DS-GA 3001.008 Probabilistic time series analysis

L10. Intro to RNNs

Instructor: Cristina Savin

NYU, CNS & CDS

The probabilistic ‘pipeline’

Domain-specific knowledge, Question

Data

Useful answers

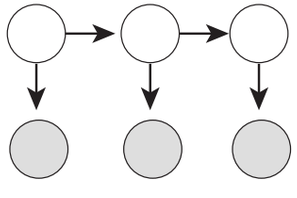
Make assumptions → **Discover patterns** → **Predictions**

$\mathrm{P}(\mathbf{x}|\mathbf{z},\theta)$

$\theta^* = \mathrm{argmax}_\theta \mathcal{L}$

$\mathrm{P}(\mathbf{z}|\mathbf{x},\theta^*)$

observations, latents, hyperparameters dependences

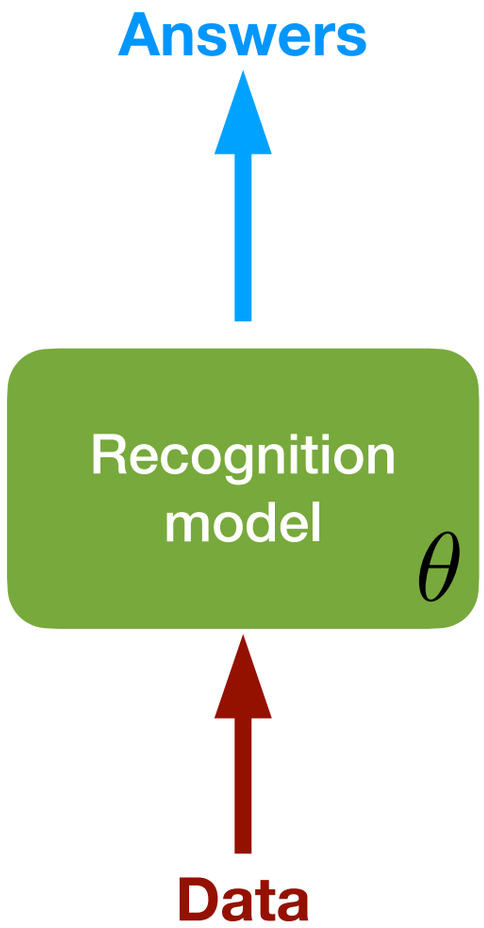


learn model

posterior inference

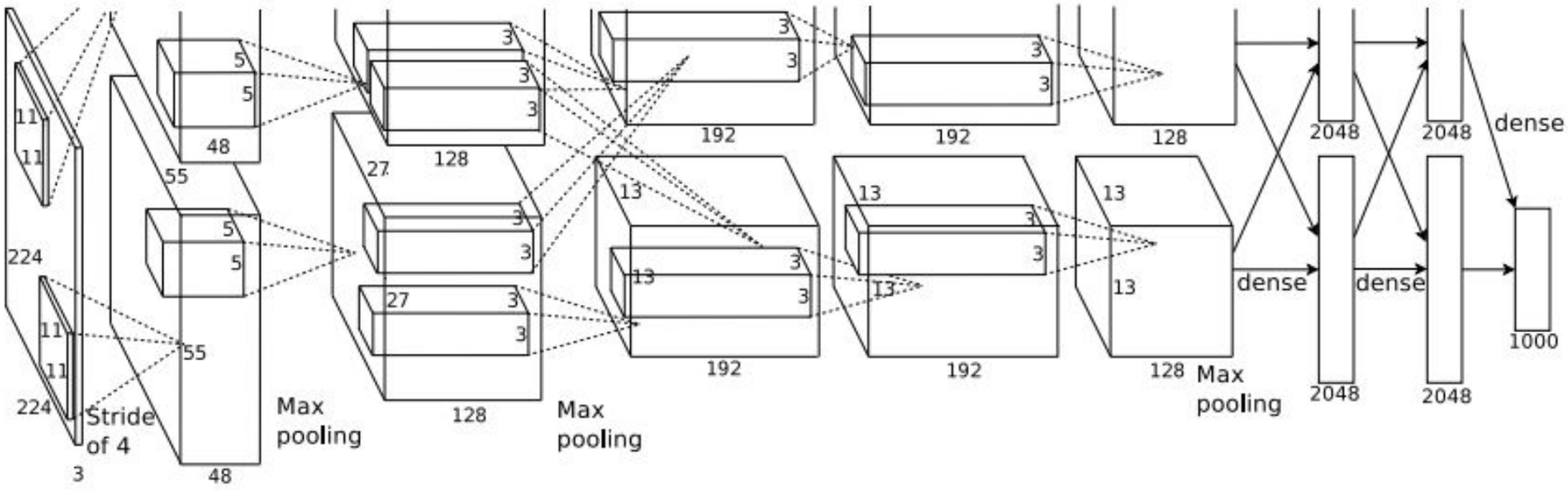
Pipeline separates **assumptions**, **computation**, **applications**

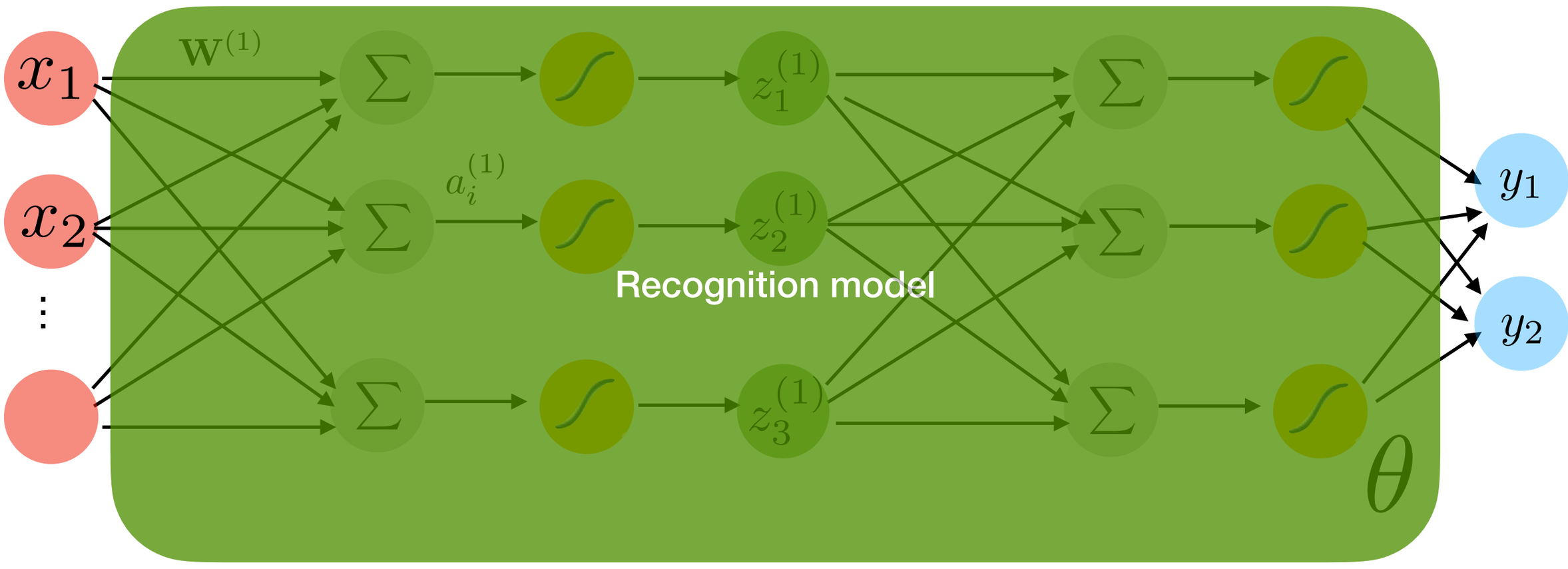
The alternative: just build a recognition model from the get go



Neural networks: flexible input-out map

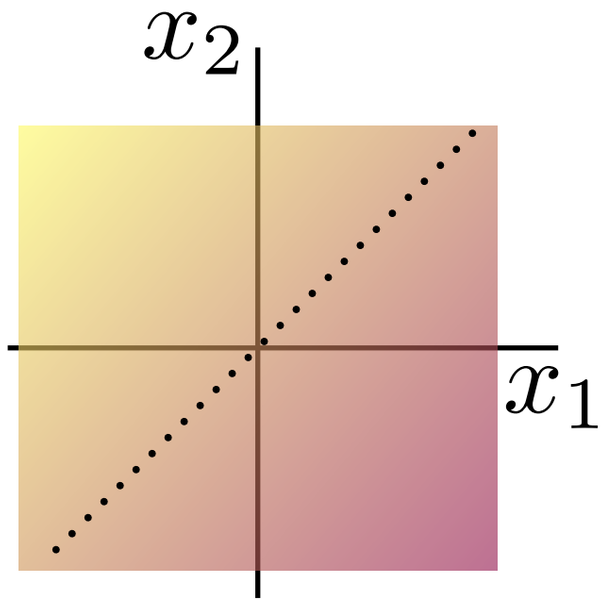
Backpropagation: how to train such models



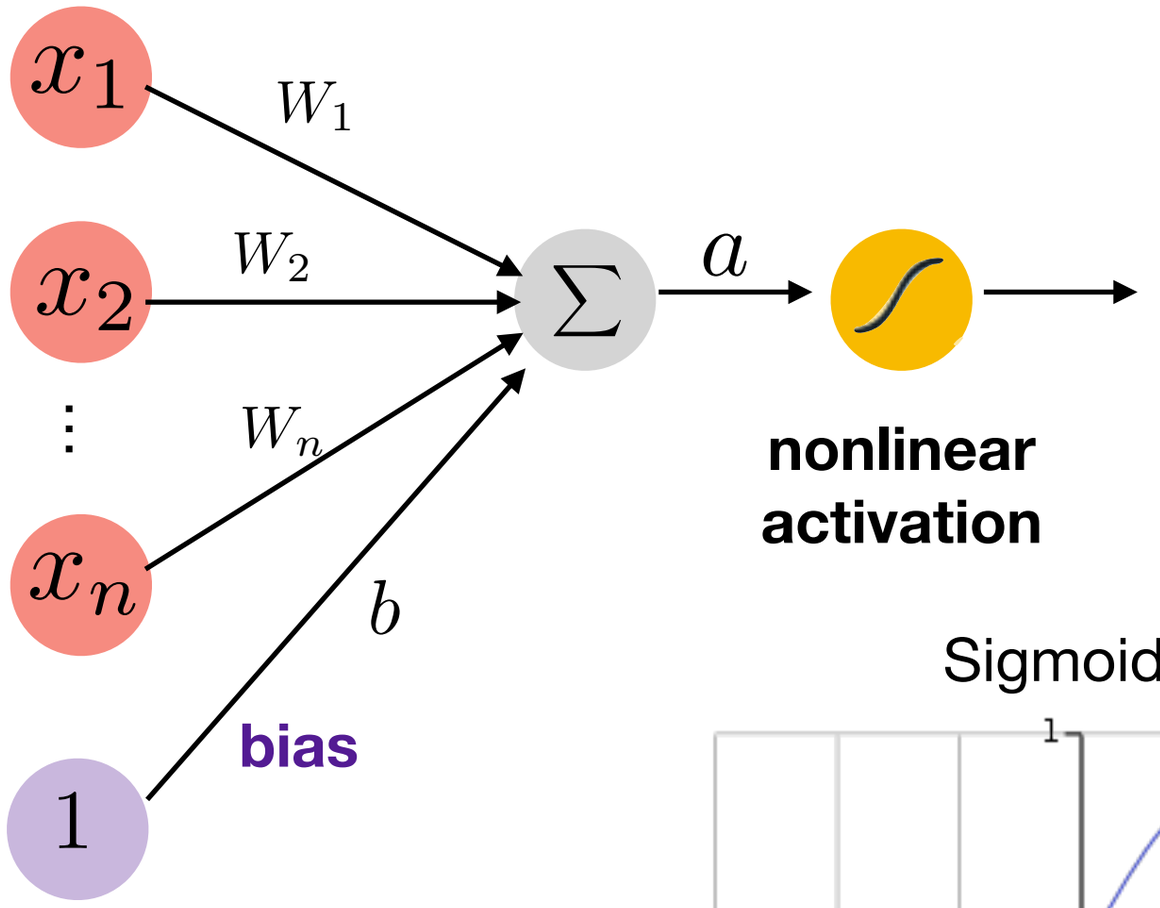


A cascade of linear-nonlinear steps-> a differentiable map

The basic building block: perceptron



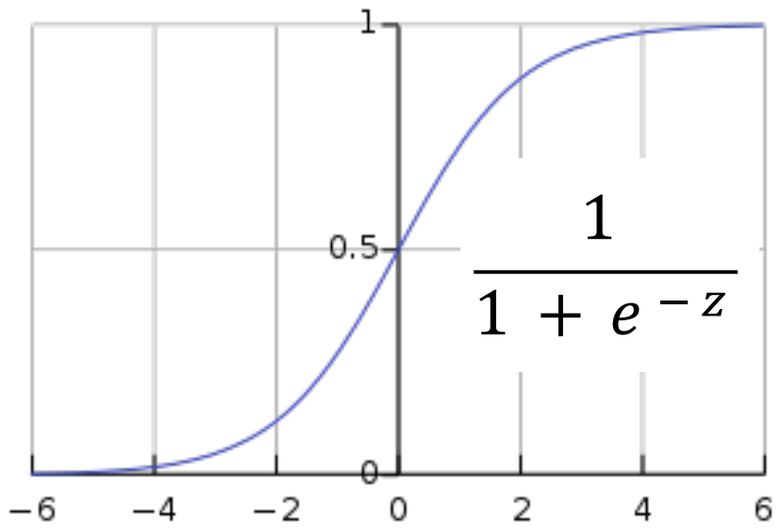
inputs



$$y = f(\mathbf{W}\mathbf{x})$$

nonlinear activation

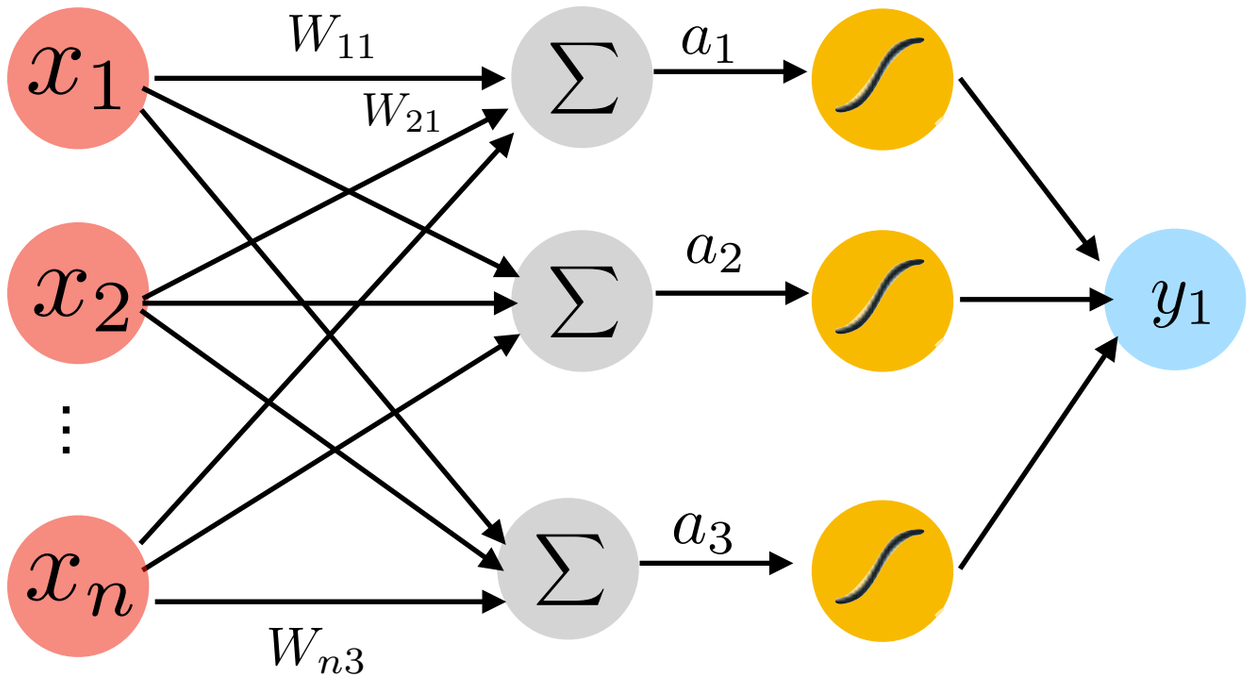
Sigmoid



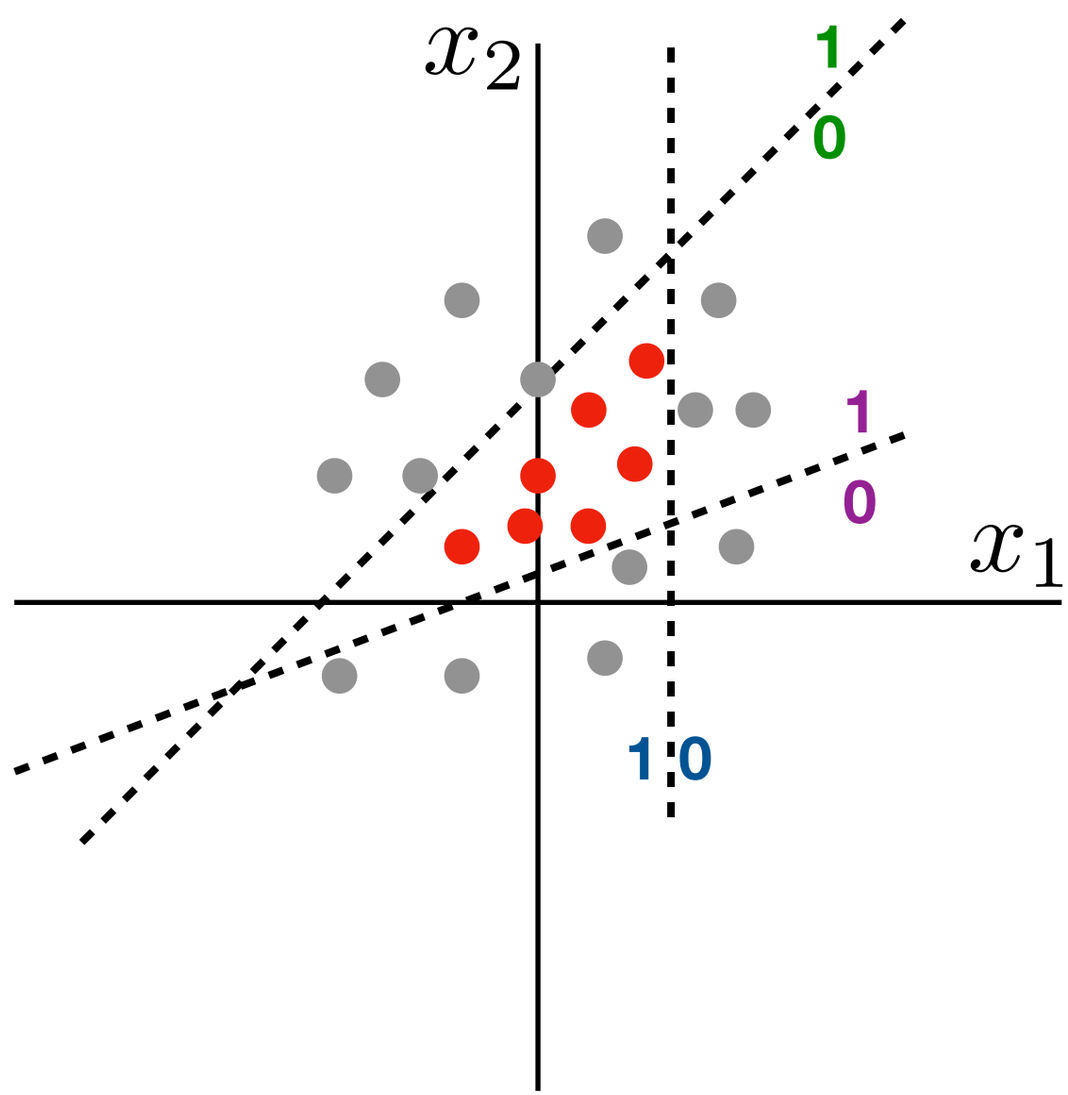
tanh: $\frac{e^{z}-e^{-z}}{e^{z}+e^{-z}}$

ReLu: $\max(0, z)$

One layer



Efficient partitioning of input space



Universal approximation theorem (Hornik, ’91)

“A single hidden layer neural network with a linear output unit can approximate any continuous function arbitrarily well, given enough hidden units.”

This works for any standard nonlinearities f.

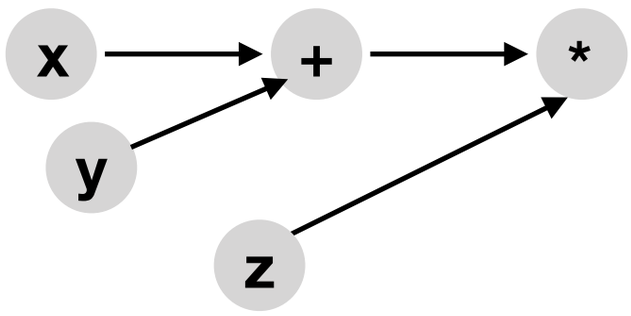
Caveat: It does not necessarily mean that there exists a learning algorithm that finds the correct solution

How do we compute the gradient?

$$\frac{\partial J(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}$$

Simple trick: repeatedly apply the chain rule

Let's take some simple examples (on the board)

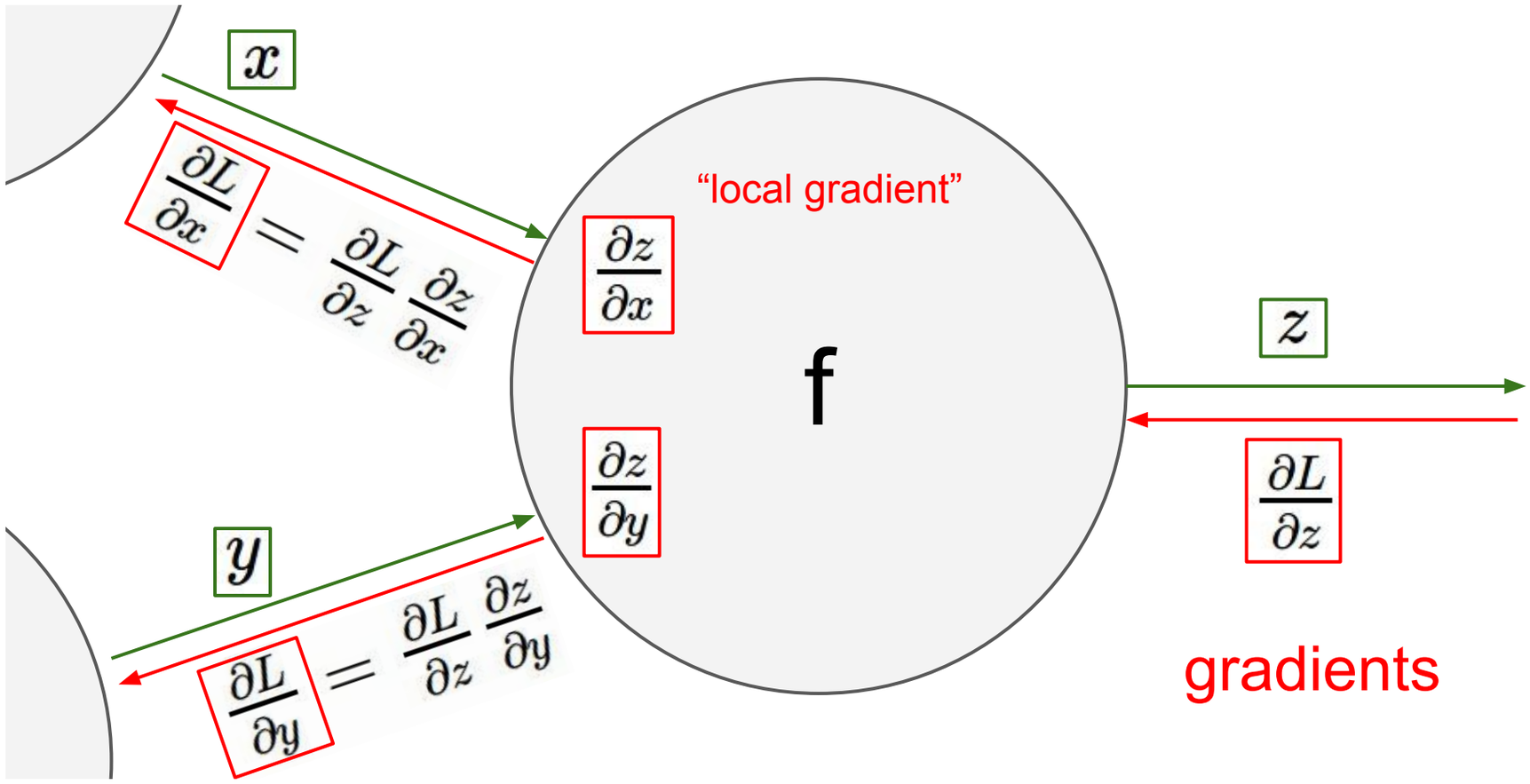


$$f(x, y, z) = (x + y)z$$

$$\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$$

Gentle introduction: "3 blue 1 brown backpropagation"

Backpropagation



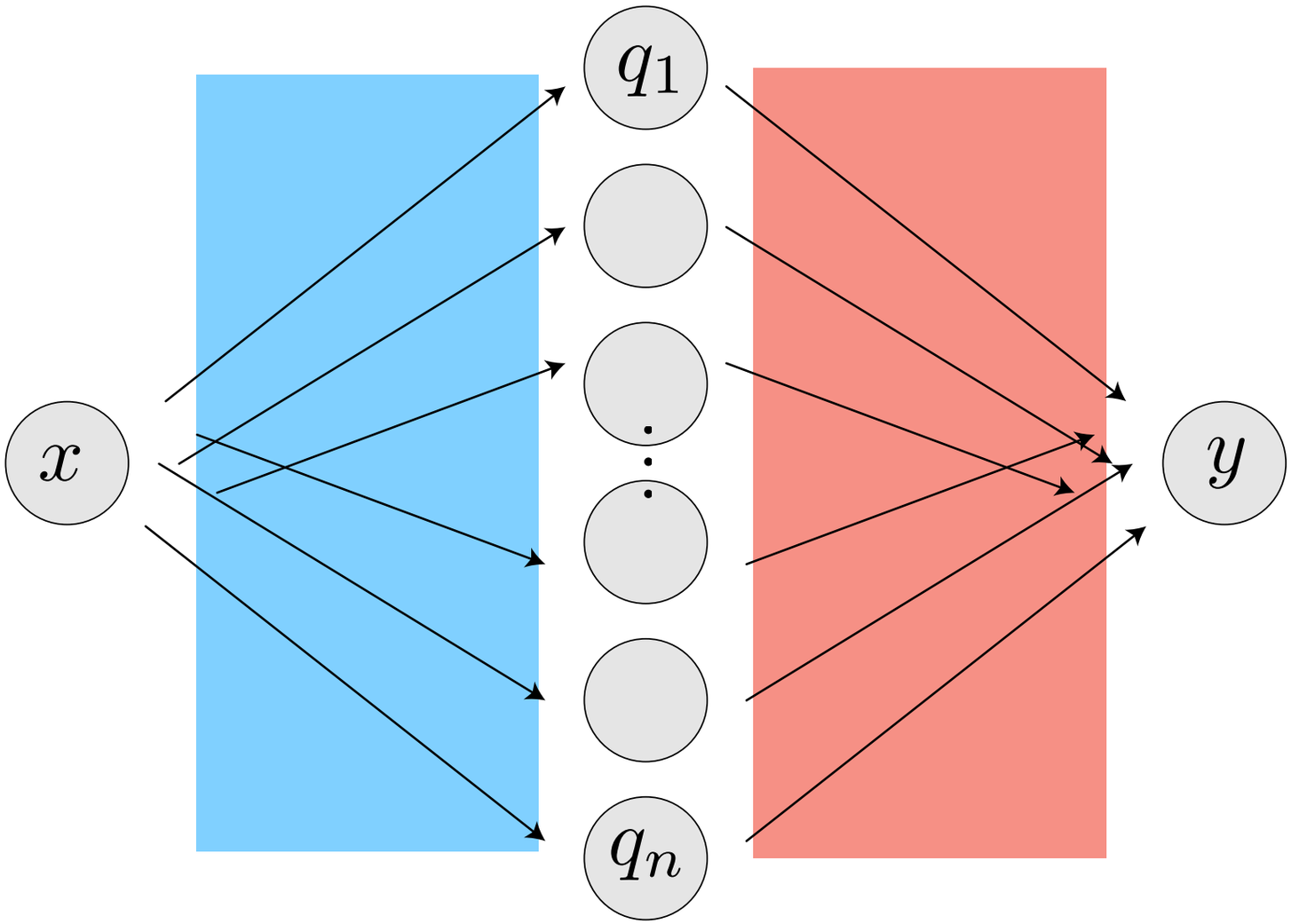
http://cs231n.github.io/optimization-2/

Key facts

$$y = W_1 x_1 + W_2 x_2 + W_3 x_3$$

$$\frac{\partial y}{\partial x_j} = W_j$$

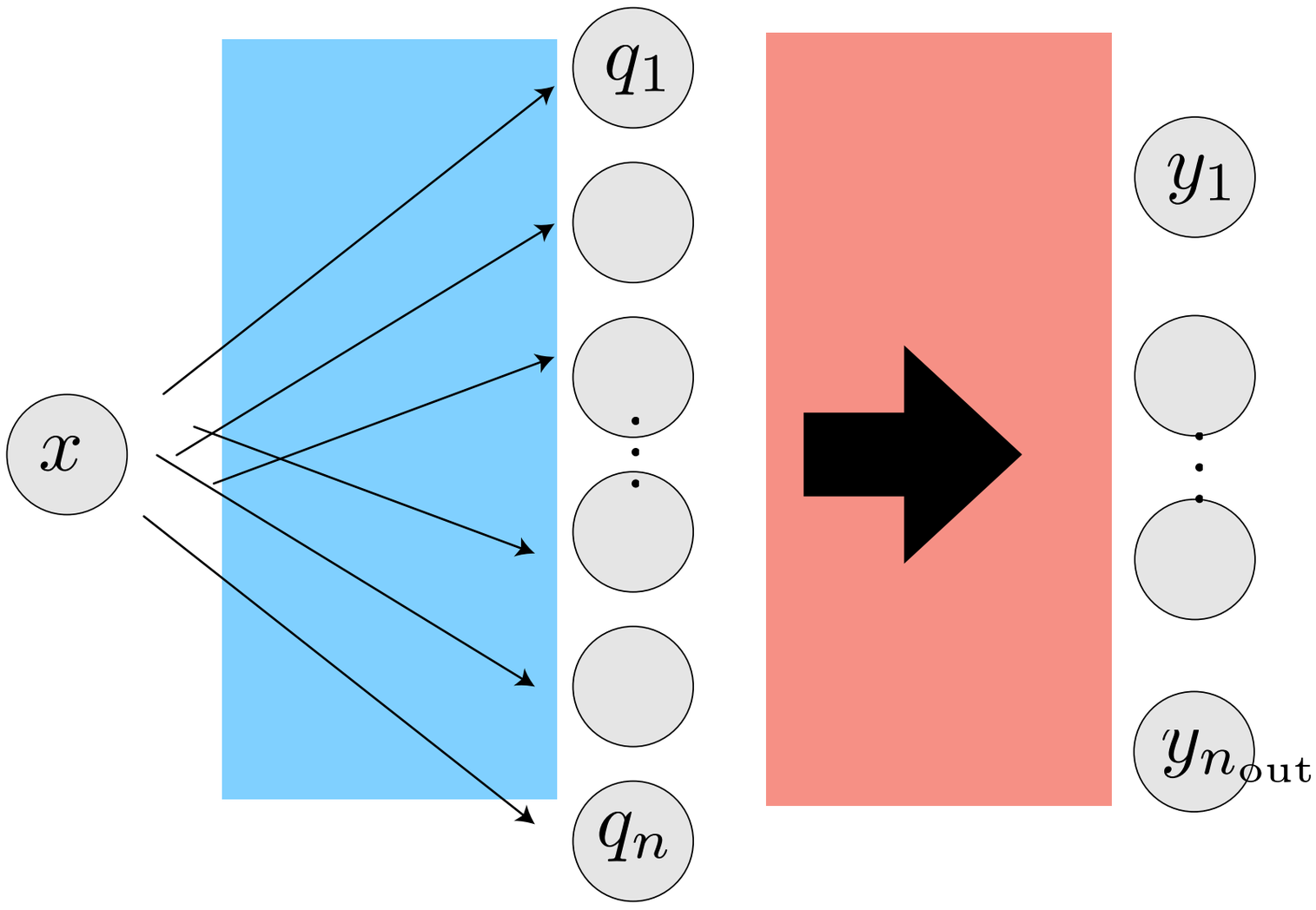
Multivariate chain rule



$$\frac{\partial y}{\partial x} = \sum_{k=1}^{n} \frac{\partial y}{\partial q_k} \frac{\partial q_k}{\partial x}$$

"dot product"

Many output, one input

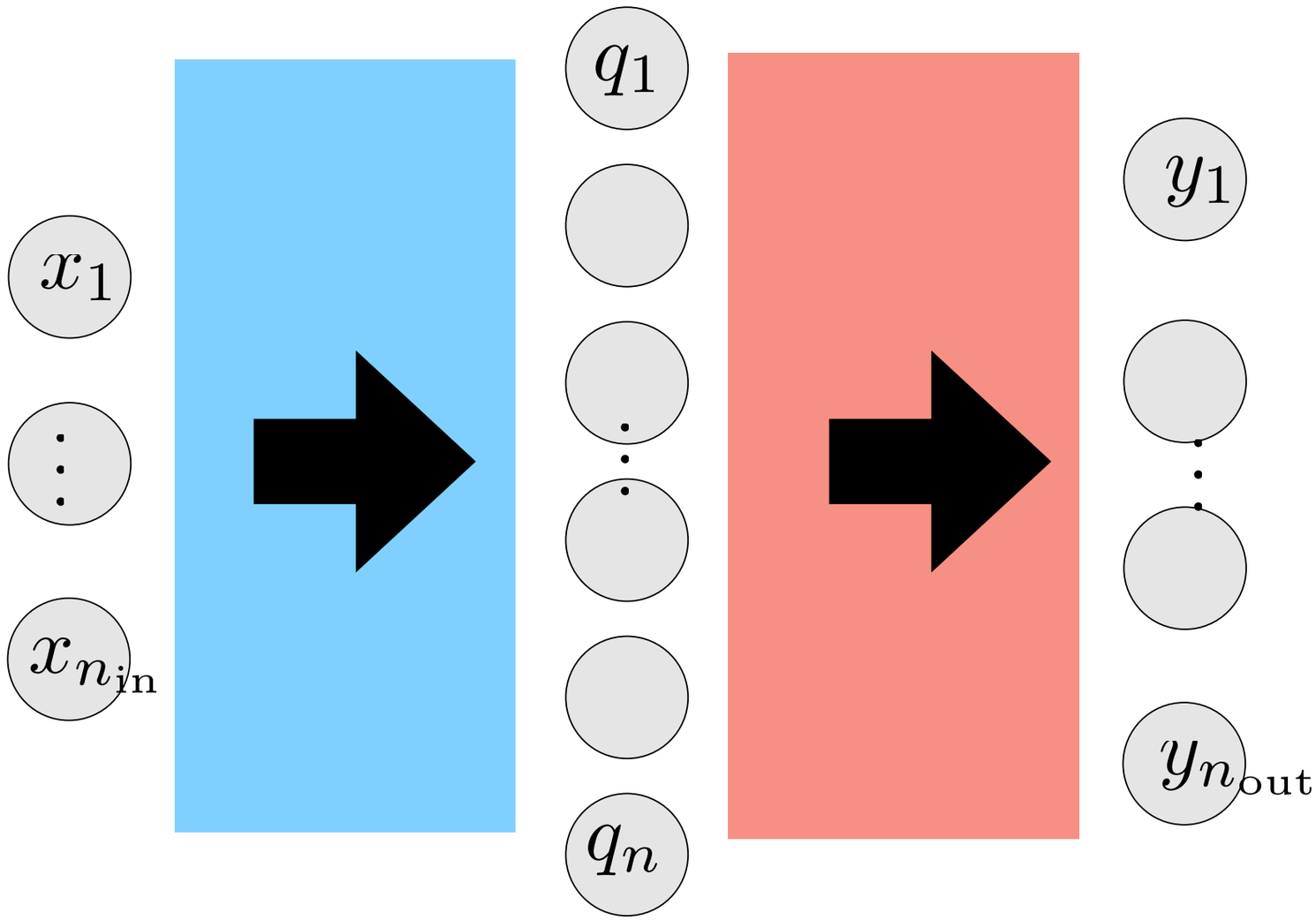


$$\frac{\partial y_i}{\partial x} = \sum_{k=1}^{n} \frac{\partial y_i}{\partial q_k} \frac{\partial q_k}{\partial x}$$

matrix-vector product

$$\frac{\partial y_i}{\partial x_j} = \sum_{k=1}^{n} \frac{\partial y_i}{\partial \phi(q_k)} \frac{\partial \phi(q_k)}{\partial q_k} \frac{\partial q_k}{\partial x_j}$$
$$= \sum_{k=1}^{n} \frac{\partial y_i}{\partial \phi(q_k)} \phi'(q_k) \frac{\partial q_k}{\partial x_j}$$

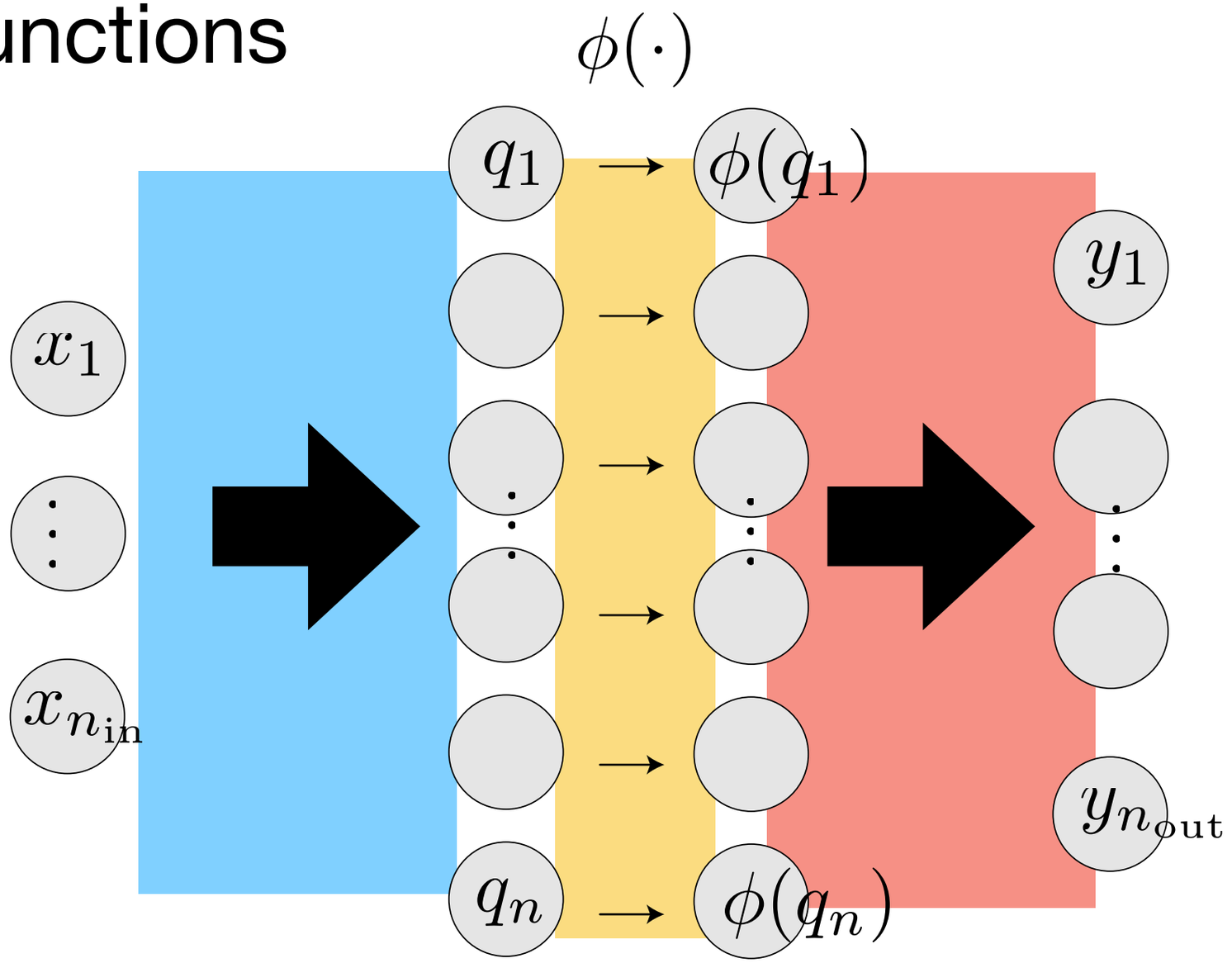
Many input, many output



$$\frac{\partial y_i}{\partial x_j} = \sum_{k=1}^{n} \frac{\partial y_i}{\partial q_k} \frac{\partial q_k}{\partial x_j}$$

matrix-matrix product

Element-wise functions



$$\frac{\partial y_i}{\partial x_j} = \sum_{k=1}^{n} \frac{\partial y_i}{\partial \phi(q_k)} \frac{\partial \phi(q_k)}{\partial q_k} \frac{\partial q_k}{\partial x_j}$$

$$= \sum_{k=1}^{n} \frac{\partial y_i}{\partial \phi(q_k)} \phi'(q_k) \frac{\partial q_k}{\partial x_j}$$

matrix-matrix product
ft. twist (rescale columns)

Exercise: take this derivative

$$\text{Input } x^{(0)}$$ (assume all vectors of dimensionality n, matrices n x n)

$$h_i^{(1)} = \sum_j W_{ij}^{(1)} x_j^{(0)}$$

$$a_i^{(1)} = \phi(h_i^{(1)})$$

$$h_i^{(2)} = \sum_j W_{ij}^{(2)} a_j^{(1)}$$

$$a_i^{(2)} = \phi(h_i^{(2)})$$

$$L = \sum_i (a_i^{(2)} - a_i^{*(2)})^2$$

$$\frac{\partial L}{\partial W_{ij}^{(1)}} = ??$$

Vanilla RNNs math

We can process a sequence of vectors **x** by applying a **recurrence formula** at every time step:

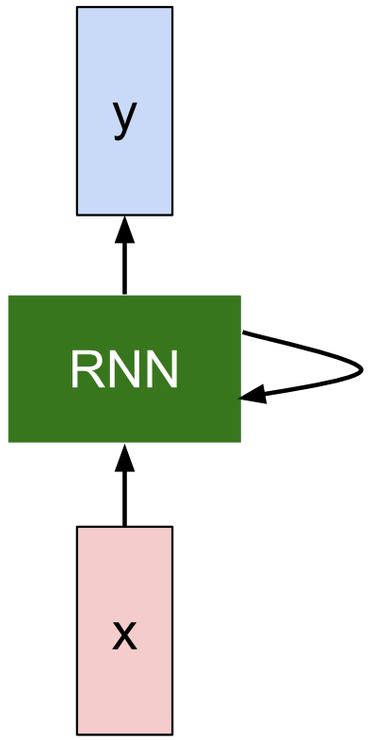
$$h_t = f_W(h_{t-1}, x_t)$$

new state

old state

input vector at some time step

some function with parameters W



$$h_t = \tanh(W_{hh} h_{t-1} + W_{xh} x_t)$$

$$y_t = W_{hy} h_t$$

RNN architectures for time series

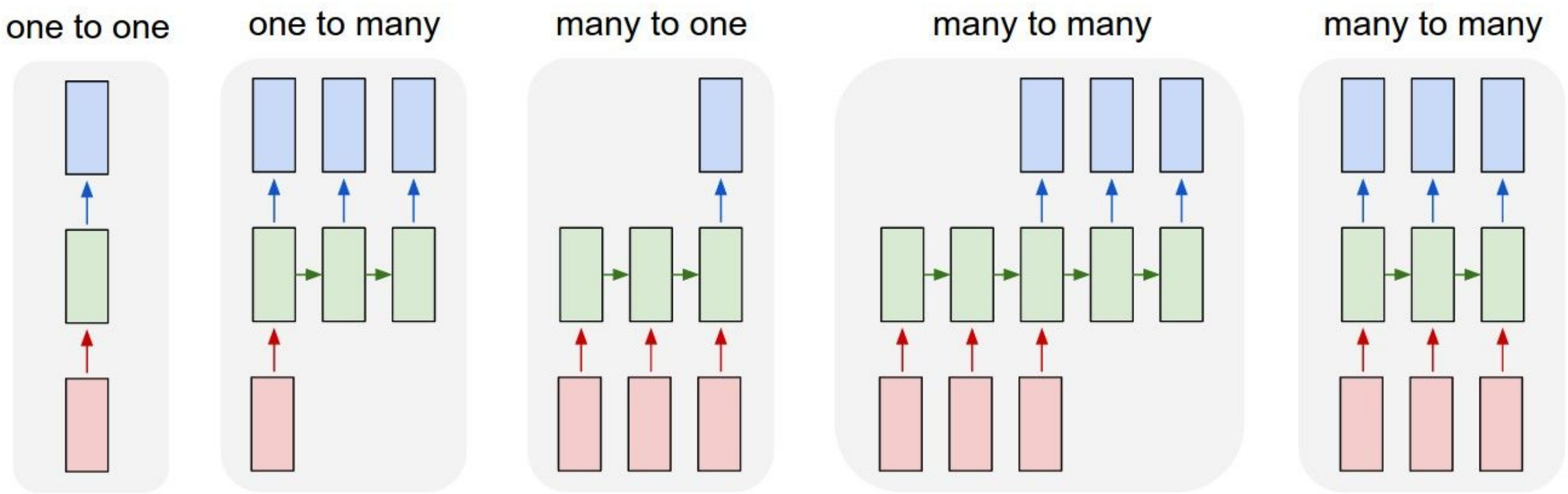


image-> caption

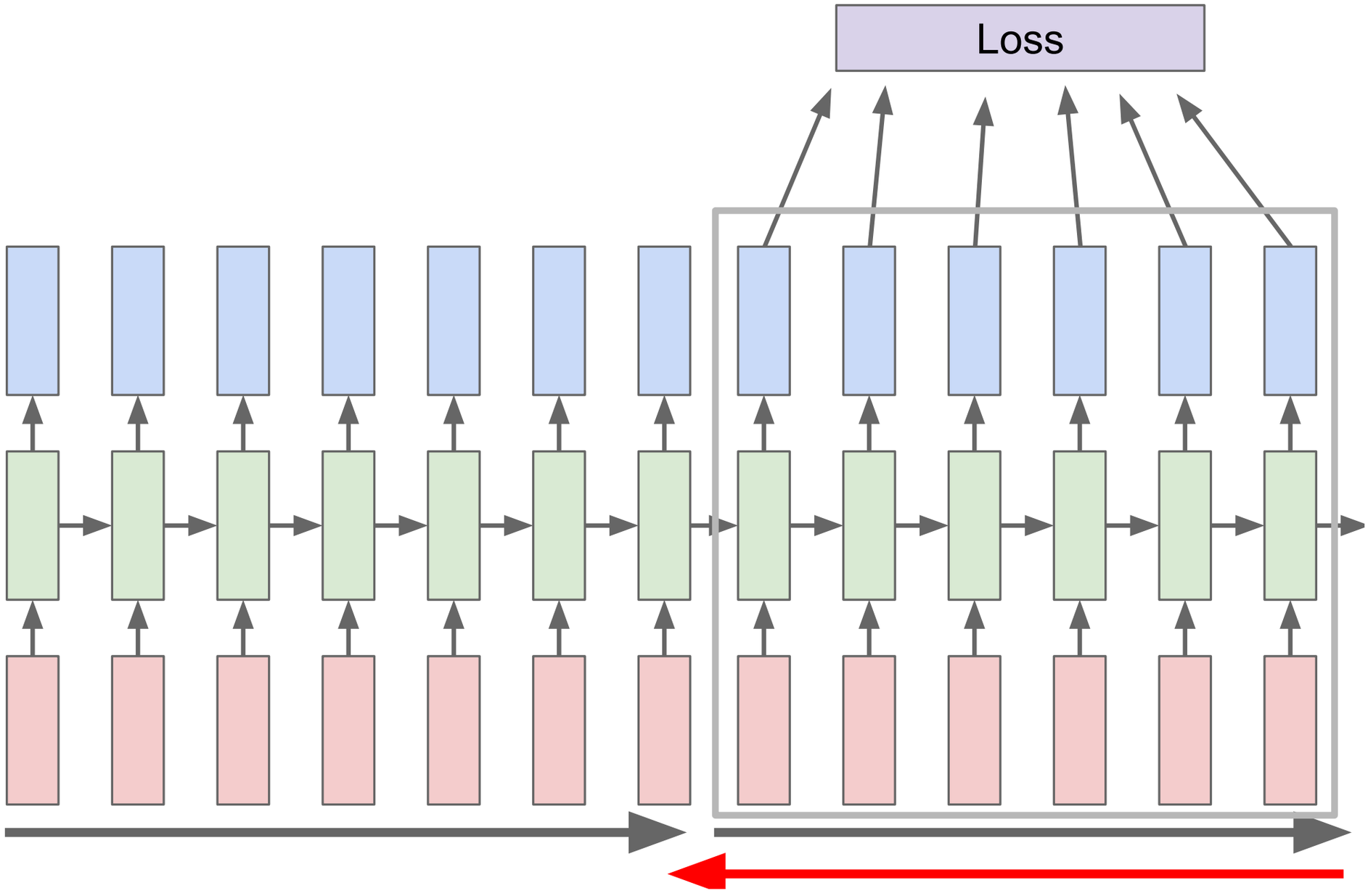
video-> label

sentence-> sentence

video-> video

text-> text

Truncated backprop through time



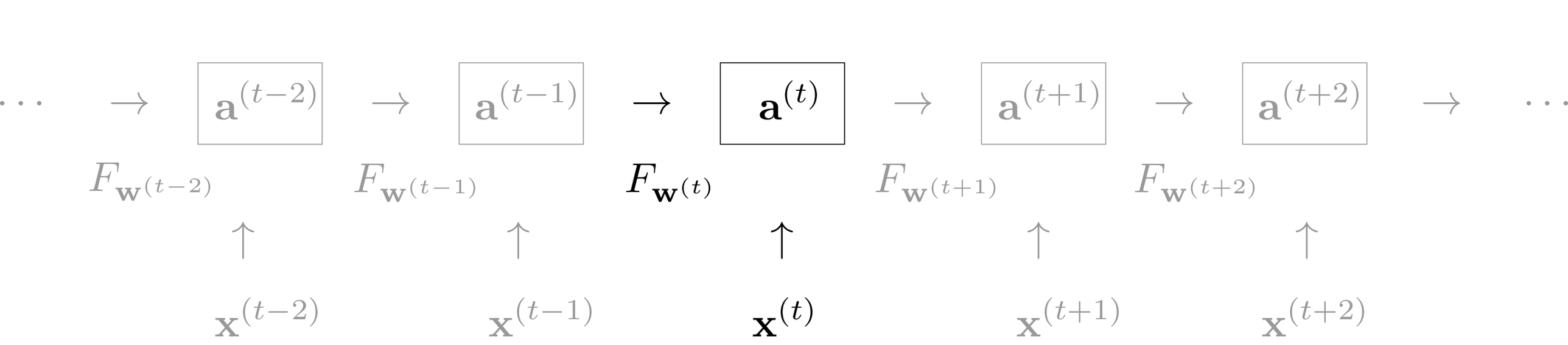
Backprop in a a “vanilla” RNN

$$\begin{aligned}\mathbf{a}^{(t)} &= F_{\mathbf{w}}(\mathbf{a}^{(t-1)}, \mathbf{x}^{(t)}) \\ &= \phi\left(\mathbf{W}\mathbf{a}^{(t-1)} + \mathbf{W}^{\text{in}}\mathbf{x}^{(t)} + \mathbf{b}^{\text{rec}}\right)\end{aligned}$$

$$h_i^{(t)} = \sum_{j \neq 1}^{n} W_{ij}\hat{a}_j^{(t-1)} + \sum_{l=1}^{n_{in}} W_{il}^{\text{in}} x_l^{(t)} + b_i^{\text{rec}}$$

$$a_i^{(t)} = \phi(h_i^{(t)}) \qquad \hat{\mathbf{a}}^{(t-1)} = \text{concat}(\mathbf{a}^{(t-1)}, \mathbf{x}^{(t)}, 1)$$

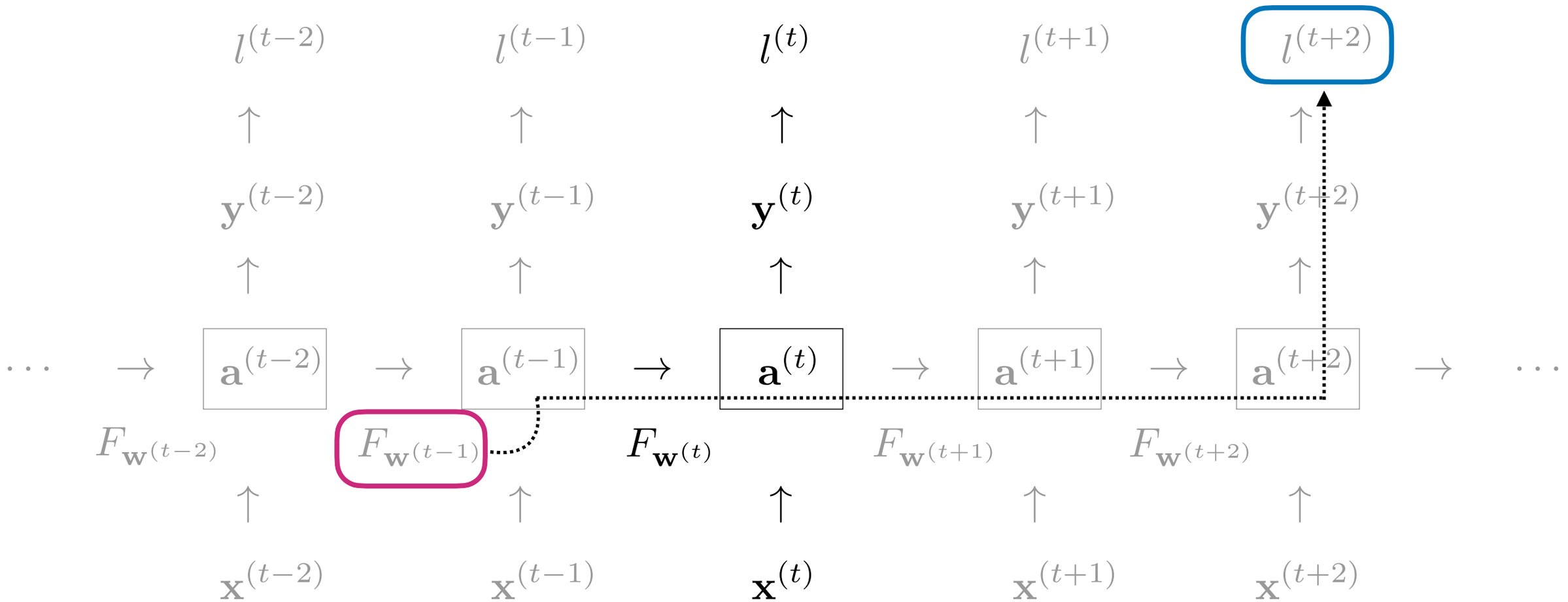
“Unrolling” a network



Gradient descent

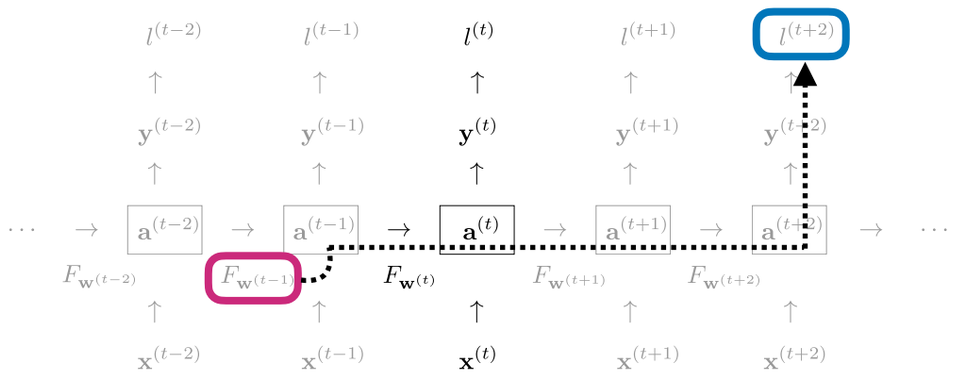
Global loss function $\mathcal{L} = \sum_t l^{(t)}$

Gradient of loss w.r.t. w



$$\frac{\partial \mathcal{L}}{\partial \mathbf{w}} = \sum_t \frac{\partial l^{(t)}}{\partial \mathbf{w}}$$

Key ingredients for calculating $\frac{\partial l^{(t)}}{\partial \mathbf{w}^{(s)}}$



$$a_i^{(t)} = \phi(h_i^{(t)}) \qquad h_i^{(t)} = \sum_j W_{ij} \hat{a}_j^{(t-1)}$$

$$l^{(t)} = \frac{1}{2} \sum_k (\sum_i W_{ki}^{\text{out}} a_i^{(t)} - y_k^{*(t)})^2$$

$$\overline{M}_{kij}^{(t)} = \frac{\partial a_k^{(t)}}{\partial W_{ij}}$$

$$J_{ij}^{(t)} = \frac{\partial a_i^{(t)}}{\partial a_j^{(t-1)}}$$

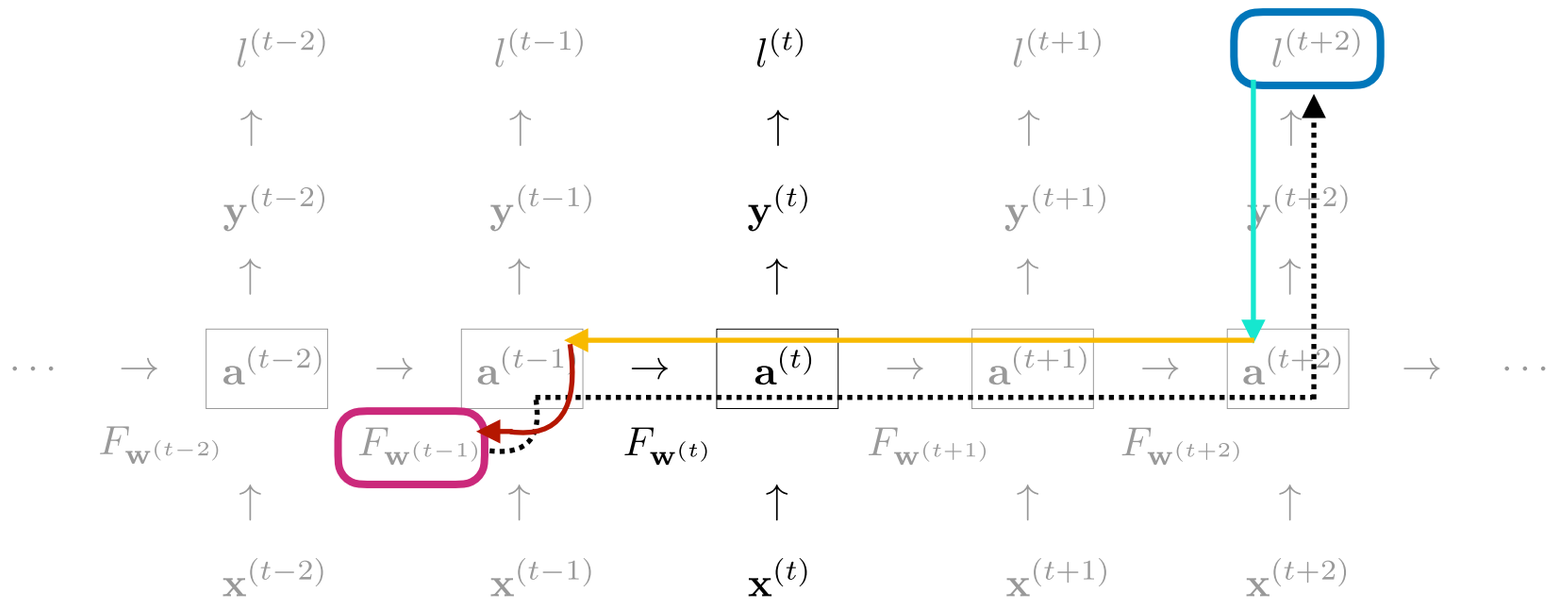
$$q_i^{(t)} = \frac{\partial l^{(t)}}{\partial a_i^{(t)}} = \sum_{l=1}^{n_{out}} (y_l^{(t)} - y_l^{*(t)}) W_{li}^{\text{out}}$$

Output derivative

Backpropagation through time

1. Unroll the graph for some number of t and s

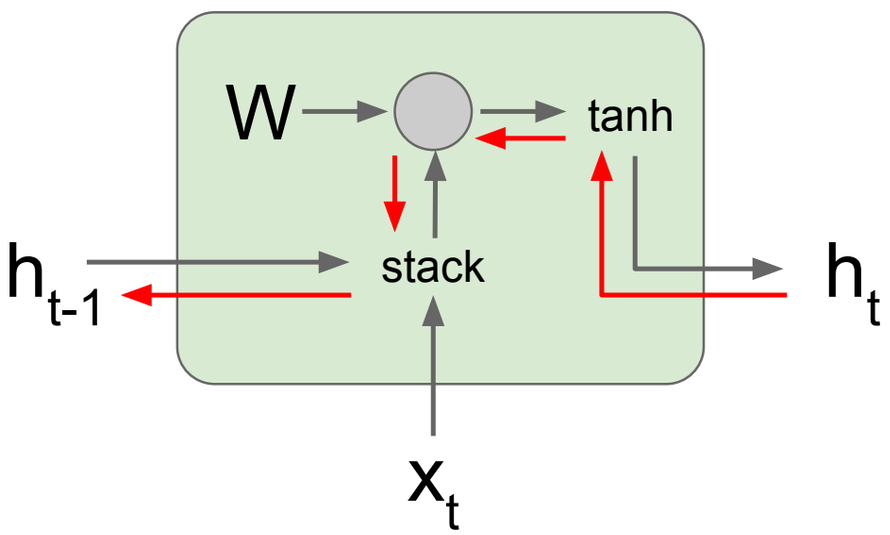
2. Calculate each instance of $\frac{\partial l^{(t)}}{\partial \mathbf{w}^{(s)}}$ going **backwards** from the losses



$$\frac{\partial l^{(t+2)}}{\partial W_{ij}^{(s-1)}} = \mathbf{q}^{(t+2)} \mathbf{J}^{(t+2)} \mathbf{J}^{(t+1)} \mathbf{J}^{(t)} \overline{\mathbf{M}}_{ij}$$

Life is harder in the RNN world

Backpropagation from h_t to h_{t-1} multiplies by W (actually W_{hh}^T)



$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$
$$= \tanh\left((W_{hh} \quad W_{hx})\begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)$$
$$= \tanh\left(W\begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)$$

K steps back: W^k

Exponential decay or explosion of gradients

If gradients are exploding: clip to a max value

Fancy regularization: unitary matrices

Smarter architectures!

Vanilla RNN

$$h_t = \tanh\left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)$$

LSTM

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

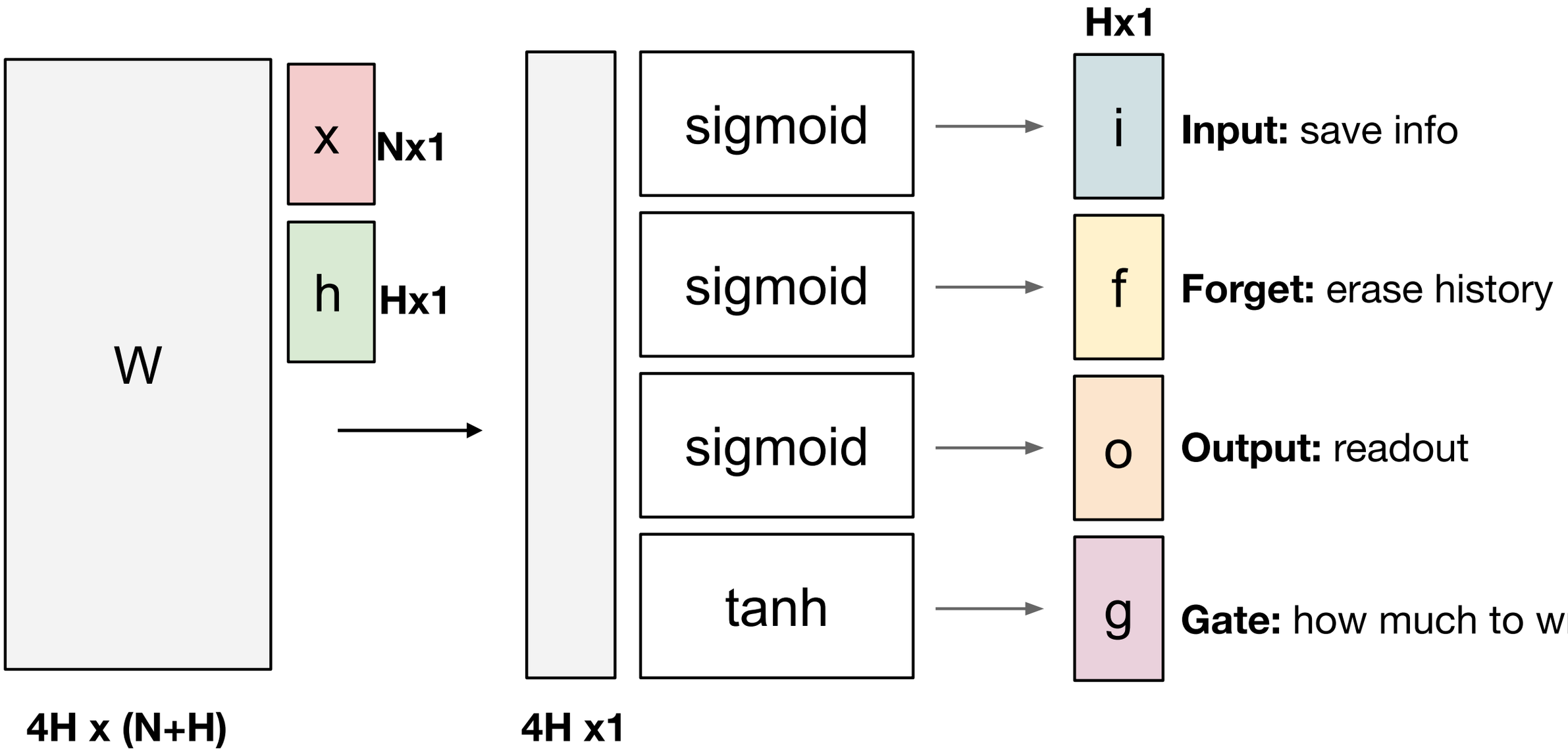
$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

Hochschreiter & Schmidhuber, 1997

$\odot$ Element-wise multiplication

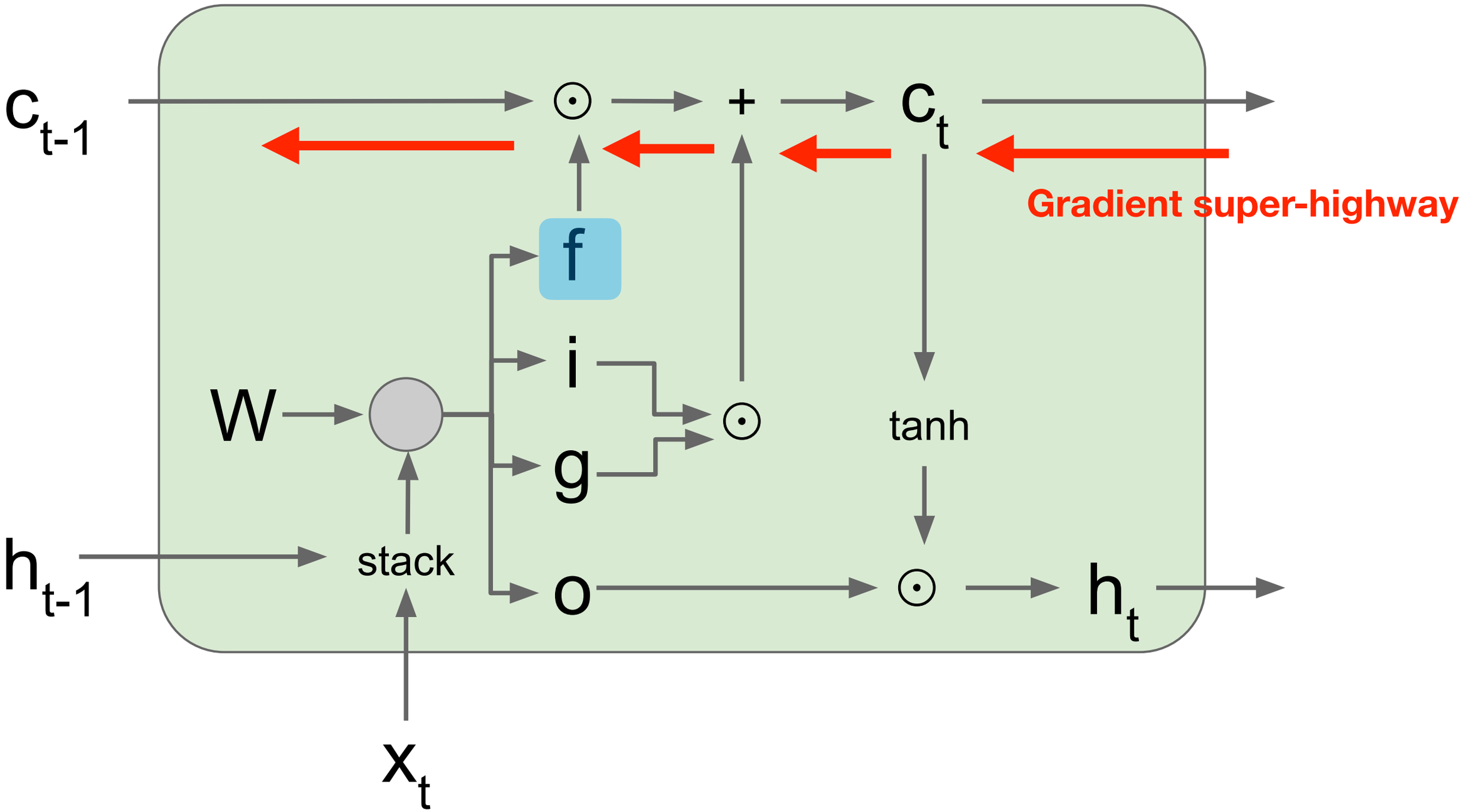
LSTM details



$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

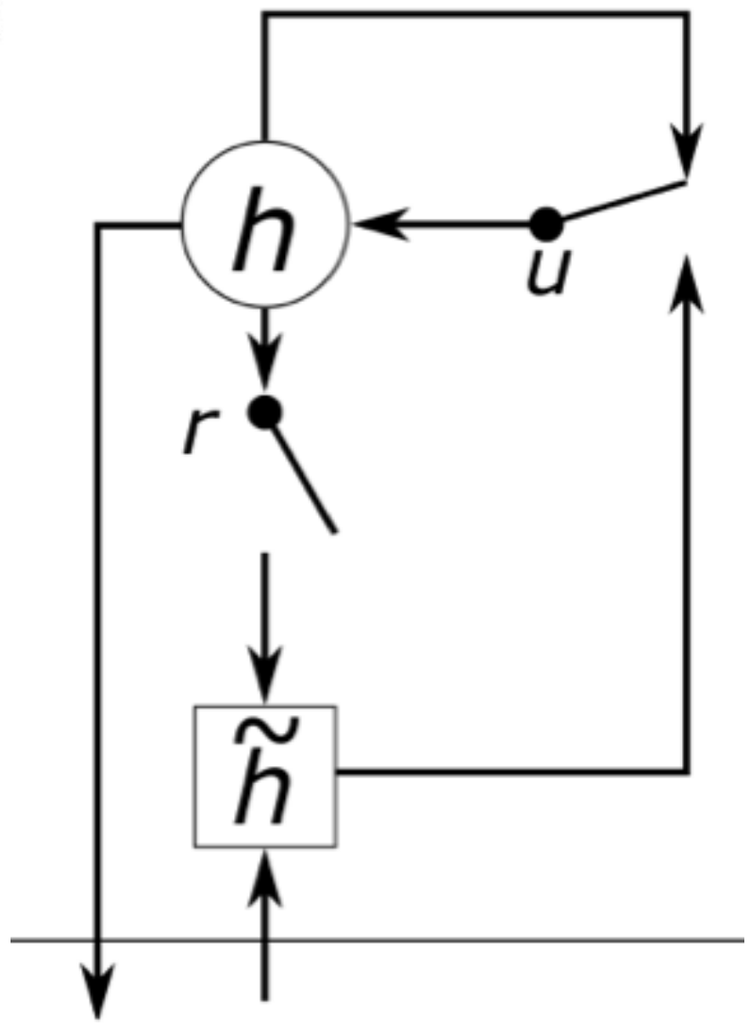
Gradient for c multiplied (element wise) with f, which varies over time and bounded so much more numerically stable



Other architectures

GRU (Cho et al, 2013)

$$r_t = \sigma(W_{xr}x_t + W_{hr}h_{t-1} + b_r)$$
$$z_t = \sigma(W_{xz}x_t + W_{hz}h_{t-1} + b_z)$$
$$\tilde{h}_t = \tanh(W_{xh}x_t + W_{hh}(r_t \odot h_{t-1}) + b_h)$$
$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot \tilde{h}_t$$



Other architectures

MUT1:

$$
\begin{aligned}
z &= \mathrm{sigm}(W_{\mathrm{xz}} x_t + b_{\mathrm{z}}) \\
r &= \mathrm{sigm}(W_{\mathrm{xr}} x_t + W_{\mathrm{hr}} h_t + b_{\mathrm{r}}) \\
h_{t+1} &= \tanh(W_{\mathrm{hh}}(r \odot h_t) + \tanh(x_t) + b_{\mathrm{h}}) \odot z \\
&+ h_t \odot (1 - z)
\end{aligned}
$$

MUT2:

$$
\begin{aligned}
z &= \mathrm{sigm}(W_{\mathrm{xz}} x_t + W_{\mathrm{hz}} h_t + b_{\mathrm{z}}) \\
r &= \mathrm{sigm}(x_t + W_{\mathrm{hr}} h_t + b_{\mathrm{r}}) \\
h_{t+1} &= \tanh(W_{\mathrm{hh}}(r \odot h_t) + W_{xh} x_t + b_{\mathrm{h}}) \odot z \\
&+ h_t \odot (1 - z)
\end{aligned}
$$

MUT3:

$$
\begin{aligned}
z &= \mathrm{sigm}(W_{\mathrm{xz}} x_t + W_{\mathrm{hz}} \tanh(h_t) + b_{\mathrm{z}}) \\
r &= \mathrm{sigm}(W_{\mathrm{xr}} x_t + W_{\mathrm{hr}} h_t + b_{\mathrm{r}}) \\
h_{t+1} &= \tanh(W_{\mathrm{hh}}(r \odot h_t) + W_{xh} x_t + b_{\mathrm{h}}) \odot z \\
&+ h_t \odot (1 - z)
\end{aligned}
$$

(Jozefowicz et al., 2015)

Online learning for RNNs

Algorithm	Facing	Tensor	Update		Memory	Time
RTRL	Past	M_{kij}	Deterministic	Closed-form	n^3	n^4
UORO	Past	$A_k B_{ij}$	Stochastic	Closed-form	n^2	n^2
KF-RTRL	Past	$A_j B_{ki}$	Stochastic	Closed-form	n^2	n^3
R-KF-RTRL	Past	$A_i B_{jk}$	Stochastic	Closed-form	n^2	n^3
r-OK	Past	$\sum_{l=1}^{r} A_{lj} B_{lki}$	Stochastic	Numerical	rn^2	rn^3
KeRNL	Past	$A_{ki} B_{ij}$	Deterministic	Numerical	n^2	n^2
RFLO	Past	$\delta_{ki} B_{ij}$	Deterministic	Closed-form	n^2	n^2
E-BPTT	–	–	Deterministic	Closed-form	nT	n^2
F-BPTT	Future	–	Deterministic	Closed-form	nT	$n^2 T$
DNI	Future	A_{li}	Deterministic	Numerical	n^2	n^2

Marschall et al. *JMLR 2020*

Bringing probabilities back in

Deep Kalman Filter *(Krishnan et al, 2016)*

Transition distribution

$$z_t|z_{t-1} \sim \mathcal{N}(F_\theta^\mu(z_{t-1}), F_\theta^\Sigma(z_{t-1}))$$

Emission distribution

$$x_t|z_t \sim \mathcal{N}(G_\theta^\mu(z_t), G_\theta^\Sigma(z_t))$$

Approximate posterior distribution

$$z_t|x_{1:T} \sim \mathcal{N}(R_\theta^\mu(x_{1:T}), R_\theta^\Sigma(x_{1:T}))$$

Feedforward neural networks

Bidirectional recurrent networks

$$\mathcal{F}(q,\theta) = \int q_\theta(z) \log p_\theta(x|z) \mathrm{d}z + \int q_\theta(z) \log \frac{p_\theta(z)}{q_\theta(z)} \mathrm{d}z$$

Directly maximize the free energy using GD and backpropagation

Summary

RNNs provide a **flexible** way for modeling temporal dependencies at scale.

Neural networks (deep learning) enable us to handle
- High-dimensional observations (often 100s-1000s)
- High-dimensional latent variables (often 100s-1000s)
- Large-scale data (often millions or more)

- Neural networks (deep learning) enable us to capture
  - Nonlinear dynamics
  - Long-term dependencies

- Neural networks (deep learning) enable us to work with
  - Intractable probabilistic models

- Neural networks enable us to transfer knowledge across problems
  - Parameter sharing across multiple datasets
  - One neural network with a task indicator input

Summary

Data hungry, and not necessarily easy to train

LSTMs and GRUs used in practice

Good at capturing long-term dependencies but not high frequency structure

Ongoing research: better architectures, theoretical understanding.

Greff et al. *LSTM: A Search Space Odyssey,* 2015.

Goodfellow, Courville, Bengio. *Deep Learning,* 2016.

Goldberg. *Neural Network Methods for Natural Language Processing,* 2017.

K Cho lecture notes: https://github.com/nyu-dl/NLP_DL_Lecture_Note