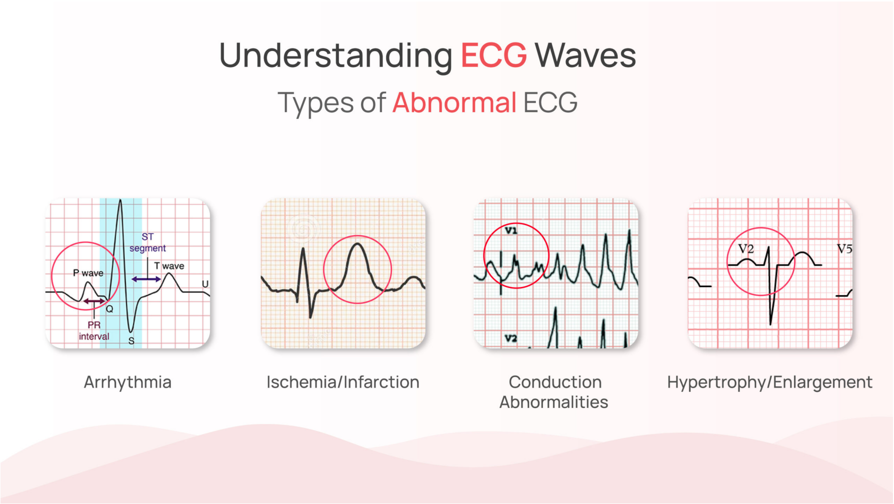
Deep Learning for Time Series

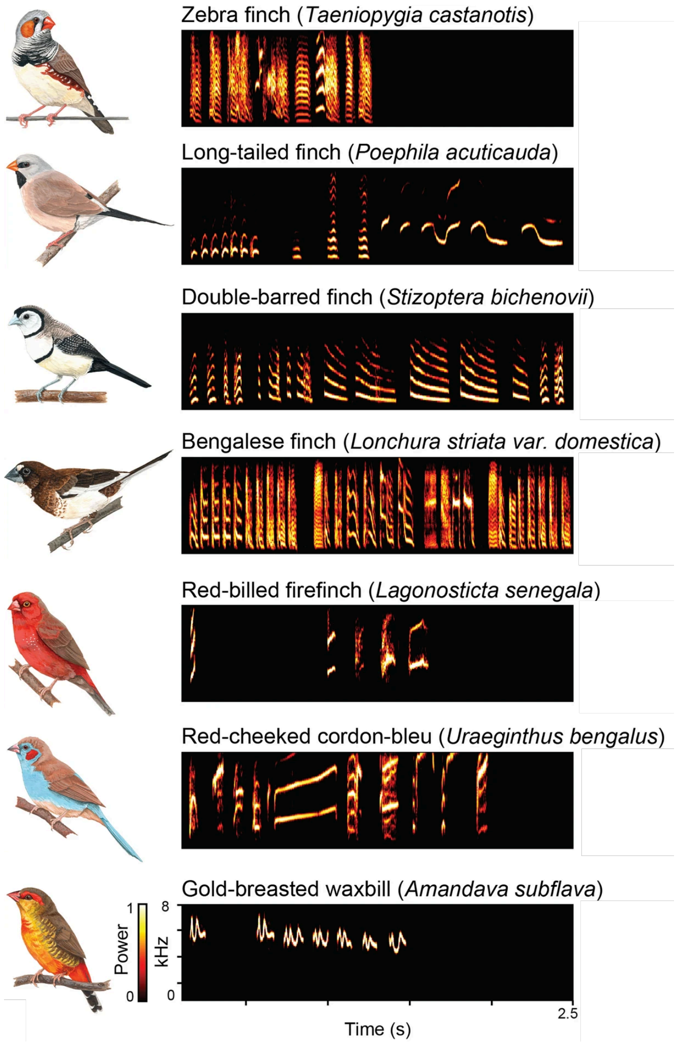
Session 5: Continuous-Time Models

Romain Tavenard

Motivation



Source: sunfox.in blog



Source: "Machine learning and statistical classification of birdsong link vocal acoustic features with phylogeny"

Motivation

- Time series are (almost always) discretization of continuous-time processes
- In real life, sensors fail
  - Missing data
  - Irregular sampling
- Basic neural architectures not tailored for such settings
  - Conv *vs* Recurrent *vs* Attention-based
  - Missing data imputation can help

$\Rightarrow$ **Can we build neural architectures that operate in continuous time?**

Neural ODEs

Ordinary Differential Equations (ODEs)

Assume the evolution of the forecast variable through time follows a system of the form:

$$\begin{cases} \dot{x}(t) & = f(x(t), t) \\ x(0) & = x_0 \end{cases}$$

- Given f and x_0, one can use an ODE solver to compute $x(t)$ for any t by approximating:

$$x(t) = x_0 + \int_0^t f(x(t), t)\,\mathrm{d}t$$

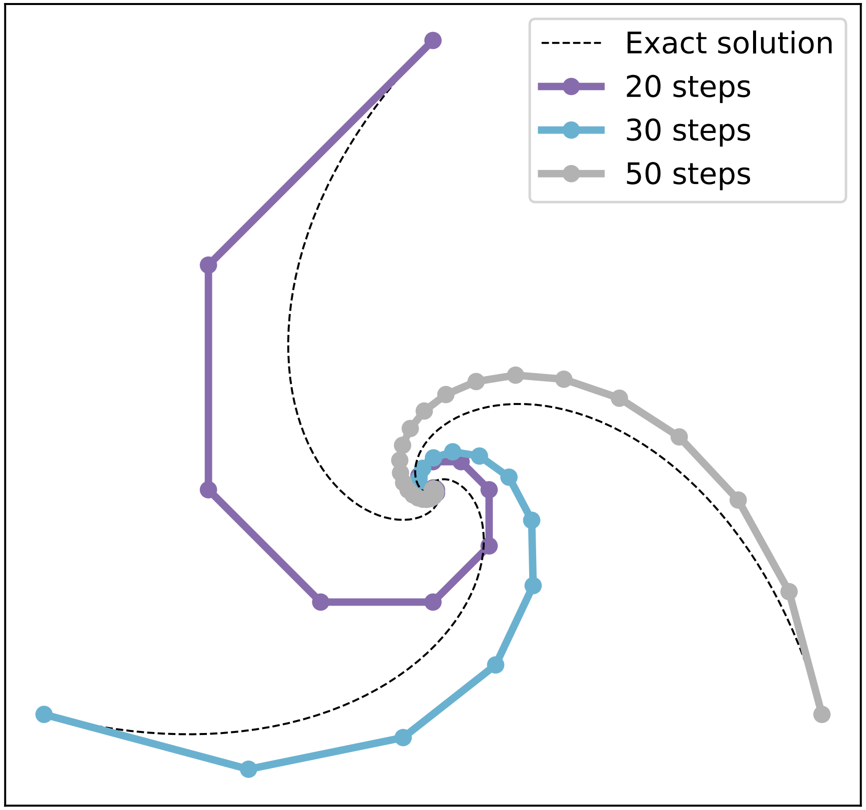
A word on ODE solvers

- Approximate the sequence of values:

$$[x(t+h), x(t+2h), ..., x(t)]$$

- Using a Taylor expansion to compute $x(\tau + h)$ for $x(\tau)$
- First-order Taylor expansion gives the Euler method:

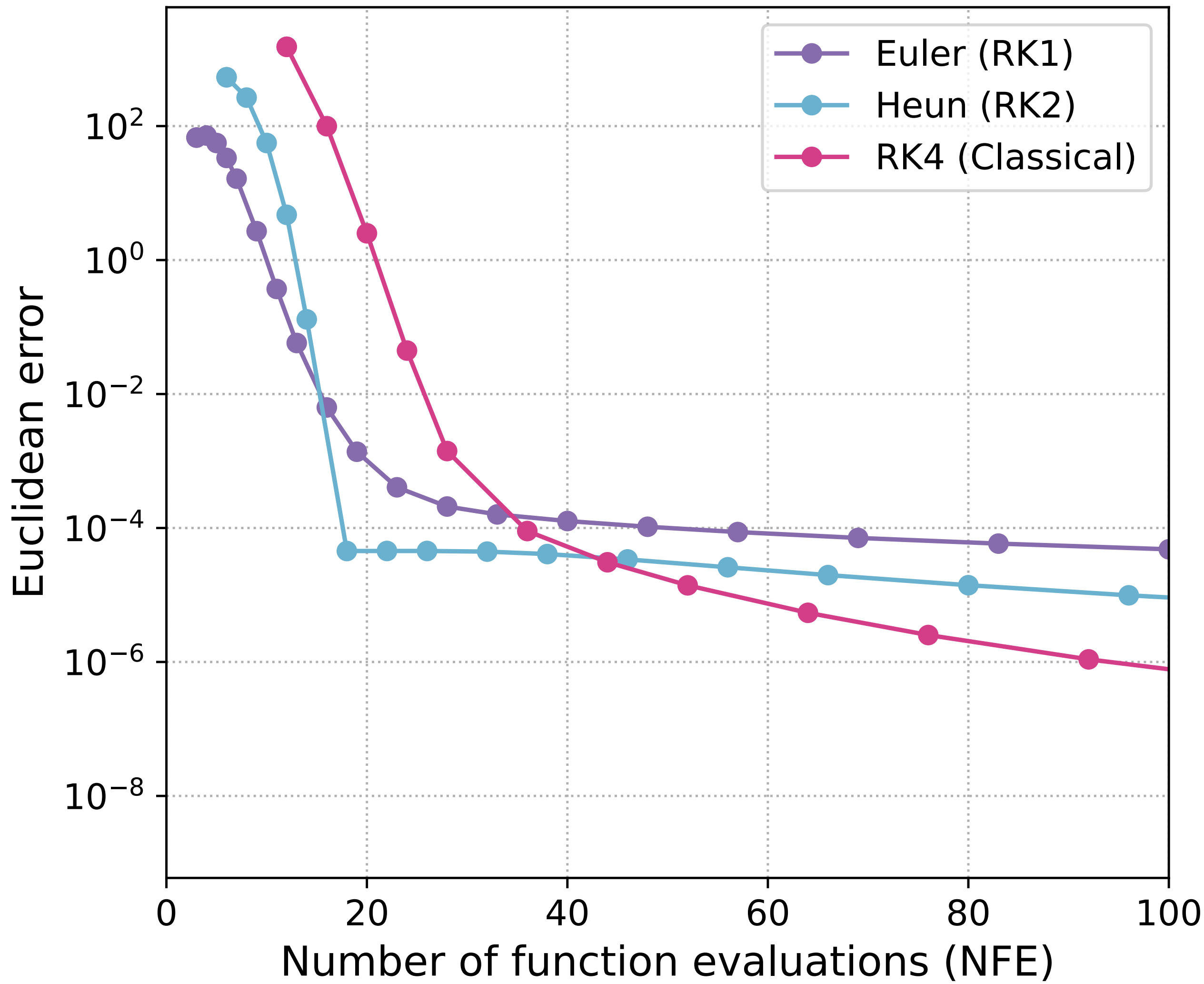
$$x(\tau + h) \approx x(\tau) + hf(x(\tau), \tau)$$



Solving a spiral ODE with a Euler scheme

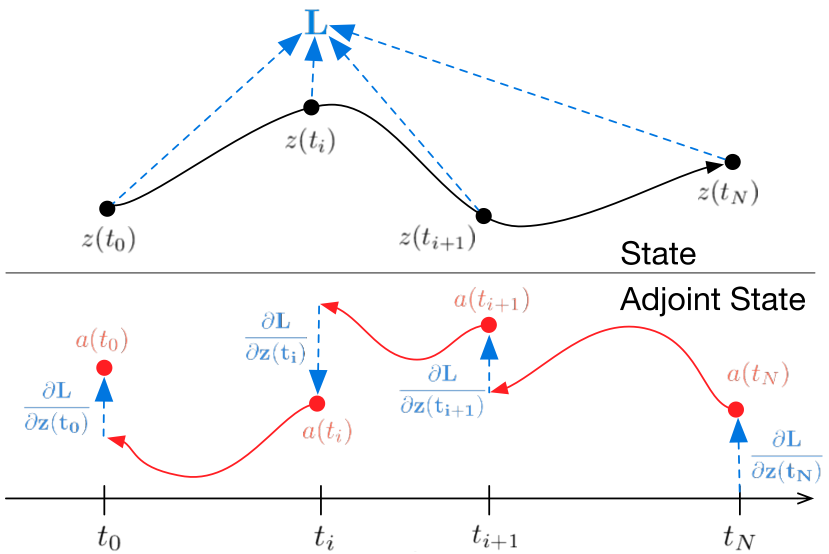
A word on ODE solvers

- More advanced ODE solvers exist
  - Runge-Kutta methods (see below), adaptive step-size, …



Neural ODEs

- ODEs for forecasting
  - What is the right f for our data?
  - Let f be a neural network f_θ and train it on our forecasting task
    - Need to compute gradients through the solver

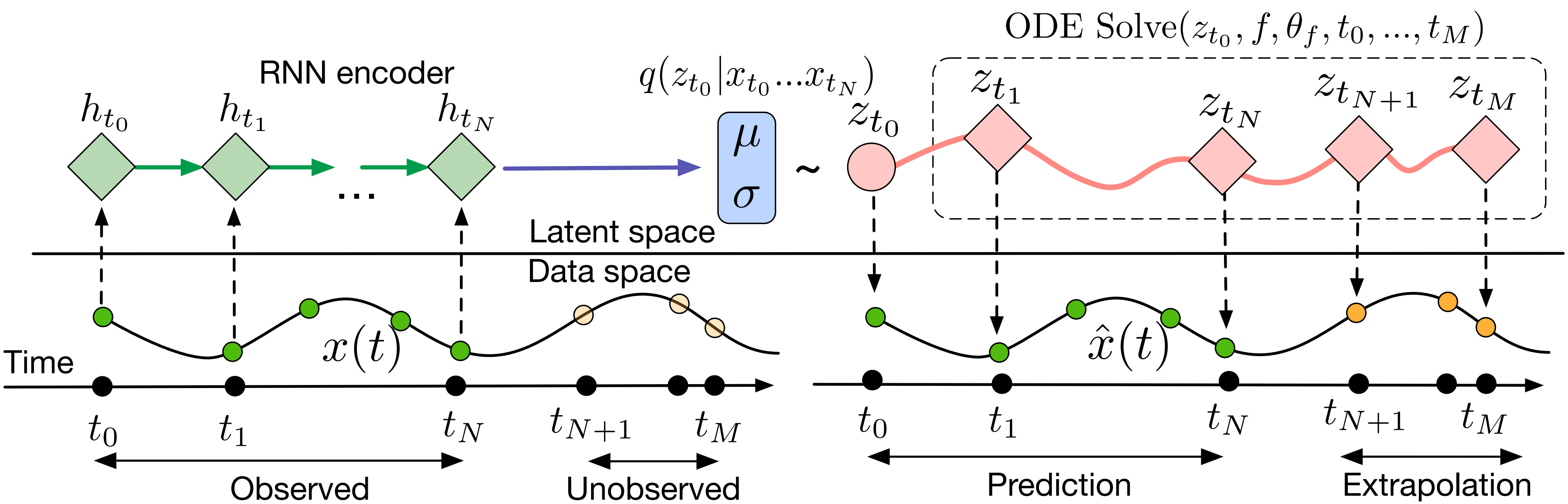


Source: “Neural ODEs”, NeurIPS’19

  - Euler scheme: akin to a ResNet with shared weights

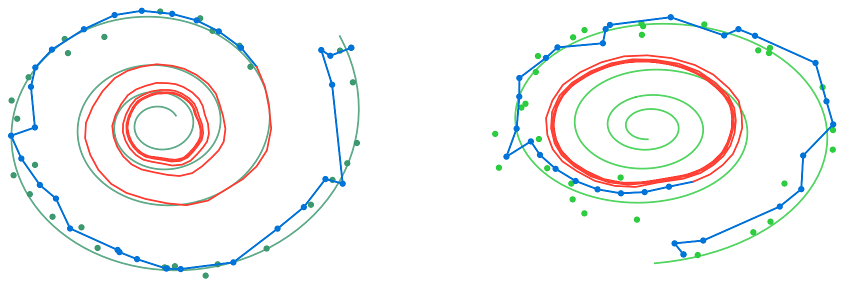
Latent NODEs

- Neural ODEs cannot operate in input space
  - Reconstruction from a single observation x_0 is a strong limitation
- Latent NODEs
  1. project in latent space to get higher-level information
  2. use RNN encoder to summarize information from the past

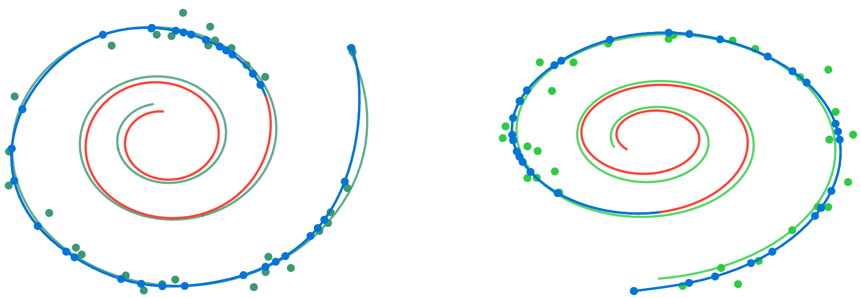


Source: “Neural ODEs”, NeurIPS’19

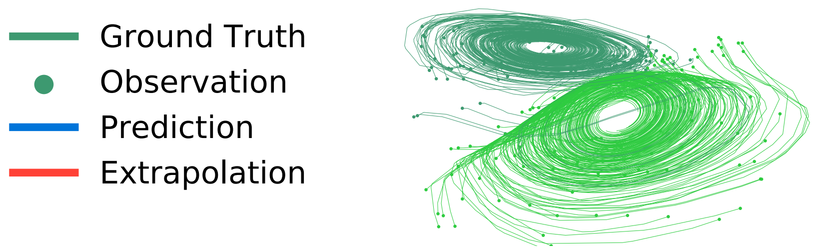
Latent NODEs



(a) Recurrent Neural Network



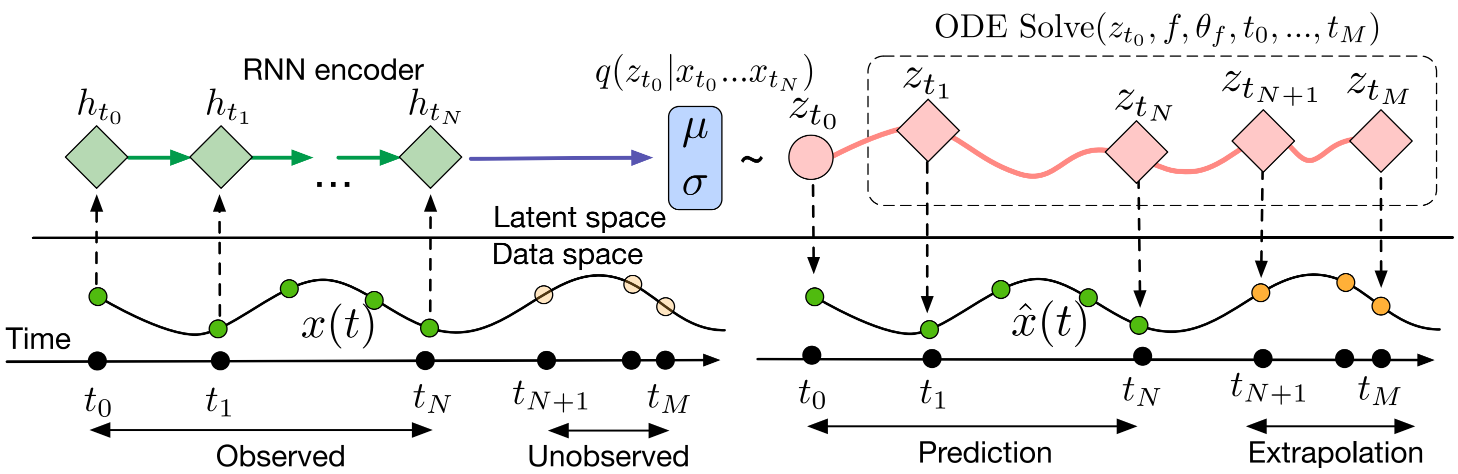
(b) Latent Neural Ordinary Differential Equation



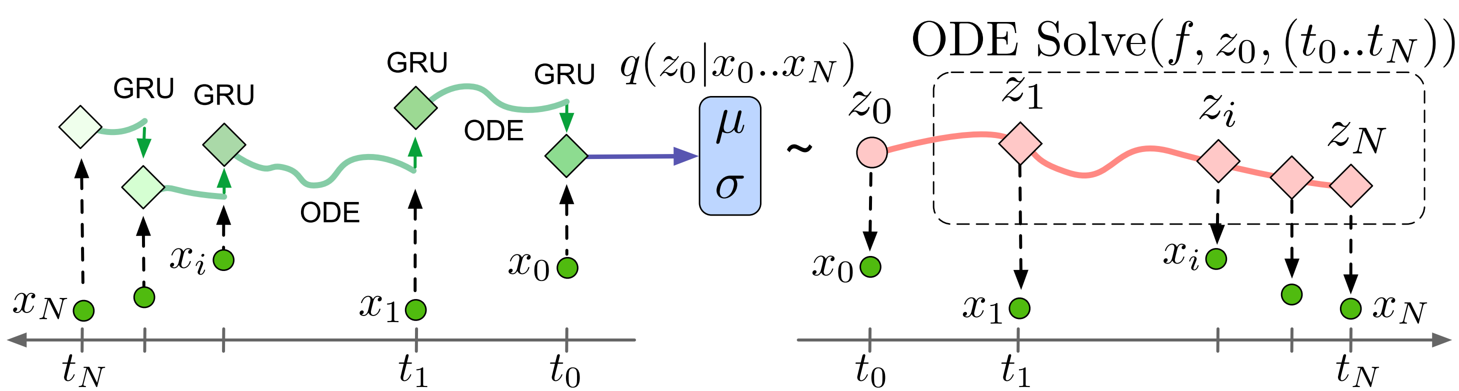
(c) Latent Trajectories

Source: “Neural ODEs”, NeurIPS’19

Latent NODEs



- RNN encoder above still assumes regular sampling



Source: “Latent ODEs for Irregularly-Sampled Time Series”, NeurIPS’20

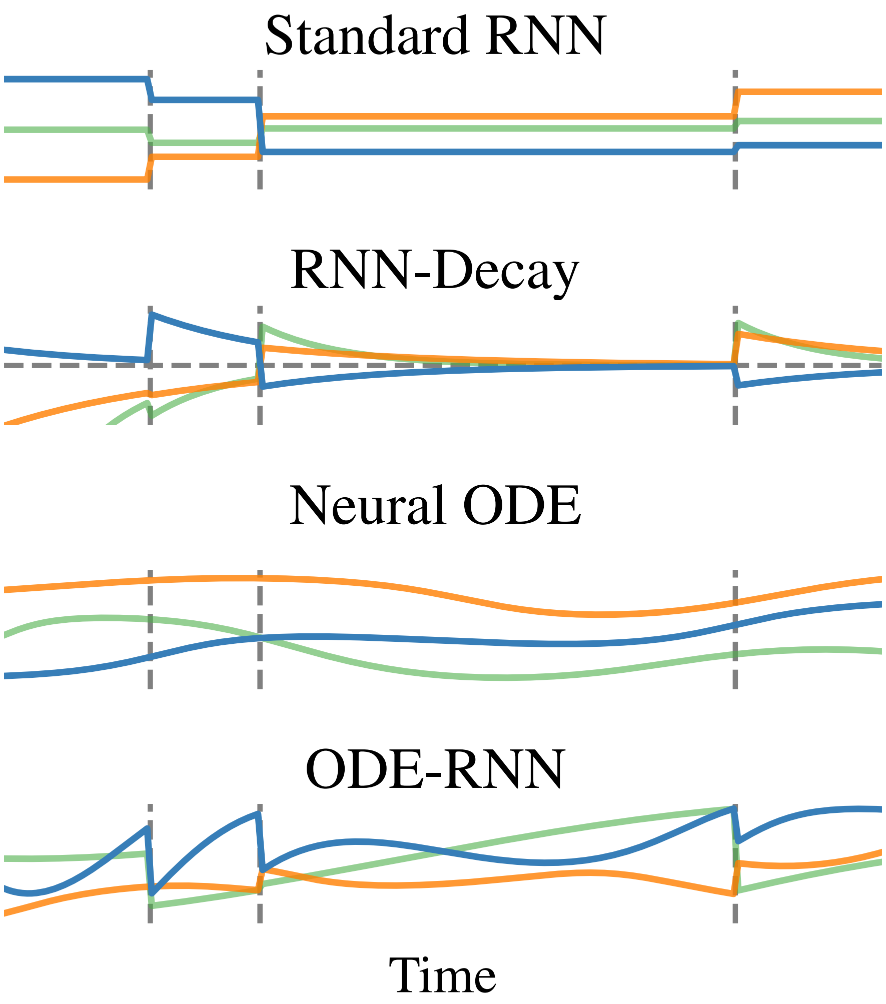
NODE-RNN

- Standard RNN cell update:

$$h_t = \mathrm{RNN}(h_{t-\Delta t}, x_t)$$

- NODE-RNN cell update:

$$\tilde{h}_t = \mathrm{ODESolve}(h_{t-\Delta t}, \Delta t)$$
$$h_t = \mathrm{RNN}(\tilde{h}_t, x_t)$$



Source: "Latent ODEs for Irregularly-Sampled Time Series", NeurIPS'20

Implicit Neural Representations

Basics

Implicit Neural Representations (INRs) model a signal as a continuous function of time:

$$
\begin{aligned}
f_\theta : \mathbb{R} &\to \mathbb{R}^d \\
t &\mapsto f_\theta(t)
\end{aligned}
$$

A common choice for f_θ is an MLP.

Time Embedding layer

- Representing time as a 1D feature is a weak representation
- In practice, an INR's first layer is often a positional encoding, eg. Fourier features (random or learnt):

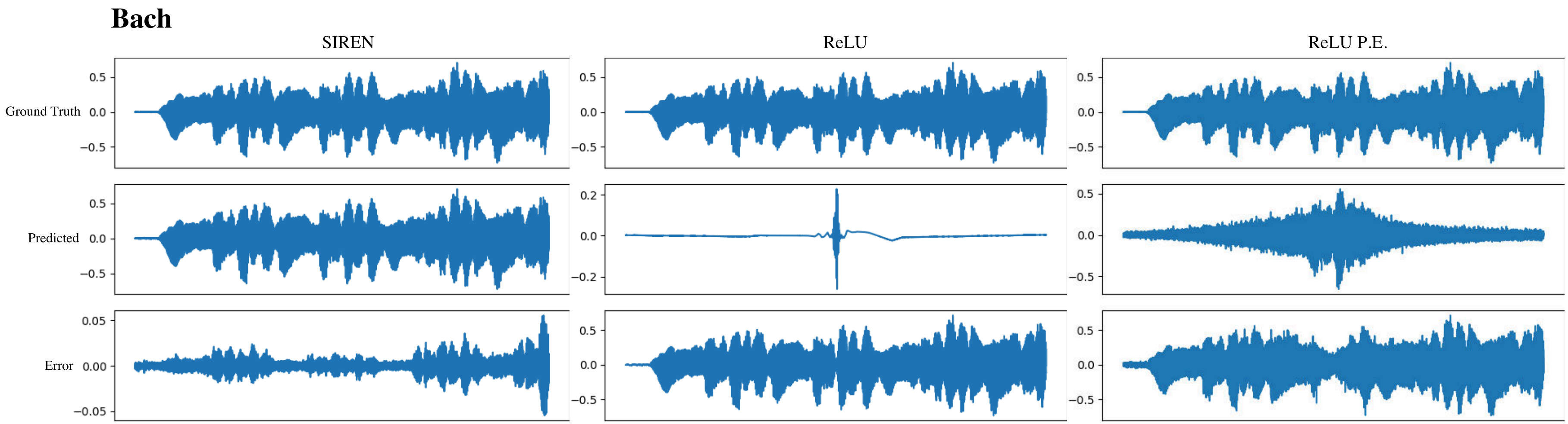
$$h_1(t) = [\sin(\omega_1 t), \cos(\omega_1 t), ..., \sin(\omega_K t), \cos(\omega_K t)]$$

SInusoidal REpresentation Networks (SIRENs)

- Use sine activation functions:

$$\Phi(x) = \sin(W \cdot x + b)$$

- Act as a learnable Fourier basis decomposition
- Have non-zero second-order derivatives



Source: “Implicit Neural Representations with Periodic Activation Functions”, NeurIPS’20

Modulated INRs

- Such INRs can learn to extrapolate a single time series:

$$\begin{aligned} f_\theta : \mathbb{R} &\to \mathbb{R}^d \\ t &\mapsto f_\theta(t) \end{aligned}$$

- In practice, we have a dataset of time series, hence the formulation:

$$\begin{aligned} f_\theta : \mathbb{R} \times \mathbb{R}^p &\to \mathbb{R}^d \\ t, z &\mapsto f_\theta(t, z) \end{aligned}$$

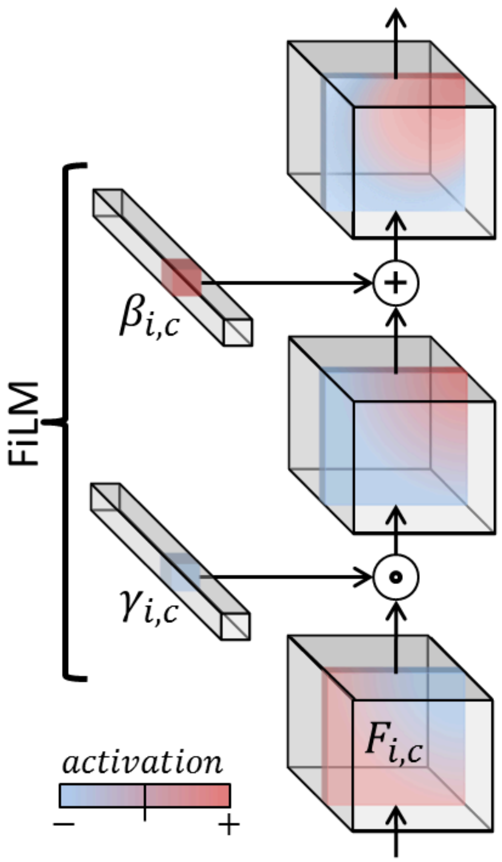
  where z is a code summarizing the content of the time series

- **Modulated INRs**: z modulates the behaviour of f_θ (activations or weights)

Feature-Wise Linear Modulation (FiLM)

1. The time series is encoded as z
2. A modulation network (shallow MLP) outputs modulation parameters $\gamma(z)$ and $\beta(z)$ for each INR layer
3. These parameters are used to modulate the INR **activations**

$$h^{\text{modulated}} = \gamma(z) \odot h^{\text{INR}} + \beta(z)$$



Source: “FiLM […]”, AAAI’18

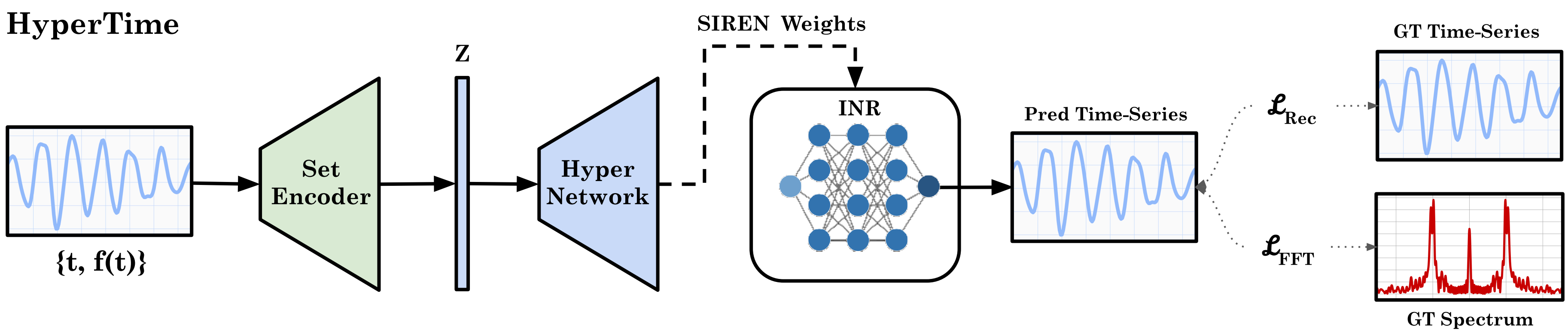
Hypernetworks

- Modulation at the INR **parameter** level:
  1. A hypernetwork learns per-parameter modulations $\psi(z)$
  2. The INR (hyponetwork) is now:

$$
\begin{aligned}
f_{\theta,\psi} : \mathbb{R} \times \mathbb{R}^p &\to \mathbb{R}^d \\
t, z &\mapsto f_{\theta+\psi(z)}(t)
\end{aligned}
$$

HyperTime

- Generates an encoding z per timestamp
- Requires global pooling along the time axis (or fixed number of observations per series)
- Uses an additional frequency-based loss

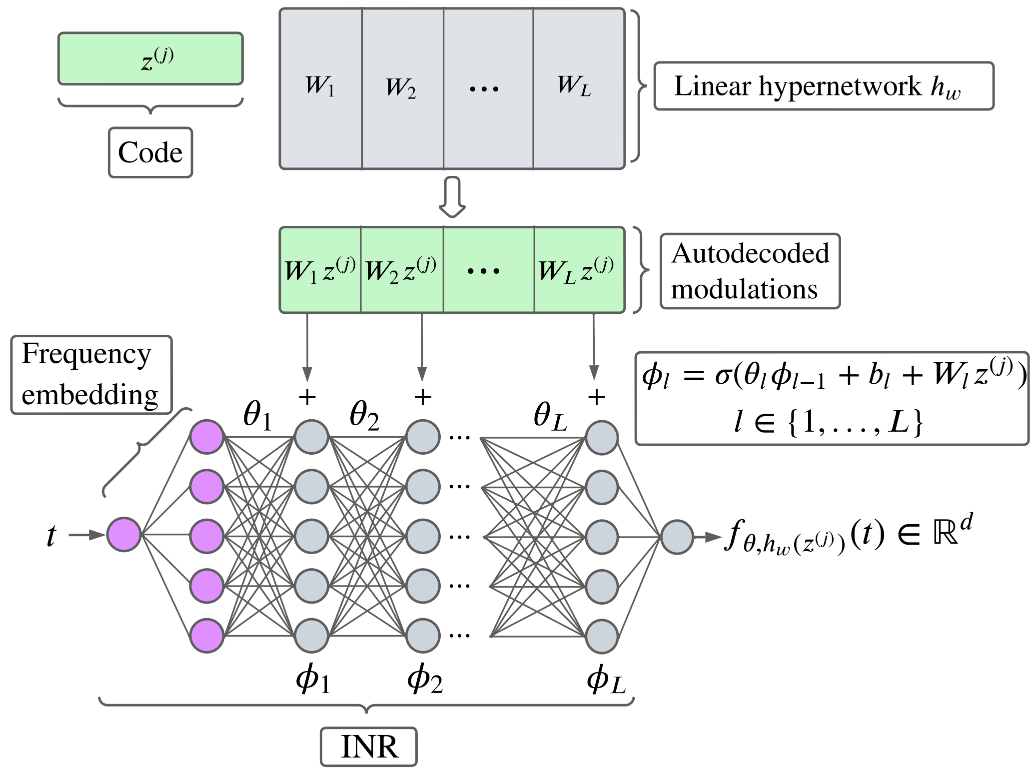


Source: “HyperTime: Implicit Neural Representation for Time Series”, NeurIPS Workshop, 2022

TimeFlow

- Activation modulations (*à la* FiLM)
- Code z is optimized through few-step gradient descent (not output by an encoder)

  $\rightarrow$ no constraint on the input time grid, no need for pooling



Source: "Time Series Continuous Modeling for Imputation and Forecasting with Implicit Neural Representations", TMLR'24