Designing Next-Generation Numerical Methods with Physics-Informed Neural Networks

Stefano Markidis

KTH Royal Institute of Technology
Talk Outline

1. How does a Physics-Informed Neural Network (PINN) Solver Work?
2. Optimization of PINN Solvers
3. Integrating PINN into Traditional Solvers
1.1 Physics-Informed Neural Networks = PINN

- PINNs have many usages:
  - data assimilation
  - uncertainty quantification
  - solving ill-defined problems (e.g., no BC or EoS)

- In this talk, I focus on PINN for solving Partial Differential Equations (PDE)
1.2 Neural Network for Solving 2D Poisson Equation

\[ \nabla^2 u(x, y) = f(x, y), (x, y) \in [0, 1] \times [0, 1] \]

How would you do it?

Let’s assume you have the analytical or the numerical solution ...
1.3 Replace a Solver with a Neural Network

\[ \tilde{u}(x,y) \]

Approximated Solution at \((x,y)\)
1.4 Training with Analytical Solution

It works if we have the analytical solution:

$$L = \frac{1}{n} \sum (\bar{u}(x,y) - u(x,y))^2$$

Sample of collocation Points (training set)
It works if we the numerical solution

\[ L = \frac{1}{n} \sum (\bar{u}(x_i, y_i) - u(x_i, y_i))^2 \]

Sample of collocation Points (training set)
1.6 Prediction with the Surrogate Model = PDE Solver

This PDE solver is gridless
1.7 Entering PINNs!

- PINNs are neural networks that encode the partial differential equations into a part of neural network, exclusively to calculate the loss function
  - We still use the surrogate to evaluate the solution!
- Two major innovations:
  1. Add a part of the network / graph to calculate the residual.
    1. This part encodes the PDE into the NN.
    2. Leverage automatic differentiation to calculate the derivatives on the network.

[https://github.com/maziarraissi/PINNs](https://github.com/maziarraissi/PINNs)
1.8 Adding a Residual Network to Calculate a Residual

\[ r = \nabla^2 \tilde{u}(x, y) - f(x, y) \quad \text{Residual function} \]

Loss function

\[ \text{MSE}_r = \frac{1}{N_{x_i, y_i}} \sum |r(x_i, y_i)|^2 \]
1.9 PINN does not Need Solutions for Training

No need for prior data (solutions )
→ unsupervised training
1.10 How do we Calculate the Derivative on the Network?

- We use a critical ML technology used in backpropagation
  - Automatic differentiation
  - Available in the TensorFlow and PyTorch

1.11 PINN Training Iteration

1. We train network first via iterations / epochs
2. We make a prediction/inference step on a grid or any point of interest
   ○ Remember that PINN are gridless

Sample of collocation Points (training set)

\[
MSE_r = \frac{1}{N_{x_i, y_i}} \sum |r(x_i, y_i)|^2
\]
For stability and convergence studies, we need to study how error changes. Possible to define errors (generalization, training, ...) and do an analytical study. It has shown that PINNs require sufficiently smooth activation functions for convergence: PINNs with ReLU, ELU and SELU do not converge.


1.13 What is the Performance of a Simple PINN?

\[ \nabla^2 u(x, y) = f(x, y) \]

\[ f(x, y) = \frac{1}{4} \sum_{k=1}^{4} (-1)^{k+1} 2k \sin(k\pi x) \sin(k\pi y) \]

- Python and DeepXDE
- Fully Connected
- 4 layers
- 50 units per layer
- tanh act. function
- 10,000 coll. points
- Adam + L-BFGS.B Optimizers
- PETSc CG took 92 seconds for full convergence on 128x128 grid!
2.1 Optimization of PINN Solvers

1. Activation functions / Adaptive Functions
2. Optimizers
3. Transfer-Learning
2.2 PINN Optimization – Activation Functions

Activation functions largely impacts the performance
Best activation function depends on the problem
LAAF activation functions introducing adaptive local scaling are best
  - Deal better with BCs

2.3 PINN Optimization – BFGS Optimizer

- In PINN two optimizers in succession
  1. Adam optimizer
  2. Broyden-Fletcher-Goldfarb-Shanno (BFGS) optimizer
     - Higher-order: BFGS uses the Hessian matrix (curvature in highly dimensional space)
     - Without using the Adam optimizer can rapidly converge to a local minimum!
       - For this reason, the Adam optimizer is used first to avoid local minima, and then the solution is refined by BFGS.

- BFGS is currently the most critical technology for PINNs as it provides much higher accuracy than available DL optimizers.
  - L-BFGS-B from in SciPy. Not available on GPUs.
  - New L-BFGS-B available in Google’s Tensorflow Probability Framework
    - Built on the top of TensorFlow
2.4 PINN Optimization – Transfer Learning

The transfer learning technique = training a network solving the Poisson equation with a different source term.

- Initialize the PINN network we intend to solve with the first fully trained network weights and biases → first PINN transfers the learned information.
3.1 Integration of PINNs into Traditional Solvers

- Even after these optimizations, PINN performance is not as good as traditional iterative solvers!
  - Especially when it comes to accuracy
- Idea: combine two approaches to get the best from the two world
  - What PINNs are good at?
3.2 DLN F-principle: Convergence of PINN on Large Scale Structures First!

**Frequency-principle (F-principle):** DNNs often fit target functions from low to high frequencies during the training process.

The F-principle implies that in PINNs, the low frequency/large scale features of the solution emerge first, while it will take several training epochs to recover high frequency/small-scale features.

\[
f(x, y) = \frac{1}{4} \sum_{k=1}^{4} (-1)^{k+1} 2k \sin(k\pi x) \sin(k\pi y)
\]

---

3.3 Traditional Jacobi and GS Solvers: Convergence on Small Scales First!

- Both the Jacobi and Gauss-Seidel methods show fast convergence for small-scale features
  - Update of unknown values involves only the values of the neighbor points
    \[
    u_{i,j}^{n+1} = \frac{1}{4}(u_{i+1,j}^n + u_{i-1,j}^n + u_{i,j+1}^n + u_{i,j-1}^n)
    \]
- Between two different iterations, the information can only propagate to neighbour cells
3.4 Combining Low Frequency and High Frequency Solvers in a Multigrid Solver

Basic Idea: optimized PINN for a coarse grid then use a MG solvers

3.5 Combining Low Frequency and High Frequency Solvers in a Multigrid Solver

1. Initialize DL Network Weights and Biases
2. Train with Adam Optimizer
   - 10 Iterations
   - Single Precision
3. Train with L-BFGS-B
   - Stopping criterium \((\text{fiol})\)
   - Single Precision
4. DL Solver Network
   - Single Precision
5. Predict on Coarse Grid
   - Single Precision
6. Refine Solution with Gauss-Seidel (GS) Method on Coarse Grid - Stopping criterium \((\delta)\)
   - Interpolation to Finer Grid
   - 1 GS Cycle per Fine Grid
   - Double Precision
3.6 Hybrid Solver - Accuracy

Direct PINN Error

Hybrid DL-solver

3.7 Hybrid Solver – Computational Performance

- Different resolutions, tolerances for hybrid, pure GS and PETSc CG
- Python implementations (CG step developed in Cython)

Conclusions

1. PINN are neural networks encoding PDEs in the network and they can be used for solving PDEs in an unsupervised fashion.

2. The PINN performance (both computational and accuracy) is still far from performance of traditional approaches, but optimization are possible: activation function tuning, high-order optimizers and transfer learning.

3. Combine traditional and PINN solver technology is a realistic approach for developing the next-generation Solvers.
   ○ We are still in the infancy of these methods: lot of work to do!