Computational Mathematics and AI

Lecture 8: High-Dimensional PDEs

Lars Ruthotto
Departments of Mathematics and Computer Science
lruthotto@emory.edu
larsruthotto

slido.com #CBMS25

Reading List

Historical Context: High-dimensional PDEs are abundant, curse of dimensionality coined in optimal control (Bellman 1961).

Key Readings:

1. Han, Jentzen, and E (2018) – Solving High-Dimensional PDEs Using Deep Learning. *PNAS*

   Deep BSDE method breaking curse of dimensionality.

2. Raissi (2018) – Forward-Backward Stochastic Neural Networks. *arXiv*

   FBSNNs for high-dimensional parabolic PDEs.

3. Hu et al. (2024) – Hutchinson Trace Estimation for PINNs. *JMLR*

   Scaling PINNs to 100,000+ dimensions.

4. Li, Verma, and Ruthotto (2024) – Neural Network for Stochastic Optimal Control *SIAM SISC*

   Neural networks for high-dimensional control problems.

Lecture Outline: Model Problem $\rightarrow$ PINNs+HTE $\rightarrow$ FBSDE $\rightarrow$ PMP-informed

High-Dimensional Semilinear Parabolic PDE

The Semilinear Parabolic PDE

We consider the **semilinear parabolic PDE** for $u : [0, T] \times \mathbb{R}^d \to \mathbb{R}$:

$$\frac{\partial u}{\partial t} + \mu(t, x) \cdot \nabla u + \frac{1}{2}\text{trace}\left(\sigma\sigma^\top \nabla^2 u\right) + f(t, x, u, \sigma^\top \nabla u) = 0, \quad \text{for } t \in [0, T)$$

- $u(T, x) = g(x)$ for $x \in \mathbb{R}^d$ — **terminal condition** at time T
- $\mu : [0, T] \times \mathbb{R}^d \to \mathbb{R}^d$ — **drift coefficient**
- $\sigma : [0, T] \times \mathbb{R}^d \to \mathbb{R}^{d \times d}$ — **diffusion coefficient**
- f — **nonlinear drift term**, depends on u and ∇u

Applications:

- Stochastic optimal control (Hamilton-Jacobi-Bellman equations)
- Financial mathematics (option pricing)
- Pattern formation and reaction-diffusion systems (Allen-Cahn equation)

Example: HJB (Han, Jentzen, and E 2018; Raissi 2018)

$$\frac{\partial u}{\partial t} + \Delta u - \|\nabla u\|^2 = 0, \quad \text{for} \quad t \in [0, T) \qquad u(T, x) = g(x)$$

Solution is the value function of the stochastic optimal control problem:

$$\min_{a \in \mathcal{A}} \mathbb{E}\left[g(X_T) + \int_0^T L(X_s, a_s)\, ds \right]$$

$$\text{s.t. } dX_s = 2a_s\, ds + \sqrt{2}\, dW_s, \quad X_0 = x$$

- **Running cost:** $L(x, a) = \|a\|^2$
- **Terminal cost:** $g(x) = \log\left(\frac{1}{2}\left(1 + \|x\|^2\right)\right)$

- Analytical solution (Hopf-Cole transform): $u(t, x) = -\log\left(\mathbb{E}\left[\exp(-g(X_T)) \mid X_t = x\right]\right)$
- Verify: Let $v = e^{-u}$. Then $\partial_t v = -\partial_t u \cdot v$, $\nabla v = -\nabla u \cdot v$, $\Delta v = (\|\nabla u\|^2 - \Delta u)v$.
  HJB $\Rightarrow \partial_t v = \Delta v$, i.e., v solves the **heat equation**!
- $d = 100$ is completely intractable for grid-based methods

Today: illustrate curse of dimensionality with four representative approaches

General Paradigm: NNs for High-Dimensional Control

Offline: Learn control (high computational cost)

1. Parameterize control/value function with neural net
2. Sample state space: uniform? random walk?
3. Define loss function: PDE residual, terminal matching, control objective, ...
4. Train weights via SGD, Adam, ...

Challenge: Avoid curse of dimensionality in network size, sample complexity, time

Online: Evaluate policy (very fast, real-time)

Evaluate trained policy and measure performance with control objective (can be different from loss)

Today: Compare three neural approaches for high-D stochastic control

PINNs with Hutchinson Trace Estimation

PINNs for Semilinear Parabolic PDEs

Idea: Parameterize $u_\theta(t,x)$ as a neural network
Loss function: Minimize expected PDE residual

$$\mathcal{L}(\theta) = \mathbb{E}_{(t,x)} \left|\mathcal{N}[u_\theta](t,x)\right|^2$$

with PDE residual $\mathcal{N}[u] = \partial_t u + \mu \cdot \nabla u + \frac{1}{2}\mathrm{trace}(\sigma\sigma^\top \nabla^2 u) + f$

Computational challenge: Hessian Computation
For our PDE, we need $\mathrm{trace}(\sigma\sigma^\top \nabla^2 u)$

- Hessian matrix: $d \times d = 100 \times 100 = 10,000$ entries
- Computing full Hessian: $O(d^2)$ memory, $O(d^2)$ compute
- For $d = 1000$: Compute/memory becomes prohibitive

Problem: Standard PINNs fail at $d > 1000$ due to Hessian cost
Solution: Hutchinson Trace Estimation (HTE)

The Hutchinson Trace Estimator

For our PDE: $\mathrm{trace}(\sigma\sigma^\top \nabla^2 u) = \mathrm{trace}(\sigma^\top \nabla^2 u\, \sigma) = \mathrm{trace}(A)$ where $A = \sigma^\top \nabla^2 u\, \sigma$

Hutchinson's Trick (1990):
For any matrix A and random vector v with $\mathbb{E}[vv^\top] = I$:

$$\mathrm{trace}(A) = \mathbb{E}[v^\top A v]$$

Monte Carlo Approximation:

$$\mathrm{trace}(A) \approx \frac{1}{V} \sum_{i=1}^{V} v_i^\top A v_i$$

where $v_i \sim$ Rademacher (entries ± 1 with probability $1/2$)

Using AD for Hessian-Vector Products (HVP)

- $v^\top \sigma^\top \nabla^2 u\, \sigma v$ can be computed via autodiff in $O(1)$ memory!
- Taylor-mode AD: forward-over-reverse differentiation

Result: $O(d^2) \rightarrow O(1)$ memory/compute: enables PINNs in high dimensions!

HTE-PINN: Where Does It Sample?

The HTE-PINN Learning Problem:

$$\min_{\theta} \mathbb{E}_{(t,x),v} \left| \partial_t u_\theta + \mu \cdot \nabla u_\theta + \frac{1}{2} v^\top \sigma^\top \nabla^2 u_\theta \, \sigma v + f \right|^2$$

where $(t, x) \sim \text{Uniform}([0, T] \times \Omega)$ and $v \sim \text{Rademacher}(\pm 1)$

The Sampling Strategy:

- Sample (t_i, x_i) uniformly from $[0, T] \times \Omega$
- Use mini-batch SGD/Adam

Discussion:

- does the added noise from HTE impact convergence?
- uniform sampling does not beat curse of dimensionality

Next: Exploit problem structure with relation to SDEs

FBSDE-Based Methods

PDE $\rightarrow$ SDE: Forward-Backward SDE System

$$\frac{\partial u}{\partial t} + \mu(t,x) \cdot \nabla u + \frac{1}{2}\text{trace}\left(\sigma\sigma^\top \nabla^2 u\right) + f(t,x,u,\sigma^\top \nabla u) = 0, \quad \text{for } t \in [0,T), \quad u(T,x) = g(x)$$

Forward SDE: $dX_t = \mu(t,X_t)\,dt + \sigma(t,X_t)\,dW_t, \qquad X_0 = x_0$

What is the evolution of $u(t,X_t)$ along SDE trajectory? Ito's lemma gives:

$$\begin{aligned} du(t,X_t) &= \left(\frac{\partial u}{\partial t} + \mu(t,x) \cdot \nabla u + \frac{1}{2}\text{trace}\left(\sigma\sigma^\top \nabla^2 u\right)\right) dt + (\nabla_x u)^\top \sigma\, dW_t \\ &= -f(t,X_t,u,\sigma^\top \nabla u)\,dt + (\nabla_x u)^\top \sigma\, dW_t, \quad u(T,X_T) = g(X_T) \end{aligned}$$

Backward SDE: $Y_t = g(X_T) + \int_t^T f(s,X_s,Y_s,Z_s)\,ds - \int_t^T Z_s^\top\, dW_s$

If (X_t, Y_t, Z_t) solves FBSDE system, then $Y_t = u(t,X_t)$ and $Z_t = [\sigma(t,X_t)]^\top (\nabla_x u)(t,X_t)$

Method 1: Deep BSDE (Han, Jentzen, and E 2018)

Key Idea: Optimize NN Approximation Z_θ and Y_0 Directly

- Learnable scalar $Y_0 \approx u(0, X_0)$
- Stack of $N-1$ neural networks: $Z_k(X_k) = Z_\theta(t_k, X_k)$ for each time step
- Each subnet: 4 layers, width $d+10$, BatchNorm + ReLU

Loss Function (Terminal Matching):

$$\mathcal{L}(\theta) = \mathbb{E}\left[\left| Y_N - g(X_N) \right|^2 \right]$$

Where Y_N is computed by simulating the FBSDE forward in time:

$$\begin{aligned} X_{k+1} &= X_k + \mu(t, X_k)\Delta t + \sigma \Delta W_k \\ Y_{k+1} &= Y_k + f(t, X_k, u, Z_k(X_k))\Delta t + Z_k(X_k)^\top \Delta W_k \end{aligned}$$

Gives only pointwise estimate of $u(0, X_0)$ at initial state!

Method 2: Fwd/Bwd Stochastic NN (Raissi 2018)

Key Idea: Optimize NN approximation $u_\theta(t,x)$ using FBSDE Residuals

- Scalar-valued neural network $u_\theta(t,x)$ shared across all times
- Gradient $\nabla_x u_\theta$ via automatic differentiation
- Advantages over Deep BSDE: Parameter efficiency, can evaluate u_θ anywhere

Loss Function:

$$\mathcal{L}(\theta) = \mathbb{E}\left[|Y_N - g(X_N)|^2 + \alpha \sum_{k=0}^{N-1} |Y_{k+1} - Y_k + f_k \Delta t - Z_k^\top \Delta W_k|^2 \right]$$

where $Y_k = u_\theta(t_k, X_k)$ and $Z_k = \sigma^\top \nabla_x u_\theta(t_k, X_k)$

Σ : PINNs, Deep BSDE, and FBSNN

PINNs: Minimize PDE residual over domain

- In high-D: Use Hutchinson trace estimator for Hessian trace
- Sampling: Random collocation in $[0, T] \times \Omega$

Deep BSDE: Learn Y_0 and Z_k per time step via terminal matching

- avoids Hessian computation by working with FBSDE
- Sampling: Forward SDE $dX = \mu_t dt + \sigma_t\, dW$
- Optimization: Find Y_0 and Z_k to minimize $\|Y_N - g(X_N)\|^2$

FBSNN: Learn $u_\theta(t, x)$ via FBSDE residuals

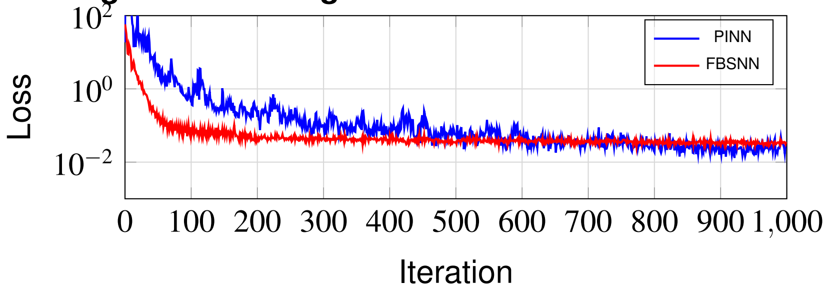
- Advantage over Deep BSDE: single NN for all times, can evaluate anywhere
- Sampling: Forward SDE $dX = \mu_t dt + \sigma_t\, dW$
- Optimization: Minimize residuals at each time step + terminal matching

Common disadvantage for HJB: Sampling independent of optimal control!

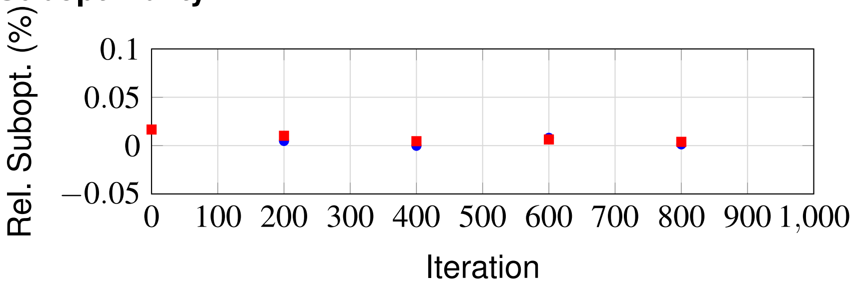
Numerical Experiments

100D HJB Benchmark: Results (Centered Target)

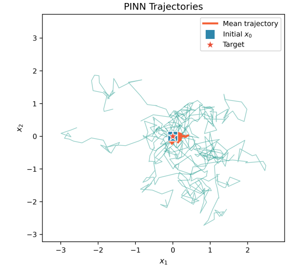
Training Loss Convergence:

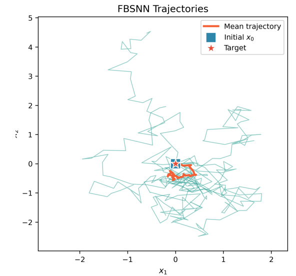


Suboptimality:



Trajectories (2D projection):





100D HJB Benchmark

$$-\frac{\partial u}{\partial t} + \Delta u - \|\nabla u\|^2 = 0, \quad \text{for} \quad t \in [0, T) \qquad u(T, x) = g(x)$$

Use terminal cost from literature: $g(x) = \log\left(\frac{1}{2}\left(1 + \|x\|^2\right)\right)$, i.e., $x_{\text{target}} = \mathbf{0} \in \mathbb{R}^{100}$.

- Initial states: $X_0 \sim \mathcal{N}(0, 0.1^2 I_{100})$ (near origin)
- Terminal time: $T = 1.0$
- Ground truth: Monte Carlo with 10^6 samples

Method	Relative Suboptimality	Training Time	Status
PINNs + HTE	$<$1%	$\sim$9 min	✓ Success
FBSNN	$<$1%	$\sim$17 min	✓ Success

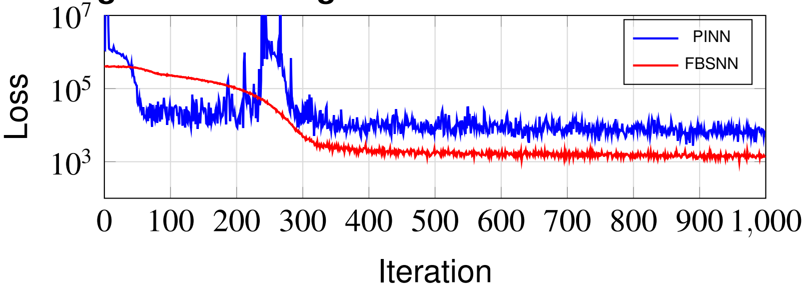
Why It Works:

- Random samples (collocation or random walk) stay near origin
- Minimizer of terminal cost is at origin $\Rightarrow$ samples cover the important region!
- Network learns the value function where it matters

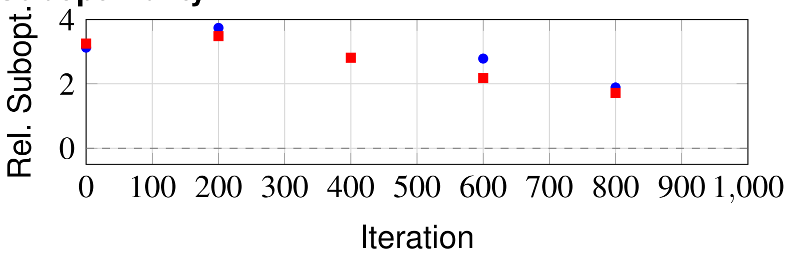
Both methods achieve $<$1% suboptimality in 100 dimensions!

100D HJB Benchmark: Results (Shifted Target)

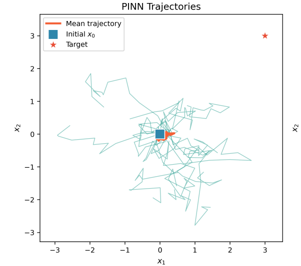
Training Loss Convergence:

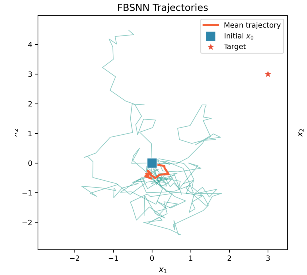


Suboptimality:



Trajectories (2D projection):





Modified 100D HJB Benchmark: Shifted Target

$$-\frac{\partial u}{\partial t} + \Delta u - \|\nabla u\|^2 = 0, \quad \text{for} \quad t \in [0, T) \qquad u(T, x) = g(x)$$

Use modified terminal cost: $g(x) = 1000 \log\left(\frac{1}{2}\left(1 + \|x - \mathbf{3}\|^2\right)\right)$, i.e., $x_{\text{target}} = \mathbf{3} \in \mathbb{R}^{100}$.

Method	Relative Suboptimality	Convergence	Status
PINNs	238%	looks good	× Fails
FBSNN	147%	looks good	× Fails

What Happened?

- Target distance: $\|x_{\text{target}}\| = 3\sqrt{100} = 30$
- Typical random walk distance: $\|X_T\| \sim \sqrt{2 \cdot 100} \approx 14$
- Random collocation: uniform in bounded domain, misses far target
- Samples rarely reach the target region!

As suspected: Both methods FAIL when the target shifts far away!

Neural SDEs for Stochastic Optimal Control

HJB and Pontryagin Maximum Principle

Consider the value function of the stochastic optimal control problem:

$$u(t,x) = \min_a \left\{ \mathbb{E}\left[\int_t^T L(X_s, a_s)\, ds + g(X_T) \right], \quad \text{s.t. } dX_s = \mu(X_s, a_s)\, ds + \sigma\, dW_s,\ X_t = x \right\}$$

Key facts from optimal control theory:

1. **HJB equation:** The value function satisfies

$$-\partial_t u + \sup_a \mathcal{H}(t, x, \nabla u, \sigma \nabla^2 u, a) = 0, \quad u(T,x) = g(x)$$

   where $\mathcal{H}(t,x,p,M,a) = \frac{1}{2}\text{trace}(\sigma M) + p^\top \mu(x,a) - L(x,a)$

2. **Feedback form (PMP):** Optimal control is given by

$$a^*(t,x) \in \text{argmax}_a \mathcal{H}(t, x, \nabla u(t,x), \sigma \nabla^2 u(t,x), a)$$

For our HJB benchmark: $L(x,a) = \|a\|^2$, $\mu(x,a) = 2a$, $\sigma = \sqrt{2}$

$\Rightarrow$ Optimal control: $\boxed{a^*(t,x) = -\nabla u(t,x)}$

Challenges: forward-backward structure, nonlinearity, high-dimensionality

The Controlled Forward SDE

Key Insight: Use PMP to Define the Forward SDE

FBSDE Methods (earlier section)

Random walk:

$$dX_t = \sqrt{2}\, dW_t$$

- Sampling independent of θ
- Trajectories don't reach target
- × Fails when target shifts!

Neural SOC (This section)

PMP-guided:

$$dX_t = \underbrace{-2\nabla u_\theta(t, X_t)}_{2a}\, dt + \sqrt{2}\, dW_t$$

- Sampling depends on θ
- Trajectories guided toward target
- ✓ Works for any target!

Consequence: The drift $-2\nabla u_\theta$ guides samples to low-cost regions.

What is the backward SDE along the controlled trajectory?

Itô's Lemma: Deriving the Backward SDE

Setup: Let u solve the HJB: $\partial_t u + \Delta u - \|\nabla u\|^2 = 0$, $u(T,x) = g(x)$
Consider $u(t, X_t)$ along the **controlled trajectory**: $dX_t = -2\nabla u(t, X_t)\, dt + \sqrt{2}\, dW_t$

Apply Itô's lemma:

$$\begin{aligned} du(t, X_t) &= \partial_t u\, dt + \nabla u^\top dX_t + \frac{1}{2}\text{trace}(\nabla^2 u \cdot 2I)\, dt \\ &= \partial_t u\, dt + \nabla u^\top \left(-2\nabla u\, dt + \sqrt{2}\, dW_t\right) + \Delta u\, dt \\ &= \left(\partial_t u - 2\|\nabla u\|^2 + \Delta u\right) dt + \sqrt{2}\nabla u^\top dW_t \end{aligned}$$

Using HJB: $\partial_t u + \Delta u = \|\nabla u\|^2$, we get:

$$\partial_t u - 2\|\nabla u\|^2 + \Delta u = \|\nabla u\|^2 - 2\|\nabla u\|^2 = -\|\nabla u\|^2$$

With $a^* = -\nabla u$ and running cost $L = \|a^*\|^2 = \|\nabla u\|^2$:

$$\boxed{du(t, X_t) = -L(X_t, a_t^*)\, dt + \sqrt{2}\nabla u^\top dW_t}$$

The backward SDE has drift $= -$(running cost)! Terminal: $u(T, X_T) = g(X_T)$

Comparing: Random Walk vs. Controlled FBSDE

Random Walk FBSDE

Forward: $dX_t = \sqrt{2}\, dW_t$
Backward:

$$dY_t = +\|\nabla u\|^2\, dt + Z_t^\top dW_t$$

- $Y_t = u(t, X_t)$, $Z_t = \sqrt{2}\nabla u$
- Drift is positive!
- Value *increases* along random paths (drifting into high-cost regions)

Controlled FBSDE

Forward: $dX_t = -2\nabla u_\theta\, dt + \sqrt{2}\, dW_t$
Backward:

$$dY_t = -\|\nabla u\|^2\, dt + Z_t^\top dW_t$$

- $Y_t = u(t, X_t)$, $Z_t = \sqrt{2}\nabla u$
- Drift is negative!
- Value *decreases* by running cost along optimal paths

Martingale verification: Define $M_t = Y_t + \int_0^t L\, ds$. Then $dM_t = Z_t^\top dW_t$ (martingale!)

$$\Rightarrow \quad u(0, x_0) = \mathbb{E}\left[\int_0^T L(X_t, a_t^*)\, dt + g(X_T)\right] \quad \text{(control objective!)}$$

Why Random Sampling Methods Fail

What Went Wrong with PINNs, Deep BSDE, FBSNN?

- **PINNs + HTE:** Random collocation points in bounded domain
- **Deep BSDE:** Random walk $dX = \sqrt{2}\,dW$
- **FBSNN:** Random walk $dX = \sqrt{2}\,dW$

All ignore the **optimal control structure** of the problem!

The Solution: PMP-Informed Sampling

Instead of random sampling, use the **controlled dynamics**:

$$\boxed{dX_t = -2\nabla u_\theta(t, X_t)\,dt + \sqrt{2}\,dW_t}$$

Benefits:

- Trajectories are guided toward optimal paths (even with crude initial u_θ)
- Backward SDE becomes simple: just integrates running cost
- Loss function directly measures control objective

We use the current network estimate to guide sampling!

PMP-Informed Neural SDE Solver: The Training Loop

1. **Initialize** value network $u_\theta(t,x)$
2. **For each training iteration:**
   (a) **Compute optimal control:** $a^*_\theta(t,x) = -\nabla_x u_\theta(t,x)$
   (b) **Sample trajectories with PMP-guided drift:**

$$X_{k+1} = X_k + 2a^*_\theta(t_k, X_k)\Delta t + \sqrt{2\Delta t}\,\xi_k, \quad \xi_k \sim \mathcal{N}(0,I)$$

   (c) **Compute loss:**

$$\mathcal{L}(\theta) = \mathbb{E}\left[\sum_{k=0}^{N-1} L(X_k, a^*_k)\Delta t + g(X_N)\right] + \lambda_{\text{HJB}} P_{\text{HJB}} + \lambda_T P_T + \lambda_{\nabla T} P_{\nabla T}$$

(penalty terms enforce HJB and terminal conditions)

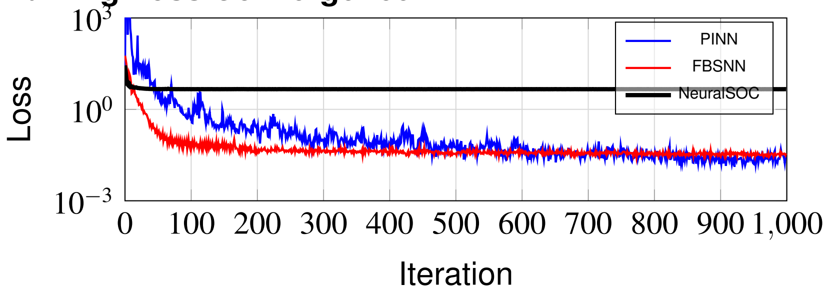
   (d) **Update** θ via gradient descent
3. **Return** trained network u_θ

Key Difference from Random-Sampling Methods:

- Trajectories are **guided by current policy estimate**
- Crude estimate initially, iterations pulls trajectories toward relevant regions
- **Must backprop through the SDE!** (X_k depends on θ via a^*_θ)

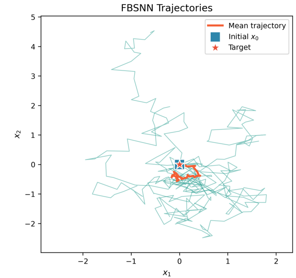
100D HJB Benchmark: Results Neural SOC (Centered)

Training Loss Convergence:

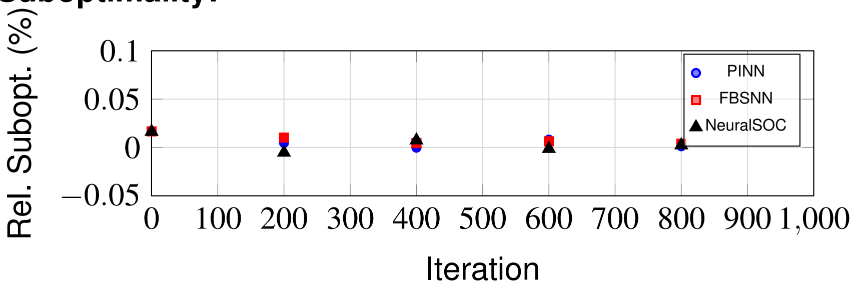


Trajectories:

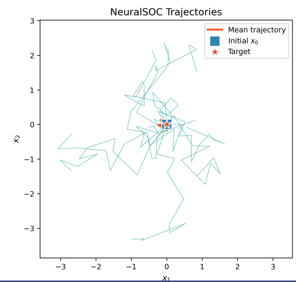
FBSNN:



Suboptimality:

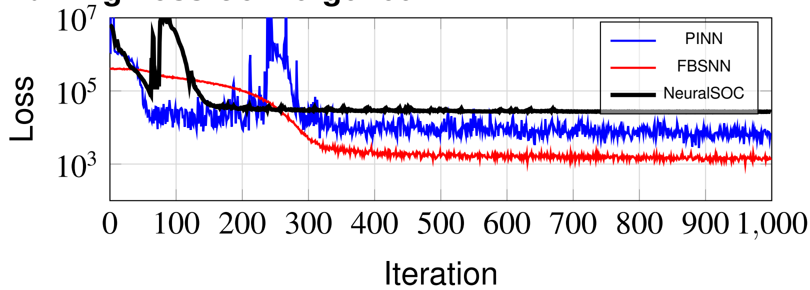


NeuralSOC:

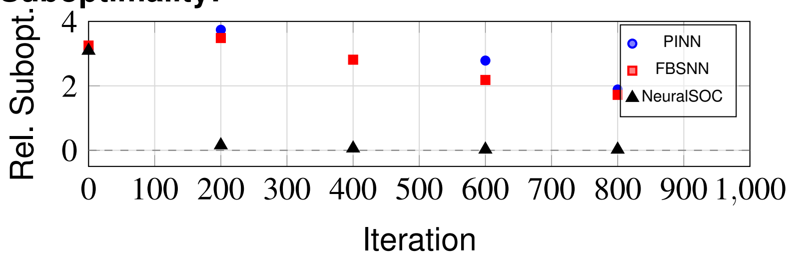


100D HJB Benchmark: Results with Neural SOC (Shifted)

Training Loss Convergence:

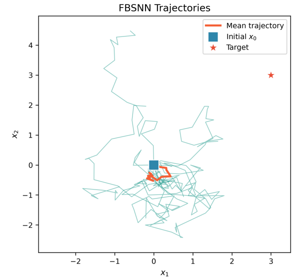


Suboptimality:

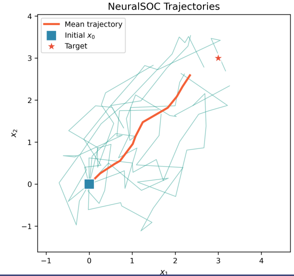


Trajectories:

FBSNN:



NeuralSOC:



Results: PMP-Informed Succeeds Where Others Fail

The Hard Case: $x_{\textbf{target}} = (3, \ldots, 3)^\top \in \mathbb{R}^{100}$

Method	**Sampling**	**Rel. Suboptimality**	**Status**
PINNs	Random collocation	238%	× Fails
FBSNN	$dX = \sigma\, dW$	147%	× Fails
Neural SOC	$dX = -2\nabla u_\theta\, dt + \sigma\, dW$	1.2%	✓ Success!

Three ingredients for solving high-dimensional HJB:

1. FBSDE reformulation (continuous-time dynamics)
2. Neural network approximation (meshfree representation)
3. Smart sampling (guided by current policy)

Local vs. Global vs. Semi-global Solutions

Different types of solutions for optimal control:

- **Local solution:** Find optimal trajectory for one given initial state x_0
  - Standard shooting methods, adjoint methods
  - Must resolve for each new initial condition
- **Global solution:** Find optimal policy for all states $(t, x) \in [0, T] \times \Omega$
  - Requires solving HJB on full domain
  - Curse of dimensionality: impossible for $d \gg 1$!
- **Semi-global solution:** Find policy that is optimal in *the subset of state space likely to be visited*
  - Realistic goal for high-dimensional problems
  - Learn u_θ along (approximately) optimal trajectories
  - Generalizes to nearby initial conditions

Key insight: PMP-informed sampling gives semi-global solutions!

You get good policies **where you sample** $\Rightarrow$ sample where it matters!

Outlook and Summary

Outlook: Reinforcement Learning and HJBs

Reinforcement Learning for Control

- Alternative approach for solving (stochastic) optimal control problems
- Example: Actor-critic methods for games
- Only observations needed (of system and objective)
- Attractive when model is complex, incomplete, or unavailable
- Challenge: Sample efficiency (Scientific ML is not an Atari game!)

RL + HJB: Best of Both Worlds?

- Exploit that objective function is **known** (unlike pure RL)
- Learn control-affine dynamics model:

$$dX_t = f_\mu(t, X_t)\, dt + B_\mu(t, X_t)\, a_t\, dt + \sigma\, dW_t$$

  with learnable parameters μ
- Use model to estimate u and guide sampling $\rightarrow$ reduce sample complexity

HJB RL: Use structure when available, see Verma et al. 2024

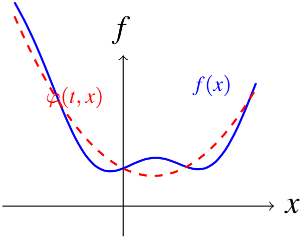
Outlook: HJB in Global Optimization

Goal: Find global minimum of non-convex $f(x)$

Algorithm:

1. Compute Moreau envelope:

$$\varphi(t,x) = \min_y f(y) + \frac{1}{2t}\|x-y\|^2$$



2. Gradient descent: $x_{k+1} = x_k - \alpha_k \nabla \varphi(t_k, x_k)$
3. Increase t (smoothing parameter)

Key observation: φ solves Burgers-type HJB!

$$\partial_t \varphi(t,x) + \|\nabla \varphi(t,x)\|^2 = 0, \quad \varphi(0,x) = f(x)$$

Connection to this lecture: Add viscosity $\delta > 0$, use Cole-Hopf transform:

$$\nabla \varphi(t,x) = -\delta \nabla \log v^\delta(t,x)$$

Same sampling ideas apply. See Heaton, Fung, and Osher 2022

Outlook: Mean Field Games and Control

Setup: Large population of interacting agents, each solving optimal control

Individual Agent:

- State $X_t \in \mathbb{R}^d$, control a_t
- Dynamics depend on population density ρ
- Cost depends on ρ (congestion, competition)

Population Level:

- Density $\rho(t,x)$ evolves via Fokker-Planck
- Nash equilibrium: no agent wants to deviate
- Limit of N-player game as $N \to \infty$

Coupled PDE System:

$$\text{HJB (backward):} \quad -\partial_t u + H(x, \nabla u, \rho) = 0, \quad u(T,x) = g(x, \rho_T)$$
$$\text{Fokker-Planck (forward):} \quad \partial_t \rho + \nabla \cdot (\rho\, v^*(x, \nabla u)) = \Delta \rho, \quad \rho(0) = \rho_0$$

Neural Approaches: Similar ideas from today! (Ruthotto et al. 2020)

- Parameterize $u_\theta(t,x)$ and $\rho_\varphi(t,x)$ with neural networks
- Avoid spatial grids $\to$ handle $d = 100+$ dimensions

Applications: crowd motion, traffic, finance, multi-agent RL, generative models

Σ: High-Dimensional Optimal Control

Key Takeaways

- **Three common approaches:** PINNs+HTE, Deep BSDE, FBSNN
  - All use neural approximation, no spatial grid, polynomial cost in d
- **All succeed on "easy" problems** (centered targets)
  - Random sampling happens to cover the important region
- **All fail when the target shifts!**
  - Random sampling misses the important region in high-D
- **PMP-informed sampling succeeds** where others fail
  - Use optimal control structure; feedback loop improves sampling

Open Research Challenges

- **Sampling for general semilinear PDEs**
  - HJB structure enables PMP-guided sampling — what about non-HJB?
- **Avoiding time integration**
  - Flow matching: learn velocity field directly, skip SDE simulation?

References I

- Han, J., A. Jentzen, and W. E (2018). "Solving High-Dimensional Partial Differential Equations Using Deep Learning". In: *Proceedings of the National Academy of Sciences* 115.34, pp. 8505–8510.
- Heaton, Howard, Samy Wu Fung, and Stanley Osher (2022). *Global Solutions to Nonconvex Problems by Evolution of Hamilton-Jacobi PDEs*. arXiv: 2202.11014 [math.OC]. URL: https://arxiv.org/abs/2202.11014.
- Hu, Z., Z. Shi, G. E. Karniadakis, and K. Kawaguchi (2024). "Hutchinson Trace Estimation for High-Dimensional and High-Order Physics-Informed Neural Networks". In: *Computer Methods in Applied Mechanics and Engineering* 424, p. 116883.
- Li, X., D. Verma, and L. Ruthotto (2024). "A Neural Network Approach for Stochastic Optimal Control". In: *SIAM Journal on Scientific Computing* 46.5, A3094–A3117. DOI: 10.1137/23M155832X.
- Raissi, M. (2018). "Forward-Backward Stochastic Neural Networks: Deep Learning of High-Dimensional Partial Differential Equations". In: *arXiv preprint arXiv:1804.07010*.

References II

Ruthotto, L., S. J. Osher, W. Li, L. Nurbekyan, and S. W. Fung (2020). "A Machine Learning Framework for Solving High-Dimensional Mean Field Game and Mean Field Control Problems". In: *Proceedings of the National Academy of Sciences* 117.17, pp. 9183–9193.

Verma, Deepanshu, Nick Winovich, Lars Ruthotto, and Bart van Bloemen Waanders (2024). *Neural Network Approaches for Parameterized Optimal Control*. arXiv: 2402.10033 [math.OC]. URL: https://arxiv.org/abs/2402.10033.