System Intelligence Laboratory

Snakes and Sirens: Can Sines Help Deep Learning?

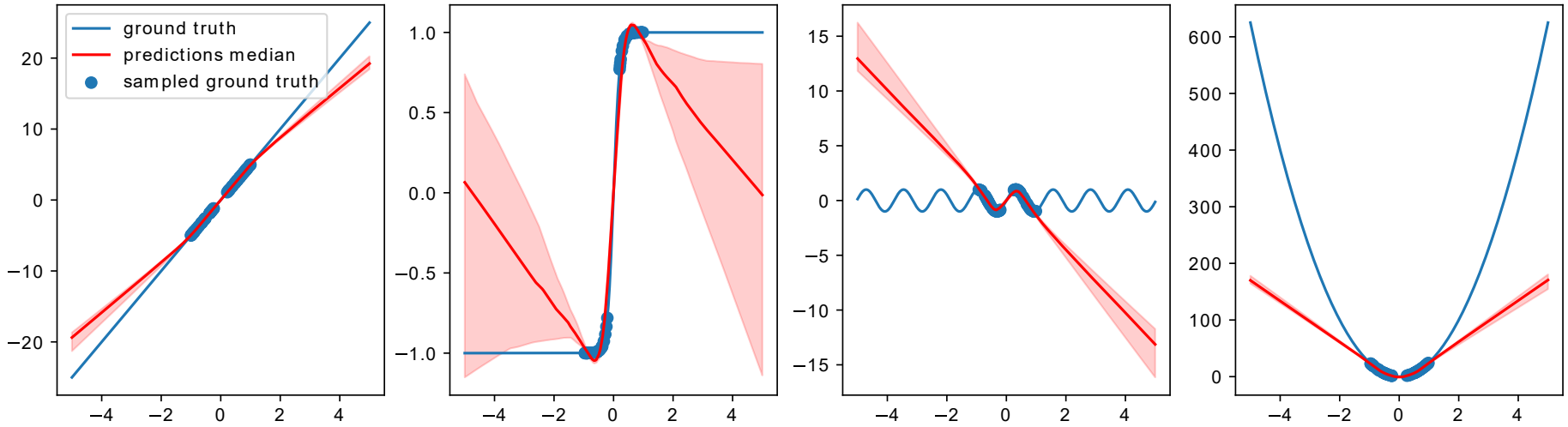
Federico Berto

Contents

- Neural Networks Fail to Learn Periodic Functions and How to Fix It (NeurIPS 2020)
- Implicit Neural Representations with Periodic Activation Functions (NeurIPS 2020, Oral)
- Extra material (referenced during presentation)

Motivation

- Activation functions are rarely taken into account in model design
- Certain functions e.g. ReLU perform well while *interpolating*
- How about *extrapolating*? Can we learn well outside of the boundaries?



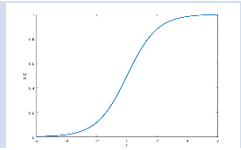
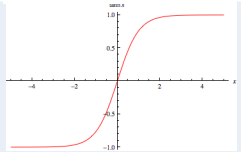
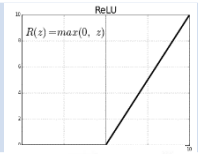
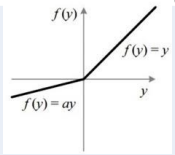
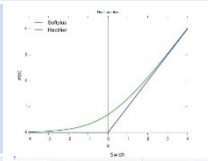
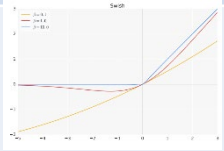
ReLU interpolation and extrapolation with different functions

Activation Fuctions

- Activation functions $\sigma(x)$ define outputs of a network
  - We insert *non-linearities* to represent complex functions
  - Otherwise, deep neural network becomes an affine transformation (i.e. linear + bias)

- Most commonly used activations:
  - Sigmoid
  - Tanh
  - ReLU
  - Leaky ReLU
  - Softplus
  - Swish

Overview of Commonly Used Activation Fuctions

Function	Formula	Graph	Range	Saturation	Computation	$\mathcal{C}^1$	Extrapolation
Sigmoid	$\frac{1}{1+e^{-x}}$		$[0,1]$	Negative and positive values	Intensive	✓	✗
Tanh	$\frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$		$[-1,1]$	Negative and positive values	Intensive	✓	✗
ReLU	$\begin{cases} x, if\ x>0 \\ 0, otherwise \end{cases}$		$[0,+\infty]$	Negative values (*dying ReLU*)	Easy	✗	✗
Leaky ReLU	$\begin{cases} x, if\ x>0 \\ ax, otherwise \end{cases}$		$[-\infty,+\infty]$	None	Easy	✗	✗
Softplus	$\ln(1+e^{x})$		$[-\infty,+\infty]$	Large negative values	Intensive	✓	✗
Swish	$\frac{x}{1+e^{-\beta x}}$		$[-\infty,+\infty]$	Depending on β	Intensive	✓	✗

Why can’t these functions extrapolate?

Given a scalar *z* and a unit vector *u*:

- Theorem for Tanh:

$$\lim_{z\to\infty} \| f_{tanh}(zu) - v_u \|_2 = 0$$

where v_u is a constant vector depending on *u* → Tanh converges to a constant!

- Theorem for ReLU:

$$\lim_{z\to\infty} \| f_{ReLU}(zu) - zW_u u - b_u \|_2 = 0$$

where W_u is a constant matrix depending on *u* → ReLU converges to a linear transformation!

→ This results holds true for functions in the ReLU family as well e.g. Leaky ReLU and Swish

Sinusoidal Activation Functions

- Some work has been done on them such as:

  - Sinusoidal activations[1]

$$\sigma(x) = \sin(ax)$$

  - Fourier Neural Networks[2]

$$\sigma_i(x) = \alpha_i \cos(A_i x + b_i) + \beta_i \sin(C_i x + d_i)$$

where $\sigma_i(x)$ is a single activation function of the network:

$$f_L : x \to \gamma_0 + \sum_{i=0}^{n} \alpha_i \cos(A_i x + b_i) + \beta_i \sin(C_i x + d_i)$$

1. Taming the waves: sine as activation function in deep neural networks (https://openreview.net/forum?id=Sks3zF9eg)
2. Fourier Neural Networks: A Comparative Study (https://arxiv.org/abs/1902.03011)

Sinusoidal Activation Functions: Pros and Cons

- Pros
  - Universal extrapolation properties
  - Can learn some *a priori* known periodic signals better than alternatives
- Cons
  - Difficult to train (degeneracy in local minima)
  - Computationally Expensive
  - Cannot compete against ReLU on standard tasks

→ In these formulations, sinusoidal activations seem to be not competitive against alternatives!

Snake

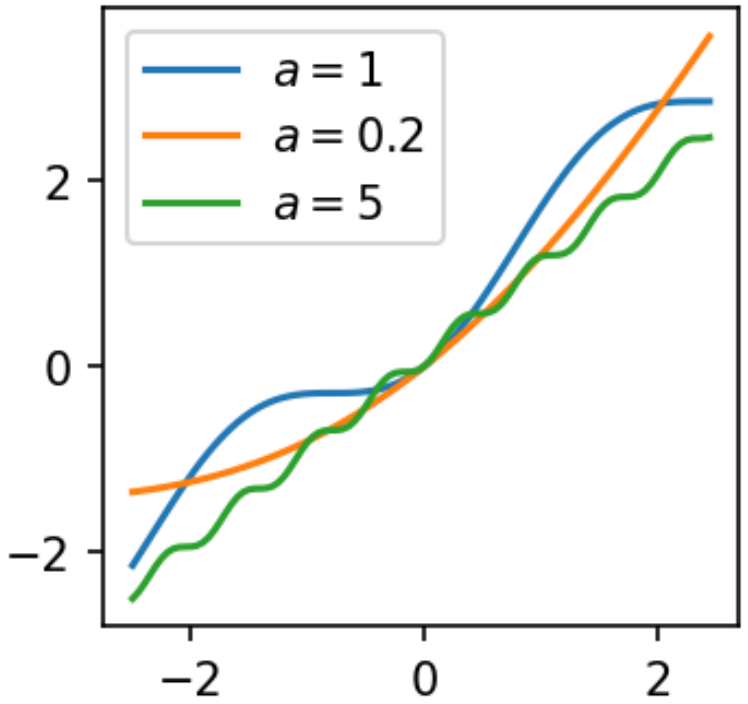
- Formulation of *Snake*[1] :

$$Snake_a := x + \frac{1}{a}\sin^2(ax)$$

where *a* can be set as learnable

Improvements over previous periodic functions:

- Monotonic (easier to train)
- Easy drop-in replacement
- Universal approximation properties
- Competitive with ReLU and other activation



Snake with different a: this parameter regulates the frequency

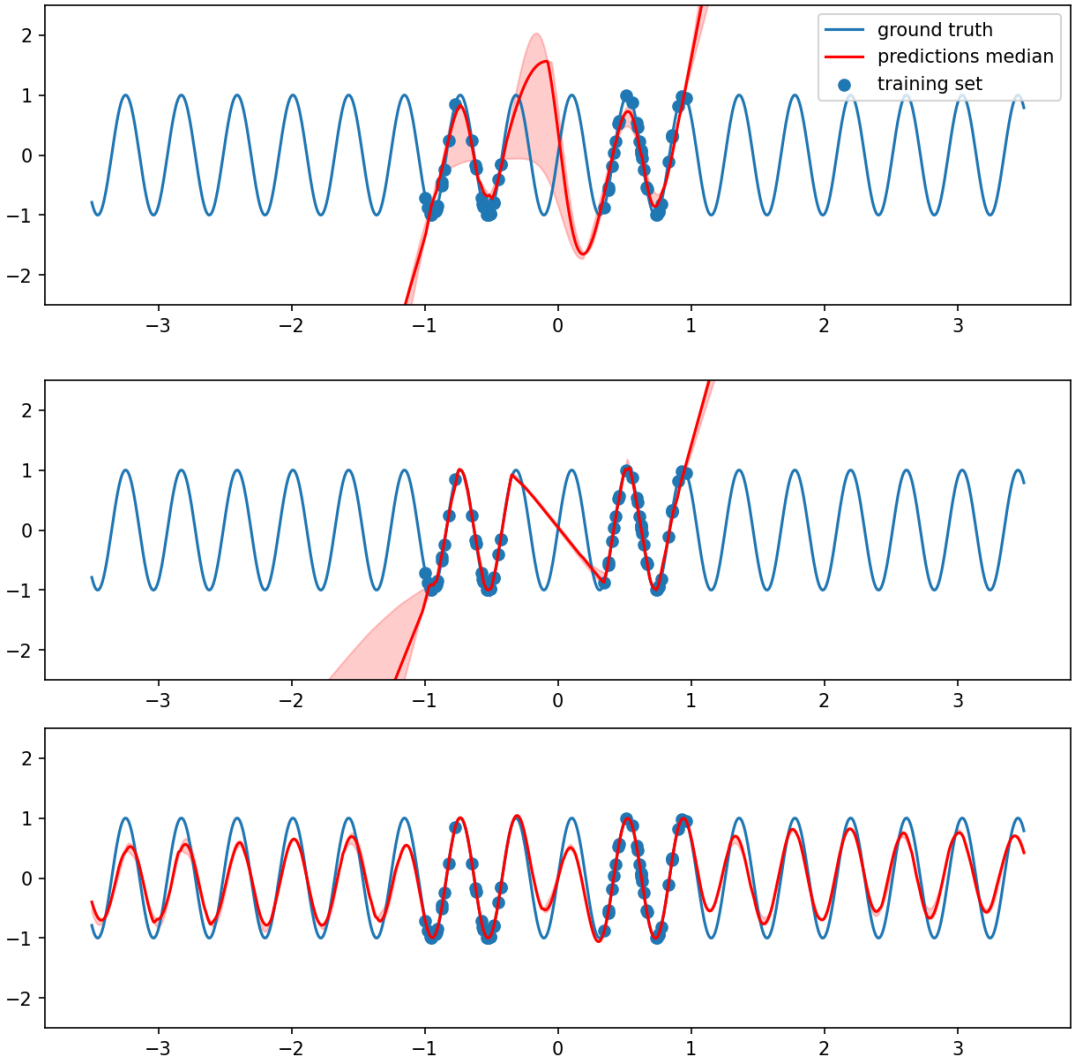
1. Implementation at https://github.com/EdwardDixon/snake

Snake Extrapolation

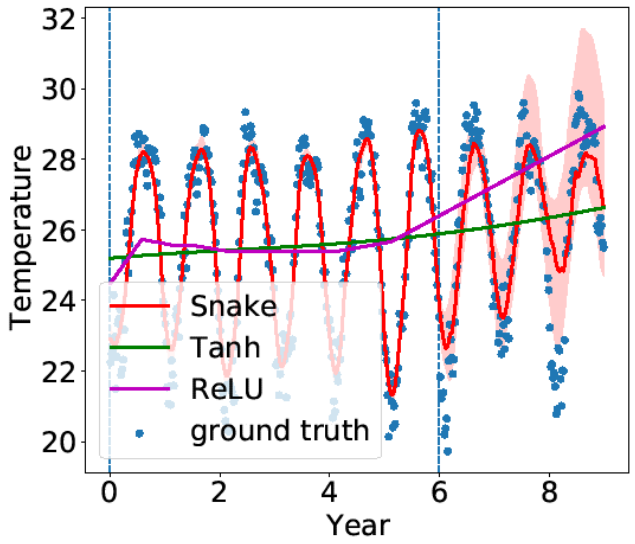
$$\text{Tanh(x)} = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$$

$$\text{ReLU(x)} = \begin{cases} x, if\ x > 0 \\ 0, otherwise \end{cases}$$

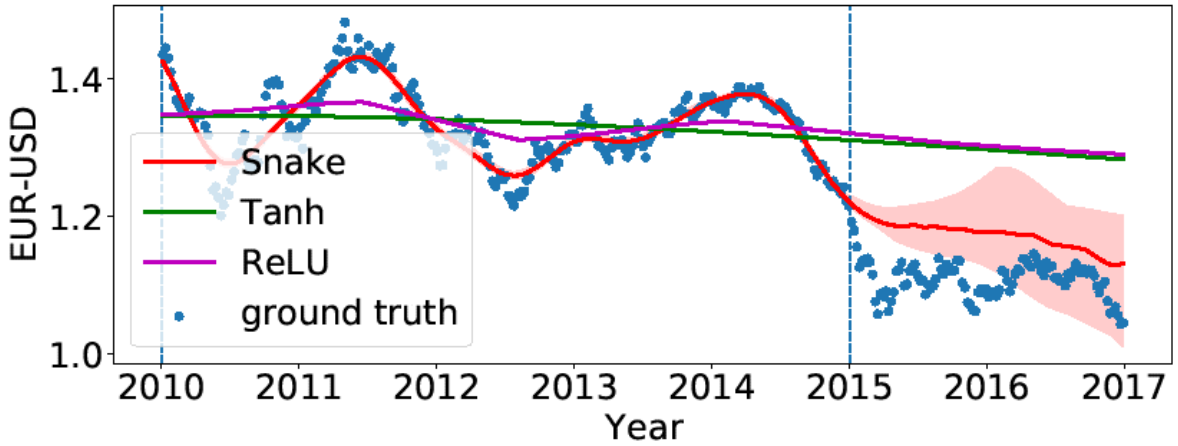
$$\text{Snake}\ (x) = x + \frac{1}{a}\sin^2(ax)$$



Competitiveness against Other Activation Functions – Periodic Data



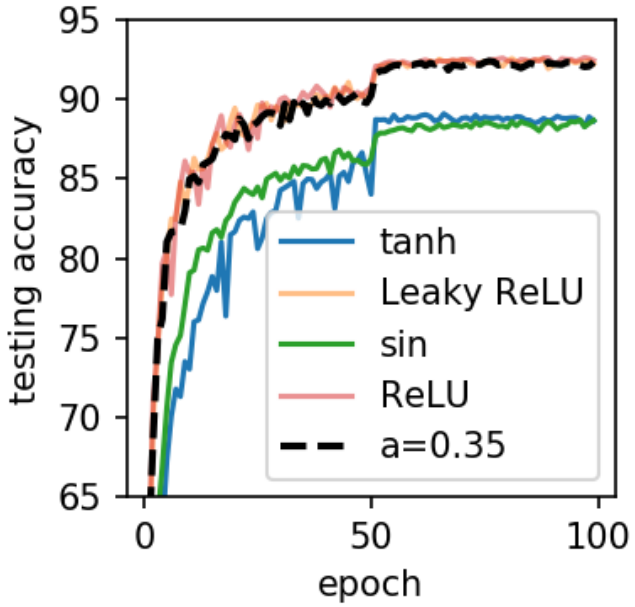
Atmospheric temperature prediction



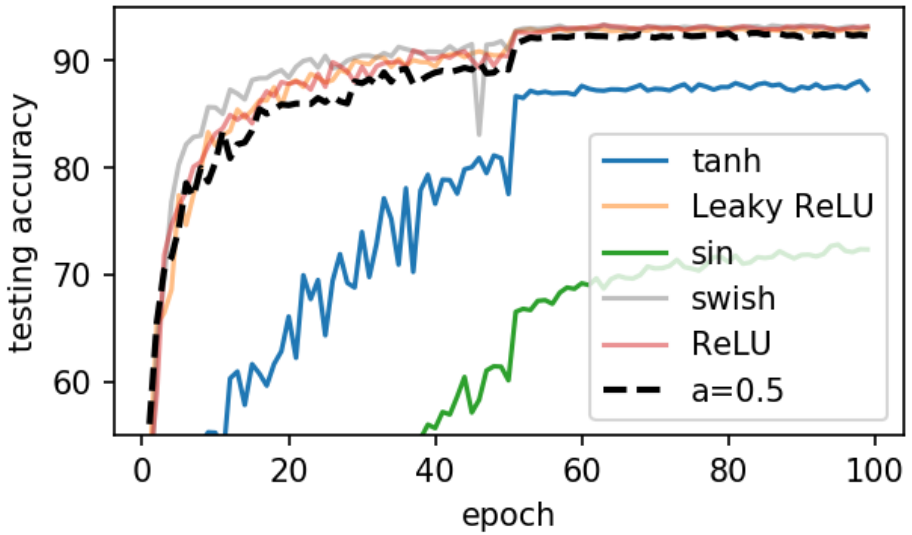
EUR-USD predictions

→ Snake easily outperforms other activation functions on extrapolation tasks of functions with a periodic component

Competitiveness against Other Activation Functions – Non-Periodic Data



ResNet18 on CIFAR-10



ResNet100 on CIFAR-10

→ Snake remains competitive against the other best-performing functions, while even retaining a similar learning curve due to its monotonicity

Background: Implicit Neural Representations

- We want to learn a function Φ such that:

$$F(x, \Phi, \nabla_x \Phi, \nabla_x^2 \Phi, \dots) = 0, \qquad \Phi: x \rightarrow \Phi(x)$$

with $x \in \mathbb{R}^m$ are spatio-temporal coordinates

- Φ is a neural network implicitly defined by the constraints imposed by F → *implicit neural representation*
- *Examples:*
  - *Images, audio, video*
  - *Solving boundary value problems, e.g. partial differential equations*

From: Scene Representation Networks: Continuous 3D-Structure-Aware Neural Scene Representations (https://arxiv.org/abs/1906.01618)

Background: Implicit Neural Representations (continued)

- Problem cast as:

$$\text{find } \Phi(x) \;\; subject\ to \;\; \mathcal{C}_m(a(x), \Phi(x), \nabla\Phi(x), \dots) = 0, \qquad \forall x \in \Omega_m, m = 1, \dots, M$$

where $\mathcal{C}_m(\cdot)$ are the sets of constraints (*boundary value problem)*

- The loss can be written as:

$$\mathcal{L} = \int_{\Omega} \sum_{m=1}^{M} \mathbf{1}_{\Omega_m}(x) \, \|\mathcal{C}_m(a(x), \Phi(x), \nabla\Phi(x), \dots)\| dx$$

where $\mathbf{1}_{\Omega_m}(x) = 1$ when $x \in \Omega_m$, and $\mathbf{1}_{\Omega_m}(x) = 0$ when $x \notin \Omega_m$

→We are enforcing the sampling inside of the domain

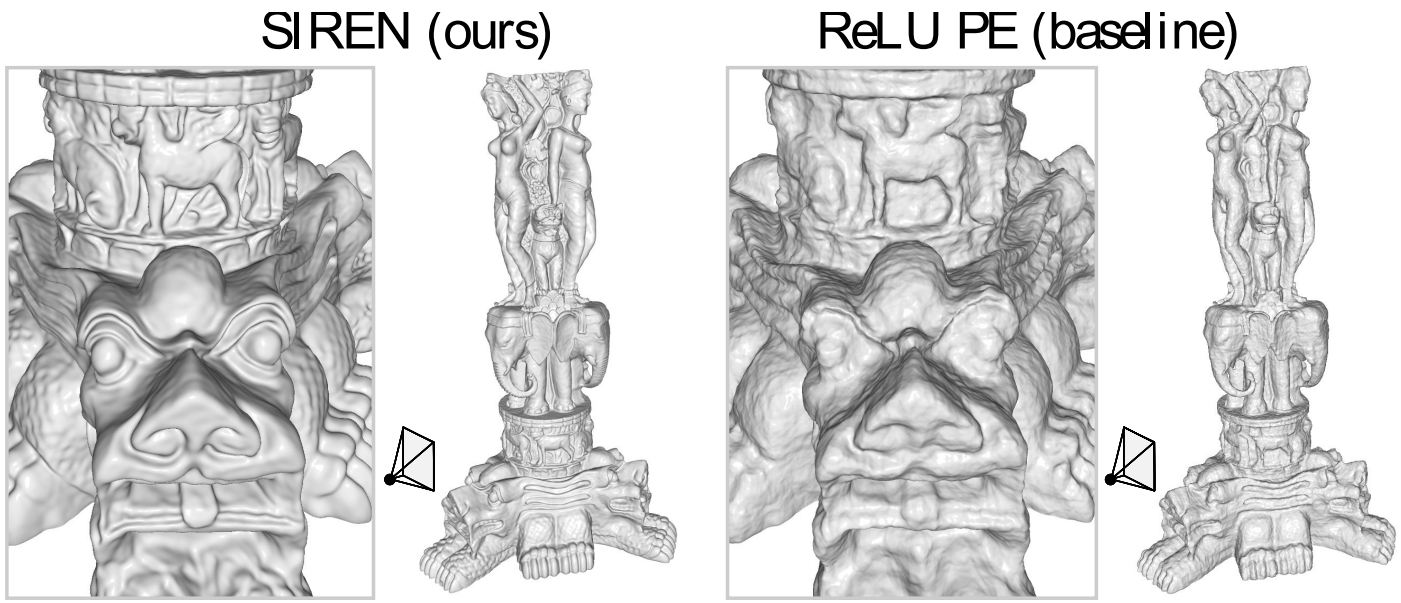
→The loss penalizes deviations from the constraints

SIREN

- SIREN: Sinusoidal Neural Representation[1]

$$\Phi(x) = W_n(\sigma_{n-1} \circ \sigma_{n-2} \circ \cdots \circ \sigma_0)(x) + b_n \quad \text{where } \sigma_i(x) = \sin(W_i x + b_i)$$

→ Affine transformation composed with nested sine activation functions



1. Extra material: videos, code playground and more (https://vsitzmann.github.io/siren/)

SIREN Properties

- SIREN: Sinusoidal Neural Representation

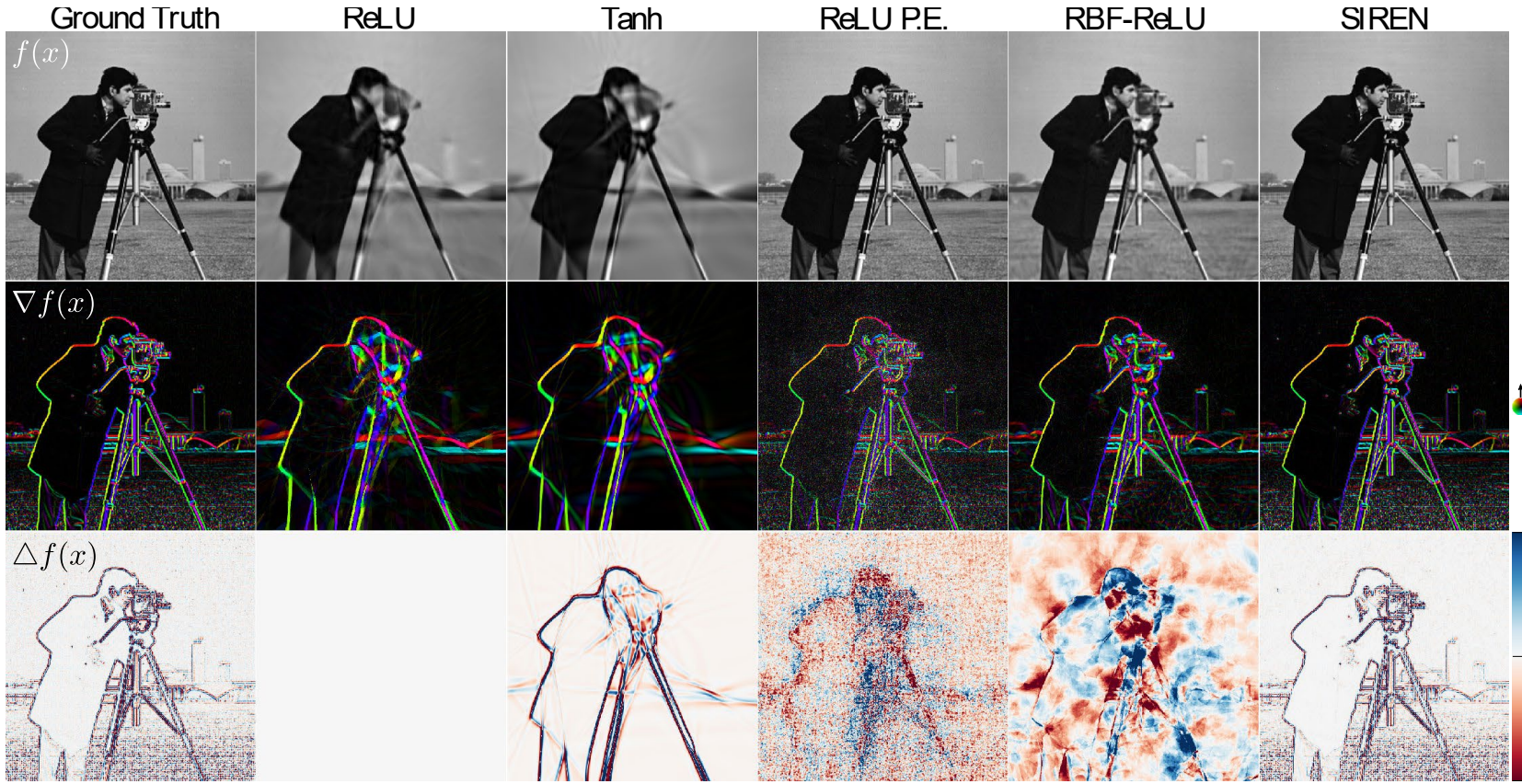
$$\Phi(x) = W_n(\sigma_{n-1} \circ \sigma_{n-2} \circ \cdots \circ \sigma_0)(x) + b_n \quad \text{where } \sigma_i(x) = \sin(W_i x + b_i)$$

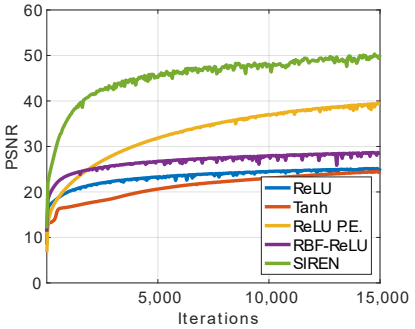
Properties:

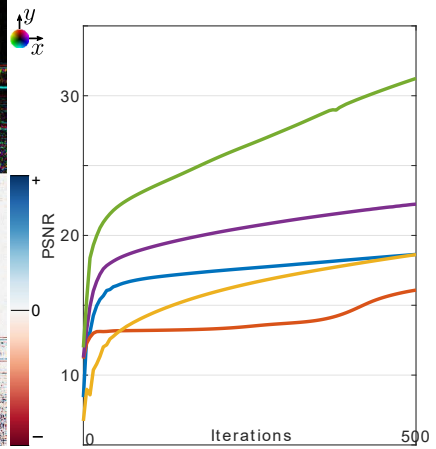
- Derivatives of a SIREN is a phase-shifted SIREN
- Continuously differentiable function ($\mathcal{C}^\infty$)
- Can capture signals and fine details thanks to *sine*
- Suitable for *implicit neural representations*

Image Learning

- Image: Φ: $\mathbb{R}^2 \rightarrow \mathbb{R}^3$
- Constraints: $\mathcal{C}\big(f(x), \Phi(x)\big) = \Phi(x_i) - f(x_i) = 0$
- Loss: $\mathcal{L} = \sum_i \|\Phi(x_i) - f(x_i)\|^2$

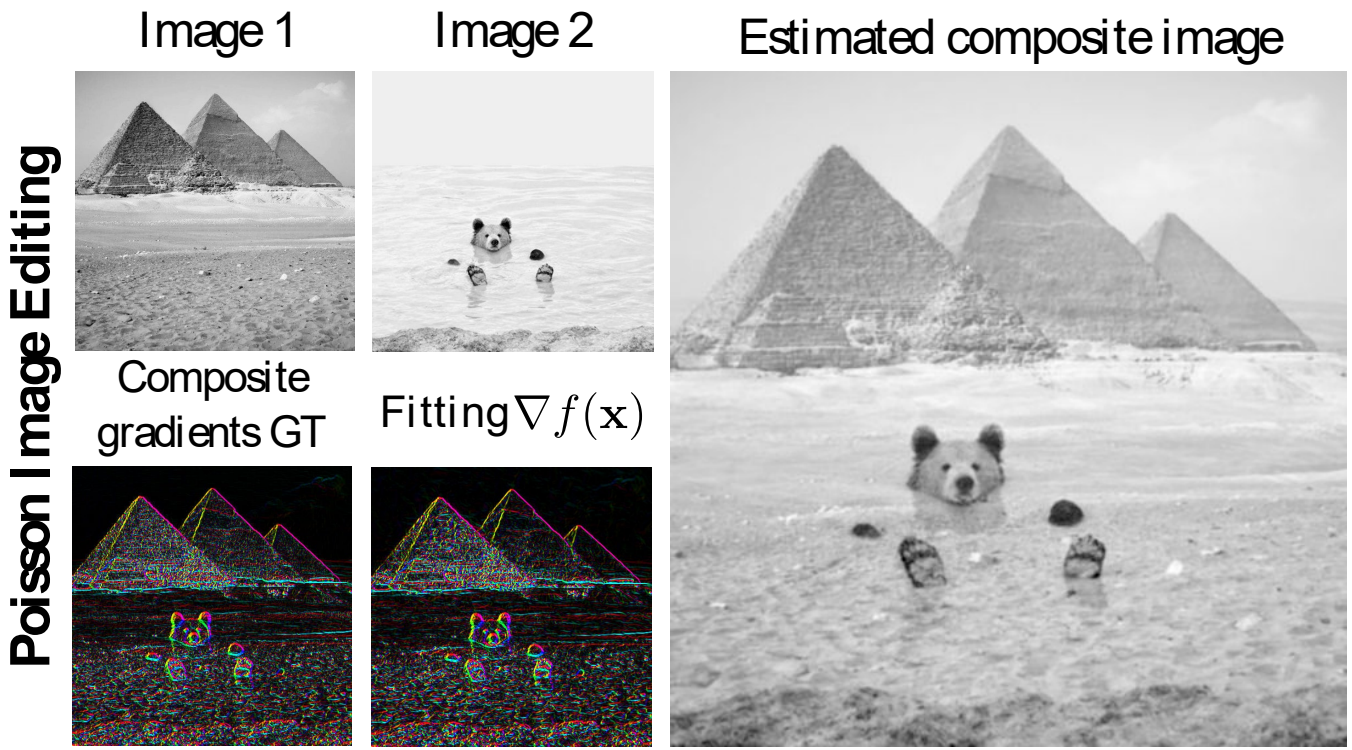






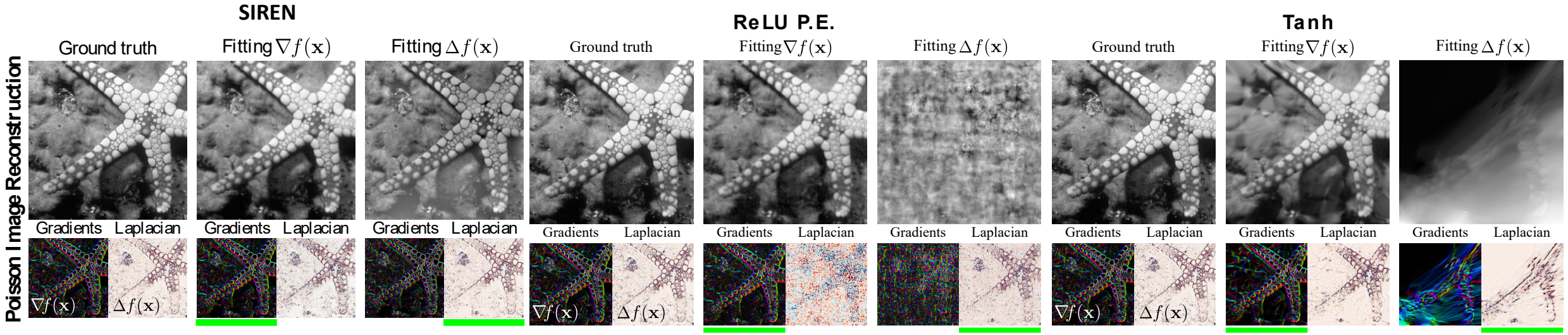
Supervising on Gradients and Laplacians

- Image: Φ: $\mathbb{R}^2 \rightarrow \mathbb{R}^3$
- Constraints: $\mathcal{C}\big(f(x), \Phi(x)\big) = \nabla_x \Phi(x) - \nabla_x f(x) = 0 \;\; or \; \Delta\Phi(x) - \Delta \mathrm{f(x)} = 0$
- Loss: $\mathcal{L} = \int_\Omega \|\nabla_x \Phi(x) - \nabla_x f(x)\| dx \quad or \quad \mathcal{L} = \int_\Omega \|\Delta\Phi(x) - \Delta \mathrm{f(x)}\| dx$



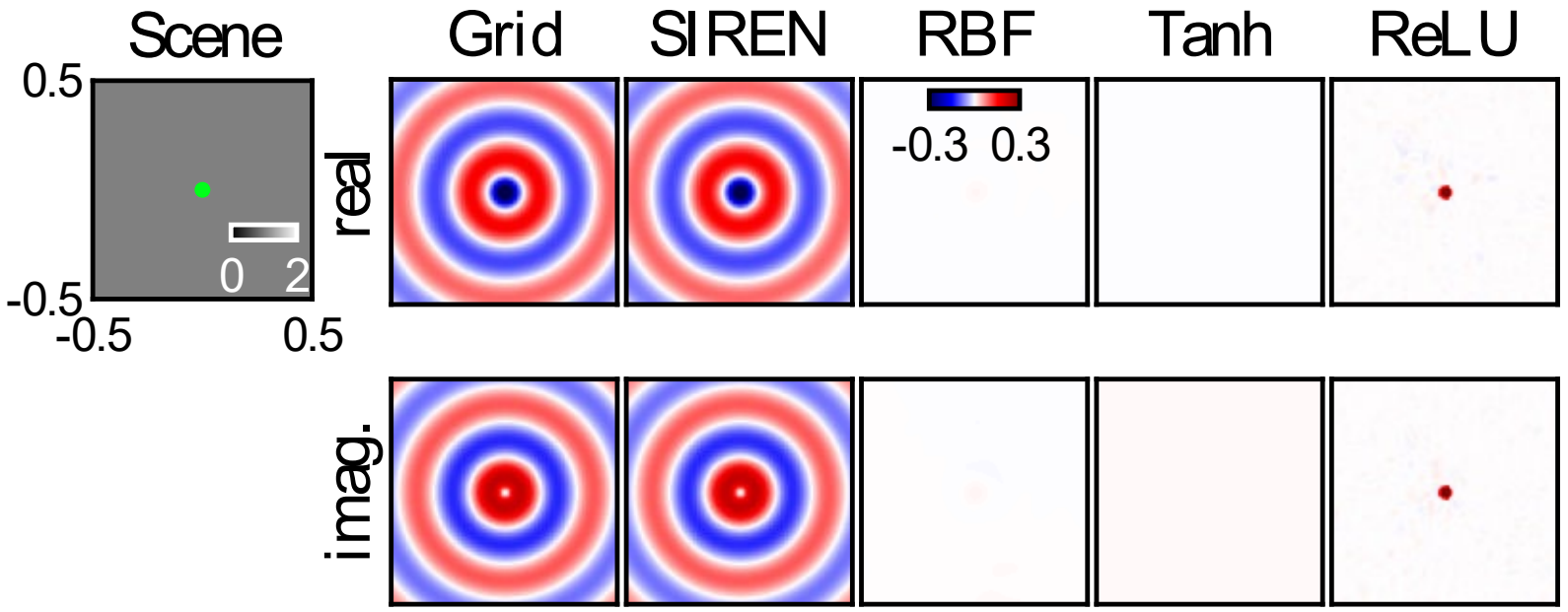
Supervising on Gradients and Laplacians (continued)

- Comparing SIREN with other activation functions
  - ReLU P.E. (with positional encoder) cannot fit Laplacians
  - Same applies to Tanh
  - SIREN handles well all situations



Solving Differential Equations

- Helmoltz Wave Equation: $\underbrace{(\Delta + m(x)\omega^2)}_{H(m)} \Phi(x) = -f(x)$
- Constraints: $\mathcal{C}(f(x), \Phi(x), H(m)) = H(m)\Phi(x) + f(x) = 0$
- Loss: $\mathcal{L} = \int_\Omega \lambda(x) \| H(m)\Phi(x) + f(x) \|_1 dx \;\; where \;\; \lambda(x) = k \in \mathbb{R}$



Experiments: Hypersolvers[1] for Controlled Dynamical Systems

- Controlled dynamical system

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

→This is a *boundary value problem*!

- Discretization

$$\begin{cases} x_{k+1} = x_k + \epsilon\psi_\epsilon(t_k, x_k, u_k) \\ x_{k=0} = x_0 \end{cases}$$

for $k \in 0, \ldots, K-1$, ψ_ϵ is an ODE solver (e.g., Euler, Runge Kutta…)

1. Hypersolvers: Toward Fast Continuous-Depth Models (https://arxiv.org/abs/2007.09601)

Experiments: Hypersolvers[1] for Controlled Dynamical Systems (continued)

- *Hypersolver*: network g_w approximating residuals

$$\begin{cases} x_{k+1} = x_k + \epsilon f\,(x_k, u_k) + \epsilon^2 g_w(x_k, f\,(x_k, u_k), u_k) \\ x_{k=0} = x_0 \end{cases}$$

Residuals are the difference between the real and approximated solution:

$$R = \Phi(x(t_k), t_k, t_{k+1}) - x(t_k) - \epsilon f(x_k, u_k)$$

Where Φ is the *nominal* solution of the discretization i.e., the *real* x_{k+1}

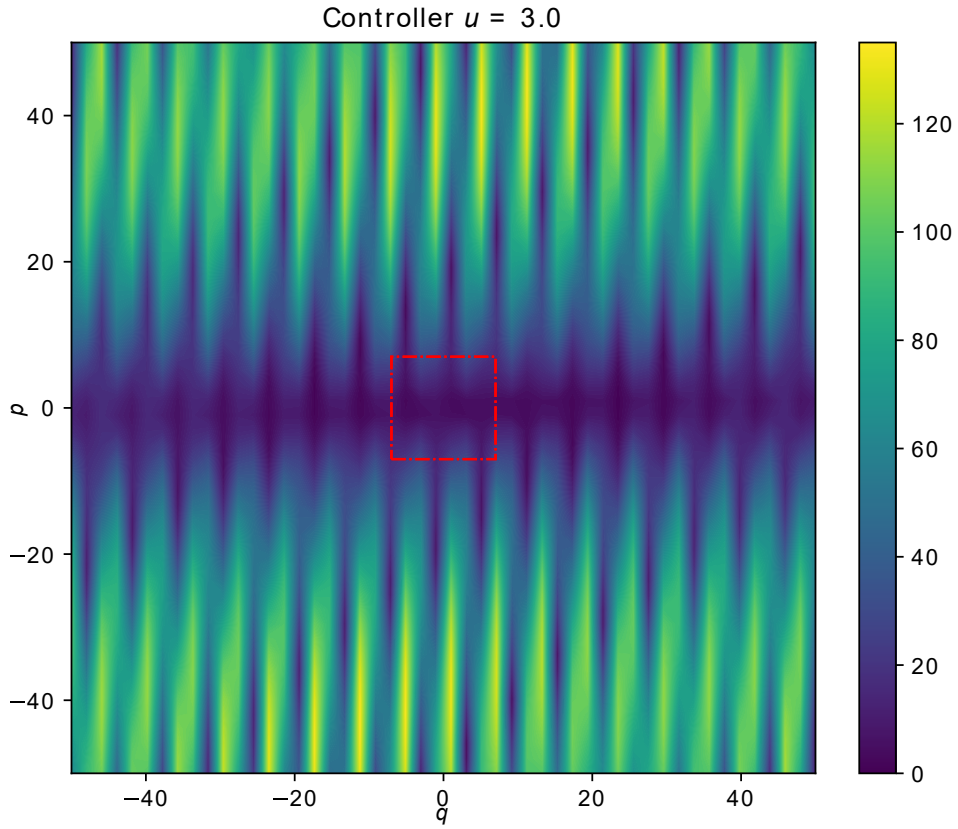
→ Idea: we can train g_w on the normalized residuals and learn the actual update next state with a *cheap* neural network instead of an *expensive* solver!

1. Hypersolvers: Toward Fast Continuous-Depth Models (https://arxiv.org/abs/2007.09601)

Residuals on the Inverted Pendulum

- Inverted pendulum model*:*

$$\begin{bmatrix}\dot{p}\\ \dot{q}\end{bmatrix} = \begin{bmatrix}0 & \frac{1}{m}\\ -k & -\frac{\beta}{m}\end{bmatrix}\begin{bmatrix}p\\ q\end{bmatrix} - \begin{bmatrix}0\\ mgl\sin(q)\end{bmatrix} + \begin{bmatrix}0\\ 1\end{bmatrix}u$$



Residual landscape at training start ($g_w(\cdot) = 0$)

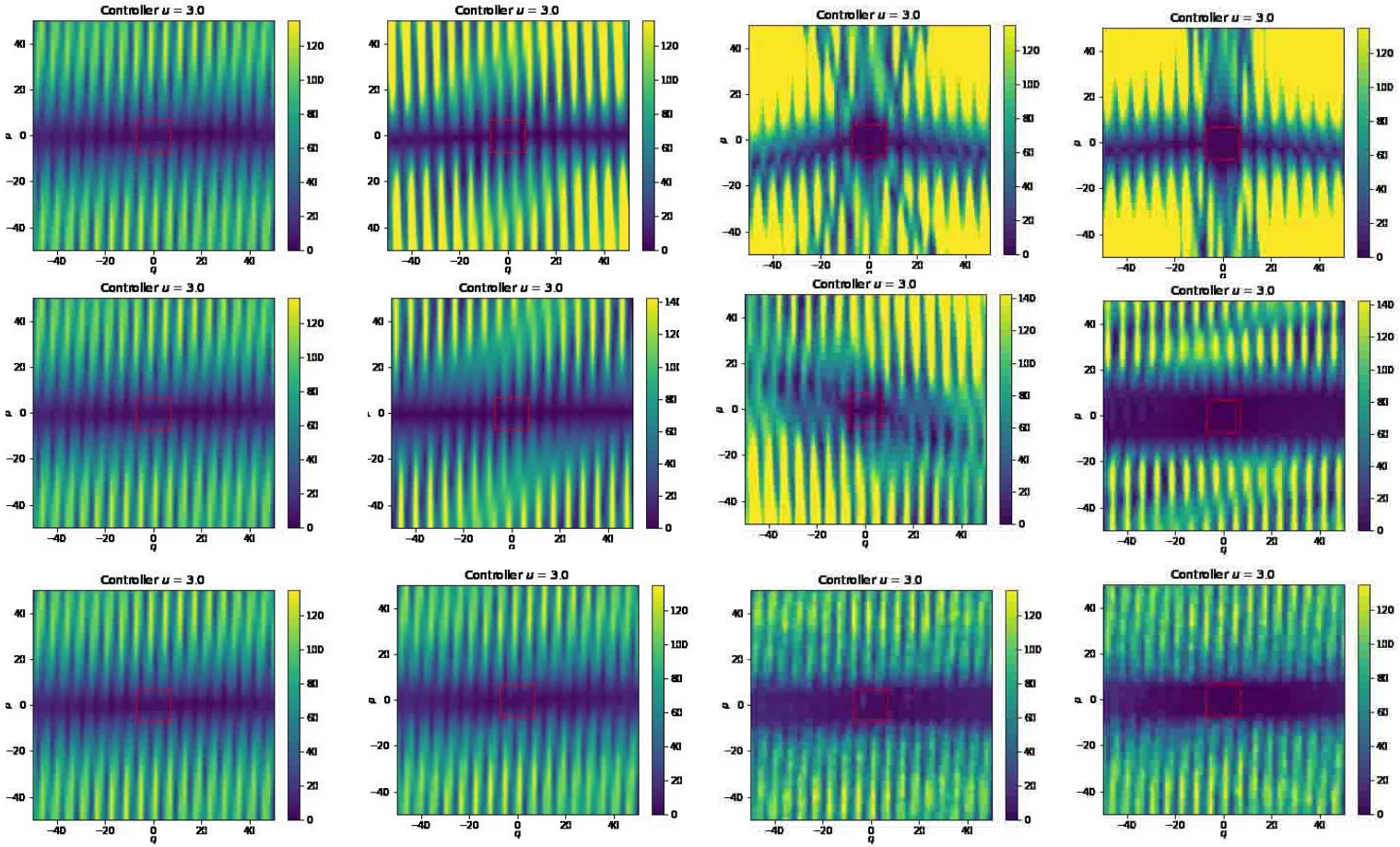
1. Hypersolvers: Toward Fast Continuous-Depth Models (https://arxiv.org/abs/2007.09601)

Residual Shape Comparison with Activation Functions

Softplus

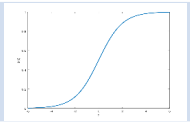
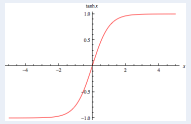
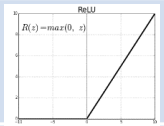
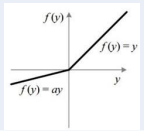
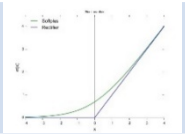
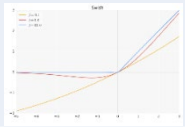
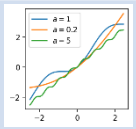
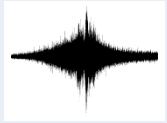
Snake

SIREN



Training Epochs

Revised Overview of Activation Functions

Function	Formula	Graph	Range	Saturation	Computation	$\mathcal{C}^1$	Extrapolation
Sigmoid	$\frac{1}{1+e^{-x}}$		$[0, 1]$	Negative and positive values	Intensive	✓	✗
Tanh	$\frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$		$[-1, 1]$	Negative and positive values	Intensive	✓	✗
ReLU	$\begin{cases} x, if\ x > 0 \\ 0, otherwise \end{cases}$		$[0, +\infty]$	Negative values (*dying ReLU*)	Easy	✗	✗
Leaky ReLU	$\begin{cases} x, if\ x > 0 \\ ax, otherwise \end{cases}$		$[-\infty, +\infty]$	None	Easy	✗	✗
Softplus	$\ln(1+e^{x})$		$[-\infty, +\infty]$	Large negative values	Intensive	✓	✗
Swish	$\frac{x}{1+e^{-\beta x}}$		$[-\infty, +\infty]$	Depending on β	Intensive	✓	✗
Snake	$x+\frac{1}{a}\sin^2(ax)$		$[-\infty, +\infty]$	None	Intensive†	✓	✓
SIREN	$\sin(W_i x + b_i)$ $\forall i \in [0, \mathrm{n}-1]$		$[-\infty, +\infty]$	None	Intensive†	✓	✓

†Performance of *sine* is similar to Tanh as described in the paper, depends on software and hardware implementation

Thanks for your attention!