Solving partial differential equations with neural networks

Anna Gribkovskaya, Andrei Kukharenka, Anton Fofanov
FYS-STK4155

December 18, 2018

Abstract

Abstract

We studied the numerical solution of the 1+1-dimensional diffusion problem with neural networks approach. Forward Euler solution was found to be better both in terms of accuracy and runtime than unoptimized neural network result. However the sigmoid activation function and the Adam optimizer with one-hidden layer network outperformed Forward Euler in terms of accuracy. The error for the neural network solver is 3.157×10^{-5}, while for explicit scheme 2.537×10^{-3}.

Introduction

Differential equations are hard to solve analytically, hence numerical methods are employed. In this project we aim to obtain the solution of partial differential equation using historically popular finite difference methods. We are using a standard explicit scheme and neural networks approach to solve the same equations. The motivation behind this is to test a well-developed numerical method designed for such problems against a neural network that is a quite recently developed technique. After both these methods are implemented we want to compare their accuracy and efficiency and make critical comments about these essentially different approaches dedicated to solve the same task.

The aim of the theory part is to present the diffusion equation and show the importance of this topic. We start our discussion on the solution of the diffusion problem with a simple Explicit Scheme for Solving PDE. Last sections of this part contain introduction into neural networks algorithm used to solve the diffusion equation in this project. The final parts of this report are devoted to results and conclusion.

1 The diffusion equation

The diffusion equation is an important ”concept” in the modern science. It came from the Brownian motion and can be solved by stochastic methods. It represents model for a Markov chain. This equation is even used in quantum mechanics and dedicated to solve the electron theory problems. Based on the diffusion equation such advanced methods in quantum chemistry as diffusion Monte Carlo[4] and Coupled Cluster Quantum Monte Carlo[7] were created. All mentioned was possible due to the similarity of the Scrödinger equation and the diffusion equation. We see now that the mathematically isomorphic form of the diffusion equation can be found in totally different fields of science. And that is why the new methods of solving this problem are extremely important to study.

2 Selected methods for solving PDE

A differential equation is an equation where the solution is a function. For a problem like this one usually consider some relations between the functions derivatives that should additionally satisfy some given constrains. The solution in case of 1+1-dimensional diffusion equation depends on both spatial and time variables. Finite difference methods replace derivatives with numerical approximations for them based on Taylor expansions.

2.1 Explicit Scheme for Solving PDE

We consider a simple example of the diffusion equation in one dimension which is written as follows:

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

where $t > 0$, and $x \in [0, L]$. We define the length of the rod L to have value $L = 1$.The equation describes how the derivatives of the function behaves in a given domain with some conditions. The boundary conditions for this task are the following:

$$u(0,t) = 0 \qquad t \geq 0, \tag{2}$$

$$u(L,t) = 0 \qquad t \geq 0. \tag{3}$$

Conditions at $t = 0$ can be expressed as follows:

$$u(x,0) = \sin(\pi x) \qquad 0 < x < L. \tag{4}$$

It is easy to see that in equation (4) function u depends only on position x, and in equations (2) and (3) this function is represented by a constant, however in more general case can be a function of time.

The analytical solution is easy to obtain and it is expressed as follows:

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

To solve this equation numerically it is natural to start with standard approximations for the derivatives for both sides of equation (1). For the left hand side it is possible to write the following expression:

$$\frac{\partial^2 u(x,t)}{\partial x^2} \approx \frac{u(x_i + \Delta x, t_j) - 2u(x_i, t_j) + u(x_i - \Delta x, t_j)}{\Delta x^2}. \tag{5}$$

Viselike for the fight hand side of (1)

$$\frac{\partial u(x,t)}{\partial t} \approx \frac{u(x_i, t_j + \Delta t) - u(x_i, t_j)}{\Delta t} \tag{6}$$

where x and t were discretized as follows:

$$step = \frac{max - min}{number\ of\ steps},$$

or more detailed for x,

$$\Delta x = \frac{x_{\text{max}} - x_{\text{min}}}{n_{\text{steps}}},$$

where Δx is a step length and n_{steps} is a number of gridpoints. One can note that approximation error for the time variable goes as $\mathcal{O}(\Delta t)$ and for the spatial component as $\mathcal{O}(\Delta x^2)$. x_{max} and x_{min} are given by the boundary conditions.

$$x_i = x_{\text{min}} + i\Delta x \qquad i = 0, 1, 2, \ldots, n_{\text{step}} \tag{7}$$

Applying this to our case one can express that the position on step i at time-step j as follows:

$$t_j = j\Delta t,$$

$$x_i = i\Delta x,$$

for $j \geq 0$ and $0 \geq i \geq n + 1$. n_{steps} was denoted as n for simplicity. Now the one-dimensional diffusion equation (4) can be expressed in its discretized version by utilization of equations 5 and 6. The result of substitution the expressions for the derivatives in (4) can be obtained as follows:

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

Equation (8) can be reformulated into explicit form for function u:

$$u_{i,j+1} = \alpha u_{i-1,j} + (1-2\alpha)u_{i,j} + \alpha u_{i+1,j}, \tag{9}$$

where $\alpha = \frac{\Delta t}{\Delta x^2}$. Taking into account the boundary conditions (equations (2) and (3)), one can write $u_{0,j} = u_{n+1,j} = 0$. The most straightforward implementation is rather simple. The main drawback of this simple method is a very weak stability condition, given by[3]:

$$\frac{\Delta t}{\Delta x^2} \leq \frac{1}{2} \tag{10}$$

In conclusion of this section it is worth mentioning that equation can be rewritten as a tridiagonal matrix-vector multiplication[3] and might be somehow numerically optimized. Brute force solution might not be the best choice as matrix contains a lot of zeros. However these technicalities are out of our attention in this project as the main goal is to compare different methods and their precision. The main focus is the neural networks method and it's application to the problem which is presented in the following section.

2.2 Neural networks. The main concepts

In this section we provide a brief overview of the basic ideas behind neural networks, followed by a discussion of their application to the solution of the differential equations.

Neuron networks approach was inspired by the biological concepts. By the analogy how neurons connected in a human brain the mathematical model was established which can be used for both classification and regression problem. Starting from the simplest possible example the neural network can be represented as the following(see figure 1). The neural network has input and

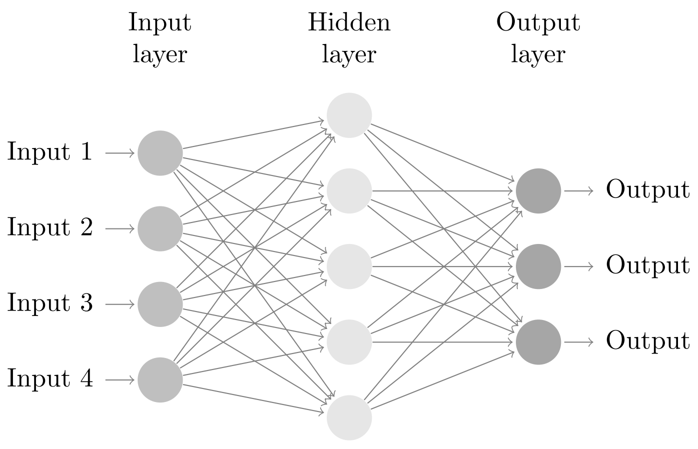


Figure 1: Neural network scheme.

output layers, alongside with some amount of the hidden layers located between input and output. The way neural network operates is activations in one layer determine the activation of the next layer. It can be assigned a weight to each one of the connections between our neuron and the neuron from the following layer. Then the weighed sum of all activations from the input layer is calculated. To be able to handle numerically the values of the activation sum(these values in general can have any values depending on the size of neural network), one needs to have these values to be in interval [0, 1]. To do this an activation function is used. A common function to use is the sigmoid function. The different activation functions we tested in this project will be described below in a greater details.

So the activation function is a basically a way to determine how positive this particular weighed sum is. To find a threshold for activation of one neuron one needs to add some number to the weighed sum. To be more precise, to subtract it from the weighed sum. This number is called the bias and it determines how large the weighted sum needs to be to activate a neuron. Described procedure was for just a one neuron from the hidden layer, where all the neurons from the input layer were connected with. This trail of thoughts can be repeated for the each neuron in the hidden layer, so each neuron has an individual weight and one dedicated bias. We need to find all the right weights and biases. Mathematically it can be expressed via linear algebra language. Organizing all the inputs into a vector A, such so for input layer($l = 0$):

$$A^{(0)} = \begin{bmatrix} x_1^{(0)} \\ x_2^{(0)} \\ x_3^{(0)} \\ \vdots \\ x_m^{(0)} \end{bmatrix} \tag{11}$$

We can construct a weight matrix ($W^{(l)}$)associated with each layer, where each row represents a connection between one layer and a particular neuron in the next layer. By denoting via vector B a row-vector, containing all the biases, the activation of the next layer can be expressed via the following expression:

$$A^{(l+1)} = \sigma(W^{(l)}A^{(l)} + B) \tag{12}$$

For the input layer one applies the latter equation to the input vector and then its activation function. The second layer receives the output of the first layer and the whole scheme is repeated up to output layer.

The learning process of the neuron network is just an adjustment of the weights and biases. So when the neural network gets an unknown data as an input, the output will be correct. We define a cost function which might be seen as a representation of the correlation between input and output. Squared difference between desired result and the output of the neural network can be considered as a measure here. So the cost function is a multidimensional function of all the weights and biases which should provide us a score how good our neural network performs. It is naturally to employ some kind of multi-dimensional minimization algorithms to find a minimum of the cost function. The most simple approach is to use the steepest descent algorithm[6].

2.3 Activation functions

The sigmoid or so-called logistic function has been widely used in the hidden layers of the neural network:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

As an alternative one might use the hyperbolic tangent as the activation function. Another example is the rectified linear unit (ReLU), which is motivated by a biological analogy(neurons would be reither activated or not):

$$\mathrm{ReLU}(x) = \max(0, x)$$

The rectified linear unit usually performs the best [6]. In our project we use all mentioned activation functions.

2.4 Solving partial differential equations with neural networks

We consider a simple example of the diffusion equation in one spatial dimension, so called 1+1-dimentional task. Mathematically the task was formulated in section *subsection* 2.1. Namely we want to solve equation 1 with initial and boundary conditions described by equations 2, 3 and 4. Similarly to the explicit scheme where our first step was to approximate derivatives, we need to reformulate task into an equation in such a form as it will be suitable for a neural network to solve. The simplest possible way is to evaluate RHS and LHS independently and construct the cost function based on obtained values:

$$C = N_x^{-1} N_t^{-1} (\frac{\partial u}{\partial t} - \frac{\partial^2 u}{\partial x^2})^2.$$

Differentiation of this cost function with respect to the weights and biases is a difficult task. In this project we utilize Tensorflow library [1] which provides required functionality.

Then PDE is reformulated into an equation which can be solved by the neural network as a standard minimization problem. The described work-flow works not only for the PDE but also for the ordinary differential equations as well, because they are basically differ only by a number of variables. The starting point of our calculations is the construction of the trial function, which satisfy our equation. In general case it can be expressed as follows:

$$g_t(x) = h_1(x) + h_2(x, N(x, P)), \tag{13}$$

where $N(x, P)$ denotes constructed neural network with weights and biases described by P. The aim to have h_2 in the expression above is to make our trial function 13 to satisfy the solution of the problem under given set of initial(boundary) conditions. With other words, $h_2 = 0$ for the case when $g_t(x)$ is evaluated at the values of x when we know that solution is $h_1(x)$. In this project we used the following trial function:

$$g_t(x) = u(x, t) = sin(\pi x) + tx(1 - x)N(x, t)$$

And finally one needs to choose a neural network architecture. This contains setting up the following parameters: the type of the activation function, number of hidden layers along with the number of neurons within each layer.

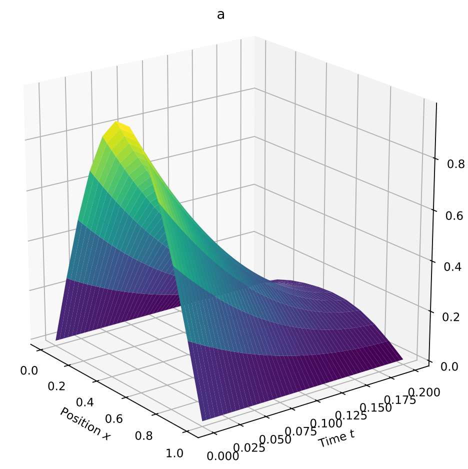
3 Results and discussion

Fortunately the considered problem has an analytical solution, and as a logical start of our research is to ensure if the code is correct. One possible validation is to compare results of neural network and explicit solvers with the analytical ones for the 1+1-dimensional diffusion equation. Neuron Network solver had the following parameters:one hidden layer with 90 neurons, sigmoid activation function, gradient descend optimizer and 10^5 learning iterations. The same discretization scheme was used for both solvers, namely 10 integration points for the spatial variable and 200 point for the time variable. x domain was defined in the problem $x \in [0, 1]$, for the t domain we used $t \in [0, 0.2]$ to satisfy convergence criteria of the explicit solver.

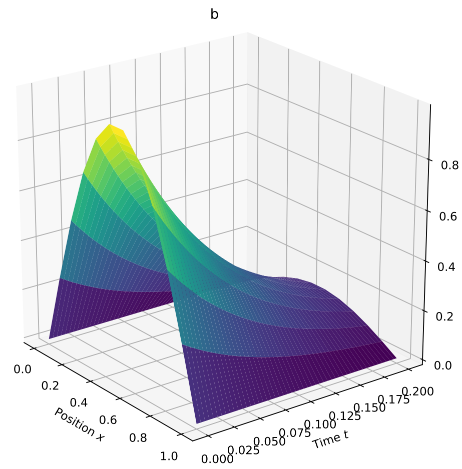
Neural network solver took approximately 31 minute to run while the explicit solver used seconds even if was written on non-compiled language. C++ or Fortran implementation could be even faster. Maximum absolute difference between neural network solver and analytical solution is 0.03104, while this value is almost ten times lower 0.00251 for the explicit scheme. On figure 2 results of two solvers along with the maximum absolute difference between the solutions are presented. From figures 2 (d) and (c) one can notice similar pattern for the error for both solvers. The maximum difference between exact and explicit solver is located where the surface has a larges slope (2) (f).

Visual inspection of the solution together with the values of the absolute difference between numerical and analytical solutions lead us to the assumption that the implementation is correct. The next step is to determine optimal parameters for the neural network and check if it can outperform the explicit scheme solver.

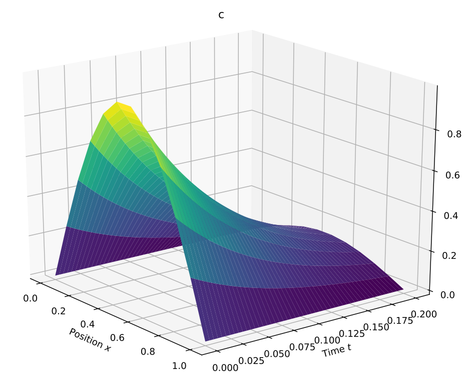
After the first run we could say that implementation is correct and trial function choice is right. We test neural network solver with different parameters and optimizers. Additionally we change the architecture of the hidden layers and try to find a somehow better parameters, so we can try to compare improved neural network solver with the explicit solver again. In real-life problems one might deal with more complicated tasks and an analytical expression for the exact solution might not be possible to find. However in our case we compare neural network solution against the exact one. Additionally to make execution time lower we use a lower number of grid point for the time variable. Fortunately we are not limited by the weak convergence criteria as it was the case for the explicit scheme solver. We tried various network architectures, used different number of hidden layers along with number of neurons within each layer.Gradient descent and Adam methods were used to minimize the cost function. ReLU and sigmoid were chosen for initial tests as an activation functions. Obtained results are presented in tables (2) and (1). It is apparent from this table that gradient descent optimizer works faster for our task. Execution time may differ rougly from 3% to 15% depending on the neural network architecture. Table shows that generally sigmoid activation function provides better results for our particular case. For example, for a neural network with one hidden layer and 1000 neurons change of the activation function from ReLU to sigmoid results in decrease of the difference again analytical solution by 5 orders. What is interesting in this data(tables (2) and (1)) is that network with one layer and large number on neurons gives the best results. Neural network with the same amount of neurons(1000) which are split into 10 layers performed ten times worse, however worked 3 times faster. Strong evidence that one layer neural network with sigmoid activation function provides the best result was found. Adam algorithm gives better results despite the fact that it works slower. So, Adam algorithm and one-layered network were chosen for further analysis of activation functions. The maximum absolute difference with the analytical solution for this case is 5.676×10^{-4} against sigmoid result 6.527×10^{-5}. The latter provided us with the input parameters for the neural network solver which we compared with the explicit scheme solver. For the final calculation the following parameters were used: $x \in [0, 1](\Delta x = 0.1)$, $t \in [0, 0.2](\Delta t = 0.002)$, single hidden layer with 300 neurons, sigmoid activation function and Adam optimization algorithm[5]. The maximum difference with the analytical solution for the neural network solver is 3.157×10^{-5}, while for explicit scheme 2.537×10^{-3}. So one can clearly see that neural network outperforms explicit solver. However in terms of computational costs it performs worse.



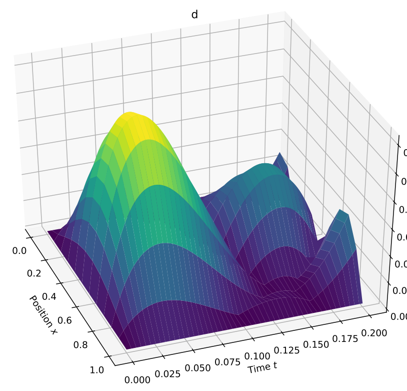
(a) Neural network solver



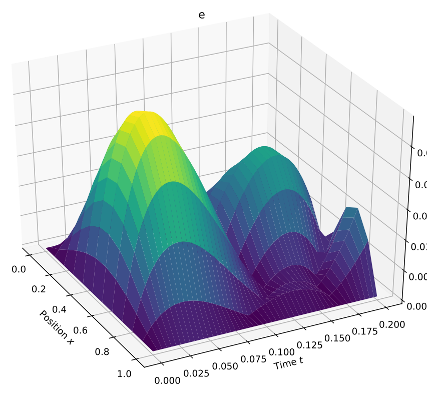
(b) Exact solution



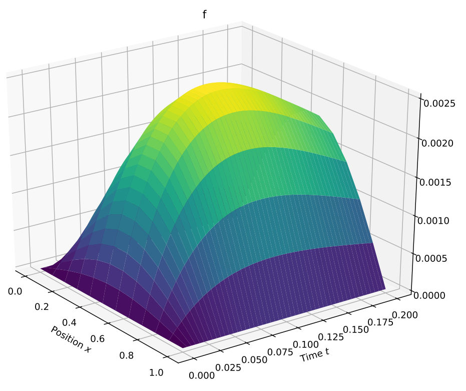
(c) Explicit scheme solver



(d) Neural network vs. exact solution



(e) Neural network vs. explicit scheme



(f) Explicit scheme vs. exact

Figure 2: Results of neural network(a) and explicit(c) solvers with the analytical(b) ones for the 1+1-dimensional diffusion equation(1).$x \in [0, 1](\Delta x = 0.1)$, $t \in [0, 0.2](\Delta t = 0.001)$.

Table 1: Results of the solution of the 1+1-dimensional diffusion problem(1) by neural network. Table explains how different neural network parameters influence on the final result.$x \in [0,1](\Delta x = 0.1)$, $t \in [0, 0.1](\Delta t = 0.1)$. $\max(|\Delta|)$ represents maximum absolute difference with the analytical solution.

Min	Activation	Hidden layers	Neutrons within hidden layer	MSE	R^2	$\max(\|\Delta\|)$	time,s
Adam	ReLU	1	50	1.703×10^{-3}	0.9712	0.0859	84
		1	100	1.703×10^{-3}	0.9712	0.08586	109
		1	500	2.367×10^{-3}	0.959954	0.1736	226
		1	1000	2.141×10^{-3}	0.9638	0.1602	456
		2	20	1.592×10^{-3}	0.9731	0.0851	94
		2	40	1.658×10^{-3}	0.9719	0.08558	126
		2	60	1.703×10^{-3}	0.9712	0.08586	187
		2	80	3.195×10^{-3}	0.9459	0.1945	214
		3	10	3.271×10^{-3}	0.9447	0.1897	95
		3	20	1.703×10^{-3}	0.9712	0.08586	115
		3	40	3.283×10^{-3}	0.9444	0.1898	166
		3	60	2.890×10^{-3}	0.9511	0.1782	261
		5	10	1.590×10^{-3}	0.9731	0.08498	119
		10	10	1.703×10^{-3}	0.9712	0.08586	213
	σ	1	50	8.947×10^{-7}	0.9(4)	2.079×10^{-3}	75
		1	100	1.956×10^{-10}	0.9(8)	4.718×10^{-5}	103
		1	500	3.499×10^{-10}	0.9(7)	7.446×10^{-5}	413
		1	1000	3.565×10^{-10}	0.9(7)	6.527×10^{-5}	960
		2	20	2.623×10^{-9}	0.9(6)	1.625×10^{-4}	114
		2	40	6.076×10^{-9}	0.9(5)	2.209×10^{-4}	166
		2	60	8.884×10^{-7}	0.9(4)	2.143×10^{-3}	233
		2	80	5.881×10^{-9}	0.9(7)	3.563×10^{-4}	270
		3	10	7.7996×10^{-9}	0.9(5)	1.784×10^{-4}	122
		3	20	1.456×10^{-8}	0.9(5)	2.541×10^{-4}	148
		3	40	3.791×10^{-8}	0.9(5)	5.865×10^{-4}	214
		3	60	9.572×10^{-7}	0.9(3)	2.438×10^{-3}	315
		5	10	4.803×10^{-10}	0.9(7)	5.335×10^{-5}	163
		10	10	1.411×10^{-8}	0.9(7)5	4.945×10^{-4}	294

Table 2: Results of the solution of the 1+1-dimensional diffusion problem(1) by neural network. Table explains how different neural network parameters influence on the final result. $x \in [0,1](\Delta x = 0.1)$, $t \in [0,0.1](\Delta t = 0.1)$. $\max(|\Delta|)$ represents maximum absolute difference with the analytical solution.

Min	Activation	Hidden layers	Neutrons within hidden layer	MSE	R^2	$\max(\|\Delta\|)$	time,s
Dradient Descent	ReLU	1	50	2.122×10^{-3}	0.9641	0.1279	61
		1	100	1.721×10^{-3}	0.9709	0.1124	89
		1	500	1.617×10^{-3}	0.9726	0.09868	265
		1	1000	1.873×10^{-3}	0.9683	0.1105	526
		2	20	2.585×10^{-3}	0.9563	0.1832	90
		2	40	2.766×10^{-3}	0.9532	0.1740	124
		2	60	1.566×10^{-3}	0.9735	0.1211	183
		2	80	2.271×10^{-3}	0.9616	0.1234	218
		3	10	3.322×10^{-3}	0.9438	0.1916	91
		3	20	3.081×10^{-3}	0.9479	0.1830	106
		3	40	2.479×10^{-3}	0.9581	0.1711	192
		3	60	7.051×10^{-4}	0.9881	0.07227	261
		5	10	2.903×10^{-3}	0.9509	0.1750	123
		10	10	2.754×10^{-3}	0.9534	0.1432	193
	σ	1	50	6.097×10^{-5}	0.9990	1.850×10^{-2}	66
		1	100	1.025×10^{-4}	0.9983	2.231×10^{-2}	90
		1	500	1.524×10^{-4}	0.9974	3.218×10^{-2}	350
		1	1000	1.696×10^{-3}	0.9713	8.597×10^{-2}	790
		2	20	1.389×10^{-6}	0.9(3)	2.459×10^{-3}	110
		2	40	2.643×10^{-6}	0.9(3)	4.412×10^{-3}	153
		2	60	2.861×10^{-6}	0.9(3)	4.366×10^{-3}	215
		2	80	9.375×10^{-7}	0.9(3)	3.123×10^{-3}	228
		3	10	1.347×10^{-6}	0.9(3)	2.380×10^{-3}	114
		3	20	4.126×10^{-7}	0.9(4)	1.847×10^{-3}	139
		3	40	1.554×10^{-5}	0.9(2)	9.331×10^{-3}	210
		3	60	6.283×10^{-5}	0.9989	1.711×10^{-2}	316
		5	10	1.067×10^{-2}	0.8194	2.211×10^{-1}	153
		10	10	1.069×10^{-2}	0.8192	2.213×10^{-1}	250

4 Conclusion and future prospects

The aim of this thesis was to study neural networks applied to the solution of the partial differential equations. First we started with a brief review of the methods we used.

The computer program using the neural network approach was written in python[2] with a heavy usage of the tensorflow library[1]. As a test method we chose the simplest possible algorithm for solving PDE, namely explicit Euler method. Considered problem have an analytical solution, which was used as a benchmark in the beginning of the project. A lot of test runs for the neural network solver were performed in order to find optimal parameters and network structure for given problem. It was found that the single hidden layer with a large number of neurons gives best results. Additionally several activation functions were tested, and turned out that sigmoid is the best choice. With these optimized parameters it was found that the neural network outperforms the explicit solver. The value of the maximum error compared to the exact solution was better by two orders of magnitude for neural network solver. In conclusion we can say that NN looks as a promising technology for the solution of the differential equations. However additional research is required. In our work we explored just a 1+1-dimentional diffusion problem. It might be interesting how full 3+1-dimentional task can be solved by neural network. Additional study of the layered structure is needed, as our approach was rather simple.

References

[1] Martin Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems.* Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[2] *code:* code repo. URL: https://github.com/andrei-fys/pr3.

[3] Morten Hjorth-Jensen. *Computational Physics. Lecture Notes Fall 2015.* 2015.

[4] Morten Hjorth-Jensen, Maria Paola Lombardo, and Ubirajara Van Kolck, eds. *An Advanced Course in Computational Nuclear Physics: Bridging the Scales from Quarks to Neutron Stars.* eng. Lecture notes in physics volume 936. OCLC: 988901009. Cham: Springer, 2017. ISBN: 978-3-319-53336-0 978-3-319-53335-3.

[5] Diederik P. Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: *CoRR* abs/1412.6980 (2014). arXiv: 1412.6980. URL: http://arxiv.org/abs/1412.6980.

[6] Pankaj Mehta et al. “A high-bias, low-variance introduction to machine learning for physicists”. In: *arXiv preprint arXiv:1803.08823* (2018).

[7] James S. Spencer and Alex J. W. Thom. “Developments in Stochastic Coupled Cluster Theory: The Initiator Approximation and Application to the Uniform Electron Gas”. en. In: *The Journal of Chemical Physics* 144.8 (Feb. 2016), p. 084108. ISSN: 0021-9606, 1089-7690. DOI: 10.1063/1.4942173.