Machine Learning: Bra Tittel

Vetle Nevland, Vetle Vikenes & Sigurd SÃÿrlie Rustad

FYS-STK4155 âĂŞ Applied Data Analysis and Machine Learning
Autumn 2021
Department of Physics
University of Oslo

December 2, 2021

Abstract: Coming soon!

Contents

1 Introduction

We will in no way answer all questions linked to the aforementioned methods. So that anyone can reproduce or continue our studies, we list all the code, results and instructions on running the code in our GitHub repository[1].

2 Theory

In the theory-section we aim to give a brief explanation of the main concepts and terminology used in this report.

2.1 The diffusion equation

The full diffusion equation reads

$$\frac{\partial u(\mathbf{r},t)}{\partial t} = \nabla \cdot \left[D(u,\mathbf{r}) \nabla u(\mathbf{r},t) \right], \tag{2.1}$$

where $\mathbf{r}$ is a positional vector and $D(u,r)$ the collective diffusion coefficient. If $D(u,\mathbf{r}) = 1$ the equation simplifies to a linear differential equation

$$\frac{\partial u}{\partial t} = \nabla^2 u(\mathbf{r},t), \tag{2.2}$$

or

$$\left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2} \right) u(x,y,z,t) = \frac{\partial u(x,y,z,t)}{\partial t} \tag{2.3}$$

in cartesian coordinates. In this report we are going to study a one dimensional rod of length $L = 1$. I.e. we need the one dimensional diffusion equation

$$\frac{\partial^2 u(x,t)}{\partial x^2} = \frac{\partial u(x,t)}{\partial t}, \tag{2.4}$$

with boundary conditions

$$u(x,0) = \sin(\pi x) \quad 0 \leq x \leq L, \tag{2.5}$$

$$u(0,t) = 0 \quad t \geq 0 \text{ and} \tag{2.6}$$

$$u(L,t) = 0 \quad t \geq 0. \tag{2.7}$$

2.2 Analytical solution

An analytical solution of the 1D diffusion equation can be dervied using the method of separation of variables.

$$u(x,t) = X(x)T(t)$$

[1] https://github.com/sigurdru/FYS-STK4155/tree/main/project3

The solution is separated into a function X only depending on the independent variable x, and a function T only depending on the independent variable t. Equation 2.4 can then be rewritten as

$$\begin{aligned}\frac{\partial^2 X(x)T(t)}{\partial x^2} &= \frac{\partial X(x)T(t)}{\partial t} \\ T(t)\frac{\partial^2 X(x)}{\partial x^2} &= X(x)\frac{\partial T(t)}{\partial t} \\ \frac{1}{X}\frac{\partial^2 X(x)}{\partial x^2} &= \frac{1}{T}\frac{\partial T(t)}{\partial t}\end{aligned}$$

The core of the method now becomes clear. The independent variables are separated and put on either side of the equation. Because x and t are independent we may fix one of them, say x, while letting the other (t) vary. The left side of the equation is thus constant, and since we have equality the expression on the right side must equal the same constant, for arbitrary t. Therefore, we can set the left side and right side equal to a constant $-k^2$. The reason for defining a negative constant is to prevent a growing solution, which will be clear on the derivation.

$$\frac{1}{X}\frac{\partial^2 X(x)}{\partial x^2} = \frac{1}{T}\frac{\partial T(t)}{\partial t} = -k^2$$

This is solved for the functions X and T separately. ... Nevland: FullfÃÿr utledning

Explicit forward Euler

In this section we want to cover the explicit forward Euler. By explicit we mean that the value at the next grid point is determined entirely by known or previously calculated values.

The one-dimensional diffusion equation (2.4) reads

$$\frac{\partial^2 u(x,t)}{\partial x^2} = \frac{\partial u(x,t)}{\partial t} \quad \text{or} \quad u_{xx} = u_t. \tag{2.8}$$

In this report we are going to study a one dimensional rod of length $L = 1$, with boundary conditions

$$u(x,0) = \sin(\pi x) \quad 0 \leq x \leq L, \tag{2.9}$$
$$u(0,t) = 0 \quad t \geq 0 \text{ and} \tag{2.10}$$
$$u(L,t) = 0 \quad t \geq 0. \tag{2.11}$$

To approximate the solution, we have to discretize the position and time coordinates. We can choose $\Delta x = L/N$ and Δt as small steps in x-direction and time, respectively, where N

are the number of discretized points in x-direction. Then we can define the value domain of t and x,

$$t_j = j\Delta t, \quad j \in \mathbb{N}_0 \quad \wedge \quad x_i = i\Delta x, \quad \{i \in \mathbb{N}_0 | i \leq N\}.$$

The algorithm for explicit forward Euler in one dimension (from [1] chapter 10.2.1) reads

$$u_{i,j+1} = \alpha u_{i-1,j} + (1 - 2\alpha)u_{i,j} + \alpha u_{i+1,j} \tag{2.12}$$

where

$$\alpha = \frac{\Delta t}{\Delta x^2}.$$

This has a local approximate error of $O(\Delta t)$ and $O(\Delta x^2)$. Experiments show that the following bound on α ensures stability.

$$\alpha = \frac{\Delta t}{\Delta x^2} \leq \frac{1}{2}. \tag{2.13}$$

Even though $\alpha = 0.5$ is at the transition between stable and unstable solutions, experiments show that it is a safe choice for the diffusion equation that produces stable solutions [2]. Thus, the stability factor $\alpha = 0.5$ is used as default in this project.

3 Solving Differential equations with deep learning

Many of the concepts we use in this section are covered in a previous report [5]. There, in the theory section, we cover central concepts like deep neural networks, cost functions, gradient descent, etc. In this report we will follow closely the work by Maziar Raissi et.al. [4] and this Jupyter-Notebook[2]. This method works with any partial differential equation (PDE), however for simplicity we will look at the diffusion equation in one dimension (2.4).

The idea is that we have some neural network N with parameters θ, including x and t as inputs, which returns $u_\theta(x,t)$. After training the neural network, one obtains the parameters $\hat{\theta}$ that best approximates its output with the actual solution, $u(x,t)$, of the PDE

$$u_{\hat{\theta}}(x,t) \approx u(x,t).$$

The process starts by defining a trial function $u_\theta(x,t)$ which is an initial guess of the actual solution $u(x,y)$, defined as

$$u_\theta(x,t) = u_0(x,t) + f\Big(x, N(x,\theta)\Big), \tag{3.1}$$

[2] https://colab.research.google.com/github/janblechschmidt/PDEsByNNs/blob/main/PINN_Solver.ipynb#scrollTo=1PlqQM9aZEkd

where u_0 is a function specifically designed to satisfy the initial and boundary conditions of the PDE. The function f is the part that is to be optimized by the algorithm. It depends on x and the parameters θ through the neural network N, and should return 0 for the initial and boundary conditions. Constructing the function f should beneficially incorporate prior knowledge about how the solution behaves, because this will accelerate the learning process.

We then define the residual $r_\theta(t, x)$, which is the output of the neural network inserted back into the PDE (2.4)

$$r_\theta(x, t) \equiv \frac{\partial u_\theta}{\partial t} - \frac{\partial^2 u_\theta}{\partial x^2}. \tag{3.2}$$

Ideally this should be zero, impying a perfect reproduction of the actual solution. The residual becomes our main object for training, as it gives a measure of how good the neural network approximates the actual solution. For the neural network to improve its predictions a loss function is required, which is to minimized. For this we use the mean sum of squared residuals, including the initial and boundary points, across all data samples n.

$$L(x, t, \theta) = \frac{1}{n}\sum_{i=1}^{n}[r_\theta(x_i, t_i)]^2 + \frac{1}{n_B}\sum_{i=1}^{n_B}(u_\theta(x_b, t_i) - u(x_b, t_i))^2 + \frac{1}{n_I}\sum_{i=1}^{n_I}(u_\theta(x_i, 0) - u(x_i, 0))^2,$$

where n_B and n_I is the number of training points on the boundary and the initial time step, respectively. Boundary points are given by x_b. Optimization is then done through gradient descent, minimizing the loss with respect to the parameters θ

$$\theta \leftarrow \theta - \eta \nabla_\theta L(x, t, \theta),$$

where η is the learning rate, which can either be constant or variable, depending on the optimization strategy. The loss function converges towards zero for the optimal parameters θ, in which case the gradient of the loss function vanishes. As a result, the parameters will hardly update anymore through gradient descent, indicating that we have reached an optimal solution.

4 Method

Unit testing

Before the numerical explicit scheme is applied to a particular problem, it is important to test that the discretized equations return expected results. Unit tests are constructed

to test the implementation. This is done by manually calculating the solution of the first two time steps given the initial condition, and comparing the result with that obtained by the numerical scheme. Since the same recursive formula is used for all time steps, it is sufficient to test the two first time steps. To avoid too much computations we choose five equally sized intervals between $x = 0$ and $x = L = 1$, that is $\Delta x = 0.2$. The time step is set to $\Delta t = 0.01$, ensuring stability. For the first time step $j = 1$, that is $t = \Delta t$, we have the following two boundary values and four interior points

$$
\begin{aligned}
u_0^1 &= u_5^1 = 0 \\
u_i^1 &= \frac{\Delta t}{\Delta x^2} u_{i+1}^0 + (1 - 2\frac{\Delta t}{\Delta x^2}) u_i^0 + \frac{\Delta t}{\Delta x^2} u_{i-1}^0 \\
&= 0.05 u_{i+1}^0 + 0.9 u_i^0 + 0.05 u_{i-1}^0
\end{aligned}
$$

Using the initial condition $u_*^0 = \Big(\sin(0), \sin(0.2\pi), \sin(0.4\pi), \sin(0.6\pi), \sin(0.8\pi), \sin(\pi)\Big)$, we get

$$
\begin{aligned}
u_0^1 &= 0 \\
u_1^1 &= 0.531656755 \\
u_2^1 &= 0.8602387 \\
u_3^1 &= 0.8602387 \\
u_4^1 &= 0.531656755 \\
u_5^1 &= 0
\end{aligned}
$$

The forward euler scheme prints out the following values for the first time step

$$(0.,\ 0.53165676,\ 0.8602387,\ 0.8602387,\ 0.53165676,\ 0.)$$

Hence, the result from forward euler is equal to the manual calulations up to machine precision.

Using the values from the first time step, we can calculate the solution at second time step. The recursive formula makes the manual calculations straightforward.

$$
\begin{aligned}
u_0^2 &= 0 \\
u_1^2 &= 0.480888053 \\
u_2^2 &= 0.778093215 \\
u_3^2 &= 0.778093215 \\
u_4^2 &= 0.480888053 \\
u_5^2 &= 0
\end{aligned}
$$

The forward euler scheme gives the following result for the second time step

$$(0.,\ 0.48088805,\ 0.77809321,\ 0.77809321,\ 0.48088805,\ 0.)$$

Once again, the two approaches provide the same result up to machine precision. Thus, the internal functionality of forward euler is validated. It remains to verify the accuracy of the scheme by comparing it with the analytical solution of the diffusion equation 2.4 with provided initial and boundary conditions.

Burde man heller skrive en mer kompakt metode for oppdatering, og deretter ha resultatene i tabell? Man kan ha Ãľn rad for manuell og Ãľn for numerisk, med i-verdier pÃě kolonnene. Da blir det mye mer kompakt. Burde det forresten vÃęre i resultater?

Neural network setup

Sigurd: Skriv hvordan du setter opp neural network her. Skriv ogsÃě hva du plotter (MSE) og hvilke ting ved NN du skal teste, og hvilke du ikke skal teste. Da kan vi bruke seksjonen under til Ãě forklare hvordan vi studerer resultater for bÃěde Euler og NN, og hvordan vi skal sammenlikne

Accuracy assessment

The unit test shows that the forward euler scheme works as expected, but to what accuracy can it reproduce the analytical solution? To address this we compute the deviation of the numerical solution to the analytical solution for two different mesh resolutions in space, $\Delta x = 0.1$ and $\Delta x = 0.01$. For comparison purposes the time step is adjusted in each case to ensure stability, and is defined as $\Delta t = 0.5 \cdot \Delta x^2$. The applied error measure at a given time step is simply the maximum absolute difference between the numerical solution u and the analytical solution u_e across the entire spatial domain. Skal det ikke vÃęre max i ligningen under?

$$e^n = \underset{i \in [0, N_x]}{\operatorname{argmax}} |u_i^n - u_{e,i}^n|$$

The total accumulated error is given by

$$E = \sum_{n=1}^{N} e^n,$$

which is the error measure to be used when testing how well the numerical solution approximates the analytical solution. Er det likningen over som er brukt for resultatene i tabell 1? - Ja, men mulig vi fjerner den siden tabell 1 maa skrives om (verdiene skal heller vere MSE)

We will also study how the error evolves over time. To do this we plot the largest absolute value of the error at 50 different times Burde vi ikke heller bruke summen av feilen fremfor max her? - Jo det burde vi. Vi holder oss konsistent til MSE.. We expect the error to be zero at $t = 0$, which decreases up to a certain point. Since the diffusion equation approaches $u(x, t) = 0$ as t increases, we expect the error to decrease towards the end. We choose $\Delta x = 0.01$ for this analysis, with Δt calculated with the stability criterion.

As a final test, we would like to study the difference between the analytical and numerical solution at two specific times. We choose the times $t_1 = 0.1$ and $t_2 = 0.5$, where the solution $u(x, t_1)$ behaves relatively smooth while still being significantly curved. For t_2 we get $u(x, t_2) < 0.01$, i.e. a solution that is almost linear. These two time periods will serve as the foundation for comparing the performance of our explicit numerical scheme to that of a neural network. During this initial analysis, we will plot the mean squared error

$$\frac{1}{N} \sum_{i=0}^{N-1} (u_i^n - u_{e,i}^n)^2$$

at t_1 and t_2 using both $\Delta x = 0.1$ and $\Delta x = 0.01$, where Δt is calculated in the same way as before. This allows us to check whether the relative difference for the two mesh resolutions are identical, which we expect due to the fixed value of α. We expect the difference between the numerical and analytical solutions at the two times to be negative for the explicit numerical scheme, due to discrete time steps advancing our model too much at each iteration. We repeat this analysis for the neural network as well and compare the two models.

For the neural network, the total error is given by the sum of squared residuals (3.2). This accumulates the error in the approximated solution for each time step, just as is done for the error in forward euler. As a result, one can make a feasible comparison of the errors.

Hele seksjonen over kan skrives litt om nÃěr vi har fÃětt pÃě plass alt av resultater vi skal ha med etc.

Eigenvalues

TBD

	$\Delta x = 0.1$		$\Delta x = 0.01$	
	FE	**NN**	**FE**	**NN**
$t_1 = 0.1$	$1.72 \cdot 10^{-5}$		$1.81 \cdot 10^{-9}$	
$t_2 = 0.5$	$1.50 \cdot 10^{-7}$		$1.69 \cdot 10^{-11}$	

Table 1. MSE as function of spatial step Δx for two different time levels, for forward euler scheme and neural network. Forward euler is abbreviated as FE and neural network as NN.

5 Results

Error of Forward Euler scheme

To assess the accuracy of our implemented numerical scheme, we need to test the how well the produced solution fits the analytical solution. In this project we use the mean squared error to compare the approximated solutions with the analytical solution, both for forward euler and the neural network. Figure 1 shows the MSE as function of time for the forward euler scheme. The time step is chosen to give a stability factor $\alpha = 0.5$. The error increases significantly for early time steps, reaches a peak at around $t = 0.1$ and quickly reduces thereafter. Already at $t = 0.5$ the MSE is converging towards zero. Include analysis of Figure 1 in beginning of discussion.

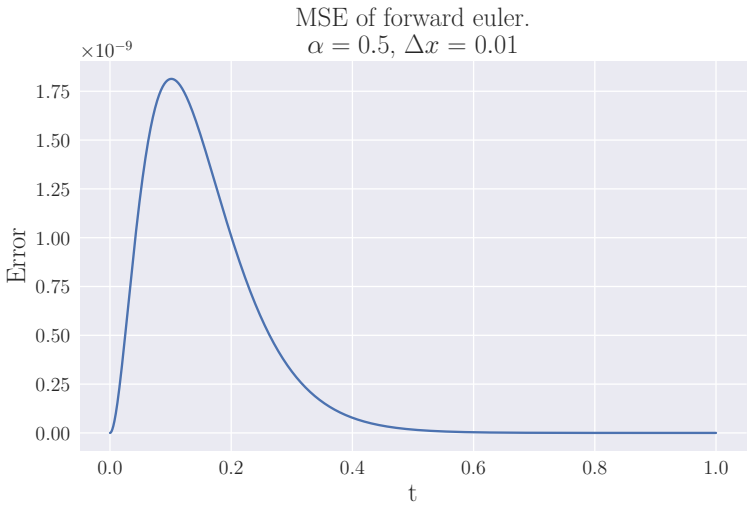

Figure 1. Mean squared error of approximated solution by forward euler, using $\Delta x = 0.01$ and time step Δt dictated by the stability criterion 2.13.

Figure **??** shows the MSE from the neural network with the architecture that provided best results. Sigurd: Legg til MSE plottet her for det beste nettverket.

Table 1 compares the MSE obtained for forward euler and neural network at two selected time steps t and two different spatial steps Δx.

Error of Neural Network

Figure **??** shows the error of the neural network when approximating the solution u_θ to the actual solution u as a function of training iterations. Initially, the error decreases extremely

fast - it reduces by a factor of ten for just a couple of iterations, and the approximated solution improves significantly. Afterwards, the error decreases by a much slower rate with some minor fluctuations. Some time after 100 iterations the error is subject to a large jump, but quickly diminishes agains. For later iterations the error slowly converges towards zero, possessing some irregular, minor fluctuations.

PUT THIS IN DISUSSION?: The reason that the error flucuates instead of decreasing monotonically is that we use the Adam optimizer [https://arxiv.org/abs/1412.6980]. This is a type of stochastic optimization method, providing more favorable features than traditional stochastic methods. The stochasticity allows the neural network to explore more of the convex parameter space by trying out new, potentially worse solutions. The hope is that, after escaping the local optima, that we find an even better solution at some later stage. The apparent jump in Figure **??** is an example of escaping a local optima at the expense of further exploration. In this case, it does not seem like the new solutions lead to better performance than the previous optima. It eventually converges to zero, which is something the solutions prior to the jump did as well.

Comparison of error

IS THIS THE RIGHT WAY TO COMPARE ERROR ??

6 Discussion

Performance of Forward Euler

Figure 1 verifies that simulating the forward euler scheme of the diffusion equation with spatial step $\Delta x = 0.01$ yields an excellent correspondance between the numerical and analytical solution. The maximum error is obtained at an early time step, around $\Delta t = 0.1$. For time levels beyond this, the MSE decreases exponentially and eventually converges towards zero. Notice that the error is exactly zero initially, because the numerical scheme has enforced the solution at the initial time step to satisfy the initial condition from the analytical solution.

The shape of the plot is in good correspondance with what was initially expected, as elaborated on in Methods. The large initial increase of the MSE is a result of the curved initial solution. The spatial gradients are significant and the approximation with finite differences becomes less accurate. In the beginning of the diffusion process, for low t, the solution changes rather quickly. The forward euler scheme is first order accurate in time, $O(\Delta t)$, and second order accurate in space, $O(\Delta x^2)$, implying that the numerical solution is more sensitive to the temporal evolution than the spatial evolution. Hence, the forward euler scheme will produce larger errors for the initial time levels due to the rapid change in solution [2].

After time $t = 0.1$ the MSE decreases. This can be explained by the discretization parameters Δt and Δx. The magnitude of the gradients are diminished because the solution becomes more linear and less curved. As a result, the solution changes more slowly, allowing the finite difference scheme to approximate the changes more accurately both spatially and temporally. Eventually, the exact solution is approximately constant at zero. In this case, the gradients of the solution are indiscernible and the numerical scheme is able to approximate the solution with machine precision. Figure 1 illustrates this by the MSE converging towards zero sufficiently deep into the diffusion process.

Performance of neural network

Potential for solving differential equations

The forward euler method and neural networks are two quite different methods for solving differential equations. The former relies on discretization of the derivative to arrive at an explicit recursive set of equations to solve for each time step. The latter uses the residual error from the approximation as a basis for backpropagation to update weights and biases to improve the approximation to the actual solution. Figure **??** visualizes how bad the approximated solution of the initial run of the neural network is. Just after a few iterations though, the error decreases tremendously and goes below 0.083 (which is the total error obtained from forward euler) after around 100 iterations. After this, the total error fluctuates mildly but eventually converges towards zero.

The forward euler method clearly has an advantage in its simple recursive formula and fast computation time. The proposed solution is calculated within a few milliseconds, with a total error of only 0.083. On the other hand, it is limited by a stability requirement in order to produce realistic results. This effectively puts a restriction on applicable mesh resolutions.

A notable drawback of the neural network is the demanding training process. To compute an approximated solution of 2.4 requires calculating gradients and accumulating chained derivatives backwards through the hidden layers. This is a significantly larger computational effort than calculating the simple recursive formula of the forward euler scheme. As a result, the neural network suffers from a large computational runtime. Still, if training sufficiently long, the neural network will produce a better solution than forward euler and eventually yield a near perfect approximation to the exact solution.

The forward euler scheme has proven to obtain good accuracy on the numerical solution. However, forward euler has an inherent disadvantage in that it is conditionally unstable with stability criteria given by 2.13. In general it means that the forward euler scheme is not a stable method for solving differential equations as it requires that we carefully assign the mesh discretization steps. Neural networks are not limited to any kind of discretization. Neural networks benefit from good convergence properties. That is, given enough time to train it is able to approximate the actual solution of a differential equation with an error of

approximately zero. Moreover, the neural network model 3.1 has the additional flexibility in that the function f can be tweaked to give a better initial guess of the solution.

Overall, the method of choice for solving differential equations is a tradeoff between the accuracy of the approximated solution and the computational runtime. Forward euler is definitely the desired method if we want a quick representation of the solution. If the importance is how precise the representation is, a neural network is a better choice.

Det meste av diskusjonen over kan beholdes da det gjelder generelt, men de faktiske tallene nevnt maa oppdateres med resultatene fra MSE.

7 Conclusion

References

[1] Morten Hjorth-Jensen. Computational physics, lecture notes fall 2015. *Department of Physics, University of Oslo*, August 2015. https://github.com/CompPhysics/ComputationalPhysics/blob/master/doc/Lectures/lectures2015.pdf.

[2] Svein Linge and Hans Petter Langtangen. *Diffusion Equations*, pages 207–322. Springer International Publishing, Cham, 2017.

[3] Pankaj Mehta, Marin Bukov, Ching-Hao Wang, Alexandre G.R. Day, Clint Richardson, Charles K. Fisher, and David J. Schwab. A high-bias, low-variance introduction to machine learning for physicists. *Physics Reports*, 810:1âĂŞ124, May 2019.

[4] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations, 2017.

[5] HÃěkon Olav Torvik, Vetle Vikenes, and Sigurd SÃŷrlie Rustad. Machine learning: Using regression and neural networks to fit continuous functions and classify data. *University of Oslo*, October 2021.