WORK SUMMARY

ANNIE ZHANG

ABSTRACT. This work explores neural network-based PDE solvers for high-dimensional financial models, integrating three approaches: Deep Backward Stochastic Differential Equations (Deep BSDE) [1], Deep Galerkin Methods (DGM) [2], and Sliced Mixture Density Networks (Sliced MDN). Deep BSDE reformulates nonlinear parabolic PDEs as backward stochastic processes, enabling efficient simulation and optimization in high dimensions. DGM is applied to both backward Kolmogorov and Fokker-Planck equations [3], providing flexible solutions for option pricing, density estimation, and sensitivity analysis in stochastic volatility models such as SABR. We also employed Sliced MDN [4] with reference points to approximate probability densities, converting grid-sampled estimations into continuous functions. Together, these methods form a unified framework for solving PDEs in derivative pricing and risk management, demonstrating strong accuracy and scalability.

Keywords. Machine Learning, PDE, Option Pricing, Kolmogorov Equation, Fokker-Planck Equation, Feynman–Kac

1. DEEP BACKWARD STOCHASTIC DIFFERENTIAL EQUATIONS (DEEP BSDE) [1]

In this section, we apply the Deep BSDE method to solve nonlinear parabolic PDEs arising in derivative pricing. We begin by presenting the general theory and the corresponding numerical scheme. Next, we demonstrate its application first to the Black–Scholes model and then to the SABR stochastic volatility model. Finally, we report numerical benchmarks that validate the accuracy of our approach.

1.1. **General Method.** We begin with a general semilinear parabolic partial differential equation (PDE) of the form:

$$\begin{aligned}\frac{\partial u}{\partial t}(t,x) &+ \frac{1}{2}\mathrm{Tr}\left[\sigma\sigma^{\top}(t,x)\cdot \mathrm{Hess}_x u(t,x)\right] \\ &+ \nabla u(t,x)\cdot\mu(t,x) + f\big(t,x,u(t,x),\sigma^{\top}(t,x)\nabla u(t,x)\big) = 0,\end{aligned} \tag{1.1}$$

subject to the terminal condition:

$$u(T,x) = g(x). \tag{1.2}$$

Equation (1.1) represents a broad class of PDEs encountered in derivative pricing, where $u(t,x)$ typically represents the option price as a function of time t and underlying asset prices x. The various components have clear financial interpretations:

- $u : [0, T] \times \mathbb{R}^d \to \mathbb{R}$ is the unknown solution (e.g., option price),
- $\mu : [0, T] \times \mathbb{R}^d \to \mathbb{R}^d$ represents the drift terms of underlying assets,
- $\sigma : [0, T] \times \mathbb{R}^d \to \mathbb{R}^{d\times m}$ captures volatility and correlation structure,
- $f : [0, T] \times \mathbb{R}^d \times \mathbb{R} \times \mathbb{R}^m \to \mathbb{R}$ is a nonlinear generator (often related to discounting),
- $g : \mathbb{R}^d \to \mathbb{R}$ defines the payoff at maturity (e.g., $\max(S - K, 0)$ for a call option).

The BSDE Connection: From PDEs to Stochastic Processes. The key insight behind Deep BSDE is the Feynman-Kac theorem, which establishes a fundamental connection between PDEs and stochastic differential equations. This connection allows us to replace the challenging task of solving PDE (1.1) with the more manageable problem of simulating stochastic processes.
We first define a forward stochastic differential equation (SDE) that describes the evolution of the underlying state variables:

$$dX_t = \mu(t, X_t)dt + \sigma(t, X_t)dW_t, \quad X_0 = \xi, \tag{1.3}$$

where W_t is an m-dimensional Brownian motion representing market randomness, and ξ is the initial state.

The mathematical bridge between the PDE (1.1) and SDE (1.3) is provided by Itô's formula. When we apply this fundamental stochastic calculus result to the process $u(t, X_t)$, we obtain:

$$\begin{aligned} du(t, X_t) = \Bigg[&\frac{\partial u}{\partial t}(t, X_t) + \nabla u(t, X_t)^\top \mu(t, X_t) \\ &+\frac{1}{2}\mathrm{Tr}\left(\sigma\sigma^\top(t, X_t) \cdot \mathrm{Hess}_x u(t, X_t)\right)\Bigg] dt \\ &+ \nabla u(t, X_t)^\top \sigma(t, X_t)dW_t. \end{aligned} \tag{1.4}$$

Now comes the crucial step: we can substitute PDE (1.1) into equation (1.4) to eliminate the drift term, yielding the backward stochastic differential equation:

$$\begin{aligned} du(t, X_t) = &-f\left(t, X_t, u(t, X_t), \sigma^\top(t, X_t)\nabla u(t, X_t)\right) dt \\ &+ \nabla u(t, X_t)^\top \sigma(t, X_t)dW_t. \end{aligned} \tag{1.5}$$

Equation (1.5) is remarkable because it expresses the PDE solution as a stochastic process that can be simulated via Monte Carlo methods. In integral form, this becomes:

$$\begin{aligned} u(t, X_t) - u(0, X_0) = &-\int_0^t f\left(s, X_s, u(s, X_s), \sigma^\top(s, X_s)\nabla u(s, X_s)\right) ds \\ &+ \int_0^t \nabla u(s, X_s)^\top \sigma(s, X_s)dW_s. \end{aligned} \tag{1.6}$$

Numerical Implementation: Making it Computationally Tractable. While equation (1.6) provides the theoretical foundation, practical implementation requires careful discretization. We partition the time interval $[0, T]$ into N uniform steps:

$$0 = t_0 < t_1 < \cdots < t_N = T, \quad \Delta t = \frac{T}{N}. \tag{1.7}$$

The forward SDE (1.3) is simulated using the standard Euler-Maruyama numerical scheme:

$$X_{n+1} = X_n + \mu(t_n, X_n)\Delta t + \sigma(t_n, X_n)\Delta W_n, \tag{1.8}$$

where $\Delta W_n \sim \mathcal{N}(0, \Delta t \cdot I_m)$ represents the Brownian increments.

The backward equation (1.5) is similarly discretized as:

$$\begin{aligned} u(t_{n+1}, X_{n+1}) = u(t_n, X_n) &- f\left(t_n, X_n, u(t_n, X_n), \sigma^\top(t_n, X_n)\nabla u(t_n, X_n)\right)\Delta t \\ &+ \nabla u(t_n, X_n)^\top \sigma(t_n, X_n)\Delta W_n. \end{aligned} \tag{1.9}$$

The Neural Network The fundamental challenge in implementing equation (1.9) is that the gradients $\nabla u(t_n, X_n)$ are unknown—they are precisely what we're trying to solve for! This is where the Deep BSDE method introduces its key innovation: we approximate these unknown gradients using neural networks. Specifically, we define:

$$\nabla u(t_n, X_n) \approx G_{\theta_n}(X_n), \tag{1.10}$$

where G_{θ_n} is a feedforward neural network with parameters θ_n for each time step n.

The complete neural network architecture consists of:

- **Initial value network:** $U_\phi(x) \approx u(0, x)$ approximates the solution at time zero, with parameters ϕ.
- **Gradient networks:** $\{G_{\theta_n}\}_{n=0}^{N-1}$ approximate the gradients at each discrete time step.

Training Strategy: Learning from Terminal Conditions. The training process leverages the known terminal condition (1.2). We simulate the forward process using equations (1.8), then use our neural network approximations in equation (1.9) to compute:

$$\begin{aligned} Y_0 &= U_\phi(X_0), \\ Y_{n+1} &= Y_n - f(t_n, X_n, Y_n, Z_n)\Delta t + Z_n^\top \Delta W_n, \\ Z_n &= G_{\theta_n}(X_n), \end{aligned} \tag{1.11}$$

where Y_n approximates $u(t_n, X_n)$ and Z_n approximates $\nabla u(t_n, X_n)$.

The network parameters $\Theta = \{\phi, \theta_0, \theta_1, \ldots, \theta_{N-1}\}$ are trained by minimizing the mean squared error between our predicted terminal value and the known payoff:

$$\mathcal{L}(\Theta) = \mathbb{E}\left[|Y_N - g(X_N)|^2\right]. \tag{1.12}$$

The loss function (1.12) is minimized using standard optimization techniques such as Adam, with the expectation approximated through Monte Carlo sampling. This approach effectively ”learns” the PDE solution by ensuring that our neural network-based simulation reproduces the correct terminal payoffs across many randomly sampled paths.

Why This Approach Works. The Deep BSDE method succeeds because it transforms a high-dimensional PDE—which traditionally requires solving on expensive grids that scale exponentially with dimension—into a machine learning problem that scales much more favorably. The method learns the essential features of the solution (the gradients) needed to simulate the stochastic process accurately, rather than trying to represent the entire solution function explicitly.

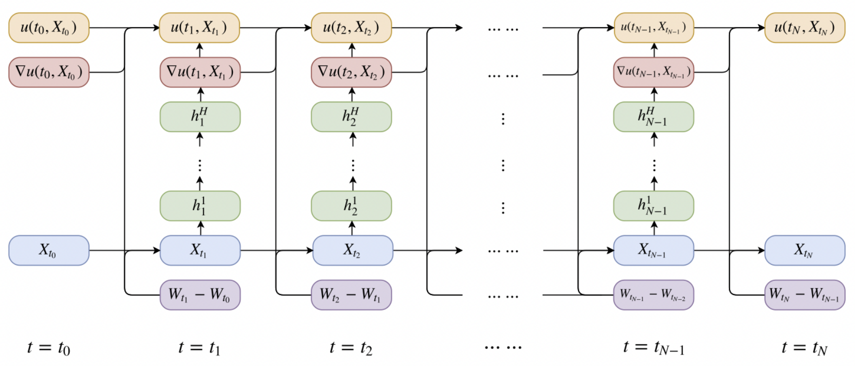


FIGURE 1. The neural network architecture used in the Deep BSDE method.

1.2. Deep BSDE for the Black–Scholes Model.

The Black–Scholes PDE for a European call option is:

$$\frac{\partial u}{\partial t}(t,S) + rS\frac{\partial u}{\partial S}(t,S) + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 u}{\partial S^2}(t,S) = ru(t,S), \tag{1.13}$$

with terminal condition:

$$u(T,S) = \max(S-K, 0). \tag{1.14}$$

Here:

- $u(t,S)$ is the option price at time t and stock price S,
- r is the risk-free interest rate,
- σ is the volatility of the underlying asset,
- K is the strike price.

Forward SDE. Under the risk-neutral measure $\mathbb{Q}$, the stock price evolves according to:

$$dS_t = rS_t\,dt + \sigma S_t\,dW_t, \quad S_0 = s, \tag{1.15}$$

where W_t is a standard Brownian motion.

BSDE Reformulation. The solution satisfies:

$$u(t, S_t) = \mathbb{E}^{\mathbb{Q}}\left[e^{-r(T-t)}\max(S_T - K, 0) \mid \mathcal{F}_t\right]. \tag{1.16}$$

Applying Itô's lemma to $u(t, S_t)$ gives:

$$\begin{aligned} du(t, S_t) &= \frac{\partial u}{\partial t}(t, S_t)\,dt + \frac{\partial u}{\partial S}(t, S_t)\,dS_t + \frac{1}{2}\frac{\partial^2 u}{\partial S^2}(t, S_t)\,d\langle S\rangle_t \\ &= \left[\frac{\partial u}{\partial t} + rS_t\frac{\partial u}{\partial S} + \frac{1}{2}\sigma^2 S_t^2\frac{\partial^2 u}{\partial S^2}\right]dt + \sigma S_t\frac{\partial u}{\partial S}(t, S_t)\,dW_t. \end{aligned} \tag{1.17}$$

Using PDE (1.13), the drift term simplifies to $ru(t, S_t)$, yielding:

$$du(t, S_t) = ru(t, S_t)\,dt + \sigma S_t\frac{\partial u}{\partial S}(t, S_t)\,dW_t. \tag{1.18}$$

Time Discretization. We discretize the interval $[0, T]$ into N steps:

$$0 = t_0 < t_1 < \cdots < t_N = T, \quad \Delta t = \frac{T}{N}. \tag{1.19}$$

The forward SDE (1.15) is simulated using Euler–Maruyama:

$$S_{n+1} = S_n + rS_n\Delta t + \sigma S_n\Delta W_n, \tag{1.20}$$

where $\Delta W_n \sim \mathcal{N}(0, \Delta t)$. The backward dynamics from (1.18) are discretized as:

$$\begin{aligned} u(t_{n+1}, S_{n+1}) = u(t_n, S_n) &+ ru(t_n, S_n)\Delta t \\ &+ \sigma S_n\frac{\partial u}{\partial S}(t_n, S_n)\Delta W_n. \end{aligned} \tag{1.21}$$

Neural Network Approximation. The Deep BSDE method approximates:

- The initial value $u(0, S_0)$ as a learnable parameter or network $U_\phi(S_0)$.
- At each time step t_n, the gradient $\frac{\partial u}{\partial S}(t_n, S_n)$ by a neural network $G_{\theta_n}(S_n)$.

The recursive approximation becomes:

$$\begin{aligned} Y_0 &= U_\phi(S_0), \\ Y_{n+1} &= Y_n + rY_n\Delta t + \sigma S_n G_{\theta_n}(S_n)\Delta W_n, \end{aligned} \tag{1.22}$$

where Y_n approximates $u(t_n, S_n)$.

Loss Function. At maturity, we enforce the terminal condition (1.14):

$$Y_N \approx \max(S_N - K, 0). \tag{1.23}$$

The training objective minimizes:

$$\mathcal{L}(\Theta) = \mathbb{E}_{\text{MC}}\left[|Y_N - \max(S_N - K, 0)|^2\right], \tag{1.24}$$

where $\Theta = \{\phi, \theta_0, \theta_1, \ldots, \theta_{N-1}\}$ and the expectation is approximated by Monte Carlo sampling over stock price trajectories. The neural network parameters are optimized using Adam optimizer to minimize loss (1.24).

1.2.1. *Results & Benchmark.* We evaluate the performance of the Deep BSDE method by comparing its predicted option prices and Deltas with the analytical Black–Scholes (BS) solutions. The Monte Carlo simulated trajectories are used to compute the predicted values at different time steps.

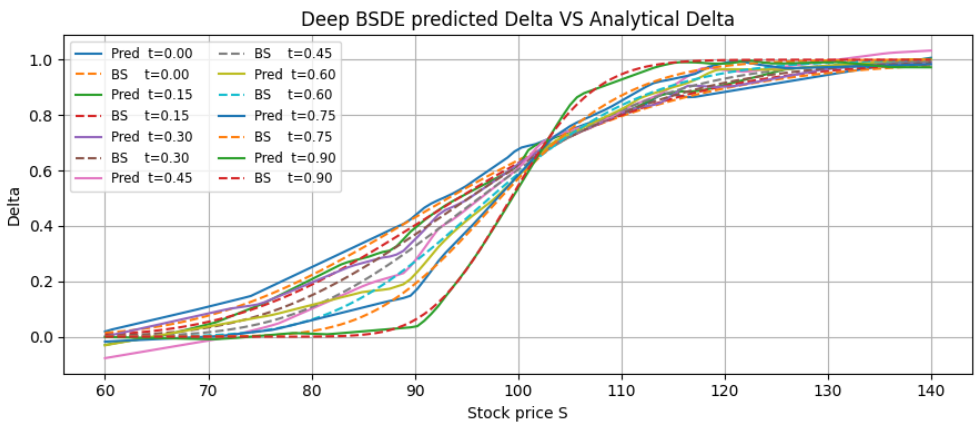


FIGURE 2. Comparison of Deep BSDE predicted Deltas and analytical Black–Scholes Deltas at multiple time steps.

Figure 2 shows that the Deep BSDE model accurately captures the shape and behavior of the Delta function across a wide range of stock prices. The predicted Deltas closely follow the analytical values, with only slight deviations in regions where the option is deep in- or out-of-the-money.

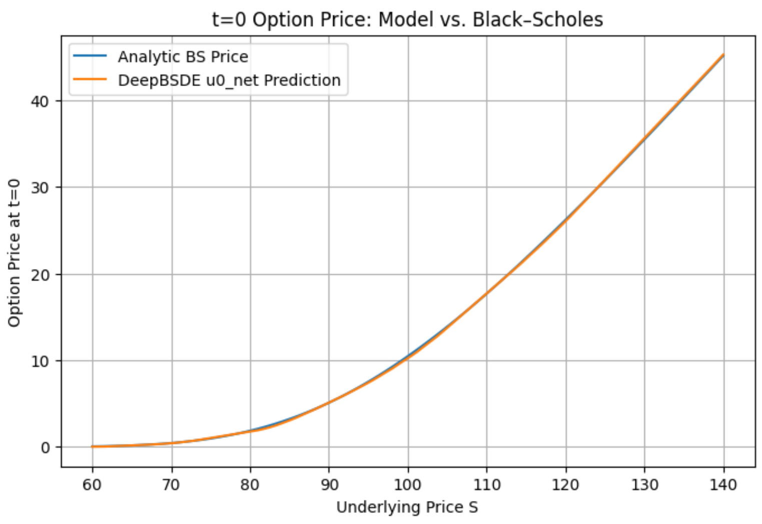


FIGURE 3. Comparison of Deep BSDE predicted option prices at $t = 0$ and the analytical Black–Scholes prices.

Figure 3 demonstrates that the Deep BSDE method produces highly accurate option prices at $t = 0$, matching the analytical Black–Scholes prices almost perfectly.

This suggests that the neural network approximation for $u(0, S)$ generalizes well across the input domain.

1.3. **Deep BSDE for the SABR Model.**

The SABR model under a risk-neutral measure is defined by the coupled SDE system:

$$\begin{aligned} dF_t &= \alpha_t F_t^\beta dW_t, \\ d\alpha_t &= \nu \alpha_t dZ_t, \\ dW_t dZ_t &= \rho dt, \end{aligned} \tag{1.25}$$

with initial values $(F_0, \alpha_0) = (f, \alpha)$. Here F_t represents the forward price, α_t is the stochastic volatility, β controls the backbone dynamics, ν is the volatility of volatility, and ρ is the correlation between price and volatility shocks. For a European call option with payoff $\max(F_T - K, 0)$, we define:

$$u(t, F, \alpha) = \mathbb{E}\left[\max(F_T - K, 0) \mid F_t = F, \alpha_t = \alpha\right]. \tag{1.26}$$

By the nonlinear Feynman-Kac theorem, u satisfies the two-dimensional PDE:

$$\frac{\partial u}{\partial t} + \frac{1}{2}\alpha^2 F^{2\beta} \frac{\partial^2 u}{\partial F^2} + \rho\nu\alpha^2 F^\beta \frac{\partial^2 u}{\partial F \partial \alpha} + \frac{1}{2}\nu^2\alpha^2 \frac{\partial^2 u}{\partial \alpha^2} = 0, \tag{1.27}$$

with terminal condition:

$$u(T, F, \alpha) = \max(F - K, 0). \tag{1.28}$$

BSDE Reformulation. Applying Itô's formula to $u(t, F_t, \alpha_t)$ and using PDE (1.27) to eliminate the drift term, we obtain:

$$\begin{aligned} du(t, F_t, \alpha_t) = \alpha_t F_t^\beta \frac{\partial u}{\partial F}(t, F_t, \alpha_t) dW_t \\ + \nu \alpha_t \frac{\partial u}{\partial \alpha}(t, F_t, \alpha_t) dZ_t. \end{aligned} \tag{1.29}$$

Time Discretization. We partition $[0, T]$ into N steps: $0 = t_0 < \cdots < t_N = T$ with $\Delta t = T/N$. The forward system (1.25) is simulated using:

$$\begin{aligned} F_{n+1} &= F_n + \alpha_n F_n^\beta \Delta W_n, \\ \alpha_{n+1} &= \alpha_n + \nu \alpha_n \Delta Z_n, \end{aligned} \tag{1.30}$$

where $\Delta W_n, \Delta Z_n \sim \mathcal{N}(0, \Delta t)$ with $\mathbb{E}[\Delta W_n \Delta Z_n] = \rho \Delta t$ to preserve the correlation structure. The backward evolution from (1.29) is discretized as:

$$u_{n+1} \approx u_n + \alpha_n F_n^\beta u_F^n \Delta W_n + \nu \alpha_n u_\alpha^n \Delta Z_n, \tag{1.31}$$

where we use the notation:

$$u_n \equiv u(t_n, F_n, \alpha_n), \quad u_F^n \equiv \frac{\partial u}{\partial F}(t_n, F_n, \alpha_n), \quad u_\alpha^n \equiv \frac{\partial u}{\partial \alpha}(t_n, F_n, \alpha_n). \tag{1.32}$$

Neural Network Approximation. The SABR implementation requires approximating two-dimensional gradients:

- The initial value $u(0, F_0, \alpha_0)$ is treated as a learnable parameter u_0.
- At each time step n, both partial derivatives (u_F^n, u_α^n) are approximated by a neural network:
$$(u_F^n, u_\alpha^n) \approx G_{\theta_n}(F_n, \alpha_n), \tag{1.33}$$
where $G_{\theta_n} : \mathbb{R}^2 \to \mathbb{R}^2$ takes the two-dimensional state (F_n, α_n) as input and outputs the gradient vector.

The recursive approximation becomes:

$$\begin{aligned} Y_0 &= u_0, \\ Y_{n+1} &= Y_n + \alpha_n F_n^\beta Z_n^{(1)} \Delta W_n + \nu \alpha_n Z_n^{(2)} \Delta Z_n, \\ (Z_n^{(1)}, Z_n^{(2)}) &= G_{\theta_n}(F_n, \alpha_n), \end{aligned} \tag{1.34}$$

where Y_n approximates $u(t_n, F_n, \alpha_n)$ and $(Z_n^{(1)}, Z_n^{(2)})$ approximate (u_F^n, u_α^n).

Loss Function. At maturity, we enforce the terminal condition (1.28):

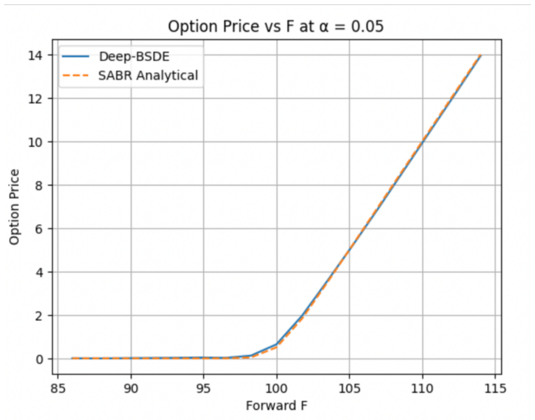
$$Y_N \approx \max(F_N - K, 0). \tag{1.35}$$

The training objective minimizes the Monte Carlo approximated mean-squared error:

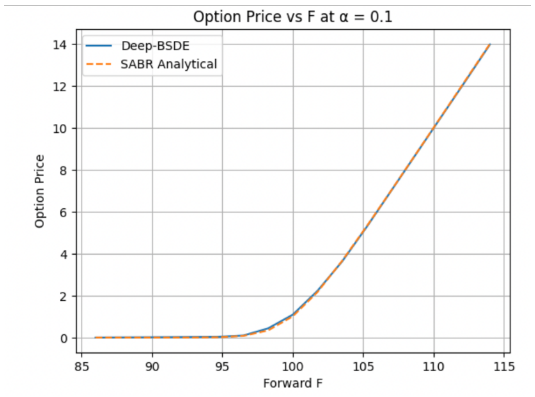
$$\mathcal{L}(\Theta) = \mathbb{E}_{\mathrm{MC}}\left[|Y_N - \max(F_N - K, 0)|^2 \right], \tag{1.36}$$

where $\Theta = \{u_0, \theta_0, \theta_1, \ldots, \theta_{N-1}\}$ includes both the initial value and all gradient network parameters. The parameters are optimized using Adam optimizer to ensure that the neural network learns both the initial value function and the time-dependent gradient structure needed to reproduce the correct terminal payoffs across the two-dimensional state space.

1.3.1. *Result & Benchmark.* We test the Deep BSDE solver under different initial volatilities α_0 and initial forward price F_0. In each case, we compare its predicted option prices against the Hagan's formula.



(A) $\alpha_0 = 0.05$.



(B) $\alpha_0 = 0.10$.

FIGURE 4. Deep BSDE vs. analytical SABR option prices at $t = 0$ for two volatilities.

Figure 4 shows that in both low- and high-volatility regimes ($\alpha_0 = 0.05$ and 0.10), the Deep BSDE method reproduces the SABR-analytical option prices almost exactly across the entire range of forwards F. This confirms that the model has learned both the initial price surface $u(0, F, \alpha)$ and its dependence on α with high accuracy.

1.4. **Experiment Setup.**

1.4.1. *Black–Scholes Model.* For the Black–Scholes experiments, we consider European call options with various initial stock prices to test the method's generalization across different moneyness levels. The forward stock price process follows the standard geometric Brownian motion under the risk-neutral measure.

TABLE 1. Black–Scholes Model Parameters

Parameter	**Value**
Risk-free rate (r)	0.05
Volatility (σ)	0.2
Strike price (K)	100
Maturity (T)	1.0 year
Initial stock price (S_0)	$\mathcal{U}[60, 140]$

The model parameters represent typical market conditions with a 5% risk-free rate and 20% annual volatility. We sample initial stock prices uniformly between 60 and 140 to cover deep out-of-the-money, at-the-money, and deep in-the-money scenarios. This broad range tests the neural network's ability to learn the option pricing function across different regimes.

Table 2. Black–Scholes Neural Network Architecture

Component	**Specification**
Initial value network $u(0, S_0)$	$1 \to 64 \to 64 \to 1$
Gradient networks $\partial_S u(t_n, S_n)$	$\{1 \to 64 \to 64 \to 1\}_{n=0}^{99}$
Activation function	ReLU + BatchNorm
Time discretization	$N = 100$ steps, $\Delta t = 0.01$
Monte Carlo batch size	1024

The neural network architecture uses relatively compact networks with two hidden layers of 64 neurons each. Since the Black–Scholes model is one-dimensional in state space, the networks have simple input-output structures. We employ 100 separate gradient networks, one for each time step, allowing the method to capture the time-varying nature of the option's Delta. The choice of 1024 Monte Carlo paths per batch provides sufficient statistical accuracy while maintaining computational efficiency.

Table 3. Black–Scholes Training Configuration

Parameter	**Value**
Loss function	$\mathbb{E}[\lvert u(T, S_T) - \max(S_T - K, 0)\rvert^2]$
Optimizer	Adam
Learning rate	10^{-3}
Training epochs	1500
Training time	≈ 190 seconds (GPU)

Training uses the Adam optimizer with a learning rate of 10^{-3}, which provides stable convergence without requiring learning rate scheduling. The terminal loss function enforces the boundary condition by minimizing the mean squared error between the predicted option value and the true payoff at maturity. With 1500 training epochs, the network consistently converges to accurate solutions across different random initializations.

1.4.2. *SABR Model.* The SABR model presents a significantly more challenging test case due to its two-dimensional state space and stochastic volatility dynamics. The model captures realistic volatility smile effects observed in financial markets, making it an important benchmark for testing the Deep BSDE method's scalability.

Table 4. SABR Model Parameters

Parameter	**Value**
Beta parameter (β)	0.7
Vol-of-vol (ν)	0.4
Correlation (ρ)	-0.3
Strike price (K)	100
Maturity (T)	1.0 year
Initial forward (F_0)	$\mathcal{U}[85, 115]$
Initial volatility (α_0)	$\mathcal{U}[0.05, 0.15]$

The SABR parameters are chosen to represent realistic market conditions. The beta parameter of 0.7 creates a moderate CEV effect, while the vol-of-vol parameter $\nu = 0.4$ introduces significant stochastic volatility. The negative correlation $\rho = -0.3$ between forward price and volatility movements captures the leverage effect commonly observed in equity markets. We sample both initial forward prices and volatilities from uniform distributions to test the method across various market regimes.

Table 5. SABR Neural Network Architecture

Component	**Specification**
Initial value network $u(0, F_0, \alpha_0)$	$2 \to 64 \to 64 \to 1$
Time-step gradient networks	$\{2 \to 64 \to 64 \to 2\}_{n=0}^{99}$
Activation function	ReLU + BatchNorm
Time discretization	$N = 100$ steps, $\Delta t = 0.01$
Monte Carlo batch size	4096
Correlation implementation	Cholesky decomposition

The SABR implementation requires two-dimensional networks that take both forward price and volatility as inputs. Each gradient network outputs a 2D vector representing the partial derivatives with respect to both state variables. We use a larger batch size of 4096 to ensure sufficient sampling of the two-dimensional state space. The correlated Brownian motions are generated using Cholesky decomposition to maintain the specified correlation structure throughout the simulation.

Table 6. SABR Training Configuration

Parameter	**Value**
Loss function	$\mathbb{E}[\lvert u(T, F_T, \alpha_T) - \max(F_T - K, 0)\rvert^2]$
Optimizer	Adam
Learning rate	5×10^{-4}
Training epochs	1500
Training time	≈ 260 seconds (GPU)

For the SABR model, we use a slightly lower learning rate of 5×10^{-4} to account for the increased complexity of the two-dimensional problem. The training process remains stable across different initializations, typically converging within 1000-1200 epochs. Despite the higher dimensional state space and increased network complexity, the training time remains comparable to the Black–Scholes case due to efficient GPU implementation and optimized network architectures.

1.5. **Summary.** The Deep BSDE method demonstrates remarkable effectiveness in solving high-dimensional parabolic PDEs arising in derivative pricing. Our implementation validates the approach on both the classical Black–Scholes model and the more complex SABR stochastic volatility model. For Black–Scholes, the predicted option prices and Delta values closely match analytical solutions across different moneyness levels, while the SABR results accurately reproduce Hagan's approximation and capture realistic volatility smile effects. The method successfully learns both initial value functions and time-dependent gradient structures through separate neural networks.

Key advantages include natural handling of high-dimensional problems, robust convergence properties, and computational efficiency. The approach scales well with dimension, requiring only modest increases in network complexity when moving from one-dimensional to two-dimensional state spaces, with training times remaining practical (under 300 seconds) even for complex models. Overall, the Deep BSDE method establishes itself as a powerful and practical tool for solving pricing PDEs in mathematical finance, offering both theoretical rigor and computational advantages for high-dimensional derivative pricing problems.

References

1. Han, J., Jentzen, A., & E, W. (2018). Solving high-dimensional partial differential equations using deep learning. *Proceedings of the National Academy of Sciences*, 115(34), 8505–8510. https://doi.org/10.1073/pnas.1718942115
2. Sirignano, J., & Spiliopoulos, K. (2018). DGM: A deep learning algorithm for solving partial differential equations. *Journal of Computational Physics*, 375, 1339–1364. https://doi.org/10.1016/j.jcp.2018.08.029
3. Su, H., Tretyakov, M. V., & Newton, D. P. (2025). Deep learning of transition probability densities for stochastic asset models with applications in option pricing. *Management Science*, 71(4), 2922–2952. https://doi.org/10.1287/mnsc.2022.01448
4. Zhai, J., Dobson, M., & Li, Y. (2021). A deep learning method for solving Fokker-Planck equations. *Proceedings of Machine Learning Research*, 145, 568–597. https://proceedings.mlr.press/v145/zhai21a.html