Numerically Solving PDEs using Deep Neural Networks

Alex Havrilla & Alden Pritchard

November 2020
https://github.com/atapritchard/DPDEs

1 Abstract

Our goal is to numerically solve a specific class of quasi-linear Partial Differential Equations (PDEs) using a deep neural network through a process called the Deep Galerkin Method (DGM). In practice, it often takes significant effort to solve even a single PDE well using conventional methods. This is also true of the DGM approach, except that now the challenge is finding the right setup for the method. This means optimizing our choice of, among other things, the DGM model architecture, neural network hyper-parameters, and interior point sampling methods.

We will start by trying to solve the heat equation as this is a common PDE and will be a good proof of concept for us. Once we have done this, we will attempt to apply similar a methodology to harder equations (we will choose these later). If we can successfully apply our methodology to other PDEs, then this will also enable us to use DGMs to solve not just PDEs, but also arbitrary functions such as the Ray-Tracing equation(perhaps, this bears further thought).

2 Motivation

Standard methods for solving PDEs include Euler steps, Runge-Kutta, Galerkin methods, and others. These methods mesh the domain in which we want to solve the PDE and then take small steps to solve the PDE. However, as the dimension increases, the number of gridpoints and steps grows exponentially and stability decreases, making this process infeasible. This leaves us with a lack of good methods for solving PDEs in higher dimensional spaces.

3 Advantages of the Deep Galerkin Method

The Deep Galerkin Method was first proposed in 2018 by Justin Sirignano and Konstantinos Spiliopoulos. They used the DGM to numerically solve the Black-Scholes equation (a well-studied PDE that has many important uses in finance, but also has no analytical solution). This main advantage of this method is that it is mesh-free, and as a result does not suffer from the curse of dimensionality, which allows us to compute values in feasible time. These authors 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. In doing all of these, the output will successfully give a numerical solution to the PDE.

4 Implementation Details

4.1 Overall Architecture

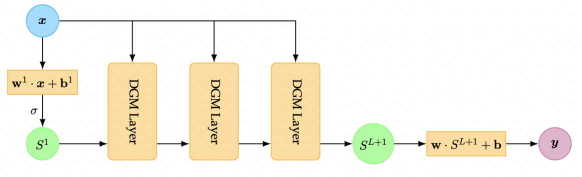
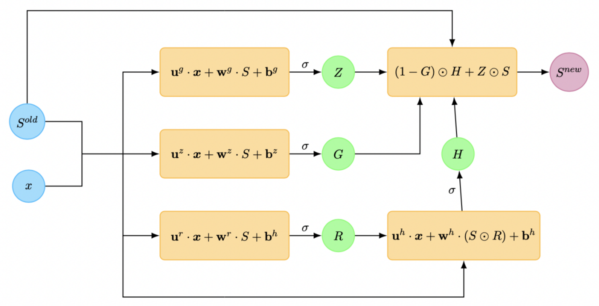

x
$\mathbf{w}^1 \cdot x + \mathbf{b}^1$
σ
S^1
DGM Layer
DGM Layer
DGM Layer
S^{L+1}
$\mathbf{w} \cdot S^{L+1} + \mathbf{b}$
y

Figure 5.2: Bird's-eye perspective of overall DGM architecture.

The overall architecture consists of stacked DGM layers that take as input the original x and the output of the previous DGM layer, as well as two linear layers, one at the start and one at the end.

4.2 Individual DGM layer



S^{old}
x
$\mathbf{u}^g \cdot \boldsymbol{x} + \mathbf{w}^g \cdot S + \mathbf{b}^g$
$\mathbf{u}^z \cdot \boldsymbol{x} + \mathbf{w}^z \cdot S + \mathbf{b}^z$
$\mathbf{u}^r \cdot \boldsymbol{x} + \mathbf{w}^r \cdot S + \mathbf{b}^h$
σ
Z
G
R
$(1 - G) \odot H + Z \odot S$
H
$\mathbf{u}^h \cdot \boldsymbol{x} + \mathbf{w}^h \cdot (S \odot R) + \mathbf{b}^h$
S^{new}

Figure 5.3: *Operations within a single DGM layer.*

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.

5 Room for Improvement

Some PDEs involve second or higher order derivatives. One way to compute these is to use backpropagation, but this is approach quadratic in the dimension of the input variables. The authors avoided using backpropagation by using a finite difference method. We believe there is room for improvement here in the finite difference approximation used. There are also questions of sampling methodologies and which distributions are most appropriate for each equation. There are also considerations regarding network structure and computation time where we anticipate being able to try several different approached. It should be noted that solving just one PDE well is valuable, and as a result, we believe finding a way to approximate even one PDE well using DGM will be a significant contribution.

6 Baseline

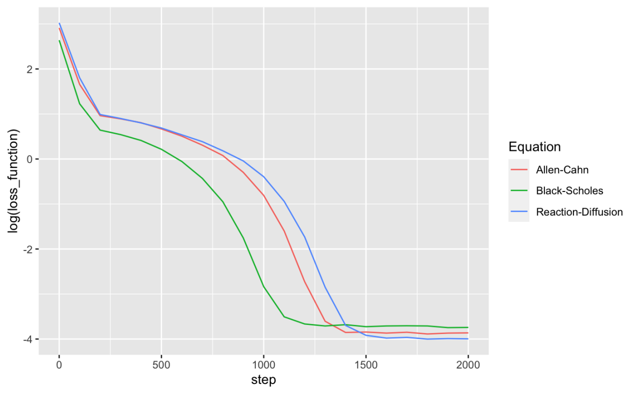


Fig 1: Loss Curves for Different PDEs

This graph shows the log-loss for the Deep Galerkin Method on the Allen-Cahn, Black-Scholes, and Reaction-Diffusion PDEs. Our goal is to produce similar results for the Heat Equation.

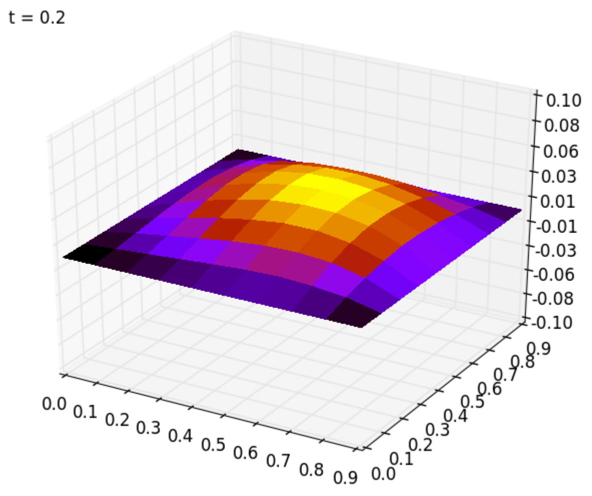


Fig 2: A solution to the Heat Equation at time t=.2

As a baseline for the heat equation we have a finite element method implementation(in 2 dimensions for concreteness). Ideally our implementation of a DGM network for the heat equation will provide a similar accuracy to the finite element method at a fraction of the speed in high dimensions.

7 Work Division

Task 1. Basic Research
Learning about PDEs and existing frameworks for numerically solving – Alex and Alden.

Task 2. Constructing a DGM Network
Base architecture – Alex
Hyperparameter optimization – Alden

3. Testing
Solving with existing methods – Alden

4. Application & Visualization
Creating visualizations to compare DGM performance to numerical approximations – Alex

References

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

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

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

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