Project 3 : Revenge of the neural network

Morten T. Berg, Henrik H. Carlsen and Jonas Rønning

September 2021

Abstract

In this project we have used a simple feed forward neural network to solve differential equations. The network have been used to solve the heat equation and for finding eigenvalues. We have explored using different activation functions and learning rates to minimise the error obtained by the solution found with the network. The solution was, for the heat equation compared to the analytic solution and the Euler method, while the eigenvalue solver was compared to library functions. The lowest MSE obtained by the neural networks solution to the heat equation was 0.04, compared to 0.0005 obtained by explicit Euler. The solution found by the neural network was good for times close to the initial condition $t = 0$, but deviated visibly when $t \to 1$. The lowest MSE obtained when finding eigenvalues was 0.008, The eigenvalue solver was however unstable, especially when trying to find the eigenvector with lowest eigenvalue.

1 Introduction

The evolution of many physical systems can be described by differential equations. These equations are often impossible to solve analytically and you have to rely on numerical methods, based on discretization of the domain, to find an approximate solution[1]. Using neural networks have become a more viable option for solving differential equations as the computational power of computers have surged in the last decades. Problems that are able to be mapped as differential equations, such as eigenvalue problems, can also be solved by using similar networks.

In this project we are solving a PDE, the one dimensional heat equation, with the use of a neural network. The solution obtained by the network is then compared to the analytical solution and the numerical Euler method. Our network is also applied on the problem of trying to find the largest and smallest eigenvalue of a 6×6 matrix.

In section 2 the theoretical background on ordinary differential equations, partial differential equations, the Forward Euler scheme and use of a neural network to solve the heat equation and the eigenvalue problem is laid forth. We

present the results in section 3 followed by a discussion of the results and our method in section 4. A conclusion is given in section 5. Feedback on the task and course as a whole can be found in section 6.

2 Theory

Here we present the theory that is used in this project. We will not discuss topics that is covered in the previous reports, like for example Neural Networks and Back Propagation [2, 3] .

2.1 Ordinary differential equations

In physics many systems can be described by differential equations. This are equations that relate a function to its derivatives. If the function is a function of only one variable the equation is an ordinary differential equation (ODE) [4]. A typical ODE takes the form

$$\dot{x} = f(x, t), \tag{1}$$

where the dot represent a time derivative and x is the unknown function. In addition to the equation initial conditions are needed to solve the ODE uniquely [5]. Not all ODE's have an unique solution and we are therefore interested in conditions that guarantees that a unique solution exists. To do this we need two definitions; First we need to know if the function is Lipschits. A function $f : X \to Y$, where X and Y are metric spaces with the respective distance functions ρ_X and ρ_Y, is Lipscuts if for all $x_1, x_2 \in X$ there exists a constant K so that [4]:

$$\rho_Y(f(x_1), f(x_2)) \leq K\rho_X(x_1, x_2). \tag{2}$$

The smallest K is the Lipschitz constant. Restricting our self to the spaces $X = Y = \mathbb{R}$ with the standard Euclidean metric the condition reads:

$$|f(x_1) - f(x_2)| \leq K|x_1 - x_2|. \tag{3}$$

So the condition is more strict than saying that f has to be continuous. The second thing we need is the definition of a ball. A ball of radius r centered in x_0 is defined as the closed set:

$$B_r(x_0) = \{x \in \mathbb{R}^n : |x - x_0| \leq r\}. \tag{4}$$

An initial value problem of the form:

$$\dot{x} = f(x), \text{ with } x(t_0) = x_0, \tag{5}$$

has an unique solution if it satisfies the Picard-Lindelof theorm:
For an $x_0 \in \mathbb{R}$, there exists $a, b > 0$ s.t $f : B_b(x_0) \to \mathbb{R}$ is Lipschitz. Then the initial value problem has an unique solution $x(t)$ for $t \in [t_0 - a, t_0 + a]$ if $a = b/M$ with $M = \max_{x \in B_b(x_0)} |f(x)|$ [4].

This does not mean that an analytical solution exists. There are few initial value problems that are analytically solvable and we are therefore often required to use numerical methods for finding the solution. This could be, for example, the exponential time differencing method or just a standard Runge-Kutta scheme.

2.2 Partial differential equations

The partial differential equation(PDE) is the generalisation of the ODE to functions with more than one variable [5]. PDE's are encountered everywhere in physics. Here we will restrict our attention to the one-dimensional heat equation:

$$\partial_x^2 u = \partial_t u, \tag{6}$$

where ∂_β is the partial derivative with respect to the variable β, in our case time, t, and position, x. This equation describes a function that is evened out due to a flattening of local maxima ($\partial_x^2 u < 0$) and minima ($\partial_x^2 u > 0$). If we want to solve this equation we need in addition to the boundary conditions(BC's), an initial condition. Given these we can solve the equation analytically with ease (some times). Suppose now that we want to model the heat dissipating in a rod of length $L = 1$ given the initial conditions $u(x, 0) = sin(\pi x)$ and the boundary conditions (BCs) $u(0, t) = u(L, t) = 0$ for all $t \leq 0$ (i.e the ends of the rod have fixed temperature that we have set to zero). There are many ways to solve this, and the easiest is by inspection. Due to experience with such equations we see that the solution has to be

$$u(x, t) = \sin(\pi x) e^{-\pi^2 t}, \tag{7}$$

which we can check that satisfies the equation as well as the BC's and initial condition. The same analytical solution can also be found in the lecture notes [6].

2.3 The forward Euler scheme

As mentioned not all PDE's are analytically solvable and we often have to rely on numerical methods. One common method for solving differential equations is the Euler method. We start out by discretising the derivative as follows:

$$\partial_t u \approx \frac{u(x, t + \Delta t) - u(x, t)}{\Delta t}, \tag{8}$$

$$\partial_x^2 u \approx \frac{u(x + \Delta x, t) - 2u(x, t) + u(x - \Delta x, t)}{\Delta x^2}. \tag{9}$$

Where the time and space coordinates are discretised on a grid with spacing Δt and Δx. The function is now evaluated at grid points $\{x_i, t_j\}$ and we introduce the notation $u(x_i, t_j) = u_{i,j}$. In this notation the derivatives reads

$$\partial_t u_{i,j} = \frac{u_{i,j+1} - u_{i,j}}{\Delta t} \ , \tag{10}$$

$$\partial_x^2 u_{i,j} = \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{\Delta x^2} \ . \tag{11}$$

Inserting this into eq. (6) we find that providing the temperature at time $t = t_i$ we can find the temperature the next timestep as

$$u_{i+1,j} = u_{i,j} + \Delta t \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{\Delta x^2} \ . \tag{12}$$

This can be written in matrix form as

$$u_{i+1} = (I + A)u_i, \tag{13}$$

with the matrix A

$$A = \frac{\Delta t}{\Delta x^2} \begin{pmatrix} 0 & 0 & 0 & ... & 0 & 0 & 0 \\ 1 & -2 & 1 & ... & 0 & 0 & 0 \\ 0 & 1 & -2 & 1 & 0 & ... & 0 \\ ... & & & & & & \\ 0 & ... & 0 & 1 & -2 & 1 & 0 \\ 0 & ... & 0 & 0 & 1 & -2 & 1 \\ 0 & .. & 0 & 0 & 0 & 0 & 0 \end{pmatrix} . \tag{14}$$

The first and last row is set assuming that u_0 and u_N is fixed by the BC's. When implementing the scheme it is required that $\delta t/\delta x^2 \leq 0.5$ in order for the solution to be stable.

2.4 Solving the heat equation using a neural network

We now want to solve a PDE using a neural network. A general PDE is of the form [6]:

$$f\left(x_1, \ldots, x_N, \frac{\partial g(x_1, \ldots, x_N)}{\partial x_1}, \ldots, \frac{\partial g(x_1, \ldots, x_N)}{\partial x_N}, \frac{\partial g(x_1, \ldots, x_N)}{\partial x_1 \partial x_2}, \ldots, \frac{\partial^n g(x_1, \ldots, x_N)}{\partial x_N^n}\right) = 0 \ . \tag{17}$$

We approximate the function g by a trial function given as:

$$g_t(\mathbf{x}, P) = f_0(\mathbf{x}) + f_1(\mathbf{x}, N(\mathbf{x}, P)), \tag{15}$$

where $N(\mathbf{x}, P)$ is the output from a neural network, f_1 is a function that vanish at the boundary, including the initial time, and f_0 is a function that satisfies the boundary and initial condition. P is the weights and biases of the neural network. We now need to translate this problem to a minimisation problem that we can solve by minimising a cost function. It is clear that a function that solves eq. (17) is a minimum of the folowing costfunction [6]

$$C\left(X, P\right) = \sum_{i=1}^{M} f\left(\left(\boldsymbol{x}_i, \frac{\partial g_t(\boldsymbol{x}_i, P)}{\partial x_1}, \ldots, \frac{\partial g_t(\boldsymbol{x}_i, P)}{\partial x_N}, \frac{\partial g_t(\boldsymbol{x}_i, P)}{\partial x_1 \partial x_2}, \ldots, \frac{\partial^n g_t(\boldsymbol{x}_i, P)}{\partial x_N^n}\right)\right)^2 \ , \tag{16}$$

and the problem is now to find the parameters P that minimises this. For the heat equation the cost function becomes:

$$C\left(X,P\right)=\left(\partial_t g_t-\partial_x^2 g_t\right)^2. \tag{17}$$

With the trial function given as [6]

$$g_t(x,t)=\sin(\pi x)+x(1-x)tN(x,P)\ , \tag{18}$$

to impose the bc's and initial conditions. Methods for minimising the cost function was discussed in the previous report as well as the back propagation algorithm. In this project we have let standard packages deal with the derivatives.

2.5 Eigenvectors with neural networks

The eigenvectors, x, and eigenvalues, λ, of a matrix A is defined by the equation:

$$Ax=\lambda x. \tag{19}$$

The eigenvalue problem can be mapped to a differential equation [7]

$$\dot{x}(t)=-x(t)+f(x(t)), \tag{20}$$

with $f(x)=\left[x^TxA+(1-x^TAx)I\right]x$. One can easily show that the stationary points of the above equation corresponds to the eigenvectors of A for symmetric matrices. It was shown in [7] that if one starts with a random initial condition that is not orthogonal to the eigenspace of the largest eigenvalue then eq. 20 will converge to the eigenvector with the largest eigenvalue in the limit $t\to\infty$. Replacing A with $-A$ the equation will converge to the eigenvector with the smallest eigenvalue. Since it is unlikely that a random vector is orthogonal to the eigenspace we can just take a random vector. When the eigenvector is found the corresponding eigenvalue is obtained from eq. (19) as

$$\lambda=x^TAx(x^Tx)^{-1}. \tag{21}$$

We use the trial function

$$g_t(t)=x_0+N(t,P), \tag{22}$$

and the cost function

$$c(t,P)=(\partial_t g_t-g_t^Tg_tAg_t+g_t^TAg_tg_t)^2. \tag{23}$$

We assume that the equation converge fast and is close to convergence at $t=1$. In the article [7] they use a recurrent neural network, but we find that our standard solver is able to find eigenvectors. Note that this choice of trial function means that the initial value is $x_0+N(0,P)$ and not x_0.

3 Results

3.1 Analytical and numerical solution of diffusion equation.

The diffusion equation was solved by analytical and numerical methods for different intervals of x and t. Figure 1 shows the solution for a rough partition of x. Figure 2 shows the solution of the equation for numerical and analytical methods with a fine partition of x. Figure 3 shows the rough partition solution as a colour-map with the time on the x axis and space on the y axis. Table 1 also shows us the mean square error between the numerical and the analytical solutions.

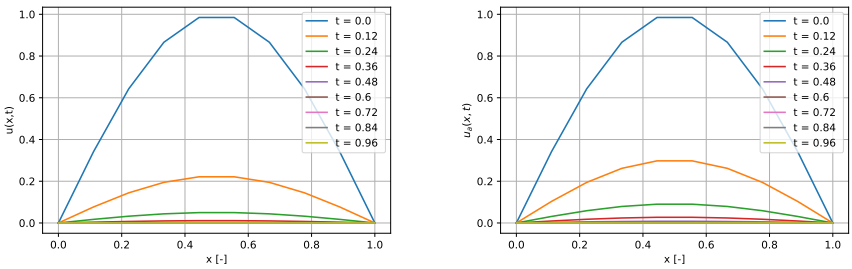

Figure 1: Left figure shows the evolution of the numerical solution to the diffusion equation as a 2D function $u(x)$. Right figure shows the same however here with the analytical solution. We have plotted over a range of different times. Either the blue or orange graphs can be viewed as t_1 and the green as t_2 mentioned in task b. Further for these figures $\Delta x = 1/10$.

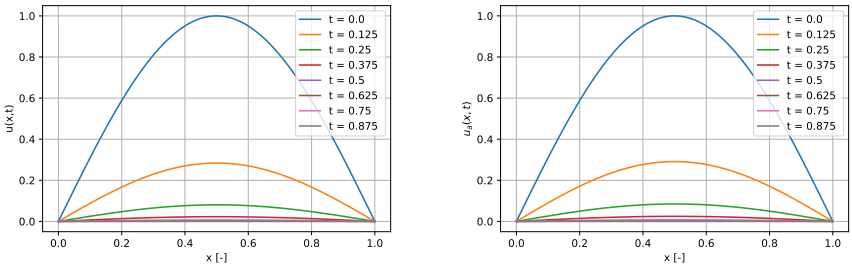

Figure 2: Left figure shows the evolution of the numerical solution to the diffusion equation as a 2D function $u(x)$. Right figure shows the same however here with the analytical solution. We have plotted over a range of different times. Either the blue or orange graphs can be viewed as t_1 and the green as t_2 mentioned in task b. Further for these figures $\Delta x = 1/100$.

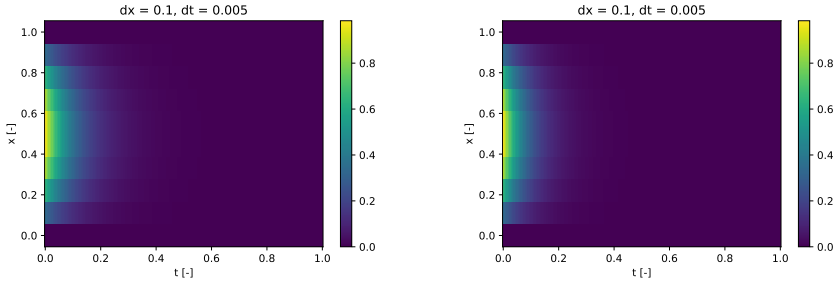

Figure 3: In this figure we can see the two solutions of the diffusion equation as a heat-map with the x axis being the time and the y axis representing the space part. The solutions in this figure was run with $\Delta x = 1/10$. The left figure is the analytical solution and the right figure is the numerical solution.

Δx	MSE
$1/10$	5.0×10^{-4}
$1/100$	5.1×10^{-6}

Table 1: The MSE between the analytical and the numerical solutions of the diffusion equation for different Δx intervals.

3.2 Homemade neural network

Figure 4 shows the MSE obtained by the neural network for different values of the activation function and learning rate. The MSE is here calculated by comparing the trial-function against the analytical solution on a 50×50 grid. The evolution of the MSE for a couple of learning cycles are shown in figure 5 together with a cross section of the obtained solution.

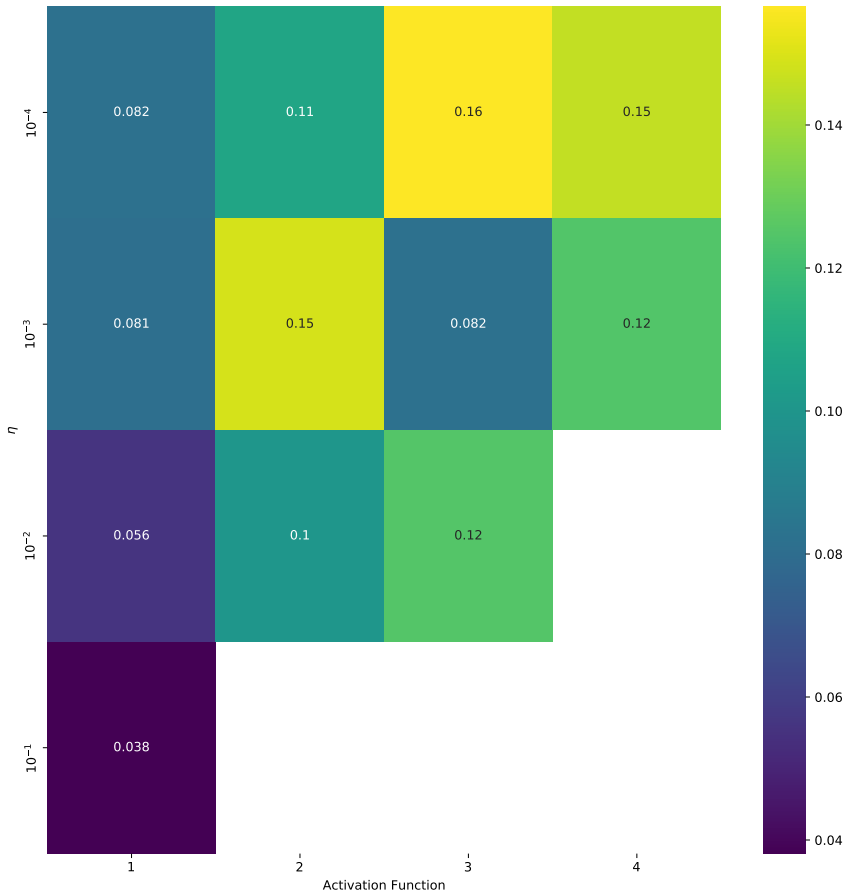

Figure 4: The MSE obtained by a neural network for different learningrates and activation functions. The activation functions are from 1 the Sigmoid, RELU, leaky-RELU and the identity. The white squares have diverged.

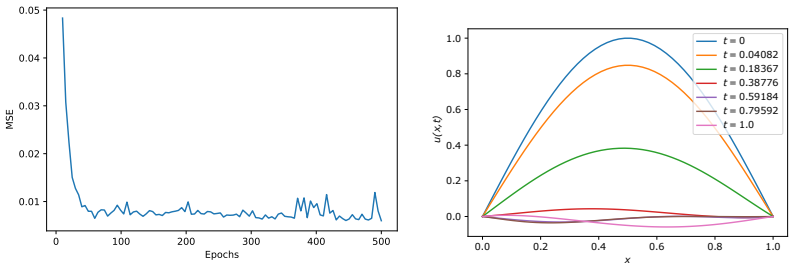

Figure 5: Left figure shows the MSE as a function of epochs. Right is the cross section of the solution obtained by the NN.

3.3 Eigenvalue problem

Figure 6 shows the MSE obtained when calculating the largest eigenvalue of a random symmetric matrix. A comparison between the value obtained by the network and the one obtained by scipy is shown in table 2 and 3. Figure 7 shows how the network converge to an eigenvector after a number of iterations.

$\lambda_{max,nn}$	λ_{max}	$\lvert\lambda_{max} - \lambda_{max,NN}\rvert$
5.000	5.025	0.025
4.568	4.574	0.006
4.980	5.017	0.037
4.760	5.271	0.511
5.493	5.499	0.006
3.732	3.742	0.010
4.411	4.662	0.250
4.474	4.488	0.015
4.976	5.016	0.040
2.770	4.511	1.741
4.169	4.176	0.006

Table 2: The biggest eigenvalue we found vs the true one for the parameters that minimalised the MSE in fig. 6. The value is found after 50 epochs.

$\lambda_{min,nn}$	λ_{min}	$\lvert\lambda_{min} - \lambda_{min,NN}\rvert$
0.420	0.340	0.08
1.338	0.093	1.245
0.016	0.012	0.004
1.278	$8.10 \cdot 10^{-5}$	1.278
2.095	0.311	1.784
4.442	0.123	4.319
1.184	0.003	1.182
3.244	0.060	3.184
2.328	0.021	2.308
0.058	0.037	0.021
1.337	$1.03 \cdot 10^{-4}$	1.337

Table 3: The smallest eigenvalue we found vs the true one for the parameters that minimalised the MSE in fig. 6. The value is found after 50 epochs.

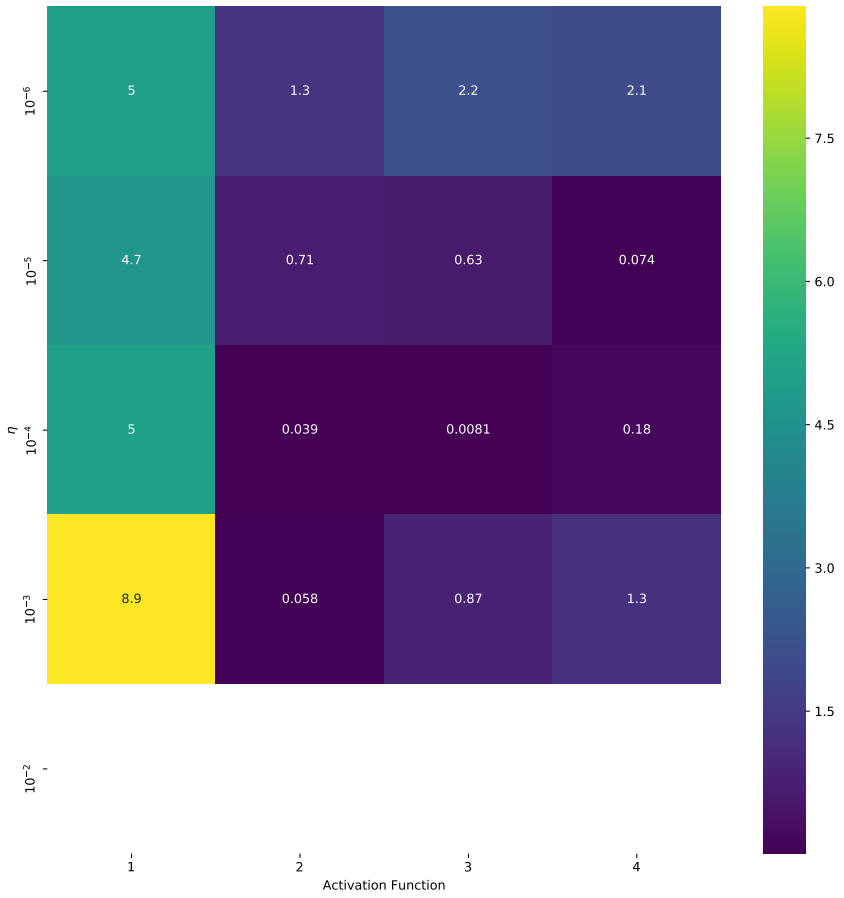

Figure 6: MSE error between the biggest eigenvalue obtained by scipy's standard function and our neural network for different learning rates and activation functions. From 1 Sigmoid, RELU, leaky-RELU and the identity. The functions are run for 100 epochs.

4 Discussion

Both the numerical Euler method and the analytical method presented similar results as we can see when comparing the solutions obtained on a low resolution grid with the solutions from the high resolution grid in figures 1 and 2. The MSE given in table 1 shows that for a higher resolution of x the MSE becomes smaller. Moreover from what we can see from the 2 dimensional representation in figure 1 and 2, the function $u(x,t)$ approaches zero quite fast.

The colormaps, while not adding any new information, show another way of

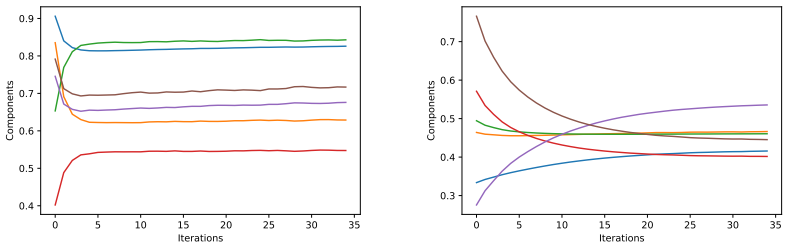

Figure 7: The components of the eigenvector as a function of iterations for eigenvectors that corresponds to the largest (left) and the lowest (right) eigenvalue.

representing the data. We can, as before, from the figures see that the numerical and the analytical solutions are very similar for the rough partition, shown in figure 3, and the fine partition of x was not included given that including the figures made the report lag, however they are very similar for both the numerical and analytical. Again it is apparent that the function approaches zero as t becomes large.

The MSE value obtained by the neural network in figure 4 shows that the best fit was given by the Sigmoid function for a learningrate of 0.1. For this optimal set-up we plot the MSE as a function of epochs and the obtained solution in figure 5. Regarding the solution given by the feed forward neural network shown in the figure, it is more accurate at early times, while at times close to $t = 1$ the solution deviates visibly from the analytical one and extra oscillations appears. The neural network actually has the solution to the diffusion equation go under 0. So while the neural network follows the same patterns as the numerical and analytical solutions in the start it seems to be incapable of finding a good solution as the equation nears it stationary state for high values of t. That the solution is good at early times is not surprising given the trial function. Note that the smallest MSE obtained by the neural network is larger, by a factor 100, to what was obtained by the Euler method for the lowest grid resolution. It is also much slower, but we have not optimized our solvers and we have only considered a network with one hidden layer. That our simple network is able to solve the differential equation to an acceptable accuracy shows that there is potential to solve PDE's with a neural network. We believe that the greatest potential for neural networks lie in solving non-linear PDEs which can be hard to solve with conventional methods.

From figure 6 it seems like leaky-RELU is the activation function that is best at finding the maximal eigenvalues. Note that the MSE is high because the network fails to find the eigenvector with the largest eigenvalue for some initial vectors, and instead converges to another eigenvector. From the cost- and trial function given in eq. (22,23) we see that this is not surprising. This

cost function have a minimum when $N(t, P)$ is constant in time and g_t being any of the eigenvectors. This means that the costfunction we are trying to minimise have multiple global minima. Despite this the function is usually very good at finding the maximal eigenvalues as can be seen in table 2. The networks ability to find the smallest eigenvalue is however disappointing as illustrated in table 3 where the network only found it 3 out of 10 times. The problem is not that the network is unable to find the minimal eigenvalue, but that it is more likely to find another eigenvalue. Because of this instability we tried initialising the matrices with different distributions and found that using a normal distribution gives the highest probability of finding the minimal eigenvalue. If however the uniform distribution was utilized to set up the problem, then the network failed completely. This is why there is no plot of the MSE for this case because more often than not the network converges to a different eigenvalue than the minimal one and we don't want to scare the grader by such high numbers. Leaky-RELU was the function that most reliably found the minimal eigenvalue. The other activation functions was poor at doing this, and we was unable to make a network with RELU or Sigmoid find it at all. A way of (almost) guaranteeing that the network converges to the smallest eigenvalue is to start the guess with the eigenvector plus some noise, but that would be cheating. Because of this we believe that our network would be able to find the smallest eigenvalue given a perfect guess of initial vector and optimal parameters. There might be some other test functions that are better at finding the eigenvalues we want, but we have not found them.

5 Conclusion

In conclusion we found that the feed forward neural networks works somewhat well for solving differential equations, yet is still worse than the analytical solution and the numerical. We also saw how a neural network could be utilised to find eigenvalues of a symmetric matrix. The error obtained by the network was higher than what was obtained by a conventional solver and it deviated considerably as $t \to 1$, but it did not diverge. For the eigenvalue problem the network seems very sensible to the parameters; learning rate, epochs and time discretisation. Our network works for finding the largest eigenvalue of a matrix, but fails to find the smallest eigenvalue unless we have a very good guess of the initial vector. Thus methods tailored to solve specific PDEs seem more fitting in order to solve the eigenvalue problem.

6 Feedback

6.1 On project

This was a great project 😎
God Jul 🎅

6.2 On the course in general

The course is fine
like red wine 🍷
if too much is changed
students will feel estranged.

The feedback was slow
but we give to you a deep bow 🙇
for all the help, lectures and group sessions.

References

[1] Tim Dockhorn. A discussion on solving partial differential equations using neural networks. *arXiv preprint arXiv:1904.07200*, 2019.

[2] Jonas Rønning Morten T. Berg, Henrik H. Carlsen. Project 1. https://github.com/MortenTryti/MachineLearning/blob/main/Project%201/Report.pdf, 2021.

[3] Jonas Rønning Morten T. Berg, Henrik H. Carlsen. Project 2. https://github.com/MortenTryti/MachineLearning/blob/main/Project2021.

[4] James D. Meiss. *Differential Dynamical Systems*. Society for Industrial and Applied Mathematics, 2017.

[5] Neha Yadav, Anupam Yadav, Manoj Kumar, et al. *An introduction to neural network methods for differential equations*. Springer, 2015.

[6] Morten Hjorth-Jensen. Week 42: Solving differential equations and convolutional neural networks. https://compphysics.github.io/MachineLearning/doc/pub/week42/html/week42.html.

[7] 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):1155–1164, 2004.