Residual-Based Adaptive Sampling for PINNs

Paper Reference

Title: A Comprehensive Study of Non-adaptive and Residual-based Adaptive Sampling for Physics-informed Neural Networks

Authors: Wu et al. (2022)

Journal: Computer Methods in Applied Mechanics and Engineering

DOI: 10.1016/j.cma.2022.115671

Code Repository: https://github.com/lu-group/pinn-sampling

Overview

This paper systematically compares **10 different sampling methods** for selecting collocation (residual) points in Physics-Informed Neural Networks (PINNs). The study includes over **6,000 PINN simulations** across 6 different PDE problems.

Sampling Methods

A. Non-Adaptive Uniform Sampling (6 methods)

These methods select points without considering the PDE residual:

Method	Description	Implementation Complexity
Grid	Regular equispaced grid points	Very Easy
Random	Uniformly random sampling	Very Easy
LHS	Latin Hypercube Sampling - stratified random	Easy (scipy)
Halton	Low-discrepancy quasi-random sequence	Easy (scipy)
Hammersley	Quasi-random sequence	Easy (scipy)
Sobol	Low-discrepancy sequence	Easy (scipy)

B. Uniform with Resampling (Random-R)

- Same as Random sampling, but **resample points every N iterations**
- The resampling period N is an important hyperparameter
- Special case of RAD with $k=0$ or $c\to\infty$

C. Adaptive Nonuniform Sampling (3 methods)

These methods use the **PDE residual** to guide point selection:

1. RAR-G (Residual-based Adaptive Refinement with Greed) - Algorithm 1

- Original method from Lu et al. (2019)
- After every certain number of iterations, **adds new points** only in locations with the **largest PDE residuals**
- Greedy approach - focuses only on maximum residual regions
- Points **accumulate** over training (dataset grows)
- Special case of RAR-D with $k \to \infty$

```
Algorithm 1: RAR-G
1. Initialize residual points S randomly
2. Train PINN for some iterations
3. Evaluate residual ε(x) on dense candidate set
4. Add points with largest residuals to S
5. Repeat from step 2
```

2. RAD (Residual-based Adaptive Distribution) - Algorithm 2

- **Resamples ALL points** according to a probability density function (PDF)
- Does NOT accumulate points - replaces entire training set
- PDF formula:

$$p(\mathbf{x}) \propto \frac{\varepsilon^k(\mathbf{x})}{\mathbb{E}[\varepsilon^k(\mathbf{x})]} + c$$

Where:

- $\varepsilon(\mathbf{x}) = |f(\mathbf{x}; \hat{u}(\mathbf{x}))|$ is the PDE residual
- $k \geq 0$ controls how much to focus on high-residual regions
- $c \geq 0$ ensures some uniform coverage (regularization)

Special cases:

Parameters	Equivalent Method
$k=0$ or $c \to \infty$	Random-R (uniform resampling)
$k=1$, $c=0$	Nabian et al.'s method
$k=2$, $c=0$	**Recommended default**

```
Algorithm 2: RAD
1. Initialize residual points S randomly
2. Train PINN for some iterations
3. Evaluate residual ε(x) on dense candidate set S₀
4. Compute probability p(x) ∝ εᵏ(x)/E[εᵏ(x)] + c
5. Resample ALL points from S₀ according to p(x)
6. Repeat from step 2
```

3. RAR-D (Residual-based Adaptive Refinement with Distribution) - Algorithm 3

- **Hybrid** of RAR-G and RAD
- **Repeatedly adds new points** (like RAR-G)
- But new points are **sampled according to the RAD PDF** (not just max residual)
- Points accumulate over training

Special cases:

Parameters	Equivalent Method
$k \to \infty$	RAR-G (greedy)
k=2, c=0	**Recommended default**

```
Algorithm 3: RAR-D
1. Initialize residual points S randomly
2. Train PINN for some iterations
3. Evaluate residual ε(x) on dense candidate set S₀
4. Compute probability p(x) ∝ εᵏ(x)/E[εᵏ(x)] + c
5. Sample M new points from S₀ according to p(x)
6. Add new points to S (S grows)
7. Repeat from step 2
```

Benchmark PDEs

The paper tests all methods on **6 PDEs** (4 forward + 2 inverse problems):

Forward Problems

Problem	PDE	Domain	Network	Optimizer
Diffusion	$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} + f(x,t)$	$x \in [-1,1], t \in [0,1]$	4×32	Adam
Burgers'	$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = \nu \frac{\partial^2 u}{\partial x^2}$	$x \in [-1,1], t \in [0,1]$	4×64	Adam + L-BFGS
Allen-Cahn	$\frac{\partial u}{\partial t} = D\frac{\partial^2 u}{\partial x^2} + 5(u - u^3)$	$x \in [-1,1], t \in [0,1]$	4×64	Adam + L-BFGS
Wave (multi-scale)	$\frac{\partial^2 u}{\partial t^2} = 4\frac{\partial^2 u}{\partial x^2}$	$x \in [0,1], t \in [0,1]$	6×100	Adam + L-BFGS

Inverse Problems

Problem	PDE	Parameters to Infer	Network	Optimizer

Problem	PDE	Parameters to Infer	Network	Optimizer
Diffusion-Reaction	$\frac{\partial u}{\partial t} = D\frac{\partial^2 u}{\partial x^2} + ku^2$	D, k	4×20	Adam
Korteweg-de Vries (KdV)	$\frac{\partial u}{\partial t} + \lambda_1 u\frac{\partial u}{\partial x} + \lambda_2 \frac{\partial^3 u}{\partial x^3} = 0$	λ_1, λ_2	4×100	Adam

Key Experimental Results

Forward Problems (L^2 Relative Error)

Method	Diffusion	Burgers'	Allen-Cahn	Wave
Grid	0.66%	13.7%	93.4%	81.3%
Random	0.74%	13.3%	22.2%	68.4%
LHS	0.48%	13.5%	26.6%	75.9%
Halton	0.24%	4.51%	0.29%	60.2%
Hammersley	0.17%	3.02%	**0.14%**	58.9%
Sobol	0.19%	3.38%	0.35%	57.5%
Random-R	**0.12%**	1.69%	0.55%	**0.72%**
RAR-G	0.20%	**0.12%**	0.53%	0.81%
RAD	**0.11%**	**0.02%**	**0.08%**	**0.09%**
RAR-D	**0.14%**	**0.03%**	**0.09%**	**0.29%**

Bold = top 3 performers

Key Findings

1. **RAD performs best overall** among all 10 sampling methods for both forward and inverse problems
2. **For PDEs with complicated solutions** (Burgers', multi-scale Wave):
    - RAD and RAR-D are **dramatically better** (errors magnitudes lower)
    - 0.02% vs 13% for Burgers' (650× improvement!)
3. **For PDEs with smooth solutions** (Diffusion, Diffusion-Reaction):
    - Some uniform methods (Hammersley, Random-R) also perform well
    - Adaptive methods still best, but margin is smaller
4. **Low-discrepancy sequences** (Halton, Hammersley, Sobol) generally outperform Random and LHS

5. **Resampling helps**: Random-R consistently beats fixed Random sampling
6. **Recommended hyperparameters**:
    - RAD: `k=2, c=0` (default)
    - RAR-D: `k=2, c=0` (default)
    - Resampling period: problem-dependent, typically every 100-1000 iterations

---

Implementation Recommendations for pinn-point

Easy to Implement (High Priority)

1. **Random-R**: Simply resample uniform random points every N epochs
    - Minimal code change from current random sampling
    - Good baseline improvement
2. **RAD**: At each resampling step:
    - Evaluate residuals on dense candidate set
    - Compute weights: `w = residual^k / mean(residual^k) + c`
    - Sample points proportionally to weights
    - Replace training set
3. **Low-discrepancy sequences**: Use scipy's `qmc` module

```
from scipy.stats import qmc
sampler = qmc.Halton(d=2)  # or Sobol, LatinHypercube
points = sampler.random(n=1000)
```

Moderate Complexity

4. **RAR-D**: Similar to RAD, but:
    - Keep existing points
    - Add M new points sampled by PDF
    - Dataset grows over time
5. **RAR-G**: Like current adaptive refinement, but:
    - Use top-k residual points instead of mesh refinement
    - Add points greedily

Comparison with Current pinn-point Approach

The current `pinn-point` adaptive method uses **mesh refinement** based on FEM error estimation, which is conceptually similar to **RAR-G** but:

Aspect	pinn-point (current)	RAR-G (paper)
Refinement basis	FEM mesh + error estimator	PDE residual magnitude
Point placement	Mesh vertices	Any location (meshless)
Geometry	Complex geometries via NGSolve	Simple domains
Computational cost	Higher (mesh operations)	Lower

The paper's methods are **meshless** and potentially simpler to implement, but our mesh-based approach handles complex geometries naturally.

References

- [1] Wu et al. (2022) - This paper
- [2] Lu et al. (2019) - Original RAR method (DeepXDE)
- [3] Nabian et al. (2021) - PDF-based resampling