Project Proposal

Jiajing Guan
Advisor: Howard Elman

University of Maryland

September 29, 2020

Overview

1. Problem Formulation
2. Method
3. Sample Problem
4. Specific Goals
5. Starting Point
6. Milestones

Problem of Interests

Motivation

Simulations of models defined as parameter-dependent PDEs.

We are interested in obtaining solutions u to the following problem many times with different $\boldsymbol{\mu}$:

$$u_t + \mathcal{N}[u; \boldsymbol{\mu}] = 0,$$

where $\boldsymbol{\mu}$ denotes parameters and $\mathcal{N}$ denotes a nonlinear differential operator.

Background

When explicit solutions are not readily available, we need to look for a surrogate function $\hat{u}(\boldsymbol{x}, t; \boldsymbol{\mu})$ that approximate solution $u(\boldsymbol{x}, t; \boldsymbol{\mu})$.

This leads to two phases of producing solutions:

1. Offline: train/fit the surrogate function $\hat{u}(\boldsymbol{x}, t; \boldsymbol{\mu})$.
2. Online: use $\hat{u}(\boldsymbol{x}, t; \boldsymbol{\mu})$ to produce approximations to $u(\boldsymbol{x}, t; \boldsymbol{\mu})$.

Background

Traditional approximation methods include:

- Finite Difference Method
- Finite Element Method
- Multigrid Methods
- etc

Even though these methods don't require an offline training process, these numerical schemes needed to be run for every set of parameters during online phase, i.e. traditional methods become expensive in this setting.

Background

Traditional approximation methods include:

- Finite Difference Method
- Finite Element Method
- Multigrid Methods
- etc

Even though these methods don't require an offline training process, these numerical schemes needed to be run for every set of parameters during online phase, i.e. traditional methods become expensive in this setting.

$\Longrightarrow$ **use neural networks to reduce the cost of computation.**

Comparison

<table>
<tr><th colspan="3">Comparison of the Online Phase</th></tr>
<tr><th></th><th>Traditional Methods</th><th>Neural Networks</th></tr>
<tr><td>Pro</td><td>well-studied accuracy of solution</td><td>speedy performance</td></tr>
<tr><td>Con</td><td>need to run the numerical scheme for each choice of parameters
solutions limited by grid positions</td><td>not thoroughly studied, unpredictable performance
limited accuracy</td></tr>
</table>

Problem Definition

Let $u(\boldsymbol{x}, t; \boldsymbol{\mu})$ be the exact solution to the following parameter-dependent PDE problem:

$$u_t + \mathcal{N}[u; \boldsymbol{\mu}] = 0,$$

where $\boldsymbol{\mu}$ denote parameters. Let $f(u(\boldsymbol{x}, t; \boldsymbol{\mu})) := u_t + \mathcal{N}[u; \boldsymbol{\mu}]$.

We would like to obtain approximations $u_{nn}(\boldsymbol{x}, t; \boldsymbol{\mu})$ produced by trained neural networks at discrete points $\{(\boldsymbol{x}_{test}, t_{test})^{(j)}\}_{j=1}^{N_{grid}}$ and set of parameters $\{\boldsymbol{\mu}_{test}^{(i)}\}_{i=1}^{N_{test}}$.

Project Definition

Goal

Studying existing machine learning methods that approximate solutions to parameter-dependent PDEs.

Existing Methods:

- Non-intrusive reduced order modeling of nonlinear problems using neural networks[1]
- Physics-Informed Neural Networks[2]

[1] J. Hesthaven and S. Ubbiali, "Non-intrusive reduced order modeling of nonlinear problems using neural networks," *Journal of Computational Physics*, vol. 363, pp. 55 –78, 2018, ISSN: 0021-9991.

[2] M. Raissi, P. Perdikaris, and G. Karniadakis, "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations," *Journal of Computational Physics*, vol. 378, pp. 686 –707, 2019, ISSN: 0021-9991.

POD-NN RB [1]

Procedure:

1. sample N_{train} number of parameters $\{\boldsymbol{\mu}_{train}^{(j)}\}_{j=1}^{N_{train}}$;
2. compute a collection of snapshots $\{\boldsymbol{u}_h(\boldsymbol{\mu}_{train}^{(j)})\}_{j=1}^{N_{train}}$ at $\{(\boldsymbol{x}_{test}, t_{test})^{(j)}\}_{j=1}^{N_{grid}}$ using traditional solvers such as FDM;
3. using proper orthogonal decomposition (POD) Galerkin Reduced Basis (RB) method, create change of basis matrix V from left singular vectors of snapshots. Need to choose dimension of reduced basis L.
4. produce sample outputs $\{V^T \boldsymbol{u}_h(\boldsymbol{\mu}_{train}^{(j)})\}_{j=1}^{N_{train}}$ and train neural network with loss function

$$L(\boldsymbol{\theta}) = \frac{1}{2N_{train}} \sum_{j=1}^{N_{train}} \left\| \boldsymbol{u}_{nn}(\boldsymbol{\mu}_{train}^{(j)}; \boldsymbol{\theta}) - V^T \boldsymbol{u}_h(\boldsymbol{\mu}_{train}^{(j)}) \right\|_2^2,$$

   where $\boldsymbol{u}_{nn}(\boldsymbol{\mu}; \boldsymbol{\theta})$ is output of the neural network given parameter $\boldsymbol{\mu}$ and weights $\boldsymbol{\theta}$.
5. produce approximations with parameter set $\{\boldsymbol{\mu}_{test}^{(i)}\}_{i=1}^{N_{test}}$ as $\{V \boldsymbol{u}_{nn}(\boldsymbol{\mu}_{test}^{(i)}; \boldsymbol{\theta})\}_{i=1}^{N_{test}}$.

PINN [2]

Procedure:

1. sample over initial and boundary training data as $\{(\boldsymbol{x}_{IB}, t_{IB}, \boldsymbol{\mu}_{IB}, u_{IB})^{(j)}\}_{j=1}^{N_i}$ and points in the domain as $\{(\boldsymbol{x}_F, t_F, \boldsymbol{\mu}_F)^{(z)}\}_{z=1}^{N_f}$.
2. train neural network with the following loss function:

$$L(\boldsymbol{\theta}) = \frac{1}{2N_i}\sum_{j=1}^{N_i}(u_{nn}(\boldsymbol{x}_{IB}^{(j)}, t_{IB}^{(j)}, \boldsymbol{\mu}_{IB}^{(j)}; \boldsymbol{\theta}) - u_{IB}^{(j)})^2 + \frac{1}{2N_f}\sum_{z=1}^{N_f}(f(u_{nn}(\boldsymbol{x}_F^{(z)}, t_F^{(z)}, \boldsymbol{\mu}_F^{(z)}; \boldsymbol{\theta})))^2$$

3. produce approximations at discrete points $\{(\boldsymbol{x}_{test}, t_{test})^{(j)}\}_{j=1}^{N_{grid}}$ and parameter set $\{\boldsymbol{\mu}_{test}^{(i)}\}_{i=1}^{N_{test}}$ as $\{\{u_{nn}(\boldsymbol{x}_{test}^{(j)}, t_{test}^{(j)}, \boldsymbol{\mu}_{test}^{(i)}; \boldsymbol{\theta})\}_{j=1}^{N_{grid}}\}_{i=1}^{N_{test}}$.

Error Evaluation

In order to quantitatively evaluate the performance of each method, we use the following metric:

- For POD-NN RB:

$$E_{rel} = \frac{1}{N_{test}} \sum_{i=1}^{N_{test}} \frac{\left\| V \boldsymbol{u}_{nn}(\boldsymbol{\mu}_{test}^{(i)}; \boldsymbol{\theta}) - \boldsymbol{u}^*(\boldsymbol{\mu}_{test}^{(i)}) \right\|_2}{\left\| \boldsymbol{u}^*(\boldsymbol{\mu}_{test}^{(i)}) \right\|_2},$$

  where $\boldsymbol{u}^*(\boldsymbol{\mu})$ is the true solution with parameter $\boldsymbol{\mu}$ evaluated at $\{(\boldsymbol{x}_{test}, t_{test})^{(j)}\}_{j=1}^{N_{grid}}$.

- For PINN:

$$E_{rel} = \frac{1}{N_{test}} \sum_{i=1}^{N_{test}} \frac{\sqrt{\sum_{j=1}^{N_{grid}} (u_{nn}(\boldsymbol{x}_{test}^{(j)}, t_{test}^{(j)}, \boldsymbol{\mu}_{test}^{(i)}; \boldsymbol{\theta}) - u^*(\boldsymbol{x}_{test}^{(j)}, t_{test}^{(j)}, \boldsymbol{\mu}_{test}^{(i)}))^2}}{\sqrt{\sum_{j=1}^{N_{grid}} (u^*(\boldsymbol{x}_{test}^{(j)}, t_{test}^{(j)}, \boldsymbol{\mu}_{test}^{(i)}))^2}}$$

  where $u^*(\boldsymbol{x}, t, \boldsymbol{\mu})$ is the true solution evaluated at $(\boldsymbol{x}, t)$ with parameter $\boldsymbol{\mu}$.

Comparison between two methods

Comparison between POD-NN RB and PINN		
	POD-NN RB	PINN
Benefits	• Smaller dimension of sample inputs • Simpler loss function	• No need to produce snapshots • Approximations can be done on flexible $(\boldsymbol{x}, t)$

Sample Problem

Suppose we are approximating the solutions to the following problem:

$$
\begin{aligned}
-\xi u'' + u' &= 0 \quad \text{for } x \in (0,1) \\
u(0) &= 1 - e^{-1/\xi} \\
u(1) &= 0
\end{aligned}
$$

Here, the parameter is $\xi = 10^a$, where a is chosen from a uniform distribution of $[-4, 0]$.

We could compute the exact analytical solution to be $u(x) = 1 - e^{(x-1)/\xi}$.

Solutions to Sample Problem for Various ξ

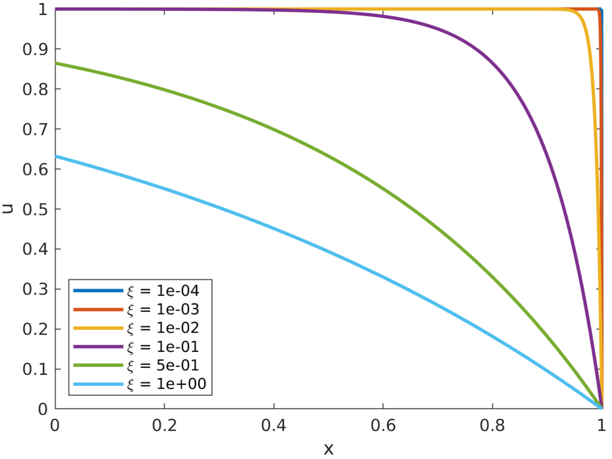

Figure: This figure shows how solutions change as ξ increases from 10^{-4} to 1.

Approximation Results

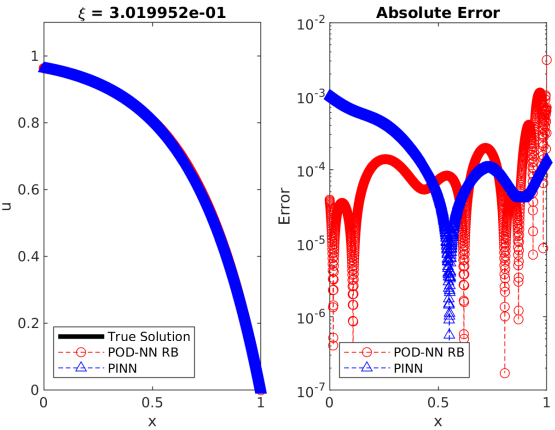

Figure: This figure shows approximations done by POD-NN RB and PINN when $\xi \approx 0.3$.

Approximation Results

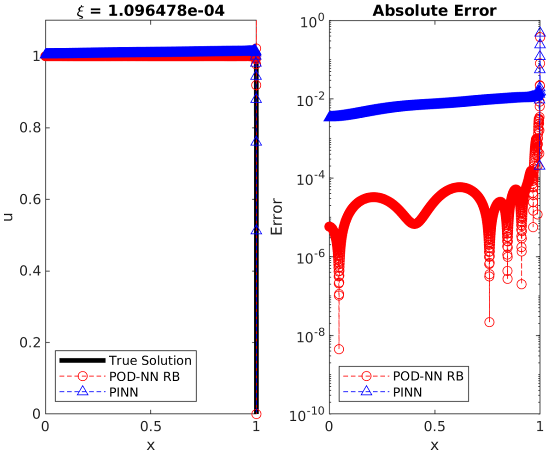

Figure: This figure shows approximations done by POD-NN RB and PINN when $\xi \approx 10^{-4}$.

Approximation Details

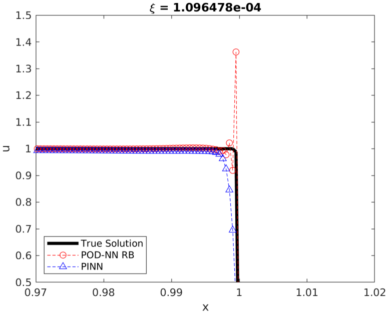

Figure: This figure shows the details of the approximation near $x = 1$ when $\xi \approx 10^{-4}$.

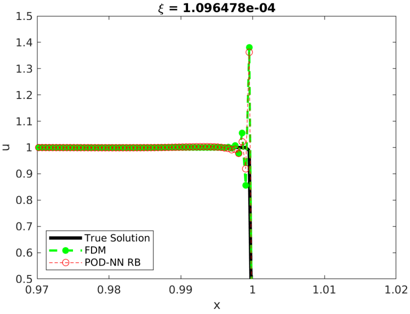

Figure: This figure shows the finite difference method approximations along with the true solution.

Setup Comparison

	POD-NN RB ($L = 25$)	PINN
Network	2 hidden layers with 32 neurons per layer	2 hidden layers with 32 neurons per layer
Optimization	234 epochs of Levenberg-Marquardt	1000 epochs of Levenberg-Marquardt
Training Time (s)	≈ 120	≈ 20000
Relative Error	1.005×10^{-3}	3.958×10^{-3}

Gaps

What hasn't been established:

- POD-NN RB:
  - performance of deeper networks
  - performance of ResNet and other network structures
- PINN:
  - performance of ResNet and other network structures
  - application on parameter-dependent PDEs
  - performance of Levenberg-Marquardt optimization

Specific Goals

Investigate the impact of different factors on performance. The factors include but are not limited to:

- Network Structures
- Network Setups
- Sampling Methods
- Optimization Methods
- Types of Problems

Details

Network Structures:

- Dense Neural Networks
- ResNet
- Recurrent Networks (LSTM)

Network Setups:

- Number of Hidden Layers
- Number of Neurons per Layer

Sampling Methods:

- Latin-Hypercube
- Domain-Specific

Details

Optimization Methods:

- Gradient-Based Methods (SGD)[3]
- Quasi-Newton Methods (L-BFGS)[4]
- Levenberg-Marquardt

Types of Problems:

- Unsteady Burger's Equations
- Nonlinear Diffusion Equation
- Convection-Diffusion Equations

[3] L. Bottou, F. E. Curtis, and J. Nocedal, "Optimization methods for large-scale machine learning," *SIAM Review*, vol. 60, no. 2, pp. 223–311, 2018.

[4] M. T. Hagan and M. B. Menhaj, "Training feedforward networks with the marquardt algorithm," *IEEE Transactions on Neural Networks*, vol. 5, no. 6, pp. 989–993, 1994.

Type of Problems in Details

- Unsteady Burger's Equations

$$\begin{cases} u_t + uu_x = \nu u_{xx}, & \text{for } x \in [-1, 1],\ t \in [0, 1] \\ u(0, x) = -\sin(\pi x), & \\ u(t, -1) = u(t, 1) = 0 & \end{cases},$$

where $\nu = 10^p$, where p is sampled on an uniform distribution of $[-4, 0]$.

Type of Problems in Details

- Nonlinear Diffusion Equation

$$\begin{cases} -(exp(u(x;\boldsymbol{\mu}))u(x;\boldsymbol{\mu})')' = s(x;\boldsymbol{\mu}), & \text{for } x \in (-\frac{\pi}{2}, \frac{\pi}{2}) \\ u(\pm\pi/2;\boldsymbol{\mu}) = \mu_2 \sin(2 \pm \frac{\mu_1 \pi}{2}) exp(\pm \frac{\mu_3 \pi}{2}) \end{cases}$$

where $\boldsymbol{\mu} = (\mu_1, \mu_2, \mu_3)$ are sampled on uniform distribution of $[1,3] \times [1,3] \times [-0.5, 0.5]$ and

$$\begin{aligned} s(x;\boldsymbol{\mu}) = & -\mu_2 exp(\mu_2 \sin(2+\mu_1 x) exp(\mu_3 x) + \mu_3 x) \\ & * [2\mu_1\mu_3 \cos(2+\mu_1 x) + (\mu_3^2 - \mu_1^2)\sin(2+\mu_1 x) \\ & + exp(\mu_3 x)[\mu_1 \cos(2+\mu_1 x) + \mu_3 \sin(2+\mu_1 x)]^2] \end{aligned}$$

Here, $s(x;\boldsymbol{\mu})$ is calculated such that the exact solution is

$$u_{ex}(x;\boldsymbol{\mu}) = \mu_2 \sin(2+\mu_1 x) exp(\mu_3 x)$$

Type of Problems in Details

- Convection-Diffusion Equations

$$\begin{cases} -\xi_0 \Delta u + [(\xi_1 \vec{\omega_1} + \xi_2 \vec{\omega_2})/(\xi_1 + \xi_2)] \cdot \nabla u = 0 & \text{for } (x, y) \in (-1, 1) \times (-1, 1) \\ u(-1, y) = \xi_1 (1 - ((1 + y)/2))^3 / (\xi_1 + \xi_2) & \text{for } y \in [-1, 1] \\ u(1, y) = (\xi_1 (1 - ((1 + y)/2))^2 + \xi_2)/(\xi_1 + \xi_2) & \text{for } y \in [-1, 1] \\ u(x, -1) = \xi_1 / (\xi_1 + \xi_2) & \text{for } x \in (-1, 1) \\ u_n(x, 1) = 0 & \text{for } x \in (-1, 1) \end{cases}$$

where $\vec{\omega_1} = (0, 1 + \frac{(x+1)^2}{4})$, $\vec{\omega_2} = (2y(1 - x^2), -2x(1 - y^2))$.

Parameters ξ_1 and ξ_2 are sampled on an uniform distribution of $[0, 1]$, $\xi_0 = 10^p$, where p is sampled on an uniform distribution of $[-4, 0]$.

Starting Point

Done:

- Implementation of dense neural network
- Implementation of Levenberg-Marquardt
- Implementation of problems
    - include functions that produce discrete approximations obtained by FDM

To do:

- Implement and test ResNet and recurrent networks
- Implement and test different optimization methods (SGD, L-BFGS)
- Investigate better sampling method

All implementation are done on Python3, utilizing Tensorflow.

Validation

We will have two approaches to validate our implementations:

- Reproduce results from [1] and [2].
- Using exact solution of simple problems to validate the implementation.
- When exact solution is not available, compare results with approximation obtained by traditional approaches (FDM, FEM, etc).

Deliverables

- Implementation of POD-NN RB and PINN
- Implementation of dense net, ResNet, LSTM
- Implementation of various optimization methods
- Report comparing performance of the different frameworks above

Milestones

- End of September: ResNet and optimization methods implemented and network setup comparisons done.
- End of October: Recurrent networks implemented and starting comparing the different network structures.
- End of November: More effective sampling method found and figuring out minimal setup to achieve satisfatory performance on problems.

References

[1] J. Hesthaven and S. Ubbiali, "Non-intrusive reduced order modeling of nonlinear problems using neural networks," *Journal of Computational Physics*, vol. 363, pp. 55 –78, 2018, ISSN: 0021-9991.

[2] M. Raissi, P. Perdikaris, and G. Karniadakis, "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations," *Journal of Computational Physics*, vol. 378, pp. 686 –707, 2019, ISSN: 0021-9991.

[3] L. Bottou, F. E. Curtis, and J. Nocedal, "Optimization methods for large-scale machine learning," *SIAM Review*, vol. 60, no. 2, pp. 223–311, 2018.

[4] M. T. Hagan and M. B. Menhaj, "Training feedforward networks with the marquardt algorithm," *IEEE Transactions on Neural Networks*, vol. 5, no. 6, pp. 989–993, 1994.