University of Caen

Machine Learning

Solving Partial differential equation with neural networks

Students:

Andrea Espinosa

Vasily Sorokin

Vimal Vijayan

Machine Learning Project for Erasmus Mundus International NuPhys Master Programme

Abstract

In this paper deep forward artificial neural network has been used to solve the one dimensional diffusion equation for a heat conduction problem. Optimization was first performed to select the best parameters for the neural network. It was subsequently trained with one layer and 90 neurons to satisfy the initial and boundary conditions. Results and differences with respect to the analytical solution were compared with the numerical solution given by the Euler method. Results show that in all the cases neural network solutions present a less degree of accuracy and are much more computationally expensive.

Contents

1 Introduction **3**
1.1 Basic Idea of Artificial Neural Network 3

2 Formalism of Neural Networks for partial differential equation **5**

3 Diffusion equation **6**
3.1 Euler's method 6
3.2 Analytical solution 8
3.3 Neural networks 9

4 Analysis **12**
4.1 $\Delta x = 1/10$ 12
4.2 $\Delta x = 1/100$ 15

5 Conclusion **18**

1 Introduction

Finding the solution of a differential equation is one of the essential task in any field of science. In specific, partial differential equations are often encountered in many of the physics problems. The usual method to solve these differential equations are to obtain an analytical solution directly or by making approximations as much as possible. But, analytical solutions are not always possible for which cases numerical techniques like finite difference method comes as an handful approach.

Here, we specifically focus on a method by which we solve the differential equation using the concept of artificial neural networks, as the use of Neural Networks in the field of Physics is vastly extensive and growing (some examples are given in [1]). Starting with the basic concepts and ideas behind the Neural Network, we try to implement these techniques to a simple known diffusion equation. Later, we also analyze the quality of the method by comparing the results with both analytical and numerical solution.

1.1 Basic Idea of Artificial Neural Network

The basic concept of artificial neural network is inspired computer programs on the biological neural networks designed to mimic the way in which the human brain processes an information. It involves things like detecting patterns and recognizing relationships in any given information data to make predictions in the future.

As similar to the case of humans, this process is tried to be mimicked based on three fundamental levels known as layers. They are the input layer, the output layer and an intermediate layer or the layer known as the hidden layer.

The input layer is the first one where all the input data of an information is fetched. This fetching point is called the neurons of the network (or the nodes). In each of these neurons or nodes, to mimic the essential output from the complete given set of data, a processing function or an activation function is used. The objective of this function is basically to identify the basic and crucial information and get rid of all other unimportant details from the input data. This is more similar to the way in which our brain process any information.

The activation function in each node fundamentally works based on two parameters called weights and biases. These serve as the decision parameters of a logistic function. The weights are accompanied to determine the importance of specific variable out of all possible variables and the bias determines whether that variable passes the minimum threshold criteria to be considered for the output (this is similar to the firing of neurons in the brain).

Each neuron in a layer fetches the output of each neuron from the previous layer. For instance, from the input layer with N input data $\vec{x} = (x_1, \ldots, x_N)$, each neuron in the first hidden layer (i^{th} neuron) fetches each output of the input layer (x_j) and accompanies a corresponding weight w_{ij}^{hidden} (i.e. coefficient which is an adjustable parameters) and bias b_i^{hidden} as,

$$z_i^{\text{hidden}} = w_{ij}^{\text{hidden}} x_j + b_i^{\text{hidden}}. \tag{1}$$

In matrix notation writing the input $\vec{x}$ as the column matrix $\mathbf{x}$ we have,

$$\mathbf{z}^{\text{hidden}} = \mathbf{w}^{\text{hidden}}\mathbf{x} + \mathbf{b} \tag{2}$$

where $\mathbf{b}^{\text{hidden}}, \mathbf{z}^{\text{hidden}}$ being the column matrix of the bias and the final input variable for the hidden layer and $\mathbf{w}$ being the matrix of weight factors in the hidden layer.

Now, depending on the value of $\mathbf{z}^{\text{hidden}}$, each neuron gives an output using an activation function f as,

$$\mathbf{x}^{\text{hidden}} = f\big(\mathbf{z}^{\text{hidden}}\big). \tag{3}$$

The most common choices of activation functions from the literature are, the ones that includes nonlinearities in a step-function (perceptrons), sigmoids and the hyperbolic tangent functions.

Thus, the process is repeated with each layer's output simultaneously given as the subsequent layer's input, where neurons in each layer generates an output signal whenever an activation threshold is exceeded. In general, the number of nodes in each layer is arbitrary and manually fixed for better performance. In this model, nodes in each layer is connected to all nodes in the consecutive layer but there is no any inter connection between the nodes of the same layer.

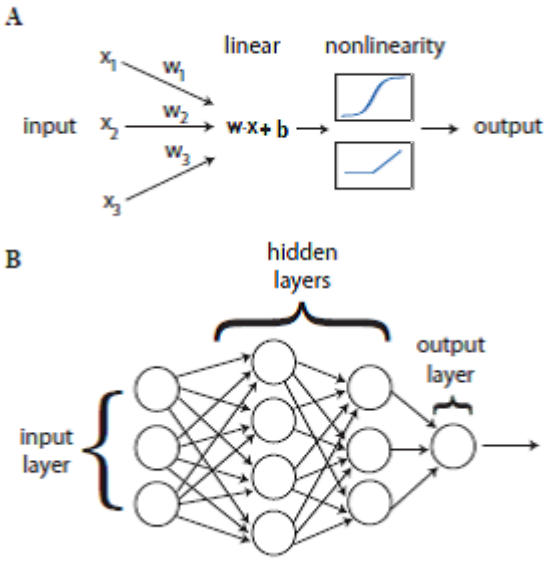


Figure 1: Basic architecture of neural networks. (A) Neurons being the basic components of the neural network takes a linear transformation of weights and bias to determine the importance of various inputs by a non-linear activation function. (B) Neurons are arranged into different layers where the output from each layer serves as the input to the consecutive layer (from [1]).

Application of neural networks in solving a differential equations is promising from Uniqueness theorem which states that a neural network with an input, output and one hidden layer can approximate any arbitrary function. It is based on the fact that, hidden neurons allows neural network to generate step functions with arbitrary offsets and heights which can be

added together to approximate any arbitrary function [2]. More complicated functions requires more hidden nodes (or free parameters). But, the approximation is always possible that makes it to be applicable to any differential equation.

However it also should be noted that, though neural networks are able to approximate any solution, it is not always possible to infer physical meaning from weight factors and biases. Sometimes, physical reasoning may not be adequate just from looking at the solution of a Neural Network. It is one of the drawbacks of using Neural Networks for problems involving unknown physics.

2 Formalism of Neural Networks for partial differential equation

In general, a partial differential equation involves multiple variables, and combinations of possible derivatives expressed as,

$$f\left(x_1, \ldots, x_N, \frac{\partial g(x_1,\ldots,x_N)}{\partial x_1}, \ldots, \frac{\partial g(x_1,\ldots,x_N)}{\partial x_N}, \frac{\partial g(x_1,\ldots,x_N)}{\partial x_1 \partial x_2}, \ldots, \frac{\partial^n g(x_1,\ldots,x_N)}{\partial x_N^n}\right) = 0 \quad (4)$$

where the function f involves all mixed derivatives of g up to order n.

The problem is approached from starting with a trial wave function of the format,

$$g_t(x_1,\ldots,x_N) = h_1(x_1,\ldots,x_N) + h_2(x_1,\ldots,x_N, N(x_1,\ldots,x_N,P)) \quad (5)$$

where $h_1(x_1,\ldots,x_N)$ is used as a function that ensures that $g_t(x_1,\ldots,x_N)$ the trial solution satisfies the given initial conditions of the solution. The contribution from the neural network is expressed by $N(x_1,\ldots,x_N,P)$ where P is used to describe the weights and biases of the network. The output of the neural network is given as the $h_2(x_1,\ldots,x_N,N(x_1,\ldots,x_N,P))$ function. The role of this function is to ensure that, the contribution of neural network vanishes at initial conditions as the initial conditions of the trial function is satisfied solely by $h_1(x_1,\ldots,x_N)$ function.

Once, we model our trial solution, the neural network works based on the principle of the minimization of a function known as the cost function. The cost function is defined by,

$$c\left(x_1,\ldots,x_N,P\right) = \left(f\left(x_1, \ldots, x_N, \frac{\partial g(x_1,\ldots,x_N)}{\partial x_1}, \ldots, \frac{\partial g(x_1,\ldots,x_N)}{\partial x_N}, \frac{\partial g(x_1,\ldots,x_N)}{\partial x_1 \partial x_2}, \ldots, \frac{\partial^n g(x_1,\ldots,x_N)}{\partial x_N^n}\right)\right)^2.$$

Letting $\vec{x} = \left(x_1,\ldots,x_N\right)$ be the array containing the different variables $x_1,\ldots,x_N$ respectively, and M being the number of set of values for $\vec{x}$ (that is $\vec{x}_i = \left(x_1^{(i)},\ldots,x_N^{(i)}\right)$ for $i = 1,\ldots,M$), we have the generalized cost function as,

$$c\left(\vec{x}_i,P\right) = \sum_{i=1}^{M} f\left(\left(\vec{x}_i, \frac{\partial g(\vec{x}_i)}{\partial x_1}, \ldots, \frac{\partial g(\vec{x}_i)}{\partial x_N}, \frac{\partial g(\vec{x}_i)}{\partial x_1 \partial x_2}, \ldots, \frac{\partial^n g(\vec{x}_i)}{\partial x_N^n}\right)\right)^2. \quad (6)$$

The neural network now tries to minimize this cost function using Feed Forward Neural Network algorithm. For instance, in the case of a single hidden layer, there are no weights and bias from the input layer. It gives, $P = \{P_{hidden}, P_{output}\}$.

For a given N_{input} input values and N_{hidden} neurons there are $P_{hidden} = N_{hidden} \times (1+N_{input})$ weights and bias variables contributes to the neural network. Similarly in the output layer, for N_{output} neurons, there are $P_{output} = N_{output} \times (1 + N_{hidden})$ weights and bias variables contribute to the Neural Network.

The minimization of this cost function is done in FFNN where minimization in each step is done by Gradient descent method. For the simple case of one hidden layer, the cost function is minimized with respect to two set of weights and bias namely P_{hidden} and P_{output} that are updated as,

$$P_{\text{hidden,new}} = P_{\text{hidden}} - \lambda \nabla_{P_{\text{hidden}}} c(\vec{x}_i, P) \tag{7}$$
$$P_{\text{output,new}} = P_{\text{output}} - \lambda \nabla_{P_{\text{output}}} c(\vec{x}_i, P) \tag{8}$$

λ known as the learning rate that determines the step size of the algorithm in the direction of the gradient.

Finding the analytic expression for the gradient of the cost function is often very difficult that is automatically done using the libraries known as Autograd when using Tensorflow.

3 Diffusion equation

The aim of this work has been to use neural networks for solving a partial differential equation. In concrete, the problem has been the temperature distribution of a rod of length L=1 placed at the origin (x=0). It can be modeled by the the diffusion equation in the form of a 1D heat conduction problem, whose mathematical expression would then be:

$$\frac{\partial^2 u(x,t)}{\partial x^2} = \frac{\partial u(x,t)}{\partial t}, \quad t > 0, x \in [0, L] \tag{9}$$

The following amplitude and boundary conditions have been considered:

$$u(x,0) = \sin(\pi x) \quad 0 < x < L,$$
$$u(0,t) = 0 \quad t \geq 0,$$
$$u(L,t) = 0 \quad t \geq 0.$$

The time interval studied in all the cases has been set to $t \in [0, T]$ with T=1.
In order to test the validity of our results and make a deeper analysis of the problem, the equation has been solved using three different approaches which are detailed below:

3.1 Euler's method

Euler's method is a numerical scheme used to solve ordinary differential equations with a giving set of initial conditions. The derivatives are replaced by differential quotients that are obtained using Taylor series to approximate the different order derivatives:

$$u'(x) \approx \frac{u(x + \Delta x) - u(x)}{\Delta x} = \frac{u_{i+1} - u_i}{\Delta x} \tag{10}$$

$$u''(x) \approx \frac{u(x+\Delta x) - 2u(x) + u(x-\Delta x)}{\Delta x^2} = \frac{u_{i+1} - 2u_i + u_{i-1}}{\Delta x^2} \tag{11}$$

In the case of the diffusion equation, it is necessary to consider the two variables of the problem (t,x). Therefore first order derivative must be calculated at time t_j whereas second order one at a position x_i:

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

The solution at any time can be then obtained as:

$$u_i^{j+1} = (1-2d)u_i^j + d(u_{i-1}^j + u_{i+1}^j) \tag{13}$$

with $d = \frac{\Delta t}{\Delta x^2}$. The basis of this algorithm's implementation are shown below.

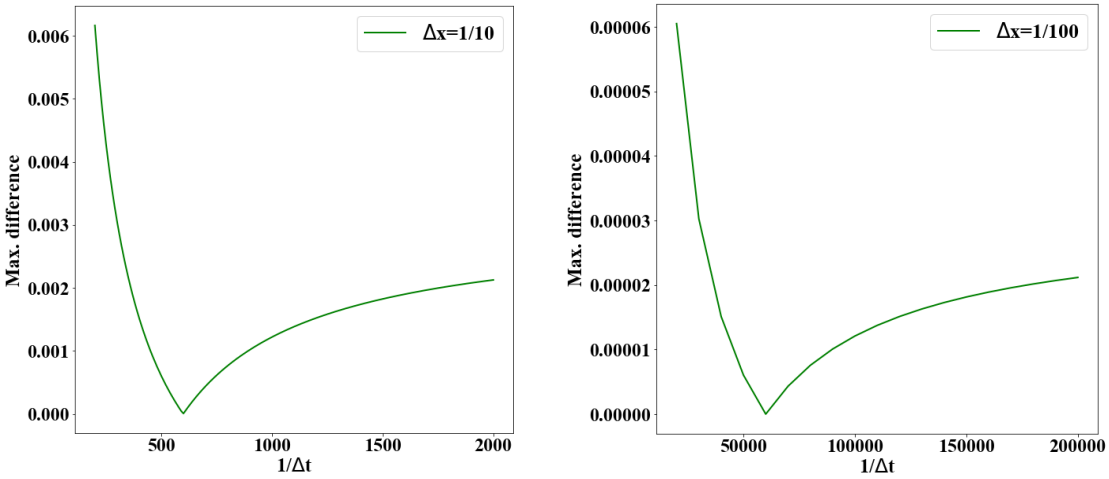
Algorithm .1: Euler's method implementation

```
1: Boundary conditions;
u[x,0]=sin(πx);
u[0,t]=0;
u[L,t]=0;
2: Solving the equation;
for j=1,...N_t do
    for i=1,...N_x do
        u[i, j+1] = (1-2d)u[i, j] + d(u[i-1, j] + u[i+1, j])
    end
end
```

On a first step, a set of values are assigned to the function so that it satisfies the initial conditions. After, its value can be evaluated in the space and time intervals with the recurrence relation derived in 13.
Solutions have been evaluated at two different Δx values using this method: (a) Δx=1/10 and (b) Δx=1/100.
For each of them, the stability criteria needed in this scheme has been considered. It is given by:

$$\frac{\Delta t}{\Delta x^2} \leq \frac{1}{2} \tag{14}$$

To selecth the Δt values, maximum difference has been calculated for different time steps. Then, the value giving the smallest maximum difference between the analytical and numerical solutions has been chosen. To do that it has been calculated for different number of time intervals. Results are ploted in Fig.2a and 2b. The minima corresponds to Δt=1/600 and Δt=1/60000 respectively.



(a) (b)

Figure 2: Maximun difference between the analytical solution and the Euler method for different number of time intervals. In each case, the value given the smallest maximun difference has been chosen to compute the final solution.

3.2 Analytical solution

The analytical solution of the partial differential equation giving by the diffusion equation can be obtained using separation of variables [3]:

$$u\left(x,t\right)=X\left(x\right)T\left(t\right) \tag{15}$$

Then 9 can be rewritten as a system of two ODE's:

$$\begin{aligned} X''(x)+\lambda X(x)&=0\\ T'(t)+\lambda T(t)&=0 \end{aligned} \tag{16}$$

which have respectively the following solutions:

$$X(x)=c_1\cos\left(\sqrt{\lambda}\,x\right)+c_2\sin\left(\sqrt{\lambda}\,x\right) \tag{17}$$

$$T\left(t\right)=B_n\mathbf{e}^{-k\lambda_n t}=c\mathbf{e}^{-k\left(\frac{n\pi}{L}\right)^2 t} \tag{18}$$

The coefficients B_n can be in turn obtained by:

$$B_n=\frac{2}{L}\int_0^L f\left(x\right)\sin\left(\frac{n\,\pi x}{L}\right)\,dx \tag{19}$$

where f(x)=sin(πx) correspond to the initial amplitude giving as an initial condition in our problem.
Solving separately each of them and imposing boundary conditions is possible to obtain the final solution, which is given by:

$$u(x,t)=\exp(-\pi^2 t)\sin(\pi x) \tag{20}$$

3.3 Neural networks

Solving PDEs with a neural network as an approximation is a natural idea, and has been considered in various forms previously [4, 5, 6, 7]. It can be obtained imposing the boundary conditions that are given in this problem in the trial solution (eq. 5). In this case a possible trial function to solve the problem would be then:

$$g_t(x,t) = (1-t)\sin(\pi x) + x(1-x)tN(x,t,P) \tag{21}$$

where $h_1(x,t) = (1-t)\sin(\pi x)$ is chosen to satisfy the initial conditions, and $h_2(x,t) = x(1-x)tN(x,t,P)$ ensures that the output from $N(x,P)$ is zero when the trial function is evaluated at the values of x where the given conditions must be satisfied, in this case at $x,t=0$. The neural network has been created with the facilities that are provided by tensorflow [8]. Originally created by Google, it is an open-source software library intended for automatic learning programming. It allows to built and train neural networks easily. This is represented in Fig.3a and Fig.3b respecively.

```
#-----------------------------------------
# STEP 2: Construction of the neural network.|
#-----------------------------------------

# 1.Definition of the number of neurons within each layer
# and total number interactions
num_hidden_neurons = [90]
num_iter = 1000

X = tf.convert_to_tensor(X)
T = tf.convert_to_tensor(T)

# 2.Construction of the deep neural network with a given activati
with tf.variable_scope('dnn'):

    num_hidden_layers = np.size(num_hidden_neurons)

    # a)Input layer.
    previous_layer = points

    # b)Hidden layers
    for l in range(num_hidden_layers):
        current_layer = tf.layers.dense(previous_layer, num_hidde
                                        activation=tf.nn.sigmoid)
        previous_layer = current_layer

    # c)Output layer
    dnn_output = tf.layers.dense(previous_layer, 1)
```

(a)

```
#-------------------------------------
# STEP 3: Training the neural network.|
#-------------------------------------

def u(x):
    return tf.sin(np.pi*x)

# 1.Definition of the cost (loss) function.
with tf.name_scope('loss'):

    # a)Trial solution wich satisfies the boundary conditions.
    g_trial = (1 - t)*u(x) + x*(1-x)*t*dnn_output
    g_trial_dt =  tf.gradients(g_trial,t)
    g_trial_d2x = tf.gradients(tf.gradients(g_trial,x),x)
    # b)Calculation of the cost function as the mean square error.
    loss = tf.losses.mean_squared_error(zeros, g_trial_d2x[0] - g_trial_dt[0])

# 2.Minimization of the cost function using gradient descent technique.
learning_rate = 0.065

with tf.name_scope('train'):
    optimizer = tf.train.GradientDescentOptimizer(learning_rate)
    traning_op = optimizer.minimize(loss)

init = tf.global_variables_initializer()

g_analytic = tf.sin(np.pi*x)*tf.exp(-(np.pi**2)*t)
g_dnn = None
```

(b)

Figure 3: Code developed using the tensorflow library for the creation and training of the neural network used in this work.

As can be observed, tensorflow allows to build the neural network with different configurations (number of neurons and layers) and activation functions. Training is done following a gradient descent approach (eq.7) to minimize the cost function. The value of the learning rate can also be selected. Furthermore, to develop and implement the neural network, we have studied the influence of the different hyperparameters on the neural network performance and accuracy to match the analytical solution. In general, since there are not theoretical methods for determining their appropriate value, a trial-and-error approach has been mostly used to design the ANN. In concrete the following hyperparameters have been considered:

1. **Network complexity**. Solutions of the neural network have been calculated with different layers and neurons within each of them. For the shake of simplicity, the number of nodes has kept constant in the multilayers models in the trial-and-error approach.

Moreover, other network configurations has been chosen from the literature, according to previous research already made in this area [9, 10, 11, 13, 12].

2. **Learning rate.**

3. **Activation function.**

4. **Number of iterations.**

Then, the implementation procedure has been done as follows.
First, role of the learning rate and activation function have been studied. We have looked at 100 linearly spaces values for the learning rate λ between 10^{-5} and 10^{-1}. This has been done for three different activation functions included in the tensor flow library: `tf.nn.sigmoid`, `tf.nn.tanh` and `tf.nn.elu`. For each of them the values of the maximum difference between the neural network solution and the analytical one as well as the cost function have been calculated.

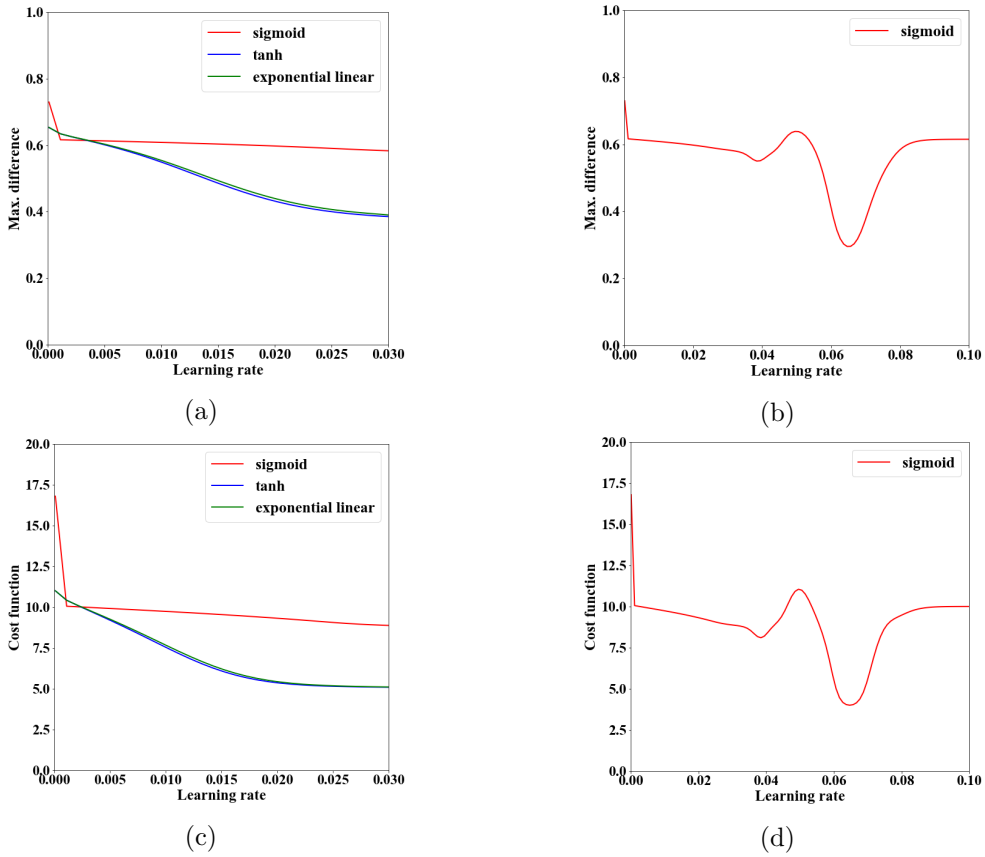


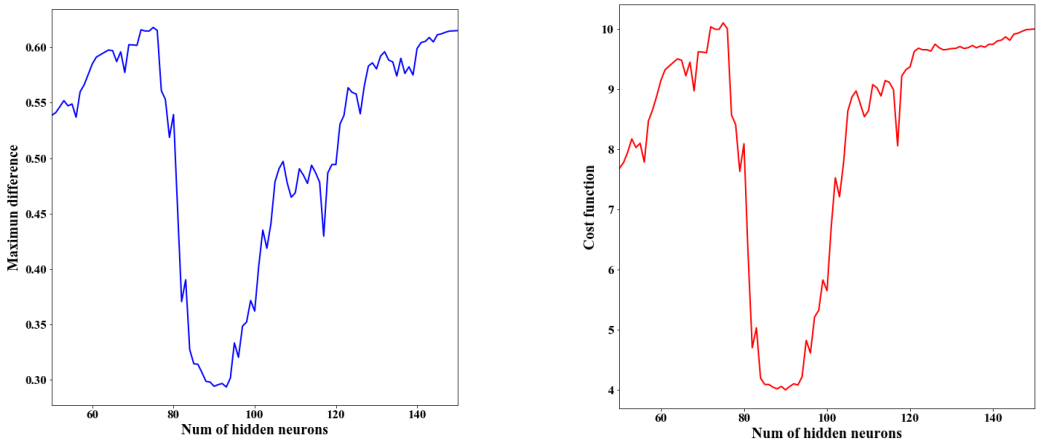
Figure 4: Values of the maximum difference between the analytical and numerical solution ((a) and (b)) and the cost function ((c) and (d)) obtained with the dnn for different values of the learning rate parameter and three diverse activation functions. Both the `tanh` and `elu` give a better result for λ values up to 0.03. However, they start to diverge whereas the `sigmoid` function presents a deep minimum which corresponds to the absolute one at higher λ ((c) and (d)).

To save computation time, the number of iterations has been fixed to 100. Following [13] we have also used a general configuration for the network, with L=1 and N=90 where L and N represent the number of layers and nodes respectively. The results are presented in Fig.4. The `tanh` and `elu` functions show a better performance for $\lambda \leq 0.03$ (Fig.4a and 4c). However, they diverge for higher values. In contrast, the `sigmoid` activation function gives the best overall minimization for both the cost function and the maximum difference between the analytical and numerical solutions. In both cases it presents a deep minimum at a higher value λ=0.065 (Fig.4b, 4d). Therefore, the `sigmoid` activation function and λ=0.065 have been selected and kept fixed in all the other cases.
Secondly, the maximum difference between the analytical and numerical solutions has been again calculated for different network complexities. Different layers and nodes within each layer have been considered. The results obtained for the maximum difference between the numerical and analytical solutions are represented in table 1. The best possible value of 0.294 corresponds to a single layer neural network with N=90. Interestingly, all the other configurations result in a bigger value of the maximum difference between the two solutions with is similar for all of them. Just the other single layer configuration with N=120 gives a smaller value of 0.494.

Table 1: Maximum difference between the analytical and numerical solutions to the diffusion equation for different artificial neural network configurations. Each number corresponds to the number of neurons in each layer. The values correspond to a 10x10 mesh, with a learning rate λ=0.065, a `sigmoid` activation function and a total of 100 iterations.

Reference	**ANN configuration**	**Maximun difference**
[9]	7-25-1	0.614
	7-10-7-1	0.615
[10]	50-50-50	0.615
[11]	2-10-10-1	0.615
[12]	120	0.494
	20-20	0.601
	14-14-14	0.614
	12-12-12-12	0.615
	10-10-10-10-10	0.615
[13]	90	0.294
	100-50-25	0.615

For this reason, we have gone one step further and compute the maximum difference and the cost function for a single layer neural network with different nodes. In concrete, 100 linearly equally spaced nodes (N) between 50 and 150 have been considered. The results are presented in Fig.5. In both cases it can be observed that initially both quantities increase. However, as N starts getting closer to 80, they start to decrease. For instance it shows a minimum at N=90 after which it starts increasing again. According to this, the neural network that has been employed in this work is a single layer neural network with 90 nodes.



(a) (b)

Figure 5: Maximun difference between the analytical and numerical solution (a) and the cost function (b) for different nodes in the single layer neural network configuration.

Finally, the maximum number of iterations is limited to the storage, memory bandwidth, and computational resources. We have try to use the maximum possible number keeping always the computation time below 6 hours.

As in the Euler's method case, solutions have been calculated for two different Δx values: (a) Δx=1/10 and (b) Δx=1/100. In the first case, the value of Δt has been set so that the resulting meshes had the same number of points for both methods. Thus Δt=600. Number of iterations has been 10^5. However, in the second case, it has been found that is not possible to make the computation with the same amount of points in the two employed numerical methods due to the limited capacity of `tensorflow`. Then, since it is not able to run tensors of such big dimensions, in order to compute the solutions with the neural network, for Δx=1/100 the time interval employed has been Δt=1/2000. In this case, due to the needed computation time to perform all the calculations, code has been run just with 1000 iterations.
All the calculations have been performed using a Lenovo Ideapad Yoga 13 portable computer. It posses a Intel (R) Core(TM) i5-3337U CPU @ 1.8 GHz processor with 8.00 GB of RAM.

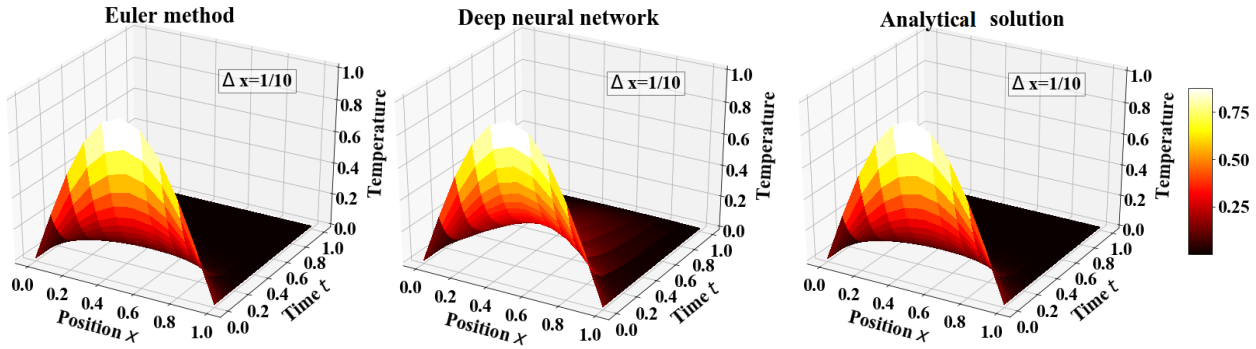
4 Analysis

The different solutions obtained for the 1D diffusion partial differential equation (eq.9) will be presented in this section. As already mentioned, they have been tested at two different Δx values: (a) Δx=1/10 and (b) Δx=1/100. For the neural network, all the calculations have been computed with a single layer neural network model consisting of 90 nodes, a learning rate λ=0.065 and a sigmoid activation function.

4.1 $\Delta x = 1/10$

Fig. 6 depicts the two numerical 3D solutions of the diffusion equation obtained with the Euler's method and the neural network together with the analytical solution. In order to meet the stability criteria given by (14), all the solutions here presented correspond to a time step Δt=1/600. Differences with respect to the analytical solution are not appreciable. In the

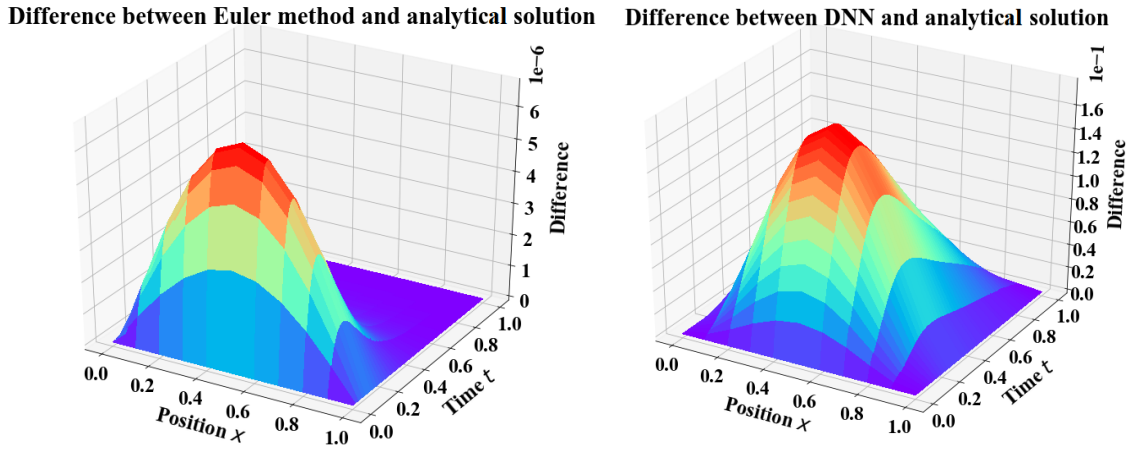
three cases the temperature shows the same behaviour: it initially raises to become maximum in the centre of the rod. However, as time increases, temperature quickly decreases, and the function becomes flat. This would correspond to the stationary state.



(a) (b) (c)

Figure 6: 3D solutions obtained for the diffusion partial differential equation with Δx=1/10 and Δt=1/600.

Nevertheless, computation time has been very different between the two numerical methods. The neural network is rather computationally expensive since the time needed to perform the full calculation is around 4 hours. In contrast, the Euler's method is pretty quickly and solution can be determined in less than one minute. Neural network algorithm presents more complexity. At each iteration all the parameters must be recalculated from the previous ones by minimizing the difference (calculated in terms of a cost function). This means that during training, activations from a forward pass must be retained until they can be used to calculate the error gradients in the backwards pass.
To study these results in more detail, the difference between the numerical and analytical solutions have been calculated for both methods at each point. The results are presented in Fig.7.

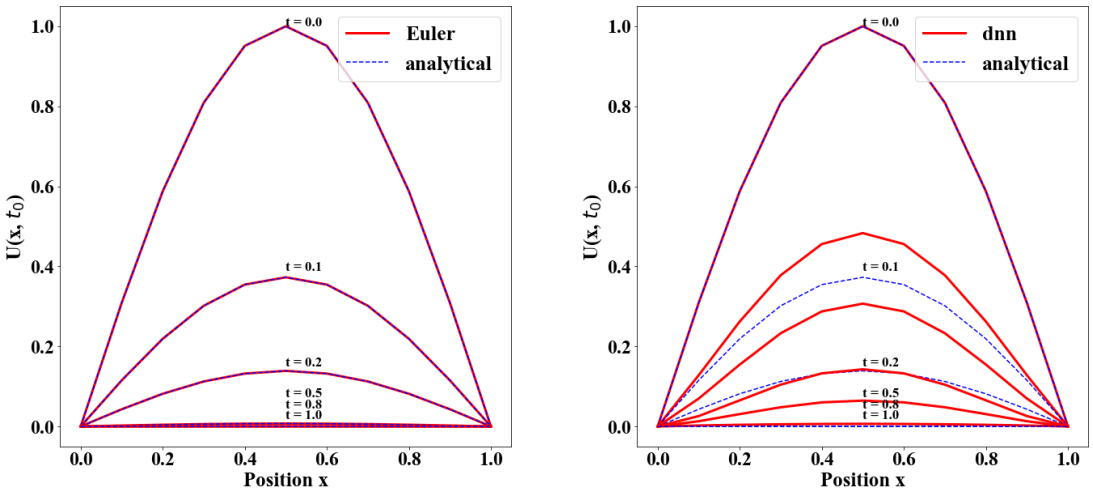


(a) (b)

Figure 7: Differences between the numerical and the analytical solutions.

For the same number of points, it can be observed that the solution obtained with the Euler method is much closer to the analytical one. The differences are indeed five orders of magnitude smaller than the differences obtained with the neural network solution. Moreover, the difference pattern is also diverse. The Euler's method solution fails in the regions where the function presents a bigger curvature whereas the stationary state is rather well approximated. In these areas the curvature of the function could affect the performance of this method. Euler's method uses the derivative of the function to determine its value at each new point. Each new point is then computed from the tangent line at the point given by the previous value. However, this approximation is good only over a small interval. This limitation becomes more noticeable wen the derivative of the function is changing quickly, i.e, where the curvature of the function is bigger. The solution obtained with the neural network, in contrast, presents a uniform difference distribution. The bigger differences are presented in the central region. These points are far from the boundary regions where the initial conditions have been imposed.

Solutions have also been studied at different fixed times. They are depicted in Fig.8 for 6 different time values. Again, Euler's method shows a high degree of accuracy and differences with the analytical solution are not appreciable at any time. In contrast, the differences between the neural network solution and the analytical one are clearly visible. The biggest differences are present at t=0.2 and t=0.5, which are the time values closer to the central region of the function. Moreover, at t=0.2 the function presents a sharp curvature (see Fig.6c) which could also explain the bad performance of the neural network at that given time.



(a) (b)

Figure 8: Evaluation of the solution at different fixed times with the two numerical solutions.

Time evolution of the full function has also been studied at x=0.5, as shown in Fig.9. At t=0 and t=1, the three solutions match. However, the neural network soon moves away. It results in an overall bigger value in all the analyzed time interval which again accounts for the bigger inaccuracy of this second method.

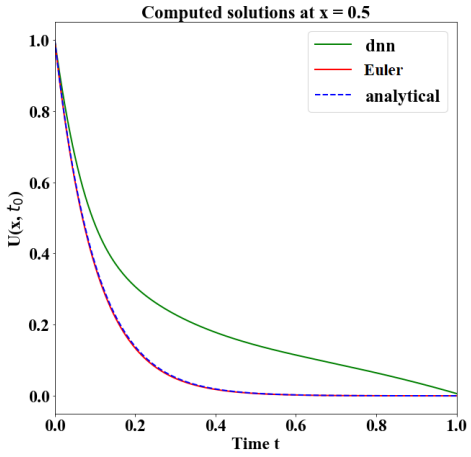
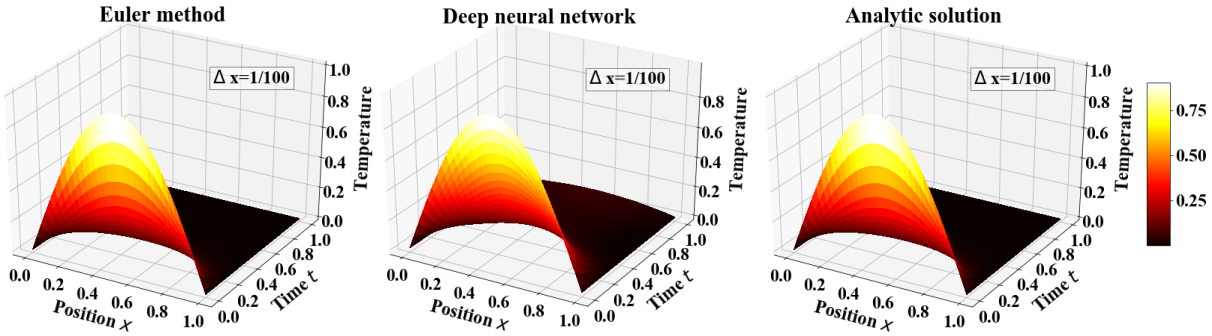


Figure 9: Evaluation of the solution with the two numerical methods at x=0.5

4.2 $\Delta x = 1/100$

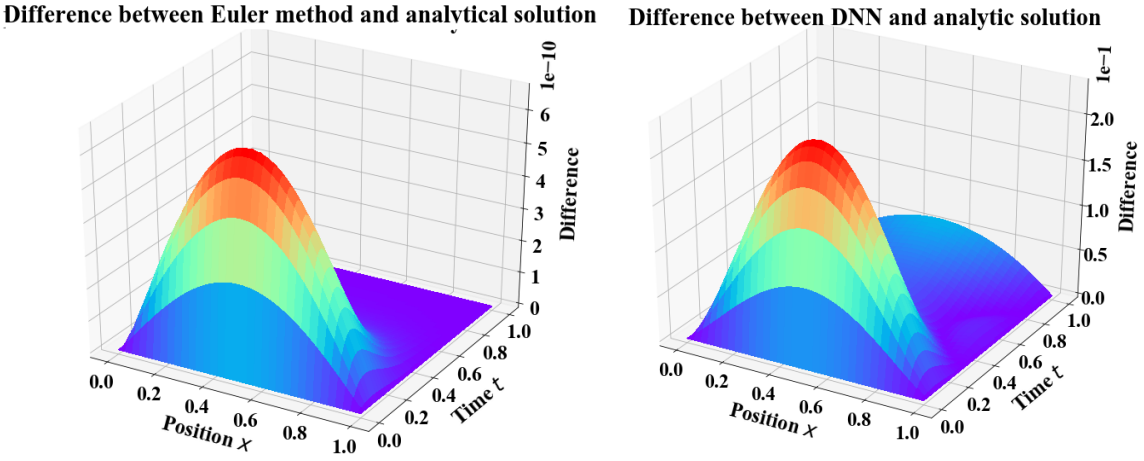
Solutions have been also calculated for Δx=1/100 steps. The 3D surfaces are presented in Fig.10. To meet the stability criteria for the Euler's method, and following the optimization criteria the time step corresponds to Δt=1/60000. As already mentioned, due to the limited capacity of `tensorflow`, in the case of the neural network it has been Δt=1/2000.



(a) (b) (c)

Figure 10: 3D solutions obtained for the diffusion partial differential equation with Δx=1/100.

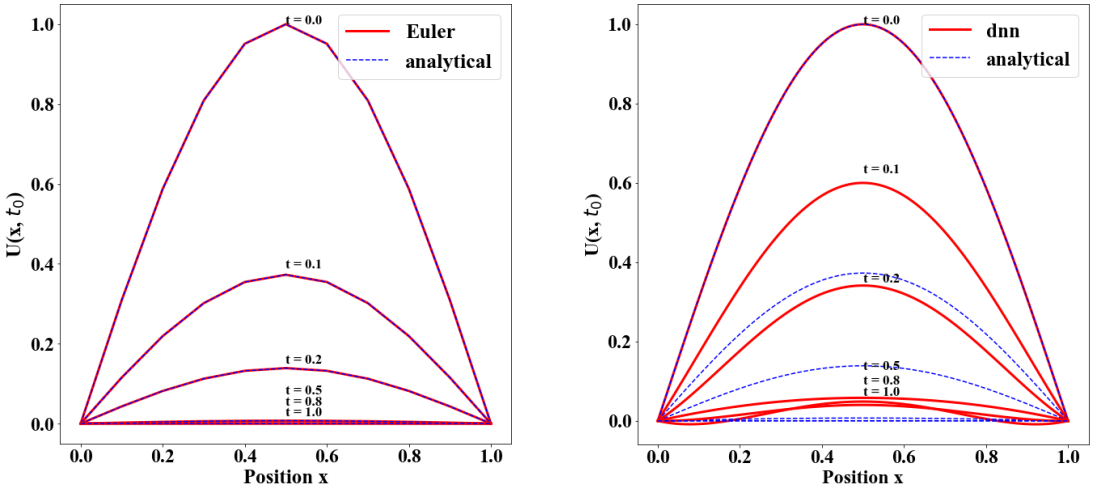
The differences of these two methods with the analytical solution are depicted in Fig.11. They present the same pattern that in the previous case. Euler's method presents the worst performance at initial times, when the curvature of the function is more pronounced. When it reaches the stationary state, the difference reduces to zero. In contrast, dnn fails also in the central result and results in a noticeable difference with respect to to the analytical solution also in the region corresponding to the stationary state.



(a) (b)

Figure 11: Differences between the numerical and the analytical solutions.

Furthermore, it must be noticed that reducing the step (i.e, increasing the number of points and intervals) drastically reduces the differences and error in the case of the Euler's method. Indeed, reducing the steps from Δx=1/10, Δt=1/600 to Δx=1/100, Δt=1/60000 results in a decrease of 4 order of magnitudes in the differences with the analytical solution. In contrast, the differences in the case of the neural network are overall bigger. Although the number of points has increased by reducing the steps from Δx=1/10, Δt=1/600 to Δx=1/100, Δt=1/2000, the number of intervals in which the network is trained has been reduced from 100000 to 1000. This has been due to the fact that computation time required to run over such big number of data points does not allow to run over the same number of iterations in this case. Again, Euler method present a considerable advantage in terms of computational time. Time needed to perform the total computation with the dnn is close to six hours whereas with Euler method is still less than one minute. This has been indeed one of the biggest limitations of the dnn method in this case since the computation time is strongly dependant of the total amount of data points and iterations.
As in the previous case, solutions at different fixed times have been determined with the above number of data points and intervals. The results are presented in Fig.12. As in the previous case, Euler method really succeeds to give a solution close to the analytical one. In contrast, neural networks fails to approximate the analytical solution at all times. Moreover, as already mentioned, the differences are bigger due to the reduced number of iterations employed.



(a) (b)

Figure 12: Evaluation of the solution at different fixed times.

Time evolution of the full function has also again been studied at x=0.5, as shown in Fig.13. Still neural network solution results in a overall bigger value. In this case however, approximation is better at bigger times, in the flat region of the function whereas it results in a worst performance at the beginning.

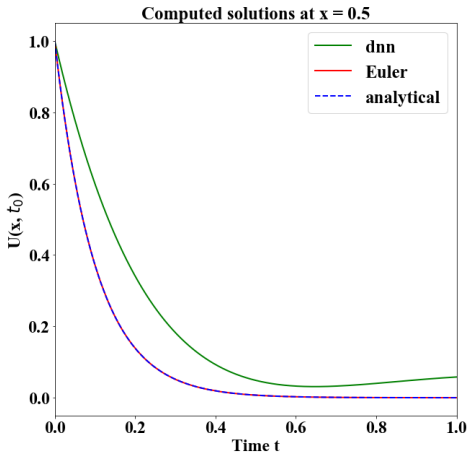


Figure 13: Evaluation of the solution with the two numerical methods at x=0.5

Searching for new ways to improve the performance and improve the solution obtained with the neural network is difficult due to its "black box" nature. When working with neural networks is not possible to certainly know how and why they will came up with a certain output. Their interpretabillity is thus highly reduced.

5 Conclusion

Diffusion equation for a heat conduction problem has been solved in this work using two different numerical methods: Euler's method and an artificial neural network. In both cases solutions have been evaluated with a python-based algorithm at two different space steps: Δx=1/10 and Δx=1/100.
Optimization has been done in both cases to reduce the maximun difference between the analytical and the numerical solution obtained with these two methods. In the case of the Euler method stability criteria sets a minimum limit for the time step value. Optimization is computationally expensive for neural networks due to the lengthy training process that has to be repeated each time a new hyperparameter setting is tested.
3D plots give a similar result with the two numerical methods and the analytical solution. Nevertheless, reducing the steps for which derivative is evaluated significantly improve the results obtained with the Euler method and differences between the numerical and analytical solution can be reduced up to 10^{-10}. In contrast, neural network accuracy is very limited. Increasing the number of points leads to a reduction in the number of iterations over which the neural network can be trained due to the needed computation time. Moreover, total number of points is also limited by the capacity of the `tensorflow` library. This results in a small margin of improvement and in all the cases the differences with the analytical result are of the order of 10^{-1}.
Popularity of neural networks has increased in the last years due to its computational power, algorithm development and increase in the amount of data which significantly improves its performance. However, their training is very slow and adding the tuning of the hyperparameters into that makes it even slower. This introduces a serious cost-benefit tradeoff which also impose a limit in the number of data points and iterations. Due to its black box nature is also difficult to determine the role of the different features in the process.
Considering all of this, traditional numerical methods such us the Euler method are preferred over artificial neural networks for the solution of the one dimensional diffusion equation because for the same number of data points they allow they show a higher degree of accuracy and are computational time much more cheaper. However, they use of more powerful computation systems could highly reduce this limitations. Therefore, more research should be made in the future to test the capability of neural networks algorithms with most powerful computers and different codes and libraries.

References

[1] Pankaj Mehta, Marin Bukov, Ching-Hao Wang, Alexandre G.R. Day, Clint Richardson, Charles K. Fisher, David J. Schwab, A high-bias, low-variance introduction to Machine Learning for physicists, (2009), https://arxiv.org/abs/1803.08823v2.

[2] Nielsen, Michael A (2015), Neural networks and deep learning (Determination Press).

[3] Aslak T., Ragnar W., Introduction to Partial Differential Equations: A Computational Approach, Springer, (2005), Ch. 10

[4] I. Lagaris, A. Likas, D. Fotiadis, Artificial neural networks for solving ordinary and partial differential equations, IEEE Trans. Neural Netw. 9(5) (1998) 987–1000.

[5] I. Lagaris, A. Likas, D. Papageorgiou, Neural-network methods for boundary value problems with irregular boundaries, IEEE Trans. Neural Netw. 11(5) (2000) 1041–1049.

[6] H. Lee, Neural algorithm for solving differential equations, J. Comput. Phys. 91 (1990) 110–131.

[7] K. Rudd, Solving Partial Differential Equations Using Artificial Neural Networks, PhD Thesis, Duke University, 2013.

[8] 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 https://www.tensorflow.org/ Software available from tensorflow.org.

[9] R. Beigzadeha, M. Rahimia, S. R.Shabaniana, Developing a feed forward neural network multilayer model for prediction of binary diffusion coefficient in liquids, Fluid Phase Equilibria, 331 (2012) 48-57.

[10] J.Sirignano, K. Spiliopoulos, DGM: A deep learning algorithm for solving partial differential equations, Fluid Phase Equilibria,331 (2012), 48-57.

[11] M. W. M. G. Dissanayake, N. Phan-Thien, Nueral-network-based aaproximations for solving partial differential eqution., Communications in numerical methods in engineering 10 (1994) 195-201.

[12] Berg, J., Nyström, K.: A unified deep artificial neural network approach to partial differential equations in complex geometries. arXiv preprint arXiv:1711.06464 (2017)

[13] https://compphysics.github.io/MachineLearning/doc/pub/odenn/html/odenn-bs.html