A Machine learning approach to partial differential equations

Hans Mathias Mamen Vege

December 18, 2018

Abstract

nan

1 Introduction

There exists several different approaches that can be used for investigating one of the fundamental building blocks of natural sciences, partial differential equations. The PDEs can appear in almost any field of science, and can give us information about a wide range of topics, such as heat dispersion, wave dynamics, quantum mechanical systems[see 4, ch. 10].

We will be investigating two different approaches to solving partial differential equations(PDEs). The first method being Deep Neural Networks, and for the second approach will be looking at finite differences. For the latter, we will be using Forward Euler. We will first and foremost be investigating solutions with Deep Neural Networks, and use Forward Euler as a method for comparison.

We will begin in section 2 to look at the PDE we are investigating and quickly derive its analytical solution. Then we will quickly go through the finite differences method of Forward Euler to solve the same PDE, and finally go through the basics of a Deep Neural Network(DNN). In section 3 we will briefly sketch out the parameters we will run for, as well as point out where to find relevant code. In section 4 we will go through the results for different combinations of hyper parameters used in the DNN, and compare it with both the analytical results and the Forward Euler results. Then, we will proceed with discussing these results in section 5, and finally in section 6 we will summarize our findings and make appropriate conclusions.

2 Theory

2.1 A solution to the heat equation

The partial differential equation we will be investigating is commonly dubbed the heat equation, and has the following shape

$$\frac{\partial^2 u(x,y)}{\partial x^2} = \frac{\partial u(x,t)}{\partial t} \tag{1}$$

given that $t > 0$, $x \in [0, L]$. As a shorthand notation, we can write this system as $u_{xx} = u_t$. The initial conditions are given as,

$$u(x, 0) = \sin(\pi x/L) \tag{2}$$

for $0 < x < L$ and $L = 1$. The boundary conditions are given as,

$$\begin{aligned} u(0, t) &= 0 \\ u(0, L) &= 0 \end{aligned} \tag{3}$$

To derive an analytical solution for our PDE, we begin by assuming that the variables are separable,

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

Inserting this into the PDE (1), we get

$$\frac{\partial^2 \left(X(x)T(t)\right)}{\partial x^2} = \frac{\partial \left(X(x)T(t)\right)}{\partial t}.$$

This becomes,

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

in which we see that each side does not depend on the other, such that they are equal to a constant. Calling this constant $-k^2$, we can solve each of them separately. We begin with the temporal part,

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

which has the solution

$$V(t) = a\mathrm{e}^{-k^2 t}.$$

For the spatial part, we get

$$\frac{1}{X(x)} \frac{\partial^2 X(x)}{\partial x^2} = -k^2 \tag{4}$$

which we can see has a solution given as[2],

$$X(x) = b \sin(kx). \tag{5}$$

We now need to find a, b and k. From the initial conditions requirement(2), we get

$$u(x, t = 0) = b \sin(kx) \cdot 1$$

Requiring $b = 1$ fulfills the initial condition requirement(2). If we then look at the boundary conditions(3),

$$\begin{aligned} u(0, t) &= \sin(k \cdot 0)\mathrm{e}^{-k^2 t} = 0 \\ u(L, t) &= \sin(k \cdot L)\mathrm{e}^{-k^2 t} = 0, \end{aligned}$$

with the last line giving us that,

$$\begin{aligned} kL &= \sin(0) \\ kL &= n\pi \\ k &= \frac{n\pi}{L}. \end{aligned}$$

Summing up, we then have our analytical solution

$$u(x,t) = \sin\left(\frac{n\pi x}{L}\right) \mathrm{e}^{-\left(\frac{n\pi}{L}\right)t} \tag{6}$$

We can further set $n = 1$ and $L = 1$, as that will match the initial conditions in equation (2).

2.2 Forward Euler and finite differences

Using an explicit scheme, namely forward Euler[4, ch. 10.2.2], the basis for this scheme is in how to approximate the derivative. Using the familiar forward formula for the derivative, we can write the right hand side of (1) as,

$$\begin{aligned} \frac{\partial u(x,t)}{\partial t} &\approx \frac{u(x_i, t_j + \Delta t) - u(x_i, t_j)}{\Delta t} \\ u_t &= \frac{u_{i,j+1} - u_{ij}}{\Delta t}. \end{aligned}$$

Rewriting the left hand side of (1) using a second derivative approximation, we get

$$\begin{aligned} \frac{\partial^2 u(x,t)}{\partial x^2} &\approx \frac{u(x_i + \Delta x, t_j) - 2u(x_i, t_j) + u(x_i - \Delta x, t_j)}{\Delta x^2} \\ u_{xx} &= \frac{u_{i+1,j} - 2u_{ij} + u_{i-1,j}}{\Delta x^2}. \end{aligned}$$

Setting these two numerical approximations equal to each other, we can solve for the next iteration in the time direction.

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

with $\alpha = \frac{\Delta t}{\Delta x^2}$. Solving for $u_{i,j+1}$, we get

$$u_{i,j+1} = \alpha u_{i+1,j} + (1 - 2\alpha)u_{ij} + \alpha u_{i-1,j}. \tag{7}$$

With a keen eye we can recognize this as a simple iterative matrix equation,

$$V^{j+1} = (I - \alpha D)V^j, \tag{8}$$

with I simply being the identity matrix and D being a tridiagonal matrix of with -1 along the of-diagonals and 2 at the diagonal. The stability condition of Forward Euler is given as $\alpha = \frac{\Delta t}{\Delta x^2} \leq 1/2$. In other words, $\Delta t \leq 1/2\Delta x^2$. We will simply refer to Hjorth-Jensen [4, ch. 10.2.1] for a derivation of this condition.

2.3 Deep Neural Networks and PDEs

A full derivation of DNNs will not be given here[1], as we will mainly focus on how to apply DNNs to solving PDEs. The derivation will closely follow the one given by Hein [3]. The neural net we will utilize is a *Multilayer Perceptron*. To apply it to solving a PDE, we must begin with defining an expression for our PDE. We write our general PDE as,

$$f\left(x, t, g(x,t), g'(x,t), \dots, g^{(n)}(x,t)\right) = 0. \tag{9}$$

We have f given to use as some function of x, $g(x,t)$ and its derivatives. Without specifying the boundary conditions and initial conditions, this function may have several solutions[2]. The general trial solution(the solution which we will use to approximate $g(x,t)$ with), is given as

$$g_t(x,t) = h_1(x,t) + h_2(x,t,N(x,t,P)). \tag{10}$$

$N(x,t,P)$ is the output from the neural network and P is the weights and biases of the network. As we can see (10) is written as a sum of two parts, $h_1(x,t)$ and $h_2(x,t,N(x,t,P))$, where $h_1(x,t)$ ensures that $g_t(x,t)$ satisfies a set of boundary conditions[6]. The boundary conditions (3) and the initial conditions (2), allows us to make a guess for the $h_1(x,t)$ and $h_2(x,t,N(x,t,P))$.

Having $h_1(x,t)$ as,

$$h_1(x,t) = (1-t)\sin(\pi x/L) \tag{11}$$

which at $t = 0$ gives $h_1(x,0) = \sin(\pi x/L)$ which satisfies (2). And for $x = 0, L$ we get $h_1(x,0) = h_1(x,L) = 0$. We now look towards h_2.

$$h_2(x,t,N(x,t,P)) = x(1-x/L)tN(x,t,P) \tag{12}$$

We see that at $t = 0$ this equation is zeros, and will also remain zero at the boundaries. Thus, the trial solution(10) fulfills our requirements.

All that now remains is a cost function to use in the network. We want the network $N(x,t,P)$ to approximate the left hand side with the right hand side of our PDE (1) as best as possible[3, p. 5], this the minimization of this expression can be used as our cost function. We can write,

$$\min_P \left\{ \frac{1}{N, N_t} \sum_{i=1}^{N} G(x,t,P) \right\} \tag{13}$$

with $G(x,t,P)$ being

$$G(x,t,P) = \frac{\partial^2 u(x,y)}{\partial x^2} - \frac{\partial u(x,t)}{\partial t} \tag{14}$$

The gradient of this cost function will be handled symbolically by TensorFlow[1].

[1] A *more* complete derivation can be found here.

3 Implementation

The code used to run this project can be viewed at GitHub[2]. Further, the code was mainly built using TensorFlow[1].

3.1 Neural Net setup

For the neural net, two different optimizers were used, Gradient Descent and the Adam optimizer[3].

Further, we used compared different kind of activation functions which can be seen in the following table,

- Sigmoid
- tanh
- Relu
- LeakyRelu

These is also summarized in the appendixB.

We also ran for a wide set of different layers, as seen in the following table

Table 1: Number of neurons found in each hidden layer.

Layers	Neurons
1	10
1	20
1	50
1	100
1	1000
2	10
2	20
2	40
2	80
3	10
3	20
3	40
3	100
5	10
10	10

[2]`https://github.com/hmvege/FYSSTK4155-project3`

[3]See the documentation for gradient descent and the Adam algorithm[5] for more details.

4 Results

4.1 Deep Neural Net

We will now proceed with presenting results for a deep neural net using the Gradient Descent optimizer in table 2, put together using different hyper parameters. Similar hyper parameters but with the Adam optimizer can be viewed in the table 3. Combinations where put together from different amount of layers, neurons on each layer, activation functions and optimizers. The results were retrieved running $N_{\text{e}} = 10^5$ epochs. To evaluate the performance we look at the R^2 score(15) and the MSE score(16). The $\varepsilon_{\text{abs}} = |u - \hat{u}|$, where u is the numerical results and $\hat{u}$ is the analytical results.

4.1.1 Testing out optimal hyper parameters

From the table 3 with Adam as optimizer, the results which gave us the best R^2 and MSE were chosen to be run for $N_{\text{e}} = 10^6$ epochs. The results for this run can be seen in table 4.

We will use the hyper parameters given in table 5 as the optimal settings for our neural network. It is worth mentioning that the exact R^2 score this combination provided was $R^2 = 0.9999999985$. The cost at this point was $\mathcal{C} = 6.74 \cdot 10^{-7}$ based on equation (13).

4.1.2 Cost and Error

The cost and error of a run with three layers and 10 neurons in each layers run for 10^6 epochs can be viewed in figure 1. The error is given as $\max\left(|u_{\text{analytic}} - u_t|\right)$, where u_t is the numerical approximation by the neural network. The Adam optimizer were used.

4.2 Comparing DNN and Forward-Euler against the analytic solution

From the table 4, we will use the hyper parameters as given in table 5 in further analysis and comparison against the Forward-Euler finite difference method. The Forward-Euler method was run with $N_x = 10$ and $N_t = 100$, and a $\alpha = \Delta t/\Delta x^2 = 0.005$ in order to safely maintain the stability condition.

In figure 2 we see the analytical solution as given for by (6) for $N_t = 10$ and $N_x = 10$.

In figure 3 we see the results from the Forward-Euler method together with the absolute difference between the analytical solution as given by (6). Comparing with the analytical, we have $R^2 = -0.12$ and $MSE = 9.87 \cdot 10^{-4}$

As an example, an increased spatial resolution can be seen in figure 4. This yielded $R^2 = 0.88$ and $MSE = 1.01 \cdot 10^{-5}$.

In figure 5 we see the resulting surface plot given by the neural net for a optimal set of parameters5.

4.3 Timing

A quick plot of different timing values for both Forward Euler and the neural net can be viewed in figure 6. The DNN was run for 10^5 epochs.

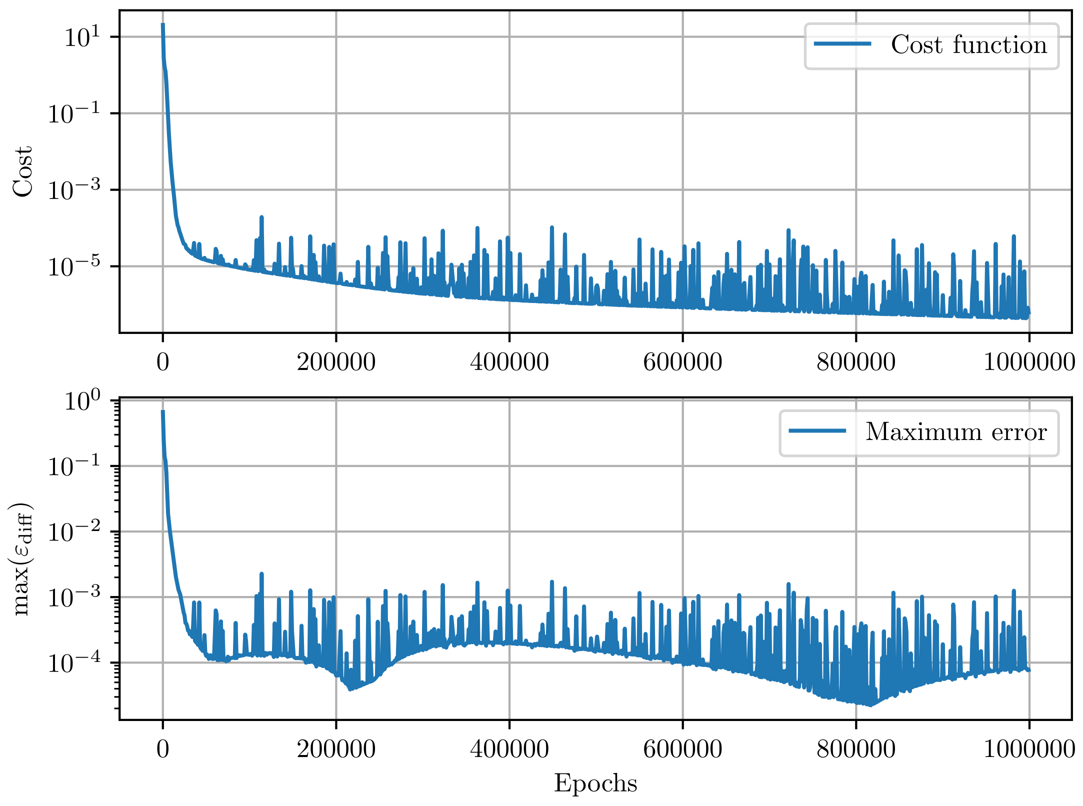


Figure 1: The cost and error as it evolves using hyper parameters given in table 5 and $N_\text{e} = 10^6$ epochs. Adam was used as an optimizer.

Table 2: Results for a DNN with different hyper parameters. The number of epoch was set to $N_\text{e} = 10^5$. Results presented are obtained with *gradient descent* as optimizer.

Activation	Layers	Neurons	$\max(\varepsilon_\text{abs})$	R^2	MSE
Leaky relu	1	100	$9.88 \cdot 10^{-2}$	$9.71 \cdot 10^{-1}$	$1.27 \cdot 10^{-3}$
Leaky relu	1	1,000	$1.33 \cdot 10^{-1}$	$9.49 \cdot 10^{-1}$	$2.23 \cdot 10^{-3}$
Leaky relu	2	80	$1.25 \cdot 10^{-1}$	$9.29 \cdot 10^{-1}$	$3.08 \cdot 10^{-3}$
Leaky relu	3	40	$1.19 \cdot 10^{-1}$	$9.52 \cdot 10^{-1}$	$2.4 \cdot 10^{-3}$
Leaky relu	5	10	$1.06 \cdot 10^{-1}$	$9.65 \cdot 10^{-1}$	$1.76 \cdot 10^{-3}$
Leaky relu	10	10	$1.9 \cdot 10^{-1}$	$8.75 \cdot 10^{-1}$	$6.23 \cdot 10^{-3}$
Relu	1	100	$1.25 \cdot 10^{-1}$	$9.68 \cdot 10^{-1}$	$1.41 \cdot 10^{-3}$
Relu	1	1,000	$1.6 \cdot 10^{-1}$	$9.14 \cdot 10^{-1}$	$3.74 \cdot 10^{-3}$
Relu	2	80	$1.5 \cdot 10^{-1}$	$9.33 \cdot 10^{-1}$	$2.92 \cdot 10^{-3}$
Relu	3	40	$5.33 \cdot 10^{-2}$	$9.9 \cdot 10^{-1}$	$5 \cdot 10^{-4}$
Relu	5	10	$2.18 \cdot 10^{-1}$	$8.59 \cdot 10^{-1}$	$7 \cdot 10^{-3}$
Relu	10	10	$1.08 \cdot 10^{-1}$	$9.53 \cdot 10^{-1}$	$2.36 \cdot 10^{-3}$
Sigmoid	1	10	$1.14 \cdot 10^{-2}$	$1 \cdot 10^{0}$	$1.76 \cdot 10^{-5}$
Sigmoid	1	20	$1.32 \cdot 10^{-2}$	$9.99 \cdot 10^{-1}$	$2.43 \cdot 10^{-5}$
Sigmoid	1	50	$1.21 \cdot 10^{-2}$	$1 \cdot 10^{0}$	$1.78 \cdot 10^{-5}$
Sigmoid	1	100	$1.36 \cdot 10^{-2}$	$1 \cdot 10^{0}$	$2.14 \cdot 10^{-5}$
Sigmoid	1	1,000	$6.25 \cdot 10^{-1}$	$-4.75 \cdot 10^{-1}$	$6.4 \cdot 10^{-2}$
Sigmoid	2	10	$2.66 \cdot 10^{-2}$	$9.99 \cdot 10^{-1}$	$3.36 \cdot 10^{-5}$
Sigmoid	2	20	$2.14 \cdot 10^{-2}$	$9.99 \cdot 10^{-1}$	$2.3 \cdot 10^{-5}$
Sigmoid	2	40	$1.62 \cdot 10^{-2}$	$1 \cdot 10^{0}$	$1.21 \cdot 10^{-5}$
Sigmoid	2	80	$1.31 \cdot 10^{-2}$	$9.99 \cdot 10^{-1}$	$3.58 \cdot 10^{-5}$
Sigmoid	3	10	$6.59 \cdot 10^{-3}$	$1 \cdot 10^{0}$	$2.82 \cdot 10^{-6}$
Sigmoid	3	20	$4.18 \cdot 10^{-3}$	$1 \cdot 10^{0}$	$1.26 \cdot 10^{-6}$
Sigmoid	3	40	$3.07 \cdot 10^{-3}$	$1 \cdot 10^{0}$	$1.18 \cdot 10^{-6}$
Sigmoid	5	10	$2.73 \cdot 10^{-2}$	$9.97 \cdot 10^{-1}$	$1.57 \cdot 10^{-4}$
Sigmoid	10	10	$4.54 \cdot 10^{-1}$	$1.96 \cdot 10^{-1}$	$3.99 \cdot 10^{-2}$
Tanh	1	10	$1.71 \cdot 10^{-2}$	$9.99 \cdot 10^{-1}$	$4.53 \cdot 10^{-5}$
Tanh	1	20	$2.26 \cdot 10^{-2}$	$9.99 \cdot 10^{-1}$	$2.48 \cdot 10^{-5}$
Tanh	1	50	$9.99 \cdot 10^{-3}$	$1 \cdot 10^{0}$	$5.7 \cdot 10^{-6}$
Tanh	1	100	$1.11 \cdot 10^{-2}$	$9.99 \cdot 10^{-1}$	$2.6 \cdot 10^{-5}$
Tanh	1	1,000	$2.13 \cdot 10^{-2}$	$9.98 \cdot 10^{-1}$	$7.97 \cdot 10^{-5}$
Tanh	2	10	$9.26 \cdot 10^{-3}$	$1 \cdot 10^{0}$	$1.52 \cdot 10^{-5}$
Tanh	2	20	$2.91 \cdot 10^{-2}$	$9.99 \cdot 10^{-1}$	$4.15 \cdot 10^{-5}$
Tanh	2	40	$1.86 \cdot 10^{-2}$	$9.99 \cdot 10^{-1}$	$2.49 \cdot 10^{-5}$
Tanh	2	80	$2.23 \cdot 10^{-2}$	$9.99 \cdot 10^{-1}$	$3.14 \cdot 10^{-5}$
Tanh	3	10	$7.38 \cdot 10^{-3}$	$1 \cdot 10^{0}$	$4.38 \cdot 10^{-6}$
Tanh	3	20	$5.05 \cdot 10^{-3}$	$1 \cdot 10^{0}$	$2.49 \cdot 10^{-6}$
Tanh	3	40	$5.1 \cdot 10^{-3}$	$1 \cdot 10^{0}$	$1.53 \cdot 10^{-6}$
Tanh	5	10	$1.34 \cdot 10^{-2}$	$1 \cdot 10^{0}$	$1.27 \cdot 10^{-5}$
Tanh	10	10	$4.86 \cdot 10^{-3}$	$1 \cdot 10^{0}$	$6.5 \cdot 10^{-6}$

Table 3: Results for a DNN with different hyper parameters. The number of epoch was set to $N_{\text{e}} = 10^5$. Results presented are obtained with the *Adam* optimizer[5].

Activation	Layers	Neurons	$\max(\varepsilon_{\text{abs}})$	R^2	MSE
Leaky relu	1	100	$1.42 \cdot 10^{-1}$	$9.56 \cdot 10^{-1}$	$1.9 \cdot 10^{-3}$
Leaky relu	1	1,000	$1.21 \cdot 10^{-1}$	$9.38 \cdot 10^{-1}$	$2.71 \cdot 10^{-3}$
Leaky relu	2	80	$1.6 \cdot 10^{-1}$	$9.2 \cdot 10^{-1}$	$3.49 \cdot 10^{-3}$
Leaky relu	3	40	$2.6 \cdot 10^{-1}$	$8.89 \cdot 10^{-1}$	$5.52 \cdot 10^{-3}$
Leaky relu	5	10	$2.29 \cdot 10^{-1}$	$8.32 \cdot 10^{-1}$	$8.33 \cdot 10^{-3}$
Leaky relu	10	10	$2.16 \cdot 10^{-1}$	$8.7 \cdot 10^{-1}$	$6.46 \cdot 10^{-3}$
Relu	1	100	$2.36 \cdot 10^{-1}$	$8.65 \cdot 10^{-1}$	$5.87 \cdot 10^{-3}$
Relu	1	1,000	$1.08 \cdot 10^{-1}$	$9.64 \cdot 10^{-1}$	$1.58 \cdot 10^{-3}$
Relu	2	80	$8.36 \cdot 10^{-2}$	$9.76 \cdot 10^{-1}$	$1.04 \cdot 10^{-3}$
Relu	3	40	$2.07 \cdot 10^{-1}$	$8.95 \cdot 10^{-1}$	$5.2 \cdot 10^{-3}$
Relu	5	10	$2.28 \cdot 10^{-1}$	$8.4 \cdot 10^{-1}$	$7.93 \cdot 10^{-3}$
Relu	10	10	$2.31 \cdot 10^{-1}$	$8.33 \cdot 10^{-1}$	$8.32 \cdot 10^{-3}$
Sigmoid	1	10	$6.46 \cdot 10^{-2}$	$9.95 \cdot 10^{-1}$	$2.17 \cdot 10^{-4}$
Sigmoid	1	20	$1.08 \cdot 10^{-2}$	$1 \cdot 10^{0}$	$6.04 \cdot 10^{-6}$
Sigmoid	1	50	$1.16 \cdot 10^{-2}$	$1 \cdot 10^{0}$	$6.12 \cdot 10^{-6}$
Sigmoid	1	100	$2.22 \cdot 10^{-3}$	$1 \cdot 10^{0}$	$3.68 \cdot 10^{-7}$
Sigmoid	1	1,000	$4.98 \cdot 10^{-4}$	$1 \cdot 10^{0}$	$3.44 \cdot 10^{-8}$
Sigmoid	2	10	$1.77 \cdot 10^{-1}$	$9.62 \cdot 10^{-1}$	$1.67 \cdot 10^{-3}$
Sigmoid	2	20	$1.98 \cdot 10^{-1}$	$9.3 \cdot 10^{-1}$	$3.04 \cdot 10^{-3}$
Sigmoid	2	40	$1.94 \cdot 10^{-2}$	$1 \cdot 10^{0}$	$1.85 \cdot 10^{-5}$
Sigmoid	2	80	$1.89 \cdot 10^{-2}$	$1 \cdot 10^{0}$	$1.73 \cdot 10^{-5}$
Sigmoid	3	10	$2.67 \cdot 10^{-3}$	$1 \cdot 10^{0}$	$5.72 \cdot 10^{-7}$
Sigmoid	3	20	$2.08 \cdot 10^{-3}$	$1 \cdot 10^{0}$	$2.37 \cdot 10^{-7}$
Sigmoid	3	40	$1.02 \cdot 10^{-4}$	$1 \cdot 10^{0}$	$1.23 \cdot 10^{-9}$
Sigmoid	5	10	$8.81 \cdot 10^{-4}$	$1 \cdot 10^{0}$	$4.46 \cdot 10^{-8}$
Sigmoid	10	10	$4.3 \cdot 10^{-3}$	$1 \cdot 10^{0}$	$1.75 \cdot 10^{-6}$
Tanh	1	10	$7.23 \cdot 10^{-2}$	$9.94 \cdot 10^{-1}$	$2.59 \cdot 10^{-4}$
Tanh	1	20	$1.87 \cdot 10^{-2}$	$1 \cdot 10^{0}$	$1.84 \cdot 10^{-5}$
Tanh	1	50	$9.84 \cdot 10^{-4}$	$1 \cdot 10^{0}$	$4.31 \cdot 10^{-8}$
Tanh	1	100	$3.4 \cdot 10^{-3}$	$1 \cdot 10^{0}$	$5.54 \cdot 10^{-7}$
Tanh	1	1,000	$6.97 \cdot 10^{-4}$	$1 \cdot 10^{0}$	$3.14 \cdot 10^{-8}$
Tanh	2	10	$4.08 \cdot 10^{-2}$	$9.98 \cdot 10^{-1}$	$7.65 \cdot 10^{-5}$
Tanh	2	20	$6.67 \cdot 10^{-2}$	$9.95 \cdot 10^{-1}$	$2.12 \cdot 10^{-4}$
Tanh	2	40	$5.9 \cdot 10^{-2}$	$9.96 \cdot 10^{-1}$	$1.64 \cdot 10^{-4}$
Tanh	2	80	$1.04 \cdot 10^{-2}$	$1 \cdot 10^{0}$	$5.32 \cdot 10^{-6}$
Tanh	3	10	$8.65 \cdot 10^{-4}$	$1 \cdot 10^{0}$	$7.01 \cdot 10^{-8}$
Tanh	3	20	$1.21 \cdot 10^{-2}$	$1 \cdot 10^{0}$	$7.93 \cdot 10^{-6}$
Tanh	3	40	$1.43 \cdot 10^{-2}$	$1 \cdot 10^{0}$	$1.67 \cdot 10^{-5}$
Tanh	5	10	$7.23 \cdot 10^{-3}$	$1 \cdot 10^{0}$	$2.97 \cdot 10^{-6}$
Tanh	10	10	$8.54 \cdot 10^{-3}$	$1 \cdot 10^{0}$	$3.94 \cdot 10^{-6}$

Table 4: Results for a DNN $N_e = 10^6$ run with a select set of hyper parameters which yielded the best R^2 scores in 3. Most of the hyper parameters inspected yields a R^2 score of more than four repeating digits.

Activation	Layers	neurons	$\max(\varepsilon_{\text{abs}})$	R^2	MSE	Duration
Sigmoid	3	10	$2.6 \cdot 10^{-3}$	$1 \cdot 10^{0}$	$3.79 \cdot 10^{-7}$	$5.19 \cdot 10^{2}$
Sigmoid	3	40	$2.38 \cdot 10^{-4}$	$1 \cdot 10^{0}$	$7.56 \cdot 10^{-9}$	$1.42 \cdot 10^{3}$
Sigmoid	5	10	$3.1 \cdot 10^{-4}$	$1 \cdot 10^{0}$	$6.67 \cdot 10^{-9}$	$7.39 \cdot 10^{2}$
Sigmoid	1	50	$7.08 \cdot 10^{-5}$	$1 \cdot 10^{0}$	$2.86 \cdot 10^{-10}$	$4.42 \cdot 10^{2}$
Sigmoid	1	100	$1.06 \cdot 10^{-4}$	$1 \cdot 10^{0}$	$6.86 \cdot 10^{-10}$	$8.59 \cdot 10^{2}$
Sigmoid	1	1,000	$7.14 \cdot 10^{-5}$	$1 \cdot 10^{0}$	$2.53 \cdot 10^{-10}$	$5.17 \cdot 10^{3}$
Tanh	3	10	$2.38 \cdot 10^{-3}$	$1 \cdot 10^{0}$	$2.4 \cdot 10^{-7}$	$5.7 \cdot 10^{2}$
Tanh	3	40	$3.21 \cdot 10^{-2}$	$9.99 \cdot 10^{-1}$	$7.16 \cdot 10^{-5}$	$2.15 \cdot 10^{3}$
Tanh	5	10	$4.64 \cdot 10^{-3}$	$1 \cdot 10^{0}$	$1.18 \cdot 10^{-6}$	$8.38 \cdot 10^{2}$
Tanh	1	50	$4.35 \cdot 10^{-5}$	$1 \cdot 10^{0}$	$7.27 \cdot 10^{-11}$	$4.28 \cdot 10^{2}$
Tanh	1	100	$2.31 \cdot 10^{-4}$	$1 \cdot 10^{0}$	$2.56 \cdot 10^{-9}$	$1.12 \cdot 10^{3}$
Tanh	1	1,000	$2.84 \cdot 10^{-4}$	$1 \cdot 10^{0}$	$4.44 \cdot 10^{-9}$	$4.98 \cdot 10^{3}$

Table 5: Optimal parameters as dictated by table 4.

Optimizer	Activation	Layers	Neurons
Adam	tanh	1	50

5 Discussion

5.1 Deep neural network

From table 2 and 3 we performed a run with two different optimizers, gradient descent and Adam[5] for $N_e = 10^5$ epochs. The first thing that stands out is that both results for activation functions Leaky Relu and Relu are rather poor. Already, a few of the results have been removed(for $R^2 \lesssim 0.8$). For the remaining results, we have in both cases that Sigmoidal activation and tanh activation clearly outperforms the Relu methods.

From comparing the gradient descent results and Adam results, we chose to go forward with the Adam optimizer as our optimizer of choice, as the resulting $\max \varepsilon_{\text{abs}}$, R^2 and MSE all appeared to be slightly better. A deep investigation was not performed, and is of consideration for the future. From these results, we then selected the layer combinations which appeared the strongest, and moved on to perform a larger and more thorough run with $N_e = 10^6$ epochs.

5.2 Testing out optimal hyper parameters

By using a set of hyper parameters as discussed in the previous paragraph, we retrieved the results which can be viewed in table 4. From this, we see that several combinations are more or less on the same order of magnitude in both error and proximity to the analytical results. A choice was made, and a given set of particular fit parameters where chosen. These were the Adam optimizer, tanh activation function and one layer with 50 neurons5. We then moved

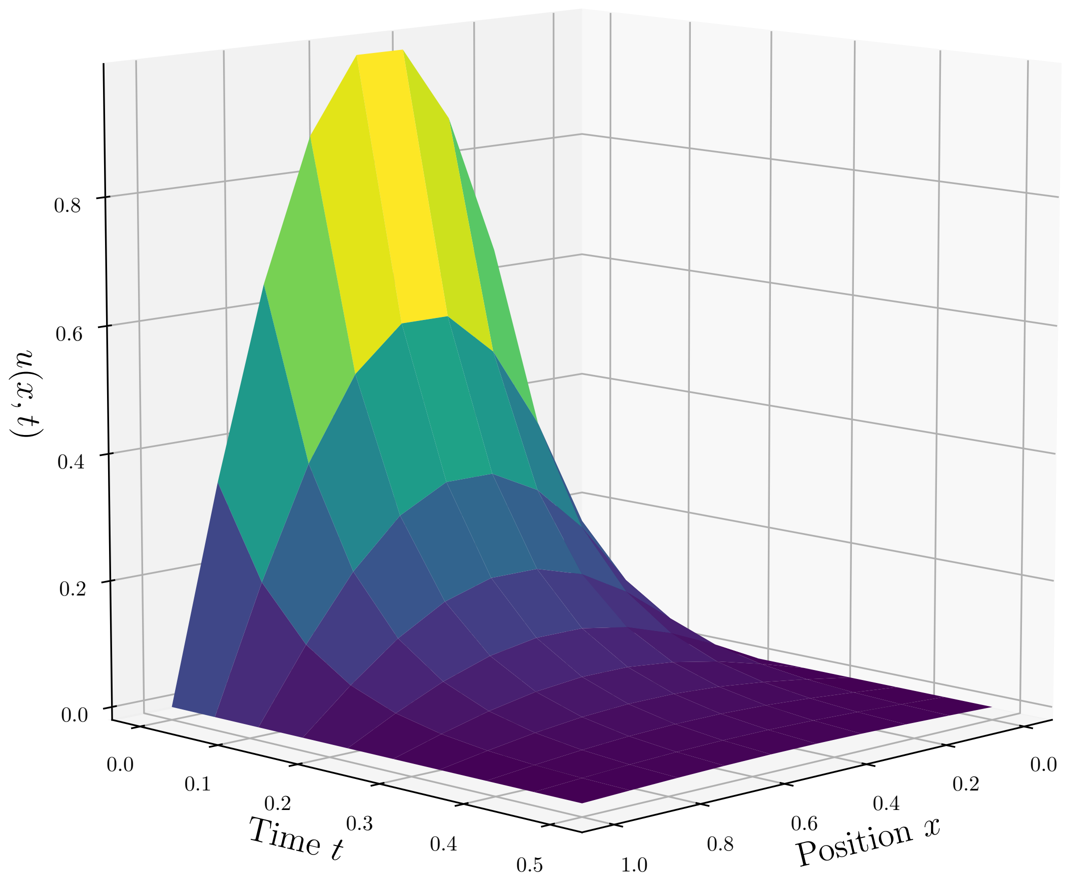


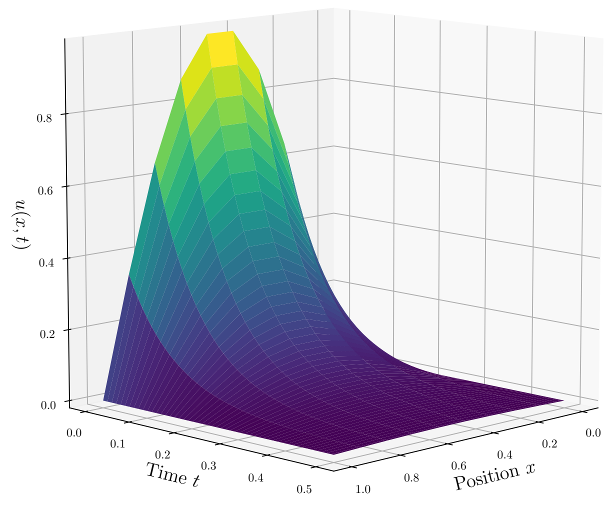
Figure 2: The analytical solution as given by (6) for $N_t = 10$ and $N_x = 10$.

on to perform further comparisons with the finite difference method of Forward Euler and the Neural Network result based of the optimal parameters.

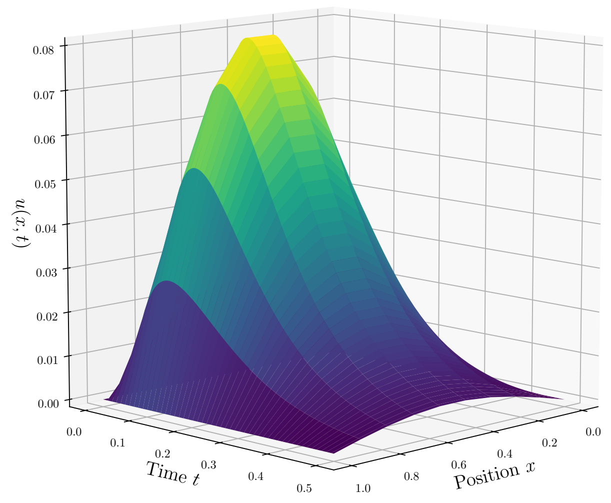
5.3 Cost and Error

The cost and absolute error evolution of the epochs were investigated in the figure 1. We observe that the cost function is decreasing with a monotonic lower bound, although it fluctuates within. The same cannot be said for the maximum difference, as we can see et keeps oscillating between an maximum absolute error of 10^{-3} and 10^{-5}. The behavior of this is not well understood, and it would be interesting to investigate this with different type of optimizers. The behavior of the Adam optimizer may be easily understood considering it is a stochastic optimizer[4]. The main point to make from 1 is that one may leave the network in a state where is is at somewhat unbound upper error, and not a lower bound error. In fact, judging from the fluctuations, it is likely that many of the other hyper parameters are just as

[4]See this page for a discussing surrounding the Adam optimizer.



(a) Forward-Euler.



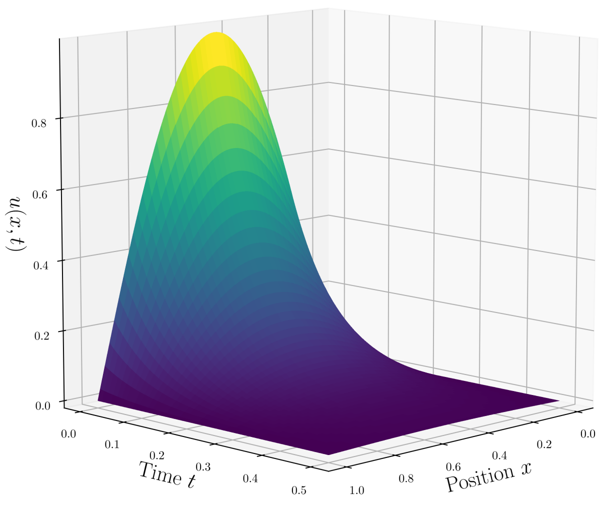
(b) Absolute error between analytical and Forward Euler.

Figure 3: The forward Euler results 3a, and the absolute error between analytical and Forward Euler3b. The Forward Euler was run with $N_x = 10$ and $N_t = 100$ and $\alpha = \Delta t / \Delta x^2 = 0.5$.

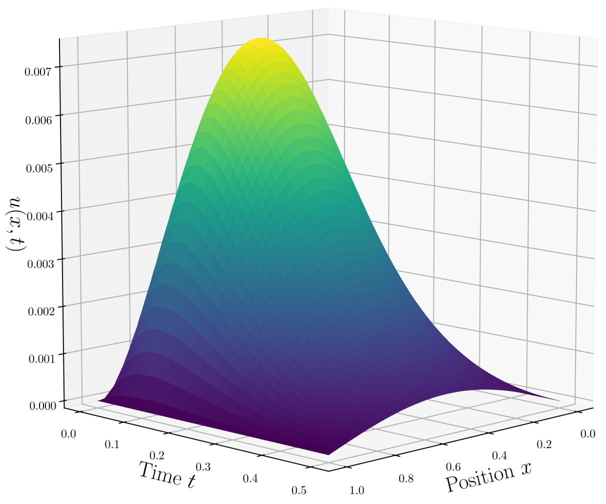
good as the one chosen in table 5 or better.

5.4 Comparing DNN and Forward-Euler against the analytic solution

In figure 2 we presented the analytical solution as given for $N_t = 10$ and $N_x = 10$ points. With this figure in mind, we can take a look at the figure for the Forward-Euler results 3. Due



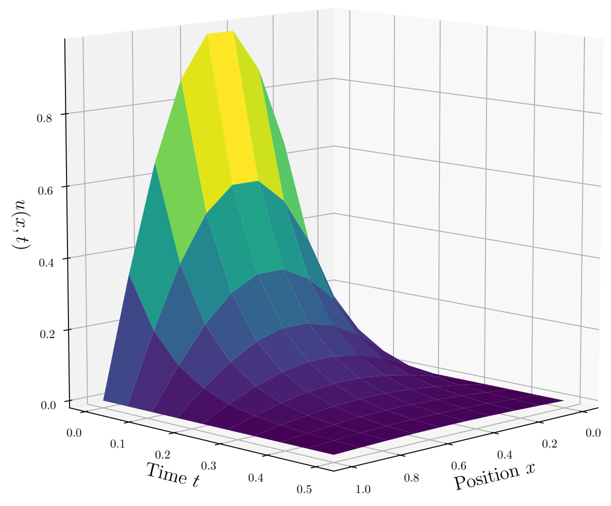
(a) Forward-Euler.



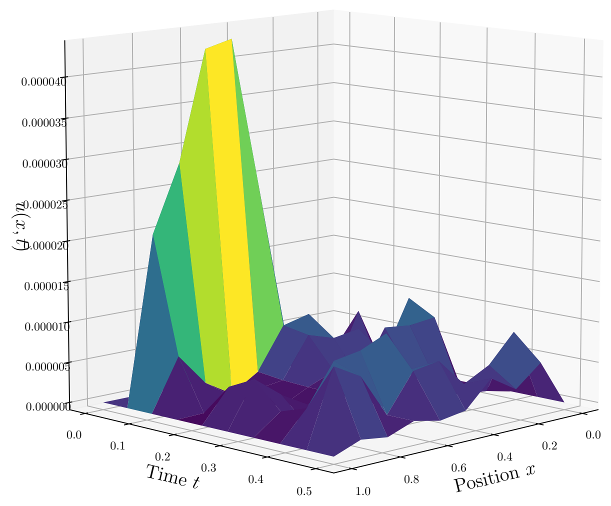
(b) Absolute error between analytical and Forward Euler.

Figure 4: The forward Euler results 3a, and the absolute error between analytical and Forward Euler3b. The Forward Euler was run with $N_x = 100$ and $N_t = 10000$ and $\alpha = \Delta t/\Delta x^2 = 0.5$.

to the stability condition of Forward-Euler $\alpha = \frac{\Delta t}{\Delta x^2} \leq 1/2$, we have $N_t = 100$. This makes direct comparisons between Forward-Euler and the Deep Neural Net somewhat tricky, but we can still draw out some of the main results. By comparing with figure 5, we can see that the overall absolute error is at least three orders of magnitude greater. This is slightly due to the amount of points we are integrating for along the spatial dimension, and can easily be increased to decrease the error, as see in figure 4.



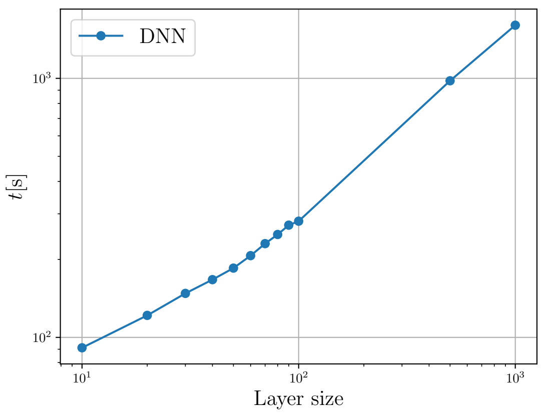
(a) Neural Network.



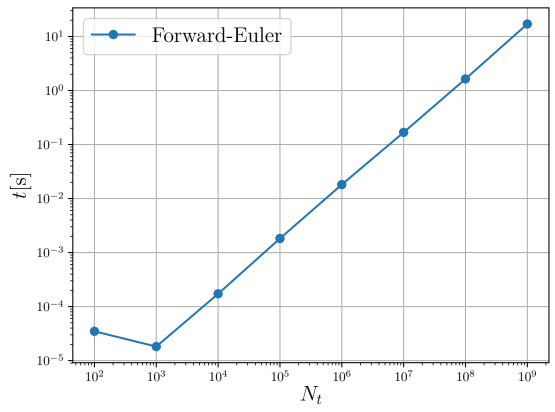
(b) Absolute difference between neural network and analytical results.

Figure 5: The neural net results5a and the absolute difference between the neural net and the analytical results5b. Results run for $N_x = 10$ and $N_t = 10$, with the Adam optimizer, one hidden layer of 50 neurons and tanh as activation function.

Another issue with comparing the results, is that the validity of the results presented are compared against the analytical solution. When solving for a differential equation without any analytical solution, the cost function will be the only available metric to judge the result.



(a) TensorFlow timing values.



(b) Forward Euler timing values.

Figure 6: Timing values for Forward Euler and Neural Net output. The Neural network was calculated using Adam and Sigmoidal activation function. Note, that we are not comparing the same values along the x-axis.

5.5 Timing

A a closing comment, I would note that the time it takes to run a neural network is far greater than that of finite differences, although that has only briefly been touched upon here in figure 6. Even though the x-axis are not equal in any sense, it is clear that the effectiveness of the results from a DNN is heavily dependent on the number of layers. This is also clear from the optimal parameter table, as that contains the Duration of each run in the righter most column 4.

Since the accuracy of Forward-Euler is mainly dependent on the number of integration points, the fact that it is by default a quick algorithm provides is with motivation for utilizing it.

6 Conclusion

A comparison between Forward Euler and Deep Neural Networks, more specifically as multilayer perceptron has been made. From investigating many different combinations of hyper parameters and two different optimizers as seen in table 2 and 3, we selected the most promising results which we saw in table 4. While many of the results found where highly promising, we decided to move on with using the Adam optimizer, one hidden layer with 50 neurons and the tanh activation function. From this we managed to have the maximum difference drop to $\max(\varepsilon_{\text{abs}}) = 4.35 \cdot 10^{-5}$, achieve a $R^2 \approx 1.0$ and MSE score of $7.27 \cdot 10^{-11}$ and get a cost of $\mathcal{C} = 6.74 \cdot 10^{-7}$. The result of this can be seen in figure 5, with the absolute error can be seen in 5b

We also investigated the time for different single hidden layer sizes in 6a and the time dependence on the number of integration points in the temporal direction N_t in figure 6b. What we found was that already for a simple single hidden layer with 10 neurons and 10^5 epochs, we use around 15 seconds, with this climbing steeply to well above 1000 seconds for 1000 neurons. For Forward-Euler the accuracy is mainly dependent on the number of integration points, while for the DNN we are heavily reliant on several factors such as layer size and number of epochs. This makes a direct time comparison difficult, but from what we can see the major drawback with a DNN is the time it takes to train.

6.1 Future works and thoughts

Possible improvements would be to test different $h_1(x,t)$ and $h_2(x,t,P)$ functions.

Test for different optimizers, more layers, larger layers, more input points, and learning rates. Test for more Nx points and Nt points in neural net, as well as forward Euler

Appendices

A Evaluating the performance

Since analytical results is achievable, we will evaulate our final results using the R^2 score,

$$R^2(\boldsymbol{y}, \tilde{\boldsymbol{y}}) = 1 - \frac{\sum_{i=0}^{N-1}(y_i - \hat{y}_i)^2}{\sum_{i=0}^{N-1}(y_i - \bar{\boldsymbol{y}})^2} \tag{15}$$

with $\bar{\boldsymbol{y}}$ defined as

$$\bar{\boldsymbol{y}} = \frac{1}{N}\sum_{i=0}^{n-1} y_i$$

and the MSE score,

$$\text{MSE} = \frac{1}{N}\sum_{i=0}^{N-1}(y_i - \hat{y}_i)^2, \tag{16}$$

where the $\hat{y}$ is the analytical results, and y is the numerical results.

B Neural Net activation functions

B.1 Sigmoid

The **sigmoid** activation function is given as,

$$\sigma_{\mathrm{sig}}(z) = \frac{1}{1 + \mathrm{e}^{-z}} \tag{17}$$

B.2 Hyperbolic tangens

The **hyperbolic tangens** activation function is given as

$$\sigma_{\tanh}(z) = \tanh(z) \tag{18}$$

B.3 Relu

The **relu** or rectifier activation is given as,

$$\sigma_{\mathrm{relu}}(z) = \begin{cases} z & \text{if } z \geq 0 \\ 0 & \text{if } z < 0 \end{cases} \tag{19}$$

B.4 Leaky relu

The **leaky relu** or rectifier activation is given as,

$$\sigma_{\mathrm{lrelu}}(z) = \begin{cases} z & \text{if } z > 0 \\ 0.01z & \text{if } z \leq 0 \end{cases} \tag{20}$$

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `http://tensorflow.org/`. Software available from tensorflow.org.

[2] Mary L. Boas. *Mathematical methods in the physical sciences*. Wiley, Hoboken, NJ, 3rd ed edition, 2006. ISBN 978-0-471-19826-0 978-0-471-36580-8.

[3] Kristine Baluka Hein. *Data Analysis and Machine Learning: Using Neural networks to solve ODEs and PDEs*. November 2018. URL `https://github.com/CompPhysics/MachineLearning/blob/master/doc/pub/odenn/pdf/odenn-minted.pdf`.

[4] Morten Hjorth-Jensen. *Computational Physics, Lecture Notes Fall 2015*. 2015. URL https://github.com/CompPhysics/ComputationalPhysics/blob/master/doc/Lectures/lectures2015.pdf.

[5] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *arXiv:1412.6980 [cs]*, December 2014. URL http://arxiv.org/abs/1412.6980. arXiv: 1412.6980.

[6] I.E. Lagaris, A. Likas, and D.I. Fotiadis. Artificial neural networks for solving ordinary and partial differential equations. *IEEE Transactions on Neural Networks*, 9(5):987–1000, September 1998. ISSN 10459227. doi: 10.1109/72.712178. URL http://ieeexplore.ieee.org/document/712178/.