Deep Learning Architectures for the 1D Burgers' Equation: Evaluating PINN and CNN Methods

Presented by : Cheng Li

Mentors: Dr. Hsien Shang, Dr. Somdeb Bandopadhyay

August 29, 2023

Introduction

Deep Learning for PDEs

- ➔ Deep Learning has shown promise in solving PDEs using data-driven methods.
- ➔ Neural networks can approximate complex functions, enabling the solution of PDEs in scenarios where traditional numerical methods face challenges.

1D Burgers' Equation

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = \nu\frac{\partial^2 u}{\partial x^2}$$

- ➔ The 1D Burgers' Equation is a fundamental partial differential equation used to model various physical phenomena, including fluid dynamics and shock waves.
- ➔ The equation combines convection and diffusion terms, making it nonlinear in nature and also has exact solution for specific cases

Motivation

- ➔ Physics-Informed Neural Networks (PINNs) and Convolutional Neural Networks (CNNs) are two popular approaches for data-driven analysis.
- ➔ This presentation aims to evaluates the effectiveness of PINNs and CNNs in solving the 1D Burgers' Equation.
- ➔ We also focus on evaluating the presence of bias in our sample used for training the NN
- ➔ In essence this work is an initiative towards solving PDEs governing real world applications using NN

Objective

- ➔ Use CNN to approximate the solution of the Burgers equation for the prediction of fluid dynamics.
- ➔ Use biased samples and compare the results of the biased and unbiased samples.

Key Concept of the Training Models

- ➔ **By providing data such as sampling points (x,t), initial points, boundary points**
- ➔ **Different parts of the loss function guide the model**
- ➔ **Perform the optimization process**

$$u(x,t) = \frac{x}{t+1} \div \left(1 + \sqrt{\frac{t+1}{t_0}} \cdot \exp\left(\frac{R_{\text{num}} \cdot x^2}{4 \cdot t + 4}\right)\right)$$

```
def exact_solution(Rnum, data_points):
    x = data_points[:, 0]
    t = data_points[:, 1]
    t0 = np.exp(Rnum / 8.0)
    return (x / (t + 1)) / (1.0 + np.sqrt((t + 1) / t0) * np.exp(Rnum * (x * x) / (4.0 * t + 4)))
```

Convolutional Neural Network (CNN) : Literature

1) Raissi et al. (2017) [arXiv:1711.10561] : Proposed using CNNs along with physics-based loss functions to solve nonlinear PDEs.

2) Han et al. (2018) [JCP, 399:108929] : Demonstrated using CNNs to approximate solutions of many-body Schrodinger equation for quantum systems. Achieved good accuracy compared to exact methods.

3) Zhu et al. (2018) [JCP,394:56-81] : Used CNNs with physics-based constraints for uncertainty quantification in problems with high-dimensional random inputs. Applied to stochastic PDEs.

4) Sun et al. (2020) [arXiv preprint arXiv:2006.15063] : Proposed a physics-informed CNN to learn mappings from boundary conditions to velocity fields for fluid flow without simulation data.

5) Jin et al (2021) [arXiv preprint arXiv:2104.09406] : Presented an unsupervised learning framework using CNN autoencoders and GANs to model fluid flows by learning from visual data.

Convolutional Neural Network (CNN)

- ➔ CNNs have convolutional layers that act as filters to extract features from input images.
- ➔ These convolutional layers detect locally connected features using learnable kernels that are convolved across the input.
- ➔ CNNs have pooling layers that downsample the feature maps to reduce dimensions and computational load.
- ➔ Multiple convolutional and pooling layers extract hierarchical features from low-level edges to high-level shapes.
- ➔ Fully connected layers at the end act as classifiers by mapping features to output categories based on learned weights.

Summary →

1. Convolutional layers detect local features
2. Pooling layers downsample feature maps
3. Multiple layers extract hierarchical features
4. Fully connected layers classify based on learned weights

Physics Informed Neural Network (PINN) : Literature

1) Raissi et al. (2019) [JCP 378:686-707] : Introduced PINN framework and demonstrated applications in solving various nonlinear PDEs forward and inverse problems.

2) Zhu et al. (2019) [JCP, 394: 56-81] :Proposed using PINN for high-dimensional surrogate modeling and uncertainty quantification in problems with stochastic inputs but no labeled data.

3) Sun et al. (2020) [Computer Methods in Applied Mechanics and Engineering, 361 (2020): 112732] : Proposed PINN model incorporating physics constraints to learn mappings from boundary conditions to velocity fields for fluid flows without simulation data.

4) Wang et al. (2020) [Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pp. 1457-1466. 2020] : Demonstrated convolutional PINN model for predictive modeling of turbulent flows using physics-informed loss functions.

5) Jin et al. (2021) [JCP, 426: 109951] : Presented PINN model called NSFnets to solve incompressible Navier-Stokes equations and predict various canonical turbulent flows.

Physics Informed Neural Network (PINN)

- ➔ PINNs are neural networks that incorporate physics knowledge like PDEs into the loss function.
- ➔ The loss function penalizes violations of governing equations and boundary/initial conditions.
- ➔ Automatic differentiation is used to compute residuals of PDEs during training.
- ➔ PINN is trained to minimize losses from data mismatch, boundary conditions, and PDE residuals.
- ➔ Once trained, PINN can rapidly predict solutions by forward pass, without solving PDEs explicitly.

Summary →

1. Loss function encodes physics knowledge
2. Automatic differentiation for PDE residuals
3. Minimizes losses from data, boundary conditions, PDEs
4. Rapid prediction after training, no explicit PDE solving
5. Has potential to replaces traditional numerical PDE solvers

Code Overview

```python
import numpy as np
import torch
import torch.nn as nn
import matplotlib.pyplot as plt
from torch.autograd import grad
import csv
import pandas as pd
from shapely.geometry import Point, Polygon

def generate_input_points(num_x_points=128, num_t_points=100):
    ...

def generate_sample_points(input_points, num_samples=100, bias_regions=None):
    ...

def generate_boundary_conditions(Rnum, num_bc_points):
    ...

def exact_solution(Rnum, data_points):
    ...

def plot_contour(xx, tt, data, title=''):
    ...

def plot_losses_from_file(filename):
    ...

def plot_lines(x, data, labels, title=''):
    ...

```

```python
class CNN_1DBurgers(nn.Module):
    ...
```

```python
class PINN_1DBurgers(nn.Module):
    ...

def get_optimizer(name, parameters, learning_rate):
    ...

def get_loss(name):
    ...

def get_scheduler(optimizer, scheduler_params):
    ...

def train_step(model, optimizer, loss_name, x_train, y_train, \
               input_points, initial_points, initial_data, boundary_points, boundary_data):
    ...

def train_model(model, num_epochs, optimizer_name, loss_name, learning_rate,
                input_points, sample_points, sample_data, initial_points,
                initial_data, boundary_points, boundary_data, scheduler_params):
    ..

def main():
    ...

if __name__ == "__main__":
    main()
```

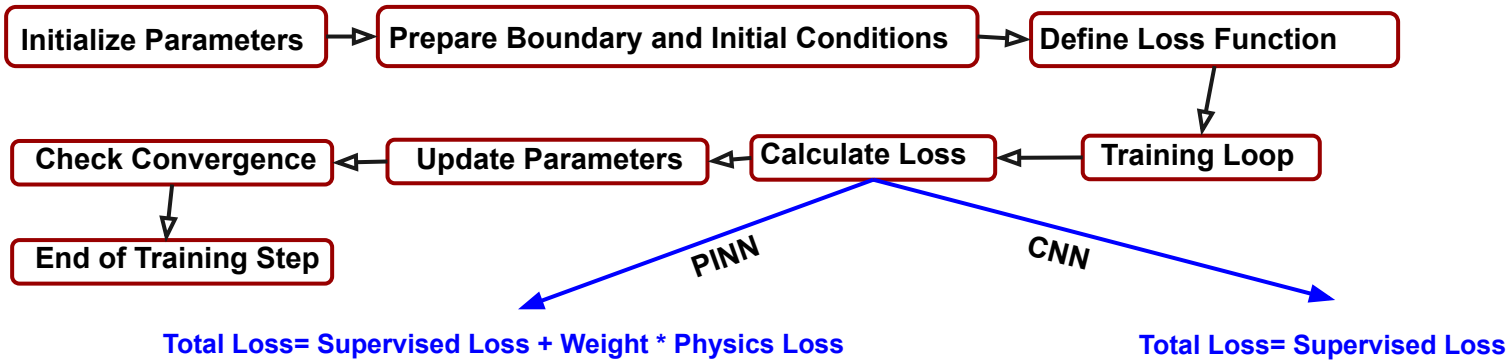
A Generic Approach for Training Model

1) **Initialize Optimizer and Scheduler:**
    a) **Initialize an optimizer using the specified optimizer name, model parameters, and learning rate.**
    b) **Initialize a learning rate scheduler using the specified scheduler parameters.**
2) **Initialize Tracking Variables:**
    a) **Initialize best-loss with a value of positive infinity.**
    b) **Set the maximum number of consecutive epochs without loss improvement to trigger early stopping.**
    c) **Initialize variable stop_iter as zero to track consecutive epochs without loss improvement.**
3) **Batching and Iterating Through Epochs:**
    a) **Calculate the number of iterations needed per epoch based on the batch size and sample points.**
    b) **Epoch Loop:**
        i) For each epoch, execute some specific steps (to be discussed later)
        ii) Continue to the next epoch until the specified number of epochs is reached.
4) **End of Training:**
    a) **The train_model function completes after all epochs.**
    b) **The losses are logged in the CSV file.**

Training Model : Epoch Loop

1) Initialize variables for loss calculation(for PINN this includes total-loss, l2-loss, pde-residual, boundary-loss and initial-condition-loss while for CNN we have total loss)
2) Batch Iteration: For each batch in the epoch, do the following:
    a) Training Step and Loss Calculation:
        i) Call the train_step function to perform a single training step.
        ii) Calculate loss terms
        iii) Perform backpropagation to update the model's weights based on the computed losses.
    b) Loss Accumulation for Epoch: Accumulate various loss terms (L2 loss, PDE residual loss, etc. for PINN and total loss for CNN) for the current epoch.
    c) Check for Loss Improvement:
        i) If the total loss for the current epoch is lower than best_loss, update best_loss with the new value.
        ii) Reset stop_iter to zero to track consecutive epochs without loss improvement.
    d) Check for Early Stopping: If stop_iter exceeds max_stop_iter :- exit the loop.
3) Learning Rate Scheduling: Call the scheduler's step() method to adjust the learning rate according to the chosen strategy.
4) Post-Processing :
    a) Log Losses: Write the losses for the current epoch to a CSV file named "losses.csv".
    b) Print Loss Updates: Every 5 epochs, print the losses for a on-the-fly recording

A Comparative Overview of Training Methods



Supervised Loss

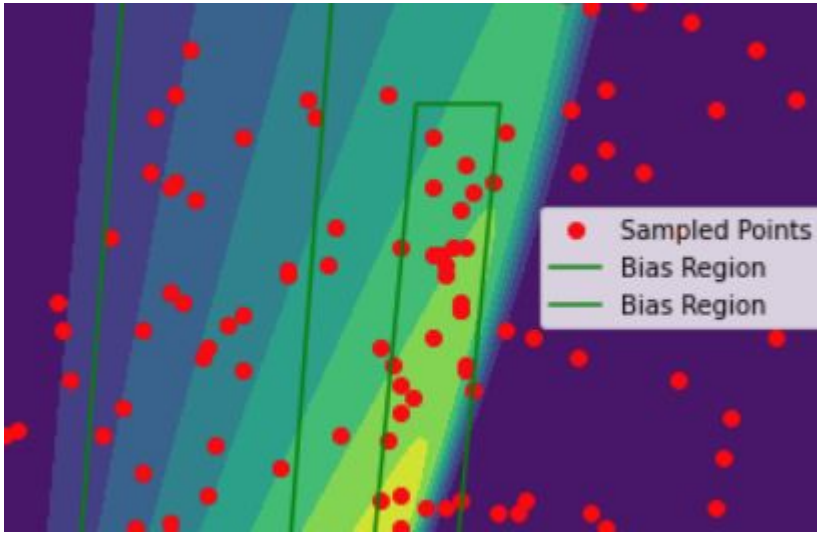
1) Measures the discrepancy between the predicted solution at the sampled points and the true solution.
2) It's a standard regression loss, such as mean squared error (MSE) or L1 loss.

Physics Loss

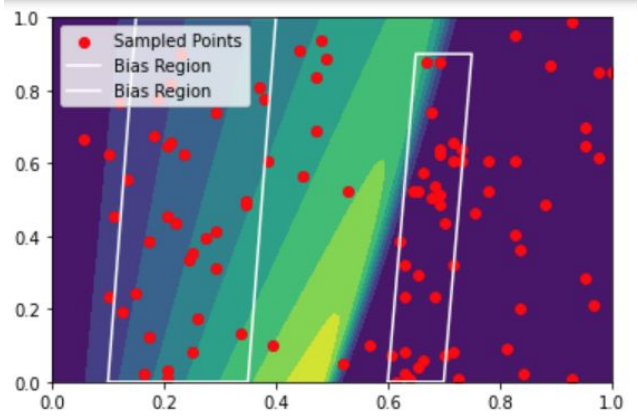
1) Enforces the physics constraints by penalizing the deviation of the neural network's predictions from the PDE
2) Computed using the PDE itself and involves partial derivatives of the neural network's output wrt input variables.

Biased Samples VS Unbiased Samples

- ➔ **better capture the complexity or interesting dynamics**
- ➔ **allocate more training points in specific regions**
- ➔ **challenging for neural networks to learn the properties without sufficient samples**



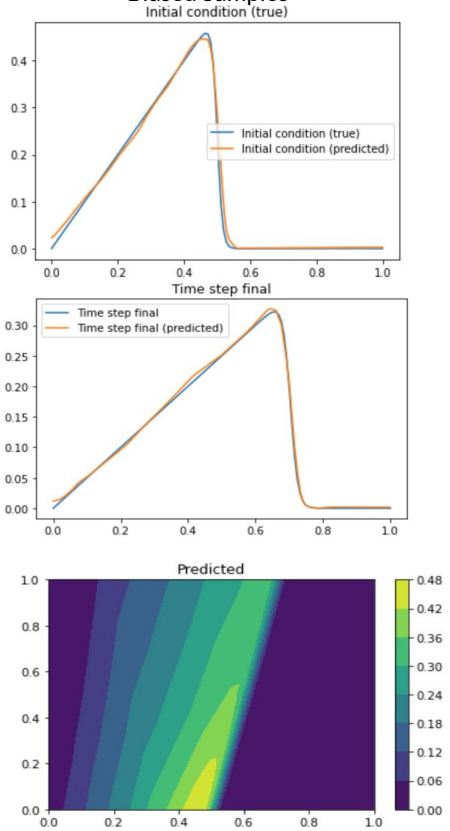
Sample Results (CNN)



CNN:

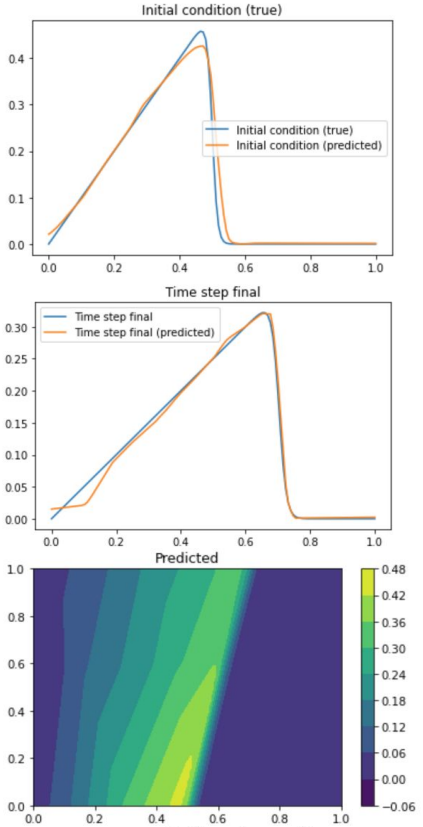
bias_regions = [([(0.1, 0.0), (.15, 1.0), (0.4, 1.0), (0.35, 0.0)], 30),
([(0.45, 0.0), (0.55, 0.90), (0.75,0.90),(0.65,0.0)], 30)]
hidden_sizes =[16, 32, 32, 64, 32, 32]
best_loss = 90.0
max_stop_iter = 90
Rnum = 500.0
num_epochs = 15000
learning_rate = 0.001

Biased Samples



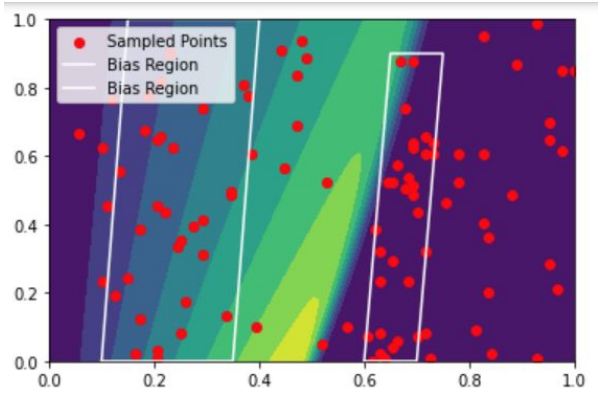
Losses for Epoch [735/15000], Loss: 0.0002)

Unbiased Samples



Losses for Epoch [205/15000], Loss: 0.0014)

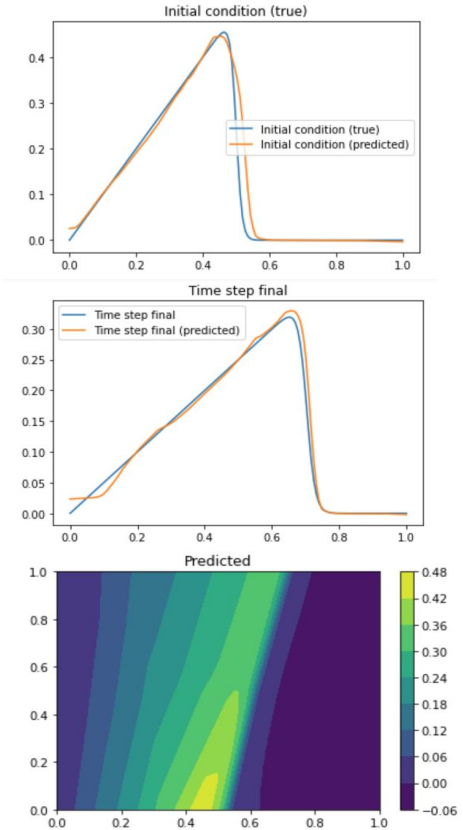
Sample Results (CNN)



CNN:

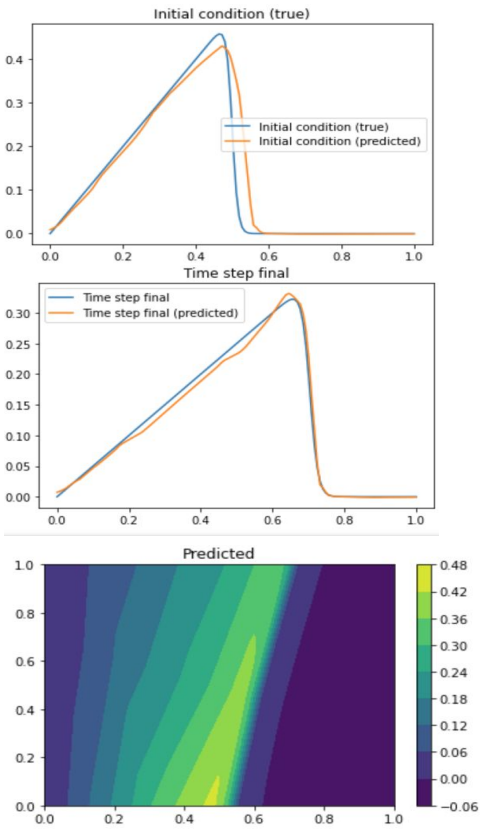
bias_regions = [([(0.1, 0.0), (.15, 0.9), (0.4, 0.9), (0.35, 0.0)], 30),
([(0.45, 0.0), (0.55, 0.95), (0.75,0.95),(0.65,0.0)], 30)]
hidden_sizes =[16, 32, 64, 128, 32, 32] (change neurons)
best_loss = 90.0
max_stop_iter = 90
Rnum = 500.0
num_epochs = 15000
learning_rate = 0.001

Biased Samples



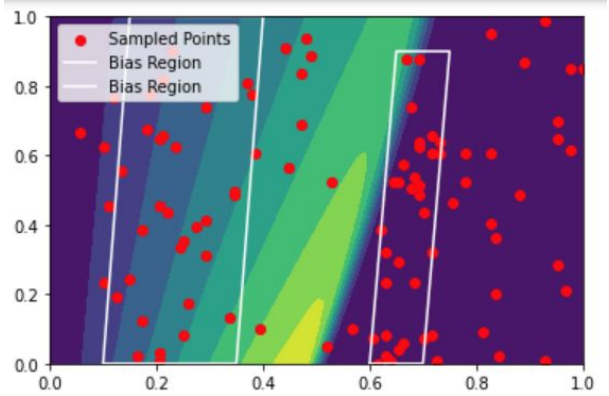
Losses for Epoch [395/15000], Loss: 0.0001)

Un based Samples



Losses for Epoch [230/15000], Loss: 0.0002)

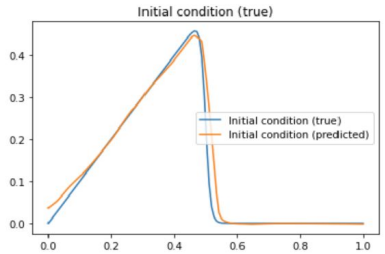
Sample Results (CNN)

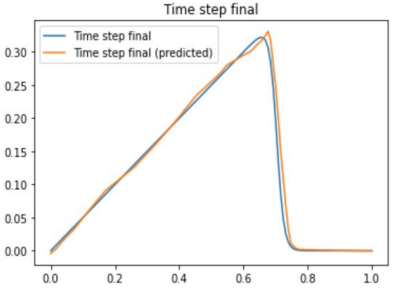


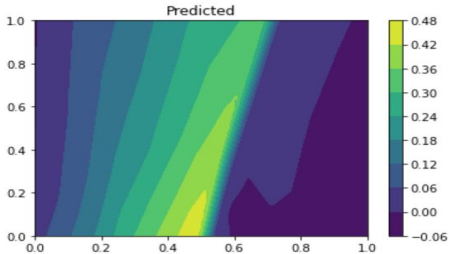
CNN:

bias_regions = [([(0.1, 0.0), (.15, 0.9), (0.4, 0.9), (0.35, 0.0)], 30),
([(0.45, 0.0), (0.55, 0.95), (0.75,0.95),(0.65,0.0)], 30)]
hidden_sizes =[16, 32, 32, 64, 32] (change hidden layers)
best_loss = 90.0
max_stop_iter = 90
Rnum = 500.0
num_epochs = 15000
learning_rate = 0.001

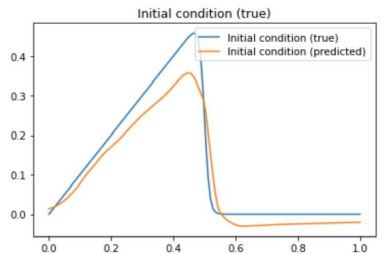
Biased Samples

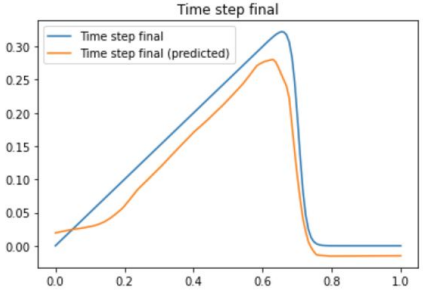


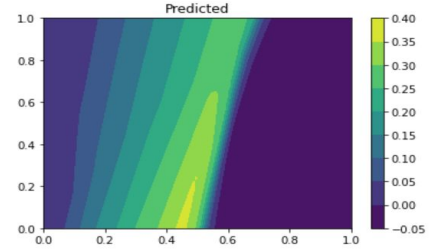




Unbiased Samples

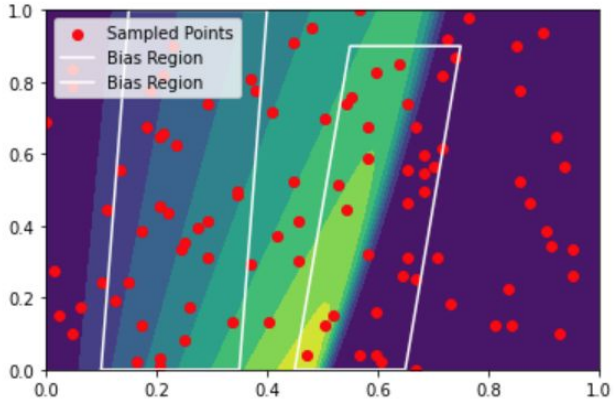






Losses for Epoch [675/15000], Loss: 0.0000) Losses for Epoch [470/15000], Loss: 0.0192)

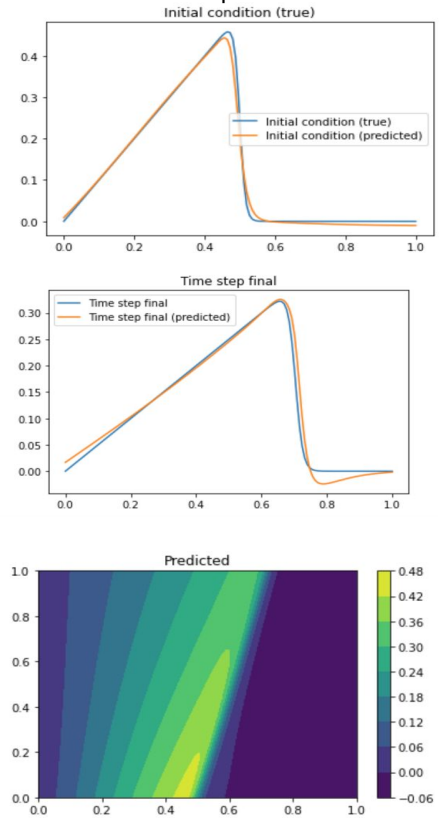
Sample Results (PINN)



PINN:

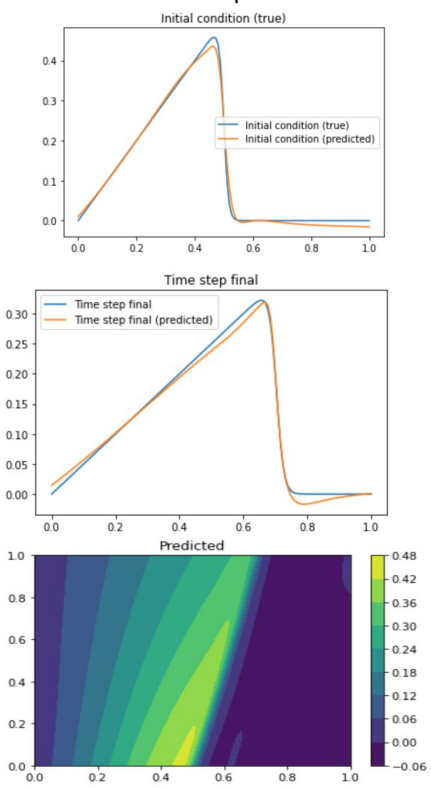
bias_regions = [([(0.1, 0.0), (.15, 1.0), (0.5, 1.0), (0.35, 0.0)], 30),
([(0.45, 0.0), (0.55, 0.90), (0.75,0.90),(0.65,0.0)], 30)]
num_epochs = 15000
learning_rate = 0.001
Rnum = 500.0
hidden_sizes = [40,40,40,40]
max_stop_iter = 90
stop_iter = 0
batch_size = 20

Biased Samples



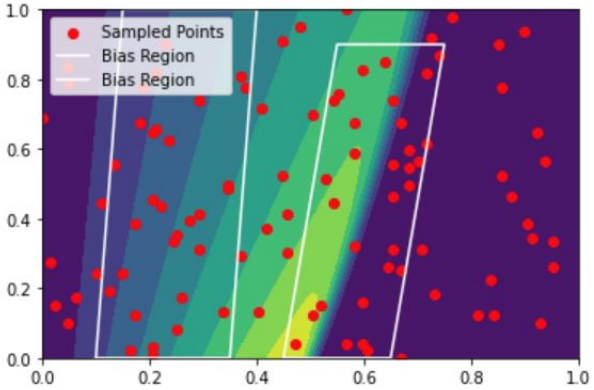
Losses for Epoch [900/15000], L2: 0.0002, PDE: 0.0005, IC: 0.0002, BC: 0.0001
Early stopping at epoch: 903

Unbiased Samples



Losses for Epoch [795/15000], L2: 0.0002, PDE: 0.0005, IC: 0.0002, BC: 0.0002
Early stopping at epoch: 797

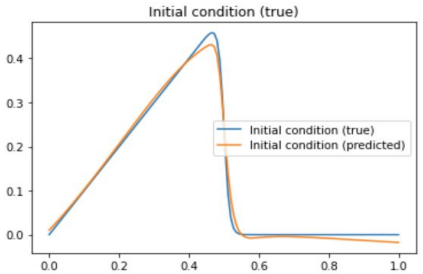
Sample Results (PINN)

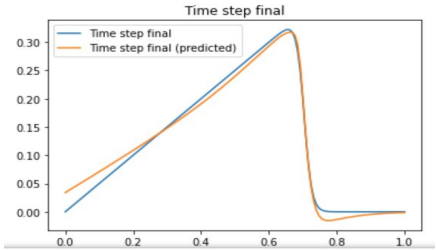


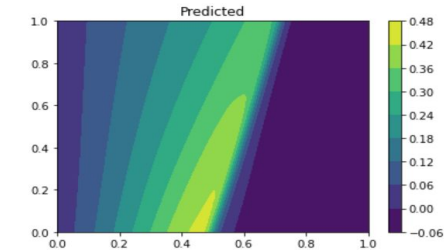
PINN:

bias_regions = [([(0.1, 0.0), (.15, 1.0), (0.4, 1.0), (0.35, 0.0)], 30),
([(0.45, 0.0), (0.5, 0.8), (0.60, 0.8),(0.55, 0.0)], 30)]
num_epochs = 15000
learning_rate = 0.001
Rnum = 500.0
hidden_sizes = [40,45,45,40] (change neurons)
max_stop_iter = 90
stop_iter = 0
batch_size = 20

Biased Samples

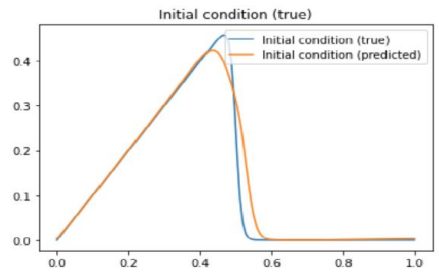


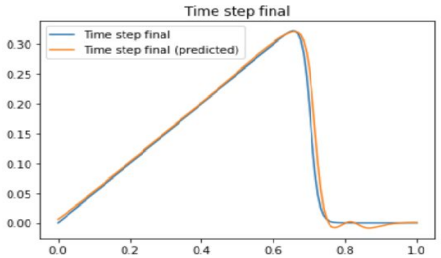


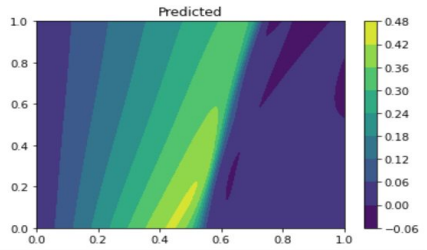


Losses for Epoch [680/15000], L2: 0.0003, PDE: 0.0005, IC: 0.0002, BC: 0.0002
Early stopping at epoch: 681

Unbiased Samples

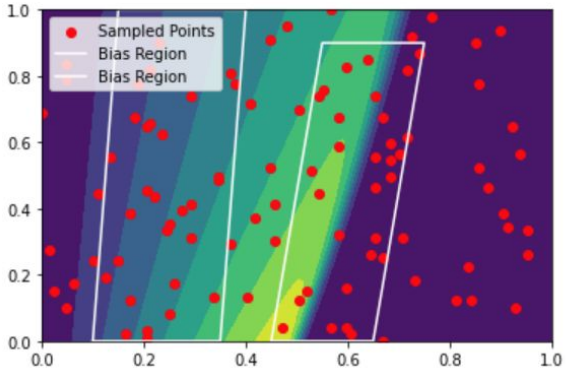






Losses for Epoch [955/15000], L2: 0.0003, PDE: 0.0001, IC: 0.0001, BC: 0.0001

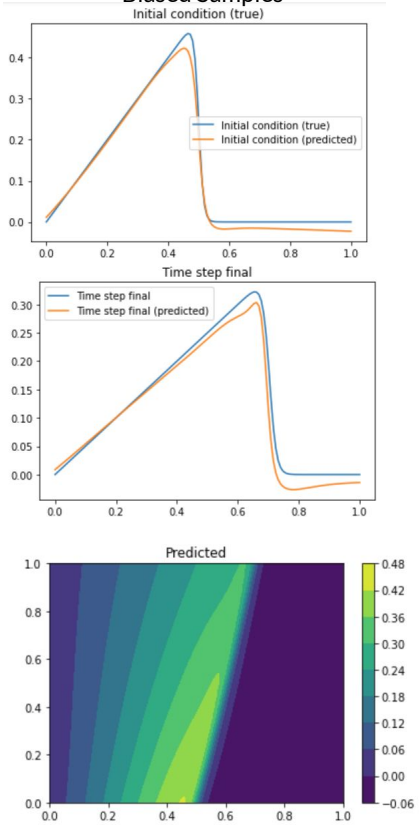
Sample Results (PINN)



PINN:

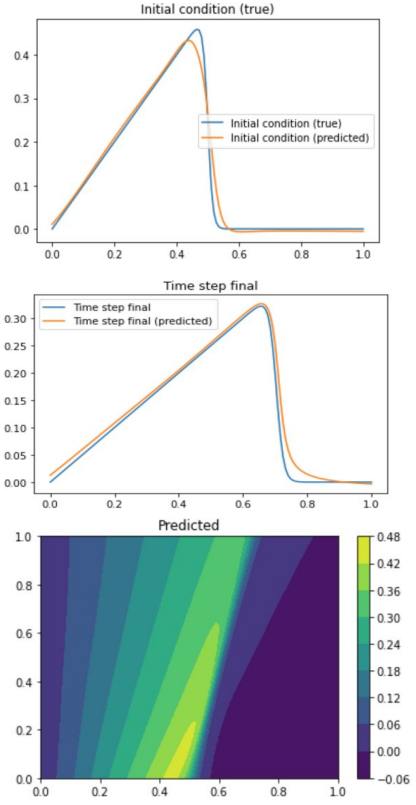
bias_regions = [([(0.1, 0.0), (.15, 1.0), (0.5, 1.0), (0.35, 0.0)], 30),
([(0.45, 0.0), (0.55, 0.90), (0.75,0.90),(0.65,0.0)], 30)]
num_epochs = 15000
learning_rate = 0.001
Rnum = 500.0
hidden_sizes = [40,40,40,40,40] (change hidden layers)
max_stop_iter = 90
stop_iter = 0
batch_size = 20

Biased Samples



Losses for Epoch [610/15000], L2: 0.0006, PDE: 0.0005, IC: 0.0003, BC: 0.0002
Early stopping at epoch: 614

Unbiased Samples



Losses for Epoch [770/15000], L2: 0.0027, PDE: 0.0001, IC: 0.0006, BC: 0.0005

Discussion & Remarks

- ➔ **The result shows that there are demonstrable effect of sample-bias in the NN models**
- ➔ **The effect of very primitive sample bias is more prominent in the CNN models compared to PINN models**
- ➔ **It seems like, selection of sample bias is easier for CNN over PINN. This can be explained by following the loss function calculation in PINN and hence, the bias that we selected needs to be picked after thorough consideration of all different losses instead of just focusing on the nonlinearity of the problem itself**

- ➔ **In future, we need to go through a more rigorous experiments of bias regions**
- ➔
- ➔ **We should also check the effect of the % of the bias per region**
- ➔ **CNN seems to be less effective for nonlinear PDEs and as such focus needs to be given on PINN more**

Thanks!