# On the Solution of Poisson's Equation using Deep Learning

Riya Aggarwal
School of Engineering
University of Newcastle
Newcastle, Australia
Email: Riya.Aggarwal@uon.edu.au

Hassan Ugail
Centre for Visual Computing
University of Bradford
Bradford, United Kingdom
Email: h.ugail@bradford.ac.uk

***Abstract*—We devise a numerical method for solving the Poisson's equation using a convolutional neural network architecture, otherwise known as deep learning. The method we have employed here uses both feedforward neural systems and backpropagation to set up a framework for achieving the numerical solutions of the elliptic partial differential equations - more superficially the Poisson's equation. Our deep learning framework has two substantial entities. The first part of the network enables to fulfill the necessary boundary conditions of the Poisson's equation while the second part consisting of a feedforward neural system containing flexible parameters or weights gives rise to the solution. We have compared the solutions of the Poisson's equation arising from our deep learning framework subject to various boundary conditions with the corresponding analytic solutions. As a result, we have found that our deep learning framework can obtain solutions which are accurate as well as efficient.**

***Index Terms*—Deep Learning, PDE Solutions, Poisson's Equation**

## I. Introduction

The Poisson's equation is represented as $\nabla^2\varphi = -f$, where $\nabla$ is the standard elliptic Laplacian operator, $\varphi$ and $f$ are real valued functions on a manifold. Here, given $f$, $\varphi$ is usually sought subject to a certain set of boundary conditions. This equation is due to the French mathematician, S.D. Poisson who dedicated his life to discovering and teaching mathematics, as the words of Poisson himself highlight it, *'life is good for only two things, discovering mathematics and teaching mathematics'*.

The Poisson's equation can be invoked whenever we are required to study a field in which the gradient of it associated to a force on an entity in which the force is conserved, i.e. the energy difference between any two points in the force field is independent of the path it takes. In this sense, the Poisson's equation is widely applicable to many practical problems in engineering and physics. Examples include, modelling gravitational and electrostatic fields [1], surface reconstruction [2], image processing [3] as well as other applications in geometric design [4], [5], [6], [7], [8], [9], [18].

Many techniques for finding solutions of partial differential equitations (PDEs) have been proposed. These include analytic methods such as the separation of variables, Green's functions, Fourier analysis [10] and many notable numerical schemes such as the finite element method [11], the finite difference method [12] and the finite volume method [13].

Though a great variety of numerical solution techniques for solving the Poisson's equation do exist, the efficiency, as well as the computational complexities of such techniques, are still questionable. As a result, our aim here is to see if emerging machine learning techniques such as deep learning can be utilised to efficiently solve PDEs.

The ability of deep learning in tackling problems such as image classification have, for example, shown great promise [14], [15], [16], [17]. One of the main features of deep learning is that the effectiveness by which it can represent very complex functions when compared to traditional methods such as the use of finite elements or basis functions with a large number of free parameters. Thus, the success of deep learning in many computing related domains has inspired us to see the possibility of utilising it in solving PDEs - more specifically the Poisson's equation in this case.

The rest of this paper is structured as follows. In Section II, we discuss some of the basic concepts of deep learning. In Section III, we discuss the formulation of our deep learning approach to solve the Poisson's equation. In Section V, we discuss some relevant examples to demonstrate the computational accuracy as well as the ease at which our solution scheme can be implemented. Finally, in Section VI, we conclude this paper.

## II. Deep Learning

Deep Learning is a domain of machine learning in which a constructed model learns to perform a given task directly from some knowledge of the data presented to it - be it images, sound or text. Deep learning is often enforced by the exploitation of neural network techniques. The term deep itself means that a variety of layers are involved in the learning process and in general, the more additional layers there are, the better the model performs on the assigned task.

978-1-7281-2741-5/19/$31.00 

As far as the definition goes, a neural network is a parallel distributed information processing structure in the form of a directed graph, where the nodes of the graph are called processing elements and the links are known as the connections. Each processing element can contain any number of incoming and outgoing connections, but the signal will be the same. A single node is insufficient for practical problems and to avoid this, networks with a large number of nodes are generally used. In Figure 1, we illustrate the general form of a neural network system.

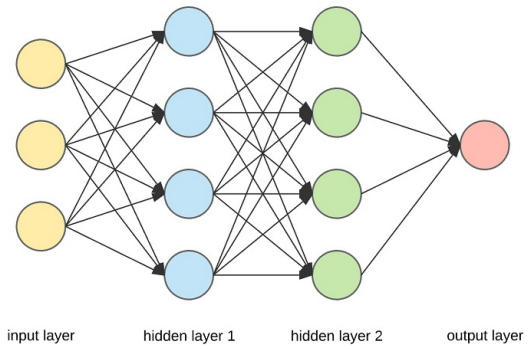


Fig. 1. The general structure of a neural network system.

Deep learning systems are extensions of neural network systems which are derived from perceptrons. A multilayer perceptron (MLP) is a class of feedforward artificial neural network. An MLP comprises of no less than three layers of nodes. With the exception of the info nodes, every neuron has a nonlinear element in it. An MLP uses a directed learning method called backpropagation. Its numerous layers and non-straight initiation differentiates the MLP from a direct perceptron. A solitary layer perceptron shapes half-plane choice districts, a two-layer perceptron can frame a convex (polygon) and a three-layer perceptron can frame subjectively complex choice locales in the info space.

### *A. The Backpropagation Algorithm*

A backpropagation method of artificial neural networks (ANN) has three layers of neurons - an input, a hidden and an output layer - where there are no connections between the layers but they are fully connected between two consecutive layers. There are two connections weight matrices between them. They are,

(1) between an input and the hidden layer,
(2) and between the hidden and an output layer,

which can be used to calculate the gradient of error of the network with respect to the network's modifiable weights. In addition to this, there is a learning rate, $\alpha$, indicating how much change in weight should influence the current weight change.

To apply backpropagation learning procedure, the following entities are required.

1. A set of normalised training patterns.
2. The value for the learning rate.
3. A criterion that terminates the algorithm.
4. A method to update the weights accordingly.
5. An activation function.
6. The initial values for the chosen weights.

## III. Problem Formulation

In this Section, we outline how we developed the deep learning method to formulate the solution of Poisson's equation. In particular, we discuss how the neural network architecture can be laid out and how the necessary gradient computations can be performed for developing the general solutions of the Poisson's equation with given boundary conditions.

### *A. The Neural Network Architecture for Solving Poisson's Equation*

The Poisson's equation is a very powerful tool for modelling the behaviour of electrostatic systems. Often, numerical simulations are utilised in order to model the behaviour of complex geometries with the practical values. The neural network approach as an efficient and computationally accurate alternative is explored here. Here we illustrate this in terms of the following general Poisson's equation subject to general boundary conditions, i.e. both Dirichlet and/or Neumann boundary conditions, such that,

$$-\Delta u = f \text{ in } \Omega,$$
$$u = g_D \text{ on } \partial\Omega,$$
$$u = g_N \text{ on } \partial\Omega.$$

The unknown function $u$ is a two variable entity, $u = u(x, y)$, such that it is defined over a two-dimensional domain $\Omega$.

Solving a boundary value problem such as the Poisson's equation by our proposed method consists of following steps.

1. Identify the computational domain, the PDE (in this case the Poisson's equation), boundary conditions and the source term, $f$.
2. Identify a trial solution for the problem which satisfies all the boundary conditions.
3. Reformulate the trial solution by adding a feedforward neural network term with an input and weight parameters.
4. Train the network and calculate appropriate weights.

The above functionalities can be performed using a computer programming environment. In our case, we have written a Python program which defines the computational domain, the boundary value problem, trail solution, training routine and analytical solution to compare with the numerical solution.

### B. Method of Multilayer Perceptron (MLP) Neural Network

There are different neural networks to find solution of differential equations. Here we are concentrating on the trial solution techniques similar to those proposed in [19], [20], [21]. A method of MLP neural networks is based on the function approximation capabilities of feedforward neural networks and the construction of a trial solution which is in a differentiable and closed analytic form. It employs a feedforward neural network as the basic approximation element whose parameters (or weights) are adjusted to minimise an appropriate error function. Optimisation techniques are used to minimise the loss function which in turn require the computation of error gradient with respect to the input and network parameters.
Let us consider the general Poisson's problem to be solved as,

$$F(\vec{x}, u(\vec{x}), \nabla u(\vec{x}), \nabla^2 u(\vec{x})) = G(\vec{x}), \;\; \vec{x} \in D, \qquad (1)$$

subject to certain boundary conditions where $x \in \mathbf{R}^n$ is independent variable over the domain $D \subset \mathbf{R}^n$ and $u(x)$ is an unknown solution. To construct the trial solution $u_t(x, p)$, we assume that the trial function satisfies the given boundary conditions and it is the sum of two terms where the first term is independent of adjustable parameters (or weights) and the other is with weight parameters. For instance, a suppose trial solution is written as follows,

$$u_t(x, p) = A(\vec{x}) + G(\vec{x}) * N(\vec{x}, \vec{p}), \qquad (2)$$

where, $N(x, p)$, is a feedforward neural network with weight parameter $p$. The task is to learn the parameter $p$ such that Equation 1 is approximately solved by the form in Equation 2. to do this computation, we discretise the domain $D$ and its boundary $S$ into a set of discrete points, $\hat{D}$ and $\hat{S}$ respectively. The problem is then transformed into a system of equations,

$$F(\vec{x_i}, u(\vec{x_i}), \nabla u(\vec{x_i}), \nabla^2 u(\vec{x_i})) = G(\vec{x_i}), \;\; \vec{x_i} \in \hat{D}, \qquad (3)$$

subject to the constraints imposed by the boundary conditions. This relaxation is general and independent of the form in Equation 2 because with a given neural network it may not be possible to exactly satisfy Equation 3 at each discrete point. The problem is further relaxed to find a trial solution that nearly satisfies Equation 3 by minimising a related error index. If $u_t(x, p)$ is the trial solution with the adjustable parameters, $p$, then the problem is transformed to an optimisation problem which is the minimisation problem of the following form,

$$min_{\vec{p}} \sum_{x_i \in \vec{D}} F(\vec{x_i}, u(\vec{x_i}, \vec{p}), \nabla u(\vec{x_i}, \vec{p}), \nabla^2 u(\vec{x_i}, \vec{p})) = G(\vec{x_i}). \qquad (4)$$

subject to the constraints imposed by boundary conditions.
A subtle point is that $A(\vec{x})$ must often be constructed from piecewise boundary conditions. Furthermore, for a given problem there are multiple ways to construct $A(\vec{x})$ and $G(\vec{x})$.

### C. Gradient Computation

Minimisation of the error function can also be treated as a procedure of training the network where the error corresponding to each input vector $x_i$ is the value $f(x_i)$ which has to become zero. Computation of this error value requires network output as well as the derivative of the output with respect to the input vectors. Therefore, while computing the error we need to compute not only the gradient of the network but also the gradient of network derivatives with respect to its inputs.

Consider a multilayer perceptron with $n$ input units, hidden layer with $H$ sigmoid units and a output unit. For a given input vector $\vec{x}$ the output of the network is $N = \sum_{i=H}^{H} v_i \sigma(z_i)$ where $z_i = \sum_{j=1}^{N}, w_{ij} x_j + h_i$, $w_{ij}$ is the weigth from: input unit $(j) \rightarrow$ hidden unit $(i)$, $v_i$ is the weigth from: hidden unit $(i) \rightarrow$ output value, $\sigma(z)$ is the sigmoid transfer function and $h_i$ is the hidden unit $i$. Then it is easy to see that:

$$\frac{\partial^k N}{\partial x_j^k} = \sum_{i=1}^{H} v_i w_{ij}^k \sigma(z_i)^{(k)},$$

$\sigma(z_i)^{(k)}$ denotes the $k^{th}$ order derivative of the sigmoid.

### D. Formulation for the Poisson's Equation

Considering the same problem as in Equation 5, subject to the boundary conditions and initial conditions over domain $D$, the boundary value problem for the general trial solution can be cast as,

$$u_t(x, y) = A(x, y) + x(1 - x)y(1 - y)N(x, y, p),$$

where $A(x, y)$ is chosen as to satisfy the set of given boundary conditions,

$$\begin{aligned} A(x, y) = (1 - x)f_0(y) + xf_1(y) + \\ (1 - y)\{g_0(x) - [(1 - x)g_0(0) + xg_0(1)]\} + \\ y\{g_1(x) - [(1 - x)g_1(0) + xg_1(1)]\}. \end{aligned} \qquad (5)$$

For mixed boundary conditions of the form,

$$\begin{aligned} u(0, y) &= f_0, 0 < y < 1, \\ u(1, y) &= f_1, 0 < y < 1, \\ u(x, 0) &= g_0, 0 < x < 1, \\ \frac{\partial}{\partial y} u(x, 1) &= g_1, 0 < x < 1, \end{aligned}$$

where we have Dirichlet conditions on one part of boundary and Neumann boundary conditions elsewhere, the trial solution will look like,

$$u_t(x, y) = B(x, y) + x(1 - x)y[N(x, y, p) - N(x, 1, p) - E,$$

where,

$$E = \frac{\partial}{\partial y} N(x, 1, p)],$$

where $B(x,y)$ is chosen to satisfy the set of given boundary conditions,

$$B(x,y) = (1-x)f_0(y) + xf_1(y) + \{g_0(x) - [(1-x)g_0(0) + xg_0(1)]\} + y\{g_1(x) - [(1-x)g_1(0) + xg_1(1)]\}. \quad (6)$$

### *E. Alternative Way of Writing a Trial Solution*

The analytic solution of Equation 5 can be written as,

$$\Sigma_{n=1}^{N} X(u,v) = A_0(u) + \Sigma\, A_n(u)cos(nv) + B_n(u)sin(nv),$$

where,

$$A_0(u) = a_{00} + a_{01}u,$$
$$A_n(u) = a_{n1}e^{nu} + a_{n2}e^{-nu},$$
$$B_n(u) = b_{n1}e^{nu} + b_{n2}e^{-nu},$$

where, $a_{00}, a_{01}, a_{n1}, a_{n2}, b_{n1}, b_{n2}$ are all vector-valued constants, whose values are determined by the imposed boundary conditions. We can write $A(x,y)$ in the trial solution as the analytic solution since we need this term to be satisfying the boundary conditions. The benefit of writing an analytic solution as the first term of trial solution is we can reduce the minimisation error and the value of loss function, as can can be seen later in the examples.

Note that the second term of the trial solution does not affect the boundary conditions since it vanishes at the part of boundary where Dirichlet boundary conditions are imposed and its gradient component normal to the boundary vanishes at the part of the boundary where Neumann boundary conditions are imposed. In the above PDE problems, the error to be minimised is given by,

$$E(p) = \sum_i \{\frac{\partial^2 u(x_i,y_i)}{\partial x^2} + \frac{\partial^2 u(x_i,y_i)}{\partial y^2} - f(x_i,y_i)\}^2, \quad (7)$$

where $(x_i, y_i)$ are points in $[0,a] \times [0,b]$.

## IV. Comparison with Finite Difference

The Poisson's equation in the region of computation with Dirichlet boundary condition can be described as,

$$\nabla^2 u(x,y) = f(x,y), \quad (8)$$

where $D = 0 < x < a, 0 < y < b$, with boundary conditions as follows,

$$u(0,y) = f_1, 0 < y < b,$$
$$u(a,y) = f_2, 0 < y < b,$$
$$u(x,0) = g_1, 0 < x < a,$$
$$u(x,b) = g_2, 0 < x < a.$$

In order to compare our neural network that solves the above equation, the second order spatial derivatives should be approximated using finite difference method. The spatial domain is discretised in $x$ and $y$ direction by equidistant nodes, i.e., $\triangle x = \triangle y = h$,

$$\frac{\partial^2 u}{\partial x^2} = \frac{1}{h^2}[u(i+1,j) - 2u(i,j) + u(i-1,j)], \quad (9)$$

$$\frac{\partial^2 u}{\partial y^2} = \frac{1}{h^2}[u(i,j+1) - 2u(i,j) + u(i,j-1)], \quad (10)$$

where, $u(i,j) = u(ih, jh), \quad 1 \leq i,j \leq N$. Using Equations 8 and 9 in Equation 10, we get an ordinary differential equation such that,

$$u_{ij} = -au_{ij} + A^{\star}u_{ij}, \quad (11)$$

where the spatial correlation operator $\star$ is defined as,

$$A^{\star}u_{ij} = \sum_{C(k,l)\in N(i,j)} A(k-i, l-j)u_{kl},$$

$a > 0 \;\; 1 \leq i,j \leq N$, A is a $3 \times 3$ matrix known as the **network template**. Here $A$ is taken to be,

$$A = \begin{bmatrix} 0 & 1/h^2 & 0 \\ 1/h^2 & -4/h^2 + a & 1/h^2 \\ 0 & 1/h^2 & 0 \end{bmatrix}.$$

## V. Experiments and Results

In this section, we discuss a number of experiments and their results whereby we show how solutions of the Poisson's equation in various forumlations can be generated and compared upon.

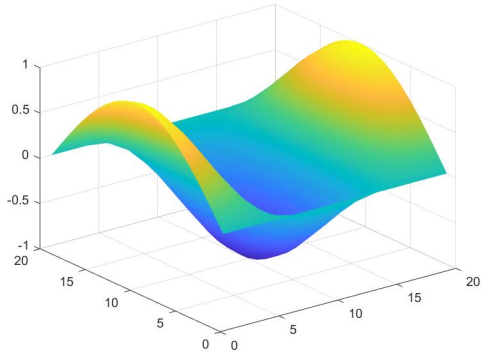


Fig. 2. The analytic solution, for the Example 1.

### *A. Example 1*

In this example, we consider the Poisson's problem subject to boundary conditions over the domain $D = (0,1) \times (0,1)$, such that,

$$\nabla^2 u(x,y) = -5\pi^2 sin(\pi x)cos(2\pi y), \quad (12)$$

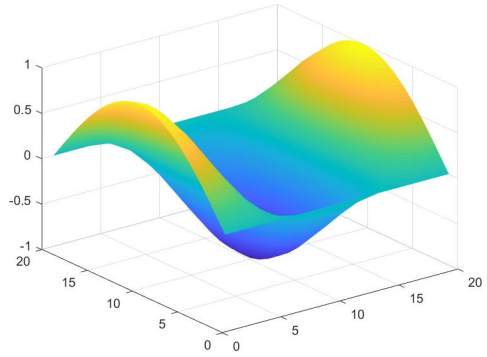

Fig. 3. The Numerical Solution using the our deep learning method, for the Example 1.

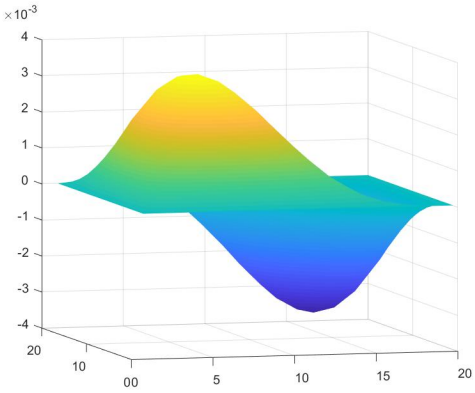

Fig. 4. Error of the numerical solution from our approach against the exact solution, for the Example 1.

where, $D = 0 < x < 1, 0 < y < 1$, with boundary conditions as follows,

$$u(0, y) = 0, 0 < y < 1,$$
$$u(1, y) = 0, 0 < y < 1,$$
$$u(x, 0) = sin(\pi x), 0 < x < 1,$$
$$u(x, 1) = sin(\pi x), 0 < x < 1.$$

For the above boundary value problem, the general trial solution is taken as,

$$u_t(x, y) = A(x, y) + x(a - x)y(b - y)N(x, y, p),$$

where $A(x, y)$ is chosen as to satisfy the set of given boundary conditions,

$$A(x, y) = (1 - x) * 0 + x * 0 + (1 - y)\{sin(\pi x) - [(1 - x) * 0 + x * 0]\} + y\{sin(\pi x) - [(1 - x * 0 + x * 0]\},$$
$$A(x, y) = (1 - y)sin(\pi x) + ysin(\pi x) = sin(\pi x).$$

Figures 2, 3, 4 and 5 show the results for this example.

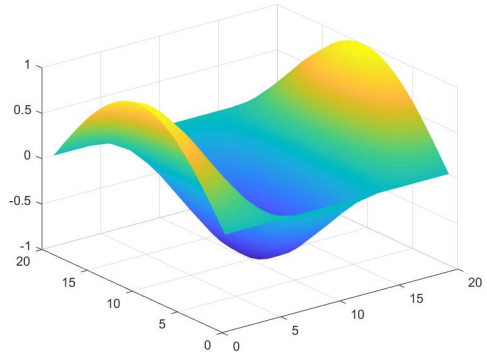

Fig. 5. Numerical Solution using the Finite Difference method, for the Example 1.

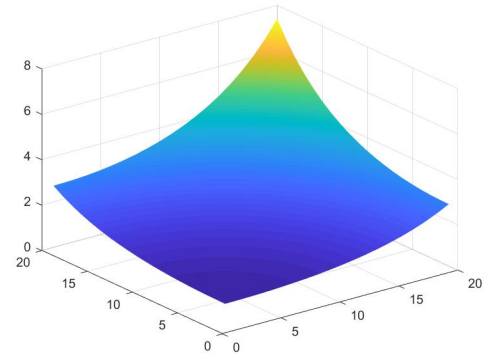

Fig. 6. The analytic solution, for the Example 2.

### B. Example 2

We consider,

$$\nabla^2 u(x, y) = -4e^{(x^2+y^2)}(x^2 + y^2 + 1), \tag{13}$$

where, $D = 0 < x < 1, 0 < y < 1$ with boundary conditions as follows,

$$u(0, y) = e^{y^2}, 0 < y < 1,$$
$$u(1, y) = e^{y^2+1}, 0 < y < 1,$$
$$u(x, 0) = e^{x^2}, 0 < x < 1,$$
$$u(x, 1) = e^{x^2+1}, 0 < x < 1,$$

For the boundary value problem the general trial solution can be cast as,

$$u_t(x, y) = A(x, y) + x(a - x)y(b - y)N(x, y, p),$$

where, $A(x, y)$ is chosen as to satisfy the set of given boundary conditions,

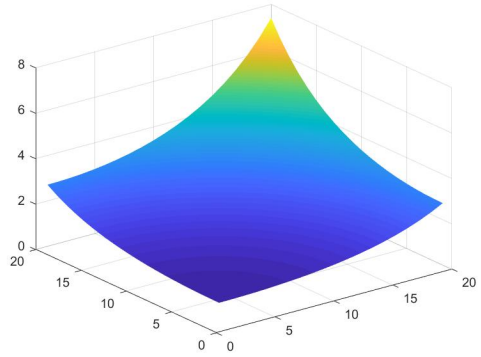


Fig. 7. The Numerical Solution using the our deep learning method, for the Example 2.

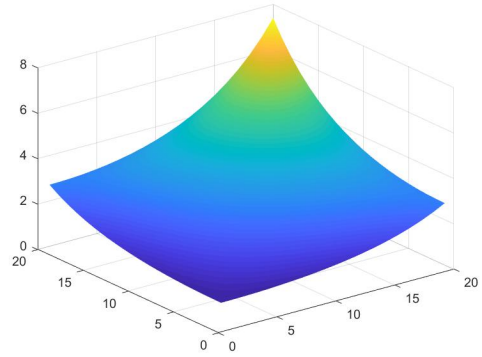


Fig. 9. Numerical Solution using the Finite Difference method, for the Example 2.

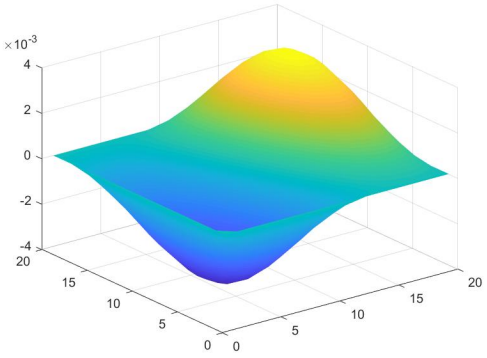


Fig. 8. Error of the numerical solution from our approach against the exact solution, for the Example 2.

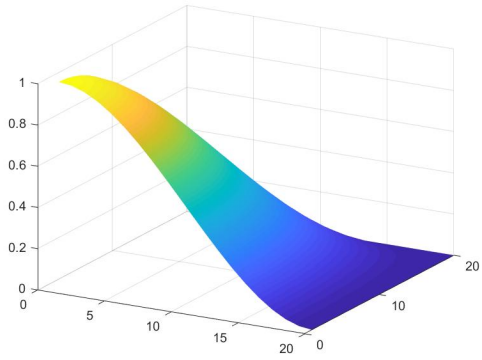


Fig. 10. The analytic solution, for the Example 3.

$$A(x,y) = (1-x)e^{y^2} + xe^{y^2+1} + (1-y)\{e^{x^2} - [(1-x)+xe]\} + y\{e^{x^2+1} - [(1-x)e + xe^2]\}.$$

Figures 6, 7, 8 and 9 show the results for this example.

### C. Example 3

We consider the Poisson's problem for mixed boundary conditions,

$$\nabla^2 u(x,y) = -2(2y^3 - 3y^2 + 1) + 6(1-x^3)(2y-1), \quad (14)$$

$D = 0 < x < 1, 0 < y < 1$, with boundary conditions as follows,

$$u(0,y) = 2y^3 - 3y^2 + 1, 0 < y < 1,$$
$$u(1,y) = 0, 0 < y < 1,$$
$$u(x,0) = 1 - x^2, 0 < x < 1,$$
$$u_y(x,1) = 0, 0 < x < 1.$$

For above boundary value problem, the general trial solution can be cast as,

$$u_t(x,y) = B(x,y) + x(1-x)y[N(x,y,p) - N(x,1,p) - E$$

where

$$E = \frac{\partial}{\partial y} N(x,1,p)],$$

where, $B(x,y)$ is chosen as the same as the analytic solution which is already satisfying the set of given boundary conditions or, can be chosen by the formula given in Equation 6, with,

$$B(x,y) = (1-x^2)(2y^3 - 3y^2 + 1),$$

or

$$B(x,y) = (1-x)(2y^3 - 3y^2 + 1) - x^2 + x.$$

Figures 10, 11, 12 and 13 show the results for this example.

### D. Example 4

We consider,

$$\nabla^2 u(x,y) = 0, \quad (15)$$

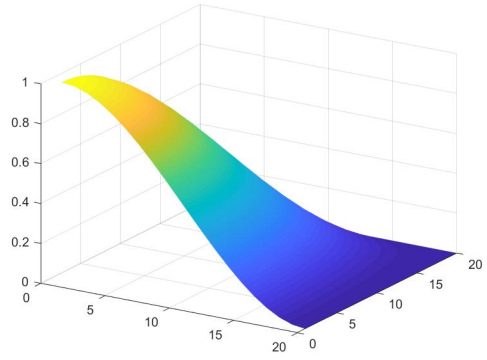


Fig. 11. The Numerical Solution using the our deep learning method, for the Example 3.

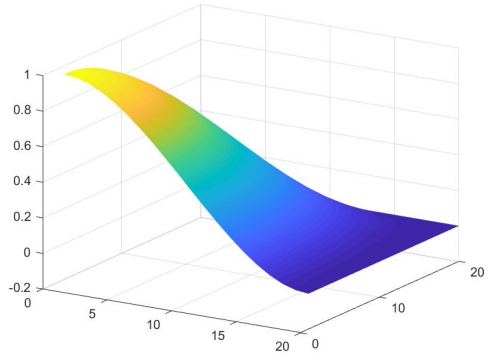


Fig. 13. Numerical Solution using the Finite Difference method, for the Example 3.

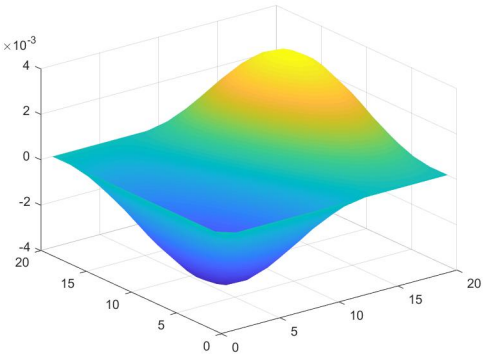


Fig. 12. Error of the numerical solution from our approach against the exact solution, for the Example 3.

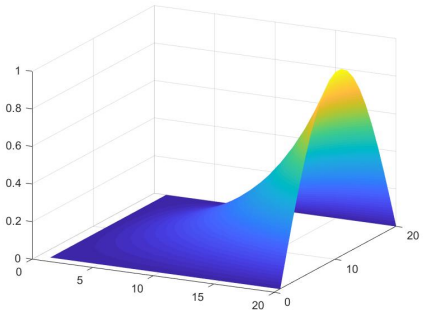


Fig. 14. The analytic solution, for the Example 3.

where $D = 0 < x < 1, 0 < y < 1$, with boundary conditions as follows,

$$u(x, y) = 0, \forall x \in \{(x, y) \in D | x = 0, x = 1, or y = 0\}$$

$$u(x, y) = sin(\pi x), \forall x \in \{(x, y) \in D | y = 1\}.$$

For the above boundary value problem, the general trial solution is taken as,

$$u_t(x, y) = A(x, y) + x(a - x)y(b - y)N(x, y, p),$$

where, $A(x, y)$ is chosen as to satisfy the set of given boundary conditions,

$$A_1(x, y) = ysin(\pi x),$$

or,

$$A_2(x, y) = \frac{1}{(e^{\pi} - e^{-\pi})} \; sin(\pi x)(e^{\pi y} - e^{-\pi y}).$$

Figures 14, 15 and 16 show the results for this example.

## VI. Conclusion

In this paper, we discuss a method of solving Poisson's equation that relies on the ability of function approximation capabilities of the feedforward neural networks and provides accurate and differentiable solutions in a closed analytic form. Thus, a designed and trained cellular neural network is shown to be able to simulate the solutions of PDEs.

We show how a convolutional neural network modified by a backpropagation algorithm can be utilised to numerical solve PDEs. Results show that designed neural network can be as precise as the analytic solutions. The success of the method can be attributed to two factors. The first is the employment of neural networks that are excellent in approximating functions that are of complex nature and the second is the form of the trial solution that satisfies by way construction of the boundary conditions. Therefore the constrained optimisation problem becomes substantially simpler than an unconstrained problem. Unlike most previous approaches, the method is general and can be applied to solve both ordinary as well as partial differential equations by constructing the appropriate form of the trial solution.

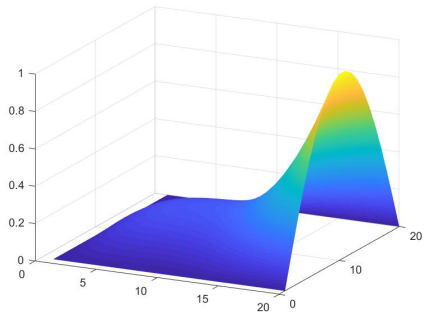


Fig. 15. The Numerical Solution using the our deep learning method, with $A_1(x, y)$, for the Example 4.

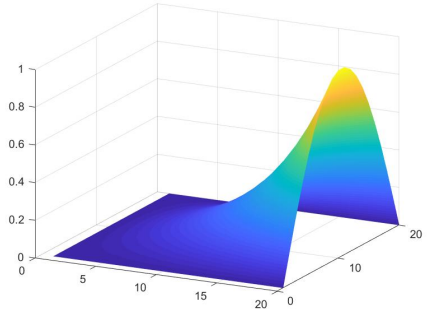


Fig. 16. The Numerical Solution using the our deep learning method, with $A_2(x, y)$, for the Example 4..

## VII. Acknowledgement

This work was supported by the European Union's Horizon 2020 Programme H2020-MSCA-RISE-2017, under the project PDE-GIR with grant number 778035.

## References

[1] R.J. Blakely, Potential Theory in Gravity and Magnetic Applications, Cambridge University Press, (2010).

[2] M. Kazhdan, M. Bolitho and H. Hoppe, Poisson Surface Reconstruction, in Eurographics Symposium on Geometry Processing, (2006).

[3] P. Pérez, M. Gangnet and A. Blake, Poisson Image Editing, ACM Transactions on Graphics, 22(3), 313-318, (2003).

[4] E. Chaudhry, S. J. Bian, H. Ugail, X. Jin and L. You, Dynamic Skin Deformation using Finite Difference Solutions for Character Animation, Computers and Graphics, 46(1), 294-305, (2015).

[5] N. Ahmat, G. González-Castro and H. Ugail, Automatic Shape Optimisation of Pharmaceutical Tablets using Partial Differential Equations, Computers and Structures, 130 1–9, (2013).

[6] H. Ugail, Spine Based Shape Parameterisaion for PDE Surfaces, Journal of Computing, 72 195-206, (2004).

[7] Q. Shen, Y. Sheng, C. Chen, G. Zhang and H Ugail, A PDE Patch-Based Spectral Method for Progressive Mesh Compression and Mesh Denoising, The Visual Computer, 34(11), 1563-1577, (2018).

[8] H. Ugail and N.B. Ismail, Method of Modelling Facial Action Units using Partial Differential Equations, in Kawulok M, Celebi E M and Smolka B (Eds.), Advances in Face Detection and Facial Image Analysis, 129-143, Springer, (2016).

[9] A. Arnal, J. Monterde and H. Ugail, Explicit Polynomial Solutions of Fourth Order Linear Elliptic Partial Differential Equations for Boundary Based Smooth Surface Generation, Computer Aided Geometric Design, 28(6), 382-394, (2011).

[10] M. Andersson, R. Sigurdsson and M. Passare, Analytic Solutions to Partial Differential Equations, in Complex Convexity and Analytic Functionals, Progress in Mathematics, 225, 129-150, (2004).

[11] A. Iserles, A First Course in the Numerical Analysis of Differential Equations, Cambridge University Press, (2008).

[12] E.F. Toro, Riemann Solvers and Numerical Methods for Fluid Dynamics, Springer-Verlag, (1999).

[13] E.F. Anley, Numerical Solutions of Elliptic Partial Differential Equations by Using Finite Volume Method, Pure and Applied Mathematics Journal, 5(4), 120-129, (2015).

[14] A. M. Bukar and H. Ugail, Automatic age estimation from facial profile view, IET Computer Vision, 11(8), 650-655, (2017).

[15] H. Ugail and A. Al-dahoud, Is gender encoded in the smile? A computational framework for the analysis of the smile driven dynamic face for gender recognition, The Visual Computer, 34(9), 1243–1254, (2018).

[16] A. Elmahmudi and H. Ugail, Deep face recognition using imperfect facial data, Future Generation Computer Systems, 19, 213-225, (2019).

[17] A. Elmahmudi and H. Ugail, Experiments on Deep Face Recognition Using Partial Faces, 2018 International Conference on Cyberworlds (CW), Singapore, pp. 357-362, (2018).

[18] A. Elmahmudi and H. Ugail, The Biharmonic Eigenface, Signal, Image and Video Processing, https://doi.org/10.1007/s11760-019-01514-4, (2019).

[19] L. Jianyu, L. Siwei, Q. Yingjian, H. Yaping, Numerical Solution of differential equations by radial basis function neural networks. Proc. Int. Jt Conf. Neural Netw. 1, 773–777 (2002)

[20] N. Mai-Duy, T. Tran-Cong, Numerical solution of differential equations using multiquadric radial basis function networks. Neural Netw. 14, 185–199 (2001)

[21] L.O. Chua, L. Yang, Cellular neural networks: theory. IEEE Trans. Circuits Syst. 35, 1257–1272 (1988)