Solving differential equations using neural networks

Eirik Nordgård

Geophysical Institute, University of Oslo

2019
December

Abstract

All material for this project may be found on `https://github.com/eirikngard/FYS-STK4155/tree/master/Project_2`

1 Introduction

Solving differential equations (DE's) is an important and central problem in physics, meteorology and many other fields. Doing so efficiently and accurately is therefore of great interest, and various algorithms have been used to sole this problem. Complex DE's typically require high order algorithms while simple DE's can be solved using first or second order algorithms such as Forward Euler Scheme. Time and computer power consumption is the main problem solving DE's. A coupled model computing several equations for thousands or millions of grid-boxes may suffer badly from a poor DE solving algorithm, both in terms of accuracy and time consumption. Intro on the field of differential equations and solving them. How to use a NN for solving a problem like this. The paper by [?] et. al. (1998) is one of many that suggests solving DE's can be done with high precision using Neural Networks (NN's). However, the computational cost may be quite large. The question of interest regarding the use of NN's is whether they have the ability to outperform traditional algorithms in speed and efficiency. In this project this will be investigated comparing the Forward Euler approach with the Neural Network approach. Furthermore, following the approach in the paper by [?] et. al. (year), the NN will be used to compute extrema eigenvectors and corresponding eigenvalues. In this project the Diffusion Equation will be solved. Physically this equation can for example represent the temperature gradient through a rod, describing how the temperature decays along the rod with time. Very short and specific on what can be found in the differnt sections.

2 Theory

2.1 The general problem

The problem to be solved is the simple diffusion equation

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

Another way to write this problem is $u_{xx} = u_t$. Initial conditions is necessary. Using $L = 1$, the initial condition at $t = 0$ is given by

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

Furthermore, Dirichlet boundary conditions are also used, given by

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

As an example, this differential equation and its initial and boundary conditions could represent the temperature of a heated rod. As time progress the heat is transported through the rod while the temperature decreases along the way.

2.2 Exact solution of the diffusion equation

To investigate how the solutions to the partial differential equation performs it is necessary to compare them with some benchmark solution. The exact solution serves this purpose, and it will be calculated in this section. Through separation of variables, the (WHAT EQ) equation can be expressed as

$$u(x,t) = X(x)T(t) \tag{3}$$

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

$$\frac{X''(x)}{X(x)} = \frac{T'(t)}{T(t)} \tag{4}$$

As the to sides of (4) are not dependant on the same variables, they must both be equal to a constant. For convenience the constant is chosen to be $-\omega^2$. This gives the two equations.

$$\begin{aligned} X''(x) &= -\omega^2 X(x) \\ T'(t) &= -\omega^2 T(t) \end{aligned} \tag{5}$$

Now the solution X can appear in three possible ways given by the characteristic equation. In order to satisfy the initial condition (2), $X(x)$ must be on the form

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

The initial condition then rules $C = 0, \omega = \pi$. For $T(t)$ in (5) the solution is on the form

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

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

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

Finally, from the initial condition, we know that $A \cdot B = 1$, hence the exact solution denoted *exact* must be

$$u_{exact}(x,t) = e^{-\pi^2 t}\sin(\pi x) \tag{6}$$

2.3 Solution using explicit Euler scheme

Now it is desired to solve the equation with a Euler scheme. To make this possible, eq. (1) must be discretized in both time and space. 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). This scheme is given as

$$u(x,t)t \approx \frac{u(x,t+\Delta t) - u(x,t)}{\Delta t} \tag{7}$$

For the spatial discretization a centred difference is used, which has an error proportional to Δx^2 (SOURCE), given by

$$u(x,t)x = \frac{u(x+\Delta x,t) - 2u(x,t) + u(x-\Delta x,t)}{\Delta x^2} \tag{8}$$

on a discrete time and space grid, $u(x,t) = u(x_i,t_n)$, $t + \Delta t_n = t_{n+1}$ and so on. Simplifying this notation yields $u_i^n = u(x_i,t_n)$. On a discrete form eq. (1) is then

$$\begin{aligned} u_{xx} &= u_t \\ [u_{xx}]_i^n &= [u_t]_i^n \\ \frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{\Delta x^2} &= \frac{u_i^{n+1} - u_i^n}{\Delta t} \end{aligned} \tag{9}$$

2

Solving this for u_i^{n+1} provides the solution u to (1) for each spatial point i:

$$u_i^{n+1} = \frac{\Delta t}{\Delta x^2}(u_{i+1}^n - 2u_i^n + u_{i-1}^n) + u_i^n \quad (10)$$

(10) is stable (SOMETHING MORE ON WHY THIS IS THE FACT) for the following grid resolutions:

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

2.3.1 Implementation

Maybe include here some implementation of the method. Posibly include how it performs here compared to the exact?

2.4 Solution using a Neural Network

Solving the PDE can also be done using a neural network. BRIEF INTRODUCTION TO WHAT A NN IS. For more information on neural networks and how they work, see (LINK TO PLACE WITH INFO/PROJECT2). In this project the neural network functionality within *TensorFlow* for python3 is used, as this is stable, fast and simple to use compared to building a neural network from scratch.

In order to solve PDEs with a neural network, we approximate the true function u with a trial function $\Theta(x,t)$. Thus the aim is to calculate Θ as close to the true function u as possible [?] (SOURCE). When aiming to solve Eq. (1), the corresponding equation for the trial function is

$$\Theta(x,t)x = \Theta(x,t)t \quad t > 0 \quad x \in [0, L]$$

The residual of this approximation is then

$$E = \Theta(x,t)x - \Theta(x,t)t \quad (11)$$

The cost function to be minimized by the Neural Network is the sum of E, evaluated at each point in the space and time grid. This equivalent to minimizing the mean squared error.(CORRECT?) The neural network iterates a given number of times. For each iteration the trial function is updated based on the networks previous calculated trial function. Choosing the correct "form" of the trial function is important to restrain the residual, and this "form" is based on the order of the PDE and its initial condition. To satisfy both the initial condition and the Dirichlet condition of Eq. (1), the following form is chosen:

$$\Theta(x,t) = (1-t)I(x) + x(1-x)tN(x,t,p), \quad (12)$$

where $I(x)$ is the initial condition, $N(x,t,p)$ is the output from the neural network and p is the weights and biases of the network.

Ideally the process should work accordingly: For each iteration in the network, the partial derivatives of Θ is calculated according to the new state of the network $N(x,t,p)$, updating the cost along the way. As the cost is minimized, the error term E gets closer to zero, and the trial function $\Theta(x,t)$ approaches the true solution of the PDE. In theory the cost can practically reach zero if the number of iterations is big enough, but since it is of small interest to run infinite iteration a *minimum cost value* is chosen to 10^{-3} CHECK WHAT VALUE TO ACTUALLY USE. Learning rate, the structure of the neural network in terms of the number of hidden layers and the number of nodes in each layer is also important in minimizing the cost.

2.5 Computing eigenpairs with Neural Networks

Why it is important/desired to find the eigenvalues and eigenvectors, and what the largest and smallest mean/represent.

eigenvector Λ_{max} corresponding to the largest eigenvalue λ_{max} of the $n \times n$ matrix A

In Yi et. al. [?] a neat way to compute Λ_{max} and λ_{max} is provided. This computation is done by solving the ordinary differential equation

$$\frac{d\Lambda(t)}{dt} = -\Lambda(t) + f(\Lambda(t)), \quad t \geq 0 \qquad (13)$$

where $\Lambda = [\Lambda, \Lambda, \ldots, \Lambda_n]^T$ and $f(\Lambda)$ is given as Here it is an equation for f from YI. Here, I is the $n \times n$ identity matrix.

According to [?], when $t \to \infty$, any random non-zero initial Λ, will approach Λ_{max} if it is not orthogonal to Λ_{max}. The corresponding eigenvalue λ_{max}, is computed by the equation

$$\lambda = \frac{v^T A v}{v^T v} \qquad (14)$$

In (14) A is a symmetric matrix given by

$$A = \frac{Q^T + Q}{2} \qquad (15)$$

where Q is a random, real matrix.

After finding the largest eigenvalue λ_{max}, the smallest eigenvalue λ_{min} corresponding to the eigenvector Λ_{min} is easily found by substituting A with $-A$ in Eq. (15)[?].
UNTIL HERE

As described earlier, a trial function is needed to solve Eq. (??) with a Neural Network. Since $v \in \mathbb{R}^n$, we choose a trial function $\Psi(x, t)$ dependent both on position x and time t, so that for each time step, the approximated eigenvector is given as $[v(1, t), v(2, t), \ldots, v(n, t)]^T$. Eq. (??) can then be rewritten as

$$\frac{\partial \Psi(x, t)}{\partial t} = -\Psi(x, t) + f(\Psi(x, t)) \qquad (16)$$

with $t \geq 0$ and $x = 1, 2, \ldots, n$. We defined the trial function as

$$\Psi(x, t) = v_0 + tN(x, t, p)$$

where v_0 is the initial v, chosen at random. The error is then the difference between the two hand sides of Eq. (16).

2.6 Metrics used to evaluated solutions

The main metric used to evaluate the performance of the neural network compared to the analytic scheme is the Mean Squared Error(MSE). As the name suggests, this metric quantifies the mean of the squares of the error between the prediction $\hat{x}_i$ and the observed value x_i. Thus,

$$MSE = \frac{1}{n} \sum_{i=1}^{n} (x_i - \hat{x}_i)^2$$

where n is the number of samples.

2.7 Data description - the function

3 Method

3.1 Execution of Python Code

4 Results

5 Conclusion

6 Future Work

7 Appendix

References

[1] Hastie, Trevor. Tibshirani, Robert. Friedman, Jerome. *The Elements of Statistical Learning. Data Mining, Interference, and Prediction.*
Second Edition.
Springer, 2009.
Chapter 4, Chapter 11

[2] M. Hjorth-Jensen Lecture Notes in FYS-STK4155. *Data Analysis and Machine Learning: Logistic Regression and Neural Networks.*
URL: `https://github.com/CompPhysics/MachineLearning/tree/master/doc/pub/LogReg`
URL: `https://github.com/CompPhysics/MachineLearning/tree/master/doc/pub/NeuralNet`
Unpublished, 2019.

[3] Nielsen, Michael
URL: `http://neuralnetworksanddeeplearning.com`
Downloaded 13.10.19

[4] *The comparisons of data mining techniques for the predictive accuracy of probability of default of credit card clients.*
I. Yeh and C. Lien
URL: `https://bradzzz.gitbooks.io/ga-seattle-dsi/content/dsi/dsi_05_classification_databases/2.1-lesson/assets/datasets/DefaultCreditCardClients_yeh_2009.pdf`

[5] *Default Payments of Credit Card Clients in Taiwan from 2005.*
Drugov, Vladimir G.
URL: `https://rstudio-pubs-static.s3.amazonaws.com/281390_8a4ea1f1d23043479814ec4a38dbbfd9.html`