# Approximate Solution for Heat Equation Applying Neural Network

**Zurab Kiguradze**[1,2]

[1] *Electromagnetic Compatibility Laboratory, Department of Electrical and Computer Engineering, Missouri University of Science and Technology, Rolla, MO, USA*

[2] *Ilia Vekua Institute of Applied Mathematics of Ivane Javakhishvili Tbilisi State University Tbilisi, Georgia*

*E-mails:* kiguradzz@mst.edu, zurab.kiguradze@tsu.ge

Our goal in the proposed study is to apply Neural Network (NN) capabilities to the approximate solution of the partial differential equations (PDEs). Neural networks are one of the popular approach to approximate multi-variable nonlinear functions. It has been also successfully applied to different kinds of real-world problems arising in finance, healthcare, signature verification and facial recognition, weather forecasting, etc. In this note, we consider the simple heat equation and propose its approximate solution by using NN.

In the domain $\Omega = (0,1) \times (0,T)$, $T = const > 0$, let us consider the initial-boundary value problem for the heat equation:

$$\begin{gathered} \frac{\partial U(x,t)}{\partial t} - a\,\frac{\partial^2 U(x,t)}{\partial x^2} = f(x,t), \quad (x,t) \in \Omega, \\ U(0,t) = U(1,t) = 0, \quad t \in [0,T], \\ U(x,0) = U_0(x), \quad x \in [0,1], \end{gathered} \tag{1}$$

where $a$ is a positive constant and $U_0$ is a given function.

Qualitative and quantitative properties, as well as the numerical solution for problem (1) and its even more complicated nonlinear analogs, are well-studied in the literature (see, for example, [2–5,9] and the references therein). Our purpose, as we already mentioned, is to study a different approach to solving PDEs by means of Machine Learning methods, in particular, to train the neural network so that the trained surrogate model could predict the solution of PDE at any arbitrary point $(x,t) \in \Omega$. Neural Networks could contain several layers. It necessarily contains input and output layers and could have any number of inner layers called hidden layers (see, for example, Fig. 1). In each layer, the user can choose the number of neurons (green circles).

The neural network constructs approximation for the solution of problem (1)

$$u(x,t,\theta) \approx U(x,t),$$

where $u(x,t,\theta)$ denotes the function obtained from a NN, and $\theta$ is the variable combining all NN parameters which should be optimized during the training of the NN. In general, training of the NN requires a large amount of training data, representing the NN's input. However, applying the NN to the PDEs approximate solution has the advantage due to it tacks into account the physics and therefore shortens the size of the training data (see, for example, [1, 6–8]).

The state-of-the-art machine learning software algorithms, provide automatic differentiation capabilities for functions realized by neural networks, the approximate solution $u(x,t,\theta)$, which in turn allows the residual of the nonlinear problem (1) to be evaluated at a set of training points.

$$R(x,t,\theta) = \frac{\partial u(x,t,\theta)}{\partial t} - a\,\frac{\partial^2 u(x,t,\theta)}{\partial x^2} - f(x,t). \tag{2}$$

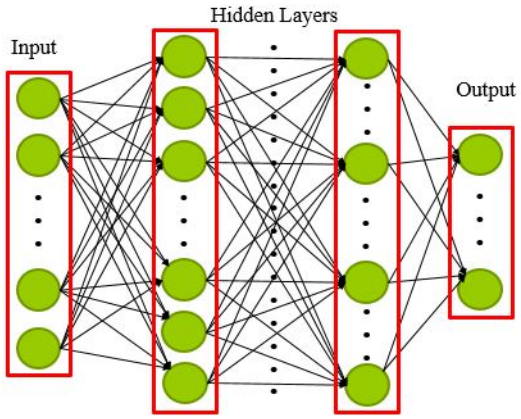


**Figure 1.** Architecture of the general Neural Network.

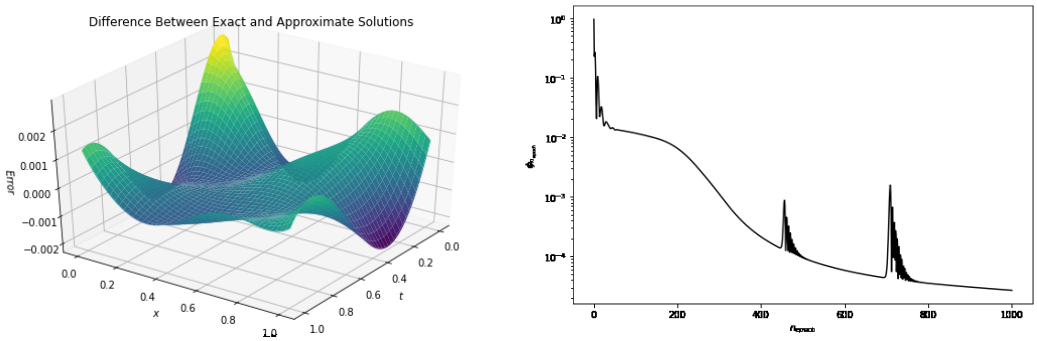


**Figure 2.** Difference between exact and numerical solutions and learning rate (1000 epochs).

Let us construct the cost function $\mathcal{F}(x,t,\theta)$ which should be minimized by a neural network during the training. The cost function should incorporate residual (2) as well as initial and boundary conditions as follows:

$$\mathcal{F}(x,t,\theta) = Err_{residual}(x,t,\theta) + Err_{initial}(x,t,\theta) + Err_{boindary}(x,t,\theta),$$

where

$$Err_{residual}(x_i^r,t_i^r,\theta) = \frac{1}{N_r}\sum_{i=1}^{N_r}\left|R(x_i^r,t_i^r,\theta)\right|^2,$$

$$Err_{initial}(x_i^0,t_i^0,\theta) = \frac{1}{N_0}\sum_{i=1}^{N_0}\left|u(x_i^0,t_i^0,\theta) - U_0(x_i^0)\right|^2,$$

$$Err_{boindary}(x_i^b,t_i^b,\theta) = \frac{1}{N_b}\sum_{i=1}^{N_r}\left|u(x_i^b,t_i^b,\theta)\right|^2.$$

The number of the training points is denoted by $N_r$, while $(x_i^r,t_i^r)$ represents a set of training data. Collection of the following points $(x_i^0,t_i^0)$, $(x_i^b,t_i^b)$ are used for initial and boundary conditions respectively.

Below the results of the numerical experiments are given. For the training of the neural network, the library of scientific computing NumPy and the library of machine learning TensorFlow are used. For the test experiments the Jupyter Notebook implementation, proposed in [1], was modified and

applied to the problem (1). The right-hand side of the problem (1) was chosen in such a way that the exact solution is $U(x,t) = x(1-x)\exp(-x-t)$ with the corresponding initial solution $U_0(x) = x(1-x)\exp(-x)$.

For the initial line and the boundaries we set $N_0 = N_b = 25$ and for the inner points, training data of size $N_r = 1000$ was chosen.

For the neural network architecture, we set five hidden layers and 10 neurons in each layer. The surfaces in Fig. 3 depict exact and numerical solutions when for NN training 1000 epochs (iterations) were used.

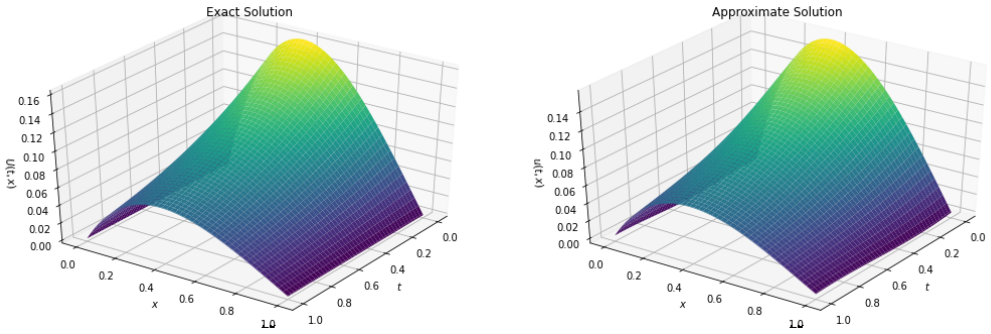


**Figure 3.** Exact and numerical solutions (1000 epochs).

The difference between exact and numerical solutions is given in Fig. 2 (left). In the same figure, the NN learning rate is given on right.

We also ran NN training for 5000 epochs. The results for the difference between exact and numerical solutions and the learning rate are given in Fig. 4.

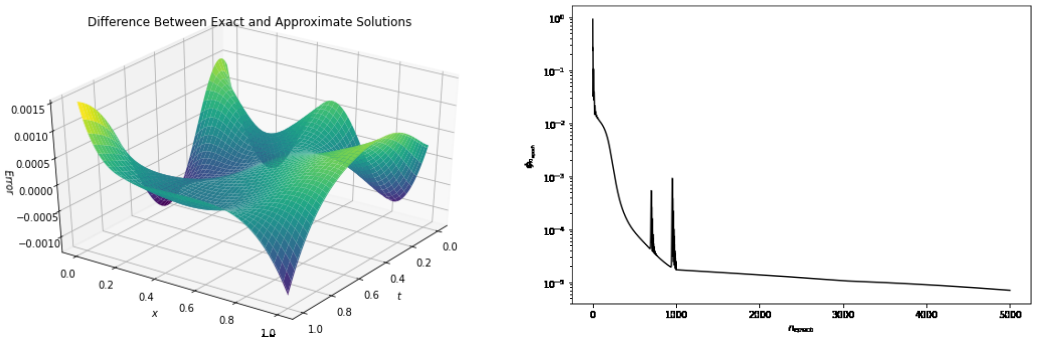


**Figure 4.** Difference between exact and numerical solutions and learning rate (5000 epochs).

## Acknowledgements

This work is supported by the Shota Rustaveli National Science Foundation (SRNSFG), FR-21-2101.

## References

[1] J. Blechschmidt and O. G. Ernst, Three ways to solve partial differential equations with neural network-a review. *GAMM-Mitt.* **44** (2021), no. 2, Paper no. e202100006, 29 pp.

[2] T. Jangveladze, Investigation and numerical solution of nonlinear partial differential and integro-differential models based on system of Maxwell equations. *Mem. Differ. Equ. Math. Phys.* **76** (2019), 1–118.

[3] T. Jangveladze, Z. Kiguradze and B. Neta, *Numerical Solutions of Three Classes of Nonlinear Parabolic Integro-Differential Equations*. Elsevier/Academic Press, Amsterdam, 2016.

[4] Z. Kiguradze, A Bayesian optimization approach for selecting the best parameters for weighted finite difference scheme corresponding to heat equation. *Abstracts of the International Workshop on the Qualitative Theory of Differential Equations – QUALITDE-2019, Tbilisi, Georgia, December 7-9*, pp. 108–111;
http://www.rmi.ge/eng/QUALITDE-2019/Kiguradze_Z_workshop_2019.pdf.

[5] Z. Kiguradze, Gaussian process for heat equation numerical solution. Abstracts of the International Workshop on the Qualitative Theory of Differential Equations – QUALITDE-2017, Tbilisi, Georgia, December 18-20, pp. 120–123;
`http://www.rmi.ge/eng/QUALITDE-2021/Kiguradze_Z_workshop_2021.pdf`.

[6] M. Raissi, P. Perdikaris and G. E. Karniadakis, Physics informed deep learning (Part I): data-driven solutions of nonlinear partial differential equations. *arXiv 1711.10561*, 2017; https://arxiv.org/abs/1711.10561.

[7] M. Raissi, P. Perdikaris and G. E. Karniadakis, Physics informed deep learning (Part II): data-driven discovery of nonlinear partial differential equations. *arXiv 1711.10566*, 2017; https://arxiv.org/abs/1711.10566.

[8] M. Raissi, P. Perdikaris and G. E. Karniadakis, Physics-informed neural networks: a deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *J. Comput. Phys.* **378** (2019), 686–707.

[9] A. A. Samarskii, The Theory of Difference Schemes. (Russian) Nauka, Moscow, 1977.