Neural Ordinary Differential Equations

Nicola Farenga

Supervisor: Prof. Stefania Fresca

Computational Statistics (Prof. Andrea Manzoni)
M.Sc. Mathematical Engineering, Politecnico di Milano

March 1th, 2022

Table of Contents

1. Introduction
2. Application I: Spiral ODE
3. Application II: Quantifying NODEs Predictions' Uncertainty
4. Application III: Parametrized Neural ODE

Deep Implicit Layers

Modern deep learning architectures are made by a stack of multiple layers, whose computational structure is explicitly defined, so that, given the input x, the output y of a layer $f : \mathcal{X} \times \mathbb{R}^{|\theta|} \to \mathcal{Y}$ can be directly computed as

$$y = f(x; \theta)$$

Deep Implicit Layers

Modern deep learning architectures are made by a stack of multiple layers, whose computational structure ~~is explicitly defined~~ can be **implicitly** defined, so that the output y of a layer $f : \mathcal{X} \times \mathbb{R}^{|\theta|} \to \mathcal{Y}$ can be computed by

$$\text{Find } y \ : \ f(x;\theta) - y = 0$$

Deep Implicit Layers

Modern deep learning architectures are made by a stack of multiple layers, whose computational structure ~~is explicitly defined~~ can be **implicitly** defined, so that the output y of a layer $f : \mathcal{X} \times \mathbb{R}^{|\theta|} \to \mathcal{Y}$ can be computed by

$$\text{Find } y \ : \ g(x, y; \theta) = 0$$

with $g : \mathcal{X} \times \mathcal{Y} \times \mathbb{R}^{|\theta|} \to \mathbb{R}^n$.

Deep Implicit Layers

Modern deep learning architectures are made by a stack of multiple layers, whose computational structure ~~is explicitly defined~~ can be **implicitly** defined, so that the output y of a layer $f : \mathcal{X} \times \mathbb{R}^{|\theta|} \to \mathcal{Y}$ can be computed by

$$\text{Find } y \ : \ g(x, y; \theta) = 0$$

with $g : \mathcal{X} \times \mathcal{Y} \times \mathbb{R}^{|\theta|} \to \mathbb{R}^n$.

$\Rightarrow$ *Root finding problem* $\to$ Bisection method, Newton's method...

Deep Implicit Layers

Modern deep learning architectures are made by a stack of multiple layers, whose computational structure ~~is explicitly defined~~ can be **implicitly** defined, so that the output y of a layer $f : \mathcal{X} \times \mathbb{R}^{|\theta|} \to \mathcal{Y}$ can be computed by

$$\text{Find } y \ : \ g(x, y; \theta) = 0$$

with $g : \mathcal{X} \times \mathcal{Y} \times \mathbb{R}^{|\theta|} \to \mathbb{R}^n$.

$\Rightarrow$ *Root finding problem* $\to$ Bisection method, Newton's method...

$\Rightarrow$ Decouple the *definition* of the layer and from its *computational structure*.

Deep Implicit Layers

Modern deep learning architectures are made by a stack of multiple layers, whose computational structure ~~is explicitly defined~~ can be **implicitly** defined, so that the output y of a layer $f : \mathcal{X} \times \mathbb{R}^{|\theta|} \to \mathcal{Y}$ can be computed by

$$\text{Find } y \ : \ \text{arbitrary condition on } x, y \text{ is satisfied}$$

Deep Implicit Layers

Modern deep learning architectures are made by a stack of multiple layers, whose computational structure ~~is explicitly defined~~ can be **implicitly** defined, so that the output y of a layer $f : \mathcal{X} \times \mathbb{R}^{|\theta|} \to \mathcal{Y}$ can be computed by

$$\text{Find } y \ : \ \text{arbitrary condition on } x, y \text{ is satisfied}$$

Introduced in the 80s, during the developement of *recurrent backpropagation*, when fixed-point equations layers (Almeida 1987) and differential equations layers (Pineda 1987) first appeared.

Deep Implicit Layers

Modern deep learning architectures are made by a stack of multiple layers, whose computational structure ~~is explicitly defined~~ can be **implicitly** defined, so that the output y of a layer $f : \mathcal{X} \times \mathbb{R}^{|\theta|} \to \mathcal{Y}$ can be computed by

$$\text{Find } y \ : \ \text{arbitrary condition on } x, y \text{ is satisfied}$$

Introduced in the 80s, during the developement of *recurrent backpropagation*, when fixed-point equations layers (Almeida 1987) and **differential equations layers** (Pineda 1987) first appeared.

Deep Implicit Layers

Ordinary Differential Equations Layers

Modern deep learning architectures are made by a stack of multiple layers, whose computational structure ~~is explicitly defined~~ can be **implicitly** defined, so that the output y of a layer $f : \mathcal{X} \times \mathbb{R}^{|\theta|} \to \mathcal{Y}$ can be computed by

$$\text{Find } y(t^*) \;:\; \frac{dy}{dt}(t) = f(t, y(t); \theta), \qquad y(0) = y_0$$

Deep Implicit Layers

Ordinary Differential Equations Layers

Modern deep learning architectures are made by a stack of multiple layers, whose computational structure ~~is explicitly defined~~ can be **implicitly** defined, so that the output y of a layer $f : \mathcal{X} \times \mathbb{R}^{|\theta|} \to \mathcal{Y}$ can be computed by

$$\text{Find } y(t^*) \;:\; \frac{dy}{dt}(t) = f(t, y(t); \theta), \qquad y(0) = y_0$$

requiring the IVP is satisfied, where $f : \mathbb{R} \times \mathbb{R}^d \times \mathbb{R}^{|\theta|} \to \mathbb{R}^d$ is a neural network, $\theta = \{(W_l, b_l)_{l=1}^{L}\}$ is the set of learnable parameters of its L-layers, and $y_0 \in \mathbb{R}^d$.

$\Rightarrow$ **Neural Ordinary Differential Equation**.

Derivating Neural ODEs from ResNets

Continuous-depth architectures

The t-th block in a ResNet architecture consists of

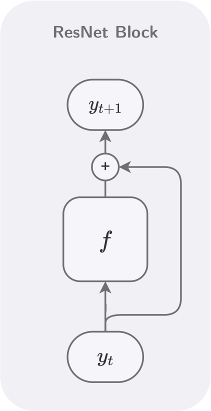
$$y_{t+1} = y_t + f(y_t; \theta_t)$$

and resambles the Explicit Euler's Method discretization with $h = 1$

$$y_{t+1} - y_t = hf(y_t; \theta_t)$$

corresponding to the continuous formulation

$$\frac{dy}{dt}(t) = f(t, y(t); \theta)$$



Computing the output of an ODE-Layer

In order to compute the output of the implicitly defined ODE-layers we rely on numerical integration methods

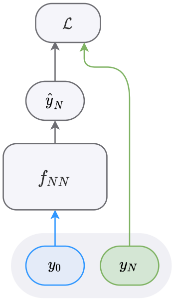
$$y_{t_0}, \ldots, y_{t_N} = \mathrm{ODEint}(f, y_0, [t_0, \ldots, t_N])$$

implemented in the `torchdiffeq` library, together with the *Adjoint Backpropagation Method* (Chen et al. 2018).

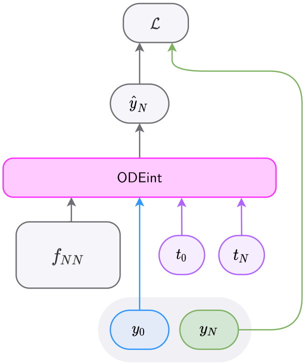
Architectures differences

Backpropagation in ODE-Nets

FFNN



ODE-Net

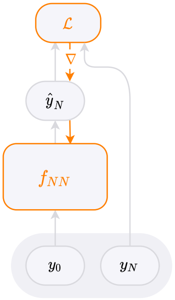


Architectures differences

Backpropagation in ODE-Nets

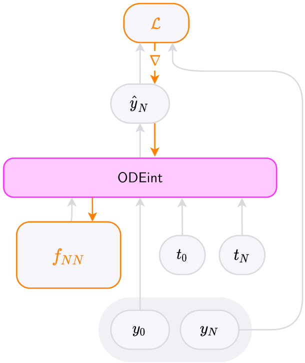
FFNN

$$\mathcal{L}(\hat{y}_N) = \mathcal{L}(f_{NN}(y_0;\theta))$$



ODE-Net

$$\mathcal{L}(\hat{y}_N) = \mathcal{L}(\texttt{odeint}(f_{NN,\theta}, y_0, t_0, t_N))$$



Architectures differences

Backpropagation in ODE-Nets

Adjoint Backpropagation Method (Chen et al. 2018) can be employed for backpropagating through the ODE solver:

Input: Params θ, start time t_0, stop time t_N, final state $y(t_N)$, loss gradient $\frac{\partial \mathcal{L}}{\partial y(t_N)}$

$s_0 = [y(t_N), \frac{\partial \mathcal{L}}{\partial y(t_N)}, 0_{|\theta|}]$

def `AugmentedDynamics`$([y(t), a(t), \cdot], t, \theta)$**:**

  return $[f(y(t), t, \theta), -a(t)^T \frac{\partial f}{\partial y}, -a(t)^T \frac{\partial f}{\partial \theta}]$;

$[y(t_0), \frac{\partial \mathcal{L}}{\partial y(t_0)}, \frac{\partial \mathcal{L}}{\partial \theta}] = \mathrm{ODEint}(\texttt{Augmented Dynamics}, s_0, t_N, t_0, \theta)$

return $\frac{\partial \mathcal{L}}{\partial y(t_0)}, \frac{\partial \mathcal{L}}{\partial \theta}$;

Architectures differences

Backpropagation in ODE-Nets

Adjoint Backpropagation Method (Chen et al. 2018) can be employed for backpropagating through the ODE solver:

Input: Params θ, start time t_0, stop time t_N, final state $y(t_N)$, loss gradient $\frac{\partial \mathcal{L}}{\partial y(t_N)}$

$s_0 = [y(t_N), \frac{\partial \mathcal{L}}{\partial y(t_N)}, 0_{|\theta|}]$

def $\texttt{AugmentedDynamics}([y(t), a(t), \cdot], t, \theta)$:

  return $[f(y(t), t, \theta), -a(t)^T \frac{\partial f}{\partial y}, -a(t)^T \frac{\partial f}{\partial \theta}]$;

$[y(t_0), \frac{\partial \mathcal{L}}{\partial y(t_0)}, \frac{\partial \mathcal{L}}{\partial \theta}] = \text{ODEint}(\texttt{Augmented Dynamics}, s_0, t_N, t_0, \theta)$

return $\frac{\partial \mathcal{L}}{\partial y(t_0)}, \frac{\partial \mathcal{L}}{\partial \theta}$;

Practically we already know how to backpropagate through some solvers since they can be viewed as a ResNet (i.e. Euler).

Learning Latent ODE

Framework

The aim of the project is to employ Neural ODEs to learn the latent dynamics of time-dependent processes, by exploiting NODEs' continuous-depth (time-continuous) architecture:

$$\dot{\mathrm{u}} = \mathrm{f}(t, \mathrm{u}(t); \mu)$$

Learning Latent ODE

Framework

The aim of the project is to employ Neural ODEs to learn the latent dynamics of time-dependent processes, by exploiting NODEs' continuous-depth (time-continuous) architecture:

$$\dot{\mathrm{u}} = \mathrm{f}(t, \mathrm{u}(t); \mu) \simeq \mathrm{f}_{NN}(t, \mathrm{u}(t); \theta)$$

Learning Latent ODE

Framework

The aim of the project is to employ Neural ODEs to learn the latent dynamics of time-dependent processes, by exploiting NODEs' continuous-depth (time-continuous) architecture:

$$\dot{\mathrm{u}} = \mathrm{f}(t, \mathrm{u}(t); \mu) \simeq \mathrm{f}_{NN}(t, \mathrm{u}(t); \theta)$$

Observation: the learned function has direct interpretation, we will refer to it as the *learned vector field*.

Spiral ODE

Application I - Framework

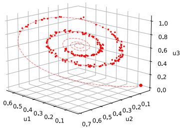
Lorenz ODE-IVP

$$\begin{cases} \dot{u}_1 = \sigma(u_2 - u_1) \\ \dot{u}_2 = u_1(\rho - u_3) - u_2 \\ \dot{u}_3 = u_1 u_2 - \beta u_3 \\ \mathbf{u}(0) = \mathbf{u}_0 \end{cases}$$

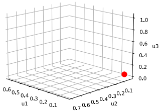
$$\sigma = 14,\ \rho = 10, \beta = 8/3$$

Synthetic Generation of Evenly & Irregularly Spaced Noisy Data

$$\mathcal{D} = \{x_i, y_i\}_{i=1}^{n_b(=20)}$$
$$x_i = u(t_0)$$
$$y_i = [u(t_1), \dots, u(t_N)]$$
$$\epsilon \sim \mathcal{N}(0, 1\text{e-}3)$$



Train

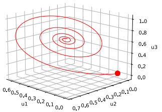


Initial State

ODE-Block: Linear | Tanh | Linear

ODEint

ODE-Net



Predicted Dynamics

Spiral ODE

Application I - Framework

Lorenz ODE-IVP

$$\begin{cases} \dot{u}_1 = \sigma(u_2 - u_1) \\ \dot{u}_2 = u_1(\rho - u_3) - u_2 \\ \dot{u}_3 = u_1 u_2 - \beta u_3 \\ \mathbf{u}(0) = \mathbf{u}_0 \end{cases}$$

$$\sigma = 14,\ \rho = 10, \beta = 8/3$$

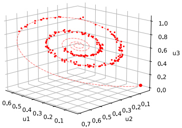
Synthetic Generation of Evenly & Irregularly Spaced Noisy Data

$\mathcal{D} = \{x_i, y_i\}_{i=1}^{n_b(=20)}$

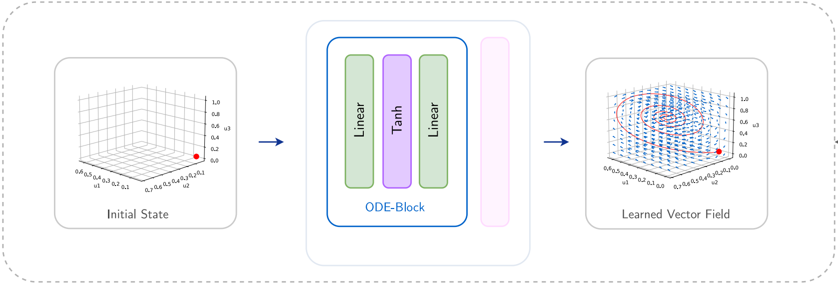
$x_i = u(t_0)$

$y_i = [u(t_1), \dots, u(t_N)]$

$\epsilon \sim \mathcal{N}(0, 1\text{e-}3)$



Train



Integration Methods Comparison

Application I - Training

		Evenly spaced				Irregularly spaced			
	Method	**Training time [min]**	**Time/Epoch [s]**	**MSE**	**Max NFE**	**Training time [min]**	**Time/Epoch [s]**	**MSE**	**Max NFE**
Fixed Step	Euler	7.721	0.231	7.62e-03	199	7.987	0.239	9.17e-03	199
	Euler+Adjoint	17.283	0.518	1.37e-02	398	17.339	0.520	1.45e-02	398
	RK4	22.984	0.689	6.31e-03	796	23.838	0.715	1.12e-02	796
	RK4+Adjoint	25.541	0.765	1.02e-02	1592	25.612	0.768	1.12e-02	1592
Adaptive Step	Bosh3	39.452	1.183	4.89e-03	1007	58.671	1.760	4.91e-03	1883
	Bosh3+Adjoint	112.364	3.370	2.88e-03	3328	140.651	4.219	4.16e-03	5230
	DorPri5	12.263	0.367	2.83e-03	212	17.311	0.519	3.70e-03	380
	DorPri5+Adjoint	73.300	2.198	1.10e-02	1786	87.391	2.621	6.43e-03	2578

Integration Methods Comparison

Application I - Training

		Evenly spaced				Irregularly spaced			
	Method	**Training time [min]**	**Time/Epoch [s]**	**MSE**	**Max NFE**	**Training time [min]**	**Time/Epoch [s]**	**MSE**	**Max NFE**
Fixed Step	Euler	7.721	0.231	7.62e-03	199	7.987	0.239	9.17e-03	199
	Euler+Adjoint	17.283	0.518	1.37e-02	398	17.339	0.520	1.45e-02	398
	RK4	22.984	0.689	6.31e-03	796	23.838	0.715	1.12e-02	796
	RK4+Adjoint	25.541	0.765	1.02e-02	1592	25.612	0.768	1.12e-02	1592
Adaptive Step	Bosh3	39.452	1.183	4.89e-03	1007	58.671	1.760	4.91e-03	1883
	Bosh3+Adjoint	112.364	3.370	2.88e-03	3328	140.651	4.219	4.16e-03	5230
	DorPri5	12.263	0.367	2.83e-03	212	17.311	0.519	3.70e-03	380
	DorPri5+Adjoint	73.300	2.198	1.10e-02	1786	87.391	2.621	6.43e-03	2578

- Adjoint Backprop Method: up to 4x slower training & degradated predictive performances.

Integration Methods Comparison

Application I - Training

		Evenly spaced				Irregularly spaced			
	Method	**Training time [min]**	**Time/Epoch [s]**	**MSE**	**Max NFE**	**Training time [min]**	**Time/Epoch [s]**	**MSE**	**Max NFE**
Fixed Step	Euler	7.721	0.231	7.62e-03	199	7.987	0.239	9.17e-03	199
	Euler+Adjoint	17.283	0.518	1.37e-02	398	17.339	0.520	1.45e-02	398
	RK4	22.984	0.689	6.31e-03	796	23.838	0.715	1.12e-02	796
	RK4+Adjoint	25.541	0.765	1.02e-02	1592	25.612	0.768	1.12e-02	1592
Adaptive Step	Bosh3	39.452	1.183	4.89e-03	1007	58.671	1.760	4.91e-03	1883
	Bosh3+Adjoint	112.364	3.370	2.88e-03	3328	140.651	4.219	4.16e-03	5230
	DorPri5	12.263	0.367	2.83e-03	212	17.311	0.519	3.70e-03	380
	DorPri5+Adjoint	73.300	2.198	1.10e-02	1786	87.391	2.621	6.43e-03	2578

- Adjoint Backprop Method: up to 4x slower training & degradated predictive performances.
- Dormand-Prince 5 has been selected due to training performances and low MSE.

ODE-Block: Linear/Tanh/Linear, 50 hidden units per layer.
Data noise $\varepsilon \sim \mathcal{N}(0, 1\text{e-}3)$ added to globally max-scaled data. Irregular spacing obtained by uniformly sampling from a fixed batch time interval of 200 time-steps.

ODE-Block Configurations

Application I - Hyperparameters Tuning

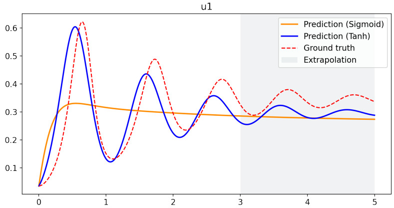
ODE-Block structure:

Layer	Input Size	Output Size
1	3	n hidden units
	activation	
2	n hidden units	3

Tested configurations:

- Hidden Units: 20, 50, 100
- Activations: Sigmoid, Tanh

Activation	Hidden units	Time/Epoch [s]	Train MSE	Test MSE
Sigmoid	20	0.151	1.58e-02	1.51e-03
	50	0.336	2.45e-02	2.01e-03
	100	0.350	2.68e-02	3.97e-03
Tanh	20	0.401	1.99e-02	1.88e-03
	50	0.453	2.38e-03	8.97e-04
	100	0.484	6.96e-03	8.24e-04

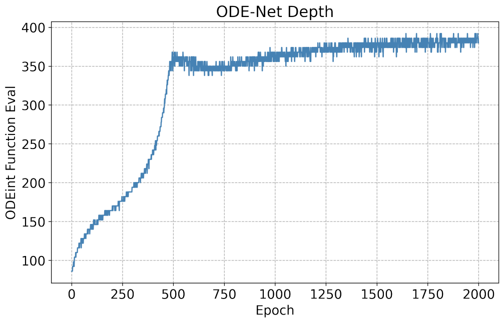


Sigmoid with 50 hidden units underfits.

What about ODE-Nets' depth?

Application I

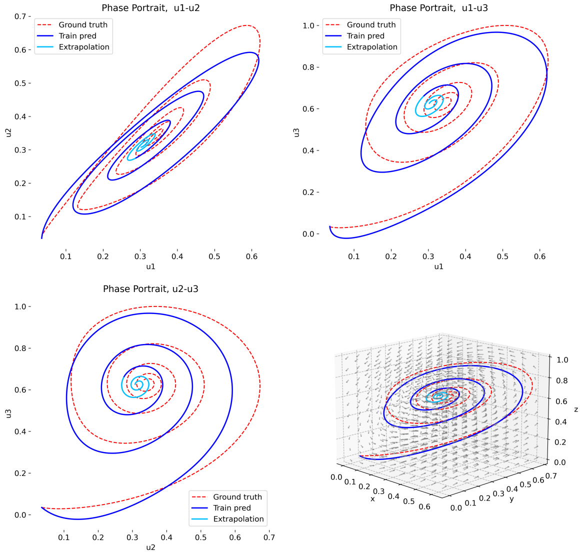
ODE-Nets' depth depends on the number of function evaluations taken by the adaptive-step solver, and it grows as the training proceeds, resulting in 4x deeper networks, wrt classical ResNets.



Observations

Application I

- ODE-Nets are able to capture the general behavior of the true underlying dynamics.
- A major role is played by the employed integration method, both in terms of training performances and predictive capabilities.



Augmenting Neural ODE Framework

1. Improve predictive accuracy and robustness wrt irregularly sampled data
2. Quantify predictions' uncertainty

A natural way to model uncertainty in a NN-framework is to employ Bayesian Neural Networks (BNNs), enabling to model weights and biases as distributions.

Problem: BNNs are computationally demanding, thus the training process is critically slowed down when dealing with very deep architectures (like ODE-Nets).

Solution: Employ an *ensembling*-based approximated Bayesian NNs framework.

Approximately Bayesian Ensembling

The framework employed, introduced by Pearce et al. 2018, exploits the fact that by adding a regularization term to the loss, the parameters that minimize the regularized loss can be interpreted as MAP estimates.

$$\mathcal{L}(\mathrm{y}_i, \hat{\mathrm{y}}_i) = \|\mathrm{y}_i - \underbrace{\texttt{odeint}(f_\theta, x_i, t_0, t_N)}_{\hat{\mathrm{y}}_i}\|^2 + \lambda\|\theta\|^2$$

Problem: Regularization reduces diversity between differently initialized models, reducing the ensemble's capabilities to capture uncertainty.

Anchored Bayesian Ensembling

The framework employed, introduced by Pearce et al. 2018, exploits the fact that by adding a regularization term to the loss, the parameters that minimize the regularized loss can be interpreted as MAP estimates.

$$\mathcal{L}_{\text{anc}}(\mathrm{y}_i, \hat{\mathrm{y}}_i) = \|\mathrm{y}_i - \underbrace{\texttt{odeint}(f_\theta, x_i, t_0, t_N)}_{\hat{\mathrm{y}}_i}\|^2 + \lambda\|\theta - \theta_{\text{anc}}\|^2$$

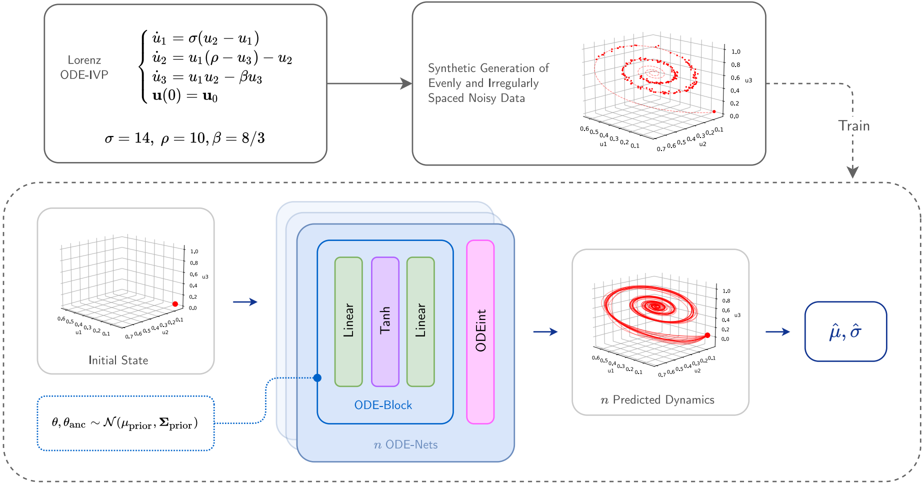
Problem: Regularization reduces diversity between differently initialized models, reducing the ensemble's capabilities to capture uncertainty.

Solution: Introduce an anchoring r.v. $\theta_{\text{anc}} \sim \mathcal{N}(\mu_{\text{prior}}, \Sigma_{\text{prior}})$ to:

- inject noise into the regularization term, ensuring diversity
- anchor the learned distribution to the prior

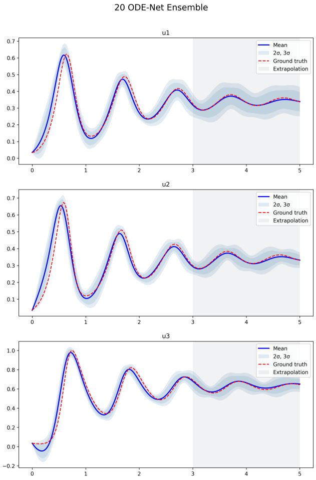
Anchored Bayesian Ensembling

Application II - Framework



Results

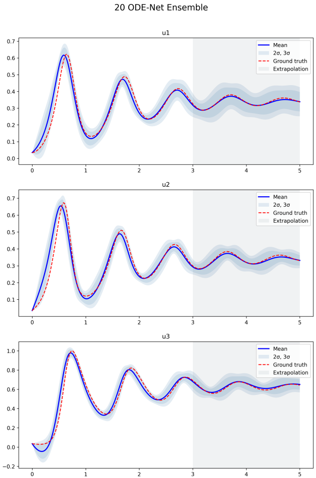
Application II



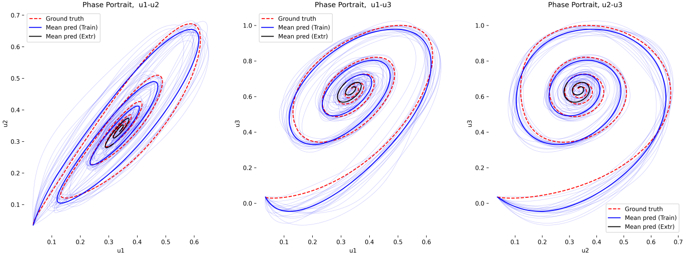
σ^2_{prior}	**Ensemble Size**	**Training Time**	**Mean Prediction** Train MSE	**Mean Prediction** Test MSE	**% within $\pm 2\sigma$** u_1	**% within $\pm 2\sigma$** u_2	**% within $\pm 2\sigma$** u_3
1	3	∼45 min	8.39e-03	7.33e-04	40.2%	62.2%	23.8%
	5	∼1h 15 min	1.06e-02	8.31e-04	39.7%	58.4%	32.0%
	10	∼2h 30 min	1.01e-02	8.90e-04	34.5%	51.4%	27.3%
	20	∼5h	1.05e-02	8.52e-04	34.2%	52.6%	29.0%
20	3	∼45 min	3.17e-03	1.38e-03	80.4%	83.9%	72.7%
	5	∼1h 15 min	2.43e-03	1.24e-03	85.6%	81.4%	79.6%
	10	∼2h 30 min	2.27e-03	9.02e-05	88.9%	83.0%	78.5%
	20	∼5h	2.30e-03	5.36e-05	94.5%	88.3%	87.7%

Results

Application II



σ^2_{prior}	Ensemble Size	Training Time	Mean Prediction Train MSE	Mean Prediction Test MSE	% within $\pm 2\sigma$ u_1	% within $\pm 2\sigma$ u_2	% within $\pm 2\sigma$ u_3
20	3	∼45 min	3.17e-03	1.38e-03	80.4%	83.9%	72.7%
	5	∼1h 15 min	2.43e-03	1.24e-03	85.6%	81.4%	79.6%
	10	∼2h 30 min	2.27e-03	9.02e-05	88.9%	83.0%	78.5%
	20	∼5h	2.30e-03	5.36e-05	94.5%	88.3%	87.7%



Parametrized Neural ODE

Application III - Framework

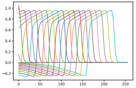
1-dimensional ODE-PDE non-linear system

Monodomain Equation + FitzHugh-Nagumo Cellular Model

$$
\begin{cases}
\mu \dfrac{\partial u}{\partial t} - \mu^2 \dfrac{\partial^2 u}{\partial x^2} + u(u-0.1)(u-1) + w = 0 & (x,t) \in (0,L) \times (0,T) \\
\dfrac{dw}{dt} + (\gamma w - \beta u) = 0 & (x,t) \in (0,L) \times (0,T) \\
\dfrac{\partial u}{\partial x}(0,t) = 5000 t^3 e^{-15t} & t \in (0,T) \\
\dfrac{\partial u}{\partial x}(L,t) = 0 & t \in (0,T) \\
u(x,0) = 0, \ w(x,0) = 0 & x \in (0,L)
\end{cases}
$$

$L = 1, \ T = 2, \ \beta = 0.5, \ \mu \in \mathcal{P} = 5 \cdot [10^{-3}, 10^{-2}]$

Synthetic Data Generation

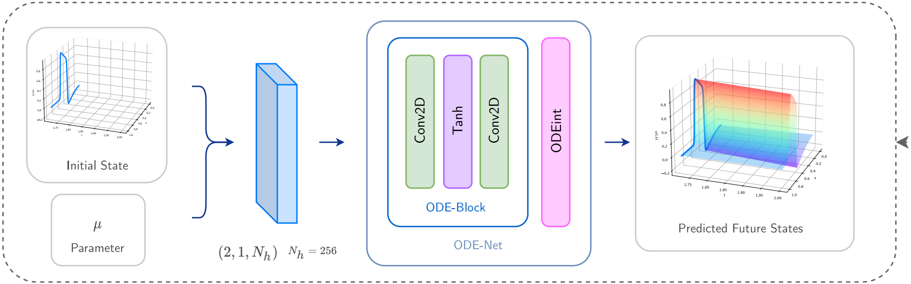


$\mathcal{D} = \{x_i, y_i\}_{i=1}^{n_b (=20)}$

$u(t_i) \in \mathbb{R}^{N_h}, \ N_h = 256$

$x_i = [u(t_0), \mu], \quad y_i = [u(t_1), \dots, u(t_N)]$

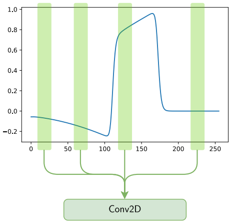
Train

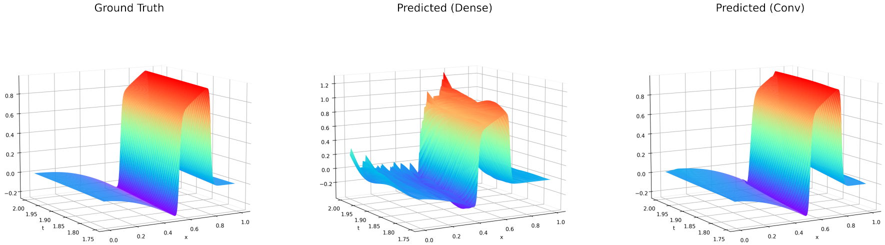


Parametrized ODE-Block Architecture

Application III

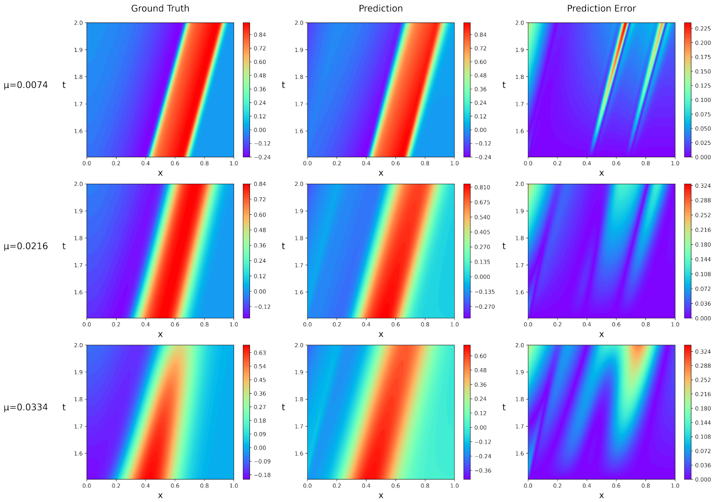
Layer	Input Shape	Output Shape	Kernel Size	n Filters	Stride	Padding
1	[2,1,256]	[16,1,256]	[1,3]	16	1	Same
Tanh						
2	[16,1,256]	[1,1,256]	[1,3]	1	1	Same





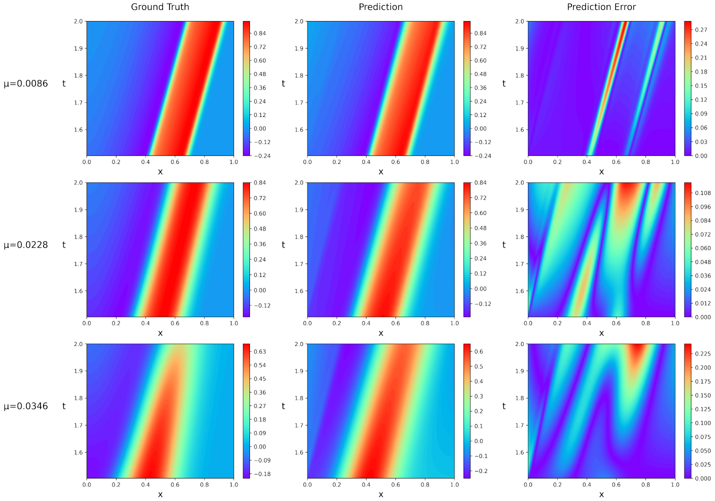
Results - Time Extrapolation

Application III



Results - Out-of-$\mathcal{P}$ Predictions

Application III



Conclusions

- Neural ODEs provide interpretability together with performances comparable to SotA DL approaches.
- *Ensembling*-based techniques (e.g. ABE) allow to both quantify predictions' uncertanity and to achieve better predictive performances.
- Parametrized Neural ODEs represent a promising *data-driven* approach to build surrogate models in the field of model-order reduction.

Thanks!

- Almeida, Luis B. (1987). *A learning rule for asynchronous perceptrons with feedback in a combinatorial environment*.
- Chen, Ricky T. Q. et al. (2018). *Neural Ordinary Differential Equations*.
- Duvenaud, David, J. Zico Kolter, and Matt Johnson (2020). *Deep Implicit Layers: Neural ODEs, Equilibrium Models, and Beyond*.
- Fresca, Stefania, Luca Dedé, and Andrea Manzoni (2021). *A Comprehensive Deep Learning-Based Approach to Reduced Order Modeling of Nonlinear Time-Dependent Parametrized PDEs*.
- Kidger, Patrick (2022). *On Neural Differential Equations*.
- Pearce, Tim et al. (2018). *Uncertainty in Neural Networks: Approximately Bayesian Ensembling*.
- Pineda, Fernando (1987). *Generalization of Back-Propagation to Recurrent Neural Networks*.