LEARNING FOR SEQUENTIAL DATA: TOOLS AND APPLICATIONS

Michele Rossi
rossi@dei.unipd.it

Dept. of Information Engineering
University of Padova, IT

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

Outline

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

UNSUPERVISED LEARNING AS VECTOR QUANTIZATION

Setup

- Input space, vectors of m elements

$$\boldsymbol{x} = [x_1, x_2, \ldots, x_m]^T$$

- i.i.d. sequentially sampled, one at a time,
  - from the same (unknown) pdf $f(\boldsymbol{x})$
  - time is discrete n = 1, 2, …

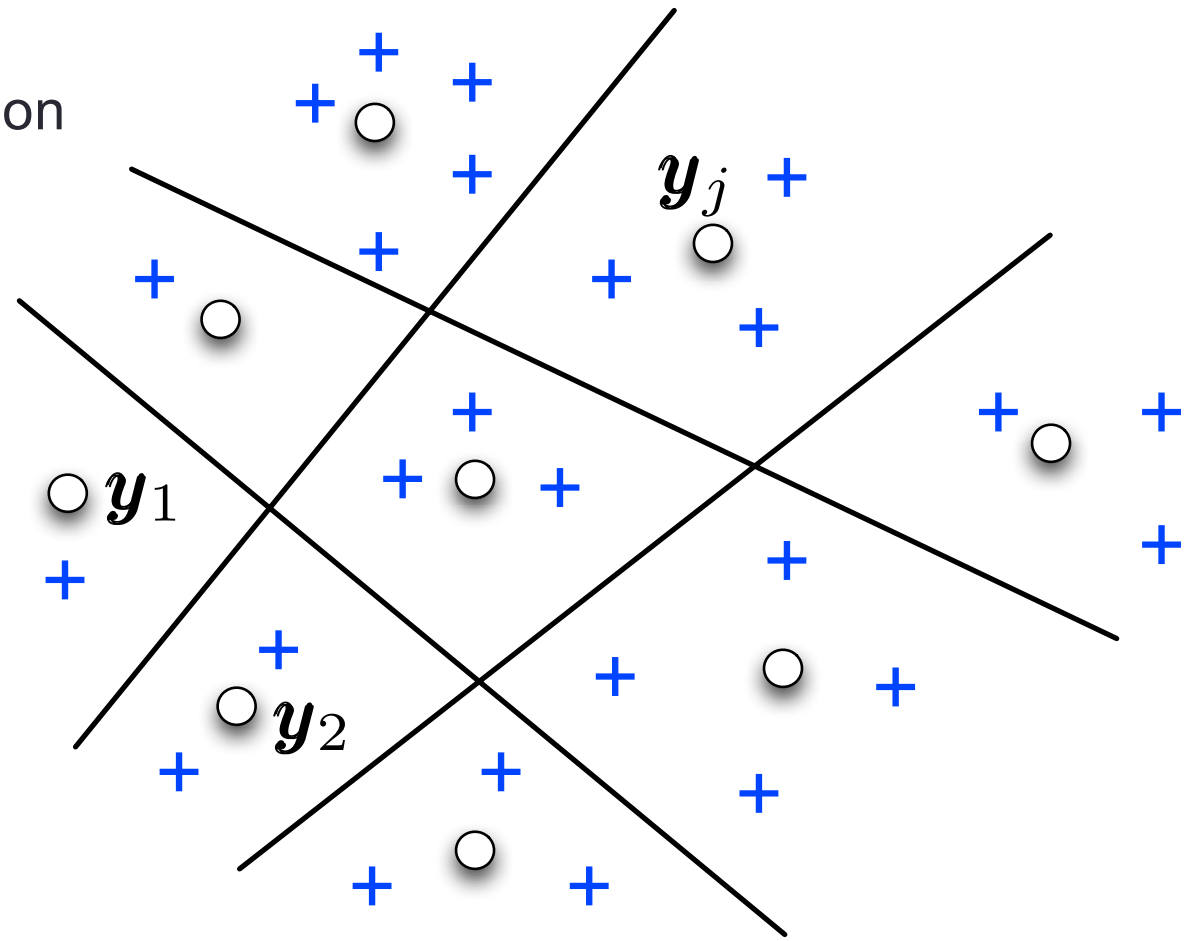
- A set of centroids is defined (e.g., cluster centers)

$$\{\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_\ell\}$$

- with: $\boldsymbol{y}_j \in \mathbb{R}^m$

Vector Quantization (1/2)

- Given an input data distribution (multi-dimensional vectors)
- We need to find:
  - A partition rule (regions)
  - A prototype for each region
    - Also called centroid



Prototype (codeword)

Input data vector

Vector Quantization (2/2)

- Distortion measure
  - Most common measure: Euclidean distance
  - Average distortion is quantified as the mean square error

$$E[d(\boldsymbol{x}, \boldsymbol{y}_j)] = \sum_{j=1}^{\ell} \int_{I_j} \|\boldsymbol{a} - \boldsymbol{y}_j\| f(\boldsymbol{a}) d\boldsymbol{a} \quad (1)$$

Region j

Centroid

$\|\boldsymbol{x}\| \rightarrow \text{norm-2}$

Input (signal) pdf

- Optimal VQ
  - Find a set of centroids and a partition rule that minimize (1)
  - Nearest neighbor condition: the optimal partition is the one returning the minimum distortion (region j)

$$I_j = \{\boldsymbol{x} : d(\boldsymbol{x}, \boldsymbol{y}_j) \leq d(\boldsymbol{x}, \boldsymbol{y}_h), j \neq h\}$$

Available tools

- A number of *online* tools exist
  - Online means that the full dataset does not have to be stored
  - The solutions adapts, one sample at a time
  - New samples are collected sequentially (i.i.d.) from the same pdf
- Most remarkable tools are
  - Self-Organizing Maps (SOM) [Kohonen82]
  - Growing Neural Gas (GNG) networks [Fritzke94]
  - Both are neural networks, whose weights
    - represent the (learned) prototypes
    - are progressively refined as new samples are acquired

[Kohonen82] Teuvo Kohonen, “Self-organized formation of topologically correct feature maps,” Biological Cybernetics, vol. 43, pp. 59–69, 1982.

[Fritzke94] Bernd Fritzke, “A Growing Neural Gas Network Learns Topologies,” Proceedings of the 7th International Conference on Neural Information Processing Systems (NIPS), 1994.

SOM & the human brain

- SOMs are motivated by features of the human brain
  - "The brain is organized in many places in a way that different sensory inputs are represented by topologically ordered computation maps"
- Key fact: sensory inputs such as
  - Tactile (Kaas, J.H., M.M. Merzenich, and H.P. Killackey, 1983. "The reorganization of somatosensory cortex following peripheral nerve damage in adult and developing mammals," *Annual Review of Neurosciences,* vol. 6, pp. 325–356)
  - Visual (Hubel, D.H., and T.N. Wiesel, 1962. "Receptive fields, binocular interaction and functional architecture in the cat's visual cortex," *Journal of Physiology,* vol. 160, pp. 106–154, London)
  - Acoustic (Suga, N., 1985. "The extent to which bisonar information is represented in the bat auditory cortex," in *Dynamic Aspects of Neocortical Function,* G.M. Edelman, W.E. Gall, and W.M. Cowan, eds. pp. 653–695, New York: Wiley)
- are mapped onto several areas of the cerebral cortex in a topologically ordered manner

Input vectors & synaptic weights

- m is the dimension of the input data space
- A vector sampled at random from the input space:

$$\boldsymbol{x} = [x_1, x_2, \ldots, x_m]^T$$

- The synaptic-weight vector of each neuron in the SOM has the same dimension of the input space. The weight vector associated with neuron j is:

$$\boldsymbol{w}_j = [w_{j1}, w_{j2}, \ldots, w_{jm}]^T, \; j = 1, 2, \ldots, \ell$$

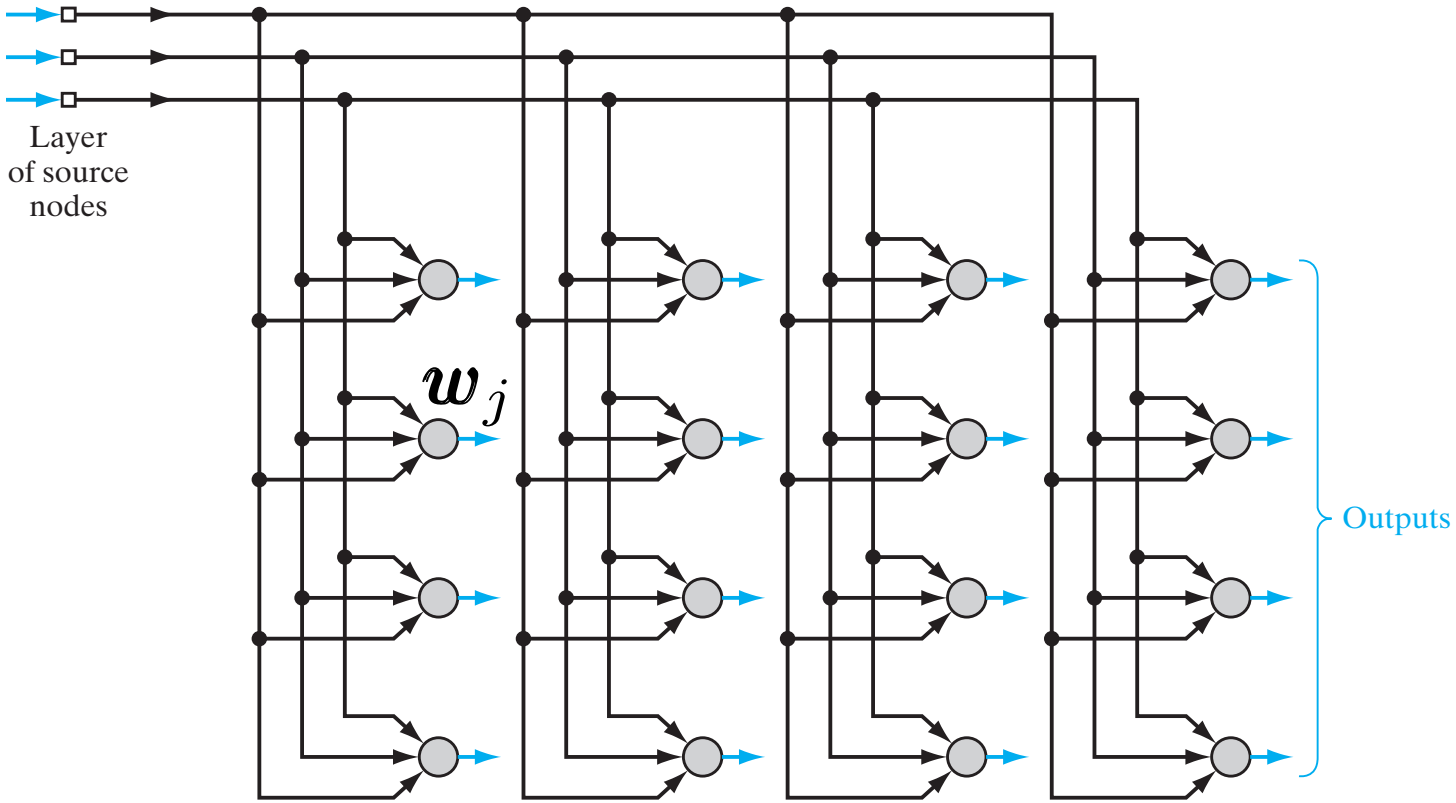
- ℓ represents the number of (output) neurons in the SOM

A 2D SOM example

2D lattice of $\ell = 4 \times 4 = 16$ neurons

Input patters are vectors with three elements

$$\boldsymbol{x} = [x_1, x_2, x_3]^T$$



1. Competition

- Let **x** be a vector from the input space

$$\boldsymbol{x} = [x_1, x_2, \ldots, x_m]^T$$

- All the neurons in the SOM lattice compete
  - The best fit neuron will win the competition
  - Let i(**x**) be the index associated with the winning neuron
  - This index i(**x**) is computed as:

$$i(\boldsymbol{x}) = \arg\min_j \|\boldsymbol{x} - \boldsymbol{w}_j\|, \; j \in \mathcal{L}$$

  - where $\mathcal{L}$ indicates the SOM lattice
  - and $\|\boldsymbol{x} - \boldsymbol{w}_j\|$ is the Euclidean distance
  - Observation: a *continuous* input space of activation patterns is mapped onto a *discrete output space* (lattice) of neurons by a process of competition among the neurons in the neural network

2. Cooperation

- The winning neuron locates the center of
  - a *topological neighborhood of cooperating neurons*
- The question now is
  - *how do we define a neighborhood that is neuro-biologically correct?*
- Answer: there is neurobiological evidence that
  - Lateral interaction occurs among excited neurons in the human brain
  - Meaning: a neuron that fires tends to *excite more the neurons that are located in its immediate neighborhood* rather than those that are located away from it

- $h_{i,j}(n)$ defines a topological neighborhood around the winning neuron i(**x**) and make it decay smoothly with lateral distance

3. Adaptation

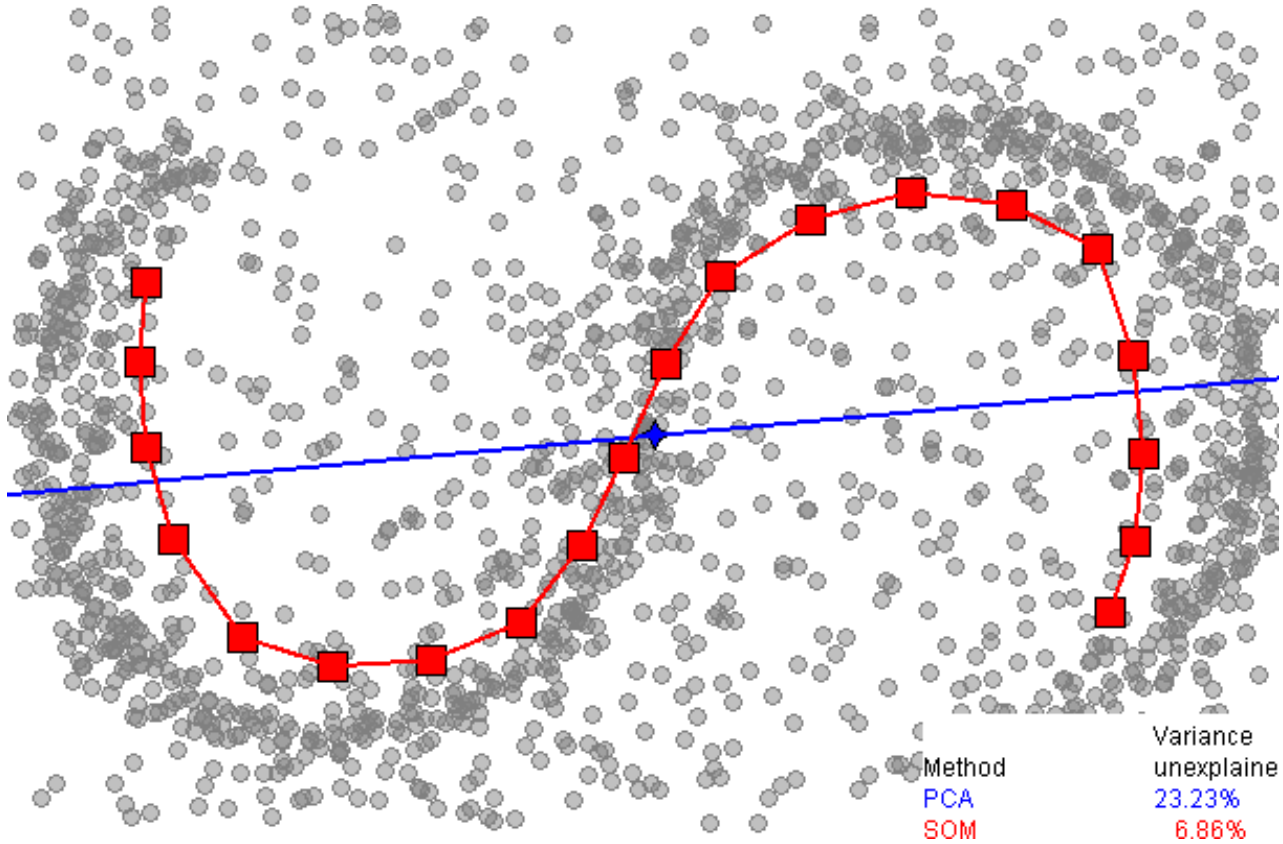
- Using discrete-time formalism
- The update equation for the weight vectors is:

$$\boldsymbol{w}_j(n+1) = \boldsymbol{w}_j(n) + \overbrace{\boxed{\eta(n) h_{i,j}(n)(\boldsymbol{x} - \boldsymbol{w}_j)}}^{\Delta \boldsymbol{w}}, \; j \in \mathcal{L}$$

- $h_{i,j}(n)$: controls the size of the neighborhood *vs* space & time
- $\eta(n)$: is a learning rate parameter
- This update equation is of a stochastic approximation type
  - The new weight vector **w**(n+1) is equal to the old one **w**(n) plus a Δ**w** (update) term, which depends on the distance between (i) the current input vector **x** and (ii) the weight vector $\mathbf{w}_j$
  - The intensity of the update depends on (i) how much **x** differs from $\mathbf{w}_j$, (ii) the learning rate parameter and (iii) the location of neuron j wrt to the winning neuron i(**x**) (note that j and i can also coincide)

1D SOM *vs* 1D PCA

- Data points rest on a non-linear manifold
- PCA performs poorly
- SOM does a better job



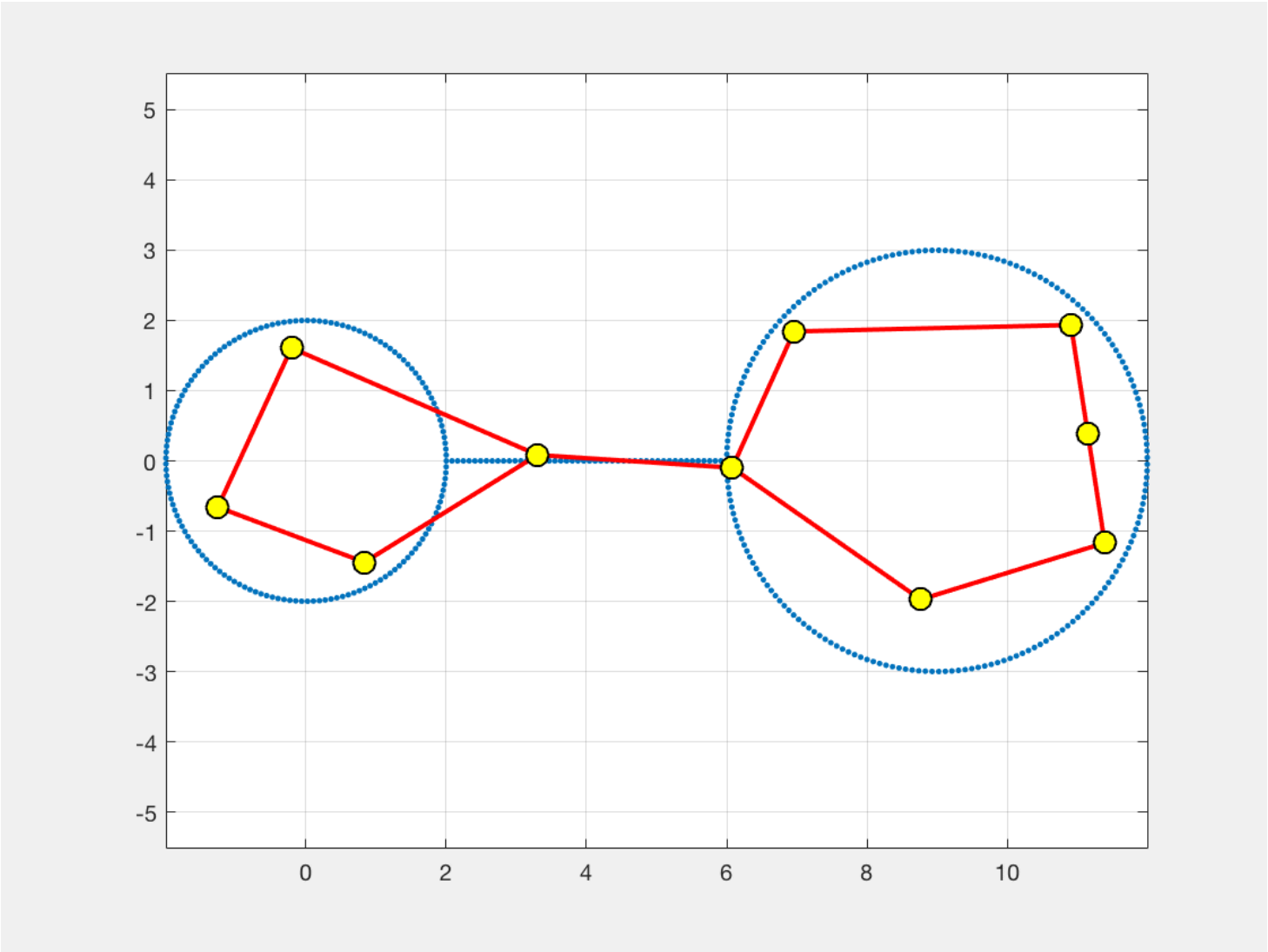
One-dimensional SOM versus principal component analysis (PCA) for data approximation. SOM is a red broken line with squares, 20 nodes. PCA is presented by a blue line. Data points are the small grey circles. For PCA, the fraction of variance unexplained in this example is 23.23%, for SOM it is 6.86%

GNG

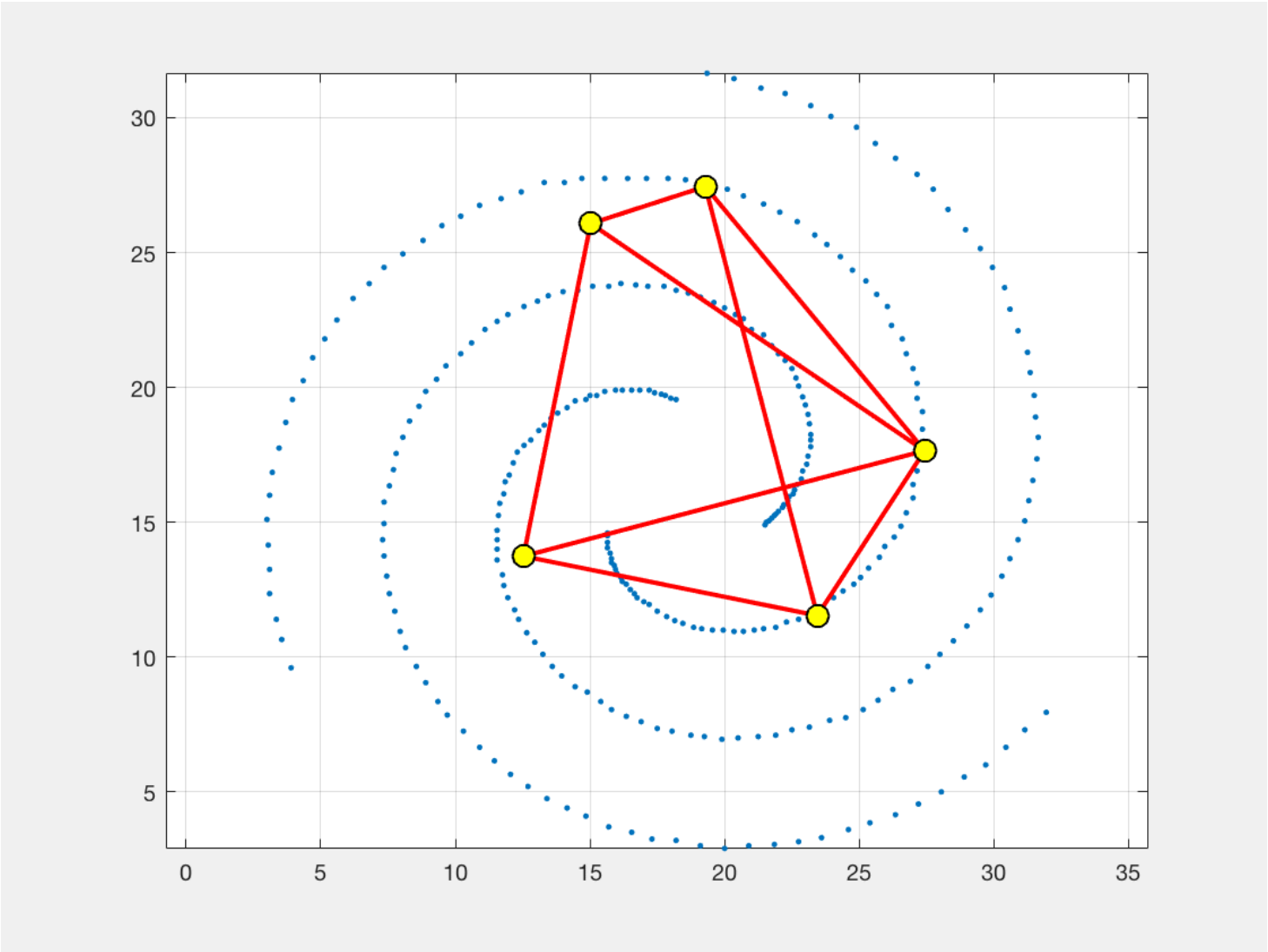
- Are more involved
  - Uses Hebbian learning's principle
    - *"Neurons that fire together wire together"* [Hebb49]
  - Competition
    - Is among synaptic links, rather than among nodes (neurons)
- Addition and removal of neurons (prototypes)
  - Is automated, as part of the algorithm
  - Depends on the maximum tolerable error (which is tunable)
- Continuous learning
  - Since neurons can be added and removed
  - The structure can be continuously refined (up to some error tolerance)
  - To some (good) extent robust to non stationary environments (pdfs)

[Hebb49] D.O. Hebb: The Organization of Behavior, Wiley, New York, 1949.

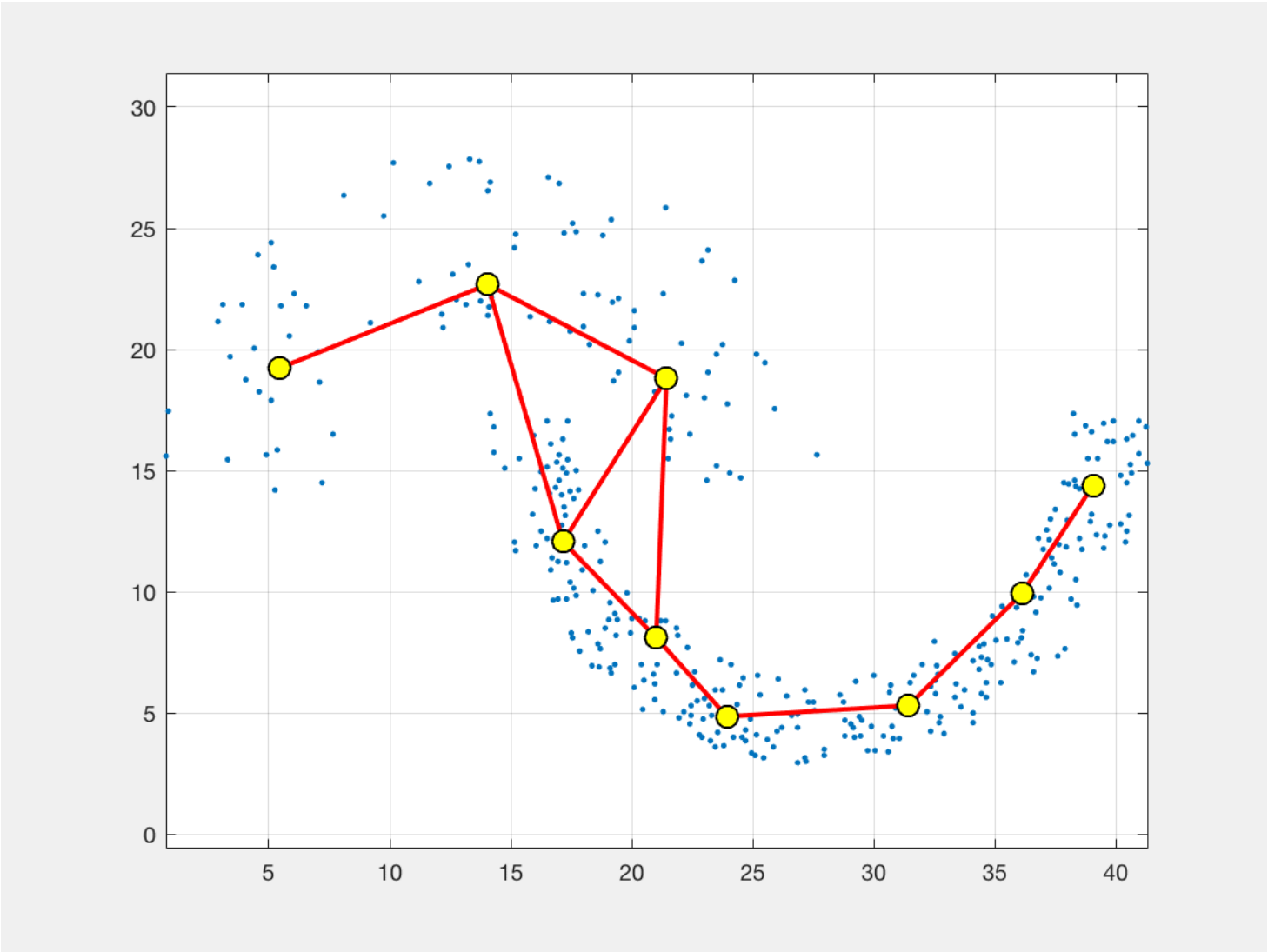
GNG, pdf domain → circles



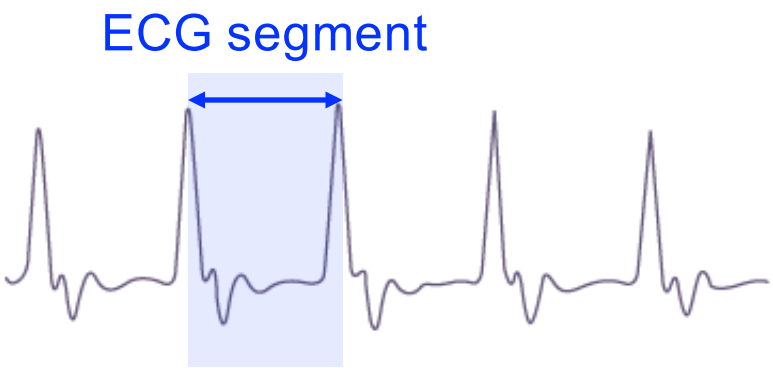
GNG, pdf domain → spiral



GNG, generic pdf support



Applications

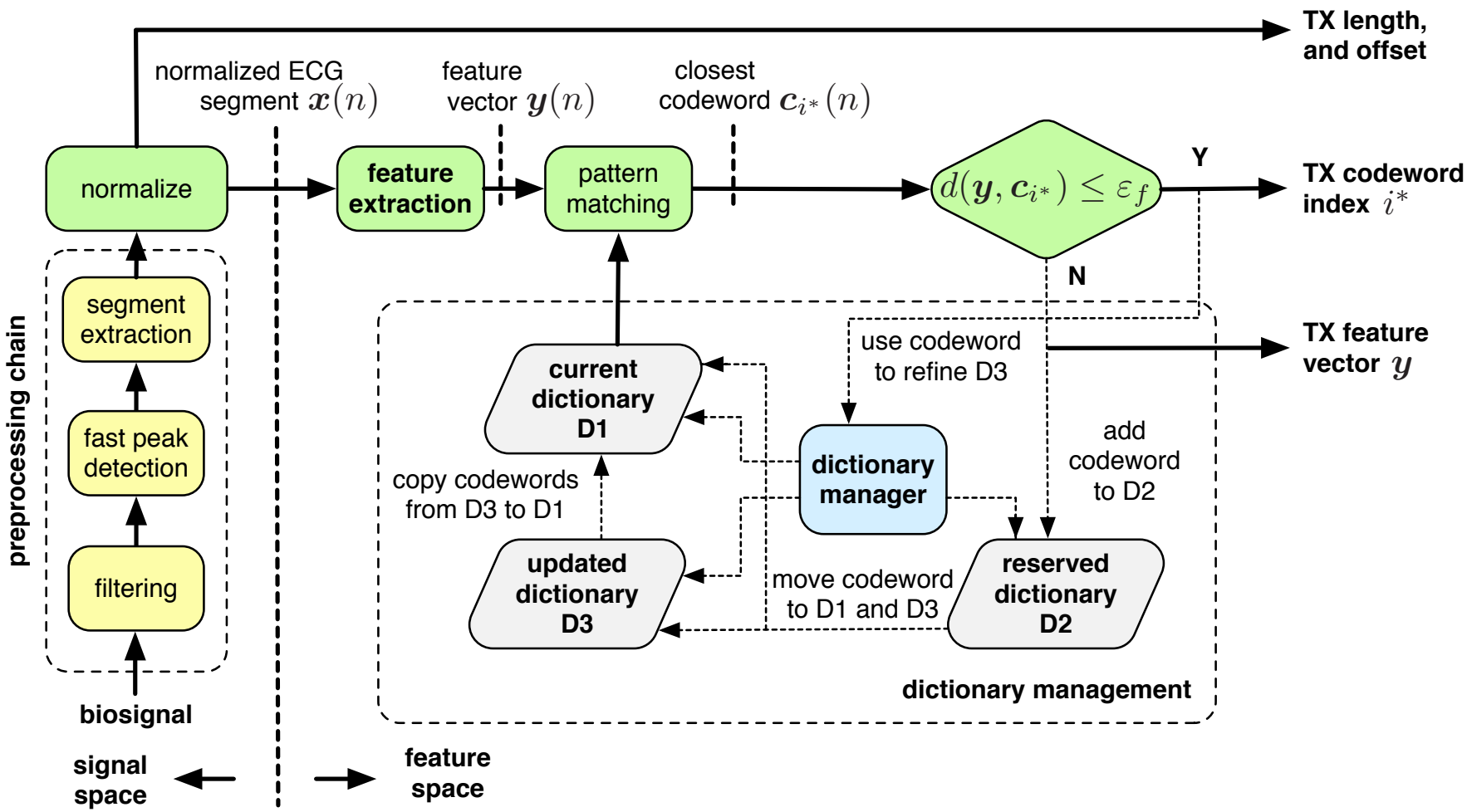


- Many applications exist

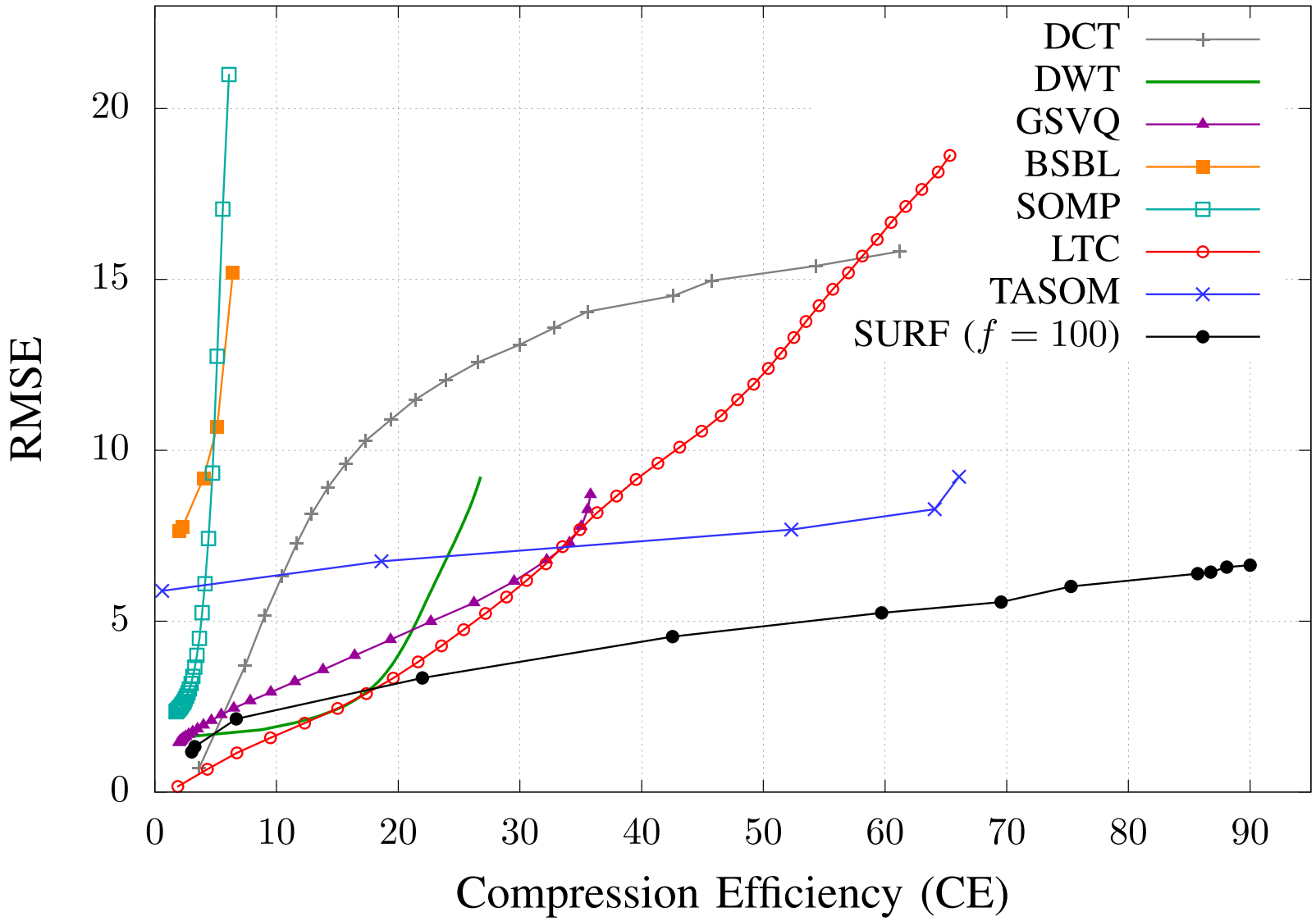
- From my research: ECG signals [Hooshmand17]
  - Capture signal patterns
  - Represented through a number of codewords (dictionary)
  - Allows high compression factors (up to 90 times)

[Hooshmand17] M. Hooshmand, D. Zordan, T. Melodia, M. Rossi, ”SURF: subject-adaptive unsupervised ECG signal compression for wearable fitness monitors,” *IEEE Access*, Vol. 5, pp. 19517 - 19535, 7 September 2017.

Applications: ECG signal compression



Applications: ECG signal compression

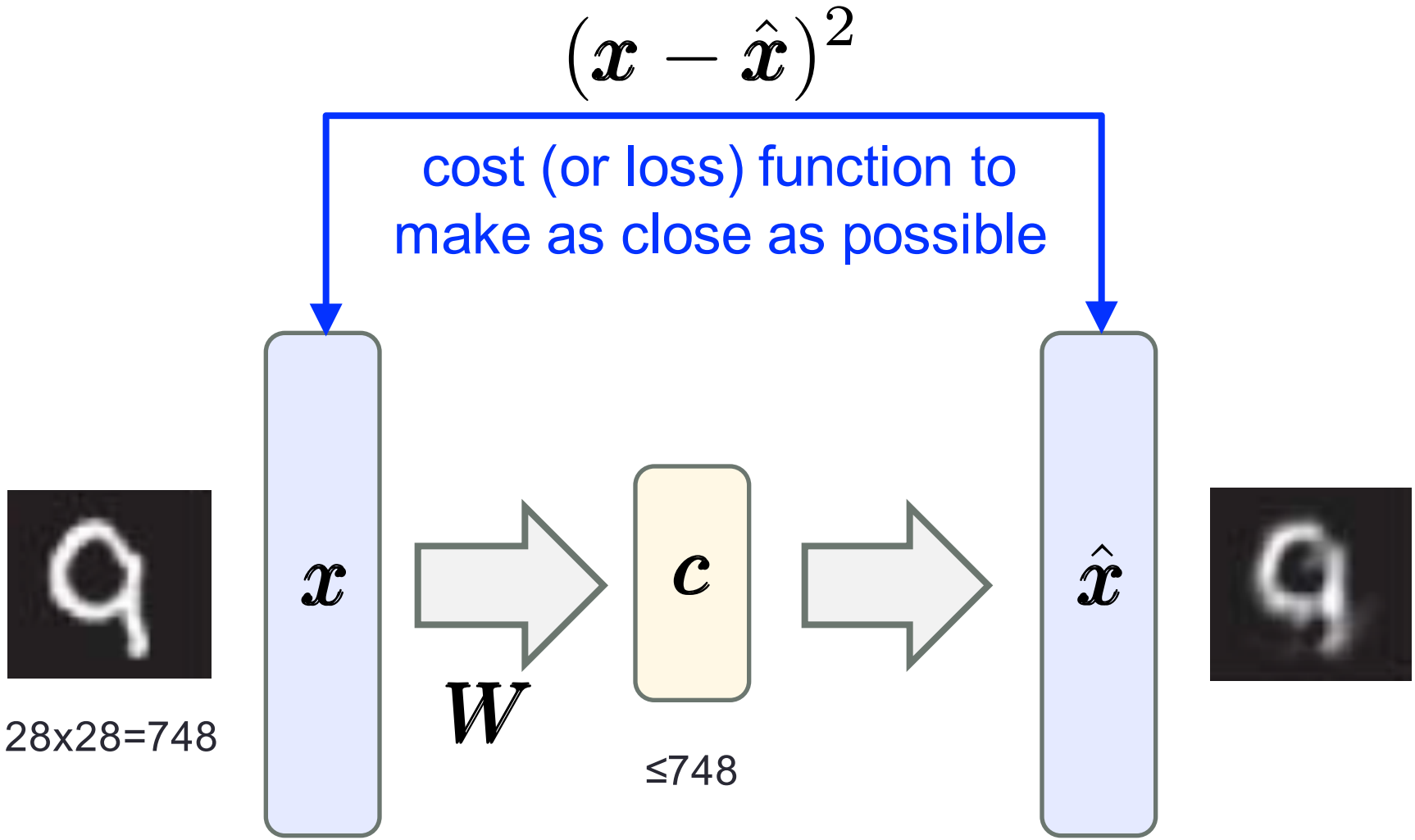


AUTOENCODERS

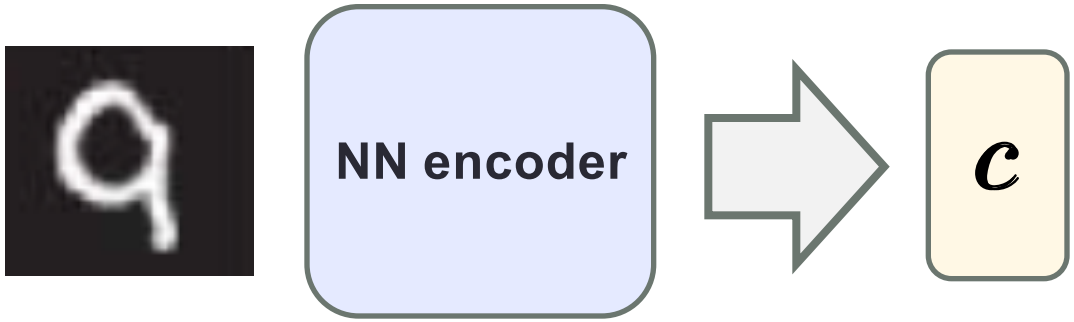
Principal Component Analysis (PCA)

- 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

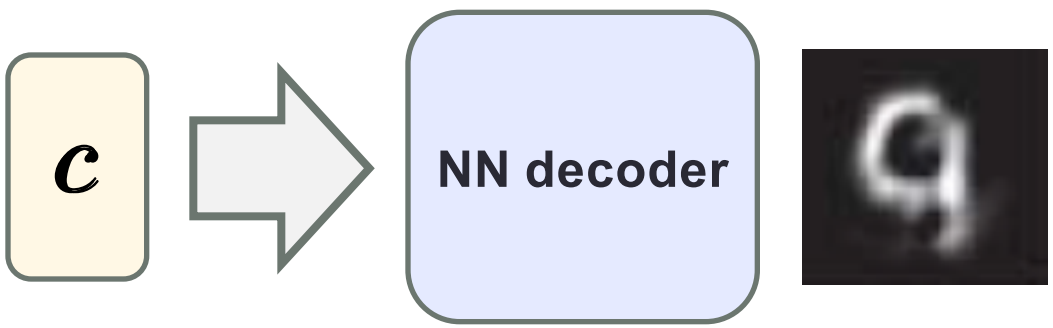
Principal Component Analysis (PCA)



Autoencoder [Hinton06]



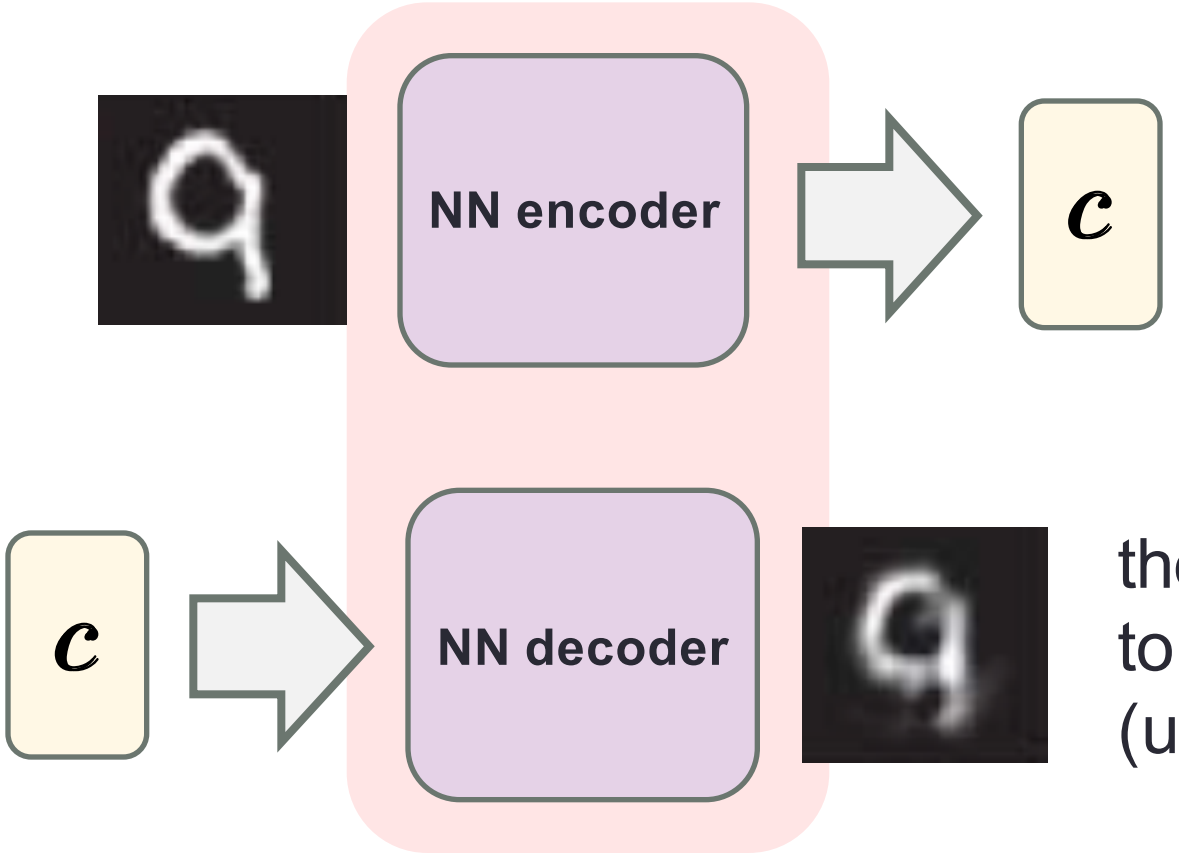
the code: a compact representation of the input object



the code: can be used to reconstruct the object (up to some accuracy)

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

Autoencoder



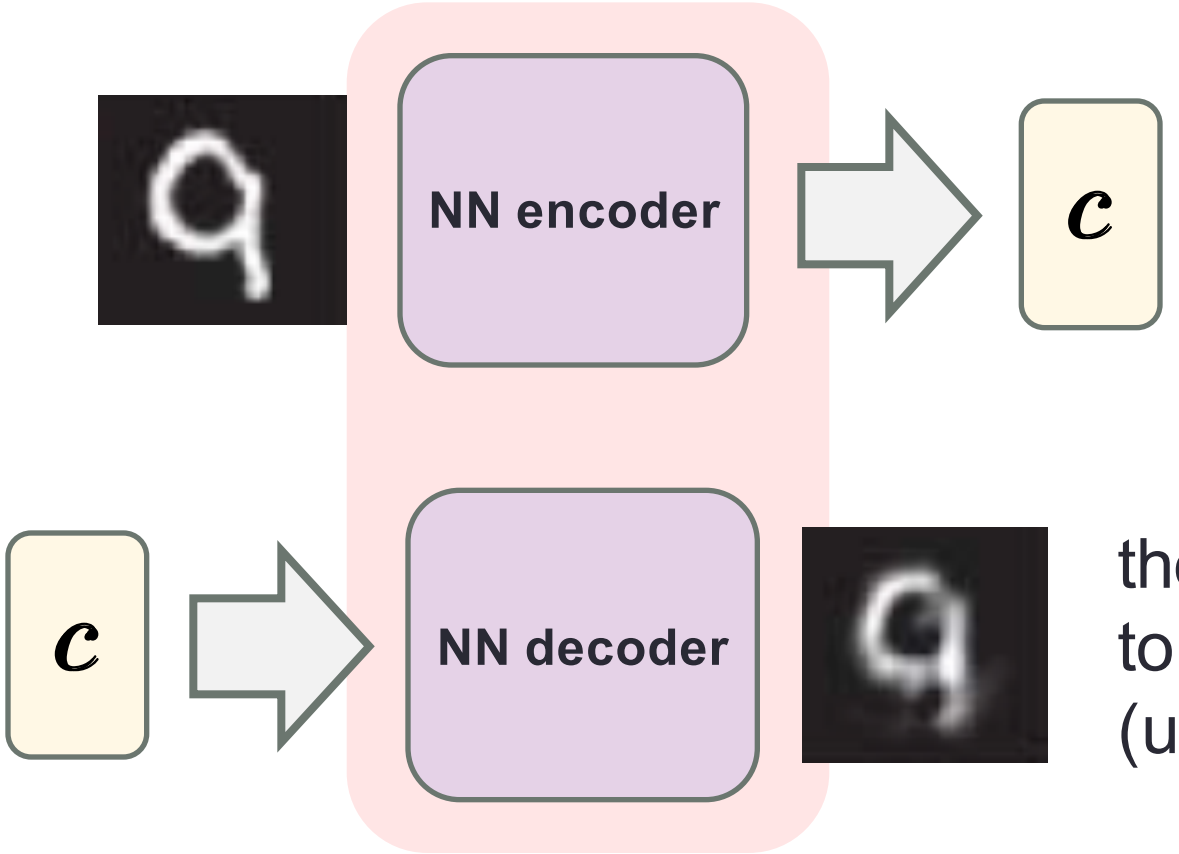
the code: a compact representation of the input object

the code: can be used to reconstruct the object (up to some accuracy)

Key point 1: encoder and decoder

- are jointly trained

Autoencoder



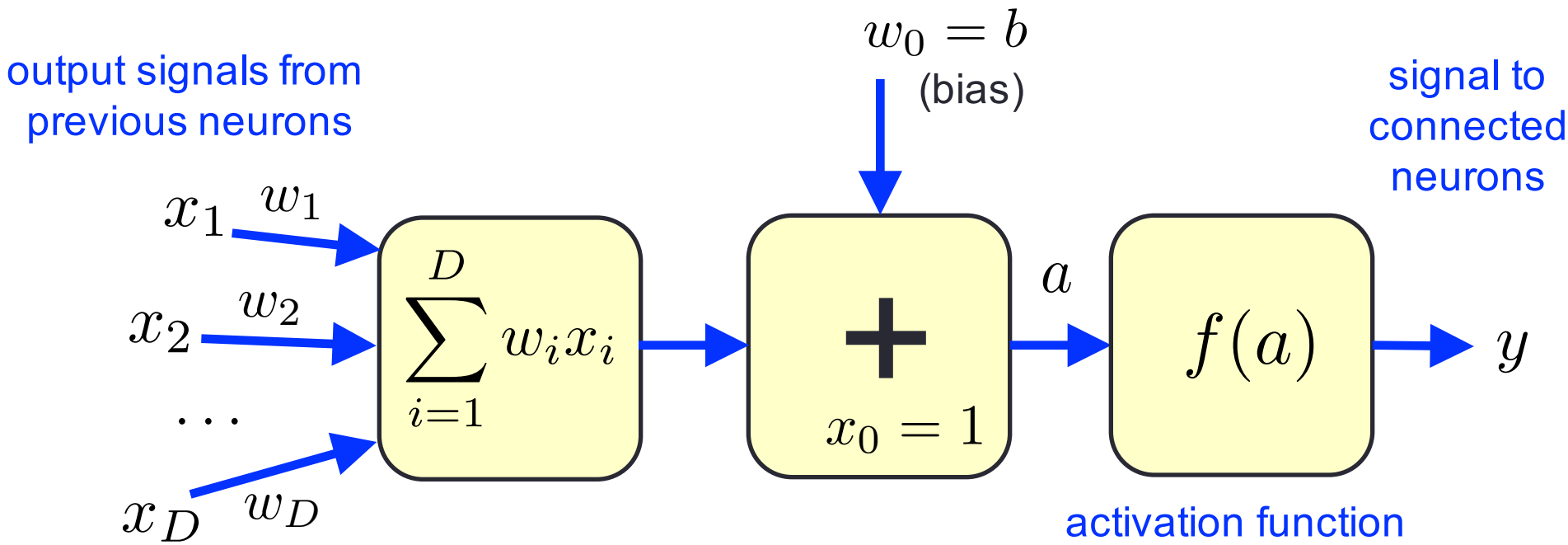
the code: a compact representation of the input object

the code: can be used to reconstruct the object (up to some accuracy)

Key point 2: encoder and decoder

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

The artificial neuron



$$y = f(a) = f\left(\sum_{j=1}^{D} w_j x_j + b\right) = f\left(\sum_{j=0}^{D} w_j x_j\right)$$

The bias b is included in the weight vector, $b=w_0$, by extending the input vector, i.e., adding element $x_0=1$

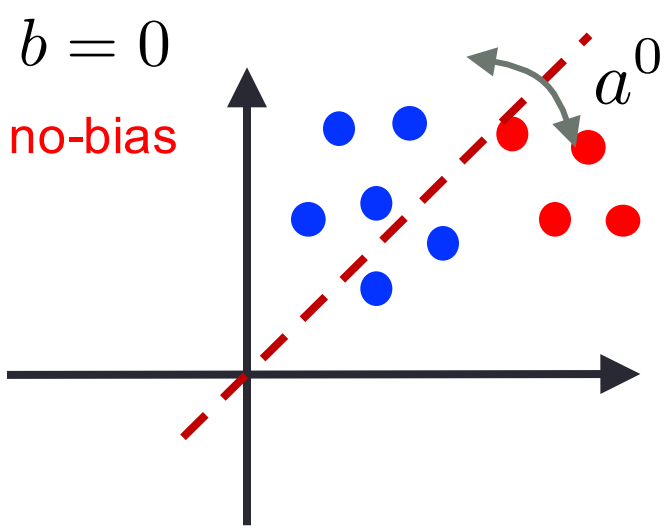
Importance of the bias

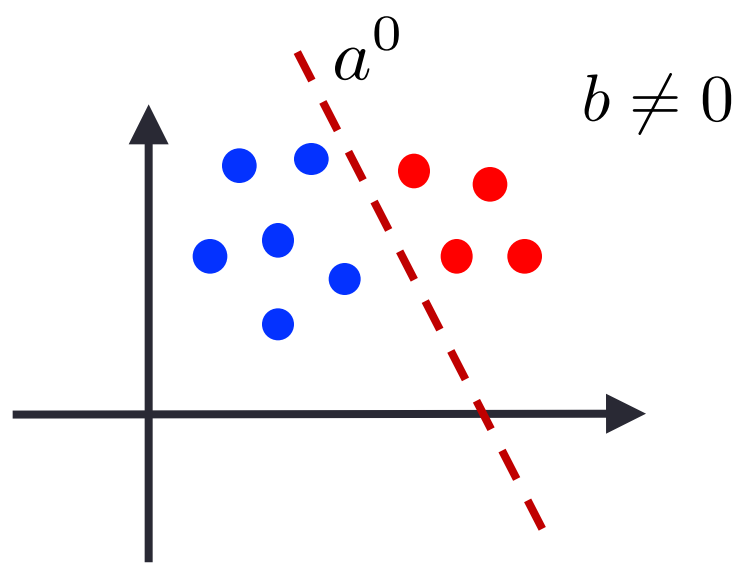
- Output of a neuron:

$$y(a) = f(a) = f\left(\sum_{i=1}^{D} w_i x_i + b\right) = f\left(\sum_{i=0}^{D} w_i x_i\right)$$

for all activation functions, the discriminating plane is that passing through zero:

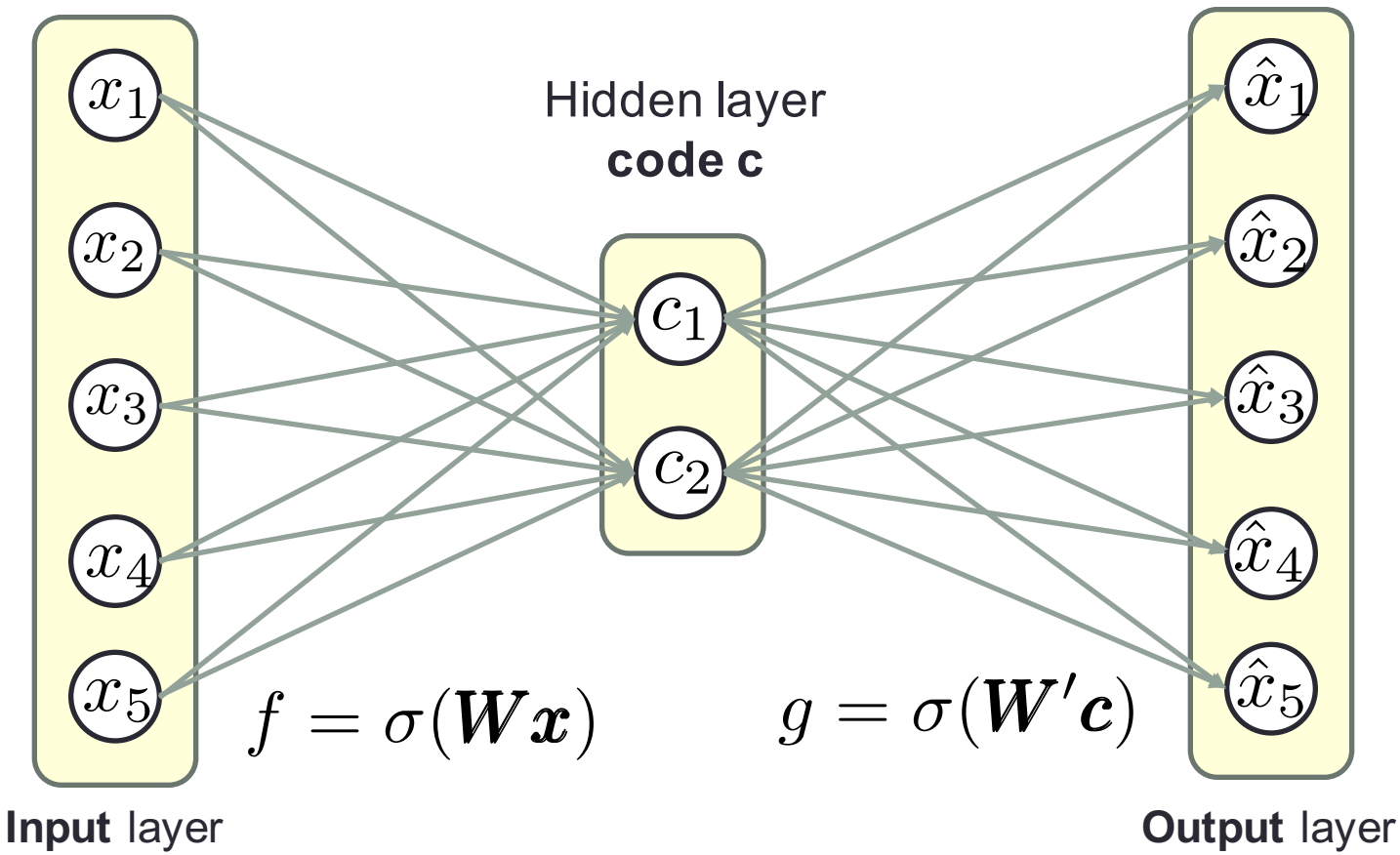
$$a^0 \triangleq \sum_i w_i x_i = 0$$





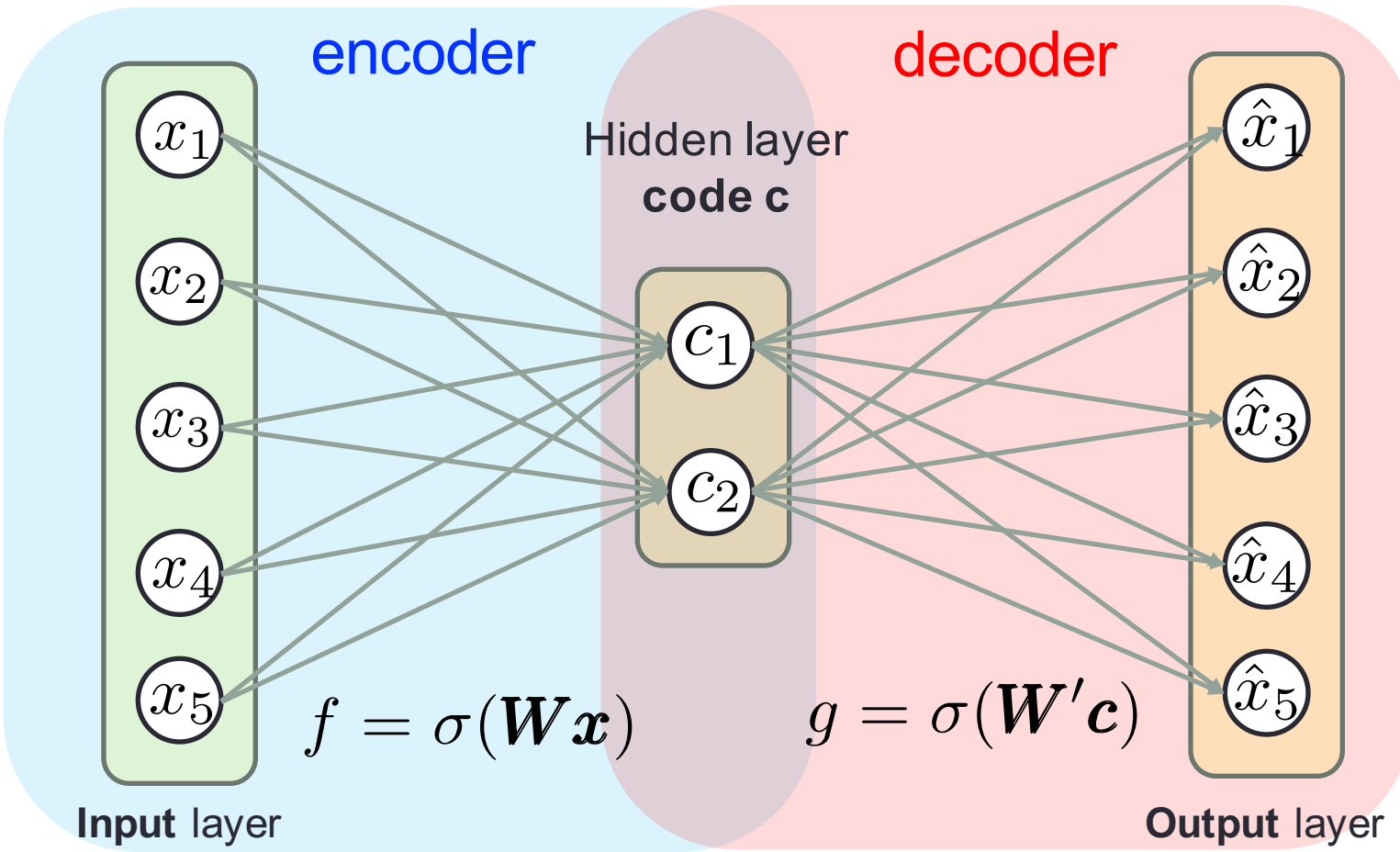
Autoencoder through FFNNs

- 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

Batch-mode learning

- Every input vector $\boldsymbol{x}_n$ is evaluated to get the derivatives $\dfrac{\partial E_n(\boldsymbol{w})}{\partial w_{ij}}$
- These are summed up to obtain the total derivative

$$\frac{\partial E(\boldsymbol{w})}{\partial w_{ij}} = \sum_n \frac{\partial E_n(\boldsymbol{w})}{\partial w_{ij}}$$

- Algorithm (initialize t=0)
  - Measure $\boldsymbol{x}_n$ and perform forward pass obtaining $\hat{\boldsymbol{x}}_n$
  - Evaluate error at the output layer $\boldsymbol{x}_n - \hat{\boldsymbol{x}}_n$
  - Back-propagate error to obtain the derivatives $\partial E_n(\boldsymbol{w})/\partial w_{ij}$
  - Add up these derivatives to obtain the total derivative
  - Update the weights:

$$w_{ij}(t+1) = w_{ij}(t) - \eta \frac{\partial E(\boldsymbol{w})}{\partial w_{ij}}$$

  - t=t+1 and repeat

Stochastic Gradient Descent (SGD)

- Weight updates are performed for each input pattern
- Generally patterns are picked at random from the data set

- Algorithm (initialize t=0)
  - Pick an input vector $\boldsymbol{x}_n$ at random from the data set
  - Forward propagate it through the network to obtain output $\hat{\boldsymbol{x}}_n$
  - Obtain error at output layer $\boldsymbol{x}_n - \hat{\boldsymbol{x}}_n$
  - Back-propagate error, obtaining $\partial E_n(\boldsymbol{w}) / \partial w_{ij}$
  - Update weights immediately:

$$w_{ij}(t+1) = w_{ij}(t) - \eta \frac{\partial E_n(\boldsymbol{w})}{\partial w_{ij}}$$

  - t=t+1 and repeat

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 [Vincent04]:

> *"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****..."*

[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.

Denoising AE (1/3)

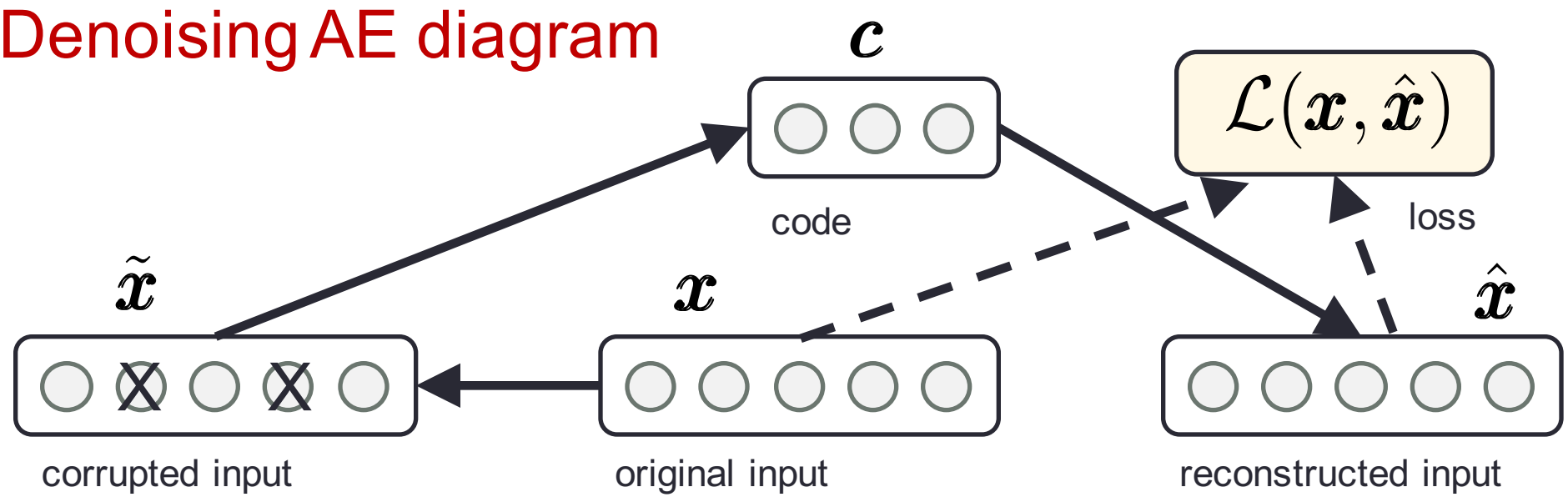
- Rather than constrain the representation (the code)
  - Train the AE for a more challenging task:
  - cleaning partially corrupted input (denoising)
- From [Vincent04]:

"...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..."*

[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.

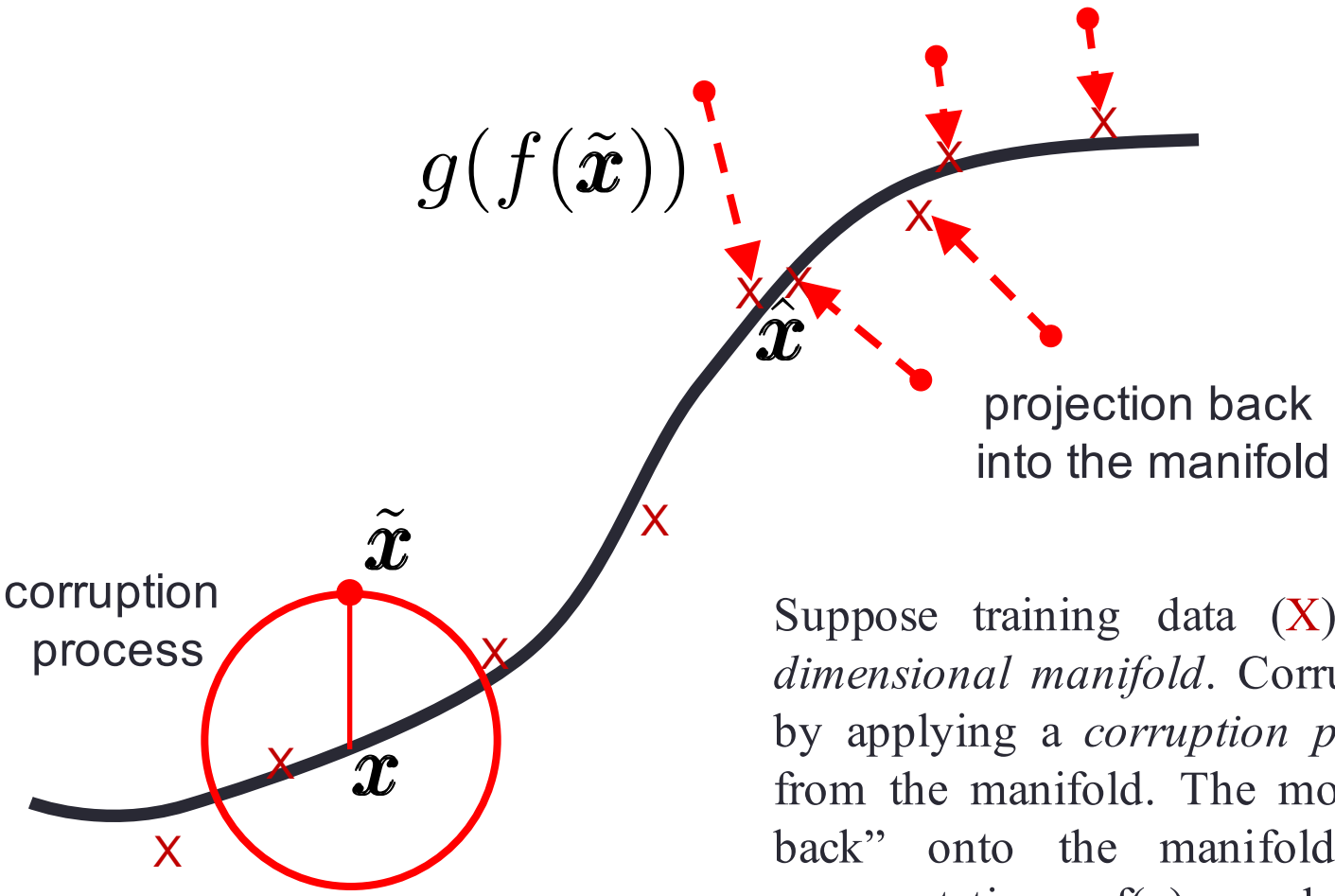
Denoising AE (2/3)

- 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



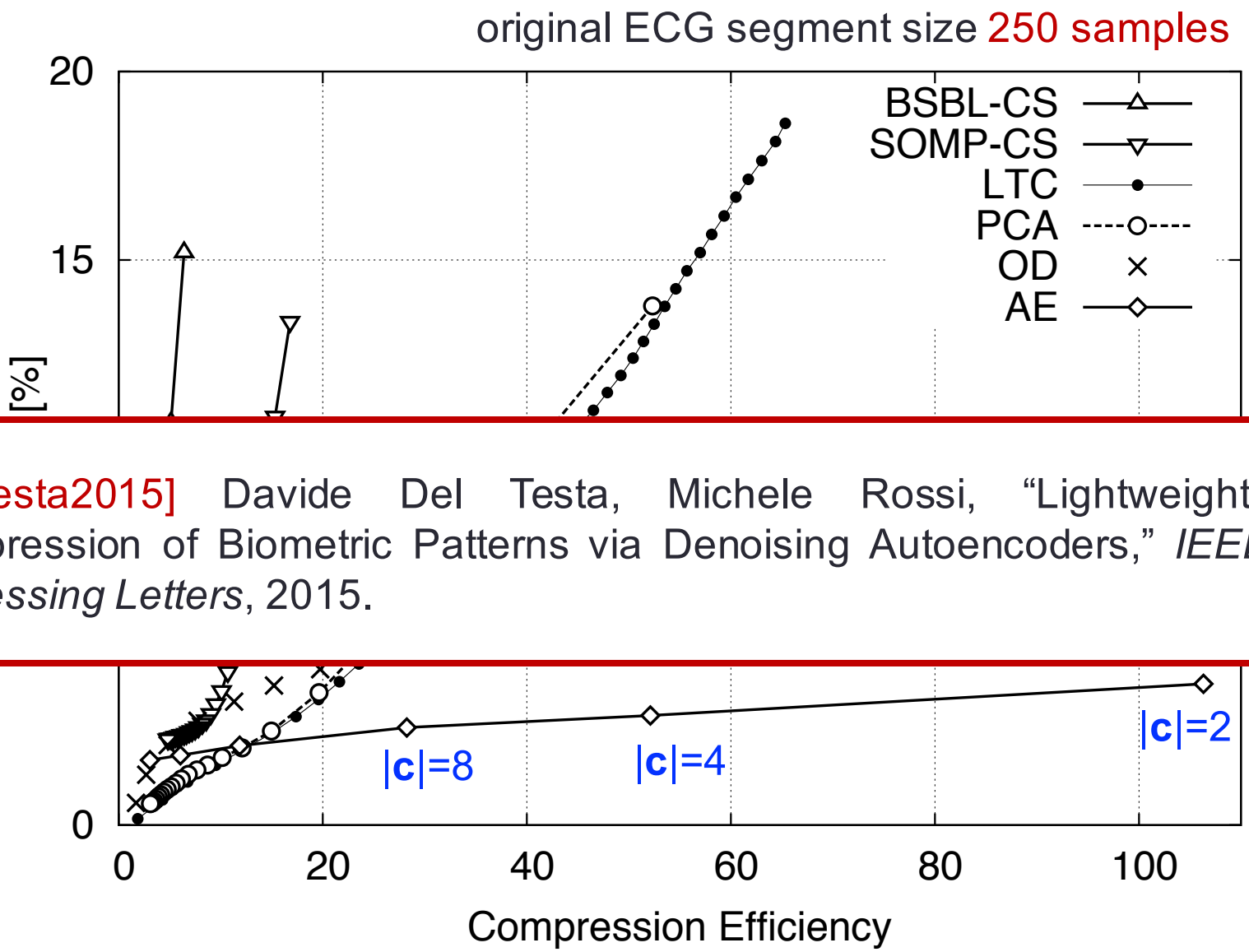
Denoising AE (3/3)

- 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.

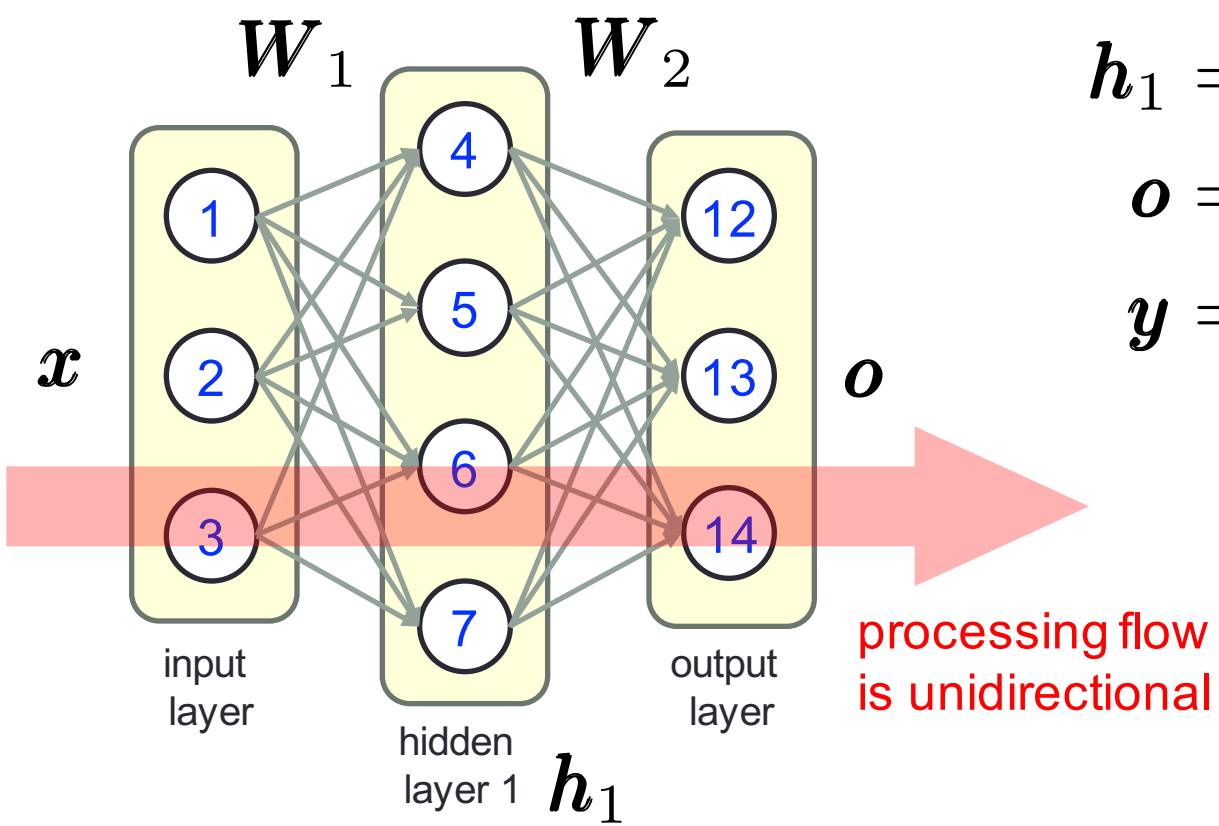
So far

- 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 1998-2004: Hidden Markov Models
  - 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



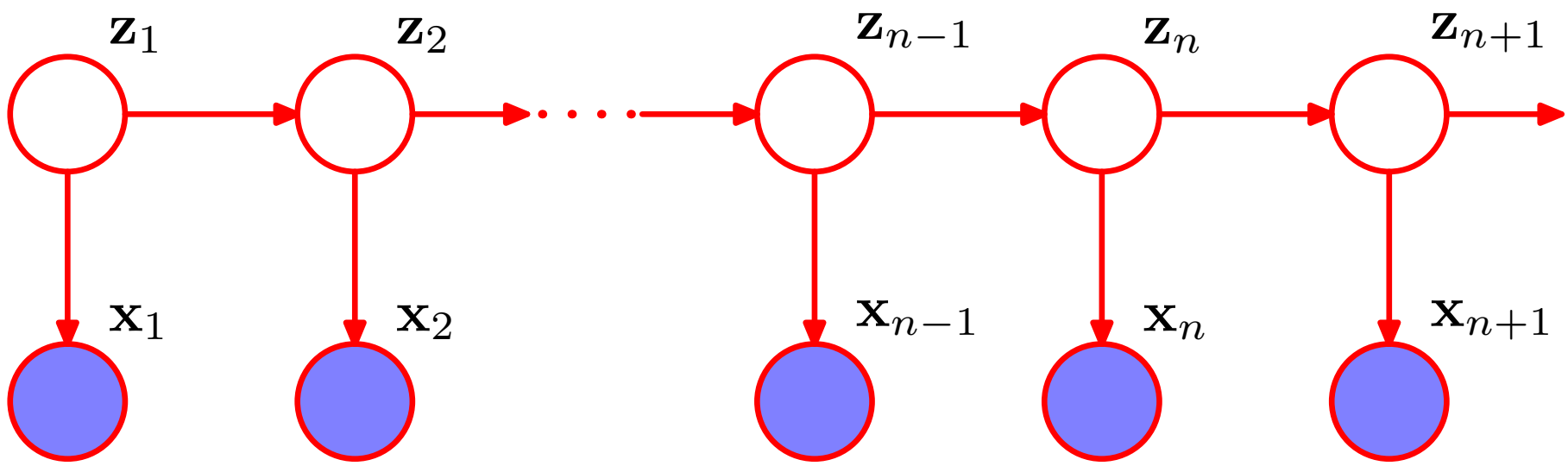
$$\boldsymbol{h}_1 = \sigma(\boldsymbol{W}_1\boldsymbol{x} + \boldsymbol{b}_1)$$
$$\boldsymbol{o} = \sigma(\boldsymbol{W}_2\boldsymbol{h}_1 + \boldsymbol{b}_2)$$
$$\boldsymbol{y} = \mathrm{softmax}(\boldsymbol{o})$$

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

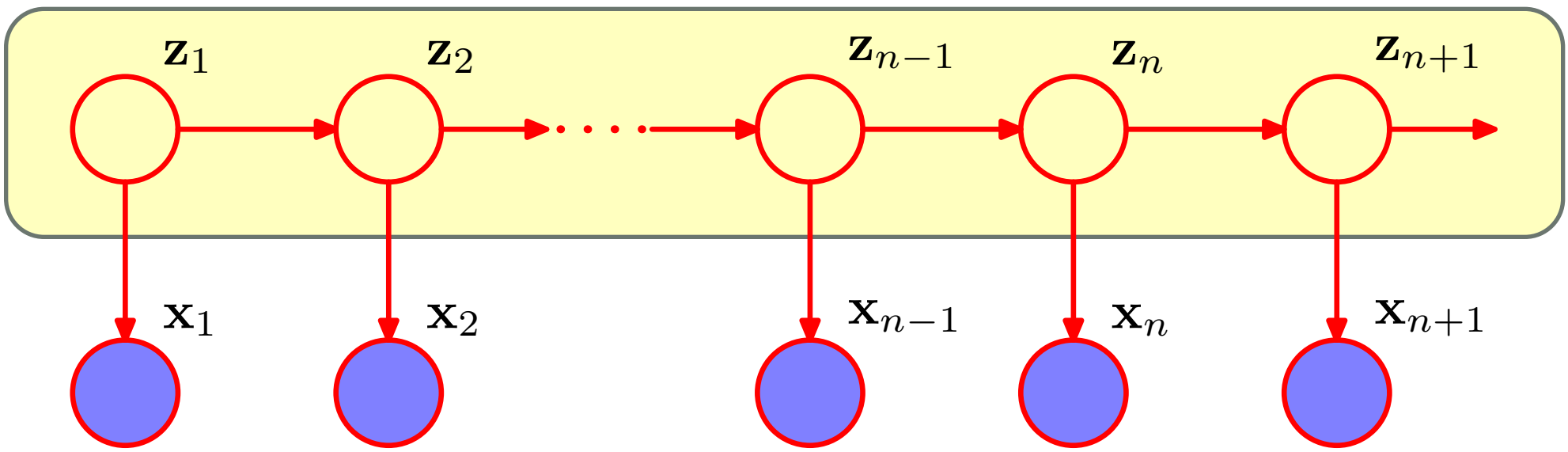
Hidden Markov Model (HMM) (1/4)

- Is a graphical model
- States: white filled circles
- Emissions or observations: blue filled circles
- Arrows: dependencies



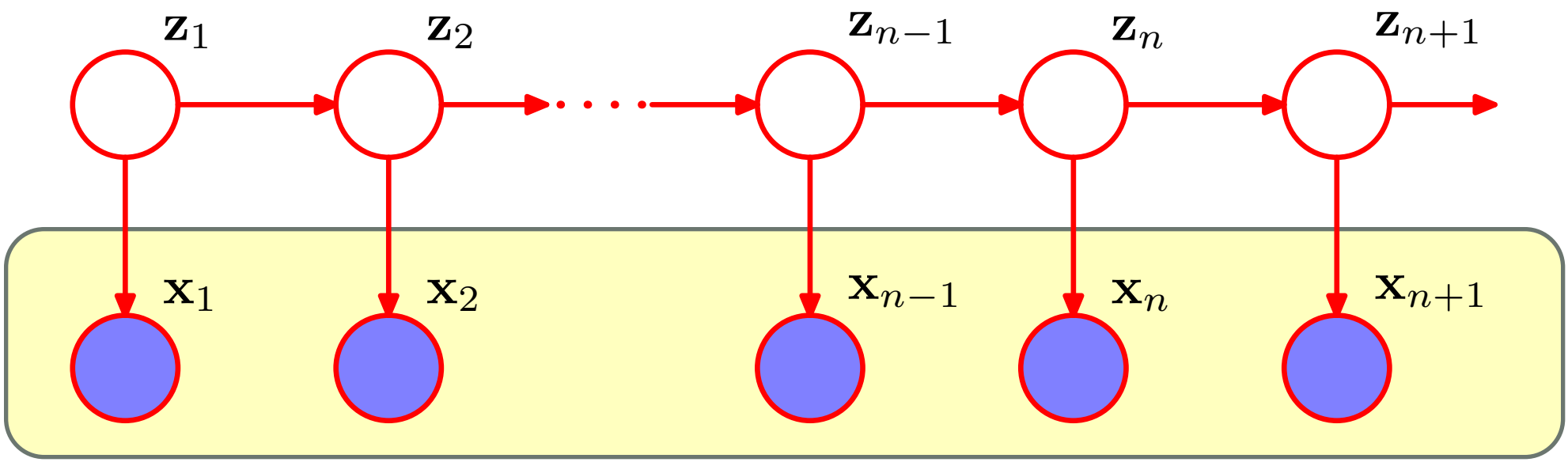
HMM (2/4)

- Hidden states (internal variables)
  - Each $\mathbf{z}_i$ vector has a 1-of-K coding (K values)
  - Discrete and finite
  - Are also called "latent variables" $\mathbf{z}_i$



HMM (3/4)

- Observations (or emissions)
  - Represent the measured, *noisy* sequence
  - Can be either discrete or continuous r.v. (e.g., GM PDF)
  - Each observation $\mathbf{x}_i$ is conditioned on the corresponding variable $\mathbf{z}_i$
    - and does not depend on other observations

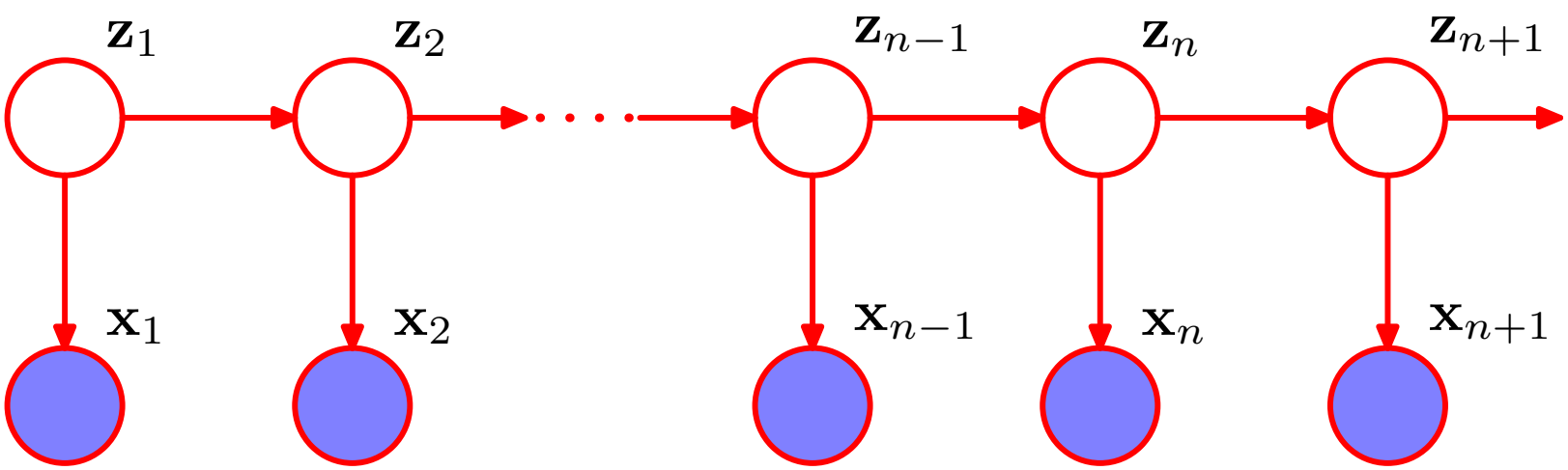


HMM (4/4)

- We allow $\mathbf{z}_n$ to depend on $\mathbf{z}_{n-1}$ through a conditional PDF

$$p(\boldsymbol{z}_n|\boldsymbol{z}_{n-1})$$

- Considering the 1-of-K representation of $\mathbf{z}_n$
  - Conditional PDF is compactly represented through a matrix $\boldsymbol{A} = [A_{jk}]$
  - We have: $A_{jk} \triangleq p(z_{nk} = 1|z_{n-1,j} = 1)$



HMM formalism

- Let us consider N observations

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

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

$$\boldsymbol{\theta} = \{\boldsymbol{\pi}, \boldsymbol{A}, \boldsymbol{\phi}\}$$ 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)

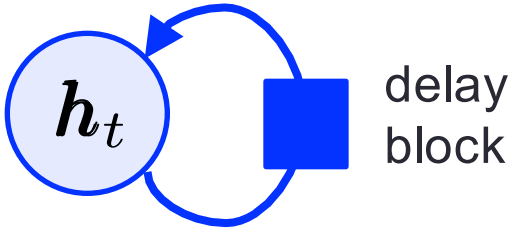
What's wrong with HMM?

- 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 growth in 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

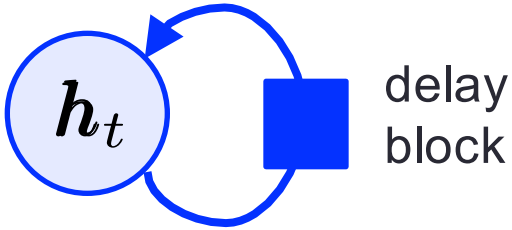
$$\boldsymbol{h}_t = f(\boldsymbol{h}_{t-1}; \boldsymbol{\theta})$$



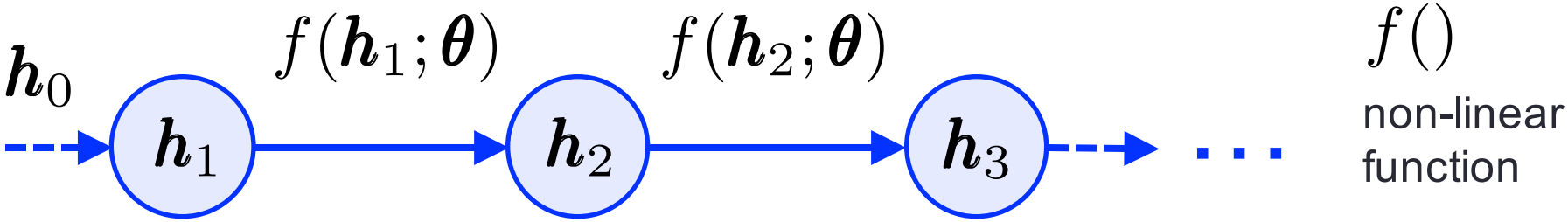
RNN – unfolding

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

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



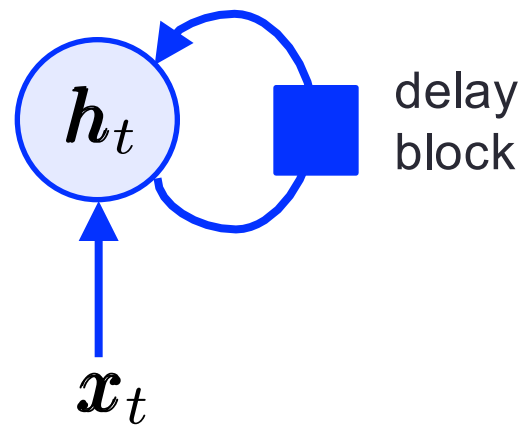
- Unfolding
  - Lead to a Directed Acyclic Graph (DAG) structure
  - Much preferable for training weights



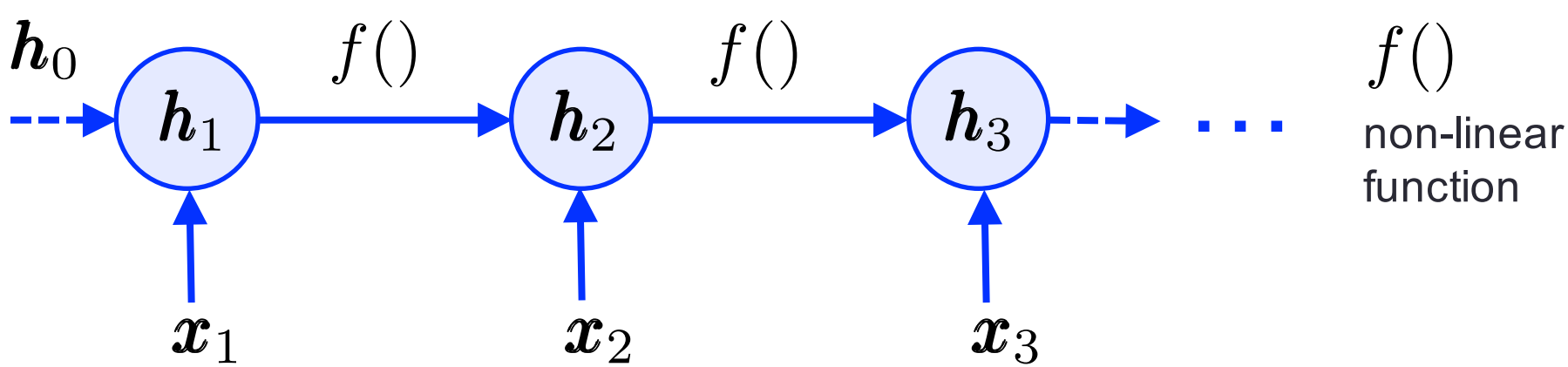
RNN – external input

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

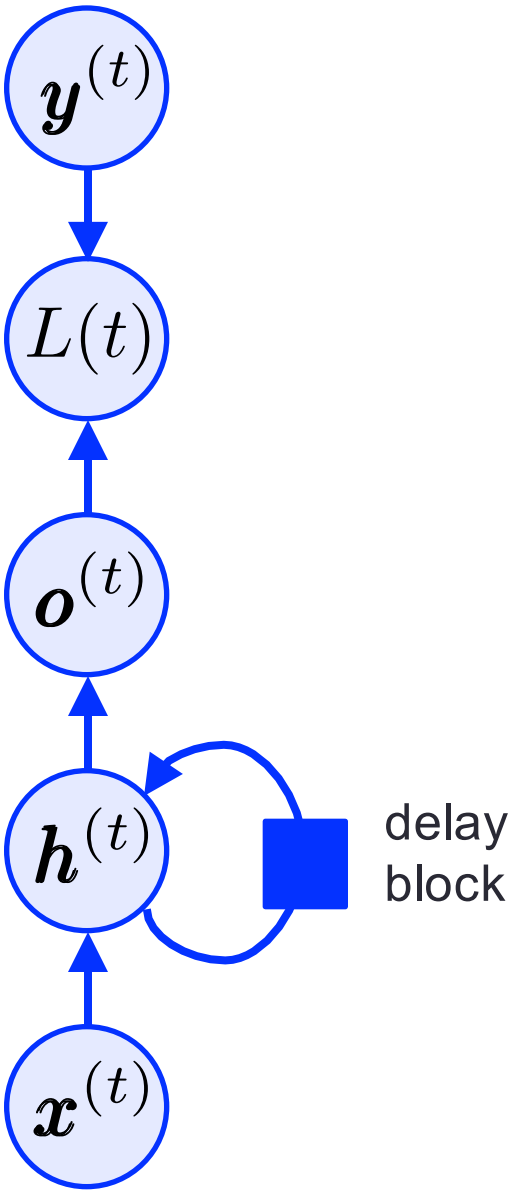
$$\boldsymbol{h}_t = f(\boldsymbol{h}_{t-1}, \boldsymbol{x}_t; \boldsymbol{\theta})$$



- 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

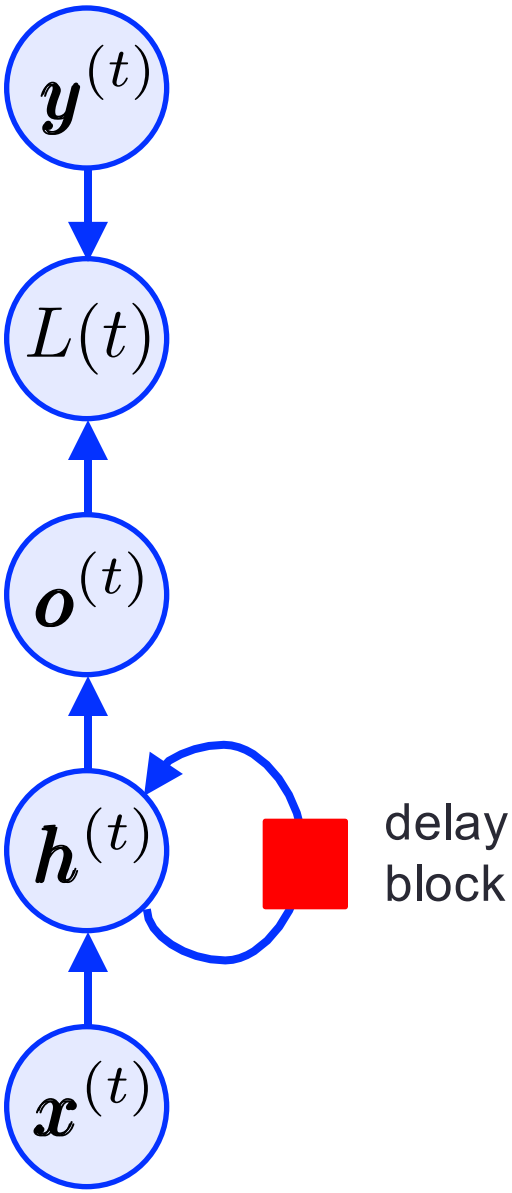
$$\boldsymbol{h}^{(t)} = \tanh\left(\boldsymbol{a}^{(t)}\right)$$

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)

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

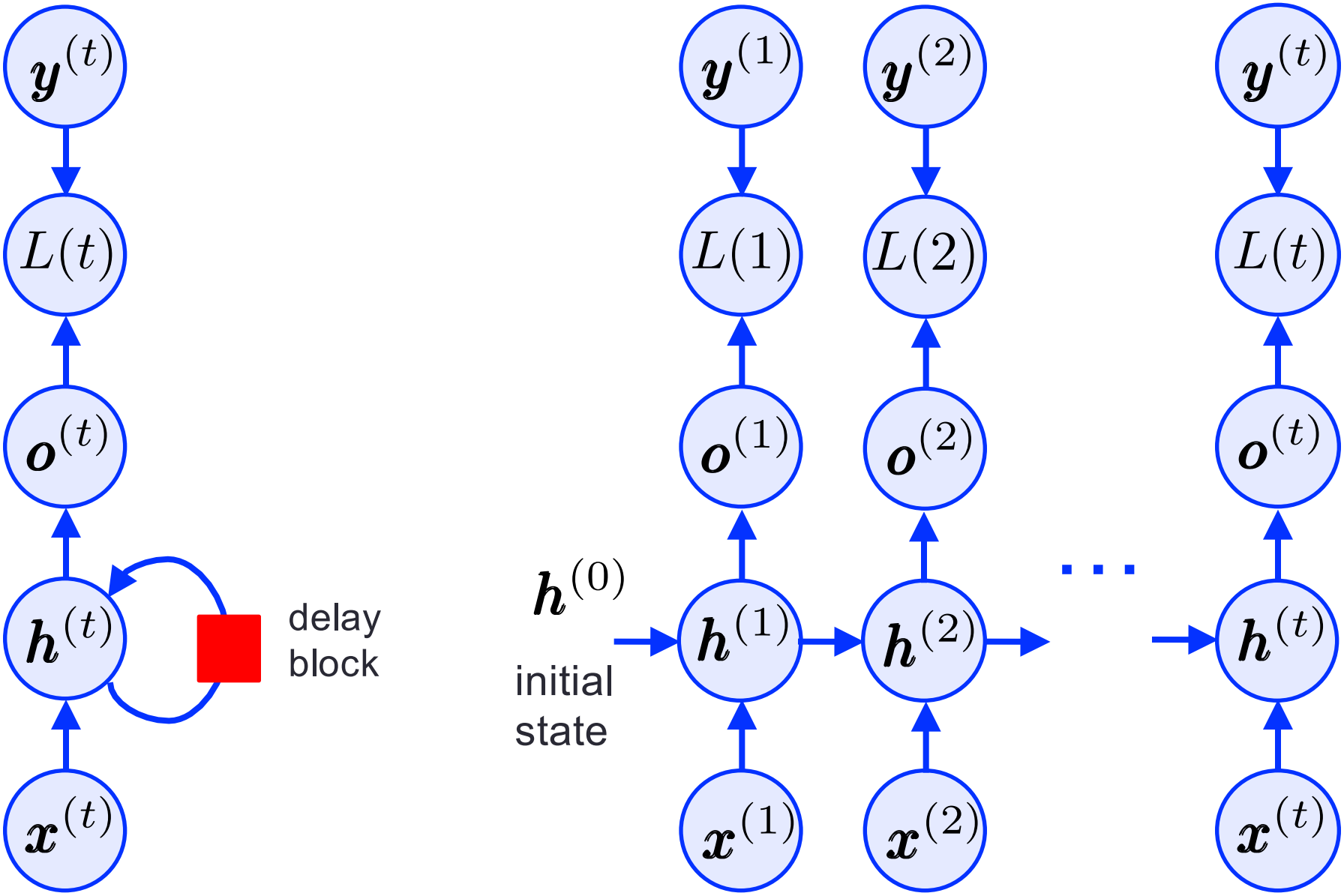
$$\boldsymbol{a}^{(t)} = \boldsymbol{U}\boldsymbol{x}^{(t)} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{b}$$

recurrence

Input at time t

RNN unfolding

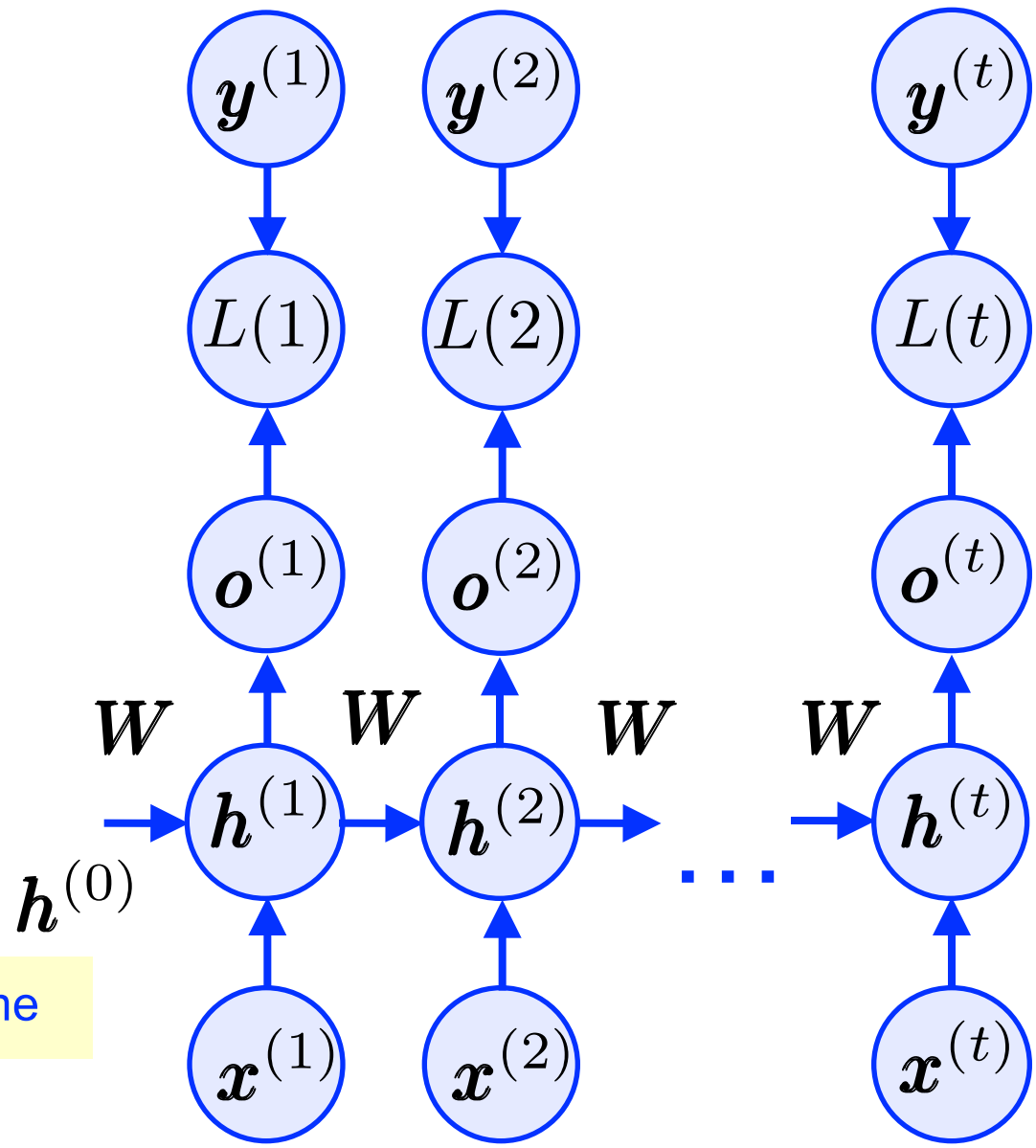
Unfolded graph is a standard DAG



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
  - 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 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 regression, 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 convNets)
  - 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 to T and find derivatives of total error wrt all network parameters (weights, biases):

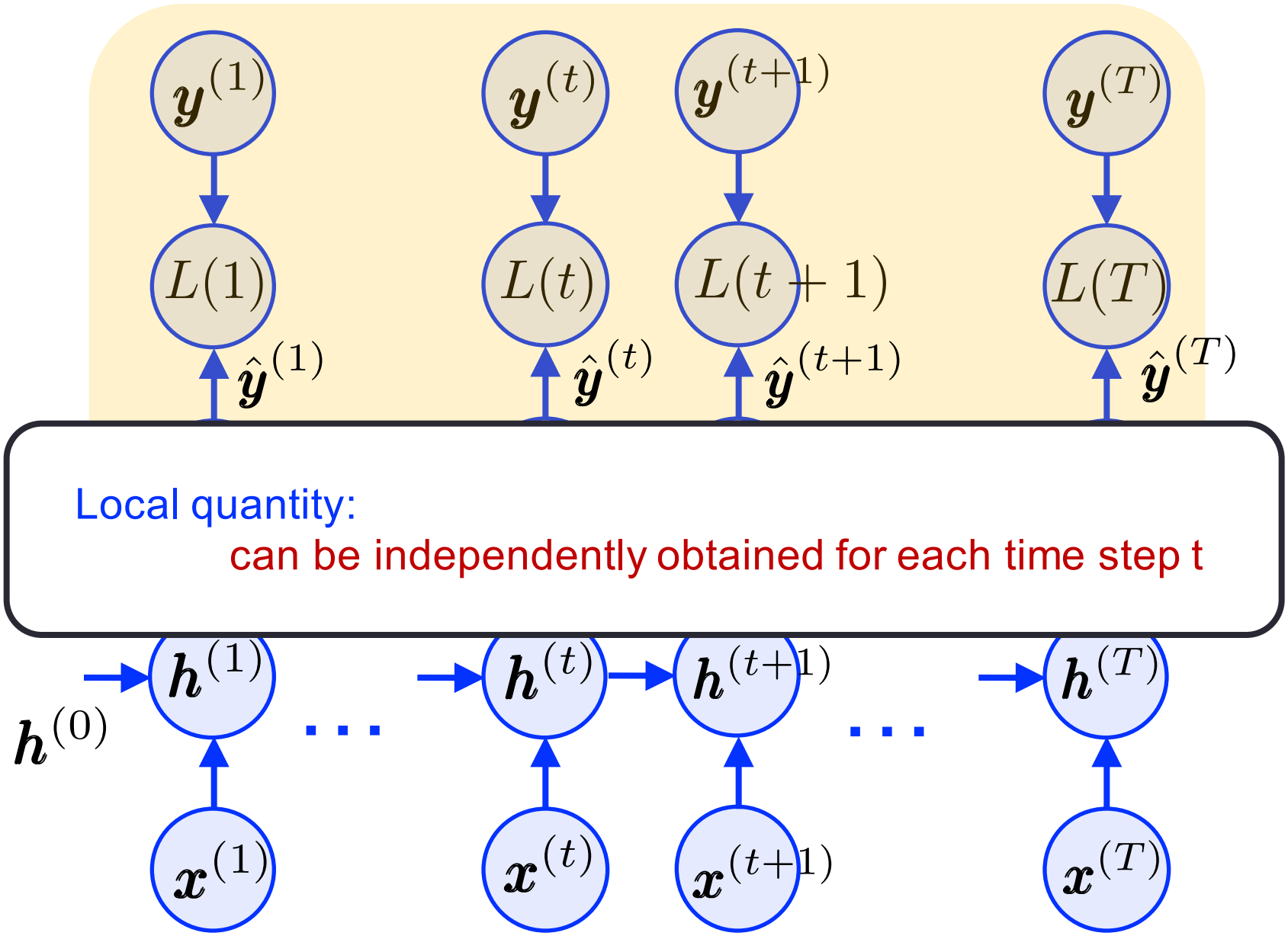
$$\frac{\partial L}{\partial \boldsymbol{V}}, \frac{\partial L}{\partial \boldsymbol{W}}, \frac{\partial L}{\partial \boldsymbol{U}}, \nabla_{\boldsymbol{b}}(L), \nabla_{\boldsymbol{c}}(L)$$

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

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

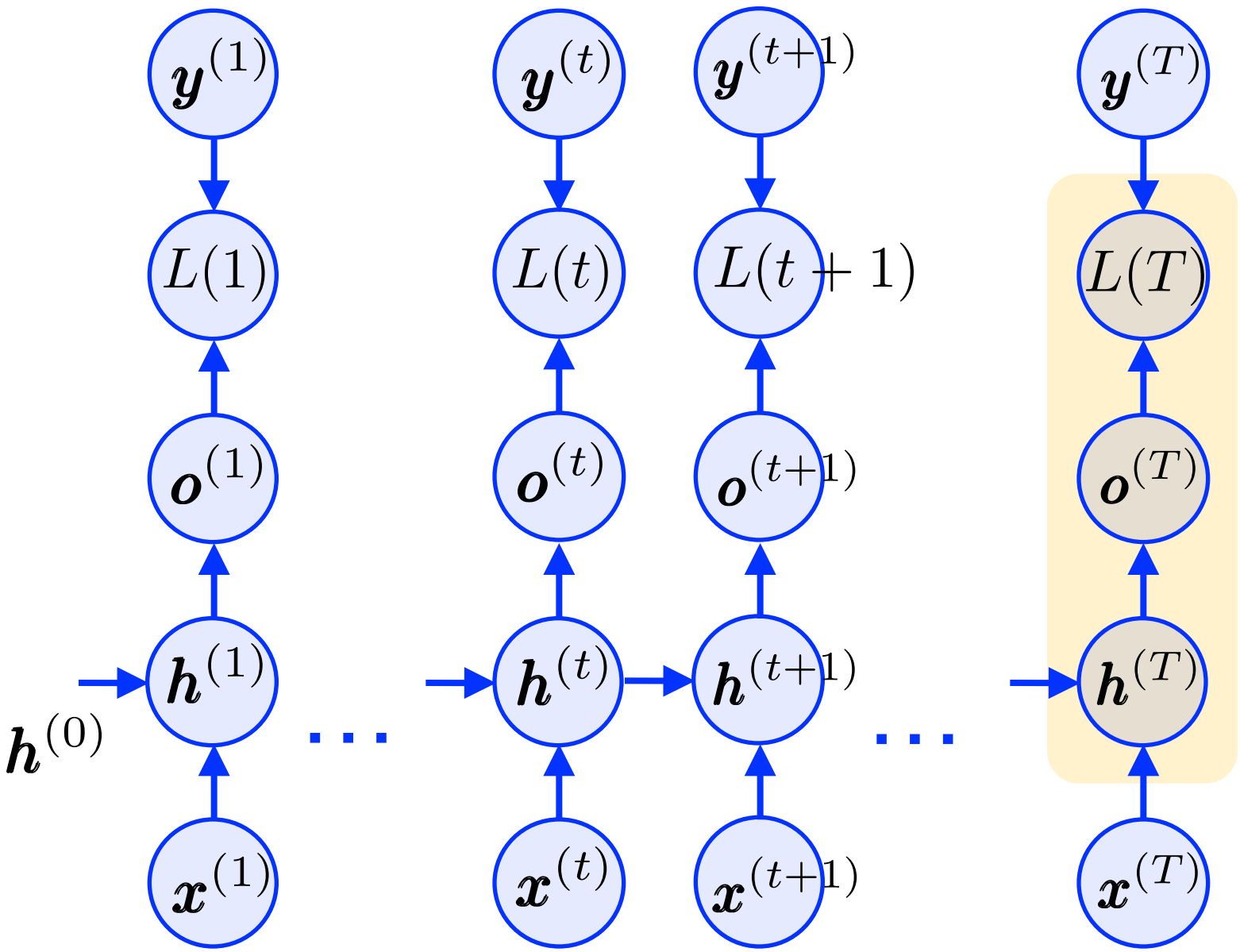
BTT – unfolded graph

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



BTT – time T

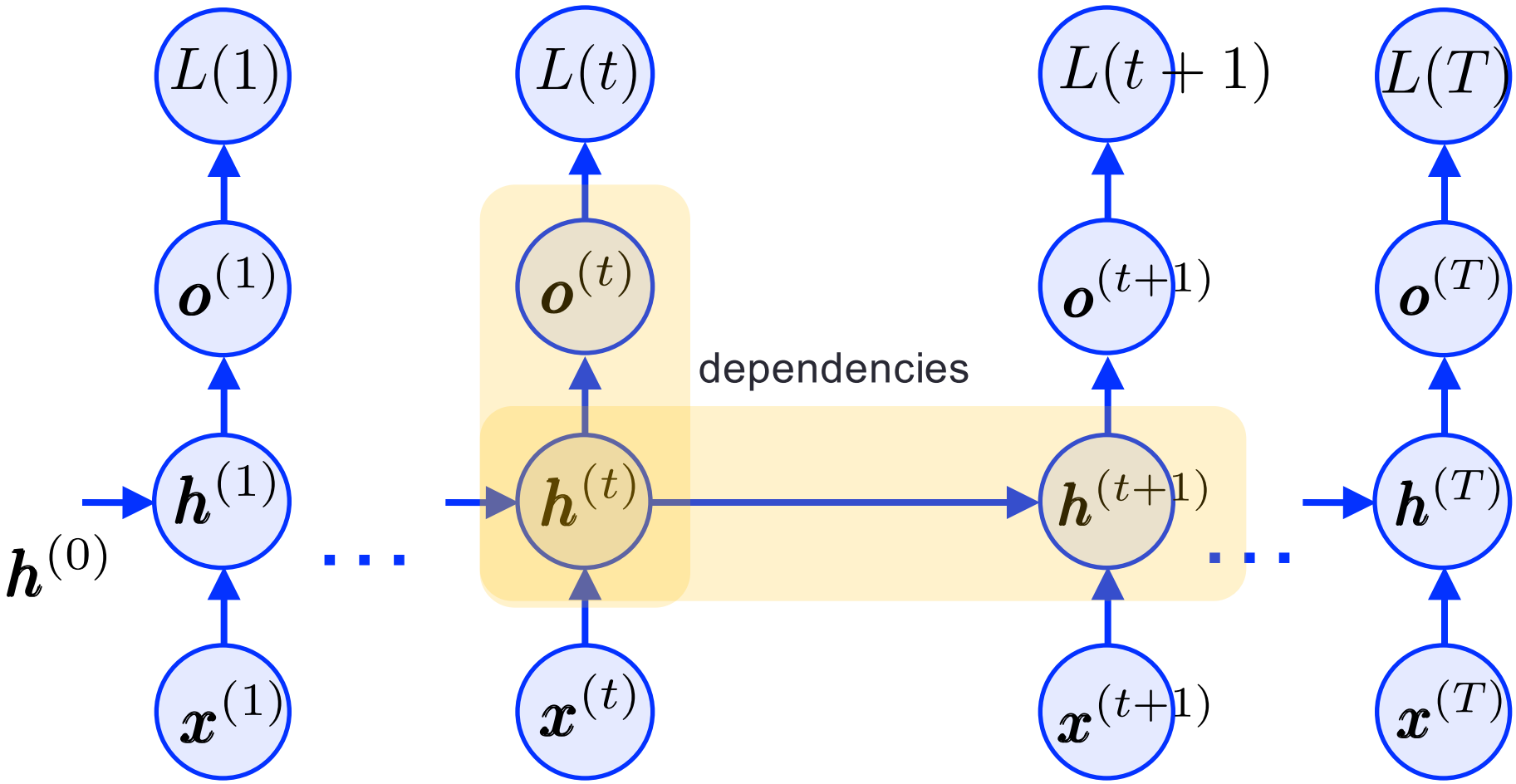
$$\nabla_{\boldsymbol{h}^{(T)}} L = (\boldsymbol{V})^T \nabla_{\boldsymbol{o}^{(T)}} L$$



BTT – time 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

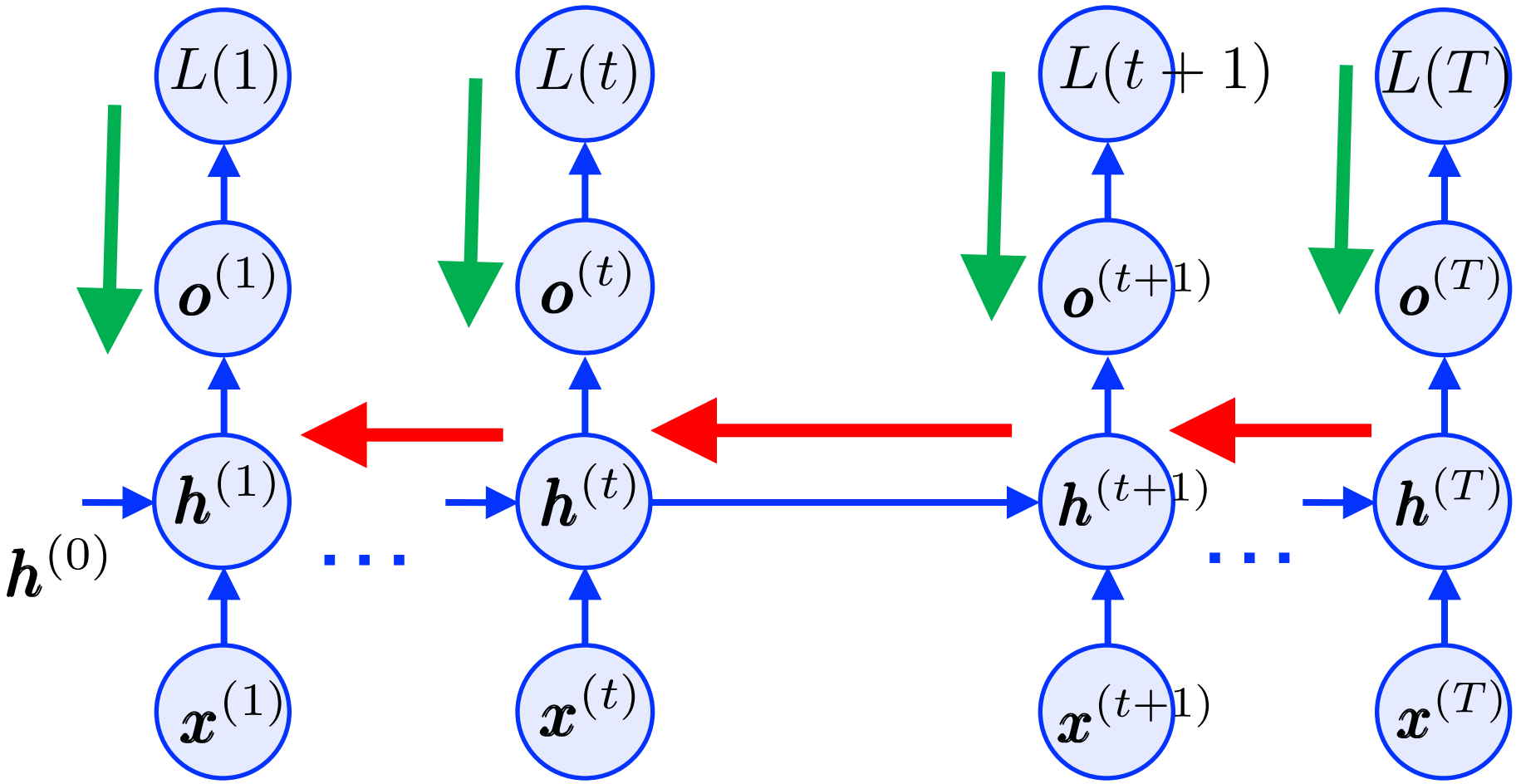
gradient of $\mathbf{h}^{(t)}$ is computed recursively

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

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]

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

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

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

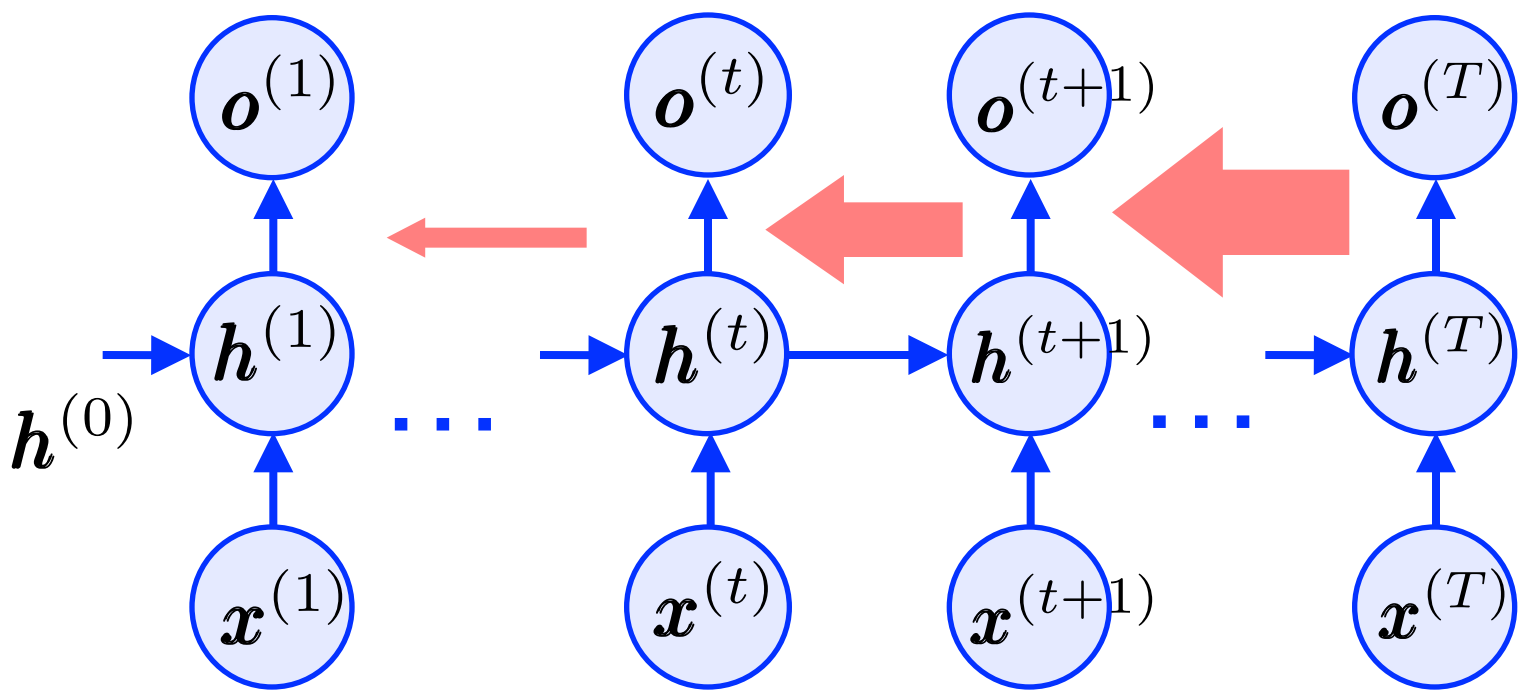
$$\nabla_{\boldsymbol{U}} L = \sum_{t=1}^{T} \operatorname{diag}\left(1 - \left(\boldsymbol{h}^{(t)}\right)^2\right) \left(\nabla_{\boldsymbol{h}^{(t)}} L\right) \left(\boldsymbol{x}^{(t)}\right)^T$$

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

BTT – vanishing gradients problem

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

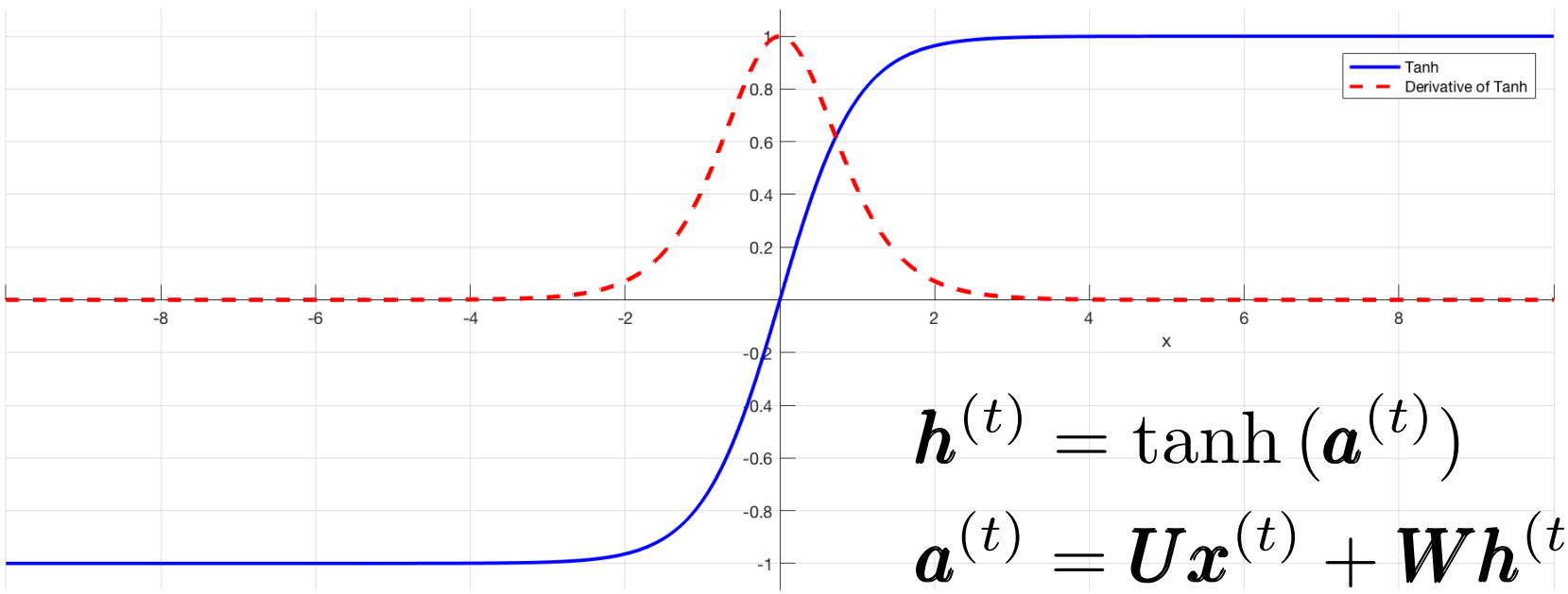
Note that: $\operatorname{diag}(1 - (\boldsymbol{h}^{(t)})^2) = \dfrac{\partial \tanh(\boldsymbol{h}^{(t)})}{\partial \boldsymbol{h}^{(t)}}$



BTT – vanishing gradients problem

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

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



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

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

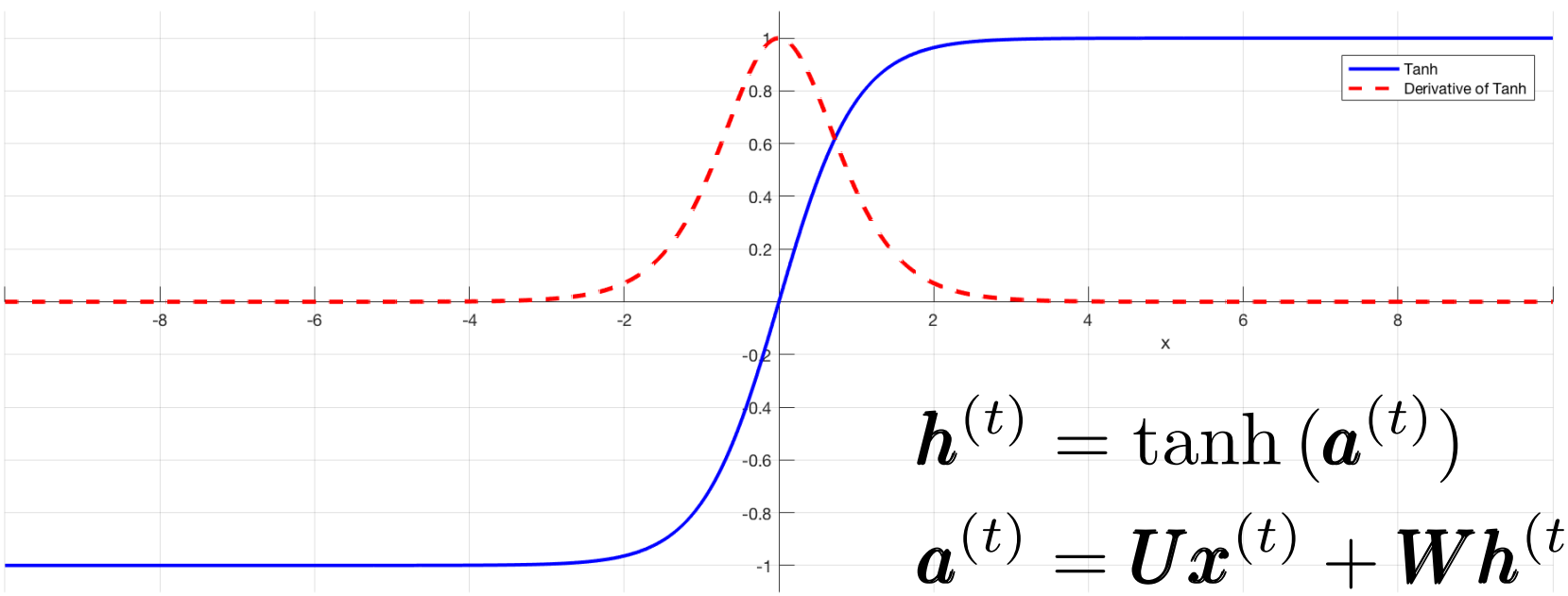
BTT – vanishing gradients problem

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

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

The tanh gradient vanishes:

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



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

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

BTT – vanishing gradients problem

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

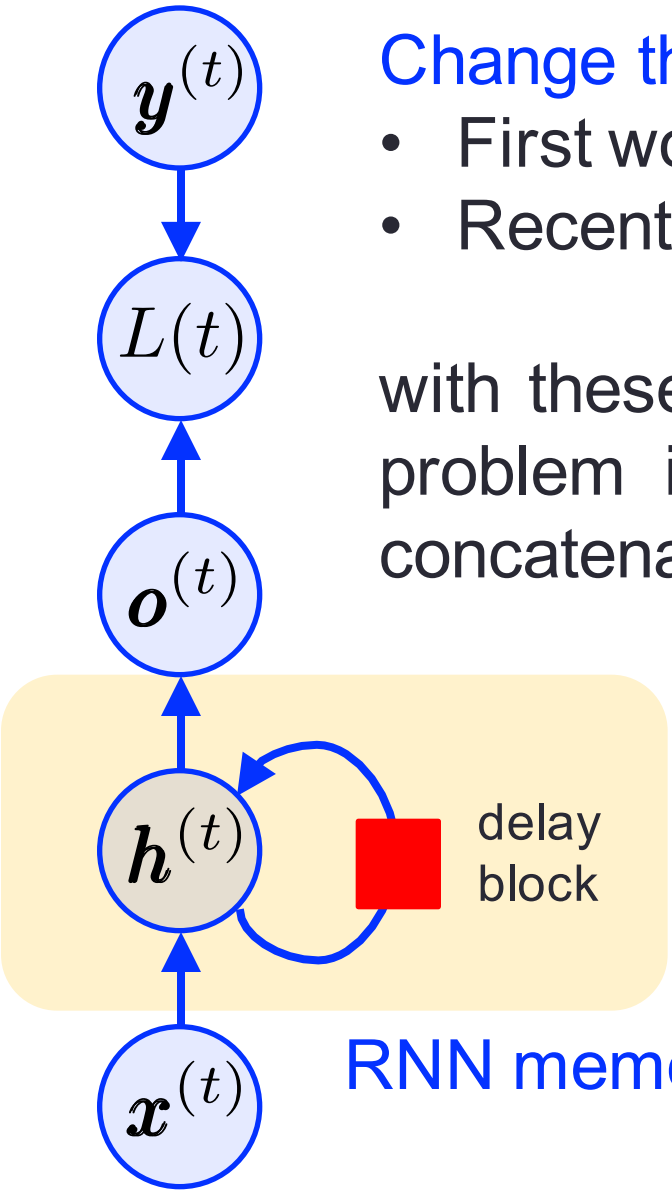
The tanh gradient vanishes:

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

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

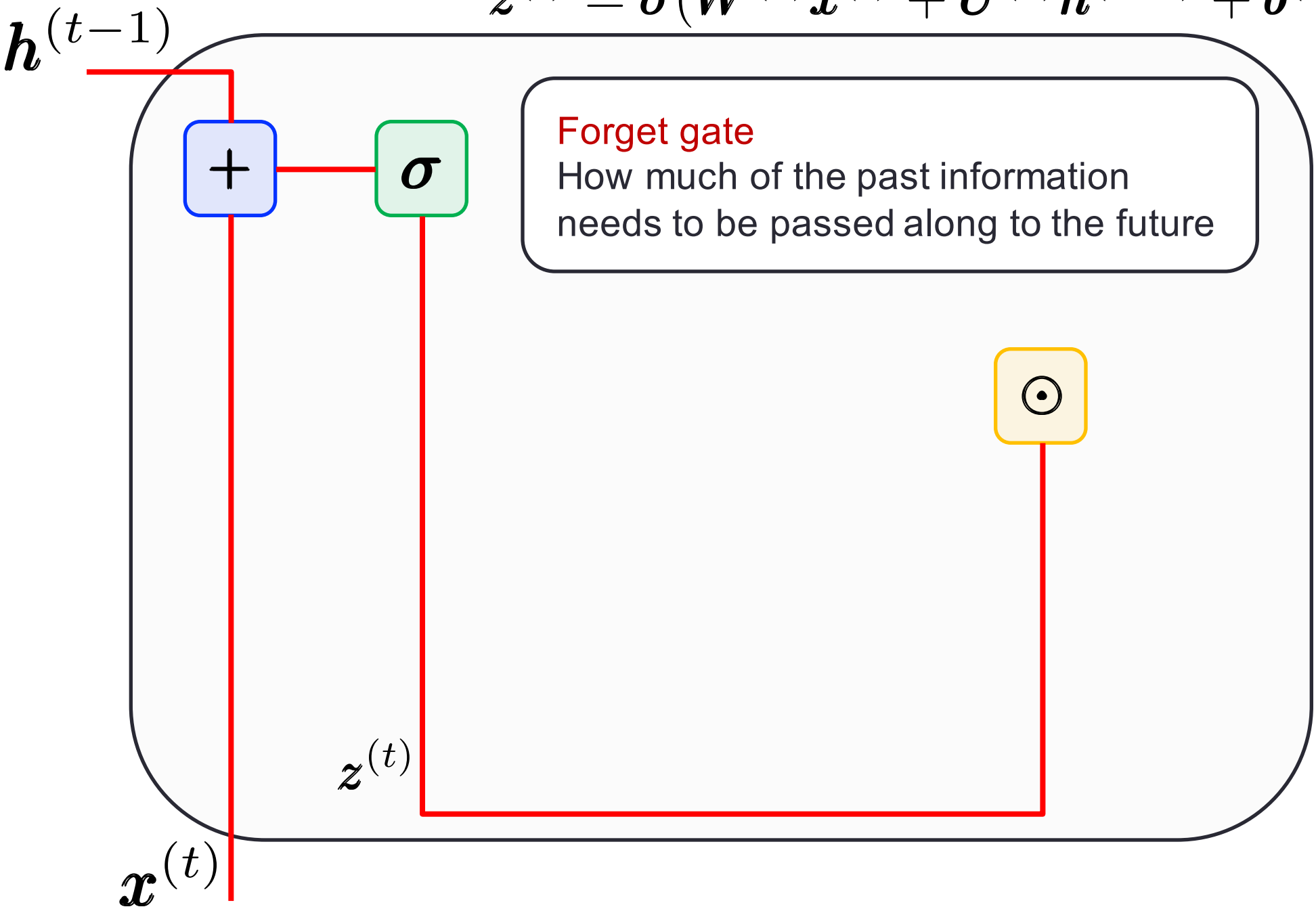
GRU cells in four steps…

Definitions

$\tanh$	hyperbolic tangent (element-wise)
σ	sigmoid nonlinearity (element-wise)
$+$	vector addition
$\odot$	element-wise product between vectors

Step 1: forget gate

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

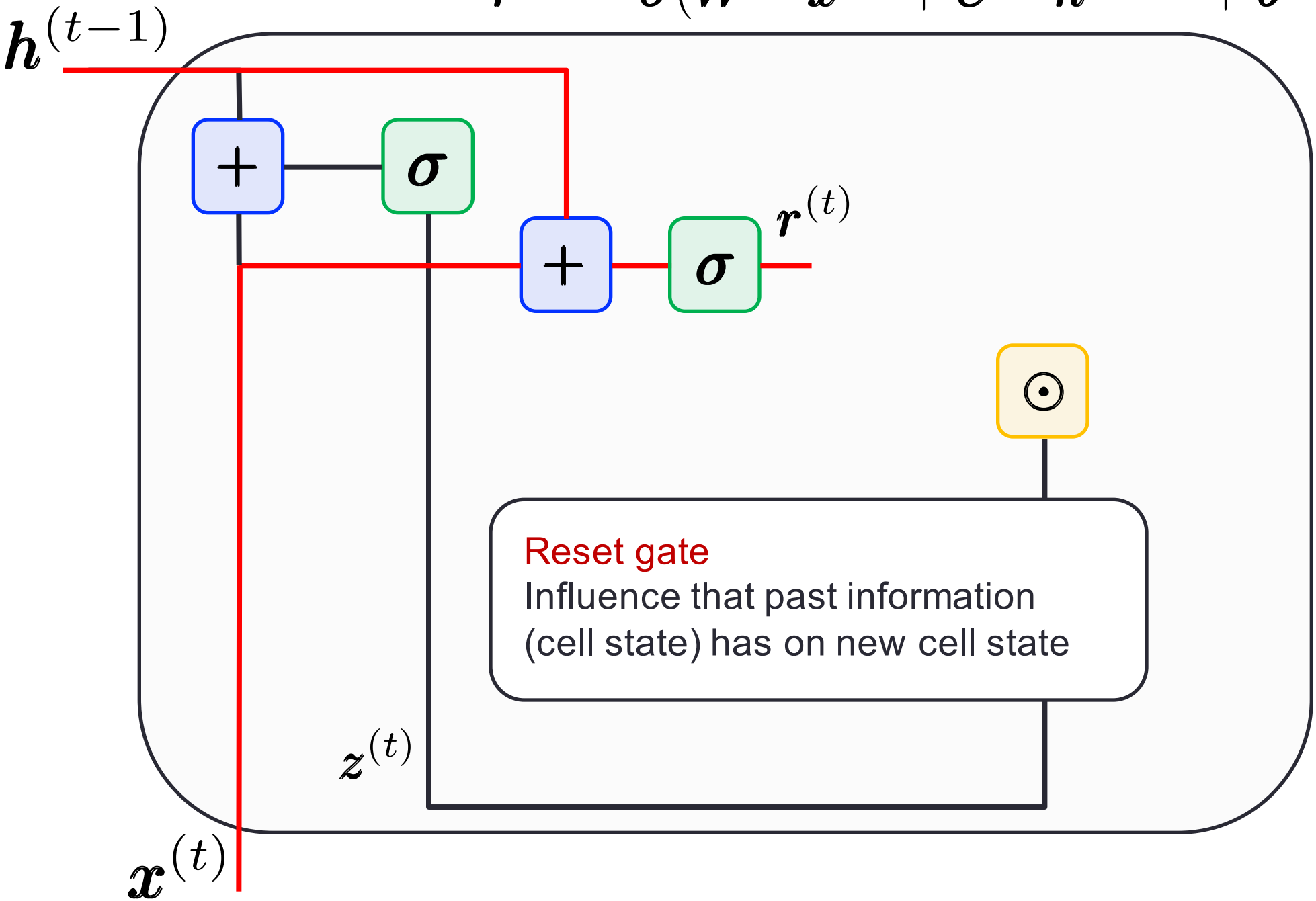


Forget gate

How much of the past information needs to be passed along to the future

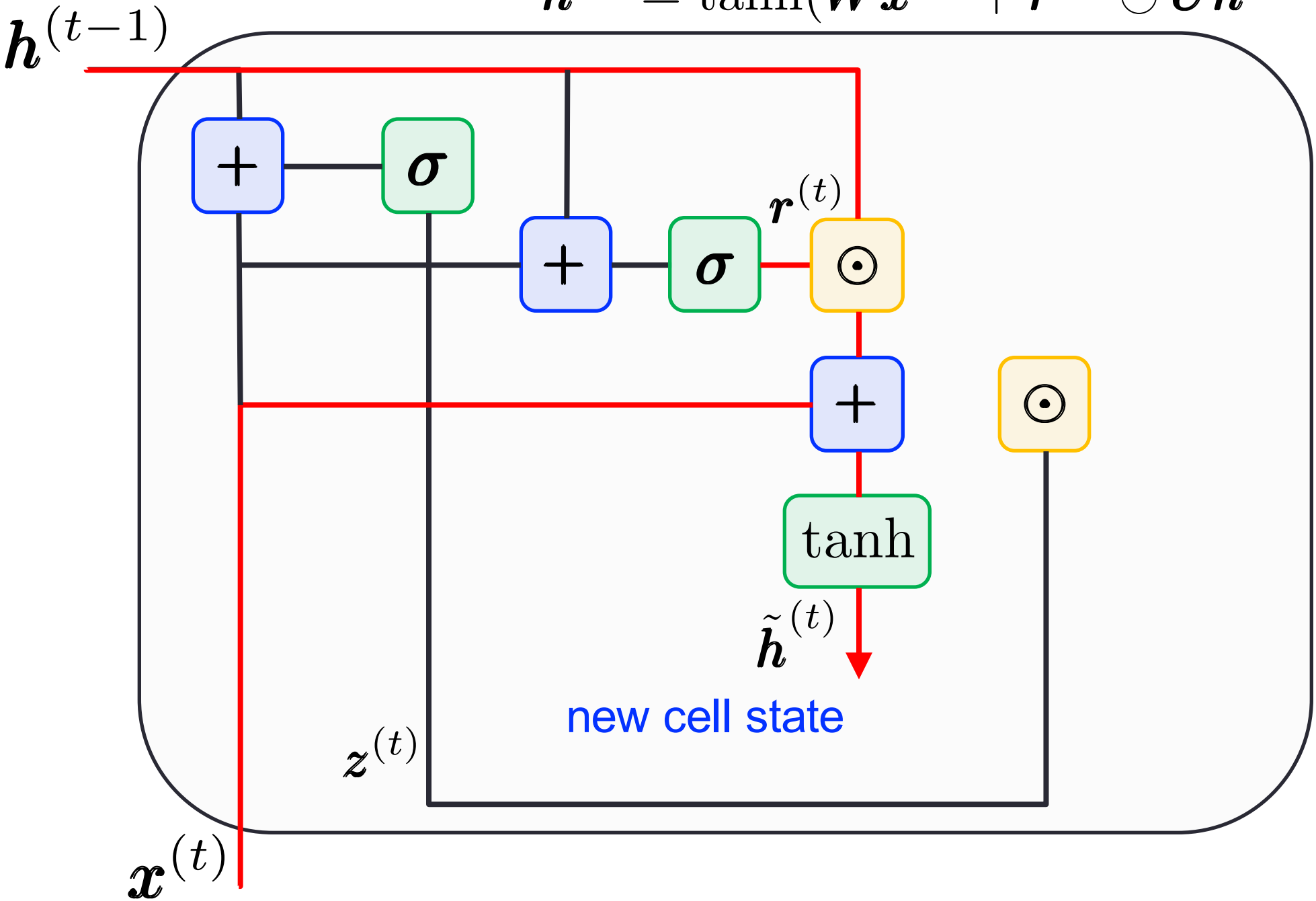
Step 2: reset gate

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



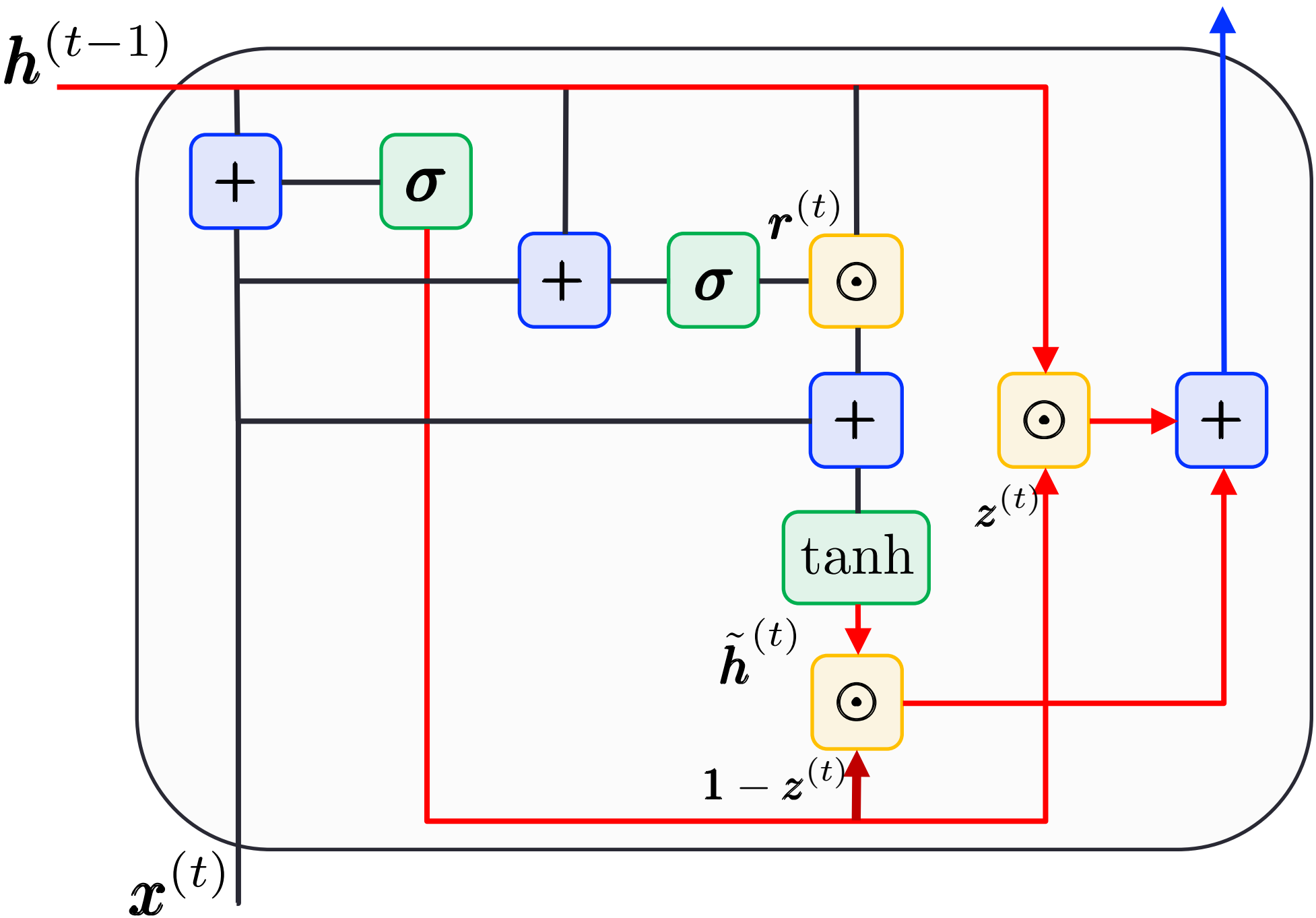
Step 3: internal state

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

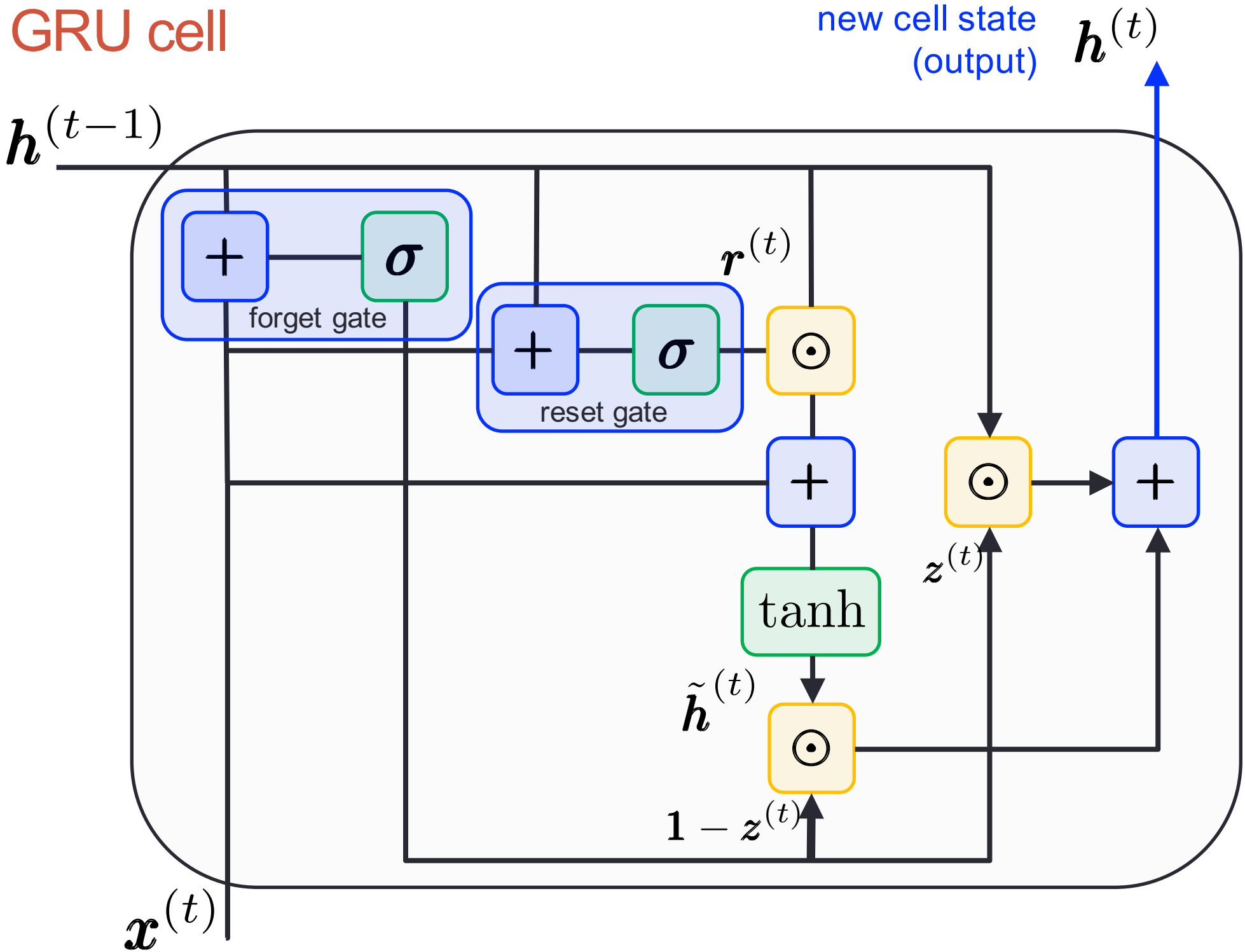


Step 4: final state

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



GRU cell


new cell state
(output)
$h^{(t)}$
$h^{(t-1)}$
$+$
σ
forget gate
$r^{(t)}$
$+$
σ
reset gate
$\odot$
$+$
$\odot$
$+$
$z^{(t)}$
tanh
$\tilde{h}^{(t)}$
$\odot$
$1 - z^{(t)}$
$x^{(t)}$

GRU cell

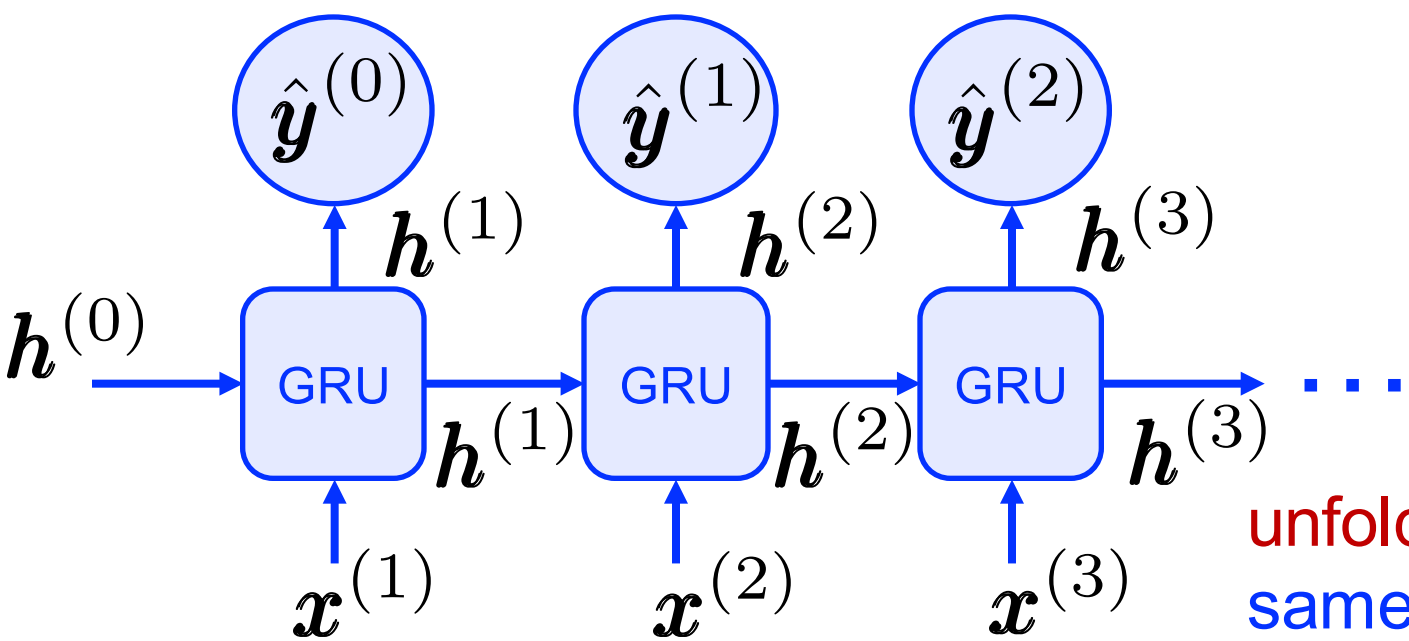
$$\boldsymbol{z}^{(t)} = \boldsymbol{\sigma}(\boldsymbol{W}^{(z)}\boldsymbol{x}^{(t)} + \boldsymbol{U}^{(z)}\boldsymbol{h}^{(t-1)} + \boldsymbol{b}^{(z)})$$

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

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

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

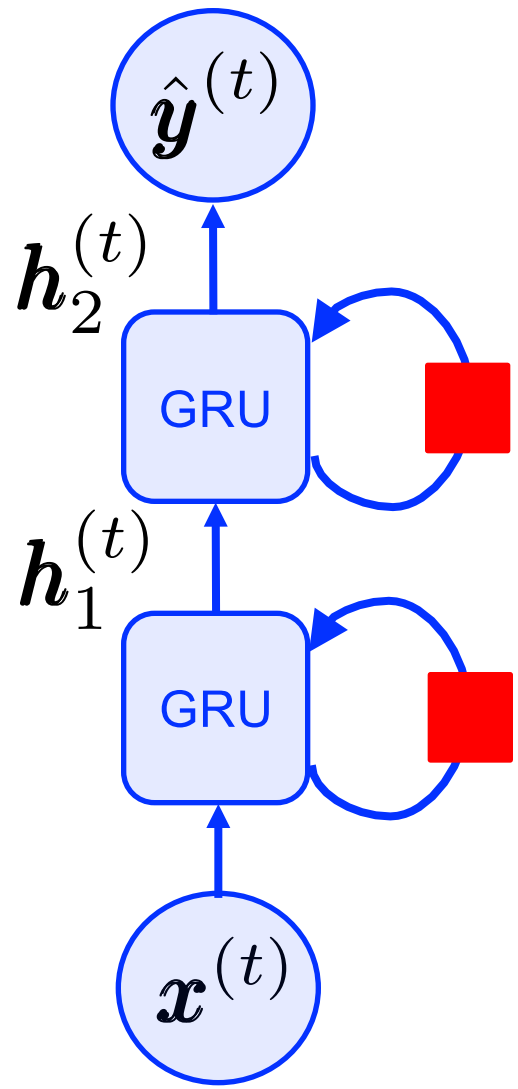
fully gated unit



unfolded graph is the same as simple RNN

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)

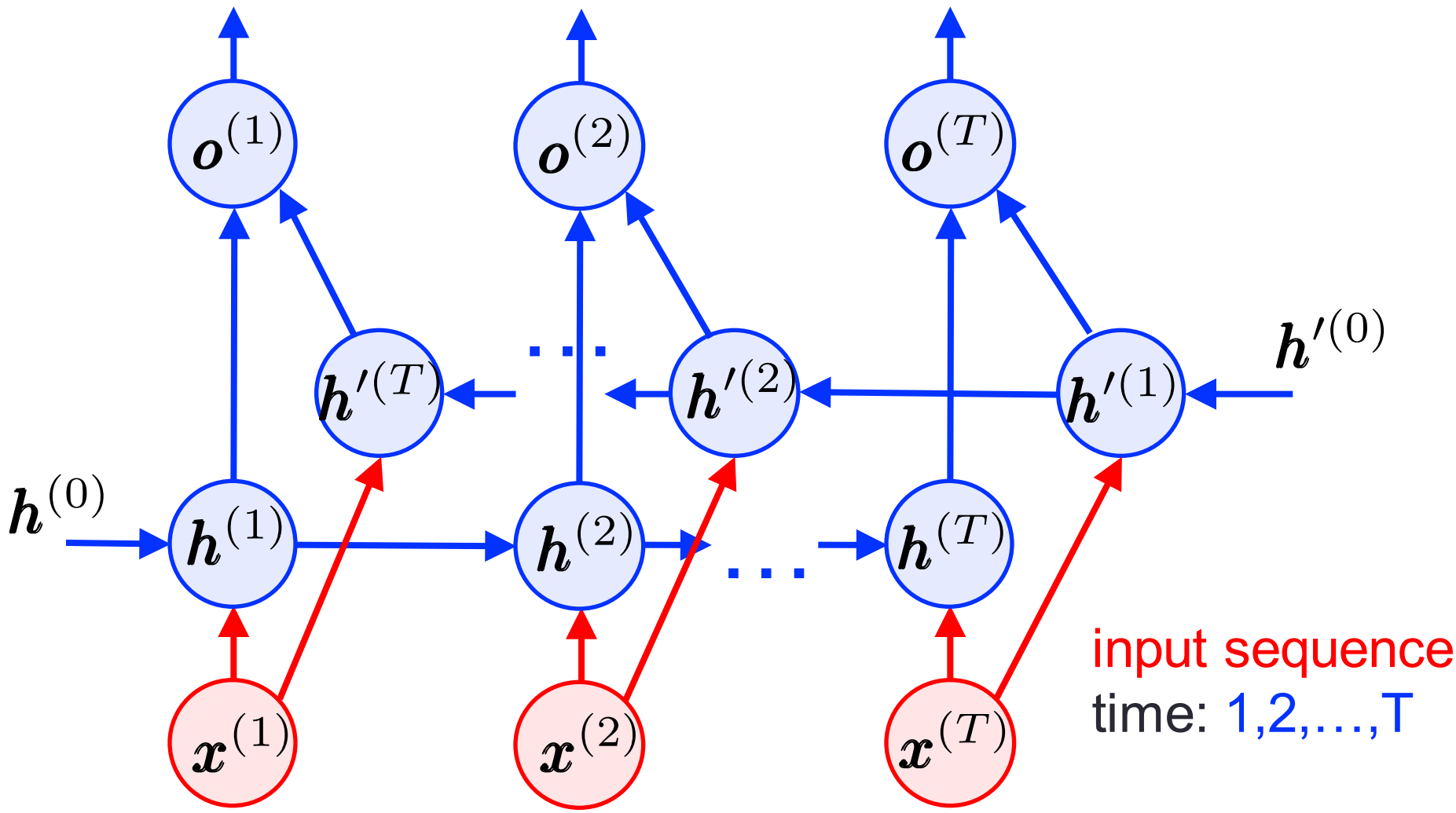
- Advantage
  - Connect present output to *past* and *future* samples
- Example from speech

  "I am really angry as I have been exercising a lot today"

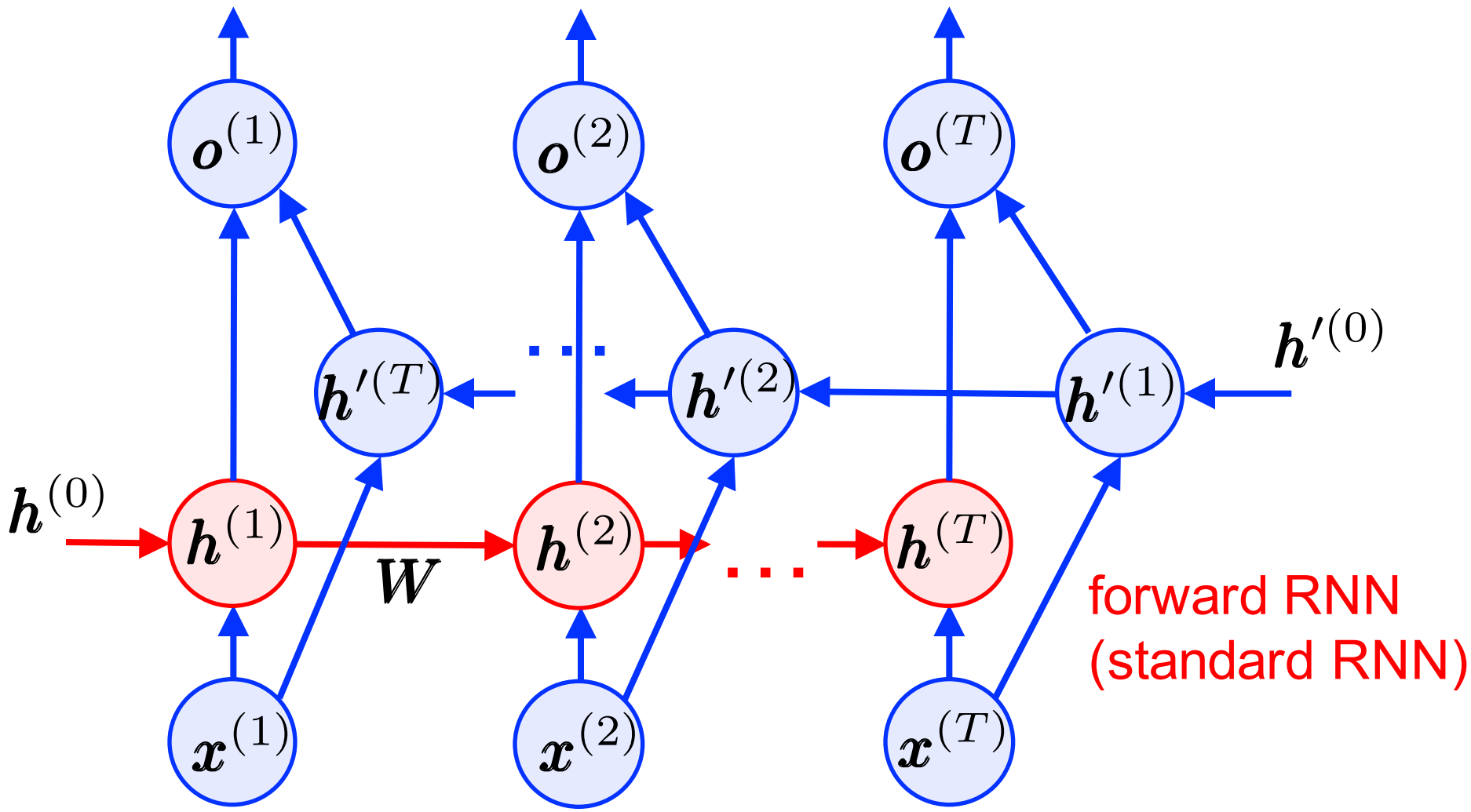
  - Once you have inputted "I am really angry": you are unable to say anything about the correctness of "angry"
  - But if you look forward, through the whole sequence "as I have been exercising a lot today"

  → you realize that *angry is wrong*

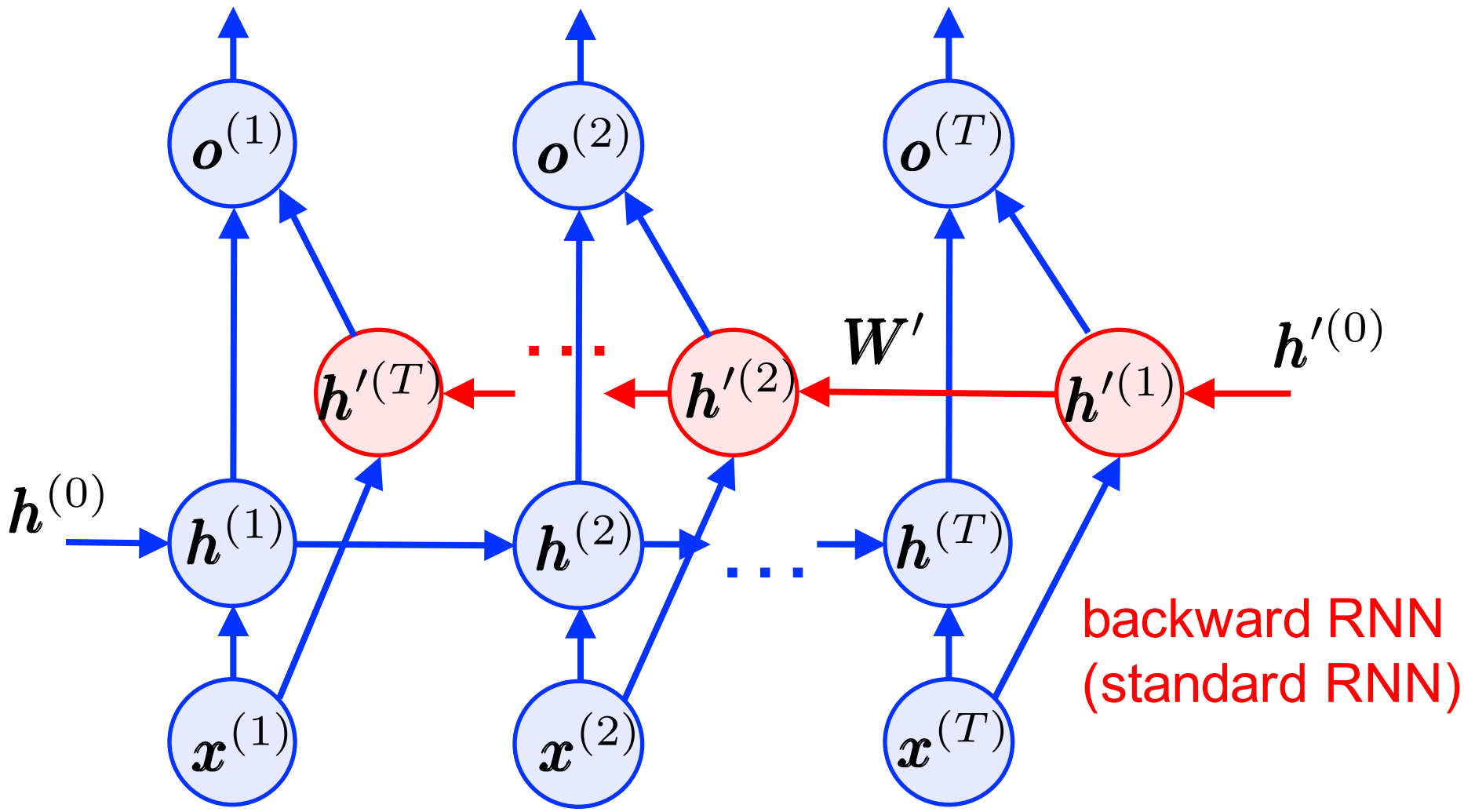
Bi-directional RNN (unfolded)



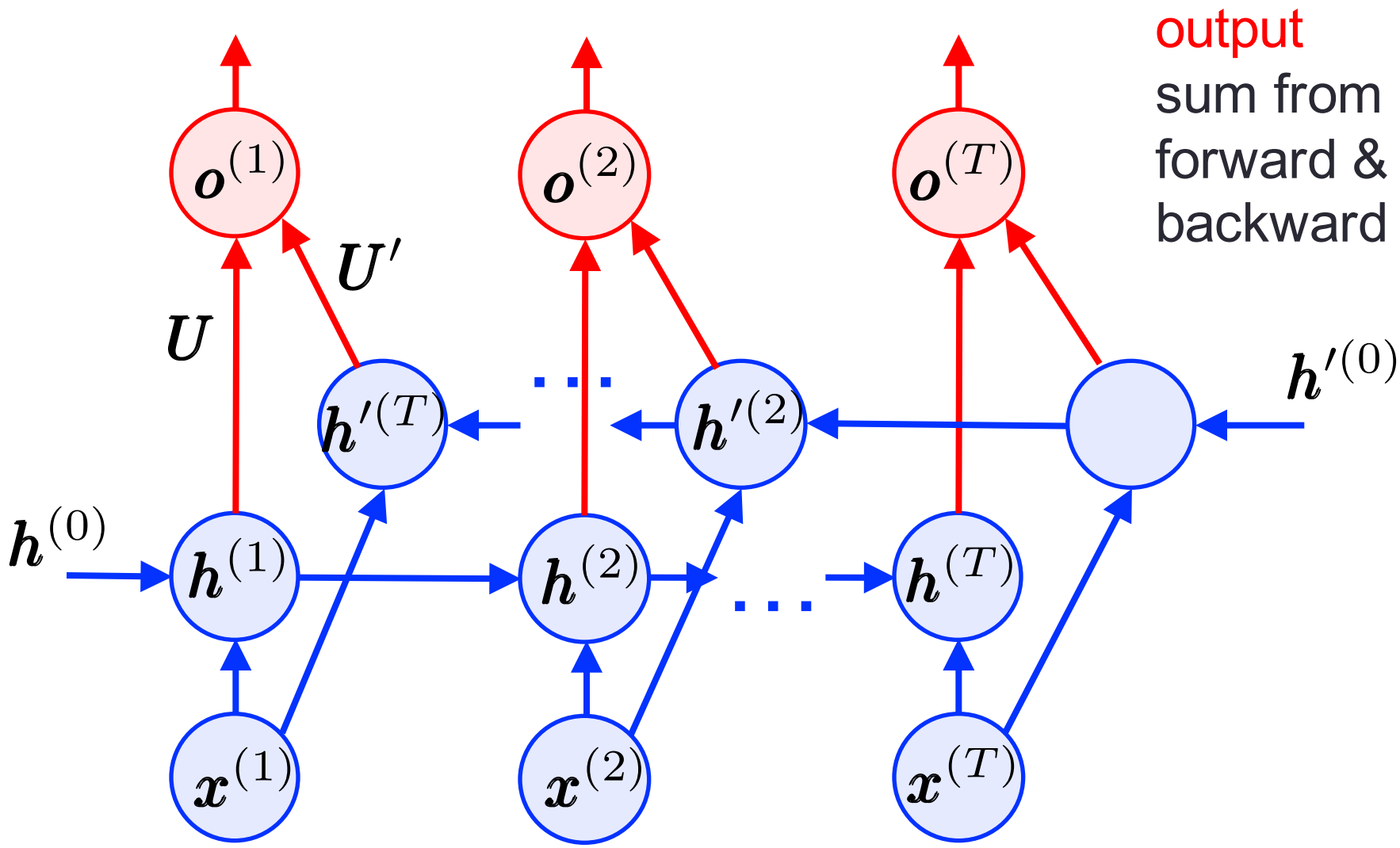
Bi-directional RNN (unfolded)



Bi-directional RNN (unfolded)

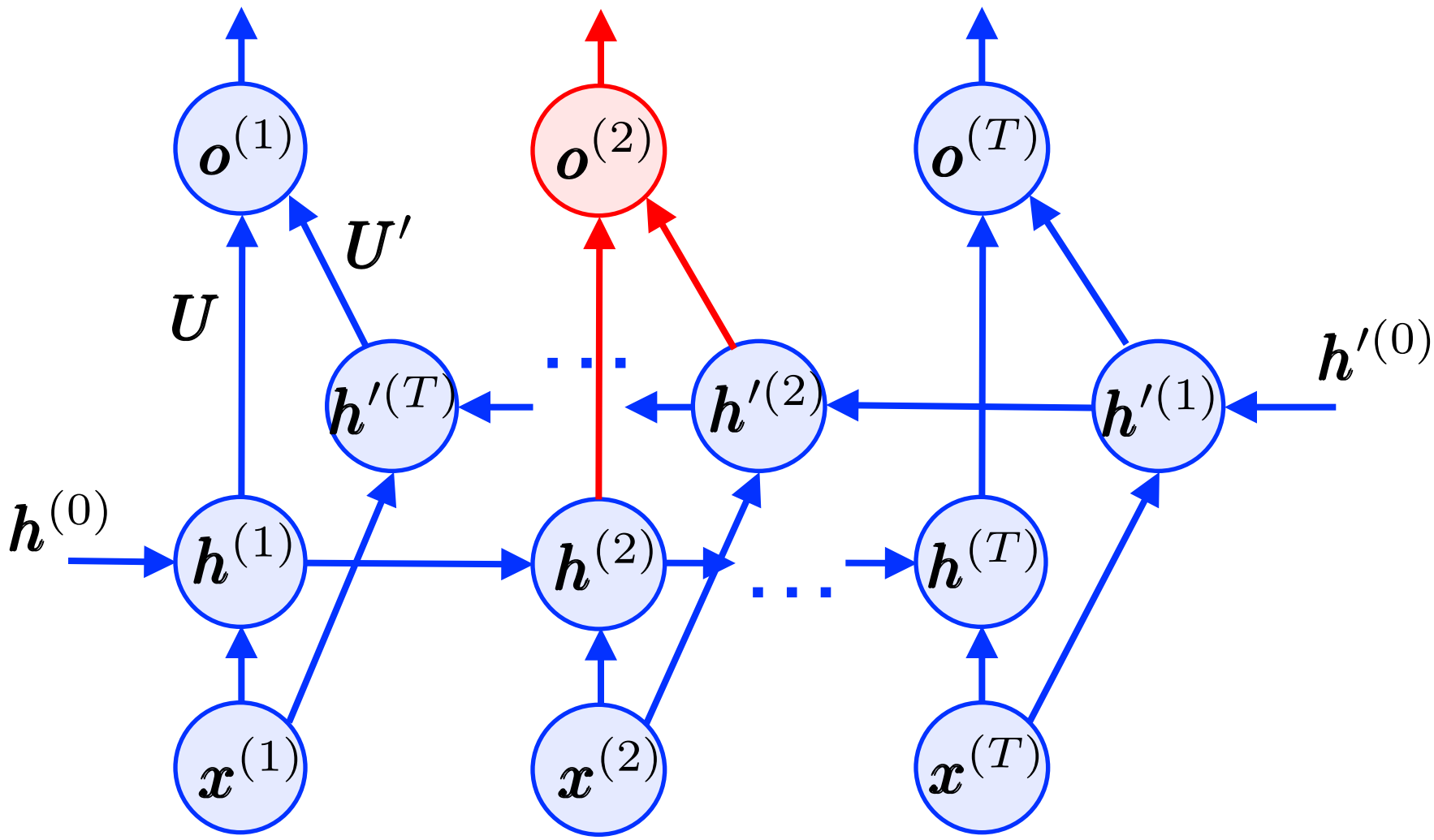


Bi-directional RNN (unfolded)

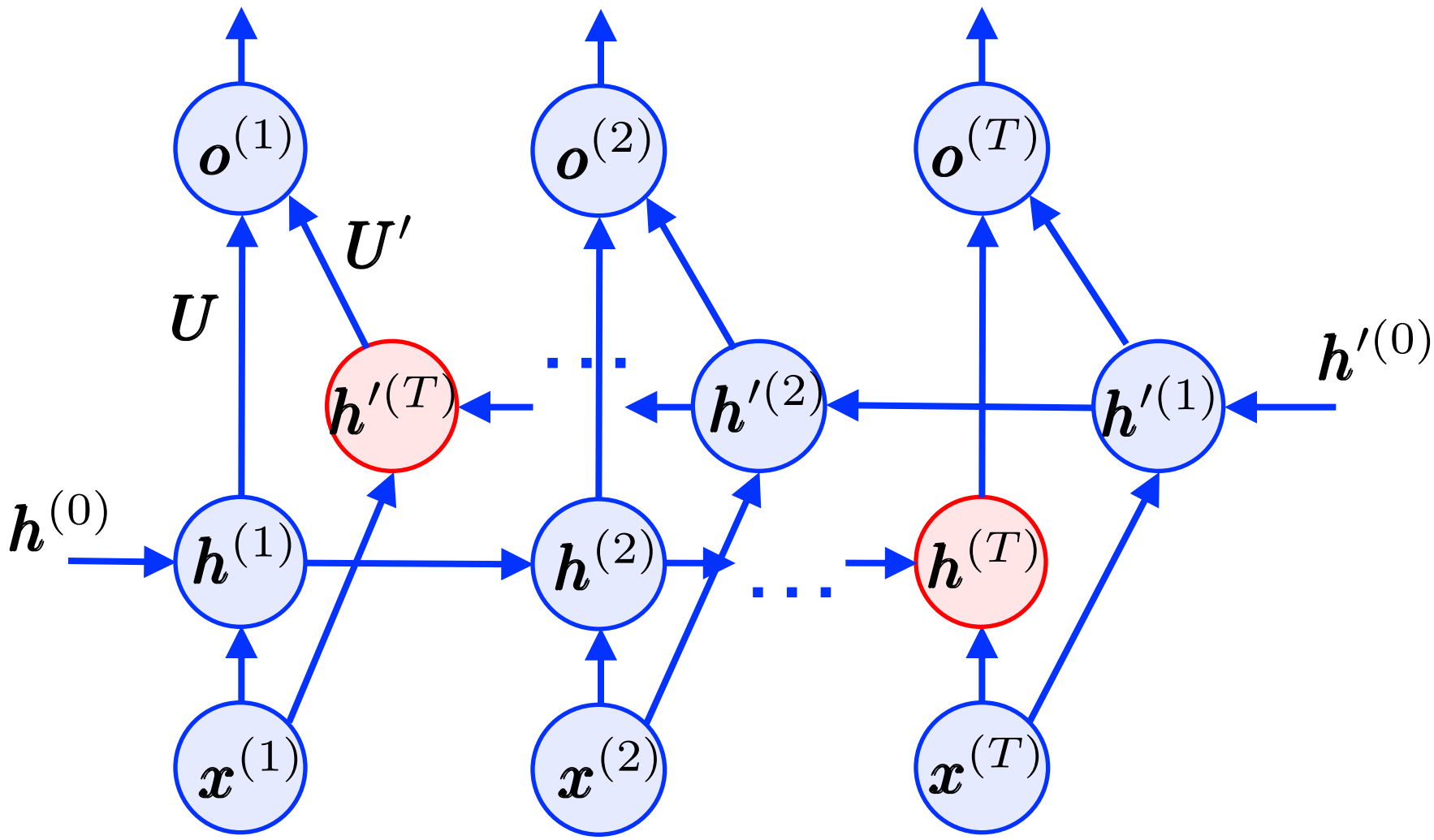


output at generic time i

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

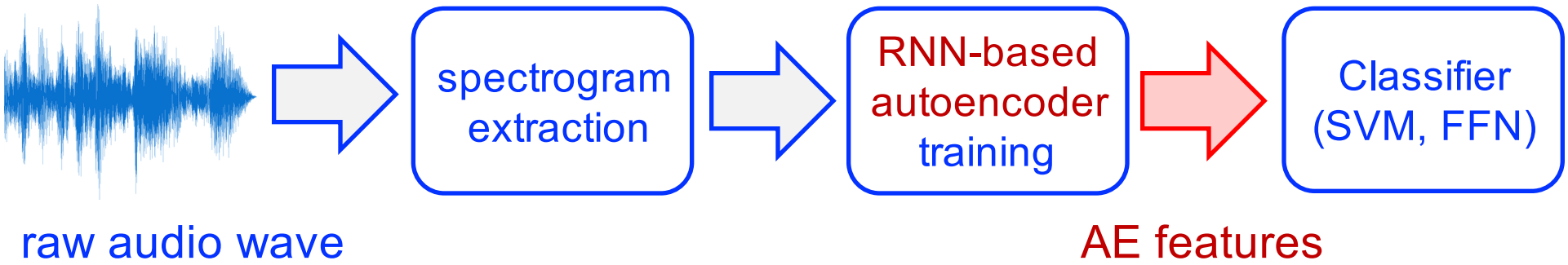


RNN final state



APPLICATION: UNSUPERVISED REPRESENTATION LEARNING OF AUDIO WAVEFORMS

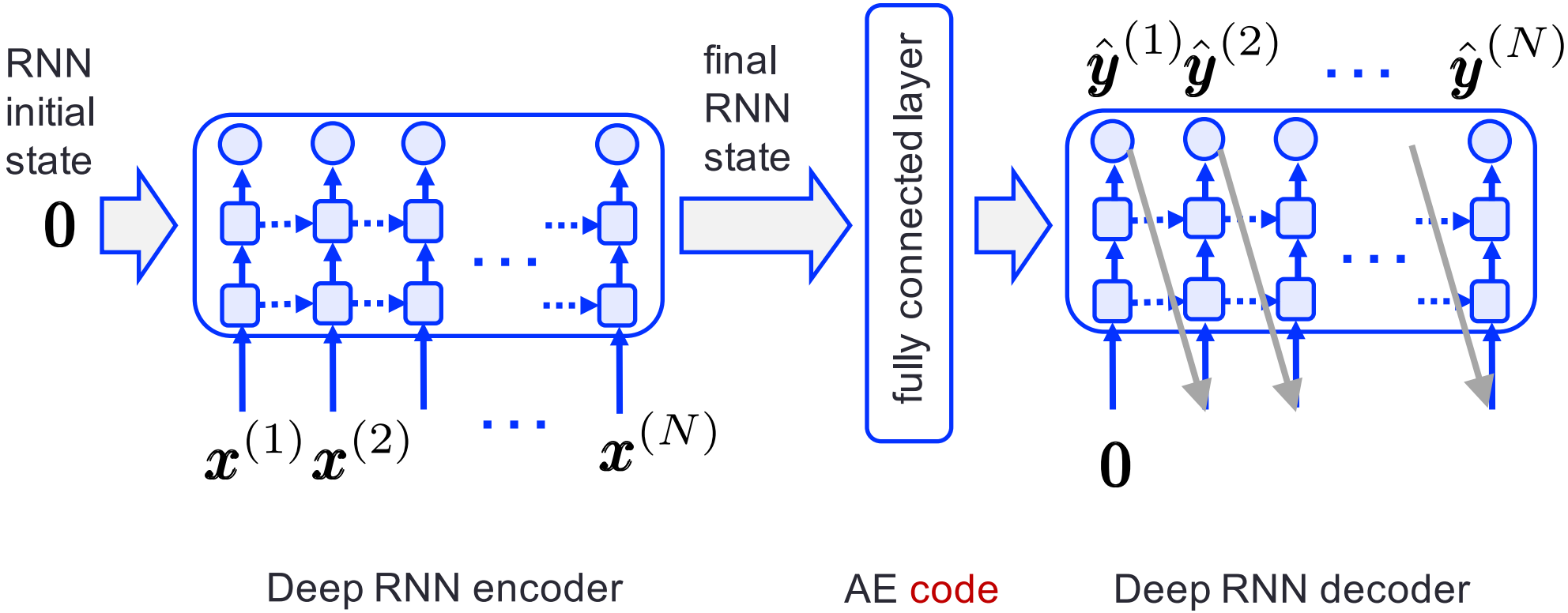
Unsupervised representation of audio



Signal processing pipeline from [Amiriparian17]

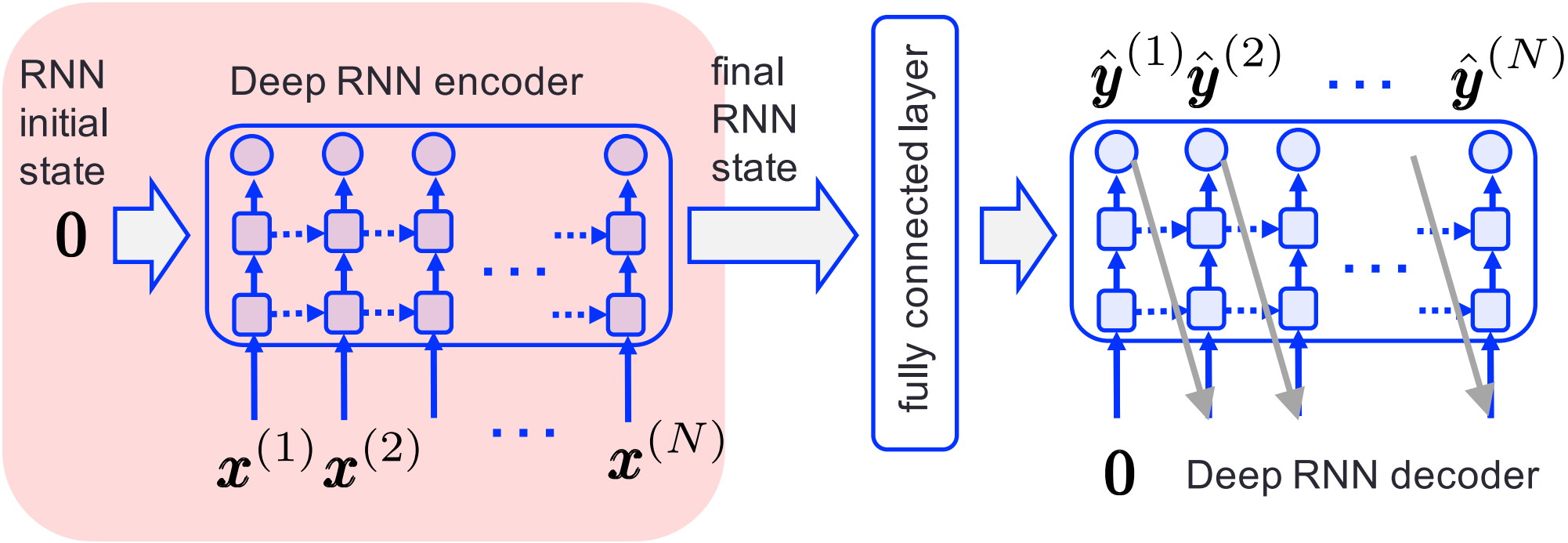
[Amiriparian17] S. Amiriparian, M. Freitag, N. Cummins, B. Schuller, “Sequence to Sequence Autoencoder for Unsupervised Representation Learning from Audio,” Detection and Classification of Acoustic Scenes and Events (DCASE), Munich, Germany, November 2017.

Deep RNN – AE (1/5)



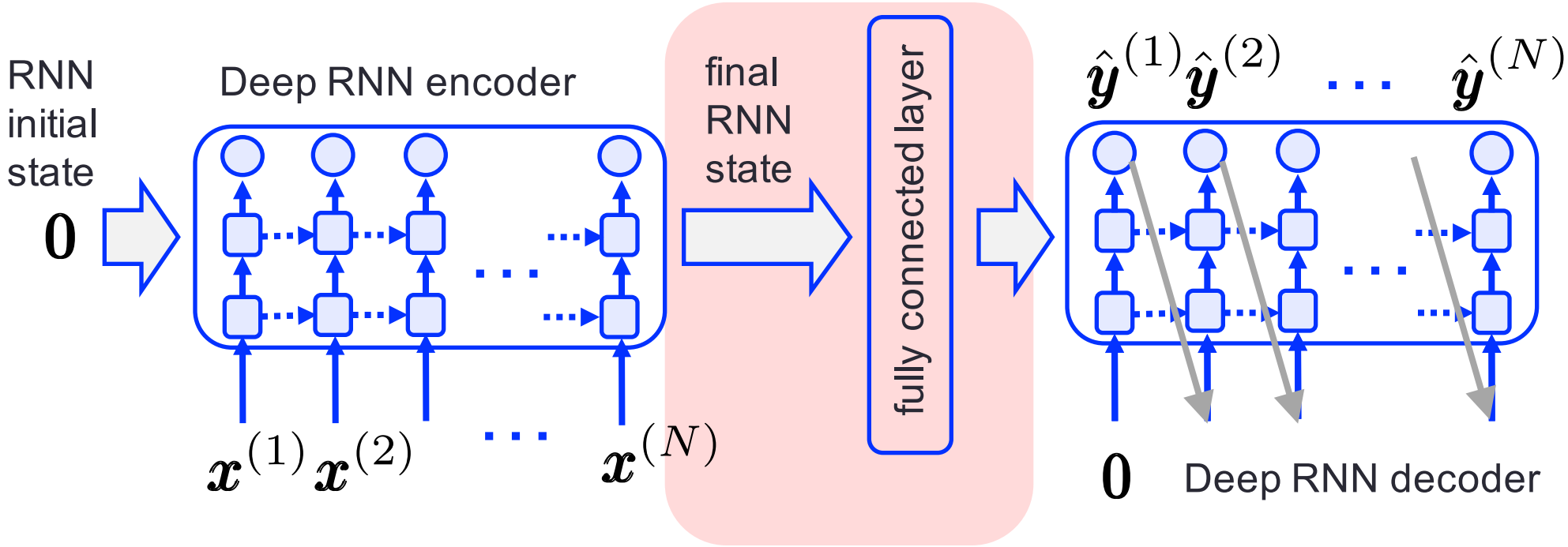
Training objective is: $\hat{\boldsymbol{y}}^{(i)} \simeq \boldsymbol{x}^{(i)}\ ,\ i = 1, 2, \ldots, N$

Deep RNN – AE (2/5)



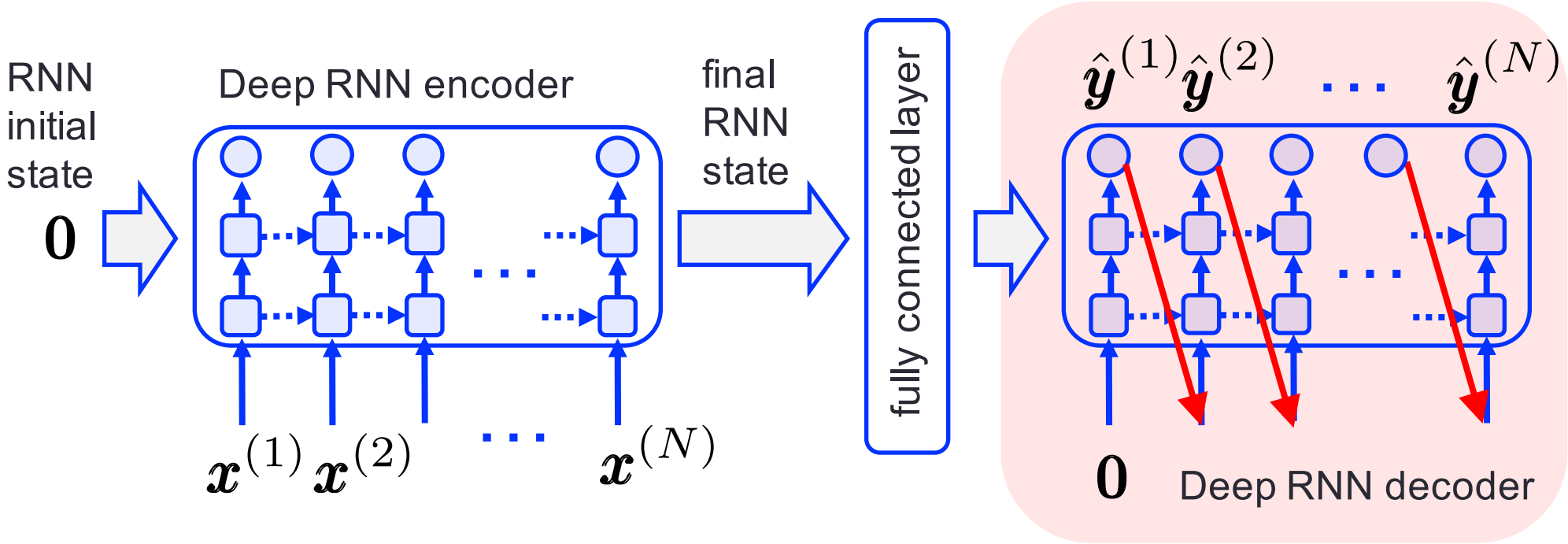
- RNN encoder
  - RNN state → zero vector
  - Input vector sequence has N time steps n=1,2,…,N
  - Input one feature vector at a time from time 1 to time N

Deep RNN – AE (3/5)



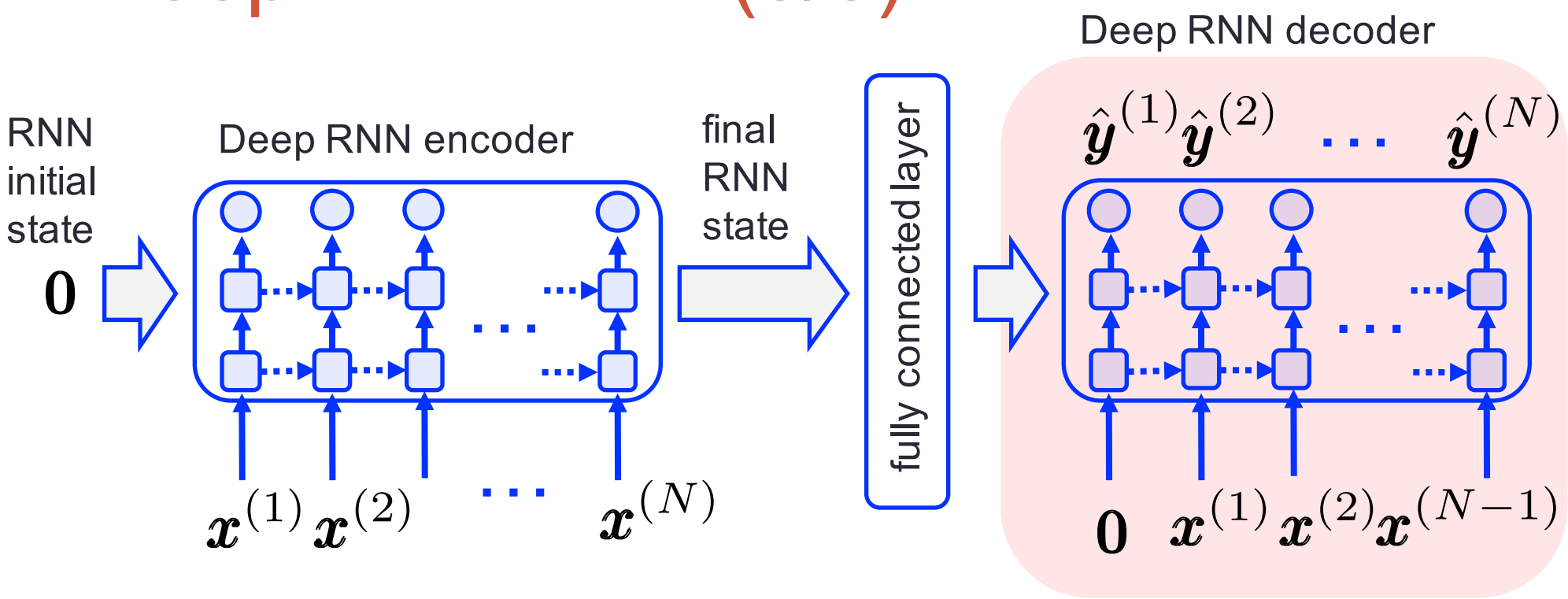
- After N time steps
  - Final RNN state flattened
  - Inputted into a fully connected layer
    - with tanh activation functions

Deep RNN – AE (4/5)



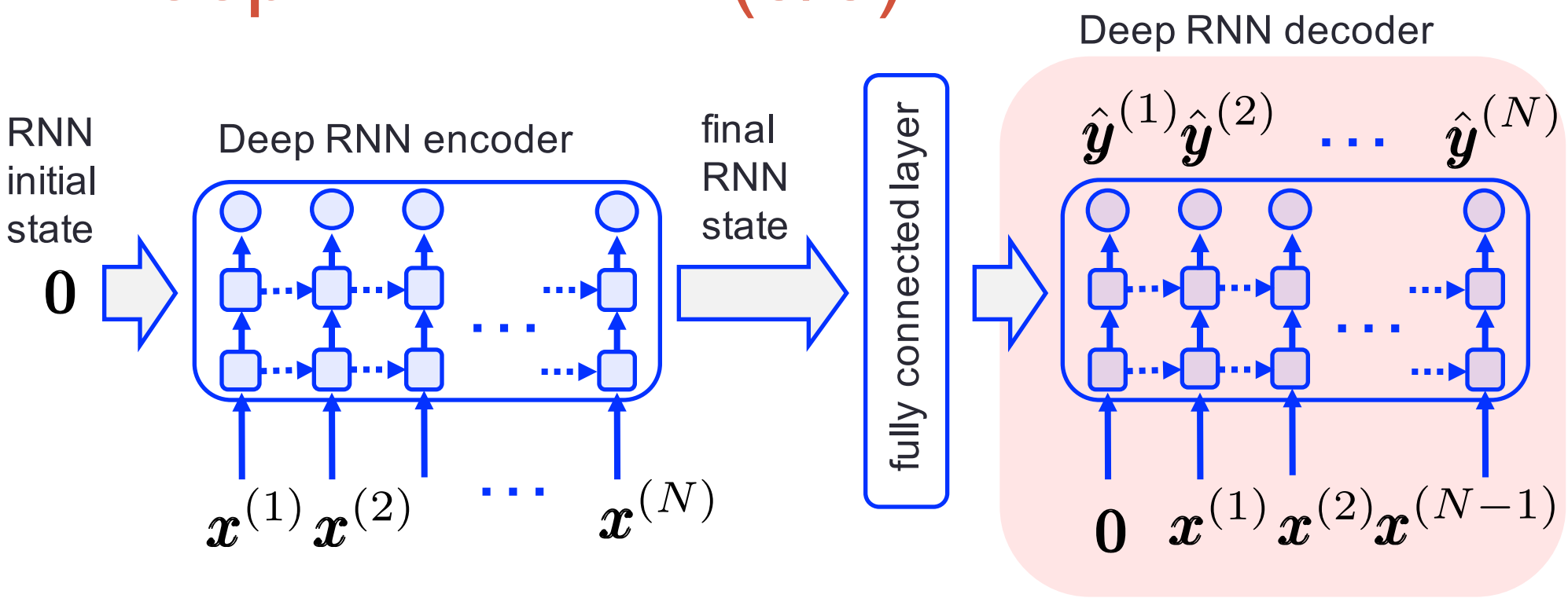
- RNN decoder
  - Initial decoder state = AE code
  - Decoder is a predictor of the input sequence
  - Output from time t-1 used as input for time t
    - to predict output at time t

Deep RNN – AE (5/5)



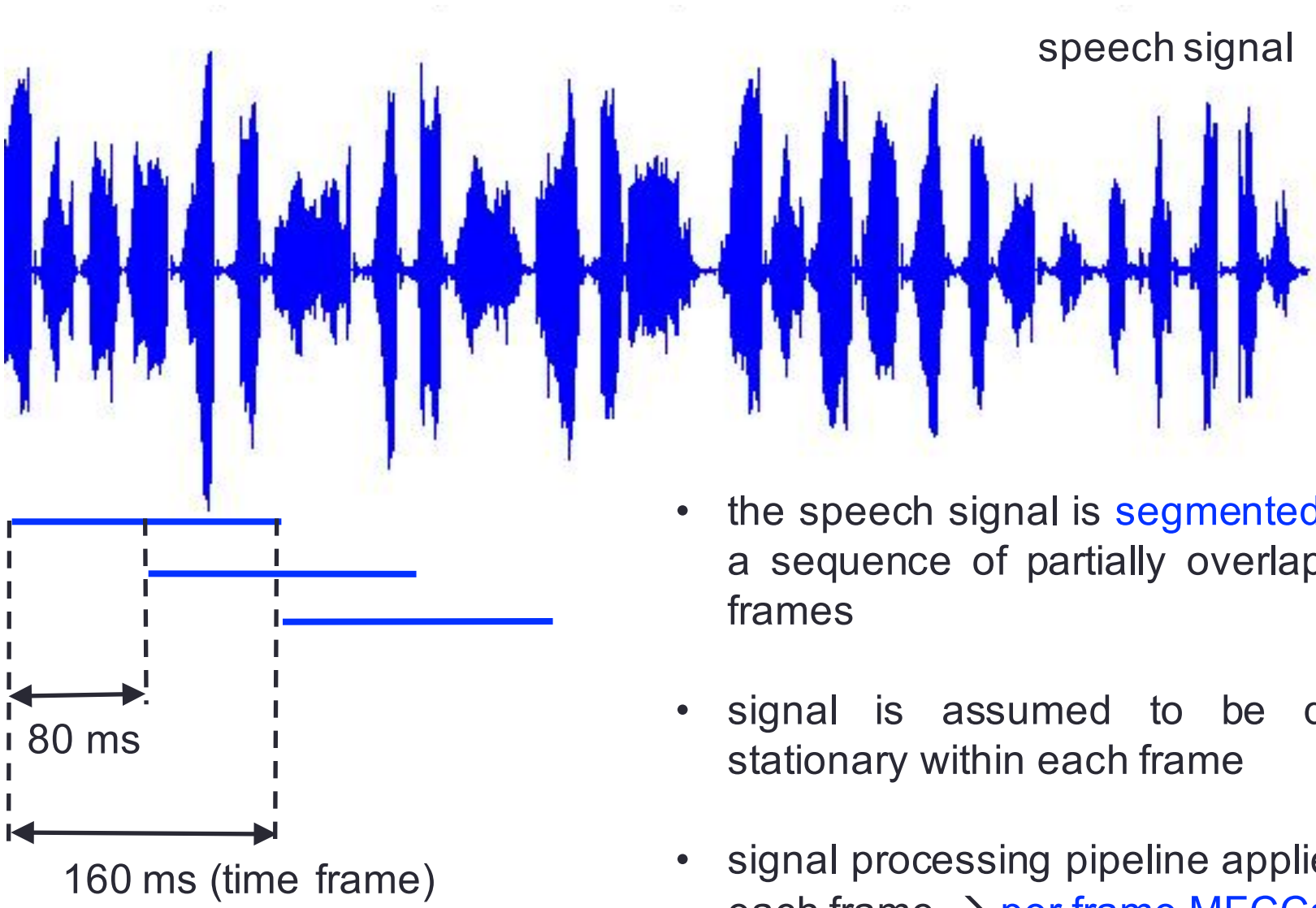
- RNN decoder
  - Trick no. 1: the *shifted* input sequence is used
    - in place of the one outputted by the decoder
    - In [Amiriparian17] it is proven that
      - this leads to faster training
      - performance degradation is negligible

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

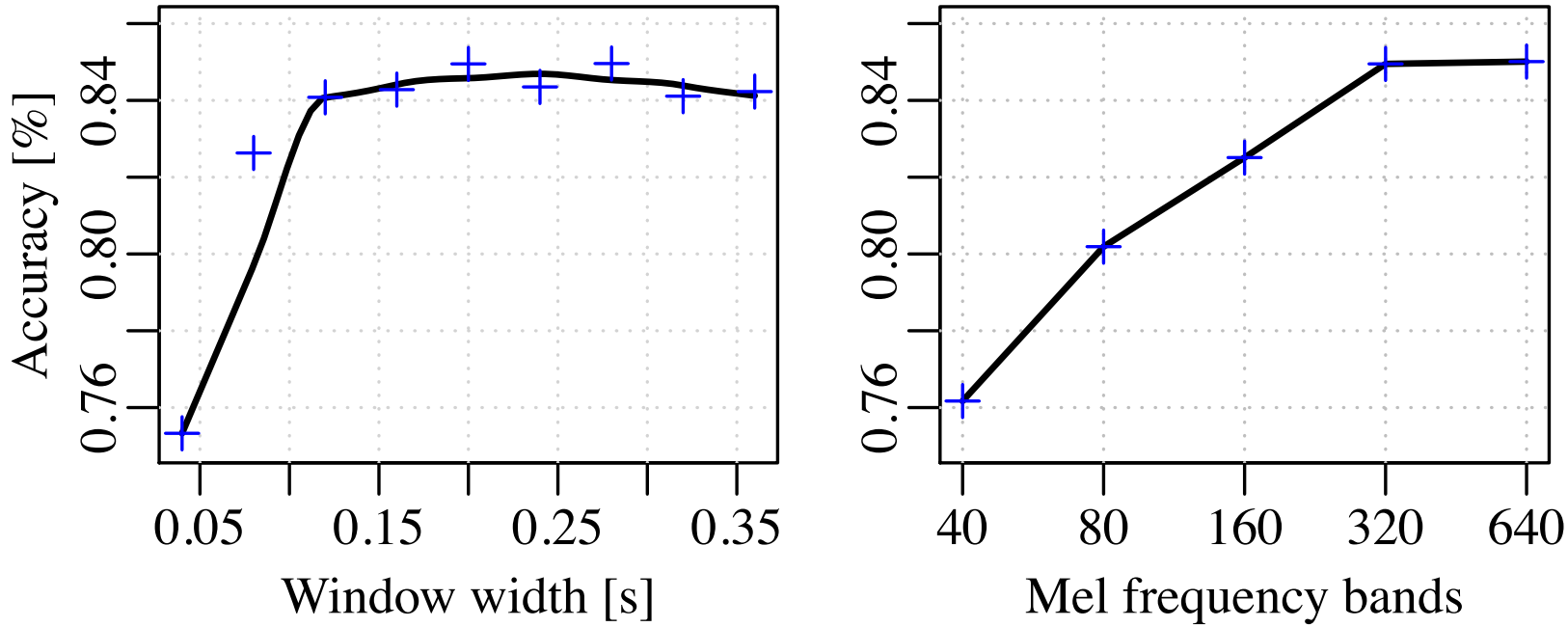
[Mesaros2017] A. Mesaros, T. Heittola, A. Diment, B. Elizalde, A. Shah, E. Vincent, B. Raj, and T. Virtanen, “DCASE 2017 Challenge Setup: Tasks, Datasets and Baseline System,” in Detection and Classification of Acoustic Scenes and Events (DCASE2017), Munich, Germany, Nov 2017.

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

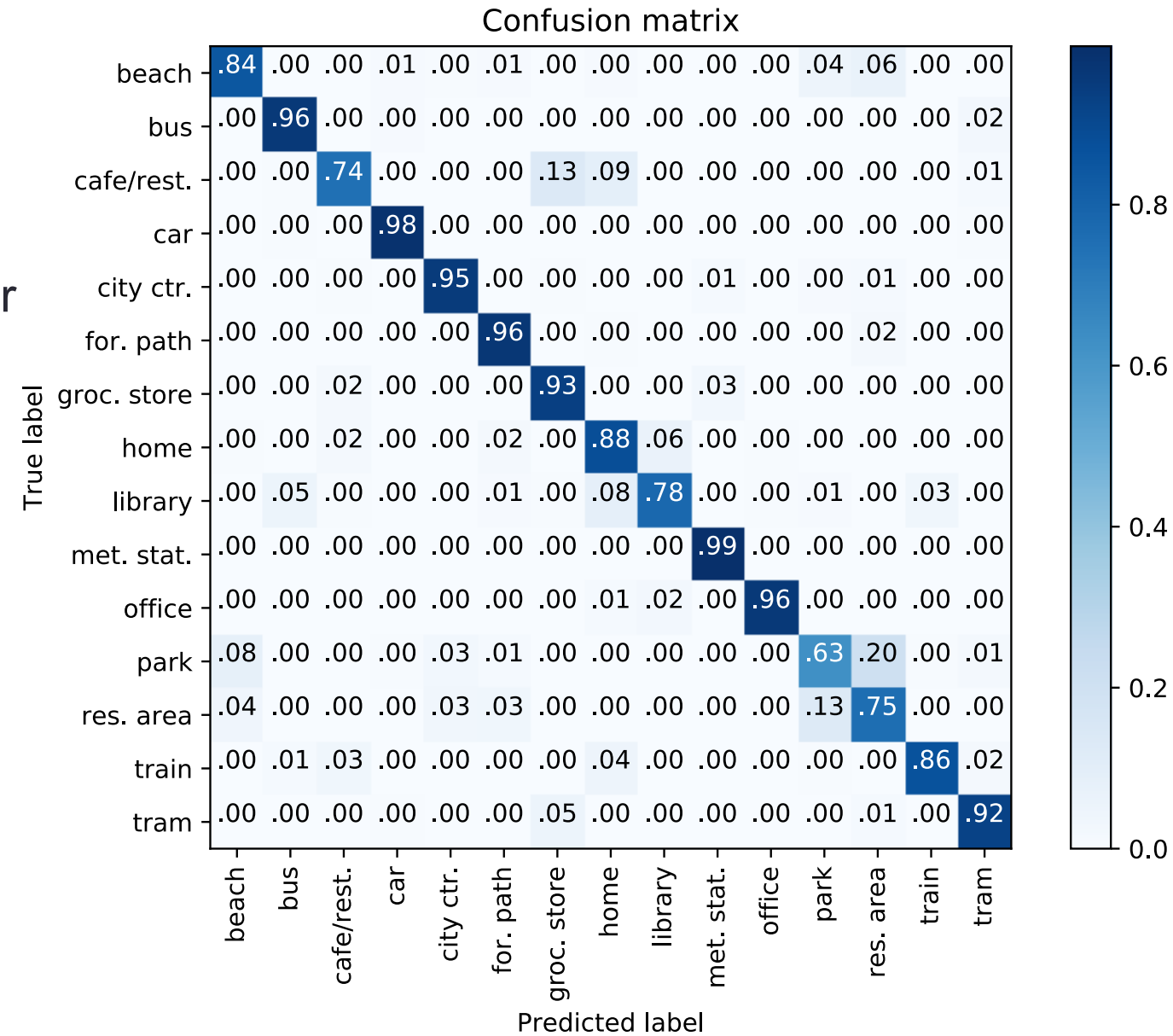


Grid search for preprocessing parameters

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

Confusion matrix

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



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.

Unsupervised clustering:

[Kohonen82] Teuvo Kohonen, “Self-organized formation of topologically correct feature maps,” Biological Cybernetics, vol. 43, pp. 59–69, 1982.

[Fritzke94] Bernd Fritzke, “A Growing Neural Gas Network Learns Topologies,” Proceedings of the 7th International Conference on Neural Information Processing Systems (NIPS), 1994.

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:

[Hooshmand17] M. Hooshmand, D. Zordan, T. Melodia, M. Rossi, ”SURF: subject-adaptive unsupervised ECG signal compression for wearable fitness monitors”, *IEEE Access*, Vol. 5, September 2017.

[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 !!!

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

LEARNING FOR SEQUENTIAL DATA: TOOLS AND APPLICATIONS

Michele Rossi
rossi@dei.unipd.it

Dept. of Information Engineering
University of Padova, IT

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE