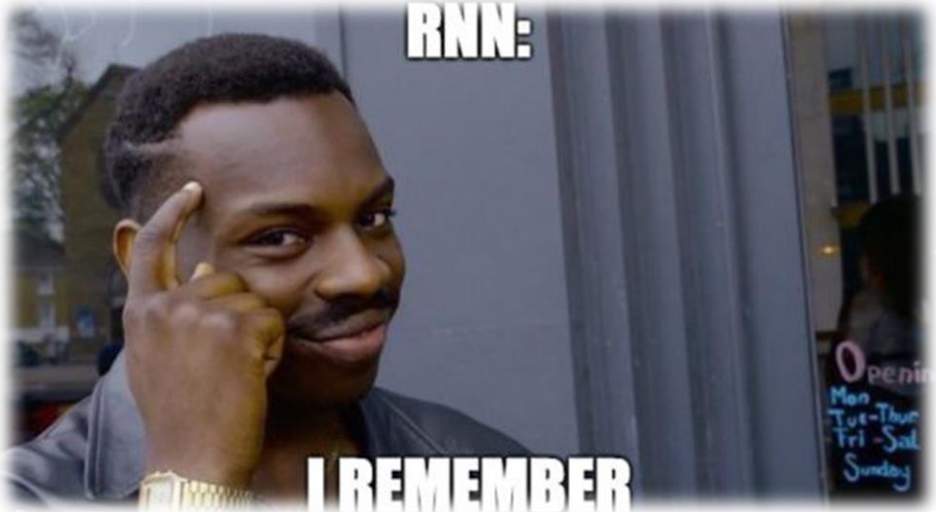
Recurrent Neural Networks

Alberto TONDA, Ph.D. (Senior permanent researcher, DR)

UMR 518 MIA-PS, INRAE, AgroParisTech, Université Paris-Saclay
UAR 3611, Institut des Systèmes Complexes de Paris Île-de-France

Outline

- Dynamical systems
- What is a Recurrent Neural Network (RNN)?
- First architectures and issues
- Long-Short-Term Memory Networks (LSTM)



Dynamical systems

- Systems where the output does not depend only on input
  - But also on a history of *previous inputs*
  - Typical of **time-series** analysis and forecasting
  - But in general, any type of sequence

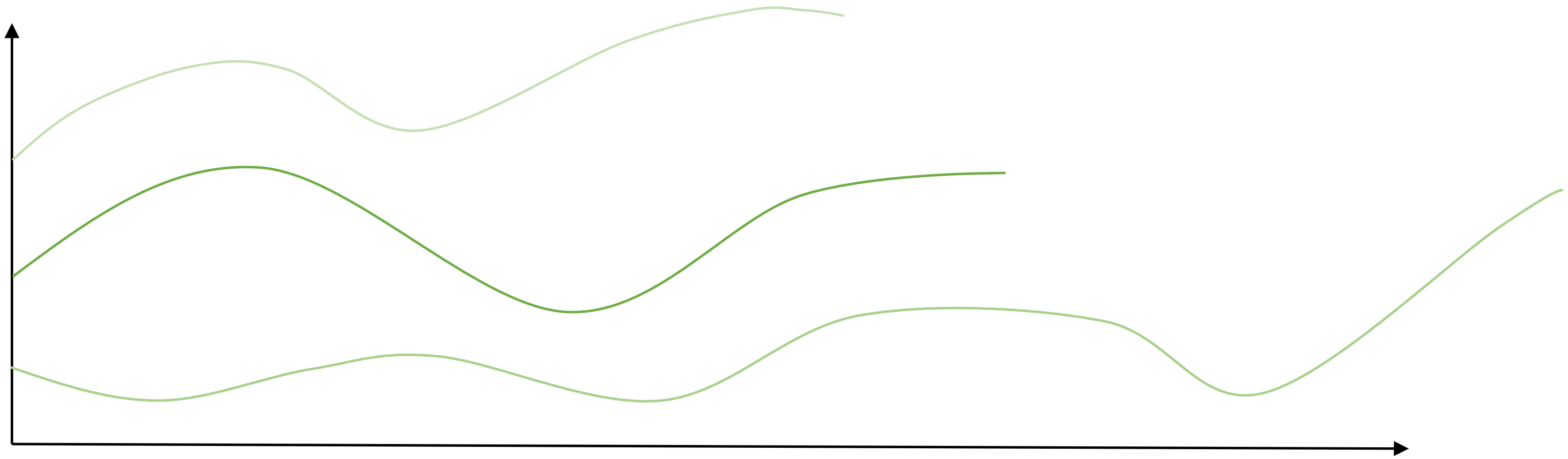
- $x_{t+1} = f(x_t, h_t)$, with $h_t = g(x_0, x_1, \dots, x_{t-1})$
- h_t can also be called **state**, **hidden state**, history of system

Dynamical systems

- Alternatives to computing and storing h_t?
- Autoregressive models
  - Assumption: $x_{t+1} = f(x_t, h_t)$, but $h_t = g(x_{t-1}, x_{t-2}, \dots, x_{t-N})$
  - In other words, $x_{t+1} = f(x_t, x_{t-1}, x_{t-2}, \dots, x_{t-N})$
- They might look simple, but good performance
  - Auto-Regressive Moving Average (ARMA)
  - Auto-Regressive Integrated Moving Average (ARIMA)

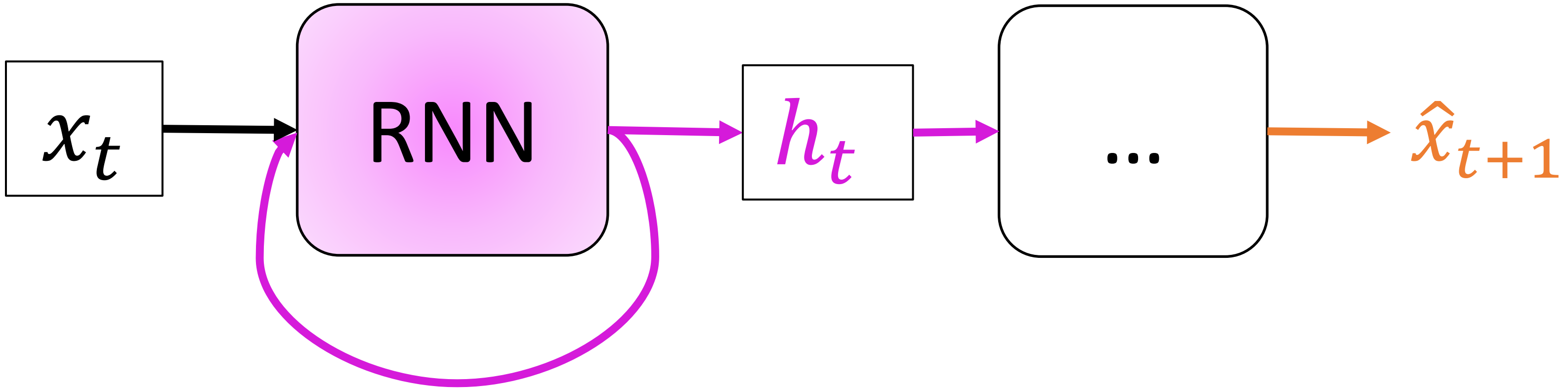
Dynamical systems

- For neural networks, this could be an issue
  - So far, input tensors for an application had the **exact same shape**
  - *Variable number of steps* in sequences for same application
  - For example, how could a CNN deal with a variable-sized input?



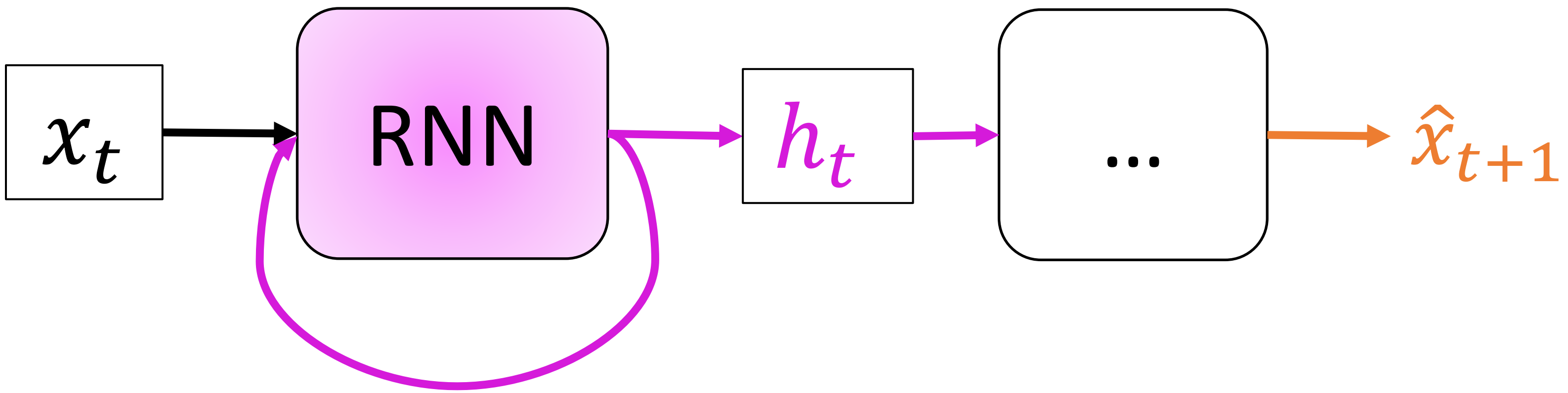
Recurrent Neural Networks

- Interestingly, first ideas are from 1925 (!)
  - “Ising model” (Lenz and Ising), model of magnetism (no learning)
  - Shin’ici Amari, 1972, version with adaptable weights
  - Popularized by John Hopfield in 1982, “Hopfield Networks”



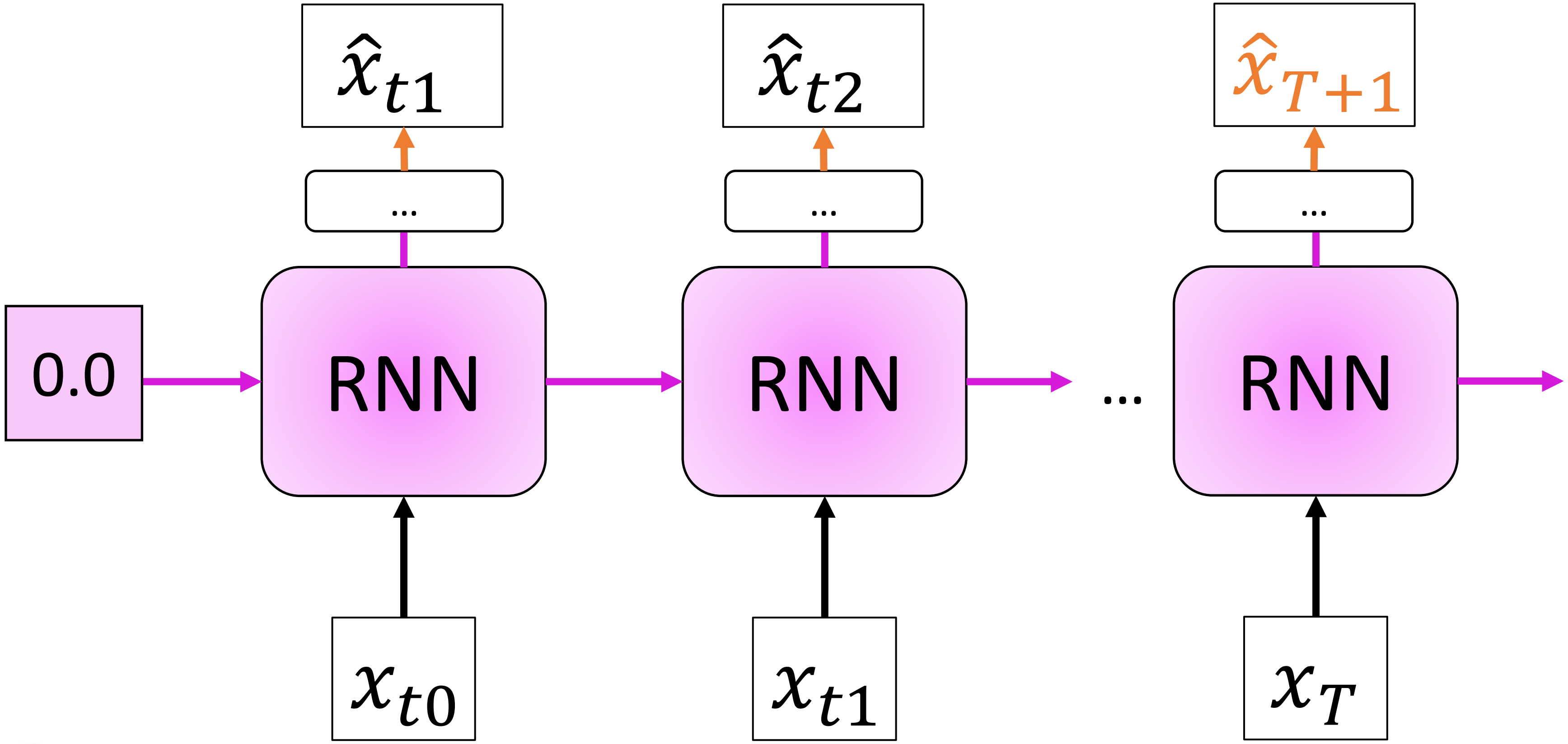
Recurrent Neural Networks

- The architecture is conceived to predict $x_{t+1} = f(x_t, h_t)$
  - RNN unit is designed to compute $h_t = f(x_t, h_{t-1})$
  - Other modules can later obtain $x_{t+1} = f(x_t, h_t)$



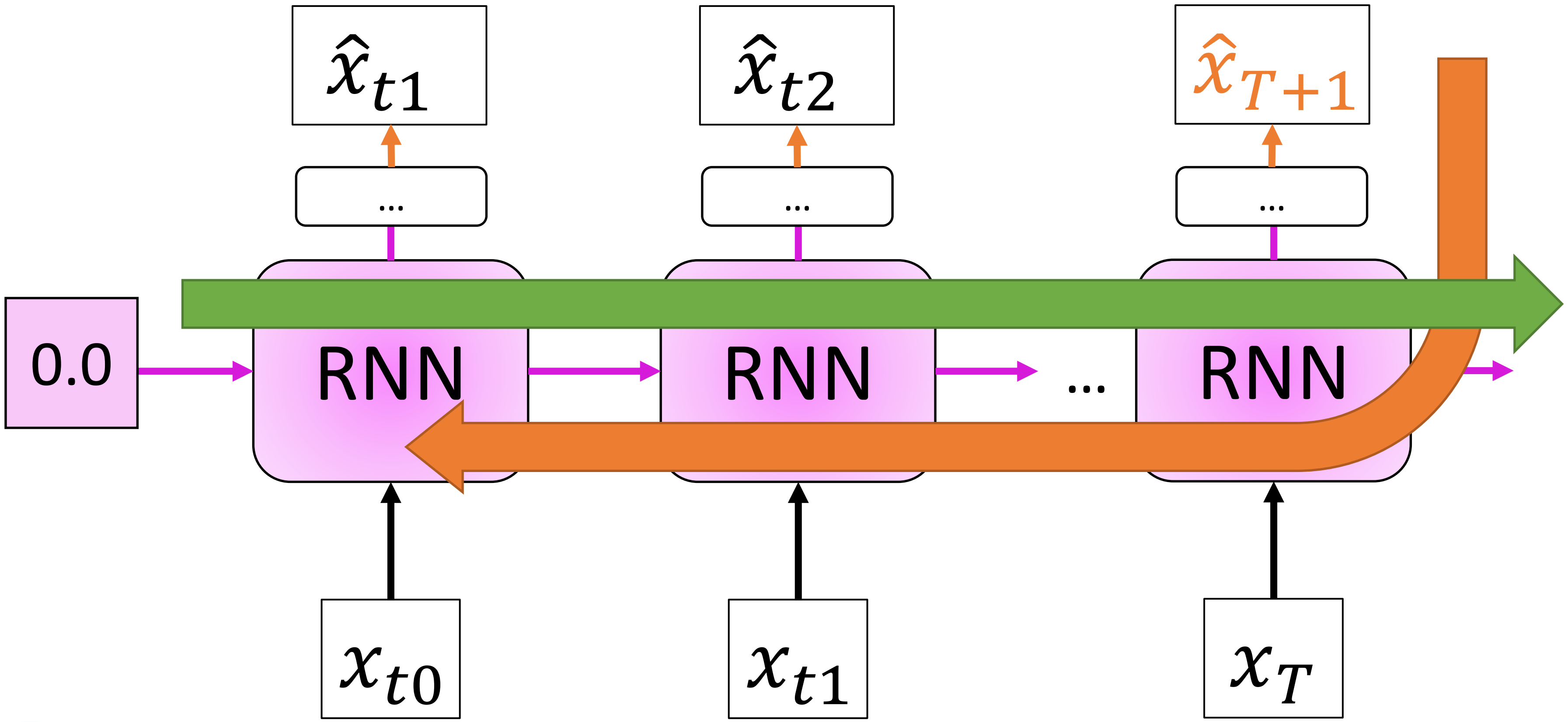
Recurrent Neural Networks

- How is a RNN trained? Unrolling!



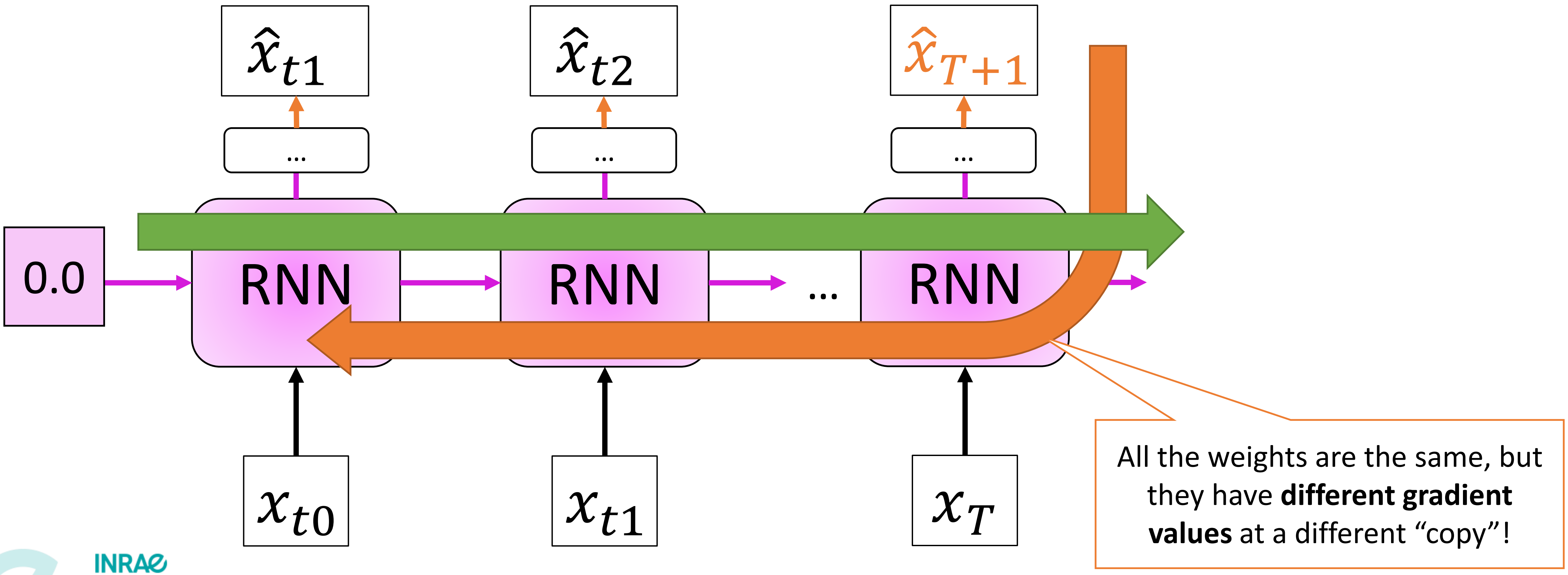
Recurrent Neural Networks

- How is a RNN trained? Unrolling!



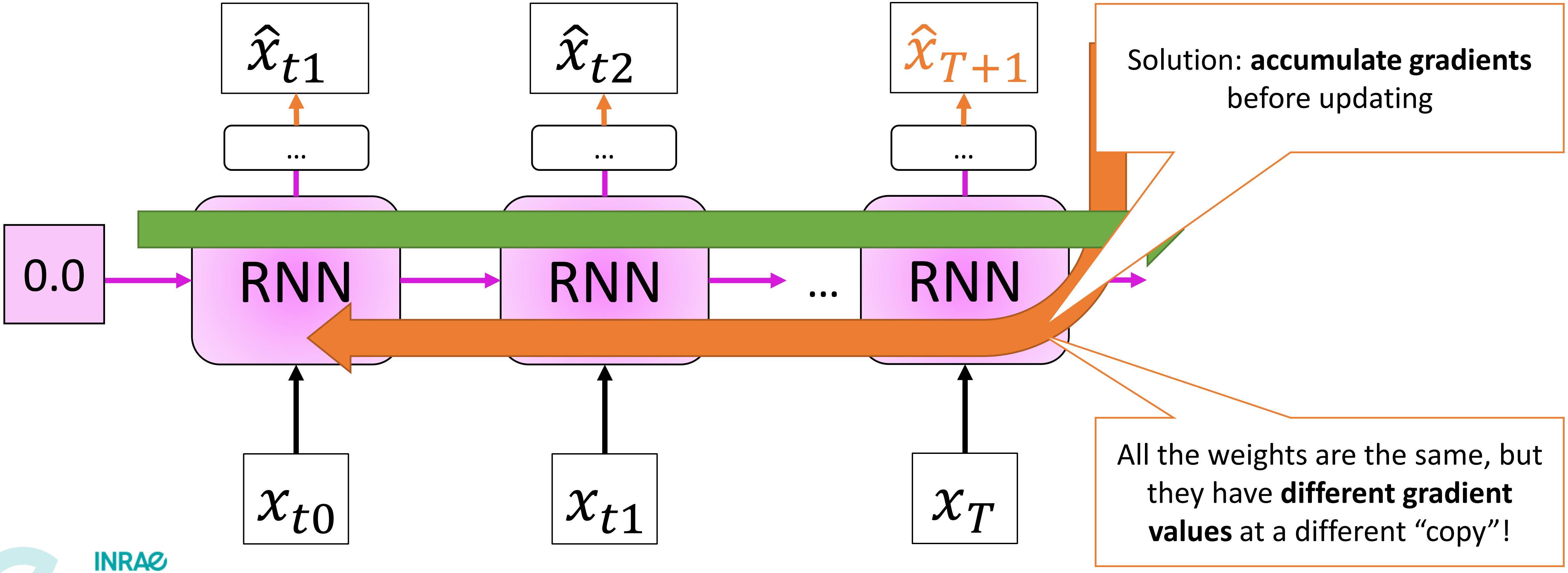
Recurrent Neural Networks

- How is a RNN trained? Unrolling!

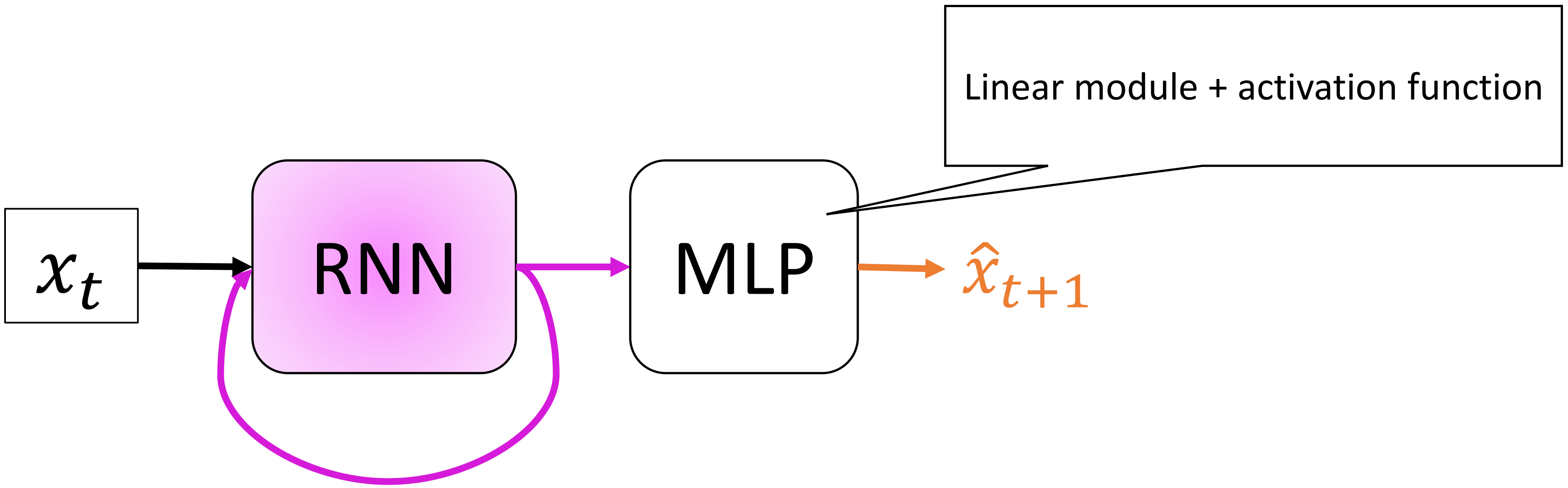


Recurrent Neural Networks

- How is a RNN trained? Unrolling!



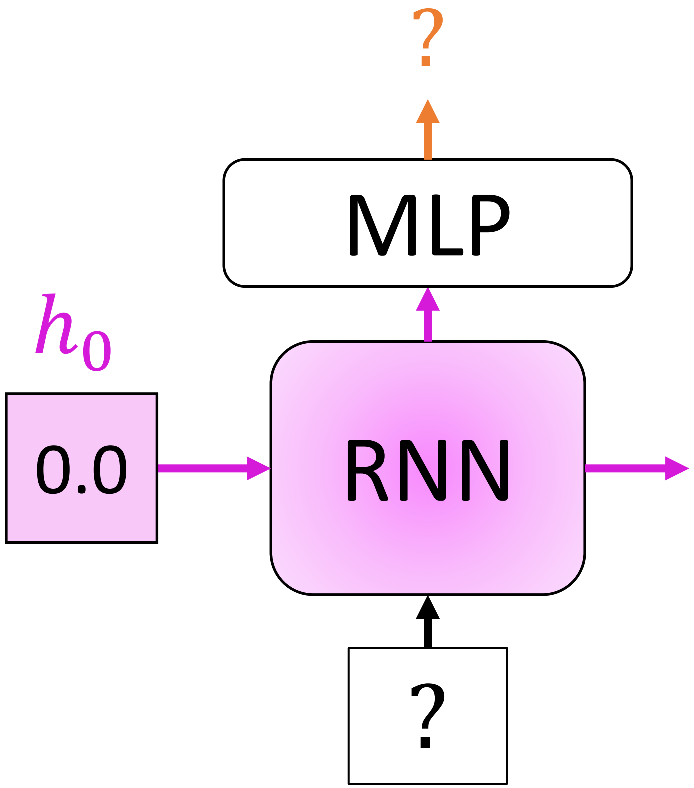
Recurrent Neural Networks



Input sequence x_{t0} | x_{t1} | … | x_T

Target $\hat{x}_{T+1}$

Recurrent Neural Networks



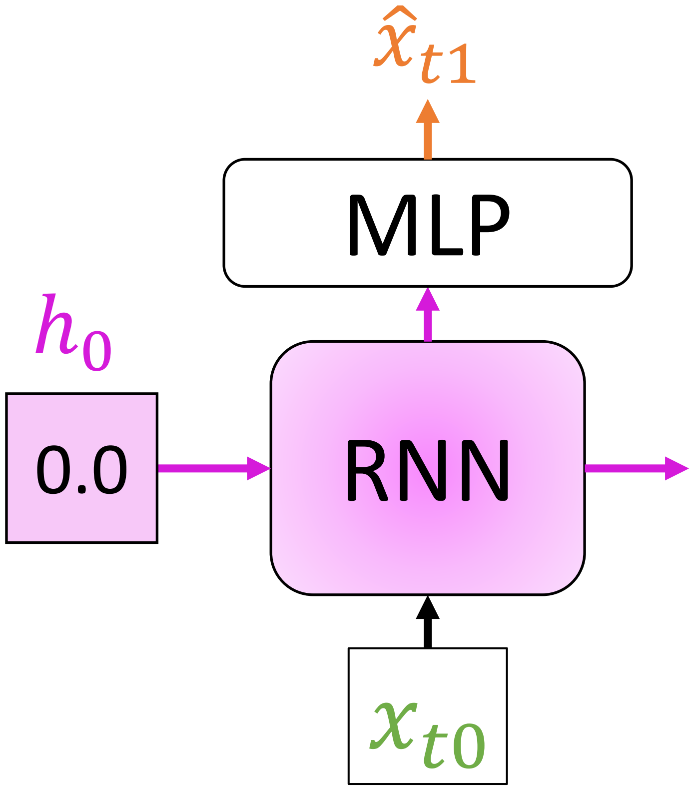
Input sequence

x_{t0}	x_{t1}	...	x_T

Target

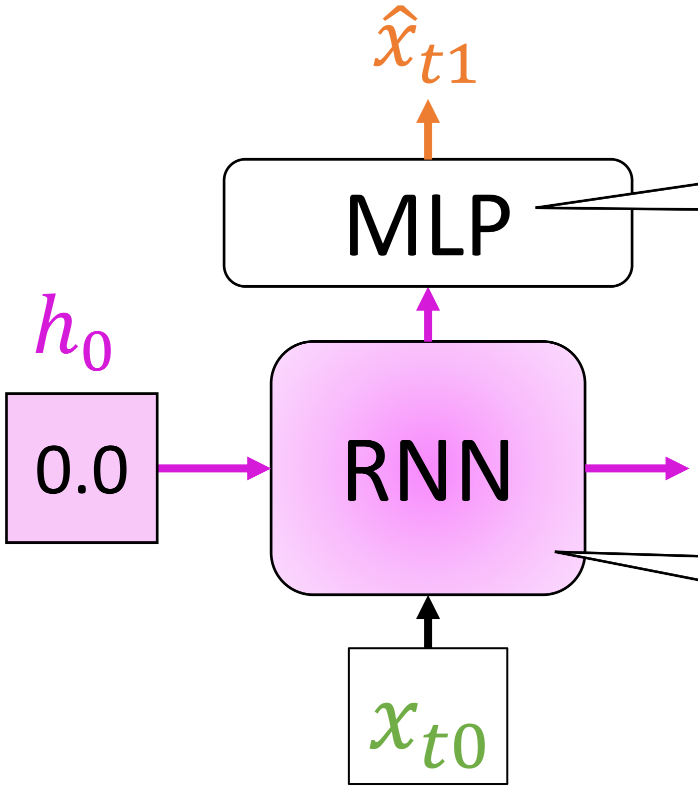
$\hat{x}_{T+1}$

Recurrent Neural Networks



Input sequence	x_{t0}	x_{t1}	...	x_T	Target	$\hat{x}_{T+1}$

Recurrent Neural Networks



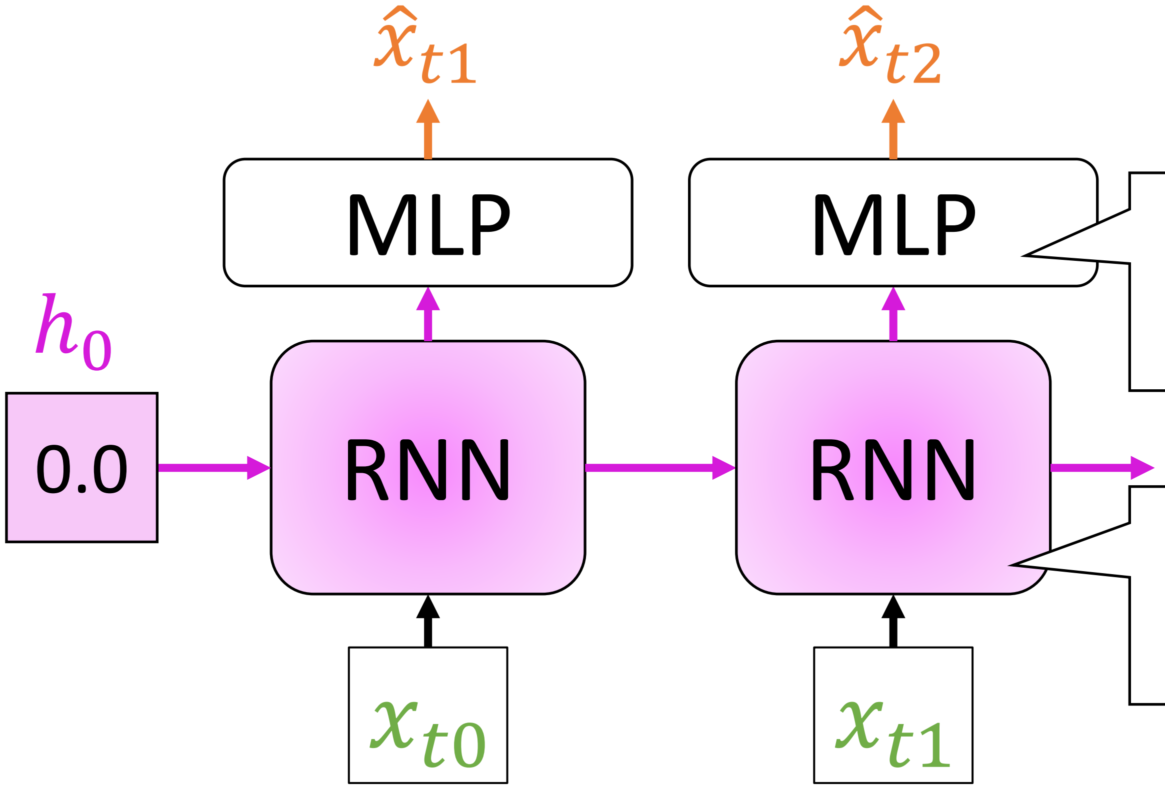
$$\hat{x}_{t1} = \tanh(w_{ho} \cdot h_1 + \beta_o)$$

$$h_1 = \tanh(w_{hh} \cdot h_0 + W_{ih} \cdot x_{t0} + \beta_h)$$

Input sequence x_{t0} | x_{t1} | … | x_T

Target $\hat{x}_{T+1}$

Recurrent Neural Networks



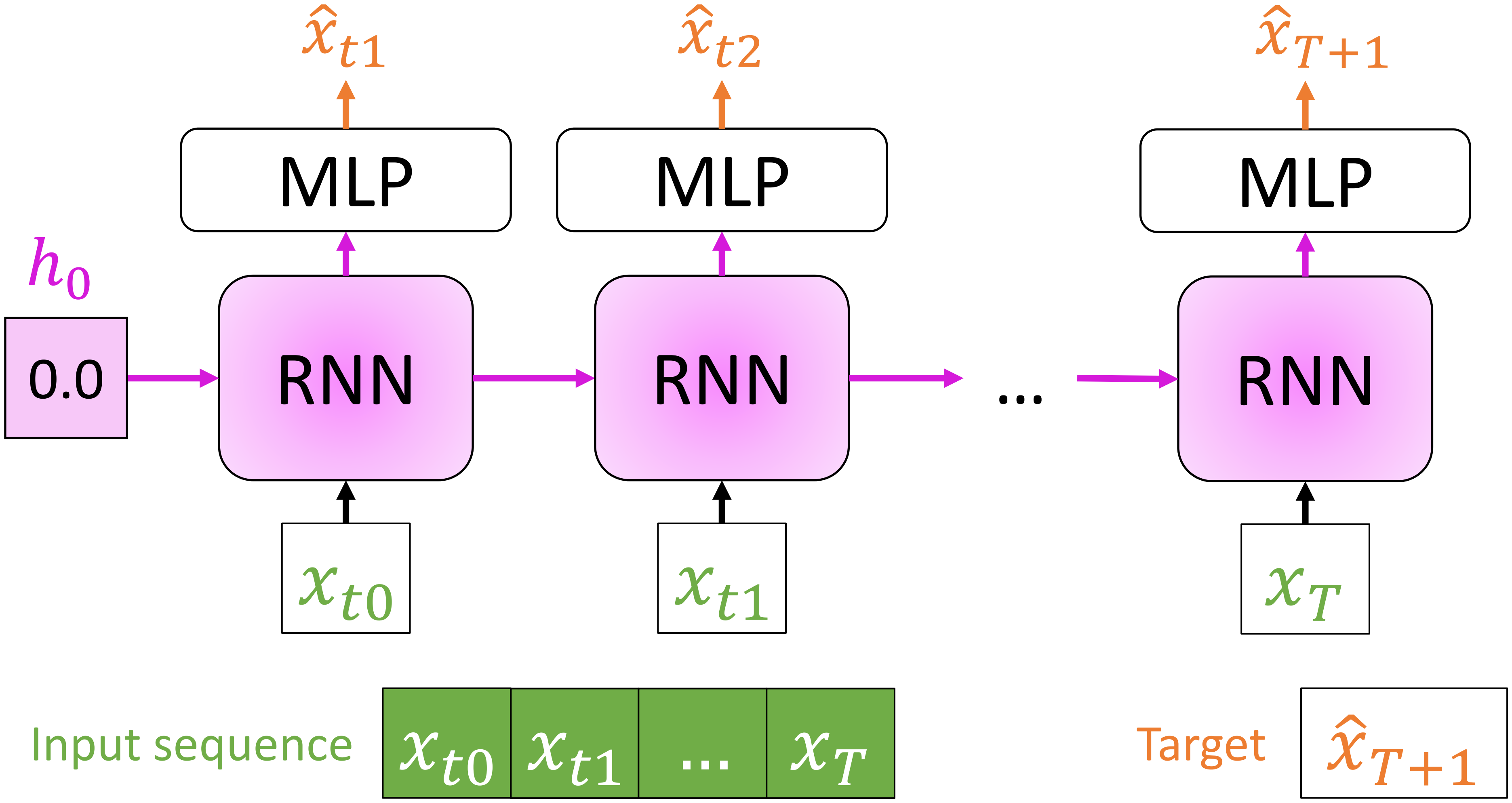
$$\hat{x}_{t2} = \tanh(w_{ho} \cdot h_2 + \beta_o)$$

$$h_2 = \tanh(w_{hh} \cdot h_1 + W_{ih} \cdot x_{t1} + \beta_h)$$

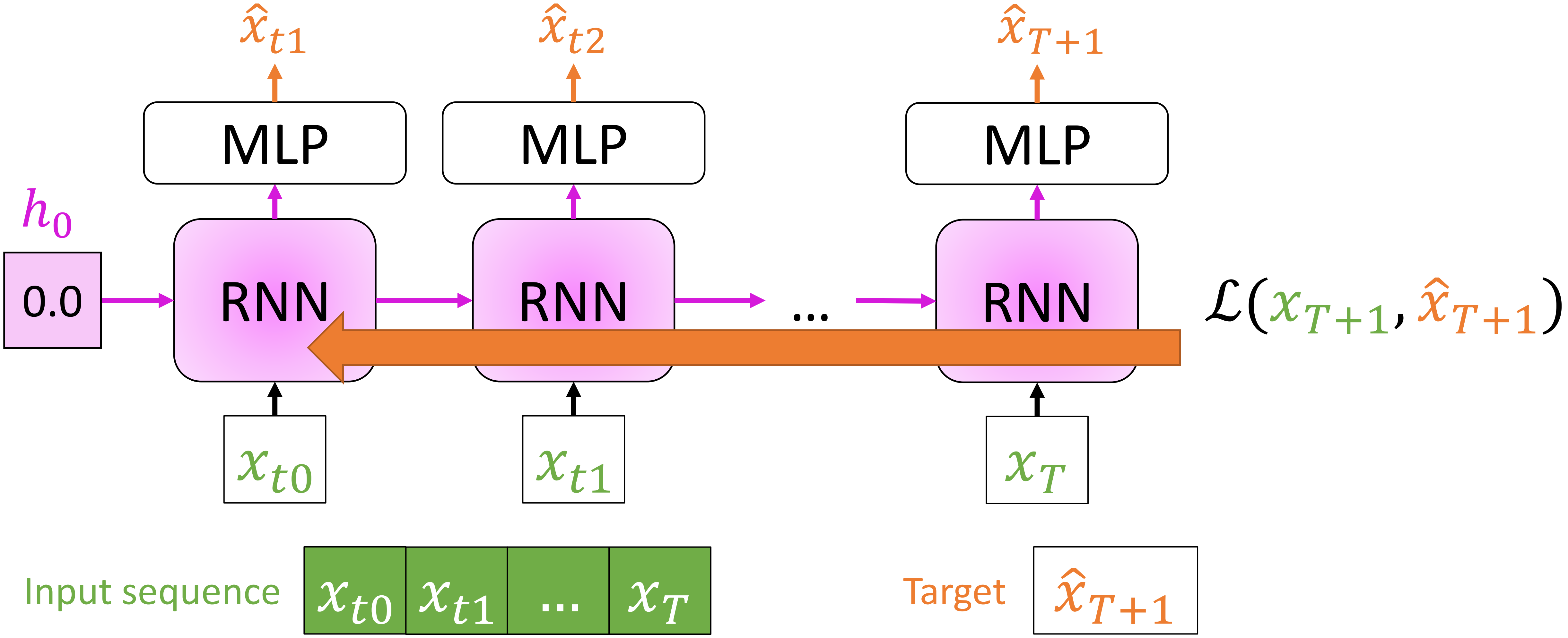
Input sequence x_{t0} x_{t1} … x_T

Target $\hat{x}_{T+1}$

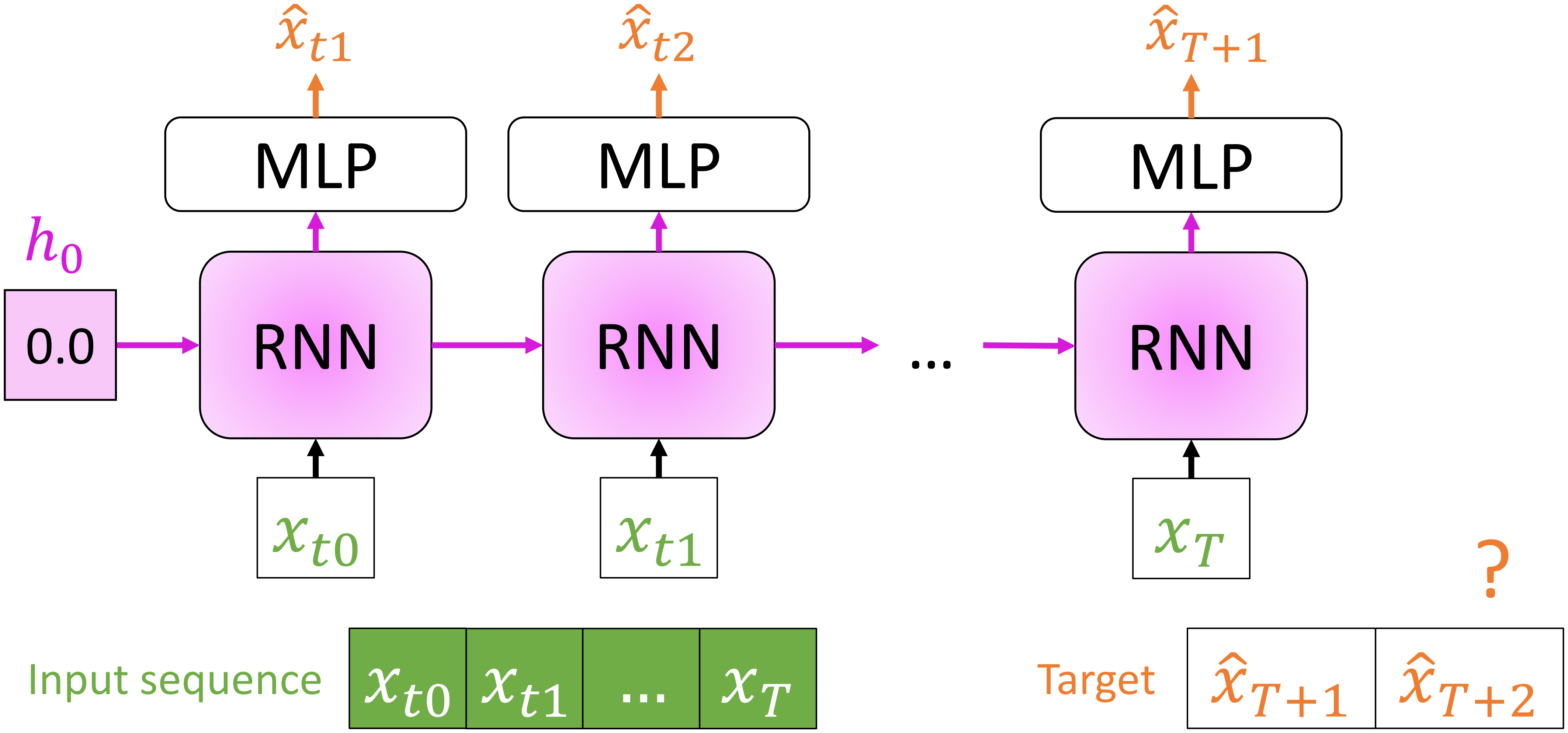
Recurrent Neural Networks



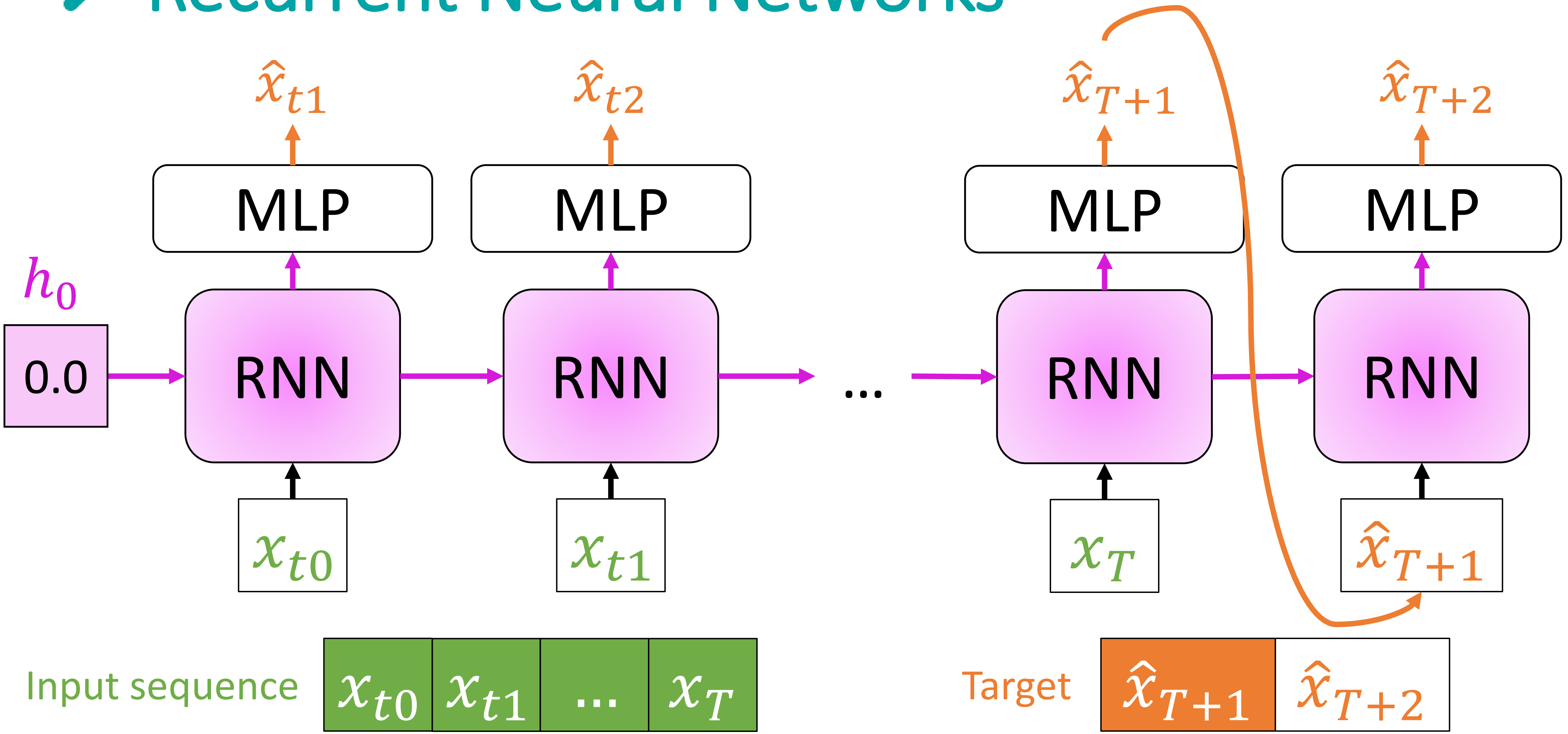
Recurrent Neural Networks



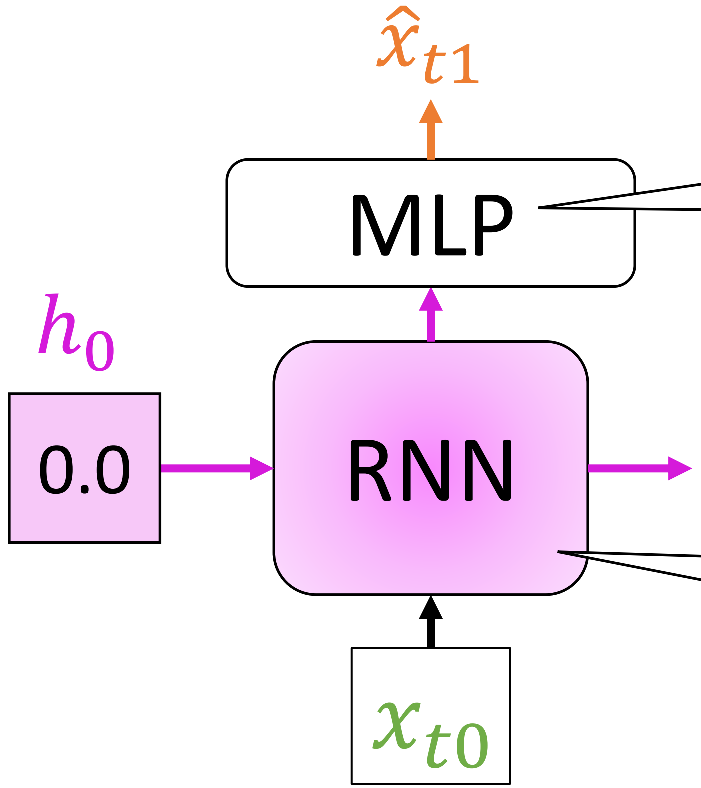
Recurrent Neural Networks



Recurrent Neural Networks



Recurrent Neural Networks



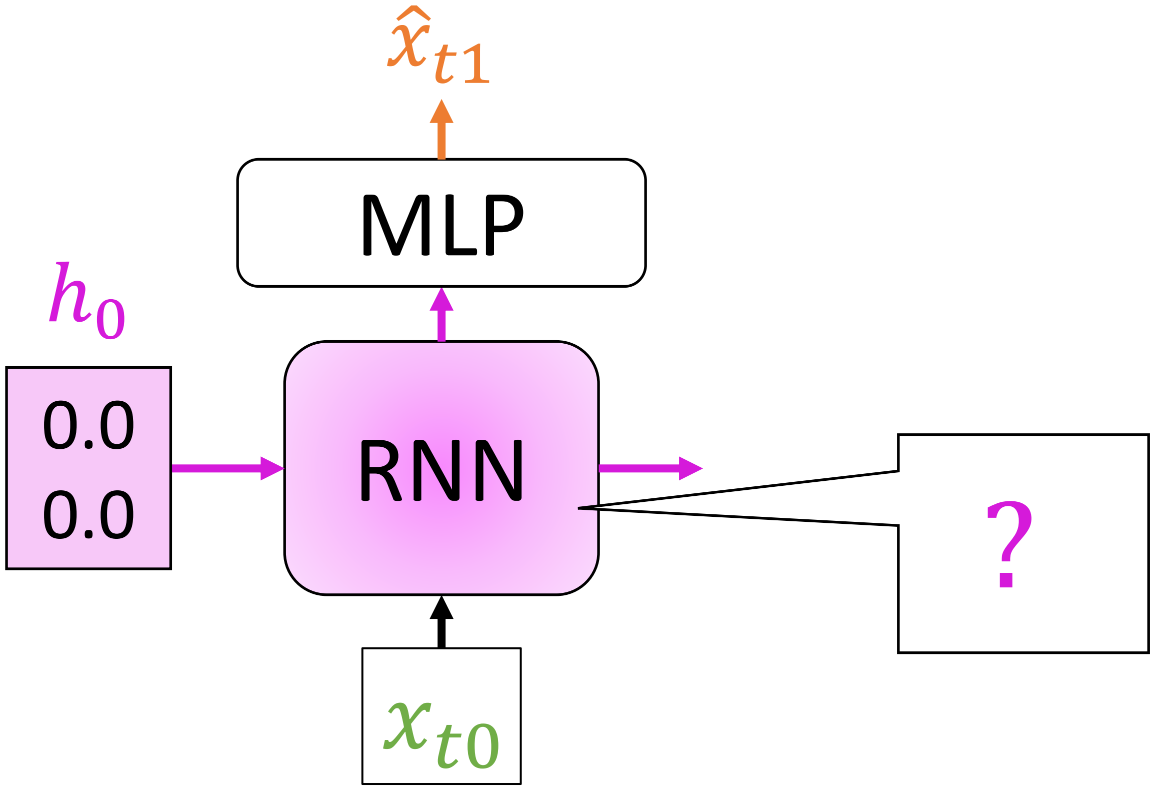
$$\hat{x}_{t1} = \tanh(w_{ho} \cdot h_1 + \beta_o)$$

$$h_1 = \tanh(w_{hh} \cdot h_0 + W_{ih} \cdot x_{t0} + \beta_h)$$

Input sequence x_{t0} | x_{t1} | ... | x_T

Target $\hat{x}_{T+1}$

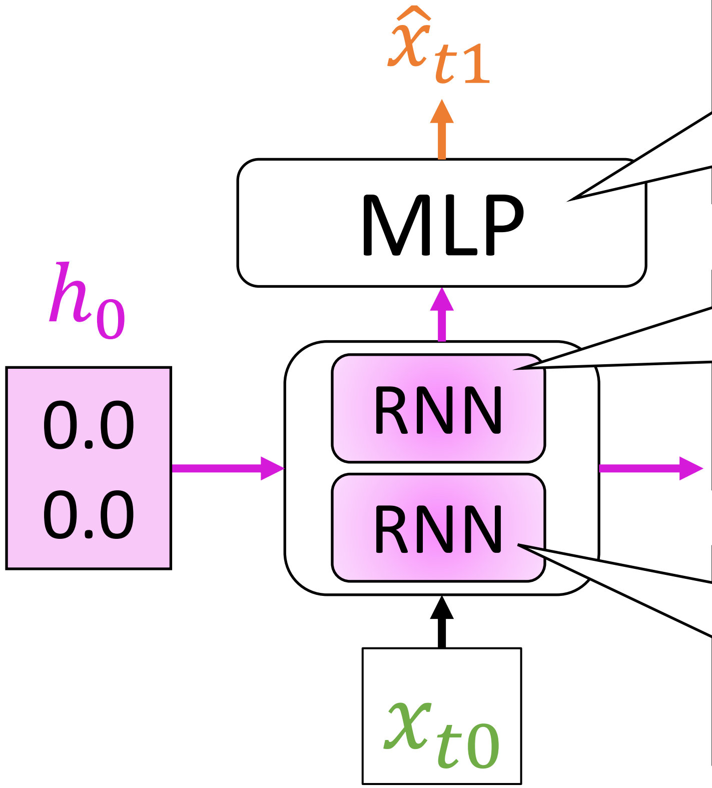
Recurrent Neural Networks



Input sequence | x_{t0} | x_{t1} | ... | x_T

Target | $\hat{x}_{T+1}$

Recurrent Neural Networks



$$\hat{x}_{t1} = \tanh(w_{ho0} \cdot h_{10} + w_{ho1} \cdot h_{11} + \beta_o)$$

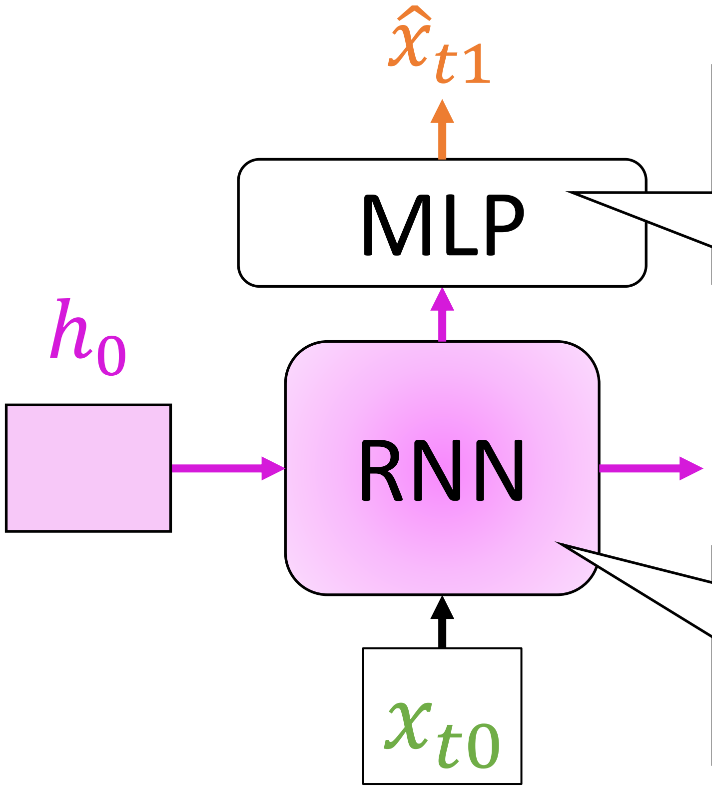
$$h_{11} = \tanh(w_{hh10} \cdot h_{00} + w_{hh11} \cdot h_{01} + W_{ih} \cdot x_{t0} + \beta_h)$$

$$h_{10} = \tanh(w_{hh00} \cdot h_{00} + w_{hh10} \cdot h_{01} + W_{ih} \cdot x_{t0} + \beta_h)$$

Input sequence x_{t0} | x_{t1} | … | x_T

Target $\hat{x}_{T+1}$

Recurrent Neural Networks



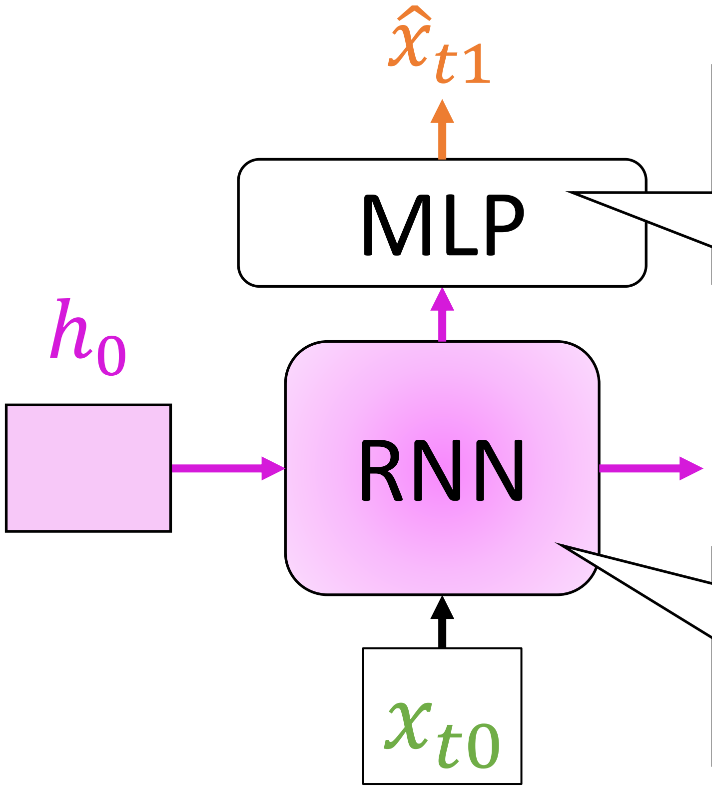
$$\hat{x}_{t1} = \tanh(W_{ho} \cdot h_0 + \beta_o)$$

$$h_1 = \tanh(W_{hh} \cdot h_0 + W_{ih} \cdot x_{t0} + \beta_h)$$

Input sequence x_{t0} | x_{t1} | … | x_T

Target $\hat{x}_{T+1}$

Recurrent Neural Networks



$$\hat{x}_{t1} = \tanh(W_{ho} \cdot h_0 + \beta_o)$$

$$h_1 = \tanh(\underbrace{W_{hh}}_{(2,2)} \cdot \underbrace{h_0}_{(2,1)} + \underbrace{W_{ih}}_{(2,3)} \cdot \underbrace{x_{t0}}_{(3,1)} + \underbrace{\beta_h}_{(2,1)})$$

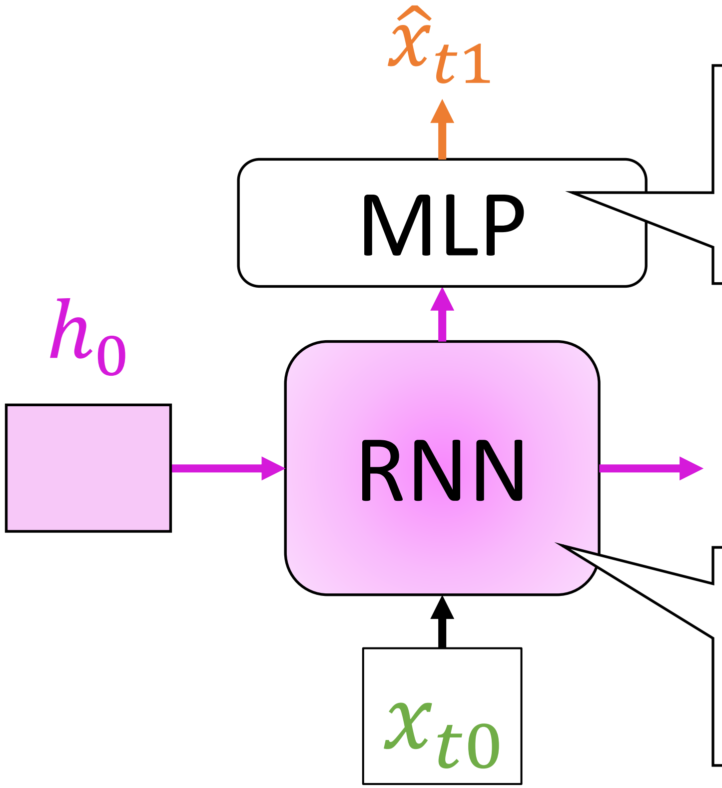
(2,1)

Input sequence

x_{t0}	x_{t1}	...	x_T

Target x_{T+1}

Recurrent Neural Networks



$$\hat{x}_{t1} = \tanh(\overset{(3,2)}{W_{ho}} \cdot \overset{(2,1)}{h_0} + \overset{(3,1)}{\beta_o})$$

$$\underset{(2,1)}{h_1} = \tanh(\overset{(2,2)}{W_{hh}} \cdot \underset{(2,1)}{h_0} + \overset{(2,3)}{W_{ih}} \cdot \underset{(3,1)}{x_{t0}} + \overset{(2,1)}{\beta_h})$$

Input sequence

x_{t0}	x_{t1}	...	x_T

Target x_{T+1}

Issues with RNNs

- Unrolling is a simple and effective technique
  - However, it factually creates *very deep* networks
  - Backpropagation through 1,000 layers can be an issue
- Exploding gradient, e.g. $(1.01)^{1000}$
  - Values so big that it's an issue representing them
  - But not only, HUGE gradient updates
- Vanishing gradient, e.g. $(0.99)^{1000}$
  - Same issues as above with internal representation
  - And super-small gradient updates (run out of epochs)

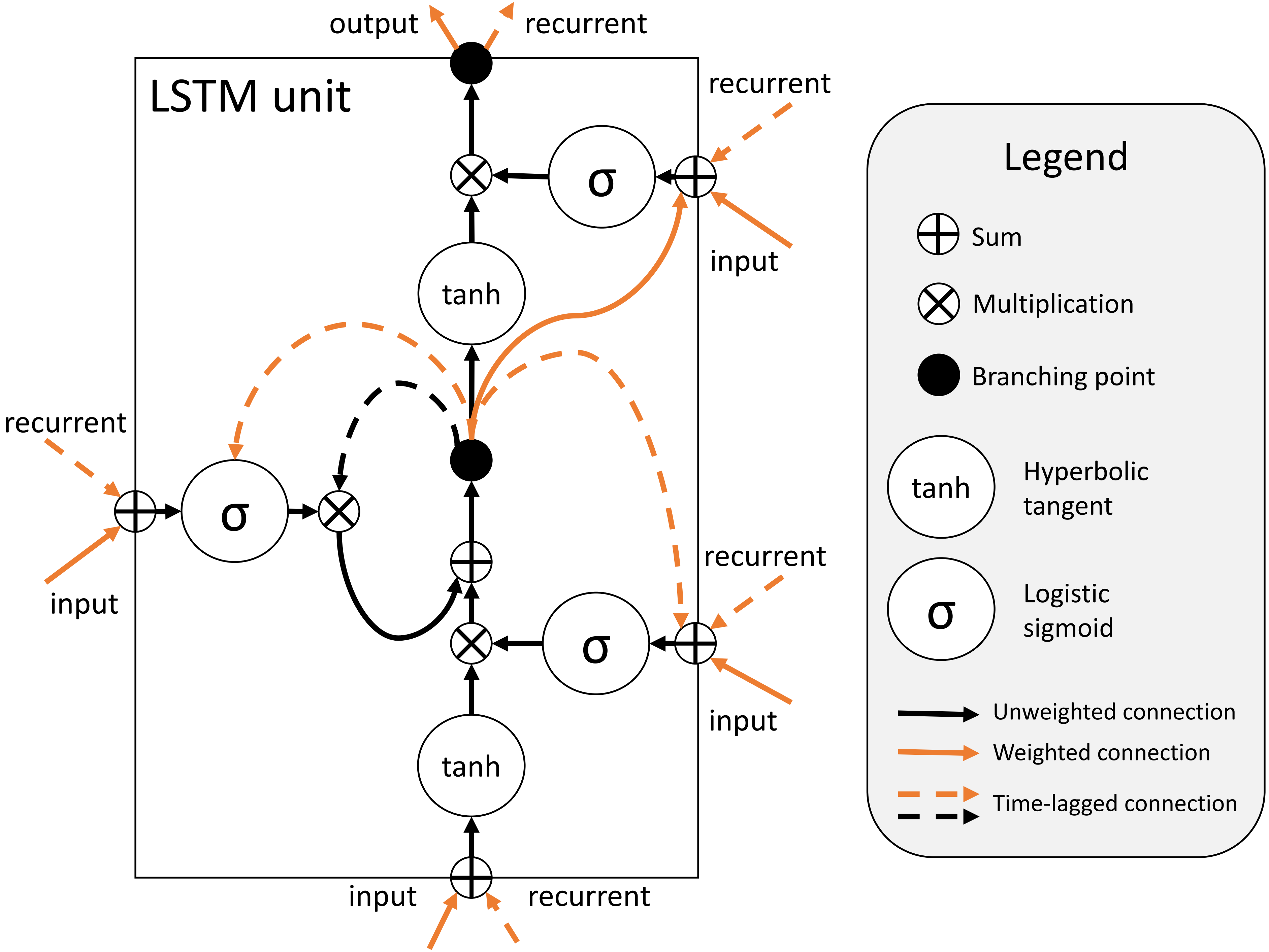
Issues with RNNs

- Solutions have been proposed
  - **Gradient clipping** to solve exploding gradient, normalize

$$\nabla_W E = \begin{cases} \frac{\nabla_W E}{\|\nabla_W E\|} & \text{if: } \|\nabla_W E\| > 1 \\ \nabla_W E & \text{else} \end{cases}$$

E here is the error function (loss)

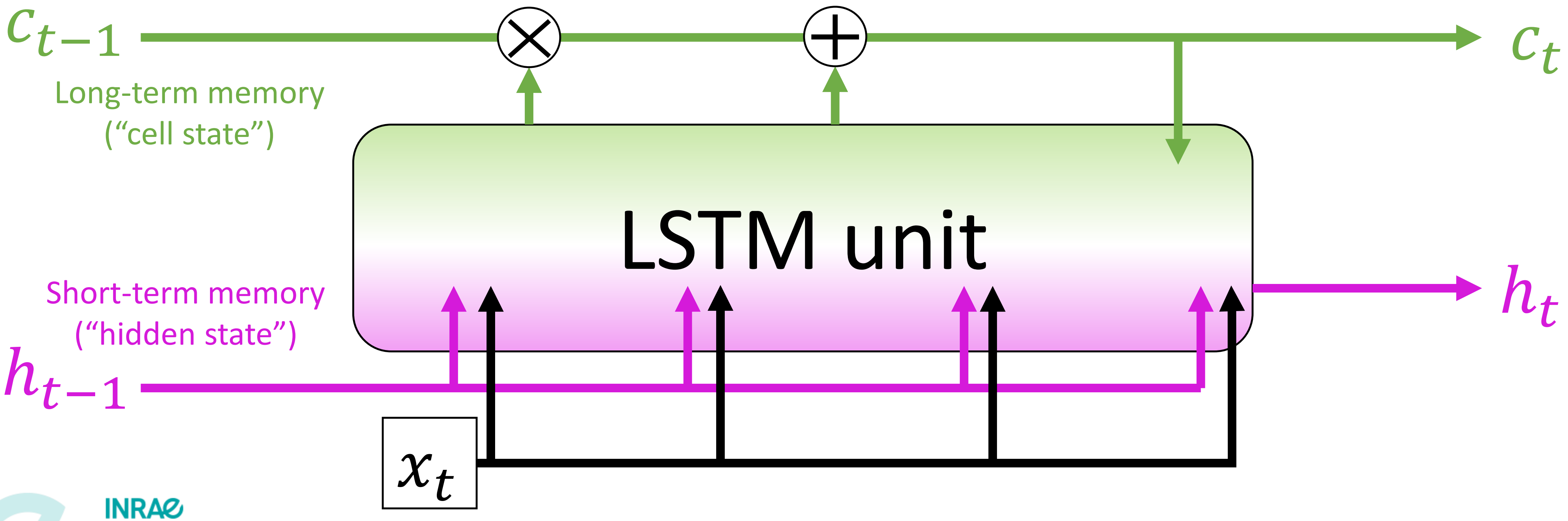
- Long-Short Term Memory Networks (LSTMs)
  - New type of module, “reset” hidden state when needed
  - No weights on the path of updating the long-term history
  - Successful in practical applications



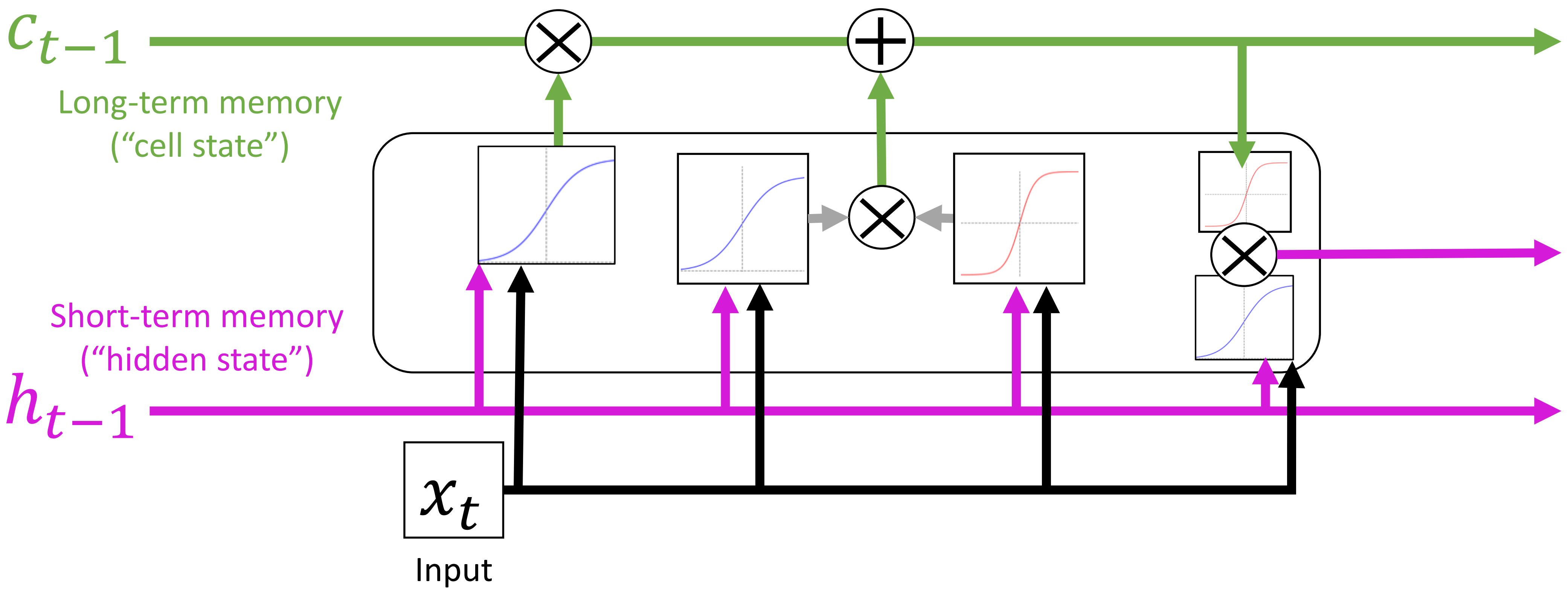
output
recurrent
LSTM unit
recurrent
input
recurrent
input
recurrent
input
input
recurrent
σ
tanh
σ
σ
tanh
Legend
Sum
Multiplication
Branching point
tanh
Hyperbolic tangent
σ
Logistic sigmoid
Unweighted connection
Weighted connection
Time-lagged connection

Long-Short Term Memory Networks

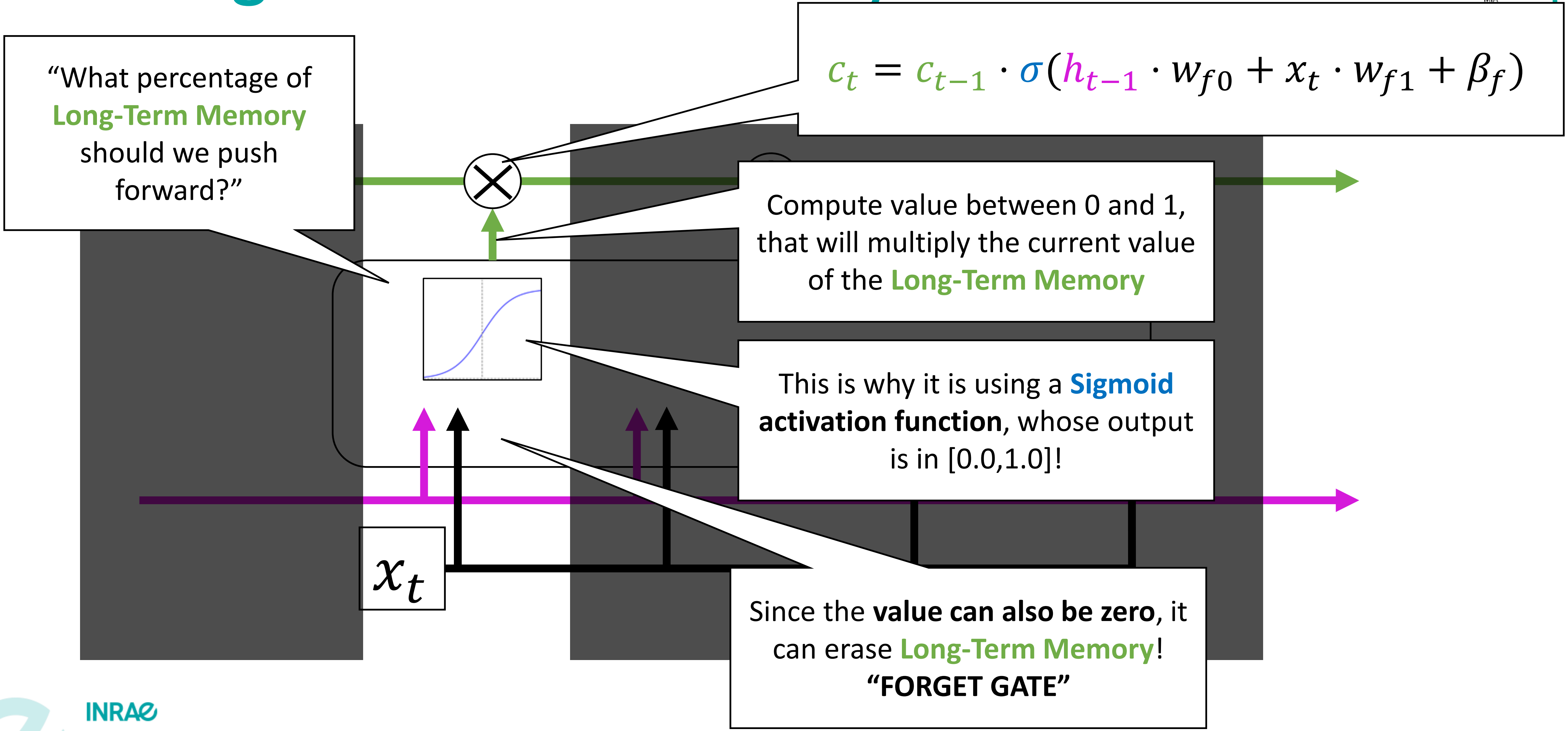
- Two "memory lanes" per unit
  - **Long-term memory**, no weights (!), **avoids exploding/vanishing**
  - **Short-term memory**, carried out from the recent past



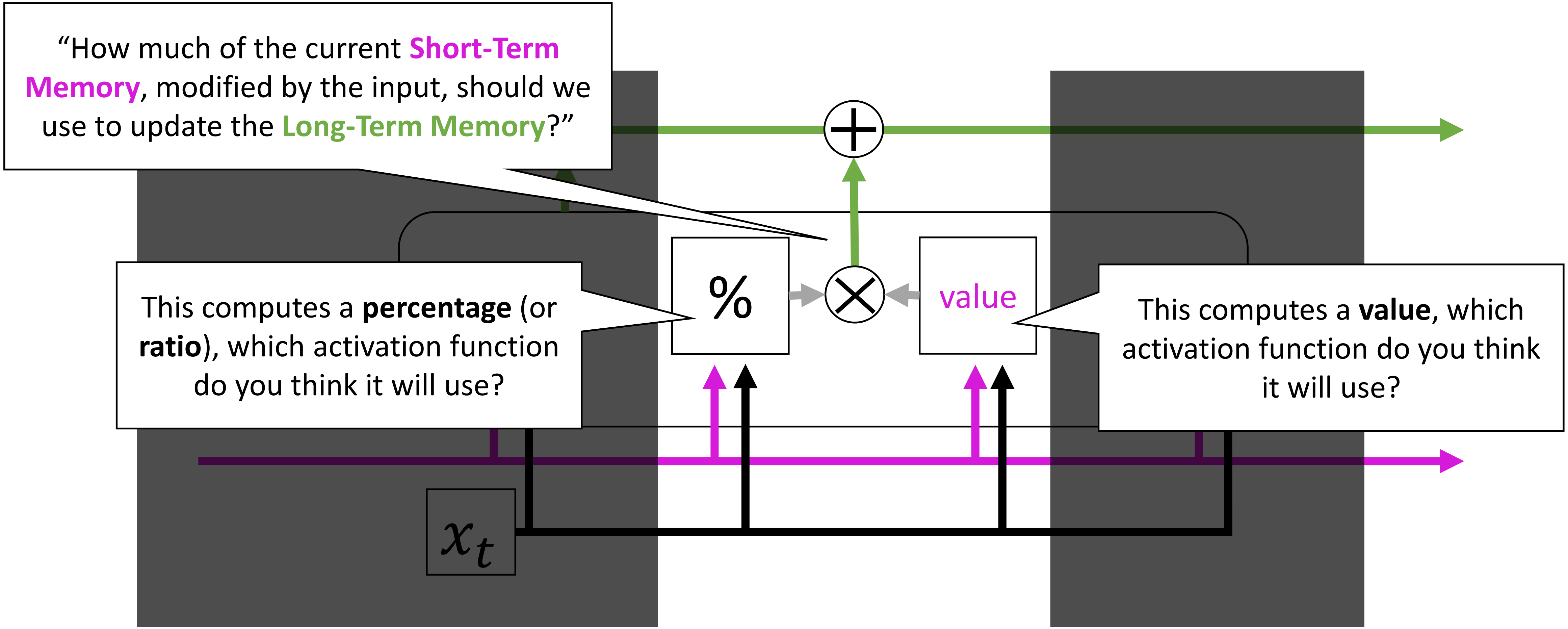
Long-Short Term Memory Unit



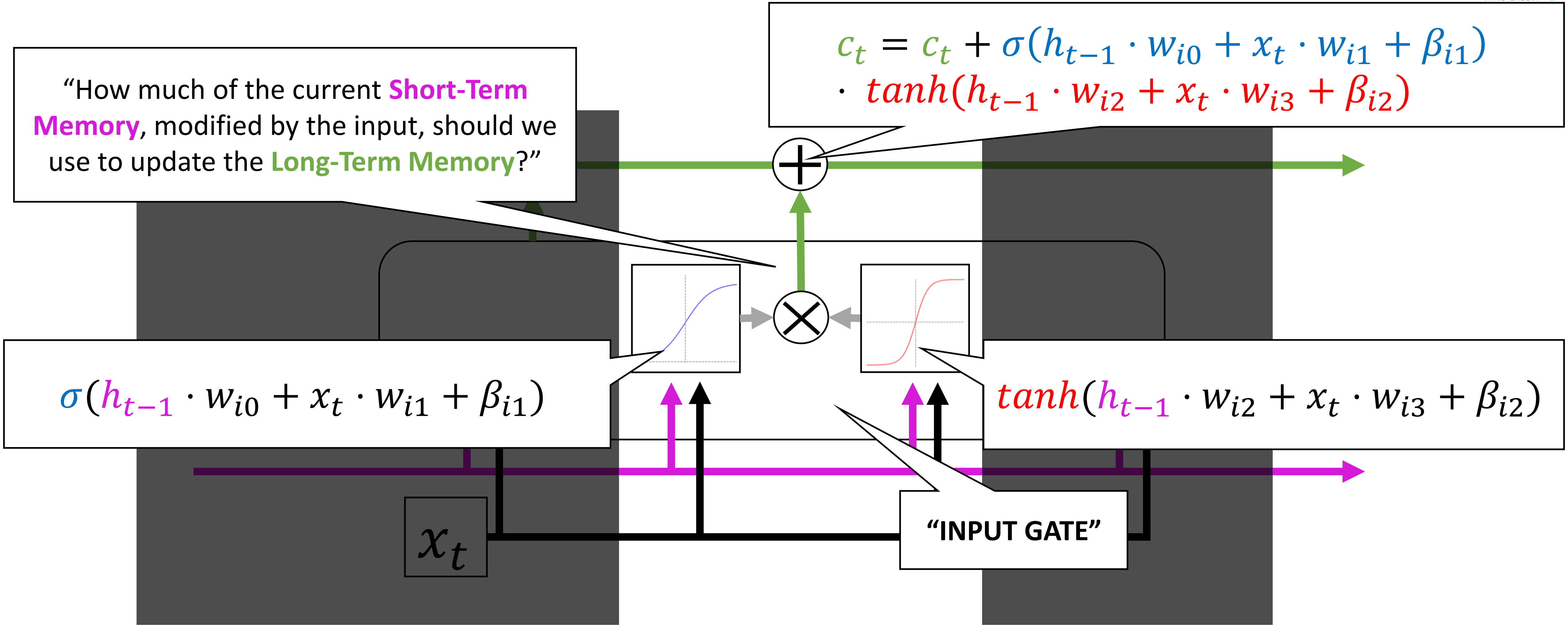
Long-Short Term Memory Unit



Long-Short Term Memory Unit



Long-Short Term Memory Unit



Parenthesis

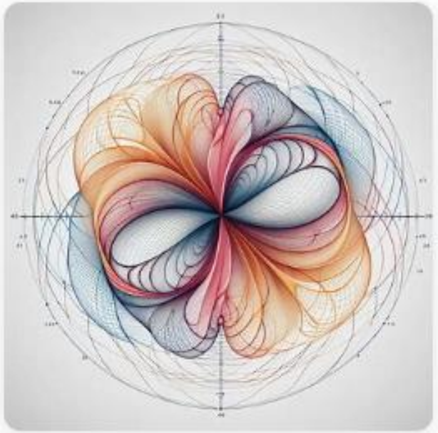
Copilot

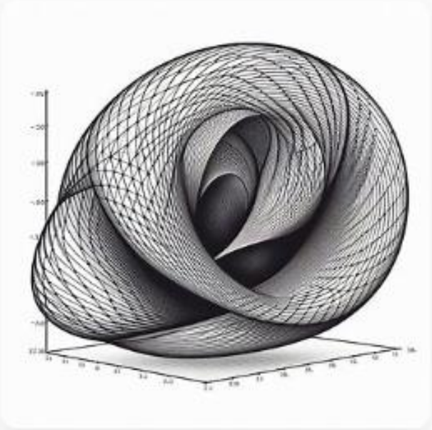
Certainly! Here’s the hyperbolic tangent function (tanh) graph between -1 and 1, with unlabeled axes:

$$f(x) = \tanh(x)$$

!Hyperbolic Tangent Graph

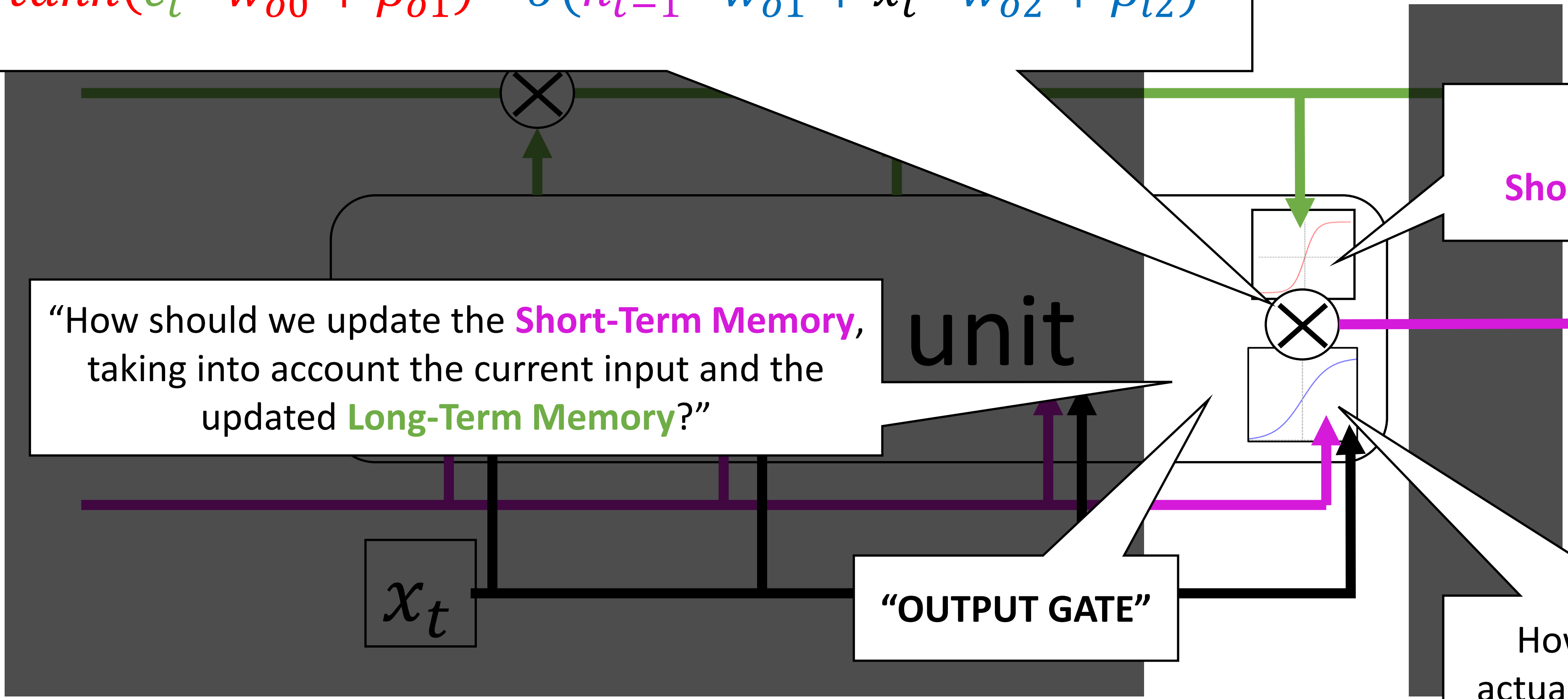
2 of 30





Long-Short Term Memory Unit

$$h_t = tanh(c_t \cdot w_{o0} + \beta_{o1}) \cdot \sigma(h_{t-1} \cdot w_{o1} + x_t \cdot w_{o2} + \beta_{i2})$$



Potential **Short-Term Memory**

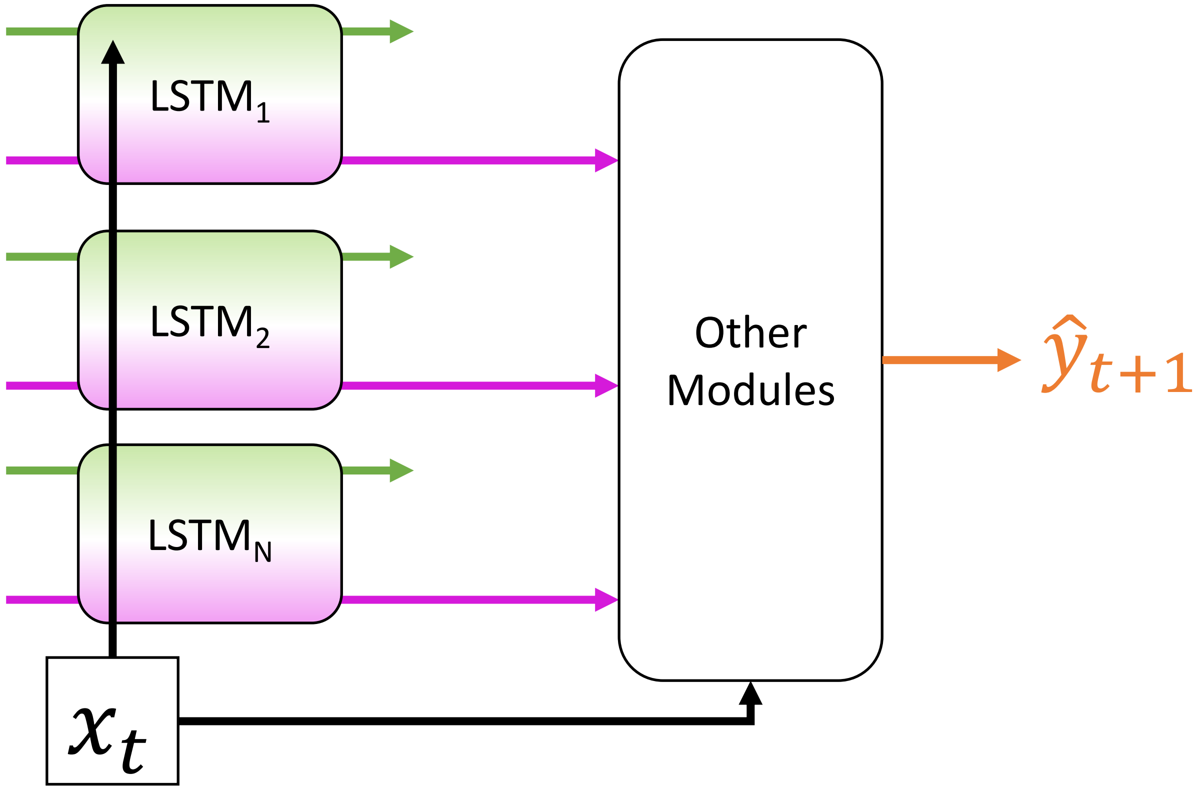
“How should we update the **Short-Term Memory**, taking into account the current input and the updated **Long-Term Memory**?”

Output h_t

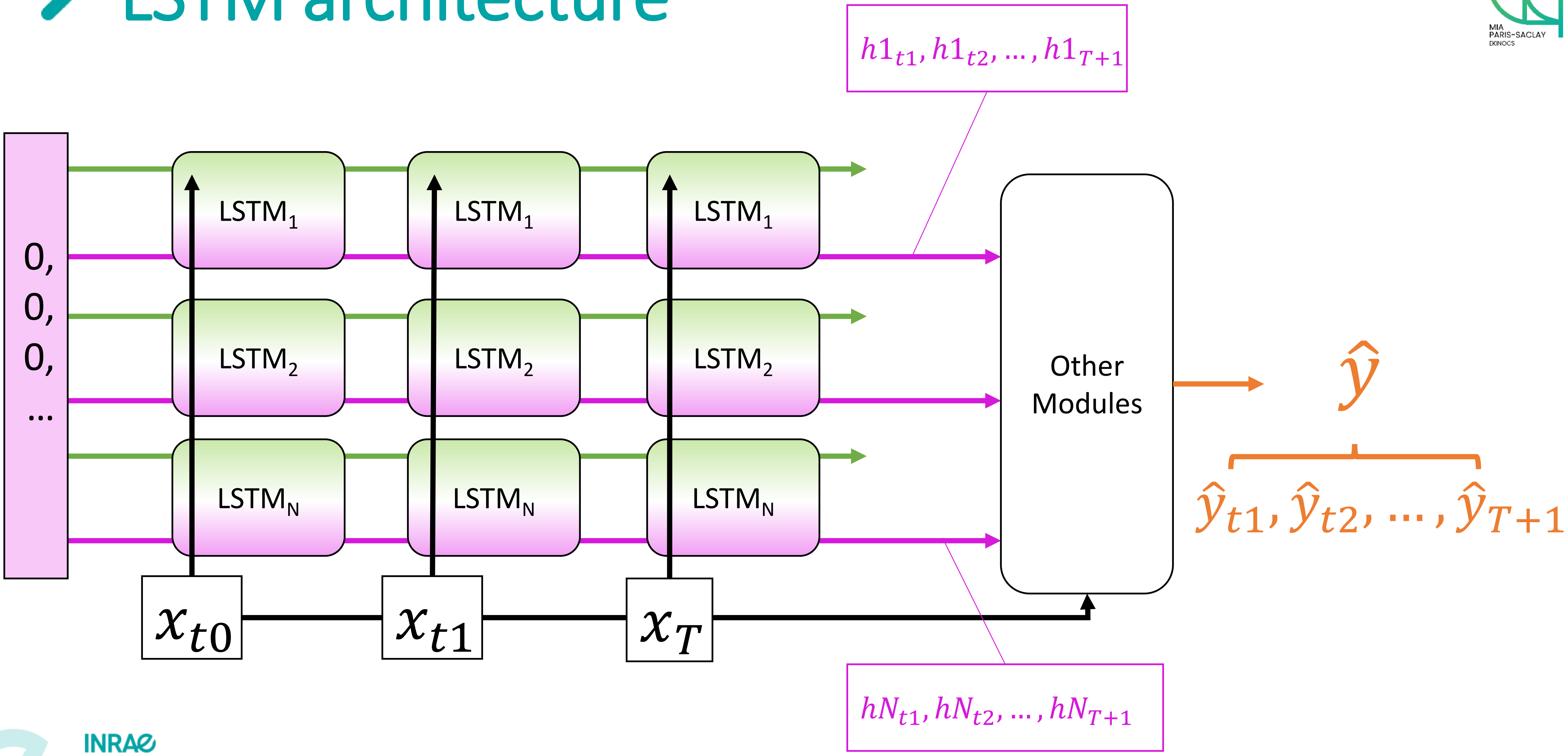
“OUTPUT GATE”

How much of it will actually be passed on to the next LSTM unit

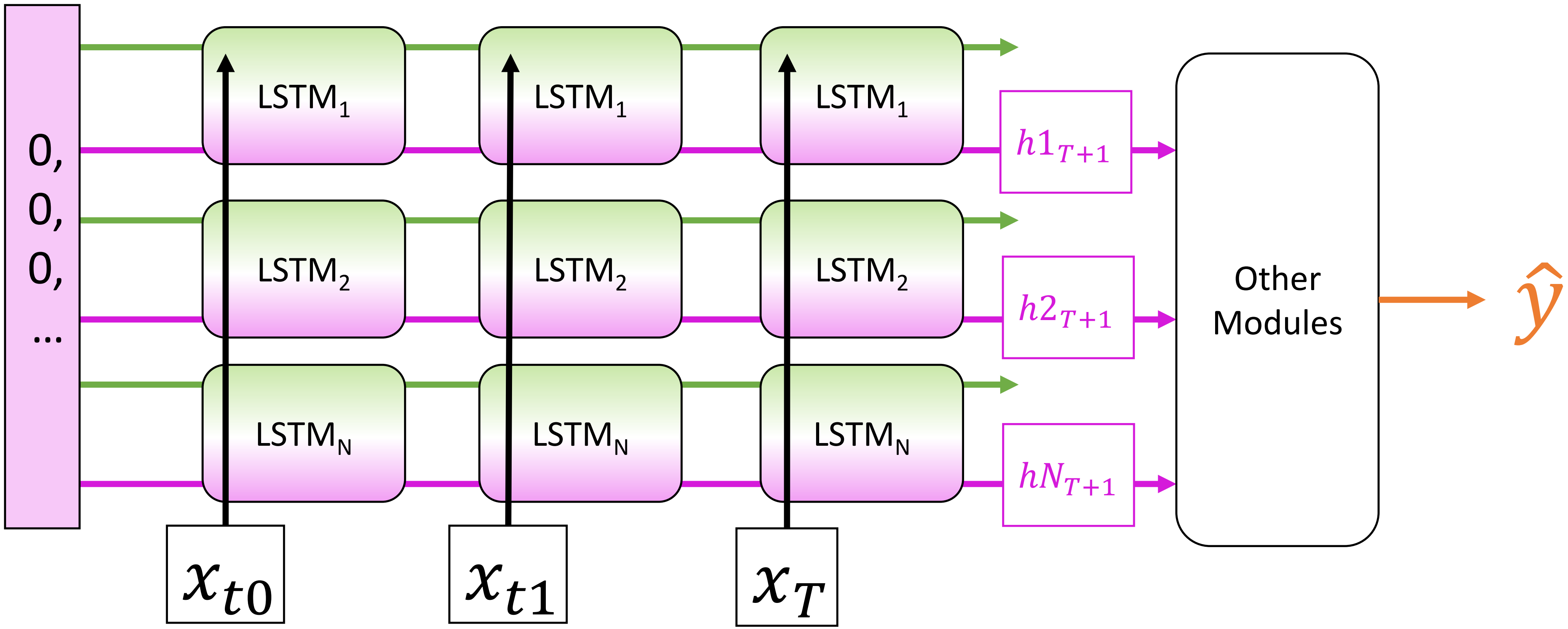
LSTM architecture



LSTM architecture



LSTM architecture



Gated Recurrent Units (GRUs)

- Variant of LSTM with less parameters per unit (12 vs 16)
  - Considered more or less just as accurate, slightly faster to train
  - More recent applications favor GRUs over LSTMs
  - But not always, in practice people *try both* and then pick

pytorch RNN

$$h_t = \tanh(x_t W_{ih}^T + b_{ih} + h_{t-1} W_{hh}^T + b_{hh})$$

pytorch LSTM

$$\begin{aligned}
i_t &= \sigma(W_{ii} x_t + b_{ii} + W_{hi} h_{t-1} + b_{hi}) \\
f_t &= \sigma(W_{if} x_t + b_{if} + W_{hf} h_{t-1} + b_{hf}) \\
g_t &= \tanh(W_{ig} x_t + b_{ig} + W_{hg} h_{t-1} + b_{hg}) \\
o_t &= \sigma(W_{io} x_t + b_{io} + W_{ho} h_{t-1} + b_{ho}) \\
c_t &= f_t \odot c_{t-1} + i_t \odot g_t \\
h_t &= o_t \odot \tanh(c_t)
\end{aligned}$$

pytorch GRU

$$\begin{aligned}
r_t &= \sigma(W_{ir} x_t + b_{ir} + W_{hr} h_{(t-1)} + b_{hr}) \\
z_t &= \sigma(W_{iz} x_t + b_{iz} + W_{hz} h_{(t-1)} + b_{hz}) \\
n_t &= \tanh(W_{in} x_t + b_{in} + r_t \odot (W_{hn} h_{(t-1)} + b_{hn})) \\
h_t &= (1 - z_t) \odot n_t + z_t \odot h_{(t-1)}
\end{aligned}$$

Questions?

Bibliography

- Amari, S. I. (1972). *Learning patterns and pattern sequences by self-organizing nets of threshold elements*. IEEE Transactions on computers, 100(11), 1197-1206. [**RNNs**]
- Cho, K., Van Merriënboer, B., Bahdanau, D., & Bengio, Y. (2014). On the properties of neural machine translation: Encoder-decoder approaches. arXiv preprint arXiv:1409.1259. [**GRUs**]
- Hochreiter, S., & Schmidhuber, J. (1997). *Long short-term memory*. Neural computation, 9(8), 1735-1780. [**LSTMs**]
- Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. Proceedings of the national academy of sciences, 79(8), 2554-2558. [**RNNs**]
- StatQuest, Long Short-Term Memory (LSTM), Clearly Explained, https://www.youtube.com/watch?v=YCzL96nL7j0