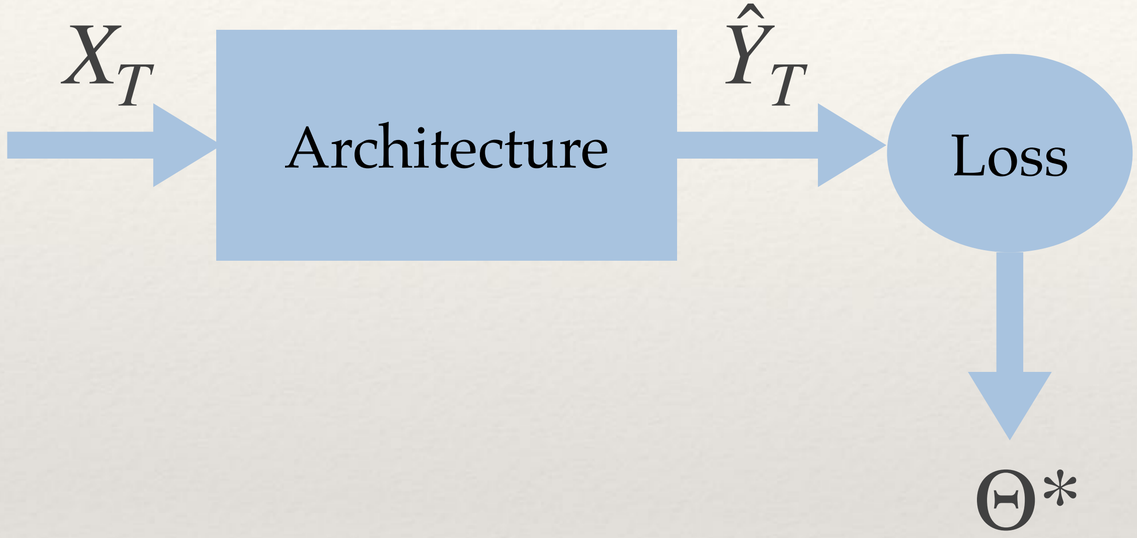
Physically-based Machine Learning: Overview

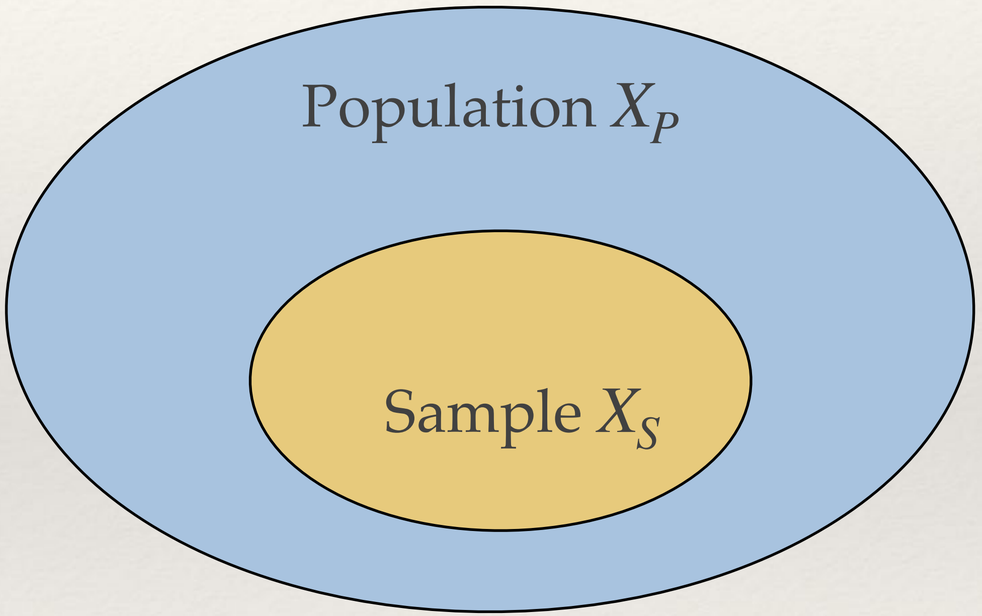
Gordon Erlebacher
October 12, 2022

What is Machine Learning?

Machine Learning is the automation of Inductive Inference
(Ulrike von Luxborg, U. Tübingen)

Minimize the loss





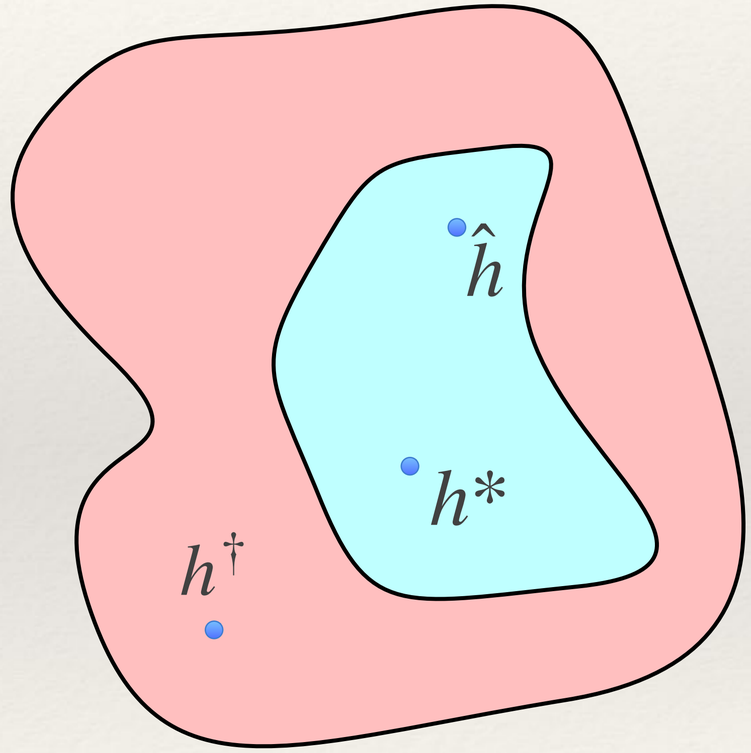
- $\Theta^* = \arg\min \mathscr{L}(Y_P, \Theta)$ is the solution to the stated problem

Ingredients of Machine Learning

- **Training data** $x_i,\ i = 0,\ N-1$
  - Scalars, vectors, tables, images, audio clips, abstract objects
- **Output data** $y_i, i = 0,\ N-1$
- An **architecture**:
  - mapping $x_i \rightarrow y_i$: $y_i = F(x_i;\ \Theta)$ parameters Θ
- A **loss function** $\mathscr{L}(X_T; \Theta)$

Model (hypothesis) Space

- $\hat{h}$: Estimated (calculate) model
  - local minimum
- h^* : Optimal model
  - given the specified family of models parameterized by Θ
- $h^\dagger$: True model
  - within a given function space (i.e., L^2 functions)



Traditional Applications of Machine Learning

- **Large Data Models**
  - Image Classification models
    - CNN + variations
  - Text to image
    - Dalle-2, Stable-Diffusion, Mid
  - Language models
    - Text generation, translation

What is Physics-Based Modeling?

PBM predicts using the tools of physics and mathematics

- Symmetries
- Invariances
  - translation, rotation, scale
- Equivariance
- Constraints
- Optimization
- Isotropy, Homogeneity

Given $x_i \in X \subset \mathbb{R}^m$,
predict $y_i \in Y \subset \mathbb{R}^n$

What is Physics-Based Modeling?

PBM predicts using the tools of physics and mathematics

- Symmetries
- Invariances
  - translation, rotation, scale
- Equivariance
- Constraints
- Optimization
- Isotropy, Homogeneity

Given $x_i \in X \subset \mathbb{R}^m$,
predict $y_i \in Y \subset \mathbb{R}^n$

High generalizability
- PDEs, ODEs
- Interpretable

Types of Physical Constraints

- Translation Invariance ==> CNN
- Rotation Invariance
- Equivariance ==> GNN
- Invariance ==> GNN
- Temporal homogeneity ==> shared weights in FFNN with recursion
- Boundary and Initial conditions
- Experimental measurements
- Divergent-free flow
- Positivity

Examples of Physical Models

- Navier-Stokes equations
  - viscous fluids, transition, turbulence, sub/trans/supersonic flows
- Transport equations
  - Geology, biological processes
- Diffusion equations
  - Heat conduction
- Electromagnetic equations
- Special/General Relativity

Examples of Physical Models

- Navier-Stokes equations

These equations applicable over a wide range of phenomena

  - viscous fluids, transition, turbulence, sub/trans/supersonic flows
- Transport equations
  - Geology, biological processes
- Diffusion equations
  - Heat conduction
- Electromagnetic equations
- Special/General Relativity

Examples of Physical Models

- Navier-Stokes equations
  - viscous fluids, transition, turbulence, sub/trans/supersonic flows
- Transport equations
  - Geology, biological processes
- Diffusion equations
  - Heat conduction
- Electromagnetic equations
- Special/General Relativity

These equations applicable over a wide range of phenomena

(Initial and) Boundary conditions $\Longrightarrow$ Solution in space (and time)

Examples of Physical Models

- Navier-Stokes equations
  - viscous fluids, transition, turbulence, sub/trans/supersonic flows
- Transport equations
  - Geology, biological processes
- Diffusion equations
  - Heat conduction
- Electromagnetic equations
- Special/General Relativity

These equations applicable over a wide range of phenomena

(Initial and) Boundary conditions $\Longrightarrow$ Solution in space (and time)

They generalize well:
- New geometry
- new B-C
- new I.C.

Examples of Physical Models

- Navier-Stokes equations
  - viscous fluids, transition, turbulence, sub/trans/supersonic flows
- Transport equations
  - Geology, biological processes
- Diffusion equations
  - Heat conduction
- Electromagnetic equations
- Special/General Relativity

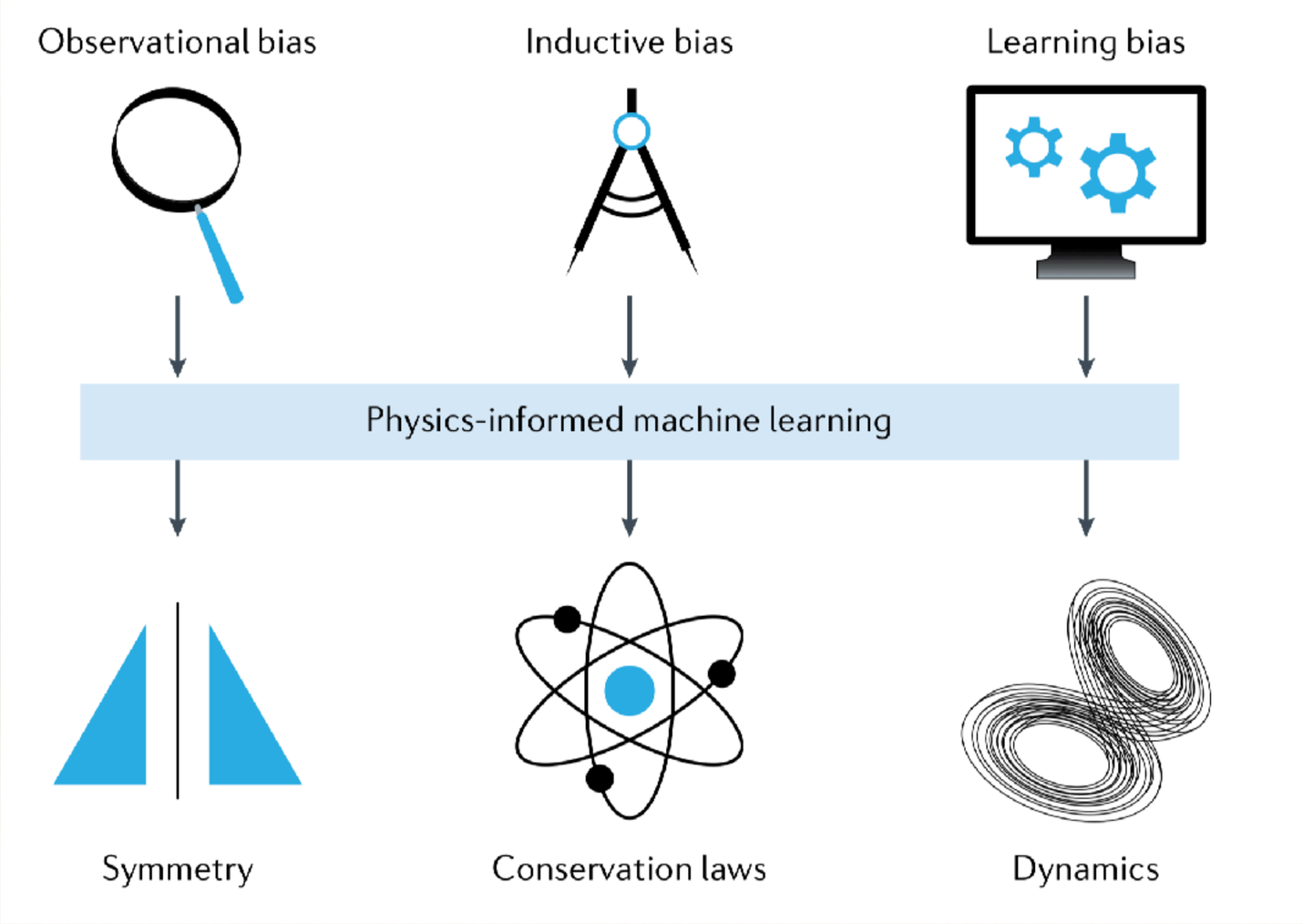
These equations applicable over a wide range of phenomena

(Initial and) Boundary conditions $\Longrightarrow$ Solution in space (and time)

They generalize well:
- New geometry
- new B-C
- new I.C.

High-quality data is often hard to acquire

Physics-Informed Machine Learning

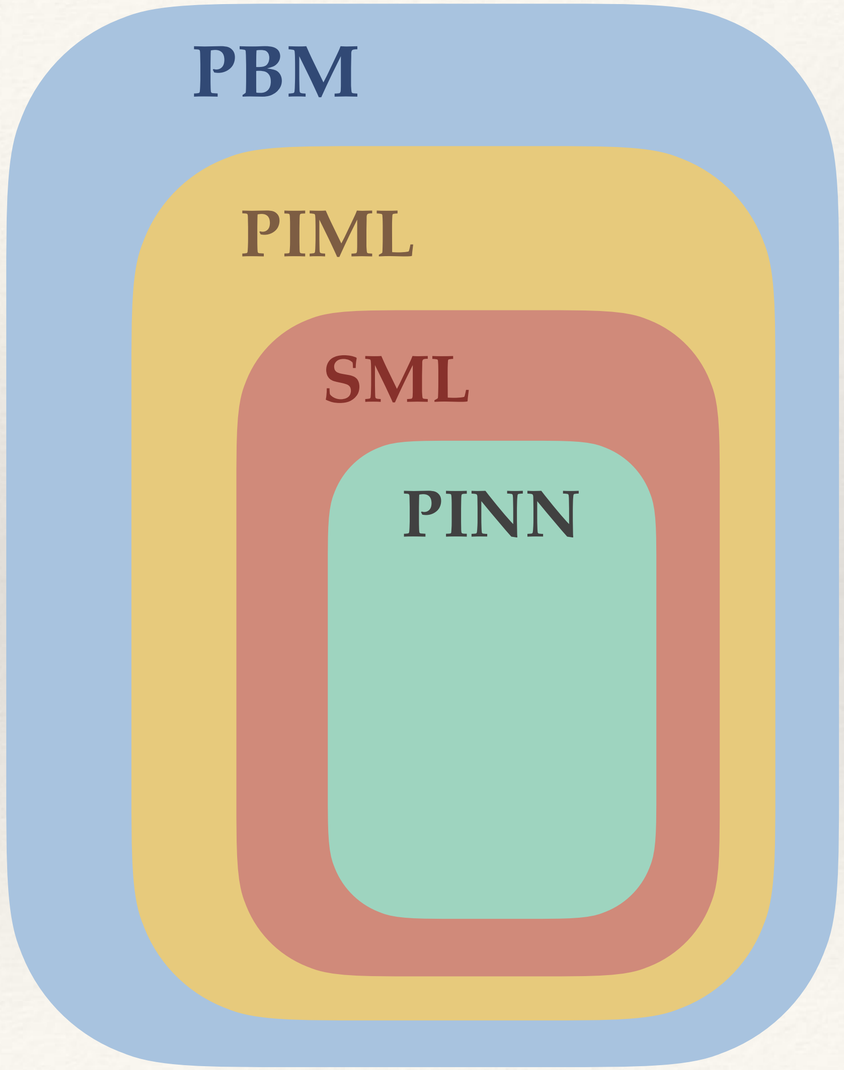


Introduce elements of
Physics into
Machine Learning Algorithms
to produce strong
inductive bias

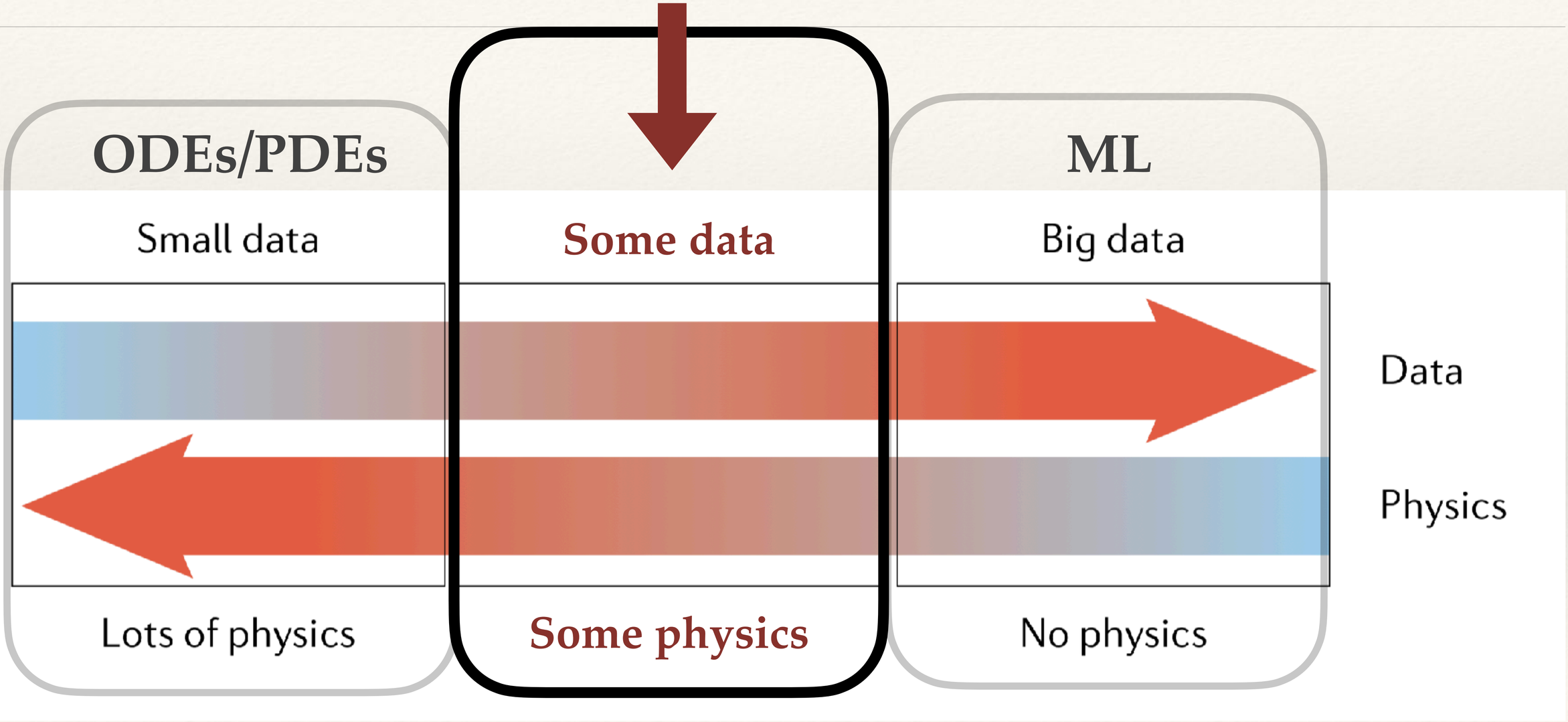
From: Physics-informed machine learning, by Karniadakis metal, Nature Reviews, 2021

High-Level Taxonomy

- Physics-Based Modeling (PBM)
- Physics-Informed Machine Learning (PIML)
- Scientific Machine Learning (SML)
- Physics-Informed Neural Networks (PINN)



SciML, PIML, PINN



Physics-Based Preprocessing

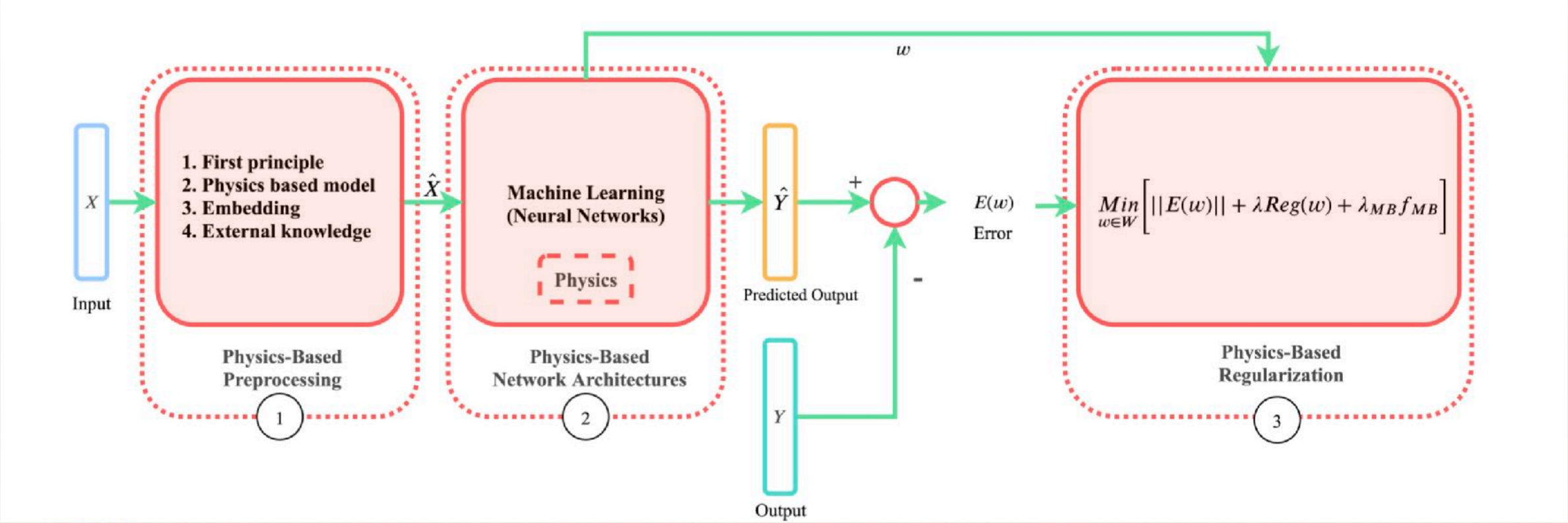
- Compressed Sensing
- Empirical Mode Decomposition
- Fast Fourier Transform
- Wavelet Transform
- Principal Component Analysis
- Data Augmentation (flips, rotations, etc.)

Objectives
(1) Reduce data
(2) Impose symmetries
(3) Impose constraints
(4) Add auxiliary features
(5) Transform to more interpretable space

Inductive Bias of Physics

- **Where is physics incorporated into the algorithms?**
  - Architectural choices
  - Loss function
  - Input structure
  - Choice of attributes

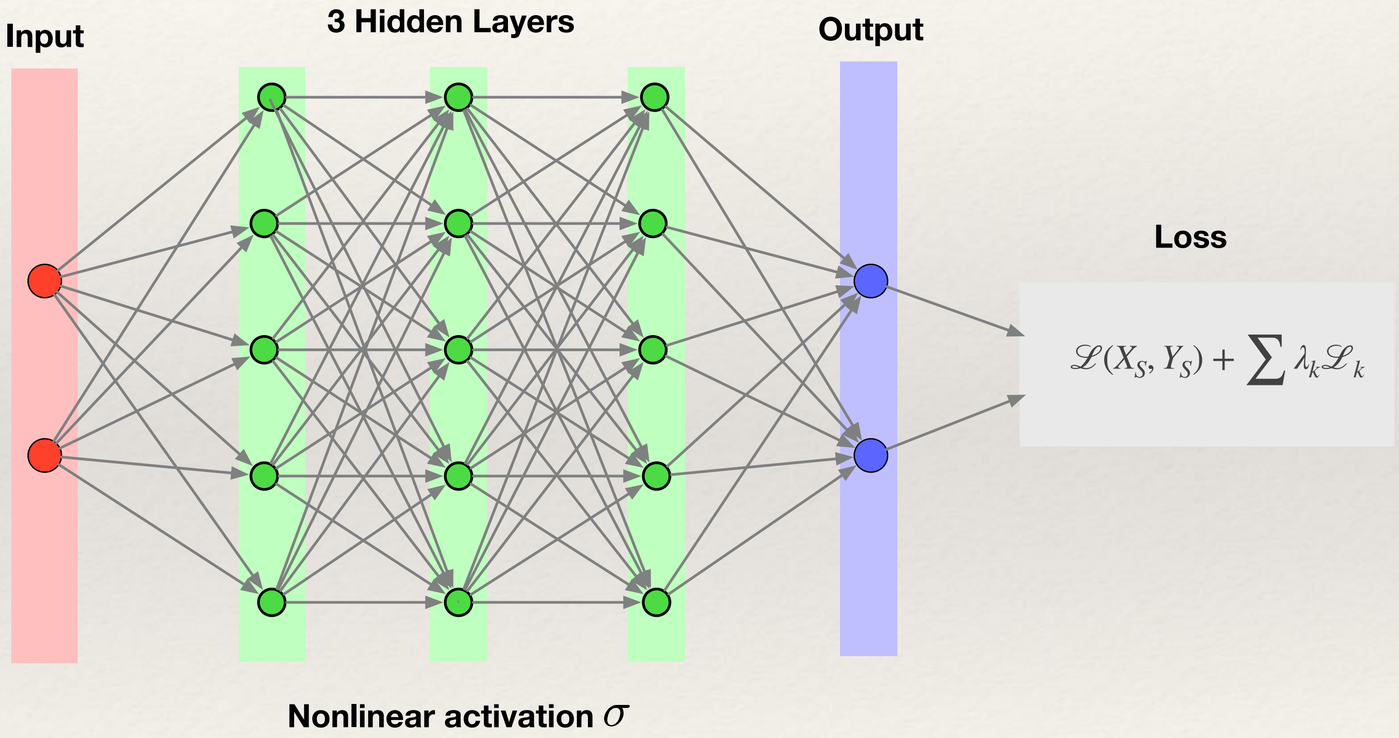
Introducing Knowledge / Physics



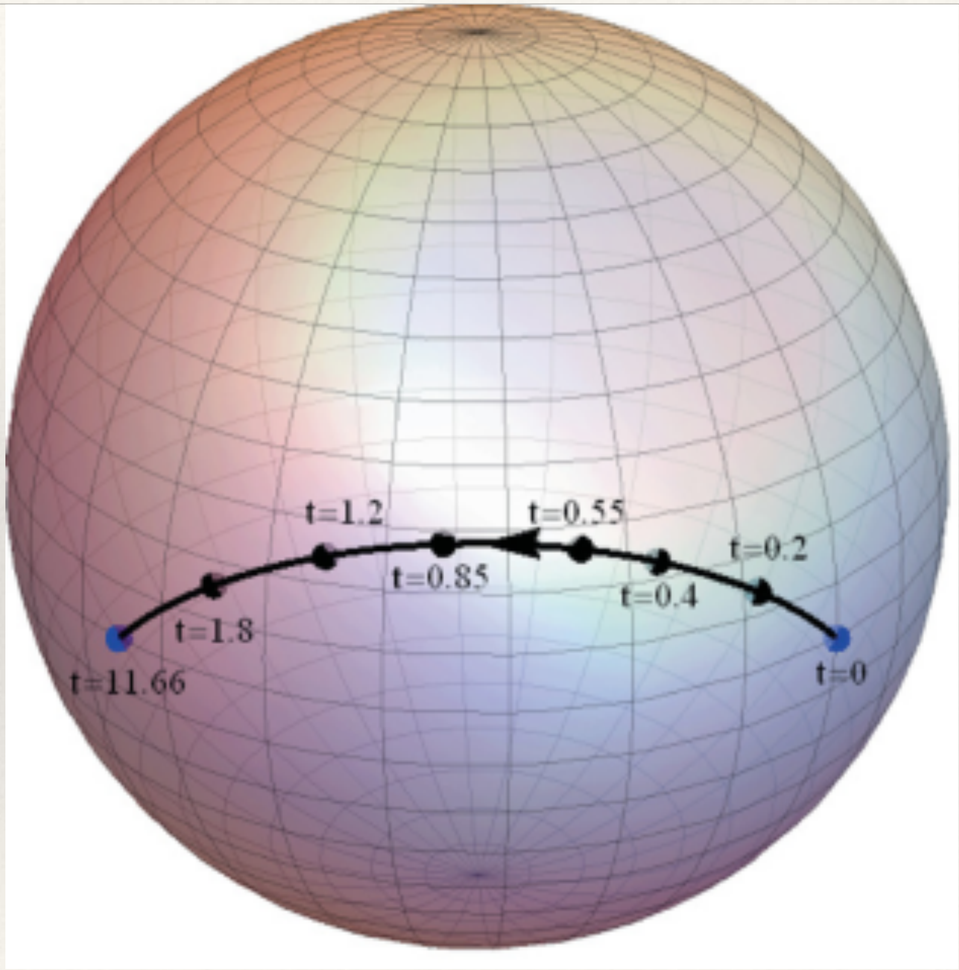
Data **Architecture** **Loss**

Prototypical Multilayer Perceptron

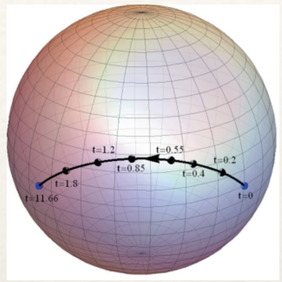
- Fully-Connected
- Feed-Forward
- Input $x_i \in X_S \subset X \subset \mathbb{R}^m$
- Output $y_i \in Y_S \subset Y \subset \mathbb{R}^n$
- Loss $\mathscr{L}(X_S, Y_S) + \sum \lambda_k \mathscr{L}_k$



Regression on a Sphere $\mathcal{S}$ of unit radius



Regression on a Sphere $\mathcal{S}$ of unit radius



- Consider N sample points (data x_i, y_i , label z_i) on $\mathcal{S}$:

$$\Gamma_N = \{x_i, y_i, z_i, i = (1, \cdots, N)\} \subset \mathcal{S}$$

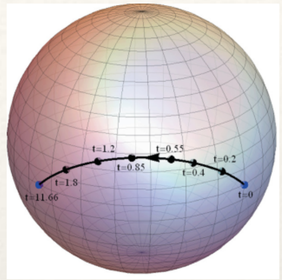
- Objective: fit a curve through Γ_N

$$\mathcal{L}_S = \sum_i (NN(x_i, y_i) - z_i^2) + \lambda \sum_i (x_i^2 + y_i^2 + z_i^2 - 1)^2$$

Curve fit — Constraint

- With large enough N and a FF network, predictions will fall *close* (perhaps *very close)* to the sphere

Regression on a Sphere $\mathcal{S}$ of unit radius



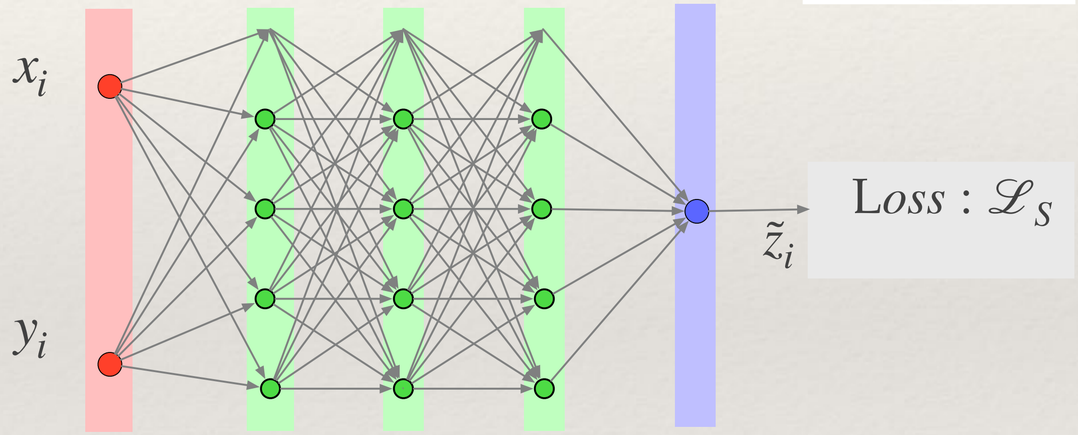
- Consider N sample points (data x_i, y_i , label z_i) on $\mathcal{S}$:

$$\Gamma_N = \{x_i, y_i, z_i, i = (1, \cdots, N)\} \subset \mathcal{S}$$

- Objective: fit a curve through Γ_N

$$\mathcal{L}_S = \sum_i (NN(x_i, y_i) - z_i^2) + \lambda \sum_i (x_i^2 + y_i^2 + \tilde{z}_i^2 - 1)^2$$

Curve fit — Constraint



Neural Network: NN

$$\tilde{z}_i = NN(x_i, y_i)$$

- With large enough N and a FF network, predictions will fall *close* (perhaps *very close)* to the sphere

Regression on Sphere $\mathcal{S}$: preprocess data

- Change of Coordinate$(x, y, z) \to (\xi, \eta)$

$$\begin{aligned} x_i &= \cos\theta_i \cos\phi_i \\ y_i &= \cos\theta_i \sin\phi_i \\ z_i &= \sin\theta_i \end{aligned} \implies \theta_i = \arctan \frac{z_i}{\sqrt{x_i^2 + y_i^2}}, \quad \phi_i = \arctan \frac{y_i}{x_i}$$

- Objective: fit a curve through Γ_N

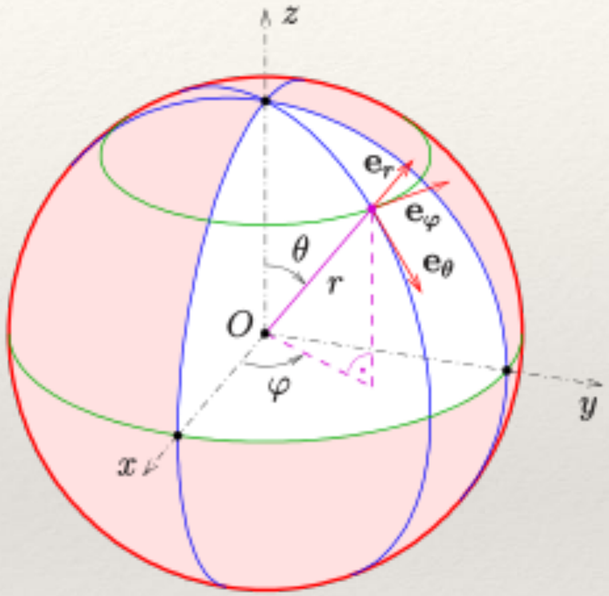
$$\mathcal{L}_S = \sum_i (NN(\phi_i) - \tilde{\theta}_i)^2$$

- Predictions fall *on the sphere* for all N! Less network *capacity* (parameters) is needed.

Regression on Sphere $\mathcal{S}$: preprocess data

- Change of Coordinate$(x, y, z) \rightarrow (\xi, \eta)$

$$
\begin{aligned}
x_i &= \cos\theta_i \cos\phi_i \\
y_i &= \cos\theta_i \sin\phi_i \\
z_i &= \sin\theta_i
\end{aligned}
\quad \Longrightarrow \quad
\theta_i = \arctan \frac{z_i}{\sqrt{x_i^2 + y_i^2}}, \quad \phi_i = \arctan \frac{y_i}{x_i}
$$

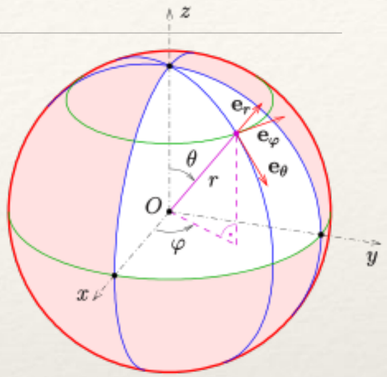


- Objective: fit a curve through Γ_N

$$
\mathcal{L}_S = \sum_i (NN(\phi_i) - \tilde{\theta}_i)^2
$$

- Predictions fall *on the sphere* for all N! Less network *capacity* (parameters) is needed.

Regression on Sphere $\mathcal{S}$: preprocess data



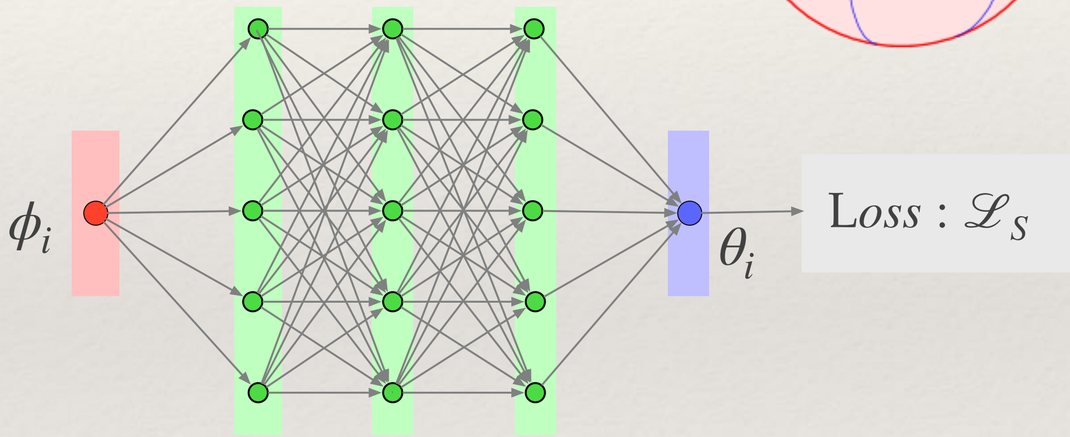
- Change of Coordinate$(x, y, z) \to (\xi, \eta)$

$$x_i = \cos\theta_i \cos\phi_i$$
$$y_i = \cos\theta_i \sin\phi_i \implies \theta_i = \arctan\frac{z_i}{\sqrt{x_i^2 + y_i^2}}, \quad \phi_i = \arctan\frac{y_i}{x_i}$$
$$z_i = \sin\theta_i$$

- Objective: fit a curve through Γ_N

$$\mathscr{L}_S = \sum_i (NN(\phi_i) - \tilde{\theta}_i)^2$$

- Predictions fall *on the sphere* for all N! Less network *capacity* (parameters) is needed.

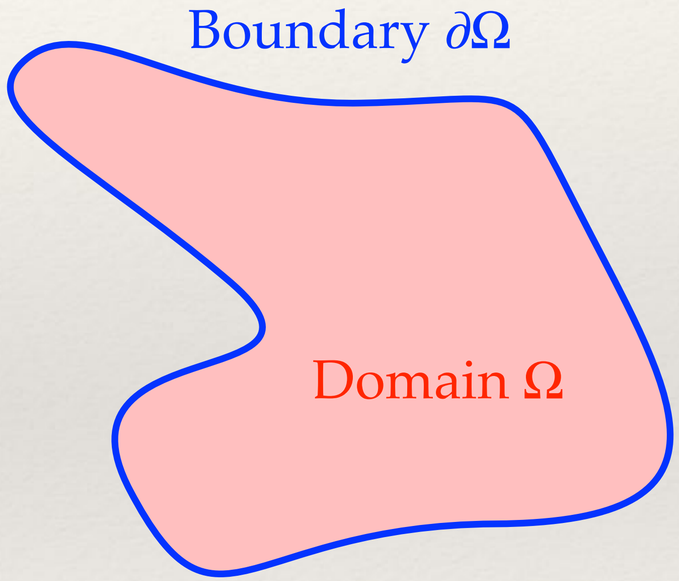


Neural Network: NN

$$\tilde{y}_i = NN(x_i)$$

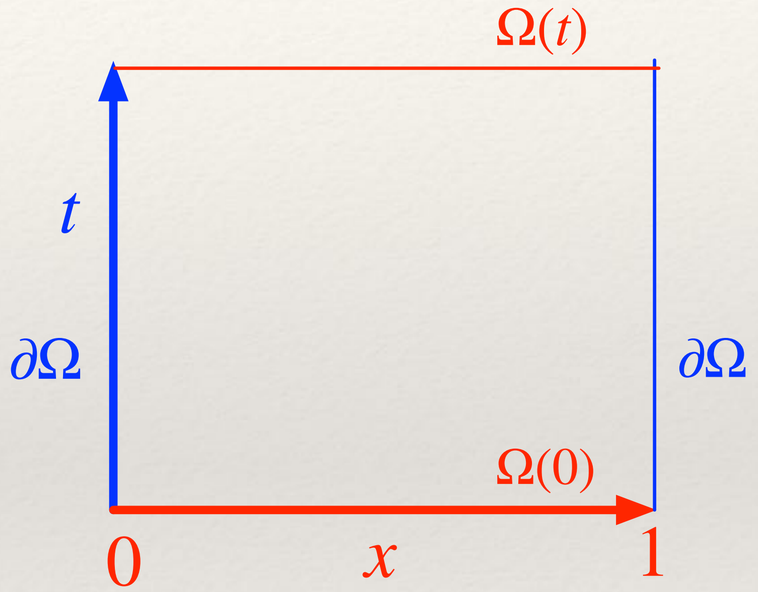
PDE Review

- $\partial_t u(x, y) = L(x, t, u, u_x, u_{xx}) + f(x, t)$
- Special cases:
  - Parabolic: $\partial_t u = \partial_{xx} u + f(x, t)$ (heat conduction)
    I.C.: $u_\Omega(x,0) = U_0(x)$, B.C.: $u_{\partial\Omega}(x, t) = U_{\partial\Omega}(x, t)$
  - Hyperbolic: $\partial_t u = a\partial_x u + f(x, t)$ (wave propagation)
    characteristic boundary conditions
  - Elliptic: $\partial_{xx} u + au = f(x, y)$ (stationary waves)
    B.C.: $u_\Omega(x) = U(x)$



How to Solve a PDE?

- Consider $\partial_t u = \partial_{xx} u + f(x,t)$
- I.C.: $u(x,0) = U_0(x), x \in [0,1]$
- BC: $u(0,t) = u_0$, $u(1,t) = u_1$



- **Next: contrast two solution algorithms**
  - Finite-Difference and Spectral Collocation
  - Use Euler methods for time-advancement

Finite-Difference

- **Discretization operators** ($\tau = \Delta t, h = \Delta x$):

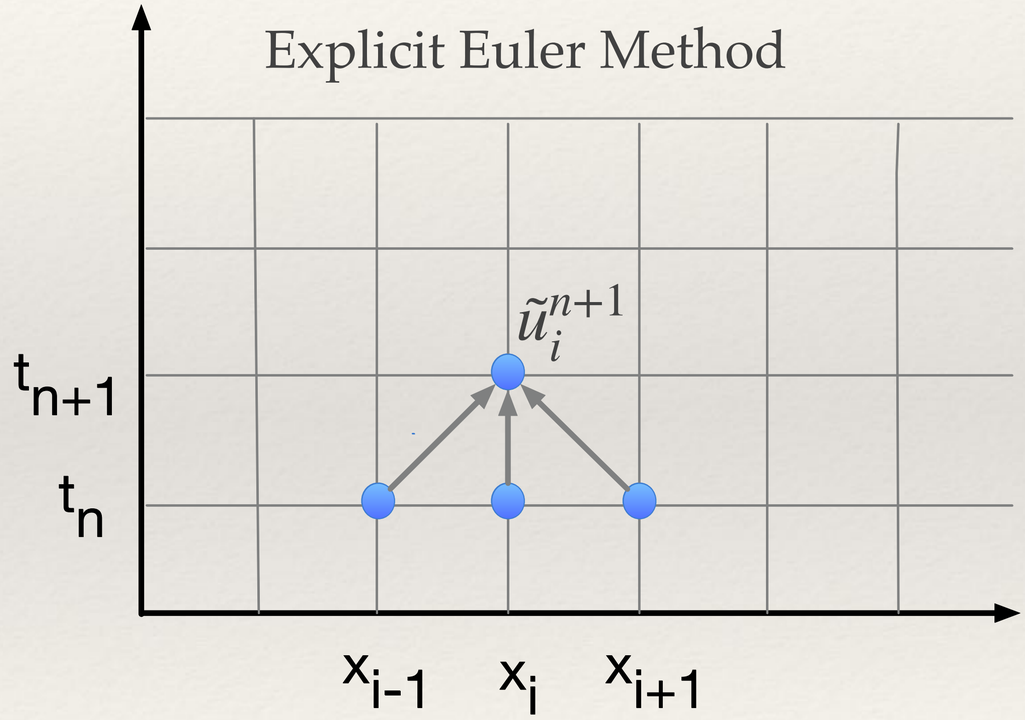
$$\partial_t u\Big|_{t=t^n} = \frac{u_i^{n+1} - u_i^n}{\tau} + O(\tau)$$

$$\partial_{xx} u\Big|_{t=t_n} = \frac{u_{i+1}^n - u_{i-1}^n}{2h} + O(h^2)$$

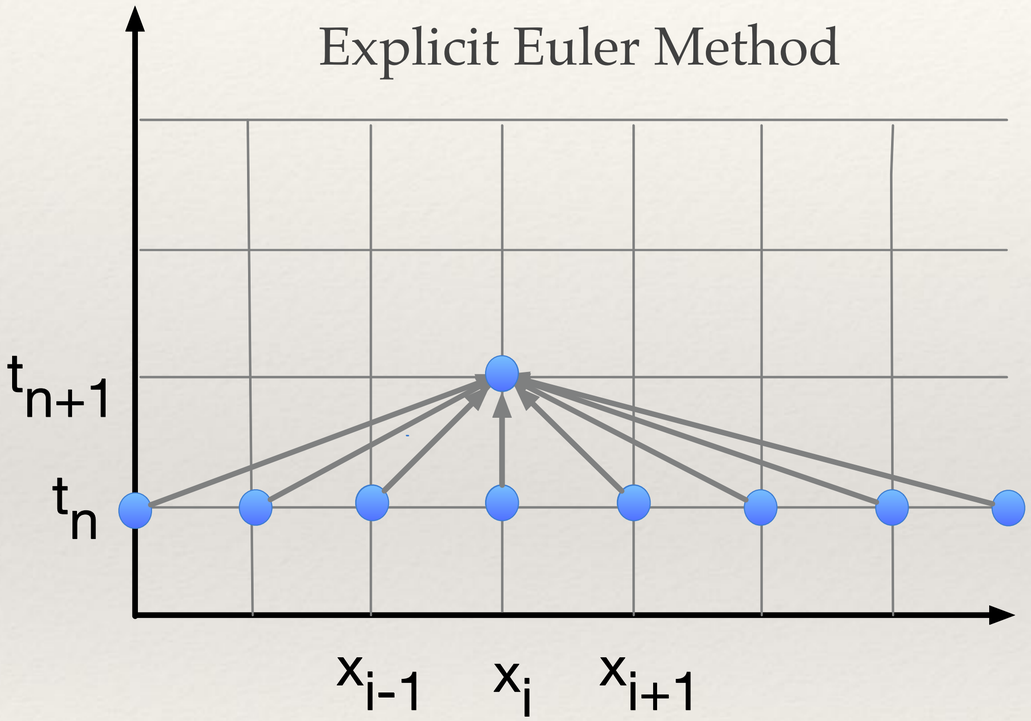
- **Time advancement**

$$\tilde{u}_i^{n+1} = \tilde{u}_i^n + \frac{\tau}{2h}(\tilde{u}_{i+1}^n - \tilde{u}_{i-1}^n),$$

$$i = 1, \cdots, N-2, n = 0, \cdots, T/\tau$$



Spectral Collocation



- Discretization

$$u(x) = \sum_n a_n \phi_n(x)$$

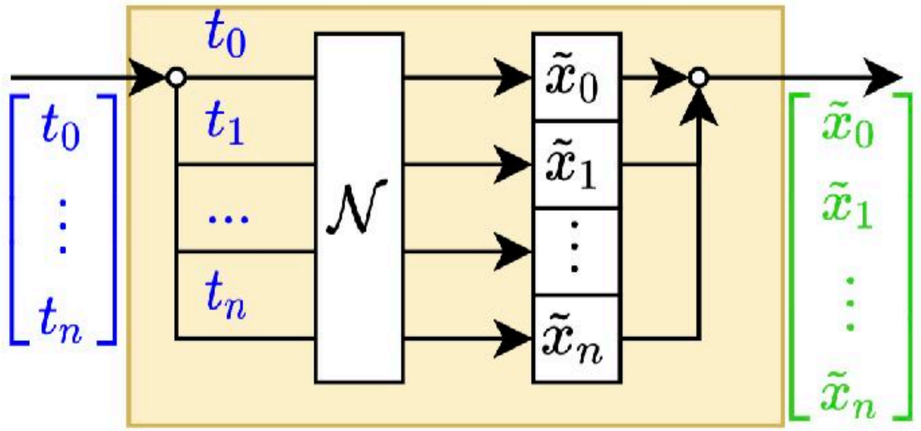
- Time advancement

$$u_i^{n+1} = u_i^n + \tau L(u, \partial_x u, \partial_{xx} u)\Big|_{u=u_i^n=u(x_i^n)}$$

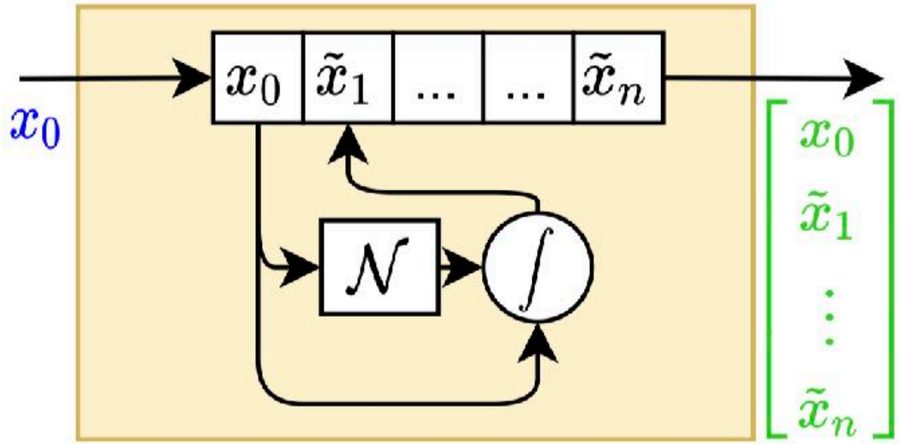
$$= u_i^n + \tau F(u_0^n, u_1^n, \cdots, u_N^n)$$

x_i are collocation points
(equidistant, Gauss points, Chebyshev, Legendre, …

Direct Solution versus Time-Stepper



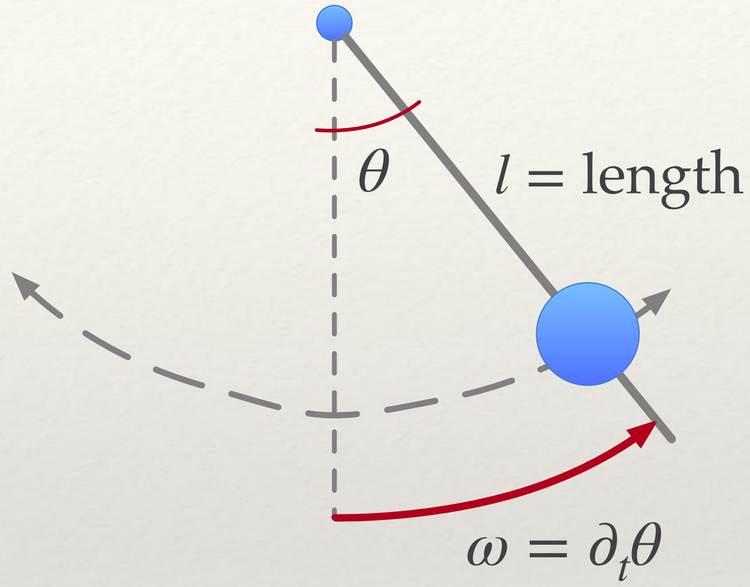
(a) Direct-solution model. A NN is used to parameterize a mapping from a time instance to the solution corresponding to that time instance.



(b) Time-stepper model. The network, $\mathcal{N}$, provides the derivative of the system at various points in state-space, which is then integrated by a numerical solver, here depicted as $\int$.

Pendulum Example

- $\frac{\partial^2\theta}{\partial t^2} + \frac{g}{l}\sin(\theta) = 0$
- $\begin{bmatrix} \partial_t\omega \\ \partial_t\theta \end{bmatrix} = \begin{bmatrix} -\frac{g}{l}\sin\theta \\ \omega \end{bmatrix} = f(x)$
- $\frac{dx}{dt} = f(x)$ = derivative function
- $x = (\omega, \theta)$ = state variables



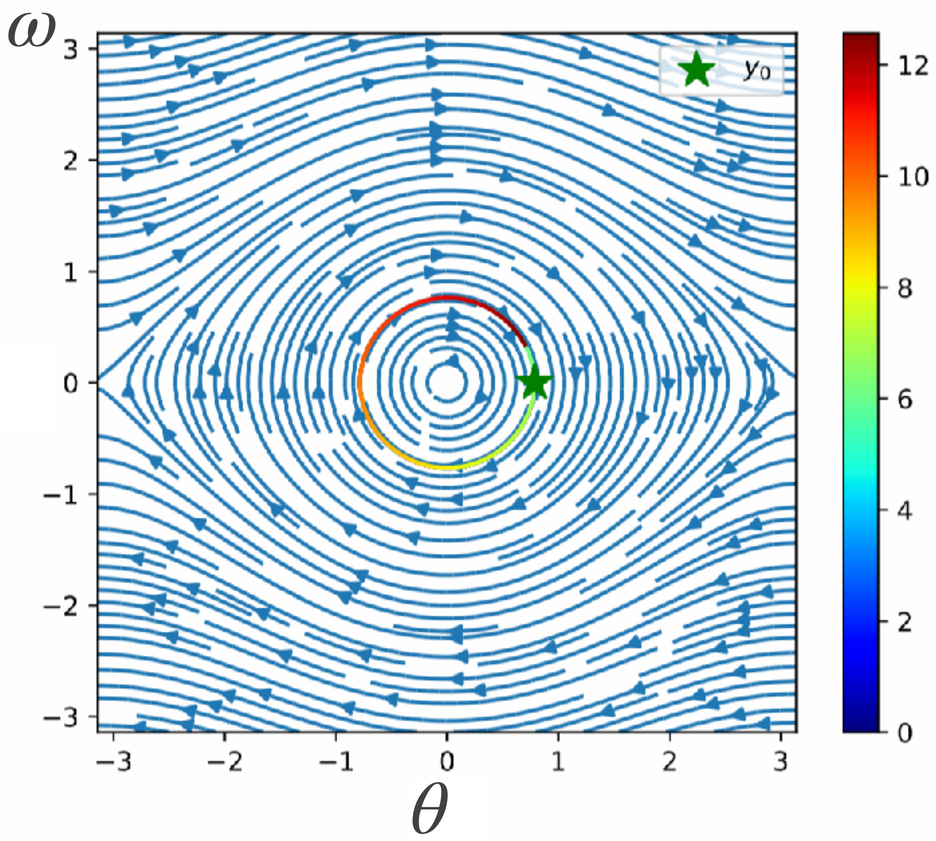
Initial conditions: $x(0) = (\omega_0, \theta_0)$

θ_0 : initial angle

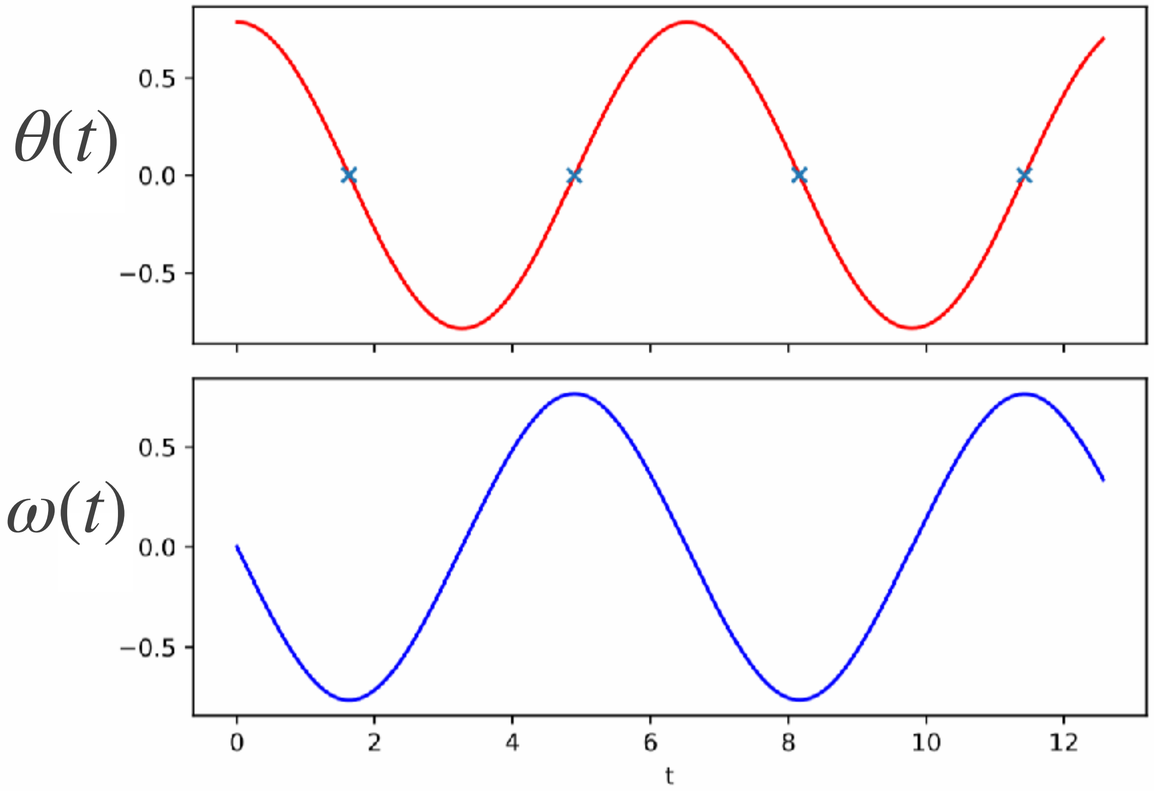
ω_0 : initial velocity

Accurate Solution to Pendulum Equations

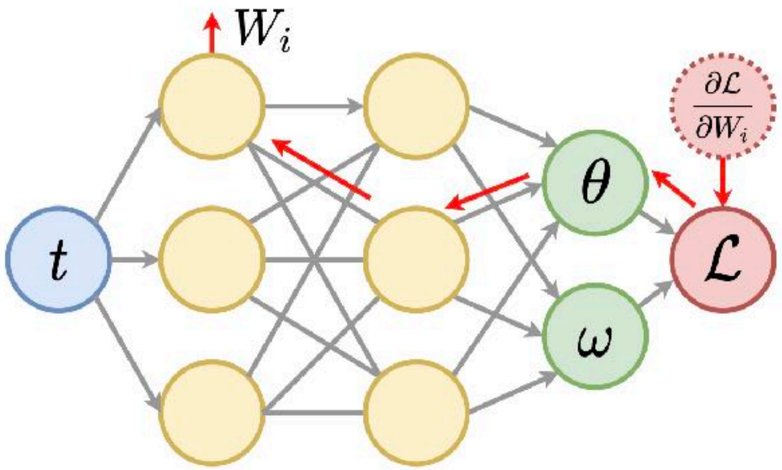
Phase Portrait



Solution



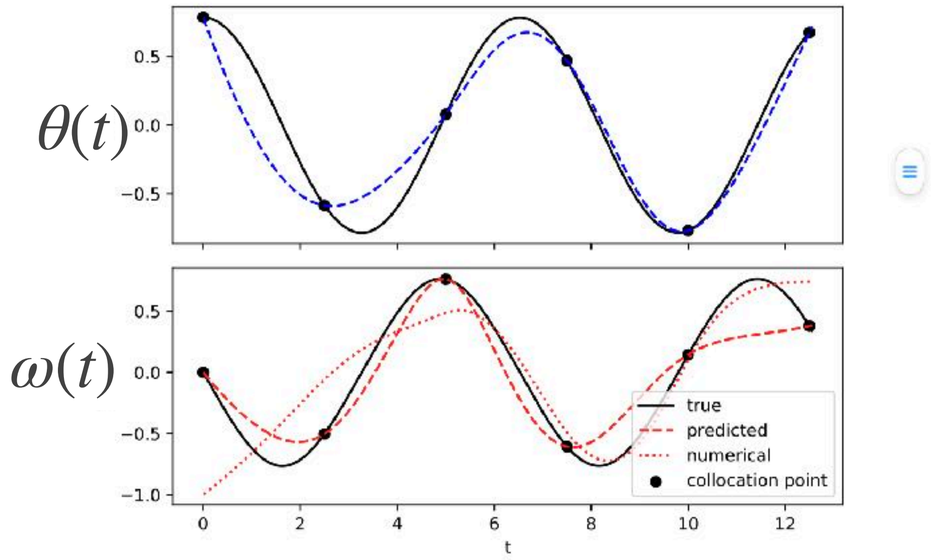
Basic Direct Solution Model



(a) Network structure.

```
θ̃,ω̃ = network(t)
loss = MSE((θ̃, ω̃), (θ, ω))
optimizer.step(loss)
```

(c) Training.

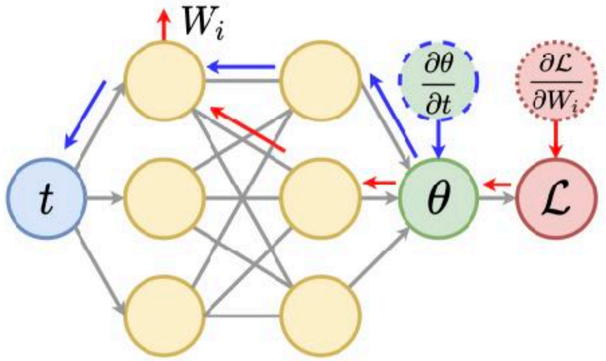


(b) Predictions.

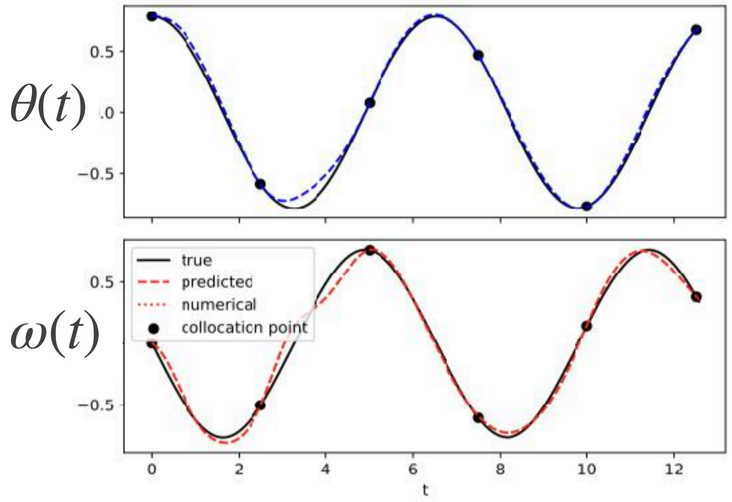
```
θ̃,ω̃ = network(t)
```

(d) Inference.

Enforce Gradient Relationships



(a) Network structure.



(b) Predictions

```
θ̃ = network(t)
ω̃ = gradient(θ̃, t)
loss = MSE((θ̃, ω̃), (θ, ω))
optimizer.step(loss)
```

(c) Training.

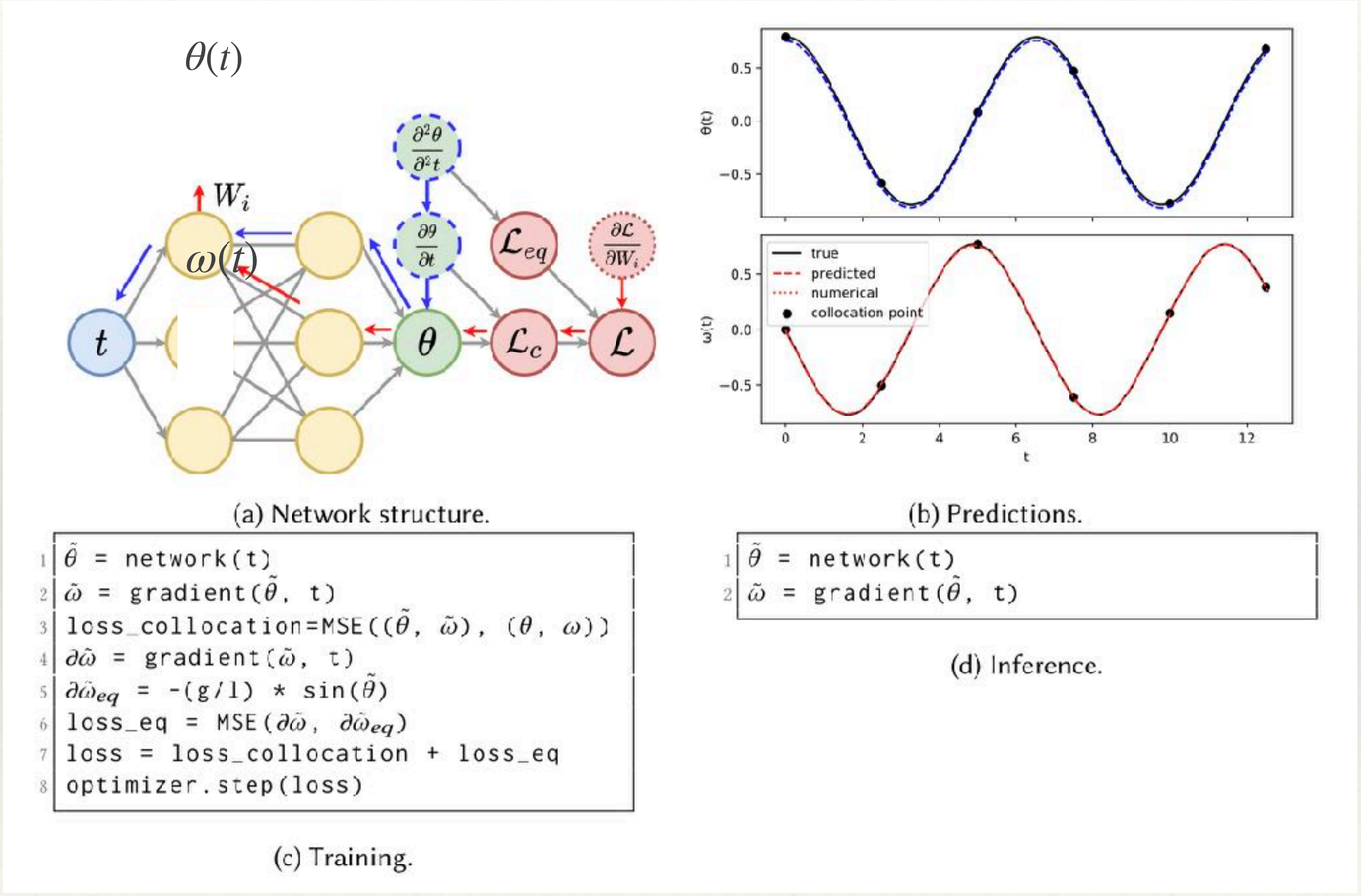
```
θ̃ = network(t)
ω̃ = gradient(θ̃, t)
```

(d) Inference.

Physics-Informed Neural Network

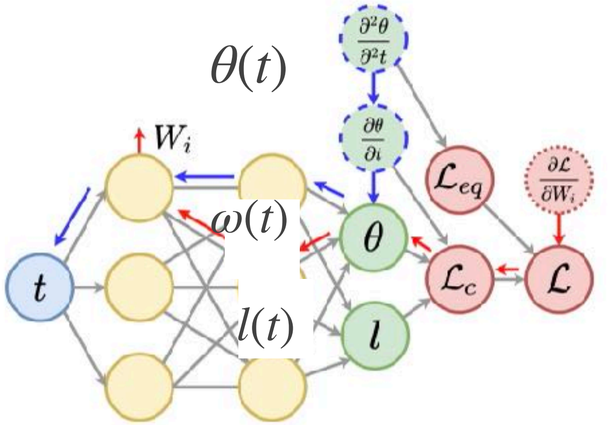
- Points chosen for loss_colloc and loss_eq can be different
- Experimental measurements can be incorporated in loss_colloc
- Loss_colloc could also take the form:

$$\sum_k (\partial_t \omega_k - \frac{g}{l} \sin\theta_k)^2 + \sum_k (\partial_t \theta_k - \omega_k)^2$$



Hidden-Physics Network

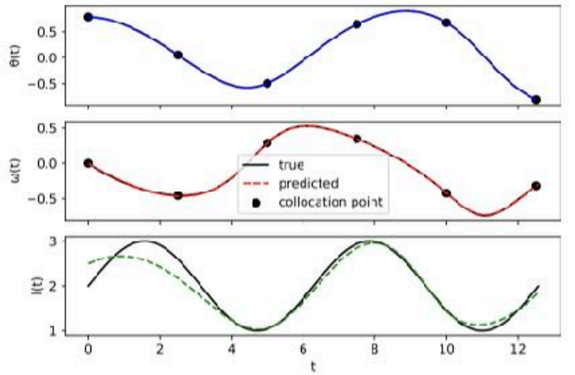
- The equations are parametrized by the pendulum length $\tilde{l}$
- If several parameters, uniqueness is not guaranteed.



(a) Network structure.

```
θ̃, l̃ = network(t)
ω̃ = gradient(θ̃, t)
loss_collocation=MSE((θ̃, ω̃), (θ, ω))
∂ω̃ = gradient(ω̃, t)
∂ω̃_eq = -(g / l̃) * sin(θ̃)
loss_eq = MSE(∂ω̃, ∂ω̃_eq)
loss = loss_collocation + loss_eq
optimizer.step(loss)
```

(c) Training.



(b) Predictions.

```
θ̃, l̃ = network(t)
ω̃ = gradient(θ̃, t)
```

(d) Inference.

Time-Stepper Methods

- $\partial_t x(t) = f(x(t), t)$
- $x(t) = x(0) + \int_0^t f(x(\tau), \tau)\, d\tau$
- The neural network approximates $f(t)$ at time t_k
- $x(t_{k+1})$ is approximated via numerical approximation of the integral

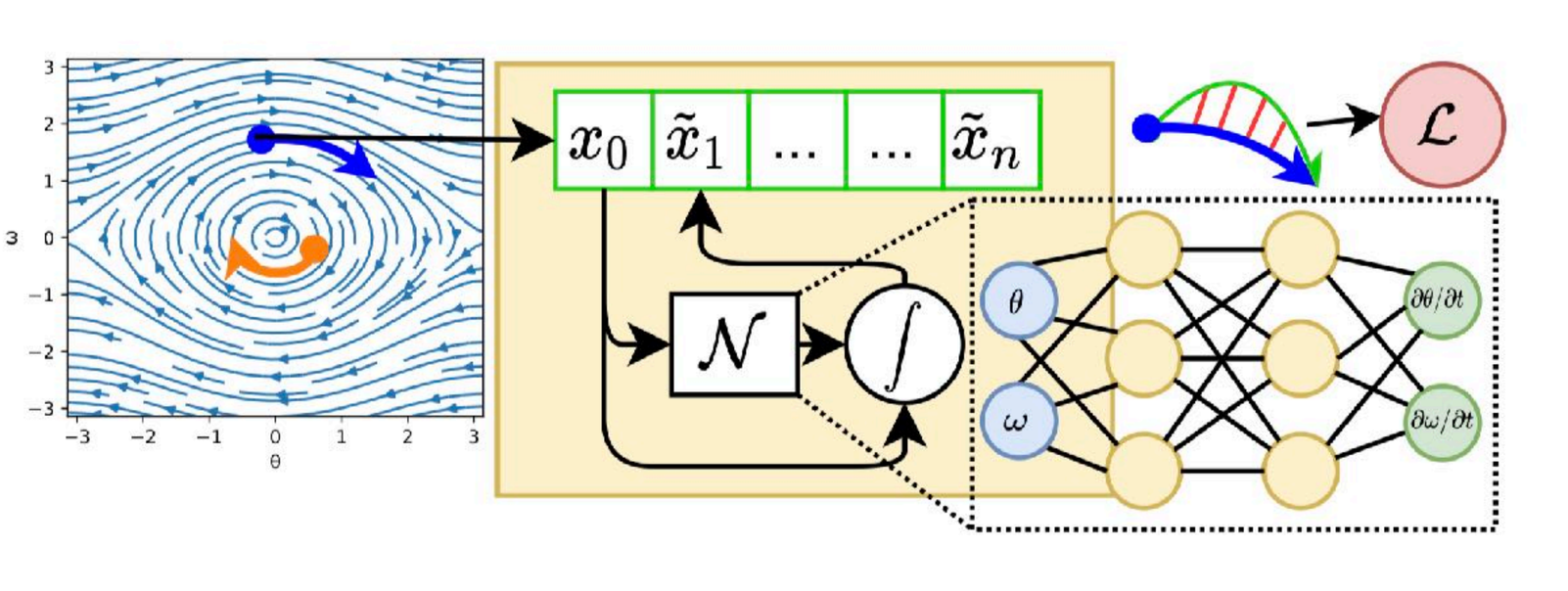
$$x_{k+1} = u_k + \int_{t_k}^{t_{k+1}} f(x(t), t)\, dt$$

Possible Explicit Schemes

- Explicit Schemes:
  - Euler:
    - $\tilde{x}_{k+1} = \tilde{x}_k + \Delta t_k \, f_k$
  - Two-step Adams-Bashford:
    - $\tilde{x}^{k+1} = \tilde{x}_k + 1 + \frac{3}{2}\Delta t f_{k+1} - \frac{1}{2} f_k$
  - Runga-Kutta schemes
  - Many more …

$$f_k = f(\tilde{x}_k, t_k)$$

Time-Stepper Models

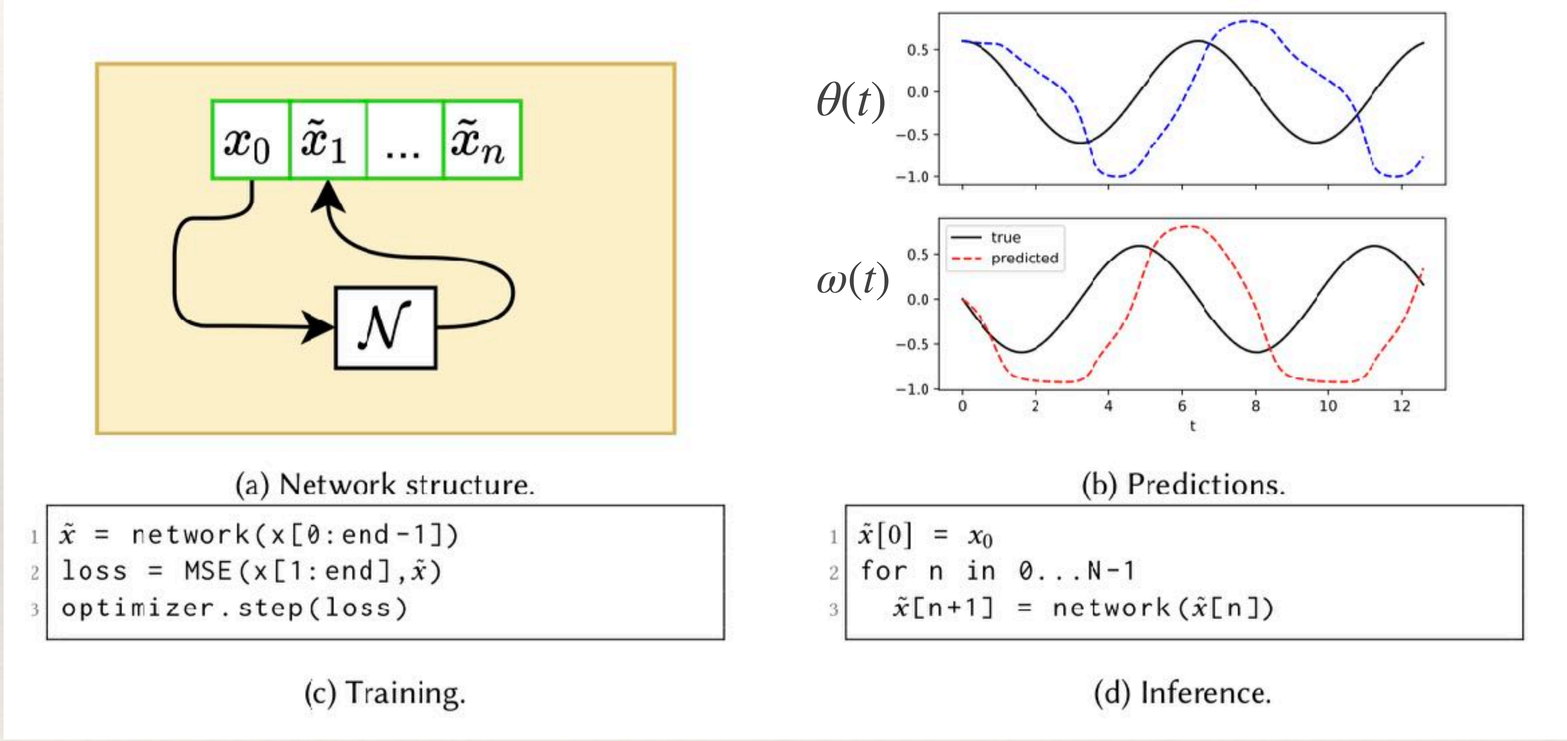


- Generate a set of N short trajectories $(x_0, \tilde{x}_1, \tilde{x}_2, \ldots, \tilde{x}_n)$
- Loss function: MSE between known and exact trajectories

- Sometimes minimize the single step error:

$$\mathscr{L} = \sum_k (\tilde{x}_k - x_k)^2$$

Direct Time-Stepper



(a) Network structure.

(b) Predictions.

```
x̃ = network(x[0:end-1])
loss = MSE(x[1:end],x̃)
optimizer.step(loss)
```

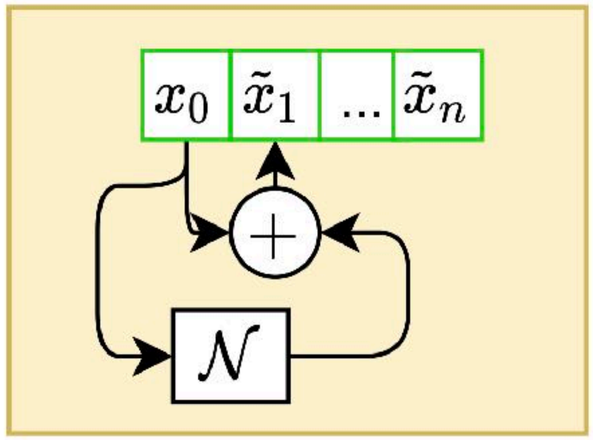
(c) Training.

```
x̃[0] = x_0
for n in 0...N-1
  x̃[n+1] = network(x̃[n])
```

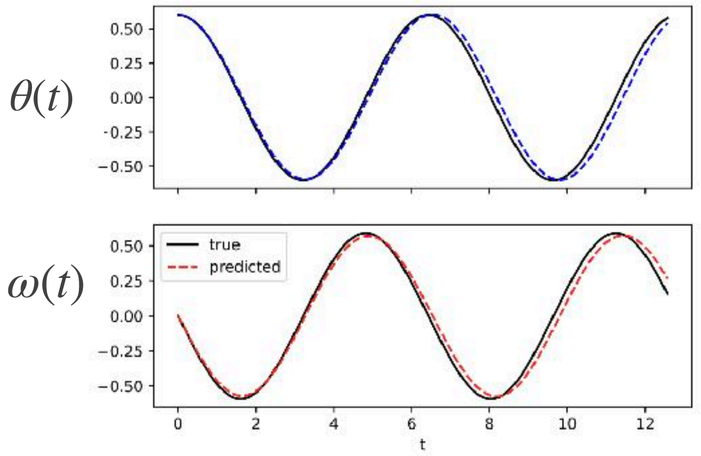
(d) Inference.

- Simplest approach
- The network must learn to model the identity and the time-derivative $f(x, t)$.

Residual Time-Stepper



(a) Network structure.



(b) Predictions.

```
Δx = network(x[0:end-1])
x̃ = x[1:end] + Δx
loss = MSE(x[0:end-1],x̃)
optimizer.step(loss)
```

(c) Training.

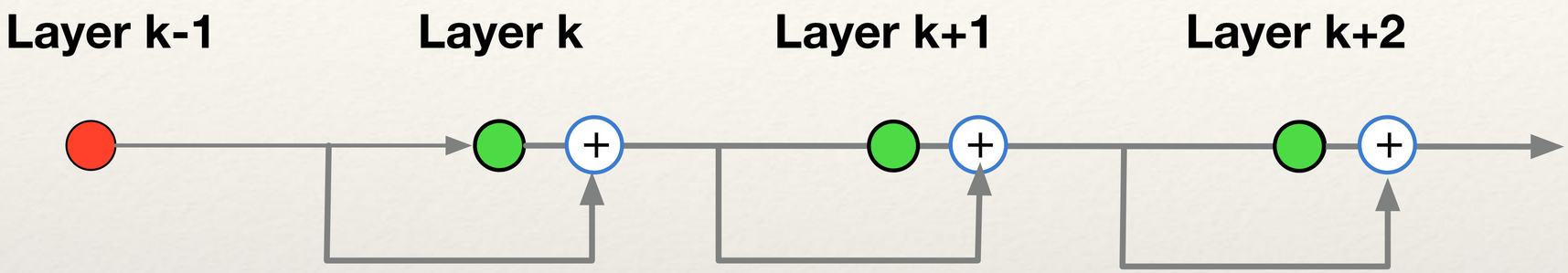
```
x̃[0] = x_0
for n in 0...N-1
  Δx = network(x̃[n])
  x̃[n+1] = x̃[n] + Δx
```

(d) Inference.

- Built on the residual network:

$$\tilde{x}_{k+1} = \tilde{x}_k + N(\tilde{x}_k, t_K)$$

Residual Network



$$x_k = x_{k-1} + f(W_{k-1}x_{k-1} + b_{k-1})$$

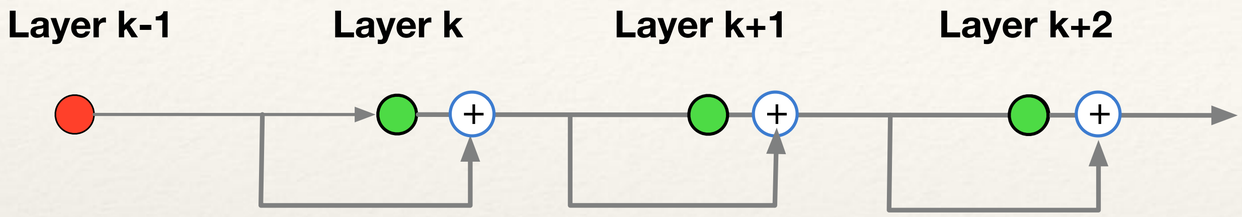
$$x_{k+1} = x_k + f(W_k x_k + b_k)$$

$$x_{k+2} = x_{k+1} + f(W_{k+1}x_{k+1} + b_{k+1})$$

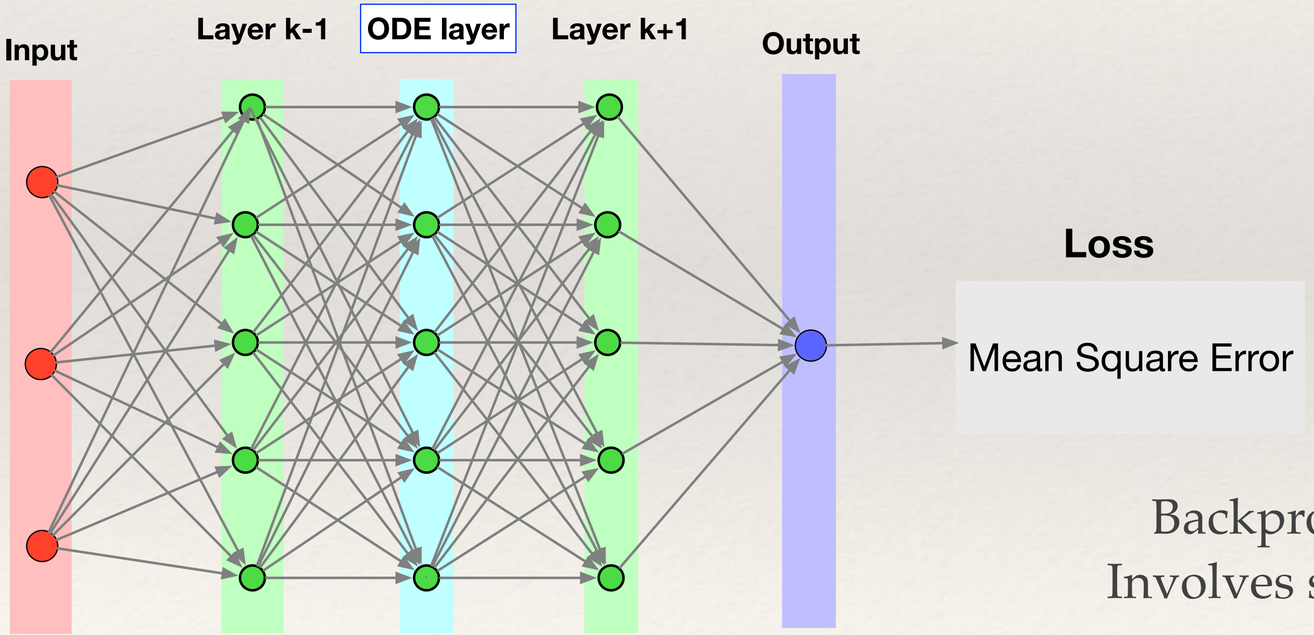
3 iterations of Euler's equation
with $\Delta t = 1$

$$\Longrightarrow \quad \frac{dx}{dt} = f(x, t)$$

Neural ODE



$$\frac{dx}{dt} = f(x, t; W_k)$$

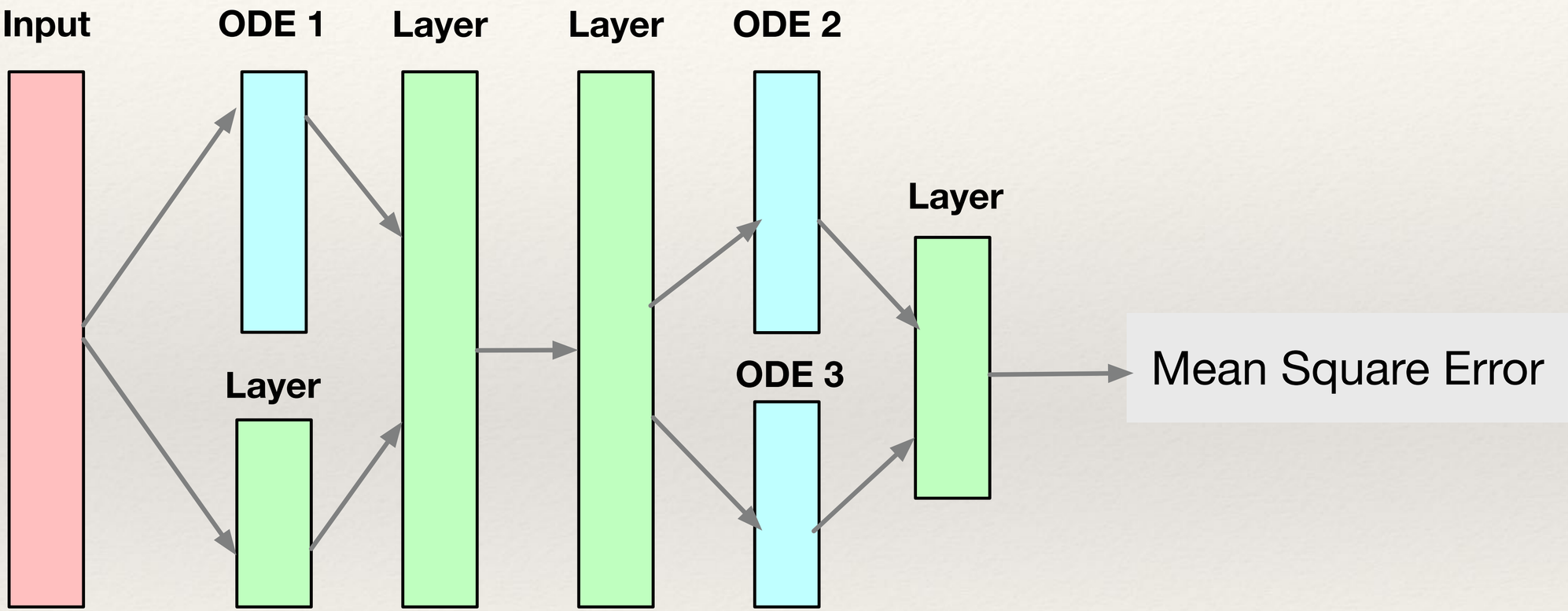


$$\text{Loss} = \sum_k \mathscr{L}_k$$

$$\mathscr{L}_k = MSE(x_k, \tilde{x}_k, t_k; W_k, b_k)$$

Backpropagation must go *through* the ODE layers
Involves solving adjoint equations backwards in time

More General NeuralODEs



Applications of Phys. Inform. ML

- Neural Network solves PDE
- Data Assimilation
- Multi-domain
- Optics
- Biology
- Physics
- Geology
- Seismology
- many more

Issues

- Convergence
- Accuracy
- Noise
- Loss functions
- Hyperparameters
- Efficienty
  - Prediction is very fast (NN is a surrogate model of the PDEs)
  - Training can be very expensive with low generalizability

Challenges

- Minimize parameters
- Choose sampling points
- Develop new loss functions
  - Collocation (standard PINN)
  - Galerkin PINN
  - C-PINN (multiple domains)
- Model operators (DeepONets)

Frameworks

- Python
  - torch physics (https://github.com/boschresearch/torchphysics)
  - deepxde (https://github.com/lululxvi/deepxde)
    - pytorch, tensor flow, Jax, Paddle
- Julia
  - SciML (https://sciml.ai)
  - https://github.com/SciML/

Source Material

(1) [Leg22] Constructing Neural Network-Based Models for Simulating Dynamical Systems, by Legaard etal, arXiv:2111.01495v2, 2022.

(2) [Rac21] Universal Differential Equations for Scientific Machine Learning, 2021.

(3) [Chen18] Neural Ordinary Differential Equations, 2018.

(4) [Rai17] Driven by Data or Derived Through Physics? A Review of Hybrid Physics-Guided ML Techniques with a CPS focus Rai et al, DOI 10.1109/ACCESS.2020.2987324

Questions?

Modeling

- Derive functional forms of equations
  $c_1 u_t + c_2 u_x + c_3 u_y + c_4 u_{xt} + c_5 u_{yt} + \ldots = 0$:
- Calculate coefficients $c_1, c_2, \cdots, c_n$

Sindy

- Given a time-dependent signal, deduce a dynamical system that describes it.

Inductive Bias

Neural Network Components

- Input
- Network parametrized by $\Theta = (\theta_1, \cdots, \theta_n)$
- Output
- Loss function

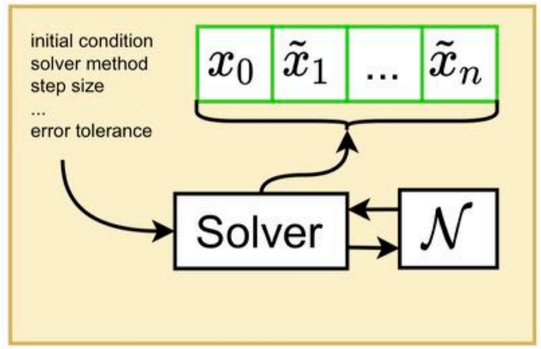
PINN: Physics-Informed Neural Networksx

Physics-informed neural networks (PINNs) are a type of universal function approximator that embeds the knowledge of any physical laws that govern a given data-set in the learning process, and can be described by partial differential equations (PDEs).

PINN Variations

- PINN
- CPINN
- XPINN
- B-PINN
- VA-PINN
- FEM-PINN

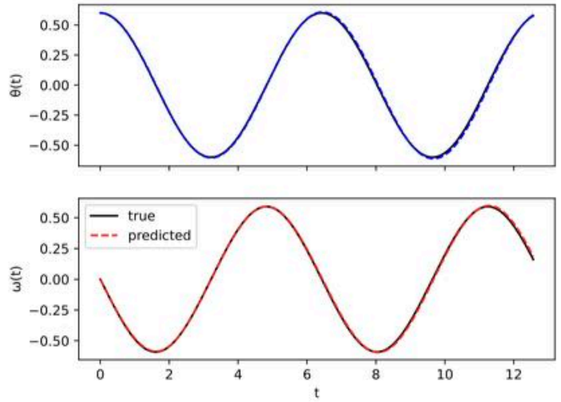
Neural Differential Equations



(a) Network Structure.

```
x̃ = odeint(net, x_0, t_start, t_start + h, "rk4")
loss = MSE(x, x̃)
optimizer.step(loss)
```

(c) Training.

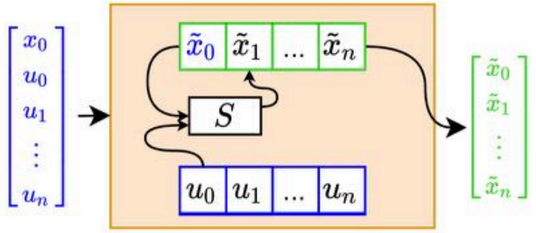


(b) Predictions.

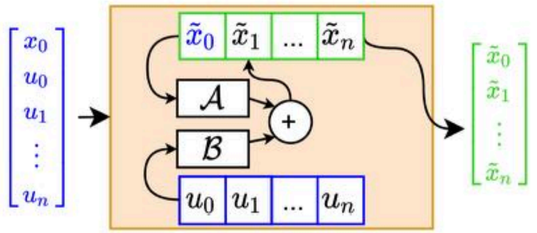
```
x̃ = odeint(net, x_0, t_start, t_end, "rk4")
```

(d) Inference.

Fig. 16. Neural ordinary differential equations. Neural ODEs generally refer to models that are constructed to use a numerical solver to integrate the derivatives through time. Unlike the previously introduced integration schemes which mapped to concrete architectures, neural ODEs refer to the idea of using well established numerical solvers inside a model. Part of neural ODEs popularity is due to the fact that it mimics the programming APIs of traditional numerical solvers, which makes it easy to switch between different types of solvers.



(a) State and input stacked and fed into the same network.



(b) State and input fed into separate networks.

Fig. 17. Incorporation of inputs in time stepping model.

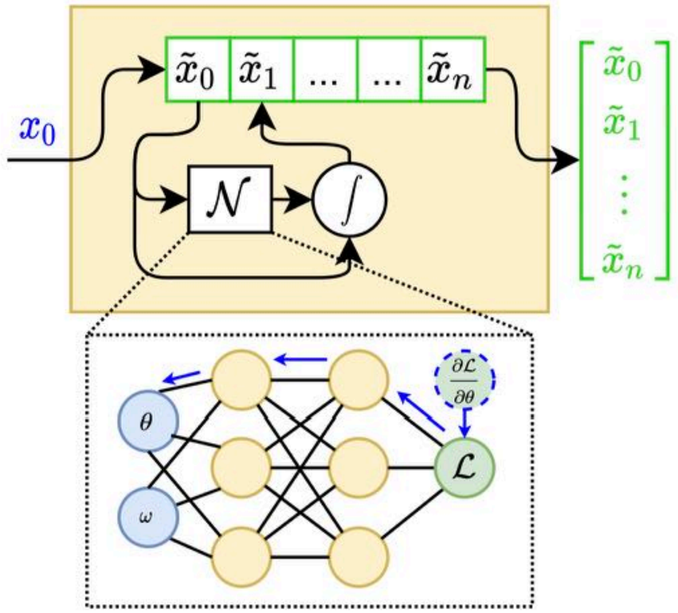


Fig. 18. Lagrangian time-stepper. The Lagrangian, $\mathcal{L}$ (not to be confused with the loss function), is differentiated using AD to obtain the derivative of the state.

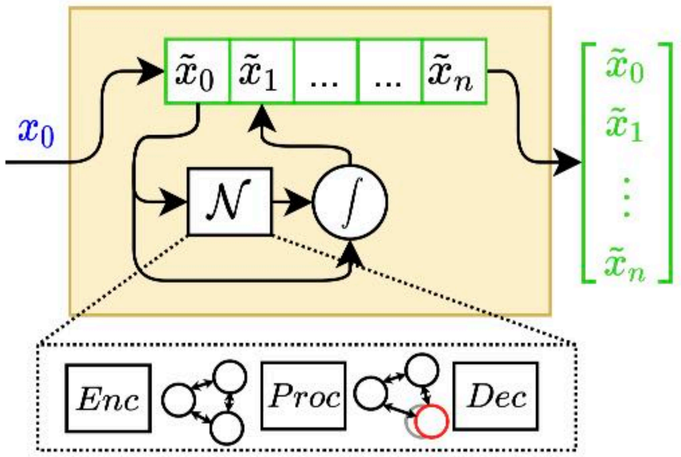


Fig. 19. Simplified view of a graph time-stepper. During each step of the simulation, the current state is encoded as a graph (Enc) which is then used to compute the change in state variable between the current and next time step (Proc). Finally, the change in state is decoded to the original state-space to update the state of the system (Dec).

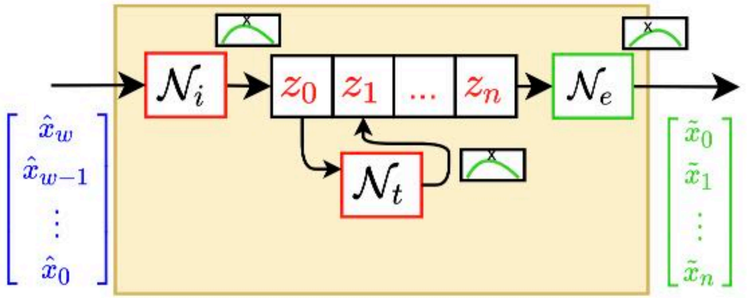


Fig. 20. Deep Markov model with inference network. The value of z_0 is estimated by an inference network N_i based on several samples of the observed variable. The transmission function, approximated by the network N_t, maps the current value of z to a distribution over z one step ahead in time. The emission function, approximated by N_e, maps each predicted latent variable to a distribution of the corresponding x value in the original observed space. Note that the output of each network is the parameters of a distribution, which is then sampled to obtain a value that can be fed into the next stage of the model.

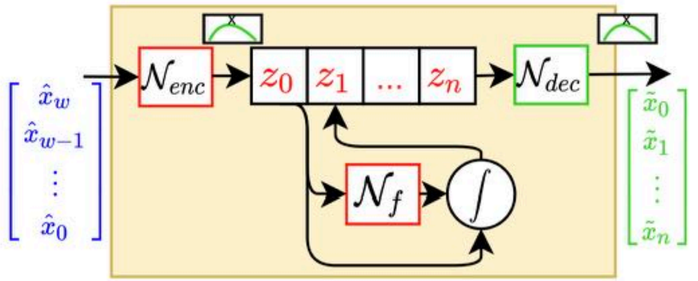


Fig. 21. Latent neural ODEs. An encoder network is used to obtain a latent representation of the system's initial state, z_0, by aggregate information from several observations of the systems $[\hat{x}_w, \hat{x}_{w-1}, ..., \hat{x}_0]$. The system is simulated for multiple steps to obtain $[z_0, z_1, ..., z_n]$. Finally, the latent variables are mapped back to the original state-space by a decoder network.

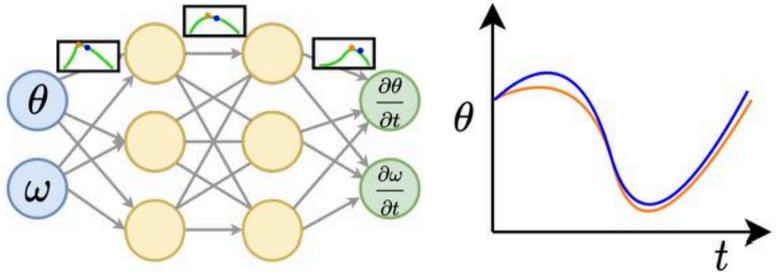


Fig. 22. Bayesian Neural Ordinary Differential Equations. The parameters of the network are characterized by a probability distribution. The parameter distributions are sampled multiple times and used to simulate the system, producing multiple trajectories as shown to the right. To get a single prediction, the predictions can be averaged.

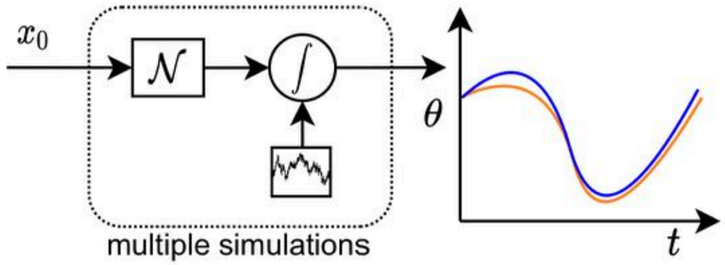


Fig. 23. Neural stochastic differential equations. The network $\mathcal{N}$ is used to approximate the deterministic drift term of the SDE and the diffusion term is a Wiener process. Multiple trajectories are produced by solving the SDE multiple times, corresponding to different realizations of the Wiener process.

Table 2. Comparison of direct-solution and time-stepper models.

Name	Advantages	Limitations
Direct-solution	+ Easy to apply to PDEs + No discretization of time and spatial coordinates + No accumulation of error during simulation + Parallel evaluation of simulation	- Fixed Initial condition - Fixed temporal and spatial domain - Difficult to incorporate inputs
Time-stepper	+ Initial condition not fixed + Easy to incorporate inputs + Leverage knowledge from numerical simulation	- Not trivial to apply to PDEs - Accumulation of error during simulation - No parallel evaluation of simulation

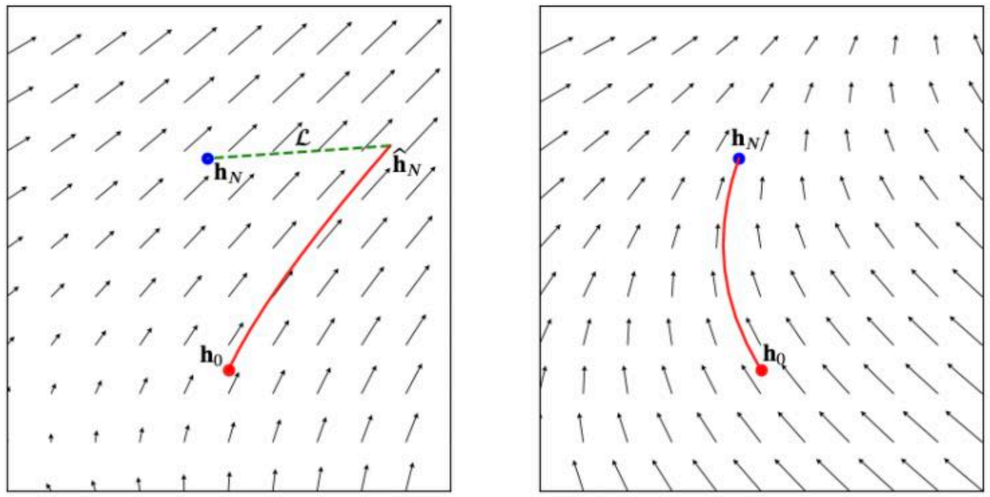


Fig.4: Effect of updating parameters of the dynamics.

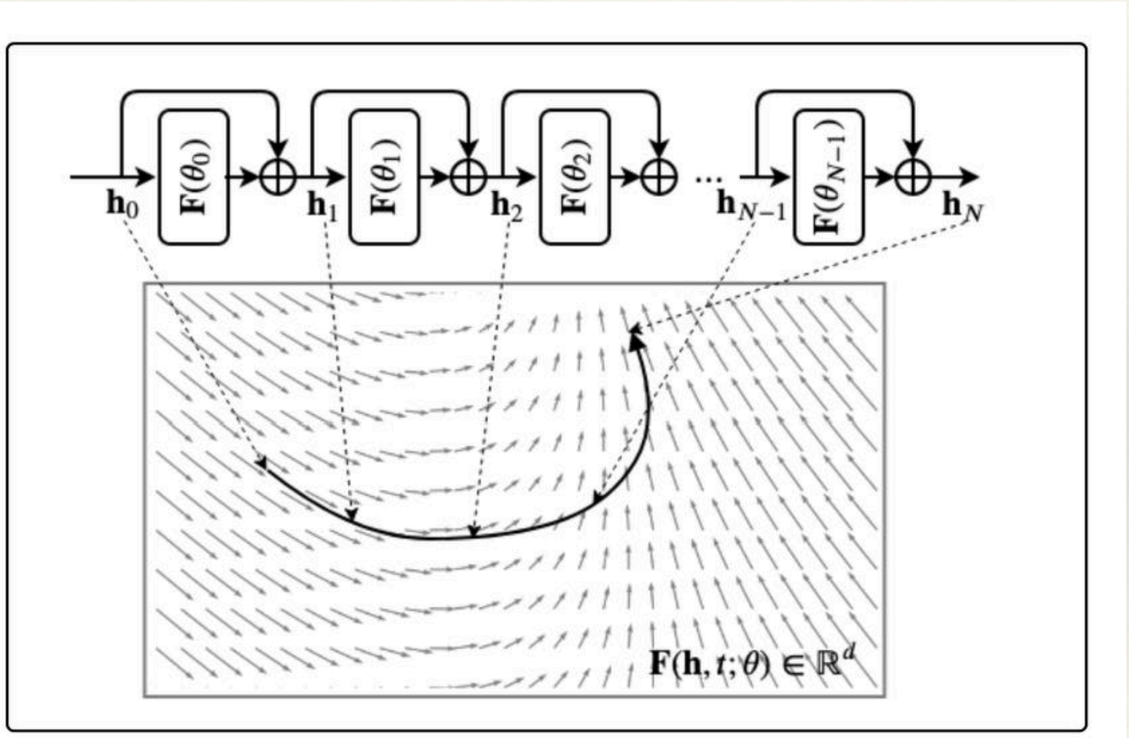


Fig.3: Resemblence of ResNet and Forward Eular's method.

Regression

- $y = \theta_1 x + \theta_2$
- $y = \theta_1 \sin(x) + \theta_2 \sin(2x) + \ldots$
- $y = \theta_1 \sin(\theta_2 x + \theta_3)$

Differential-algebraic equations

Stephen L. Campbell et al. (2008), Scholarpedia, 3(8):2849. doi:10.4249/scholarpedia.2849 revision #153375 [link to/cite this article]

- **Dr. Stephen L. Campbell**, North Carolina State University, Raleigh, NC, USA.
- **Vu Hoang Linh**, Faculty of Mathematics, Mechanics and Informatics, Vietnam National University, Hanoi, Vietnam
- **Linda R. Petzold**, Department of Mechanical Engineering, and Department of Computer Science, University of California Santa Barbara, CA

Post-publication activity

Curator: Vu Hoang Linh

A **differential-algebraic equation** (**DAE**) is an equation involving an unknown function and its derivatives. A (first order) DAE in its most general form is given by

$$F(t, x, x') = 0, \quad t_0 \le t \le t_f, \tag{1}$$

where $x = x(t)$, the unknown function, and $F = F(t, u, v)$ have N components, denoted by x_i and F_i, $i = 1, 2, \ldots, N$, respectively. Every DAE can be written as a first order DAE. The term DAE is usually reserved for the case when the highest derivative x' cannot be solved for in terms of the other terms t, x, when (1) is viewed as an algebraic relationship between three variables t, x, x' . The Jacobian $\partial F/\partial v$ along a particular solution of the DAE may be singular. Systems of equations like (1) are also called implicit systems, generalized systems, or descriptor systems. The DAE may be an initial value problem where x is specified at the initial time, $x(t_0) = x_0$, or a boundary value problem, where the solution is subject to N two-point boundary conditions $g(x(t_0), x(t_f)) = 0$.

The method of solution of a DAE will depend on its structure. A special but important class of DAEs of the form (1) is the semi-explicit DAE or ordinary differential equation (ODE) with constraints

$$\begin{aligned} y' &= f(t, y, z) \\ 0 &= g(t, y, z), \end{aligned} \tag{2}$$

which appear frequently in applications. Here $x = (y, z)$ and $g(t, y, z) = 0$ are the explicit constraints.

Contents [hide]