**PINN for Optimal Control Documentation**:

This document consists of key results that have been obtained during our experimentation with various models of PINN (physically informed neural networks).

# Progress so Far:

## Vanilla Implementation:

The simplest implementation of PINN for solving Berger’s 1D equation

Tried out:

* Varying the number of hidden layers and the number of neurons per layer
* various activation functions (sigmoid, **tanh**, ReLU, LeakyReLU): Tanh works best
* Optimizers: ADAM, **LBFGS**, Genetic Algorithms- LBFGS works the best
* Number of epochs for training

## Adaptive Training:

Implemented self-supervised training, i.e. adaptive sampling of collocation points based on:

(i) gradient and (ii) residual at the grid points

Tried out:

* Probability mass function based on simple division - with cosine annealing
* pmf using softmax function - and the effect of temperature
  + softmax function with annealing and without annealing
* Partial training with ADAM optimizer and the remaining with LBFGS
* The number of collocation points vs achieved loss

Vanilla Implementation:

| Uniform Initialisation  8 layers: 20 each  No of epochs: 10000  Training error: 6.93e-6  Test error: 6.86e-3 |
| --- |

| Xavier Weight Initialisation:   * Activation Function : Tanh * Training error : 2.08e-06 * Test error : 1.369e-03 |
| --- |

| Dropout:   * P = 0.05: training died after around 130 epochs * Training error : 0.1692 * Test error : 0.6515   Overall dropout is not useful |
| --- |

**Key Observations**:

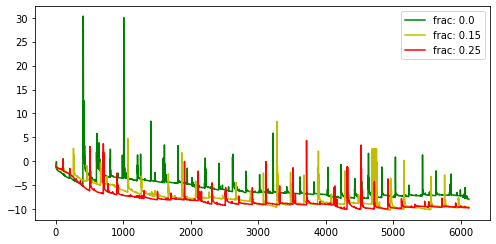
1. With ReLU and Sigmoid, training converges very early and therefore the model works extremely poor. Tanh seems to be working the best
2. Dropout for regularisation doesn’t seem to provide good training
3. LBFGS is the best optimizer so far for training (with adam and genetic algorithms, the training converges very quickly)

Adaptive Training:

| Probability function: simple division – using **residual** as proxy  Nf = 1000  Num\_epochs = 10000  Iter 8048, Loss: 9.61087e-05, Loss\_u: 2.47903e-05, Loss\_f: 7.13183e-05  CPU times: user 2min 33s, sys: 2.62 s, total: 2min 36s  Wall time: 2min 34s  Test error: Error u: 2.920005e-02    Probability function: simple division – using **gradient** as proxy  Nf = 1000  Num\_epochs = 10000  Iter 7976, Loss: 1.93796e-04, Loss\_u: 7.01633e-05, Loss\_f: 1.23633e-04  CPU times: user 2min 32s, sys: 2.62 s, total: 2min 35s  Wall time: 2min 35s  Test error: Error u: 2.049676e-02    **Uniform Sampling**  Nf = 1000  Num\_epochs = 10000  Iter 8047, Loss: 4.18818e-06, Loss\_u: 6.65361e-07, Loss\_f: 3.52282e-06  CPU times: user 2min 25s, sys: 2.89 s, total: 2min 28s  Wall time: 2min 26s  Error u: 2.825430e-01 |
| --- |

note: error obtained may vary everytime we run, but essentially it is of the order of 1e-02

## ADAM + LBFGS (with adaptive training)



## Softmax function for probabilities:

Softmax with cosine annealing:

Temperature = 20

| Iter 7893, Loss: 1.50004e-05, Loss\_u: 1.21376e-06, Loss\_f: 1.37867e-05  CPU times: user 2min 38s, sys: 2.78 s, total: 2min 41s  Wall time: 2min 39s  Error u: 9.446155e-03 |
| --- |

| Iter 7892, Loss: 1.50353e-05, Loss\_u: 3.45056e-06, Loss\_f: 1.15847e-05  CPU times: user 2min 39s, sys: 2.76 s, total: 2min 42s  Wall time: 2min 41s  Error u: 3.502753e-02 |
| --- |

Softmax without annealing:

Tried running several times, but same result:

| Temperature = 20  Iter 2024, Loss: 1.32538e+16, Loss\_u: 3.95233e+03, Loss\_f: 1.32538e+16  Training stopped!!! |
| --- |

| Temperature = 200  Iter 7895, Loss: 2.16511e-05, Loss\_u: 3.51170e-06, Loss\_f: 1.81393e-05  CPU times: user 2min 54s, sys: 2.77 s, total: 2min 57s  Wall time: 2min 55s  Error u: 4.868308e-02 |
| --- |

| Temperature = 100  Iter 879, Loss: 2.52207e+14, Loss\_u: 1.09691e+04, Loss\_f: 2.52207e+14  Iter 880, Loss: nan, Loss\_u: nan, Loss\_f: nan |
| --- |

| Temperature = 150  Iter 7827, Loss: 3.12462e-05, Loss\_u: 3.48002e-06, Loss\_f: 2.77661e-05  CPU times: user 2min 52s, sys: 2.8 s, total: 2min 54s  Wall time: 2min 52s  Error u: 6.156201e-03  Take2:  Iter 2013, Loss: 1.12617e+21, Loss\_u: 2.91748e+04, Loss\_f: 1.12617e+21  Iter 2014, Loss: nan, Loss\_u: nan, Loss\_f: nan |
| --- |

Observations:

1. For high temperatures, the results are good, comparable to that with annealing and without softmax
2. ‘Nan’ values are observed at lower epochs for low temperatures.
   1. Possible explanation is that for low lower epochs the model is not well trained and a lower temperature gives a complete outlier, for which the model gives large errors and therefore gradients explode
3. The accuracy is approx the same as that with cosine annealing. But in cosine annealing, the possibility for gradients to explode is less!
4. Training is quite temperature sensitive

Problems Faced:

1. Even with lbfgs we are sometimes getting exploding gradients: getting ‘nan’ values for loss
2. With the softmax function, we are converging to nan values much faster
   1. Possible explanation:

Whenever there is an entirely new set of points, it may lead to a huge error, therefore the gradients are more (concave upwards) and so it explodes!!!