Artificial Neural Networks For Solving Ordinary and Partial Differential Equations

Team 42

Speakers: Jacky Behrendt and Viet Duc Nguyen

Seminar: Funktionalanalysis und Data Science WS 19/20

Applied Functional Analysis Group
Technische Universität Berlin

Table of Contents

The roadmap for today

What will you learn today?

Problem: Initial value or boundary value problem

Methods to solve this problem:

- Runge Kutta
- Finite element methods
- Neural networks

Universal Approximation Theorem (Cybenko, 1989)

Every continuous function on a compact set can be arbitrarily well approximated with a neural network with one single hidden layer.

Advantages of using a neural network:

Advantages of using a neural network:

- solution is differentiable, and in closed analytic form

Advantages of using a neural network:

- solution is differentiable, and in closed analytic form
- few number of parameters to tune $\rightsquigarrow$ memory efficient

Advantages of using a neural network:

- solution is differentiable, and in closed analytic form
- few number of parameters to tune $\rightsquigarrow$ memory efficient
- very general method $\rightsquigarrow$ applicable on ODEs, system of ODEs and PDEs

Advantages of using a neural network:

- solution is differentiable, and in closed analytic form
- few number of parameters to tune $\rightsquigarrow$ memory efficient
- very general method $\rightsquigarrow$ applicable on ODEs, system of ODEs and PDEs
- takes advantage of hardware architecture $\rightsquigarrow$ use of neural processors, method is parallelizable, ...

Table of Contents

Where we are

- Problem: $G(x, u(x), \nabla u(x), \nabla^2 u(x)) = 0$ for all $x \in D$ such that u fulfills boundary condition, with domain $D \subset \mathbb{R}^n$ and boundary $S \subset \mathbb{R}^n$

- Problem: $G(x, u(x), \nabla u(x), \nabla^2 u(x)) = 0$ for all $x \in D$ such that u fulfills boundary condition, with domain $D \subset \mathbb{R}^n$ and boundary $S \subset \mathbb{R}^n$
- Goal: reformulate the original problem to an unconstrained optimization problem

- Problem: $G(x, u(x), \nabla u(x), \nabla^2 u(x)) = 0$ for all $x \in D$ such that u fulfills boundary condition, with domain $D \subset \mathbb{R}^n$ and boundary $S \subset \mathbb{R}^n$
- Goal: reformulate the original problem to an unconstrained optimization problem
- Idea: discretize D and S to $\hat{D}$ and $\hat{S}$

- Problem: $G(x, u(x), \nabla u(x), \nabla^2 u(x)) = 0$ for all $x \in D$ such that u fulfills boundary condition, with domain $D \subset \mathbb{R}^n$ and boundary $S \subset \mathbb{R}^n$
- Goal: reformulate the original problem to an unconstrained optimization problem
- Idea: discretize D and S to $\hat{D}$ and $\hat{S}$
- use the collocation method to obtain

$$G(x_i, u(x_i), \nabla u(x_i), \nabla^2 u(x_i)) = 0 \quad \forall x_i \in \hat{D}$$

such that u fulfills the boundary conditions

- Let u_N be the trial solution of the following form:

$$u_N(x) = A(x) + F(x, N_p(x))$$

- A contains no trainable parameters and satisfies boundary conditions
- F does not contribute to the boundary conditions
- N_p is a neural network with trainable parameters p

- formulate to an *unconstrained* optimization problem:

$$\min_p \sum_{x_i \in \hat{D}} (G(x_i, u_N(x_i), \nabla u_N(x_i), \nabla^2 u_N(x_i)))^2$$

- train N_p such that u_N minimizes the optimization problem by using any gradient method:

$$u_N(x) = A(x) + F(x, \underbrace{N_p(x)}_{\text{to be trained}})$$

- Consider the following differential equation

$$\begin{cases} u'(x) & = f(x, u(x)) \\ u(0) & = u_0 \end{cases}$$

- Consider the following differential equation

$$\begin{cases} u'(x) & = f(x, u(x)) \\ u(0) & = u_0 \end{cases}$$

- A trial solution may be given by:

$$u_N(x) = u_0 + x \cdot N_p(x)$$

with $A(x) = u_0$ and $F(x, N_p(x)) = xN_p(x)$.

- Consider the following differential equation

$$\begin{cases} u'(x) &= f(x, u(x)) \\ u(0) &= u_0 \end{cases}$$

- A trial solution may be given by:

$$u_N(x) = u_0 + x \cdot N_p(x)$$

with $A(x) = u_0$ and $F(x, N_p(x)) = xN_p(x)$.

- F prevents N_p from contributing to the boundaries

Idea and Description of the Method

Compute the gradient of the trial solution u_N for minimization

- Remember our goal is solving

$$\min_p E(p) = \min_p \sum_{x_i \in \hat{D}} (G(x_i, u_N(x_i), \nabla u_N(x_i), \nabla^2 u_N(x_i)))^2$$

- Remember our goal is solving

$$\min_p E(p) = \min_p \sum_{x_i \in \hat{D}} (G(x_i, u_N(x_i), \nabla u_N(x_i), \nabla^2 u_N(x_i)))^2$$

- N_p is a multilayer perceptron with n input units, one hidden layer with h hidden units and one linear output unit

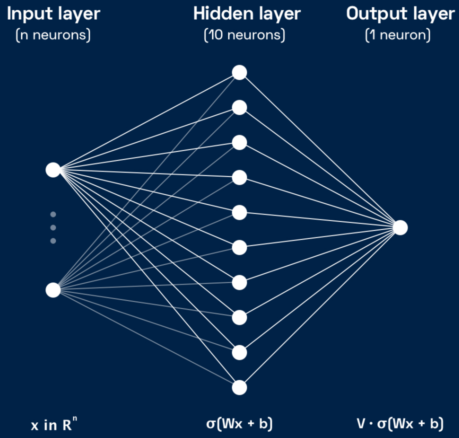
- Remember our goal is solving

$$\min_p E(p) = \min_p \sum_{x_i \in \hat{D}} (G(x_i, u_N(x_i), \nabla u_N(x_i), \nabla^2 u_N(x_i)))^2$$

- N_p is a multilayer perceptron with n input units, one hidden layer with h hidden units and one linear output unit
- For $x \in \mathbb{R}^n$ the neural network outputs $N_p(x) = \sum_{i=1}^{h} v_i \sigma(z_i)$ with $z_i = \sum_{j=1}^{n} w_{i,j} x_j + b_i$
  - the weights $w_{i,j}$ for input unit j to hidden unit i
  - bias b_i at the hidden unit i
  - weights v_i of the hidden layer
  - sigmoid function $\sigma(x) = \frac{1}{1+e^{-x}}$

Example Neural Network

Team 42



- $N_p(x) = \sum_{i=1}^{h} v_i \sigma(z_i)$ with $z_i = \sum_{j=1}^{n} w_{i,j} x_j + b_i$
- $\frac{d^k N_p(x)}{dx_j^k} = \sum_{i=1}^{h} v_i w_{i,j}^k \sigma^{(k)}(z_i)$
  with $\sigma^{(k)}$ the k-th order derivative of the sigmoid function
- we can conclude

$$\frac{d^{\lambda_1}}{dx_1^{\lambda_1}} \frac{d^{\lambda_2}}{dx_2^{\lambda_2}} \cdots \frac{d^{\lambda_n}}{dx_n^{\lambda_n}} N_p(x) = \sum_{i=1}^{n} v_i P_i \sigma^{(\Lambda)}(z_i)$$

with $P_i = \prod_{k=1}^{n} w_{i,k}^{\lambda_k}, \Lambda = \sum_{i=1}^{n} \lambda_i$

Table of Contents

Where we are

- In the following experiments a shallow neural network with one hidden layer was used
- The hidden layer contained ten neurons
- We ask the following questions:
  - How good is the approximation?
  - How long does the training take?
  - How well does it perform in comparison to other methods?

Numerical experiments

Problem

- Consider the following differential equation

$$\begin{cases} u'(x) = -\frac{1}{5}u(x) + e^{-\frac{1}{5}x}\cos(x) \\ u(0) = 0 \end{cases}, \quad x \in [0, 2]$$

- exact solution is $u_a(x) = e^{-\frac{1}{5}x}\sin(x)$
- trial solution is $u_N(x) = xN_p(x)$
- loss function is

$$\sum_{x_i \in \hat{D}} \left(u'_N(x_i) + \frac{1}{5}u_N(x_i) - e^{-\frac{1}{5}x_i}\cos(x_i) \right)^2$$

View code on
https://cutt.ly/tu-berlin-nn-01

- Solution obtained through FEM is not in closed analytic form
- At training points FEM is more accurate than the NN approach
- NN approach performes better at interpolation point

TABLE I
MAXIMUM DEVIATION FROM THE EXACT SOLUTION FOR THE NEURAL AND THE FINITE-ELEMENT METHODS

	Neural Method		*Finite Element*	
Problem No.	Training set	Interpolation set	Training set	Interpolation set
5	5×10^{-7}	5×10^{-7}	2×10^{-8}	1.5×10^{-5}
6	6×10^{-6}	6×10^{-6}	7×10^{-7}	4×10^{-5}
7	1.5×10^{-5}	1.5×10^{-5}	6×10^{-7}	4×10^{-5}

Comparison with Finite Element Method

Comparison of Parameters and Computation time

- FEM needs an excessive number of parameters $\rightsquigarrow$ high memory requirements

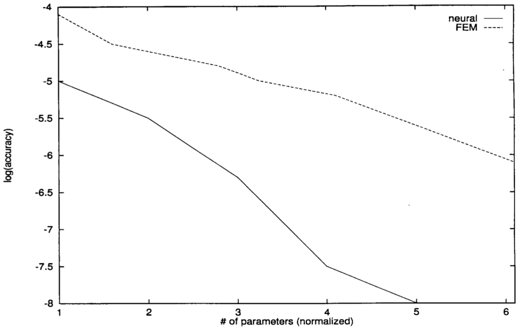


Fig. 16. Plot of logarithm of the maximum convergence error at the interpolation points as a function of the normalized number of parameters for the neural and the FEM approach.

- The neural network approach converges faster for a larger number of parameters

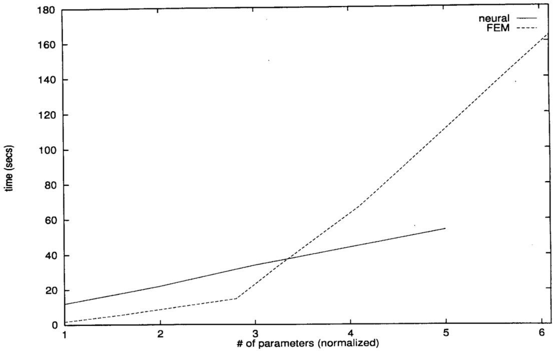


Fig. 17. Plot of the time to converge as a function of the normalized number of parameters for the neural and the FEM approach.

Table of Contents

Where we are

- The presented method provides an accurate, differential solution in closed analytic form
- Accuracy of the approximated solution is based on the ability of Neural Networks to approximate any continuous function
- The choice of the trial solution leads to an unconstrained optimization problem which can be solved any minimization technique

- Does the neural network perform better if we increase the number of hidden layers or the number of hidden units in a layer?
- How do we chose optimal training points?
- How is the performance in a high dimensional setting?

Literature

Artificial Neural Networks for Solving Ordinary and Partial Differential Equations (I. E. Lagaris, A. Likas, D. I. Fotiadis, 1997)