SOLUTION TO DIFFUSION EQUATION USING NEURAL NETWORKS

Esther Ugochukwu Amaka Okolie

Contents

1 Introduction

Almost all physical process and systems ranging from chemical reactions to fluid flow and heat transfer, are dependent on more than one variable; for instance, the diffusion process is dependent on space (position) and time and can be represented using partial differential equations (PDEs). Some of the resulting PDEs do not readily have an analytical solution, thus we used numerical methods to solve them. These approaches, such as the finite difference methods, aid in the transformation of PDEs into linear systems that may be solved using single or multistep methods such as Euler's and Runge Kutta methods. With the recent development of Machine Learning and its applications, neural networks have shown to be efficient in applied research. For example, neural networks have been used to derive numerical solutions to PDEs such as the Navier Stoke Equation and others with excellent accuracy, sometimes outperforming traditional numerical methods.

2 Background Knowledge

The general 1D Diffusion equation is given by

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

where c is the diffusion co-efficient such that $x \in [0, L]$ with initial condition $U(x, 0) = f(x) \;\; \forall x \in [0, L]$ and Dirichlet boundary conditions $u(0, t) = u(L, t) = 0 \;\; \forall t > 0$ In a discretized mesh space, we use the explicit forward finite difference scheme in to transform the PDE into

$$\frac{U_i^{j+1} - u_i^j}{\Delta t} = c(\frac{u_{i+1}^j - 2u_i^j + U_{i-1}^j}{\Delta x^2})$$

with $\alpha = \dfrac{c\Delta t}{\Delta x^2}$ we have

$$U_i^{j+1} = (1 - 2\alpha)u_i^j + \alpha(u_{i+1}^j + u_{i-1})$$

Similarly, the backward difference scheme gives

$$\frac{u_i^n - u_i^{n-1}}{\Delta t} = \frac{\alpha(u_{i+1}^n - 2u_i^n + u_{i-1}^n)}{\Delta x^2}$$

Hence these can be written as

$$AU = b$$

when $U = (U_0^n, ..., U_t^n)$
And A is a sparse tridiagonal matrix When

$$b = \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_N \end{bmatrix} \textbf{ where } \;\; b_0 = 0 \;\;\; b_i = U_i^{n-1} \;\;\; b_N = 0 \;\;\; \textbf{and} \;\;\; i = 1, \cdots, N-1$$

These numerical approach produce a good accuracy and are generally stable, but neural networks are more accurate and stable. Neural networks will be discussed more explicitly in the next section.

3 Neural Networks Description and Implementation

Neural networks are designed such that interconnected nodes or neurons learn in a way similar to the brain. The learning process of the neural network can be divided into two processes known as the forward propagation and backward propagation. During the backward pass, the network adjusts its weights and biases by minimizing the error using the cost function.

PDEs can be generally written as

$$f(x_1, x_2, \cdots, x_N, \frac{\partial u}{\partial x_1}, \frac{\partial u}{\partial x_2}, \cdots, \frac{\partial u}{\partial x_N}, \frac{\partial^2 u}{\partial x_1 \partial X_2}, \cdots, \frac{\partial^n g(x_1, \cdots x_n)}{\partial x_N^n}) = 0$$

We must have a trial solution for x, a very handy trial solution can be

$$u(x_1, x_2, \cdots, x_N) = h_1(x_1, \cdots, x_N) + h_2(x_1, \cdots, x_N, N(x_1, \cdots, x_N, P))$$

Where h_1 is a function that ensures U satisfies the conditions, $N(x_1, \cdots, x_N, P)$ represents the neural network with set of weights and biases of P, h_2 ensures that $N(x_1, x_2, \cdots, x_n, P)$ goes to zero when $u(x_1, x_2, \cdots, x_n)$ is evaluated. The cost function which can be used to minimize the error is given by squaring the function f

$$C(x, p) = \sum_{i=1}^{N} \left(f(x_i, \frac{\partial u}{\partial x_1}, \frac{\partial u}{\partial x_2}, \cdots, \frac{\partial u}{\partial x_N}, \frac{\partial^2 u}{\partial x_1 \partial x_2}, \cdots, \frac{\partial^n u}{\partial x_N^n})\right)^2$$

where $x_i = (x_1, \cdots, x_N)$
Having this, we can build a simple neural network with N inputs and N hidden neutrons and N outputs neurons.A simple implementation can be found in `https://github.com/etthereo/Diffusion_Equation_with_Neural_Network/blob/main/Diffusion_Equations.ipynb` and the result can be found at `https://github.com/etthereo/Diffusion_Equation_with_Neural_Network/blob/main/Neural_Network_Result.csv`

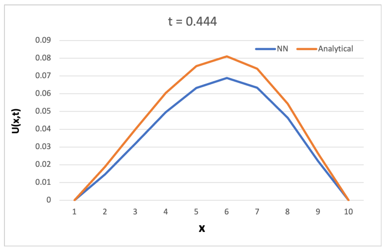
4 Result and Conclusion

Let us consider a case of the diffusion equation, where we want to find $g(x, t)$ such that

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

Where $u(0, t) = 0, t \leq 0$ $u(1, t) = 0, t \leq 0$ $u(x, 0) = \sin(\pi x)$ $x \in [0, 1]$ using the implementation discussed in section 3, we have the result below
The Analytical solution to this diffusion is given by $u(x, t) = e^{-\pi^2 t} \sin \pi x$

The graph below shows a good comparison of the neural network solution and the analytical solution.



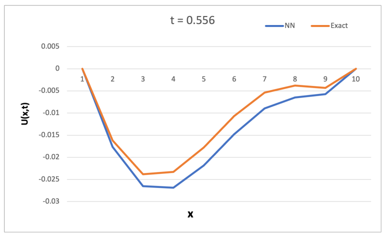
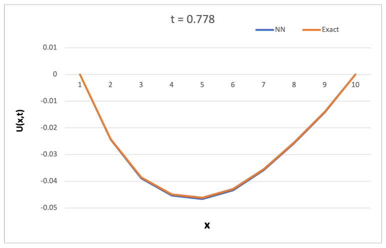


Figure 1: Exact Solution and Neural Network Solution at time $t = 0.45$ and $t = 0.56$



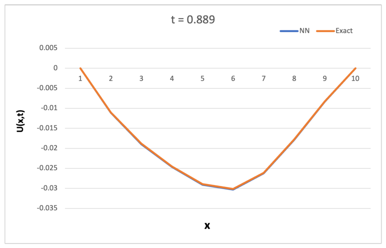


Figure 2: Exact Solution and Neural Network Solution at time $t = 0.78$ and $t = 0.89$

We can see that from the result, the neural network methods have values very close to the analytical solution as the time increases. These results can be improved by adding more hidden layers and tuning some hyperparamters such as the learning rate, batch size and so on.

References

[1] Morten Hjorth-Jensen. Deep Learning: Solvind differential equations with neural networks. In *https://compphysics.github.io/CompSciProgram/doc/pub/week6/html/week6-bs.html*, 2023.

[2] Github.com. *https://compphysics.github.io/CompSciProgram/doc/pub/week6/html/week6-bs.html*, 2023.