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 $\alpha = \frac{\Delta t}{\Delta x^2} \leq 1/2$. 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,

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

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 $N_\mathrm{e} = 10^5$ epochs. To evaluate the performance we look at the R^2 score(15) and the MSE score(16). The $\varepsilon_\mathrm{abs} = |u - \hat{u}|$, where u is the numerical results and $\hat{u}$ is the analytical results.

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 $N_\mathrm{e} = 10^6$ epochs. The results for this run can be seen in table 4.

4.2 Forward Euler and finite difference

4.3 Comparing DNN and Forward-Euler against the analytic solution

5 Discussion

Validity of results presented 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.

6 Conclusion

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.

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

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

Table 2: Results for a DNN with different hyper parameters. The number of epoch was set to $N_e = 10^5$. Results presented are obtained with the gradient descent optimizer.

Activation	Layers	Neurons	$\max(\varepsilon_{\text{abs}})$	R^2	MSE
Leaky relu	1	100	$9.88 \cdot 10^{-2}$	0.97	$1.27 \cdot 10^{-3}$
Leaky relu	1	1,000	$1.33 \cdot 10^{-1}$	0.95	$2.23 \cdot 10^{-3}$
Leaky relu	2	80	$1.25 \cdot 10^{-1}$	0.93	$3.08 \cdot 10^{-3}$
Leaky relu	3	40	$1.19 \cdot 10^{-1}$	0.95	$2.4 \cdot 10^{-3}$
Leaky relu	5	10	$1.06 \cdot 10^{-1}$	0.96	$1.76 \cdot 10^{-3}$
Leaky relu	10	10	$1.9 \cdot 10^{-1}$	0.87	$6.23 \cdot 10^{-3}$
Relu	1	100	$1.25 \cdot 10^{-1}$	0.97	$1.41 \cdot 10^{-3}$
Relu	1	1,000	$1.6 \cdot 10^{-1}$	0.91	$3.74 \cdot 10^{-3}$
Relu	2	80	$1.5 \cdot 10^{-1}$	0.93	$2.92 \cdot 10^{-3}$
Relu	3	40	$5.33 \cdot 10^{-2}$	0.99	$5 \cdot 10^{-4}$
Relu	5	10	$2.18 \cdot 10^{-1}$	0.86	$7 \cdot 10^{-3}$
Relu	10	10	$1.08 \cdot 10^{-1}$	0.95	$2.36 \cdot 10^{-3}$
Sigmoid	1	10	$1.14 \cdot 10^{-2}$	1	$1.8 \cdot 10^{-5}$
Sigmoid	1	20	$1.32 \cdot 10^{-2}$	1	$2.4 \cdot 10^{-5}$
Sigmoid	1	50	$1.21 \cdot 10^{-2}$	1	$1.8 \cdot 10^{-5}$
Sigmoid	1	100	$1.36 \cdot 10^{-2}$	1	$2.1 \cdot 10^{-5}$
Sigmoid	1	1,000	$6.25 \cdot 10^{-1}$	-0.47	$6.4 \cdot 10^{-2}$
Sigmoid	2	10	$2.66 \cdot 10^{-2}$	1	$3.4 \cdot 10^{-5}$
Sigmoid	2	20	$2.14 \cdot 10^{-2}$	1	$2.3 \cdot 10^{-5}$
Sigmoid	2	40	$1.62 \cdot 10^{-2}$	1	$1.2 \cdot 10^{-5}$
Sigmoid	2	80	$1.31 \cdot 10^{-2}$	1	$3.6 \cdot 10^{-5}$
Sigmoid	3	10	$6.59 \cdot 10^{-3}$	1	$3 \cdot 10^{-6}$
Sigmoid	3	20	$4.18 \cdot 10^{-3}$	1	$1 \cdot 10^{-6}$
Sigmoid	3	40	$3.07 \cdot 10^{-3}$	1	$1 \cdot 10^{-6}$
Sigmoid	5	10	$2.73 \cdot 10^{-2}$	1	$1.57 \cdot 10^{-4}$
Sigmoid	10	10	$4.54 \cdot 10^{-1}$	0.2	$3.99 \cdot 10^{-2}$
Tanh	1	10	$1.71 \cdot 10^{-2}$	1	$4.5 \cdot 10^{-5}$
Tanh	1	20	$2.26 \cdot 10^{-2}$	1	$2.5 \cdot 10^{-5}$
Tanh	1	50	$9.99 \cdot 10^{-3}$	1	$6 \cdot 10^{-6}$
Tanh	1	100	$1.11 \cdot 10^{-2}$	1	$2.6 \cdot 10^{-5}$
Tanh	1	1,000	$2.13 \cdot 10^{-2}$	1	$8 \cdot 10^{-5}$
Tanh	2	10	$9.26 \cdot 10^{-3}$	1	$1.5 \cdot 10^{-5}$
Tanh	2	20	$2.91 \cdot 10^{-2}$	1	$4.2 \cdot 10^{-5}$
Tanh	2	40	$1.86 \cdot 10^{-2}$	1	$2.5 \cdot 10^{-5}$
Tanh	2	80	$2.23 \cdot 10^{-2}$	1	$3.1 \cdot 10^{-5}$
Tanh	3	10	$7.38 \cdot 10^{-3}$	1	$4 \cdot 10^{-6}$
Tanh	3	20	$5.05 \cdot 10^{-3}$	1	$2 \cdot 10^{-6}$
Tanh	3	40	$5.1 \cdot 10^{-3}$	1	$2 \cdot 10^{-6}$
Tanh	5	10	$1.34 \cdot 10^{-2}$	1	$1.3 \cdot 10^{-5}$
Tanh	10	10	$4.86 \cdot 10^{-3}$	1	$7 \cdot 10^{-6}$

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

Activation	Layers	Neurons	$\max(\varepsilon_{\text{abs}})$	R^2	MSE
Leaky relu	1	100	$1.42 \cdot 10^{-1}$	0.96	$1.9 \cdot 10^{-3}$
Leaky relu	1	1,000	$1.21 \cdot 10^{-1}$	0.94	$2.71 \cdot 10^{-3}$
Leaky relu	2	80	$1.6 \cdot 10^{-1}$	0.92	$3.49 \cdot 10^{-3}$
Leaky relu	3	40	$2.6 \cdot 10^{-1}$	0.89	$5.52 \cdot 10^{-3}$
Leaky relu	5	10	$2.29 \cdot 10^{-1}$	0.83	$8.33 \cdot 10^{-3}$
Leaky relu	10	10	$2.16 \cdot 10^{-1}$	0.87	$6.46 \cdot 10^{-3}$
Relu	1	100	$2.36 \cdot 10^{-1}$	0.86	$5.87 \cdot 10^{-3}$
Relu	1	1,000	$1.08 \cdot 10^{-1}$	0.96	$1.58 \cdot 10^{-3}$
Relu	2	80	$8.36 \cdot 10^{-2}$	0.98	$1.04 \cdot 10^{-3}$
Relu	3	40	$2.07 \cdot 10^{-1}$	0.9	$5.2 \cdot 10^{-3}$
Relu	5	10	$2.28 \cdot 10^{-1}$	0.84	$7.93 \cdot 10^{-3}$
Relu	10	10	$2.31 \cdot 10^{-1}$	0.83	$8.32 \cdot 10^{-3}$
Sigmoid	1	10	$6.46 \cdot 10^{-2}$	1	$2.17 \cdot 10^{-4}$
Sigmoid	1	20	$1.08 \cdot 10^{-2}$	1	$6 \cdot 10^{-6}$
Sigmoid	1	50	$1.16 \cdot 10^{-2}$	1	$6 \cdot 10^{-6}$
Sigmoid	1	100	$2.22 \cdot 10^{-3}$	1	$0 \cdot 10^{0}$
Sigmoid	1	1,000	$4.98 \cdot 10^{-4}$	1	$0 \cdot 10^{0}$
Sigmoid	2	10	$1.77 \cdot 10^{-1}$	0.96	$1.67 \cdot 10^{-3}$
Sigmoid	2	20	$1.98 \cdot 10^{-1}$	0.93	$3.04 \cdot 10^{-3}$
Sigmoid	2	40	$1.94 \cdot 10^{-2}$	1	$1.9 \cdot 10^{-5}$
Sigmoid	2	80	$1.89 \cdot 10^{-2}$	1	$1.7 \cdot 10^{-5}$
Sigmoid	3	10	$2.67 \cdot 10^{-3}$	1	$1 \cdot 10^{-6}$
Sigmoid	3	20	$2.08 \cdot 10^{-3}$	1	$0 \cdot 10^{0}$
Sigmoid	3	40	$1.02 \cdot 10^{-4}$	1	$0 \cdot 10^{0}$
Sigmoid	5	10	$8.81 \cdot 10^{-4}$	1	$0 \cdot 10^{0}$
Sigmoid	10	10	$4.3 \cdot 10^{-3}$	1	$2 \cdot 10^{-6}$
Tanh	1	10	$7.23 \cdot 10^{-2}$	0.99	$2.59 \cdot 10^{-4}$
Tanh	1	20	$1.87 \cdot 10^{-2}$	1	$1.8 \cdot 10^{-5}$
Tanh	1	50	$9.84 \cdot 10^{-4}$	1	$0 \cdot 10^{0}$
Tanh	1	100	$3.4 \cdot 10^{-3}$	1	$1 \cdot 10^{-6}$
Tanh	1	1,000	$6.97 \cdot 10^{-4}$	1	$0 \cdot 10^{0}$
Tanh	2	10	$4.08 \cdot 10^{-2}$	1	$7.7 \cdot 10^{-5}$
Tanh	2	20	$6.67 \cdot 10^{-2}$	1	$2.12 \cdot 10^{-4}$
Tanh	2	40	$5.9 \cdot 10^{-2}$	1	$1.64 \cdot 10^{-4}$
Tanh	2	80	$1.04 \cdot 10^{-2}$	1	$5 \cdot 10^{-6}$
Tanh	3	10	$8.65 \cdot 10^{-4}$	1	$0 \cdot 10^{0}$
Tanh	3	20	$1.21 \cdot 10^{-2}$	1	$8 \cdot 10^{-6}$
Tanh	3	40	$1.43 \cdot 10^{-2}$	1	$1.7 \cdot 10^{-5}$
Tanh	5	10	$7.23 \cdot 10^{-3}$	1	$3 \cdot 10^{-6}$
Tanh	10	10	$8.54 \cdot 10^{-3}$	1	$4 \cdot 10^{-6}$

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.

Activation	Layers	neurons	$\max(\varepsilon_{\text{abs}})$	R^2	MSE
Tanh	3	10	$1.11 \cdot 10^{-2}$	1	$7 \cdot 10^{-6}$

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,

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

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,

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

B.4 Leaky relu

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

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

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/`.