PINN-Based BSDE Solvers: A Comparative Study of Deep Neural Networks for High-Dimensional Black-Scholes PDEs

Yitong Gong *

Abstract

This paper investigates the application of Backward Stochastic Differential Equation (BSDE) techniques within a Physics-Informed Neural Network (PINN) framework to solve high-dimensional Black-Scholes Partial Differential Equations (BS-PDEs). We compare the performance of several classical neural network architectures, including Fully Connected Neural Networks (FCNN), Recurrent Neural Networks (RNN), Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU), in their ability to approximate BSDE-driven PDE solutions. By incorporating PINN loss constraints, we enforce financial model consistency while leveraging the BSDE formulation to improve solution accuracy. Monte Carlo simulations are employed as a baseline to evaluate the accuracy and efficiency of these neural networks in approximating the PDE solutions. This study provides a comprehensive analysis of the strengths and the limitations of each architecture, highlighting their suitability for solving high-dimensional financial problems and contributing to the advancement of deep learning applications in ETF option hedging.

Keywords: BSDE; PINN; high-dimensional BS-PDE; option pricing; delta hedging; deep learning.

1 Introduction

In the past few decades, option pricing via solving partial differential equations (PDEs) is always in the centre of financial mathematics. However, traditional numerical methods such as finite difference methods and Monte Carlo simulations become computationally expensive in high-dimensional settings, due to the complexity of higher dimensionality [1, 2]. This limitation has become a significant challenge in pricing derivatives involving multiple assets, such as baskets or ETF/index options.

To improve computational efficiency, Backward Stochastic Differential Equations (BSDEs) have been invented as a powerful alternative. The theoretical foundation introduced by Pardoux and Peng [3] shows that many nonlinear PDEs can be equivalently reformulated as BSDEs, making it possible to solve PDEs by simulating stochastic processes forward in time and propagating value functions backward.

Based on BSDE, deep learning-based BSDE solvers were introduced as a practical tool for solving high-dimensional PDEs by approximating the option price Y_t and hedging process Z_t using neural networks [1]. This framework has inspired a wide range of extensions, including backward deep BSDE schemes for nonlinear dynamics [4], model-free control applications [5], and advanced architectures for improved scalability [6, 7].

However, despite their success, pure deep learning approaches may produce solutions that are not physically consistent with the underlying PDE unless additional structure is imposed. This motivates the incorporation of Physics-Informed Neural Networks (PINNs) which were introduced by Raissi et al. [8]. PINNs embed the residual of the governing PDE directly into the loss function, ensuring that the neural network not only fits data but also satisfies known physical or financial laws. Recent developments have further explored the synergy between PINNs and BSDEs, showing that interpolating between the two frameworks can yield more robust and interpretable solvers [9].

In this paper, we propose a comprehensive comparison of PINN-augmented deep BSDE solvers using several classical neural networks: Fully Connected Neural Networks (FCNN), Recurrent Neural Networks (RNN), Long Short-Term Memory

*Student Number: 16003146

(LSTM), and Gated Recurrent Units (GRU). We apply these models to estimate the solution of high-dimensional Black-Scholes PDEs, using Monte Carlo simulations as a baseline. Our goal is to evaluate their performance in terms of both accuracy and computational efficiency and to identify which architectures are more suitable for scalable option pricing without violating physical laws.

2 Methodology

2.1 Mathematical Backgrounds

Consider we have a d-dimensional asset portfolio, and the asset price vector $\mathbf{S}_t = (S_t^1, S_t^2, \dots, S_t^d)^T$ satisfies the geometric Brownian motion (GBM):

$$d\mathbf{S}_t = \boldsymbol{\mu} \odot \mathbf{S_t} dt + \boldsymbol{\sigma} \mathbf{S}_t \odot d\mathbf{W}_t^{\mathbb{P}}. \tag{1}$$

where

- $\boldsymbol{\mu} = (\mu_1, \mu_2, \dots, \mu_d)^T \in \mathbb{R}^d$ is the drift vector,
- $\boldsymbol{\sigma} \in \mathbb{R}^{d \times d}$ is the standard deviation / volatility matrix between assets, $\boldsymbol{\sigma}\boldsymbol{\sigma}^T$ is the covariance matrix,
- $\mathbf{W}_t^{\mathbb{P}} \in \mathbb{R}^d$ is a d-dimensional Brownian motion under physical measure $\mathbb{P}$ and $d\mathbf{W}_t^{\mathbb{P}}(d\mathbf{W}_t^{\mathbb{P}})^T = \mathbf{I}dt$.

2.1.1 High-Dimensional BS-PDE

Supposes the value of a derivative with the underlyings above is $V(t, \mathbf{S}_t)$ then it satisfies the high-dimensional Black Scholes partial differential equation

$$\frac{\partial V}{\partial t} + \sum_{i=1}^{d} r_i S^i \frac{\partial V}{\partial S^i} + \frac{1}{2} \sum_{i=1}^{d} \sum_{j=1}^{d} S^i S^j (\boldsymbol{\sigma}\boldsymbol{\sigma}^T)_{ij} \frac{\partial^2 V}{\partial S^i \partial S^j} - rV = 0 \tag{2}$$

where

- r_i is the discount factor, and usually $r_i = r$,
- the boundary condition is $h(\mathbf{S}_T) = V(T, \mathbf{S}_T)$, i.e. the payoff function at maturity.

However, there are no analytic solutions for high-dimensional BS-PDE in general. Instead, numerical methods are used to estimate the solution of PDEs, such as Monte Carlo simulation.

2.1.2 Backward Stochastic Differential Equation (BSDE)

Let $Y_t = V(t, \mathbf{S}_t)$, by applying the Itô's lemma, we can write

$$dY_t = \left(\frac{\partial V}{\partial t} + \sum_{i=1}^{d} \mu_i S_t^i \frac{\partial V}{\partial S_t^i} + \frac{1}{2} \sum_{i=1}^{d} \sum_{j=1}^{d} S_t^i S_t^j (\boldsymbol{\sigma}\boldsymbol{\sigma}^T)_{ij} \frac{\partial^2 V}{\partial S_t^i \partial S_t^j} \right) dt + \sum_{i=1}^{d} \sum_{j=1}^{d} \boldsymbol{\sigma}_{ij} S_t^i \frac{\partial V}{\partial S_t^i} dW_t^{j,\mathbb{P}} \tag{3}$$

Substitute (2) into (3), we can obtain

$$dY_t = \left(rY_t - \sum_{i=1}^{d} (r_i - \mu_i) S_t^i \frac{\partial V}{\partial S_t^i} \right) dt + \sum_{i=1}^{d} \sum_{j=1}^{d} \boldsymbol{\sigma}_{ij} S_t^i \frac{\partial V}{\partial S_t^i} dW_t^{j,\mathbb{P}}. \tag{4}$$

To obtain the backward stochastic differential equation, Girsanov theorem and Feynman-Kac formula should be introduced first.

Theorem 1 *(Girsanov theorem) [10] Suppose W_t is a standard Brownian motion under the probability space $\{\Omega, \mathcal{F}, \mathbb{P}\}$. Given a process θ_t, the process*

$$L_t = \exp\left(-\frac{1}{2} \int_0^t \theta_s^2 ds - \int_0^t \theta_s dW_s \right)$$

is an $(\mathcal{F}_t)$-martingale. If we define the probability measure $\mathbb{Q}$ on $(\Omega, \mathcal{F}_T)$ by

$$\mathbb{Q}(A) = \mathbb{E}^{\mathbb{P}}[L_T \mathbf{1}_A] \quad \textit{for } A \in \mathcal{F}_T,$$

then the process W_t^θ defined by

$$W_t^\theta = W_t + \int_0^t \theta_s dW_s$$

is a standard $(\mathcal{F}_t)$-Brownian motion with respect to the probability measure $\mathbb{Q}$.

Theorem 2 *(Feynman-Kac) [11] Consider the partial differential equation*

$$\frac{\partial u}{\partial t} + \mathcal{L}u - ru + f = 0, \quad (x,t) \in \mathbb{R}^d \times [0,T] \tag{5}$$

subject to the terminal condition $u(x,T) = g(x)$, where the parabolic operator $\mathcal{L}$ is defined as

$$\mathcal{L}u = \sum_{i=1}^d \mu_i(x,t)\frac{\partial u}{\partial x_i} + \frac{1}{2}\sum_{i,j=1}^d (\sigma\sigma^T)_{ij}\frac{\partial^2 u}{\partial x_i \partial x_j}.$$

Then the Feynman-Kac formula expresses $u(x,t)$ as a conditional expectation under the probability measure $\mathbb{Q}$

$$u(x,t) = \mathbb{E}\left[\int_t^T e^{-r(s-t)} f(\mathbf{X}_s, s)ds + e^{-r(T-t)}g(\mathbf{X}_T)|\mathbf{X}_t = \mathbf{x}\right] \tag{6}$$

where $\mathbf{X}_t = (X_t^1, X_t^2, \dots, X_t^d)^T \in \mathbb{R}^d$ is an Itô process satisfying

$$d\mathbf{X}_t = \mu(\mathbf{X}_t, t)dt + \sigma(\mathbf{X}_t, t)d\mathbf{W}_t^{\mathbb{Q}}$$

and $W_t^{\mathbb{Q}}$ a Brownian motion under $\mathbb{Q}$.

Let $Z_t^j = \sum_{i=1}^d \boldsymbol{\sigma}_{ij} S_t^i \frac{\partial V}{\partial S_t^i}$, where $\frac{\partial V}{\partial S_t^i}$ represents the sensitivity to the i-th asset, a.k.a. the Greek delta (Δ). According to Girsanov Theorem, we can obtain the form of backward stochastic differential equation

$$dY_t = -rY_t dt + \mathbf{Z}_t^T d\mathbf{W}_t^{\mathbb{Q}}, \quad Y_T = h(\mathbf{S}_T) \tag{7}$$

By Feynman-Kac formula, it is easy to prove that the solution of high-dimensional BS-PDE can be written as

$$Y_t = V(t, \mathbf{S}_t) = e^{-r(T-t)}\mathbb{E}^{\mathbb{Q}}[h(\mathbf{S}_T)|\mathbf{S}_t] \tag{8}$$

where $\mathbb{Q}$ is the risk-free measure, and under this measure, the asset prices $\mathbf{S}_t$ follow

$$dS_t^i = rS_t^i dt + \sum_{j=1}^d \sigma_{ij} S_t^i dW_t^{j,\mathbb{Q}} \tag{9}$$

Then, finding the estimation of Y_t, Z_t by (7) and (8) will not only figure out the derivative price but the delta hedging strategy (via Z_t).

Overall, we can use numerical methods to estimate the solutions of a high-dimensional BS-PDE according to equation (8). Monte Carlo is one of them because of its ability to implement and stationary with more simulations. However, as the dimensionality exceeds some level, to ensure the estimating accuracy, the MC simulation methods will need larger sample size which will be computationally expensive. To level up the computational efficiency, deep BSDE solver, which shows high performance in high-dimensional deduction, will be introduced in the following content.

2.2 Physics-Informed Neural Networks (PINN)

Physics-Informed Neural Networks (PINNs), which were introduced by Raissi et al. [8], are a class of deep learning models that obey physical laws which are described by Partial Differential Equations (PDEs) or Ordinary Differential Equations

(ODEs). Instead of relying entirely on data, PINNs enforce physical constraints by implementing differential equation residuals into the loss function.

Unlike conventional supervised learning, where neural networks are trained solely on labeled data, PINNs leverage automatic differentiation to compute the residuals of governing equations (e.g., Navier–Stokes, Black–Scholes, Schrödinger) and enforce them as soft constraints during training. This makes them particularly powerful in data-scarce regimes or high-dimensional problems, where traditional numerical solvers struggle due to the curse of dimensionality. For a PDE of the form

$$\frac{\partial u}{\partial t} + \mathcal{L}u(x,t) = f(x,t),$$

where $\mathcal{L}$ is a differential operator, PINNs define a loss function that penalized violations of the equation

$$\mathcal{L}(\theta) = \frac{1}{N_f}\sum_{i=1}^{N_f}\left|\frac{\partial u_\theta}{\partial t}(x_i,t_i) + \mathcal{L}u_\theta(x_i,t_i) - f(x_i,t_i)\right|^2 + \cdots, \tag{10}$$

where the residual is evaluated at data points (x_i, t_i), and additional terms (e.g., initial or boundary conditions) are appended as needed. This formulation allows the neural network to learn solutions which are consistent with both the data and the physical laws.

2.3 Deep Neural Networks

A Deep Neural Network (DNN) is an artificial neural network formed by multiple layers, including an input layer, several hidden layers, and an output layer. By implementing multiple nonlinear transformations, DNNs are able to capture complex patterns and further features in data. They serve as main concept of deep learning and are widely used in fields such as image recognition, natural language processing, and quantitative finance.

In this section, five classical deep neural networks will be introduced to learn the parameterized Y_t, Z_t in deep BSDE solver.

Name	**Structure**	**Mathematical Expression**	**Variable Explanation**
FCNN [12]	Dense layers	$f_\theta(x) = W_L\sigma_{L-1}(\cdots\sigma_1(W_1x+b_1)+\cdots)+b_L$	x: input; W_l, b_l: weights/biases; σ_l: activation; f_θ: output
RNN [13]	Recurrent cell	$h_t = \sigma(W_{xh}x_t + W_{hh}h_{t-1} + b_h)$ $y_t = W_{hy}h_t + b_y$	x_t: input; h_t: hidden state; y_t: output; W_*, b_*: parameters
LSTM [14]	Memory + gates	$f_t = \sigma(W_fx_t + U_fh_{t-1} + b_f)$, $i_t = \sigma(W_ix_t + U_ih_{t-1} + b_i)$ $o_t = \sigma(W_ox_t + U_oh_{t-1} + b_o)$, $\tilde{c}_t = \tanh(W_cx_t + U_ch_{t-1} + b_c)$ $c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$, $h_t = o_t \odot \tanh(c_t)$	f_t, i_t, o_t: gates; $\tilde{c}_t$: candidate state; c_t: cell; h_t: output; $\odot$: element-wise product
GRU [15]	Simplified LSTM	$z_t = \sigma(W_zx_t + U_zh_{t-1} + b_z)$, $r_t = \sigma(W_rx_t + U_rh_{t-1} + b_r)$ $\tilde{h}_t = \tanh(W_hx_t + U_h(r_t \odot h_{t-1}) + b_h)$ $h_t = (1-z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t$	z_t, r_t: gates; $\tilde{h}_t$: candidate; h_t: output; other symbols as above

Table 1: General formulae of four neural network architectures and corresponding variable notations

Aspect	FCNN	RNN	LSTM	GRU
Structure	Fully connected layers	Recurrent with hidden state	Recurrent with memory and gates	Recurrent with simplified gates
Suitable for	General input data	Sequential data (short-term)	Long-term sequence modeling	Efficient sequence modeling
Time Dependency	Not supported	Supported (short)	Supported (long)	Supported (moderate)
Parameter Sharing	No sharing	Shared over time steps	Shared over time steps	Shared over time steps
Memory Mechanism	None	Hidden state only	Cell state with gates	Gating mechanism only
Vanishing Gradient Handling	Not handled	Partially handled	Well handled via gates	Handled via gating units
Parameter Complexity	Low to moderate	Moderate	High (multiple gates)	Moderate (fewer gates)
Training Efficiency	Fast training	Moderate training speed	Slower due to complexity	Faster than LSTM

Table 2: Comparison of four classical neural network architectures

2.4 Model Training

In this section, we will introduce the components of the proposed PINN-augmented deep BSDE solver. Based on the mathematical framework in Section 2.1 and the neural architectures discussed in Section 2.3, the deep BSDE solver are asked to approximate the solution pair (Y_t, Z_t) to the BSDE (7) by applying the deduction efficiency of deep neural networks and the regularizing effect of physics-informed loss functions. Furthermore, the solver takes as input the time $t \in \mathbb{R}$ and the current asset state $S_t \in \mathbb{R}^d$, and outputs predictions for both the option value Y_t and the hedging strategy Z_t. The neural network is trained to minimize a weighted loss function that includes the terminal condition residual, the BSDE dynamics residual along sample paths, and the PDE-based physics-informed penalty. This section describes the overall architecture, data generation process, loss formulation, and optimization settings used in our experiments.

2.4.1 Network Structure

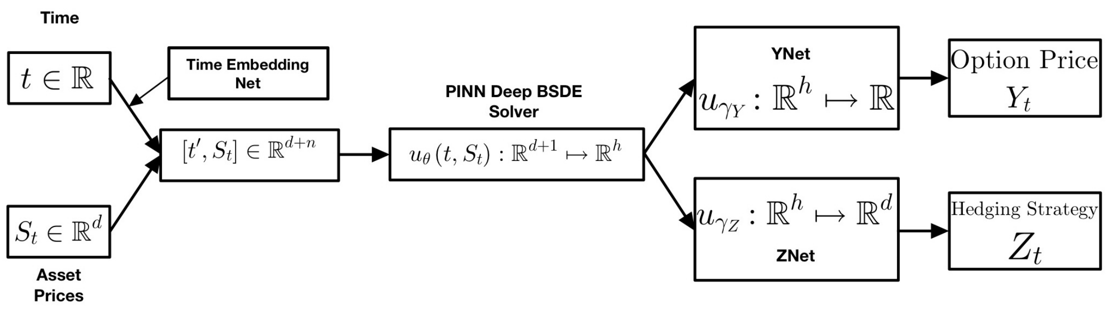

Time
$t \in \mathbb{R}$
Time Embedding Net
$[t', S_t] \in \mathbb{R}^{d+n}$
$S_t \in \mathbb{R}^d$
Asset Prices
PINN Deep BSDE Solver
$u_\theta(t, S_t) : \mathbb{R}^{d+1} \mapsto \mathbb{R}^h$
YNet
$u_{\gamma_Y} : \mathbb{R}^h \mapsto \mathbb{R}$
$u_{\gamma_Z} : \mathbb{R}^h \mapsto \mathbb{R}^d$
ZNet
Option Price Y_t
Hedging Strategy Z_t

Figure 1: The Structure of Whole PINN-augmented Deep BSDE Solver

As shown in figure 1, the entire deep BSDE solver architecture is divided into four components. The model takes as input the time t and asset prices S_t, resulting in a combined input vector of size $d + 1$. The core component of the framework is the PINN-based deep BSDE solver, which transforms the input into a latent representation (signal vector) of size h using one of the five neural network architectures discussed in Section 2.3. This latent vector is then mapped to the option value Y_t and the hedging strategy Z_t via two separate multiple layer perceptrons with two hidden layers, denoted as YNet and ZNet, respectively. In this architecture, the trainable parameters consist of the neural network parameters θ, and the parameters of the MLPs: γ_Y for Y_t, and γ_Z for Z_t.

To obtain a fair comparison among the five neural network architectures, we will control each model to have approximately the same total trainable parameter counts. This will ensure that the main performance differences come from the structural efficiency rather than the model capacities under the PINN deep BSDE solver framework. The detailed model structures for **PINN Deep BSDE Solver** part are shown as follows

Network	**Assets Dim**	**Hidden Size m**	**Output Dim h**	**Layers**	**Parameter Counts (Symbolic)**	**Parameter Counts (Fixed $h = 10$)**
FCNN	d	36	10	3 hidden layers	$(d+1)m + 2(m+1)m + (m+1)h$	$36d + 3106$
RNN	d	30	10	2 RNN layers	$m(d+m+2) + m(2m+1) + mh + h$	$30d + 3100$
LSTM	d	25	10	1 LSTM layer	$4m(d+m+2) + mh + h$	$100d + 2960$
GRU	d	28	10	1 GRU layer	$3m(d+m+2) + mh + h$	$84d + 2810$

Table 3: Architecture settings for five neural networks architectures with fixed output dimension $h = 10$

The activation in this class of architectures will be **Softplus** since Hessian matrix will be calculated in the PINN loss function and Softplus can avoid the gradients vanishing problem.

2.4.2 Data Generating

To train the PINN-augmented deep BSDE solvers, we generate a dataset containing one thousand simulated random portfolio routes based on the risk-neutral SDE (9) with a fixed time horizon $[0, T]$ with discrete time steps, a fixed risk-free rate r and a fixed standard deviation matrix $\boldsymbol{\sigma}$ under risk-neutral measure $\mathbb{Q}$. In the simulations, we assume that the Brownian motions are sampled independently for each path.

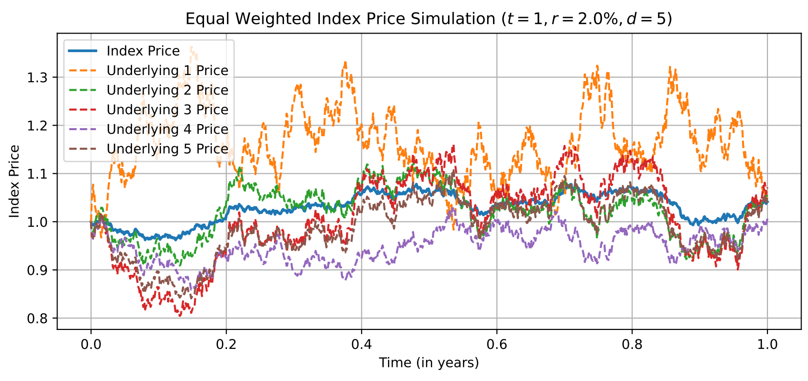


Figure 2: Sample routes of a equal weighted index and its components

2.4.3 Loss Functions

The loss function is a weighted sum of three components, loss of dynamic residues $\mathcal{L}_{\text{dynamic}}$, loss of terminal condition $\mathcal{L}_{\text{terminal}}$ and loss of PINN residues $\mathcal{L}_{\text{PINN}}$.

- **Dynamic Residue**: it enforces consistency along sample routes and is defined as follows

$$\mathcal{L}_{\text{dynamic}} = \frac{1}{N} \sum_{i=0}^{N-1} \left| Y_\phi(t_i, \mathbf{S}_{t_i}) - rY_\phi(t_i, \mathbf{S}_{t_i})\Delta t + \mathbf{Z}_{t_i}^T \Delta \mathbf{W}_{t_i} - Y_\phi(t_{i+1}, \mathbf{S}_{t_{i+1}}) \right|^2 \tag{11}$$

- **Terminal Condition**: it ensures the predicted price matches the payoff at maturity and is defined as follows

$$\mathcal{L}_{\text{terminal}} = \frac{1}{M}\sum_{i=1}^{M}\left|Y_\phi(T, \mathbf{S}_T^{(i)}) - g(S_T^{(i)})\right|^2, \text{ where } M \text{ is the sample/batch size} \tag{12}$$

- **PINN Residue / PDE Residue**: it embeds physical-informed constraints into the training process and is defined as follows

$$\mathcal{L}_{\text{PINN}} = \frac{1}{N}\sum_{i=1}^{N}|\frac{\partial Y_\phi}{\partial t} + \mathcal{L}Y_\phi - rY_\phi|^2 \tag{13}$$

- **Total Loss**

$$\mathcal{L}_{\text{total}} = \lambda_{\text{dynamic}}\mathcal{L}_{\text{dynamic}} + \lambda_{\text{terminal}}\mathcal{L}_{\text{terminal}} + \lambda_{\text{PINN}}\mathcal{L}_{\text{PINN}}, \tag{14}$$

where $\phi = \{\theta, \gamma_Y, \gamma_Z\}$ and $\lambda_{\text{dynamic}}, \lambda_{\text{terminal}}, \lambda_{\text{PINN}}$ are self-defined loss weights. In the following experiments, we will fix $\lambda_{\text{dynamic}} = 1, \lambda_{\text{terminal}} = 0.1, \lambda_{\text{PINN}} = 10$

2.4.4 Optimizer

In the following experiments, we will use Adam optimizer [16, 17] to train the model with at most 10000 epoches since it combines the advantages of momentum and adaptive learning rates and is suitable for optimizing high-dimensional function in deep learning task such as PINN-augmented deep BSDE solvers.

2.4.5 Adjustments to RNN Family

Due to the fact that CUDA does not support efficient second-order derivative computation for RNN-family architectures, directly applying PINN (Physics-Informed Neural Networks) losses during training with these models leads to either computational failure or extreme inefficiency. To address this issue, we adopt a two-stage hybrid training strategy that decouples temporal learning from PDE constraint enforcement.

In the first stage, we employ a RNN-based deep BSDE solver as the backbone network and train the model using only the dynamic loss (which enforces the BSDE evolution through Monte Carlo trajectories) and the terminal condition loss. The goal of this stage is to allow the model, particularly the YNet and ZNet components, to effectively learn the time-dependent structure of the solution under the guidance of real stochastic paths. Importantly, PINN loss is not included in this phase to avoid the need for second-order gradients, which are incompatible with cuDNN-accelerated RNNs.

Once the GRU-based model is trained, we replace the GRU encoder with a fully connected feedforward neural network (FCNN) that is smooth and differentiable. This FCNN will serve as the new backbone for the second stage. At this point, the previously trained YNet and ZNet are frozen to preserve the time-adaptive knowledge acquired in the first phase. In the second training stage, we introduce the PINN loss to enforce the solution's consistency with the associated partial differential equation (PDE), by minimizing residuals involving both first and second-order derivatives with respect to time and asset prices.

This hybrid strategy allows the model to benefit from the temporal representation capabilities of GRU during early training, while leveraging the differentiability and global approximation ability of FCNN during PDE-constrained optimization. It ensures both learning efficiency and solution accuracy under physical constraints.

2.4.6 Baseline Model

We will train our models under two scenarios, one-dimensional and multidimensional cases. In one dimensional test, Black-Scholes model will be implemented to figure out the precise result and Monte Carlo will also be used for comparison. In multidimensional case, we will only use Monte Carlo simulations as our baseline model.

3 Results

Our experiments were run on a machine equipped with an Intel Core i9-14900HX CPU and a single NVIDIA RTX 4060 GPU (8GB). The code was implemented in PyTorch 2.6.0 with CUDA 12.6. All results were obtained using fixed random seeds to ensure reproducibility.

In the experiments, we trained the model to price the option with the following payoff function at maturity

$$g(\mathbf{S}_T) = \max(\frac{1}{d}\sum_{i=1}^{d} S_T^d - K, 0) \tag{15}$$

which is a vanilla arithmetic mean basket call option, and we set $K = 99$, $\mathbf{S}_0 = 100 \cdot \mathbf{1}_d$, $\boldsymbol{\sigma} = 10\% \cdot I_d$ and $T = 1/12$ which will be divided into 10 discrete time steps.

3.1 One Dimensional Case (d = 1)

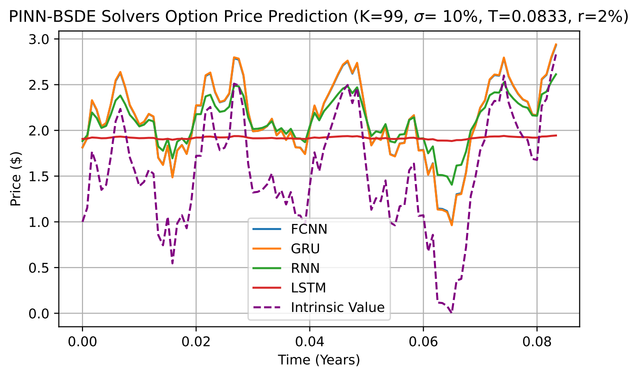
In the one-dimensional setting, we compare PINN-augmented BSDE solvers against the analytical Black-Scholes solution and Monte Carlo simulation. The Black-Scholes model serves as the theoretical benchmark. Table 4 shows that FCNN and GRU achieve the lowest error (-0.34% and -0.36%, respectively), while RNN and LSTM overestimate the option value by more than 3%.

As shown in Figure 3a, we compare the approximation capabilities of four different network architectures within the PINN-augmented deep BSDE framework for vanilla European option pricing. FCNN and GRU significantly outperform RNN and LSTM, yielding smoother and more accurate prediction curves that adhere to the practical financial principle: option prices are typically higher than their intrinsic value and converge toward it as time approaches maturity. These observations suggest that FCNN and GRU are more suitable for producing efficient and physically consistent solutions in one-dimensional option pricing problems.

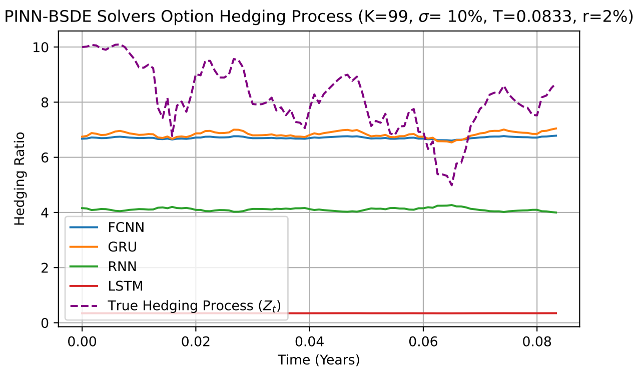
Figure 3b compares the learned hedging strategies (Z_t) across various PINN-based BSDE solvers. FCNN and GRU closely match the true delta hedging curve, indicating strong consistency with theoretical expectations. RNN and LSTM show higher deviations which may affect their reliability in real-world hedging applications.

Method	**BS Model**	**Monte Carlo (10K)**	**FCNN**	**GRU**	**RNN**	**LSTM**
Price	1.8204	1.8544	1.8142	1.8137	1.8836	1.9068
Error Rate (%)	0.0000	1.8697	-0.3374	-0.3651	3.4760	4.7493

Table 4: Option pricing and error rate comparison under different models at t = 0



(a) Comparison of price prediction



(b) Comparison of hedge processes

Figure 3: Comparison of one-dimensional PINN price and hedging process predictions

3.2 High Dimensional Case

In the high-dimensional setting ($d = 5$), we expand the evaluation to a five-asset arithmetic mean basket option. Since there is no analytic solution in this case, we use a high-precision Monte Carlo simulation with 10^6 samples as the ground truth. An MC simulation with fewer samples (10K) is also included to assess the sample accuracy of PINN-based Models.

Among the neural networks, LSTM achieves the lowest error rate (+0.34%) for estimating the theoretical price at initial point, followed closely by RNN (+0.77%) and GRU (+0.88%). FCNN, while effective in the one-dimensional case, is slightly poor here with a -1.08% deviation. These results suggest that recurrent models exhibit stronger generalization in high-dimensional, time-dependent financial systems, likely due to their memory mechanisms and parameter-sharing capabilities across time steps.

Figure 4 illustrates the predicted pricing curves over time for a representative sample path. Despite accurate point estimates at $t = 0$, LSTM and RNN fail to capture the temporal dynamics of option values across the pricing horizon. In contrast, GRU and FCNN demonstrate better alignment with the theoretical time evolution, providing more meaningful representations of the underlying price diffusion process.

Method	**Monte Carlo (1M)**	**Monte Carlo (10K)**	**FCNN**	**GRU**	**RNN**	**LSTM**
Price	1.3277	1.3189	1.3134	1.3394	1.3380	1.3322
Error Rate (%)	0.0000	-0.6653	-1.0770	0.8782	0.7750	0.3409

Table 5: Option pricing and error rate comparison under different models at t = 0 (5D Case)

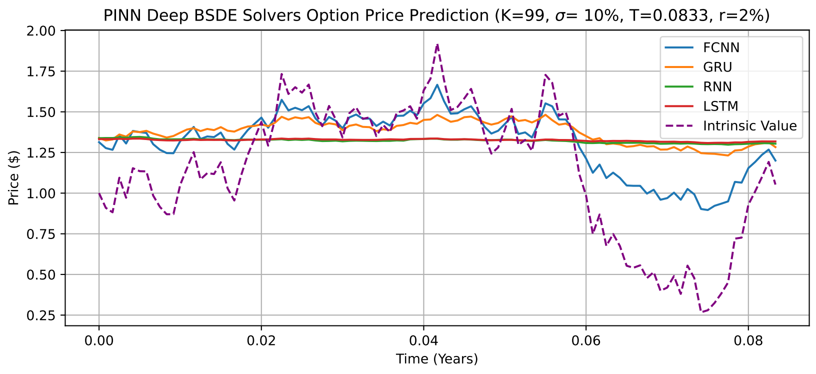


Figure 4: Comparison of 5-dimensional PINN predictions on testset with more timesteps

4 Discussion

Applicability Across Moneyness Regimes. While the proposed framework is theoretically applicable to all moneyness regimes, empirical results suggest that PINN-based BSDE solvers perform best in the at-the-money (ATM) region. This is because the option price reaches the highest curvature near ATM, providing more gradient information for training. In contrast, deep in-the-money or out-of-the-money options have flatter payoff surfaces and near-constant Delta values, which may lead to vanishing gradients and underfitting. These practical factors will limit the implementation of PINN-based BSDE solver in DITM/DOTM option pricing.

Limitations of RNN-family Layers under PINN Framework. One major limitation of implementing RNN-family architectures (such as RNN, GRU, and LSTM) into the PINN-BSDE framework is their incompatibility with second-order gradient computations. In PyTorch, these layers are typically executed via cuDNN backends, which offer efficient forward and backward operations but do not preserve the full computation graph necessary for higher-order differentiation. As a result, it becomes difficult to compute the second-order derivatives required in PINN residual loss, especially for PDE terms involving $\partial^2 V/\partial S^2$. This technical constraint makes hybrid training strategies or architectural modifications essential for RNN-based models.

Future Exploration of Transformer Architectures. To address the limitations of RNN-family layers under the PINN-BSDE framework, future research may explore the integration of Transformer-based sequence models. Transformers offer global attention mechanisms that enable parallel modeling of temporal dependencies and are fully compatible with higher-order differentiation, making them ideal candidates for PINN training involving second-order PDE constraints. Moreover, their scalability to high-dimensional input spaces and ability to model complex temporal-asset interactions make them well suited for advanced financial derivatives, such as basket or path-dependent options. Recent developments in physics-informed Transformers and neural operators further support this direction, offering promising avenues for accurate and efficient high-dimensional PDE solvers in quantitative finance.

Suitability to High-Dimensional Hedging Instruments. An important advantage of the PINN-BSDE framework is its natural suitability for modeling high-dimensional derivative products, such as basket options, ETF options, and index options. These instruments are written on portfolios of correlated assets, which introduce substantial computational challenges for traditional numerical methods due to the curse of dimensionality. In contrast, neural network-based solvers can efficiently scale with dimensionality through parameter sharing. Moreover, the PINN formulation enables simultaneous learning of both the option value and its sensitivity (i.e., Delta), providing a unified and differentiable framework for constructing real-time hedging strategies. This makes PINN-BSDE solvers particularly appealing for modeling and managing risk in portfolio-level financial instruments.

References

[1] J. Han, A. Jentzen, and W. E, "Solving high-dimensional partial differential equations using deep learning," *Proceedings of the National Academy of Sciences*, vol. 115, p. 8505–8510, Aug. 2018.

[2] P. Glasserman, *Monte Carlo Methods in Financial Engineering.* Stochastic Modelling and Applied Probability, Springer New York, 2013.

[3] E. Pardoux and S. Peng, "Adapted solution of a backward stochastic differential equation," *Systems & Control Letters*, vol. 14, pp. 55–61, 1990.

[4] Y. Yu, B. Hientzsch, and N. Ganesan, "Backward deep bsde methods and applications to nonlinear problems," 2020.

[5] Y. Wang and Y.-H. Ni, "Deep bsde-ml learning and its application to model-free optimal control," 2022.

[6] L. Kapllani and L. Teng, "Deep learning algorithms for solving high-dimensional nonlinear backward stochastic differential equations," *Discrete and Continuous Dynamical Systems - B*, vol. 29, no. 4, p. 1695–1729, 2024.

[7] D. Bussell and C. A. García-Trillos, "Deep multi-step mixed algorithm for high dimensional non-linear pdes and associated bsdes," 2023.

[8] M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations." Online, 2017.

[9] N. Nusken and L. Richter, "Interpolating between bsdes and pinns: Deep learning for elliptic and parabolic boundary value problems," *Journal of Machine Learning*, vol. 2, pp. 31–64, mar 2023.

[10] Wikipedia contributors, "Girsanov theorem — Wikipedia, the free encyclopedia," 2025. [Online; accessed 22-March-2025].

[11] Wikipedia contributors, "Feynman–kac formula — Wikipedia, the free encyclopedia," 2024. [Online; accessed 22-March-2025].

[12] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, no. 5, pp. 359–366, 1989.

[13] J. L. Elman, "Finding structure in time," *Cognitive Science*, vol. 14, no. 2, pp. 179–211, 1990.

[14] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[15] K. Cho, B. van Merriënboer, Çaglar Gülçehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder–decoder for statistical machine translation," 2014.

[16] Y. LeCun, L. Bottou, G. B. Orr, and K. R. Müller, *Efficient BackProp*, pp. 9–50. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998.

[17] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2017.