AI4Science: Physics-Aware Deep Learning for Modeling Dynamical Systems

Patrick Gallinari,

patrick.gallinari@sorbonne-universite.fr,
https://pages.isir.upmc.fr/gallinari/

Outline

- Context: AI4science
- Background: Neural networks and ordinary differential equations
  - NNs as numerical schemes for solving ODEs
  - Neural ODEs
- Modeling Spatio-temporal dynamics with Neural Networks
  - NNs as surrogate models for solving PDEs - Data-driven approaches
    - Discrete space models
    - Continuous space models
  - NNs as surrogate models for solving PDEs – Data free approaches
- Hybrid models
  - Incorporating physical knowledge in dynamics models
- Generalization in ML models for dynamics modeling

Context: AI4Science

Context: AI for Science

- AI successes mainly concern the numerical world and GAFAs/BATs applications
  - Semantic data: e.g. vision, language, and virtual worlds, e.g. games
  - Availability of massive data collected on the internet (vision, NLP) or generated (games)

- What about applications of AI in the real world for Science and Engineering?
  - The dominant paradigm is still the classical physics based one
    - Classical science and engineering rely on a scientific paradigm involving a deep understanding of the laws of nature in physics, biology, etc
  - With the availability of large amounts of data AI4Science emerged as a new paradigm:
    - How to make use of AI in scientific domains?
    - Is it possible to leverage the classical scientific paradigm together with the more recent paradigm of data science?

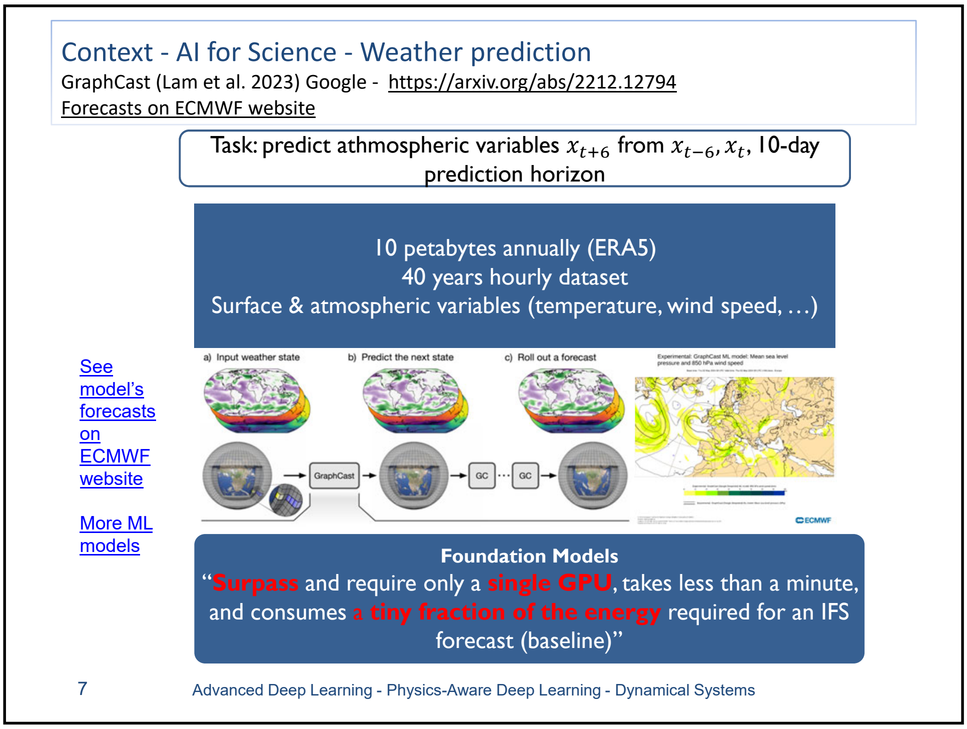
Context - AI for Science – ML Challenges - examples

- In red challenges addressed in this presentation
  - Data based approaches to emulate physical phenomena
  - Integrating scientific knowledge in ML algorithms
  - Discover new scientific knowledge
  - Physical plausibility/ interpretability of the solutions provided by ML
  - Robustness guaranties/ uncertainty
  - Generalization: how can agnostic methods be biased to generalize to different conditions/ environments

Context - AI for Science - Weather forecasting and climate

2022-2024 – Foundation Models for weather prediction (ERA5 dataset 40 years hourly reanalysis data)

- GraphCast – Google & DeepMind 2022
  https://arxiv.org/abs/2212.12794
  https://deepmind.google/discover/blog/graphcast-ai-model-for-faster-and-more-accurate-global-weather-forecasting/#:~:text=Deep%20learning%20offers%20a%20different,the%20present%20into%20the%20future.
  There is an online demo of weather prediction
- ClimaX – Msoft & UCLA 2023
  https://arxiv.org/abs/2301.10343
- Pangu-Weather – Huawei 2023
  http://arxiv.org/abs/2211.02556
- FourCastNet – NVIDIA&Lawrence Berkeley lab.&al.
  http://arxiv.org/abs/2202.11214
- Neural General Circulation Model – Google 2023
  https://arxiv.org/abs/2311.07222
- Aurora – Microsoft 2024
  - https://arxiv.org/abs/2405.13063



Context - AI for Science - Weather prediction

GraphCast (Lam et al. 2023) Google - https://arxiv.org/abs/2212.12794

Forecasts on ECMWF website

Task: predict athmospheric variables x_{t+6} from x_{t-6}, x_t, 10-day prediction horizon

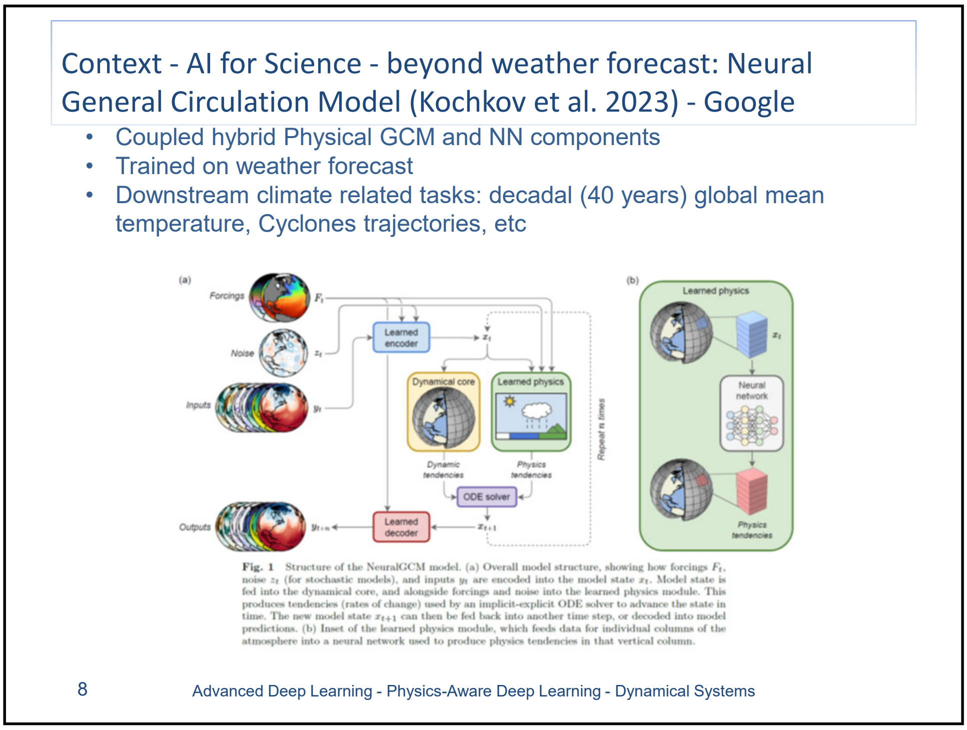
10 petabytes annually (ERA5)
40 years hourly dataset
Surface & atmospheric variables (temperature, wind speed, …)

See model's forecasts on ECMWF website

More ML models

a) Input weather state
b) Predict the next state
c) Roll out a forecast
GraphCast
GC
GC
Experimental: GraphCast ML model: Mean sea level pressure and 850 hPa wind speed
ECMWF

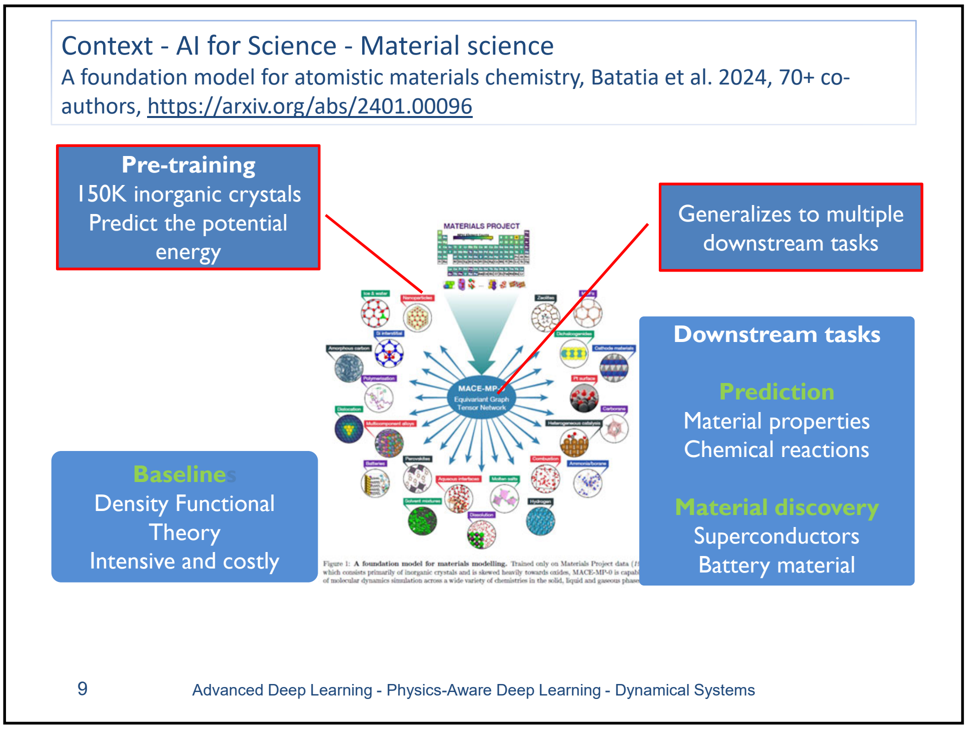
Foundation Models
"**Surpass** and require only a **single GPU**, takes less than a minute, and consumes **a tiny fraction of the energy** required for an IFS forecast (baseline)"



Context - AI for Science - beyond weather forecast: Neural General Circulation Model (Kochkov et al. 2023) - Google

- Coupled hybrid Physical GCM and NN components
- Trained on weather forecast
- Downstream climate related tasks: decadal (40 years) global mean temperature, Cyclones trajectories, etc

Fig. 1 Structure of the NeuralGCM model. (a) Overall model structure, showing how forcings F_t, noise z_t (for stochastic models), and inputs y_t are encoded into the model state x_t. Model state is fed into the dynamical core, and alongside forcings and noise into the learned physics module. This produces tendencies (rates of change) used by an implicit-explicit ODE solver to advance the state in time. The new model state x_{t+1} can then be fed back into another time step, or decoded into model predictions. (b) Inset of the learned physics module, which feeds data for individual columns of the atmosphere into a neural network used to produce physics tendencies in that vertical column.



Context - AI for Science - Material science

A foundation model for atomistic materials chemistry, Batatia et al. 2024, 70+ co-authors, https://arxiv.org/abs/2401.00096

Pre-training
150K inorganic crystals
Predict the potential energy

Generalizes to multiple downstream tasks

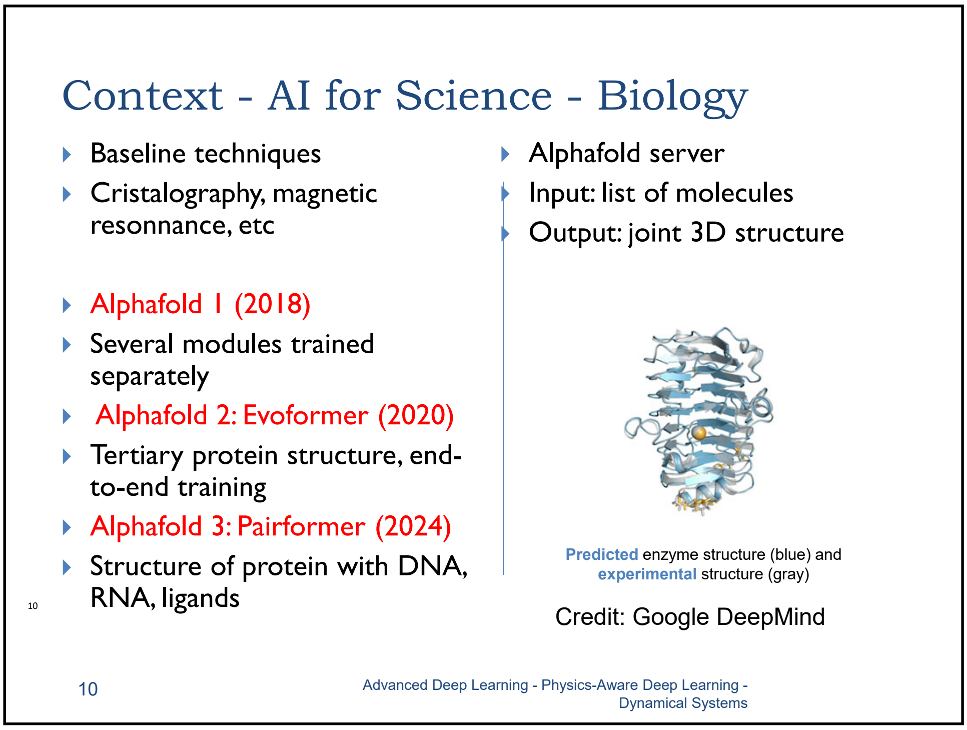
Downstream tasks

Prediction
Material properties
Chemical reactions

Material discovery
Superconductors
Battery material

Baselines
Density Functional Theory
Intensive and costly

Figure 1: **A foundation model for materials modelling.** Trained only on Materials Project data (*M*
which consists primarily of inorganic crystals and is skewed heavily towards oxides, MACE-MP-0 is capab
of molecular dynamics simulation across a wide variety of chemistries in the solid, liquid and gaseous phase



Context - AI for Science - Biology

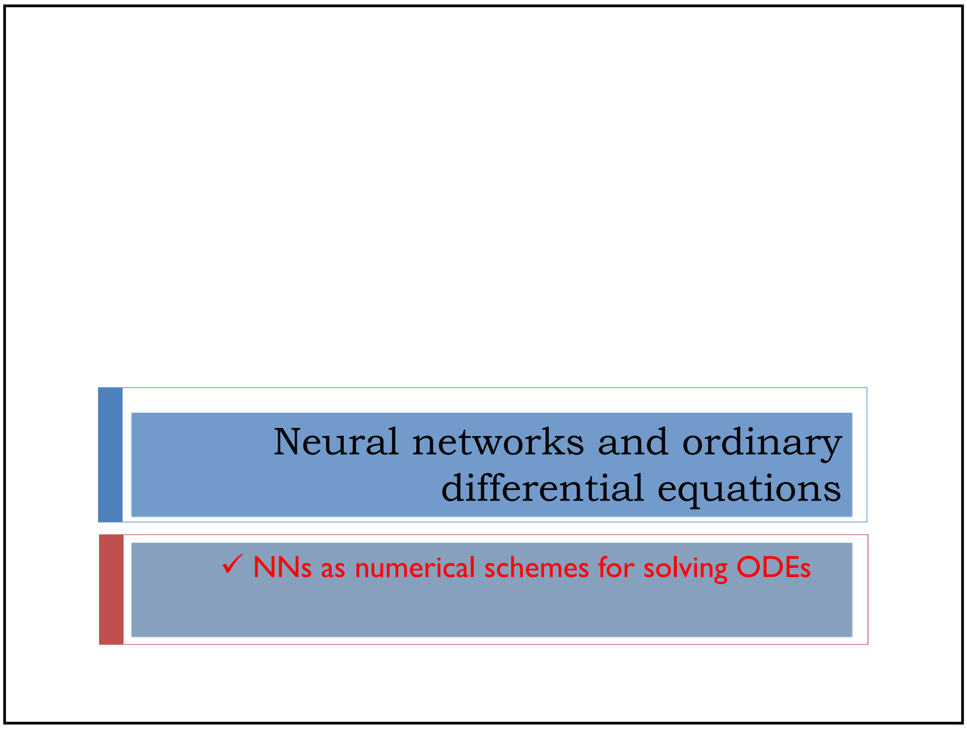
- Baseline techniques
- Cristalography, magnetic resonnance, etc

- Alphafold 1 (2018)
- Several modules trained separately
- Alphafold 2: Evoformer (2020)
- Tertiary protein structure, end-to-end training
- Alphafold 3: Pairformer (2024)
- Structure of protein with DNA, RNA, ligands

- Alphafold server
- Input: list of molecules
- Output: joint 3D structure

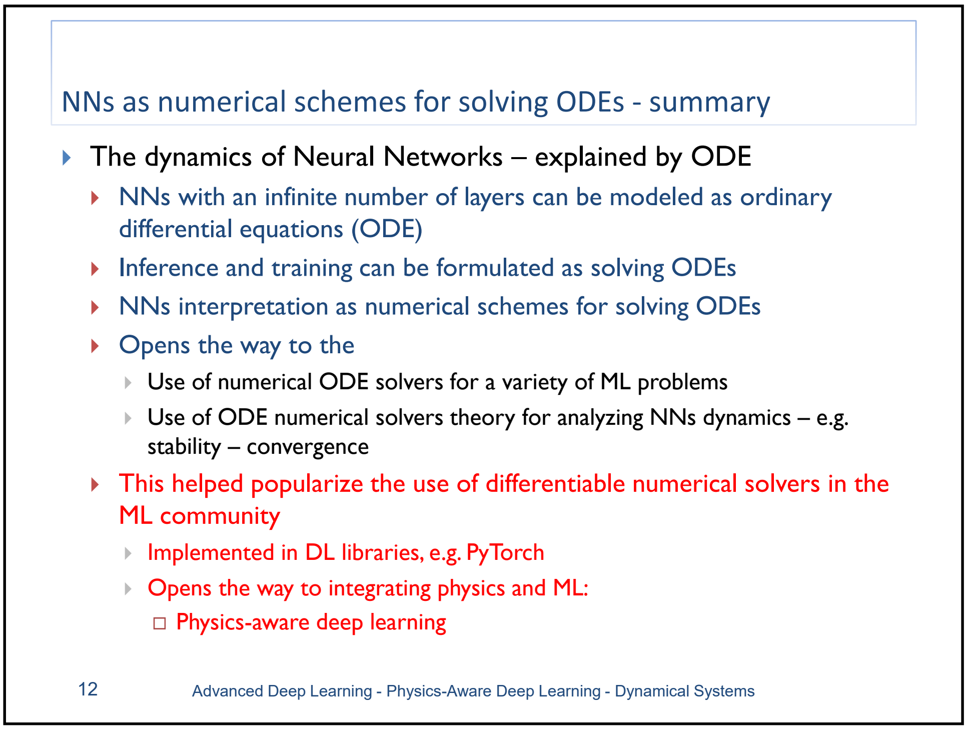
Predicted enzyme structure (blue) and experimental structure (gray)

Credit: Google DeepMind



Neural networks and ordinary differential equations

✓ NNs as numerical schemes for solving ODEs

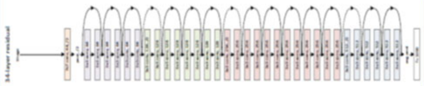


NNs as numerical schemes for solving ODEs - summary

- The dynamics of Neural Networks – explained by ODE
  - NNs with an infinite number of layers can be modeled as ordinary differential equations (ODE)
  - Inference and training can be formulated as solving ODEs
  - NNs interpretation as numerical schemes for solving ODEs
  - Opens the way to the
    - Use of numerical ODE solvers for a variety of ML problems
    - Use of ODE numerical solvers theory for analyzing NNs dynamics – e.g. stability – convergence
  - This helped popularize the use of differentiable numerical solvers in the ML community
    - Implemented in DL libraries, e.g. PyTorch
    - Opens the way to integrating physics and ML:
      - □ Physics-aware deep learning

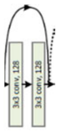
NNs as numerical schemes for solving ODEs

- Several NNs use skip connections, e.g. ResNet



Resnet Module

x_t



$x_{t+1} = x_t + f(x_t, \theta_t)$

Input x is progressively modified by a residual $f(x, \theta)$

- ODE for initial value problem
  - $\frac{dx}{dt} = f(x(t); \theta(t))$ for $t \in [0, T]$, $x(0) = x_0$
    - What is the value of $x(T)$?
- Equivalent integral formulation
  - $x(T) = x(0) + \int_O^T f\left(t, x(t)\right) dt$
  - $\int_O^T f\left(t, x(t)\right) dt$ is approximated via numerical integration
  - Exemple: Euler numerical scheme
    - $x_{t+1} = x_t + hf(x_t, \theta_t), x(0) = x_0$

Forward pass of ResNet is similar to Euler scheme for solving IVP (E 2017, Haber 2017, Chang 2018, Lu 2018, …)

NNs as numerical schemes for solving ODEs – Learning problem

- Learning problem with ResNets
  - $Min_\theta \quad L(F(x, \theta), y)$
    $s.t. \quad x_l = x_{l-1} + f_l(x_{l-1}), l = 1 \dots T, x_0 = x$

The constraint describes the Forward graph of the Resnet

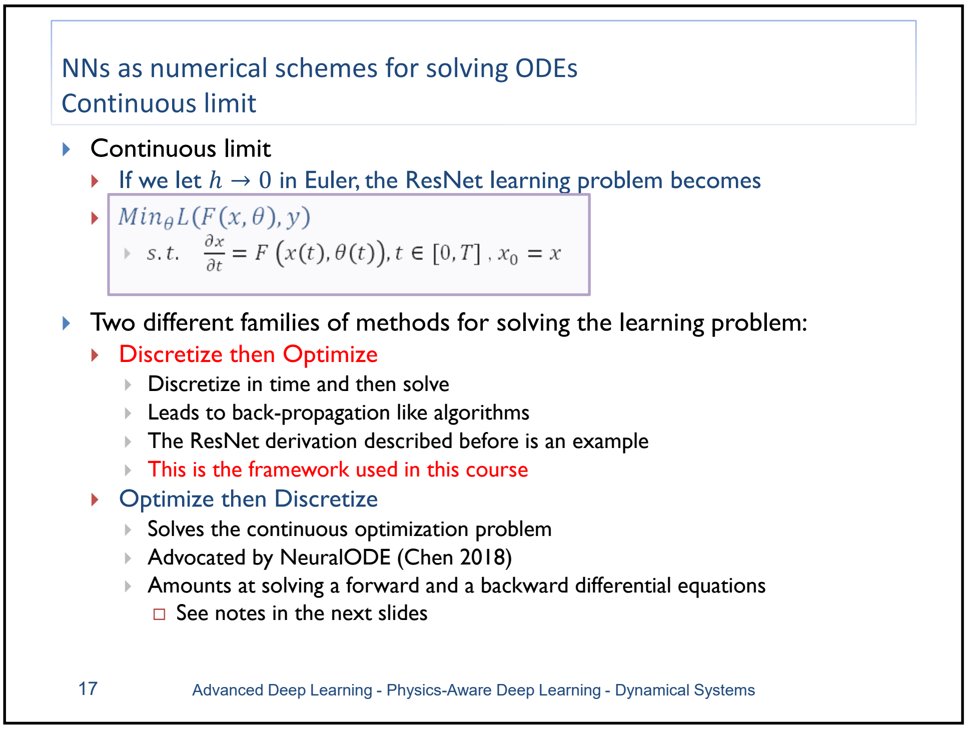
  - x input, y target, θ parameters, x_l layer l activation, T layers
  - Solving this problem requires alternating
    - Forward pass – Euler numerical scheme for solving
      - □ $\frac{dx}{dt} = f(x(t), \theta(t))$ for $t \in [0, T], x(0) = x_0$
    - Backward pass – differentiation through Euler scheme for solving
      - □ $\frac{d\theta}{dt} = -\epsilon \frac{\partial L(\theta(t))}{\partial \theta}, \theta(0) = \theta_0$
- Could this idea be generalized?
  - Replace Euler with any numerical integration scheme

NNs as numerical schemes for solving ODEs
Euler derivation

- ODE – IVP problem
  - $\frac{dx}{dt} = f(x(t))$ for $t \in [0, T], x(0) = x_0$
- Continuous to discrete time
  - Divide $[0, T]$ in intervals of size Δt: $t_n = n\Delta t$
  - The objective is to find x_n an approximation of $x(t_n)$ at each t_n
- Taylor expansion
  - $x(t_{n+1}) \approx \mathrm{x}(\mathrm{t_n}) + \Delta t \frac{dx(t_n)}{dt}$
  - $x(t_{n+1}) \approx \mathrm{x}(\mathrm{t_n}) + \Delta t f(x(t_n))$
- Discrete approximation and algorithm
  - $x_0 = x(0)$
  - $x_{n+1} \approx x_n + \Delta t f(x_n)$

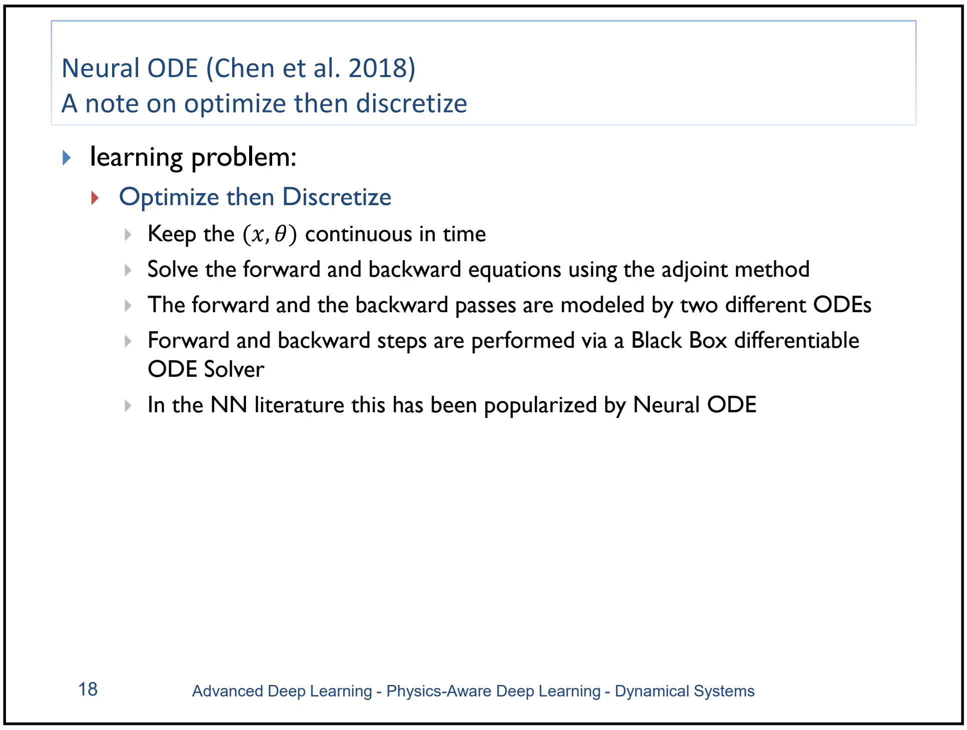
NNs as numerical schemes for solving ODEs
ODE formulation of agradient algorithm

- Steepest gradient descent
  - $\theta_{t+1} = \theta_t - \epsilon_t \nabla L(\theta_t)$, with initial value θ_0
- Continuous formulation
  - Let $\epsilon_t = \epsilon(t)\Delta \mathrm{t}$
  - $\theta(t+1) = \theta(t) - \epsilon(t)\Delta \mathrm{t} \nabla L(\theta(t))$
  - $\frac{\theta(t+1) - \theta(t)}{\Delta \mathrm{t}} = -\epsilon(t)\Delta \mathrm{t} \nabla L(\theta(t))$
  - $\frac{\partial \theta(t)}{\partial t} = -\epsilon(\mathrm{t}) \nabla L(\theta(t))$
- ODE IVP
  - $\frac{\partial \theta(t)}{\partial t} = -\epsilon(\mathrm{t}) \nabla L(\theta(t))$with $\theta(0) = \theta_0$



NNs as numerical schemes for solving ODEs
Continuous limit

- Continuous limit
  - If we let $h \to 0$ in Euler, the ResNet learning problem becomes
  - $Min_\theta L(F(x,\theta),y)$
    - $s.t. \quad \frac{\partial x}{\partial t} = F\left(x(t),\theta(t)\right), t \in [0,T]\,, x_0 = x$
- Two different families of methods for solving the learning problem:
  - Discretize then Optimize
    - Discretize in time and then solve
    - Leads to back-propagation like algorithms
    - The ResNet derivation described before is an example
    - This is the framework used in this course
  - Optimize then Discretize
    - Solves the continuous optimization problem
    - Advocated by NeuralODE (Chen 2018)
    - Amounts at solving a forward and a backward differential equations
      - □ See notes in the next slides



Neural ODE (Chen et al. 2018)
A note on optimize then discretize

- learning problem:
  - Optimize then Discretize
    - Keep the (x,θ) continuous in time
    - Solve the forward and backward equations using the adjoint method
    - The forward and the backward passes are modeled by two different ODEs
    - Forward and backward steps are performed via a Black Box differentiable ODE Solver
    - In the NN literature this has been popularized by Neural ODE

Neural ODE
A note on optimize then discretize

- Discretize – optimize (DTO) vs optimize-discretize (OTD)
  - OTD opens the way to the use of adaptive solvers and is less demanding on memory usage – no need to store the state of the system during the forward pass, they are recomputed during the backward pass (for the backward equation)
- In practice, discretize – optimize (DTO) should be prefered to optimize-discretize (OTD)
  - The gradient of the backward pass correspond to the function computed during the forward pass (not the case for OTD)
  - Well known in numerical analysis
  - By default auto-differentiation performs DTO
    - Note: DTO vs ODT is discussed in Gholami et al. 2020, Onken et al. 2020

Neural ODE
A note on optimize then discretize

- Forward pass
  - This amounts to solve
    - $\frac{\partial x}{\partial t} = F\ (x(t), \theta(), \mathrm{t} \in [0, T]$
    - $x(0) = x_0$
  - Solver call

$$x(T) = ODESolve(F(x(t), t, \theta), x(t_0), t_0 = 0, t_1 = T)$$

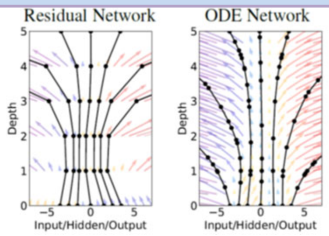


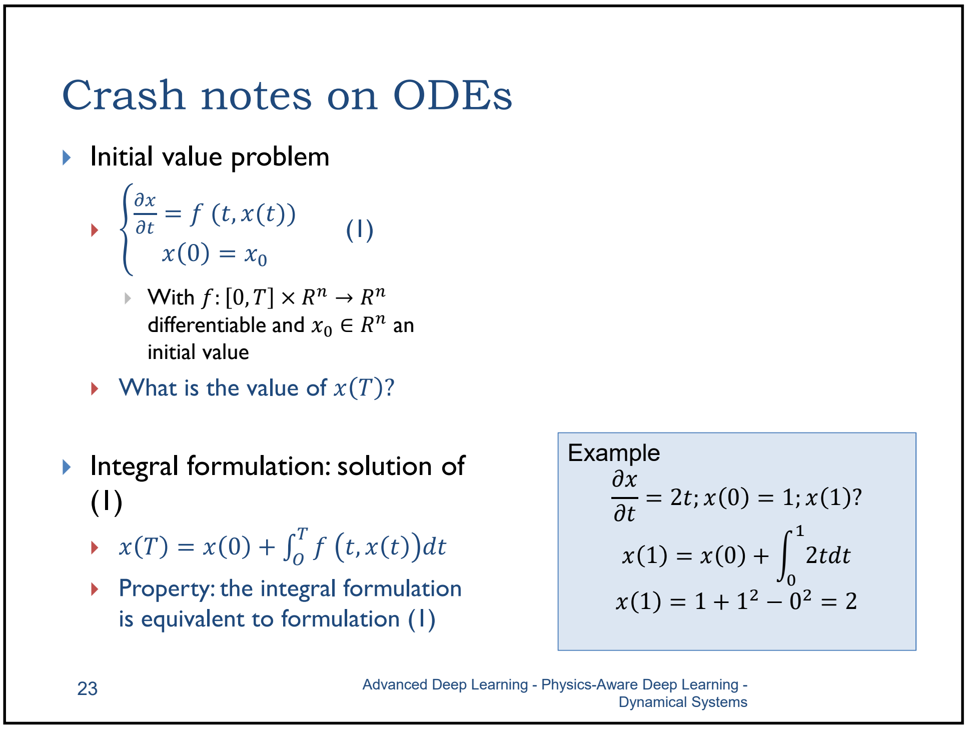
Fig. Chen et al. 2018

NNs as numerical schemes for solving ODEs

- Key ideas
  - Training of NNs can be formulated as solving ODEs with a numerical scheme
  - Different numerical schemes could be used
    - To be implemented with specific NN architectures
  - Allows us using numerical schemes theory for deriving NN properties
  - The link between NNs and differential equations will be most relevant for modeling dynamical systems

  - Note
    - ODE are central in several ML contexts involving dynamical processes such as generative models (e.g. diffusion models, flow matching models)

Interlude
Crash notes on ODEs



Crash notes on ODEs

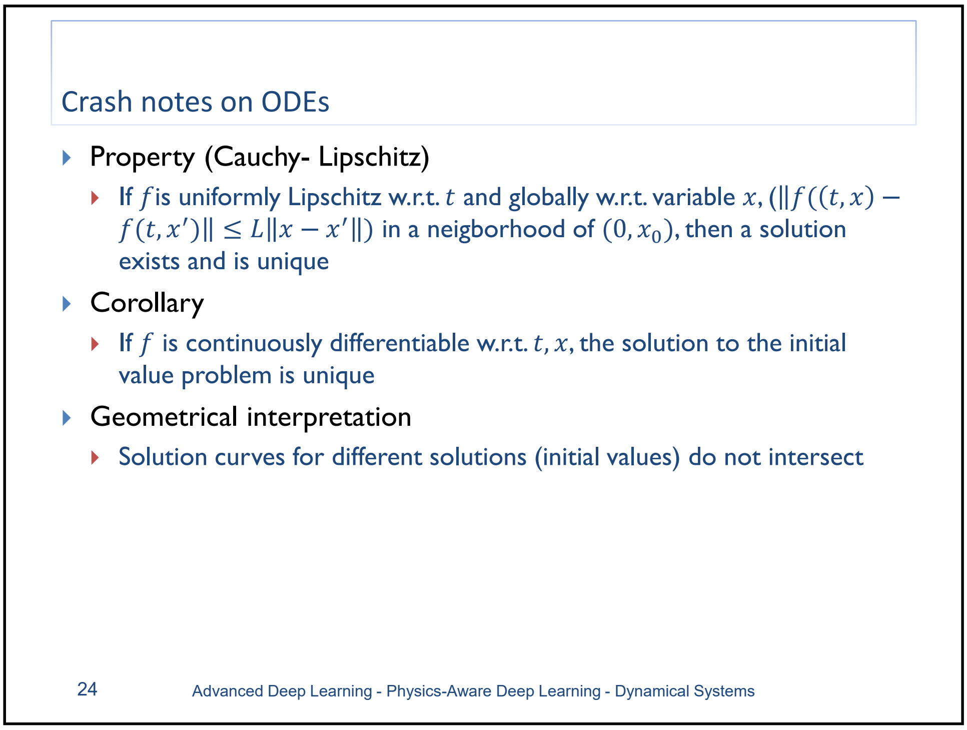
- Initial value problem
  - $$\begin{cases} \frac{\partial x}{\partial t} = f\ (t, x(t)) \\ x(0) = x_0 \end{cases} \quad (1)$$
    - With $f: [0, T] \times R^n \to R^n$ differentiable and $x_0 \in R^n$ an initial value
  - What is the value of $x(T)$?
- Integral formulation: solution of (1)
  - $x(T) = x(0) + \int_0^T f\ \big(t, x(t)\big)dt$
  - Property: the integral formulation is equivalent to formulation (1)

Example

$$\frac{\partial x}{\partial t} = 2t; x(0) = 1; x(1)?$$

$$x(1) = x(0) + \int_0^1 2tdt$$

$$x(1) = 1 + 1^2 - 0^2 = 2$$



Crash notes on ODEs

- Property (Cauchy- Lipschitz)
  - If $f$is uniformly Lipschitz w.r.t. t and globally w.r.t. variable x, ($\|f((t, x) - f(t, x')\| \leq L\|x - x'\|$) in a neigborhood of $(0, x_0)$, then a solution exists and is unique
- Corollary
  - If f is continuously differentiable w.r.t. t, x, the solution to the initial value problem is unique
- Geometrical interpretation
  - Solution curves for different solutions (initial values) do not intersect

Crash notes on ODEs

- Trajectories (solution curves)
- Flow of an ODE
  - $\phi : R \times R^n \rightarrow R^n$ of f is defined by $\phi(t, x_0) = x(t)$
- Geometric Interpretation
  - The flow traces the trajectory of the solution in the state space:
    $$\{\phi_t(x_0): t \in R\}$$
  - These trajectories are solutions of the ODE and follow f the vector field f, satisfying:
    $$\frac{d\phi_t}{dt} = f\big(\phi_t(x_0)\big)$$

i.e. it describes all the trajectories for different initial conditions

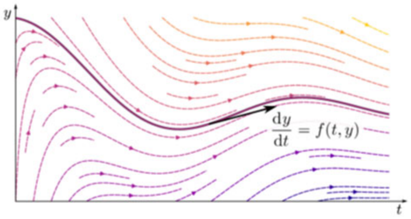
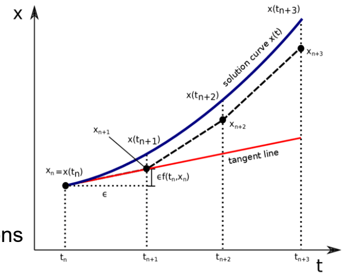


Figure 2.2.: Illustration (in dashed lines) of the continuous flow of an ODE, with a particular solution that is plotted with a solid thicker line. The black arrow represents the tangent to the highlighted solution, which is fully determined by its derivative and initial condition according to variants of the Cauchy-Lipschitz theorem (Demailly, 2006, Chapter V, Section 3.4). In this example, f is defined as $f: (t, y) \mapsto \frac{1}{t}(\cos t - y)$, the ODE admitting as solutions over $I = (0, +\infty)$ functions $y_C: t \mapsto \frac{C}{t} + \operatorname{sinc} t$ for all $C \in \mathbb{R}$.

Crash notes on ODEs

- Numerical solvers
  - $x(T) = x(0) + \boxed{\int_0^T f\,\big(t, x(t)\big)dt}$
    - What if the integral cannot be analytically integrated?
  - $\int_0^T f\,\big(t, x(t)\big)dt$ is approximated via numerical integration
    - Objective: build a sequence of values $x_0, x_1, \dots x_N$ that approximate the solution at the discretization points $x(t_0), x(t_1), \dots, x(t_N)$
- Exemple: Euler forward
  - Step size h
    - $t_{n+1} = t_n + h$
  - Update using the gradient at $f(t_n)$
    - $x_{n+1} = x_n + hf(x_n, \theta_n)$



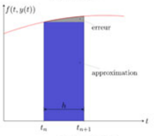
Note: the same solver can be recovered also via the differential formulation through derivative approximations

e.g. $\frac{\partial x}{\partial t} \simeq \frac{x(t+h) - x(t)}{h}$ leads to $x_{n+1} = x_n + hf(t_n, x_n)$

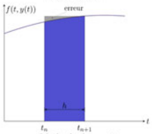
Crash notes on ODEs

- One step methods - exemples
  - $x_{n+1} = x_n + h_n \phi(t_n, x_n, h_n)$ with ϕ a function depending on f
  - Euler forward (explicit)
    - $x_{n+1} = x_n + hf(t_n, x_n)$
  - Euler backward (implicit)
    - $x_{n+1} = x_n + hf(t_{n+1}, x_{n+1})$
    - Requires solving a fixed point equation
  - Runge Kutta e.g. RK2
    - (explixit, RK4 often used as a default option)
  - $\begin{cases} x_{n,1} = x_n \\ x_{n,2} = x_n + hf(t_n, x_{n,1}) \\ x_{n+1} = x_n + \frac{h}{2} f(t_n, x_{n,1}) + \frac{h}{2} f(t_{n+1}, x_{n,2}) \end{cases}$

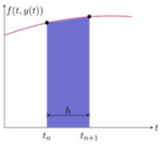
Euler explicite



Euler implicite

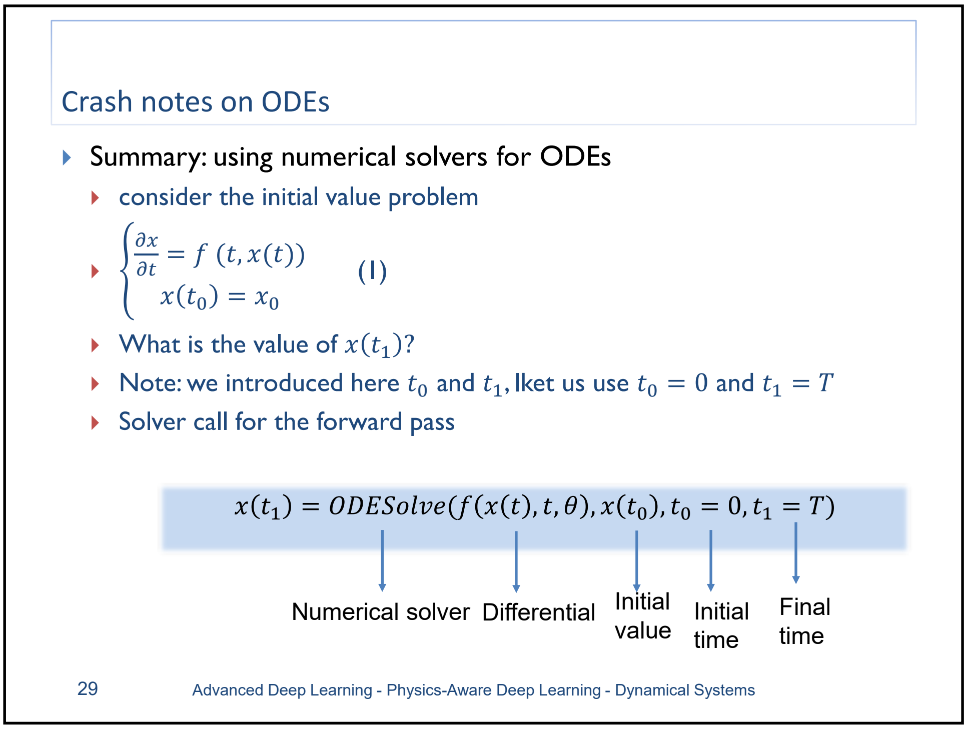


Méthode du trapèze



Crash notes on ODEs
NNs as numerical schemes for solving ODEs

- NN architectures motivated by ODE numerical schemes
  - This link between numerical schemes and NNs has been exploited by some authors
    - Different discretisation methods used in place of Forward Euler
  - Linear multi-step (Lu et al. 2018)
    - $x_{t+1} = (1 - k_t)x_t + k_t x_{t-1} + f(x_t; \theta_t)$, θ_t are the parameters of f
  - Leapfrog Network (Chang et al. 2018)
    - $x_{t+1} = 2x_t - x_{t-1} - h^2 f(x_t, \theta_t)$
  - …
  - Implicit schemes
    - e.g. backward Euler scheme
      - □ $x_{t+1} = x_t + hf(x_{t+1}; \theta_{t+1})$
      - □ Note: requires solving a non linear equation at each step
  - Each numerical scheme leads to a specific NN architecture (a la ResNet)



Crash notes on ODEs

- Summary: using numerical solvers for ODEs
  - consider the initial value problem
  - $\begin{cases} \frac{\partial x}{\partial t} = f\,(t, x(t)) \\ x(t_0) = x_0 \end{cases}$ (1)
  - What is the value of $x(t_1)$?
  - Note: we introduced here t_0 and t_1, lket us use $t_0 = 0$ and $t_1 = T$
  - Solver call for the forward pass

$$x(t_1) = ODESolve(f(x(t), t, \theta), x(t_0), t_0 = 0, t_1 = T)$$

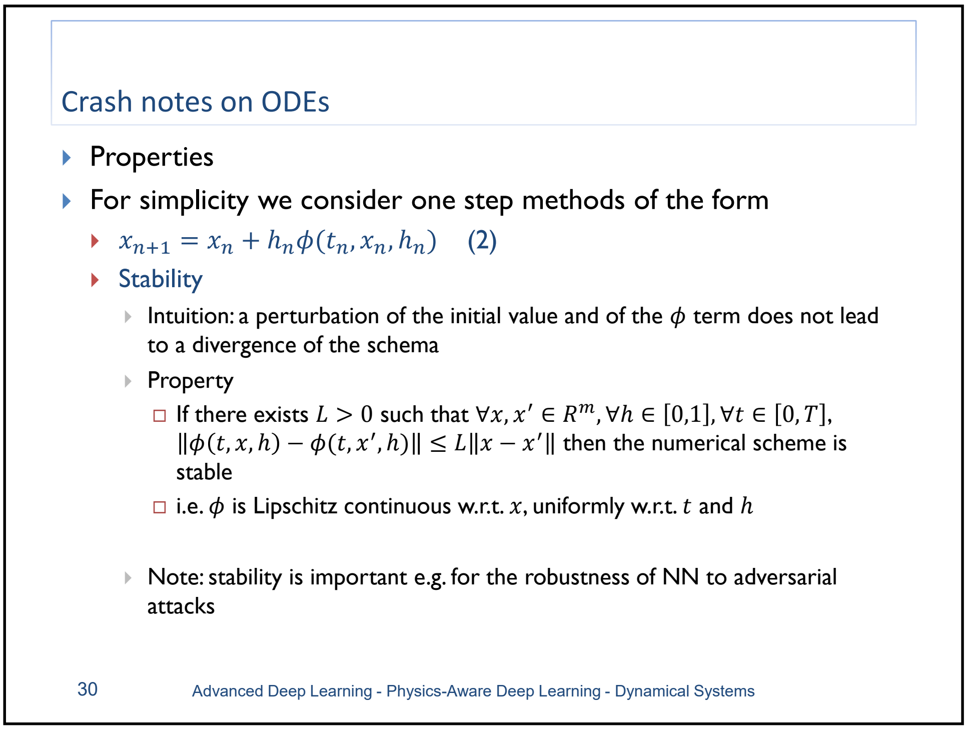
Numerical solver

Differential

Initial value

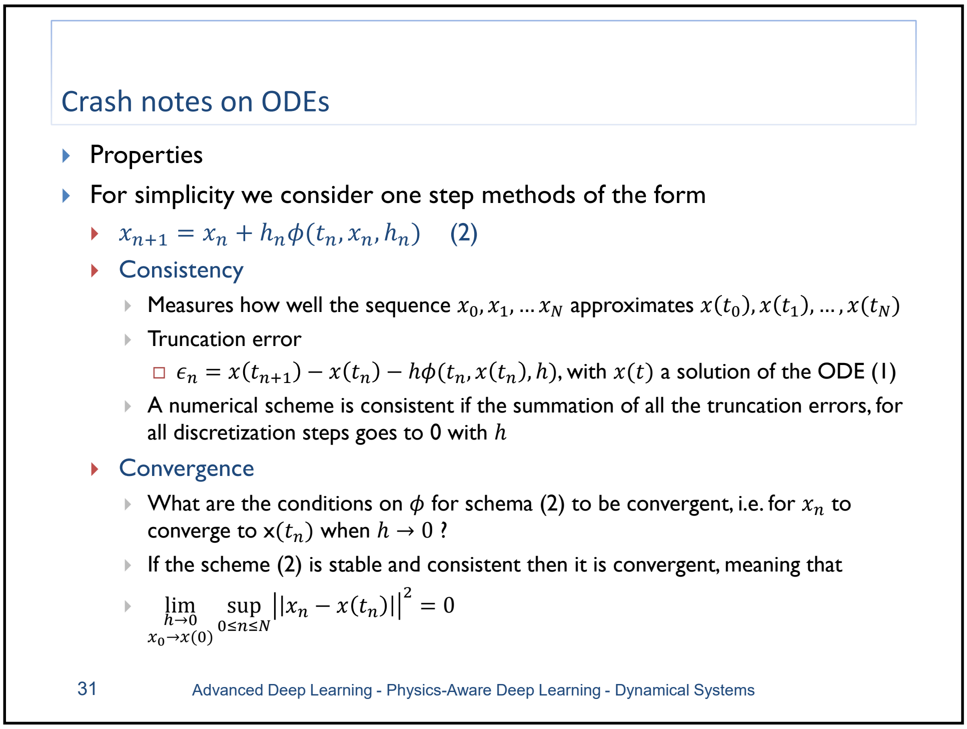
Initial time

Final time



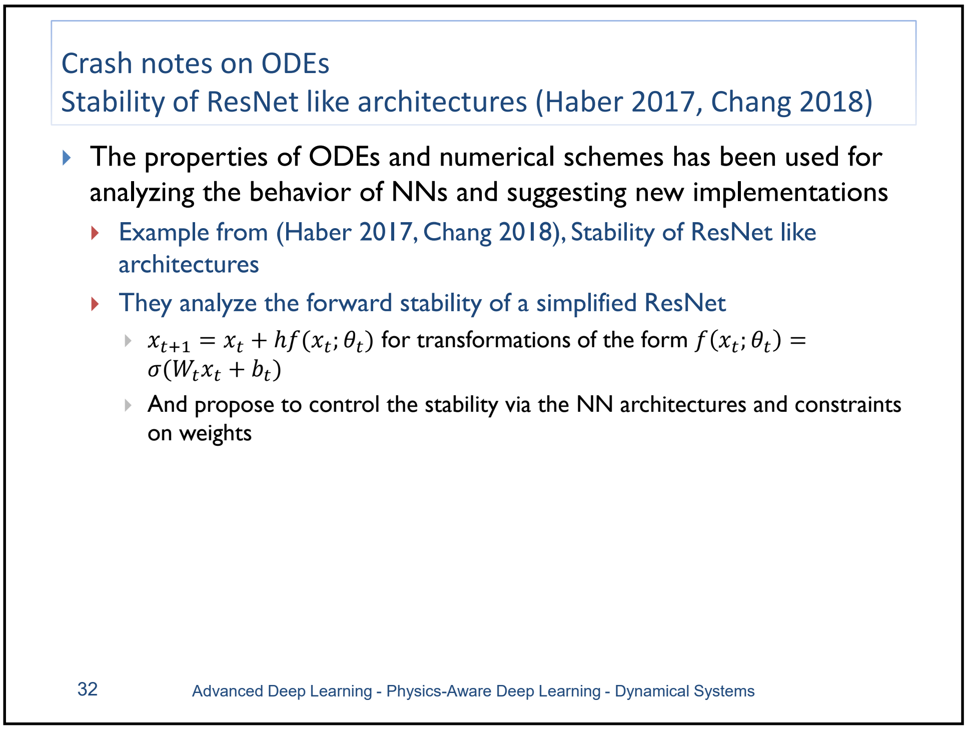
Crash notes on ODEs

- Properties
- For simplicity we consider one step methods of the form
  - $x_{n+1} = x_n + h_n \phi(t_n, x_n, h_n)$ (2)
  - Stability
    - Intuition: a perturbation of the initial value and of the ϕ term does not lead to a divergence of the schema
    - Property
      - □ If there exists $L > 0$ such that $\forall x, x' \in R^m, \forall h \in [0,1], \forall t \in [0, T]$, $\|\phi(t, x, h) - \phi(t, x', h)\| \leq L\|x - x'\|$ then the numerical scheme is stable
      - □ i.e. ϕ is Lipschitz continuous w.r.t. x, uniformly w.r.t. t and h
    - Note: stability is important e.g. for the robustness of NN to adversarial attacks



Crash notes on ODEs

- Properties
- For simplicity we consider one step methods of the form
  - $x_{n+1} = x_n + h_n \phi(t_n, x_n, h_n)$ (2)
  - Consistency
    - Measures how well the sequence $x_0, x_1, \dots x_N$ approximates $x(t_0), x(t_1), \dots, x(t_N)$
    - Truncation error
      - □ $\epsilon_n = x(t_{n+1}) - x(t_n) - h\phi(t_n, x(t_n), h)$, with $x(t)$ a solution of the ODE (1)
    - A numerical scheme is consistent if the summation of all the truncation errors, for all discretization steps goes to 0 with h
  - Convergence
    - What are the conditions on ϕ for schema (2) to be convergent, i.e. for x_n to converge to x(t_n) when $h \to 0$?
    - If the scheme (2) is stable and consistent then it is convergent, meaning that
    - $\lim_{\substack{h \to 0 \\ x_0 \to x(0)}} \sup_{0 \le n \le N} ||x_n - x(t_n)||^2 = 0$



Crash notes on ODEs
Stability of ResNet like architectures (Haber 2017, Chang 2018)

- The properties of ODEs and numerical schemes has been used for analyzing the behavior of NNs and suggesting new implementations
  - Example from (Haber 2017, Chang 2018), Stability of ResNet like architectures
  - They analyze the forward stability of a simplified ResNet
    - $x_{t+1} = x_t + hf(x_t; \theta_t)$ for transformations of the form $f(x_t; \theta_t) = \sigma(W_t x_t + b_t)$
    - And propose to control the stability via the NN architectures and constraints on weights

Crash notes on ODEs
Stability of ResNet like architectures (Haber 2017, Chang 2018)

- Informal description of the ideas
- Stability of the ODE
  - The ODE is stable if θ_t is changing sufficiently slowly and the Jacobian $J_t \triangleq \nabla_x f(x_t, \theta_t)$ satisfies (sufficient condition):
    - $\max_i Re(\lambda_i(J_t)) \leq 0, \forall t \in [0, T]$ with $\lambda_i(J_t)$ the ith eigenvalue of J_t and $Re()$ the real component
- Stability of the numerical scheme
  - Forward Euler is stable if:
    - $\max_i |1 + h\lambda_i(J_k)| \leq 1 \; \forall k = 0, \dots, l-1$ (k indexes the layers) with $J_k \triangleq \nabla_x f(x_k, \theta_k)$
- Intuition
  - The stability conditions of the ODE and of the numerical scheme should be considered in the training optimization problem
    - $\max_i Re(\lambda_i(J_t)) > 0$ amplifies the signal, and may lead to divergence
    - $\max_i Re(\lambda_i(J_t)) \ll 0$ may imply signal loss
    - They propose architectures for which $Re(\lambda_i(J_t)), i = 1..l$ is close to 0

- End of the interlude on ODEs

Modeling Spatio-temporal dynamics with Neural Networks

NNs as surrogate models for solving PDEs – Discrete space models
NNs as surrogate models for solving PDEs – Continuous space models
NNs as surrogate models for solving PDEs – Data free approaches

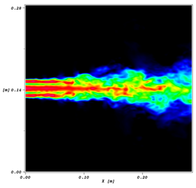
Modeling Spatio-temporal dynamics with Neural Networks Motivations

- Modeling the complex dynamics arising in natural/ physical processes
  - Objective: understanding, predicting, controling
- Physical models
  - Mathematical equations of dynamical systems
  - Often take the form of PDEs and associated numerical models
  - Stem from a deep understanding of the underlying physics
- Data driven modeling
  - In many cases data are plentiful (climate, simulations, etc)
  - Can we leverage ML for modeling these complex systems?
    - Way more complex than current ML successes (vision, language)
- Challenge: Interaction between the physical model based and the statistical paradigms

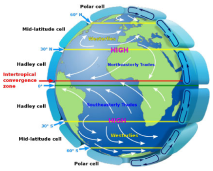
Modeling Spatio-temporal dynamics with Neural Networks Motivations

- Applications domains - examples

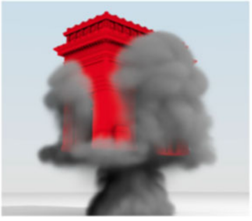
Computational Fluid Dynamics



Earth System Science - Climate



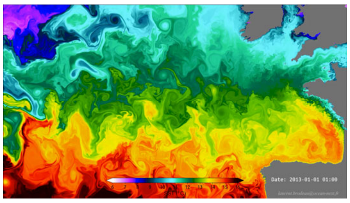
Graphical design

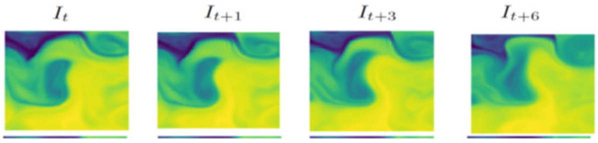


Tompson et al. 2017

- Objectives
  - Data-driven approaches: reduce computational cost – e.g. CFD: Surrogate Models/ Reduced Order Models
  - Hybrid Systems: Complement physical models

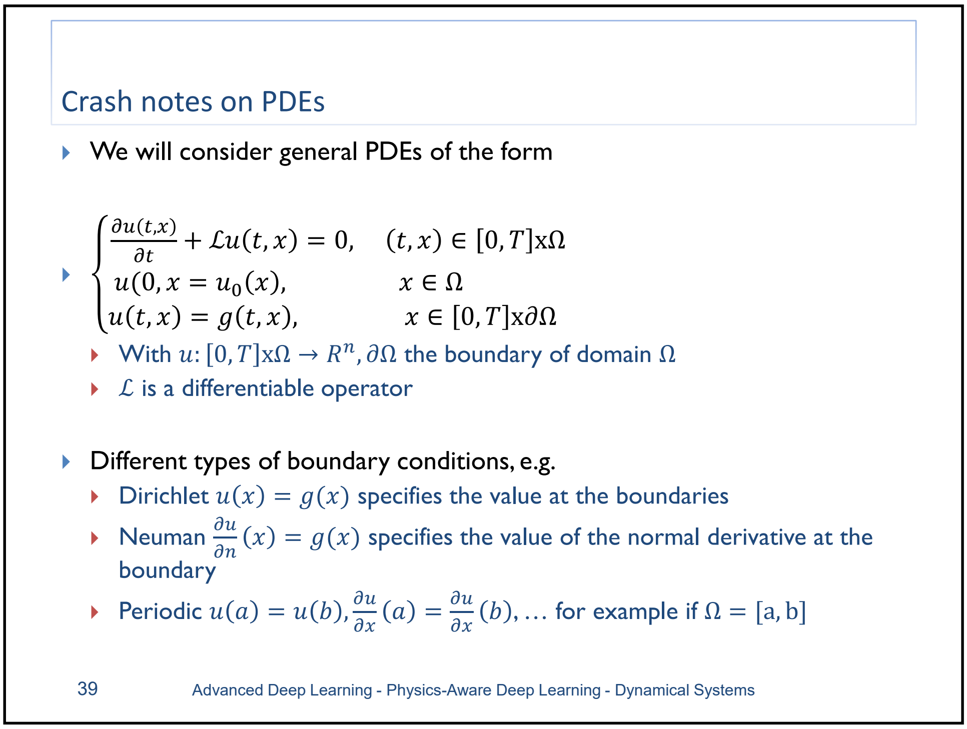
Modeling Spatio-temporal dynamics with Neural Networks Example: Sea Surface Temperature (SST)





Initial state Successive states $t+1, t+3, t+6$

Interlude - Crash notes on PDEs

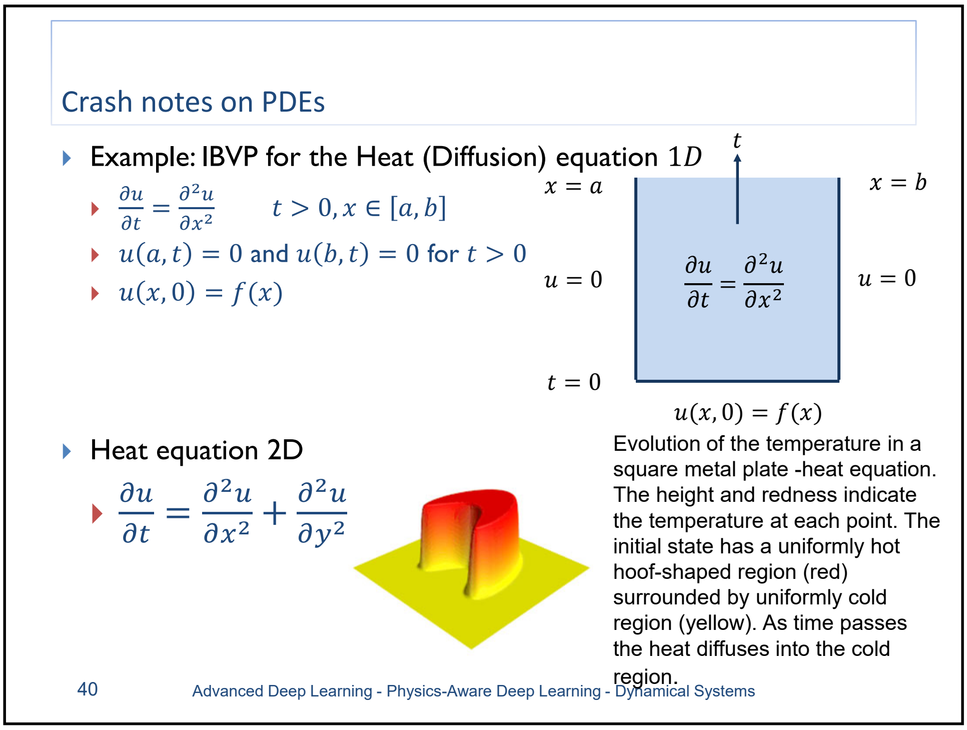


Crash notes on PDEs

- We will consider general PDEs of the form

- $$\begin{cases} \frac{\partial u(t,x)}{\partial t} + \mathcal{L}u(t,x) = 0, & (t,x) \in [0,T]\text{x}\Omega \\ u(0,x = u_0(x), & x \in \Omega \\ u(t,x) = g(t,x), & x \in [0,T]\text{x}\partial\Omega \end{cases}$$
  - With u: $[0,T]\text{x}\Omega \rightarrow R^n, \partial\Omega$ the boundary of domain Ω
  - $\mathcal{L}$ is a differentiable operator

- Different types of boundary conditions, e.g.
  - Dirichlet $u(x) = g(x)$ specifies the value at the boundaries
  - Neuman $\frac{\partial u}{\partial n}(x) = g(x)$ specifies the value of the normal derivative at the boundary
  - Periodic $u(a) = u(b), \frac{\partial u}{\partial x}(a) = \frac{\partial u}{\partial x}(b), \ldots$ for example if $\Omega = [\text{a}, \text{b}]$



Crash notes on PDEs

- Example: IBVP for the Heat (Diffusion) equation $1D$
  - $\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} \qquad t > 0, x \in [a,b]$
  - $u(a,t) = 0$ and $u(b,t) = 0$ for $t > 0$
  - $u(x,0) = f(x)$

t
$x = a$
$x = b$
$u = 0$
$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2}$
$u = 0$
$t = 0$
$u(x,0) = f(x)$

- Heat equation 2D
  - $\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}$

Evolution of the temperature in a square metal plate -heat equation. The height and redness indicate the temperature at each point. The initial state has a uniformly hot hoof-shaped region (red) surrounded by uniformly cold region (yellow). As time passes the heat diffuses into the cold region.

Crash notes on PDEs – method of lines

- Many applications of NN to PDE proceed as in the method of line (Hamdi et al. 2007)
  - Replace spatial derivatives in the PDE with algebraic approximations.
  - The spatial derivatives don't appear anymore and the only remaining independent variable is the time.
  - This transforms the PDE into a system of ODE that can be solved with classical ODE solvers using a time stepping scheme
  - This is the scheme used for most discrete space ML solvers

Crash notes on PDEs – method of lines

- Example Diffusion (heat) PDE in one dimension
  - $\frac{\partial u}{\partial t} = \mathrm{c}\frac{\partial^2 u}{\partial x^2}$ with boundary conditions $u(-L,t) = u(L,t)$
  - Spatial discretization of $u(x,t)$:
    - Denote $u(-L,t) = u_1, u(-L+\Delta\, x,t) = u_2, \dots, u(L,t) = u_{n+1}$
- Replace spatial derivatives in the PDE with algebraic approximations.
  - Discretization of the spatial derivative with a second order scheme :

$$\boxed{\frac{du_i}{dt} = \frac{c}{\Delta x^2}\,[u_{i+1} - 2u_i + u_{i-1}], \qquad \forall\, i \in \{1,n\}}$$

- The spatial derivative do not appear anymore
  - We are left with a system of n ODEs

Crash notes on PDEs – method of lines

- This system of PDE can be solved using appropriate classical ODE solvers.
  - Let us derive the ODE integration using an Euler discretization scheme for the temporal component: $\frac{\mathrm{du_i}}{\mathrm{dt}} \approx \frac{u_i^{n+1} - u_i^n}{\Delta t}$
  - The integration scheme for the system of ODEs can be rewritten as:

$$\boxed{u_i^{m+1} = u_i^m + \lambda\,(u_{i+1}^m \; - 2u_i^m + u_{i-1}^m) \qquad \forall i \in \{1, n\}}$$

with $\lambda \; = \; c\frac{\Delta t}{\Delta \mathrm{x}^2}$ the CFL (Courant-Friedrichs-Lewy) number that conditions the stability of the ODE

    - The values of u at time step $m + 1$ can be obtained from the values at time step m using neighbourhood points at space index u_i
    - Two dimensions, is slightly more complex, but the PDE can be reduced to a system of linear ODEs as in the example above
- End Interlude

Modeling Spatio-temporal dynamics with Neural Networks

✓ NNs as surrogate models for solving PDEs – Discrete space models
NNs as surrogate models for solving PDEs – Continuous space models
NNs as surrogate models for solving PDEs – Data free approaches

Basic NN architectures
Used on discrete grids

Convolutional Neural Networks
Unet
Graph Neural Networks

Convolutional Neural Networks

- Convolution of function $u(x), x \in R^n$ with kernel $g(x)$
  - $(u * g)(x) = \int_{R^d} u(y)g(y-x)dy$
  - $(u * g)(x) = \int_{R^d} u(y-\mathrm{x})g(y)dy$

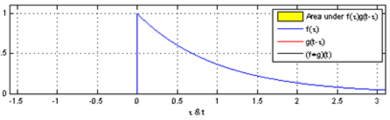
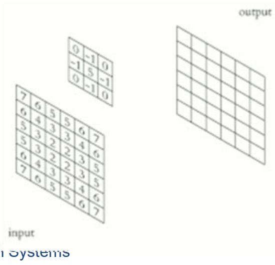
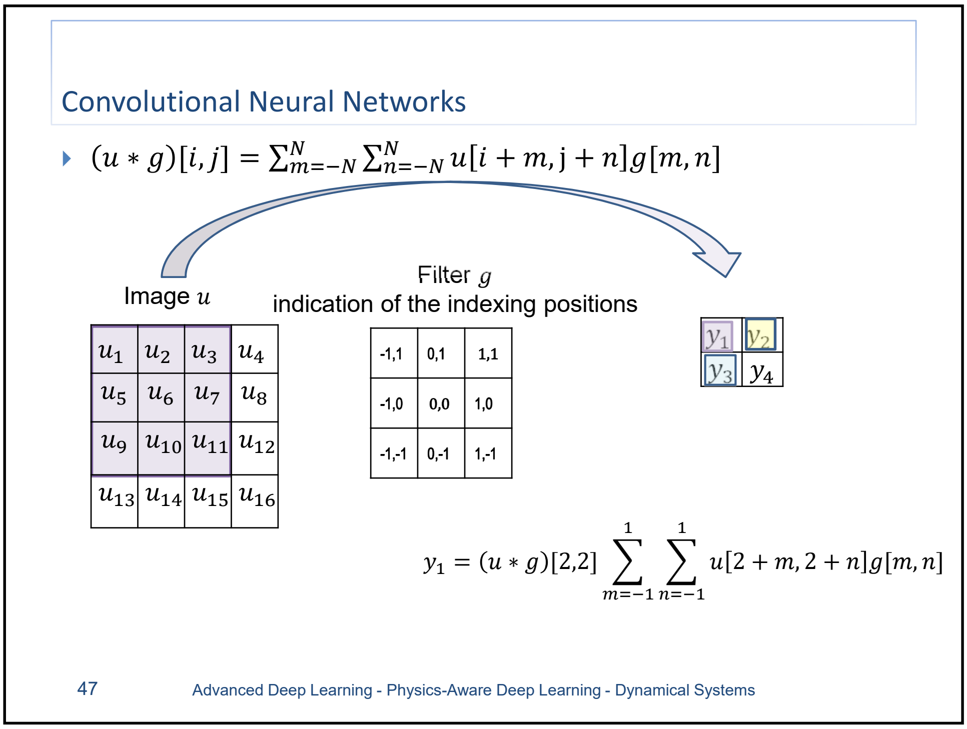


Fig. https://en.wikipedia.org/wiki/Convolution

- Discrete convolution in 2 D of $u[i,j]$ with $g[i,j]$
  - Let us suppose that g has a finite support set $\{-N, -N+1, \dots, N\}^2$
    - $(u * g)[i,j] = \sum_{m=i-N}^{i+N} \sum_{n=i-N}^{i+N} u[m,n]g[m-i, n-j]$
    - $\boxed{(u * g)[i,j] = \sum_{m=-N}^{N} \sum_{n=-N}^{N} u[i+m, \mathrm{j}+n]g[m,n]}$
      - □ The latter is the convolution used in CNNs





Convolutional Neural Networks

- $(u * g)[i, j] = \sum_{m=-N}^{N} \sum_{n=-N}^{N} u[i + m, \mathrm{j} + n] g[m, n]$

Image u

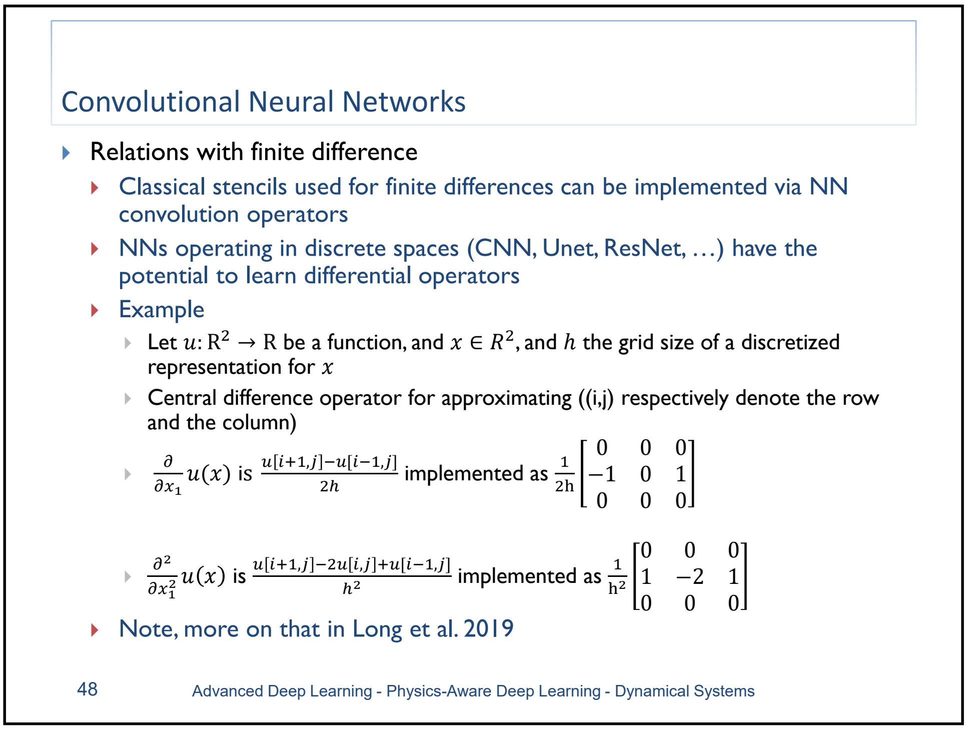
u_1	u_2	u_3	u_4
u_5	u_6	u_7	u_8
u_9	u_{10}	u_{11}	u_{12}
u_{13}	u_{14}	u_{15}	u_{16}

Filter g
indication of the indexing positions

-1,1	0,1	1,1
-1,0	0,0	1,0
-1,-1	0,-1	1,-1

y_1	y_2
y_3	y_4

$$y_1 = (u * g)[2,2] \sum_{m=-1}^{1} \sum_{n=-1}^{1} u[2 + m, 2 + n] g[m, n]$$



Convolutional Neural Networks

- Relations with finite difference
  - Classical stencils used for finite differences can be implemented via NN convolution operators
  - NNs operating in discrete spaces (CNN, Unet, ResNet, …) have the potential to learn differential operators
  - Example
    - Let $u\colon \mathrm{R}^2 \to \mathrm{R}$ be a function, and $x \in R^2$, and h the grid size of a discretized representation for x
    - Central difference operator for approximating ((i,j) respectively denote the row and the column)
    - $\frac{\partial}{\partial x_1} u(x)$ is $\frac{u[i+1,j] - u[i-1,j]}{2h}$ implemented as $\frac{1}{2\mathrm{h}} \begin{bmatrix} 0 & 0 & 0 \\ -1 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}$
    - $\frac{\partial^2}{\partial x_1^2} u(x)$ is $\frac{u[i+1,j] - 2u[i,j] + u[i-1,j]}{h^2}$ implemented as $\frac{1}{\mathrm{h}^2} \begin{bmatrix} 0 & 0 & 0 \\ 1 & -2 & 1 \\ 0 & 0 & 0 \end{bmatrix}$
  - Note, more on that in Long et al. 2019

UNets

- Introduced for image to image transformations (initially image segmentation, could be used to associate a source to the solution of a PDE)
- Encoder-decoder type architecture, V- Cycle:
  - □ First, upscale the image resolution and increase the number of channels
  - □ Then downscale to the initial resolution and reduce the number of channels
- Close to Multigrid numerical methods
  - Makes use of
    - □ Convolutions
    - □ skip connections combine information at the same resolution
  - Recent versions incorporate Attention mechanisms

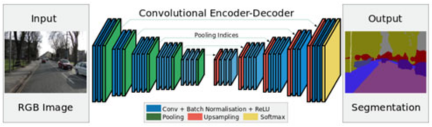


Fig. Badrinarayanan et al. 2015

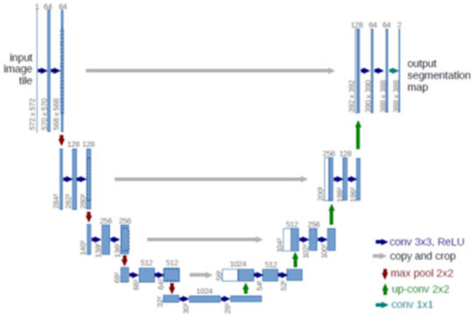
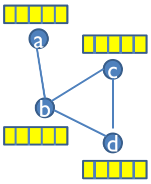


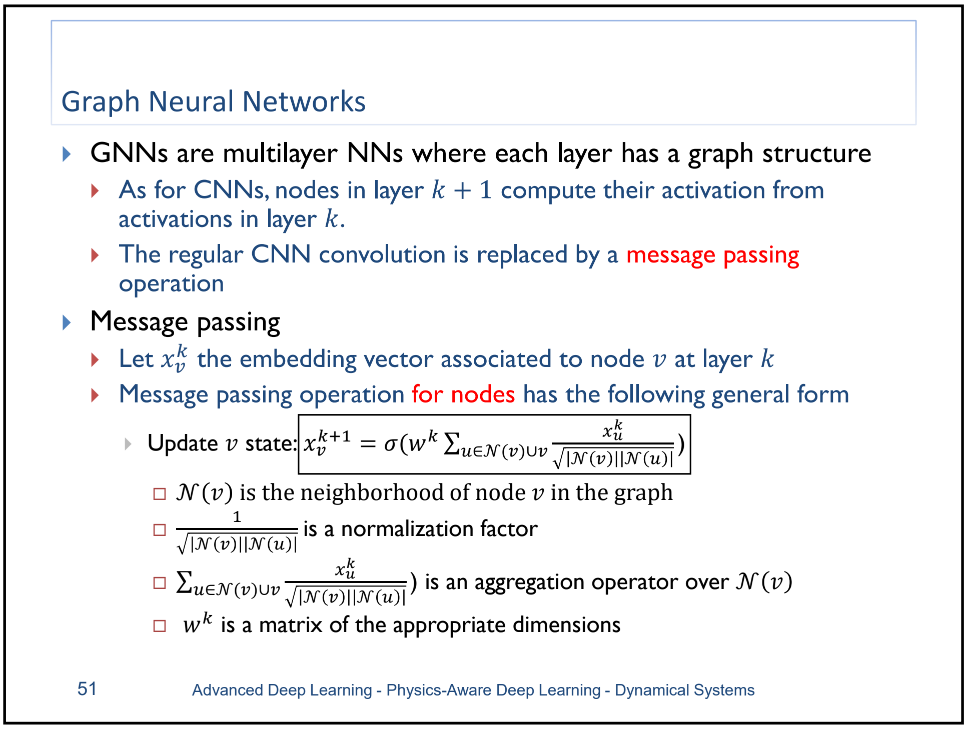
Fig. Ronneberger net al. 2015

Graph Neural Networks

- Extends the CNN ideas to irregular grids (graphs)
  - Better adapted to irregular meshes used e.g. in fluid mechanics
  - Computational cost high compared to CNNs
    - Needs to compute the neighbours for several operations – not so adapted to GPUs
  - Notations
    - $G = (V, E)$ a graph
    - To each $v \in V$, is attached a set of node features $x \in R^d$: ▭▭▭▭▭
    - $\mathcal{N}(v)$ is the neighborhood of note v in the graph
    - Alternatively, G can be described by its adjacency matrix A

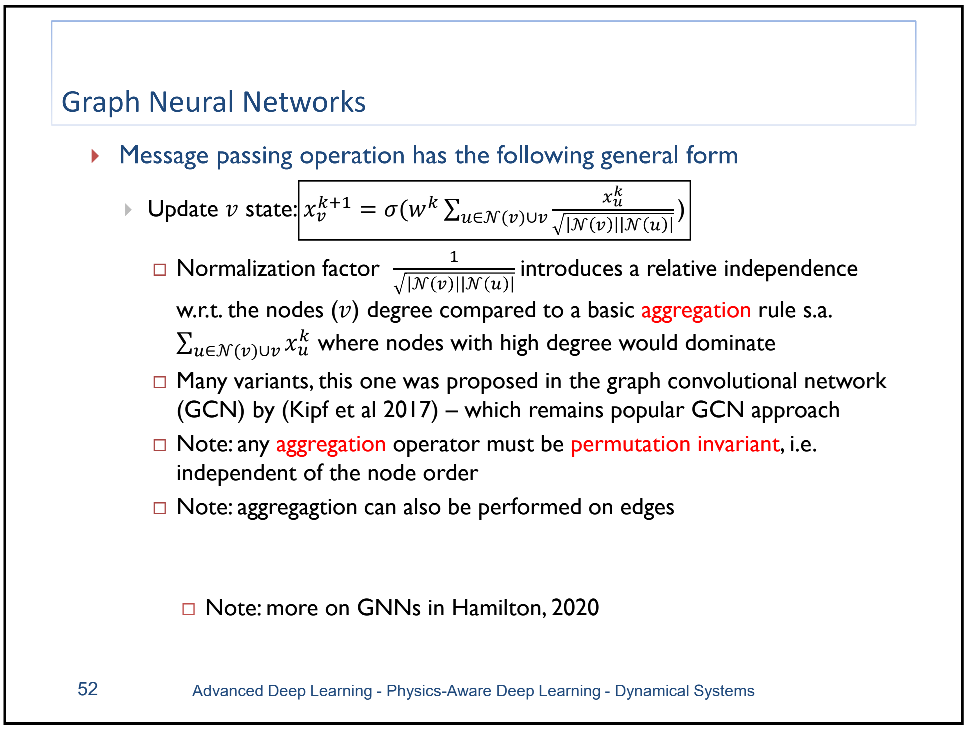


$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \end{bmatrix}$$



Graph Neural Networks

- GNNs are multilayer NNs where each layer has a graph structure
  - As for CNNs, nodes in layer $k+1$ compute their activation from activations in layer k.
  - The regular CNN convolution is replaced by a message passing operation
- Message passing
  - Let x_v^k the embedding vector associated to node v at layer k
  - Message passing operation for nodes has the following general form
    - Update v state: $\boxed{x_v^{k+1} = \sigma(w^k \sum_{u \in \mathcal{N}(v) \cup v} \frac{x_u^k}{\sqrt{|\mathcal{N}(v)||\mathcal{N}(u)|}})}$
      - □ $\mathcal{N}(v)$ is the neighborhood of node v in the graph
      - □ $\frac{1}{\sqrt{|\mathcal{N}(v)||\mathcal{N}(u)|}}$ is a normalization factor
      - □ $\sum_{u \in \mathcal{N}(v) \cup v} \frac{x_u^k}{\sqrt{|\mathcal{N}(v)||\mathcal{N}(u)|}})$ is an aggregation operator over $\mathcal{N}(v)$
      - □ w^k is a matrix of the appropriate dimensions



Graph Neural Networks

- Message passing operation has the following general form
  - Update v state: $\boxed{x_v^{k+1} = \sigma(w^k \sum_{u \in \mathcal{N}(v) \cup v} \frac{x_u^k}{\sqrt{|\mathcal{N}(v)||\mathcal{N}(u)|}})}$
    - □ Normalization factor $\frac{1}{\sqrt{|\mathcal{N}(v)||\mathcal{N}(u)|}}$ introduces a relative independence w.r.t. the nodes (v) degree compared to a basic aggregation rule s.a. $\sum_{u \in \mathcal{N}(v) \cup v} x_u^k$ where nodes with high degree would dominate
    - □ Many variants, this one was proposed in the graph convolutional network (GCN) by (Kipf et al 2017) – which remains popular GCN approach
    - □ Note: any aggregation operator must be permutation invariant, i.e. independent of the node order
    - □ Note: aggregagtion can also be performed on edges

□ Note: more on GNNs in Hamilton, 2020

Graph Neural Networks

- Attention with GCN
- A popular set aggregation rule relies on neighborhood attention
  - Attention was popularized with transformers in NLP
- Example
  - Aggregation rule $\sum_{u\in\mathcal{N}(v)\cup v} \alpha_{v,u} x_u^k$
  - With coefficients $\alpha_{v,u}$ denoting the attention on neighbor $u \in \mathcal{N}(v)$
    - $\alpha_{v,u} = \frac{\exp(a^T(Wx_v \oplus Wx_u))}{\sum_{u'\in\mathcal{N}(v)} \exp(a^T(Wx_v \oplus Wx_{u'}))}$, $\oplus$ concatenation operation
    - Wa matrix of the appropriate size
    - And possible extensions to multiple head attention as in Transformers

NNs as surrogate models for solving PDEs
Discrete space models

✓ Regular grid: Learning from partial observations – ResNets - UNets

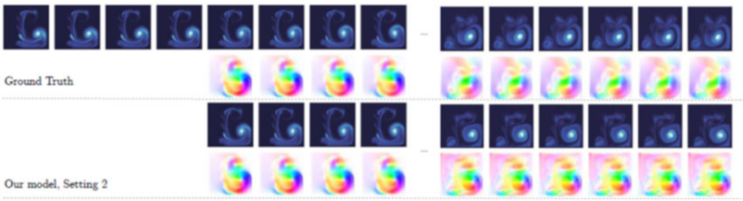
Irregular mesh: passing PDE solvers – Graph NNs

Learning from partial observations (Ayed et al. 2019- 2022)

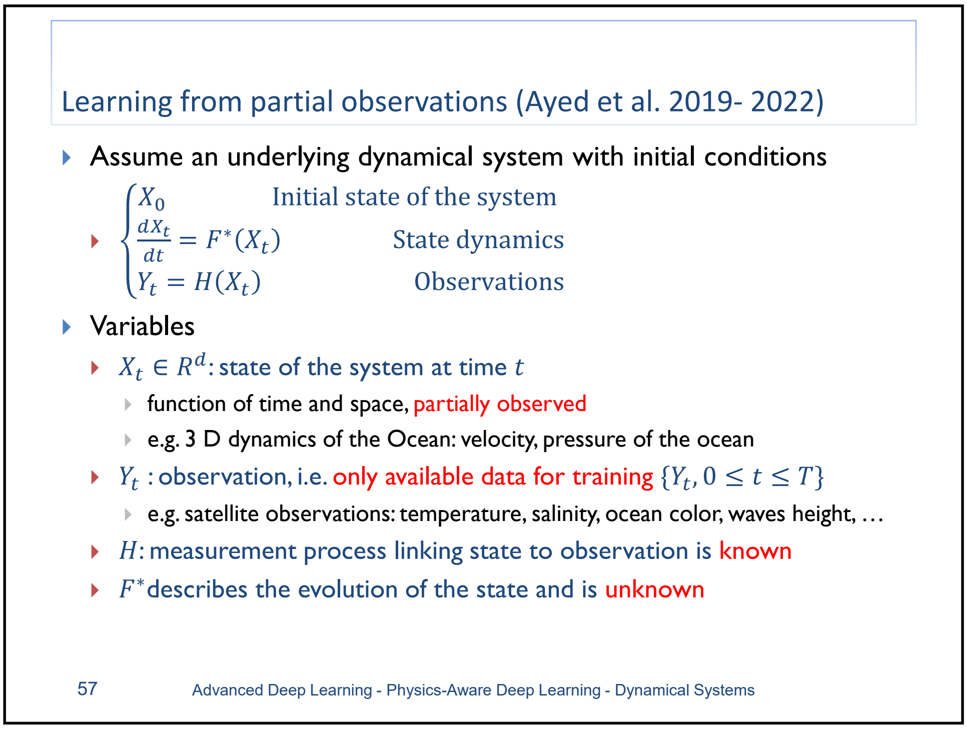
- Forecasting non linear dynamical systems from observations only
  - Data-driven approach: without any knowledge of the physics
    - The form of the PDE is unknown
  - Only assumption
    - The underlying system follows a differential equation, but the PDE is unknown
- Objective
  - Learn the evolution of this system (observations and state) from scratch with a NN
  - Discover automatically the relation between states (dynamics)

Learning from partial observations (Ayed et al. 2019- 2022)

- Illustration
  - Navier Stokes equations
    - Discretised on a spatial 64x64 grid
    - State: fluid particle density + 2D velocity field
    - Observations: density only Y
    - Initial state: true full state X_0

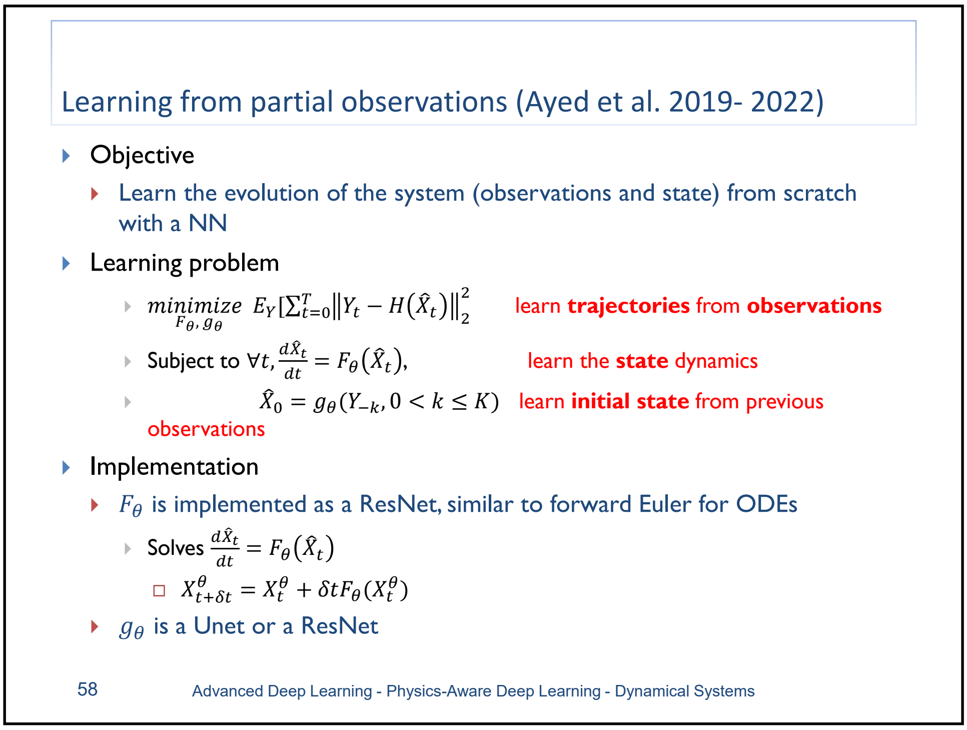


Forecasting NS – horizon prediction $T = 30$ (in blue density Y, in color 2D velocity field with color code



Learning from partial observations (Ayed et al. 2019- 2022)

- Assume an underlying dynamical system with initial conditions
  - $$\begin{cases} X_0 & \text{Initial state of the system} \\ \frac{dX_t}{dt} = F^*(X_t) & \text{State dynamics} \\ Y_t = H(X_t) & \text{Observations} \end{cases}$$
- Variables
  - $X_t \in R^d$: state of the system at time t
    - function of time and space, partially observed
    - e.g. 3 D dynamics of the Ocean: velocity, pressure of the ocean
  - Y_t : observation, i.e. only available data for training $\{Y_t, 0 \leq t \leq T\}$
    - e.g. satellite observations: temperature, salinity, ocean color, waves height, …
  - H: measurement process linking state to observation is known
  - F^*describes the evolution of the state and is unknown



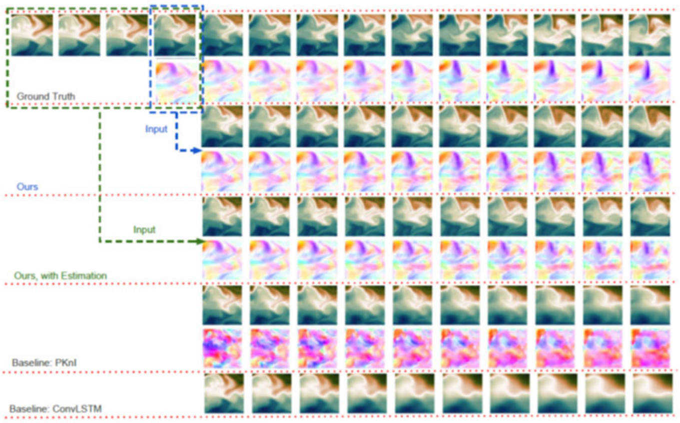
Learning from partial observations (Ayed et al. 2019- 2022)

- Objective
  - Learn the evolution of the system (observations and state) from scratch with a NN
- Learning problem
  - $\underset{F_\theta, g_\theta}{minimize}\ E_Y[\sum_{t=0}^{T} \|Y_t - H(\hat{X}_t)\|_2^2$ learn **trajectories** from **observations**
  - Subject to $\forall t, \frac{d\hat{X}_t}{dt} = F_\theta(\hat{X}_t)$, learn the **state** dynamics
  - $\hat{X}_0 = g_\theta(Y_{-k}, 0 < k \leq K)$ learn **initial state** from previous observations
- Implementation
  - F_θ is implemented as a ResNet, similar to forward Euler for ODEs
    - Solves $\frac{d\hat{X}_t}{dt} = F_\theta(\hat{X}_t)$
      - □ $X_{t+\delta t}^{\theta} = X_t^{\theta} + \delta t F_\theta(X_t^{\theta})$
  - g_θ is a Unet or a ResNet

Learning from partial observations (Ayed et al. 2019- 2022) Examples

- NEMO – Nucleus for European Modelling of the Ocean Engine
  - State: 7 variables, we make use only of 2 variables corresponding to the velocity field
  - Observations: Sea Surface Temperature
  - Initial state: interpolated from previous observations
- For all test, data are partitioned into a training and a test set
  - Horizon of 6 time steps used for the target sequence for training

Learning from partial observations (Ayed et al. 2019- 2022) NEMO – Global Ocean Physics Reanalysis



Targets (Y_t)

Targets (X_t)

Predictions $(\hat{Y}_t)$

Predictions $(\hat{X}_t)$

Figure 4. Forecasting Glorys2v4. From top to bottom: input and target observations, along with the associated ground truth partial hidden state, our model's outputs, our model variant when the initial conditions are estimated from the observations, outputs from the PKnl baseline, and from the ConvLSTM.

NNs as surrogate models for solving PDEs
Discrete space models

Regular grids: Learning from partial observations – ResNets – Unets

✓ Irregular meshes: Message passing PDE solvers – Graph NNs

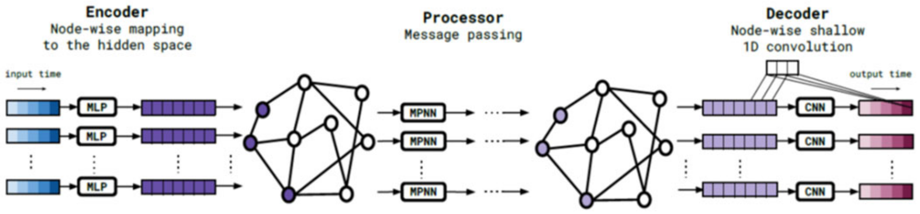
Graph Neural Networks

- GNN are well adapted to handle irregular meshes
  - Several efforts for developing PDE solvers based on graphs
    - (Sanchez-Gonzales et al. 2020, Belbute-Peres et al. 2020, Pfaff et al. 2021, …)
  - Grid cells/ nodes are mapped to a graph which is processed via message passing
- Example used in the course:
  - Brandstetter et al. 2022: Message Passing Neural PDE Solvers
    - Representative GNN solver
      - □ Handle multiple situations:
        - □ Multiple resolutions, boundary problems, parametric PDEs, etc
      - □ New improvements for training w.r.t. previous GNN PDE solvers
  - Inference
    - The mesh (precomputed) is mapped onto a graph
    - Objective: forecast spatio-temporal dynamics
    - Auto-regressive model $u(x,t) \rightarrow u(x,t+\Delta t) \rightarrow u(x,t+2\Delta t)$ …

Message Passing Neural PDE Solvers (Brandstetter et al. 2022)

Fig. Bransdtetter et al. 2022

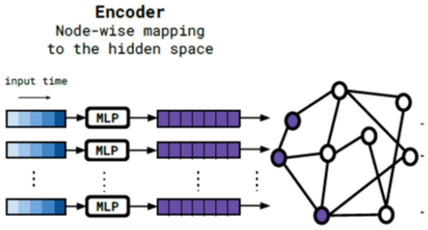
- General framework
  - Framework: Encode-Process-Decode (Sanchez-Gomzales 2020)
    - Process: message passing on the graph node embeddings



Node i, step k		
Encoding Input: last K values at each node i f_i^0 =Encode($u_i^{k-K}, \dots, u_i^k$)	**Process** M message passing steps $f_i^m, m = 1 \dots M$	**Decode** Output: next K values $u_i^{k+1}, \dots, u_i^{k+K}$

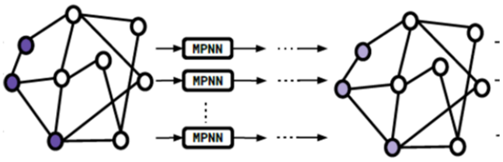
Message Passing Neural PDE Solvers (Brandstetter et al. 2022)

- Encode
  - Compute node embeddings for each node i
  - $f_i^0 = embed\left(u_i^{k-K:k}, x_i, t_k, \theta_{PDE}\right)$ (vector)
    - $u_i^{k-K:k}$: K last values, x_i: node position, t_k: time step k
    - θ_{PDE}: parameters of the equation, e.g. PDE coefficient values, boundary condition indicators etc



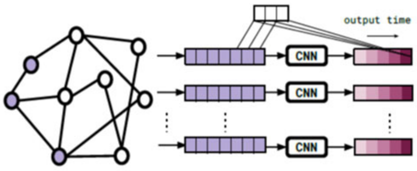
Message Passing Neural PDE Solvers (Brandstetter et al. 2022)

- Process
  - Compute M steps of node update, $f_i^1, \dots, f_i^M$ for all nodes i via message passing
  - Step m
    - Message for edge $j \to i$ $\quad m_{ij}^m = \Phi(f_i^m, f_j^m, u_i^{k-K:k} - u_j^{k-K:k}, x_i - x_j, \theta_{PDE})$
    - Update for node i $\quad f_i^{m+1} = \psi\left(f_i^m, \sum_{j\in\mathcal{N}(i)} m_{ij}^m, \theta_{PDE}\right)$
    - Φ, ψ are MLPs



Message Passing Neural PDE Solvers (Brandstetter et al. 2022)

- Decode
  - From the last node embedding f_i^M , compute next Kvalues $u_i^{k+1}, \dots, u_i^{k+K}$ for all nodes i
  - $f_i^M$is a vector and is considered as a time contiguous signal and processed through a $1D$ CNN to compute the next K predictions $u_i^{k+1}, \dots, u_i^{k+K}$



Message Passing Neural PDE Solvers (Brandstetter et al. 2022)

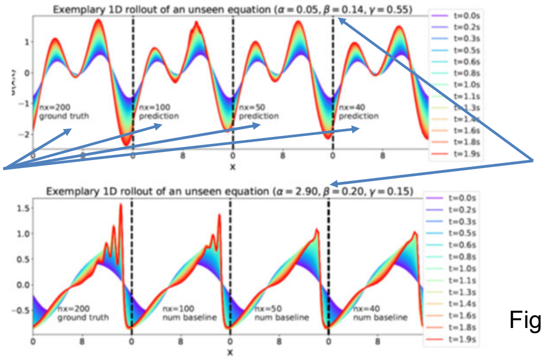
- Claim
  - Able to handle
    - parametric PDEs, with the θ_{PDE} coefficients
    - Multiple resolutions, message passing allows for multiple resolutions in the GNN
    - Multiple boundary conditions (Dirichlet, Neuman, mixture)

Message Passing Neural PDE Solvers (Brandstetter et al. 2022)

- Example
  - Generalization : family of PDE equations with different parameters and different resolutions
  - $\frac{\partial u}{\partial t} + \frac{\partial}{\partial x}\left(\alpha u^2 - \beta \frac{\partial u}{\partial x} + \gamma \frac{\partial^2 u}{\partial x^2}\right) = f(x,t)$
  - $u(0,x) = u_0(x)$
  - Encompasses several classical equations
    - $(\alpha, \beta, \gamma) = (0, \eta, 0)$ Heat equation
    - $(\alpha, \beta, \gamma) = (0.5, \eta, 0)$ Burgers equation (simplified equation for fluid flows)
    - Etc
  - $\theta_{PDE} = (\alpha, \beta, \gamma)$

Message Passing Neural PDE Solvers (Brandstetter et al. 2022) Example

- Generalization : to PDE equations with different parameters values for (α, β, γ) and different resolutions
- Not real generalization but interpolation in the range of training values for (α, β, γ)
- Colours correspond to different times



resolutions

Parameter values (α, β, γ)

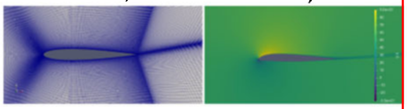
Fig. Brandstetter et al. 2022

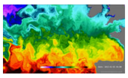
Modeling Spatio-temporal dynamics with Neural Networks

NNs as surrogate models for solving PDEs – Discrete space models
✓ NNs as surrogate models for solving PDEs – Continuous space models
NNs as surrogate models for solving PDEs – Data free approaches

Neural operators

- Classical numerical solvers operate on grids or meshes (finite differences, finite elements, finitie volumes)





- Neural solvers operate on tensors (grids) or on graphs (irregular meshes)

- Neural operators is a recent topic aiming at learning maps between function spaces instead of vector spaces
  - e.g. images are considered as continuous functions
- Key ideas
  - Functions and operators are mesh/ resolution invariant
  - They can be applied for different geometries, for multiple resolutions
- Learning operator methods are data driven

Learning operators

- Instead of learning maps between vector space, learn maps between function spaces
  - Images for example are considered as continuous functions
  - The objective is then to learn the operator mapping an input to an output function space
- Learning operator methods are data driven
  - As usual, one makes use of a training set of input-output pairs in order to learn the operator
  - Generate input/output data using a PDE solver or collect data from sensors
- Key ideas
  - Functions and operators are mesh/ resolution invariant
  - Operator learns to interpolate between function spaces
- Examples presented here
  - Neural Fourier operators a popular families starting in 2020 – with several developments
  - Implicit models and CORAL for a recent alternative approach and time discretization free approaches

NNs as surrogate models for solving PDEs – Continuous space models

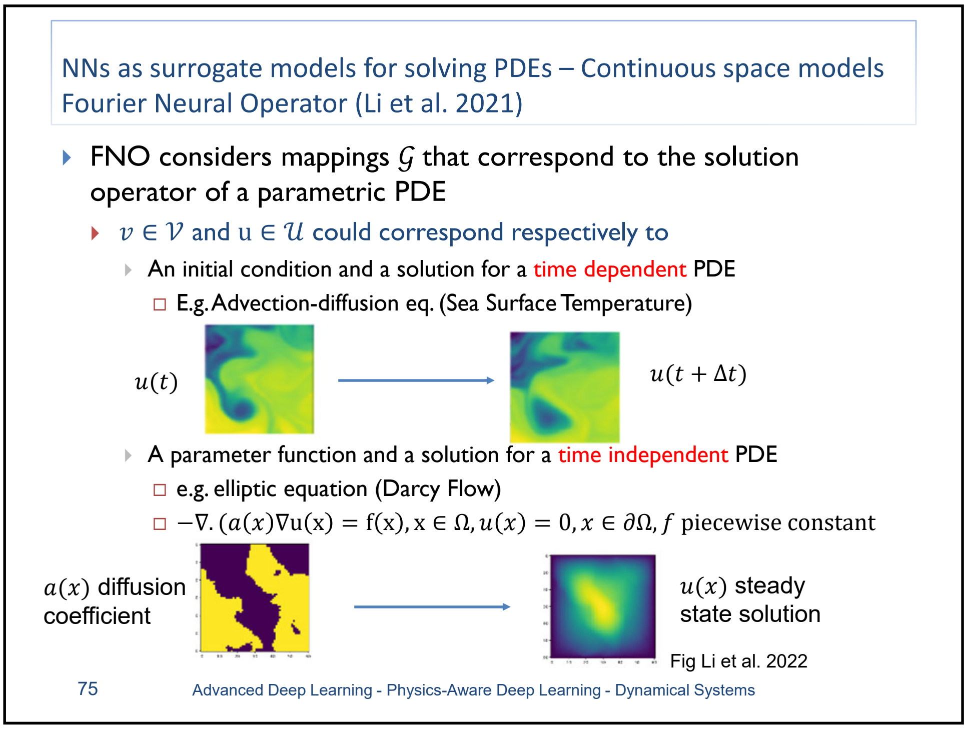
✓ Fourier Neural Operators

Implicit models

CORAL: coordinate-based model for operator learning

NNs as surrogate models for solving PDEs – Continuous space models Fourier Neural Operator (Li et al. 2021)

- We consider
  - $\mathcal{V} = \mathcal{V}(\Omega \subset R^d; R^n), \mathcal{U} = U(\Omega' \subset R^{d'}; \mathrm{R}^{\mathrm{m}})$ two function spaces
  - $\mathcal{G}: \mathcal{V} \rightarrow \mathcal{U}$ a non linear unknown mapping between the two function spaces
    - FNO considers mappings $\mathcal{G}$ that correspond to the solution operator of a parametric PDE
    - $v \in \mathcal{V}$ and $\mathrm{u} \in \mathcal{U}$ could correspond respectively to
      - □ an initial condition and a solution for a time dependent PDE
      - □ A parameter function and a solution for a time independent PDE
- Objective
  - Learn $\mathcal{G}_\theta$ an approximation of $\mathcal{G}$ from a finite set of samples
  - Samples are provided as p-points discretization of functions $v \in \mathcal{V}$ and $\mathrm{u} \in \mathcal{U}$
    - i.e. in practice we learn from discrete spaces, the representation of the continuous functions $v \in \mathcal{V}$ and $u \in \mathcal{U}$

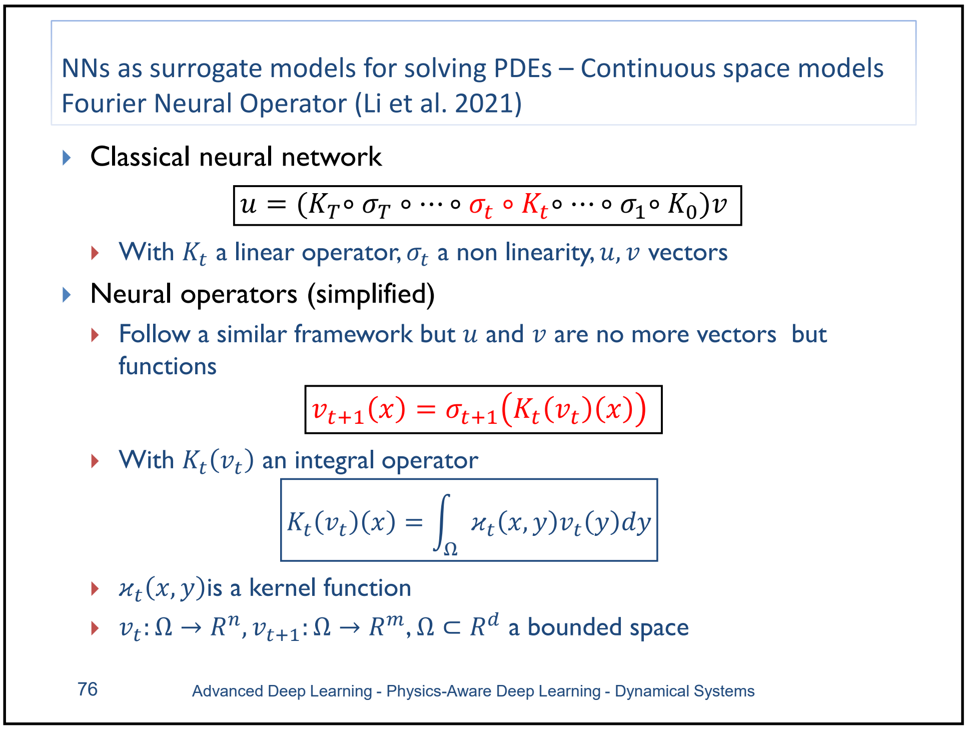


NNs as surrogate models for solving PDEs – Continuous space models Fourier Neural Operator (Li et al. 2021)

- FNO considers mappings $\mathcal{G}$ that correspond to the solution operator of a parametric PDE
  - $v \in \mathcal{V}$ and u $\in \mathcal{U}$ could correspond respectively to
    - An initial condition and a solution for a time dependent PDE
      - □ E.g. Advection-diffusion eq. (Sea Surface Temperature)

    - A parameter function and a solution for a time independent PDE
      - □ e.g. elliptic equation (Darcy Flow)
      - □ $-\nabla.(a(x)\nabla \mathrm{u(x)} = \mathrm{f(x)}, \mathrm{x} \in \Omega, u(x) = 0, x \in \partial\Omega, f$ piecewise constant

Fig Li et al. 2022



NNs as surrogate models for solving PDEs – Continuous space models Fourier Neural Operator (Li et al. 2021)

- Classical neural network

$$\boxed{u = (K_T \circ \sigma_T \circ \cdots \circ \sigma_t \circ K_t \circ \cdots \circ \sigma_1 \circ K_0)v}$$

  - With K_t a linear operator, σ_t a non linearity, u, v vectors
- Neural operators (simplified)
  - Follow a similar framework but u and v are no more vectors but functions

$$\boxed{v_{t+1}(x) = \sigma_{t+1}\big(K_t(v_t)(x)\big)}$$

  - With $K_t(v_t)$ an integral operator

$$\boxed{K_t(v_t)(x) = \int_\Omega \varkappa_t(x,y)v_t(y)dy}$$

  - $\varkappa_t(x,y)$is a kernel function
  - $v_t\colon \Omega \to R^n, v_{t+1}\colon \Omega \to R^m, \Omega \subset R^d$ a bounded space

NNs as surrogate models for solving PDEs – Continuous space models
Fourier Neural Operator (Li et al. 2021)

- How to learn the kernel function $\varkappa_t$?
- We consider the simplified update rule

$$\boxed{u(x) = K(v)(x) \ = \int_{\Omega} \varkappa(x,y)v(y)dy}$$

  - with $v, u: \Omega \to R^n$
- FNO works in Fourier space
  - Let us make $\varkappa(x,y) = \varkappa(x-y)$
    - $u(x) = \int_{\Omega} \varkappa(x-y)v(y)dy$
    - $u(x) = \int_{-\infty}^{+\infty} \varkappa(x-y)v'(y)dy$ with $v'(y) = \mathbb{1}_{\Omega}(y)v(y)$ <<<<<< convolution

$$\boxed{u(x) = (\varkappa * v')(x)}$$

  - Convolution theorem:

$$\boxed{u(x) = \mathcal{F}^{-1}(\mathcal{F}(\varkappa).\mathcal{F}(v'))(x)}$$

  - Convolution in space is equivalent to pointwise multiplication in Fourier domain
  - $\mathcal{F}(\varkappa)$ is a linear transformation

NNs as surrogate models for solving PDEs – Continuous space models
Fourier Neural Operator (Li et al. 2021)

- Fourier transform – Linear Transform – Inverse Fourier

$$\boxed{u(x) = \mathcal{F}^{-1}(\mathcal{F}(\varkappa).\mathcal{F}(v'))(x)}$$

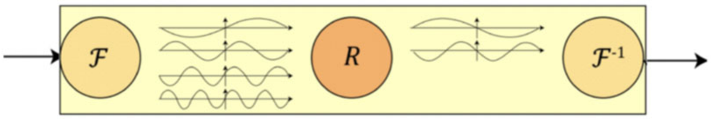


Fig Li et al., 2021

- Findings
  - In practice, it is sufficient to take the lower frequency modes
  - Fourier filters operate at the global level, different from CNN filters operating at a local level
  - R is a linear operator – implemented as a tensor

NNs as surrogate models for solving PDEs – Continuous space models Fourier Neural Operator (Li et al. 2021)

- Whole module

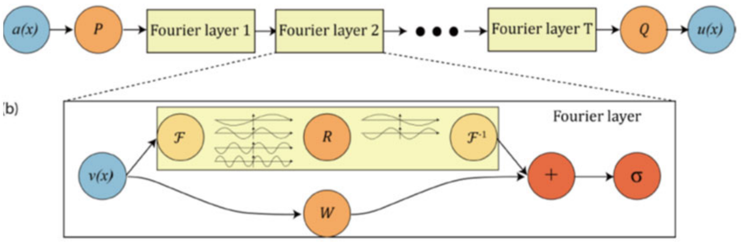


Fig Li et al. 2021

  - Fourier Layer works only with periodic conditions
  - W captures non periodic conditions
  - σ transforms are performed in the spatial domain
- In practice
  - $\mathcal{F}$ is implemented via a Fast Fourier Transform (complexity $n \log n$, n nb of spatial points)
  - Operates on regular grids only
  - But FFT is independent of the grid size
    - Could be used on resolutions different from the training ones

NNs as surrogate models for solving PDEs – Continuous space models Fourier Neural Operator (Li et al. 2021)

- Example: zero shot super-resolution
  - 2 D Navier Stokes, vorticity form, viscous incompressible fluid
    - $\frac{\partial}{\partial t} w(x,t) + u(x,t).\nabla w(x,t) = \nu \Delta w(x,t) + f(x), x \in (0,1)^2, t \in (0,T]$
    - $\nabla. u(x,t) = 0, x \in (0,1)^2, t \in (0,T)$
    - $u(x,t)$ velocity field, $w(x,t)$ vorticity, characterizes local rotation of the fluid
  - Fig. Illustrates super-resolution: trained at 64x64, test on 256x256

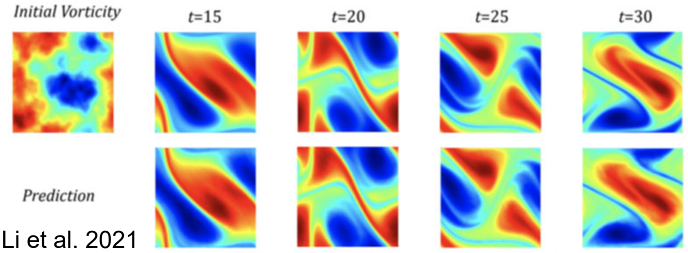


Fig Li et al. 2021

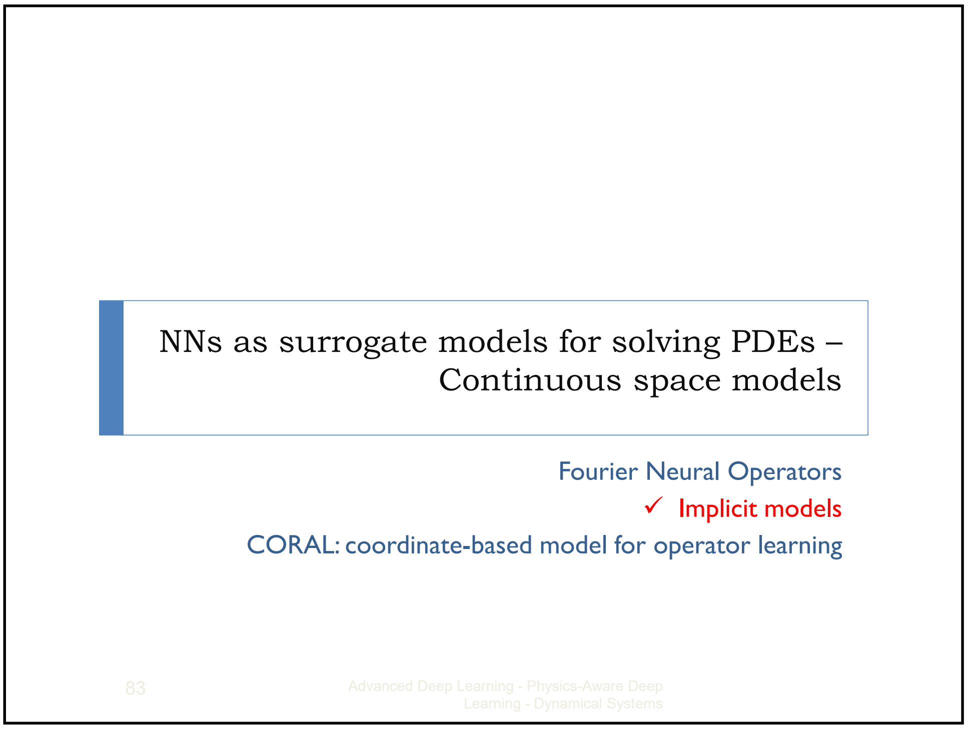
Zero-shot super-resolution: Navier-Stokes Equation with viscosity $\nu = 1\mathrm{e}{-4}$; Ground truth on top and prediction on bottom; trained on $64 \times 64 \times 20$ dataset; evaluated on $256 \times 256 \times 80$ (see Section 5.4).

NNs as surrogate models for solving PDEs – Continuous space models Fourier Neural Operator (Li et al. 2021)

- Details on N-S example
  - 2 D Navier Stokes, vorticity form, viscous incompressible fluid
    - $\frac{\partial}{\partial t}w(x,t) + u(x,t).\nabla w(x,t) = \nu\Delta w(x,t) + f(x), x \in (0,1)^2, t \in (0,T)$
    - $\nabla.u(x,t) = 0, x \in (0,1)^2, t \in (0,T)$
    - $w(x,0) = w_0(x), x \in (0,1)^2$
      - □ $\nabla u = (\frac{\partial u}{\partial x}, \frac{\partial u}{\partial y})$
      - □ $\Delta u = \nabla.\nabla \mathrm{u}$
      - □ $\nabla.\mathrm{v}= \frac{\partial v_x}{\partial x} + \frac{\partial v_y}{\partial y}$ for a 2 D vector v – divergence operator
      - □ $w = \nabla \times u$ with x the curl operator

NNs as surrogate models for solving PDEs – Continuous space models Fourier Neural Operator (Li et al. 2021)

- Many extensions/ variants
  - Irregular grids, Physics informed FNO (Li et al. 2022) Transformer FNO, large size application to weather forecasting (Pathak 2022)
- Approximation theorem
  - Universal property approximation of operator classes by (F)NO e.g. Kowachki 2022

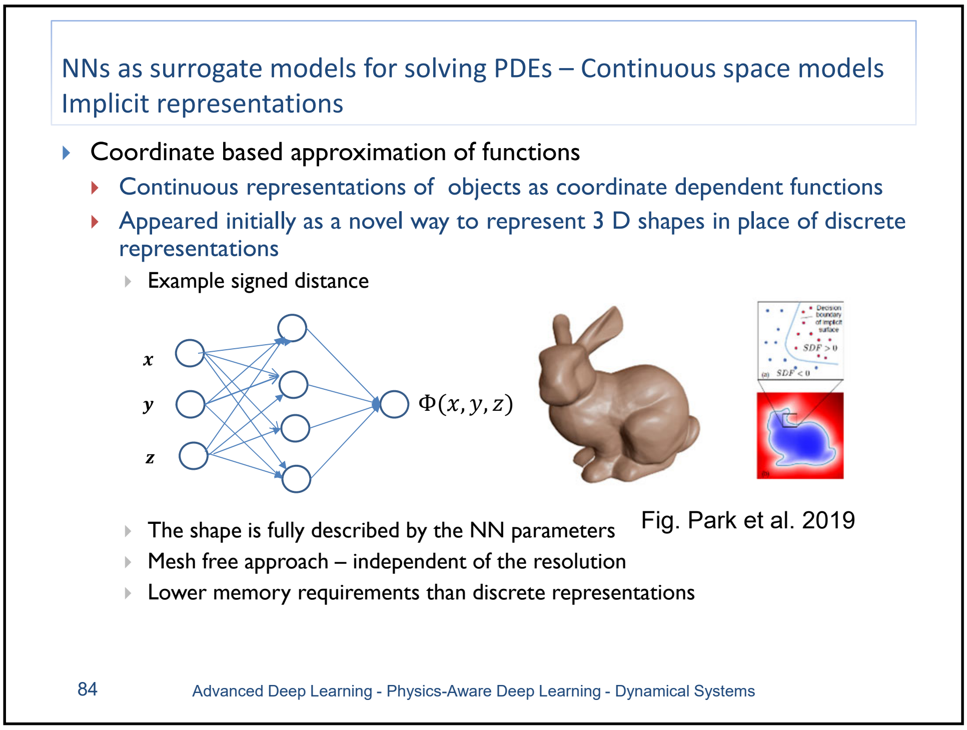


NNs as surrogate models for solving PDEs – Continuous space models

Fourier Neural Operators

✓ Implicit models

CORAL: coordinate-based model for operator learning

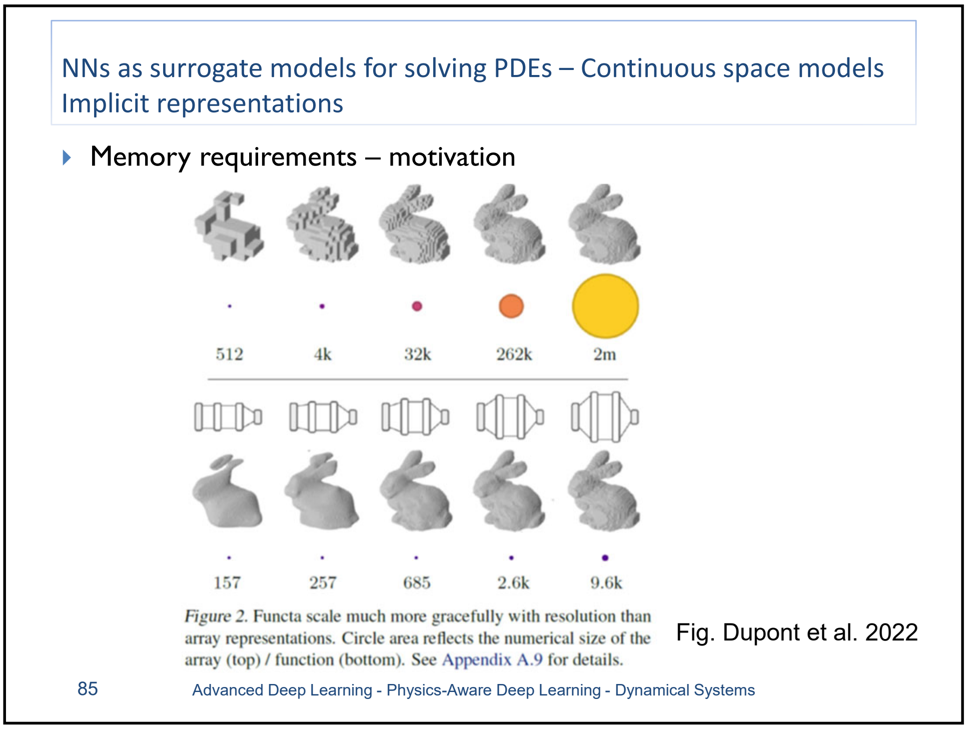


NNs as surrogate models for solving PDEs – Continuous space models Implicit representations

- Coordinate based approximation of functions
  - Continuous representations of objects as coordinate dependent functions
  - Appeared initially as a novel way to represent 3 D shapes in place of discrete representations
    - Example signed distance

Fig. Park et al. 2019

- The shape is fully described by the NN parameters
- Mesh free approach – independent of the resolution
- Lower memory requirements than discrete representations



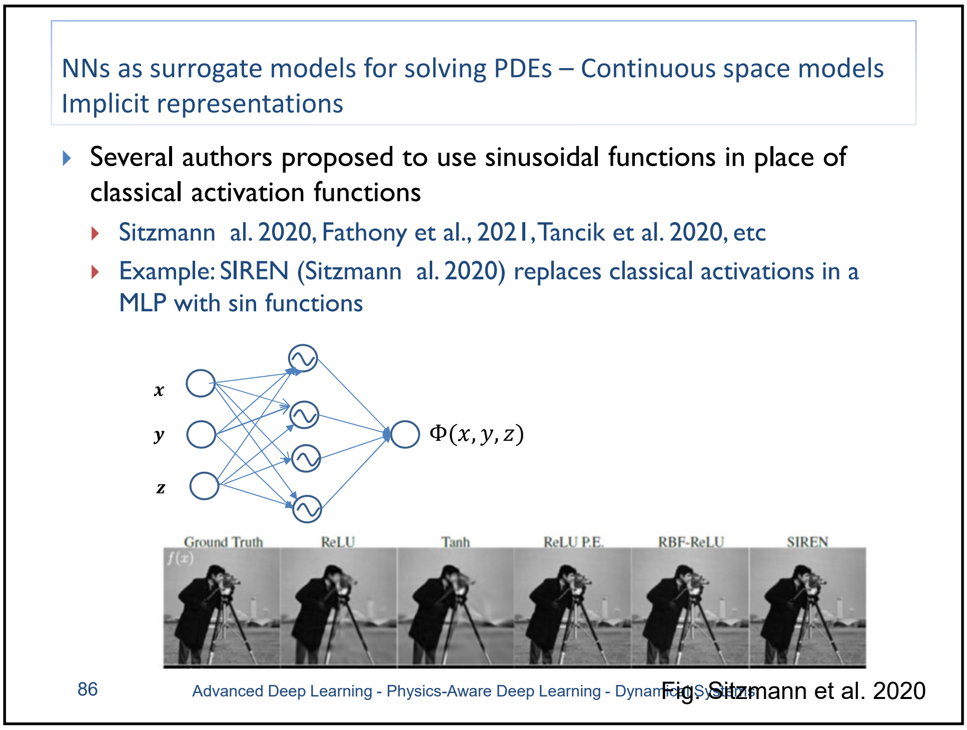
NNs as surrogate models for solving PDEs – Continuous space models Implicit representations

- Memory requirements – motivation

512
4k
32k
262k
2m
157
257
685
2.6k
9.6k

Figure 2. Functa scale much more gracefully with resolution than array representations. Circle area reflects the numerical size of the array (top) / function (bottom). See Appendix A.9 for details.

Fig. Dupont et al. 2022



NNs as surrogate models for solving PDEs – Continuous space models Implicit representations

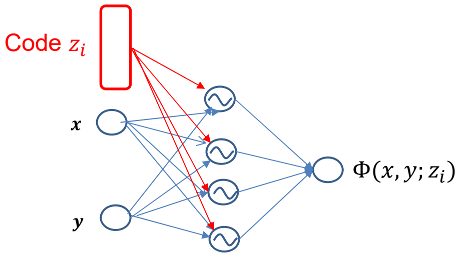
- Several authors proposed to use sinusoidal functions in place of classical activation functions
  - Sitzmann al. 2020, Fathony et al., 2021, Tancik et al. 2020, etc
  - Example: SIREN (Sitzmann al. 2020) replaces classical activations in a MLP with sin functions

x
y
z
$\Phi(x, y, z)$
Ground Truth
$f(x)$
ReLU
Tanh
ReLU P.E.
RBF-ReLU
SIREN

Fig. Sitzmann et al. 2020

Neural Fields (Implicit Neural Representations)

- Learning several images
  - A neural field model represents one image
  - How to represent multiple images using a single model?
    - Condition the neural field on a compact code specific of an image



    - This code z_i could be learned e.g. through auto encoding by gradient descent and is specific to an image
    - Conditioning is performed through an hypernetwork
    - Network weights (in blue) are shared across images

NNs as surrogate models for solving PDEs – Continuous space models

Neural Operators
DeepONet
Implicit models
✓ CORAL: coordinate-based model for operator learning

CORAL : COordinate-based model for opeRAtor Learning

- Tasks

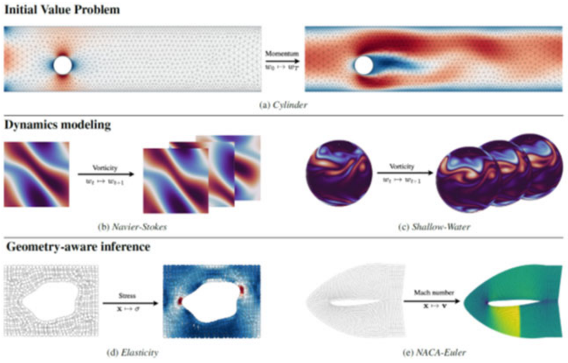
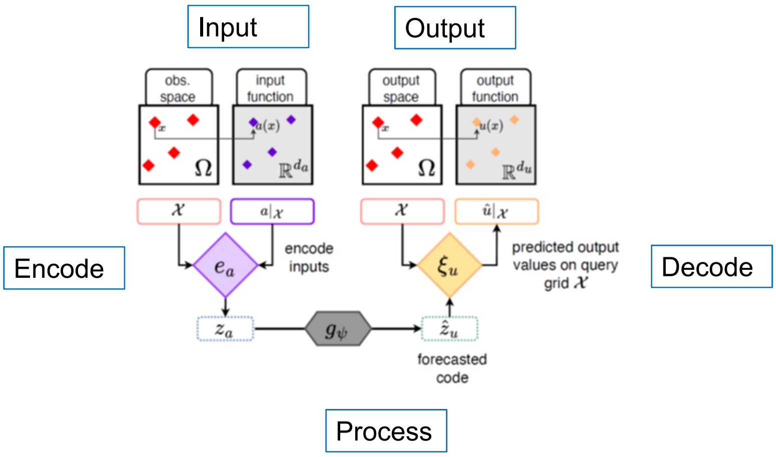


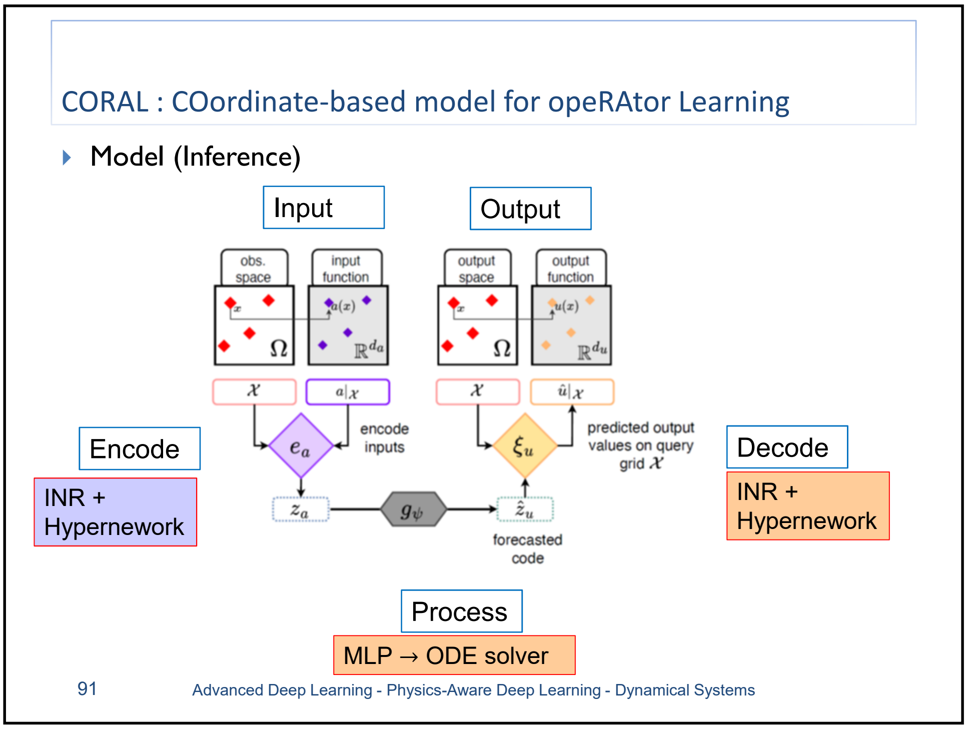
Figure 1: Illustration of the problem classes addressed in this work: Initial Value Problem (IVP) (a), dynamic forecasting (b and c) and geometry-aware inference (d and e).

- Objective
  - Learn on geometries (e.g. NACA meshes), generalize on new geometries
  - Handle general geometries

CORAL : COordinate-based model for opeRAtor Learning

- Model (Inference)





CORAL : COordinate-based model for opeRAtor Learning

- Model (Inference)

Input

Output

Encode

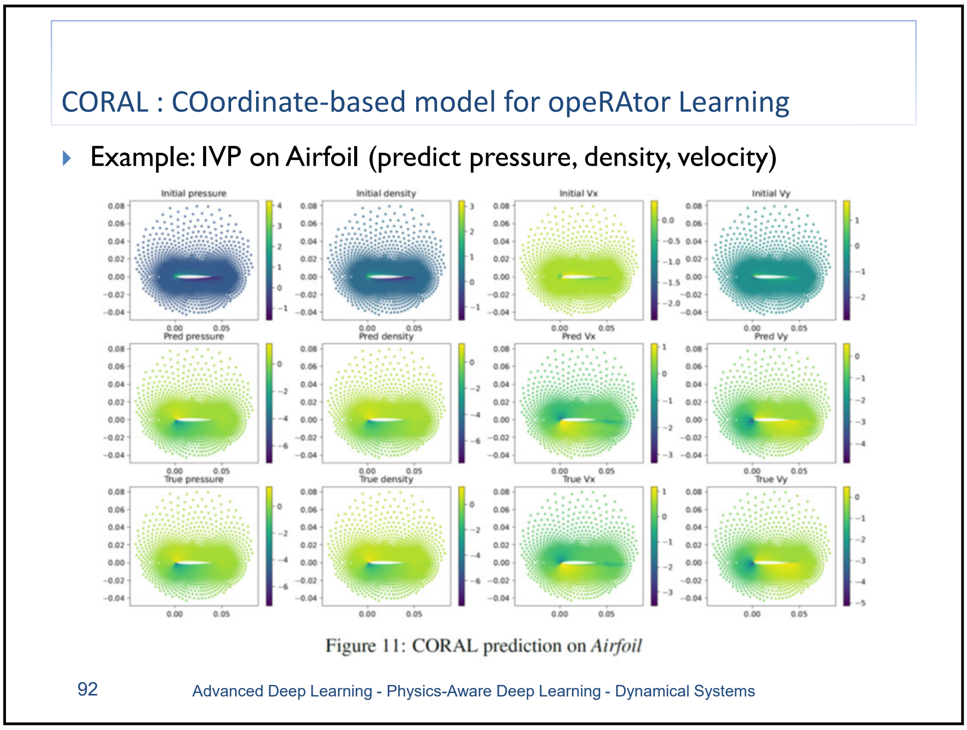
INR +
Hypernework

Decode

INR +
Hypernework

Process

MLP → ODE solver



CORAL : COordinate-based model for opeRAtor Learning

- Example: IVP on Airfoil (predict pressure, density, velocity)

Figure 11: CORAL prediction on *Airfoil*

CORAL : COordinate-based model for opeRAtor Learning

- Example: forecasting on Shallow-Water (vorticity)

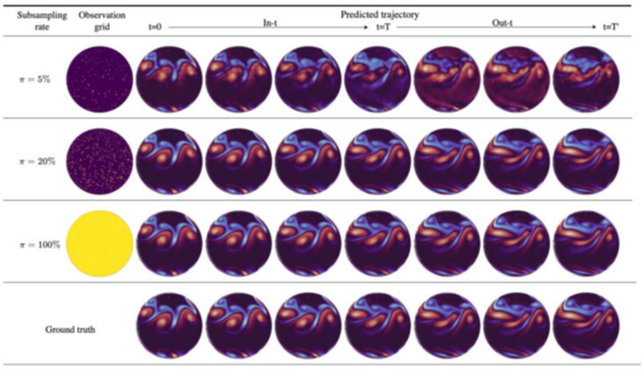


Figure 13: Prediction MSE per frame for CORAL on *Shallow-Water* with its corresponding training grid $\mathcal{X}$. Each row corresponds to a different sampling rate and the last row is the ground truth. The predicted trajectory is predicted from $t = 0$ to $t = T'$. In our setting, $T = 19$ and $T' = 39$.

CORAL : COordinate-based model for opeRAtor Learning

- Geometry aware inference: NACA-Euler (Mach number)

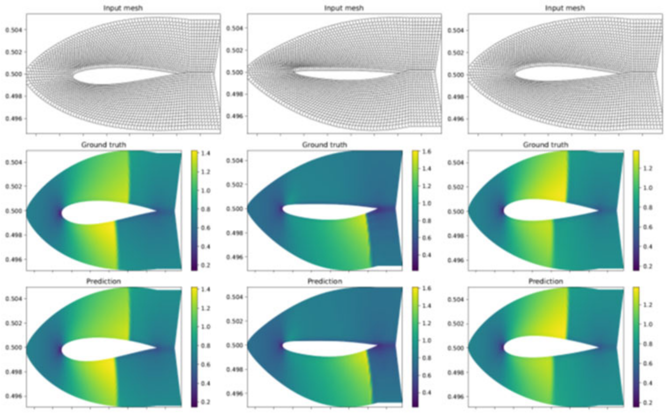
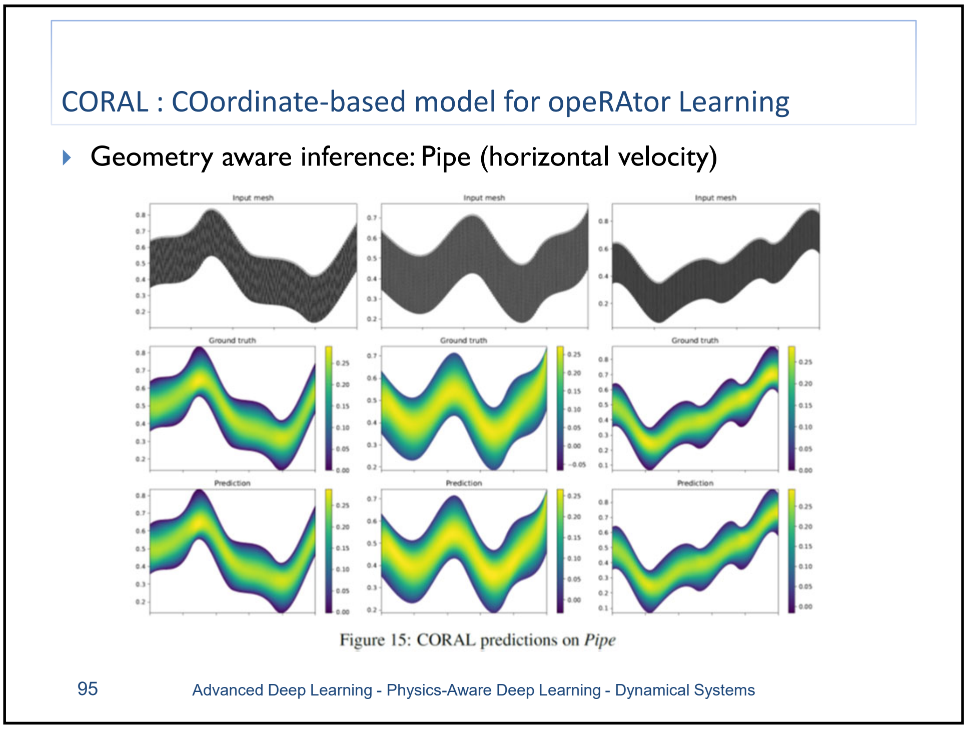


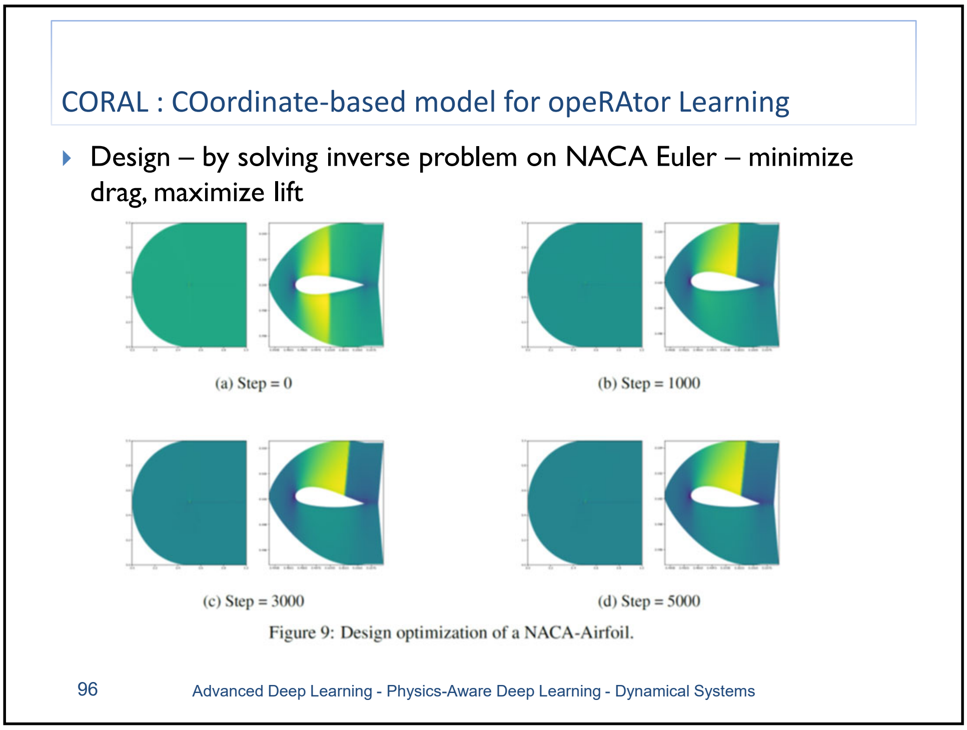
Figure 14: CORAL predictions on *NACA-Euler*



CORAL : COordinate-based model for opeRAtor Learning

- Geometry aware inference: Pipe (horizontal velocity)

Figure 15: CORAL predictions on *Pipe*



CORAL : COordinate-based model for opeRAtor Learning

- Design – by solving inverse problem on NACA Euler – minimize drag, maximize lift

(a) Step = 0

(b) Step = 1000

(c) Step = 3000

(d) Step = 5000

Figure 9: Design optimization of a NACA-Airfoil.

Modeling Spatio-temporal dynamics with Neural Networks

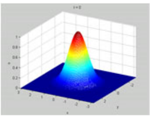
NNs as surrogate models for solving PDEs – Discrete space models
NNs as surrogate models for solving PDEs – Continuous space models
✓ NNs as surrogate models for solving PDEs – Data free approaches

NNs as surrogate models for solving PDEs – Data free approaches (Lagaris 1998, Sirignano 2018, Raissi 2019)

- Objective
  - Build a reduced order (parametric) model, implemented by a NN, to offer a cheap approximate solution of a PDE
  - Assumption: the form of the PDE is known as for classical solvers
  - Data free approach: no need for simulated data/ observations as required by all the other approaches seen before
- Results
  - The algorithm solves the PDE using a single parametric function, for all space and time conditions
  - The original algorithm solves a unique IBVP – and shall be re-trained for a new IBVP

NNs as surrogate models for solving PDEs – Data free approaches (Lagaris 1998 , Sirignano 2018, Raissi 2019)

- Problem
  - Parabolic PDE with d spatial dimensions
  - $$\begin{cases} \frac{\partial u(t,x)}{\partial t} + \mathcal{L}u(t,x) = 0, (t,x) \in [0,T] \times \Omega, \Omega \subset R^d & \text{PDE} \\ u(t=0,x) = u_0(x)\ , x \in \Omega & \text{Initial conditions} \\ u(t,x) = g(t,x)\ , x \in \partial\Omega & \text{Boundary conditions} \end{cases}$$
  - $u(t,x)$ is the spatio-temporal quantity of interest
  - $\mathcal{L}u(t,x)$ is the differential term of the PDE
    - e.g. Burgers $1D$: $\frac{\partial u(t,x)}{\partial t} = -\mathrm{u}\frac{\partial u(t,x)}{\partial x} + \nu\frac{\partial^2 u(t,x)}{\partial x^2}$



- Objective
  - Approximate $u(t,x)$ with a NN $f(t,x;\theta), \theta \in R^K$are the network parameters

NNs as surrogate models for solving PDEs – Data free approaches (Lagaris 1998 , Sirignano 2018, Raissi 2019)

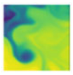
- Formulate the problem as minimizing an objective function
  - Foe simplification we only consider initial conditions (remove BC)

Initial condition loss

PDE loss: constraint

  - $$J(f) = \|f(0,x;\theta) - u_0(x)\|^2_{\Omega,\nu_1} + \left\|\frac{\partial f(t,x;\theta)}{\partial t} - \mathcal{L}f(t,x;\theta)\right\|^2_{[0,T]\times\Omega,\nu_2}$$

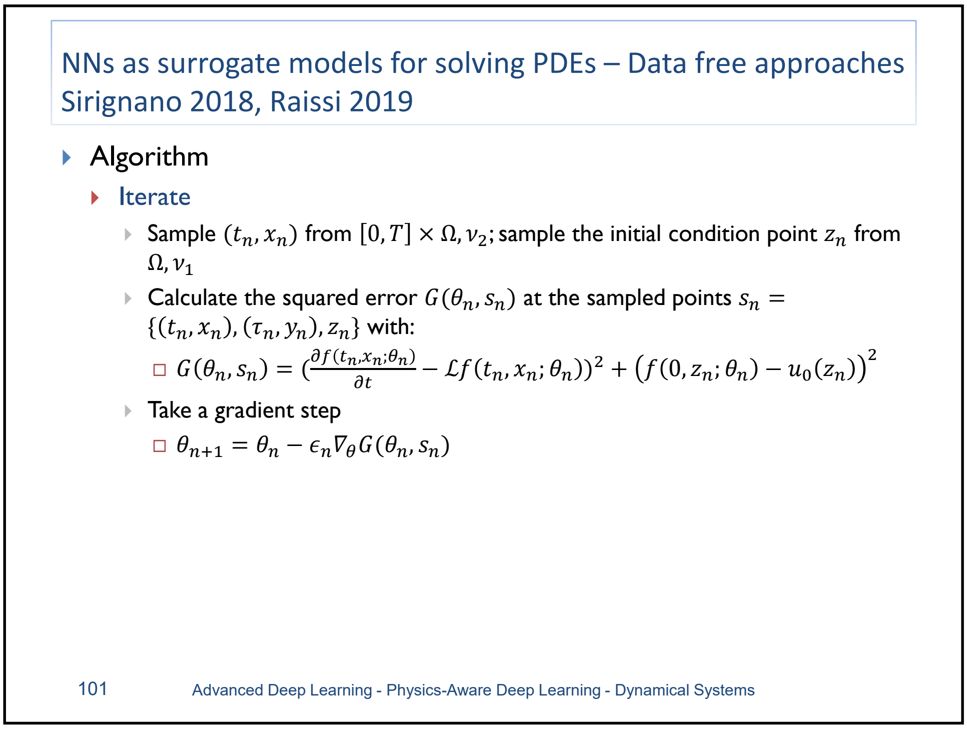
$u_0()$:



- Learn $f(O,x;\theta)$ by sampling from Ω, the initial condition
- This is a regression problem
- This provides a parametric approximation of target $u(x,t=0)$

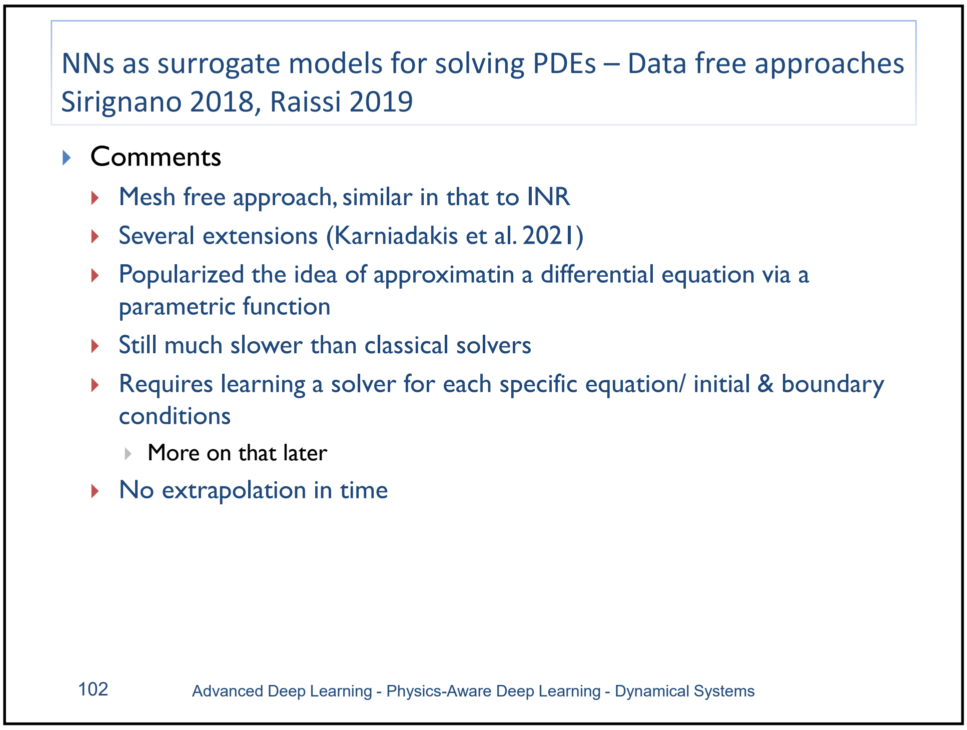
- Constrains $f(t,x;\theta)$ to follow the PDE expression by sampling uniformly from $[0,T] \times \Omega$
- $\frac{\partial f}{\partial t}$ and $\mathcal{L}u(t,x)$ computed by automatic differentiation

  - Solved using stochastic gradient descent
  - Several extensions, e.g. sampling from data from the PDE loss, etc



NNs as surrogate models for solving PDEs – Data free approaches Sirignano 2018, Raissi 2019

- Algorithm
  - Iterate
    - Sample (t_n, x_n) from $[0, T] \times \Omega, \nu_2$; sample the initial condition point z_n from Ω, ν_1
    - Calculate the squared error $G(\theta_n, s_n)$ at the sampled points $s_n = \{(t_n, x_n), (\tau_n, y_n), z_n\}$ with:
      - □ $G(\theta_n, s_n) = (\frac{\partial f(t_n, x_n; \theta_n)}{\partial t} - \mathcal{L}f(t_n, x_n; \theta_n))^2 + \big(f(0, z_n; \theta_n) - u_0(z_n)\big)^2$
    - Take a gradient step
      - □ $\theta_{n+1} = \theta_n - \epsilon_n \nabla_\theta G(\theta_n, s_n)$



NNs as surrogate models for solving PDEs – Data free approaches Sirignano 2018, Raissi 2019

- Comments
  - Mesh free approach, similar in that to INR
  - Several extensions (Karniadakis et al. 2021)
  - Popularized the idea of approximatin a differential equation via a parametric function
  - Still much slower than classical solvers
  - Requires learning a solver for each specific equation/ initial & boundary conditions
    - More on that later
  - No extrapolation in time

NNs as surrogate models for solving PDEs – Data free approaches (Sirignano 2018, Raissi 2019)

- Example: Burger equation

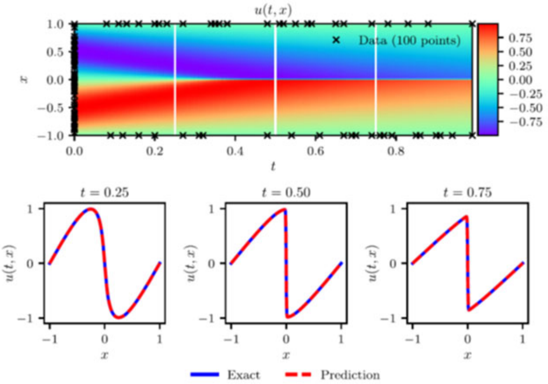


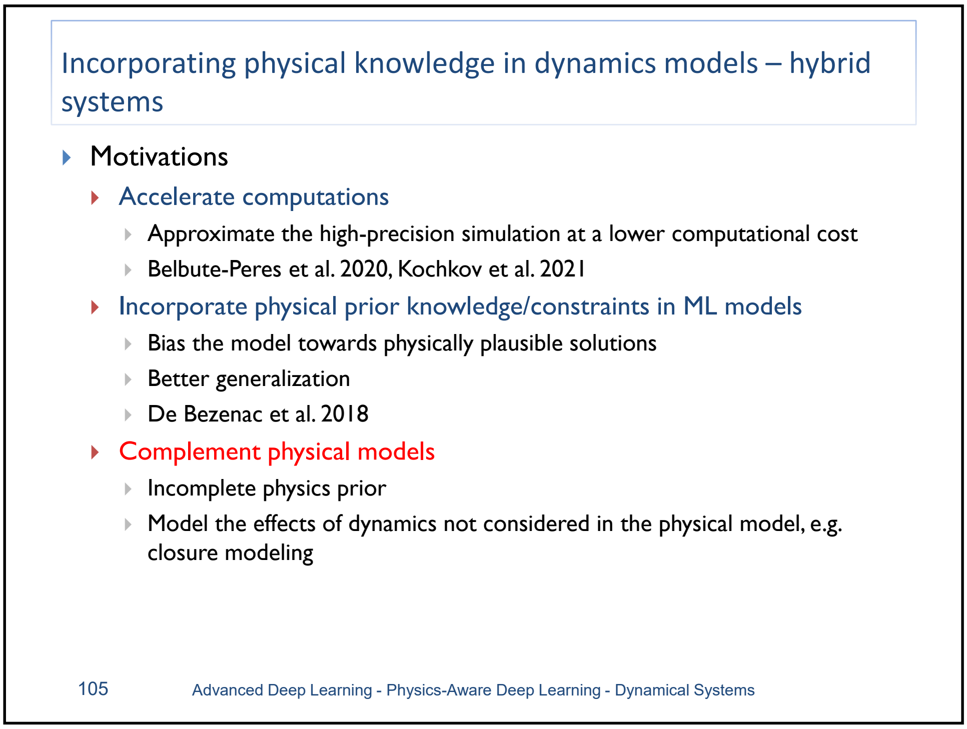
Fig. A.6. *Burgers' equation: Top:* Predicted solution $u(t, x)$ along with the initial and boundary training data. In addition we are using 10,000 collocation points generated using a Latin Hypercube Sampling strategy. *Bottom:* Comparison of the predicted and exact solutions corresponding to the three temporal snapshots depicted by the white vertical lines in the top panel. The relative $\mathbb{L}_2$ error for this case is $6.7 \cdot 10^{-4}$. Model training took approximately 60 seconds on a single NVIDIA Titan X GPU card.

Fig: Raissy 2019

Hybrids and generalization

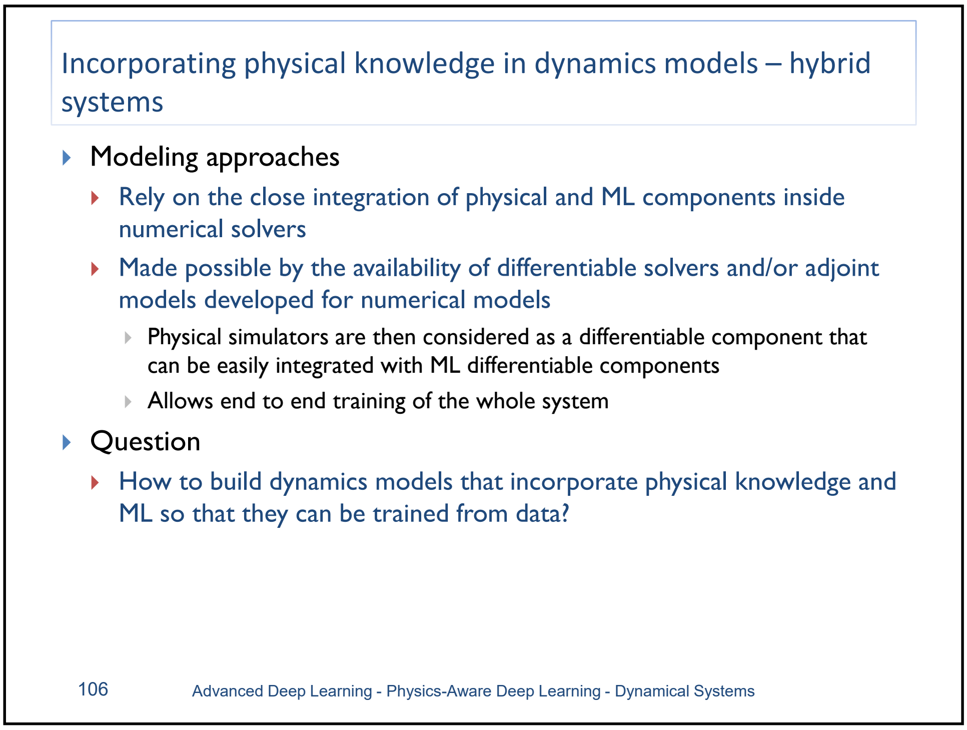
✓ Incorporating physical knowledge in dynamics models – hybrid systems

Generalization in ML models for dynamics modeling



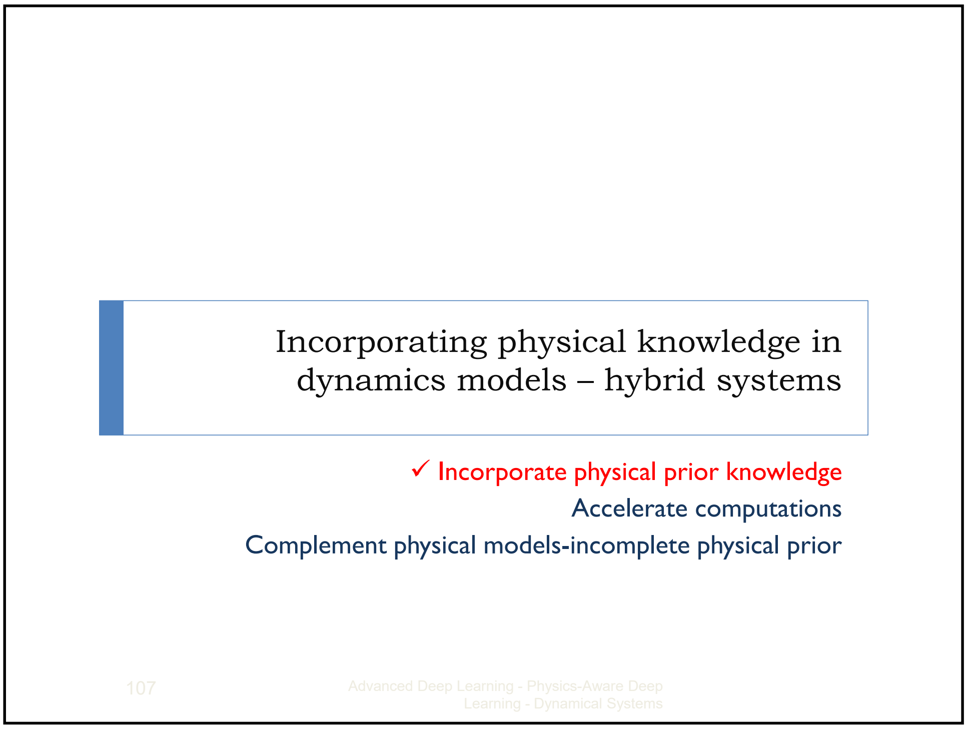
Incorporating physical knowledge in dynamics models – hybrid systems

- Motivations
  - Accelerate computations
    - Approximate the high-precision simulation at a lower computational cost
    - Belbute-Peres et al. 2020, Kochkov et al. 2021
  - Incorporate physical prior knowledge/constraints in ML models
    - Bias the model towards physically plausible solutions
    - Better generalization
    - De Bezenac et al. 2018
  - Complement physical models
    - Incomplete physics prior
    - Model the effects of dynamics not considered in the physical model, e.g. closure modeling



Incorporating physical knowledge in dynamics models – hybrid systems

- Modeling approaches
  - Rely on the close integration of physical and ML components inside numerical solvers
  - Made possible by the availability of differentiable solvers and/or adjoint models developed for numerical models
    - Physical simulators are then considered as a differentiable component that can be easily integrated with ML differentiable components
    - Allows end to end training of the whole system
- Question
  - How to build dynamics models that incorporate physical knowledge and ML so that they can be trained from data?

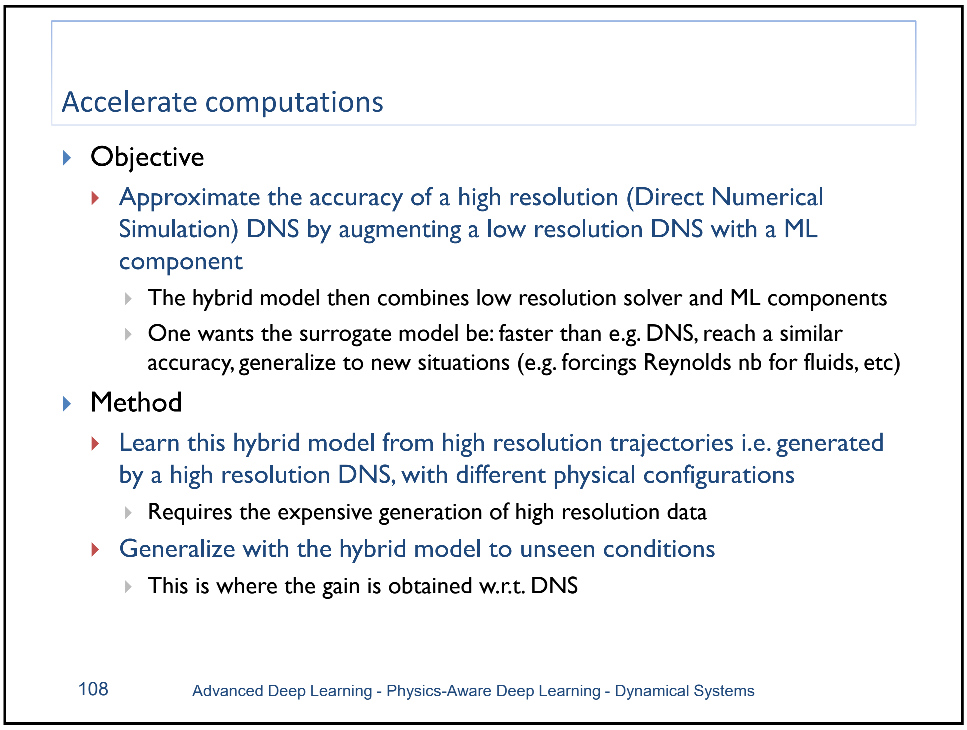


Incorporating physical knowledge in dynamics models – hybrid systems

✓ Incorporate physical prior knowledge

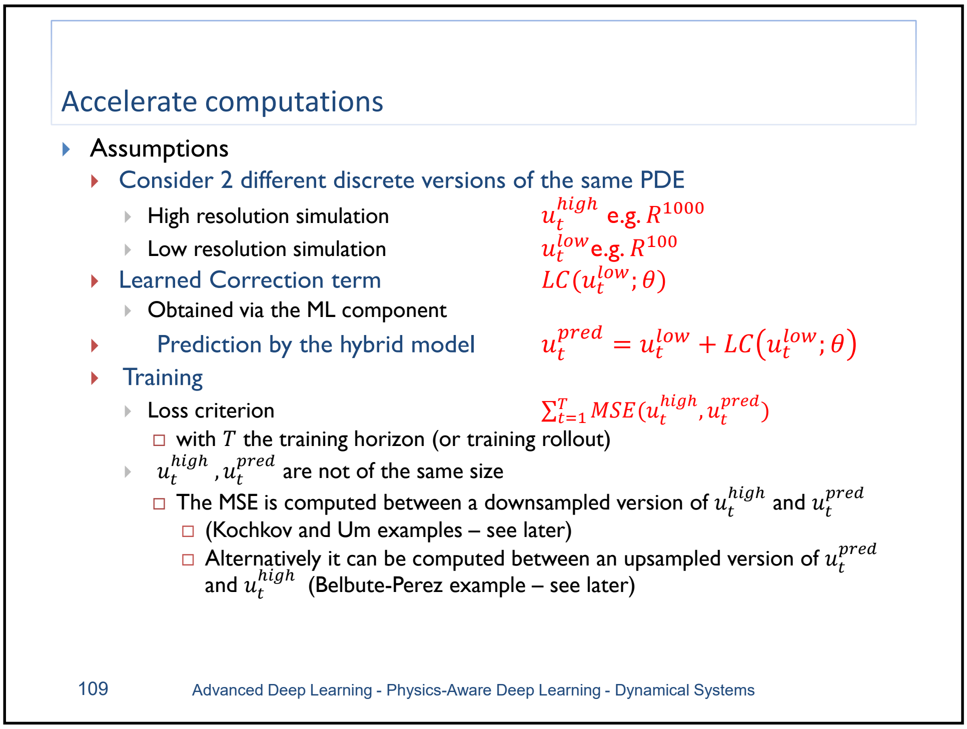
Accelerate computations

Complement physical models-incomplete physical prior



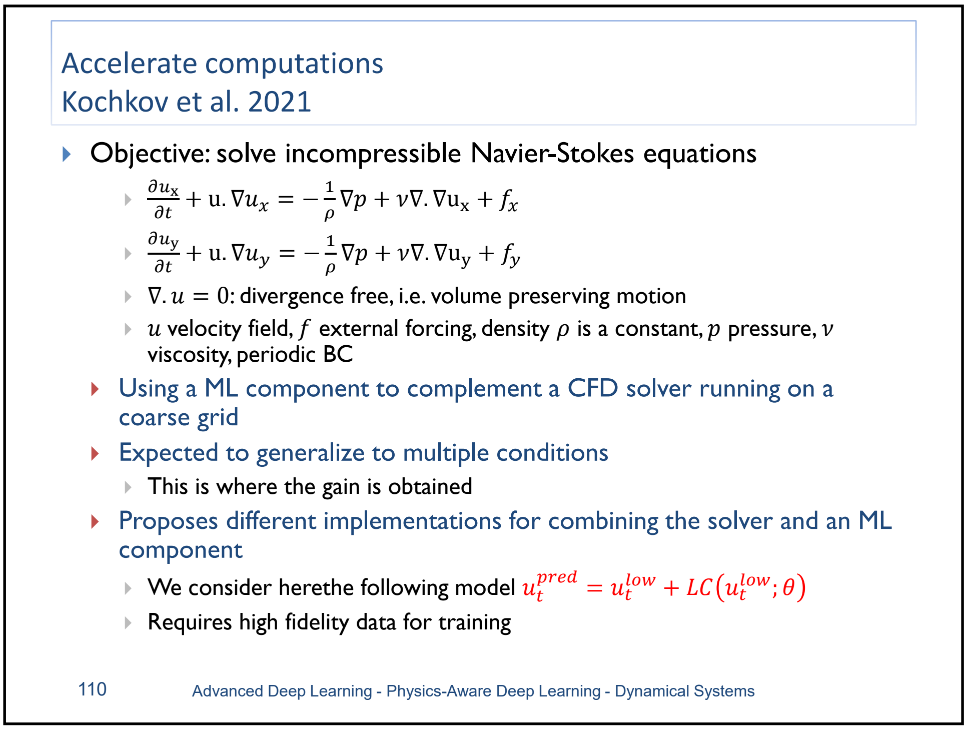
Accelerate computations

- Objective
  - Approximate the accuracy of a high resolution (Direct Numerical Simulation) DNS by augmenting a low resolution DNS with a ML component
    - The hybrid model then combines low resolution solver and ML components
    - One wants the surrogate model be: faster than e.g. DNS, reach a similar accuracy, generalize to new situations (e.g. forcings Reynolds nb for fluids, etc)
- Method
  - Learn this hybrid model from high resolution trajectories i.e. generated by a high resolution DNS, with different physical configurations
    - Requires the expensive generation of high resolution data
  - Generalize with the hybrid model to unseen conditions
    - This is where the gain is obtained w.r.t. DNS



Accelerate computations

- Assumptions
  - Consider 2 different discrete versions of the same PDE
    - High resolution simulation u_t^{high} e.g. R^{1000}
    - Low resolution simulation u_t^{low}e.g. R^{100}
  - Learned Correction term $LC(u_t^{low};\theta)$
    - Obtained via the ML component
  - Prediction by the hybrid model $u_t^{pred} = u_t^{low} + LC\left(u_t^{low};\theta\right)$
  - Training
    - Loss criterion $\sum_{t=1}^{T} MSE(u_t^{high}, u_t^{pred})$
      - □ with T the training horizon (or training rollout)
    - u_t^{high}, u_t^{pred} are not of the same size
      - □ The MSE is computed between a downsampled version of u_t^{high} and u_t^{pred}
        - □ (Kochkov and Um examples – see later)
        - □ Alternatively it can be computed between an upsampled version of u_t^{pred} and u_t^{high} (Belbute-Perez example – see later)



Accelerate computations
Kochkov et al. 2021

- Objective: solve incompressible Navier-Stokes equations
    - $\frac{\partial u_x}{\partial t} + \mathrm{u}.\nabla u_x = -\frac{1}{\rho}\nabla p + \nu\nabla.\nabla \mathrm{u_x} + f_x$
    - $\frac{\partial u_y}{\partial t} + \mathrm{u}.\nabla u_y = -\frac{1}{\rho}\nabla p + \nu\nabla.\nabla \mathrm{u_y} + f_y$
    - $\nabla.u = 0$: divergence free, i.e. volume preserving motion
    - u velocity field, f external forcing, density ρ is a constant, p pressure, ν viscosity, periodic BC
  - Using a ML component to complement a CFD solver running on a coarse grid
  - Expected to generalize to multiple conditions
    - This is where the gain is obtained
  - Proposes different implementations for combining the solver and an ML component
    - We consider herethe following model $u_t^{pred} = u_t^{low} + LC\left(u_t^{low};\theta\right)$
    - Requires high fidelity data for training

Accelerate computations
Kochkov et al. 2021

- Example – Turbulent flows
  - Figure represents scalar vorticity: $\omega = \frac{\partial}{\partial x} u_y - \frac{\partial}{\partial y} u_x$
  - Top: high fidelity reference, middle learned correction, bottom low fidelity, boxes: evolution of a single vortex
  - Learned interpolation close to high fidelity

DNS 1024x1024

Learned Interpolation 64x64

DNS 64x64

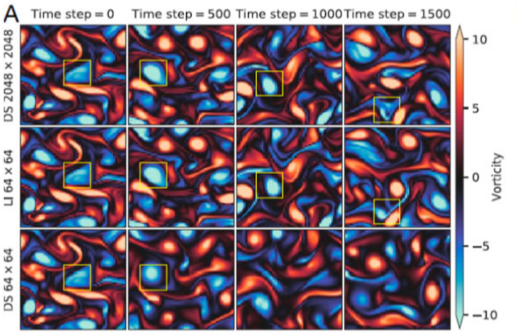


Fig. Kochkov et al. 2021

Incorporating physical knowledge in dynamics models – hybrid systems

Incorporate physical prior knowledge

✓ Accelerate computations

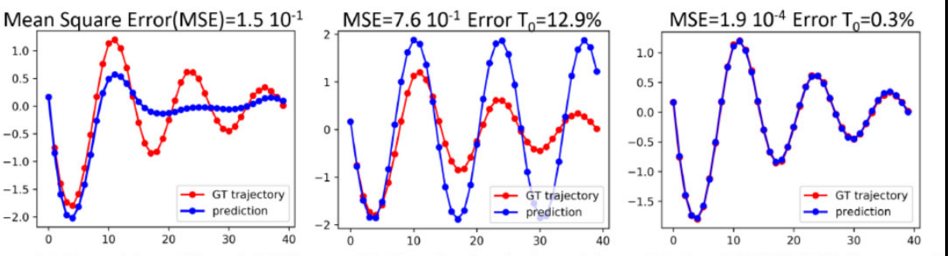
✓ Complement physical models-incomplete physical prior

Complement physical models-incomplete physical prior APHYNITY: Augmenting Physical Models with Deep Networks for Complex Dynamics Forecasting (Yin et al. 2021)

- Context
  - Incomplete background knowledge is available, e.g. PDE that only explains partially the phenomenon
  - Complement the physical model with a statistical component
  - Provide a principled framework to make model based and data based framework cooperate
  - Objective
    - Identify correctly the physical parameters (inverse problem)
    - The NN component should learn to describe the information that cannot be captured by the physics (direct problem)

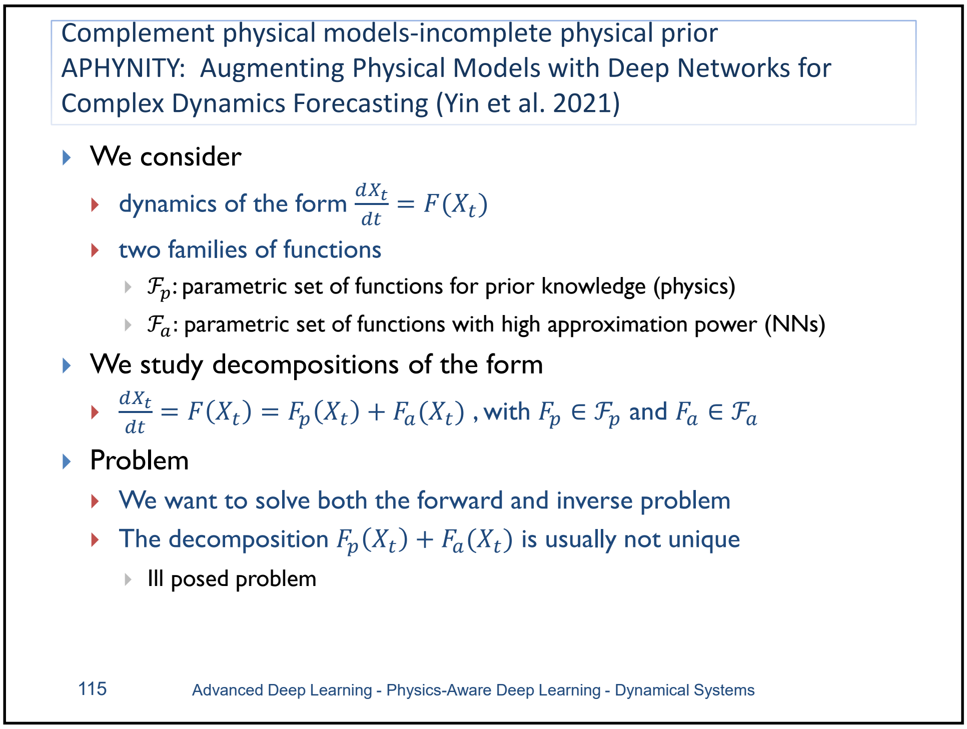
Complement physical models-incomplete physical prior APHYNITY: Augmenting Physical Models with Deep Networks for Complex Dynamics Forecasting (Yin et al. 2021)

- Illustration: damped pendulum



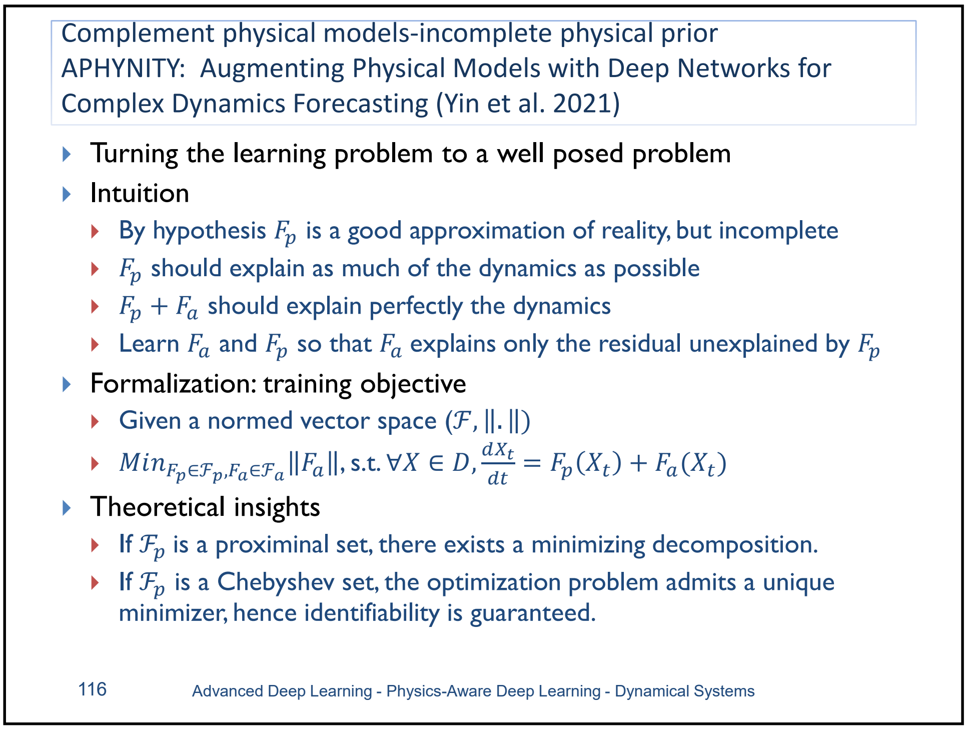
(a) Data-driven Neural ODE (b) Simple physical model (c) Our APHYNITY framework

Figure 1: Predicted dynamics for the damped pendulum vs. ground truth (GT) trajectories $\mathrm{d}^2\theta/\mathrm{d}t^2 + \omega_0^2 \sin\theta + \alpha \mathrm{d}\theta/\mathrm{d}t = 0$. We show that in (a) the data-driven approach (Chen et al., 2018) fails to properly learn the dynamics due to the lack of training data, while in (b) an ideal pendulum cannot take friction into account. The proposed APHYNITY shown in (c) augments the over-simplified physical model in (b) with a data-driven component. APHYNITY improves both forecasting (MSE) and parameter identification (Error T_0) compared to (b).



Complement physical models-incomplete physical prior APHYNITY: Augmenting Physical Models with Deep Networks for Complex Dynamics Forecasting (Yin et al. 2021)

- We consider
  - dynamics of the form $\frac{dX_t}{dt} = F(X_t)$
  - two families of functions
    - $\mathcal{F}_p$: parametric set of functions for prior knowledge (physics)
    - $\mathcal{F}_a$: parametric set of functions with high approximation power (NNs)
- We study decompositions of the form
  - $\frac{dX_t}{dt} = F(X_t) = F_p(X_t) + F_a(X_t)$, with $F_p \in \mathcal{F}_p$ and $F_a \in \mathcal{F}_a$
- Problem
  - We want to solve both the forward and inverse problem
  - The decomposition $F_p(X_t) + F_a(X_t)$ is usually not unique
    - Ill posed problem



Complement physical models-incomplete physical prior APHYNITY: Augmenting Physical Models with Deep Networks for Complex Dynamics Forecasting (Yin et al. 2021)

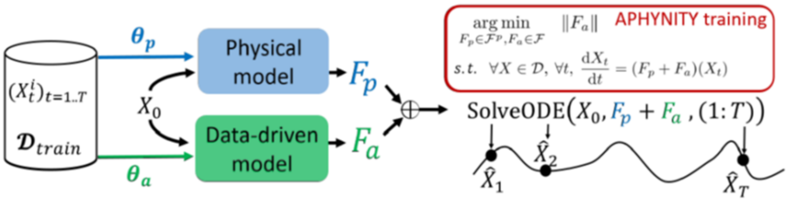
- Turning the learning problem to a well posed problem
- Intuition
  - By hypothesis F_p is a good approximation of reality, but incomplete
  - F_p should explain as much of the dynamics as possible
  - $F_p + F_a$ should explain perfectly the dynamics
  - Learn F_a and F_p so that F_a explains only the residual unexplained by F_p
- Formalization: training objective
  - Given a normed vector space $(\mathcal{F}, ||.||)$
  - $Min_{F_p \in \mathcal{F}_p, F_a \in \mathcal{F}_a} ||F_a||$, s.t. $\forall X \in D, \frac{dX_t}{dt} = F_p(X_t) + F_a(X_t)$
- Theoretical insights
  - If $\mathcal{F}_p$ is a proximinal set, there exists a minimizing decomposition.
  - If $\mathcal{F}_p$ is a Chebyshev set, the optimization problem admits a unique minimizer, hence identifiability is guaranteed.

Complement physical models-incomplete physical prior
APHYNITY: Augmenting Physical Models with Deep Networks for Complex Dynamics Forecasting (Yin et al. 2021)

- Practical instantiation
  - $\mathcal{F}_p$ is parameterized with a differential equation $F_p^{\theta_p}$
  - $\mathcal{F}_a$ is parameterized by a NN $F_a^{\theta_a}$
  - We fit trajectories instead of derivatives $\frac{dX_t}{dt}$
  - The problem is solved in the trajectory space:
    - Suppose available sequences of size n: $(\mathrm{X}_{\mathrm{t}_0}^{\mathrm{i}}, \dots, \mathrm{X}_{\mathrm{t}_\mathrm{n}}^{\mathrm{i}})$
    - Solve:
      - □ $Min_{\{\theta_a,\theta_p\}} \left\|F_a^{\theta_a}\right\| + \lambda \sum_i \sum_{\{k=1\}}^{n} ||\hat{X}_{t_k}^i - X_{\mathrm{t_k}}^{\mathrm{i}}||$
      - □ with prediction $\hat{X}_{t_k}^i$obtained by a differentiable solver (RK4 in practice)
        - □ $\hat{X}_{t_k}^i = \mathrm{X}_{\mathrm{t_{k-1}}}^{\mathrm{i}} + \int_{t_{k-1}}^{t_k} \left(F_p^{\theta_p} + F_a^{\theta_a}\right)\left(\hat{X}^i\ (\tau)\right) d\tau$

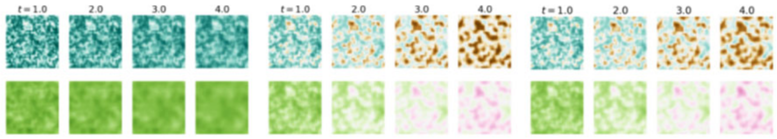
Complement physical models-incomplete physical prior
APHYNITY: Augmenting Physical Models with Deep Networks for Complex Dynamics Forecasting (Yin et al. 2021)

- Summary


θ_p
Physical model
F_p
$(X_t^i)_{t=1..T}$
$\mathcal{D}_{train}$
X_0
Data-driven model
F_a
θ_a
$\underset{F_p\in\mathcal{F}^p, F_a\in\mathcal{F}}{\arg\min} \|F_a\|$
APHYNITY training
$s.t.\ \forall X \in \mathcal{D}, \forall t, \frac{\mathrm{d}X_t}{\mathrm{d}t} = (F_p + F_a)(X_t)$
$\oplus$
$\mathrm{SolveODE}(X_0, F_p + F_a, (1:T))$
$\hat{X}_1$
$\hat{X}_2$
$\hat{X}_T$

Complement physical models-incomplete physical prior APHYNITY: Augmenting Physical Models with Deep Networks for Complex Dynamics Forecasting (Yin et al. 2021)

- Example: reaction diffusion equation
- $\frac{\partial u}{\partial t} = a\Delta u + R_u(u, v, k), \frac{\partial v}{\partial t} = \mathrm{b}\Delta v + R_v(u, v)$
  - a,b, diffusion coefficients; R_u, R_v reaction terms; Δ Laplace operator
  - Background physical knowledge
    - Diffusion with coefficients to be estimated
    - Reaction terms are ignored and shall be estimated by F_a

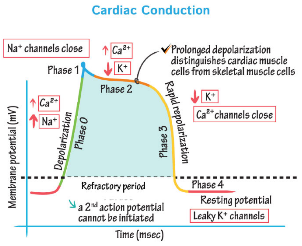


(a) Param PDE (a, b), diffusion-only (b) APHYNITY Param PDE (a, b) (c) Ground truth simulation

Figure 2: Comparison of predictions of two components u (top) and v (bottom) of the reaction-diffusion system. Note that $t = 4$ is largely beyond the dataset horizon ($t = 2.5$).

Combining NN and differential solvers – APHYN-EP Cardiac electrophysiology (Kashtanova et al. 2022)

- Objective
  - Modeling the dynamics of cardiac electrical activity
    - Normal and pathological conditions
  - Variable of interest: Action Potential (mVolts) wave propagation



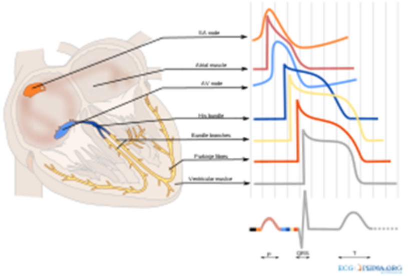


Fig. Wikipedia

Fig. drawittoknowit.com

Complement physical models-incomplete physical prior
Cardiac electrophysiology (Kashtanova et al. 2022)

- Objective
  - Modeling the dynamics of cardiac electrical activity
- Models
  - Complex models
    - e.g. Ten Tusscher-Panfilov 2004
    - # hidden variables and parameters, computationally expensive (43 variables)
  - Surrogate low fidelity models
    - e.g. Mitchell Schaeffer 2003 (2 variables)
      - □ Rapid prototyping, less precise
      - □ Reaction-diffusion model
  - Objective
    - Learn to simulate real data (here TenTusscher) using a combination of low fidelity model and residual neural network – similar to the APHYNITY framework
      - □ Idendify from examples the low fidelity parameters and provide good forecasts of the dynamics
      - □ Limited to the polarization phase

Combining NN and differential solvers – APHYN-EP
Cardiac electrophysiology (Kashtanova et al. 2022)

- In silico data - Example: polarization phase
  - Slab of 2D cardiac tissue of 24x24 elements

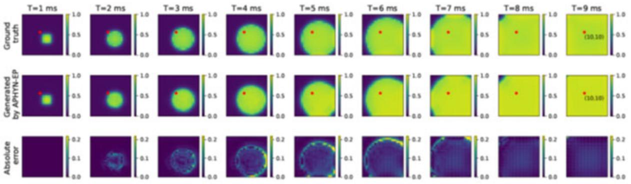


Figure 2: APHYN-EP predicted dynamics for the transmembrane potential diffusion. The figure shows a 9 ms of forecast).

- Ex-vivo data
  - Optical data from swine hearts

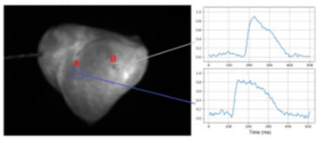


Fig. 2: Example of optical mapping data (tracings of denoised action potential waves) recorded ex vivo in a porcine heart. ROI B represents an ischaemic region characterized by a shorten action potential duration (APD) compared to the normal APD recorded in ROI A.

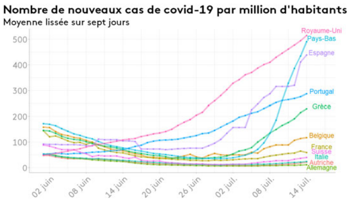
Hybrids and generalization

Incorporating physical knowledge in dynamics models – hybrid systems

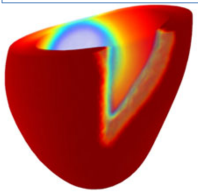
✓ Tackling the generalization problem for dynamical systems

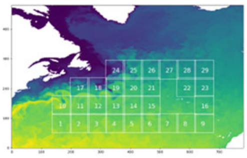
Tackling the generalization problem for dynamical systems Illustrative examples

Modelling epidemics in different countries



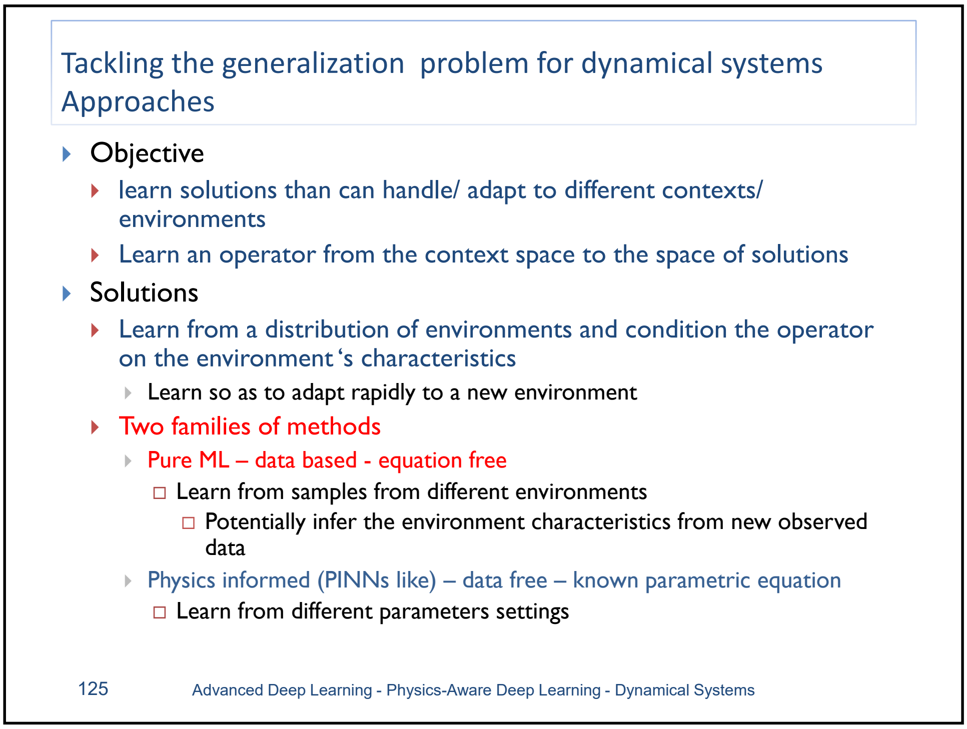
Modeling heart electrical diffusion from different patients, Fig. Fresca et al. 2020





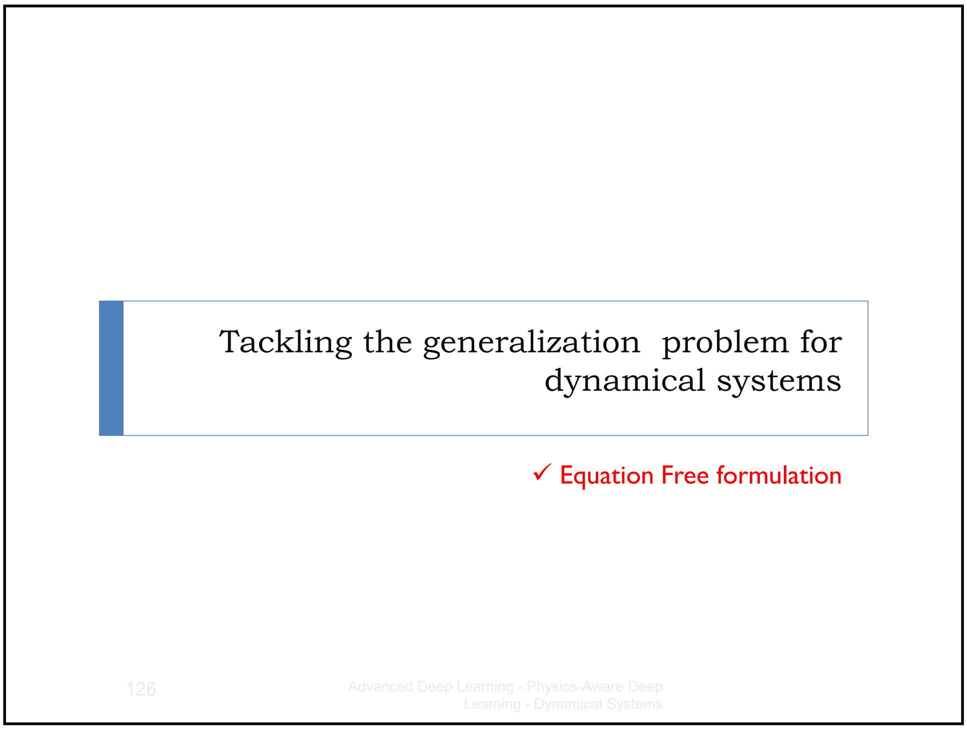
Sub regions extracted for the dataset. Test regions are regions 17 to 20.

Predictions of sea surface temperature from satellite data, Fig. Pajot et al. 2018



Tackling the generalization problem for dynamical systems Approaches

- Objective
  - learn solutions than can handle/ adapt to different contexts/ environments
  - Learn an operator from the context space to the space of solutions
- Solutions
  - Learn from a distribution of environments and condition the operator on the environment ‘s characteristics
    - Learn so as to adapt rapidly to a new environment
  - Two families of methods
    - Pure ML – data based - equation free
      - □ Learn from samples from different environments
        - □ Potentially infer the environment characteristics from new observed data
    - Physics informed (PINNs like) – data free – known parametric equation
      - □ Learn from different parameters settings



Tackling the generalization problem for dynamical systems

✓ Equation Free formulation

Tackling the generalization problem for dynamical systems
Domain Generalization

- Problem setting
  - Assumption: there exists a set of environments $E = \{e^i\}$, each governed by a differential equation $\frac{dx_t^e}{dt} = f_e(x_t^e)$
    - Sharing commonalities e.g. general form of the dynamics (shared parameters θ_c)
    - With specificities, e.g. coefficients of the PDE, initial & boundary conditions, forcings, spatio-temporal domains, etc (Specific parameters θ_e)
- Challenge
  - How to leverage this setting in order to generalize to unseen situations and new environments?

Tackling the generalization problem for dynamical systems
Domain Generalization

- Usual practice in ML (Empirical Risk Minimization)
  - Training dataset: sample environment distribution and for each environmen sample the trajectory distribution
  - Expect this will generalize to new environments
  - This assumes:
    - i.i.d. distribution, dataset large enough to cover the data distribution and represent the diversity of situations
  - Not realistic
- Claim
  - The models should leverage adaptive conditioning to the environment

Tackling the generalization problem for dynamical systems
Domain Generalization (Kassai et al. 2024)

ERM baselines vs environment adaptive conditioning

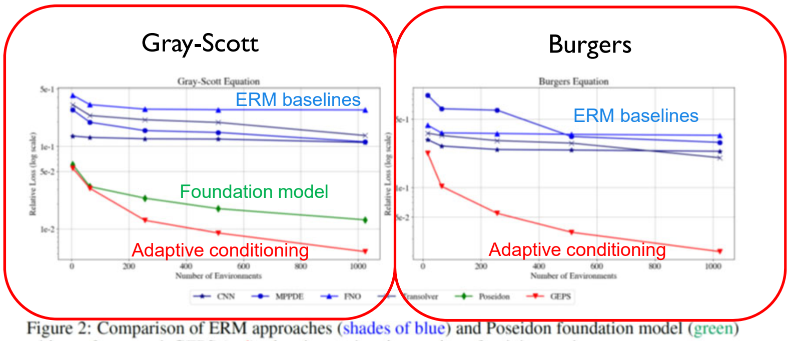
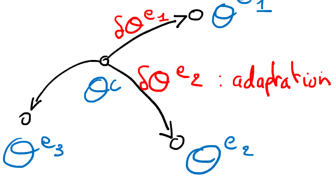


Figure 2: Comparison of ERM approaches (shades of blue) and Poseidon foundation model (green) with our framework GEPS (red) when increasing the number of training environments.

Tackling the generalization problem for dynamical systems
CODA framework (Kirchmeyer et al. 2022, Kassai et al. 2024

- How to – Intuition: Meta-learning for fast adaptation to new environments
  - Training
    - Learn on a sample of the domains' distribution (i.e. different environments)
      - □ $\theta^e = \theta^c + \delta\theta^e$
      - □ θ^c shared parameters across environments, $\delta\theta^e$ environment specific parameters
    - So that it could adapt fast and with a few shots to a new environment
  - Inference: for a new environment fast adaptation with a few samples



Tackling the generalization problem for dynamical systems CODA framework (Kirchmeyer et al. 2022)

- Dynamical function for environment e
  - $f_{\theta^e}, \theta^e = \theta^c + \delta\theta^e$
- Training objective
  - $\min_{\theta^c, \delta\theta^e; e\in E} \sum_{e\in E} \|\delta\theta^e\|^2$ s.t. $\forall x^e \in D^e, \forall t, \frac{dx^e(t)}{dt} = f_{\theta^c+\delta\theta^e}(x^e(t))$
    - θ^c learned over a training set sampled from a family of environments E
    - $\delta\theta^e$ conditioned on environment e
    - Locality constraint $\min_e \|\delta\theta^e\|^2$: θ^e should lie in the neighborhood of θ^c: $\delta\theta^e$ lies on a low dimensional manifold -> fast adaptation to environments

- Lotka-Voltera ODE
- Loss landscape for 3 environments
- Centered on the shared θ^c
- Local min θ^e indicated by arrow

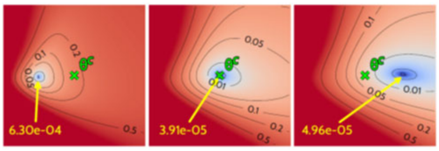
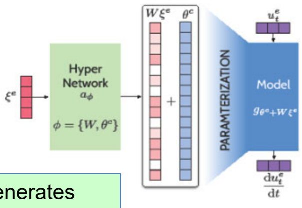


Figure 1. CoDA's loss landscape centered in θ^c, marked with ×, for 3 environments on the *Lotka-Volterra* ODE. Loss values are projected onto subspace $\mathcal{W}$, with $d_\xi = 2$. $\forall e$, → points to the local optimum $\theta^{e\star}$ with loss value reported in yellow.

Tackling the generalization problem for dynamical systems CODA framework (Kirchmeyer et al. 2022)

- Conditioning to an environment
  - $\theta^e = \theta^c + \delta\theta^e$
  - Environment specific parameters: Implementation via a hypernetwork
    - $\delta\theta^e = \mathrm{W}\xi^e$with W weight matrix and ξ^elearned code
    - Code $\xi^e$is infered from a few observations for each new environment
  - θ^c and W are shared parameters learned on the training set
- Inference: for a new environment, learn code ξ^e from a few observations and infer θ^e

(1) Environment encoding ξ^e(learned)



Neural network predictor: forecast next frames via numerical integration

(2) Hypernetwork: generates the parameters of the neural network predictor, conditionned on the environment ξ^e

Tackling the generalization problem for dynamical systems CODA framework (Kirchmeyer et al. 2022)

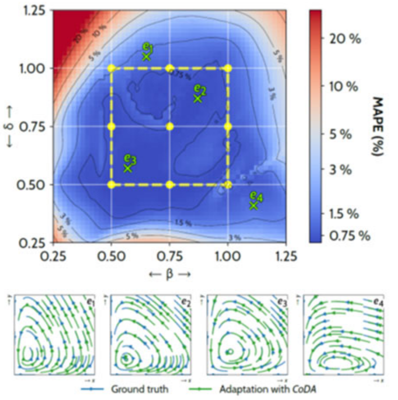


Figure 2. Adaptation results with CoDA-ℓ_1 on LV. Parameters (β, δ) are sampled in $[0.25, 1.25]^2$ on a 51×51 uniform grid, leading to 2601 adaptation environments $\mathcal{E}_{\text{ad}}$. ● are training environments $\mathcal{E}_{\text{tr}}$. We report MAPE (↓) across $\mathcal{E}_{\text{ad}}$ (Top). On the bottom, we choose four of them (×, e_1–e_4), to show the ground-truth (blue) and predicted (green) phase space portraits. x, y are respectively the quantity of prey and predator in the system in Eq. (15).

Lotka-Volterra (LV, Lotka, 1925) The system describes the interaction between a prey-predator pair in an ecosystem, formalized into the following ODE:

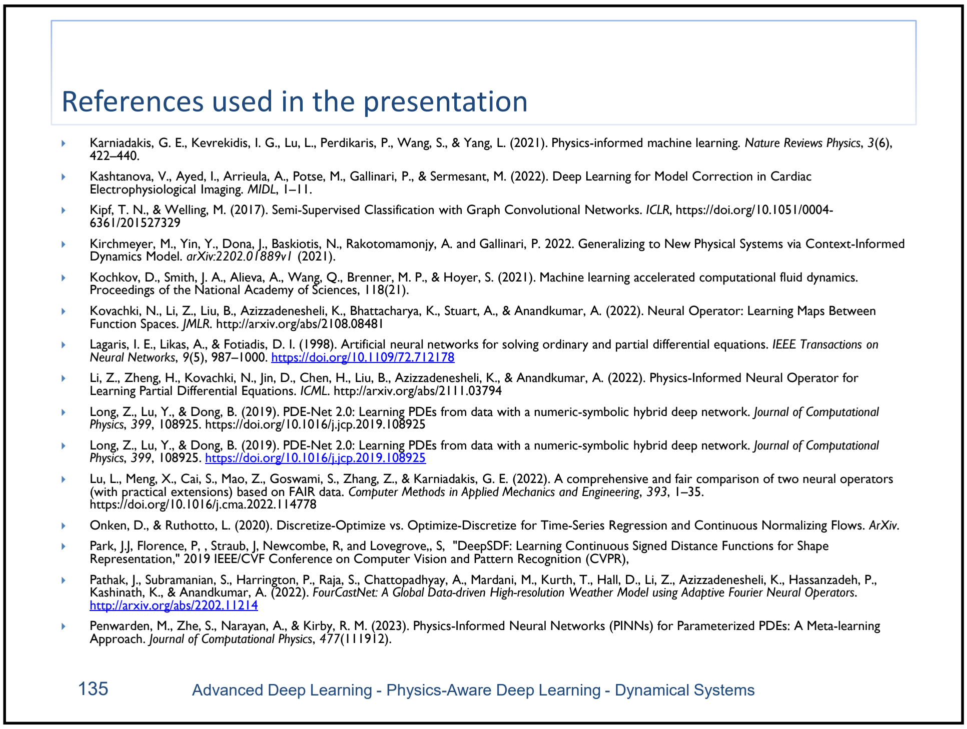
$$\frac{\mathrm{d}x}{\mathrm{d}t} = \alpha x - \beta xy \qquad (15)$$
$$\frac{\mathrm{d}y}{\mathrm{d}t} = \delta xy - \gamma y$$

where x, y are respectively the quantity of the prey and the predator, $\alpha, \beta, \delta, \gamma$ define how two species interact.

- Four parameters, two fixed (α, γ) and two (β, δ) change accross environments
- Training on 9 environments (yellow)
- Top: Evaluation on 2600 new environments
- Bottom: phase portraits for 4 new environments $e_1$to e_4
  - **Blue trajectories: ground truth**
  - **Green trajectories: predicted**

References used in the presentation

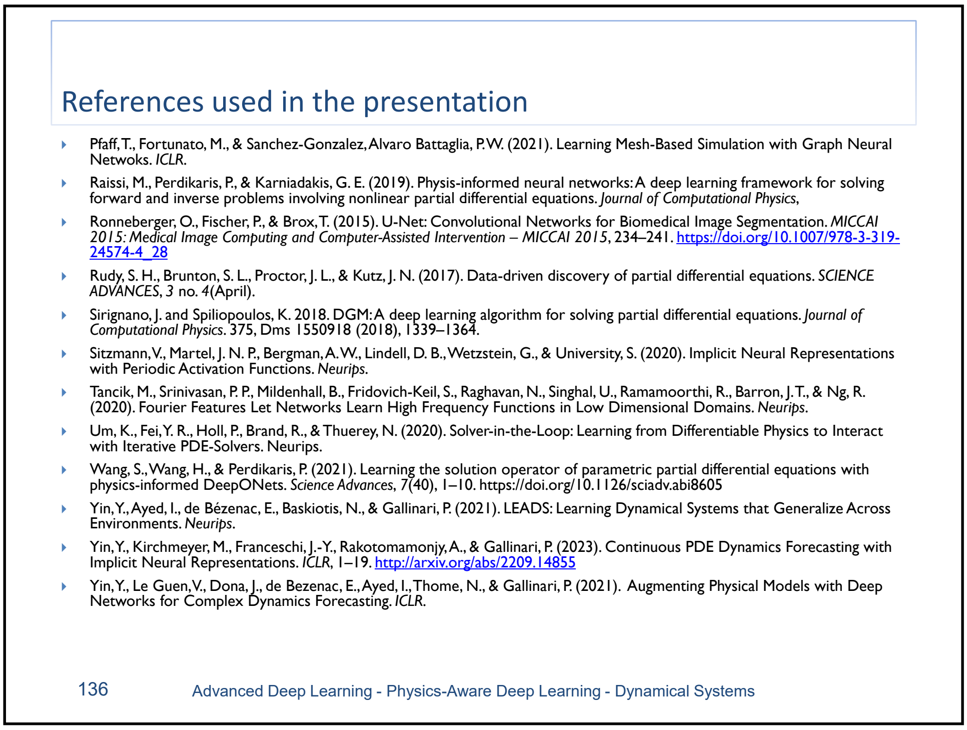
- Blogs on Neural ODE
  - https://ayandas.me/blog-tut/2020/03/20/neural-ode.html
  - https://www.depthfirstlearning.com/2019/NeuralODEs
- Ayed I., de Bezenac E., Pajot A., Brajard J. , Gallinari P. , (2019), Learning Dynamical Systems from Partial Observations, arXiv:1902.11136
- Badrinarayanan, V., Kendall, A., & Cipolla, R. (2017). SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *39*(12), 2481–2495. https://doi.org/10.1109/TPAMI.2016.2644615
- Belbute-Peres, F. de A., Economon, T. D., & Kolter, J. Z. (2020). Combining Differentiable PDE Solvers and Graph Neural Networks for Fluid Flow Prediction. *ICML*.
- Brandstetter, J., Worrall, D. E., & Welling, M. (2022). Message Passing Neural PDE Solvers. *ICLR*.
- Chen, R.T.Q., Rubanova, Y., Bettencourt, J., and Duvenaud, D. 2018. Neural Ordinary Differential Equations. *NIPS*.
- Chen, T, Chen, H. (1995) Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems. IEEE Trans. Neural Netw. 6, 911–917 (1995).
- Chen, Y., Dong, B., & Xu, J. (2022). Meta-MgNet: Meta Multigrid Networks for Solving Parameterized Partial Differential Equations. *Journal of Computational Physics*, *455*. http://arxiv.org/abs/2010.14088
- de Bezenac, E., Pajot, A., & Gallinari, P. (2018). Deep Learning For Physical Processes: Incorporating Prior Scientific Knowledge. In *ICLR, also in Journal of Statistical Mechanics: Theory and Experiment, 2019.*
- Dona, J., Dechelle, M., Gallinari, P., & Levy, M. (2022). Constrained Physical-Statistical Models for Dynamical System Identification and Prediction. *ICLR*.
- Dupont, E., Kim, H., Eslami, S. M. A., Rezende, D., & Rosenbaum, D. (2022). From data to functa: Your data point is a function and you can treat it like one. *ICML*. http://arxiv.org/abs/2201.12204
- Fathony, R., Sahu, A. K., Willmott, D., & Kolter, J. Z. (2021). Multiplicative Filter Networks. *ICLR*, 1–10.
- Fresca, S., Manzoni, A., Dedè, L., & Quarteroni, A. (2020). Deep learning-based reduced order models in cardiac electrophysiology. *PLoS ONE*, *15*(10 October), 1–32. https://doi.org/10.1371/journal.pone.0239416
- Gholami, A., Keutzer, K., & Biros, G. (2019). ANODE: Unconditionally accurate memory-efficient gradients for neural ODEs. *IJCAI International Joint Conference on Artificial Intelligence*, 730–736. https://doi.org/10.24963/ijcai.2019/103
- Haber, E. and Ruthotto, L. 2018. Stable architectures for deep neural networks. *Inverse Problems*. 34, 1 (2018).
- Hamilton, W. L. (2020). Graph Representation Learning: Foundations, Methods, Applications and Systems. In *Graph Representation Learning*. Springer. https://doi.org/10.1145/3447548.3470824
- Huang, X., Ye, Z., Liu, H., Shi, B., Wang, Z., Yang, K., Li, Y., Weng, B., Wang, M., Chu, H., Zhou, J., Yu, F., Hua, B., Chen, L., & Dong, B. (2022). Meta-Auto-Decoder for Solving Parametric Partial Differential Equations. *Neurips*, *1*, 1–20. http://arxiv.org/abs/2111.08823



References used in the presentation

- Karniadakis, G. E., Kevrekidis, I. G., Lu, L., Perdikaris, P., Wang, S., & Yang, L. (2021). Physics-informed machine learning. *Nature Reviews Physics*, *3*(6), 422–440.
- Kashtanova, V., Ayed, I., Arrieula, A., Potse, M., Gallinari, P., & Sermesant, M. (2022). Deep Learning for Model Correction in Cardiac Electrophysiological Imaging. *MIDL*, 1–11.
- Kipf, T. N., & Welling, M. (2017). Semi-Supervised Classification with Graph Convolutional Networks. *ICLR*, https://doi.org/10.1051/0004-6361/201527329
- Kirchmeyer, M., Yin, Y., Dona, J., Baskiotis, N., Rakotomamonjy, A. and Gallinari, P. 2022. Generalizing to New Physical Systems via Context-Informed Dynamics Model. *arXiv:2202.01889v1* (2021).
- Kochkov, D., Smith, J. A., Alieva, A., Wang, Q., Brenner, M. P., & Hoyer, S. (2021). Machine learning accelerated computational fluid dynamics. Proceedings of the National Academy of Sciences, 118(21).
- Kovachki, N., Li, Z., Liu, B., Azizzadenesheli, K., Bhattacharya, K., Stuart, A., & Anandkumar, A. (2022). Neural Operator: Learning Maps Between Function Spaces. *JMLR*. http://arxiv.org/abs/2108.08481
- Lagaris, I. E., Likas, A., & Fotiadis, D. I. (1998). Artificial neural networks for solving ordinary and partial differential equations. *IEEE Transactions on Neural Networks*, *9*(5), 987–1000. https://doi.org/10.1109/72.712178
- Li, Z., Zheng, H., Kovachki, N., Jin, D., Chen, H., Liu, B., Azizzadenesheli, K., & Anandkumar, A. (2022). Physics-Informed Neural Operator for Learning Partial Differential Equations. *ICML*. http://arxiv.org/abs/2111.03794
- Long, Z., Lu, Y., & Dong, B. (2019). PDE-Net 2.0: Learning PDEs from data with a numeric-symbolic hybrid deep network. *Journal of Computational Physics*, *399*, 108925. https://doi.org/10.1016/j.jcp.2019.108925
- Long, Z., Lu, Y., & Dong, B. (2019). PDE-Net 2.0: Learning PDEs from data with a numeric-symbolic hybrid deep network. *Journal of Computational Physics*, *399*, 108925. https://doi.org/10.1016/j.jcp.2019.108925
- Lu, L., Meng, X., Cai, S., Mao, Z., Goswami, S., Zhang, Z., & Karniadakis, G. E. (2022). A comprehensive and fair comparison of two neural operators (with practical extensions) based on FAIR data. *Computer Methods in Applied Mechanics and Engineering*, *393*, 1–35. https://doi.org/10.1016/j.cma.2022.114778
- Onken, D., & Ruthotto, L. (2020). Discretize-Optimize vs. Optimize-Discretize for Time-Series Regression and Continuous Normalizing Flows. *ArXiv*.
- Park, J.J, Florence, P, , Straub, J, Newcombe, R, and Lovegrove,, S, "DeepSDF: Learning Continuous Signed Distance Functions for Shape Representation," 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR),
- Pathak, J., Subramanian, S., Harrington, P., Raja, S., Chattopadhyay, A., Mardani, M., Kurth, T., Hall, D., Li, Z., Azizzadenesheli, K., Hassanzadeh, P., Kashinath, K., & Anandkumar, A. (2022). *FourCastNet: A Global Data-driven High-resolution Weather Model using Adaptive Fourier Neural Operators*. http://arxiv.org/abs/2202.11214
- Penwarden, M., Zhe, S., Narayan, A., & Kirby, R. M. (2023). Physics-Informed Neural Networks (PINNs) for Parameterized PDEs: A Meta-learning Approach. *Journal of Computational Physics*, *477*(111912).

135 Advanced Deep Learning - Physics-Aware Deep Learning - Dynamical Systems



References used in the presentation

- Pfaff, T., Fortunato, M., & Sanchez-Gonzalez, Alvaro Battaglia, P.W. (2021). Learning Mesh-Based Simulation with Graph Neural Netwoks. *ICLR*.
- Raissi, M., Perdikaris, P., & Karniadakis, G. E. (2019). Physis-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*,
- Ronneberger, O., Fischer, P., & Brox, T. (2015). U-Net: Convolutional Networks for Biomedical Image Segmentation. *MICCAI 2015: Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, 234–241. https://doi.org/10.1007/978-3-319-24574-4_28
- Rudy, S. H., Brunton, S. L., Proctor, J. L., & Kutz, J. N. (2017). Data-driven discovery of partial differential equations. *SCIENCE ADVANCES*, *3* no. *4*(April).
- Sirignano, J. and Spiliopoulos, K. 2018. DGM: A deep learning algorithm for solving partial differential equations. *Journal of Computational Physics*. 375, Dms 1550918 (2018), 1339–1364.
- Sitzmann, V., Martel, J. N. P., Bergman, A. W., Lindell, D. B., Wetzstein, G., & University, S. (2020). Implicit Neural Representations with Periodic Activation Functions. *Neurips*.
- Tancik, M., Srinivasan, P. P., Mildenhall, B., Fridovich-Keil, S., Raghavan, N., Singhal, U., Ramamoorthi, R., Barron, J. T., & Ng, R. (2020). Fourier Features Let Networks Learn High Frequency Functions in Low Dimensional Domains. *Neurips*.
- Um, K., Fei, Y. R., Holl, P., Brand, R., & Thuerey, N. (2020). Solver-in-the-Loop: Learning from Differentiable Physics to Interact with Iterative PDE-Solvers. Neurips.
- Wang, S., Wang, H., & Perdikaris, P. (2021). Learning the solution operator of parametric partial differential equations with physics-informed DeepONets. *Science Advances*, *7*(40), 1–10. https://doi.org/10.1126/sciadv.abi8605
- Yin, Y., Ayed, I., de Bézenac, E., Baskiotis, N., & Gallinari, P. (2021). LEADS: Learning Dynamical Systems that Generalize Across Environments. *Neurips*.
- Yin, Y., Kirchmeyer, M., Franceschi, J.-Y., Rakotomamonjy, A., & Gallinari, P. (2023). Continuous PDE Dynamics Forecasting with Implicit Neural Representations. *ICLR*, 1–19. http://arxiv.org/abs/2209.14855
- Yin, Y., Le Guen, V., Dona, J., de Bezenac, E., Ayed, I., Thome, N., & Gallinari, P. (2021). Augmenting Physical Models with Deep Networks for Complex Dynamics Forecasting. *ICLR*.

136 Advanced Deep Learning - Physics-Aware Deep Learning - Dynamical Systems