Key Insights on Bayesian Physics-Informed Neural Networks (B-PINNs)

Implementation Notes

February 22, 2026

Abstract

This document summarizes critical theoretical and implementation insights gained while building a from-scratch Bayesian Physics-Informed Neural Network (B-PINN) and applying Hamiltonian Monte Carlo (HMC). It clarifies apparent contradictions between standard Machine Learning terminology (loss, training) and Bayesian Inference (energy, sampling), and explains why PyTorch requires highly specific computational graph management for B-PINN sampling.

1 Why PINNs are Inherently Robust to High-Frequency Noise

When dealing with partial differential equations (PDEs), such as the 1D Poisson equation $u_{xx} = f(x)$, a PINN network maps the coordinate x to the predicted solution $u(x)$. The physics constraint enforces that the second spatial derivative of the network output must match the forcing target $f(x)$ at collocation points x_f.

When the forcing targets y_f are heavily corrupted with noise (e.g., $y_f = -\pi^2 \sin(\pi x) + \mathcal{N}(0, 1)$), one might intuitively expect the network's prediction $u(x)$ to become noisy and jagged. However, $u(x)$ typically remains perfectly smooth.

This occurs because solving the PDE $u_{xx} = f(x)$ mathematically implies integrating $f(x)$ twice. Integration inherently acts as a **low-pass filter**, systematically suppressing high-frequency noise. Furthermore, the spectral bias of neural networks naturally favors learning low-frequency (smooth) functions first. Consequently, the PINN effortlessly ignores high-frequency jitter in y_f and fits a smooth macro-structure.

2 Data: Boundary/Interior Sensors (x_u, y_u) vs. Collocation Targets (x_f, y_f)

In B-PINN literature, there is a clear distinction between the actual state measurements of a system and the physical constraints acting upon it:

- **Observations (y_u)**: These are direct sensor measurements of the true state $u(x)$.
  - If they are placed exactly on the edges of the domain, they serve strictly as **Boundary Conditions**.
  - If they are scattered throughout the interior, the setup becomes an **Inverse Problem** or Data Assimilation scenario. Here, we are trying to deduce unknown physical parameters or forcing terms from sparse, noisy thermometer or tracking data.
- **Collocation Targets (y_f)**: These represent the PDE constraints (e.g., the known load on a beam, or the known internal heat source of a plate). They do not measure the state $u(x)$ itself. We enforce the known PDE residual at random spatial coordinates x_f to effectively bound the infinite space of possible functions to only those obeying the designated physics.

3 The Illusion of "Loss" in B-PINNs: Energy Equivalence

In standard deep learning, we optimize a neural network by minimizing a Loss Function (like Mean Squared Error). In Bayesian inference, there is no "Loss" function and no optimization—we merely map an energy landscape and gather samples proportionally to the *Posterior Distribution*.

Therefore, the Yang et al. (2020) B-PINN paper never refers to a "Physics Loss". Instead, it defines the Unnormalized Posterior as a combination of independent Gaussian likelihoods:

$$p(\theta|\mathcal{D}) \propto p(\theta) \cdot \prod \mathcal{N}\Big(y_u \mid u_\theta(x_u), \sigma_u^2\Big) \cdot \prod \mathcal{N}\Big(y_f \mid f_\theta(x_f), \sigma_f^2\Big)$$

Hamiltonian Monte Carlo (HMC) generates samples by simulating a particle sliding on an energy landscape $U(\theta)$, defined as the negative log-posterior:

$$U(\theta) = -\log p(\theta|\mathcal{D})$$

Because the negative natural logarithm of a Gaussian Probability Density Function mathematically expands out to the Mean Squared Error (MSE), the Bayesian "Potential Energy" is precisely equivalent to a scaled classical PINN Loss:

$$-\log \mathcal{N}(y|\hat{y}, \sigma^2) = \frac{1}{2\sigma^2}(y - \hat{y})^2 + C \quad \propto \quad \text{MSE}$$

Thus, navigating the B-PINN posterior landscape is mathematically identical to traversing a surface defined by the (weighted) MSE Data Loss + MSE Physics Loss + L2 Regularization (Prior).

4 Boundary vs. Physics Competition: Trivial Solutions and Posterior Collapse

During practical implementation of Physics-Informed Neural Networks (particularly when encountering stiff or rigid ODEs/PDEs like the unforced Damped Harmonic Oscillator, $mu_{tt} + cu_t + ku = 0$), a critical competition emerges between the Physics constraints (e.g., matching the true dynamics over time) and the Boundary/Initial data constraints ($u(0)$, $u'(0)$).

4.1 Standard PINNs: Optimizer Laziness and Trivial Minima

In a standard PINN, the total loss function is typically the unweighted sum of the boundary Mean Squared Error (MSE) and the physics residual MSE. For systems where the governing equations permit a trivial solution (e.g., $u(t) = 0$ perfectly solves the unforced oscillator equation globally), the optimizer may find it overwhelmingly easier to achieve a perfect physics loss of 0.0 by predicting a flat zero. By doing so, the network accepts a small, localized penalty for failing to intersect the exact boundary condition (e.g., $u(0) = 1$) in exchange for a massive, domain-wide reduction in physics loss.

To correct this behavior, practitioners must explicitly introduce a **hyperparameter weighting factor** (e.g., $\lambda_{\text{boundary}} = 100$) scaled directly against the boundary loss. This forcefully communicates to the optimizer that minimizing the empirical data loss takes priority over physics satisfaction, demanding the network approximate the underlying PDE *only on the manifold of functions* that strictly satisfy the provided boundary conditions.

4.2 Bayesian PINNs: Energy Wells and Posterior Collapse

Bayesian PINNs face an analogous, probabilistically equivalent challenge during Hamiltonian Monte Carlo (HMC) trajectory sampling. The HMC sampler explores the topology of the potential energy landscape defined by $U(\theta) = -\log p(\text{data}|\theta) - \log p(\text{physics}|\theta) - \log p(\theta)$.

When the underlying physics equations are stiff or heavily constrained, randomly proposed network weights almost exclusively yield massive, non-compliant physics residuals, resulting in high potential energy and immediate sample rejection. Simultaneously, the aforementioned trivial solution $u(x) = 0$ yields zero physics energy. Thus, the HMC sampler often slides inevitably into this "zero-energy physical well" and collapses, endlessly rejecting outgoing proposals that exhibit variance or structure. This phenomenon manifests as a flatline predictive mean coupled with a suspiciously tight, zero-bound uncertainty variance.

In a pure Bayesian formulation, one cannot arbitrarily inject a "100x multiplier" into the posterior density logarithm. Instead, we must control the topological structure of the energy landscape by manipulating the underlying **likelihood variances**, σ_u (for boundary observations) and σ_f (for physics residuals).

By assigning a comparatively tiny variance to the boundary likelihood (e.g., $\sigma_u = 0.01$) relative to the looser physics likelihood (e.g., $\sigma_f = 0.1$), we mathematically inject a massive, incredibly steep probability density function spike perfectly centered precisely around the boundary coordinates. The HMC sampler is subsequently forced algorithmically to only explore regions of the model's parametric state-space that strictly adhere to the initial condition $u(0) = 1$, successfully rescuing the Markov chain out from the trivial zero-energy collapse and accurately restoring uncertainty quantification bounding actual dynamic trajectories.

5 HMC and Autograd Graph Detachment in PyTorch

Implementing B-PINN HMC from scratch in PyTorch requires extreme care regarding computational graph tracking via `torch.autograd`.

5.1 Why the Leapfrog Integrator requires `torch.no_grad()`

HMC uses a Leapfrog integrator to simulate the discrete steps of the particle across the energy landscape.

$$\theta_{t+\Delta t} = \theta_t + \Delta t \cdot r_t$$

If left unchecked, PyTorch's Autograd engine will silently interlink every single operation across hundreds of sampling steps, constructing a single, massive computational graph in memory. This results in memory exhaustion, and gradient explosions yielding `NaN` values. We must explicitly wrap the position (θ) and momentum (r) updates in `with torch.no_grad():` and `.clone().detach()` the raw tensors before the loop so PyTorch drops historical Markov chain lineage.

5.2 Why Potential Energy $U(\theta)$ requires Autograd tracking

While we must detach the chain history, we **cannot** use `torch.no_grad()` to evaluate the Hamiltonian (Energy) of a given state θ.

In standard BNNs whose energy only depends on standard data outputs (e.g., classifying cats and dogs), one could evaluate energy without forming a graph. However, the B-PINN's *Physics Energy* demands the calculation of spatial derivatives (e.g., u_x, u_{xx}).

PyTorch computes these PDE spatial derivatives precisely by constructing the forward graph connecting spatial input x to network output u, and then running `torch.autograd.grad(u, x, create_graph=True)`. If `torch.no_grad()` is active when evaluating $U(\theta)$, PyTorch leaves no graph tape to traverse, crashing the Autograd engine.

Thus, the core rhythm of PyTorch B-PINN HMC is:

1. **Turn Autograd ON** to evaluate physics targets and calculate the gradient $\nabla_\theta U(\theta)$.
2. **Turn Autograd OFF** to apply the gradient and step the physical particle forward across the landscape without building an infinitely long historical tape.