Deep Learning:

Recurrent Neural Networks

Robert Legenstein
Institute of Machine Learning and Neural Computation
robert.legenstein@tugraz.at

Deep Learning VO - WS 25/26
Lecture 8

Today

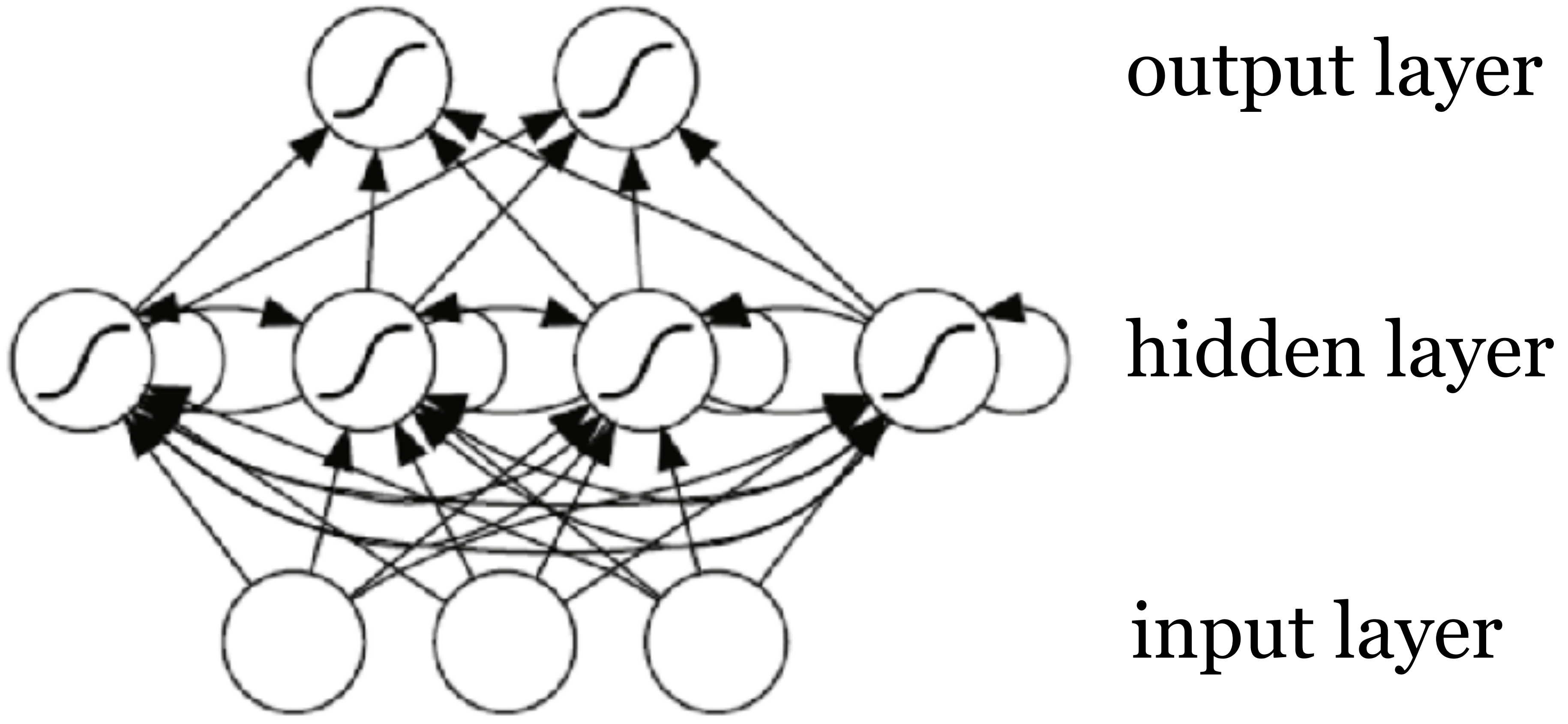
- Recurrent Neural Networks
  - Simple RNNs
  - Backpropagation Through Time
  - Architectural Variants
  - Gated RNNs (LSTM & GRU)

Recurrent Neural Networks (RNNs)

Typical input: Sequence of values (vectors) $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(\tau)}$

Typical output: Sequence of values (vectors) $\mathbf{o}^{(1)}, \ldots, \mathbf{o}^{(\tau)}$

Network graph includes recurrent connections.



At each discrete time step t,

- input $\mathbf{x}^{(t)}$ is presented,
- hidden state $\mathbf{h}^{(t)}$ is updated,
- output $\mathbf{o}^{(t)}$ is computed,
- loss $L^{(t)}$ is evaluated, and
- gradient is backpropagated through time.

(Simple) RNNs were introduced in the 1980s.

Advantages:

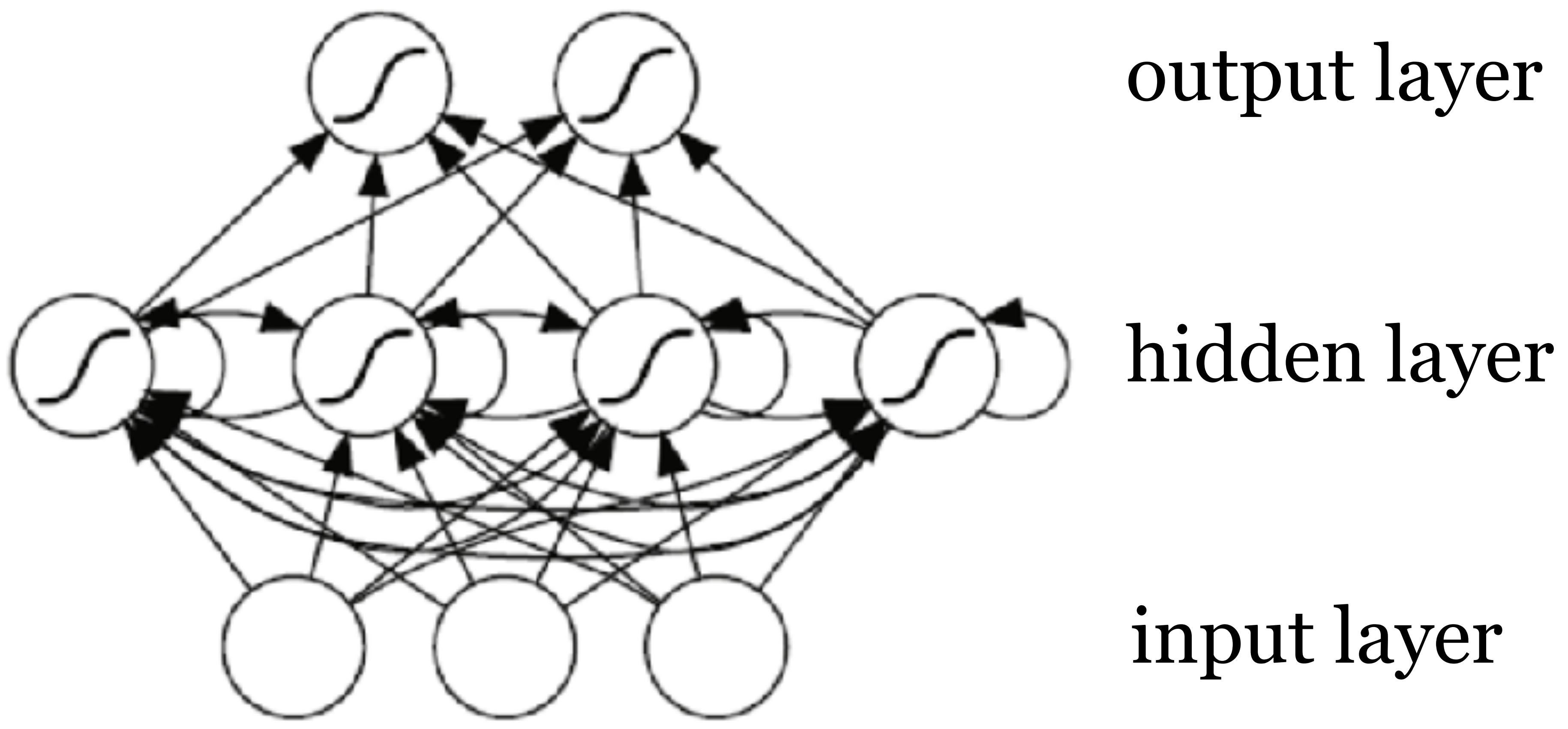
- Can process inputs of variable length.
- Implements parameter sharing over time.
- Can therefore generalize over time (compare to CNNs).

Simple RNNs: Computation

Typical input: Sequence of values (vectors) $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(\tau)}$

Typical output: Sequence of values (vectors) $\mathbf{o}^{(1)}, \ldots, \mathbf{o}^{(\tau)}$

Network graph includes recurrent connections.



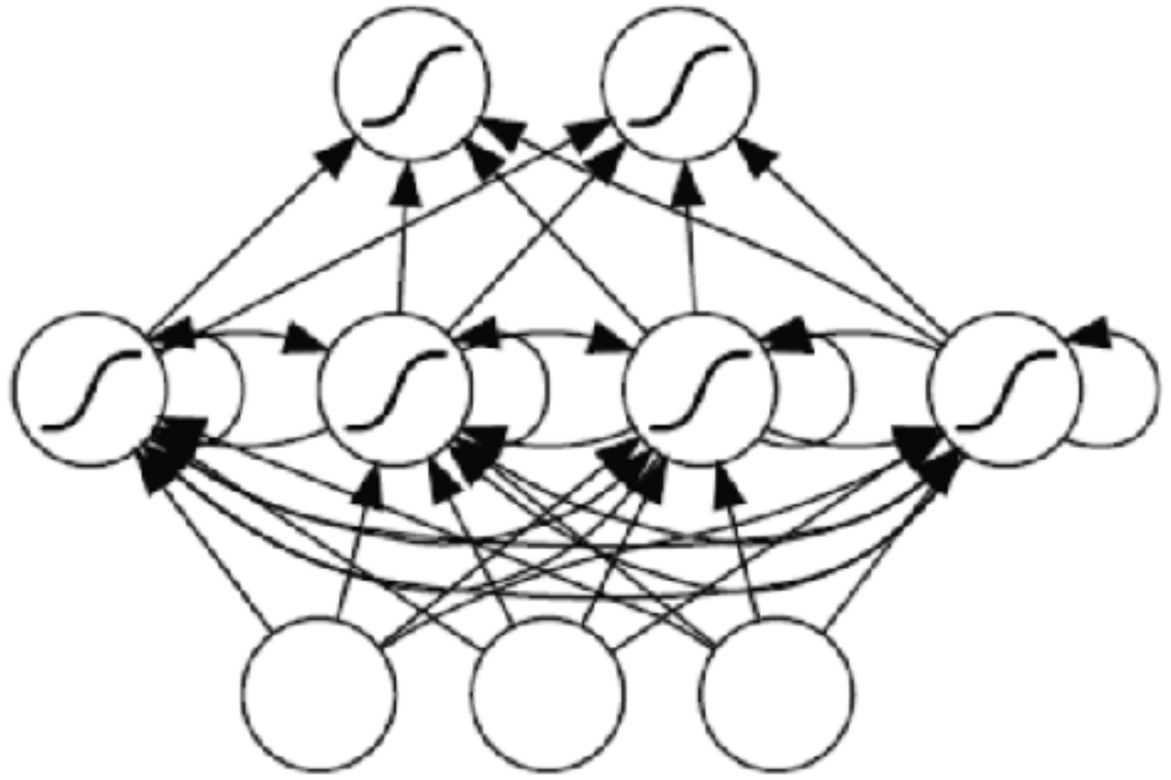
At each discrete time step t,

- input $\mathbf{x}^{(t)}$ is presented,
- hidden state $\mathbf{h}^{(t)}$ is updated,
- output $\mathbf{o}^{(t)}$ is computed,
- loss $L^{(t)}$ is evaluated, and
- gradient is backpropagated through time.

$$\mathbf{a}^{(t)} = \mathbf{b} + W\mathbf{h}^{(t-1)} + U\mathbf{x}^{(t)}$$
$$\mathbf{h}^{(t)} = \tanh(\mathbf{a}^{(t)})$$
$$\mathbf{o}^{(t)} = \mathbf{c} + V\mathbf{h}^{(t)}$$
$$\hat{\mathbf{y}}^{(t)} = \operatorname{softmax}\left(\mathbf{o}^{(t)}\right)$$

Unfolding in Time

View the RNN as a dynamical system driven by external input:



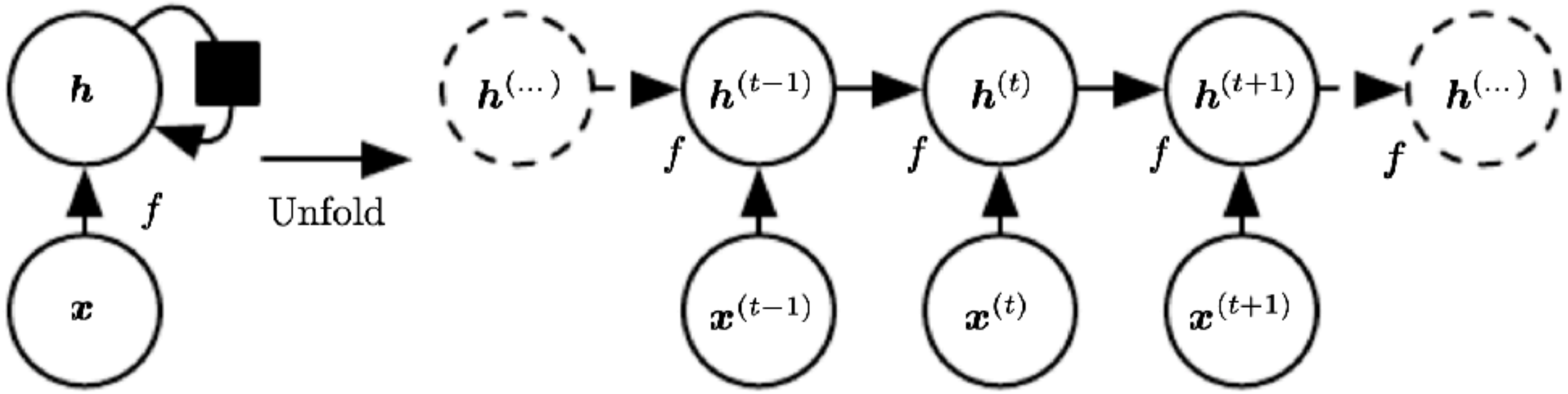
$$\mathbf{h}^{(t)} = f\left(\mathbf{h}^{(t-1)}, \mathbf{x}^{(t)}; \theta\right)$$

We can unfold the equation in time:

$$\mathbf{h}^{(3)} = f\left(\mathbf{h}^{(2)}, \mathbf{x}^{(3)}; \theta\right) = f\left(f\left(\mathbf{h}^{(1)}, \mathbf{x}^{(2)}; \theta\right), \mathbf{x}^{(3)}; \theta\right) = f\left(f\left(f\left(\mathbf{h}^{(0)}, \mathbf{x}^{(1)}; \theta\right), \mathbf{x}^{(2)}; \theta\right), \mathbf{x}^{(3)}; \theta\right)$$

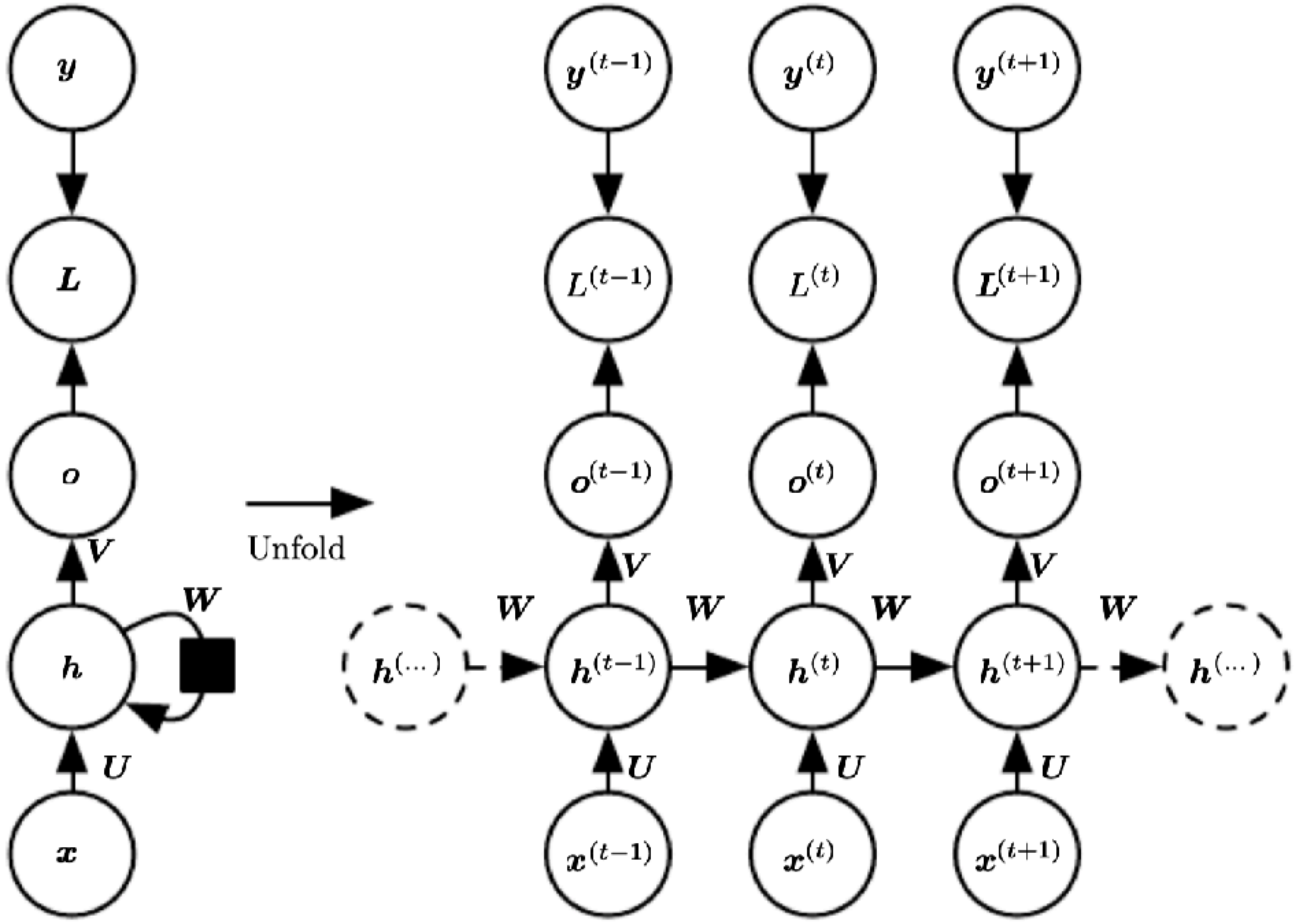
Unfolding the computational graph:

(square indicates unit time delay)



Network typically learns to use $\mathbf{h}^{(t)}$ as a lossy summary of the task-relevant aspects of the past sequence up to t.

RNN Equations and Training



$$\mathbf{a}^{(t)} = \mathbf{b} + W\mathbf{h}^{(t-1)} + U\mathbf{x}^{(t)}$$
$$\mathbf{h}^{(t)} = \tanh(\mathbf{a}^{(t)})$$
$$\mathbf{o}^{(t)} = \mathbf{c} + V\mathbf{h}^{(t)}$$
$$\hat{\mathbf{y}}^{(t)} = \text{softmax}\left(\mathbf{o}^{(t)}\right)$$

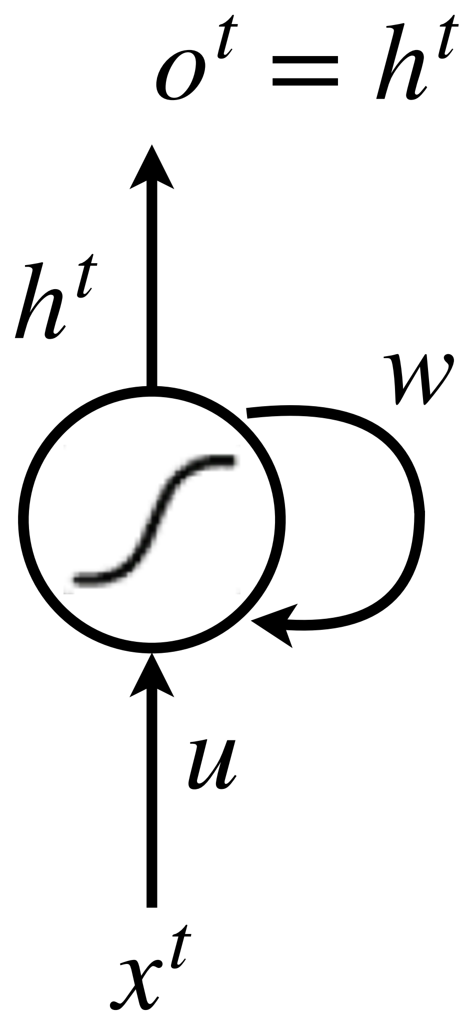
Training is via **backpropagation through time:** Compute the gradients in the unfolded computational graph.

Today

- [ ] Recurrent Neural Networks
    - [x] Simple RNNs
    - [ ] Backpropagation Through Time
    - [ ] Architectural Variants
    - [ ] Gated RNNs (LSTM & GRU)

Backpropagation Through Time

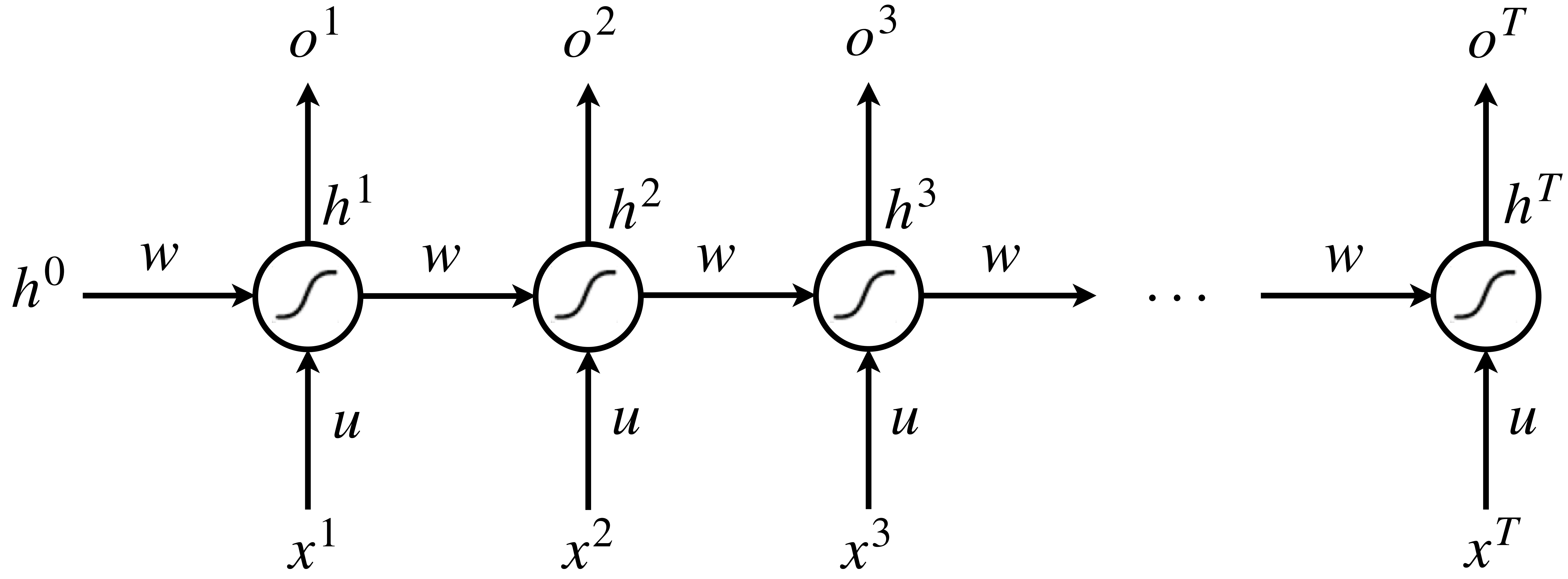
An example: a single neuron with a recurrent connection to itself.



$$h^0 = 0$$

$$h^t = \sigma\left(\underbrace{ux^t + wh^{t-1}}_{a^t = ux^t + wh^{t-1}}\right)$$

Unfolded Network:

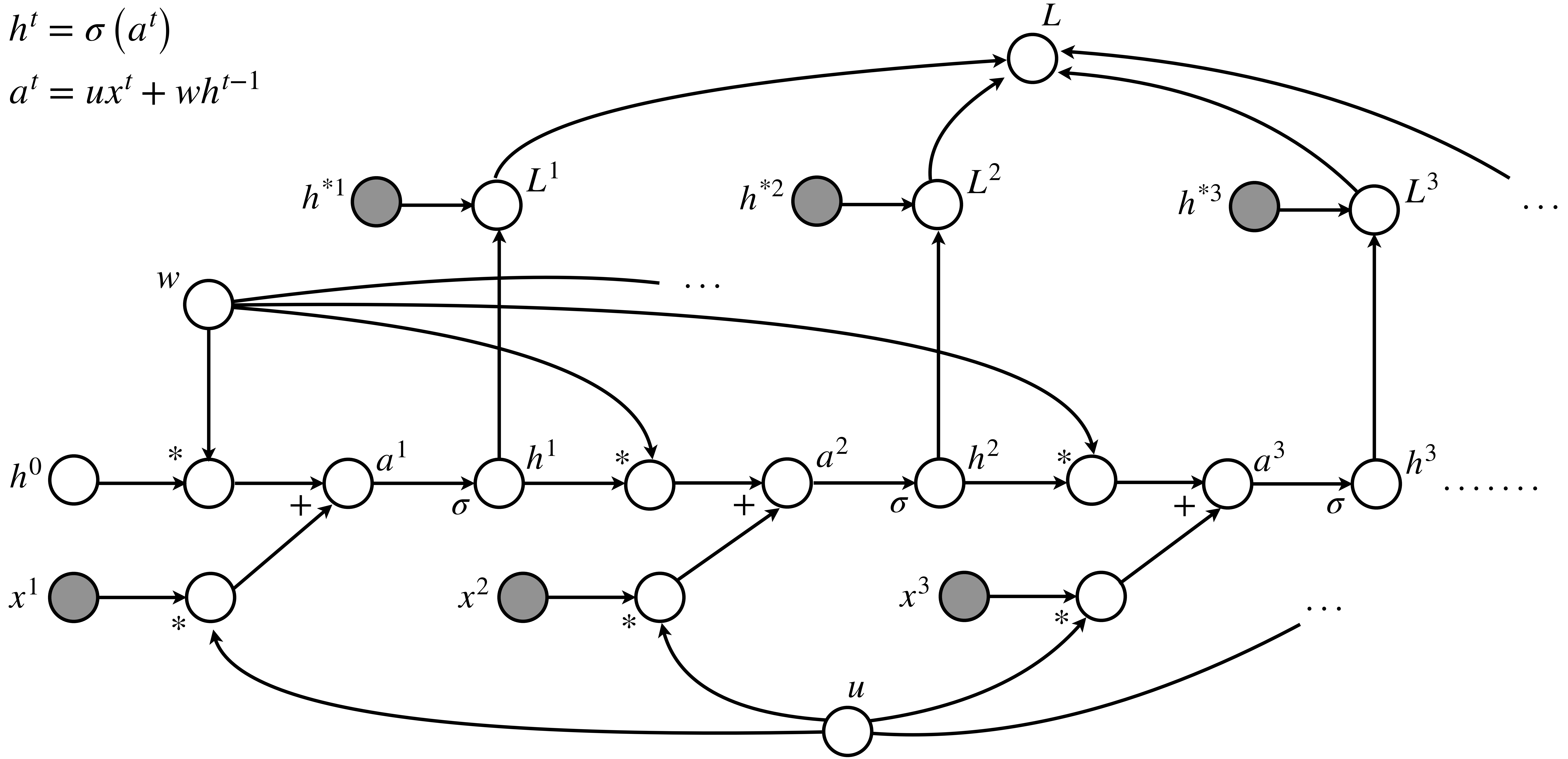


Unfolded Computational Graph

$$L = \sum_{t=1}^{T} L^t \quad L^t = \frac{1}{2}\left(h^t - h^{*t}\right)^2$$

$$h^t = \sigma\left(a^t\right)$$

$$a^t = ux^t + wh^{t-1}$$

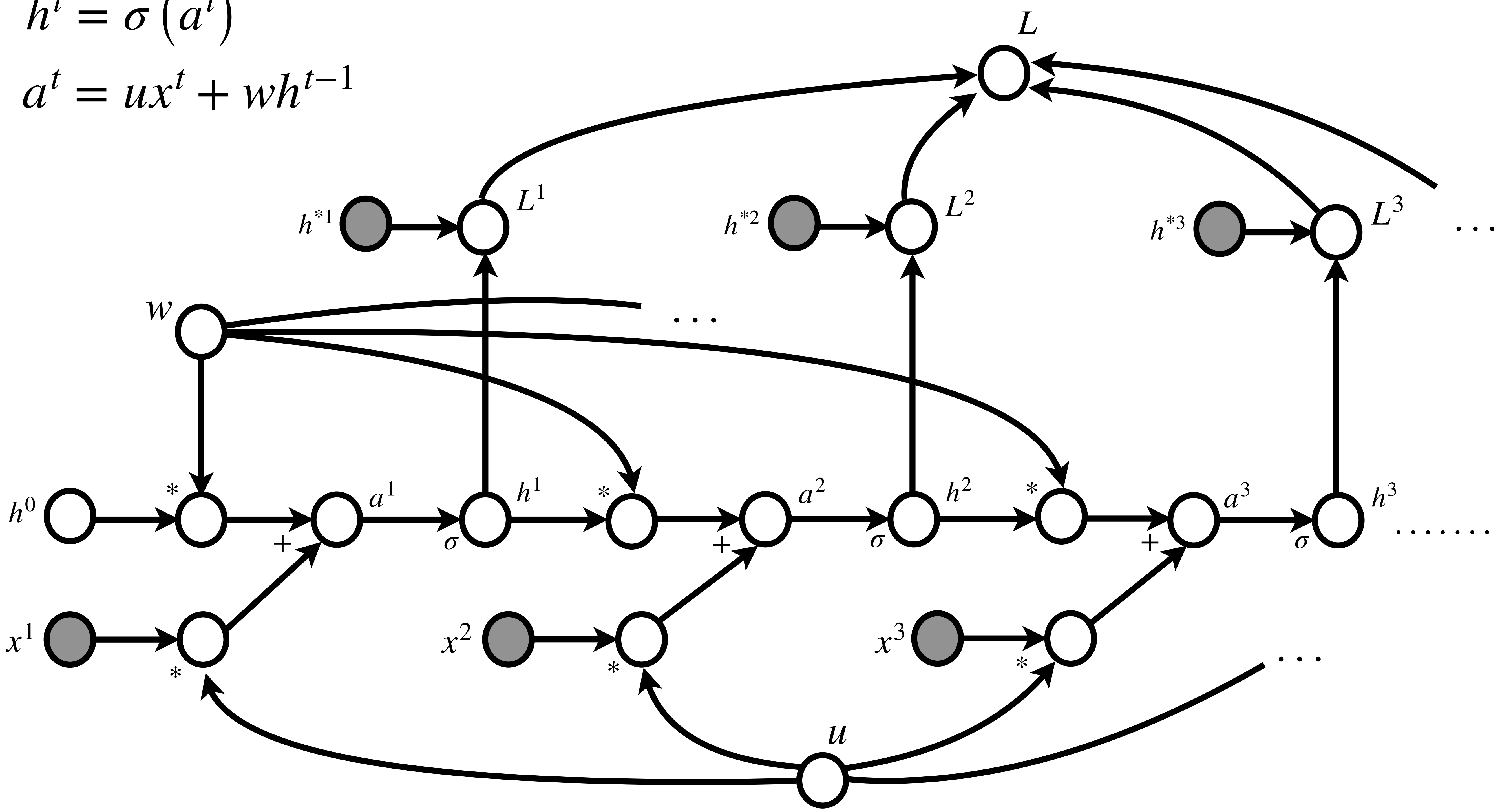


Backpropagation Through Time

$$L = \sum_{t=1}^{T} L^t \quad L^t = \frac{1}{2}\left(h^t - h^{*t}\right)^2$$

$h^t = \sigma\left(a^t\right)$

$a^t = ux^t + wh^{t-1}$



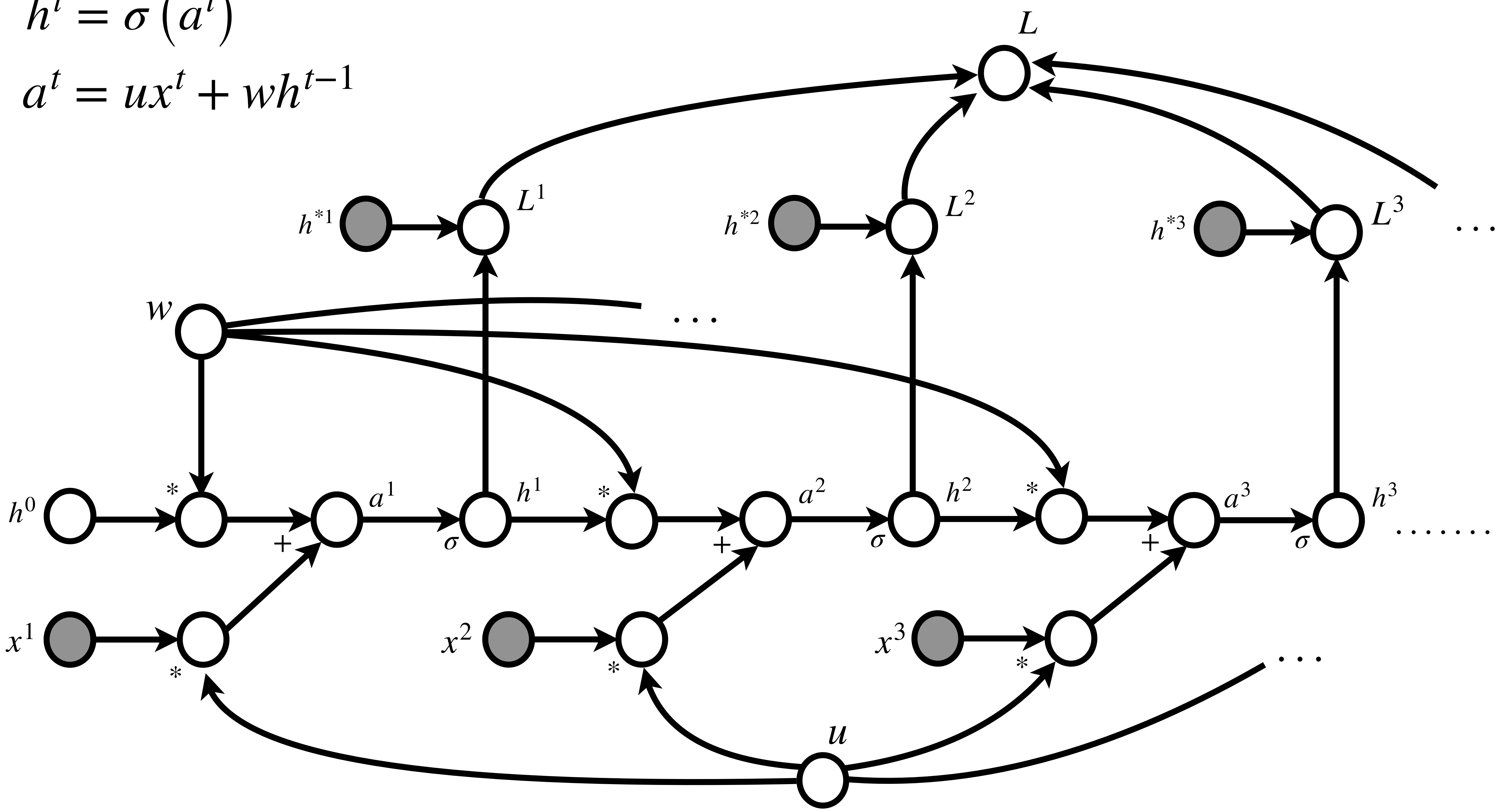
- We need $\dfrac{\partial L}{\partial w}$ and $\dfrac{\partial L}{\partial u}$ for parameter updates.
- To sketch the idea, we will show for $\dfrac{\partial L}{\partial w}$.

Backpropagation Through Time

$$L = \sum_{t=1}^{T} L^t \quad L^t = \frac{1}{2}\left(h^t - h^{*t}\right)^2$$

$$h^t = \sigma\left(a^t\right)$$

$$a^t = ux^t + wh^{t-1}$$



$$\frac{\partial L}{\partial w} = \sum_{t=1}^{T} \frac{\partial L}{\partial a^t}\frac{\partial a^t}{\partial w} = \sum_{t=1}^{T} \frac{\partial L}{\partial a^t} h^{t-1} = \sum_{t=1}^{T} \delta^t h^{t-1}$$

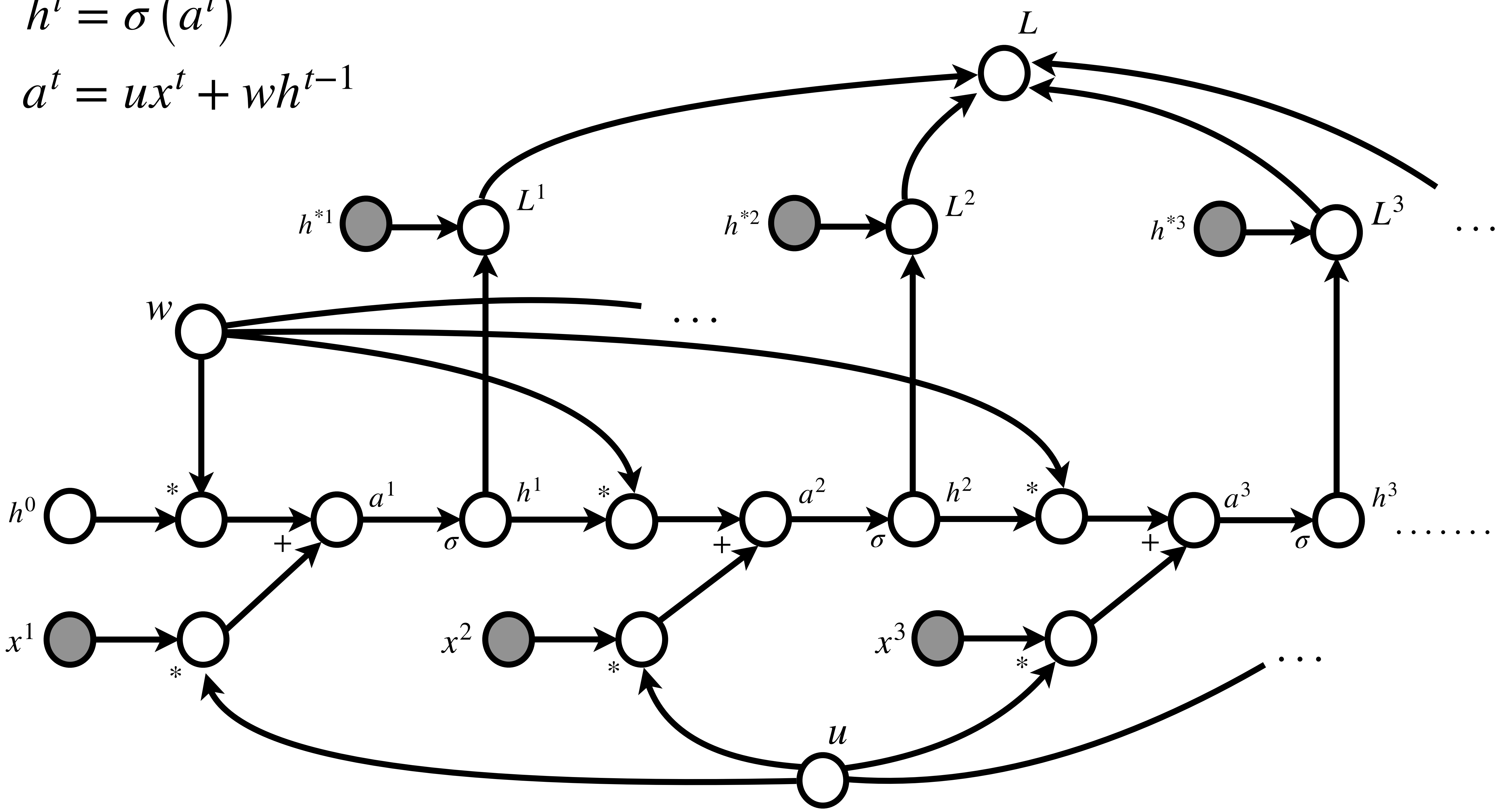
$$\boxed{\delta^t \stackrel{\text{def}}{=} \frac{\partial L}{\partial a^t}}$$

Backpropagation Through Time

$$L = \sum_{t=1}^{T} L^t \quad L^t = \frac{1}{2}\left(h^t - h^{*t}\right)^2$$

$$h^t = \sigma\left(a^t\right)$$

$$a^t = ux^t + wh^{t-1}$$



$$\frac{\partial L}{\partial w} = \sum_{t=1}^{T} \frac{\partial L}{\partial a^t}\frac{\partial a^t}{\partial w} = \sum_{t=1}^{T} \frac{\partial L}{\partial a^t} h^{t-1} = \sum_{t=1}^{T} \delta^t h^{t-1}$$

$$\delta^t \overset{\text{def}}{=} \frac{\partial L}{\partial a^t} = \frac{\partial L}{\partial h^t}\frac{\partial h^t}{\partial a^t} = \frac{\partial L}{\partial h^t}\dot{\sigma}\left(a^t\right)$$

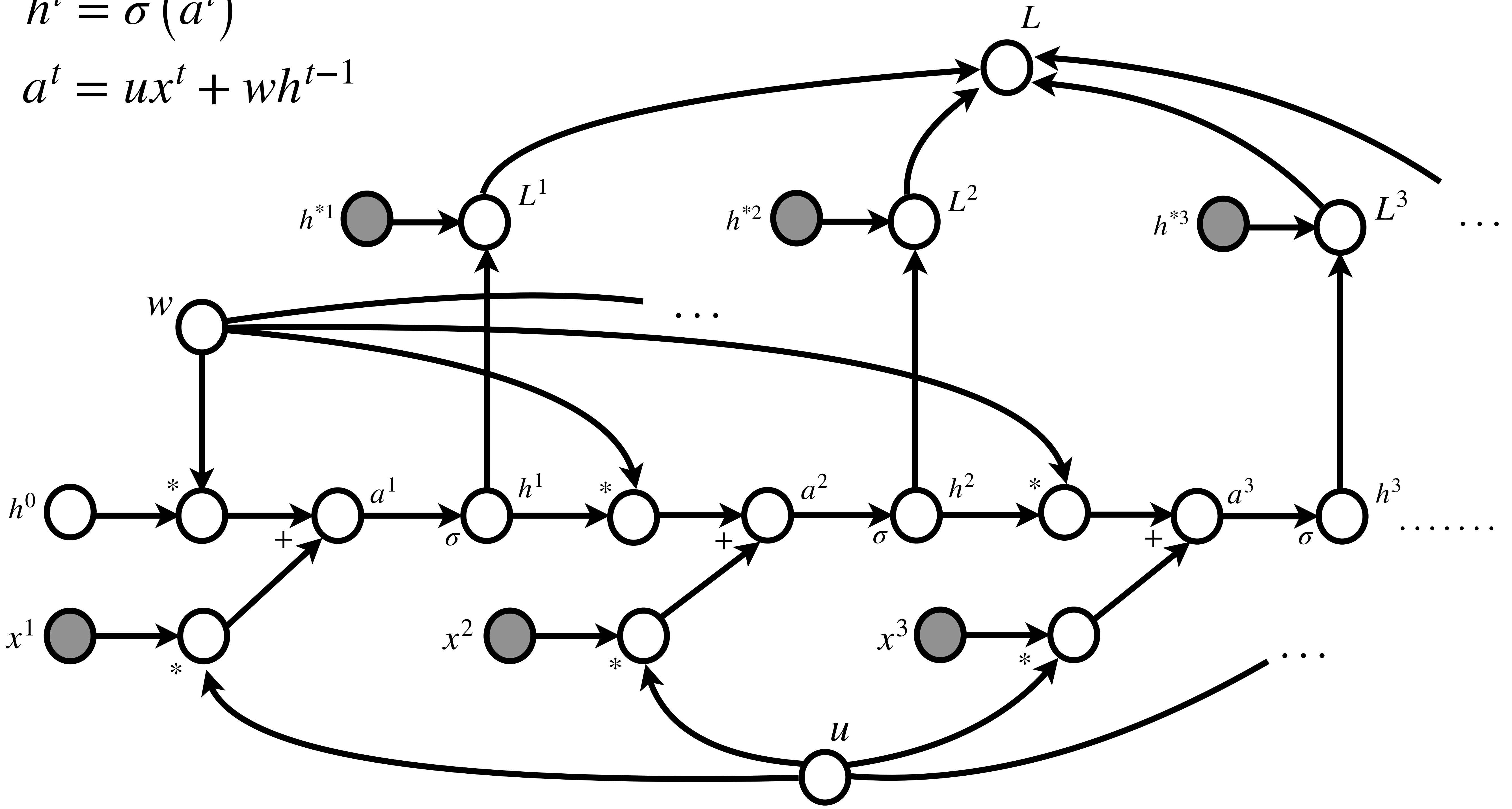
$$\frac{\partial L}{\partial h^t} = \underbrace{\frac{\partial L}{\partial L^t}}_{1}\frac{\partial L^t}{\partial h^t} + \underbrace{\frac{\partial L}{\partial a^{t+1}}}_{\delta^{t+1}}\underbrace{\frac{\partial a^{t+1}}{\partial h^t}}_{w} = \frac{\partial L^t}{\partial h^t} + \delta^{t+1}\cdot w$$

Backpropagation Through Time

$$L = \sum_{t=1}^{T} L^t \quad L^t = \frac{1}{2}\left(h^t - h^{*t}\right)^2$$

$$h^t = \sigma\left(a^t\right)$$

$$a^t = ux^t + wh^{t-1}$$



$$\frac{\partial L}{\partial w} = \sum_{t=1}^{T} \frac{\partial L}{\partial a^t}\frac{\partial a^t}{\partial w} = \sum_{t=1}^{T} \frac{\partial L}{\partial a^t} h^{t-1} = \sum_{t=1}^{T} \delta^t h^{t-1}$$

$$\delta^t \stackrel{\text{def}}{=} \frac{\partial L}{\partial a^t} = \frac{\partial L}{\partial h^t}\frac{\partial h^t}{\partial a^t} = \frac{\partial L}{\partial h^t}\dot{\sigma}\left(a^t\right)$$

$$\frac{\partial L}{\partial h^t} = \frac{\partial L}{\partial L^t}\frac{\partial L^t}{\partial h^t} + \frac{\partial L}{\partial a^{t+1}}\frac{\partial a^{t+1}}{\partial h^t} = \frac{\partial L^t}{\partial h^t} + \delta^{t+1} \cdot w$$

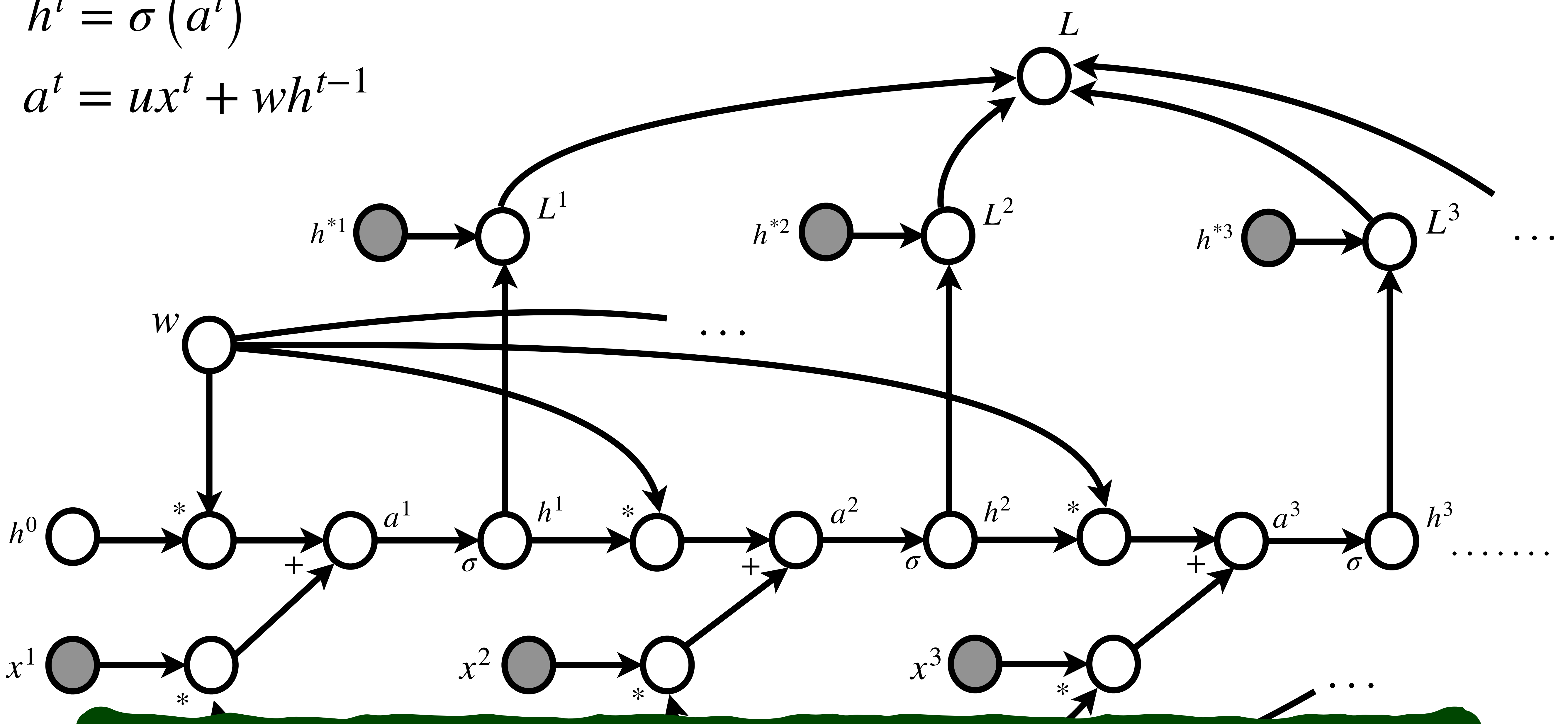
$$\delta^t = \left(\frac{\partial L^t}{\partial h^t} + \delta^{t+1} \cdot w\right) \cdot \dot{\sigma}\left(a^t\right)$$

Backpropagating error through time

Backpropagation Through Time

$$L = \sum_{t=1}^{T} L^t \quad L^t = \frac{1}{2}\left(h^t - h^{*t}\right)^2$$

$$h^t = \sigma\left(a^t\right)$$
$$a^t = ux^t + wh^{t-1}$$



Notes on backpropagation through time:

- Depends on the number of time steps.
- Before each parameter update, we need to wait and store everything until all computations are done.
  - Memory demanding for longer sequences.
  - Alternative: "truncated backprop through time".

$$\frac{\partial L}{\partial w} = \sum_{t=1}^{T} \frac{\partial L}{\partial a^t}\frac{\partial a^t}{\partial w} = \sum_{t=1}^{T} \frac{\partial L}{\partial a^t} h^{t-1} = \sum_{t=1}^{T} \delta^t h^{t-1}$$

$$\delta^t \overset{\text{def}}{=} \frac{\partial L}{\partial a^t} = \frac{\partial L}{\partial h^t}\frac{\partial h^t}{\partial a^t} = \frac{\partial L}{\partial h^t}\dot{\sigma}\left(a^t\right)$$

$$\frac{\partial L}{\partial h^t} = \frac{\partial L}{\partial L^t}\frac{\partial L^t}{\partial h^t} + \frac{\partial L}{\partial a^{t+1}}\frac{\partial a^{t+1}}{\partial h^t} = \frac{\partial L^t}{\partial h^t} + \delta^{t+1}\cdot w$$

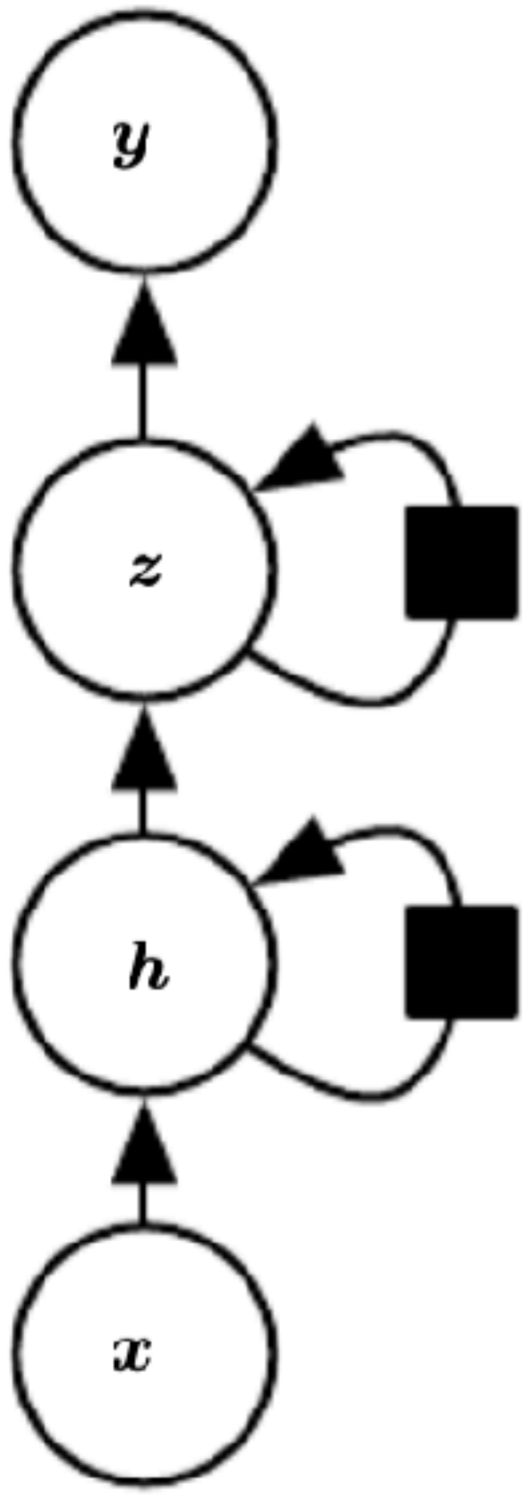
$$\delta^t = \left(\frac{\partial L^t}{\partial h^t} + \delta^{t+1}\cdot w\right)\cdot\dot{\sigma}\left(a^t\right)$$

Backpropagating error through time

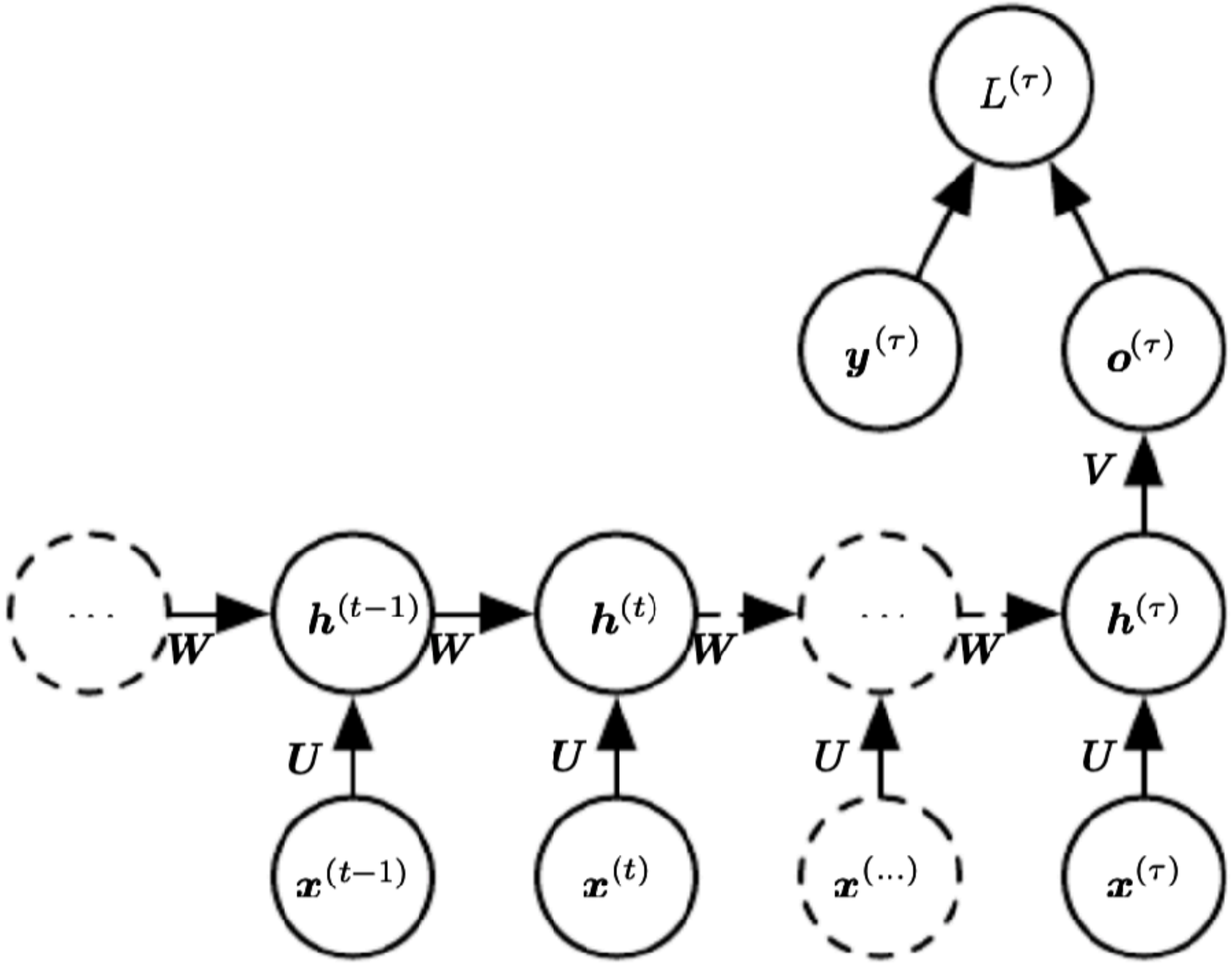
Today

- [ ] Recurrent Neural Networks
    - [x] Simple RNNs
    - [x] Backpropagation Through Time
    - [ ] Architectural Variants
    - [ ] Gated RNNs (LSTM & GRU)

(1) Deep Recurrent Network



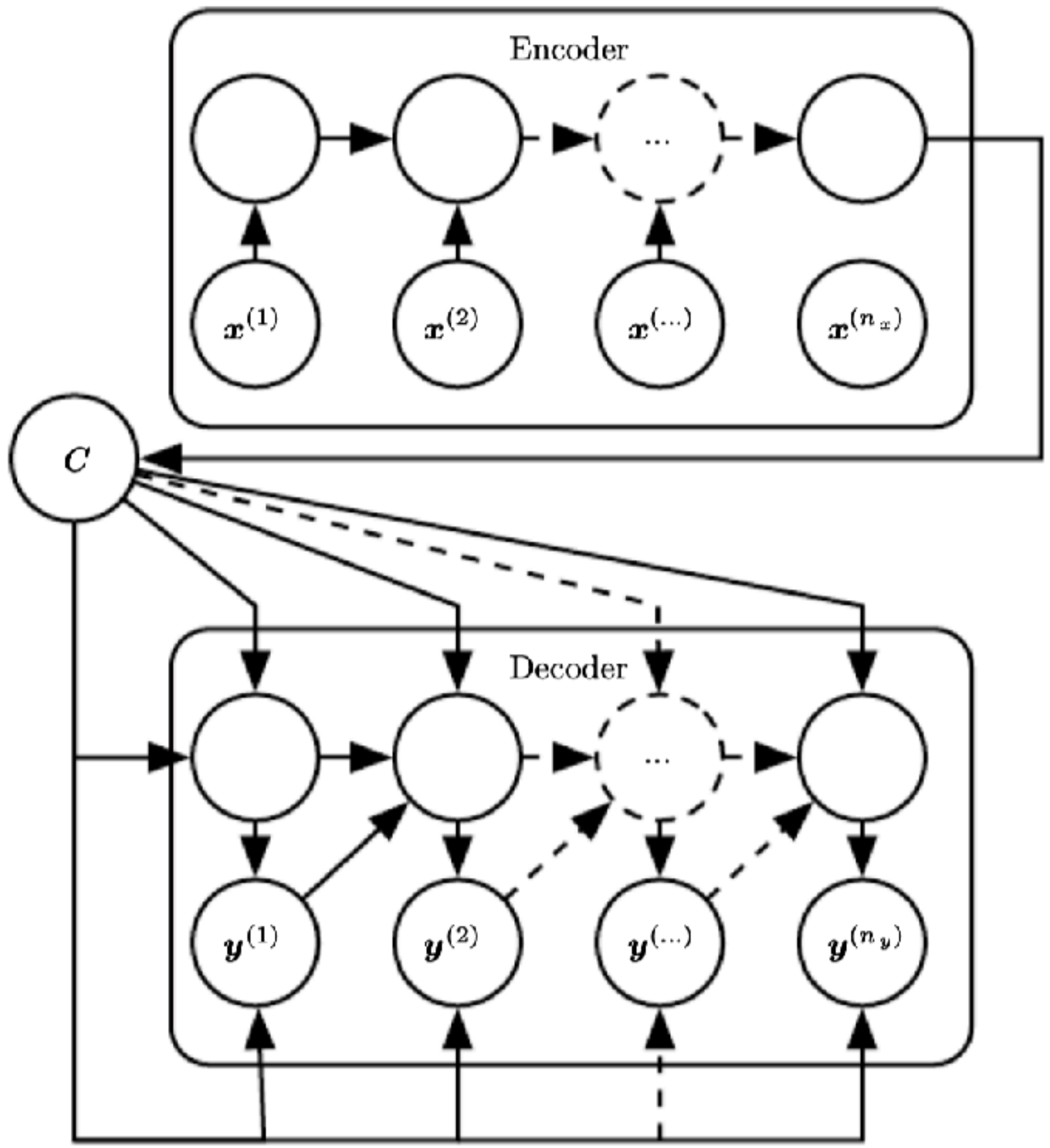
(2) Single output at the end of the sequence



Some potential use cases:

- Text classification.
- Detecting anomalies from a given measurement sequence.
- Summarize a sequence and produce a fixed-size representation, then use this as input for further processing.

(3) Encoder-Decoder (Sequence-to-Sequence) Architecture



- If the output sequence does not have the same length as input sequence, e.g. in language translation.

Input: Sequence $\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(n_x)}$

Output: Sequence $\mathbf{y}^{(1)}, \dots, \mathbf{y}^{(n_y)}$

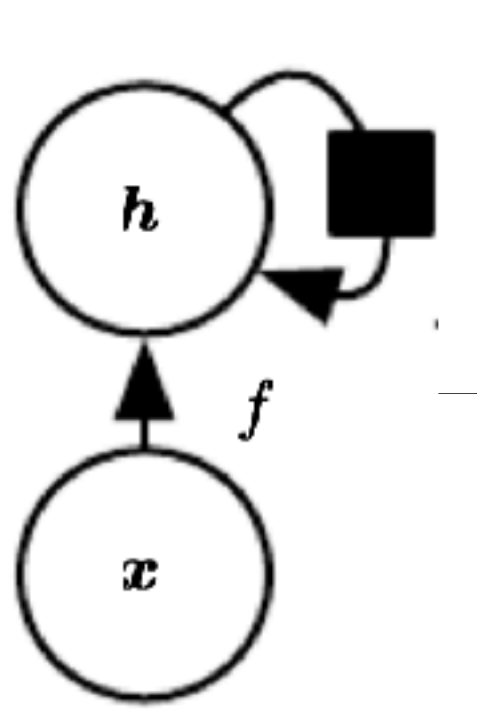
Encoder: Processes input and emits the context C, typically a simple function of its final hidden state.

Decoder: Generates the output sequence conditioned on this context C.

Today

- [ ] Recurrent Neural Networks
  - [x] Simple RNNs
  - [x] Backpropagation Through Time
  - [x] Architectural Variants
  - [ ] Gated RNNs (LSTM & GRU)

Problem: Challenging long-term dependencies



- *Vanishing/exploding gradients* problem:

Gradients propagated over many stages tend to either vanish or explode.

We can mitigate *exploding gradients* by gradient clipping.

What about *vanishing gradients*?

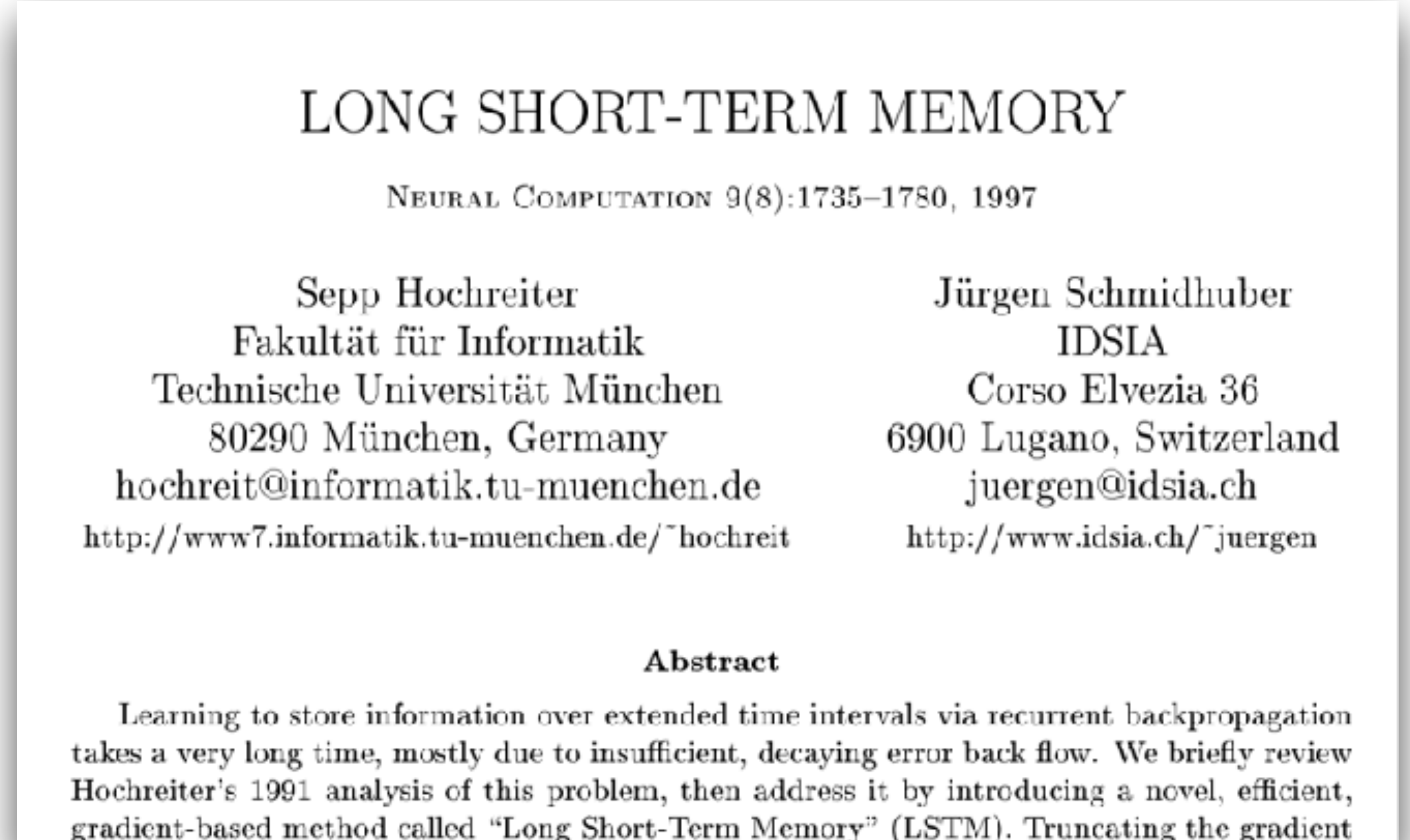
- In order to store memories in a way that is robust to small perturbations, the RNN must enter a region of parameter space where gradients vanish.
- The gradient of a long term interaction has exponentially smaller magnitude than that of a short term interaction.

Long Short-Term Memory (LSTM)

Idea: **Create paths through time that have derivatives that neither vanish nor explode.**

LSTM basic unit: A memory cell

- A linear neuron with a unit-weight self loop, where:
  - an **input gate** controls whether to load something in,
  - an **output gate** controls whether to make the content available to others,
  - a **forget gate** controls whether to forget the content.


LONG SHORT-TERM MEMORY

NEURAL COMPUTATION 9(8):1735–1780, 1997

Sepp Hochreiter
Fakultät für Informatik
Technische Universität München
80290 München, Germany
hochreit@informatik.tu-muenchen.de
http://www7.informatik.tu-muenchen.de/~hochreit

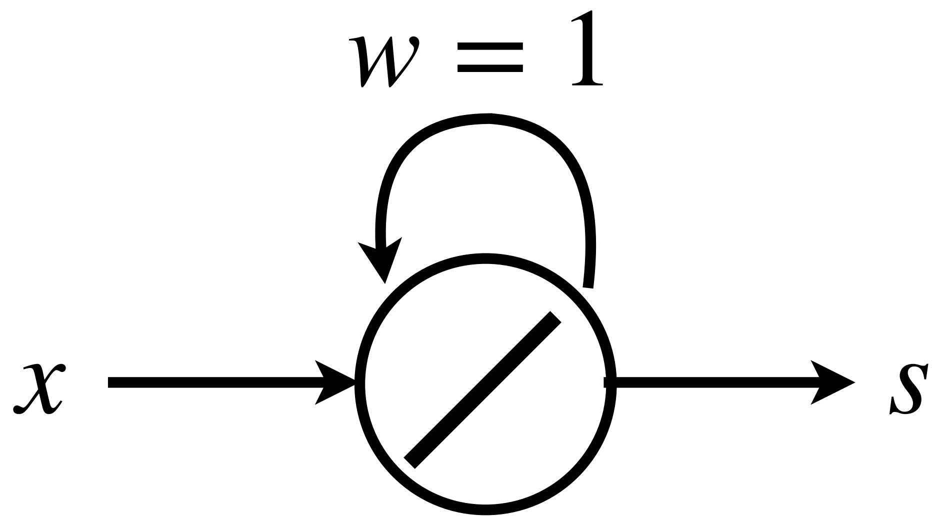
Jürgen Schmidhuber
IDSIA
Corso Elvezia 36
6900 Lugano, Switzerland
juergen@idsia.ch
http://www.idsia.ch/~juergen

Abstract

Learning to store information over extended time intervals via recurrent backpropagation takes a very long time, mostly due to insufficient, decaying error back flow. We briefly review Hochreiter's 1991 analysis of this problem, then address it by introducing a novel, efficient, gradient-based method called "Long Short-Term Memory" (LSTM). Truncating the gradient

[S. Hochreiter and J. Schmidhuber. "Long short-term memory." *Neural Computation*, 1997.]

LSTM - Memory Cell

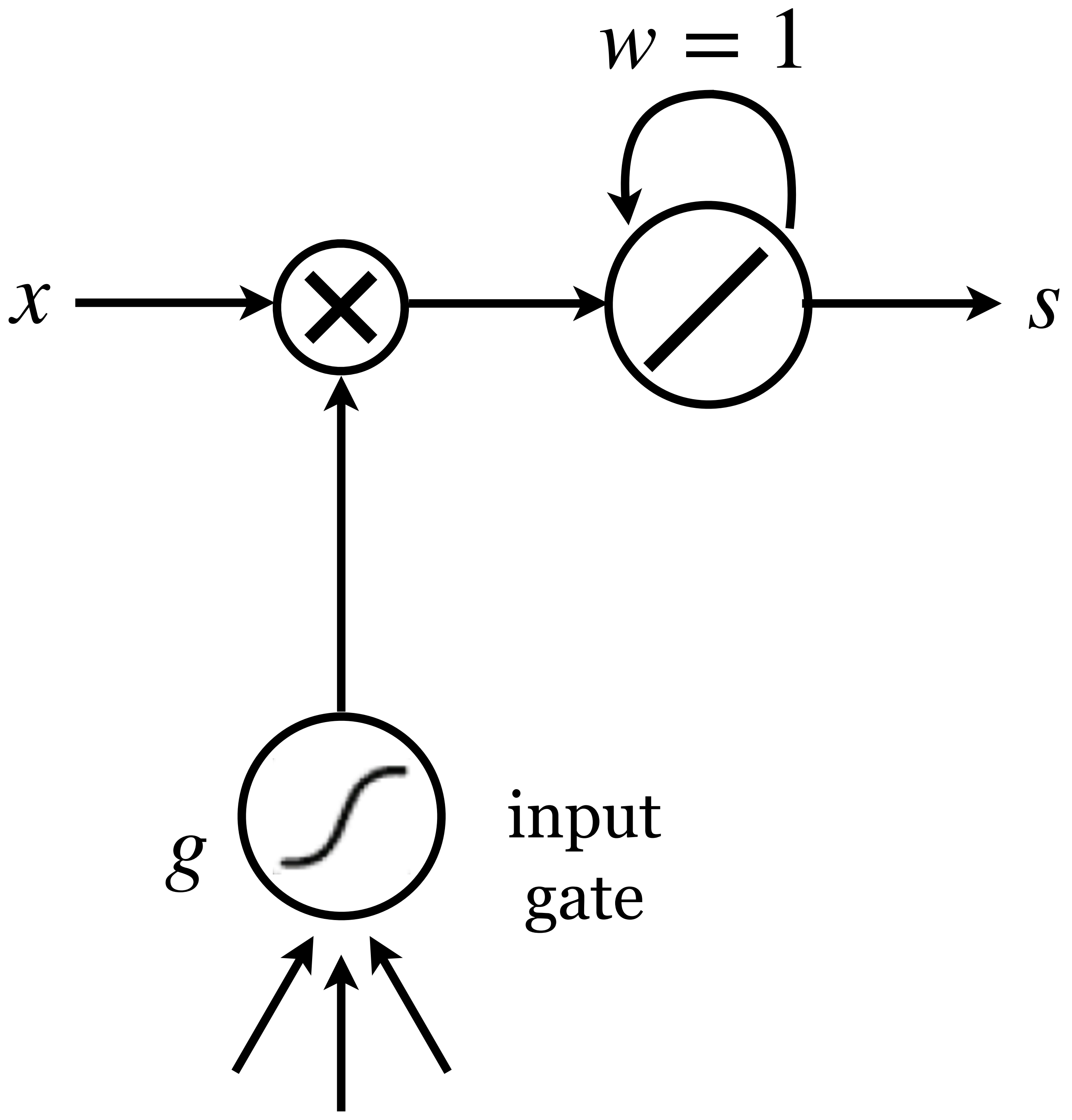


$$s^t = s^{t-1} + x^t$$

[S. Hochreiter and J. Schmidhuber. "Long short-term memory." *Neural Computation*, 1997.]

LSTM - Memory Cell

Input Gate:

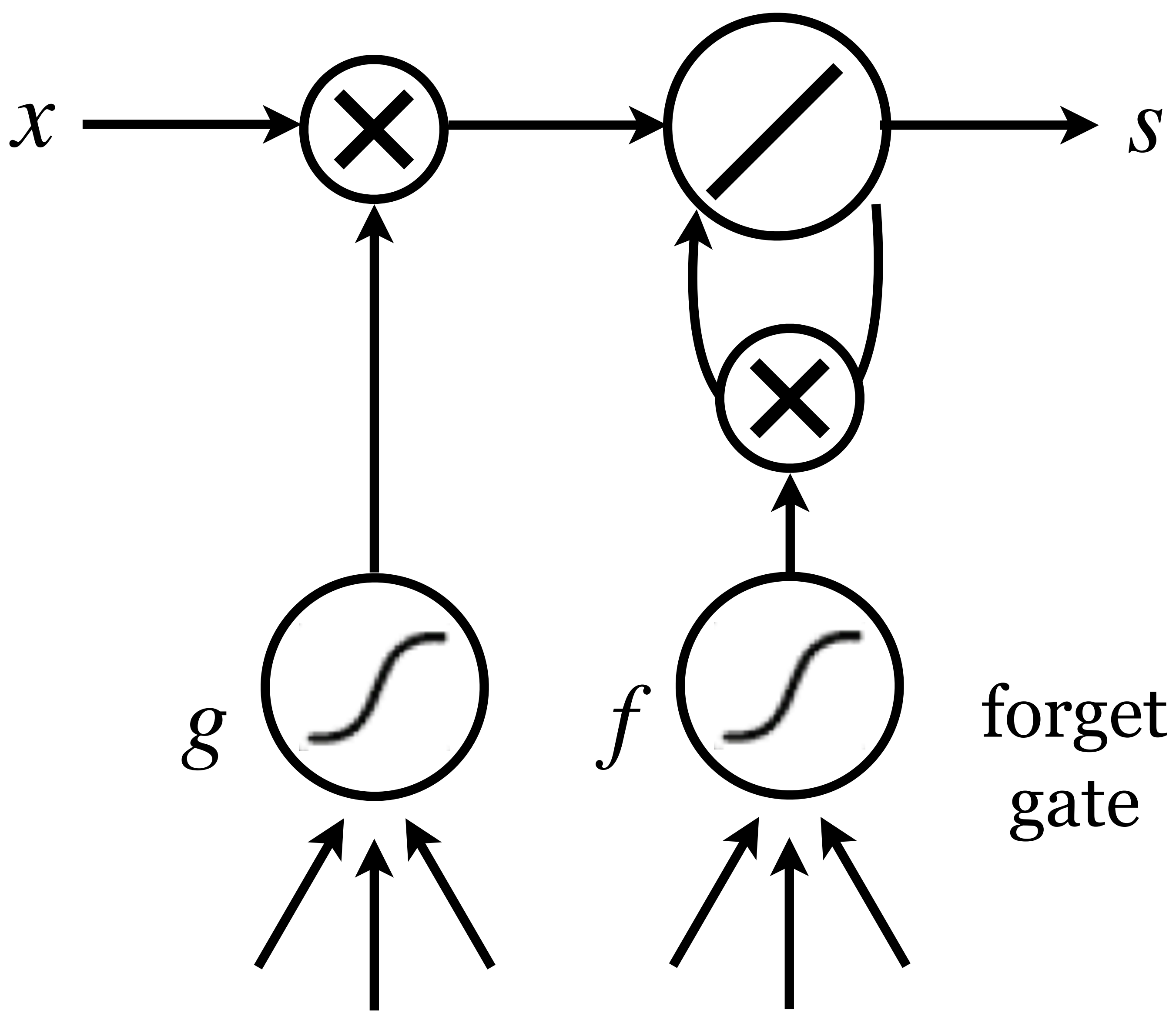


$$s^t = s^{t-1} + g^t x^t$$

[S. Hochreiter and J. Schmidhuber. "Long short-term memory." *Neural Computation*, 1997.]

LSTM - Memory Cell

Forget Gate:

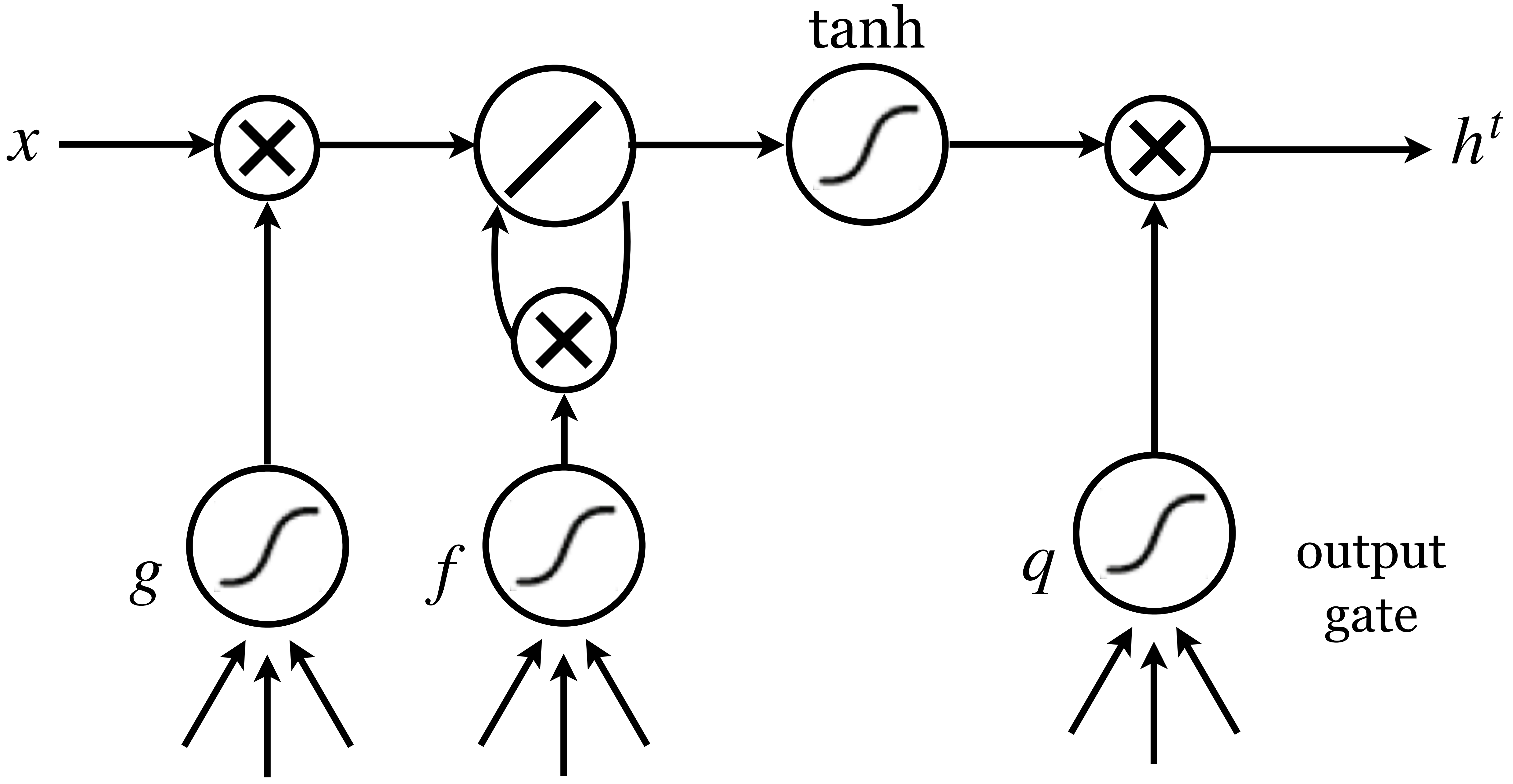


$$s^t = f^t s^{t-1} + g^t x^t$$

[S. Hochreiter and J. Schmidhuber. "Long short-term memory." *Neural Computation*, 1997.]

LSTM - Memory Cell

Output Gate:

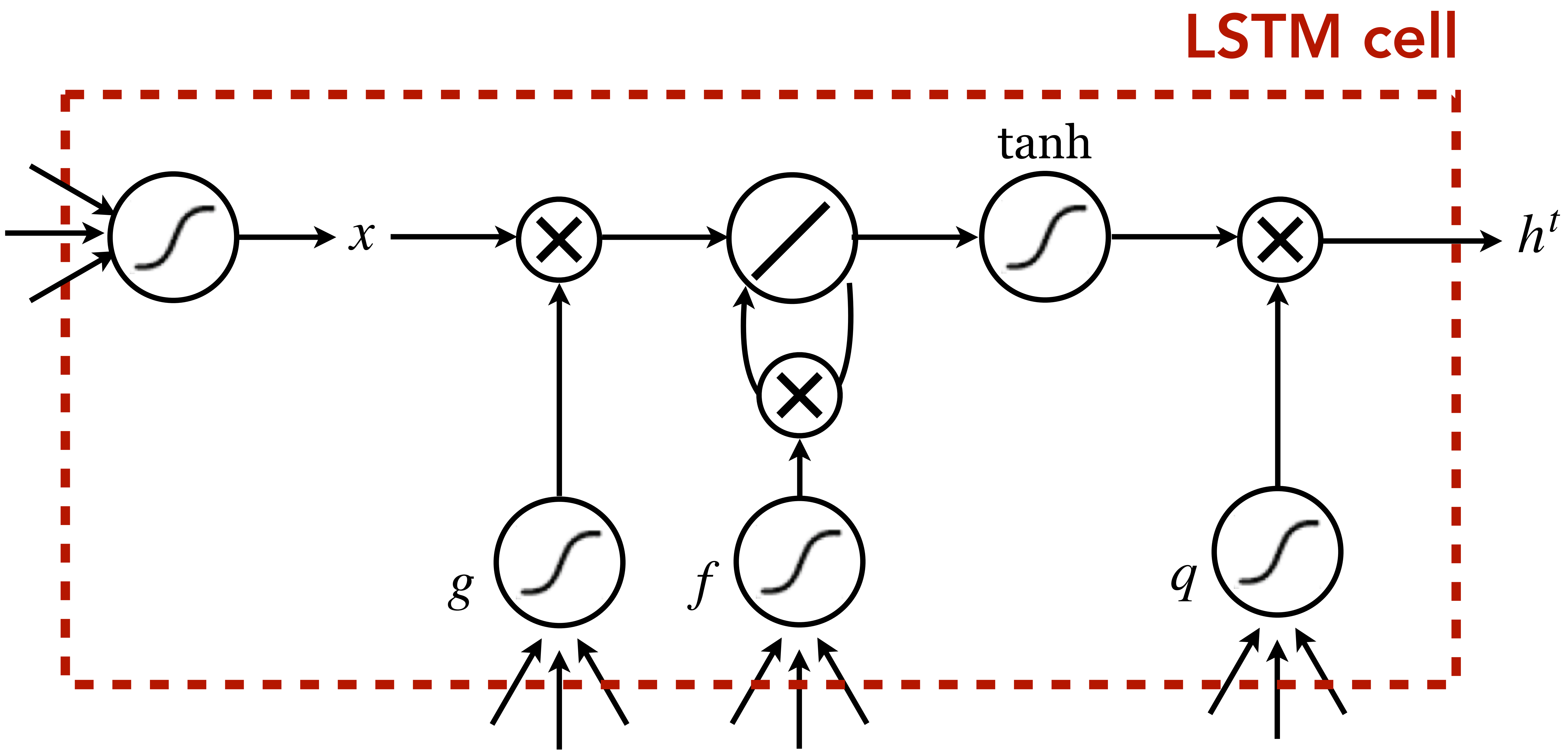


$$s^t = f^t s^{t-1} + g^t x^t$$

$$h^t = q^t \cdot \tanh\left(s^t\right)$$

[S. Hochreiter and J. Schmidhuber. "Long short-term memory." *Neural Computation*, 1997.]

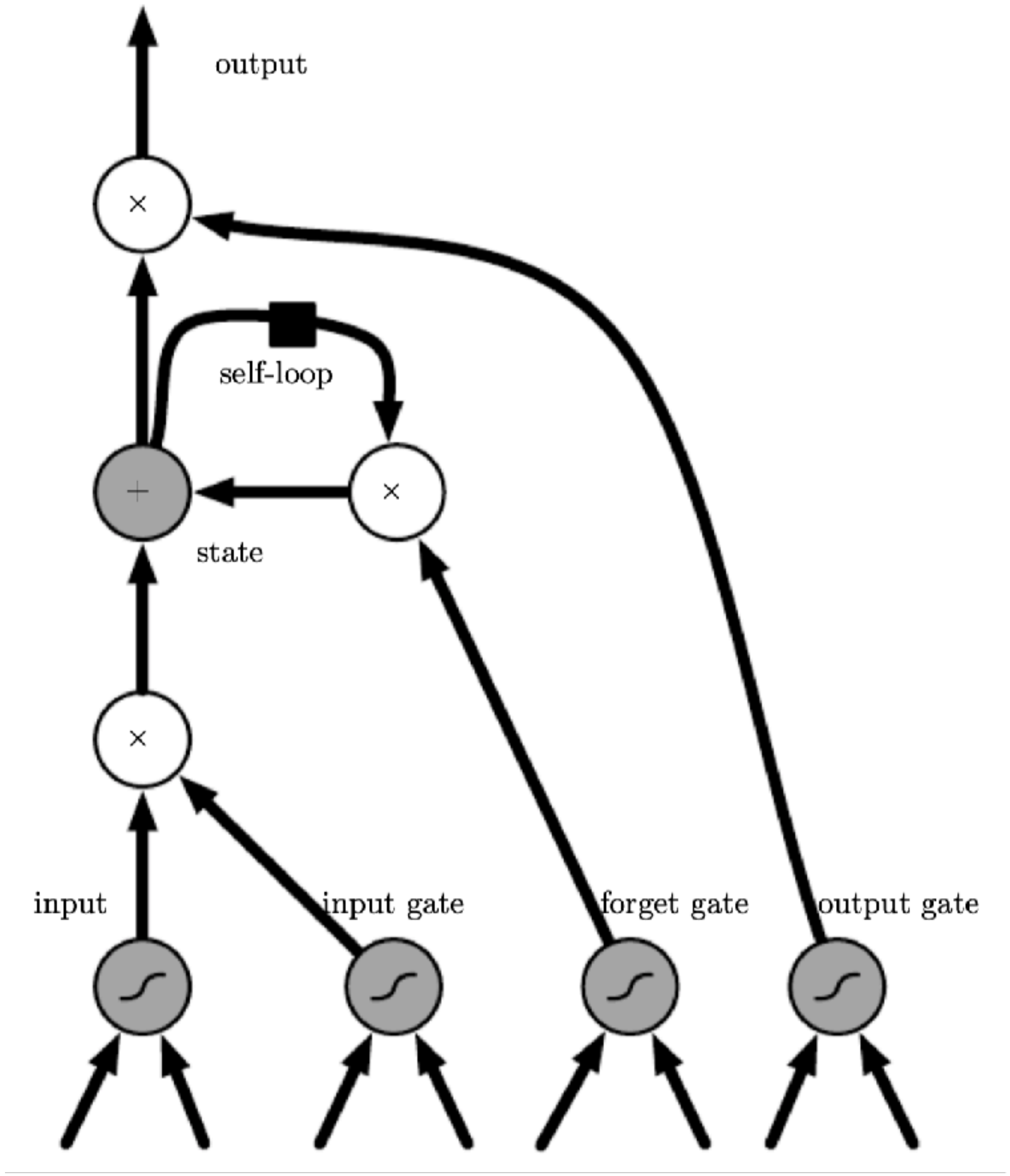
LSTM - Memory Cell



$$s^t = f^t s^{t-1} + g^t x^t$$
$$h^t = q^t \cdot \tanh\left(s^t\right)$$

[S. Hochreiter and J. Schmidhuber. "Long short-term memory." *Neural Computation*, 1997.]

LSTM - Memory Cell - Details



$$h_i^{(t)} = \tanh\left(s_i^{(t)}\right) q_i^{(t)}$$

$$s_i^{(t)} = f_i^{(t)} s_i^{(t-1)} + g_i^{(t)} \sigma\left(b_i + \sum_j U_{i,j} x_j^{(t)} + \sum_j W_{i,j} h_j^{(t-1)}\right)$$

$$g_i^{(t)} = \sigma\left(b_i^g + \sum_j U_{i,j}^g x_j^{(t)} + \sum_j W_{i,j}^g h_j^{(t-1)}\right)$$

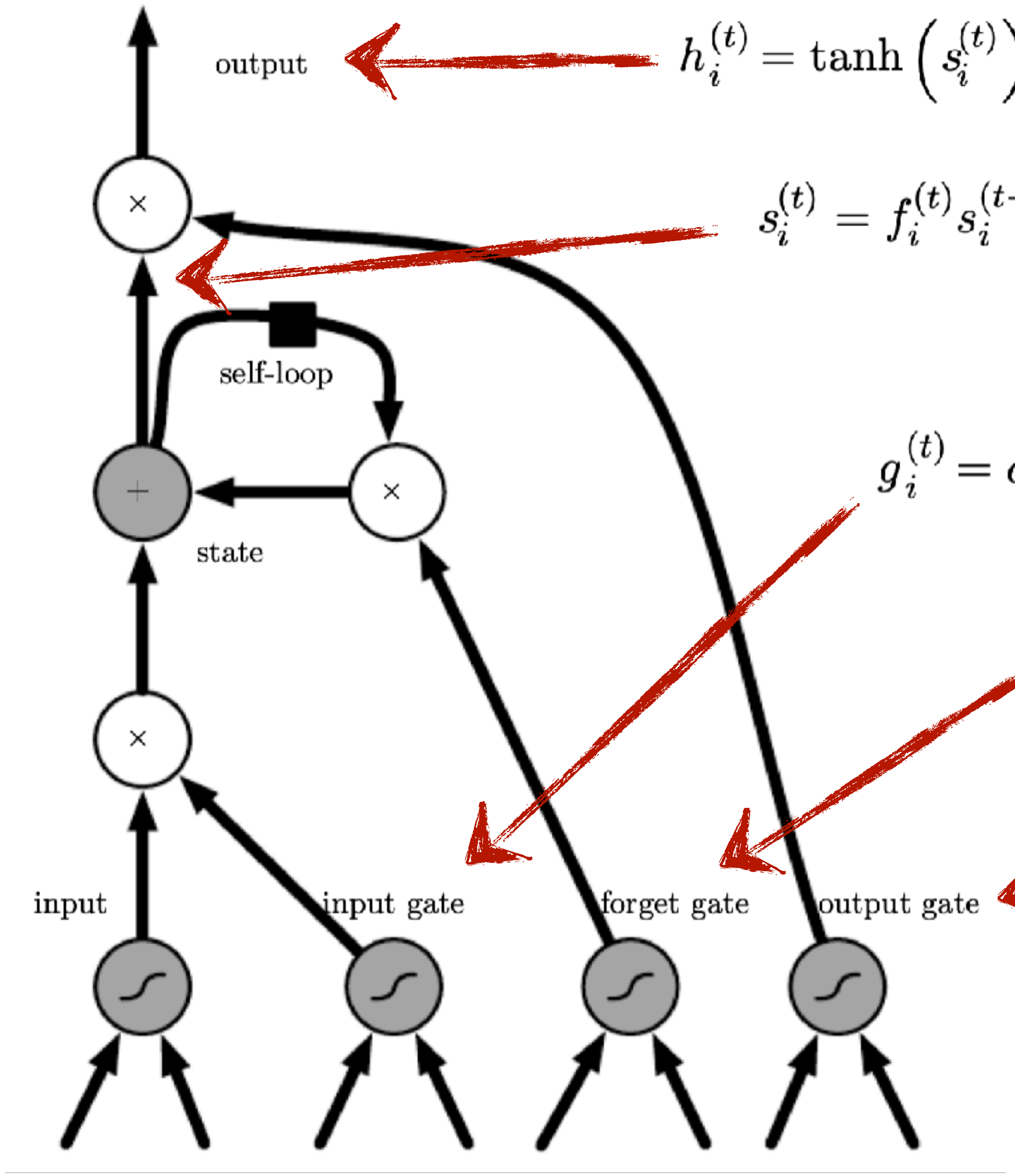
$$f_i^{(t)} = \sigma\left(b_i^f + \sum_j U_{i,j}^f x_j^{(t)} + \sum_j W_{i,j}^f h_j^{(t-1)}\right)$$

$$q_i^{(t)} = \sigma\left(b_i^o + \sum_j U_{i,j}^o x_j^{(t)} + \sum_j W_{i,j}^o h_j^{(t-1)}\right)$$

$\boldsymbol{x}^{(t)}$ is the current input vector.

$\boldsymbol{h}^{(t)}$ is the current hidden layer vector.

LSTM - Memory Cell - Details



$$h_i^{(t)} = \tanh\left(s_i^{(t)}\right) q_i^{(t)}$$

$$s_i^{(t)} = f_i^{(t)} s_i^{(t-1)} + g_i^{(t)} \sigma\left(b_i + \sum_j U_{i,j} x_j^{(t)} + \sum_j W_{i,j} h_j^{(t-1)}\right)$$

$$g_i^{(t)} = \sigma\left(b_i^g + \sum_j U_{i,j}^g x_j^{(t)} + \sum_j W_{i,j}^g h_j^{(t-1)}\right)$$

$$f_i^{(t)} = \sigma\left(b_i^f + \sum_j U_{i,j}^f x_j^{(t)} + \sum_j W_{i,j}^f h_j^{(t-1)}\right)$$

$$q_i^{(t)} = \sigma\left(b_i^o + \sum_j U_{i,j}^o x_j^{(t)} + \sum_j W_{i,j}^o h_j^{(t-1)}\right)$$

$\boldsymbol{x}^{(t)}$ is the current input vector.

$\boldsymbol{h}^{(t)}$ is the current hidden layer vector.

Gated Recurrent Unit (GRU)

A simplified version of an LSTM cell.

- an **update gate** : u simultaneously controls the forgetting factor and the input gating.
- a **reset gate** : r controls which parts of the hidden state are used for the update.

$$h_i^{(t)} = u_i^{(t-1)} h_i^{(t-1)} + (1 - u_i^{(t-1)}) \sigma \left(b_i + \sum_j U_{i,j} x_j^{(t)} + \sum_j W_{i,j} r_j^{(t-1)} h_j^{(t-1)} \right)$$

- The output values of u_i^t and r_j^t are computed in the usual way.

Recall LSTMs: $h_i^{(t)} = \tanh\left(s_i^{(t)}\right) q_i^{(t)}$

$$s_i^{(t)} = f_i^{(t)} s_i^{(t-1)} + g_i^{(t)} \sigma \left(b_i + \sum_j U_{i,j} x_j^{(t)} + \sum_j W_{i,j} h_j^{(t-1)} \right)$$

- GRUs have only two gates (instead of three with LSTMs).
- A single “update gate". (LSTMs: input & forget gates).
- Therefore usually less parameters than LSTMs.

Today

- ☑ Recurrent Neural Networks
  - ☑ Simple RNNs
  - ☑ Backpropagation Through Time
  - ☑ Architectural Variants
  - ☑ Gated RNNs (LSTM & GRU)

Questions?