Solving differential equations using neural networks

Eina B. Jørgensen, Anna Lina P. Sjur and Jan-Adrian H. Kallmyr

December 9, 2019

Abstract

1 Introduction

In physics, as well as many other fields, differential equations (DE's) play a central role in analysis of a wide variety of problems. As such, it is important to use an efficient and accurate algorithm to produce reliable results. There exists many such algorithms of different orders, and each have their uses for different DE's. More complex DE's usually require higher order algorithms such as fourth order Runge-Kutta, while simple DE's can do with a first or second order algorithm such as Forward Euler (FE), or Centered Difference (CD), respectively. Higher order methods usually perform slower, and for complex coupled models, it is important to choose just the right algorithm for solving each sub-problem, as a slow method might significantly slow down model runs. It is therefore of interest if a general method for solving DE's exist which is efficient. A possible candidate are Neural Networks (NN's), which can approximate any function. In general, NN's for solving DE's could be useful if they offer a good accuracy/efficiency trade-off for many different types of equations, or if they outperform traditional methods. We will focus on the latter, and compare a traditional CD/FE-approach with a NN-approach. Furthermore, we will also look at an immediate application of the NN-approach in finding extrema eigenpairs (Yi et al., 2004) and compare this with the standard approach.

The Partial Differential Equation (PDE) we will solve (eq. 1), as well as all background theory and methods, can be found in Section 2. Our most important results are showcased in Section 3, and mainly consists of comparisons between different methods. In Section 4 we discuss the pros and cons of each method, as well as taking a deeper look at interesting results. Finally, in section 5 we summarise the article, presenting the most important takeaways, as well as possible future uses.

2 Theory

2.1 The general Problem

The equation to be solved with different methods in this project is a simple diffusion equation

$$\frac{\partial^2 u(x,t)}{\partial x^2} = \frac{\partial u(x,t)}{\partial t} \quad t > 0 \quad x \in [0, L] \quad (1)$$

Or $u_{xx} = u_t$. Using $L = 1$, with initial condition at t=0:

$$u(x,0) = I(x) = \sin(\pi x) \quad (2)$$

and Dirichlet boundry conditions

$$u(0,t) = u(L,t) = 0 \quad t \geq 0$$

This problem can for instance model the temperature of rod that has been heated in the middle, and as time progresses the temperature is transported through the rod and falls.

2.2 Discretization

For time discretization, as time is only used in first order derivative, we will use the explicit Forward Euler Scheme, which gives an error proportional to Δt (SOURCE).

$$\frac{\partial u(x,t)}{\partial t} \approx \frac{u(x,t+\Delta t) - u(x,t)}{\Delta t} \tag{3}$$

For the spatial discretization we use centered difference, which has an error proportional to Δx^2 (SOURCE).

$$\frac{\partial^2 u(x,t)}{\partial x^2} = \frac{u(x+\Delta x,t) - 2u(x,t) + u(x-\Delta x,t)}{\Delta x^2} \tag{4}$$

On a discrete time and space grid, $u(x,t) = u(x_i, t_n)$, $t + \Delta t = t_{n+1}$ and so on. For simplicity we use the notation $u_i^n = u(x_i, t_n)$. The equation in it's discrete form is then

$$\begin{aligned} u_{xx} &= u_t \\ [u_{xx}]_i^n &= [u_t]_i^n \\ \frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{\Delta x^2} &= \frac{u_i^{n+1} - u_i^n}{\Delta t} \end{aligned} \tag{5}$$

Solving this for u_i^{n+1} we can caluclate the next time step for each spatial point i:

$$u_i^{n+1} = \frac{\Delta t}{\Delta x^2}\left(u_{i+1}^n - 2u_i^n + u_{i-1}^n\right) + u_i^n \tag{6}$$

Which has a stability level for the grid resolution

$$\frac{\Delta t}{\Delta x^2} \leq \frac{1}{2}$$

2.3 The Exact Solution

As we will be comparing the precision of the different ways of solving the partial differential equation, we need to calculate the the exact solution in order to calculate the error. Through sepeation of variables, the equation can be expressed as

$$u(x,t) = X(x)T(t) \tag{7}$$

Differentiating this according to (1) and moving some terms, we get

$$\frac{X''(x)}{X(x)} = \frac{T'(t)}{T(t)}$$

As the to sides of this equation are not dependant on the same variables, they must both be equal to a consant. (We can choose this constant to be $-\lambda^2$). This gives the tho equations.

$$\begin{aligned} X''(x) &= -\lambda^2 X(x) \\ T'(t) &= -\lambda^2 T(t) \end{aligned}$$

For X can have three possible forms given by the characteristic equation. In order to satisfy the inital condition (2), $X(x)$ must be on the form

$$X(x) = B\sin(\lambda x) + C\cos(\lambda x)$$

The inital condition then rules $C = 0, \lambda = \pi$. For $T(t)$ the solution is on the form

$$T(t) = Ae^{-\lambda^2 t}$$

As we know $\lambda = \pi$ the solution is then:

$$u(x,t) = X(x)T(t) = Ae^{-\pi^2 t}B\sin(\pi x)$$

And finally from the initial condition, we know that $A \cdot B = 1$, and the exact solution is

$$u(x,t) = e^{-\pi^2 t}\sin(\pi x) \tag{8}$$

2.4 Solving PDEs with Neural Networks

How we use gradient descent to minimize a cost function with forward feeding when using a Neural Network is discussed in detail in

(REF TIL PROSJEKT 2). Much of the same logic will here be applied to solve a partial differential equation.

A trial function $\Psi(x,t)$ can be approximated, and our aim is to get this Ψ as close to the true function u as possible Lagaris et al. (1998). Since the equation to be solved is (1), the corresponding equation is

$$\frac{\partial^2 \Psi(x,t)}{\partial x^2} = \frac{\partial \Psi(x,t)}{\partial t} \quad t > 0 \quad x \in [0, L]$$

The error (or residual) in the approximation is then

$$E = \frac{\partial^2 \Psi(x,t)}{\partial x^2} - \frac{\partial \Psi(x,t)}{\partial t} \tag{9}$$

The cost function to be mimimized by the Neural Network is the sum of this E, evaluated at each point in the space and time grid.

For each iteration in the calculations, we will update our trial function based on the Neural Network's previous calculations. Therefore we must choose a fitting form for our trial function. This is based on the order of the PDE, and its initial condition. To satisfy the inital condition and Dirichlet conditions, we choose:

$$\Psi(x,t) = (1-t)I(x) + x(1-x)tN(x,t,p) \tag{10}$$

Where $I(x)$ is the initial condition, and $N(x,t,p)$ is the output from the neural network, and p is the weigts.

Then, for each iteration in the network, the partial derivatives of Ψ is calculated, formulating the cost function and a new $N(x,t,p)$ for minimization is calculated, being used in the next iteration.

How small we are able to get the cost is dependant of the maximum number of iterations we allow the Network to execute, the learning rate, and the structure of the Neural Network in terms of the number of hidden layers, and the number of nodes in each layer.

2.5 Implementing the Neural Network

There are many ways of implementing the Neural Network method for solving this partial differential equation. In this project, we have chosen to use *TensorFlow* for python3, as it is fast, stable and relatively simple to use. For implementation, see (GITHUB REPO).

3 Results

4 Discussion

5 Conclusion

References

Isaac E Lagaris, Aristidis Likas, and Dimitrios I Fotiadis. Artificial neural networks for solving ordinary and partial differential equations. *IEEE transactions on neural networks*, 9(5):987–1000, 1998.

Zhang Yi, Yan Fu, and Hua Jin Tang. Neural networks based approach for computing eigenvectors and eigenvalues of symmetric matrix. *Computers & Mathematics with Applications*, 47(8-9):1155–1164, 2004.