DS-GA 3001.008 Modelling time series data

L8a. Advanced topics in GPs.

Instructor: Cristina Savin

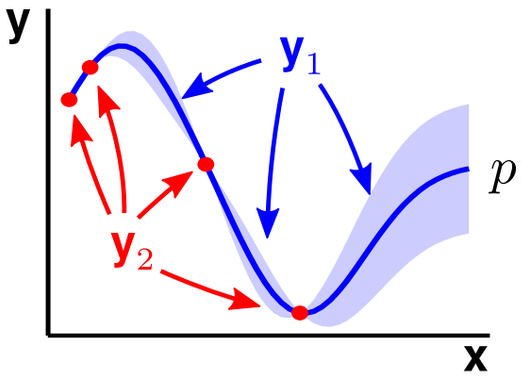
NYU, CNS & CDS

Def: A **Gaussian Process** is a collection of random variables, any finite number of which have (consistent) Gaussian distributions.

It is essentially a infinite dimensional generalization of a multivariate gaussian;

We call this a **'non-parametric'** model because the number of parameters (the covariance matrix) increases with the number of data points, N.

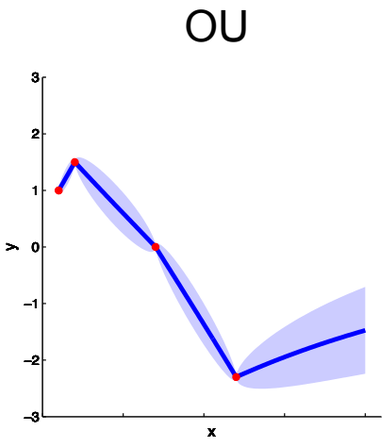
Inference

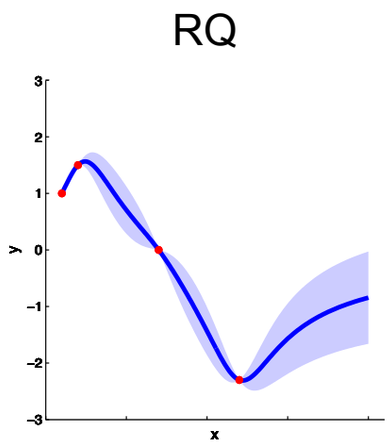


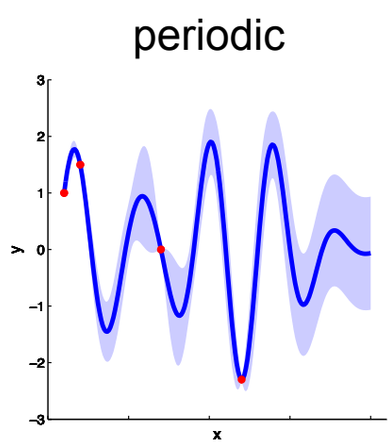
$$p(\mathbf{y}_1|\mathbf{y}_2) = \mathcal{N}(\mathbf{y}_1 ; \mathbf{a} + \mathrm{B}\mathrm{C}^{-1}(\mathbf{y}_2 - \mathbf{b}), \mathrm{A} - \mathrm{B}\mathrm{C}^{-1}\mathrm{B}^{\mathsf{T}})$$

$$\mathcal{O}(N^3)$$

Kernels:







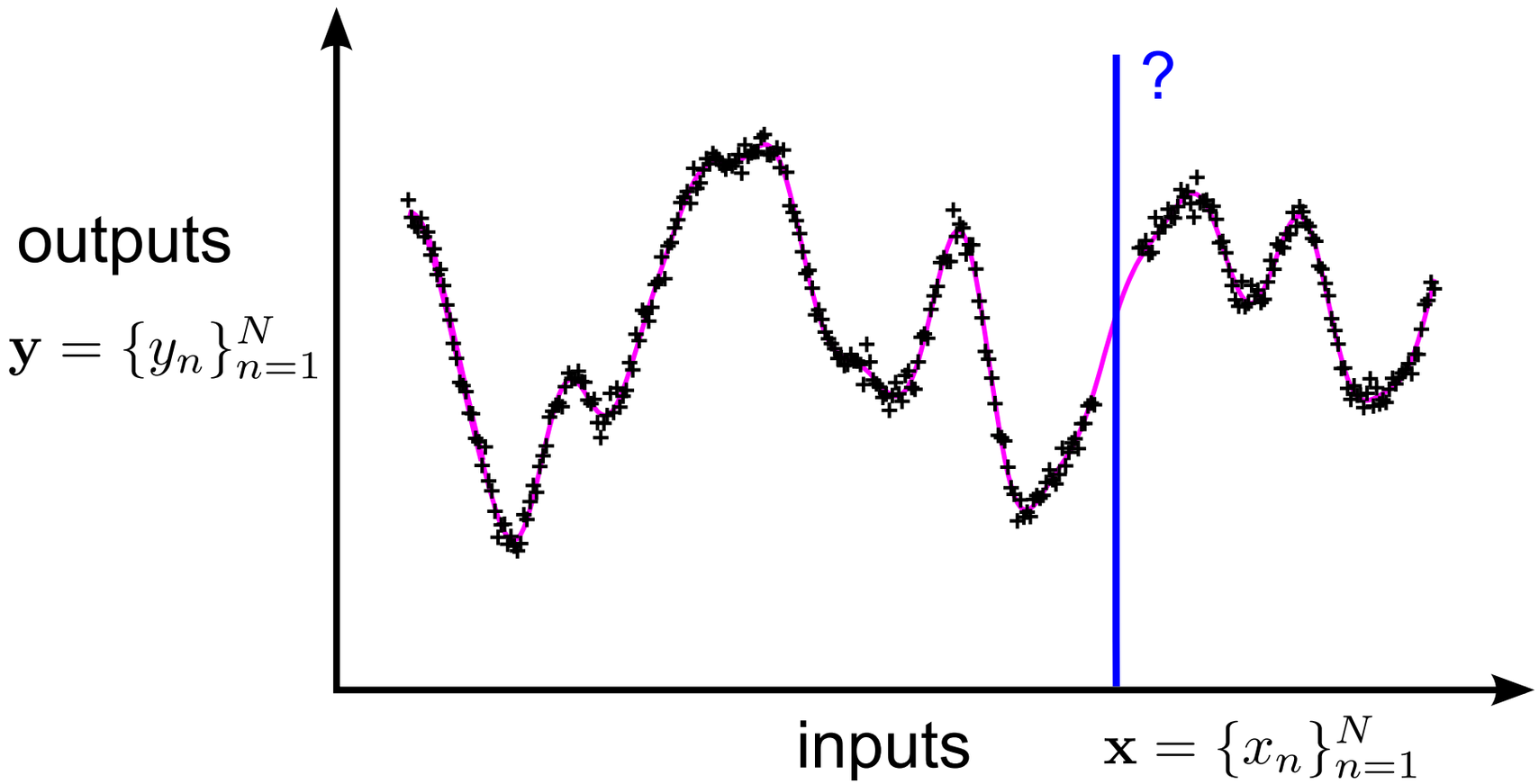
GP credits: Rich Turner

Efficient inference for large N

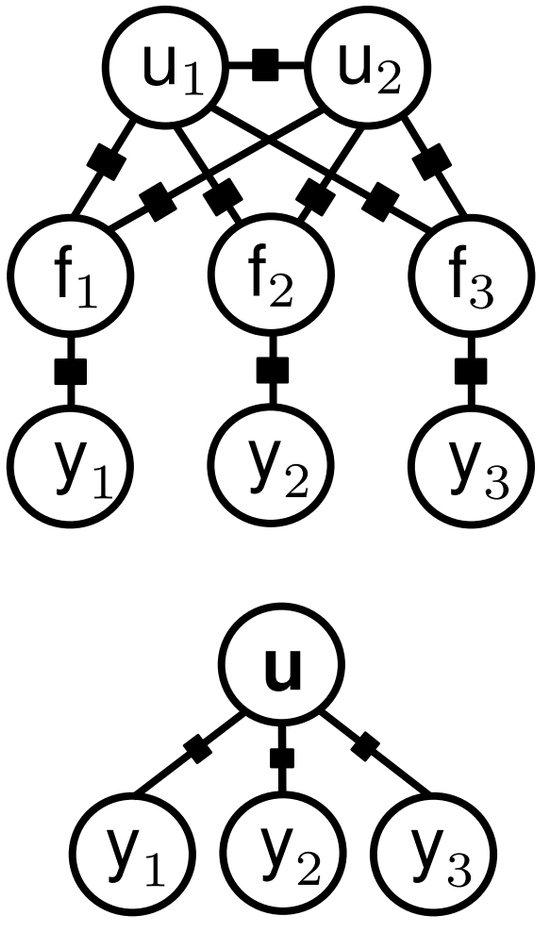
$$p(f|\theta) = \mathcal{GP}(f; 0, \mathrm{K}_\theta)$$

$$p(y_n|f, x_n, \theta)$$

IDEA: maybe we don't **really** need **all** the data



FITC: fully independent training conditional approximation



Introduce M<N inducing points (pseudo-data) **u**

These **u** variables summarize (as much as possible) the dependencies in **f**, while keeping costs low

$$p(\mathsf{f}_t|\mathbf{u}) = \mathcal{N}(\mathsf{f}_t; \mathsf{K}_{\mathsf{f}_t\mathsf{u}}\mathsf{K}_{\mathsf{uu}}^{-1}\mathbf{u}, \mathsf{K}_{\mathsf{f}_t\mathsf{f}_t} - \mathsf{K}_{\mathsf{f}_t\mathsf{u}}\mathsf{K}_{\mathsf{uu}}^{-1}\mathsf{K}_{\mathsf{uf}_t})$$

$$p(\mathbf{y}_t|\theta) = \mathcal{N}(\mathbf{y}; \mathbf{0}, \mathsf{K}_{\mathsf{fu}}\mathsf{K}_{\mathsf{uu}}^{-1}\mathsf{K}_{\mathsf{uu}}\mathsf{K}_{\mathsf{uu}}^{-1}\mathsf{K}_{\mathsf{uf}} + \mathsf{D} + \sigma_{\mathsf{y}}^2\mathsf{I})$$

A Brief History of Gaussian Process Approximations

approximate generative model
exact inference

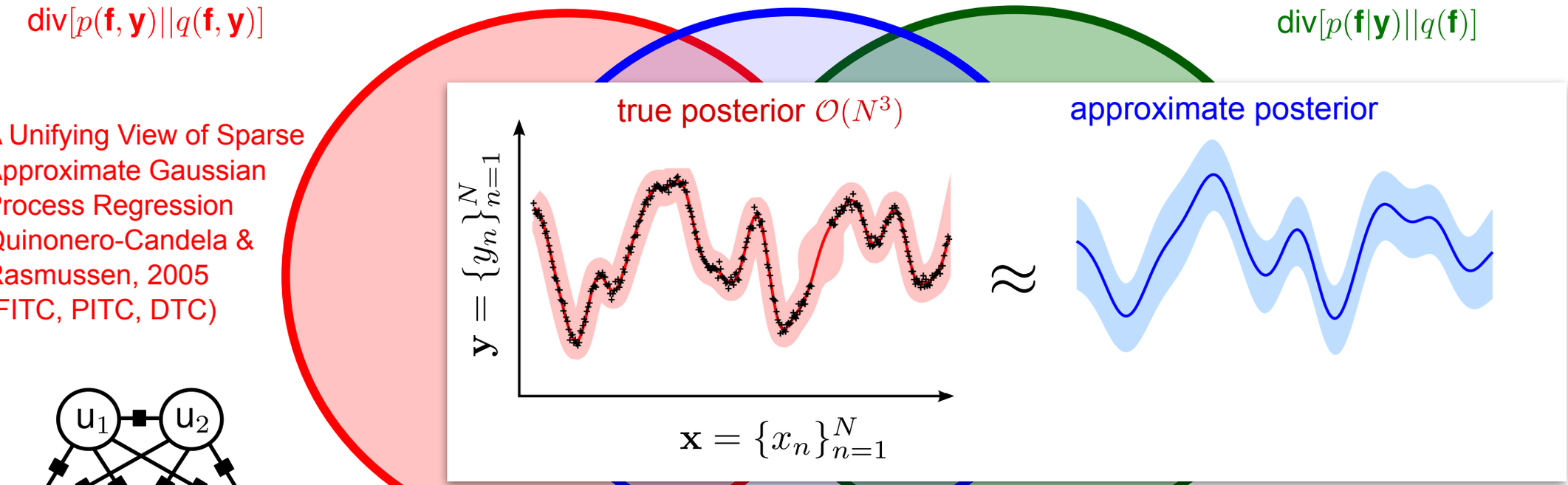
methods employing
pseudo-data

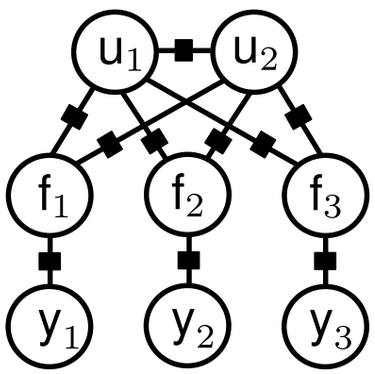
exact generative model
approximate inference

$\text{div}[p(\mathbf{f},\mathbf{y})||q(\mathbf{f},\mathbf{y})]$

$\text{div}[p(\mathbf{f}|\mathbf{y})||q(\mathbf{f})]$

A Unifying View of Sparse Approximate Gaussian Process Regression
Quinonero-Candela & Rasmussen, 2005
(FITC, PITC, DTC)





A Unifying Framework for Sparse Gaussian Process Approximation using Power Expectation Propagation
Bui, Yan and Turner, 2016
(VFE, EP, FITC, PITC ...)

FITC: Snelson et al. “Sparse Gaussian Processes using Pseudo-inputs”
PITC: Snelson et al. “Local and global sparse Gaussian process approximations”
EP: Csato and Opper 2002 / Qi et al. "Sparse-posterior Gaussian Processes for general likelihoods.”
VFE: Titsias “Variational Learning of Inducing Variables in Sparse Gaussian Processes”
DTC / PP: Seeger et al. “Fast Forward Selection to Speed Up Sparse Gaussian Process Regression”

Other tricks: building extra structure in the covariance matrix so you don't have to do the inversion the usual way. Kronecker tricks

$$\mathbf{K} = \bigotimes_i \mathbf{K}_i$$

Kronecker product

$$\mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} a_{11}\mathbf{B} & \cdots & a_{11}\mathbf{B} \\ \vdots & \ddots & \vdots \\ a_{m1}\mathbf{B} & \cdots & a_{mn}\mathbf{B} \end{bmatrix}$$

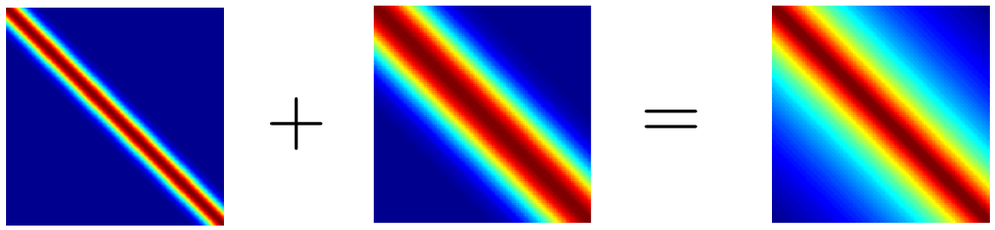
Matrix operations decompose nicely

$$\begin{aligned}
\text{Transpose}: \quad & \mathbf{K}^{\mathrm{T}} = \textstyle\bigotimes_i \mathbf{K}_i^{\mathrm{T}} \\
\text{Inverse}: \quad & \mathbf{K}^{-1} = \textstyle\bigotimes_i \mathbf{K}_i^{-1} \\
\text{Eigen} - \text{decomposition}: \quad & \mathbf{K} = \textstyle\bigotimes_i \mathbf{Q}_i \cdot \bigotimes_i \mathbf{\Lambda}_i \cdot \bigotimes_i \mathbf{Q}_i^{\mathrm{T}} \\
\text{Determinants}: \quad & \det(\mathbf{K}) = \textstyle\prod_i \det(\mathbf{K}_i)^{D_i} \\
\text{Trace}: \quad & \operatorname{tr}(\mathbf{K}) = \textstyle\prod_i \operatorname{tr}(\mathbf{K}_i) \\
\text{Vec}: \quad & \operatorname{vec}(\mathbf{AXB}) = \mathbf{A} \otimes \mathbf{B} \operatorname{vec}(\mathbf{X})
\end{aligned}$$

How to choose the kernel?

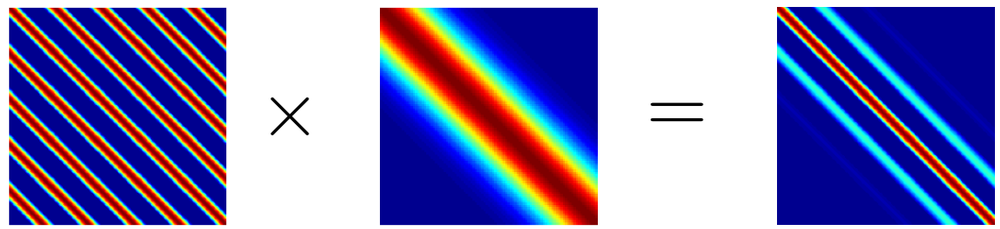
A1: mix and match

(positive) linear combinations of covariance functions



e.g. scale mixture of SE = rational quadratic

multiplication of covariance functions



e.g. periodic SE $\cos(\omega\Delta x)\exp(-\frac{1}{2l^2}\Delta x^2)$

derivative of GP = GP

$$\frac{d}{dx}y(x) = \frac{d}{dx}\sum_{k=1}^{\infty}\gamma_k g_k(x) = \sum_{k=1}^{\infty}\gamma_k \frac{d}{dx}g_k(x)$$

new basis: $g'_k(x) = \frac{d}{dx}g_k(x)$

new covariance: $K'(x,x') = \frac{d}{dx}\frac{d}{dx'}K(x,x')$

integral of GP = GP

$$\int dx\, y(x) = \sum_{k=1}^{\infty}\gamma_k \int dx\, g_k(x)$$

new basis: $g'_k(x) = \int dx\, g_k(x)$

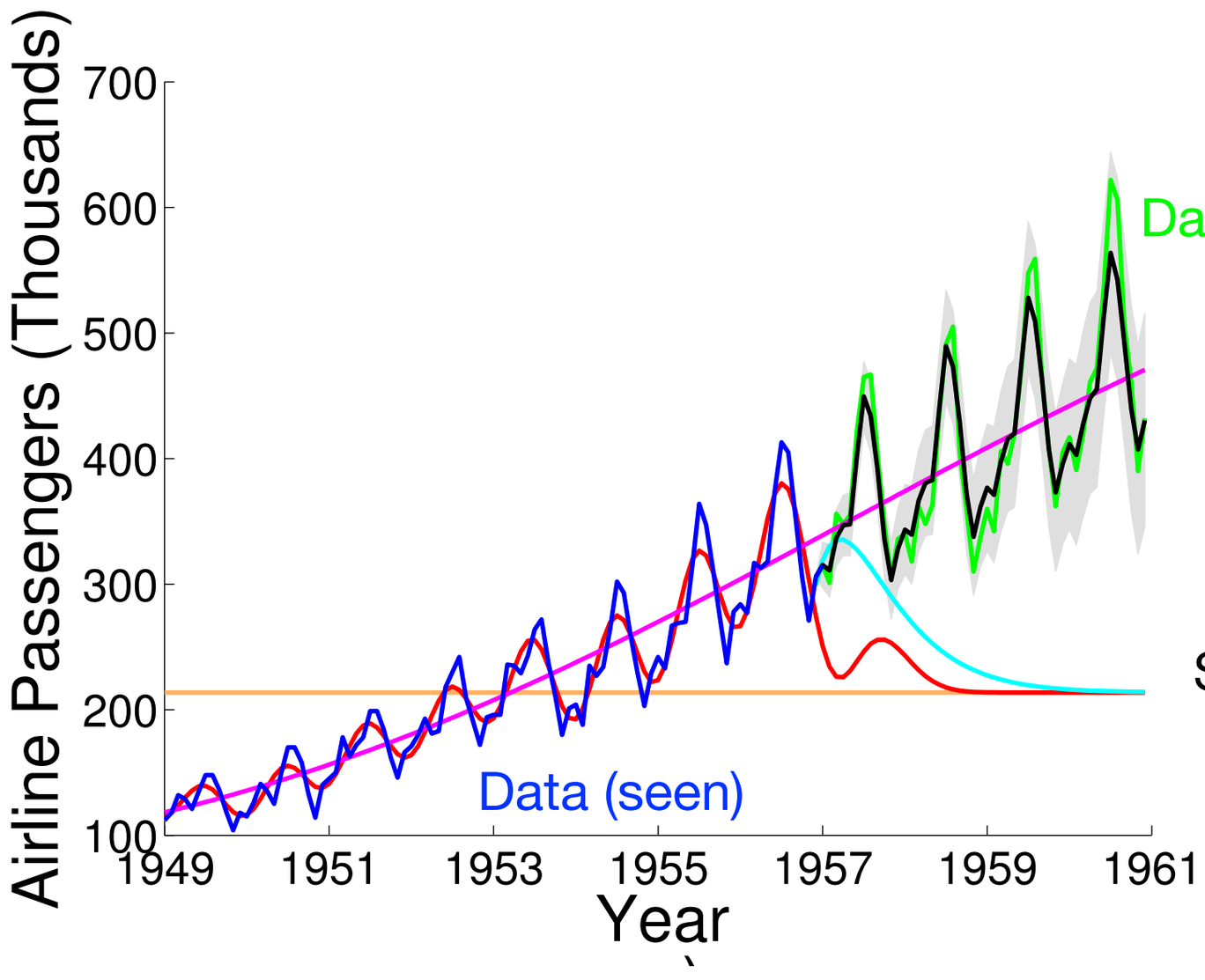
new covariance: $K'(x,x') = \int\int dx dx'\, K(x,x')$

How to choose the kernel?

A2: don't bother, learn that from data too (SM)

$$k(\tau) = \sum_{q=1}^{Q} w_q \prod_{p=1}^{P} \exp\{-2\pi^2 \tau_p^2 v_q^{(p)}\} \cos(2\pi \tau_p \mu_q^{(p)}).$$

Mixture of **Q** components Gaussian-like components (in Fourier domain)



SM tends to do much better.

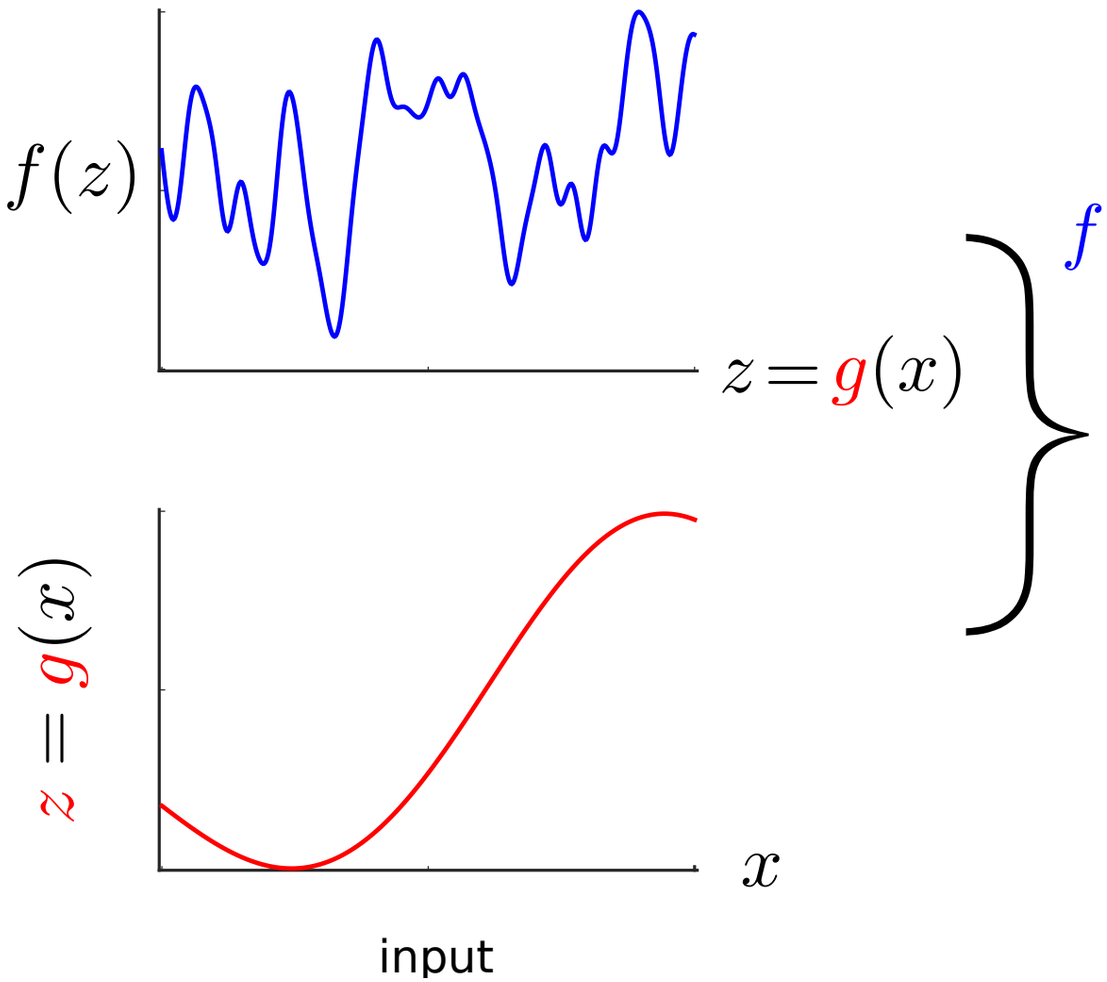
Smoothing kernels are bad at extrapolation

How to choose the kernel?

A3: go deep!

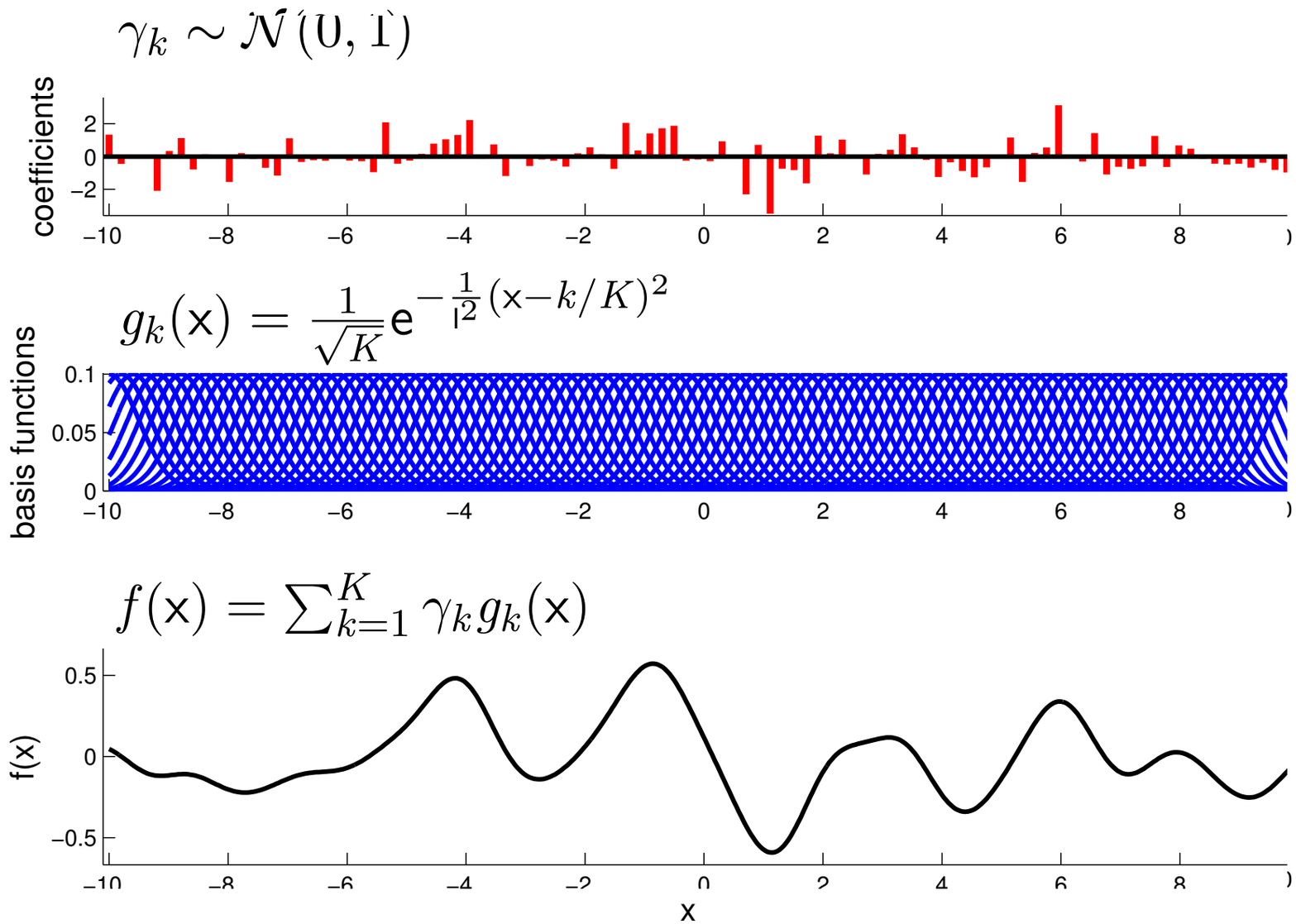
$$y(x) = f(g(x)) + \sigma_{\mathrm{y}}\epsilon$$

$$f(x) = \mathcal{GP}(0, K_f(x, x'))$$
$$g(x) = \mathcal{GP}(0, K_g(x, x'))$$



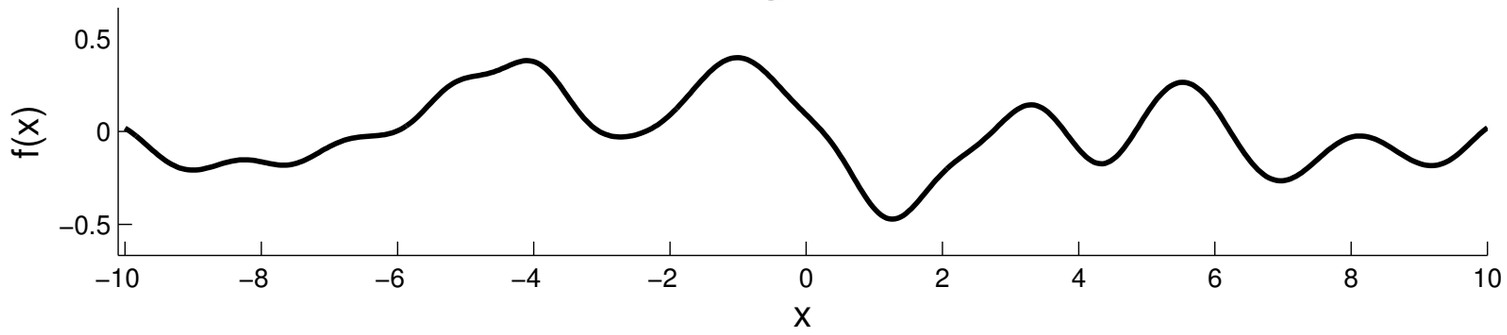
What is a kernel anyways?

We can formally link **GPs** with usual kernel methods **(SVM and co)** and **NNs**



Basis function view of Gaussian processes

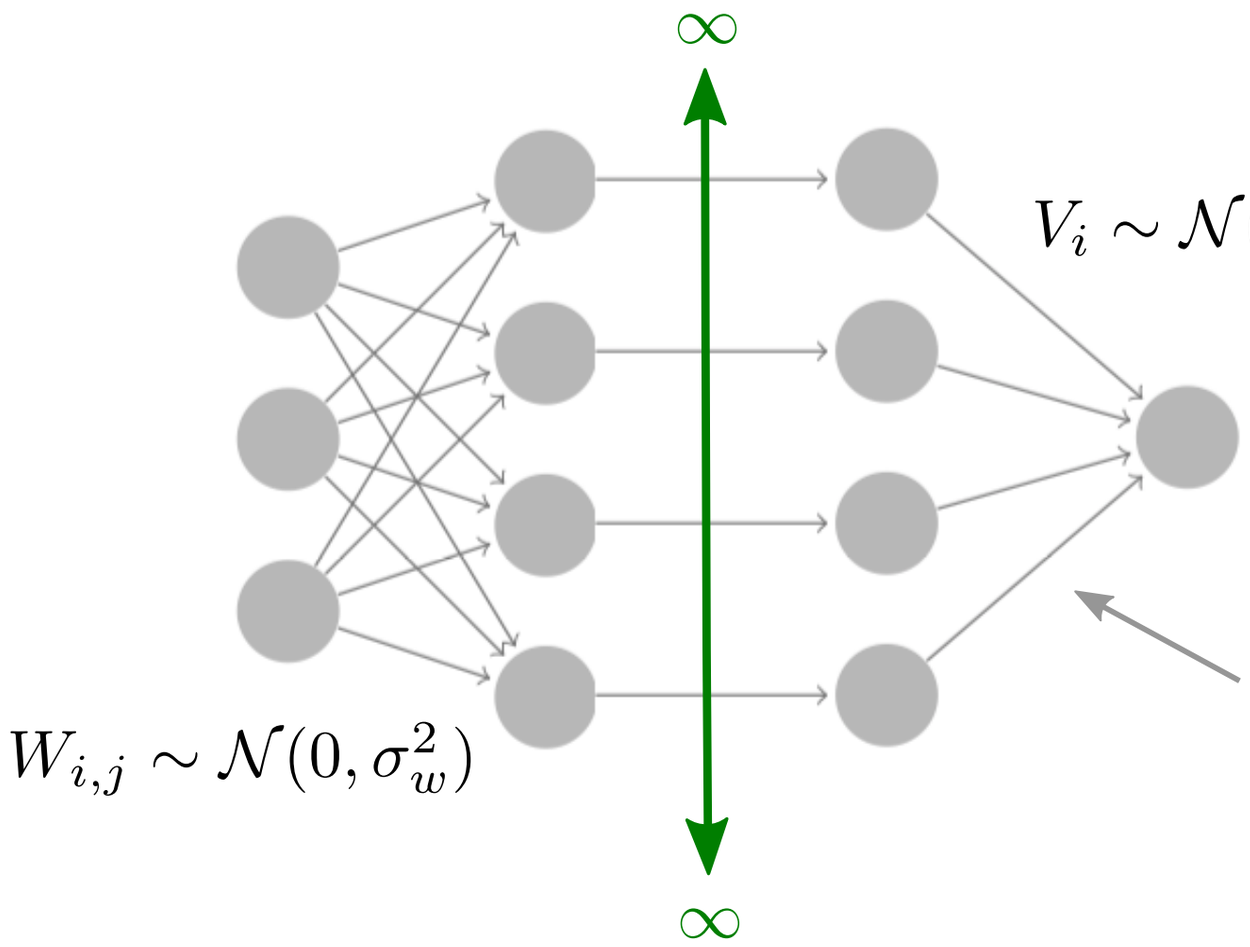
$$\left.\begin{array}{l} \gamma_k \sim \mathcal{N}(0,1) \\ \\ g_k(\mathsf{x}) = \frac{1}{\sqrt{K}} \mathsf{e}^{-\frac{1}{\mathsf{l}^2}(\mathsf{x}-k/K)^2} \\ \\ f(\mathsf{x}) = \sum_{k=1}^{K} \gamma_k g_k(\mathsf{x}) \end{array}\right\} \begin{array}{l} m(\mathsf{x}) = \langle \sum_k \gamma_k g_k(\mathsf{x}) \rangle \\ \\ \\ \mathsf{K}(\mathsf{x},\mathsf{x}') = \langle f(\mathsf{x}) f(\mathsf{x}') \rangle \end{array}$$



GP = infinite basis

Infinitely wide neural nets as GPs

inputs x | activations $a = Wx$ | activities $h = \phi(a)$ | outputs $f = Vh$



$W_{i,j} \sim \mathcal{N}(0, \sigma_w^2)$

stops the variance of the output blowing up (cf. initialisation of neural nets)

$V_i \sim \mathcal{N}(0, \sigma_v^2/D)$

$f(x) = \mathrm{NN}(x; W, V) \rightarrow$?
as $D \rightarrow \infty$

central limit kicks in: sum of independent random variables

Neal, 1996

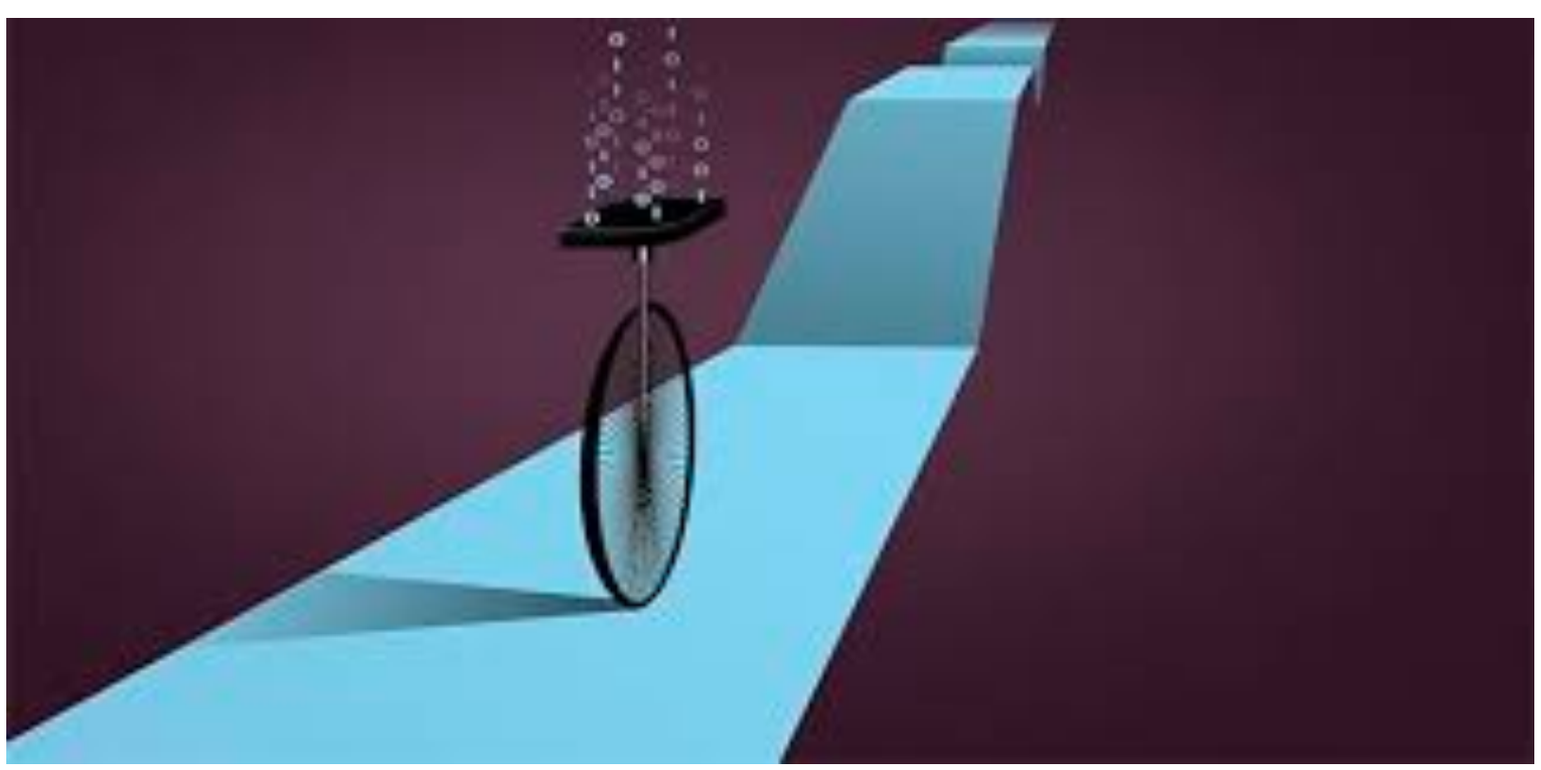
Summary

GPs are a simple and elegant way of defining distributions over functions

Building block for more complex models (audio textures in last lecture)

Proper treatment of uncertainty (optimal control, active sensing), Occam's razor

Deep mathematical connections to a variety of popular ML models



DS-GA 3001.008 Modelling time series data

L8b. Intro to RNNs

Instructor: Cristina Savin

NYU, CNS & CDS

The probabilistic ‘pipeline’

Domain-specific knowledge, Question

Data

Useful answers

Make assumptions → Discover patterns → Predictions

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

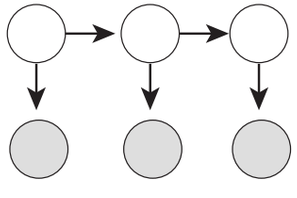
$\theta^* = \mathrm{argmax}_\theta \mathcal{L}$

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

observations, latents, hyperparameters dependences

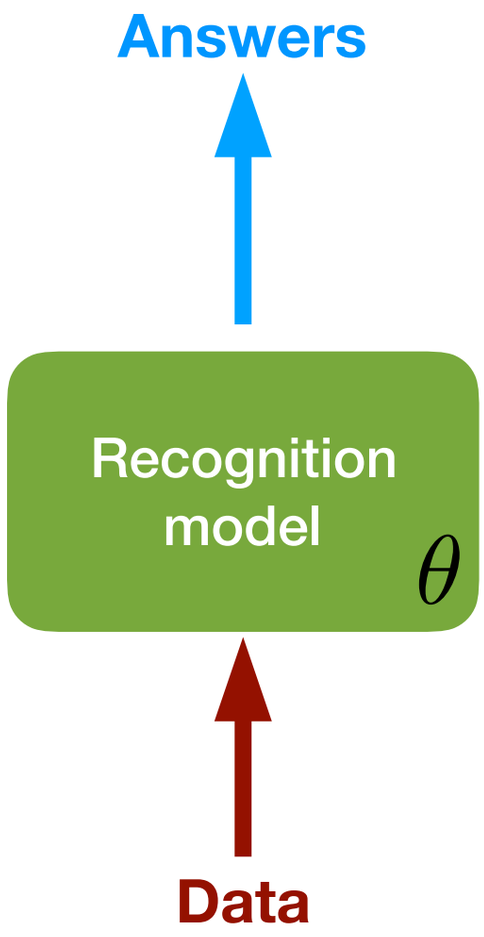
learn model

posterior inference



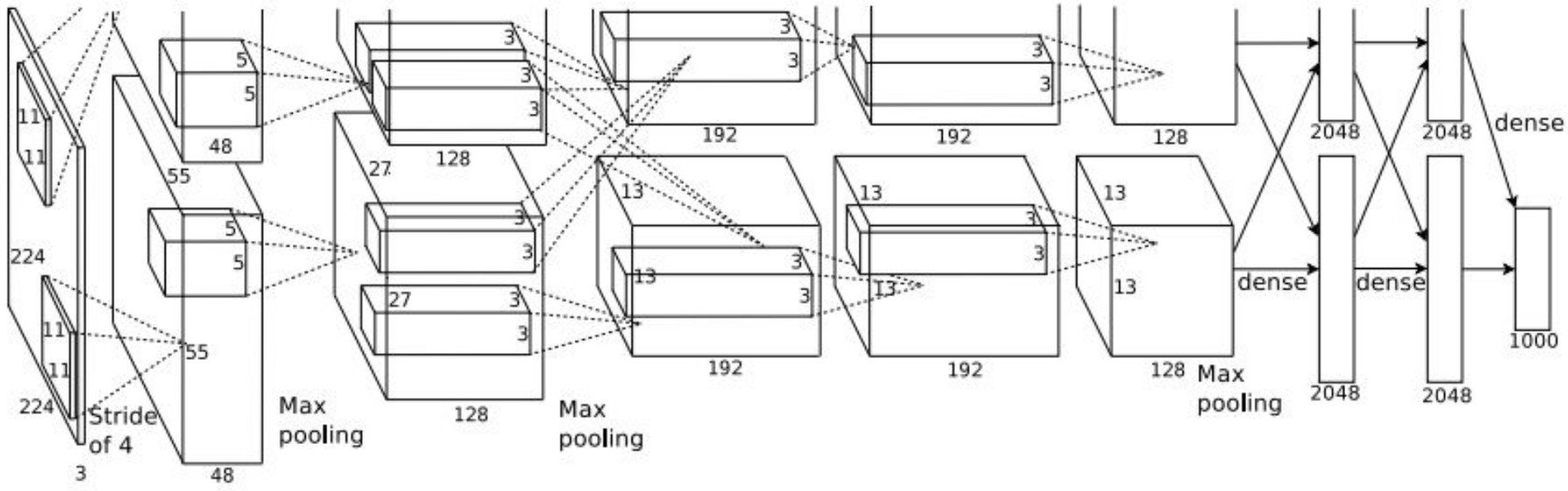
Pipeline separates **assumptions**, **computation**, **applications**

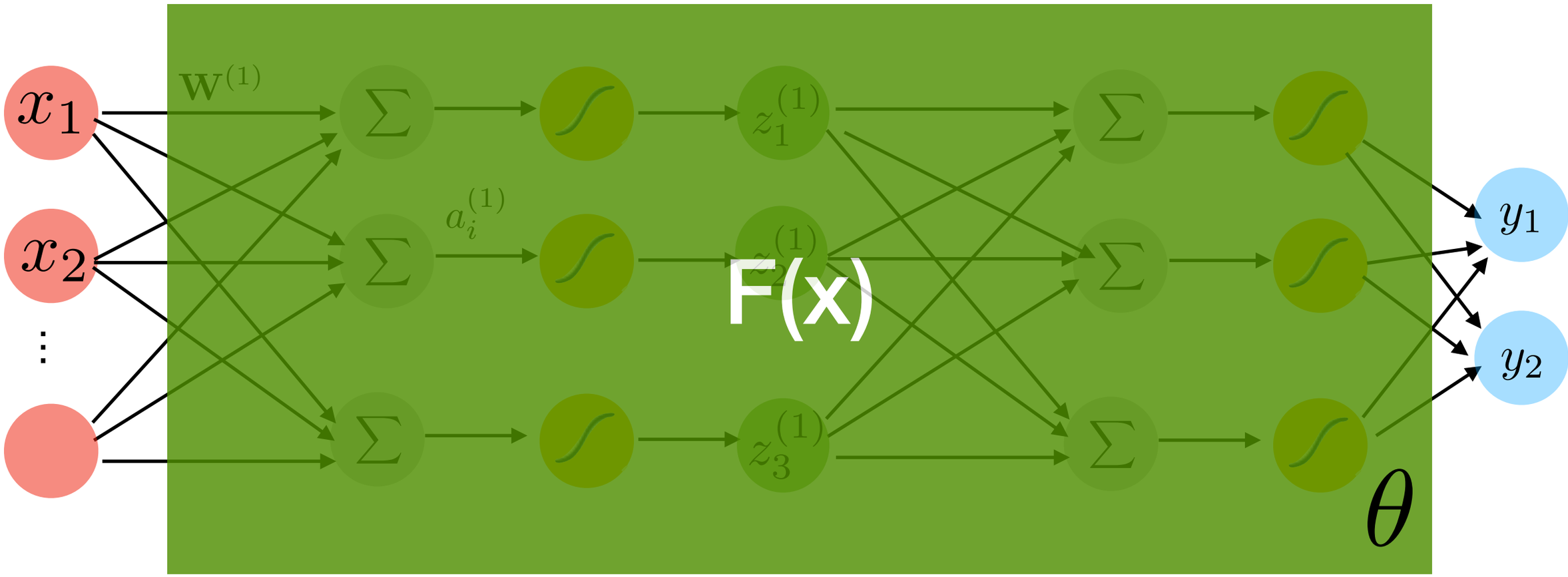
The alternative: just build a recognition model from the get go



Neural networks: flexible input-out map

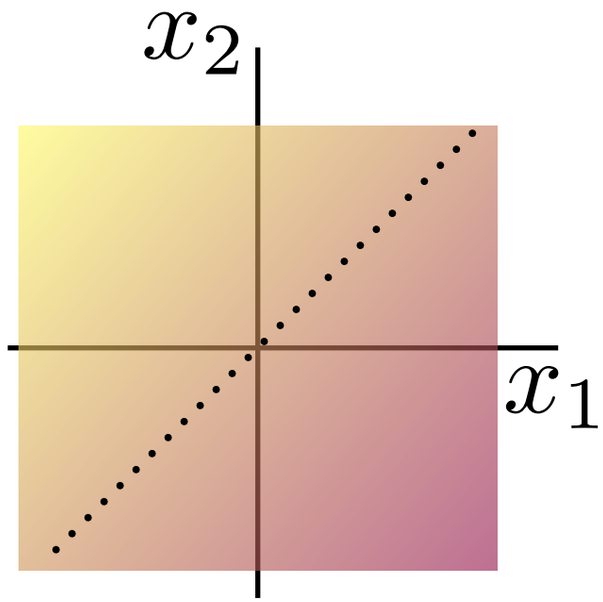
Backpropagation: how to train such models



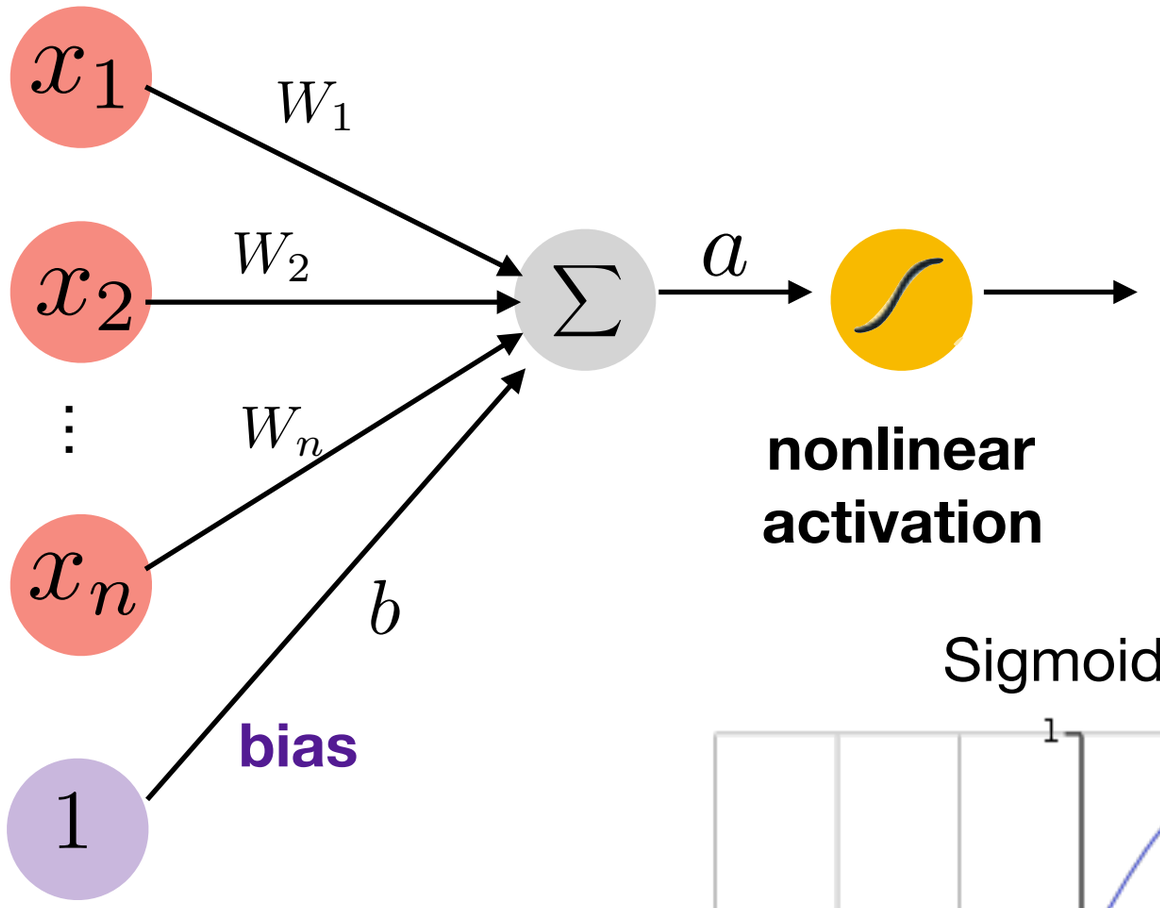


A cascade of linear-nonlinear steps-> a differentiable map

The basic building block: perceptron



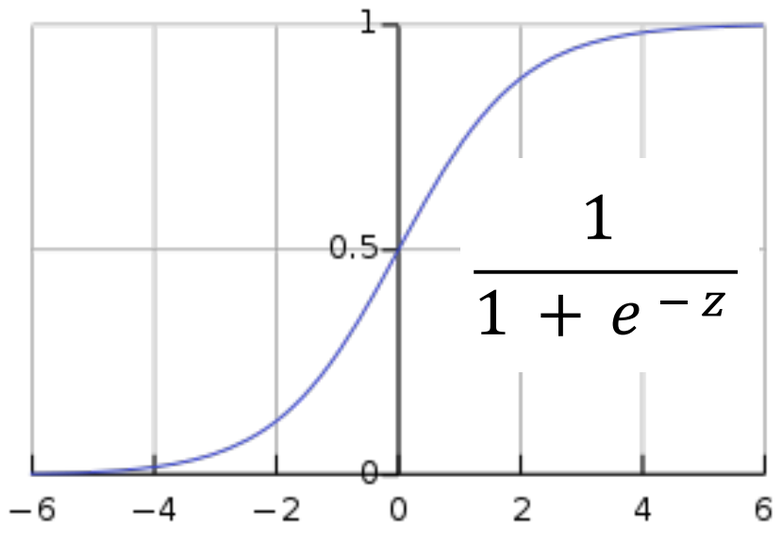
inputs



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

nonlinear activation

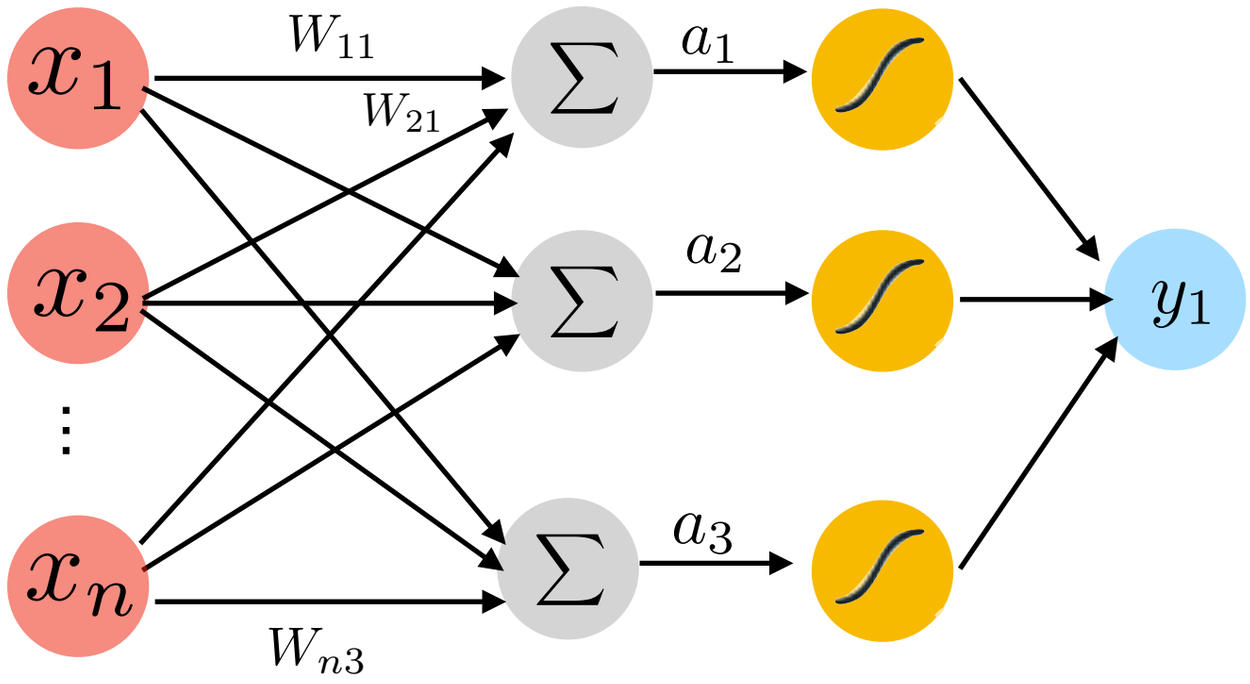
Sigmoid

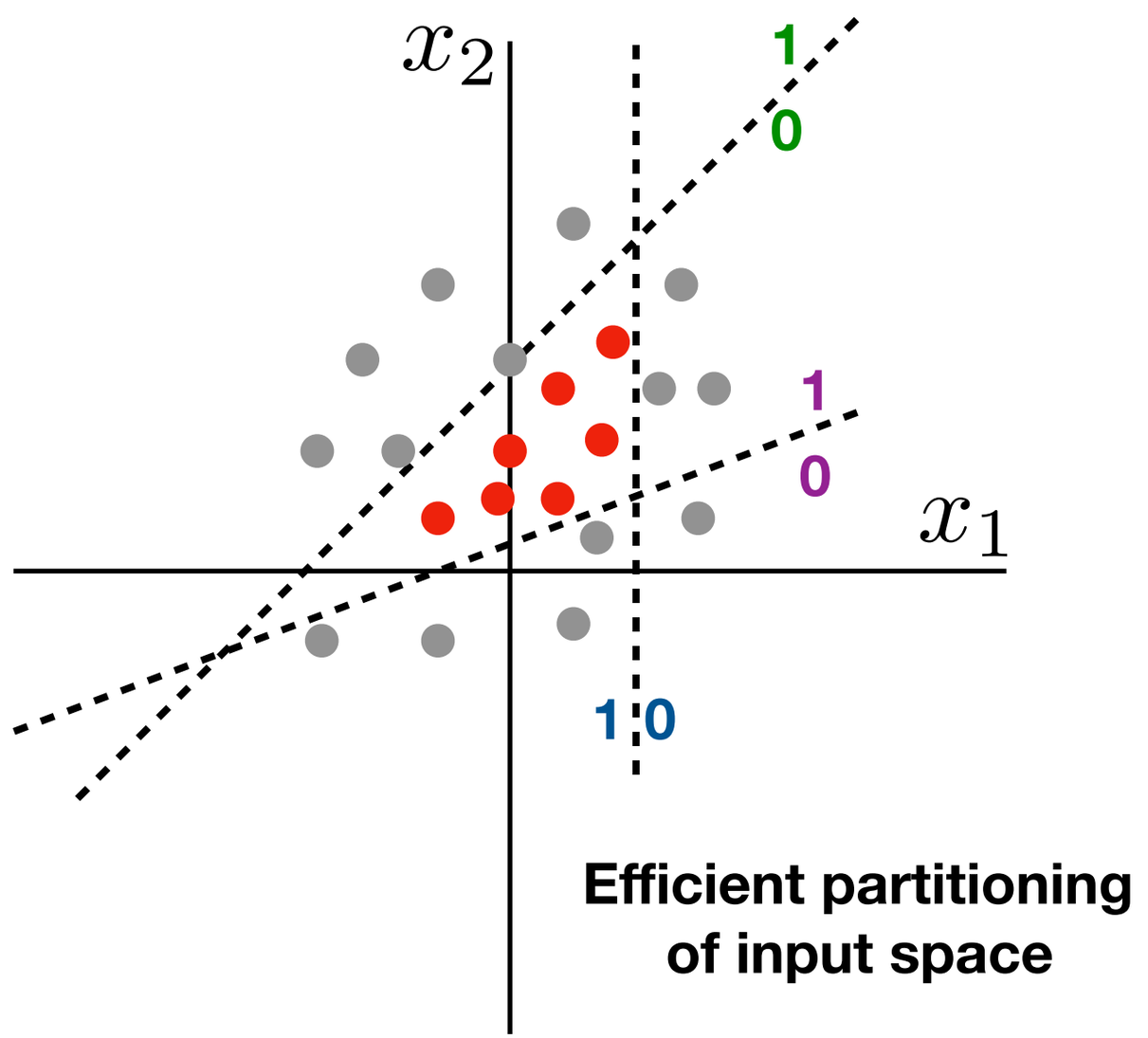


tanh: $\frac{e^{z}-e^{-z}}{e^{z}+e^{-z}}$

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.

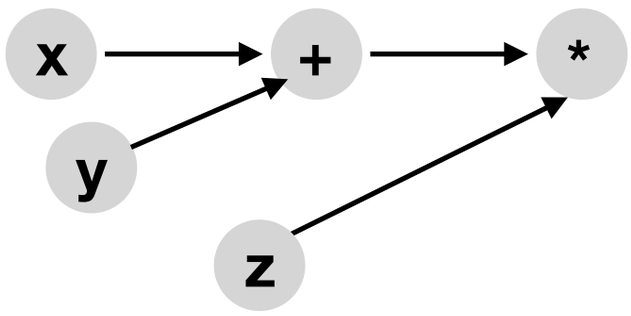
Caveat: It does not necessarily mean that there exists a learning algorithm that finds the correct solution

How do we compute the gradient?

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

Simple trick: repeatedly apply the chain rule

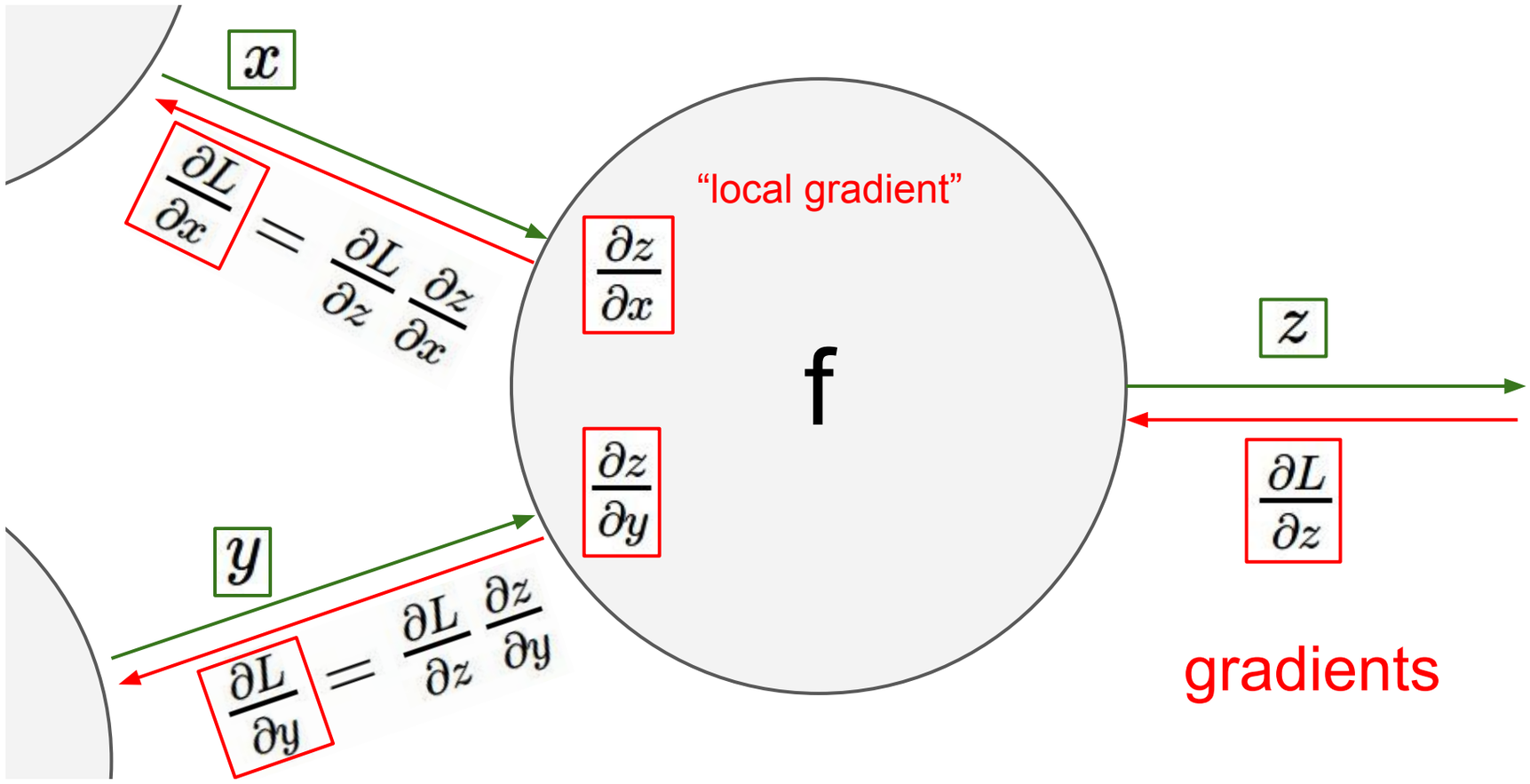
Let's take some simple examples (on the board)



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

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

Backpropagation



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

Vanilla RNNs math

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

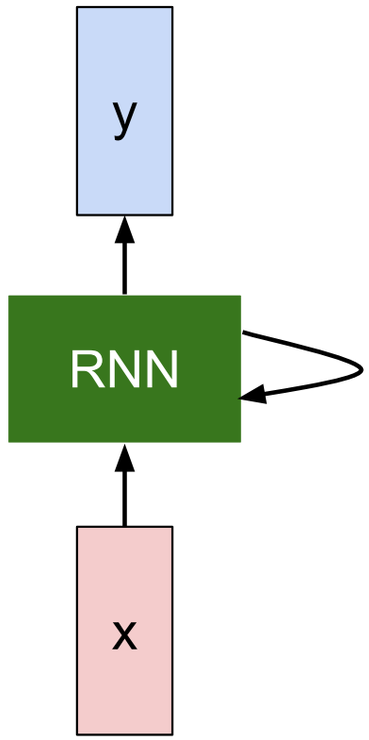
$$h_t = f_W(h_{t-1}, x_t)$$

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

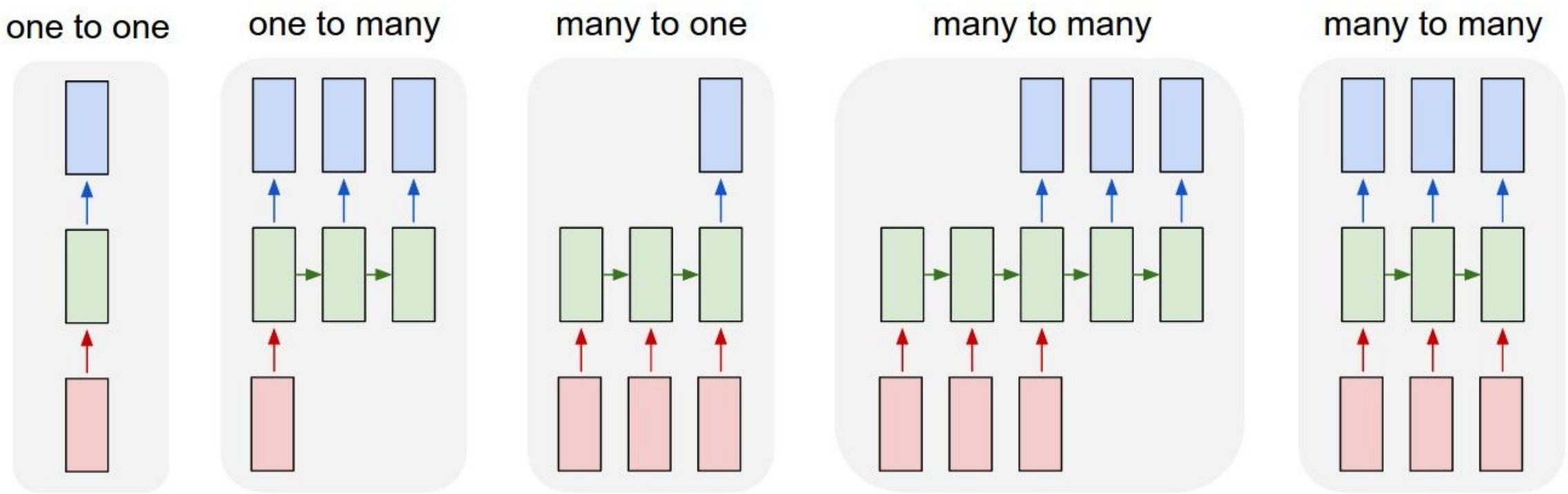


image-> caption

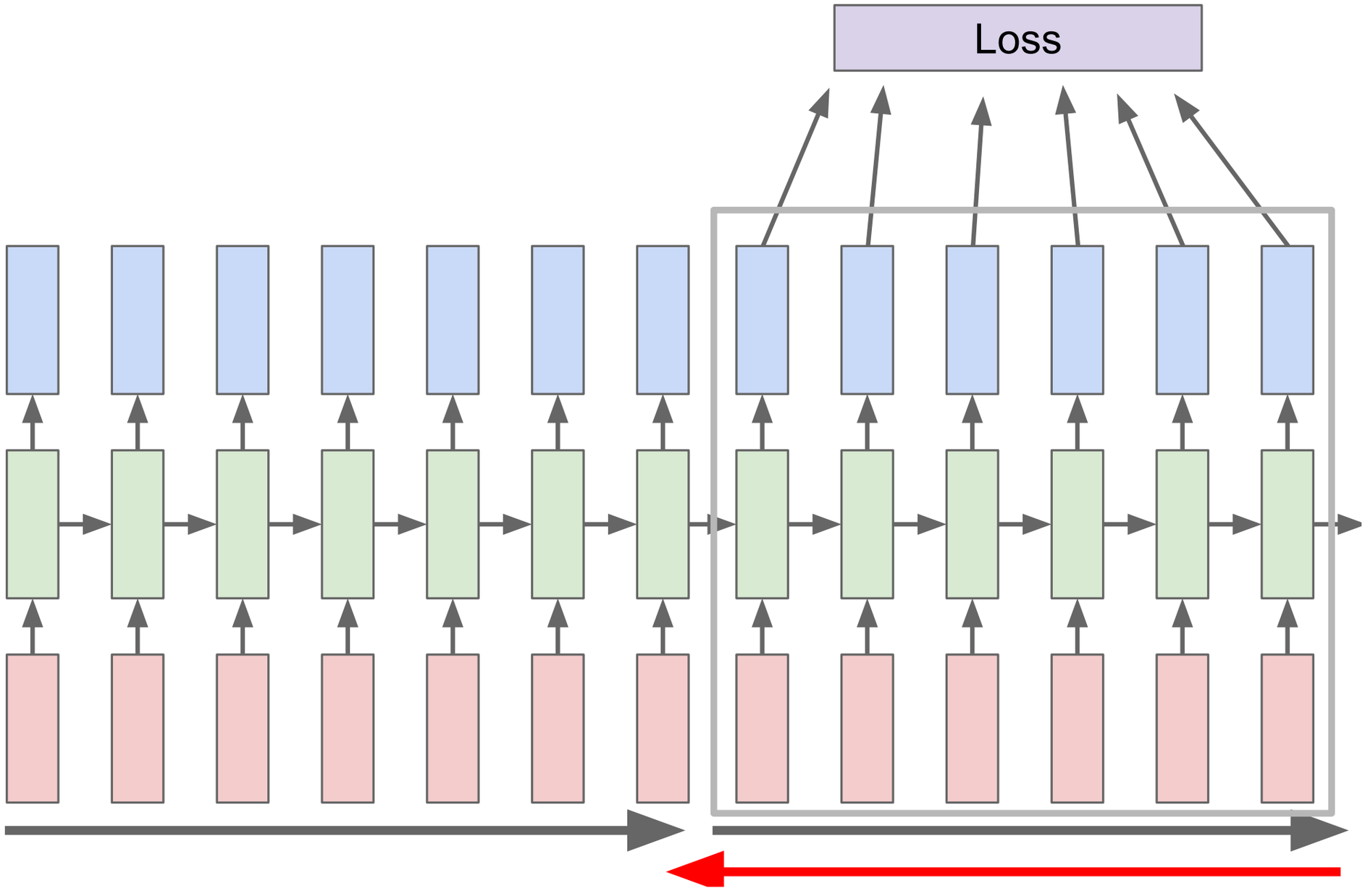
video-> label

sentence-> sentence

video-> video

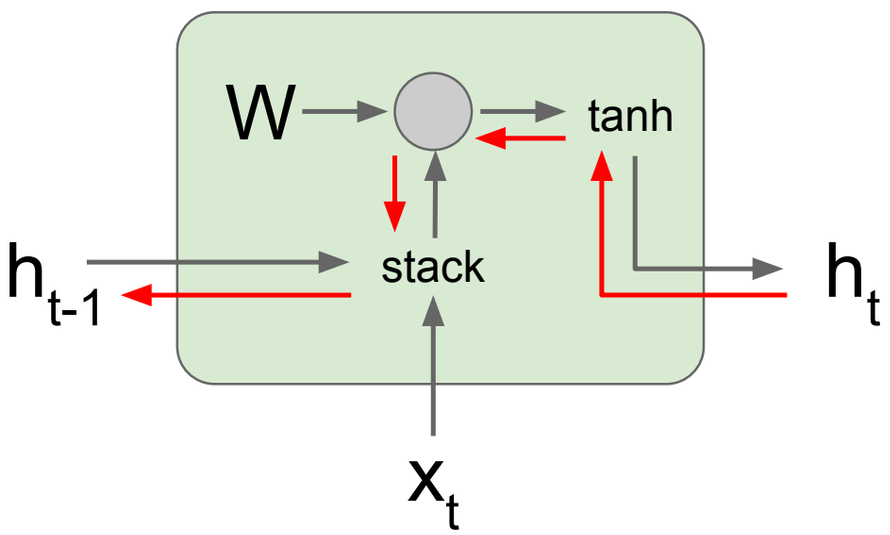
text-> text

Truncated backprop through time



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

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

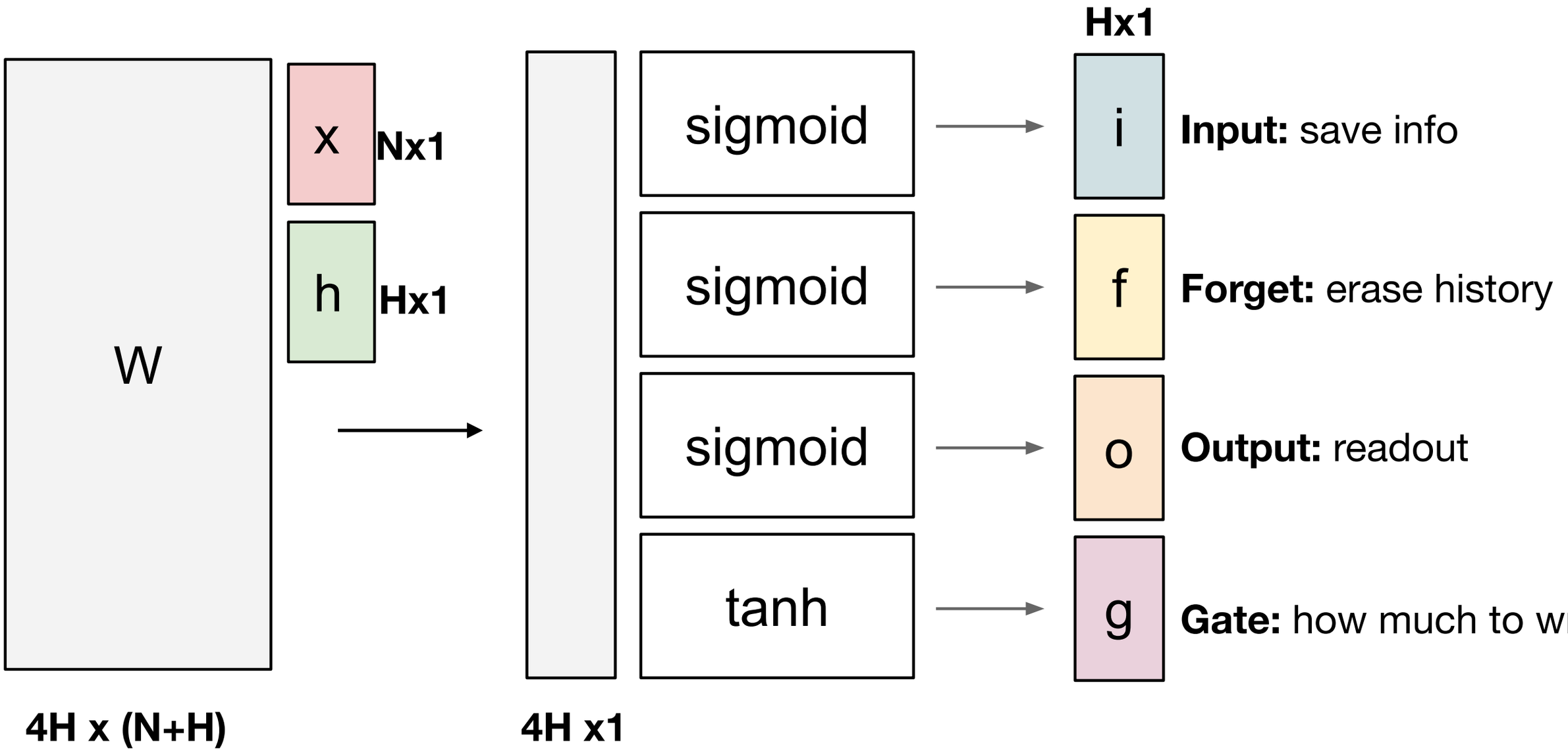
$$c_t = f \odot c_{t-1} + i \odot g$$

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

Hochschreiter & Schmidhuber, 1997

$\odot$ Element-wise multiplication

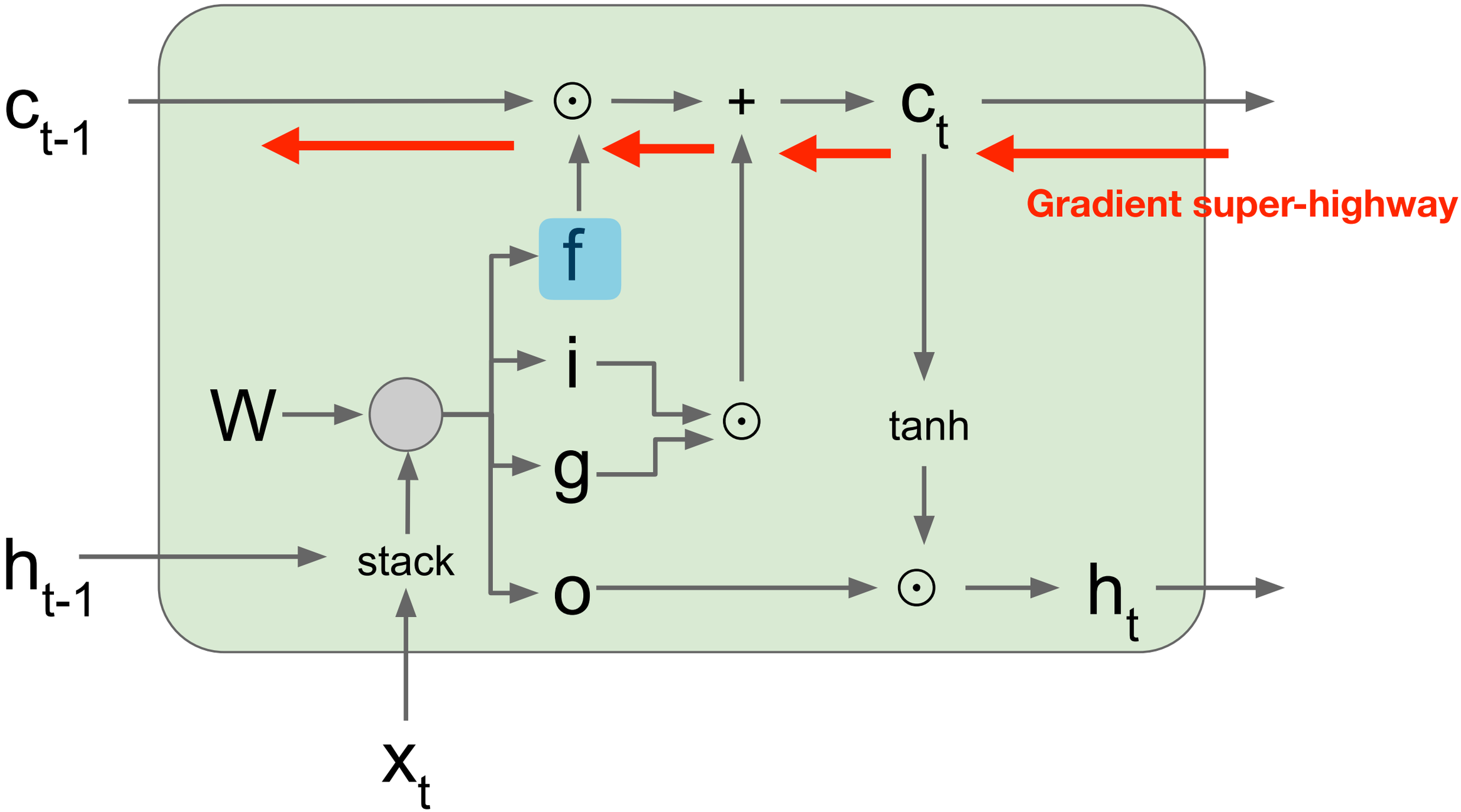
LSTM details



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

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

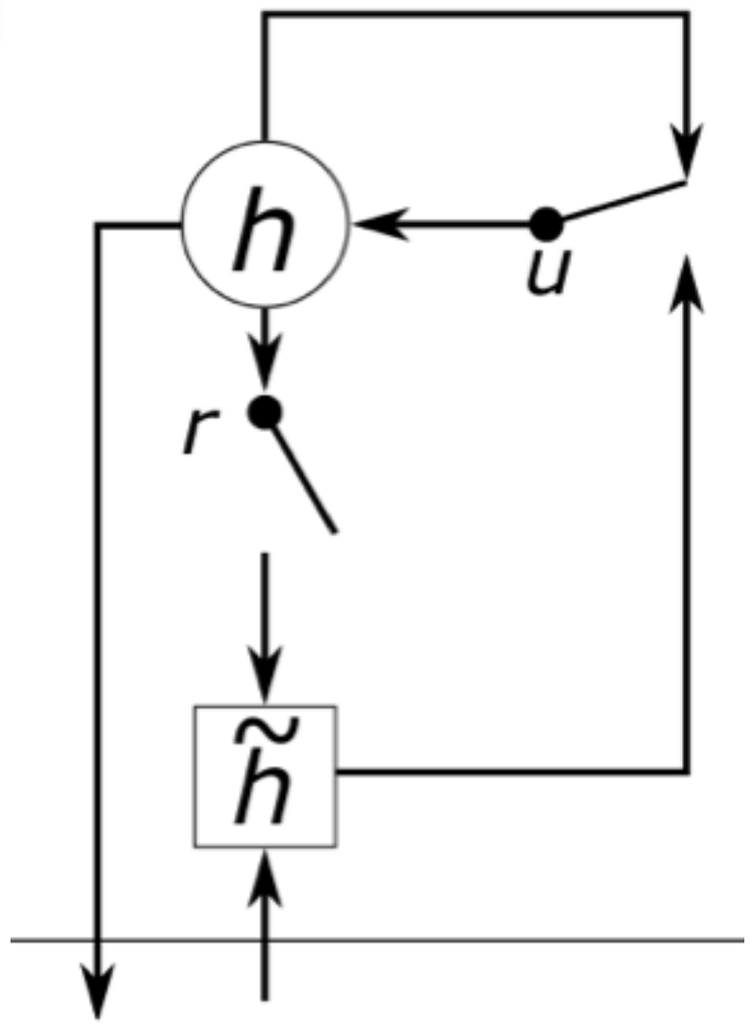
Gradient for c multiplied (element wise) with f, which varies over time and bounded so much more numerically stable



Other architectures

GRU (Cho et al, 2013)

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



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