

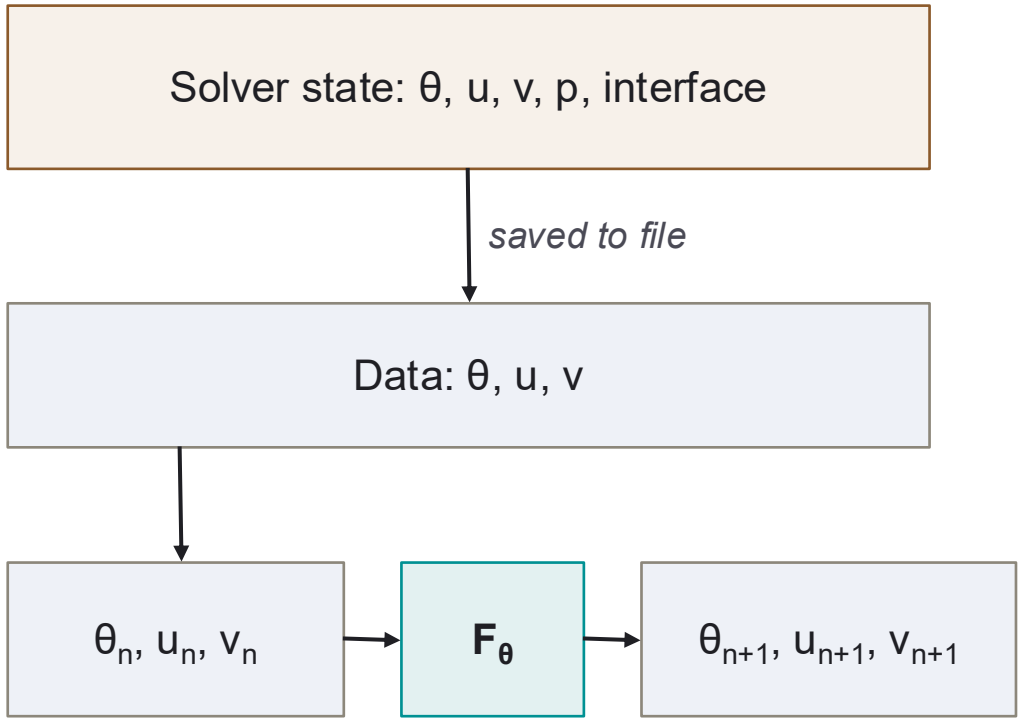
ML FOR SCIENCE · LECTURE 8

Recurrent Networks and Neural ODEs

Aparna Chandramowlishwaran · EECS 195/221 · Fall 2026

What does the next frame depend on?

- Recall lecture 7 model: F_θ maps the current frame of θ, u, v to the next frame
- The solver state also includes the pressure p and the position of the liquid-vapor interface
  - Our data file stores θ, u and v only
- Lecture 7, temporal bundling: 5 frames in and 1 frame out did not improve the rollout over 1 frame in
  - In that experiment the model received θ, u and v
- **Question**: when would past frames help the prediction?



F_θ sees θ, u, v only

θ: nondimensional temperature. u, v: velocity components. BubbleML.

Complete and partial states

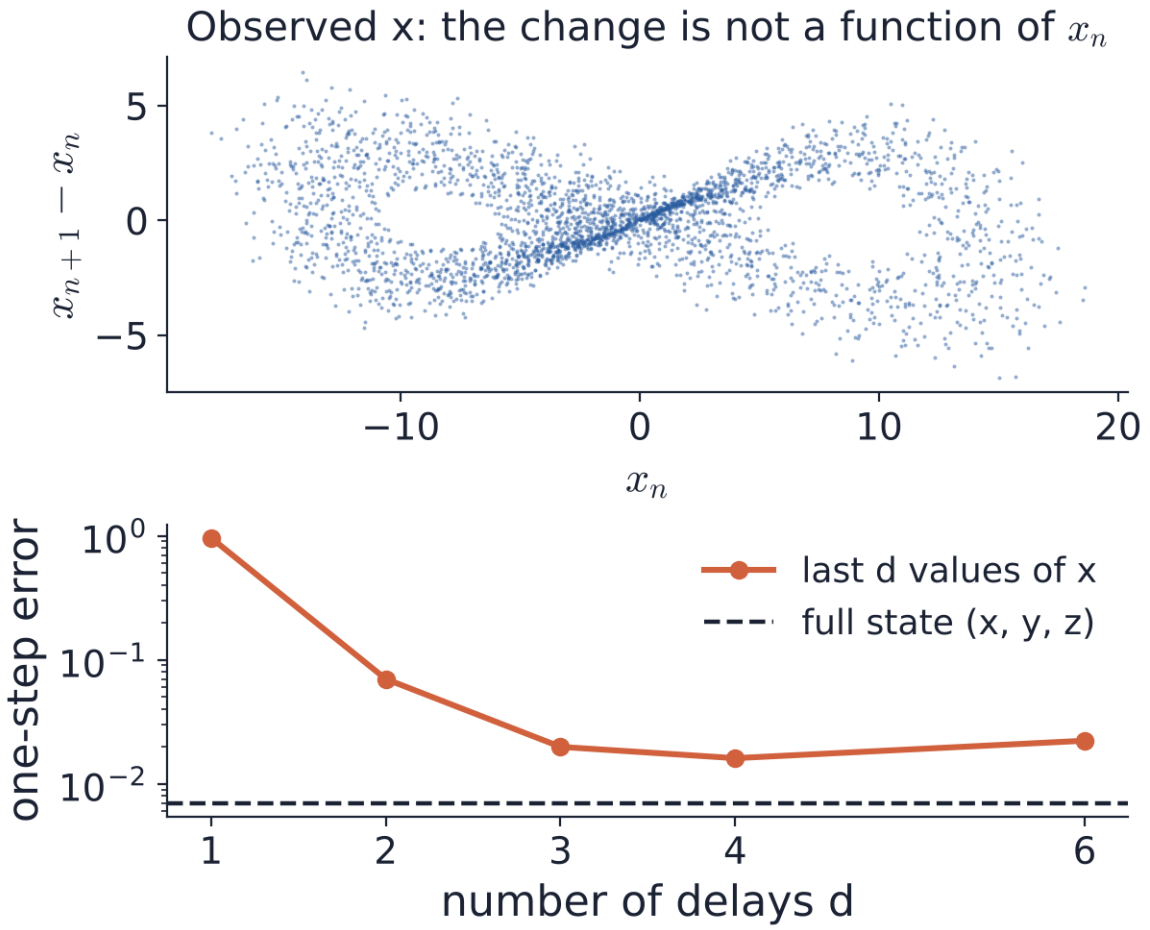
- A state u_n is **complete** if it determines the next state. Example: the heat equation

$$u_{n+1} = F(u_n)$$

- Often we observe only part of the state, through an observation map H (right)

$$y_n = H(u_n)$$

- y_{n+1} can be **multivalued** in y_n: one y_n, several possible next values
- Lorenz system (1963): state is (x, y, z). We only observe x
- Give the last d values of x, $(x_n, x_{n-1}, \ldots, x_{n-d+1})$, model recovers the information in the state (Takens 1981)



Past observations carry information about the unobserved part of the state

Lorenz-63: σ = 10, ρ = 28, β = 8/3, samples 0.05 time units apart. MLP (2 × 64, tanh) predicts $x_{n+1} - x_n$; error on a separate test trajectory. Takens 1981

Two forms of model memory

- **Frame window**: stack the **last k frames** as input channels
  - Reach of k frames
  - Input size grows with k

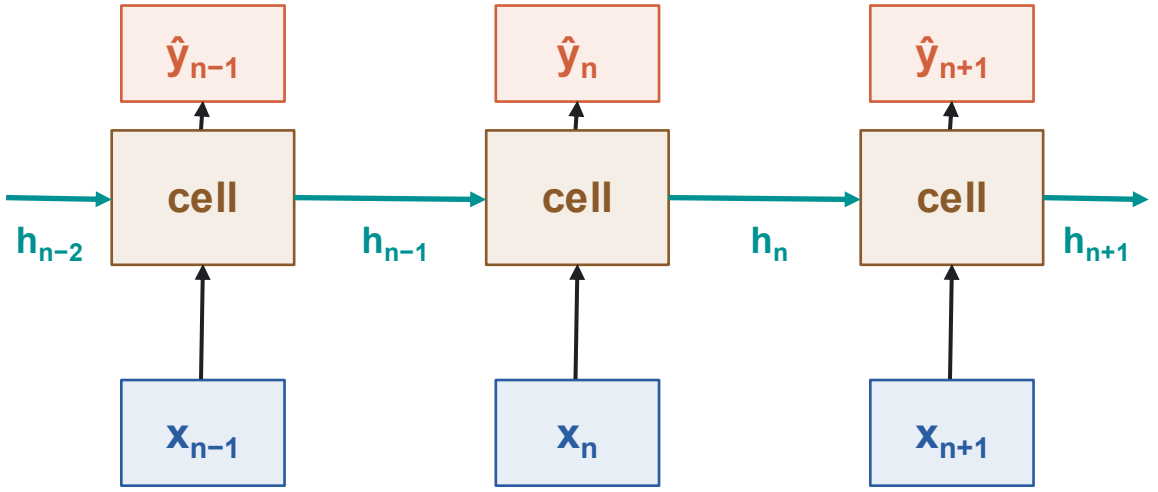
- **Hidden state h_n**: a **learned summary of past inputs**, updated at every step
  - Same cost per step for any history length
  - Can carry information over many steps

- Both are trained on sequences of frames

Frame window, k = 4



Hidden state

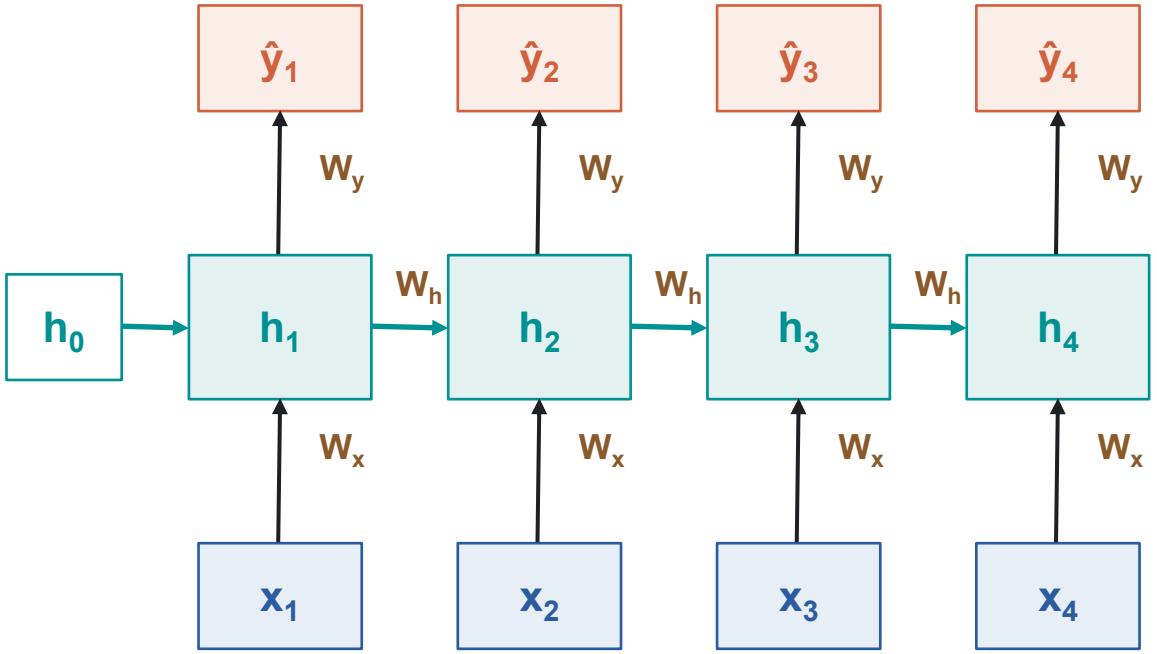


x: input **h**: hidden state **ŷ**: output **W, θ**: parameters

Recurrent neural networks

$$h_n = \tanh(W_h\, h_{n-1} + W_x\, x_n + b)\,, \qquad \hat{y}_n = W_y\, h_n + b_y$$

- $\mathbf{x_n}$: input at step n. $\mathbf{h_n}$: hidden state. $\hat{\mathbf{y}}_n$: output
- The same parameters $\mathbf{W_h}$, $\mathbf{W_x}$, $\mathbf{W_y}$, $\mathbf{b}$, $\mathbf{b_y}$ at every step
  - Sharing across time, as a convolution shares weights across space
- $\mathbf{h_0}$ = 0, or a learned vector
- Elman network (Elman 1990)
- As a rollout model: $\mathbf{x_n}$ is the current state, and the output is the change to the next state



The same cell, unrolled over 4 steps

W_h: hidden-to-hidden weights. W_x: input-to-hidden. W_y: hidden-to-output. b, b_y: biases. tanh is applied elementwise. Elman, Cognitive Science 1990

Backpropagation through time

$$\mathcal{L} = \sum_{n=1}^{N} \mathcal{L}_n, \qquad \frac{\partial \mathcal{L}}{\partial W_h} = \sum_{n=1}^{N} \sum_{k=1}^{n} \frac{\partial \mathcal{L}_n}{\partial h_n} \frac{\partial h_n}{\partial h_k} \frac{\partial h_k}{\partial W_h}$$

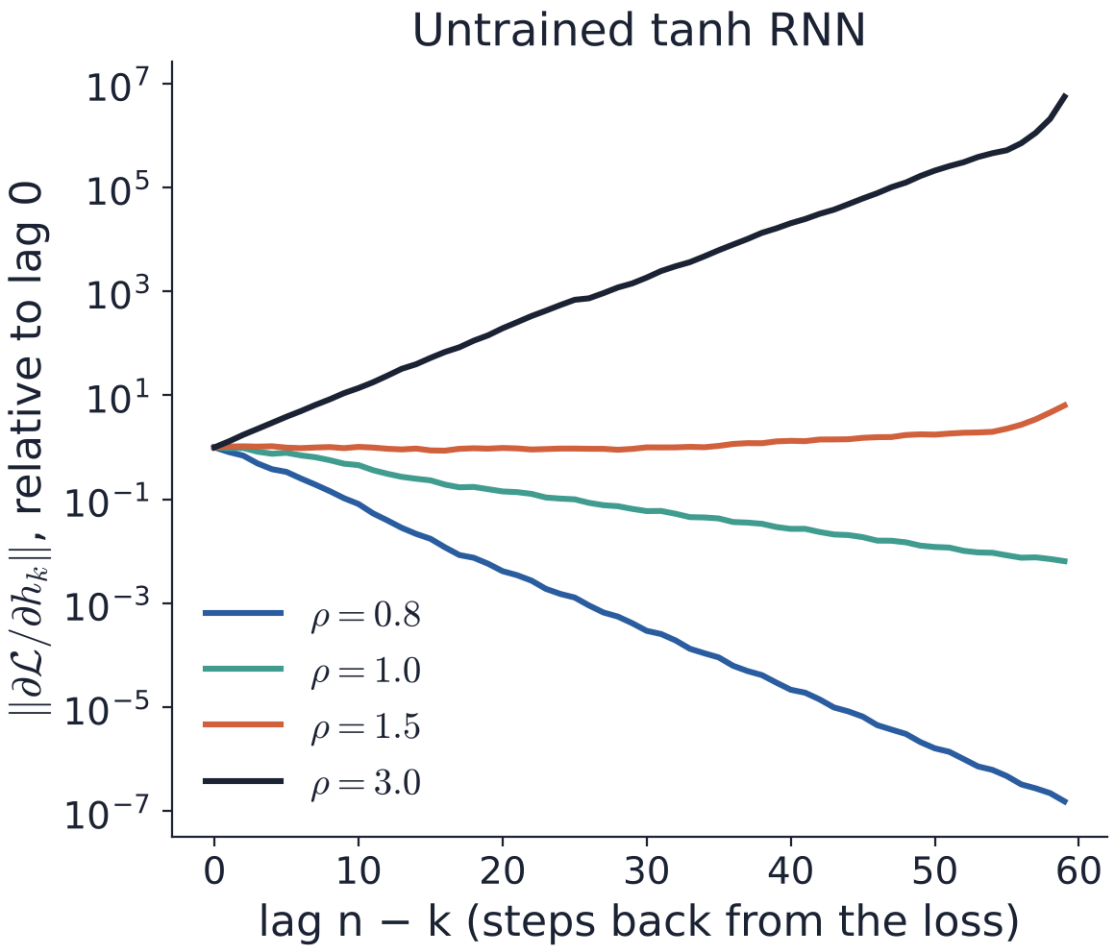
- Unroll the network over N steps and backpropagate through the unrolled graph (Werbos 1990)
- The loss at step n depends on W_h through every earlier hidden state h_k, $k \leq n$
- Same computation as **unrolled training**
  - N forward and N backward passes; memory for N steps of activations
- **Truncated BPTT**: backpropagate through the last K steps only; the hidden state is carried forward without a gradient beyond that
- **Teacher forcing**: during training, the input x_n is the true state

$\mathcal{L}_n$: loss at step n. N: sequence length. K: truncation length. Werbos, Proc. IEEE 1990

Vanishing and exploding gradients

$$\frac{\partial h_n}{\partial h_k} = \prod_{j=k+1}^{n} \frac{\partial h_j}{\partial h_{j-1}} = \prod_{j=k+1}^{n} \operatorname{diag}\left(\tanh'(z_j)\right) W_h$$

- The gradient from step n back to step k passes through n − k Jacobians
- $|\tanh'| \le 1$, so the norm is at most $\|W_h\|^{n-k}$
  - $\|W_h\| < 1$: the gradient **vanishes** exponentially with the lag n − k
  - Largest eigenvalue magnitude > 1: it can **explode**
- Lecture 7: rollout error is amplified by a product of Jacobians J in the same way
- Gradient clipping limits exploding gradients (Pascanu et al. 2013). Vanishing gradients need a change of architecture



z_j: input to tanh at step j. Figure: untrained tanh RNN, 32 hidden neurons, W_h rescaled to spectral radius ρ, random inputs. Bengio, Simard & Frasconi 1994; Pascanu, Mikolov & Bengio, ICML 2013

Gating: keep the old state or write a new one

- The **plain RNN** overwrites h at every step, so old information is lost

$$h_n = \tanh(W\,[h_{n-1}, x_n] + b)$$

- **Gated update**: a gate $z_n \in [0, 1]$ for each neuron mixes the old state and a candidate
  - $z_n = 0$: keep h_{n-1}. $z_n = 1$: replace it with $\tilde{h}_n$

$$\tilde{h}_n = \tanh(W\,[h_{n-1}, x_n] + b) \qquad \textit{candidate}$$

$$h_n = (1 - z_n) \odot h_{n-1} + z_n \odot \tilde{h}_n \qquad \textit{gated update}$$

- Right, one neuron with z_n set by hand. (A) writes at step 5 and then holds the value. (B) averages the inputs

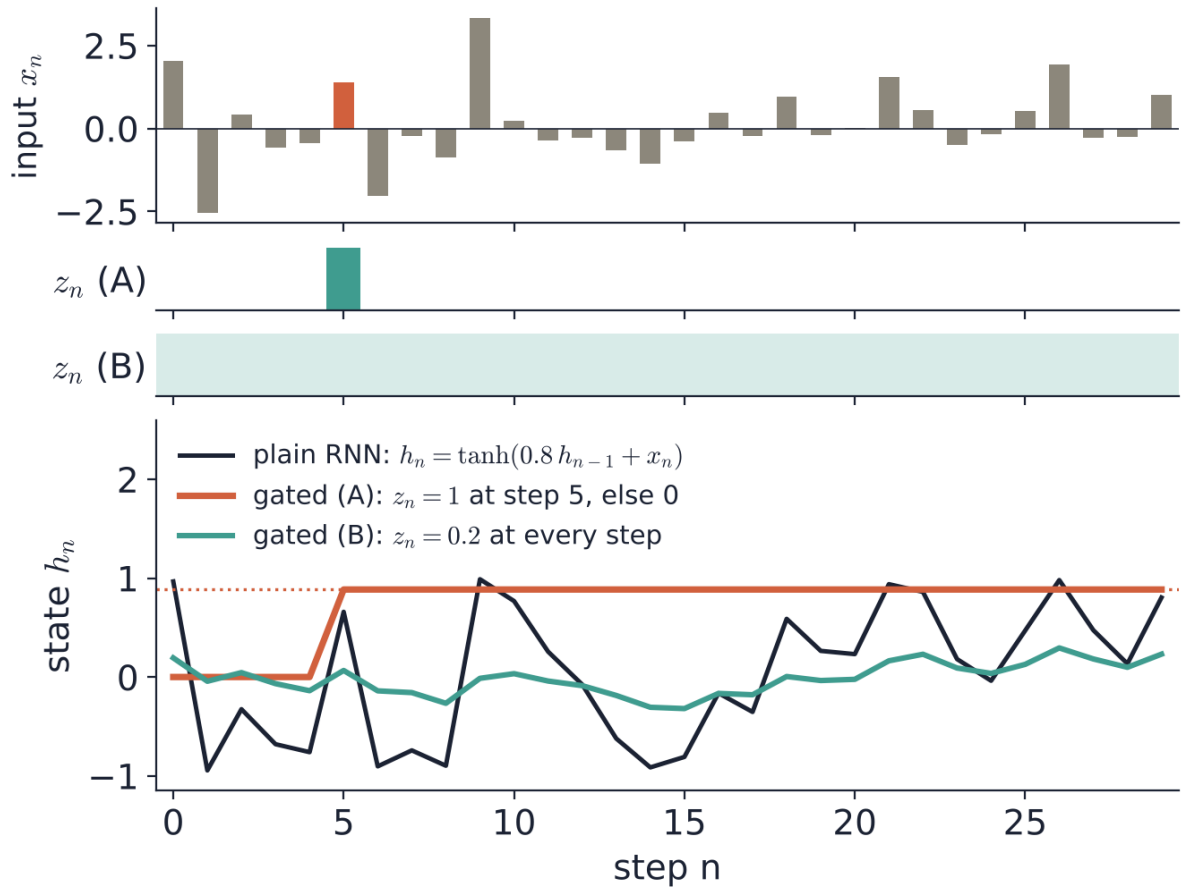


Figure: same input sequence for all three; $h_0 = 0$, $\tilde{h}_n = \tanh(x_n)$ for the gated cases; dotted line: $\tanh(x_5)$. $\odot$: elementwise product. [h, x]: concatenation

Gated recurrent unit (GRU)

- Gated update (previous slide): copy h_{n-1} or overwrite it with a candidate $\tilde{h}_n$

$$h_n = (1 - z_n) \odot h_{n-1} + z_n \odot \tilde{h}_n$$

- Update gate z_n: computed by the network from h_{n-1} and x_n; σ keeps it in (0, 1)

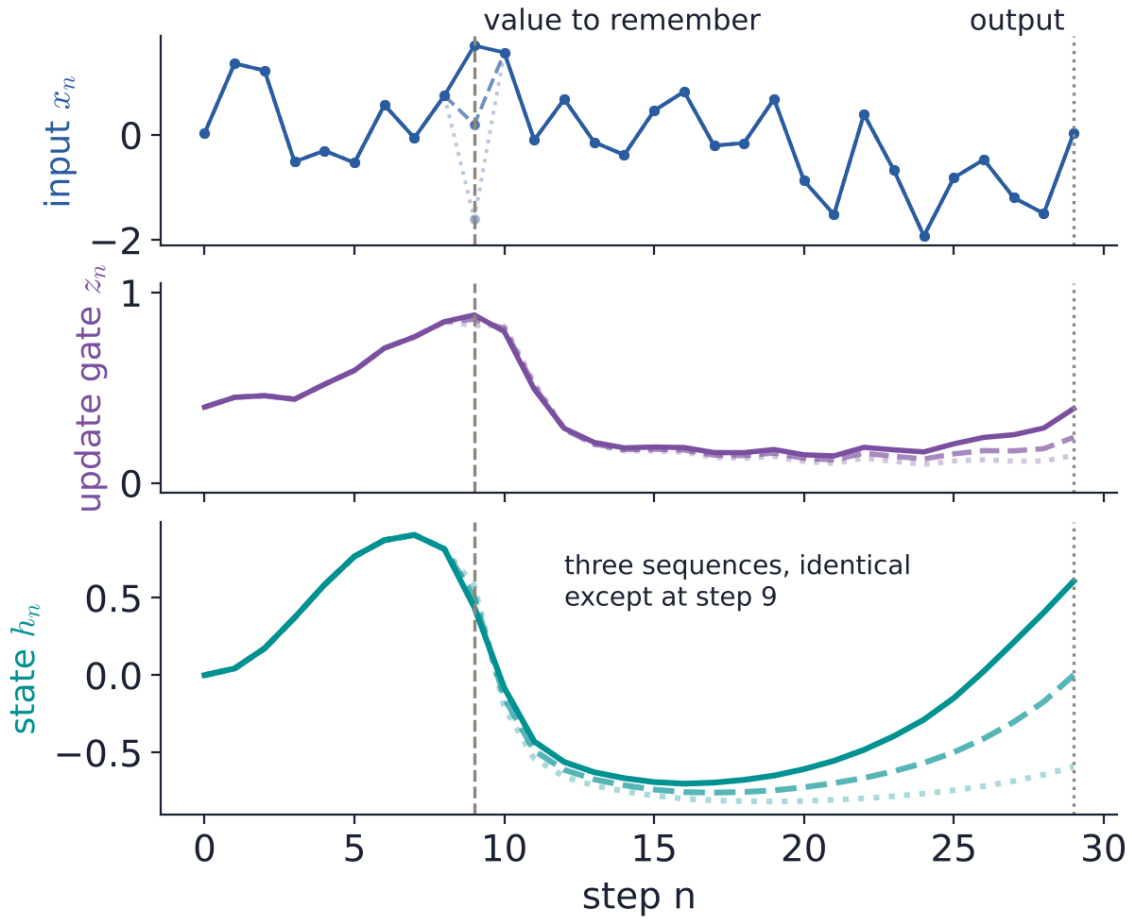
$$z_n = \sigma(W_z \, [h_{n-1}, x_n] + b_z)$$

- Reset gate r_n: how much of h_{n-1} the candidate uses. At $r_n = 0$ the candidate is built from x_n alone

$$r_n = \sigma(W_r \, [h_{n-1}, x_n] + b_r)$$

$$\tilde{h}_n = \tanh(W \, [r_n \odot h_{n-1}, \; x_n] + b)$$

- Figure: a trained GRU. z_n drops after the value arrives, and h_n holds it until the output



W, **b**: weights and biases. σ: sigmoid. Figure: GRU, 16 neurons, trained to output at step 29 the input of step 9; the neuron whose final state best predicts the target. Cho et al., EMNLP 2014

Long short-term memory (LSTM)

- A separate memory, the cell state **c_n**, and three gates computed like the GRU gates

$$f_n = \sigma(W_f\,[h_{n-1}, x_n] + b_f)$$
$$i_n = \sigma(W_i\,[h_{n-1}, x_n] + b_i)$$
$$o_n = \sigma(W_o\,[h_{n-1}, x_n] + b_o)$$

- Gated update of the cell: **f_n** keeps **c_{n-1}** (the GRU's $1 - z_n$), **i_n** writes the candidate **$\tilde{c}_n$** (the GRU's **z_n**)

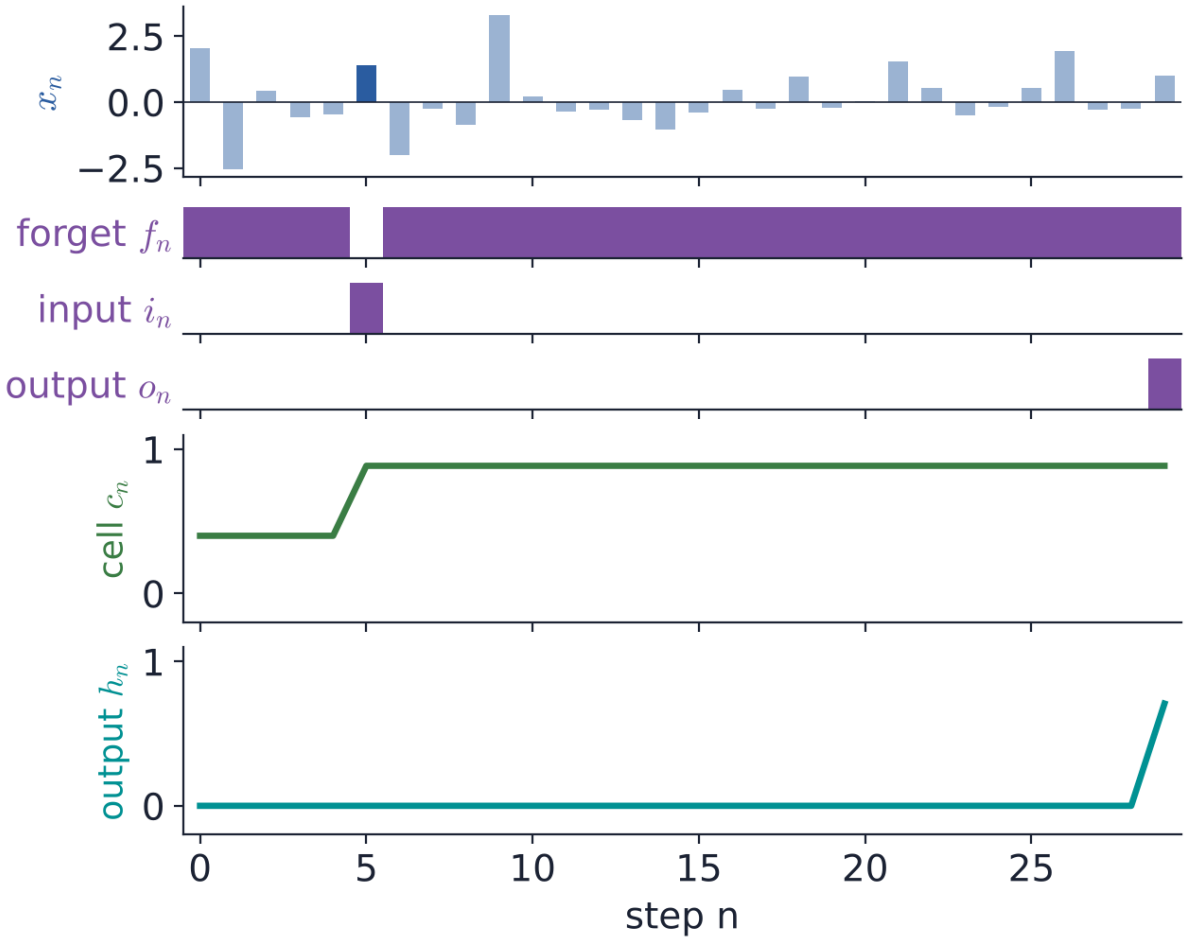
$$\tilde{c}_n = \tanh(W_c\,[h_{n-1}, x_n] + b_c)$$

$$c_n = f_n \odot c_{n-1} + i_n \odot \tilde{c}_n$$

- Output gate **o_n**: how much of the memory to show as **h_n**

$$h_n = o_n \odot \tanh(c_n)$$

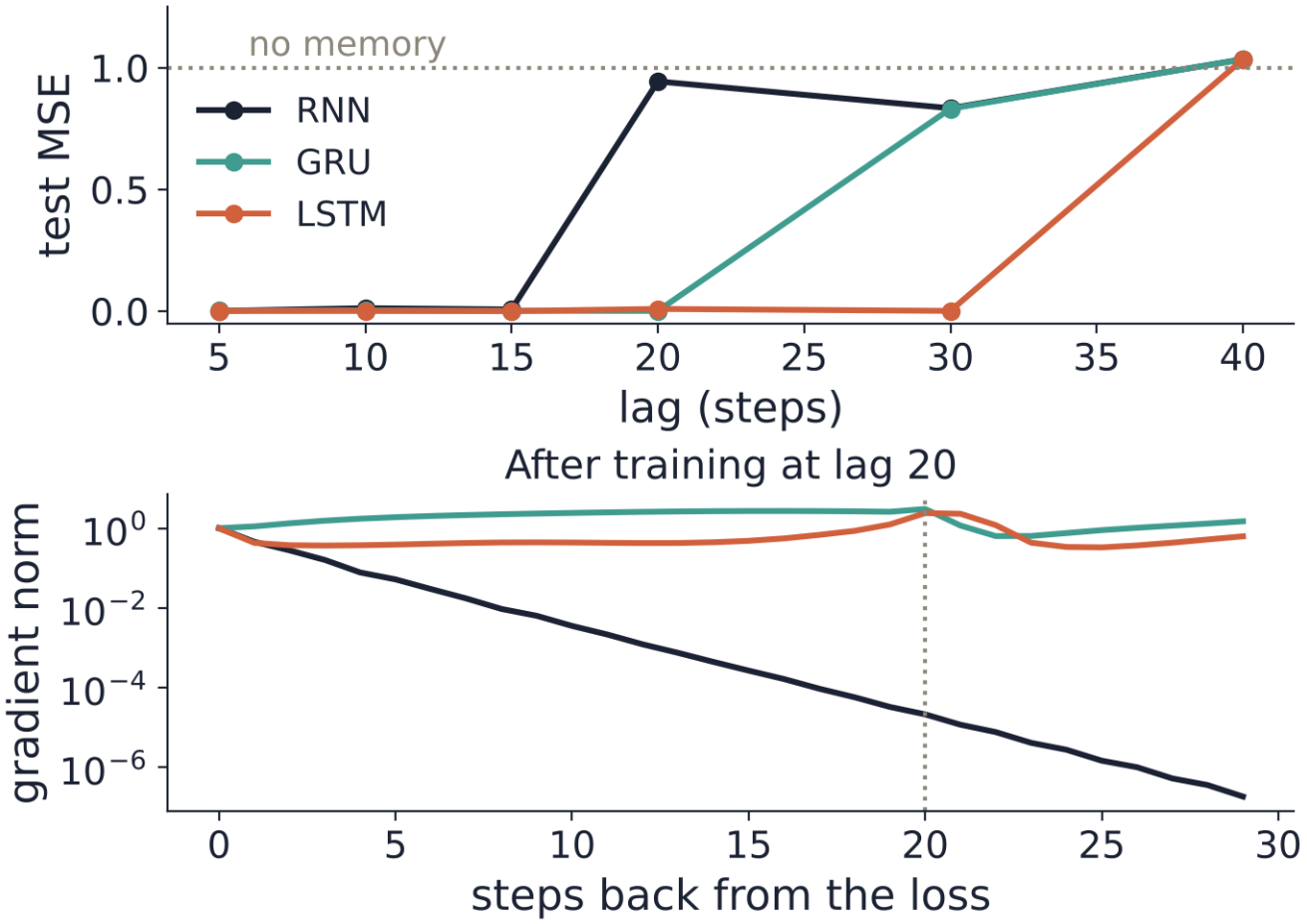
- Figure: gates set by hand: write at step 5, keep, show at step 29. With **f_n** = 1 the gradient passes from **c_n** to **c_{n-1}** unchanged



W, **b**: weights and biases. Figure: one neuron, $c_0 = 0.4$, $\tilde{c}_n = \tanh(x_n)$; purple: gate = 1, white: gate = 0. Hochreiter & Schmidhuber 1997; forget gate: Gers et al. 2000; forget-gate bias 1: Jozefowicz et al. 2015

Recurrent networks on a memory task

- **Input**: white noise, one value at each step. Target at the last step: the input *lag* steps earlier
- Mean squared error 1 is the variance of the target: no memory
- **Top**: the plain RNN fails at the shortest lag. The GRU reaches longer lags, the LSTM the longest
- **Bottom**: after training, the gradient of the GRU and LSTM stays near its size at lag 0; the RNN's decays exponentially



Hidden size 32. Adam (lr 3 $y10^{-3}$), batch 64, 2000 steps, gradient clipping at norm 1. Sequence length = lag + 10. LSTM forget-gate bias initialized to 1. Hochreiter & Schmidhuber 1997

Recurrent networks for chaotic dynamics

- **Reservoir computing**: a large recurrent network with fixed random weights. Only the output layer is trained, by linear regression
- **Kuramoto-Sivashinsky** (Lecture 3): the prediction (b) follows the true field (a) for several Lyapunov times, then diverges (c)
  - Lyapunov time: average time for a small error to grow by a factor e
- Any model of a **chaotic system** loses pointwise accuracy after a few Lyapunov times. **Long-term statistics** (spectra, means) are the test after that
- Trained LSTMs give similar short-term forecasts of reduced-order chaotic models (Vlachas et al. 2018)

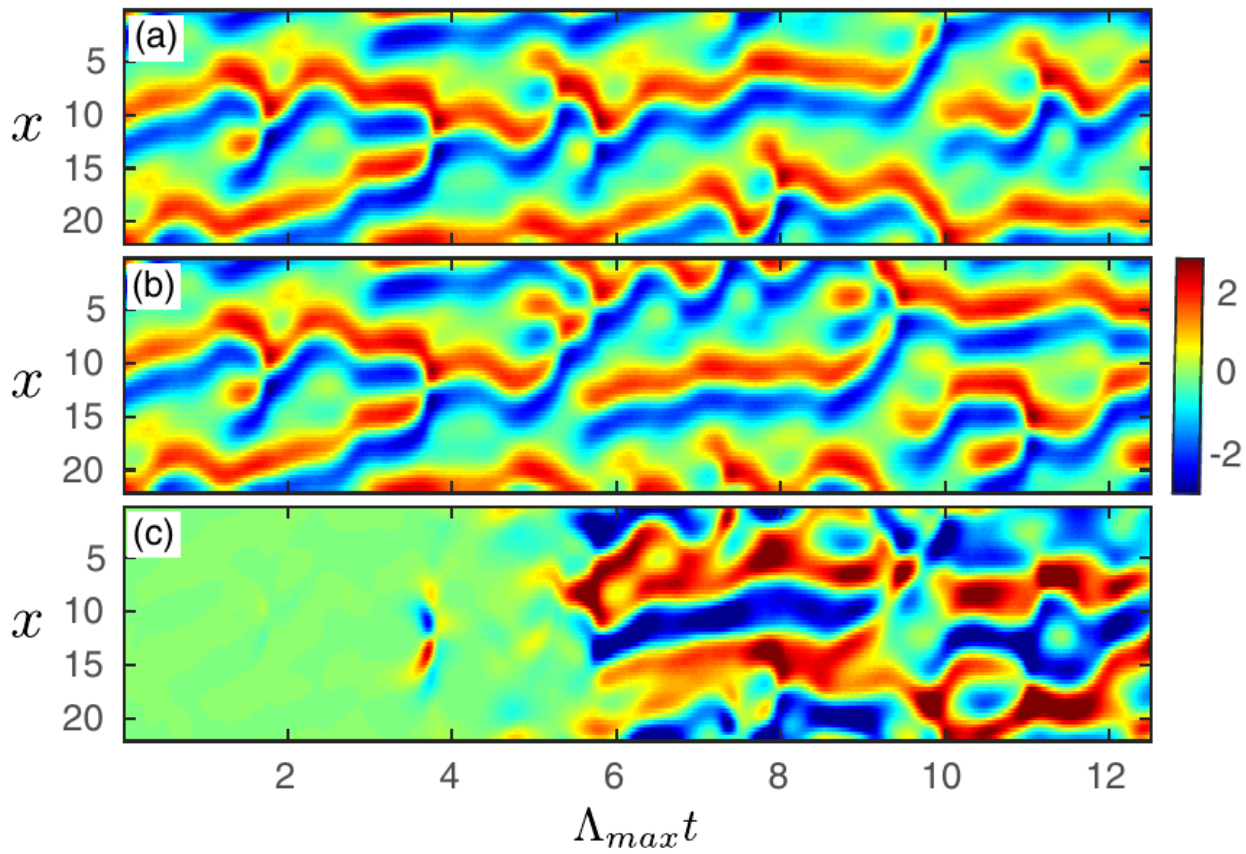
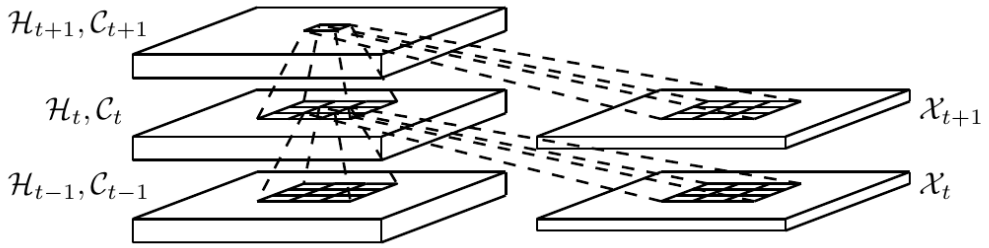


Figure: Pathak, Hunt, Girvan, Lu & Ott, Model-free prediction of large spatiotemporally chaotic systems from data, Physical Review Letters 2018
Vlachas, Byeon, Wan, Sapsis & Koumoutsakos, Proc. R. Soc. A 2018

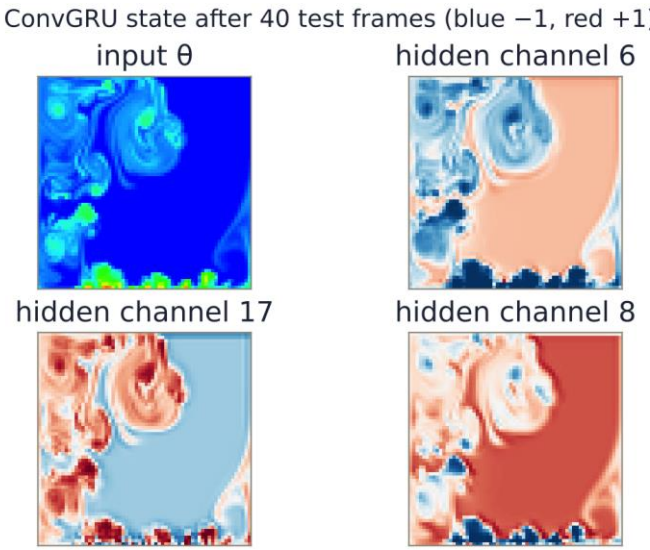
Convolutional recurrent networks

$$h_n = (1 - z_n) \odot h_{n-1} + z_n \odot \tanh(W * [\, r_n \odot h_{n-1},\ x_n \,] + b)$$

- **ConvGRU**: the GRU update (slide 9) with each matrix product **W** replaced by a convolution **W** $*$
  - Inputs $\mathbf{x_n}$, hidden state $\mathbf{h_n}$ and gates $\mathbf{z_n}$, $\mathbf{r_n}$ are fields with channels; the gates also use convolutions
- The receptive field of one step is set by the kernels. The hidden state carries information further over several steps
- **ConvLSTM**: precipitation nowcasting from radar images (Shi et al., NeurIPS 2015)
- ConvGRU for today's notebook: 3 x 3 convolution encoder, ConvGRU cell with 20 hidden channels, two 3 x 3 convolutions as decoder
  - Predicts the standardized change. Receptive field of one step: 9 x 9 cells, the same as SmallCNN

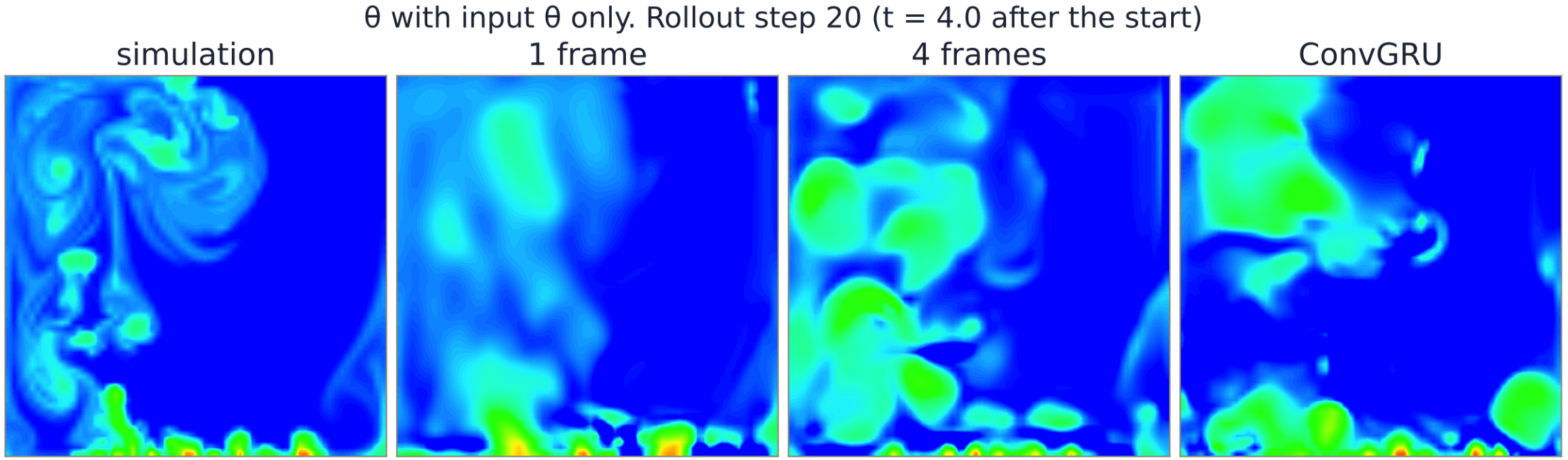


ConvLSTM: hidden and cell states are 3D tensors (Shi et al., Fig. 2)



Top figure: Shi, Chen, Wang, Yeung, Wong & Woo, Convolutional LSTM Network, NeurIPS 2015,. Bottom: ConvGRU on BubbleML (next slide)

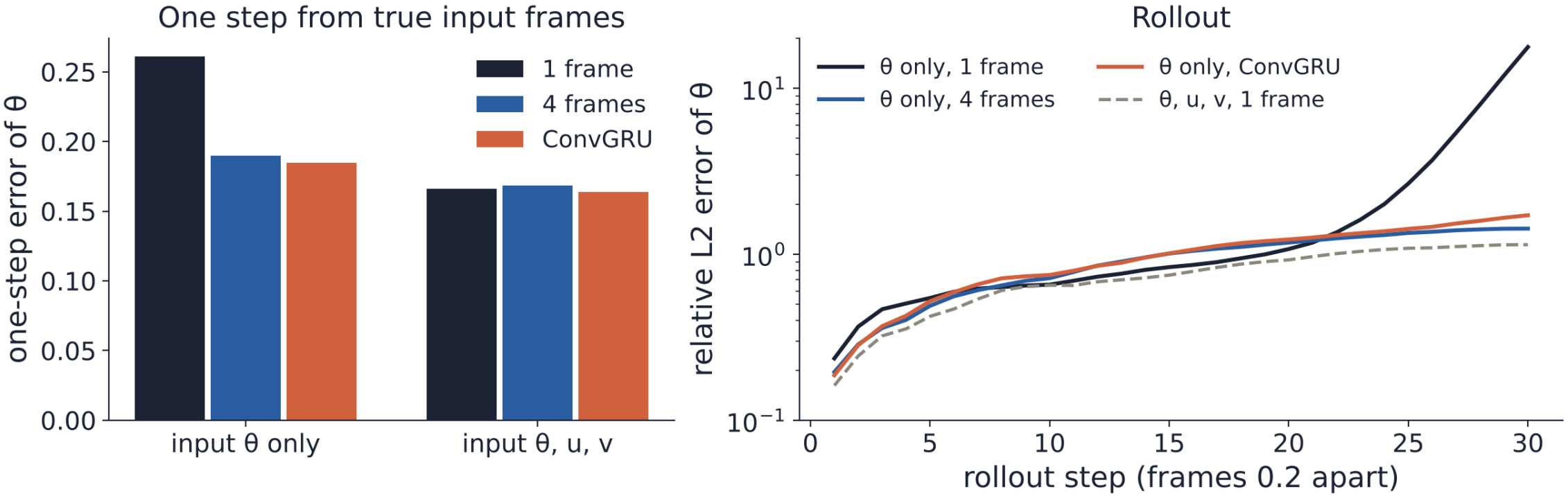
Recurrent networks on BubbleML



- Input θ only: the model has to infer the velocity from the sequence of temperature frames
- All three models start from the same 4 true test frames, then use their own predictions
- Within a few steps all three lose the thin plumes and later become unstable

BubbleML_2, T_{wall} = 106 ° C, averaged to 64 × 64, frames 0.2 apart. Train: first 600 frames; test: last 200. MSE on the standardized change

Rollout error with and without velocity input



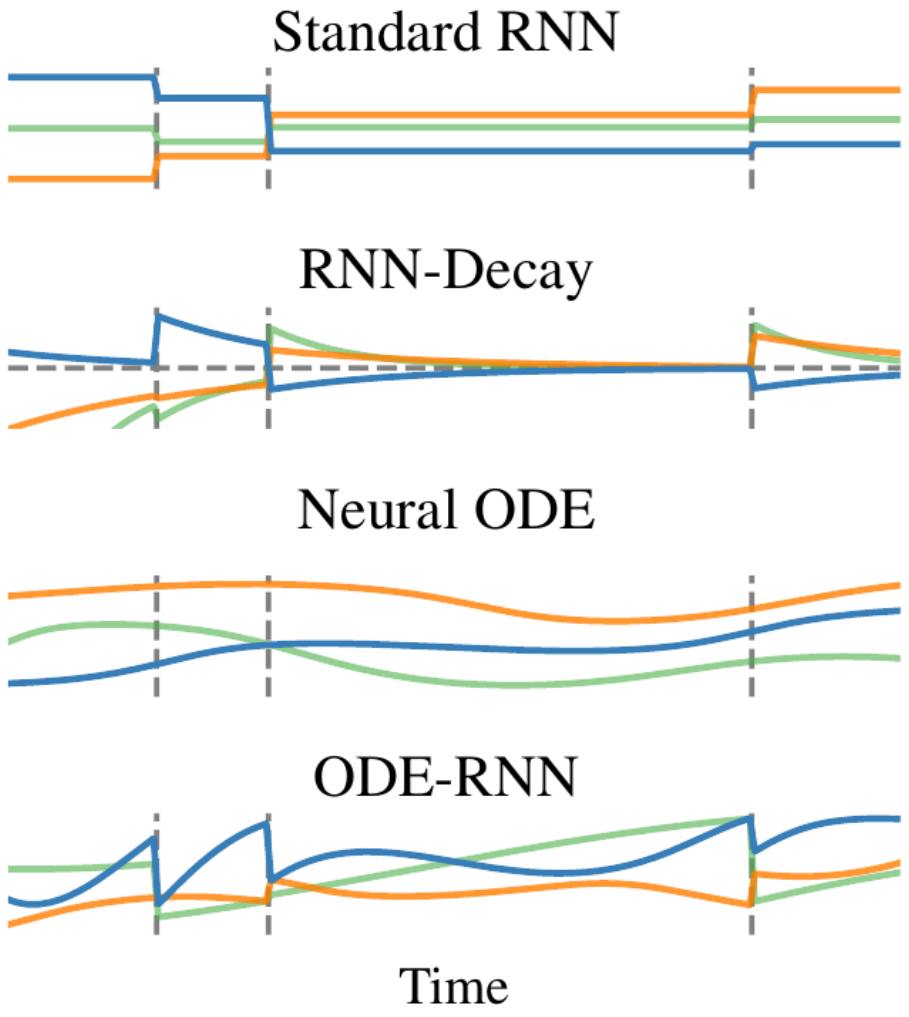
- Input θ only: past frames lower the one-step error. The 4-frame window and the ConvGRU give similar gains
- Input θ, u, v: no gain. In rollout the gain lasts a few steps, then all models drift

Memory helps when the observed state is incomplete

Relative L2 error of θ on the test frames. Same setup as the previous slide, one seed. 4 frames: SmallCNN with the last 4 frames stacked as channels

Models with a fixed time step

- A **discrete model** learns the map over one Δt. It predicts at multiples of Δt only
- Changing Δt means retraining. Lecture 7: Pangu-Weather trains four models, for 1, 3, 6 and 24-hour steps
- BubbleML frames are saved at a fixed interval. A model trained on frames 0.2 apart cannot predict at 0.1
- Data are often sampled irregularly
  - Sensor dropouts, weather stations, clinical measurements, adaptive solver output
- Recurrent networks have the same restriction: one step of the cell is one Δt. Right: a standard RNN holds its state between observations
- **Question**: can we learn the rate of change du/dt = f(u), which does not depend on Δt, and integrate it to any time?



Δt: time between frames. Figure: hidden state of four models between two observations (dashed lines); colors are three hidden dimensions.
Rubanova, Chen & Duvenaud, NeurIPS 2019

Residual networks and forward Euler

$$u_{n+1} = u_n + \Delta t\, f(u_n) \qquad \textit{forward Euler}$$

$$h^{(l+1)} = h^{(l)} + f_\theta\left(h^{(l)}\right) \qquad \textit{residual block}$$

- Recall Forward Euler: one step adds Δt times the rate of change
- A residual block adds $f_\theta(h)$ to its input: forward Euler with Δt = 1, with the layer index l as time
- With L layers and steps of size 1/L, a residual network approaches an ODE in the layer index as L grows (Weinan E 2017; Haber & Ruthotto 2017; Lu et al. 2018)
- **Figure**: a 1D hidden state passes through a residual network (fixed discrete steps) and an ODE network (solver chooses where to evaluate)

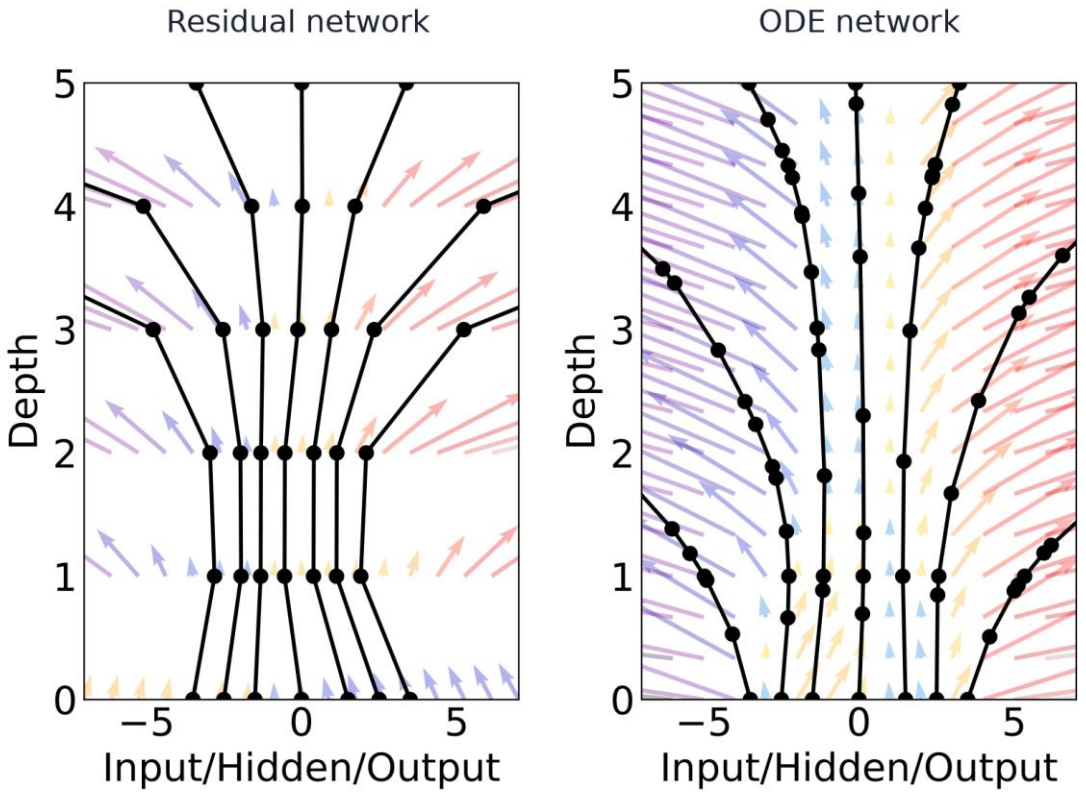


Figure: Chen, Rubanova, Bettencourt & Duvenaud, Neural Ordinary Differential Equations, NeurIPS 2018
Weinan E, Comm. Math. Stat. 2017; Haber & Ruthotto, Inverse Problems 2017; Lu et al., ICML 2018

Neural ordinary differential equations

- $\mathbf{f_\theta}$: a neural network that outputs the rate of change of the state **u**

$$\frac{du}{dt} = f_\theta(u, t)$$

- Prediction at any time t_1: integrate $\mathbf{f_\theta}$ with an ODE solver

$$u(t_1) = u(t_0) + \int_{t_0}^{t_1} f_\theta(u(t), t)\, dt = \mathrm{ODESolve}\big(f_\theta,\ u(t_0),\ t_0,\ t_1\big)$$

- Training: compare the solution with the **data** at the observed times t_i

$$\mathcal{L} = \sum_i \left\| \mathrm{ODESolve}(f_\theta, u(t_0), t_0, t_i) - u_{\mathrm{data}}(t_i) \right\|^2$$

- Observation times can be irregular and can differ between trajectories
- Chen, Rubanova, Bettencourt & Duvenaud, NeurIPS 2018. Libraries: torchdiffeq (PyTorch), Diffrax (JAX)

u(t): state at time t. $\mathbf{f_\theta}$: neural network with parameters θ. $\mathbf{u_{data}(t_i)}$: observed state. ODESolve: a numerical ODE solver, such as Runge–Kutta.
Chen et al., NeurIPS 2018

Gradients through the ODE solver

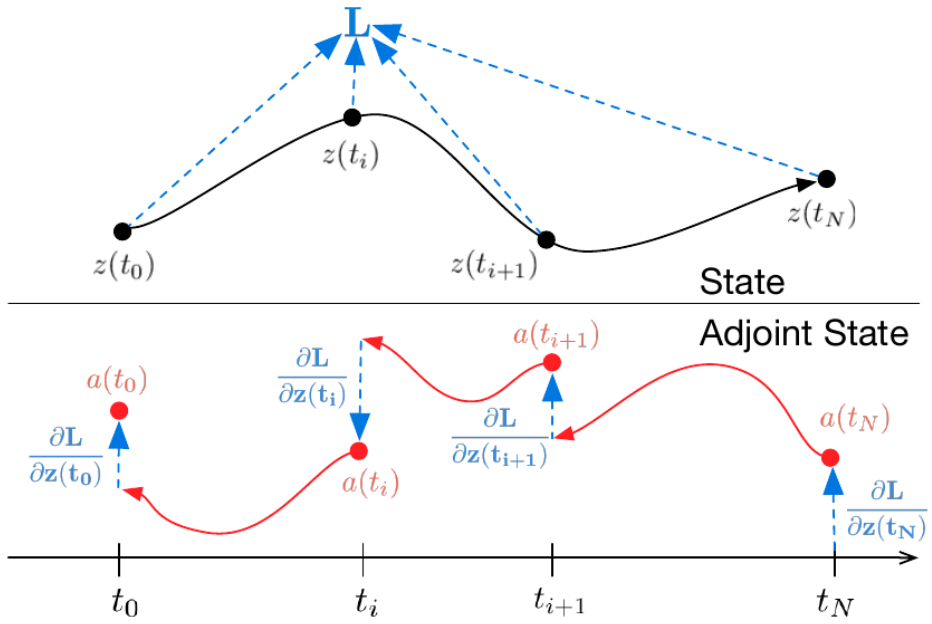
- Option 1: **backpropagate through the solver** operations
  - Exact gradient of the computed solution
  - Memory grows with the number of solver steps, as in unrolled training

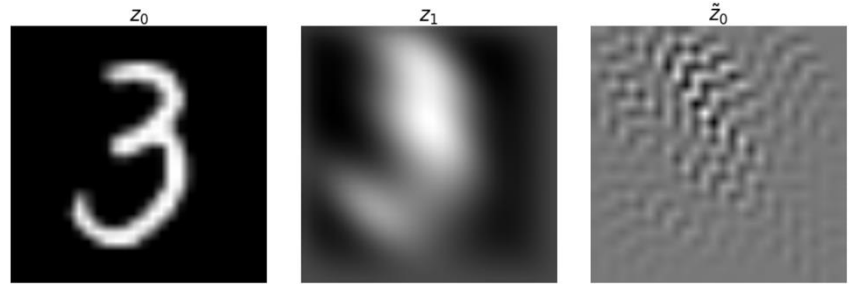
- Option 2: **adjoint method**: solve an ODE for a(t), the gradient of the loss with respect to **u(t)**, backward in time

$$a(t) = \frac{\partial \mathcal{L}}{\partial u(t)}, \qquad \frac{da}{dt} = -a(t)^\top \frac{\partial f_\theta}{\partial u}, \qquad \frac{d\mathcal{L}}{d\theta} = \int_{t_0}^{t_1} a(t)^\top \frac{\partial f_\theta}{\partial \theta}\, dt$$

  - Memory independent of the number of solver steps
  - Recomputes **u(t)** backward in time. The gradient can be **inaccurate** if that backward solve is unstable (Gholami et al. 2019)

- In practice: backpropagate through the solver when the memory is available; use the adjoint for long integrations

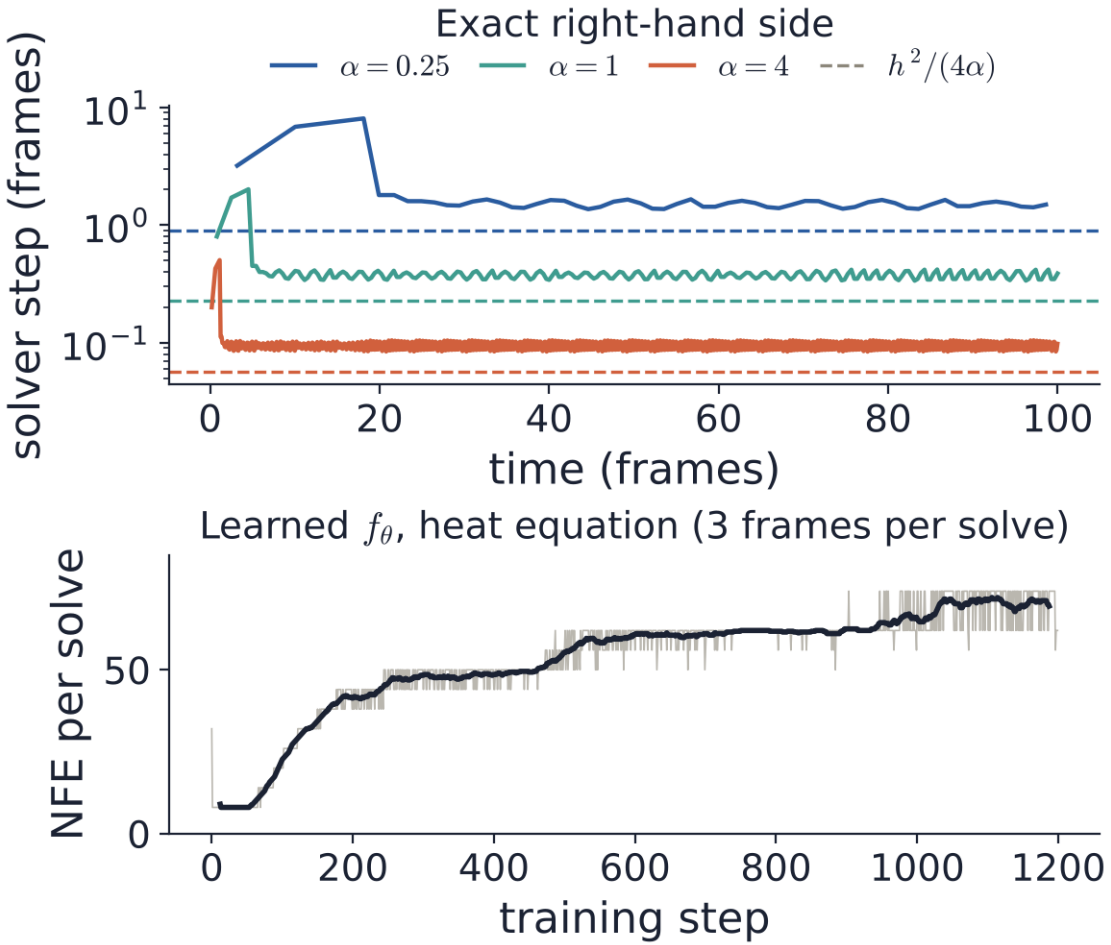




Top figure: Chen et al., NeurIPS 2018, (a(t): adjoint, z: state, L: loss)
Bottom figure: Gholami, Keutzer & Biros, ANODE, IJCAI 2019, (input z_0, output z_1, and z_0 recovered by solving backward)
Pontryagin et al. 1962

ODE solvers and their cost

- Adaptive solvers (Runge-Kutta 4(5), Dormand-Prince) choose the step to keep the estimated local error below a tolerance
- **Cost**: number of function evaluations (NFE), each a forward pass of f_θ. 6 per step for Dormand-Prince
- **Stiff** systems: an explicit solver's step is limited by stability. Heat equation with forward Euler: $\Delta t \le h^2/(4\alpha)$
- Top figure: the adaptive step settles at a fixed multiple of $h^2/(4\alpha)$ for every α, so NFE grows in proportion to α
- Bottom figure: Training can make f_θ harder to integrate, and NFE grows. Regularizers penalize it (Finlay et al. 2020; Kelly et al. 2020)



h: grid spacing. α: diffusivity. Top: 2D heat equation on 48 x 48, exact right-hand side, RK45 (rtol 10^{-4}) . Bottom: neural ODE of the next slide, Dormand-Prince

Finlay et al., ICML 2020; Kelly et al., NeurIPS 2020

Neural ODEs for PDE fields

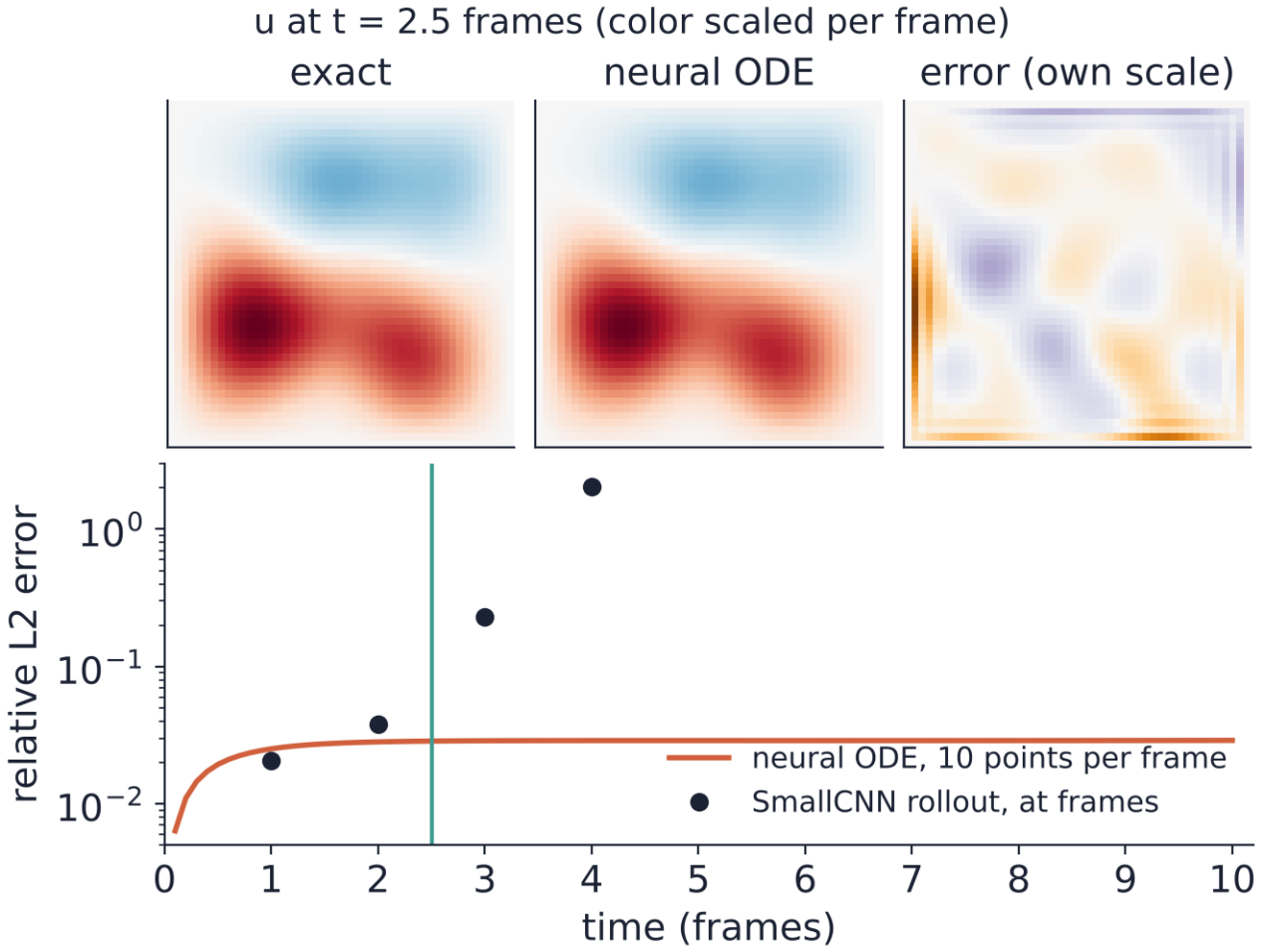
- **Method of lines**: discretize space and keep time continuous. One ODE per grid cell

$$\frac{du_{ij}}{dt} = \alpha \frac{u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1} - 4u_{ij}}{h^2}$$

- Learned version: replace the finite-difference right-hand side with a CNN $\mathbf{f_\theta}$ on the grid

$$\frac{du}{dt} = f_\theta(u)$$

- Lecture 1 heat equation, frames 5 times further apart than in Lecture 1
- Neural ODE evaluated 10 times per frame interval. SmallCNN (Lecture 1): at frames only
- The neural ODE gives the state at any time, and its error stays bounded over the 10 frames
- The SmallCNN at this larger step is accurate for the first frames, then its rollout becomes **unstable**



u_{ij}: temperature at grid cell (i, j). h: grid spacing. $\partial u/\partial t = \alpha\nabla^2 u$ on $[0, 1]^2$, u = 0 on the boundary, α = 1, 6 × 6 sine modes, 48 × 48 grid, frames 2.5 × 10^{-3} apart.
f_θ: three 3 x 3 convolutions, 16 channels, GELU. Trained on segments of 3 frames, Dormand-Prince

Known physics plus a learned term

- **Universal differential equations**: keep the terms we know and learn the rest (Rackauckas et al. 2020)

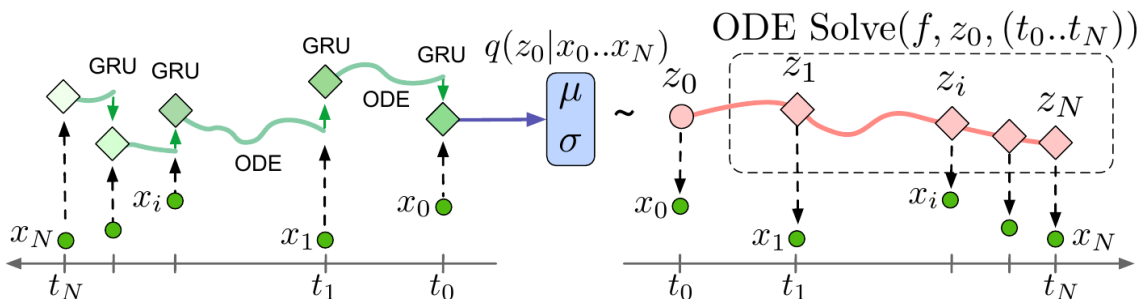
$$\frac{du}{dt} = \underbrace{a\,\nabla_h^2 u}_{\text{known form}} + \underbrace{N_\theta(u)}_{\text{learned}}$$

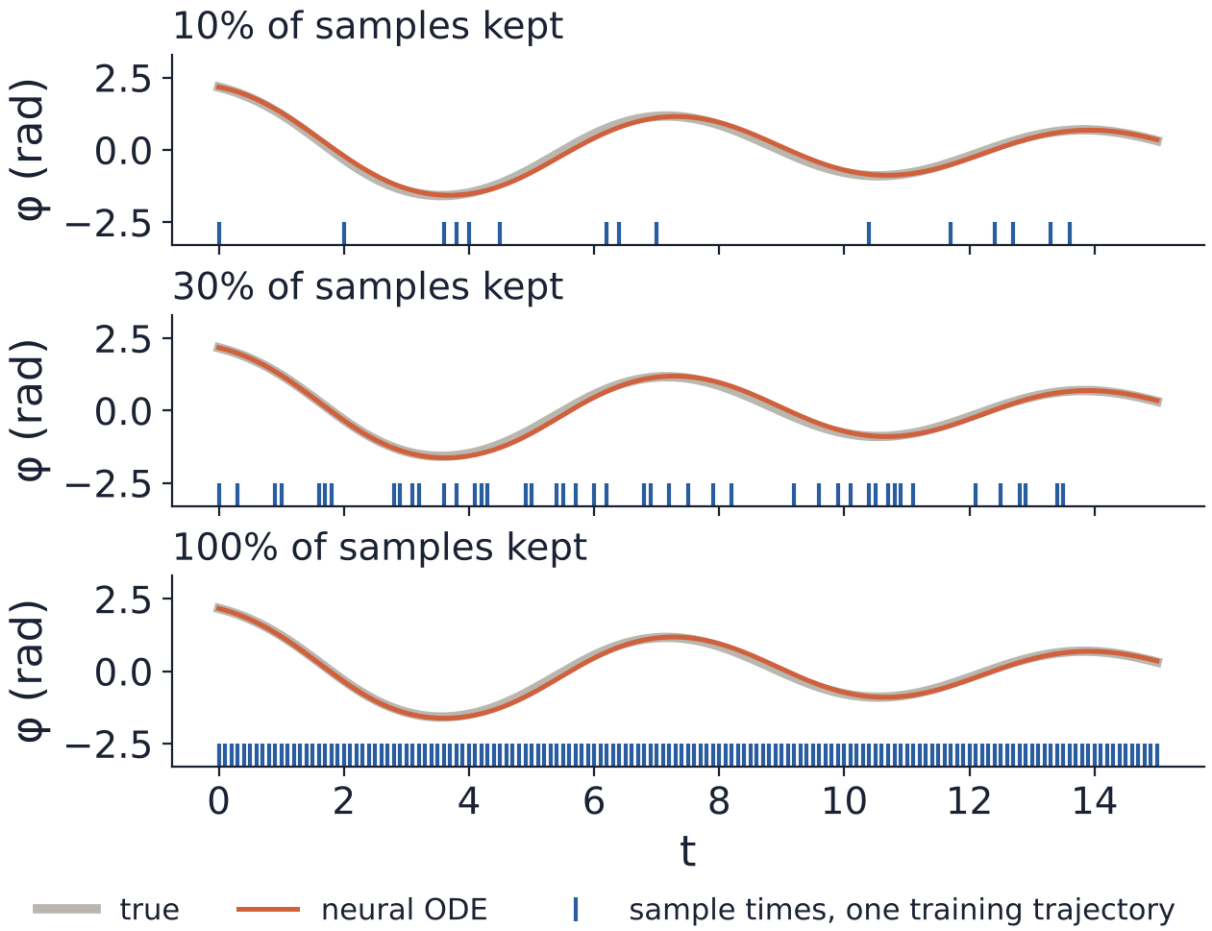
- Example: the finite-difference Laplacian ∇_h^2 with a learned coefficient **a**, plus a network $\mathbf{N_\theta}$
- The ODE solver integrates both. Gradients flow through the known terms into **a** and $\mathbf{N_\theta}$
- The known term brings its stiffness with it: the solver cost is that of the discretized PDE
- Other uses: closure terms in turbulence and climate models, unknown reaction rates
- Lecture 14: physics-constrained and hybrid learning

∇_h^2: 5-point finite-difference Laplacian. **a**, $\mathbf{N_\theta}$: learned coefficient and neural network. Rackauckas et al., Universal Differential Equations for Scientific Machine Learning, arXiv 2001.04385 (2020)

Irregularly sampled data

- Damped pendulum: neural ODE trained on 10%, 30% or 100% of the samples, at random times that differ between trajectories
- Right: one test trajectory. With fewer samples the prediction drifts more over time; it still follows the trajectory
- **Latent ODE** (below): an ODE-RNN encoder reads the data backward to give z_0; an ODE evolves z (Rubanova et al. 2019)
- Neural CDEs update the state as observations arrive (Kidger et al. 2020)





Pendulum: $d^2\varphi/dt^2 = -\sin\varphi - 0.15\, d\varphi/dt$, 16 training and 8 test trajectories, $t \in [0, 15]$. f_θ: MLP 2 x 64, tanh.
Bottom-left figure: Rubanova, Chen & Duvenaud, NeurIPS 2019
Kidger, Morrill, Foster & Lyons, NeurIPS 2020

Comparison of temporal models

	Frame window	RNN, LSTM, GRU	ConvLSTM, ConvGRU	Neural ODE
Memory	last k frames	hidden state h	hidden state field	none beyond the state; latent ODE for partial data
Time step	fixed Δt	fixed Δt	fixed Δt	any time; irregular data
Training	one-step pairs	BPTT over sequences	BPTT over sequences	backprop through solver or adjoint
Cost per prediction	one forward pass	one cell step	one cell step	NFE forward passes; grows with stiffness
Limitation	fixed reach	sequential; long lags hard	sequential; memory for BPTT	solver cost; needs a complete state
Reference	Lecture 7	Hochreiter & Schmidhuber 1997; Cho et al. 2014	Shi et al. 2015	Chen et al. 2018

Memory handles a partial state. Continuous time handles irregular sampling and a change of Δt

k: window length. BPTT: backpropagation through time. NFE: number of function evaluations

Summary

- If the observation is a partial state, the next frame depends on past frames
- Recurrent networks keep a hidden state; BPTT trains them through a product of Jacobians that can vanish or explode
- LSTM and GRU add gated, additive paths and learn dependencies over longer lags
- ConvLSTM and ConvGRU apply the same cells to fields
- Discrete models are tied to their Δt. A neural ODE learns du/dt and can be evaluated at any time
- Neural ODEs are trained through the solver or with the adjoint. Their cost grows with stiffness
- Keep the known physics terms in the ODE and learn only the missing terms with a network

After class

- Chen, Rubanova, Bettencourt & Duvenaud, Neural Ordinary Differential Equations, NeurIPS 2018
- Kidger, On Neural Differential Equations, PhD thesis, Oxford 2022, Chapters 1-3
- Rubanova, Chen & Duvenaud, Latent ODEs for Irregularly-Sampled Time Series, NeurIPS 2019
- Olah, Understanding LSTM Networks, 2015
- Goodfellow, Bengio & Courville, Deep Learning, Chapter 10 (recurrent networks, LSTM, GRU)

- **Project abstract due Thursday, Oct 22**: one page per team, problem, data, baselines, metrics, plan and GPU estimate

Next, Lecture 9 (Thu, Oct 22): Attention and transformers

NEXT: HANDS-ON

From concepts to code

→ Open the Lecture 8 notebook

1. BubbleML with temperature only: train a 4-frame SmallCNN and a ConvGRU; compare rollout error and prediction

2. Neural ODE on the heat equation: evaluate between the training frames; count function evaluations as the diffusivity grows