2/3)

- Rationale
  - 1) It is expected that a high level representation should be rather stable under a corruption of the input
  - 2) It is expected that performing the denoising task requires extracting features that capture useful features in the input distribution
- Denoising AE diagram



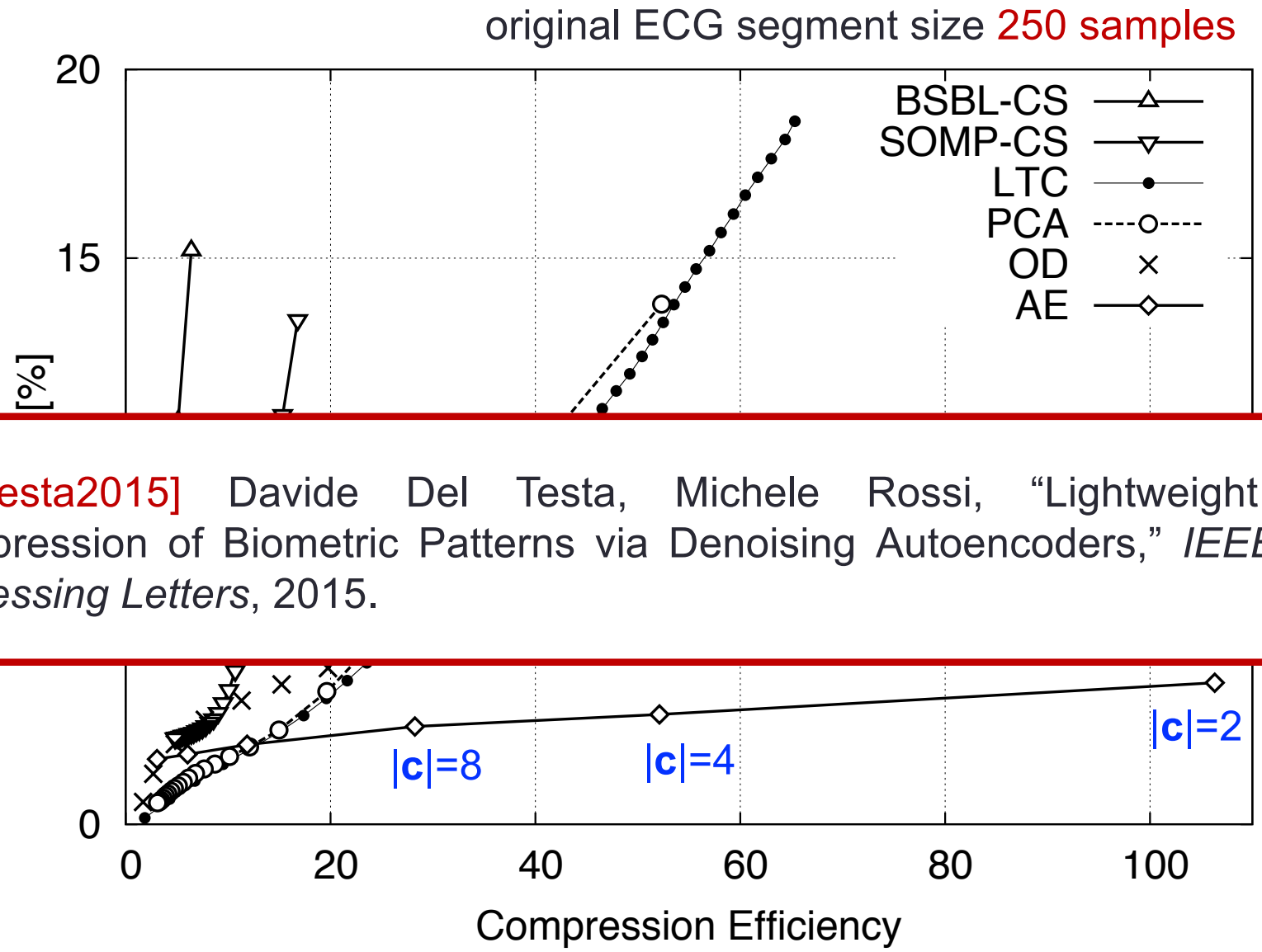
Denoising AE (3/3)

- Geometrical interpretation - manifold learning



Suppose training data (X) concentrate near a *low-dimensional manifold*. Corrupted examples (.) obtained by applying a *corruption process* generally lie farther from the manifold. The model learns to "project them back" onto the manifold. Thus, the intermediate representation **c**=f(**x**) may be interpreted as a coordinate system for points on the manifold.

Denoising AE – example results (ECG)



[DelTesta2015] Davide Del Testa, Michele Rossi, "Lightweight Lossy Compression of Biometric Patterns via Denoising Autoencoders," *IEEE Signal Processing Letters*, 2015.

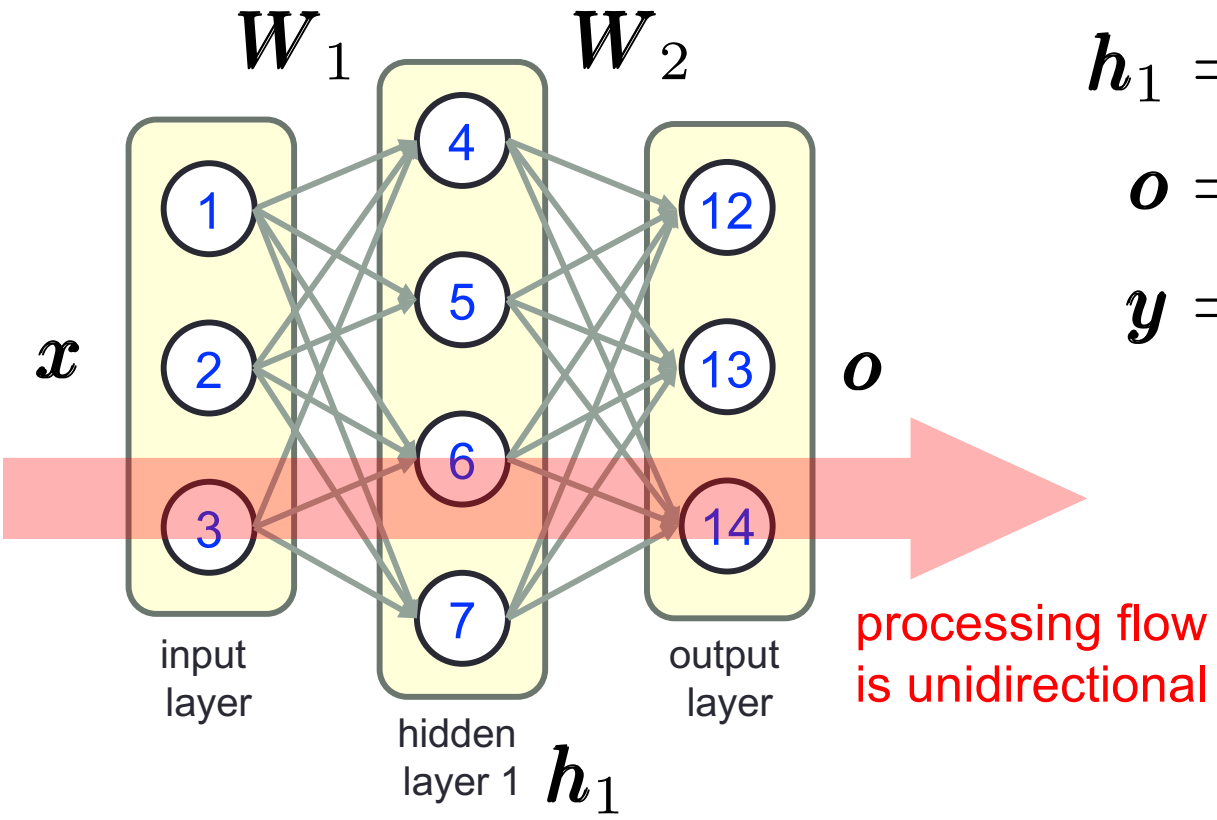
So far

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

RECURRENT NEURAL NETWORKS (RNN)

FFNN recap

- FFNN
  - Stack one layer on top of another
  - Output of last layer is a function of the input
  - Can be decomposed in layer-wise transforms



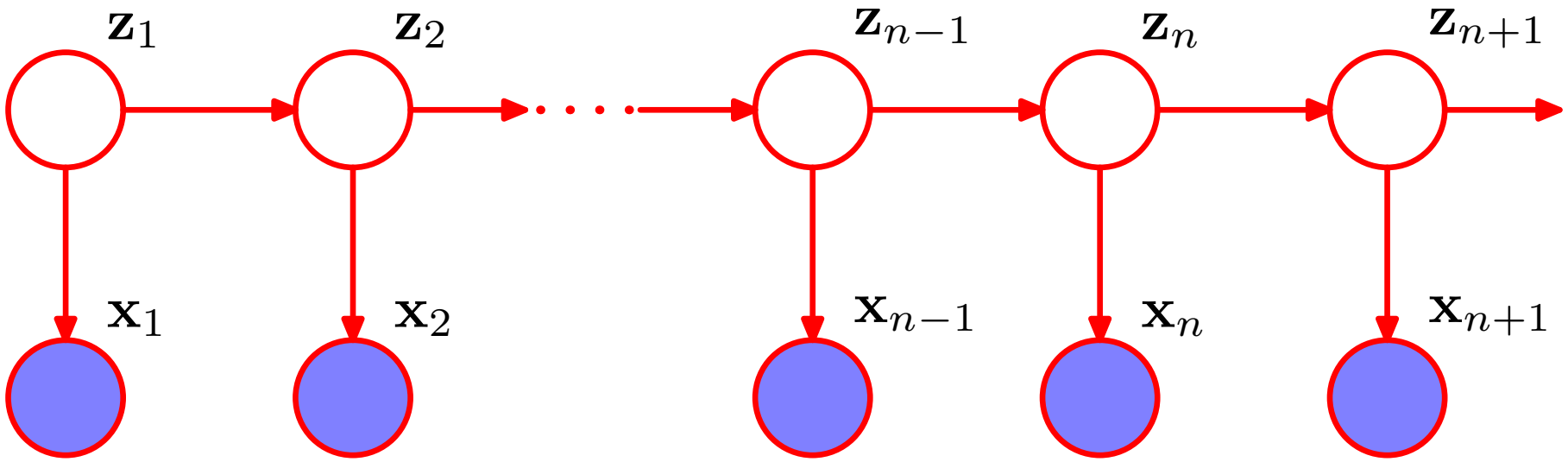
$$h_1 = \sigma(W_1 x + b_1)$$
$$o = \sigma(W_2 h_1 + b_2)$$
$$y = \mathrm{softmax}(o)$$

FFNN limitations

- FFNN
  - 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

Hidden Markov Model (HMM)

- Graphical model
- States: white filled circles
- 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
  - 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)

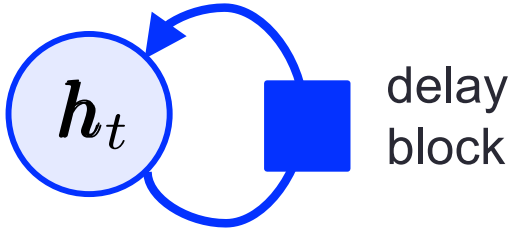
What's wrong with HMM?

- 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)

RNN – recursion

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

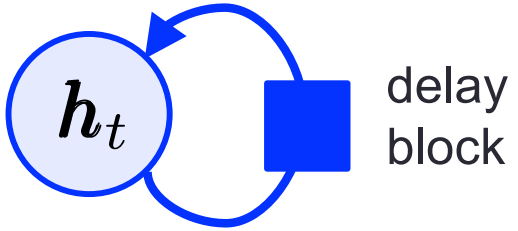
$$\boldsymbol{h}_t = f(\boldsymbol{h}_{t-1}; \boldsymbol{\theta})$$



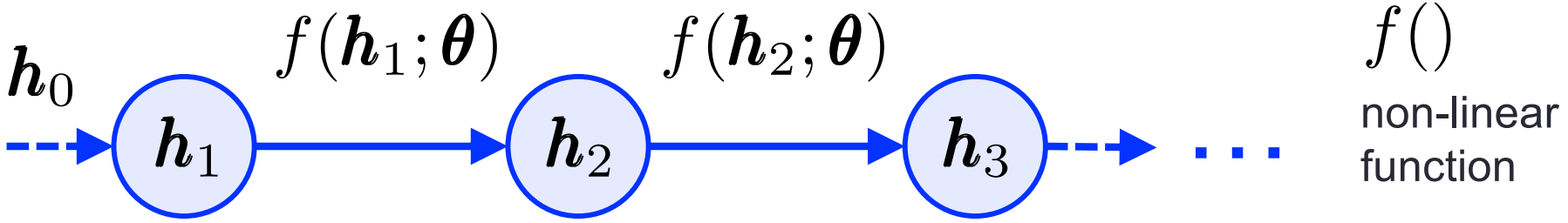
RNN – unfolding

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

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



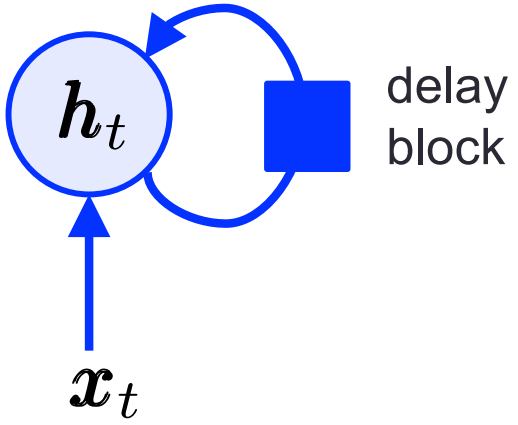
- Unfolding
  - Lead to a Directed Acyclic Graph (DAG) structure
  - Much preferable for training weights



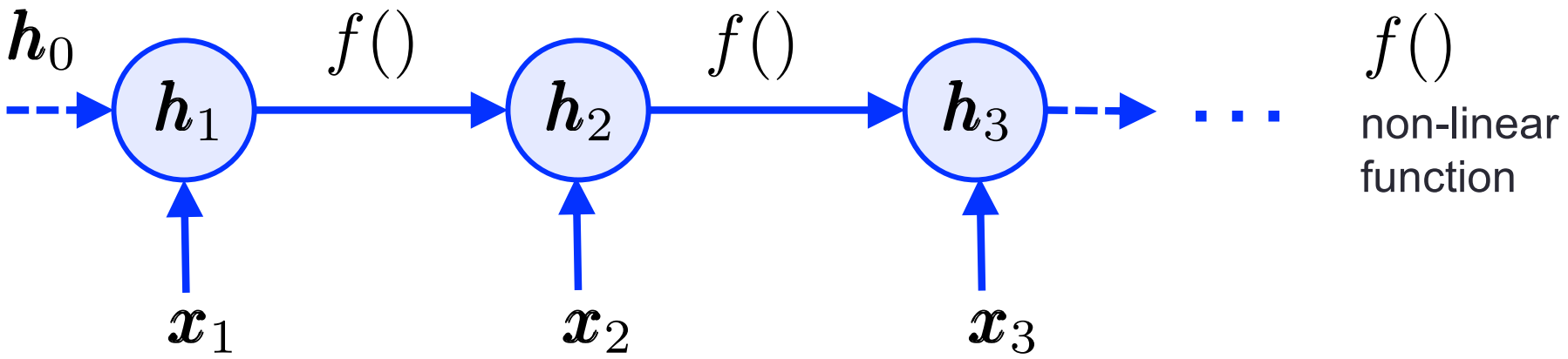
RNN with external input

- RNN
  - The recursion can be made more complex
- Recursive function, no input

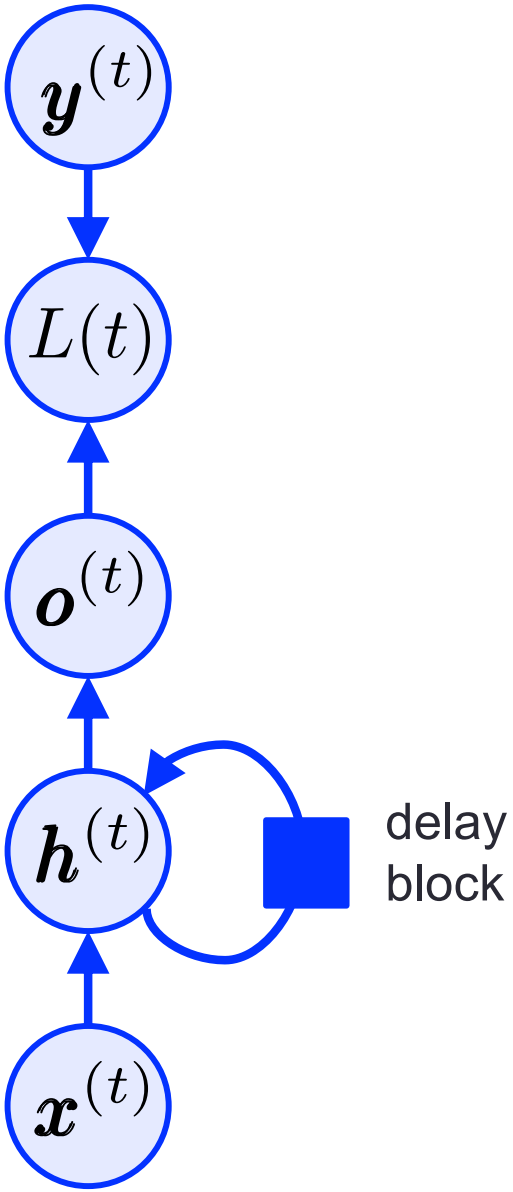
$$\boldsymbol{h}_t = f(\boldsymbol{h}_{t-1}, \boldsymbol{x}_t; \boldsymbol{\theta})$$



- 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)

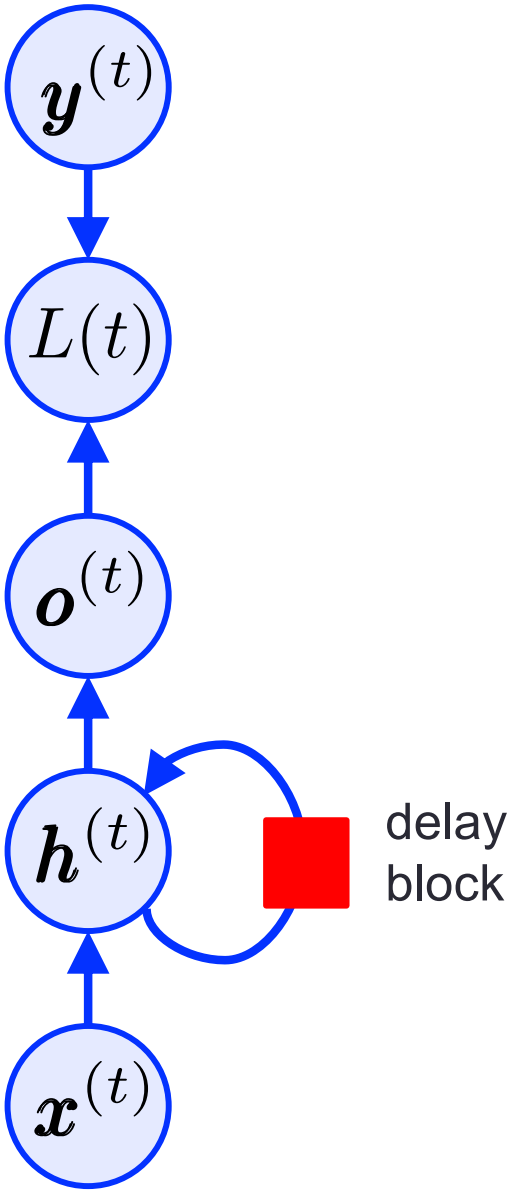
$$\boldsymbol{h}^{(t)} = \tanh\left(\boldsymbol{a}^{(t)}\right)$$

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)

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

Loss (error) function

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

Output

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

Internal state

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

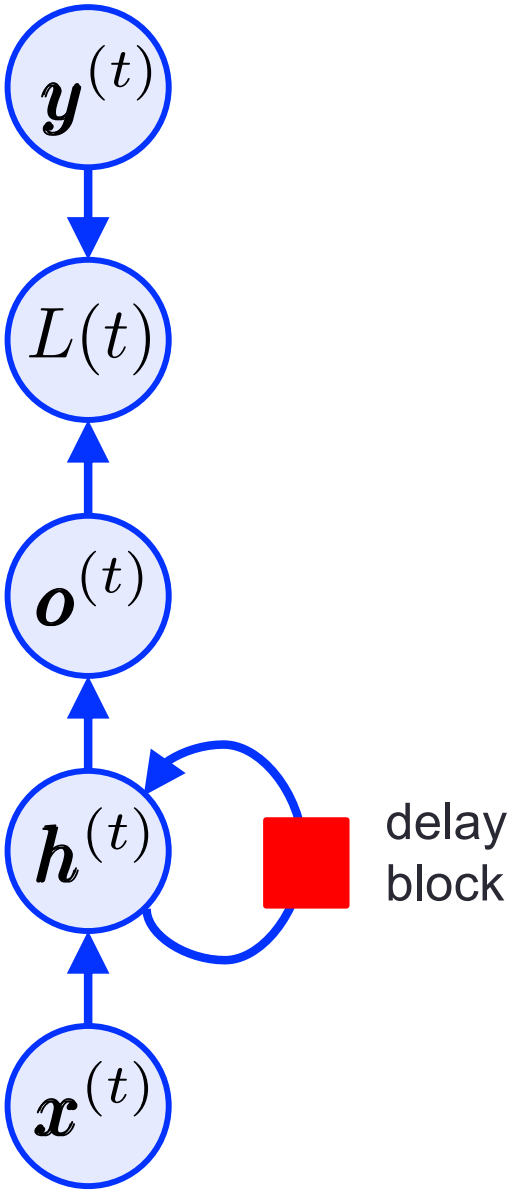
Activation

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

recurrence

Input at time t

A full-fledged RNN



Overall

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

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

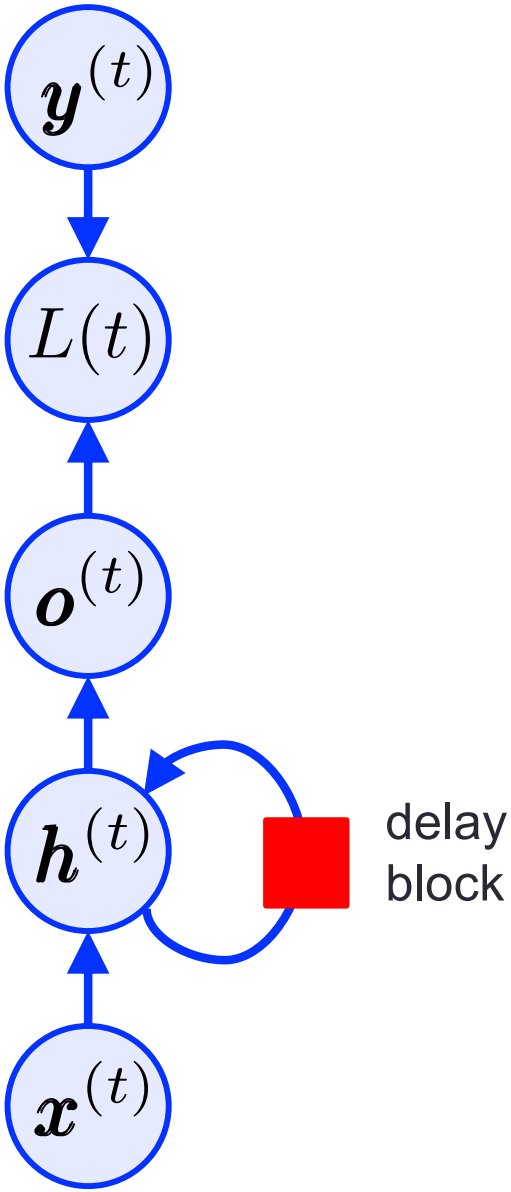
$$\boldsymbol{o}^{(t)} = \boldsymbol{V}\boldsymbol{h}^{(t)} + \boldsymbol{c} \quad \text{dense layer}$$

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

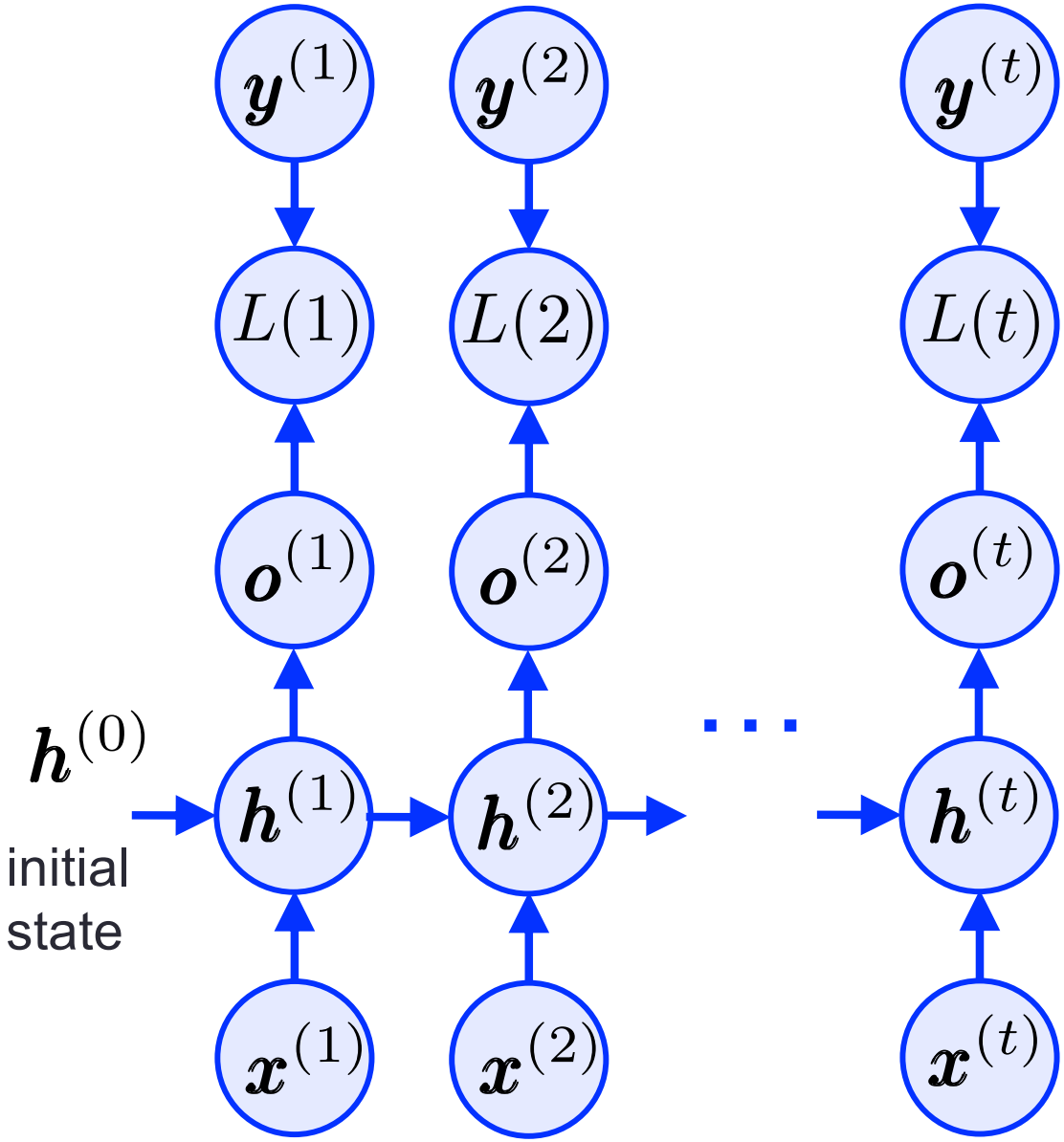
RNN hyperparameters (to be learned)

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

RNN unfolding



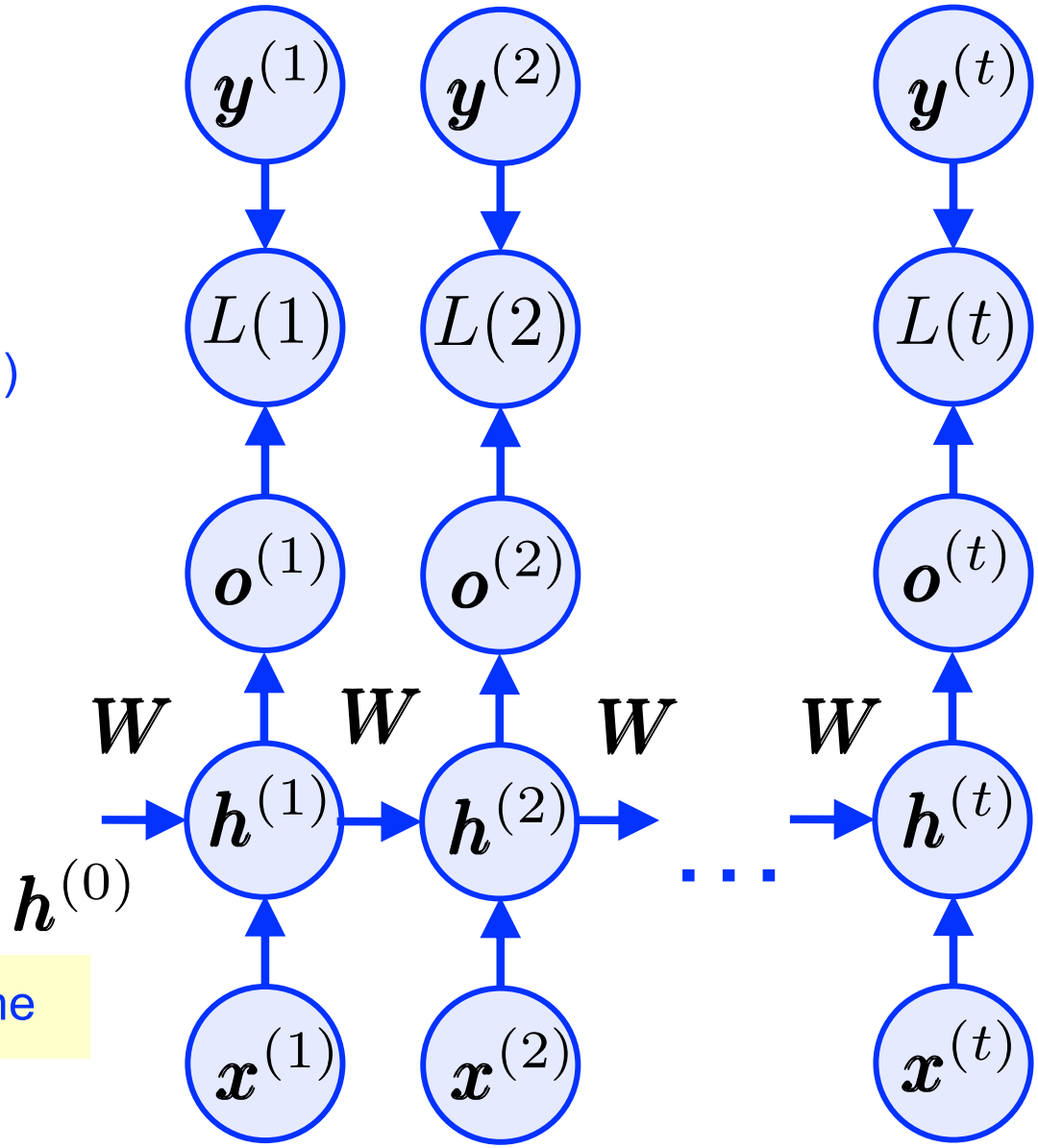
Unfolded graph is a standard DAG



Dependencies

- Let 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

Unfolded graph is a standard DAG



Input, output and loss

- Input at each t is a vector
  - For a classification problem, use *1-hot vectors*
    - K elements (number of classes)
    - only one element is 1 all other elements are 0
    - non-zero element indicates the input class
- Output is also a vector
  - Same size K, same encoding
- Loss function L(t)
  - For classification, cross-entropy loss

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

Input, output and loss

- Input at each t is a vector
  - For a regression problem, use vectors or real numbers
    - K elements
- Output is also a vector
  - Vector of real numbers
  - Same size K
- Loss function L(t)
  - For regression, e.g., Mean Square Error (MSE)

$$\hat{\boldsymbol{y}}^{(t)} = \boldsymbol{o}^{(t)}$$ (no need for softmax)

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

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** (1xm) and a scalar function y

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

the gradient (as column vector)

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 total error L 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):

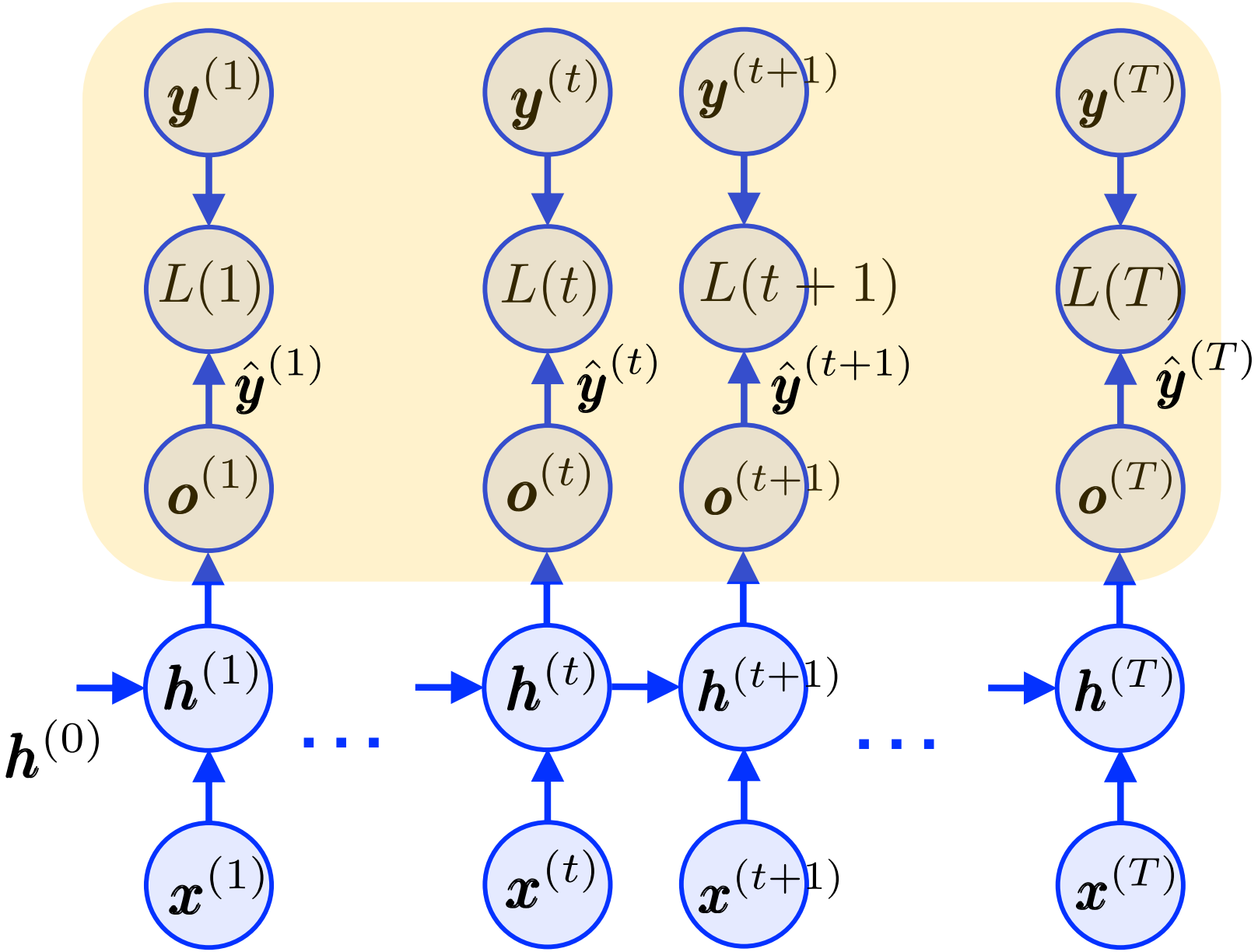
$$\frac{\partial L}{\partial \boldsymbol{V}}, \frac{\partial L}{\partial \boldsymbol{W}}, \frac{\partial L}{\partial \boldsymbol{U}}, \nabla_{\boldsymbol{b}}(L), \nabla_{\boldsymbol{c}}(L)$$

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

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

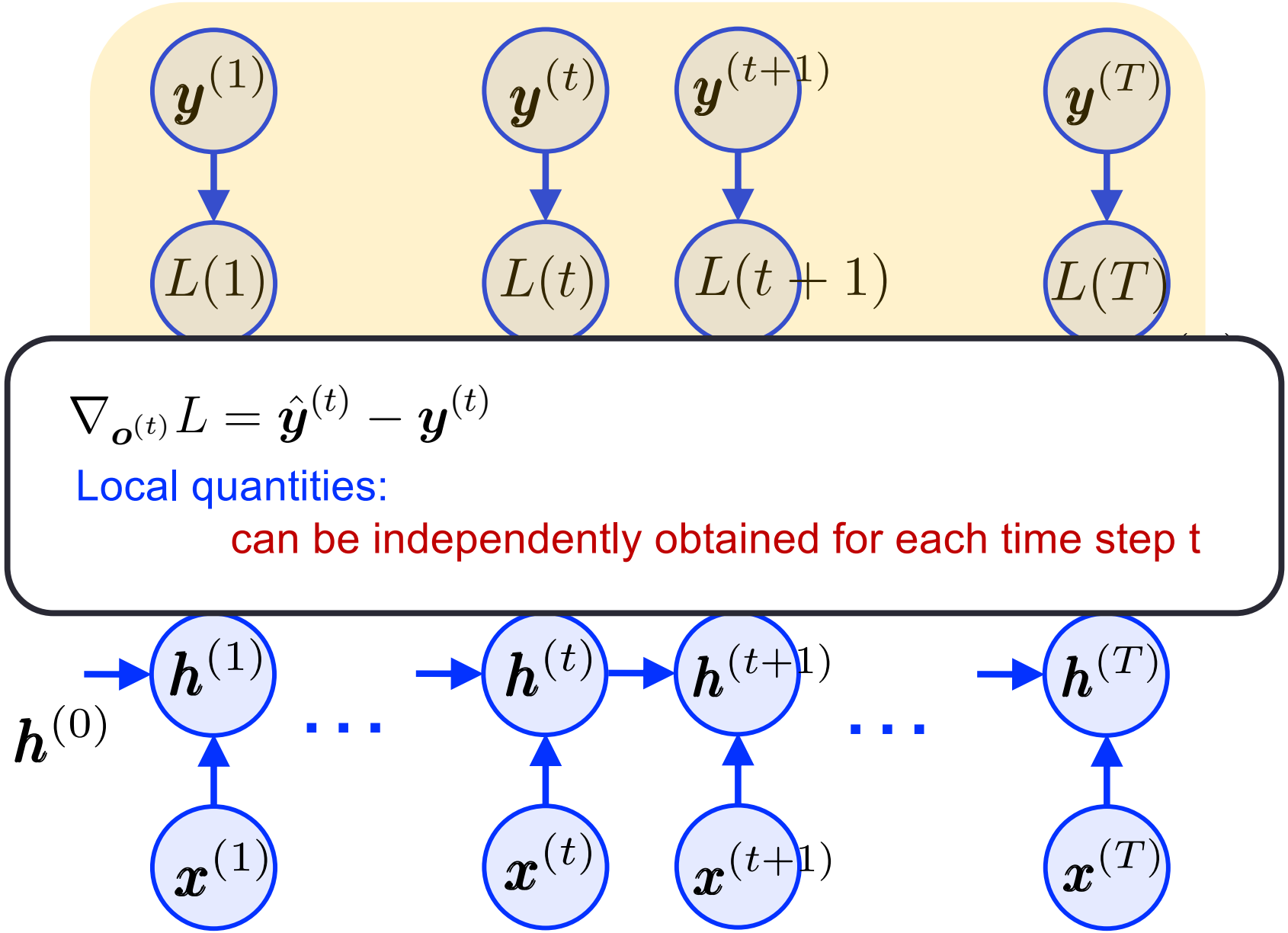
BTT – unfolded graph

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



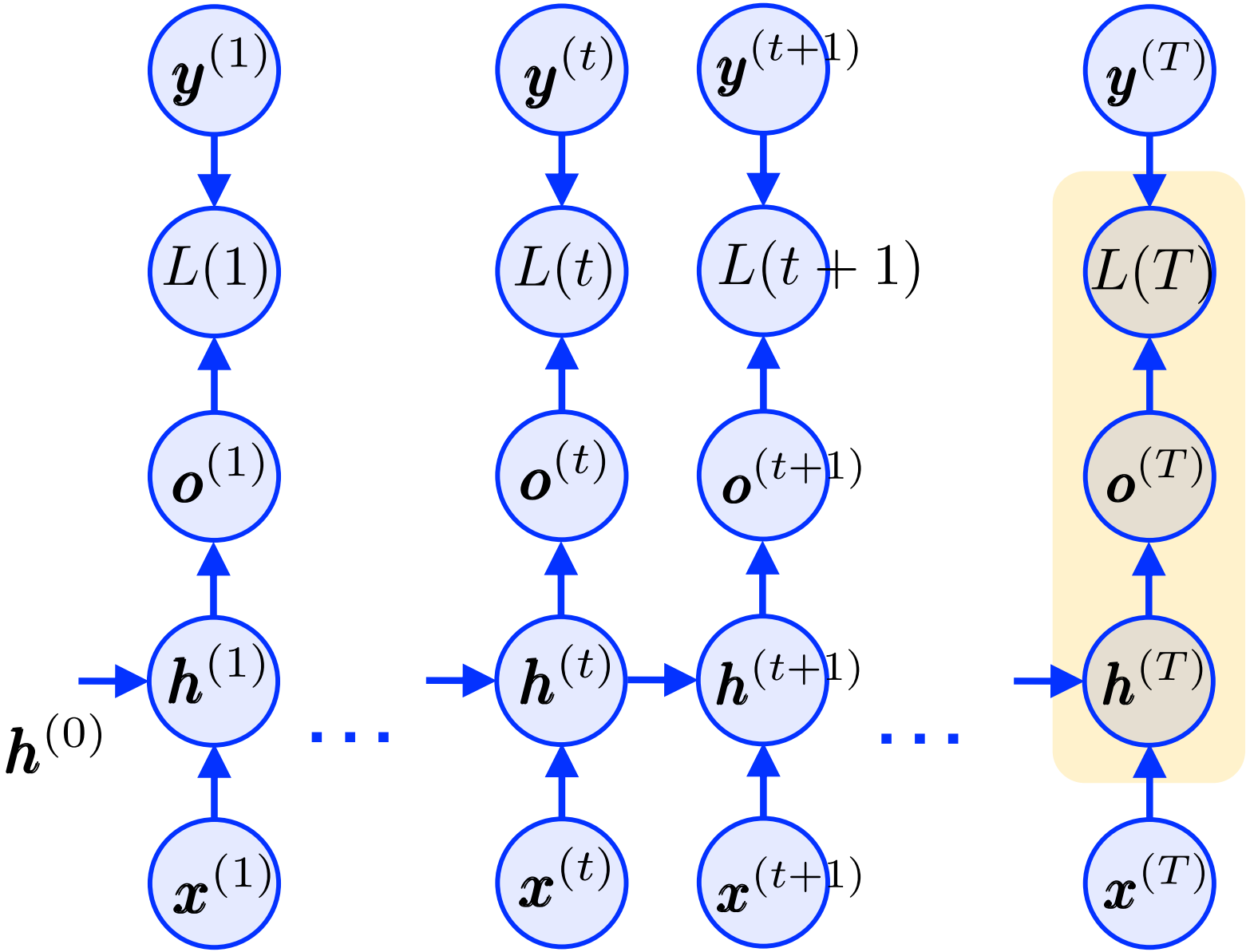
BTT – unfolded graph

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



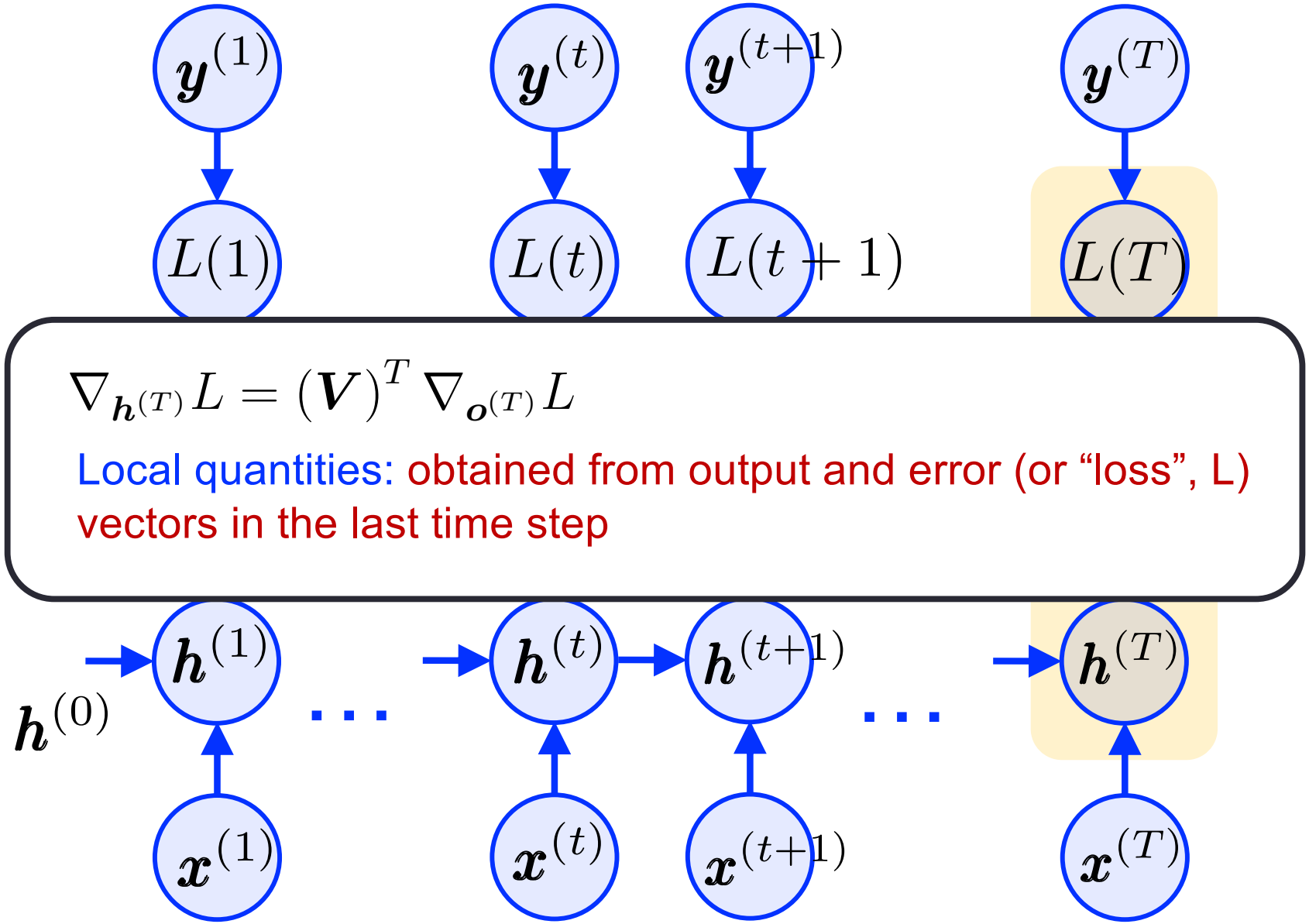
BTT – time T

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



BTT – time T

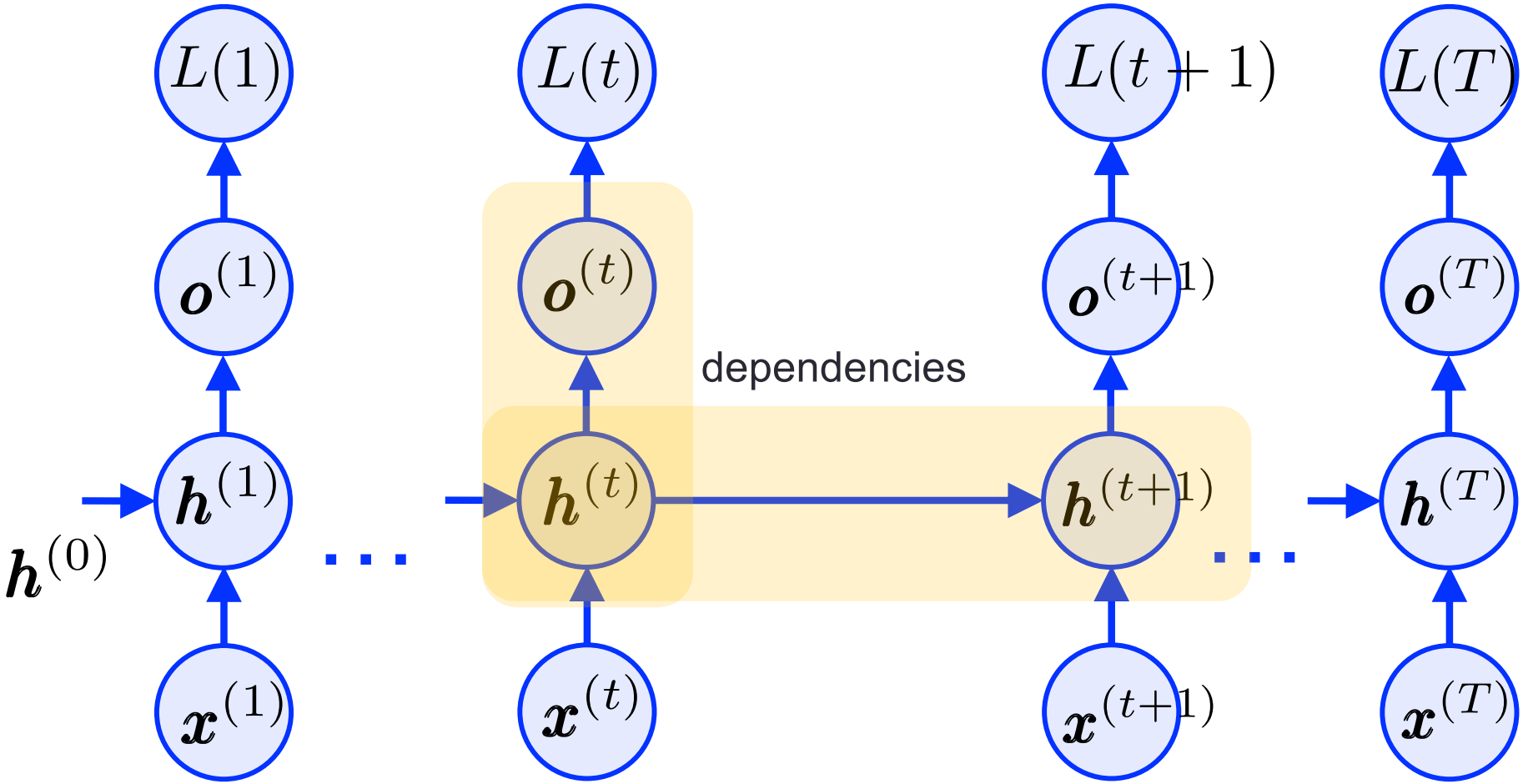
$$\nabla_{\boldsymbol{h}^{(T)}} L = (\boldsymbol{V})^T \nabla_{\boldsymbol{o}^{(T)}} L$$



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

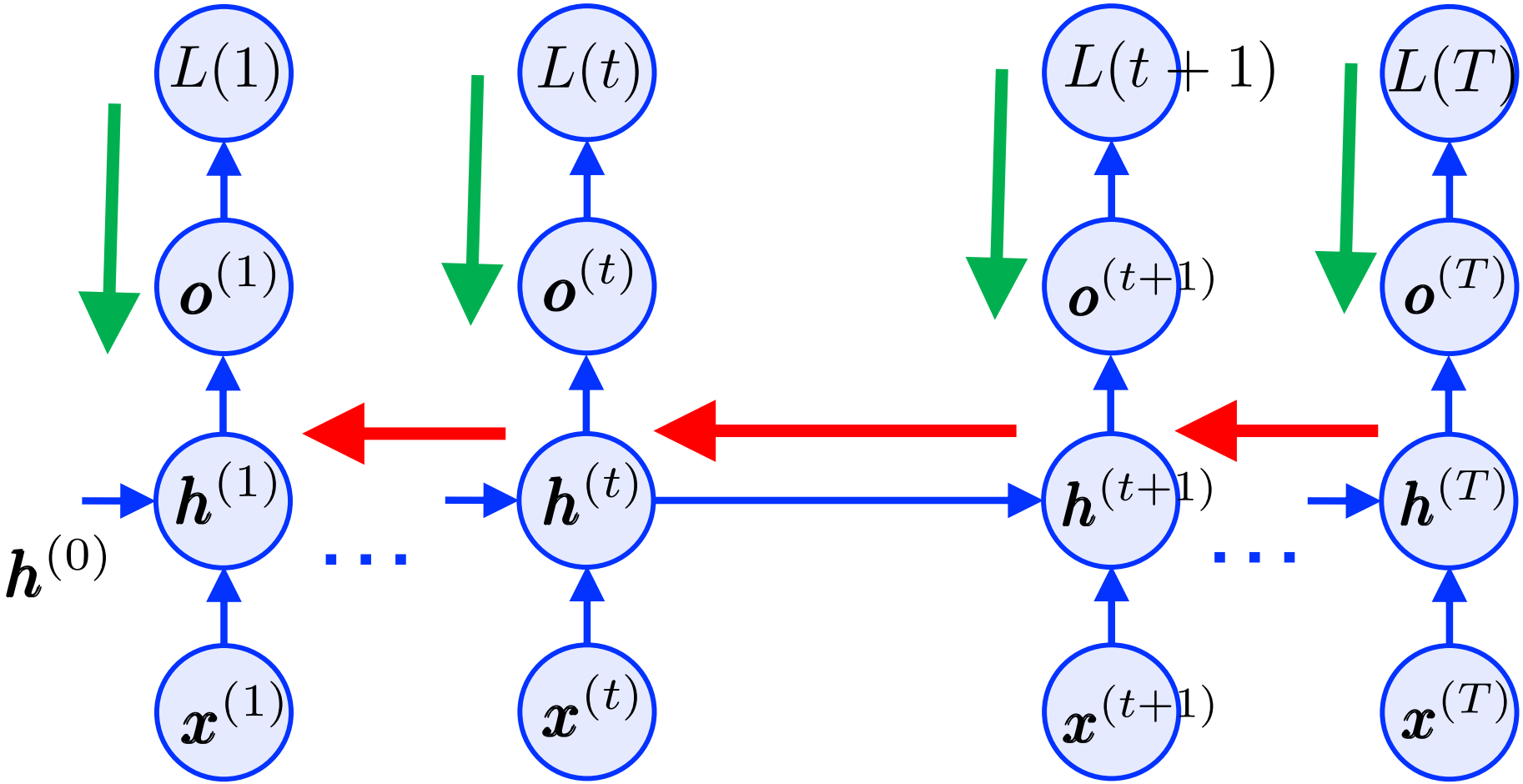
gradient of $\mathbf{h}^{(t)}$ is computed recursively

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

in compact form

recursive

local



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

The derivatives wrt the network params are [Goodfellow16]

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

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

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

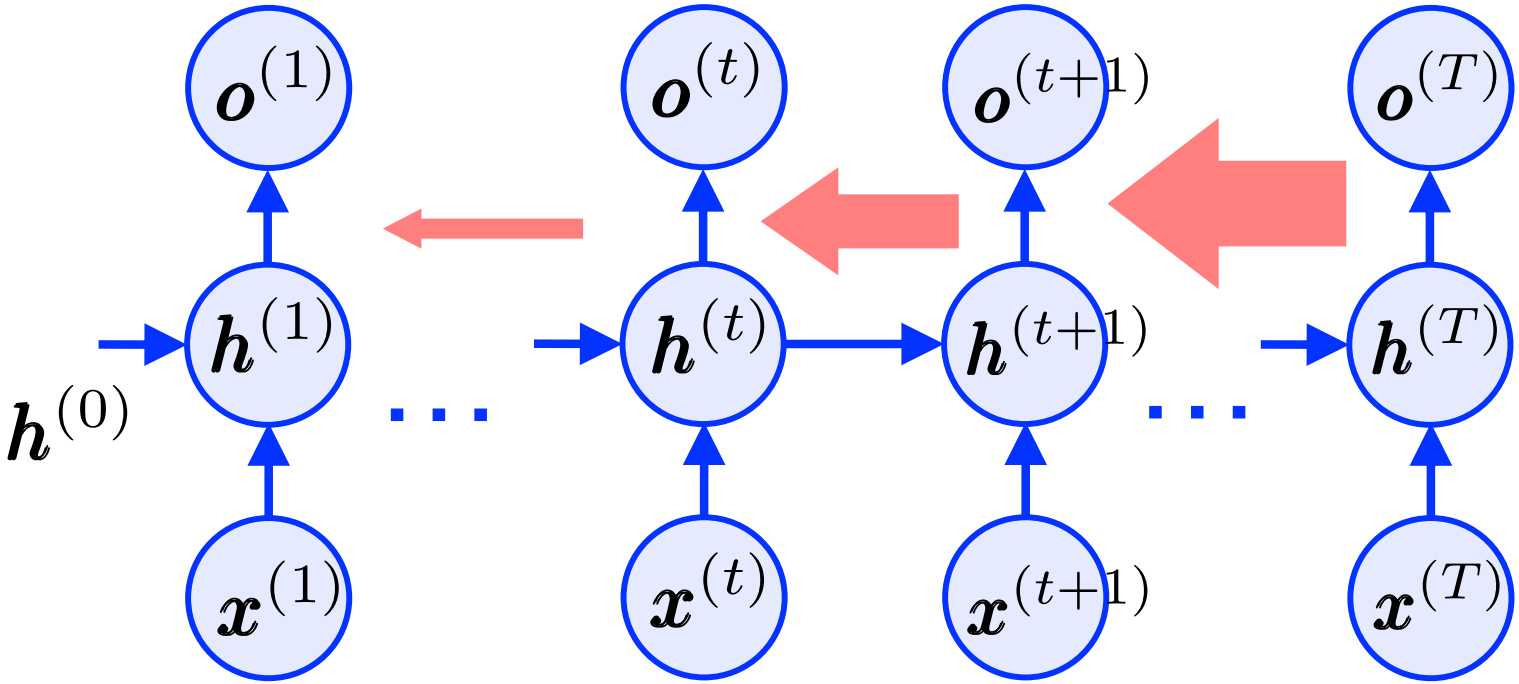
$$\nabla_{\boldsymbol{U}} L = \sum_{t=1}^{T} \operatorname{diag}\left(1 - \left(\boldsymbol{h}^{(t)}\right)^2\right) \left(\nabla_{\boldsymbol{h}^{(t)}} L\right) \left(\boldsymbol{x}^{(t)}\right)^T$$

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

BTT – vanishing gradients problem

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

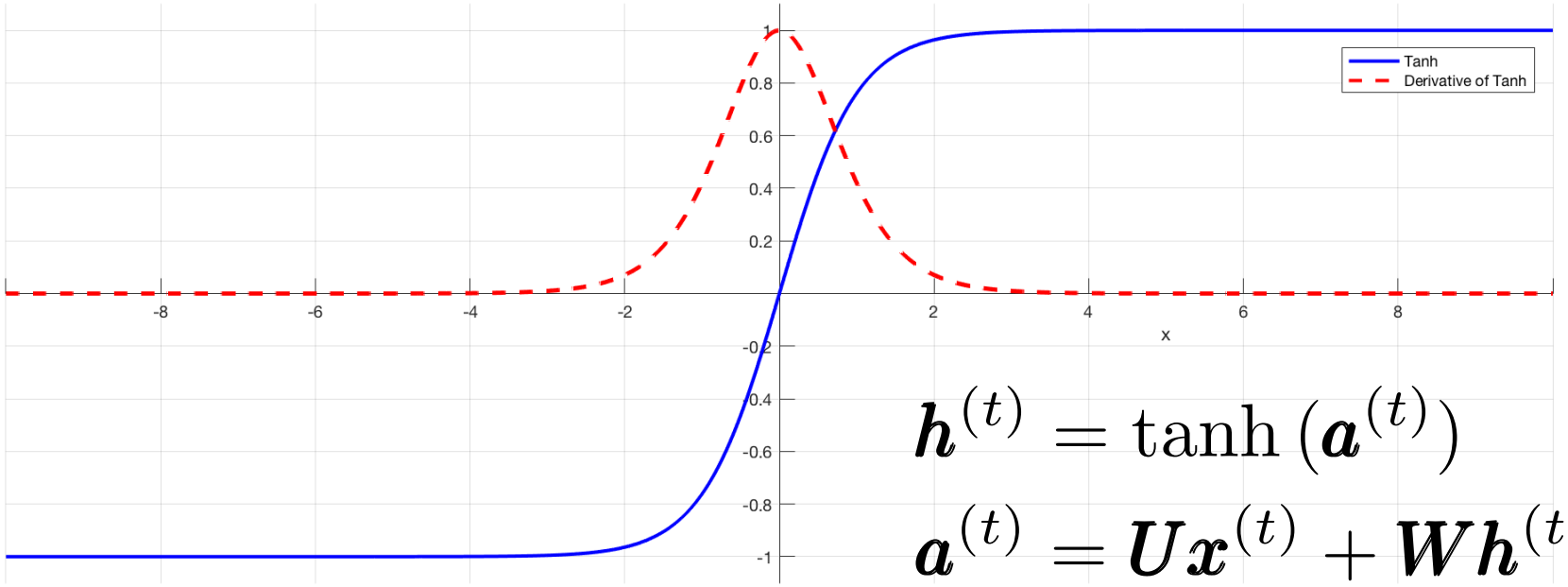
Note that: $\operatorname{diag}(1 - (\boldsymbol{h}^{(t)})^2) = \dfrac{\partial \tanh(\boldsymbol{h}^{(t)})}{\partial \boldsymbol{h}^{(t)}}$



BTT – vanishing gradients problem

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

Note that: $\operatorname{diag}(1 - (\boldsymbol{h}^{(t)})^2) = \dfrac{\partial \tanh(\boldsymbol{h}^{(t)})}{\partial \boldsymbol{h}^{(t)}}$



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

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

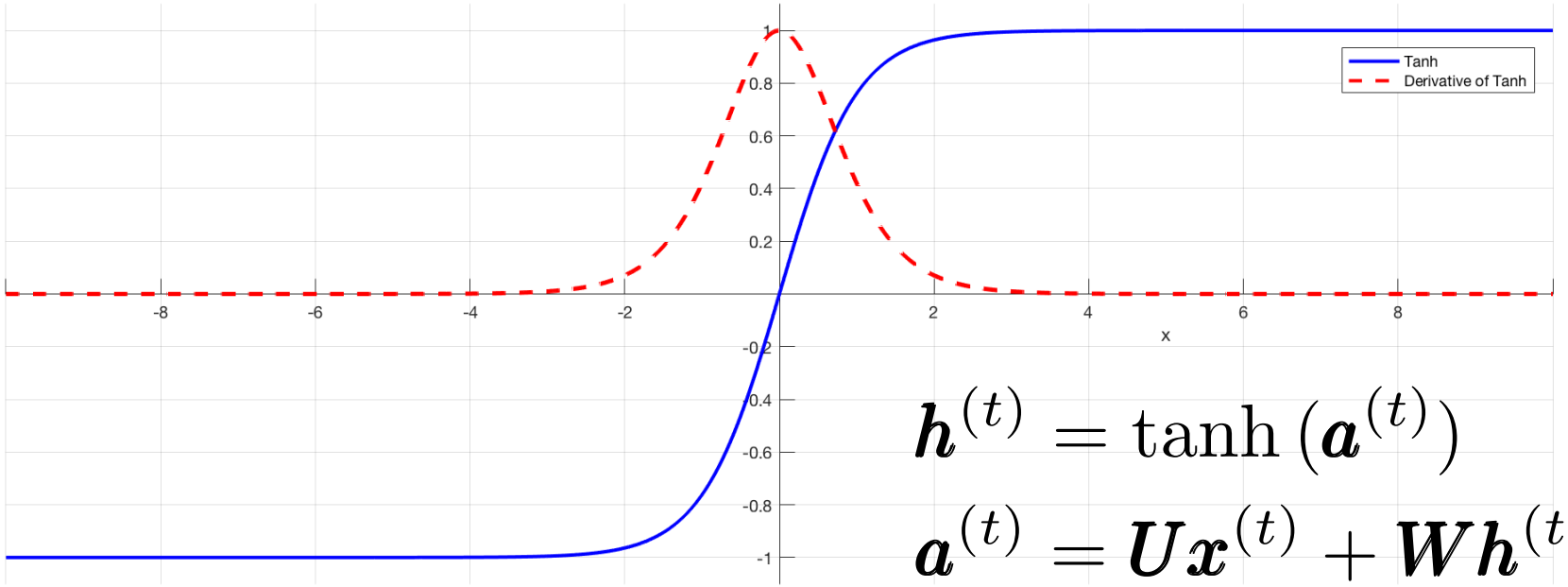
BTT – vanishing gradients problem

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

e.g., for large activations: $\boldsymbol{a}^{(t)} \uparrow \Rightarrow h_i^{(t)} \to 1$

The tanh gradient vanishes:

$$\operatorname{diag}(1 - (\boldsymbol{h}^{(t)})^2) \to \mathbf{0}$$



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

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

BTT – vanishing gradients problem

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

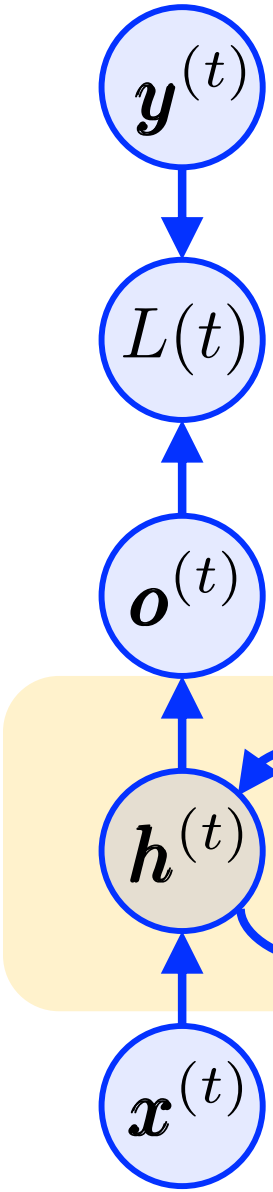
The tanh gradient vanishes:

$$\operatorname{diag}(1 - (\boldsymbol{h}^{(t)})^2) \rightarrow \mathbf{0}$$

As a consequence

- Gradients do not propagate all the way down the DAG
- Only *short term* dependencies are learned
- RNN are unable to fulfill their promise
  - very limited temporal structure learning

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 do no longer appear in a concatenated fashion in the gradient computation

GRU cells in four steps…

Definitions

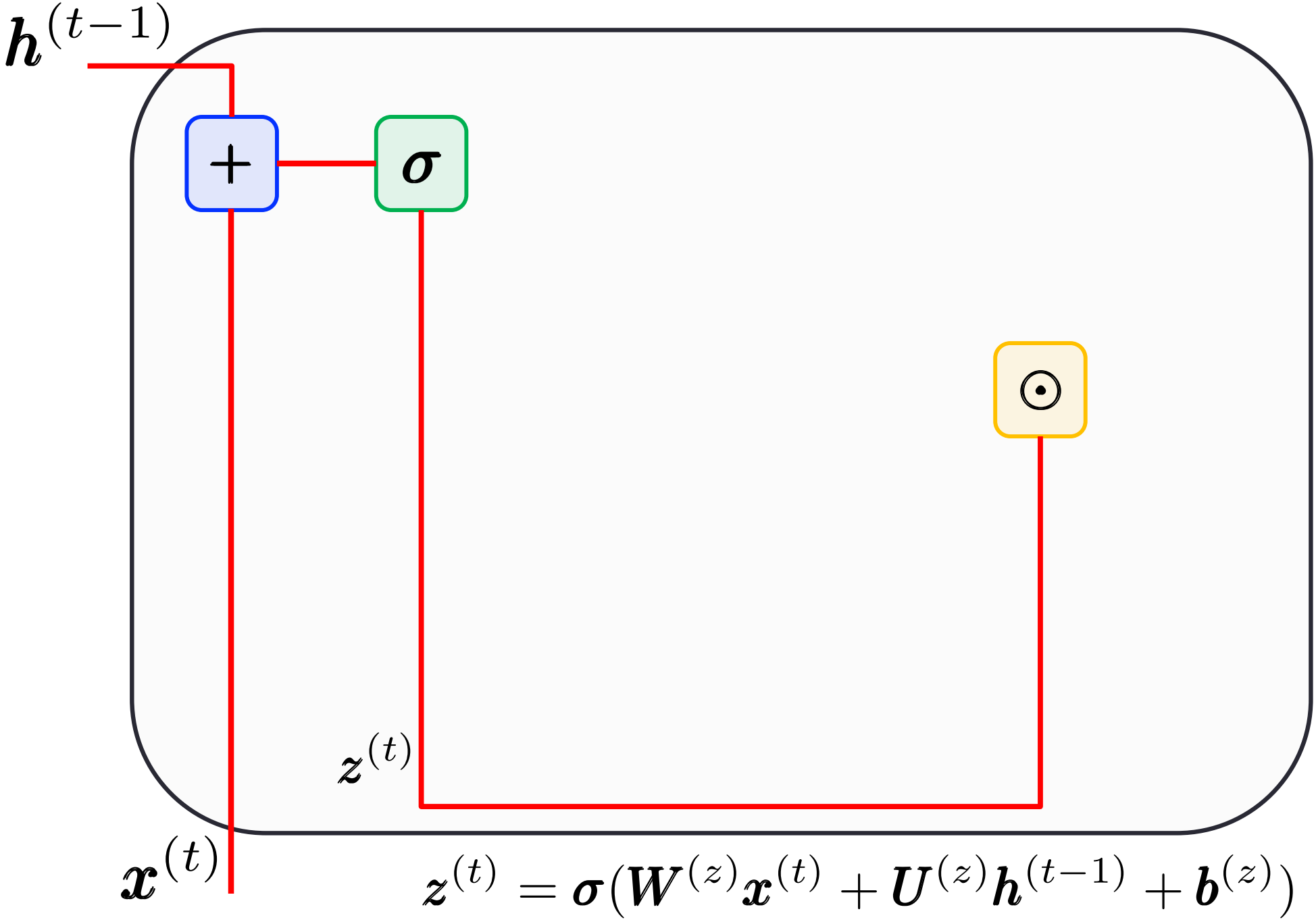
$\tanh$ hyperbolic tangent (applied element-wise)

σ sigmoid nonlinearity (applied element-wise)

$+$ vector addition

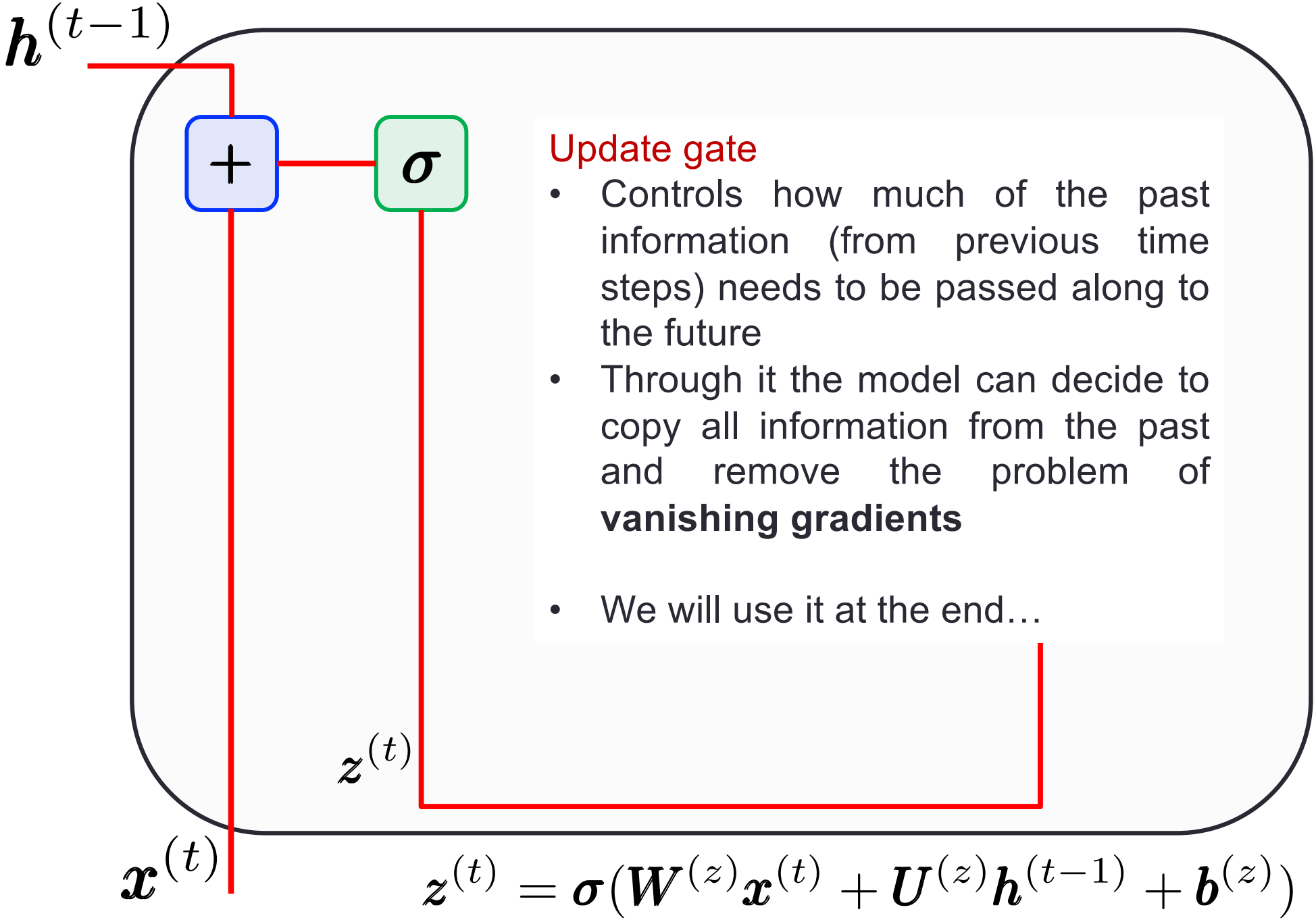
$\odot$ element-wise product between vectors
(also known as Hadamard product)

Step 1: update gate



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

Step 1: update gate



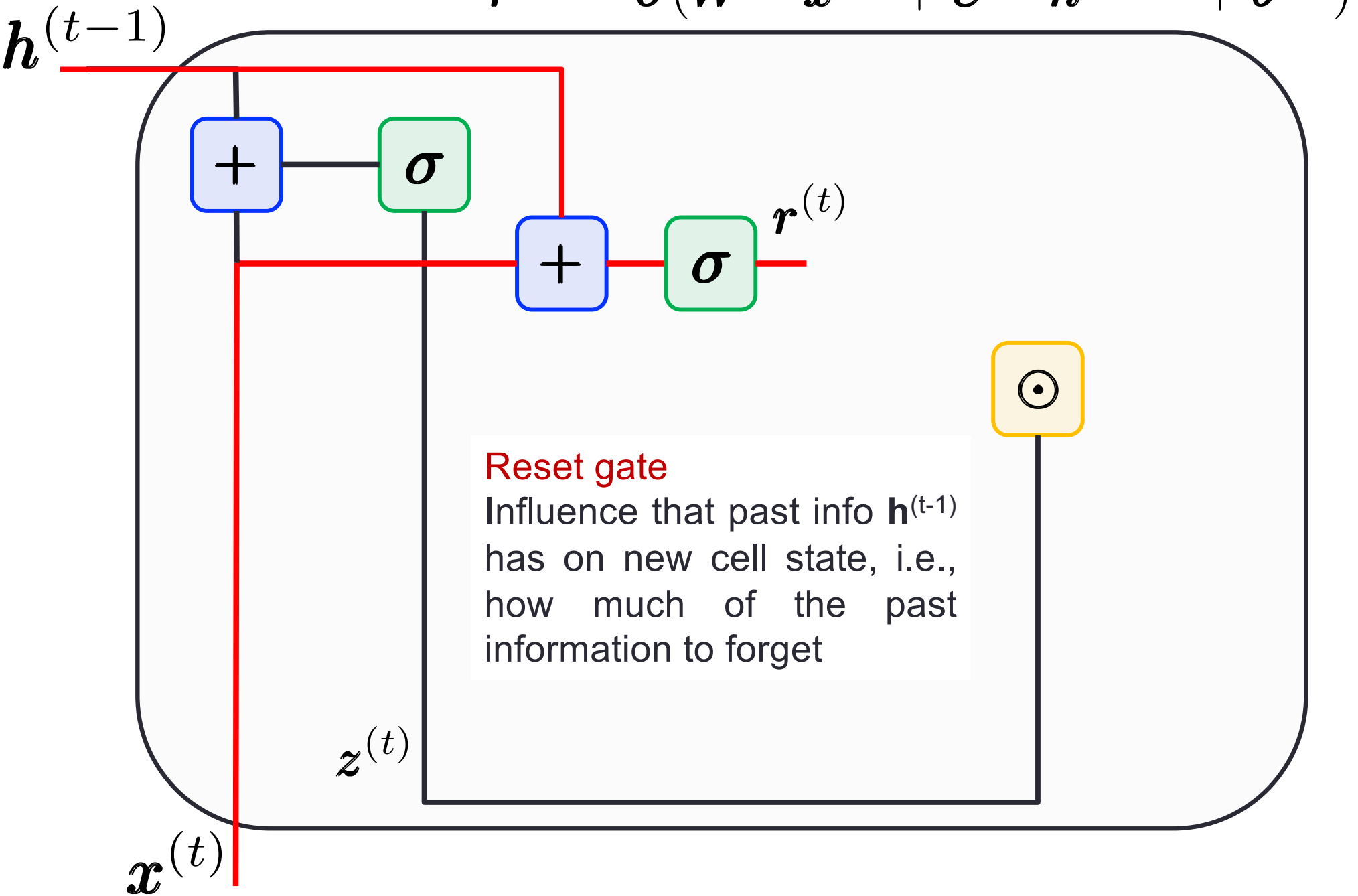
Update gate

- 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

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

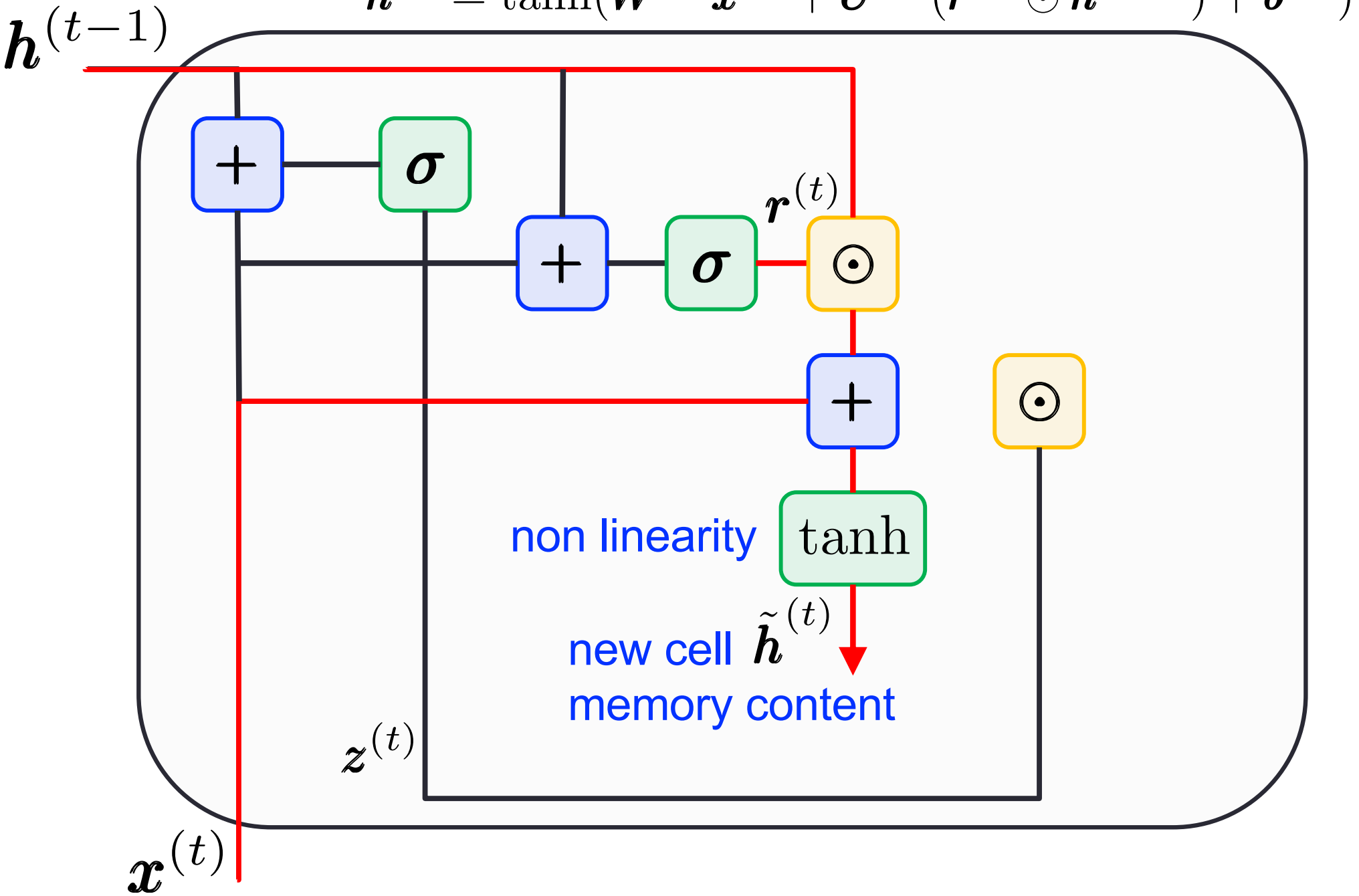


Reset gate

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

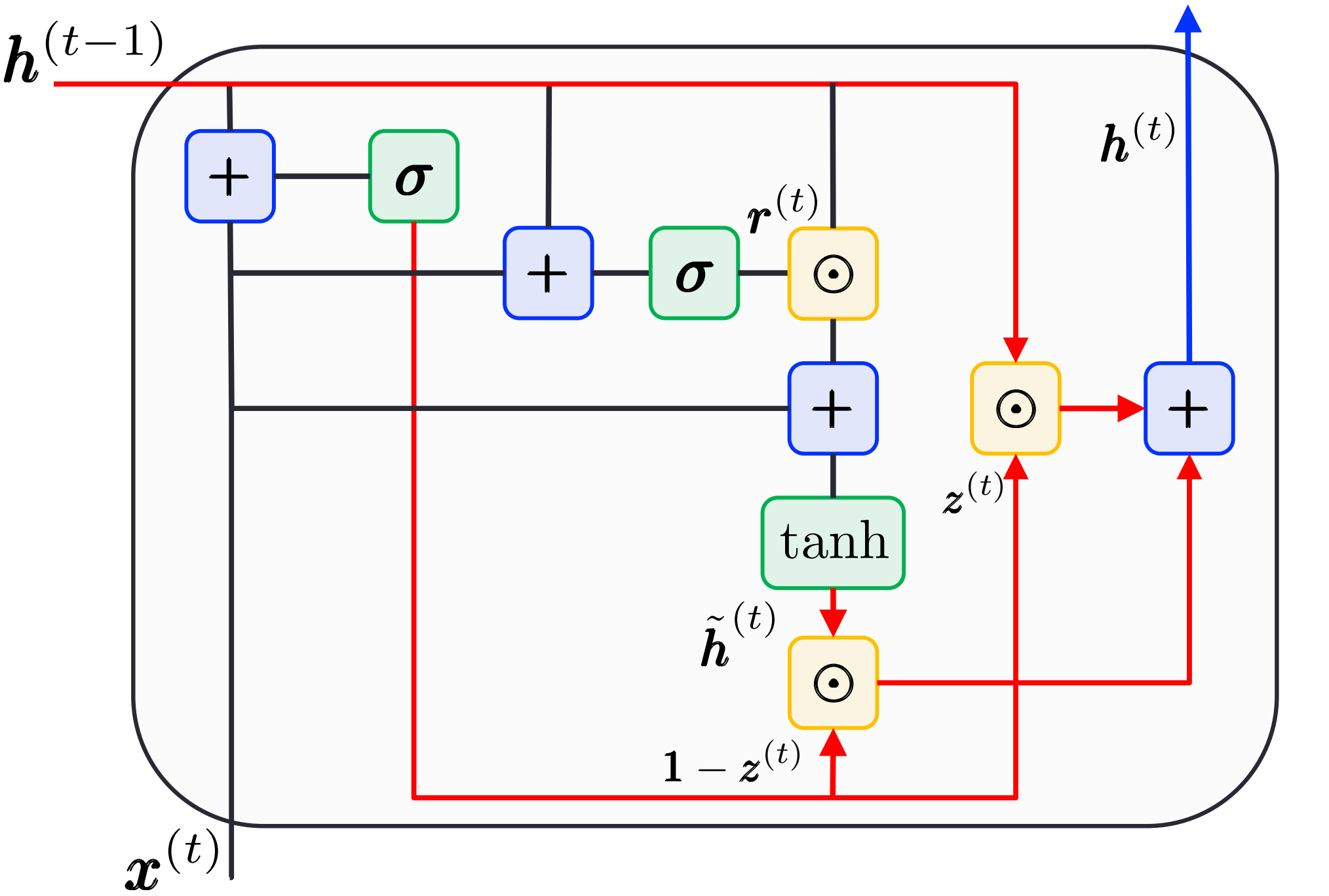
Step 3: current memory content

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

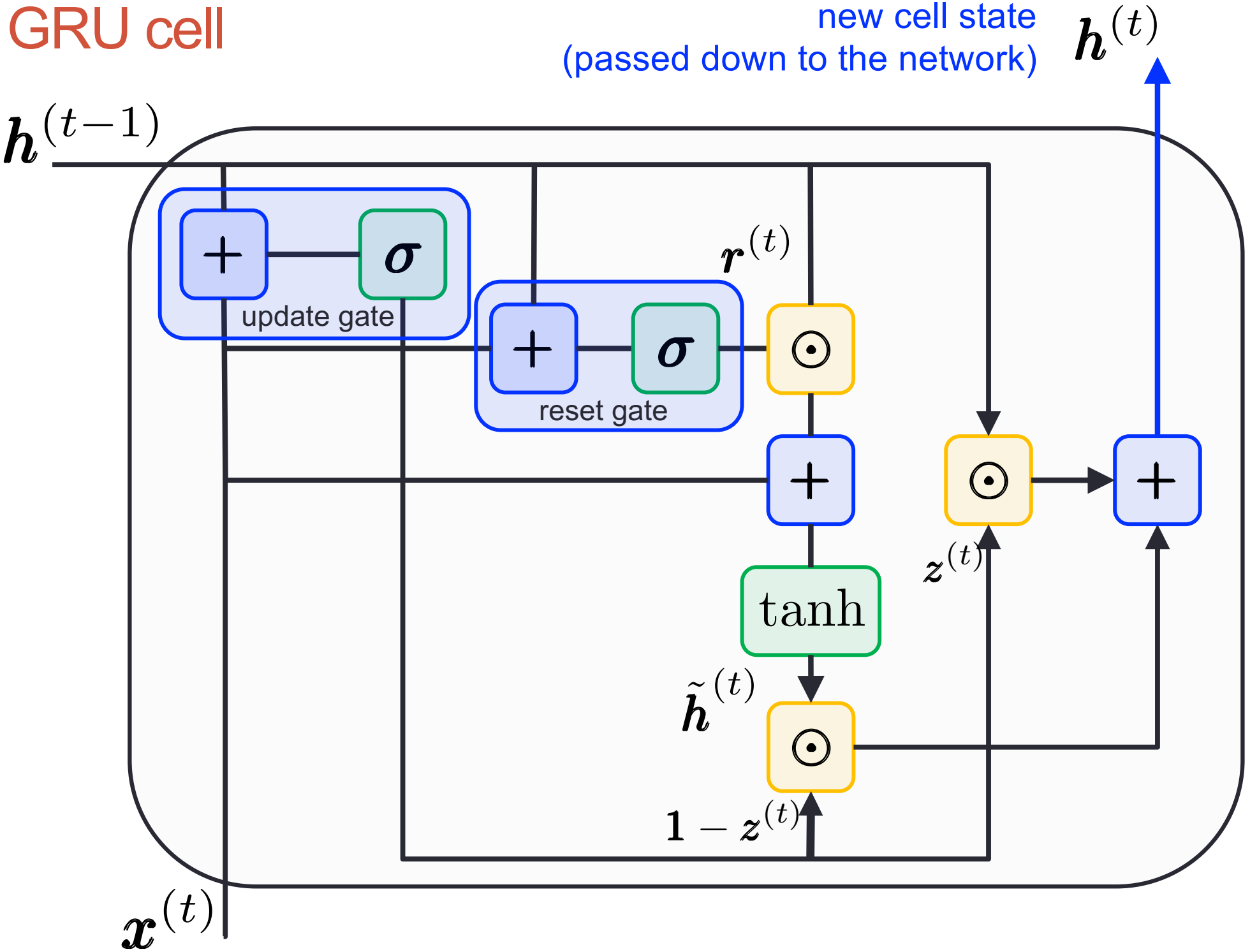


Step 4: final state

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



GRU cell


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

GRU cell "fully gated"

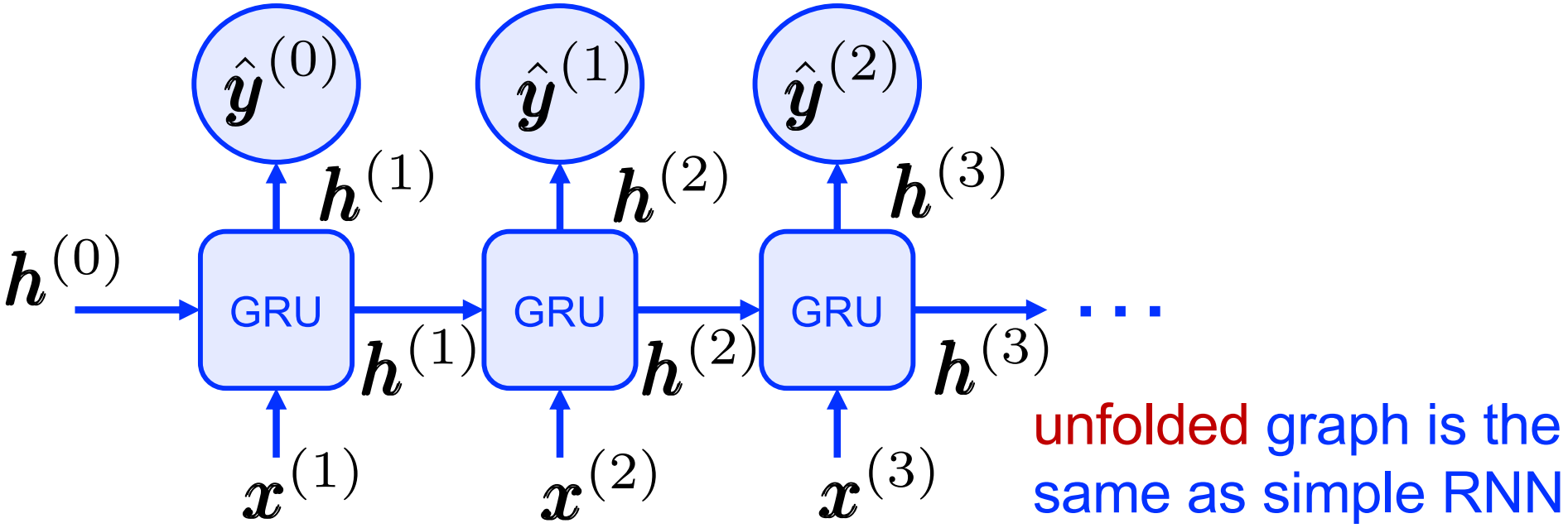
$$\boldsymbol{z}^{(t)} = \sigma(\boldsymbol{W}^{(z)}\boldsymbol{x}^{(t)} + \boldsymbol{U}^{(z)}\boldsymbol{h}^{(t-1)} + \boldsymbol{b}^{(z)})$$

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

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

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

fully gated unit



unfolded graph is the same as simple RNN

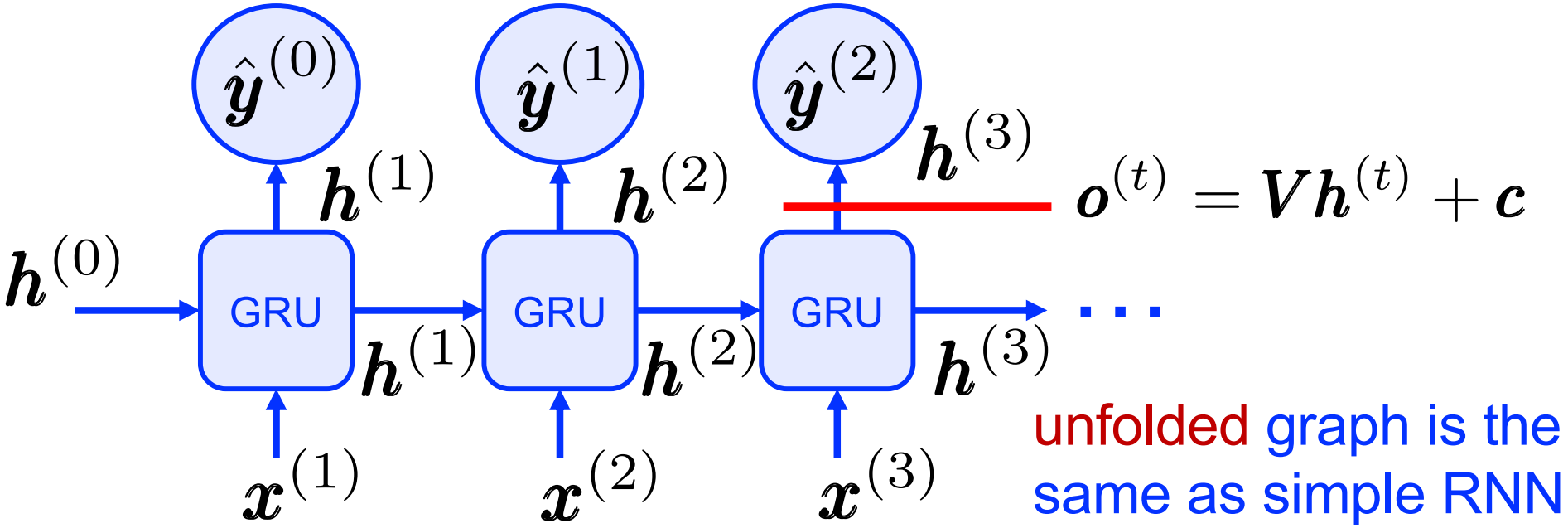
GRU cell "fully gated"

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

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

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

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



unfolded graph is the same as simple RNN

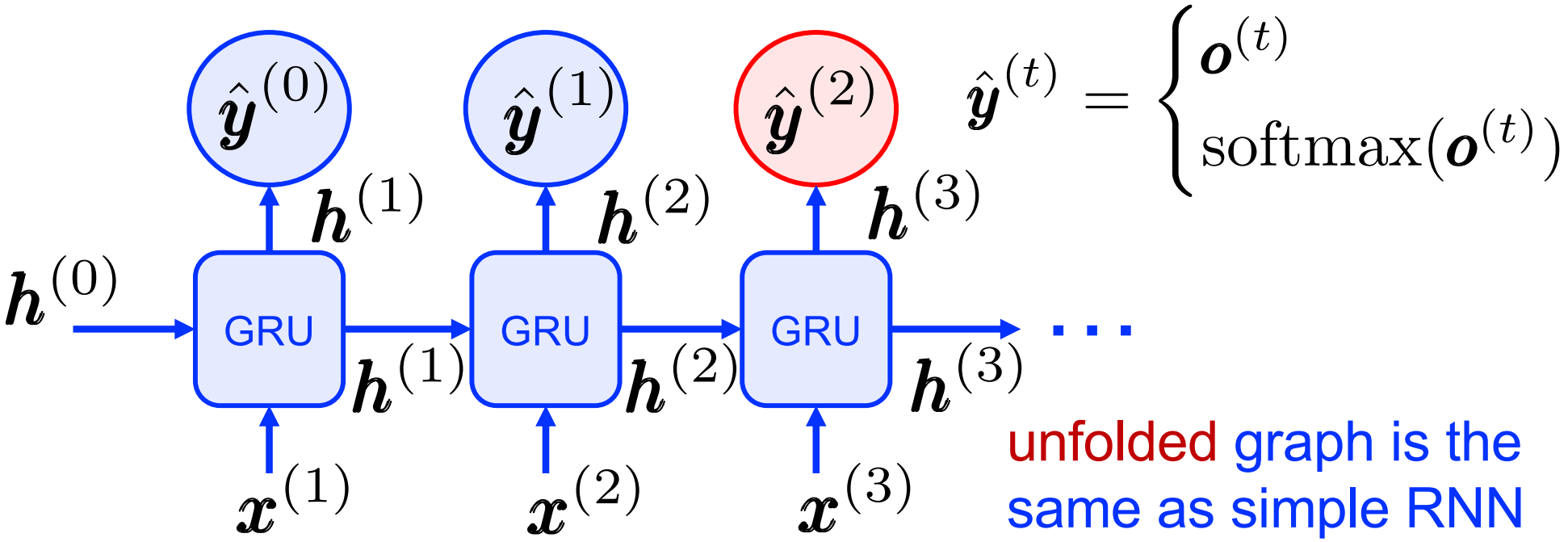
GRU cell "fully gated"

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

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

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

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



$$\hat{\boldsymbol{y}}^{(t)} = \begin{cases} \boldsymbol{o}^{(t)} \\ \text{softmax}(\boldsymbol{o}^{(t)}) \end{cases}$$

unfolded graph is the same as simple RNN

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

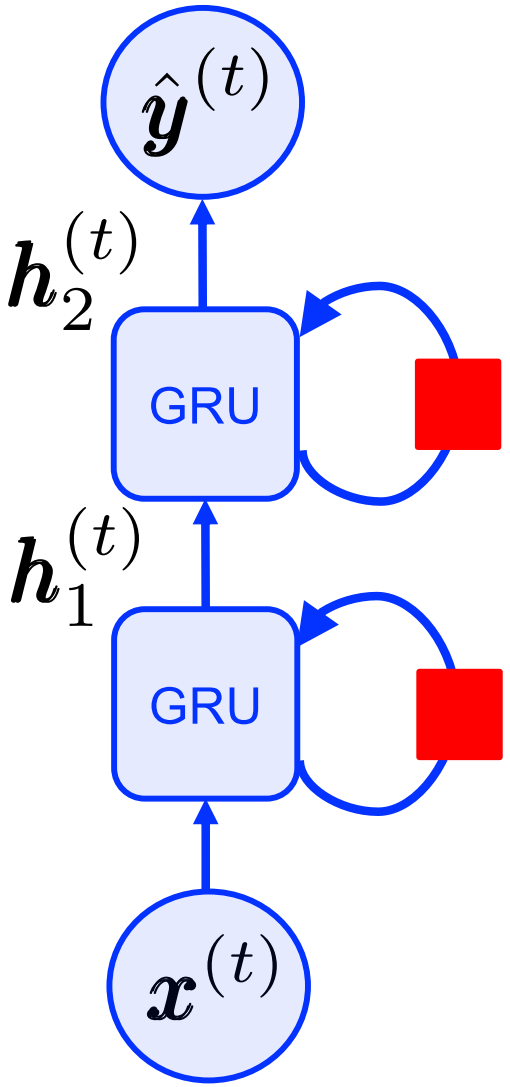
$\sigma_h(\cdot)$: tanh

a single gate is used to control

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

Deep RNN

- Obtained by “simply”
  - 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 putting two *independent* RNN together
- 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)

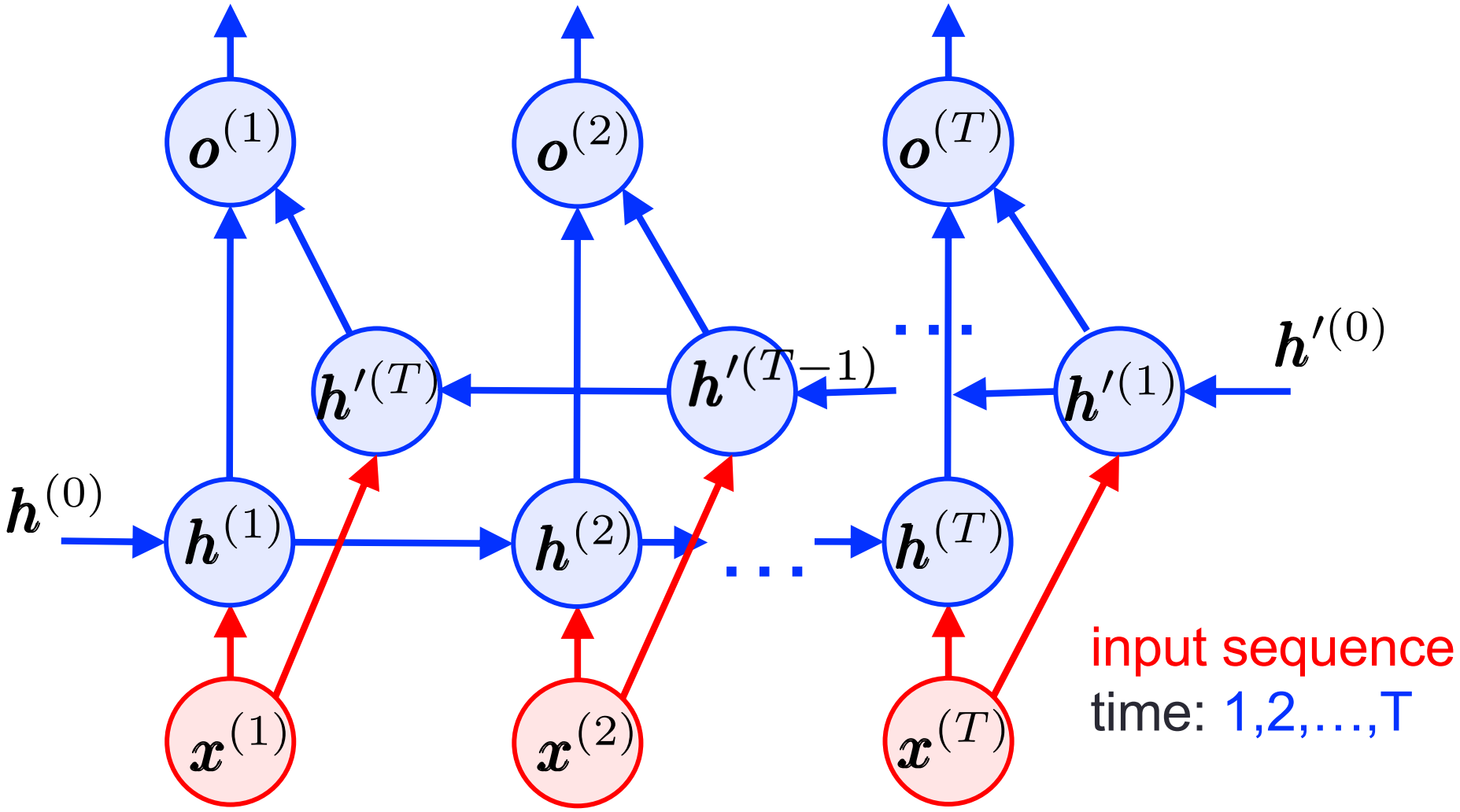
- Advantage
  - Connect present output to *past* and *future* samples
- Example from speech

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

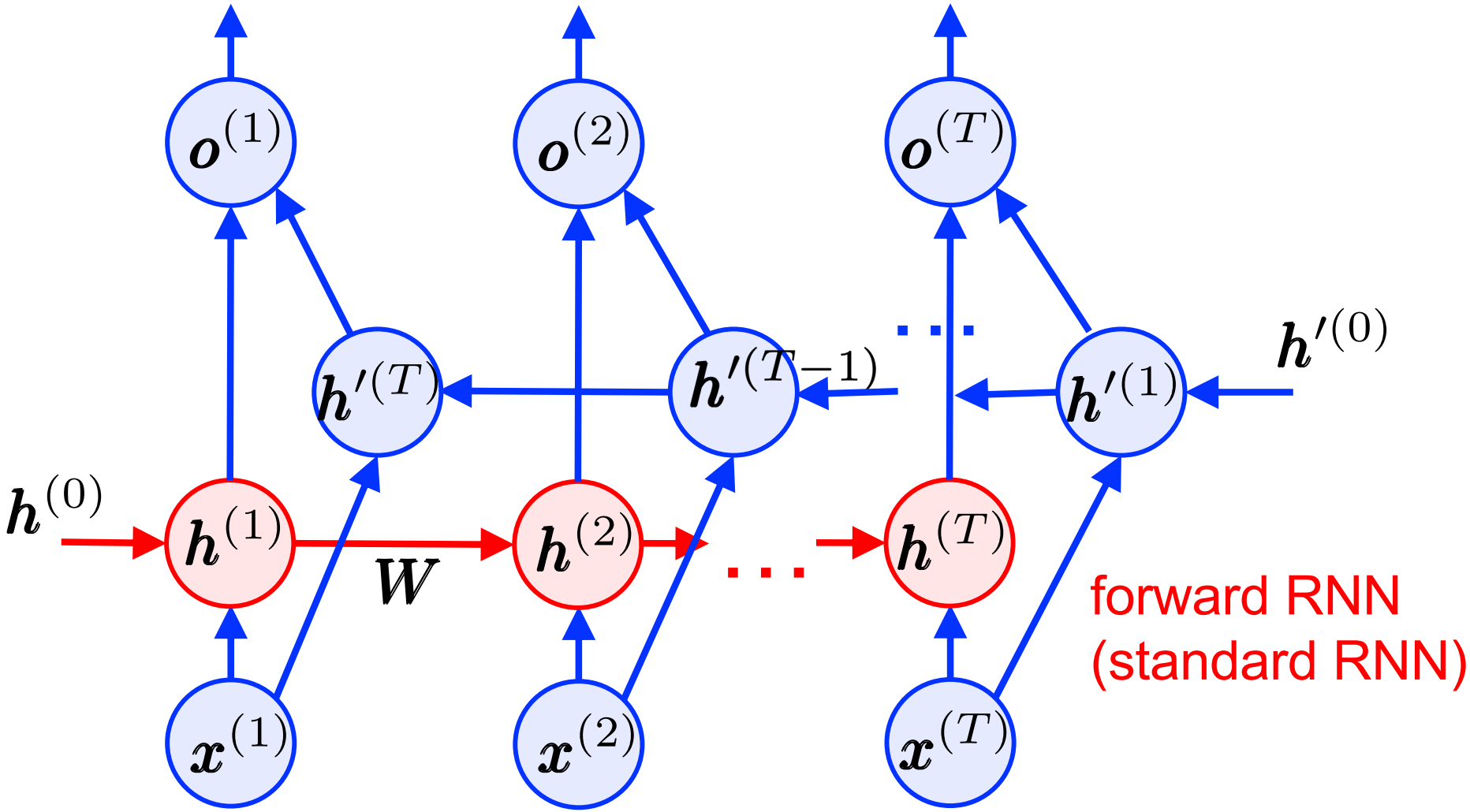
  - Once you have inputted "I am really angry": you are unable to say anything about the correctness of "angry"
  - But if you look forward, through the whole sequence "as I have been working out at the gym today"

    → you realize that *angry is wrong*

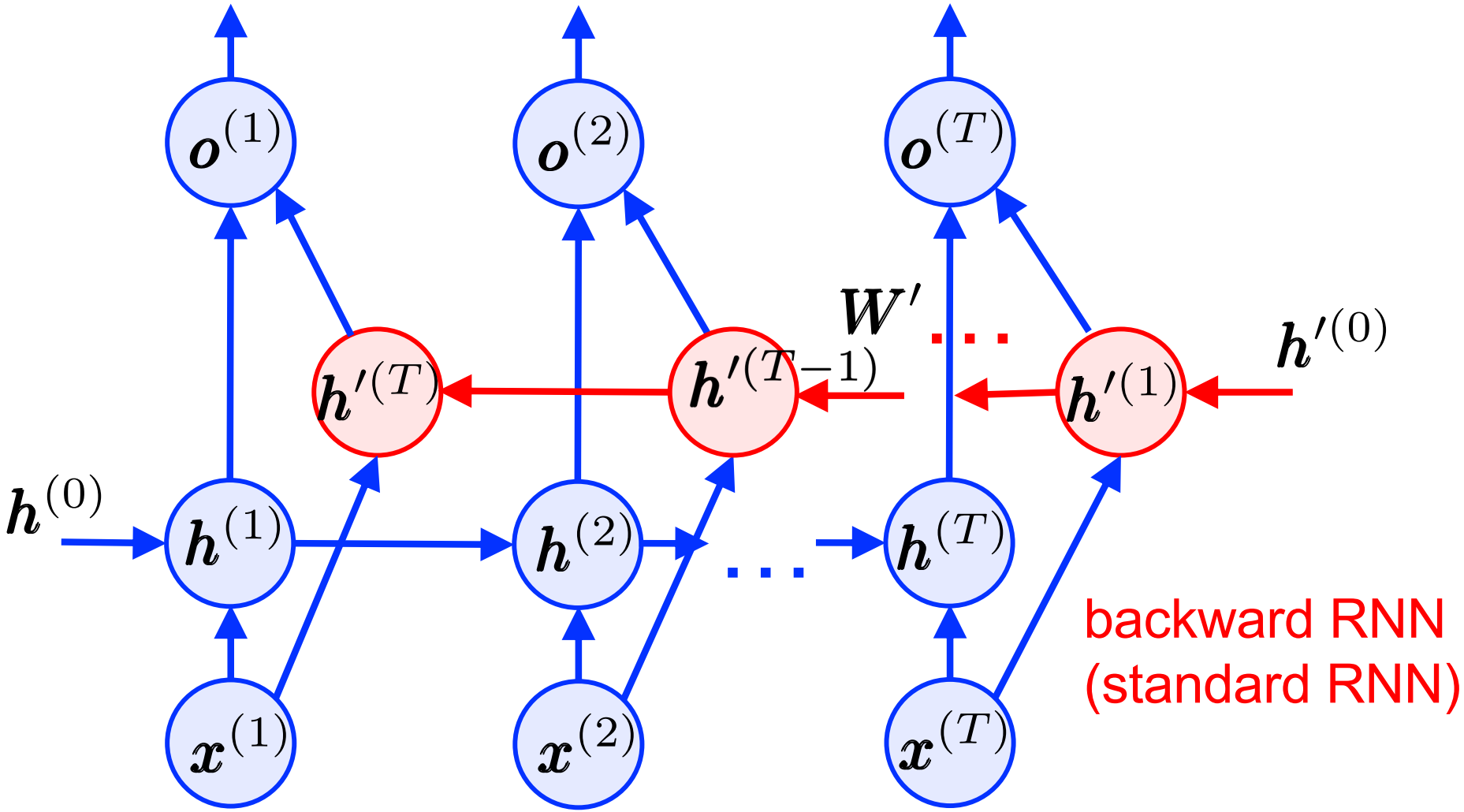
Bi-directional RNN (unfolded)



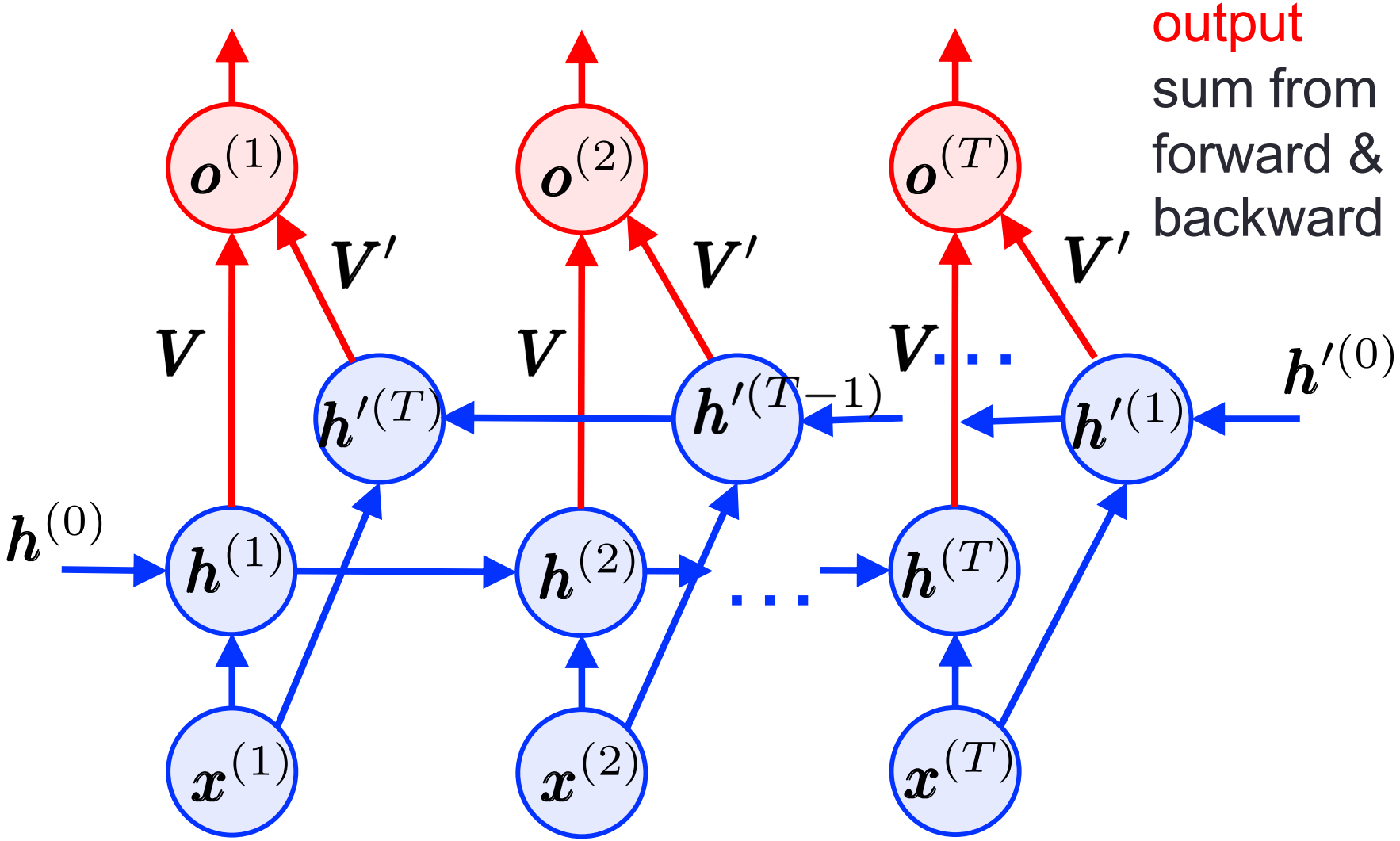
Bi-directional RNN (unfolded)



Bi-directional RNN (unfolded)



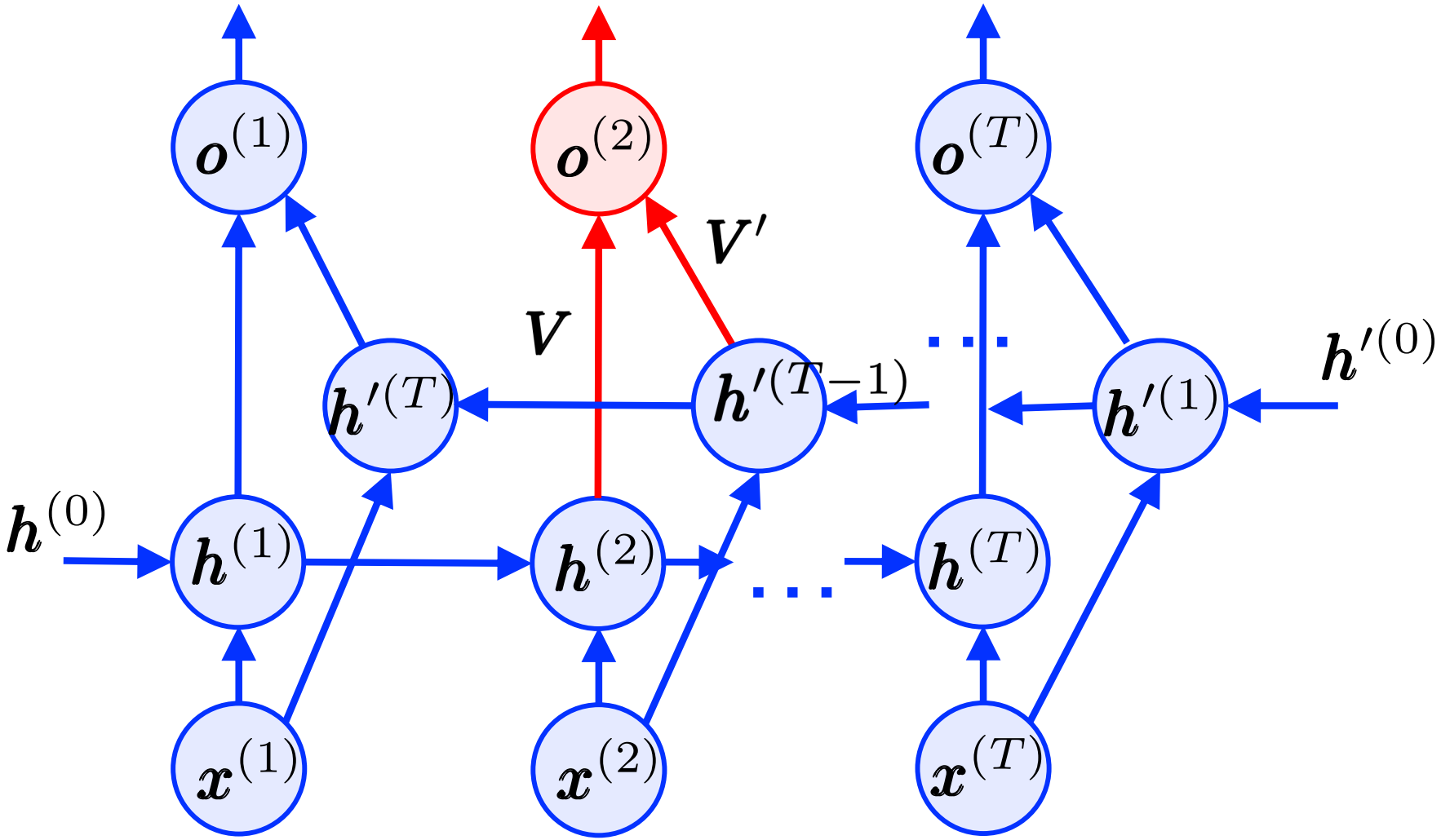
Bi-directional RNN (unfolded)



output at generic time i

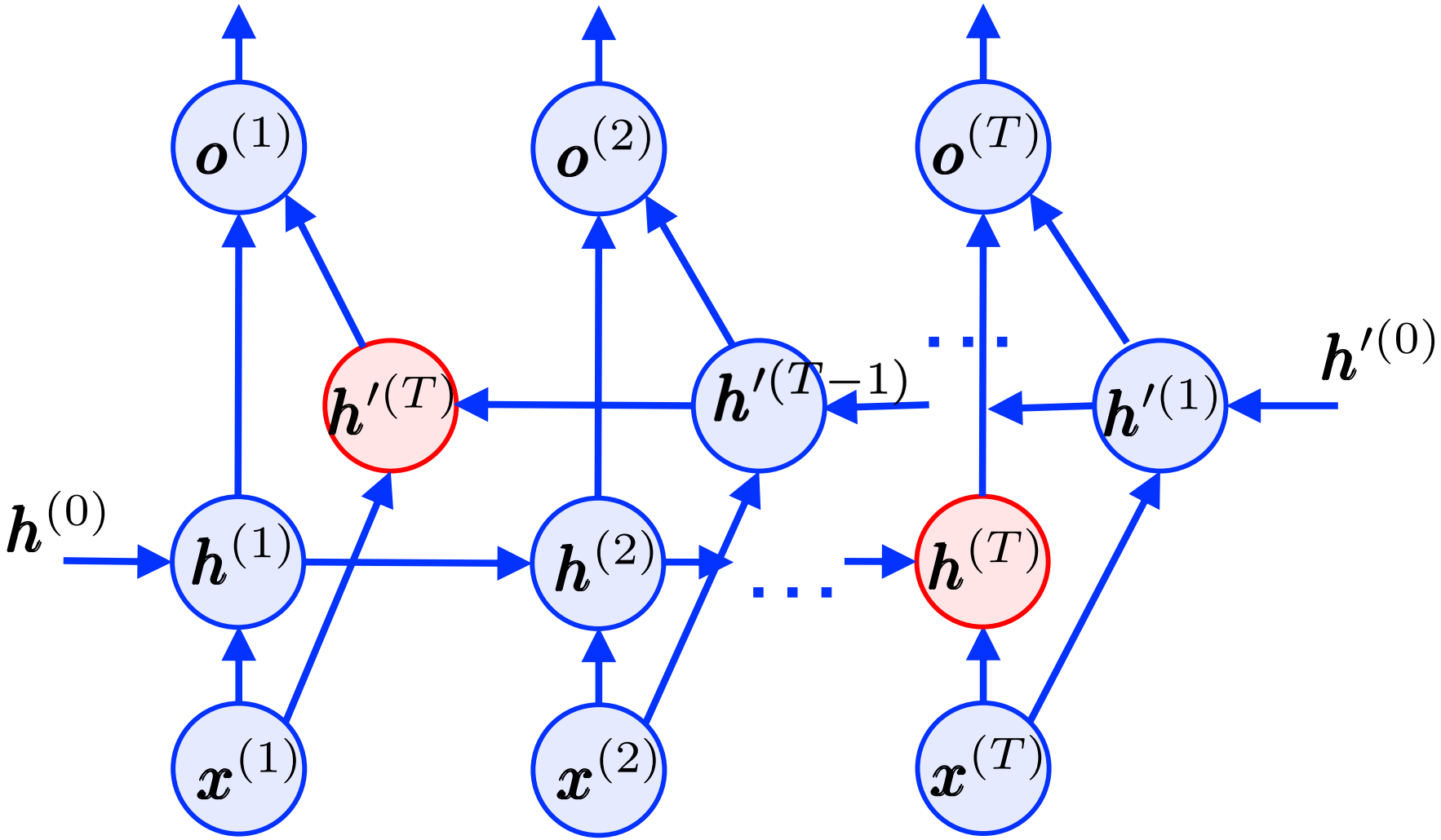
it is obtained by feeding the forward RNN with samples 1,2,…,i and the backward one with samples T,T-1,...,i

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



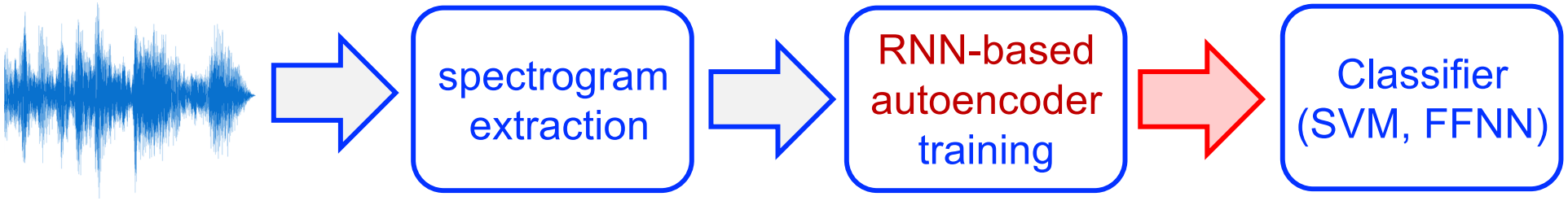
RNN final state

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



APPLICATION: UNSUPERVISED REPRESENTATION LEARNING OF AUDIO WAVEFORMS

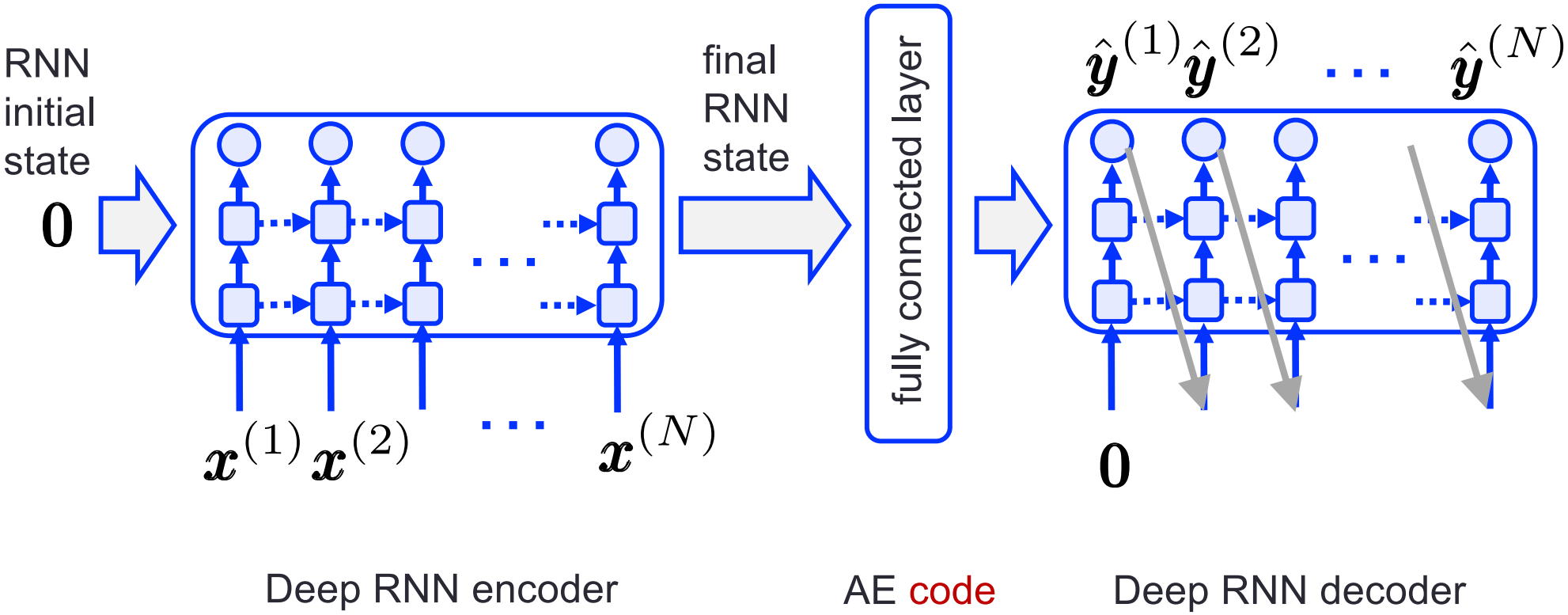
Unsupervised representation of audio



Signal processing pipeline from [Amiriparian17]

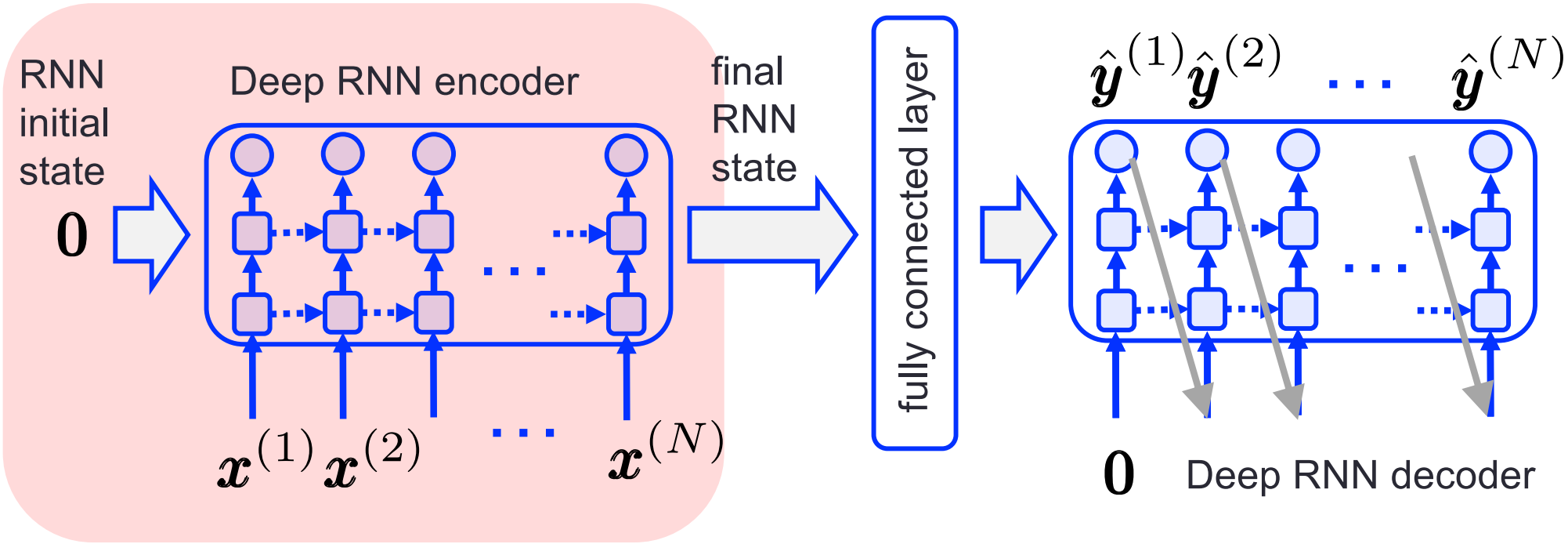
[Amiriparian17] S. Amiriparian, M. Freitag, N. Cummins, B. Schuller, “Sequence to Sequence Autoencoder for Unsupervised Representation Learning from Audio,” Detection and Classification of Acoustic Scenes and Events (DCASE), Munich, Germany, November 2017.

Deep RNN – AE (1/5)



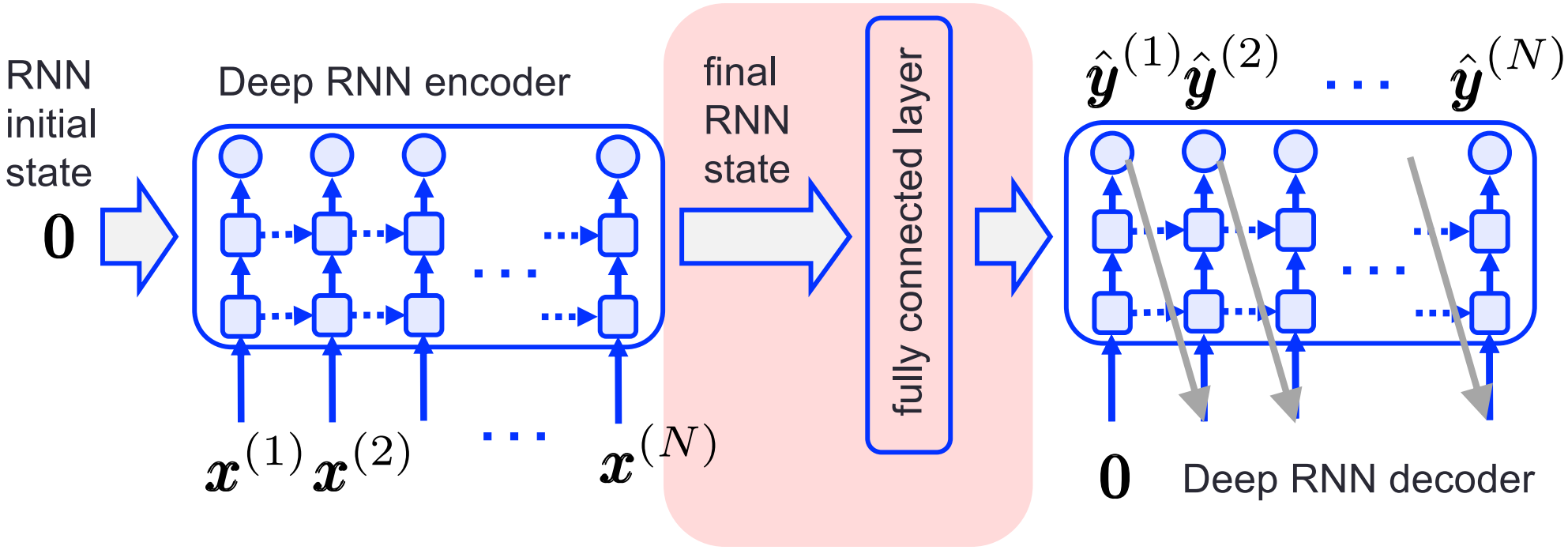
Training objective is: $\hat{\boldsymbol{y}}^{(i)} \simeq \boldsymbol{x}^{(i)}$, $i = 1, 2, \ldots, N$

Deep RNN – AE (2/5)



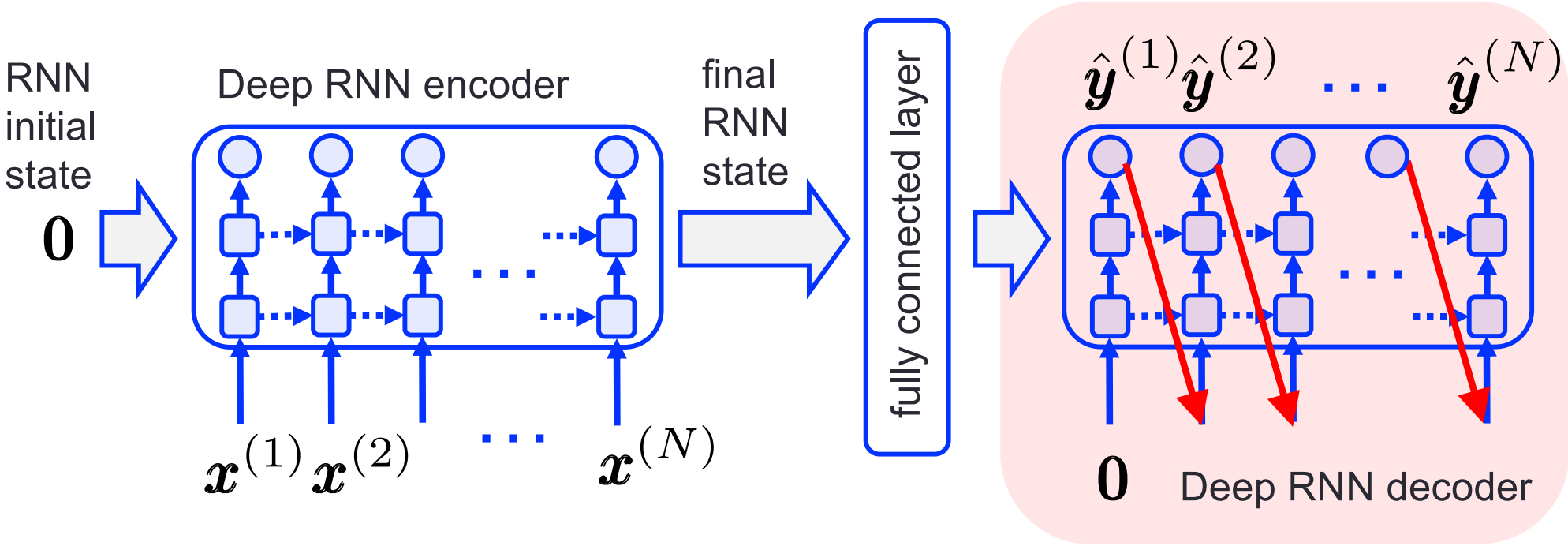
- RNN encoder
  - RNN state → zero vector
  - input vector sequence has N time steps n=1,2,…,N
  - input one feature vector at a time from time 1 to time N

Deep RNN – AE (3/5)



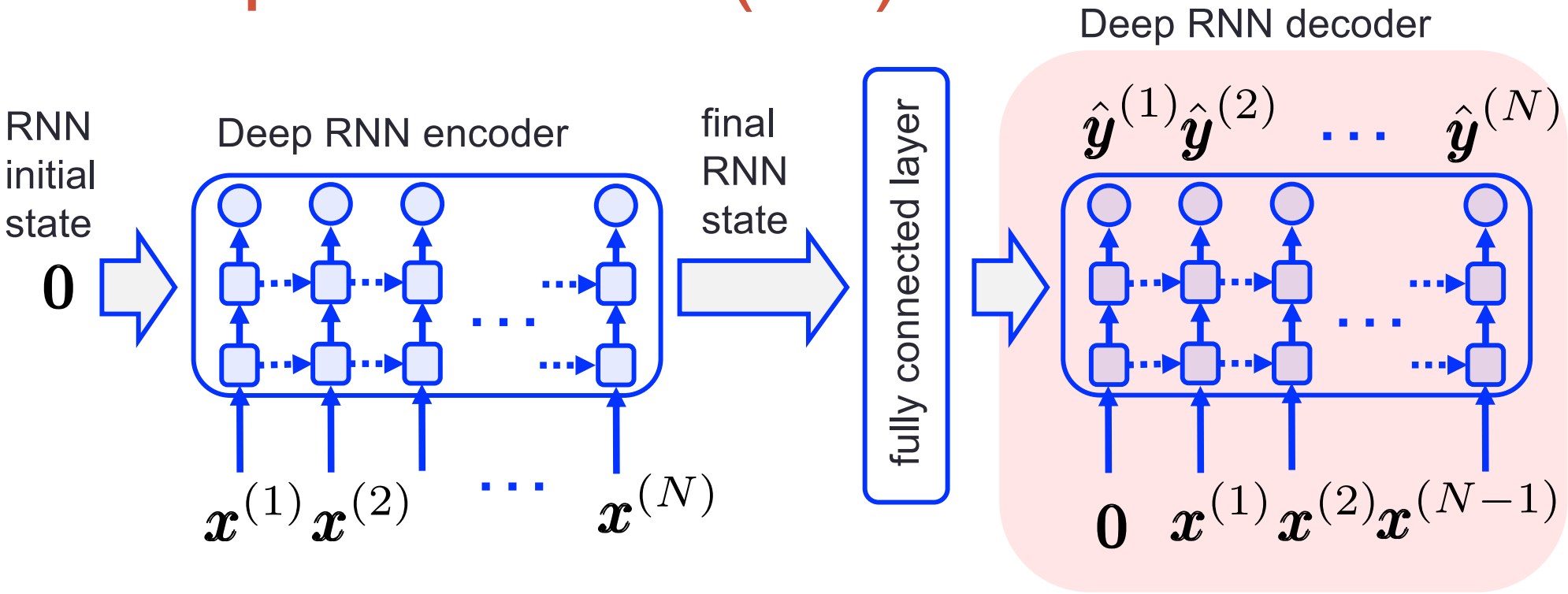
- After N time steps
  - final RNN state "flattened" into a vector (the "code")
  - code inputted into a fully connected layer
    - with tanh activation functions

Deep RNN – AE (4/5)



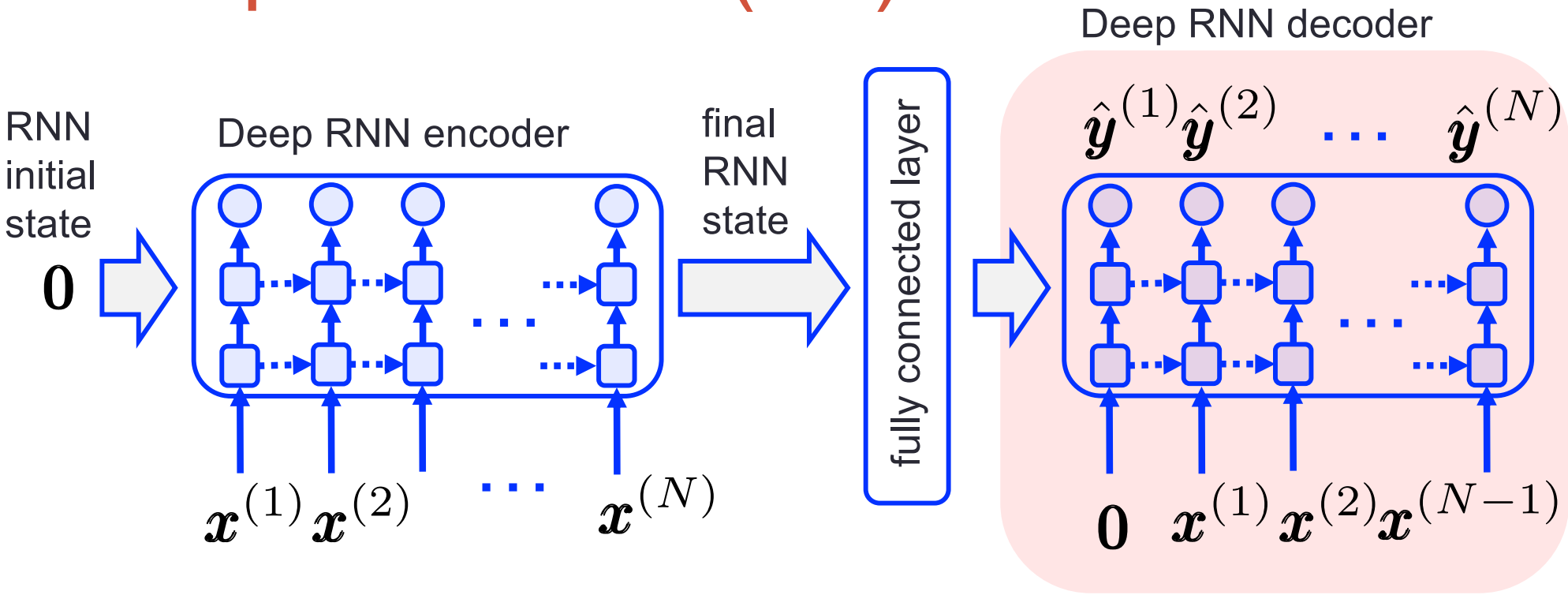
- RNN decoder
  - initial decoder state = AE code
  - decoder is a predictor of the input sequence
  - output from time t-1 used as input for time t
    - to predict output at time t

Deep RNN – AE (5/5)



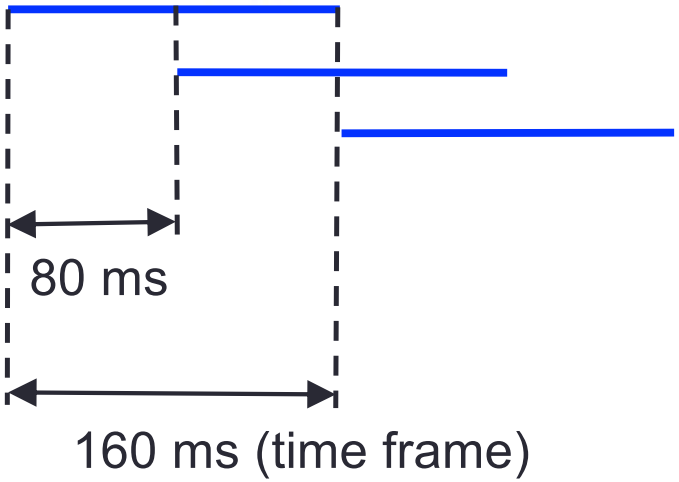
- RNN decoder
  - Trick no. 1: the *shifted* input sequence is used
    - in place of the one outputted by the decoder
    - In [Amiriparian17] it is proven that
      - this leads to faster training
      - performance degradation is negligible

Deep RNN – AE (5/5)



- RNN decoder
  - Trick no 2: it is a *bi-directional* RNN decoder

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 (MFCC)
  - First step in any speech recognition system
  - MFCC 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 unknown label

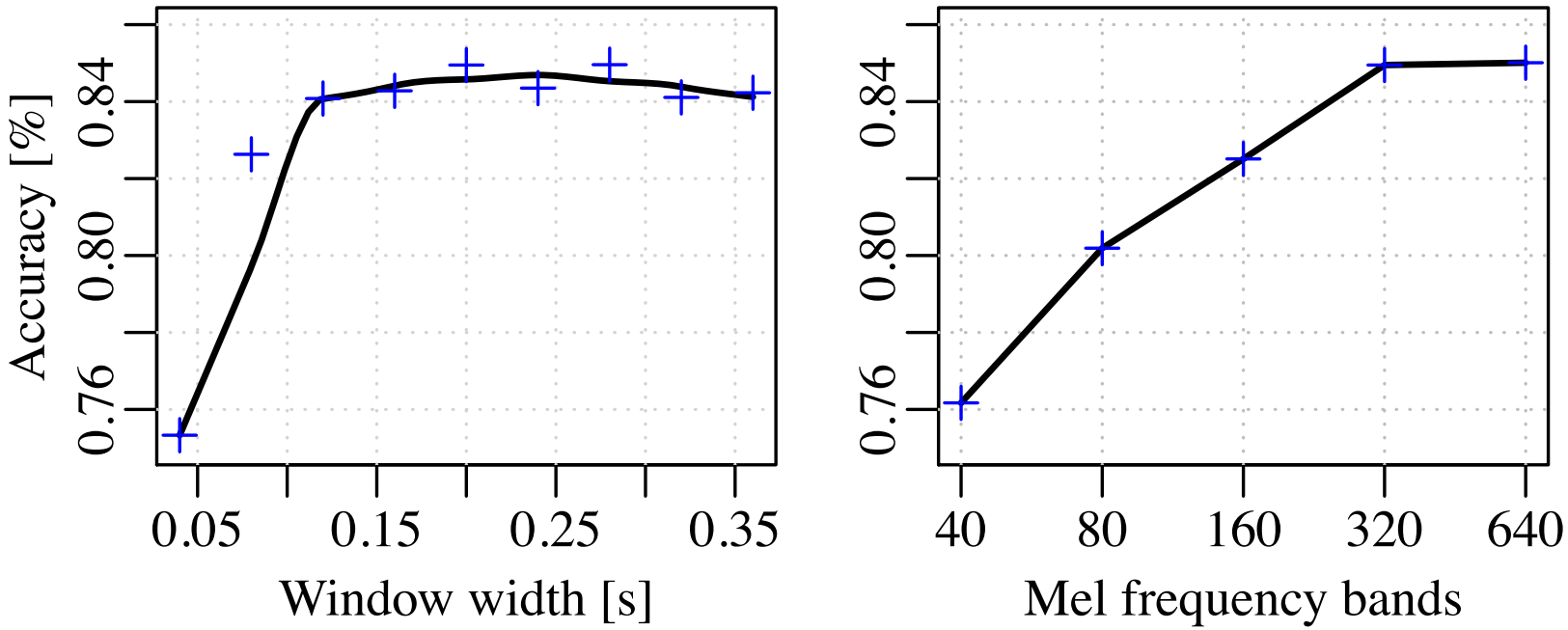
[Mesaros2017] A. Mesaros, T. Heittola, A. Diment, B. Elizalde, A. Shah, E. Vincent, B. Raj, and T. Virtanen, "DCASE 2017 Challenge Setup: Tasks, Datasets and Baseline System," in Detection and Classification of Acoustic Scenes and Events (DCASE2017), Munich, Germany, Nov 2017.

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

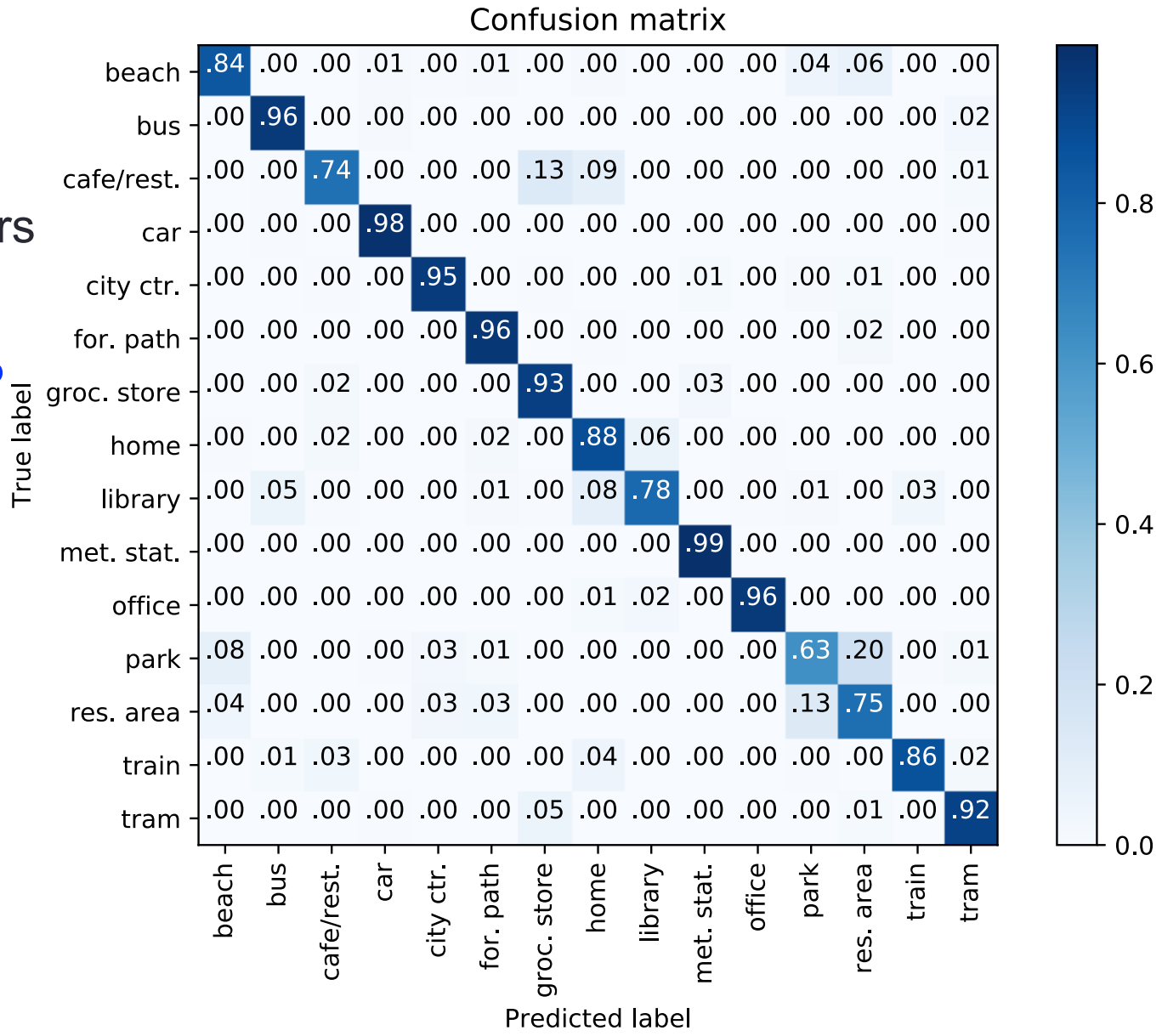


Grid search for preprocessing parameters

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

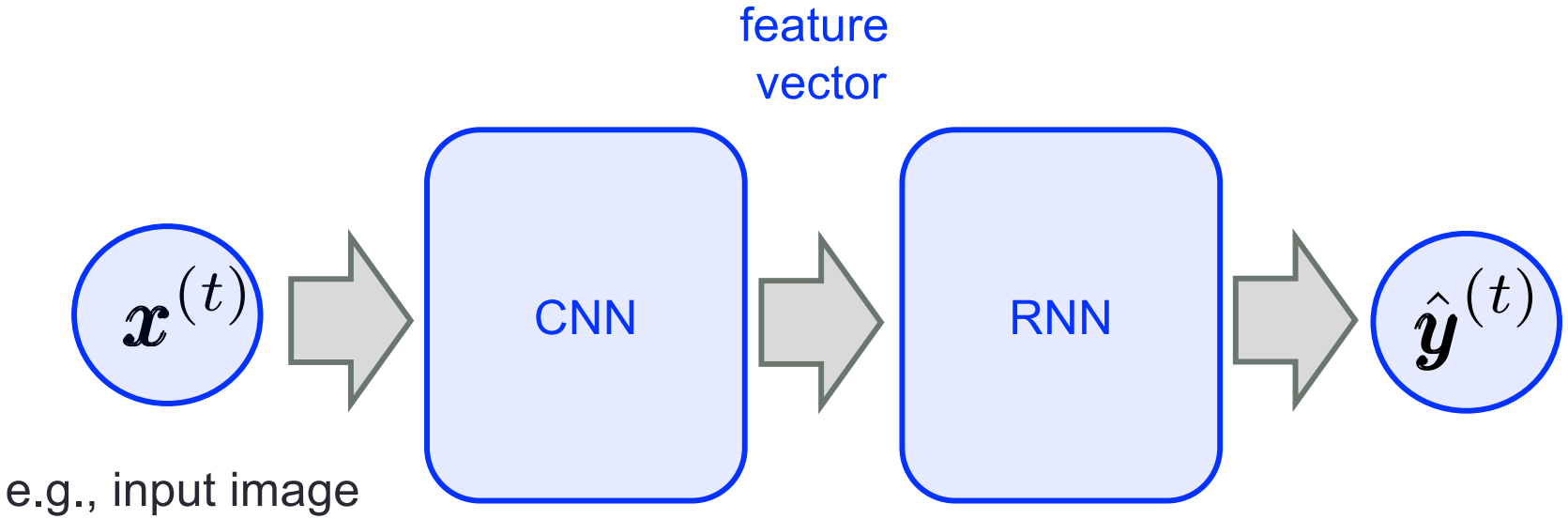
Confusion matrix

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



Combined CNN/RNN architectures

- Often
  - CNN used as feature extractor
  - Feature vector is inputted into a subsequent RNN



Bibliography (1/4)

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.

Bibliography (2/4)

Autoencoders:

[Hinton06] G. E. Hinton and R. R. Salakhutdinov, “Reducing the Dimensionality of Data with Neural Networks,” Science, July 2006.

[Vincent04] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, P.A. Manzagol, “Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion,” Journal of Machine Learning Research, 2010.

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.

Bibliography (3/4)

Seq-2-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 (4/4)

ECG:

[DelTesta2015] Davide Del Testa, Michele Rossi, “Lightweight Lossy Compression of Biometric Patterns via Denoising Autoencoders,” IEEE Signal Processing Letters, 2015.

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

Thank you !!!

thank you
danke
謝謝
ngiyabonga
teşekkür ederim
gracias
tapadh leat
merci
go raibh maith agat
mochchakkeram
obrigado
bedankt
dziękuje
sukriya
kop khun krap
terima kasih
감사합니다
спасибо
Баярлалаа
рахмат
mersi
kia ora
barka
welalin
tack
spas
dank je
misaotra
matondo
paldies
grazzi
mahalo
хвала
asante
manana
obrigada
tenki
murakoze
chokrane
faafetai lava
vinaka
спасибі
blagodaram
kiitos
dankie
nanni
nandri
dhanyavad
maururu
köszönöm
hvala
gràcje
bayarlalaa
enkosi
dankon
akun
ačiū
djiere dieuf
tau
дякую
mamnun
chnorakaloutioun
gratias ago
gràcies
sulpáy
taiku
sobodi
děkuji
sagolun
arigatô
takk
dakujem
trugarez
ありがとう
tanemirt
grazie
rahmet
mèsi
didi madloba
najis tuke
kam sah hamnida
rahmat
diolch
dhanyavadagalu
shukriya
mercê
мерси
xièxie
ευχαριστώ

LEARNING FOR SEQUENTIAL DATA: TOOLS AND APPLICATIONS

Michele Rossi

rossi@dei.unipd.it

Dept. of Information Engineering

University of Padova, IT