

# Solving the Heat Equation with Neural Networks

Alexander Havrilla & Alden Pritchard  
Carnegie Mellon University  
Pittsburgh, PA 15213  
`alumhavr@andrew.cmu.edu`; `atpritch@andrew.cmu.edu`

December 20, 2020

## 1 Introduction

The Deep Galerkin Method was first proposed in 2018 by Justin Sirignano and Konstantinos Spiliopoulos, who used the method to numerically solve the Black-Scholes partial differential equation (PDE) for options pricing. This main advantage of the DGM is that it is mesh-free, and as a result does not suffer from the curse of dimensionality, allowing us to compute values in feasible time for higher-dimensional setups. Sirignano and Spiliopoulos proved rigorously that using a deep neural network, as the number of layers goes to infinity, the network approximation converges point wise to the true solution. This is done by choosing a loss function which consists of distance from some initial condition, distance from some boundary condition, and distance from the size of the differential operator applied to the function. By minimizing the sum of these three terms, the method aims to generate a function which closely approximates the initial condition, boundary condition, and a solution on the interior of the domain, thereby giving a numerical solution to the PDE.

## 2 Background

General commentary

2, 3, 7, 4

### 2.1 PDEs and Heat Equation

Partial differential equations, or PDEs, define a subset of differential equations where derivatives with respect to different variables exist in the same equation. PDE's are commonly found in nature as they can describe many different kinds of diffusion processes. For example, the heat equation describes the diffusion of heat in a specified number of dimensions. In the one-dimensional case, there

does exist a closed-form analytical solution to the heat equation. However, in the 2-D and more general multivariate case, no analytical solution is known to exist. As a result, if we wish to know how heat will diffuse in a given area under some specified initial conditions, then we need to find an approximation to the heat equation.

## 2.2 Traditional Approximation Methods

Traditional PDE approximation methods involve creating a mesh over the domain of the problem. In the 1-D case, this amounts to breaking the domain interval into some set of  $n$  evenly-spaced points. At each point the PDE is approximated as an ODE with initial conditions satisfying the original PDE and then solved numerically at each point in time. When each of these discrete solutions is taken as a whole, we get an approximate solution to the PDE.

This process of meshing the domain from a continuous problem into a discrete set of problems and solving the individual ODEs requires a significant amount of both work and approximations, both of which contribute to making solving PDEs very numerically challenging. Since the number of mesh points increases exponentially in the dimension of the problem, it is necessary to use either a smaller domain or a larger distance between mesh points if we want to solve the PDE in a reasonable amount of time. However, as the distance between mesh points increases, the stability of the approximation breaks down. Likewise, restricting the size of the domain yields less information about PDE than we may want. This trade-off demonstrates the difficulty in numerically solving PDEs, as we are forced to choose between runtime and accuracy.

Some common methods for discretizing PDEs include Galerkin Methods, Finite Difference Methods, Finite Volume Methods, and Gradient Discretization Methods, among others. Galerkin Methods seek to approximate the PDE using a linear combination of basis functions. In this paper we use Finite Difference Methods for the 1-D case and Finite Volume Methods for the 2-D case in our approximations.


## 3 Related Work

Solving PDEs using the DGM architecture was originally proposed by Sirignano and Spiliopoulos in 2018. The authors realized that neural networks were similar in concept to Galerkin Methods through the use of linear combinations of basis functions to approximate some function. Whereas the Galerkin method uses a single linear combination of basis functions to approximate the PDE solution, the DGM uses multiple layers of basis functions. The weights for these functions are determined by training the neural network via stochastic gradient descent with backpropagation, adam optimizer, and momentum. For the loss function,

the authors propose using the sum of distances from the boundary, initial, and differential conditions to approximate the error in the solution. The theoretical inspiration for the DGM lies in its similarity to the Galerkin method. However, there is also a theoretical argument for why using a neural network might be effective in approximating PDEs. Specifically, Sirignano and Spiliopoulos proved that as the neural network depth goes to infinity, the neural network approximation converges point-wise to the PDE solution. This suggests that any PDE can be approximated with a neural network, provided we use the right hyper-parameters for the model. This parameter tuning was explored further by Al-Aradi and others in *Solving Nonlinear and High-Dimensional Partial Differential Equations via Deep Learning*.


Al-Aradi et al applied the DGM to solve various PDEs, including the Black-Scholes PDE for options pricing in finance, the Fokker-Planck PDE in statistical mechanics, Mean Field Equations for backward stochastic differential equations, and optimal control problems. Through fine-tuning model hyper-parameters, the authors were able to achieve similar performance to popular traditional methods on each PDE. This suggests that the DGM can be used for solving PDEs, and that the main obstacle is finding the right combination of parameters that best suits the model to the PDE.

### 3.1 DGM Architecture

 Architecture.png

The overall DGM network architecture consists of a fully connected layer, followed by some number of stacked DGM layers that take as input the original input  $x$  and the output of the previous DGM layer, with another fully connected layer at the end. We experimented with different depth networks in anticipation of a trade-off between training time and network accuracy as indicated by the results from Sirignano and Spiliopoulos.

### 3.2 Individual DGM layer

 DGM\_Layer.png

Each individual DGM layer consists of four sums of linear maps and an output function. In total, each layer contains eight weight matrices, four bias vectors, and four activation functions, which are eventually combined in the layer's output function.

## 4 Methods

General Commentary

- 4.1 Architecture**
- 4.2 Sampling Methodology**
- 4.3 Initial/Boundary Conditions**

## **5 Results**

General Commentary

- 5.1 Depth of Network**
- 5.2 Number of Samples**
- 5.3 Dependence on Dimension**
- 5.4 Dependence on Initial/Boundary Conditions**

## **6 Analysis**

- 6.1 Depth of Network**
- 6.2 Training Time**
- 6.3 Number of Samples**
- 6.4 Dependence on Dimension**
- 6.5 Dependence on Initial Conditions**

How these things affect time and accuracy?

## **7 Future Work**

Our DGM experiments involved varying network depth, number of nodes per layer, sampling distribution, sampling region, and number of points sampled. Future work could explore more variations such as deeper networks, networks with larger layers, as well as different sampling techniques that might help the network better understand the PDE at a given moment in time. With so many hyper parameters, there is considerable space for additional research to explore some heuristics that might lead to better performance.

One area we believe is particularly in need of additional research is the sampling problem. High dimensional sampling becomes very difficult because the space we sample from grows exponentially in size, meaning we need to use exponentially more points in order to achieve the same spatial coverage, which is important for training the network. One solution to this problem is to find the optimal distribution to sample from. In our experiments, we found that we achieved

much better results sampling from a Uniform distribution than from a Gaussian distribution. This seemed to be the case across all parameter setups, yet it is unclear why this might be true. Further research might explore which distributions are best suited to sampling tasks and whether some distributions are universally better than others for this task.

## References

References follow the acknowledgments. Use unnumbered first-level heading for the references. Any choice of citation style is acceptable as long as you are consistent. It is permissible to reduce the font size to `small` (9 point) when listing the references. **Remember that you can use a ninth page as long as it contains *only* cited references.**

[1] Al-Arabi A. Correia A. Naiff D. Jardim G. Solving Nonlinear High-Dimensional Partial Differential Equations via Deep Learning.

[2] Han. J. Solving High-dimensional PDEs Using Deep Learning. <https://www.pnas.org/content/pnas/115/34/8505.full>

[3] Penko V. <https://github.com/vitkarpenko/FEM-with-backward-Euler-for-the-heat-equation>

[4] Sirignano. J, spiliopoulos K. DGM: A deep learning algorithm for solving partial differential equations. <https://arxiv.org/abs/1708.07469>