Applications of neural networks in solving differential equations

Artem Chashchin

13.11.19

Outline

1. Predicting dynamical system evolution with residual neural networks
2. Learning finite element representation of PDE solutions with ReLU networks

Outline

1. Predicting dynamical system evolution with residual neural networks
2. Learning finite element representation of PDE solutions with ReLU networks

Motivation and Problem Statement

- Forecasting time series and time-dependent data is a common problem in many applications
- Example: solving ODE $\dot{x} = F(x)$ by using solution samples
  - Solvers cannot be used if $F(x)$ is not known explicitly
  - Neural networks could predict the solution for a timestep Δt

Objective

Exploring the ability of residual networks to predict the evolution of ODE systems from their samples

Current approaches

M. Mai et al. ***Reconstruction of ordinary differential equations from time series data*** **(2016)**:

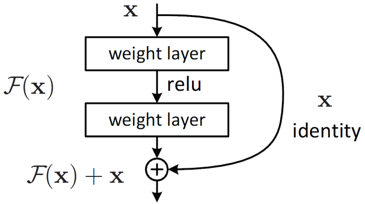
- Reconstruction of RHS from solution samples;
- Good results for 2D problems;
- Considerable divergence for 3D Lorenz system already after $T = 1$.

J. Pathak et al. ***Using machine learning to replicate chaotic attractors and calculate Lyapunov exponents from data*** **(2017)**:

- Based on reservoir modeling;
- Divergence after $T = 7$ for Lorenz system.

Proposed approach: residual network (ResNet)

ResNet blocks learn the residual of the input and the desired output



Source: He K. et al. *Deep residual learning for image recognition* (2016)

For a network of M ResNet blocks and input x_0, the output is

$$x_M = x_0 + \sum_{i=0}^{M-1} F_i(x_i)$$

Proposed approach: residual network (ResNet)

Deep Network

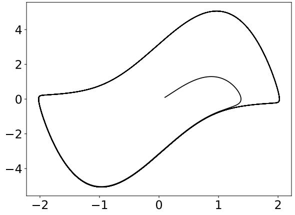
$$x_{i+1} = F_i(x_i)$$

- Consecutive multiplications in forward and backward propagations
- Vanishing/exploding gradients in very deep architectures

ResNet

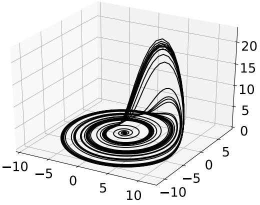
$$x_{i+1} = x_i + F_i(x_i)$$

- Summations replace some multiplications
- Vanishing/exploding gradients problem is reduced
- Resembles Euler method for ODE systems
- Successful application in image classification



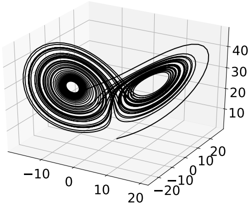
Van der Pol oscillator

$$\dot{x}(t) = y$$
$$\dot{y}(t) = \mu(1 - x^2)y - x$$



Rössler system

$$\dot{x} = -y - z$$
$$\dot{y} = x + ay$$
$$\dot{z} = b + z(x - c)$$



Lorenz system

$$\dot{x} = \sigma(y - x)$$
$$\dot{y} = x(\rho - z) - y$$
$$\dot{z} = xy - \beta z$$

Approach

- Initial conditions: 10000 points sampled from $N(0, I)$
- Solver finds solution of the ODE system in $[0, T]$
- **Two experiments** with ResNets:
  1. Training on samples from $[0, T/2]$ and predicting for $[T/2, T]$
  2. Training on samples from $[0, T/4]$ and predicting for $[T/4, T]$
- **Four ResNet architectures** with different parameters (number of blocks, number of layers in a block, layer size)

Results

Relative prediction errors for the short interval $[T/2, T]$

	Van der Pol		Lorenz		Rössler	
	ε_{avg}	ε_T	ε_{avg}	ε_T	ε_{avg}	ε_T
RN1	**0.61**	1.09	1.60e17	6.80e18	0.39	0.62
RN2	0.83	**1.07**	**0.38**	**0.57**	0.97	1.69
RN3	0.70	1.27	0.52	1.19	**0.31**	**0.53**
RN4	1.01	1.49	0.42	0.57	0.32	0.59

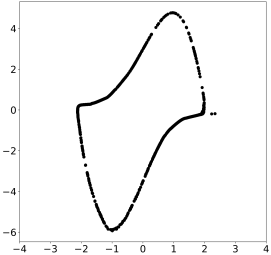
Satisfactory results for most of the experiments

Results

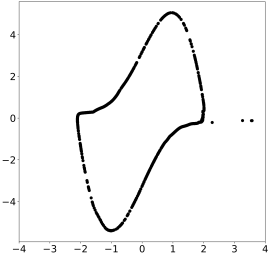
Relative prediction errors for the long interval $[T/4, T]$.

	Van der Pol		Lorenz		Rössler	
	ε_{avg}	ε_T	ε_{avg}	ε_T	ε_{avg}	ε_T
RN1	**0.42**	**0.73**	nan	nan	**0.42**	**0.65**
RN2	7.84	90.51	**0.34**	**0.58**	0.65	0.72
RN3	0.63	1.16	0.41	0.59	1.33	1.82
RN4	1.47	1.66	nan	nan	0.49	0.88

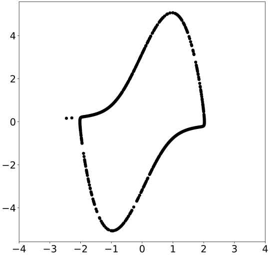
Predictions at the moment $T = 25$



ResNet, from T/2



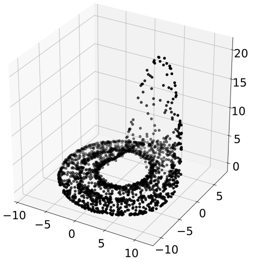
ResNet, from T/4



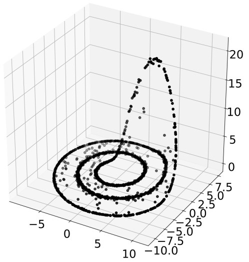
Solver

Results: Rössler attractor

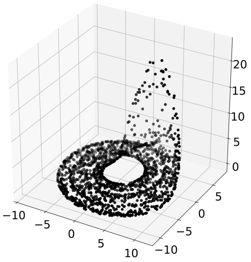
Predictions at the moment $T = 125$



ResNet, from T/2



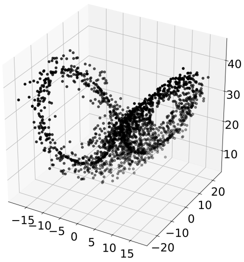
ResNet, from T/4



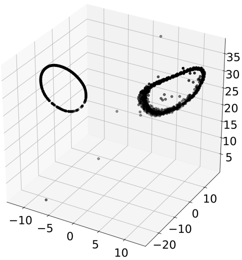
Solver

Results: Lorenz attractor

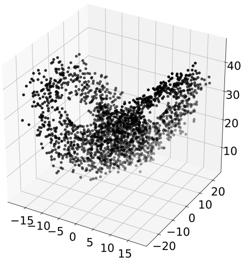
Predictions at the moment $T = 25$



ResNet, from T/2



ResNet, from T/4



Solver

Conclusion

Results:

- Successful application of residual networks to the problem of reconstructing ODE system solution
- Prediction interval is longer than for previous approaches
- The main dynamics of the systems is preserved

Future plans:

- Application of residual networks to more complex applied problems
- Experiments with ResNet modifications (e.g. RevNet - reversible residual network)

Outline

1. Predicting dynamical system evolution with residual neural networks
2. Learning finite element representation of PDE solutions with ReLU networks

Motivation and Problem Statement

- Previously, we learned the evolution of systems in time; now we try to learn the evoluion in space
- Neural networks could adaptively generate the grid and learn the solution

Objective

Constructing a neural network model for a finite element representation of PDE solution based on its samples

Related articles

He J. et al. ***ReLU deep neural networks and linear finite elements*** **(2018)**:

- Theoretical results on approximating continuous piecewise linear functions with deep ReLU networks
- Connection between FEM basis functions and ReLU functions

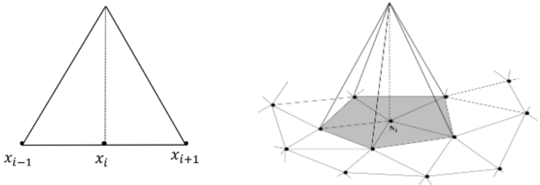
Yarotsky D. ***Error bounds for approximations with deep ReLU networks*** **(2017)**:

- Proposed NN architecture with skip connections to approximate squaring and multiplication operations

Fokina D., Oseledets I. ***Growing axons: greedy learning of neural networks with application to function approximation*** **(2019)**:

- Proposed an algorithm for efficient training of such networks

FEM approximation as a neural network



Source: He J. et al. *ReLU deep neural networks and linear finite elements* (2018)

$$u(x) \approx \sum_{i=1}^{N} \nu_i \phi_i(x)$$
$$\phi_i(x) = \frac{1}{h_i}\mathrm{ReLU}(x - x_{i-1}) - (\frac{1}{h_{i-1}} + \frac{1}{h_i})\mathrm{ReLU}(x - x_i) + \frac{1}{h_i}\mathrm{ReLU}(x - x_{i+1}),$$
where $h_i = x_{i+1} - x_i$

FEM approximation = linear combination of $\phi_i(x)$ =
linear combination of ReLUs = two-layer neural network

Architecture

Bias $\to$ ReLU $\to$ FC

- Bias $=$ a layer of biases $t_1, t_2, \ldots, t_N$
  - fixed uniform
  - trainable (uniform / Chebyshev / random init)
- FC $=$ a fully-connected layer without bias

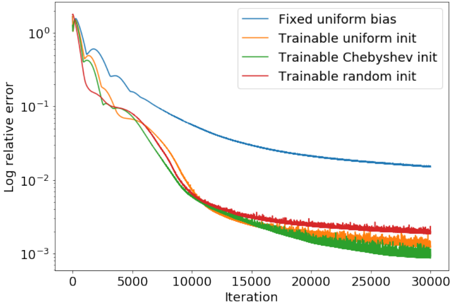
Output

$$\hat{u}(x; t, w) = \sum_{i=1}^{N} w_i \mathrm{ReLU}(x - t_i)$$

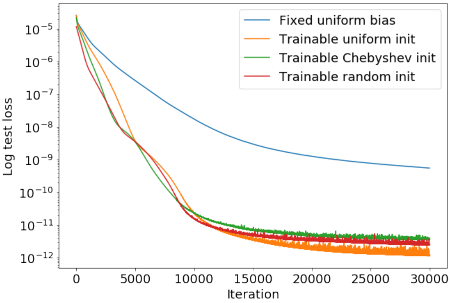
With appropriate constraints on weights and biases we could learn the FEM approximation of the function!

Experiments

$$u(x) = x(1 - x), \;\; x \in [0, 1]$$



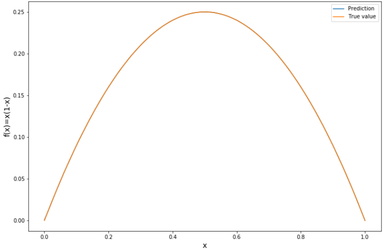
Log relative error



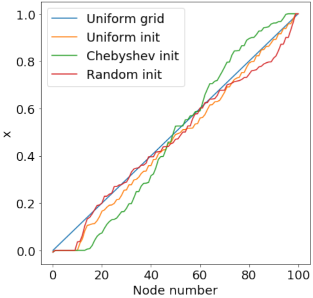
Log MSE on test set

Experiments

$$u(x) = x(1-x), \;\; x \in [0,1]$$

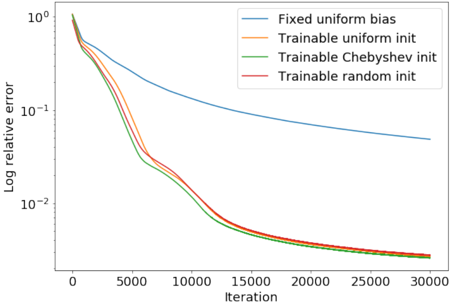


Prediction after 30000 epochs

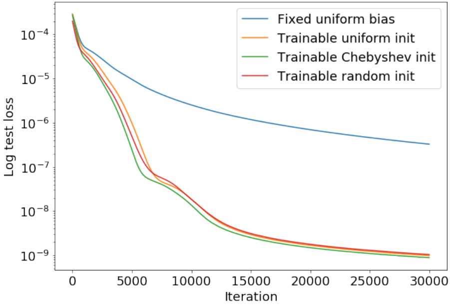


Learned grids

$$u(x) = \sqrt{x}, \; x \in [0, 1]$$



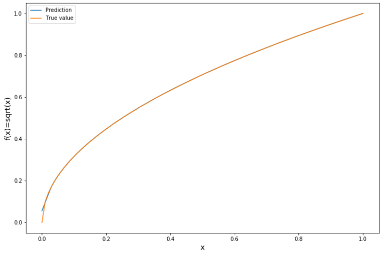
Log relative error



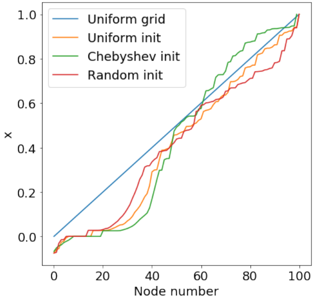
Log MSE on test set

Experiments

$u(x) = \sqrt{x}, \ x \in [0, 1]$



Prediction after 30000 epochs



Learned grids

Plans

- Adding constraints to keep the grid from getting out of bounds
- Adding variational dropout to learn sparse approximation of a function
- Considering problems of higher dimension