Physics-informed Neural Networks
Mixed Data-driven Model-based approach

Workshop Automatique et IA 10 Juin 2021

Matthieu Barreau
Division of Decision and Control Systems, KTH, barreau@kth.se

work conducted with J. Liu, M. Aguiar and K. H. Johansson

Introdution

Machine Learning

$$u^\star = \operatorname{argmin}_{\mathbf{u}} \quad \frac{1}{N}\sum_{i=1}^{N} L\left(\nu_i, y_i, \mathbf{u}\right)$$

Data-driven

Optimization problem

Introdution

Machine Learning

$$u^\star = \operatorname{argmin}_{\mathbf{u}} \quad \frac{1}{N}\sum_{i=1}^{N} L\left(\nu_i, y_i, \mathbf{u}\right)$$

Data-driven

Optimization problem

Differential Equation

$$\begin{cases} \dfrac{\partial v}{\partial t}(t,x) + \mathcal{N}[v](t,x) = 0, & (t,x) \in [0,T) \times \Omega, \\ v(\cdot,x)|_{x\in\partial\Omega} = v_{\partial\Omega}, \\ v(0,\cdot) = v_0. \end{cases}$$

Initial condition

Boundary condition

PDE $\Rightarrow$ Model-based

Introdution

Machine Learning

$$u^\star = \operatorname{argmin}_{\mathbf{u}} \frac{1}{N}\sum_{i=1}^{N} L\left(\nu_i, y_i, \mathbf{u}\right)$$

Data-driven

Optimization problem

Differential Equation

$$\begin{cases} \dfrac{\partial v}{\partial t}(t,x) + \mathcal{N}[v](t,x) = 0, & (t,x) \in [0,T) \times \Omega, \\ v(\cdot,x)|_{x\in\partial\Omega} = v_{\partial\Omega}, \\ v(0,\cdot) = v_0. \end{cases}$$

Initial condition

Boundary condition

PDE $\Rightarrow$ Model-based

Examples

Weather forecast

Stock price prediction

...

Traffic jam forecast

Trajectory estimation

Machine learning to solve PDEs?

JOURNAL OF COMPUTATIONAL PHYSICS **91**, 110–131 (1990)

Neural Algorithm for Solving Differential Equations

HYUK LEE

Department of Electrical Engineering, Polytechnic Institute of New York, Brooklyn, New York 11201

AND

IN SEOK KANG

Department of Chemical Engineering, California Institute of Technology, Pasadena, California 91125

Received August 17, 1988; revised October 6, 1989

Machine learning to solve PDEs?

IEEE TRANSACTIONS ON NEURAL NETWORKS, VOL. 9, NO. 5, SEPTEMBER 1998 987

Artificial Neural Networks for Solving Ordinary and Partial Differential Equations

Isaac Elias Lagaris, Aristidis Likas, *Member, IEEE,* and Dimitrios I. Fotiadis

abstract—We present a method to solve initial and boundary value problems using artificial neural networks. A trial solution of the differential equation is written as a sum of two parts. The first part satisfies the initial/boundary conditions and contains no adjustable parameters. The second part is constructed so as not to affect the initial/boundary conditions. This part involves a feedforward neural network containing adjustable parameters (the weights). Hence by construction the initial/boundary conditions are satisfied and the network is trained to satisfy the differential equation. The applicability of this approach ranges from single ordinary differential equations (ODE's), to systems of coupled ODE's and also to partial differential equations (PDE's). In this article, we illustrate the method by solving a variety of model problems and present comparisons with solutions obtained using the Galerkin finite element method for several cases of

of a feedforward neural network by replacing each spline with the sum of piecewise linear activation functions that correspond to the hidden units. This method considers local basis-functions and in general requires many splines (and consequently network parameters) in order to yield accurate solutions. Furthermore, it is not easy to extend these techniques to multidimensional domains.

In this article we view the problem from a different angle. We present a method for solving both ordinary differential equations (ODE's) and partial differential equations (PDE's) (defined on *orthogonal box* domains) that relies on the function approximation capabilities of feedforward neural networks and results in the construction of a solution written in a

JOURNAL OF COMPUTATIONAL PHYSICS **91**, 110–131 (1990)

Neural Algorithm for Solving Differential Equations

HYUK LEE

Department of Electrical Engineering, Polytechnic Institute of New York, Brooklyn, New York 11201

AND

IN SEOK KANG

Department of Chemical Engineering, California Institute of Technology, Pasadena, California 91125

Received August 17, 1988; revised October 6, 1989

Machine learning to solve PDEs?

Neural-Network Methods for Boundary Value Problems with Irregular Boundaries

Isaac Elias Lagaris, Aristidis C. Likas, *Member, IEEE*, and Dimitrios G. Papageorgiou

Abstract—**Partial differential equations (PDEs) with boundary conditions (Dirichlet or Neumann) defined on boundaries with simple geometry have been successfully treated using sigmoidal multilayer perceptrons in previous works. This article deals with the case of complex boundary geometry, where the boundary is determined by a number of points that belong to it and are closely located, so as to offer a reasonable representation. Two networks**

1) **infinitely differentiable closed analytic form;**
2) **superior interpolation properties as compared to well-established methods such as finite elements [1], [2];**
3) **small number of parameters;**
4) **suitability for efficient implementation on parallel computers;**

Artificial Neural Networks for Solving Ordinary and Partial Differential Equations

Isaac Elias Lagaris, Aristidis Likas, *Member, IEEE*, and Dimitrios I. Fotiadis

Abstract—**We present a method to solve initial and boundary value problems using artificial neural networks. A trial solution of the differential equation is written as a sum of two parts. The first part satisfies the initial/boundary conditions and contains no adjustable parameters. The second part is constructed so as not to affect the initial/boundary conditions. This part involves a feedforward neural network containing adjustable parameters (the weights). Hence by construction the initial/boundary conditions are satisfied and the network is trained to satisfy the differential equation. The applicability of this approach ranges from single ordinary differential equations (ODE's), to systems of coupled ODE's and also to partial differential equations (PDE's). In this article, we illustrate the method by solving a variety of model problems and present comparisons with solutions obtained using the Galerkin finite element method for several cases of**

of a feedforward neural network by replacing each spline with the sum of gaussian or linear activation functions that correspond to the hidden units. This method considers local basis-functions and in general requires many splines (and consequently network parameters) in order to yield accurate solutions. Furthermore, it is not easy to extend these techniques to multidimensional domains.

In this article we view the problem from a different angle. We present a method for solving both ordinary differential equations (ODE's) and partial differential equations (PDE's) (defined on *orthogonal box* domains) that relies on the function approximation capabilities of feedforward neural networks and results in the construction of a solution written in a

RNAL OF COMPUTATIONAL PHYSICS **91**, 110–131 (1990)

Neural Algorithm for Solving Differential Equations

HYUK LEE

Department of Electrical Engineering, Polytechnic Institute of New York, Brooklyn, New York 11201

AND

IN SEOK KANG

Department of Chemical Engineering, California Institute of Technology, Pasadena, California 91125

Received August 17, 1988; revised October 6, 1989

Machine learning to solve PDEs?

IEEE TRANSACTIONS ON NEURAL NETWORKS, VOL. 11, NO. 5, SEPTEMBER 2000 1041

Neural-Network Methods for Boundary Value Problems with Irregular Boundaries

Isaac Elias Lagaris, Aristidis C. Likas, *Member, IEEE*, and Dimitrios G. Papageorgiou

Abstract—**Partial differential equations (PDEs) with boundary conditions (Dirichlet or Neumann) defined on boundaries with simple geometry have been successfully treated using sigmoidal multilayer perceptrons in previous works. This article deals with the case of complex boundary geometry, where the boundary is determined by a number of points that belong to it and are closely located, so as to offer a reasonable representation. Two networks**

1) infinitely differentiable closed analytic form;
2) superior interpolation properties as compared to well-established methods such as finite elements [1], [2];
3) small number of parameters;
4) suitability for efficient implementation on parallel computers;

IEEE TRANSACTIONS ON NEURAL NETWORKS, VOL. 9, NO. 5, SEPTEMBER 1998 987

Artificial Neural Networks for Solving Ordinary and Partial Differential Equations

Isaac Elias Lagaris, Aristidis Likas, *Member, IEEE*, and Dimitrios I. Fotiadis

Abstract—**We present a method to solve initial and boundary value problems using artificial neural networks. A trial solution of the differential equation is written as a sum of two parts. The first part satisfies the initial/boundary conditions and contains no adjustable parameters. The second part is constructed so as not to affect the initial/boundary conditions. This part involves a feedforward neural network containing adjustable parameters (the weights). Hence by construction the initial/boundary conditions are satisfied and the network is trained to satisfy the differential equation. The applicability of this approach ranges from single ordinary differential equations (ODE's), to systems of coupled ODE's and also to partial differential equations (PDE's). In this article, we illustrate the method by solving a variety of model problems and present comparisons with solutions obtained using the Galerkin finite element method for several cases of**

of a feedforward neural network by replacing each spline with the sum of piecewise linear activation functions that correspond to the hidden units. This method considers local basis-functions and in general requires many splines (and consequently network parameters) in order to yield accurate solutions. Furthermore, it is not easy to extend these techniques to multidimensional domains.

In this article we view the problem from a different angle. We present a method for solving both ordinary differential equations (ODE's) and partial differential equations (PDE's) (defined on *orthogonal box* domains) that relies on the function approximation capabilities of feedforward neural networks and results in the construction of a solution written in a

RNAL OF COMPUTATIONAL PHYSICS **91**, 110–131 (1990)

Neural Algorithm for Solving Differential Equations

HYUK LEE

Department of Electrical Engineering, Polytechnic Institute of New York, Brooklyn, New York 11201

Solving Partial Differential Equations Using

I

Department of Chemical En
Pasa

Received August

Artificial Neural Networks

by

Keith Rudd

Department of Mechanical Engineering and Material Sciences
Duke University

Machine learning to solve PDEs?

IEEE TRANSACTIONS ON NEURAL NETWORKS, VOL. 11, NO. 5, SEPTEMBER 2000 1041

Neural-Network Methods for Boundary Value Problems with Irregular Boundaries

Isaac Elias Lagaris, Aristidis C. Likas, *Member, IEEE*, and Dimitrios G. Papageorgiou

Abstract—**Partial differential equations (PDEs) with boundary conditions (Dirichlet or Neumann) defined on boundaries with simple geometry have been successfully treated using sigmoidal multilayer perceptrons in previous works. This article deals with the case of complex boundary geometry, where the boundary is determined by a number of points that belong to it and are closely located, so as to offer a reasonable representation. Two networks**

1) infinitely differentiable closed analytic form;
2) superior interpolation properties as compared to well-established methods such as finite elements [1], [2];
3) small number of parameters;
4) suitability for efficient implementation on parallel computers;

IEEE TRANSACTIONS ON NEURAL NETWORKS, VOL. 9, NO. 5, SEPTEMBER 1998 987

Artificial Neural Networks for Solving Ordinary and Partial Differential Equations

Isaac Elias Lagaris, Aristidis Likas, *Member, IEEE*, and Dimitrios I. Fotiadis

Abstract—**We present a method to solve initial and boundary value problems using artificial neural networks. A trial solution of the differential equation is written as a sum of two parts. The first part satisfies the initial/boundary conditions and contains no adjustable parameters. The second part is constructed so as not to affect the initial/boundary conditions. This part involves a feedforward neural network containing adjustable parameters (the weights). Hence by construction the initial/boundary conditions are satisfied and the network is trained to satisfy the differential equation. The applicability of this approach ranges from single ordinary differential equations (ODE's), to systems of coupled ODE's and also to partial differential equations (PDE's). In this article, we illustrate the method by solving a variety of model problems and present comparisons with solutions obtained using the Galerkin finite element method for several cases of** of a feedforward neural network by replacing each spline with the sum of piecewise linear activation functions that correspond to the hidden units. This method considers local basis-functions and in general requires many splines (and consequently network parameters) in order to yield accurate solutions. Furthermore, it is not easy to extend these techniques to multidimensional domains.

In this article we view the problem from a different angle. We present a method for solving both ordinary differential equations (ODE's) and partial differential equations (PDE's) (defined on *orthogonal box* domains) that relies on the function approximation capabilities of feedforward neural networks and results in the construction of a solution written in a

RNAL OF COMPUTATIONAL PHYSICS 91, 11

Neural Algorithm for

Department of Chemical En
Pasa

Received August

Journal of Computational Physics 375 (2018) 1339–1364

Contents lists available at ScienceDirect

Journal of Computational Physics

www.elsevier.com/locate/jcp

DGM: A deep learning algorithm for solving partial differential equations ☆,☆☆

Justin Sirignano [a,*], Konstantinos Spiliopoulos [b]

[a] University of Illinois at Urbana-Champaign, Urbana, United States of America
[b] Department of Mathematics and Statistics, Boston University, Boston, United States of America

Department of Electrical Engineering, Polytechnic Institute of New York, Brooklyn, New York 11201

Solving Partial Differential Equations Using

Artificial Neural Networks

by

Keith Rudd

Department of Mechanical Engineering and Material Sciences
Duke University

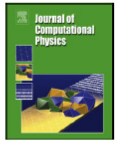
Machine learning to solve PDEs?

Journal of Computational Physics 378 (2019) 686–707

Contents lists available at ScienceDirect

Journal of Computational Physics

www.elsevier.com/locate/jcp



Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations

M. Raissi [a], P. Perdikaris [b,*], G.E. Karniadakis [a]

[a] Division of Applied Mathematics, Brown University, Providence, RI, 02912, USA

[b] Department of Mechanical Engineering and Applied Mechanics, University of Pennsylvania, Philadelphia, PA, 19104, USA

Preliminaries on neural networks

A neural network unit:

$$H_{W,b}(\nu) = \phi(\underbrace{W}_{\in \mathbb{R}^{N_n \times ?}} \times \nu + \underbrace{b}_{\in \mathbb{R}^{N_n \times 1}})$$

Preliminaries on neural networks

A neural network unit:

$$H_{W,b}(\nu) = \phi(\underbrace{W}_{\in \mathbb{R}^{N_n \times ?}} \times \nu + \underbrace{b}_{\in \mathbb{R}^{N_n \times 1}})$$

A multilayer neural network with linear output:

$$u_\theta(\nu) = W \times H_{\theta_{L-1}} \circ H_{\theta_{L-2}} \circ \cdots \circ H_{\theta_1}(\nu) + b$$

Preliminaries on neural networks

A neural network unit:

$$H_{W,b}(\nu) = \phi(\underbrace{W}_{\in \mathbb{R}^{N_n \times ?}} \times \nu + \underbrace{b}_{\in \mathbb{R}^{N_n \times 1}})$$

A multilayer neural network with linear output:

$$u_\theta(\nu) = W \times H_{\theta_{L-1}} \circ H_{\theta_{L-2}} \circ \cdots \circ H_{\theta_1}(\nu) + b$$

Theorem: Universal approximation (K. Hornik)

A neural network can approximate any smooth function arbitrarily close provided a sufficient number of neurons N_n.

Preliminaries on neural networks

A neural network unit:

$$H_{W,b}(\nu) = \phi(\underbrace{W}_{\in \mathbb{R}^{N_n \times ?}} \times \nu + \underbrace{b}_{\in \mathbb{R}^{N_n \times 1}})$$

A multilayer neural network with linear output:

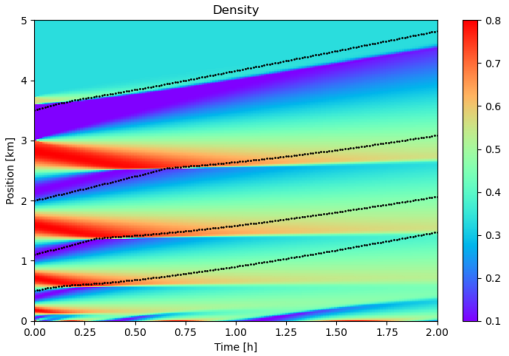
$$u_\theta(\nu) = W \times H_{\theta_{L-1}} \circ H_{\theta_{L-2}} \circ \cdots \circ H_{\theta_1}(\nu) + b$$

Proposition: Backpropagation (D. Rumelhart, G. Hinton & R. Williams)

Provided ϕ is differentiable, the backpropagation algorithm makes it possible to symbolically computes any derivative of u_θ and in particular, $\frac{\partial u}{\partial \nu}$.

Data-driven versus Model-based

Example: Traffic density estimation

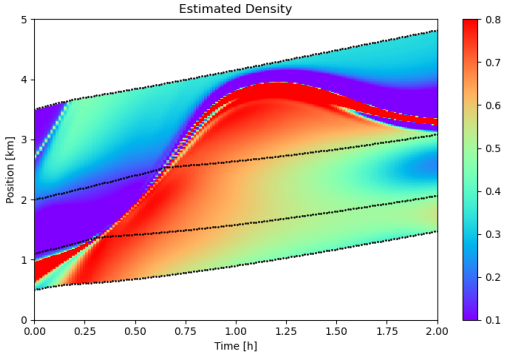


Real density - Simulation

- 2-D map (spatiotemporal diagram)
- colormap = density
- Black dots are measurements (sparse)

Data-driven versus Model-based

Example: Traffic density estimation

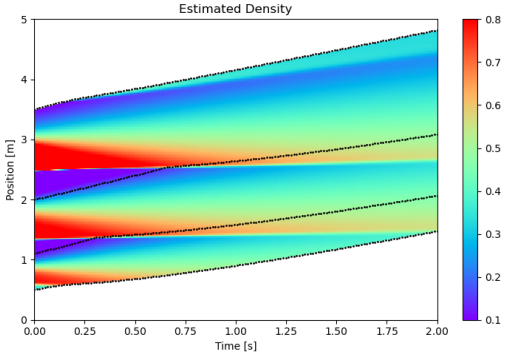


Estimated density - Data

- 2-D map (spatiotemporal diagram)
- colormap = density
- Black dots are measurements (sparse)

Data-driven versus Model-based

Example: Traffic density estimation



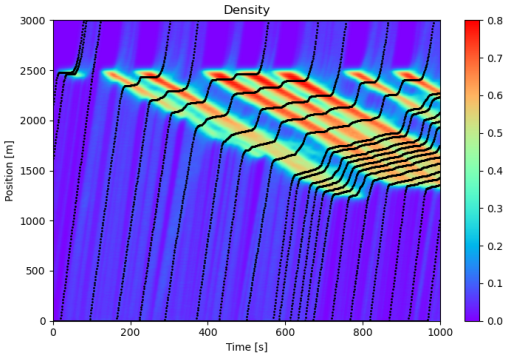
Estimated density - Model

- 2-D map (spatiotemporal diagram)
- colormap = density
- Black dots are measurements (sparse)
- Model for traffic

$$\frac{\partial u}{\partial t} + V_f(1-2u)\frac{\partial u}{\partial x} = 0$$

Data-driven versus Model-based

Example: Traffic density estimation

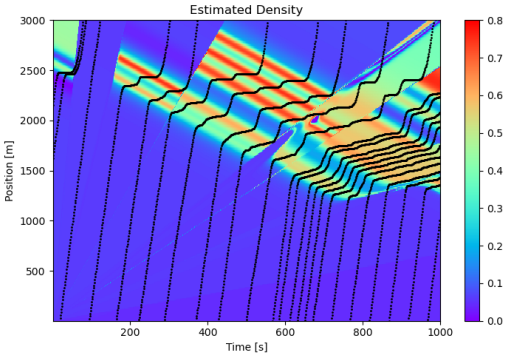


Real density - Road

- 2-D map (spatiotemporal diagram)
- colormap = density
- Black dots are measurements (sparse)

Data-driven versus Model-based

Example: Traffic density estimation



Estimated density - Model

- 2-D map (spatiotemporal diagram)
- colormap = density
- Black dots are measurements (sparse)
- Model for traffic

$$\frac{\partial u}{\partial t} + V_f(1-2u)\frac{\partial u}{\partial x} = 0$$

Data-driven versus Model-based

Data-driven

- ⊕ Good estimation if large dataset
- ⊕ Robust to noise and disturbances
- ⊕ No curse of dimensionality
- ⊕ Can infer hidden pattern
- ⊖ Large computational time
- ⊖ No convergence proof
- ⊖ Require a large dataset

Data-driven versus Model-based

Data-driven

- ⊕ Good estimation if large dataset
- ⊕ Robust to noise and disturbances
- ⊕ No curse of dimensionality
- ⊕ Can infer hidden pattern
- ⊖ Large computational time
- ⊖ No convergence proof
- ⊖ Require a large dataset

Model-based

- ⊕ No measurements needed
- ⊕ Fast to compute
- ⊕ Error bounds on the solution
- ⊖ Sometimes not robust to noise
- ⊖ Require a precise model
- ⊖ Subject to curse of dimensionality

Can we use both Data-driven and Model-based simulation tools?

Physics-Informed Neural Network (PINN)

Machine Learning

$$u^\star = \operatorname{argmin}_{\mathbf{u}} \quad \frac{1}{N}\sum_{i=1}^{N} L\left(\nu_i, y_i, \mathbf{u}\right)$$

Differential Equation

$$\begin{cases} \dfrac{\partial v}{\partial t}(t,x) + \mathcal{N}[v](t,x) = 0, & (t,x) \in [0,T) \times \Omega, \\ v(\cdot,x)|_{x\in\partial\Omega} = v_{\partial\Omega}, \\ v(0,\cdot) = v_0. \end{cases}$$

Physics-Informed Neural Networks

Optimization

$$\theta^\star = \operatorname{argmin}_{\boldsymbol{\theta}} \quad \frac{1}{N}\sum_{i=1}^{N} \left(u_{\boldsymbol{\theta}}(t_i,x_i) - y_i\right)^2$$

Measurement error

Constraint

$$\text{s. t.} \quad \frac{\partial u_{\boldsymbol{\theta}}}{\partial t}(t,x) + \mathcal{N}[u_{\boldsymbol{\theta}}](t,x) = 0, \quad (t,x) \in [0,T) \times \Omega.$$

Physics-Informed Neural Network (PINN)

Machine Learning

$$u^\star = \operatorname{argmin}_{\mathbf{u}} \quad \frac{1}{N}\sum_{i=1}^{N} L\left(\nu_i, y_i, \mathbf{u}\right)$$

Differential Equation

$$\begin{cases} \dfrac{\partial v}{\partial t}(t,x) + \mathcal{N}[v](t,x) = 0, & (t,x) \in [0,T)\times\,\Omega, \\ v(\cdot,x)|_{x\in\partial\Omega} = v_{\partial\Omega}, \\ v(0,\cdot) = v_0. \end{cases}$$

Physics-Informed Neural Networks

Optimization

$$\theta^\star = \operatorname{argmin}_{\boldsymbol{\theta}} \quad \frac{1}{N}\sum_{i=1}^{N} \left(u_{\boldsymbol{\theta}}(t_i,x_i) - y_i\right)^2$$

Measurement error

Constraint

$$\text{s. t.} \quad \frac{\partial u_{\boldsymbol{\theta}}}{\partial t}(t,x) + \mathcal{N}[u_{\boldsymbol{\theta}}](t,x) = 0, \quad (t,x) \in [0,T)\times\,\Omega.$$

$\Rightarrow$ Learning under constraint problem

Optimize a PINN

$$\theta^\star = \text{argmin}_{\boldsymbol{\theta}} \quad \sum_{i=1}^{N} \frac{\left(u_{\boldsymbol{\theta}}(t_i, x_i) - y_i\right)^2}{N}$$

$$\text{s. t.} \qquad \frac{\partial u_{\boldsymbol{\theta}}}{\partial t}(t,x) + \mathcal{N}[u_{\boldsymbol{\theta}}](t,x) = 0, \quad (t,x) \in [0,T) \times \Omega.$$

Optimize a PINN

$$\theta^\star = \text{argmin}_{\boldsymbol{\theta}} \quad \sum_{i=1}^{N} \frac{(u_{\boldsymbol{\theta}}(t_i, x_i) - y_i)^2}{N}$$

$$\text{s. t.} \quad \iint_{[0,T]\times\Omega} \left(\frac{\partial u_{\boldsymbol{\theta}}}{\partial t}(t,x) + \mathcal{N}[u_{\boldsymbol{\theta}}](t,x) \right)^2 dtdx = 0.$$

Optimize a PINN

$$\theta^\star = \text{argmin}_{\boldsymbol{\theta}} \max_{\lambda \geq 0} \sum_{i=1}^{N} \frac{\left(u_{\boldsymbol{\theta}}(t_i, x_i) - y_i\right)^2}{N} + \lambda \underbrace{\iint_{[0,T]\times\Omega} \left(\frac{\partial u_{\boldsymbol{\theta}}}{\partial t}(t,x) + \mathcal{N}[u_{\boldsymbol{\theta}}](t,x) \right)^2 dtdx}_{\text{Physics cost}}$$

Optimize a PINN

$$\theta^\star = \text{argmin}_{\boldsymbol{\theta}} \max_{\lambda \geq 0} \sum_{i=1}^{N} \frac{\left(u_{\boldsymbol{\theta}}(t_i, x_i) - y_i\right)^2}{N} + \lambda \frac{1}{N_{\text{phy}}} \sum_{j=1}^{N_{\text{phy}}} \left(\frac{\partial u_{\boldsymbol{\theta}}}{\partial t}(t_i^p, x_i^p) + \mathcal{N}[u_{\boldsymbol{\theta}}](t_i^p, x_i^p) \right)^2$$

$$= \text{argmin}_{\boldsymbol{\theta}} \max_{\lambda \geq 0} \mathcal{L}_\lambda(\theta)$$

Optimize a PINN

$$\theta^\star = \text{argmin}_{\boldsymbol{\theta}} \max_{\lambda \geq 0} \sum_{i=1}^{N} \frac{(u_{\boldsymbol{\theta}}(t_i, x_i) - y_i)^2}{N} + \lambda \frac{1}{N_{\text{phy}}} \sum_{j=1}^{N_{\text{phy}}} \left(\frac{\partial u_{\boldsymbol{\theta}}}{\partial t}(t_i^p, x_i^p) + \mathcal{N}[u_{\boldsymbol{\theta}}](t_i^p, x_i^p) \right)^2$$

$$= \text{argmin}_{\boldsymbol{\theta}} \max_{\lambda \geq 0} \mathcal{L}_\lambda(\theta)$$

- Primal-dual optimization can solve the previous problem:

$$\theta \leftarrow \theta - \alpha_\theta \nabla_\theta \mathcal{L}_\lambda(\theta) \qquad \lambda \leftarrow \theta + \alpha_\lambda \nabla_\lambda \mathcal{L}_\lambda(\theta)$$

Optimize a PINN

$$\theta^\star = \text{argmin}_{\boldsymbol{\theta}} \max_{\lambda \geq 0} \sum_{i=1}^{N} \frac{\left(u_{\boldsymbol{\theta}}(t_i, x_i) - y_i\right)^2}{N} + \lambda \frac{1}{N_{\text{phy}}} \sum_{j=1}^{N_{\text{phy}}} \left(\frac{\partial u_{\boldsymbol{\theta}}}{\partial t}(t_i^p, x_i^p) + \mathcal{N}[u_{\boldsymbol{\theta}}](t_i^p, x_i^p) \right)^2$$

$$= \text{argmin}_{\boldsymbol{\theta}} \max_{\lambda \geq 0} \mathcal{L}_\lambda(\theta)$$

- Primal-dual optimization can solve the previous problem;
- About the number of physics points N_{phy}:

Optimize a PINN

$$\theta^\star = \text{argmin}_{\boldsymbol{\theta}} \max_{\lambda \geq 0} \sum_{i=1}^{N} \frac{\left(u_{\boldsymbol{\theta}}(t_i, x_i) - y_i\right)^2}{N} + \lambda \frac{1}{N_{\text{phy}}} \sum_{j=1}^{N_{\text{phy}}} \left(\frac{\partial u_{\boldsymbol{\theta}}}{\partial t}(t_i^p, x_i^p) + \mathcal{N}[u_{\boldsymbol{\theta}}](t_i^p, x_i^p) \right)^2$$

$$= \text{argmin}_{\boldsymbol{\theta}} \max_{\lambda \geq 0} \mathcal{L}_\lambda(\theta)$$

- Primal-dual optimization can solve the previous problem;
- About the number of physics points N_{phy}:
  - New dataset: $N + N_{\text{phy}}$;

Optimize a PINN

$$\theta^\star = \text{argmin}_{\boldsymbol{\theta}} \max_{\lambda \geq 0} \sum_{i=1}^{N} \frac{\left(u_{\boldsymbol{\theta}}(t_i, x_i) - y_i\right)^2}{N} + \lambda \frac{1}{N_{\text{phy}}} \sum_{j=1}^{N_{\text{phy}}} \left(\frac{\partial u_{\boldsymbol{\theta}}}{\partial t}(t_i^p, x_i^p) + \mathcal{N}[u_{\boldsymbol{\theta}}](t_i^p, x_i^p) \right)^2$$

$$= \text{argmin}_{\boldsymbol{\theta}} \max_{\lambda \geq 0} \mathcal{L}_\lambda(\theta)$$

- Primal-dual optimization can solve the previous problem;
- About the number of physics points N_{phy}:
  - New dataset: $N + N_{\text{phy}}$;
  - The confidence in the model relates to the number of physics points;

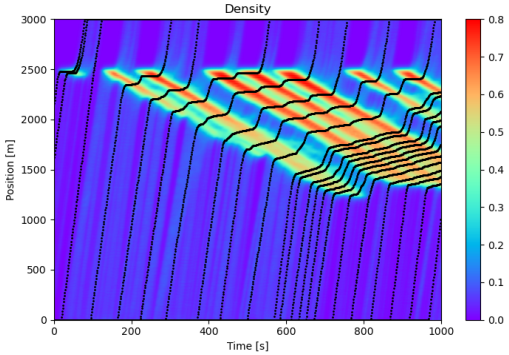
Optimize a PINN

$$\theta^\star = \text{argmin}_{\boldsymbol{\theta},\mu} \max_{\lambda \geq 0} \sum_{i=1}^{N} \frac{(u_{\boldsymbol{\theta}}(t_i, x_i) - y_i)^2}{N} + \lambda \frac{1}{N_{\text{phy}}} \sum_{j=1}^{N_{\text{phy}}} \left(\frac{\partial u_{\boldsymbol{\theta}}}{\partial t}(t_i^p, x_i^p) + \mathcal{N}_\mu[u_{\boldsymbol{\theta}}](t_i^p, x_i^p) \right)^2$$

$$= \text{argmin}_{\boldsymbol{\theta},\mu} \max_{\lambda \geq 0} \mathcal{L}_\lambda(\theta)$$

- Primal-dual optimization can solve the previous problem;
- About the number of physics points N_{phy}:
  - New dataset: $N + N_{\text{phy}}$;
  - The confidence in the model relates to the number of physics points;
- The operator $\mathcal{N}_\mu$ can be defined with uncertain parameters μ $\Rightarrow$ identification.

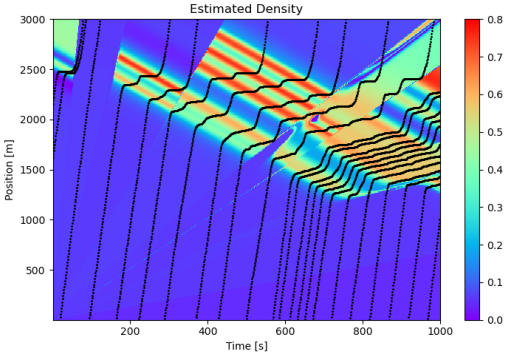
PINN example: Traffic density estimation



Real density - Road

- 2-D map (spatiotemporal diagram)
- colormap = density
- Black dots are measurements (sparse)

PINN example: Traffic density estimation

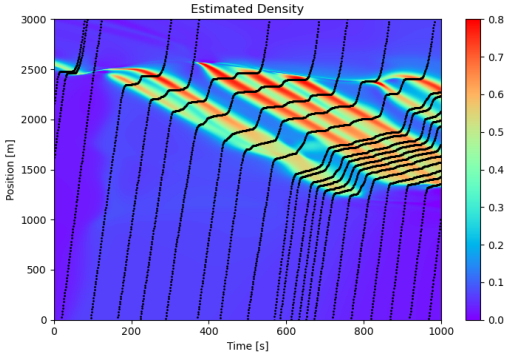


Estimated density - Model

- 2-D map (spatiotemporal diagram)
- colormap = density
- Black dots are measurements (sparse)
- Model for traffic

$$\frac{\partial u}{\partial t} + V_f(1-2u)\frac{\partial u}{\partial x} = 0$$

PINN example: Traffic density estimation



Estimated density - PINN

- 2-D map (spatiotemporal diagram)
- colormap = density
- Black dots are measurements (sparse)
- Model for traffic

$$\frac{\partial u}{\partial t} + \left(\sum_{k=0}^{N_k} \mu_k u^k \right) \frac{\partial u}{\partial x} = 0$$

Advantages and drawbacks of PINN

PINN summary

- ⊕ Good estimation with small/large dataset
- ⊕ Robust to noise and disturbances
- ⊕ No curse of dimensionality
- ⊕ Can infer hidden pattern
- ⊕ Model tuning
- ⊖ Large computational time
- ⊖ No convergence proof

Advantages and drawbacks of PINN

PINN summary

- ⊕ Good estimation with small/large dataset
- ⊕ Robust to noise and disturbances
- ⊕ No curse of dimensionality
- ⊕ Can infer hidden pattern
- ⊕ Model tuning
- ⊖ Large computational time
- ⊖ No convergence proof

Challenges

- Improve the convergence
  - Less stochastic output (robust w.r.t. training)
  - Fastest convergence
  - Conditions for convergence
- Prove approximation capacity for a broader class of operators $\mathcal{N}$
- Solve strongly coupled systems (multiple PDEs)

Perspectives for PINN

General perspectives:

- Investigate other neural network architectures (LSTM, Convolutional network ...);
- Change the training procedure (primal-dual update, ...)
- Use confidence intervals over several optimizations

Perspectives for PINN

General perspectives:

- Investigate other neural network architectures (LSTM, Convolutional network ...);
- Change the training procedure (primal-dual update, ...)
- Use confidence intervals over several optimizations

Control perspectives:

- Extend the current work to real-time observation
- Use reinforcement learning to compute a control action

Perspectives for PINN

General perspectives:

- Investigate other neural network architectures (LSTM, Convolutional network ...);
- Change the training procedure (primal-dual update, ...)
- Use confidence intervals over several optimizations

Control perspectives:

- Extend the current work to real-time observation
- Use reinforcement learning to compute a control action

Broader perspectives:

- Use the PINN methodology in other domains (fluid mechanics, signal processing, astrology...)
- Increment the framework for use in different contexts (?)

Bibliography

- M. Raissi, P. Perdikaris, G. E. Karniadakis, "Physics-informed neural net-works: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations", *Journal of Computational Physics*, 2019
- J. Sirignano, K. Spiliopoulos, "DGM: A deep learning algorithm for solving partial differential equations", *Journal of Computational Physics*, 2018
- J. Liu, M. Barreau, M.Cicic, and K. H. Johansson, "Learning-based traffic state reconstruction using probe vehicles", *CTS*, 2021
- I. Goodfellow, Y. Bengio, A. Courville, *Deep Learning*, 2016.

Thanks!