Pricing Financial Derivatives with Operator Network

Wenyong Zhang, Chak Wong

October 31, 2023

Abstract

Pricing financial derivatives is often done by a computational method under fixed model and contract coefficients. However, understanding the dependence of the derivative price on these coefficients is also important in practice. In this paper, we propose a deep learning method to learn the mapping from the space of time, states, and coefficients to the derivative price. We develop a neural network model in the operator network framework and construct a loss function to train it by solving the backward stochastic differential equation associated with the pricing problem. We test our method in several representative models on both European and path-dependent products. Numerical results demonstrate the ability of our method in learning the complete pricing map.

1 Introductions

Over the past decades, extensive research results have been developed for pricing financial derivatives. While there exist various analytically tractable cases, numerical methods are generally required for pricing, especially for path-dependent and multi-asset derivatives. A plethora of numerical methods are available, which can be classified into several basic approaches: solving the pricing partial differential equation (PDE) by the finite difference method (Duffy (2013)) and the finite element method (Hilber et al. (2013)), Markov chain approximation (Mijatović and Pistorius (2013), Zhang and Li (2019)), Monte Carlo simulation (Glasserman (2004)), and deep learning.

In recent years, the deep learning approach has become increasingly popular for derivative pricing due to its capability in approximating highly nonlinear functions even in high-dimensional problems. Sirignano and Spiliopoulos (2018) develop a deep Galerkin method to solve high-dimensional PDEs arising from option pricing and Gu et al. (2021) introduce a self-paced learning framework to improve convergence. Other papers exploit the connection between nonlinear parabolic PDEs and backward stochastic differential equations (BSDEs) and use deep learning to solve the latter. Han et al. (2017) and Han et al. (2018) propose a forward scheme to solve BSDEs by minimizing the error with the terminal condition. Huré et al. (2020) develop a backward scheme to solve BSDEs by decomposing the global problem into a sequence of smaller learning problems. Beck et al. (2021) introduce a numerical method that combines operator splitting with deep learning. Various papers price path-dependent derivatives by deep learning; see Jacquier and Oumgari (2019), Sabate-Vidales et al. (2020), Feng et al. (2021), and Liang et al. (2021).

With few exceptions, almost all numerical methods for derivative pricing are developed by assuming coefficients in the model (e.g., interest rate, volatility) and contract (e.g., moneyness) are fixed. In other words, if we want to price the derivative by changing the value of a coefficient, we have to run the numerical method again, which can be time consuming. In practice, pricing a derivative under different coefficient values is required. As an example, sensitivity analysis is often used by derivative traders to understand the impact of changing one coefficient on the derivative price. In addition, a trader may want to know the derivative price in different market scenarios as represented by varying coefficient values. Moreover, financial firms need a derivative price simulator to generate data to learn trading and hedging strategies through reinforcement learning. To achieve computational efficiency in all these applications requires that we obtain the complete mapping from the space of time, states, and coefficients to the derivative price, rather than the partial mapping from the space of time and states to the derivative price with fixed coefficients.

There is a lack of research in how to obtain the complete pricing map numerically. This problem can be much more challenging than obtaining the partial pricing map with fixed coefficients, as there is significantly more information to learn. Furthermore, some coefficients can be functions instead of real-valued constants, e.g., the risk-free rate and volatility can be time dependent and hence functions of time. This would further complicate the learning problem.

In a recent work, Berner et al. (2020) solve parametric Kolmogorov PDEs on a whole space-time region by approximating the PDE solution using a multilevel neural network. They express the PDE solution via the Feynman-Kac formula, which enable them to use supervised learning. However, their network only takes constant coefficients as inputs, and they do not consider coefficient functions. Additionally, they only test their approach on European options under the Black-Scholes model. In another work, Glau and Wunderlich (2022) employ a variant of the highway network to approximate the solution of a family of PDEs for pricing European options under the Black-Scholes model. The neural network is designed to produce the option price as a function of time, state, and model coefficients. Similar to Sirignano and Spiliopoulos (2018), they construct the loss function by directly solving the pricing PDE, where the partial derivatives are computed by automatic differentiation. Like in Berner et al. (2020), only constant coefficients are considered and it is unclear how to deal with coefficients that are functions. The success of their method in pricing European options relies on the decomposition of the option price into an no-arbitrage bound, which is known analytically, and a residual part, which is approximated by the neural network. However, for derivatives with more complex payoffs, this decomposition approach may be inapplicable.

In this paper, we take an operator perspective to characterize the complete pricing map. Specifically, any coefficient vector gives a pricing function with time and states as its variables. Thus, we have an operator that maps from the coefficient space, which is generally a function space, to the space of pricing functions. The problem then becomes how to learn this operator. Our method is based on the operator network framework, which is proposed in Lu et al. (2021) based on the universal approximation theorem for nonlinear continuous operators developed in Chen and Chen (1995). An operator network consists of two critical components: the branch net that models the effect of coefficients and the trunk net that models the effect of time and states. The outputs from these two nets are combined via inner product to yield the final output, which is the derivative price in our problem. We introduce another two components into the operator network framework for derivative pricing. The first one is a neural network called embedding net. As some coefficients are functions, we can end up with a high-dimensional representation of them. The embedding net serves the purpose of dimension reduction by obtaining a lower-dimensional latent representation of these coefficient functions, which is input into the branch net. The second component is a neural network called PI net. The prices of many derivatives are invariant after permuting the assets. The PI net is introduced to capture the property of permutation invariance, which can facilitate convergence for multi-asset derivatives as our experiment shows.

To train the proposed deep learning model, we consider the BSDE formulation of the derivative pricing problem. BSDEs provide a powerful and unified tool for pricing all kinds of derivatives under general asset price dynamics. We construct a loss function for training that better fits the purpose of our research, and it is different from the loss functions in the literature.

Compared with Berner et al. (2020) and Glau and Wunderlich (2022), our approach is mainly different in two respects. First, we adopt an operator perspective to characterize the complete pricing map, which leads to a network structure that separates the effects of coefficients and

(t, x). In comparison, a single network is used in Berner et al. (2020) and Glau and Wunderlich (2022) to learn their effects jointly. Second, our adoption of the BSDE formulation is more flexible for dealing with complex path-dependent derivatives, and it can also be extended to incorporate realistic constraints and issues into the pricing problem, such as different borrowing and lending rates and counterparty default risk.

The rest of the report is organized as follows. Section 2 introduces the operator network framework for derivative pricing. Section 3 presents our design of the pricing operator network and discusses how to construct the loss function by solving the pricing BSDE. Section 4 presents numerical results for various derivatives under some standard models. Finally, Section 6 concludes the paper with remarks for future research.

2 Operator Network

Mapping approximation is a crucial problem in machine learning. Typically, we approximate a function that often maps from a vector space to a real value space. However, there are situations where functions are included as part of the input space. This type of mapping, which incorporates additional functions as inputs, holds significant value, particularly in the context of financial engineering problems such as derivative pricing problem.

Derivative pricing involves a stochastic model that describes the dynamics of the underlying assets (such as stocks, rates, indices, etc.). The derivative price is typically viewed as a function (pricing function) of time and state of the underlying with a fixed set of coefficients that are related to the model and product (e.g. yield curve, volatility rate, mean reverting rate for models; strike price, barrier level for products, etc.) . To find the relationship between the derivative price and coefficients, this pricing function needs to be extended to a pricing functional which includes the coefficients as an additional variable (This map is called functional since coefficients includes functions). It is important to consider this extended pricing functional for various tasks in the derivative business.

We formalize this perspective mathematically. Let $u(t, x; a)$ denote a pricing function of time t and state $x \in \mathbb{R}^m$ parametrized by coefficient vector $a = (a_1, \cdots, a_p)$ where each entry of this vector can either be real value (e.g. parameters of the dynamics of underlying assets or product such as mean reverting rate, volatility rate or strike price) or real function (e.g. yield curve or volatility curve). We assume $a \in \mathcal{A}$, which is a space containing functions and real values related to parameters of the pricing function. Therefore, $\mathcal{A}$ is a function space since the components of a can be functions (e.g., the risk-free rate is often described by a curve, which is a function of time). Then, if we introduce a into the input space of the pricing function, we have the extended pricing functional which is a mapping from $\mathbb{R}^+ \times \mathbb{R}^m \times \mathcal{A}$ to $\mathbb{R}$. From another perspective, we can characterize this map by introducing an operator that maps coefficients vector a to pricing functions u. Suppose that given a, we have a $u(t, x; a) \in \mathcal{U}$, which is a Banach space containing all pricing functions. We consider an operator $\mathcal{G} : \mathcal{A} \to \mathcal{U}$, which is defined by

$$(\mathcal{G}a)(t, x) := u(t, x; a).$$

Thus, the problem of pricing the derivative under different coefficients becomes learning the operator $\mathcal{G}$. Since the component of a consists of functions that are infinite dimensional, we would like to consider how to represent them. Let $\tilde{a} = (\tilde{a}_1, \cdots, \tilde{a}_p)$ be a suitable representation of a (We suppose all entries of a are functions without loss of generality.), which we refer to as function embedding. There are generally two ideas to represent coefficient functions.

- (**Basic DeepONet**) The first one is proposed in Lu et al. (2021) which is using the value of the function $[a(x_1), ..., a(x_m)]$ on a specified grid $[x_1, ...x_m]$. However, if variables of this function have a high dimension, this approach would suffer from the curse of dimensionality.

- (**Add Kernel Operator**) The other idea is expanding the function using a set of basis functions and representing it by the vector of expansion coefficients. For better seeking the expansion coefficients, the more general method is to forward the function into a kernel operator and use some of its outputs as a representation of the input function. The kernel function is denoted as $\psi(x, y) \in C(\mathbb{R}^d \times \mathbb{R}^{\tilde{d}}, \mathbb{R})$ where $\tilde{d} \leq d$ typically. Then for the input function $a(x)$, the kernel operator is defined as:

  $$\tilde{a}(y) = \int_D \psi(x, y) a(x) dx$$

  Where D is a subset of $\mathbb{R}^{\tilde{d}}$ and we then choose $[y_1, y_2, ..., y_k]$ from the frequency domain and plug into $u(y)$. Then $\tilde{a} = [\tilde{a}(y_1), \tilde{a}(y_2), ..., \tilde{a}(y_k)]$ is the abstract vector which can better represent the function $a(x)$ in a lower dimension. For choosing of $\psi(x, y)$. We can propose at least two methods:

  1. We can use Fourier transform and output some Fourier modes as the latent representation with the Fourier kernel $\psi(x, y) = e^{-i\pi x^\top y}$. This approach is equivalent to projecting the functions onto a set of Fourier basis functions, and selecting only a few dominant Fourier modes can yield a low-dimensional representation of the functions that captures their essential features. This approach can be computationally more efficient than the grid approach, especially when the input functions have periodic or quasi-periodic behavior. However, selecting appropriate Fourier modes can be challenging and may require some trial and error or domain knowledge.
  2. We can use convolution layers to map the original functions to another domain, resulting in a low-dimensional latent representation. We can then concatenate this latent vector with the constant coefficients and input the resulting vector into an FNN to obtain the output of the branch net. This approach can be computationally more efficient than inputting discretized function values directly into the FNN when the number of grid points is large. In this approach, the kernel function is a trainable network or layers (convolution kernel with trainable parameters) which can be represented as: $\psi(x, y) = \psi(x - y; \theta_{\text{CNN}})$. Then θ_{CNN} can be involved into trainable variables of the deep operator network.

We introduce the operator network framework proposed by Lu et al. (2021). Motivated by the universal approximation theorem for continuous nonlinear operators developed in Chen and Chen (1995) (This will be shown in appendix part), Lu et al. (2021) approximate $(\mathcal{G}a)(t, x)$ as

$$(\mathcal{G}a)(t, x) \approx G_\theta(\tilde{a}, t, x) = \sum_{n=1}^{q} B_n(\tilde{a}) \cdot K_n(t, x),$$

where $B_n(\tilde{a})$ and $K_n(t, x)$ are neural networks, and θ is the vector of parameters in the operator network. This approximation separates the effects of (t, x) and a on the derivative price, so we can build separate models for them.

In practice, building $2q$ separate neural networks for learning leads to a sophisticated model, which can be difficult to train and is prone to overfitting. We consider the so-called DeepONet, which is proposed in Lu et al. (2021) and enhanced by Tan and Chen (2022). Figure 1 illustrates its architecture, which consists of two sub-networks named as *branch net* $B(\tilde{a})$ (whose output is in $\mathbb{R}^q$) and *trunk net* $K(t, x)$ (whose output is in $\mathbb{R}^q$). The branch net outputs the expansion coefficients for given $\tilde{a}$ while the trunk net outputs values of the basis functions for given (t, x). We then take the inner product of these two output vectors to obtain $G_\theta(\tilde{a}, t, x)$. Denote $< \cdot, \cdot >$ as the innerproduct of two vectors, then we approximate $(\mathcal{G}a)(t, x)$ as:

$$(\mathcal{G}a)(t, x) \approx G_\theta(\tilde{a}, t, x) =< B(\tilde{a}), K(t, x) >,$$

For the derivative pricing problem, we further add two components to this framework and will discuss the details of our design in the next section.

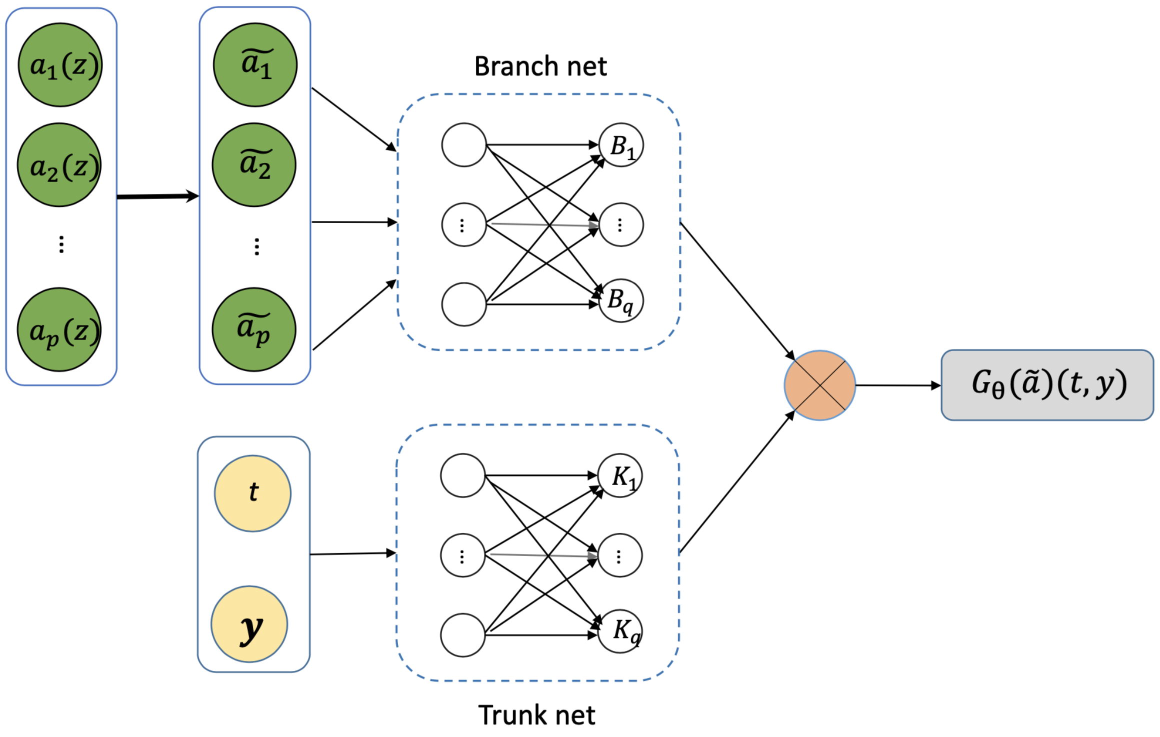


Figure 1: Architecture of DeepONet: This figure illustrate the basic structure of the operator network which is composed by branch net and trunk net. For branch net, we do function representation of $a = (a_1, ..., a_p)$ by methods provided before (sample on grid or kernel operator) and the send the representation $(\tilde{a}_1, \cdots, \tilde{a}_p)$ into the branch network and the output a vector of $\mathbb{R}^q$; For trunk net, we can do some processing on t or y (where y in the picture is exactly x.) and then input them into a neural network and output a vector of $\mathbb{R}^q$. Finally, we do the innerproduct of outputs from both networks and then obtain the price value given $(t, x; a)$.

3 Training of Pricing Operator Network

We first formulate the derivative pricing problem using BSDEs [1] , which we will solve by deep learning to obtain the pricing function. We then present our detailed design of the trunk nets in the DeepONet, and develop a loss function for training.

3.1 BSDE for Derivative Pricing

Without loss of generality, we discuss derivative pricing under the time-dependent geometric Brownian motion (GBM) model [2] [3]. Our method is applicable to more sophisticated models such as stochastic volatility (SV) and local volatility models. An example considering the

[1]In our problem, we deal with FBSDE, which is Forward-Backward Stochastic Differential Equation. In FBSDEs, we consider two stochastic processes evolving simultaneously: One forward in time and the other backward in time. The forward process (SDE) represents the state of underlying assets and related Markovian variables, while the backward process (BSDE) represents a dual process associated with the pricing function.

[2]The most general case is the SDE: $\mathrm{d}S_t = \mu(t, S_t)\mathrm{d}t + \sigma(t, S_t)\mathrm{d}W_t$. Where $a = (\mu, \sigma, \rho, \kappa)$

[3]The most general BSDE is $-\mathrm{d}Y_t = -f(t, S_t, Y_t, Z_t)\mathrm{d}t + Z_t dW_t$. Where $f(t, s, y, z)$ is a driver function depends on the product and SDE itself, which is illustrated by Ito formula.

Heston's SV model will be presented in experiment section and the example of stochastic local volatilty model (SLV) will be discussed in the extension section.

On a filtered probability space $(\Omega, \mathcal{F}, \mathbb{P}, \{\mathcal{F}_t, t \geq 0\})$ satisfying the usual hypothesis, consider d underlying assets $S_t = (S_t^1, S_t^2, ..., S_t^d)$ with the following dynamics under the risk-neutral measure:

$$\begin{aligned} \mathrm{d}S_t^i &= r(t)S_t^i \mathrm{d}t + \sigma^i(t) S_t^i \mathrm{d}W_t^i, \quad i = 1, \cdots, d, \\ dW_t^i dW_t^j &= \rho_{ij} \mathrm{d}t, \end{aligned} \tag{1}$$

where $W_t = (W_t^1, \cdots, W_t^d)$ is a correlated d-dimensional Brownian motion with respect to $\{\mathcal{F}_t, t \geq 0\}$, $r(t)$ is the forward rate function, and $\sigma^i(t)$ is the volatility function of asset i. We set $\sigma(t) = (\sigma^1(t), \cdots, \sigma^d(t))$, and $\rho = (\rho_{ij})$. We assume that $r(t)$ and every $\sigma^i(t)$ are deterministic functions of time.

The derivative under consideration matures at time T with payoff G_T, which can be path dependent. Let X_t be the vector of all related states at time t to determine the derivative price, which may contain more information than the asset prices at t. By choosing X_t appropriately, we can write $G_T = g(X_T; a)$ and each product is corresponded to a $g(X_T; a)$. The risk-neutral pricing function is given by

$$u(t, x; a) = \mathbb{E}\left[e^{-\int_t^T r(s)\mathrm{d}s} g(X_T; a) | \, X_t = x \right].$$

We present three examples of derivative products below. We denote moneyness of an option by κ, which is defined as the strike divided by the initial asset price (it becomes the average of the initial prices of all assets in the multi-asset case). Let $m_t^i = \min_{u \in [0,t]} S_u^i$, which is the running minimum price of asset i, and set $m_t = (m_t^1, \cdots, m_t^d)$. We also consider $A_t^i = \exp\left\{ \frac{1}{t} \log\left(\int_0^t S_s^i \mathrm{d}s \right) \right\}$, the geometric average price of asset i, and set $A_t = (A_t^1, \cdots, A_t^d)$.

- European geometric basket call option: $X_t = S_t$, $a = (r(t), \sigma(t), \rho, \kappa)$, and

$$g(X_T; a) = \left(\left(\prod_{i=1}^d S_T^i \right)^{1/d} - \left(\prod_{i=1}^d S_0^i \right)^{1/d} \kappa \right)^+.$$

- Floating-strike lookback call option: $X_t = (S_t, m_t)$, $a = (r(t), \sigma(t), \rho)$, and

$$g(X_T; a) = \frac{1}{d} \sum_{i=1}^d (S_T^i - m_T^j).$$

- Geometric Asian call option: $X_t = (S_t, A_t)$, $a = (r(t), \sigma(t), \rho, \kappa)$, and

$$g(X_T; a) = \left(\frac{1}{d} \sum_{i=1}^d A_T^i - \frac{\kappa}{d} \sum_{i=1}^d S_0^i \right)^+.$$

It is well known that there is a deep connection between derivative prices and solutions to BSDEs. Consider the following BSDE:

$$\begin{aligned} -\mathrm{d}Y_t &= -r(t) Y_t \, \mathrm{d}t - Z_t{}^\top \, \mathrm{d}W_t, \\ Y_T &= g(X_T; a), \end{aligned} \tag{2}$$

where W_t is the Brownian motion in (1). A solution to this BSDE is a pair of processes (Y_t, Z_t) adapted to $\{\mathcal{F}_t, \ t \geq 0\}$. In the solution of the BSDE, $Y_t = u(t, X_t; a)$, which is the derivatives price at time t.

It is important to bear in mind that X_t, W_t, Y_t, and Z_t all depend on the coefficient vector a in our problem. However, to ease notations, we do not explicitly reflect their dependence on a in their notations.

Remark 1. For path-dependent products, as the previous examples illustrate, additional information must be included in the state vector to determine their prices. While in the examples given, such information can be easily found, more generally one can consider using some representation of the path as a state. See Sabate-Vidales et al. (2020), Jacquier and Oumgari (2019), Feng et al. (2021), and Bayraktar et al. (2022) for using path signature and Saporito and Zhang (2020) for using LSTM to solve the pricing problem in various settings.

3.2 Trunk Network Design-Adding Permutation Invariant Layers

We discuss further design of the trunk net. For single asset option pricing, it is sufficient to input the time and state variables into an FNN, as shown in Lu et al. (2021). However, for multi-dimensional tasks, inputting high-dimensional states directly into the FNN leads to a significant increase in the number of parameters to train, which can cause various convergence issues. Fortunately, payoffs of multi-asset options often have a symmetric structure, meaning that the asset price variables of their pricing functions are exchangeable. Below are three common European-style multi-dimensional payoff functions (Where I is the strike price which is predetermined):

$$g(X_T;a)=\begin{cases}\left(\max\left(S_T^1,\ldots,S_T^d\right)-I\right)_+, & \text{(basket call on max)},\\ \left(\frac{1}{d}\sum_{i=1}^d S_T^i-I\right)_+, & \text{(basket call on sum)},\\ \sum_{i=1}^d 1_{\{S_T^i\geq I\}}, & \text{(basket sum of binary options)}.\end{cases}$$

In the above payoff functions, if we permute the underlying assets, the option's payoff and price remain unchanged. We call this property permutation invariant (PI).

Germain et al. (2022) proposed a symmetric neural network to parameterize PI functions, which we employ in our problem. We call it as *PI net* and explain its structure. Let x_i denote the state variable vector of asset i with length n_x, and we set $x=(x_1,...,x_d)$. For each x_i, we input it into an FNN φ, which is shared among all the assets. This network maps $x_i\in\mathbb{R}^{n_x}$ to $\varphi(x_i)\in\mathbb{R}^k$. Set $\varphi(x)=(\varphi(x_1),\cdots,\varphi(x_d))$, which is a latent representation of x. We output from the PI net

$$\Phi(x)=\mathfrak{s}\left(\varphi\left(x\right)\right),$$

where $\mathfrak{s}$ is a symmetric function in its variables. Below are some examples of symmetric functions and all of them operate component-wise.

- Max-pooling: $\mathfrak{s}(x)=\max\left(x_1,...,x_d\right)$.
- Sum: $\mathfrak{s}(x)=\sum_{i=1}^d x_i$.
- Average: $\mathfrak{s}(x)=\frac{1}{d}\sum_{i=1}^d x_i$.

The choice of the symmetric function is clearly determined by the product's payoff. For example, we use the max-pooling function for the basket call on max, and the sum function for the basket call on sum. The architecture of the PI net is illustrated in Figure 2. After the propagation of the PI net, we combine its output $\Phi(x)$ with the time variable t and feed them into the trunk net, whose architecture is shown in Figure 3. As the output of the PI net is invariant by permutation, the output of the trunk net also has this property. This allows the operator network to capture the PI property of the derivative price.

We call the operator network designed for our problem pricing operator network (PONet).

3.3 Loss Function

In the literature, the BSDE (2) is solved for fixed a . We discuss the loss function under SDE with and without jump diffusion (We discuss loss function without jump in this section, things

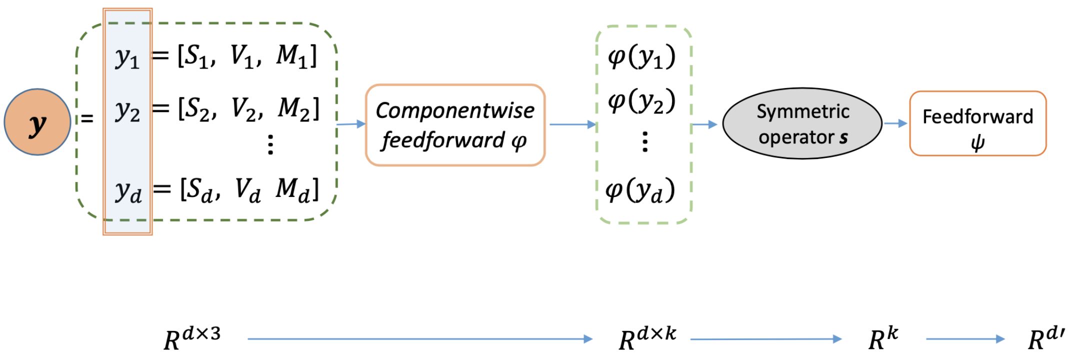


Figure 2: Architecture of the PI net: First we seperate the state of each assets and the forward the state of each asset into a neural network $\varphi(\cdot)$ seprately and the we get a matrix in $\mathbb{R}^{dxk}$. Then we pass this matrix into the symmetric operator (e.g. we calculate the average on the dimension of d). Finally we forward the output into a ordinary FNN and we have the output which is invariant of the permutation of underlying assets.

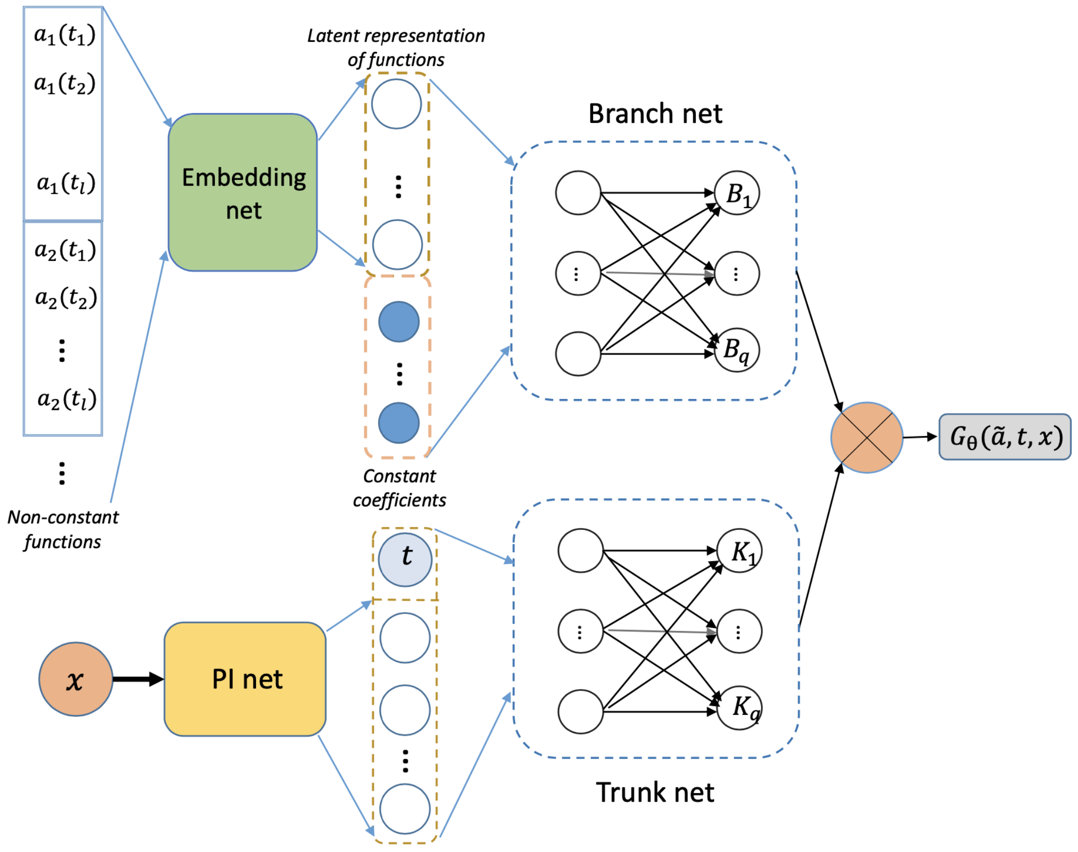


Figure 3: Architecture of the pricing operator network (PONet): This is the detailed forward propagation of out network. For branch network: we first forward the functions into a embedding net which plays a role on function representation and it can be parametrized and unparameterized based on the kernel operator we have chosen then we concatencate the representation and real valued coefficients to a unified tensor and send it into the branch network; For trunk network: we first forward state variables into the permutation invariant network and concatencate the output with the time variable and finally send them into the trunk network.

with jump is leaved in the extension section.). For simplicity, for non-jump case, we use time-dependent GBM as the example to discuss the loss. We also need to obtain the dependence of the BSDE's solution on a. The entire mapping from (a, t, x) to the derivative price is approximated by the PONet $G_\theta(\tilde{a}, t, x)$. To learn the optimal θ, we need a loss function for training.

Given a, we construct a loss function $\mathcal{L}(\theta; a)$ based on the discretized BSDE. Then we take expectation over a distribution μ from which we sample the coefficient vector a. This yields the following loss function for the PONet:

$$\mathcal{J}(\theta) = \mathbb{E}_{a\sim\mu}\left[\mathcal{L}(\theta; a)\right].$$

3.3.1 Loss Function under Diffusion Processes

In this report, we discuss the construction of the loss function in the context of diffusion processes (non-jump processes). To simplify the explanation, we utilize a time-dependent Geometric Brownian Motion (GBM) process as an example (Other underlying assets follow the same routine of loss function construction.). This choice allows us to better illustrate the formulation of the loss function. However, it's important to note that for processes involving jump diffusion, the corresponding Backward Stochastic Differential Equation (BSDE) and loss function differ slightly but pose a greater challenge. Therefore, we will address pricing under jump processes in the extension section. We next explain how to construct $\mathcal{L}(\theta; a)$ for given a under GBM model. Recall the BSDE in intergal form:

$$Y_t = g(X_T; a) - \int_t^T r(s) Y_s \, \mathrm{d}s - \int_t^\top {Z_s}^\top \, \mathrm{d}W_s, \quad t \in [t_0, T].$$

We consider a time grid: $0 = t_0 < t_1 < ... < t_N = T$ which is indexed by $\{0, 1, \cdots, N\}$. For simplicity, we assume the time points are equally spaced with time step Δt. Let $\Delta W_{t_n} := W_{t_{n+1}} - W_{t_n}$. We discretize the SDE (1) using the Euler scheme as

$$S^i_{t_{n+1}} \approx S^i_{t_n} + r(t_n) S^i_{t_n} \Delta t + \sigma^i(t_n) S^i_{t_n} \Delta W_{t_n}, \quad i = 1, \cdots, d,$$

For the BSDE, we discretize it as

$$\begin{aligned} Y_{t_N} &= g(X_{t_N}; a), \\ Y_{t_n} &\approx Y_{t_{n+1}} - r(t_{n+1}) Y_{t_{n+1}} \Delta t - Z_{t_n}^\top \Delta W_{t_n}, \quad n = N-1, \cdots, 0. \end{aligned}$$

The output from the PONet, $G_\theta(\tilde{a}, t_n, x)$ approximates Y_{t_n} given $X_{t_n} = x$. Then we construct the loss function. The loss function is seperated into two parts. The one is interior loss, which is to minimize the square error of the residual part of the BSDE 2 (The first line of the equation); the other is the terminal loss, which is to minize the distance between the value yield from the network and the exact terminal payoff (The second line of the equation). Then we sum them up to form the loss given a. We give the formulation of these two loss functions:

- The interior loss which is to optimize the interior condition is:

$$\mathcal{L}_{\text{interior}}(\theta; a) = \frac{1}{M} \sum_{m=1}^{M} \sum_{n=0}^{N} \left(g(X^m_{t_N}; a) - G_\theta(\tilde{a}, t_n, X^m_{t_n}) - \sum_{k=n}^{N-1} r(t) G_\theta(\tilde{a}, t_k, X^m_{t_k}) \Delta t + (Z^m_{t_k})^\top \Delta W^m_{t_k} \right)^2$$

  which means that we need to enforce the approximated value match all the ground truth value at each time step stemed from the terminal payoff through the BSDE 2. In this function, $X^{i,m}_{t_n}$ denotes the state vector at time t_n of asset i on path m (In GBM case, $X^m_{t_n} = S^m_{t_n}$; in stochastic volatility case, $X^m_{t_n} = (S^m_{t_n}, V^m_{t_n})$; In Asian option case, $X^m_{t_n} = (S^m_{t_n}, A^m_{t_n})$ etc.) and $\Delta W^m_{t_k}$ denotes the Brownian increment in the time interval $[t_k, t_{k+1}]$ of the m-th path. The control term $Z^m_{t_n}$ for $n = 0, ..., N-1$ and $m = 1, ..., M$ is given as:

$$Z^m_{t_n} = \sigma(t) \circ S_{t_n} \circ \nabla_X G_\theta(a, t_n, X^m_{t_n}),$$

- The terminal loss function is given as:

$$\mathcal{L}_{\text{terminal}}(\theta; a) = \frac{1}{M}\sum_{m=1}^{M}\left(G_\theta(\tilde{a}, t_N, X_{t_N}^m) - g(X_{t_N}^m; a)\right)^2.$$

  This loss enforce the value given at the terminal date match the exact terminal payoff $g(X_{t_n}^m, a)$.

Then the loss function $\mathcal{L}(\theta; a)$ is given by:

$$\mathcal{L}(\theta; a) = \mathcal{L}_{\text{interior}}(\theta; a) + \mathcal{L}_{\text{terminal}}(\theta; a)$$

To obtain $\mathcal{J}(\theta)$, we need to sample a from distribution μ. Suppose that we have a batch of sampled coefficients $\{a_1, \cdots, a_B\}$. We calculate $\mathcal{J}(\theta)$ by

$$\mathcal{J}(\theta) = \frac{1}{B}\sum_{j=1}^{B}\mathcal{L}(\theta; a_j).$$

To learn θ, we minimize $\mathcal{J}(\theta)$ by stochastic gradient descent. In each iteration, a batch of B coefficients are sampled and for each sampled coefficient vector a_j, M price paths are generated to calculate $\mathcal{L}(\theta; a_j)$. A summary of the training algorithm is presented in 3.3.2.
Testing Design: One method to test the loss function is to plug the analytical solution into the loss function to check whether the loss function value is close to zero. For BSDE under GBM, we use the Black-Scholes formula as the operator network and calculate the loss function on this model.

3.3.2 Algorithm of European Style Products

We summarize the training framework of European style problem on a certain time interval $[T_1, T_2]$. In this framework, the initial time t_0 is corresponded to T_1 and terminal time T is corresponded to T_2. The algorithm is states as 1:

Algorithm 1 Training of the Pricing Operator Network on $[T_1, T_2]$

```
Specify the sampling distribution $\mu$ and initialize $\theta$.
for $i = 1, 2, ...$ do
    Sample a batch of coefficients $\{a_j,\ j = 1, \cdots, B\}$ from $\mu$.
    For each $j$, generate $M$ asset price paths on $[T_1, T_2]$ under $a_j$ and calculate $\mathcal{L}(\theta; a_j)$. .
    Calculate $\mathcal{J}(\theta)$.
    Update $\theta$ by gradient descent.
end for
return The well trained operator network $u_\theta(t, x; a)$ defined on $[T_1, T_2]$.
```

For a specific product, we need to clarify several issues:

- The coefficients a from underlying SDE (e.g. rate curve, volatility curve, correlation coefficients, etc.) and product (e.g. barrier level, strike price, etc.);

- The terminal payoff $g(X_{\tau_N}; a)$ which can be path dependent;

- The Markovian variables for path-dependent product.

3.4 Framework with Early Exercise Features

The most common type of early exercise derivatives is Bermudan options, one can exercise the product at a series of early exercise times. Denote $\mathcal{T} := \{\tau_0 = 0, \tau_1, \ldots, \tau_k, \cdots, \tau_N = T\}$. be a collection of pre-determined time stamps on $[0, T]$. For Bermudan type options, the buyer has the right to exercise the option and obtain payoff $g(\tau, X_\tau; a), \tau \in \mathcal{T}$. Consequently, the fair price at time t can be written as:

$$u(t, x; a) = \sup_{\tau \in \mathcal{T}, \tau \geq t} \mathbb{E}\left[e^{-\int_\tau^T r(s) ds} g(\tau, X_\tau; a) \mid X_t = x\right]$$

Fortunately, from the method in Gao et al. (2022) and Wang et al. (2018), we have the formulation of the BSDE of Bermudan derivatives. We express the BSDE with integral form::

$$\begin{cases} Y_T = g(T, X_T; a) \\ Y_t = Y_{\tau_{k+1}} - \int_t^{\tau_{k+1}} r(s) Y_s \mathrm{d}s - \int_t^{\tau_{k+1}} Z_s{}^T \, \mathrm{d}W_s, \quad t \in (\tau_k, \tau_{k+1}), k = N-1, \ldots, 0, \\ Y_{\tau_k} = \max\left(g(\tau_k, X_{\tau_k}; a), Y_{\tau_{k+1}} - \int_{\tau_k}^{\tau_{k+1}} r(s) Y_s^a \mathrm{d}s - \int_{\tau_k}^{\tau_{k+1}} Z_s{}^T \, \mathrm{d}W_s\right). \end{cases}$$

Then this kind of problem can be decomposed into several sub-problem while each problem is corresponded to an European pricing problem.. Then the problem can be solved by a recursive procedure:

- When $t \in (\tau_{N-1}, \tau_N]$, solve the BSDE problem:

  $$\begin{aligned} Y_{\tau_N} &= g(\tau_N, X_{\tau_N}; a) \\ Y_t = Y_{\tau_N} &- \int_t^{\tau_N} r(s) Y_s \mathrm{d}s - \int_t^{\tau_N} Z_s{}^T \, \mathrm{d}W_s, \quad t \in (\tau_{N-1}, \tau_N] \end{aligned}$$

  Then we have the solution $u(t, X_t; a) \approx Y_t$ on the time interval $(\tau_{N-1}, \tau_N]$. We denote this sub solution as $u^N(t, x; a)$.

- When $t \in (\tau_{k-1}, \tau_k]$, where $k = 1, ..., N-1$. Solve the BSDE problem:

  $$\begin{aligned} Y_{\tau_k} &= \max\left\{g(\tau_k, X_{\tau_k}; a), u^{k+1}(\tau_k, X_{\tau_k}; a)\right\} \\ Y_t = Y_{\tau_k} &- \int_t^{\tau_k} r(s) Y_s \mathrm{d}s - \int_t^{\tau_k} Z_s{}^T \, \mathrm{d}W_s, \quad t \in (\tau_{k-1}, \tau_k] \end{aligned}$$

  Then we have the solution $u(t, X_t; a) \approx Y_t$ on the time interval $(\tau_{k-1}, \tau_k]$. We denote this sub solution as $u^k(t, x; a)$. .

- We finally interpolate the solution as a unified solution as:

  $$u(t, x; a) = \sum_{k=1}^N u^k(t, x; a) \mathbb{I}_{\{\tau_{k-1} \leq t \leq \tau_k\}}$$

3.4.1 General Algorithm of Early Exercise Products

We translate the above recursic=ve procedure into operator network training, see 2.

For a specific product, we need to clarify several issues:

- The coefficients a from underlying SDE and product (e.g. barrier level, strike price, etc.);
- The terminal payoff $g(\tau_N, X_{\tau_N}; a)$ which can be path dependent;
- The early exercise payoff $g(\tau_k, X_{\tau_k}; a)$ which can be path dependent;

Algorithm 2 Training of the Operator Network with Early Exercise Feature

```
1: (Sampling) Sample $B$ input functions $\{a_j\}_{j=1}^B$ from a distribution. Then sample $M$ paths for each $a$ on the time horizon $[\tau_0, \tau_N]$.
2: (Slicing) Slice the data corresponded to $[\tau_{N-1}, \tau_N]$ and train the operator network $u_{\theta^N}(t, x; a)$ with terminal condition $g(\tau_N, X_{\tau_N}; a)$ by algorithm 3.3.2 with $T_1 = \tau_{N-1}$ and $T_2 = \tau_N$.
3: (Training European Operators Recursively) For $k$ in $1, \cdots, N-1$. Slice the data corresponded to $[\tau_{k-1}, \tau_k]$ and train the operator network $u_{\theta^k}(t, x; a)$ with terminal condition $\max\{g(\tau_k, X_{\tau_k}; a), u_{\theta^{k+1}}(\tau_k, X_{\tau_k}; a)\}$ by algorithm 3.3.2 with $T_1 = \tau_{k-1}$ and $T_2 = \tau_k$.
4: Finally obtain the operator network by:
       $$u_\theta(t, x; a) = \sum_{k=1}^{N} u_{\theta^k}(t, x; a)\mathbb{I}_{\{\tau_{k-1} \le t \le \tau_k\}}$$
   Where $\theta$ is the parameter for the unified network which is composed by $\{\theta^1, \cdots, \theta^N\}$ and each $\theta^k$ is corresponded for each time interval between two consecutive early exercise dates $\tau_{k-1}$ and $\tau_k$.
```

- The Markovian variables for path-dependent product.

The most representative products are Bermudan swaption and callable yield note (CYN), which will be introduced and things to be clarified will also be presented in following subsubsections. For discussion of each specific product, we will discuss with the following procedure:

1. The stochastic differential equations (SDE) of the underlyings.
2. The payoff structure of the product of interest.
3. The recursive procedure of the product and clarified issues of the problem.
4. several unit tests.

Test Design For Bermudan Algorithm: Since the Bermudan algorithm is a combination of European algorithm on different time intervals. We can set the early exercise payoff be zero. Then we check whether the result is consistent to the corresponded European product on the entire time interval. For example. We can do this test with analytical solution: Under GBM, we just price the Bermudan put with only one early exercise date and then at the exercise date τ. The terminal value is $\max\{K - S, BS(S, K, \tau, T, r, \sigma)\}$. And we can check two things:

- Whether the Bermudan with exxercise payoff being zero is close to European product.
- Whether the product under GBM with one exercise date is close to the European with terminal payoff as $\max\{K - S, BS(S, K, \tau, T, r, \sigma)\}$.

3.4.2 A Typical Case: Bermudan Swaption under Hull-White Model

Hull-White Model Hull and White explored extensions of the Vasicek model that provide an exact fit to the initial term structure. We denote W_t as a Brownian motion under risk neutral measure $\mathbb{Q}$. One version of the extended Vasicek model that they consider is:

$$dR_t = (\theta(t) - aR_t)dt + \sigma dW_t$$

where a and σ are constants. σ is the instantaneous standard devation of the short rate. $\theta(t)$ is a function of time chosen to ensure that the model fits the initial term structure. At time t, the short rate reverts to $\theta(t)/a$ at rate a. We always interested in the zero coupon bond value which is defined as:

$$P(t,T) = \mathbb{E}\left[e^{-\int_t^T R_s \mathrm{d}s} \middle| R_t \right].$$

Given the forward rate curve as $F(t,T)$ which is defined as $F(t,T) = -\partial \ln P(t,T)/\partial T$. The mean reversion function $\theta(t)$ can be shown as:

$$\theta(t) = F_t(0,t) + aF(0,t) + \frac{\sigma^2}{2a}\left(1 - e^{-2at}\right)$$

The last term in this equation is usually fairly small. If we ignore it, the equation implies that the drift of the process for R at time t is $F_t(0,t) + a(F(0,t) - R_t)$. This shows that, on average, R follows the slope of the initial instantaneous forward rate curve. When it deviates from that curve, it reverts back to it at rate a. Under Hull-White model, the price of zero coupon bond is given by an analytical affine model:

$$P(t,T) = A(t,T)e^{-B(t,T)R_t}$$

$P(t,T)$ is a function of (t,T,R_t) where

$$B(t,T) = \frac{1 - e^{-a(T-t)}}{a}$$

and

$$\ln A(t,T) = \ln \frac{P(0,T)}{P(0,t)} + B(t,T)F(0,t) - \frac{1}{4a^3}\sigma^2\left(e^{-aT} - e^{-at}\right)^2\left(e^{2at} - 1\right)$$

Interest Rate Swap Before we introduce the payoff structure of interest rate swap, we define forward-looking reference rate whose term is $[t_s, t_e]$ and which is observed ar t where $t \le t_s \le t_e$:

$$L_t(t_s,t_e) = \frac{1}{t_e - t_s}\left(\frac{P(t,t_s)}{P(t,t_e)} - 1\right).$$

According to Yamakami and Takeuchi (2022). We can express the value of swap as a portfolio of discounted bonds. For example, let T_0 be the beginning of the swap period, and $T_1, \cdots, T_M$ is the interest payment date of the swap. We express the time t value of the swap starting at the nearest time stamp after t as $g(t, R_t; a)$ and The valuation of the swap which receives a fixed interest rate K with the same frequency on the fixed and floating legs is:

$$\begin{aligned} g(t,R_t;a) &= \sum_{i=1}^{M} K\left(T_i - T_{i-1}\right)P\left(t,T_i\right) - \sum_{i=1}^{M} L_t\left(T_{i-1},T_i\right)\left(T_i - T_{i-1}\right)P\left(t,T_i\right) \\ &= \sum_{i=1}^{M-1} K\left(T_i - T_{i-1}\right)P\left(t,T_i\right) \\ &\quad + \left\{1 + K\left(T_i - T_{i-1}\right)\right\}P\left(t,T_M\right) - P\left(t,T_0\right). \end{aligned}$$

Therefore, in Hull-White model, we have $a = (\theta(t), a, \sigma, K)$. We can also find that $P(t,T)$ is also related with a.

Bermudan Swaption A swaption is an option on a swap. A Bermudan swaption allows multiple times at which the holder can exercise the option and enter into the underlying swap. With T_M the maturity and T_0 expiry date of the option, we let T_r be the reset date with $0 < r \le M$. The holder of the option may exercise it at any tenor between T_r and T_M, but may not exercise it before time T_r. To value a Bermudan swaption, we need to find the optimal stopping time τ taking values in $\mathcal{T} := \{r, r+1, \ldots, M\}$ which maximises the option value:

$$u\left(T_0, R_{T_0}; a\right) = \max_{\tau \in \mathcal{T}} \mathbb{E}\left[e^{-\int_T^{T_\tau} R_s ds} max\left\{g(T_\tau, R_{T_\tau}; a), 0\right\}\right]$$

Then the value $u\left(T_0, R_{T_0}; a\right)$ can be obtained from the following recurrent formula:

$$\begin{aligned}
u(T_M, R_{T_M}; a) &= g(T_M, R_{T_M}; a) = 0 \\
v(T_n, R_{T_n}; a) &= \underbrace{\mathbb{E}\left[e^{-\int_{T_n}^{T_{n+1}} R_s ds} u\left(T_{n+1}, R_{T_{n+1}}; a\right) \mid R_{T_n}\right]}_{\text{European operator training on } [T_n, T_{n+1}]} (n = 0, \cdots, M-1), \\
u(T_n, R_{T_n}; a) &= \max\left\{g(T_n, R_{T_n}; a), v(T_n, R_{T_n}; a)\right\}
\end{aligned}$$

BSDE The BSDE under Hull White model is given by:

$$-\mathrm{d}Y_t = -R_t Y_t \mathrm{d}t - Z_t{}^\top \mathrm{d}W_t$$

Then the product can be priced and hedged with the algorithm 3.4.1. The clarified issue is:

1. Coefficients: $a = (\theta(t), a, \sigma, K)$.
2. The terminal payoff is 0.
3. The early exercise payoff is:

$$\begin{aligned}
g(t, R_t; a) = &\sum_{i=1}^{M-1} K\left(T_i - T_{i-1}\right) P\left(t, T_i\right) \\
&+ \left\{1 + K\left(T_i - T_{i-1}\right)\right\} P\left(t, T_M\right) - P\left(t, T_0\right).
\end{aligned}$$

4. no Markovian Variable.

Test Design In this product, we introduce two classes `HullWhiteModel` and `Swaption`. We write unit test as follows: For `HullWhiteModel`, we intend to give following test method:

- `test_drift`: This is to test the correctness of the drift function, we test the consistence of output shape and value of the function and the ground truth.
- `test_diffusion`: This is to test the correctness of the diffusion function, we test the consistence of output shape and value of the function and the ground truth.
- `test_correlation_matrix`: This is to test the correctness of the correlation matrix of multi-dimensional Brownian motion, we test the consistence of output shape and value of the function and the ground truth. We also test the property of positive-difinite.
- `test_martingale`: This is to test the martingale property for the SDE under risk-neutral measure via Monte Carlo simulation.

For `Swaption`, we intend to give following test method:

- `test_payoff` This function is to test the payoff at each certain time point. We just need to test the shape and the output value
- `test_martingale` This is to test whether the MC price given by the class is close to the analytical solution given by HW model.

4 Experiments

Thanks to Abadi et al. (2015), we can implement our idea with deep learning framework in tensorflow. We evaluate the performance of our algorithm in several cases.

- (Equity product)Time-dependent GBM model: single-asset European call, floating-strike lookback call, geometric Asian options; multi-asset European geometric basket call option.
- (Equity product) Heston's SV model: forward.
- (Fix-income product)Hull-White model: zero coupon bond, Bermudan swaption (with early exercise)

We consider such cases because they allow us to obtain accurate benchmarks from closed-form formulas of derivative prices, which are reviewed in the appendix

4.1 Experiment Specifications

In problems of equity products, we fix the maturity time $T = 1$, and use $N = 100$ time steps to discretize the asset price SDE. In problems of fix-income products, things are divided as: 1. For,zero coupon bond, we fix the maturity time $T = 1$; 2. For interest rate swap and swaption, we let $T_m = 3$ (T_m is the maturity date). 3. For Bermudan swaption, we let $T_m = 3$ too. For all products, we set $N = 10$ time steps per year to discretize the short rate process. To sample the coeffcients, we use two types of distributions. $U(a, b)$ refers to the uniform distribution over $[a, b]$ and $N(\mu, \sigma)$ refers to the normal distribution with mean μ and standard deviation σ. We always sample the initial asset price from $N(1, 0.2)$ and truncate them at 0.05 and 3 if the generated price falls outside this range.

Representation of coefficients. In our implementation, for example, we represent $r(t)$ and $\sigma_i(t), i = 1, \cdots, d$ by their values on a time grid with 20 uniform time steps over $[0, T]$, the lifetime of the derivative. In the time-homogeneous models, although $r(t)$ and $\sigma_i(t)$ for GBM and $\theta(t)$ for HW model are constant coefficients, we still treat them as if they were functions of t.

Design of PONet. For the PONet, we summarize the width of the layers (hidden and output) in Table 1. For example, when $d = 1$, the branch net has two hidden hidden layers and one output layer and there are 15 neurons on each layer. We increase the number of hidden layers in the branch and trunk nets as the dimension d becomes greater. In all the cases, we use the ReLU function defined by $\max\{x, 0\}$ as the activation function of each layer with only one exception for the forward value under the Heston model. In that case, as the forward value can become negative, we use the identity function for activation in the output layers of the branch and trunk nets.

Training. We summarize the training setting in Table 2. We use the popular Adam optimizer (Kingma and Ba (2014)) to minimize the loss function $\mathcal{J}(\theta)$. For every component network, we initialize its parameters using Xavier initialization (Glorot and Bengio (2010)), which can prevent the initial weights in a deep network from being either too large or too small. This method sets the weight of the j-th layer to follow the uniform distribution over $[-1/\sqrt{n_j}, 1/\sqrt{n_j}]$, where n_j is the number of neurons on the j-th layer. We also apply batch normalization to the input of the branch and trunk nets. In all the problems, we observe convergent behavior after running 1000 iterations. For training data, we sample B (batch size) coefficients from a coefficient distribution μ and for each sampled coefficient, we generate M (sample size) asset price paths. See Table 2 for the values of B and M for training.

Testing. We discuss the testing for European style products at this section and leave the Bermudan products to the following sub-section. For testing data, we sample B (batch size) coefficients from a coefficient distribution μ and for each sampled coefficient, we generate M

(sample size) asset price paths. See Table 2 for the values of B and M for testing. For each triplet $(a_j, t_n, X^m_{t_n})$ ($j = 1, \cdots, B$, $m = 1, \cdots, M$, $n = 0, 1, \cdots, N$ (time steps on the time horizon)), we calculate the percentage error of the price from our PONet by using the price from the closed-form formula as the benchmark. We report the mean and standard deviation of the percentage error over all the triplets. It should be noted that in our testing, we have sampled paths on the distribution same as what is from training data.

Network	$d = 1$	$d = 10$
Branch net	$[15, 15, 15]$	$[15, 15, 15, 15]$
Trunk net	$[15, 15, 15]$	$[15, 15, 15, 15]$
Embedding net	$[20, 20]$	$[20, 20]$
PI net	$[10, 10]$	$[15, 15]$

Table 1: Structure of PONet: Each list corresponded to the structure of a feed forward neural network, for example $[15, 15, 15]$ means that this FNN has three layers and each layer has 15 neurons and so on so forth.

name	value
optimizer	Adam
no. of iterations	1000
learning rate	0.005
sample size(M)	200
sample size(M)	100

Table 2: Setting of training and testing: B is the number of batches which is same as that in the loss function given from last section; M is the number of sample paths generated from a single set of input coefficient vectors.

4.2 Equity Products under Time-dependent GBM and Heston Model

4.2.1 Time-dependent GBM Model

We parametrize $r(t)$ as:

$$r(t) = r_0 + r_1 t + r_2 t^2, \quad r_0, r_1, r_2 > 0,$$

This is the parametric form of the interest rate function $r(t)$ where r_0 means the average level of the rate curve while r_1 and r_2 represents the linear and quadratic coefficients of calendar time. We parametrize $\sigma_i(t)$ as:

$$\sigma_i(t) = \bar{\sigma}_i e^{-\beta_i (T-t)}, \quad \bar{\sigma}_i, \beta_i > 0, \quad i = 1, \cdots, d$$

This class of functions shows exponential decay with respect of time to maturity and $\bar{\sigma}_i$ is the initial volatility rate of the i-th asset and β_i is the decay rate. We sample the coefficient functions by sampling their parameters. For simplicity, we assume that the assets have the same $\bar{\sigma}_i$ and β_i and they also share the same correlation. To represent these coefficient functions, we discretize $r(t)$ and $\sigma_i(t)$ on a time grid with 20 equal time steps. See Table 3 for the wide and narrow distributionsof the parameters.

4.2.2 Heston Model

We consider the Heston SV model (Heston (1993)) for d assets, whose dynamics under the risk-neural measure is given by

$$\mathrm{d}S^i_t = r S^i_t \mathrm{d}t + \sqrt{V^i_t} S^i_t \mathrm{d}W^{i,S}_t, \quad i = 1, \cdots, d,$$

parameter	distribution
r_0	$U(0.025, 0.65)$
r_1	$U(0.0015, 0.0035)$
r_2	$U(0.001, 0.008)$
$\bar{\sigma}_i, i = 1 \cdots, d$	$U(0.3, 0.7)$
$\beta_i, i = 1 \cdots, d$	$U(0.02, 0.03)$
ρ_S	$U(-\frac{1}{d} + 0.1, 0.6)$
κ	$N(1.0, 0.2)$

Table 3: Sampling distributions of coefficients under the time-dependent GBM model: The 1st column denotes the symbol of each coefficient of the time-dependent geometric Brownian motion (GBM) model; The 2nd colume denotes the distribution of coefficients.

$$\mathrm{d}V_t^i = \alpha_i(\theta_i - V_t^i)\mathrm{d}t + \sigma_i\sqrt{V_t^i}\mathrm{d}W_t^{i,V}, \quad i = 1, \cdots, d,$$
$$\mathrm{d}W_t^{i,S}\mathrm{d}W_t^{j,S} = \rho_{ij}^S\mathrm{d}t,\ \mathrm{d}W_t^{i,S}\mathrm{d}W_t^{i,V} = \rho_i^{S,V}\mathrm{d}t,\ \mathrm{d}W_t^{i,V}\mathrm{d}W_t^{j,V} = 1_{\{i=j\}}\mathrm{d}t.$$

For simplicity, we assume that the Brownian motions that drive the variance rates of different assets are uncorrelated. In this model, $V_t = (V_t^1, \cdots, V_t^d)$ is a new state variable and should be added to the state vector and then $X_t = (S_t^1, \cdots, S_t^d, V_t^1, \cdots, V_t^d)$.

We consider a forward contract on the portfolio of these assets. Suppose that F is the forward price determined before time 0, and let κ equal F divided by the simple average of the initial asset prices. The fair value of the forward contract at time t is given by

$$u(t, S_t, V_t; a) = \frac{1}{d}\sum_{i=1}^{d} S_t^i - \kappa e^{-r(T-t)}\frac{1}{d}\sum_{i=1}^{d} S_0^i.$$

In our experiment, we assume θ_i, κ_i, and σ_i are the same among the assets. We also let $\rho_{ij}^S = \rho_S$ for all pairs of (i, j) and $\rho_i^{S,V} = \rho_{S,V}$ for all i. See Table 4 for the distribution of the parameters.

parameter	distribution
r	$U(0.025, 0.65)$
$\theta_i, i = 1 \cdots, d$	$U(0.3, 0.9)$
$\alpha_i, i = 1 \cdots, d$	$U(0.1, 0.15)$
$\sigma_i, i = 1 \cdots, d$	$U(0.015, 0.02)$
ρ_S	$U(-\frac{1}{d} + 0.1, 0.6)$
$\rho_{S,V}$	$U(-0.5, 0.01)$
κ	$N(1.0, 0.2)$

Table 4: Sampling distributions of coefficients under the Heston model: The 1st column denotes the symbol of each coefficient of the Heston model; The 2nd colume denotes the distribution of coefficients.

4.2.3 Numerical Results

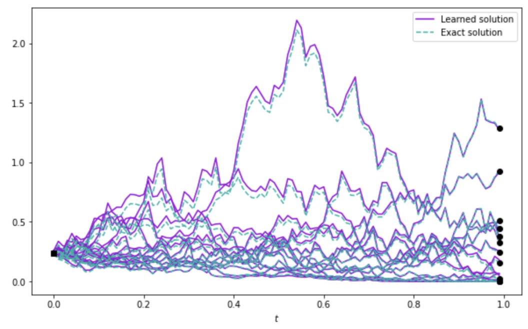
We report the test errors in Table 5 on both wide and narrow distribution. We also plot the simulated option price paths from the PONet and the closed-form benchmark in 4. In each plot, we give detailed description.

4.3 Fixed-Income Products

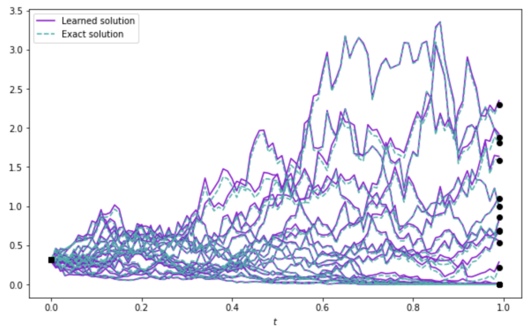
In Hull-White model, we initialize the forward rate curve with the flatten curve, and there is a constraint that $\theta_r = R_0$. We also input the curve into operator network as a function. Then

Derivative	iteration	Percentage Error mean (stdev)
GBM European ($d = 1$)	100	0.2893 (0.2473)
	500	0.0432 (0.0418)
	1000	0.0263 (0.0202)
GBM European ($d = 10$)	100	0.1776 (0.1065)
	500	0.0459 (0.0443)
	1000	0.0278 (0.0235)
GBM Lookback ($d = 1$)	100	0.1726 (0.2473)
	500	0.0486 (0.0403)
	1000	0.0330 (0.0262)
GBM Geometric Asian ($d = 1$)	100	0.2812 (0.1635)
	500	0.0556 (0.0461)
	1000	0.0463 (0.0388)
Heston forward ($d = 1$)	100	0.2956 (0.2473)
	500	0.0191 (0.0200)
	1000	0.0119 (0.0097)
Heston forward ($d = 10$)	100	0.1868 (0.1144)
	500	0.0184 (0.0160)
	1000	0.0135 (0.0121)

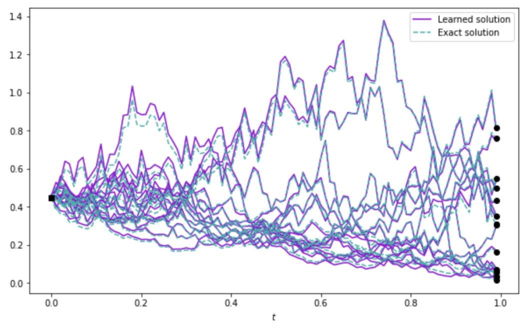
Table 5: Test errors of some options under the time-dependent GBM model and Heston stochastic volatility model. The 3rd column shows the percentage error corresponded to asset paths sampled from the distribution given in the table 3 and 4 (distribution for both training and testing). For each product, we illustrate the test percentage error for different numbers of iterations where we can see a rapid decrease for first 100 iterations. We also note the standard deviation of the test error and we can find that the standard deviation also converge to a low level. First four blocks notes the results under time-dependent GBM for vanilla European options, basket options, lookback options and geometric Asian options and the last two blocks shows the results of forward under 1 dimension and 10 dimensions.



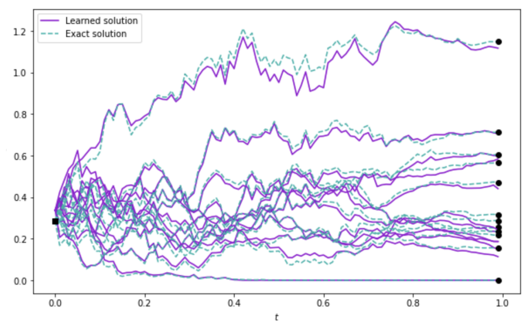
(a) GBM European $d = 1$



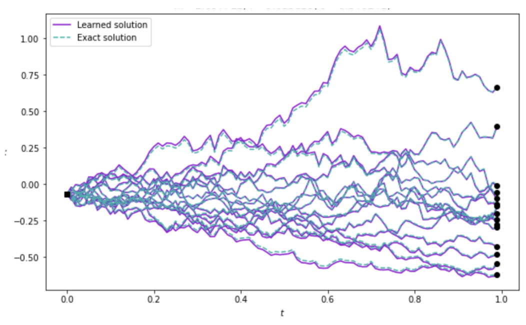
(b) GBM European $d = 10$



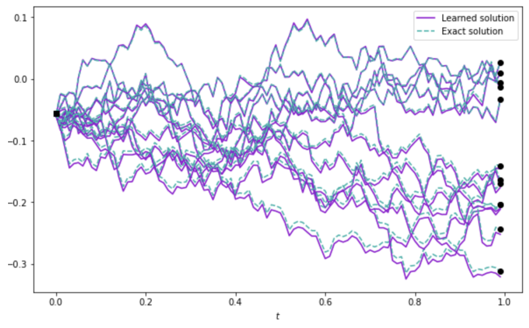
(c) GBM Lookback



(d) GBM Asian ($d = 1$)



(e) Heston forward $d = 1$



(f) Heston forward $d = 10$

Figure 4: Analytical price and predicted price by PONet under the time dependent GBM model and Heston stochastic volatility model. For each subplot, the purple curves are simulated derivative paths from the learned operator network and we legend this as learned solution while the light blue curves are corresponded to analytical solutions.

the table 6 gives the distribution of parameters:

parameter	distribution
$\theta(t) = R_0$	$U(0.04, 0.06)$
κ	$U(0.3, 0.5)$
σ	$U(0.02, 0.04)$

Table 6: Sampling distributions of coefficients under the Hull White model: The equation $\theta(t) = R_0$ means that there is a constrain in HW model and then we just need to only sample θ rather than sample both θ and R_0. Since we let the rate curve become flat, we just need to sample $\theta = \theta(t)$ in a uniform distribution.

4.3.1 Zero Coupon Bond

Since the bond price $P(t,T)$ is the most crucial component in the payoff structure of almost every fixed income instrument. We begin by testing the price of a zero-coupon bond $P(t,T)$, which can be considered as a special case where the strike of a bond option is zero. In this case, the dynamics of the zero-coupon bond follow the Heath-Jarrow-Morton (HJM) model. For a bond with maturity T, we have the following Backward Stochastic Differential Equation (BSDE):

$$\begin{aligned} &dP(t,T) = R_t P(t,T)dt + \sigma B(t,T)P(t,T)dW_t \\ &P(T,T) = 1 \end{aligned}$$

where $B(t,T) = \frac{1-e^{-a(T-t)}}{a}$. Then plug this into our European framework. We can yield the result as following plot, which is a comparison between analytical solution and value given by operator:

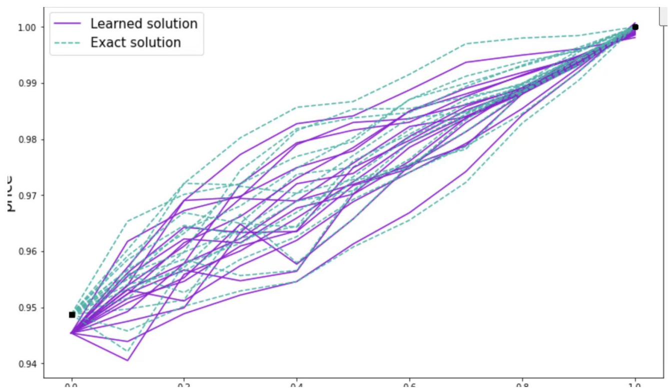


Figure 5: result of zero coupon bond. We can find the the learned solution converge to the analytical solution and the initial value and interiors value both come close to the path of analytical solution.

4.3.2 Bermudan Swaption

Building upon the successful results obtained thus far, we proceed with the testing of the Bermudan swaption. For data, we sample coefficients of Hull-White model under the same distribution as in 6. Then we list the elements of the Bermudan swaption of interest as following table.

To establish a benchmark for evaluation, we utilize the numerical solution provided by the `tf_quant_finance` library. We evaluate the initial price of the swaption at various states

item	value
expiry date	1y
maturity date	3y
swap increment	1y
exercise date	1y, 2y

and parameters, comparing them to the benchmark to assess the level of closeness. For these experiments, we set the fixed rate as zero and the product is equivalent as the swap rate. Then we change the volatility and report the result again. From 7, We observe that as the input

σ	$\theta_r = r_0$	price from operator	benchmark
0.03	0.052	0.0846	0.0901
	0.053	0.0854	0.0917
	0.055	0.0872	0.0949
	0.057	0.0888	0.0982
	0.058	0.0896	0.0997
0.05	0.052	0.0846	0.0835
	0.053	0.0854	0.0851
	0.055	0.0872	0.0885
	0.057	0.0884	0.0917
	0.058	0.0897	0.0933

Table 7: Result of the Bermudan Swaption price comparison with $\kappa = 0.4$ and $\sigma = [0.03, 0.05]$. The first colume give the level of the flatten forward rate curve at initial time; The second colume (price from operator) gives the price of Bermudan Swaption calculated from the trained operator at initial time; The third colume is the same price given by the benchmark from `tf_quant_finance` library

parameter approaches the sample boundary, the magnitude of the error increases significantly. Furthermore, the operator demonstrates the capability to learn the relationship between price and volatility, exhibiting monotonic behavior.

Another test to assess the Bermudanarity of the swaption involves substituting the analytical solution with the continuation value of the last sub-time interval [4]. Since the continuation value of the final time interval corresponds to the European swaption, which possesses a closed-form solution, this approach effectively eliminates the error introduced by the terminal condition. Consequently, we can examine whether the resulting error is smaller than the benchmark or not. The price yield from the analytical continuation value is denoted as price (cont. from ana.) while the price from the last network inference is denoted as price (cont. from net). We can find that the main issue of this algorithm is that training a single operator step by step may accumulate error. If the error of the last time step is not tiny enough, the continuation value given by the last step will not be accurate and therefore the final price would be far from the true value. However this issue happens in all numerical schemes and what we need to do is to polish our deep learning algorithm and train more steps with more suitable hyper parameters so that we can price long time period Bermudan products with more accuracy.

[4]We just substitute the continuation value in the last exercise period into the analytical ones since only in the last time period does it have analytical value which is the European swaption or Bond option

σ	$\theta_r = R_0$	price (cont. from net)	price (cont. from ana.)	benchmark
	0.052	0.0846	0.0875	0.0901
	0.053	0.0854	0.0883	0.0917
0.03	0.055	0.0872	0.0901	0.0949
	0.057	0.0888	0.0920	0.0982
	0.058	0.0896	0.0927	0.0997
	0.052	0.0846	0.0875	0.0835
	0.053	0.0854	0.0884	0.0851
0.05	0.055	0.0872	0.0902	0.0885
	0.057	0.0884	0.0919	0.0917
	0.058	0.0897	0.0928	0.0933

Table 8: The first colume give the level of the flatten forward rate curve at initial time; The second colume (price from operator) gives the price of Bermudan Swaption calculated from the trained operator at initial time; The third colume gives the Bermudan Swaption price when the continuation value of the last exercise period is calculated by analytical solution rather than the neural operator trained from the last time interval (We just substitute the continuation value in the last exercise period into the analytical ones since only in the last time period does it have analytical value which is the European swaption or Bond option). The fourth colume is the same price given by the benchmark from `tf_quant_finance` library.

5 Extensions

5.1 Under Jump Processes

5.1.1 Loss Function

We only discuss finite activity case. In this subsection we assume that the jumps are those of a compound Poisson process. Without loss of generality, we discuss the Merton diffusion model under time dependent GBM. We denote the intensity as λ for each asset and the levy measure for each asset is $\nu^i(\mathrm{d}z) = \lambda\varphi(z)\mathrm{d}z, i = 1, \cdots, d$ with $\lambda > 0$ and $\varphi(z) = \frac{1}{\sqrt{2\pi}\sigma_J} e^{-\frac{1}{2}\left(\frac{x-\mu_J}{\sigma_J}\right)^2}$ for each z^i of each asset. Then the SDE of this process of the i-th underlying asset under risk neutral measure can be written as:

$$\frac{\mathrm{d}S_t^i}{S_{t-}^i} = r(t)\mathrm{d}t + \sigma^i(t)\mathrm{d}W_t^i + \int_{\mathbb{R}} \left(e^{z^i} - 1\right) \tilde{N}^i(\mathrm{d}z, \mathrm{d}t)$$

Then the related BSDE is:

$$\begin{aligned} -\mathrm{d}Y_t &= -r(t)Y_t\mathrm{d}t - (Z_t)^\top \mathrm{d}W_t - \int_{\mathbb{R}} U(t,z)\tilde{N}(\mathrm{d}z, \mathrm{d}t) \\ Y_T &= g(X_T; a) \end{aligned}$$

where $U(t,z) = u(t, S_{t-}e^z) - u(t, S_{t-})$. We now discuss the discretized scheme of both SDE and BSDE with jump diffusion. For SDE, we have the solution, for any $i = 1, \cdots, d$:

$$S_T^i = S_t^i \exp\left\{\left(\tilde{r} - \frac{\tilde{\sigma}^{2,i}}{2}\right)(T-t) + \tilde{\sigma}^i W_t^i - \int_{\mathbb{R}} (e^z - 1 - z)\,\nu^i(\mathrm{d}z)(T-t) + \int_t^T \int_{\mathbb{R}} z\tilde{N}^i(\mathrm{d}z, \mathrm{d}s)\right\}$$

Then it is suffice to simulate the process of $ln(S_t)$. We have the following discretization scheme:

$$\ln S_{t_{n+1}}^m = \ln S_{t_n}^m + \left(r(t_n)\mathbf{1} - \frac{\sigma^2(t_n)}{2} - \lambda\left(e^{\mu_J + \frac{\sigma_J^2}{2}} - 1\right)\mathbf{1}\right)\Delta t + \sigma(t_n) \circ \Delta W_{t_n}^m + \sum_{j=N_{t_n}}^{N_{t_{n+1}}} z_j$$

For the BSDE, we discretize it as

$$\begin{aligned}
&Y_{t_N} = g(X_{t_N}; a),\\
&Y_{t_n} \approx Y_{t_{n+1}} - r(t_{n+1})Y_{t_{n+1}}\Delta t - Z_{t_n}^{\top}\Delta W_{t_n} - \sum_{j=N_{t_n}}^{N_{t_{n+1}}} \left(u(t_n, S_{t_n}e^{z_j}) - u(t_n, S_{t_n})\right) -\\
&\quad \Delta t \int_{\mathbb{R}^d} \left(u(t_n, S_{t_n}e^{z}) - u(t_n, S_{t_n})\right)\nu(\mathrm{d}z), n = N-1, \cdots, 0.
\end{aligned}$$

For the discretization of BSDE, we take care about the term: $\int_{\mathbb{R}} U(t,z)\tilde{N}(\mathrm{d}z, \mathrm{d}t)$. It can be decomposed as: $\sum_{j=N_{t_n}}^{N_{t_{n+1}}} \left(u(t_n, S_{t_n}e^{z_j}) - u(t_n, S_{t_n})\right) - \Delta t \int_{\mathbb{R}^d} \left(u(t_n, S_{t_n}e^{z}) - u(t_n, S_{t_n})\right)\nu(\mathrm{d}z)$. The question is of approximating the compensator term. The first option, which is feasible when the dimension d is low, is to employ a standard quadrature rule. When the dimension increases on can resort to Monte Carlo integration. However, the training of the networks we introduced involves a further Monte Carlo simulations during the training phase, so that such approach would lead to costly nested Monte Carlo simulations. The alternative approach is proposed by Gnoatto et al. (2022) where they approximate the compensator with another set of neural networks. Thus in this problem we approximate the term with an another unified operator network $w_\rho(t, x; a)$. Then we can put this constraint into the penalty function:

$$\mathbb{E}\left[\left(\sum_{j=N_{t_n}}^{N_{t_{n+1}}} \left(u(t_n, S_{t_n}e^{z_j}) - u(t_n, S_{t_n})\ \right) - \Delta t w_\rho\left(t_n, S_{t_n}; a\right)\right)^2\right]$$

Then we give the loss function construction of the inner loss $\mathcal{L}(\theta, \rho; a)$ via following steps:

- For $m = 1, \cdots, M$, generate a path of the price process for each asset using Euler scheme. On each path, calculate the related states at each time point. We denote the state vector at time t_n of asset i on path m by $X_{t_n}^{i,m}$ and $X_{t_n}^{m}$ is the collection of all $X_{t_n}^{i,m}$. We also keep the generated Brownian increment path $\{\Delta W_{t_n}^m : n = 0, \cdots, N-1\}$ and jump increments.
- Step 2: For $n = 0, ..., N-1$ and $m = 1, ..., M$, calculate

  $$Z_{t_n}^m = \sigma(t) \circ S_{t_n} \circ \nabla_X G_\theta(a, t_n, X_{t_n}^m),$$

  This is the hedging ratio for the product during the time interval without early exercise.
- Step 3: Evaluate the BSDE value $Y_{t_n}^m$ with:

  $$\begin{aligned}
  &Y_{t_N} = g(X_{t_N}; a),\\
  &Y_{t_n} \approx Y_{t_{n+1}} - r(t_{n+1})Y_{t_{n+1}}\Delta t - Z_{t_n}^{\top}\Delta W_{t_n} - \sum_{j=N_{t_n}}^{N_{t_{n+1}}} \left(u(t_n, S_{t_n}e^{z_j}) - u(t_n, S_{t_n})\right) -\\
  &\quad \Delta t w_\rho\left(t_n, S_{t_n}; a\right), n = N-1, \cdots, 0.
  \end{aligned}$$

- Step 4: The loss function of the model $\mathcal{L}_{model}(\theta, \rho; a)$ is given by the output from step 4 in the last subsubsection, i.e.,

  $$\mathcal{L}_m(\theta, \rho; a) = \frac{1}{M}\sum_{m=1}^{M}\sum_{n=0}^{N}\left(G_\theta(\tilde{a}, t_n, X_{t_n}^m) - Y_{t_n}^m\right)^2.$$

  and the penalty function is given by:

  $$\mathcal{L}_{penalty}(\theta, \rho; a) = \frac{1}{N}\sum_{n=0}^{N-1}\left(\sum_{j=N_{t_n}}^{N_{t_{n+1}}} \left(u(t_n, S_{t_n}e^{z_j}) - u(t_n, S_{t_n})\ \right) - \Delta t w_\rho\left(t_n, S_{t_n}; a\right)\right)^2$$

Then the loss function is:

$$\mathcal{L}(\theta, \rho; a) = \mathcal{L}_{model}(\theta, \rho; a) + \mathcal{L}_{penalty}(\theta, \rho; a)$$

To obtain $\mathcal{J}(\theta)$, we need to sample a from distribution μ. Suppose that we have a batch of sampled coefficients $\{a_1, \cdots, a_B\}$. We calculate $\mathcal{J}(\theta)$ by

$$\mathcal{J}(\theta, \rho) = \frac{1}{B} \sum_{j=1}^{B} \mathcal{L}(\theta, \rho; a_j).$$

Testing Design: On method to test the loss function is to plug the analytical solution into the loss function to check whether the loss function value is close to zero. For BSDE under Merton jump diffusion model, the analytical solution can be written as a series form of BS formulas with different implied volatility values. We can expand this series to a certain order and check whether the loss function value is below a certain value. This series form solution as given as:

$$V(S, K, T, r, \sigma, m, s, \lambda) = \sum_{k=0}^{\infty} \frac{e^{-\bar{m}\lambda T}(\bar{m}\lambda T)^k}{k!} V_{BS}(S, K, T, r_k, \sigma_k)$$

where $\sigma_k = \sqrt{\sigma^2 + k\frac{s^2}{T}}$ and $r_k = r - \lambda(\bar{m} - 1) + \frac{k\ln(\bar{m})}{T}$ where $\bar{m} = e^{m+s^2/2}$.

5.1.2 Algorithm of European Style Products

We summarize the training framework of European style problem on a certain time interval $[T_1, T_2]$. In this framework, the initial time t_0 is corresponded to T_1 and terminal time T is corresponded to T_2. The algorithm is states as following procedure:

Algorithm 3 Training of the Pricing Operator Network on $[T_1, T_2]$ under Jump Diffusions

```
Specify the sampling distribution $\mu$ and initialize $\theta$.
for $i = 1, 2, ...$ do
    Sample a batch of coefficients $\{a_j,\ j = 1, \cdots, B\}$ from $\mu$.
    For each $j$, generate $M$ asset price paths on $[T_1, T_2]$ under $a_j$ and calculate $\mathcal{L}(\theta, \rho; a_j)$. .
    Calculate $\mathcal{J}(\theta, \rho)$.
    Update $(\theta, \rho)$ by gradient descent.
end for
return RETURN The well trained operator network $u_\theta(t, x; a)$ defined on $[T_1, T_2]$.
```

Then for early exercise cases, things are same as previous discussion.

5.1.3 Case 1: Callable Yield Note under Stochastic Volatility Jump Model

Stochastic Volatility Jump (SVJ) There is strong evidence that stochastic volatility models cannot handle short-term smiles properly. The addition of jumps makes these models more realistic: Under the model first proposed by Bates Bates (1996), the jump term is the Merton jump diffusion and the distribution of jump size is under logrithm normal distribution. The dynamics can be described as:

$$\begin{aligned} dS_t &= (r(t) - \lambda m)S_t dt + \sqrt{V_t} S_t dW_t^s + \left(e^J - 1\right) S_t dN_t \\ dV_t &= \kappa(\theta - V_t)dt + \varepsilon\sqrt{V_t} dW_t^v \end{aligned}$$

where κ is a mean-reverting rate, θ is a long-term variance, ϵ is a volatility of volatility, W_t^v and W_t^s) are correlated Wiener processes with constant correlation ρ. λ is the intensity J is the

random variable representing the jump size with the law $\nu(\cdot)$ and $m = \mathbb{E}_\nu\left[\left(e^J - 1\right)\right]$ which is the relative mean of jump size.
BSDE By Ito lemma, the BSDE under SVJ is given as:

$$-\mathrm{d}Y_t = -r(t)Y_t\mathrm{d}t - (Z_t)^\top \mathrm{d}\boldsymbol{W}_t - \int_{\mathbb{R}} U(t,z)\tilde{N}(\mathrm{d}z,\mathrm{d}t)$$

where $\boldsymbol{W}_t = (W_t^s, W_t^v)$ and:

$$\int_{\mathbb{R}} U(t,z)\tilde{N}(\mathrm{d}z,\mathrm{d}t) = \left(u(t_n, S_{t_n}e^J) - u(t_n, S_{t_n})\right)\mathrm{d}N_t - \Delta t\int_{\mathbb{R}^d}\left(u(t_n, S_{t_n}e^z) - u(t_n, S_{t_n})\right)\nu(\mathrm{d}z)$$

Then since we have BSDE, we introduce the callable yield note.
Callable Yield Note: Callable Yield Notes allow investors to receive interest payments, regardless of the movements in the underlying. The CYNs will return the principal amount if the underlying does not reach or breach the Knock-In Level at the end of the trade (Some CYN is at any time during the trade). Otherwise, investors may receive less than 100% of the principal amount relative to the percentage change of the underlying. Furthermore, the Issuer has the right to call the CYNs. If the CYNs are called, investors will receive 100% of the principal amount plus any accrued but unpaid interest. The upside of CYN is that it provides high yield if the notes are called prior to maturity on one of the specified call dates, or if held to maturity and the underlying has not breached the specified Knock-In Level. The downside is that If the notes are not called prior to maturity, the notes are not principal protected and if the Knock-In Level is breached, adverse performance of the underlying could result in a loss of principal. Then we try to formuarize the payment of the CYN. Denote the callable dates as $\mathcal{T} := \{\tau_0 = 0, \tau_1, \ldots, \tau_k, \cdots, \tau_N = T\}$. Denote the annualized coupon as c and the barrier level of knock-in as B. Then from the perspective of the investor, the payoff of the trade is:

$$g(\tau_k, S_{\tau_k}; a) = c\cdot(\tau_k - \tau_0), k = 1, \cdots, N-1$$
$$g(\tau_N, S_{\tau_N}; a) = c\cdot(\tau_N - \tau_0) - \max(B - p_T, 0)$$

where p_t is the relative percentage of the initial value fot the basket of assets:

$$p_t = \min_{j\in\{1,2,\cdots,d\}}\left[\frac{S_t^j}{S_0^j}\right]$$

Then the dynamic programming rule is given as:

$$\begin{aligned}
u(\tau_N, S_{\tau_N}; a) &= g(\tau_N, S_{\tau_N}; a)\\
v(\tau_k, S_{\tau_k}; a) &= \mathbb{E}\left[e^{-\int_{\tau_k}^{\tau_{k+1}} r(s)ds} u\left(\tau_{k+1}, S_{\tau_{k+1}}; a\right) \mid S_{\tau_k}\right], (k = 0, \cdots, N-1),\\
u(\tau_k, S_{\tau_k}; a) &= \min\left\{g(\tau_k, S_{\tau_k}; a), v(\tau_k, S_{\tau_k}; a)\right\}
\end{aligned}$$

Since the issuer is the early exercise side, the investor should use the minimum to do the dynamic programming. Finally, we can use the general algorithm to price this CYN. Then the product can be priced and hedged with the algorithm 3.4.1. The clarified issue is:

1. Coefficients: $a = (r(t), \kappa, \theta, \epsilon, \lambda, m, s, \rho, c, B, K)$. (For callable Asian $a = (r(t), \kappa, \theta, \epsilon, \lambda, m, s, \rho, K)$)
2. The terminal payoff is $g(\tau_N, S_{\tau_N}; a) = h(\tau_N, S_{\tau_N}) = c\cdot(\tau_N - \tau_0) - \max(B - p_T, 0)$. (For callable Asian the payoff is $\left(\frac{1}{d}\sum_{i=1}^d A_T^i - K\right)^+$)
3. The early exercise payoff is: $g(\tau_k, S_{\tau_k}; a) = h(\tau_k, S_{\tau_k}) = c\cdot(\tau_k - \tau_0), k = 1, \cdots, N-1$. (For callable Asian the payoff is $\left(\frac{1}{d}\sum_{i=1}^d A_t^i - K\right)^+$)

4. no Markovian Variable. (For callable Asian, the variable is A_t which is defined before)

Test Design In this product, we introduce two classes `StochaticVolJump` and `CallableYieldNote`. For `StochaticVolJump`, we intend to give following test method:

- `test_drift`: This is to test the correctness of the drift function, we test the consistence of output shape and value of the function and the ground truth.
- `test_diffusion`: This is to test the correctness of the diffusion function, we test the consistence of output shape and value of the function and the ground truth.
- `test_jumpdiffusion`: This is to test the correctness of the jump diffusion function, we test the consistence of output shape and value of the function and the ground truth.
- (Reduce test) This test is just let the jump intensity $\lambda = 0$ and test whether all the test related to stochastic volatility without jump can pass.
- `test_martingale`: This is to test the martingale property for the SDE under risk-neutral measure via Monte Carlo simulation.

For `CallableYieldNote`, we intend to give following test method:

- `test_payoff` This function is to test the payoff at each certain time point. We just need to test the shape and the output value
- `test_martingale` We set the coupon become zero and the product reduced to the European put option. Then we test the MC price under this constraint and check whether this is closed to the Black-Scholes solution (In this test we just use the GBM as the SDE).

5.1.4 Case 2: Stochastic Local Volatility Jump Model under Hull White Interest Rate

Based on Heston Model, Merton jump diffusion model, Hull and White model and we combine all these issue togather and also introduce the local volatility surface $v(S,t)$ which fits the market prices and dividend payment $k(t)$. For simplicity, we use one factor Hull and White model to model the stochastic interest rate. Then the final version of the most complicated model is demonstrated as follows:

$$\begin{aligned} dR_t &= \kappa_r(\theta_r - R_t)dt + \varepsilon_r dW_t^r \\ dS_t/S_t &= (R_t - k(t) - \lambda m)dt + v(S_t,t)\sqrt{V_t}dW_t^s + \left(e^J - 1\right)dN_t \\ dV_t &= \kappa(\theta - V_t)dt + \varepsilon\sqrt{V_t}dW_t^v \end{aligned}$$

And as in previous discussion, we assume $J \sim \mathcal{N}(\mu_J, \sigma_J^2)$. And $dW_t^r dW_t^s = 0$, $dW_t^r dW_t^v = 0$, $dW_t^v dW_t^s = \rho^{s,v}dt$. Then, we can find that The BSDE can be represented as:

$$-\mathrm{d}Y_t = -R_tY_t\mathrm{d}t - (Z_t)^\top \mathrm{d}\boldsymbol{W}_t - \int_{\mathbb{R}} U(t,z)\tilde{N}(\mathrm{d}z,\mathrm{d}t)$$

Z_t is a function of (R_t, S_t, V_t), $\boldsymbol{W}_t = (W_t^r, W_t^s, W_t^v)^\top$. And

$$\begin{aligned} \int_{\mathbb{R}} U(t,z)\tilde{N}(\mathrm{d}z,\mathrm{d}t) &= \left(u(t_n, R_{t_n}, S_{t_n}e^J, V_{t_n}) - u(t_n, R_{t_n}, S_{t_n}, V_{t_n})\right)\mathrm{d}N_t \\ &\quad - \left(\int_{\mathbb{R}^d} \left(u(t_n, R_{t_n}, S_{t_n}e^z, V_{t_n}) - u(t_n, R_{t_n}, S_{t_n}, V_{t_n})\right)\nu(\mathrm{d}z)\right)\mathrm{d}t \end{aligned}$$

In these BSDE solving problem. The coefficients are $a = (\kappa_r, \varepsilon_r, \theta_r, \lambda, m, \kappa, \theta, \varepsilon; k(t), v(s,t); \rho^{s,v})$ Then, again, we face with the callable yield note, the dynamic programming rule is given as:

$$u(\tau_N, R_{\tau_N}, S_{\tau_N}, V_{\tau_N}; a) = g(\tau_N, S_{\tau_N}; a)$$
$$v(\tau_k, R_{\tau_k}, S_{\tau_k}, V_{\tau_k}; a) = \mathbb{E}\left[e^{-\int_{\tau_k}^{\tau_{k+1}} R_s ds} u\left(\tau_{k+1}, R_{\tau_{k+1}}, S_{\tau_{k+1}}, V_{\tau_{k+1}}; a\right) \mid R_{\tau_k}, S_{\tau_k}, V_{\tau_k}\right], (k = 0, \cdots, N-1),$$
$$u(\tau_k, R_{\tau_k}, S_{\tau_k}, V_{\tau_k}; a) = \min\{g(\tau_k, S_{\tau_k}; a), v(\tau_k, R_{\tau_k}, S_{\tau_k}, V_{\tau_k})\}$$

The clarified issue is:

1. Coefficients: $a = (\kappa_r, \varepsilon_r, \theta_r, \lambda, m, \kappa, \theta, \varepsilon; k(t), v(s,t); \rho^{s,v}, c, B, K)$. (For callable Asian $a = (\kappa_r, \varepsilon_r, \theta_r, \lambda, m, \kappa, \theta, \varepsilon; k(t), v(s,t); \rho^{s,v}, K)$)
2. The terminal payoff is $g(\tau_N, S_{\tau_N}; a) = c \cdot (\tau_N - \tau_0) - \max(B - p_T, 0)$. (For callable Asian the payoff is $\left(\frac{1}{d}\sum_{i=1}^d A_T^i - K\right)^+$)
3. The early exercise payoff is: $g(\tau_k, S_{\tau_k}; a) = c \cdot (\tau_k - \tau_0), k = 1, \cdots, N-1$. (For callable Asian the payoff is $\left(\frac{1}{d}\sum_{i=1}^d A_t^i - K\right)^+$)
4. no Markovian Variable. (For callable Asian, the variable is A_t which is defined before)

Test Design We introduce the class `StochaticLocalVolJump`. We test this class with the same rule as testing previous classes related to SDEs. Since the drift and diffusions are almost same as previous ones. We just test the Euler step method to check whether the value at the next time step has the potential distribution.

- `test_conditional_distribution`: Do a one step forward simulation with sufficient large number of paths. We test whether the mean and the variance is consistence with the drift and diffusions.
- `test_cholesky`: Sample a vector and calculate the quadatic form corresponded to the decomposed matrix to check whether it is non negative.
- `test_martingale`: This is to test the martingale property for the SDE under risk-neutral measure via Monte Carlo simulation.

5.2 Experiments

To be completed...

6 Conclusions

1. Conclusions In the study of derivatives pricing with operator network, we have developed a neural network model to learn the mapping from the space of time, states, and coefficients to the derivative price. Our method is based on the representation of the pricing function by basis expansion, which separates the effect of coefficients on the price from that of time and states. We use separate neural networks called branch and trunk nets to capture these effects. We also introduce the embedding net to generate latent representations of the coefficient functions and the PI net to produce the theoretical property of permutation invariance that holds in many multi-asset derivatives. To train the neural network model, we construct a loss function by solving the BSDE associated with the pricing problem. The BSDE approach is flexible as it provides a unified framework to price different types of derivatives in general price models. Various numerical experiments demonstrate the ability of our model in learning this complicated

mapping even in the multi-dimensional case. In future research, we plan to extend the current method to consider options with early exercise features and price models with jumps.
2. Problems In Bermudan pricing, the key challenge is to approximate the continuation value functions recursively. This process involves solving a sequence of conditional expectation problems, which can be computationally expensive. However, if we can solve this problem effectively, the Bermudan problem can be reduced to a European and path-dependent problem. This is because the Bermudan option can be viewed as a series of European options with the possibility of early exercise at specific dates. Furthermore, since the option's payoff depends on the underlying asset's price path, it is also a path-dependent option. Therefore, if we can accurately approximate the continuation value operators, we can effectively reduce the Bermudan problem to a combination of European and path-dependent options. This can significantly simplify the pricing process and make it more computationally efficient. However, as mentioned earlier, the challenge lies in efficiently approximating the continuation value operators, which is a non-trivial task.
3. Extensions To prepare for future extensions, one interesting topic is operator training under jump models. The Merton diffusion model and SVJ (stochastic volatolity jump) model are the counterparts of GBM and Heston models, respectively. Fortunately, Gnoatto et al. (2023) has proposed a framework for Deep BSDE under jump models. The innovation lies in their use of an extra neural network to approximate the expectation of the jump term in high-dimensional cases, which allows for the extension of all deep BSDE without jump using this technique. When it comes to Bermudan pricing, we realize that the approximation of continuation value functions (operators) is independent of the formulation of the underlying asset dynamics or stochastic volatility. Thus, all we need to do is perform least squares with paths generated by Monte Carlo simulation. Once we have the value network, we can use dynamic programming to obtain the intrinsic value on each exercise date under each scenario. We can then use the framework of Gnoatto et al. (2023) once again to train the pricing operator with PricingONet proposed by our paper. In other words, whether under a jump model or not, the core problem is to design a framework to efficiently train the operator mapping from coefficients to continuation value functionsB. Finally, experiments with jump need to be tested in future research given all kinds of products without jump pass test.

A Some Analytical Solutions

Here we list benchmark solutions under time dependent GBM.

B Analytical Solutions under Time-dependent GBM Cases

We first show the analytical solution of vanilla European call option under time-dependent GBM, where we denote $N(\cdot)$ as the cumulative distribution function of standard normal distribution.
1-dimensional Black Scholes formula: The closed form solution of European call option is:

$$u\left(t, S_t; (r, \sigma, \kappa)\right) = S_t N\left(d_1\right) - \kappa S_0 e^{-\tilde{r}(T-t)} N\left(d_2\right),$$

where

$$d_1 = \frac{\log\left(\frac{S_t}{\kappa S_0}\right) + \left(\tilde{r} + \frac{\tilde{\sigma}^2}{2}\right)(T-t)}{\tilde{\sigma}\sqrt{T-t}}, \quad d_2 = d_1 - \sigma\sqrt{T-t}.$$

The spot rate and volatility terms under time dependent GBM is given as:

$$\tilde{r} = \frac{1}{T-t}\int_t^T r(s)ds, \quad \tilde{\sigma}^2 = \frac{1}{T-t}\int_t^T \sigma^2(s)ds.$$

1-dimensional float lookback option: Follow the expression of the $\tilde{r}$ and $\tilde{\sigma}^2$, we give the formula of float strike minimum look back option and geometric Asian call option.

$$u(t, S_t, m_t; (r, \sigma)) = S_t N(a_1) - m_t e^{-\tilde{r}(T-t)} N(a_2) - S_t \frac{\tilde{\sigma}^2}{2\tilde{r}} \left(N(-a_1) - e^{-\tilde{r}(T-t)} \left(\frac{m_t}{S_t}\right)^{2\tilde{r}/\tilde{\sigma}^2} N(-a_3) \right).$$

where

$$a_1 = \frac{\log(S_t/m_t) + \left(\tilde{r} + \tilde{\sigma}^2/2\right)(T-t)}{\tilde{\sigma}\sqrt{T-t}}, \quad a_2 = a_1 - \tilde{\sigma}\sqrt{T-t}, \quad a_3 = a_1 - \frac{2\tilde{r}}{\tilde{\sigma}}\sqrt{T-t}.$$

1-dimensional geometric Asian call option formula: The corresponded time dependent Asian formula under time dependent GBM is give as:

$$u(t, S_t, A_t; (r, \sigma, \kappa)) = e^{-\tilde{r}(T-t)} \left(A_t^{t/T} S_t^{1-t/T} e^{\hat{\mu} + \hat{\sigma}^2/2} N(d_1) - \kappa S_0 N(d_2) \right),$$

where

$$A_t = \exp\left\{ \frac{1}{t} \int_0^t \log S_s ds \right\}, \quad \hat{\mu} = \left(\tilde{r} - \frac{\tilde{\sigma}^2}{2}\right) \frac{1}{2T}(T-t)^2, \quad \hat{\sigma} = \frac{\tilde{\sigma}}{T}\sqrt{\frac{1}{3}(T-t)^3},$$
$$d_2 = \frac{(t/T)\log A_t + (1 - t/T)\log S_t + \hat{\mu} - \log K}{\tilde{\sigma}}, \quad d_1 = d_2 + \hat{\sigma}.$$

Multi-dimensional geometric basket call option formula: Then we show the analytical solution of geometric basket call:

$$u(t, S_t; (r, \sigma, \rho_S, \kappa)) = F N(d_1) - \kappa \bar{S}_0 e^{-\tilde{r}(T-t)} N(d_2).$$

where $\bar{S}_0 = \left(\prod_{i=1}^d S_0^i\right)^{\frac{1}{d}}$ denotes the geometric average of the initial price.

$$d_+ = \frac{\ln \frac{F}{\kappa \bar{S}_0} + \frac{1}{2}\tilde{\sigma}^2(T-t)}{\tilde{\sigma}\sqrt{(T-t)}}, \quad d_- = d_+ - \tilde{\sigma}\sqrt{(T-t)}, F = \bar{S}_0 \left(\left(\tilde{r} - \frac{(\sigma^2 - \tilde{\sigma}^2)}{2} \right)(T-t) \right)$$
$$\sigma^2 = \frac{1}{n}\sum_{i=1}^n \frac{1}{T-t}\int_t^T \sigma_i^2(s) ds, \quad \tilde{\sigma}^2 = \frac{1}{n^2}\sum_{i,j=0}^n \rho_{ij} \frac{1}{T-t}\int_t^T \sigma_i(s)\sigma_j(s) ds$$

under time dependent GBM, the spot rate $\tilde{r}$ is same as what in one dimensional cases

References

Abadi, M., A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

Bates, D. S. (1996). Jumps and stochastic volatility: Exchange rate processes implicit in deutsche mark options. *The Review of Financial Studies 9*(1), 69–107.

Bayraktar, E., Q. Feng, and Z. Zhang (2022). Deep Signature Algorithm for Path-dependent American Option Pricing. *arXiv preprint arXiv:2211.11691*.

Beck, C., S. Becker, P. Cheridito, A. Jentzen, and A. Neufeld (2021). Deep Splitting Method for Parabolic PDEs. *SIAM Journal on Scientific Computing 43*(5), A3135–A3154.

Berner, J., M. Dablander, and P. Grohs (2020). Numerically solving parametric families of high-dimensional Kolmogorov partial differential equations via deep learning. *Advances in Neural Information Processing Systems 33*, 16615–16627.

Chen, T. and H. Chen (1995). Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems. *IEEE Transactions on Neural Networks 6*(4), 911–917.

Duffy, D. J. (2013). *Finite Difference Methods in Financial Engineering: a Partial Differential Equation Approach.* John Wiley & Sons.

Feng, Q., M. Luo, and Z. Zhang (2021). Deep Signature FBSDE Algorithm. *arXiv preprint arXiv:2108.10504*.

Gao, C., S. Gao, R. Hu, and Z. Zhu (2022). Convergence of the Backward Deep bsde Method with Applications to Optimal Stopping Problems. *arXiv preprint arXiv:2210.04118*.

Germain, M., M. Laurière, H. Pham, and X. Warin (2022). DeepSets and Their Derivative Networks for Solving Symmetric PDEs. *Journal of Scientific Computing 91*(2), 63.

Glasserman, P. (2004). *Monte Carlo Methods in Financial Engineering*, Volume 53. Springer.

Glau, K. and L. Wunderlich (2022). The Deep Parametric PDE Method and Applications to Option Pricing. *Applied Mathematics and Computation 432*, 127355.

Glorot, X. and Y. Bengio (2010). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 249–256. JMLR Workshop and Conference Proceedings.

Gnoatto, A., M. Patacca, and A. Picarelli (2022). A Deep Solver for BSDEs with Jumps. *arXiv preprint arXiv:2211.04349*.

Gnoatto, A., A. Picarelli, and C. Reisinger (2023). Deep xva Solver: A Neural Network-Based Counterparty Credit Risk Management Framework. *SIAM Journal on Financial Mathematics 14*(1), 314–352.

Gu, Y., H. Yang, and C. Zhou (2021). Selectnet: Self-paced learning for high-dimensional partial differential equations. *Journal of Computational Physics 441*, 110444.

Han, J., A. Jentzen, et al. (2017). Deep Learning-based Numerical Methods for High-dimensional Parabolic Partial Differential Equations and Backward Stochastic Differential Equations. *Communications in Mathematics and Statistics 5*(4), 349–380.

Han, J., A. Jentzen, and W. E (2018). Solving high-dimensional partial differential equations using deep learning. *Proceedings of the National Academy of Sciences 115*(34), 8505–8510.

Heston, S. L. (1993). A closed-form solution for options with stochastic volatility with applications to bond and currency options. *The Review of Financial Studies 6*(2), 327–343.

Hilber, N., O. Reichmann, C. Schwab, and C. Winter (2013). *Computational Methods for Quantitative Finance: Finite Element Methods for Derivative Pricing.* Springer Science & Business Media.

Huré, C., H. Pham, and X. Warin (2020). Deep Backward Schemes for High-dimensional Nonlinear PDEs. *Mathematics of Computation 89*(324), 1547–1579.

Ioffe, S. and C. Szegedy (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *International Conference on Machine Learning*, pp. 448–456. pmlr.

Jacquier, A. J. and M. Oumgari (2019). Deep PPDEs for rough local stochastic volatility. *Available at SSRN 3400035*.

Kingma, D. P. and J. Ba (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Lapeyre, B. and J. Lelong (2021). Neural network regression for Bermudan option pricing. *Monte Carlo Methods and Applications 27*(3), 227–247.

Liang, J., Z. Xu, and P. Li (2021). Deep learning-based least squares forward-backward stochastic differential equation solver for high-dimensional derivative pricing. *Quantitative Finance 21*(8), 1309–1323.

Longstaff, F. A. and E. S. Schwartz (2001). Valuing American options by simulation: a simple least-squares approach. *The Review of Financial Studies 14*(1), 113–147.

Lu, L., P. Jin, G. Pang, Z. Zhang, and G. E. Karniadakis (2021). Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators. *Nature Machine Intelligence 3*(3), 218–229.

Mijatović, A. and M. Pistorius (2013). Continuously monitored barrier options under Markov processes. *Mathematical Finance 23*(1), 1–38.

Sabate-Vidales, M., D. Šiška, and L. Szpruch (2020). Solving path dependent PDEs with LSTM networks and path signatures. *arXiv preprint arXiv:2011.10630*.

Saporito, Y. F. and Z. Zhang (2020). PDGM: a neural network approach to solve path-dependent partial differential equations. *arXiv preprint arXiv:2003.02035*.

Sirignano, J. and K. Spiliopoulos (2018). DGM: A deep learning algorithm for solving partial differential equations. *Journal of Computational Physics 375*, 1339–1364.

Tan, L. and L. Chen (2022). Enhanced DeepONet for modeling partial differential operators considering multiple input functions. *arXiv preprint arXiv:2202.08942*.

Wang, H., H. Chen, A. Sudjianto, R. Liu, and Q. Shen (2018). Deep learning-based BSDE solver for LIBOR market model with application to Bermudan swaption pricing and hedging. *arXiv preprint arXiv:1807.06622*.

Yamakami, T. and Y. Takeuchi (2022). Pricing Bermudan swaption under two factor Hull-White model with fast Gauss transform. *arXiv preprint arXiv:2212.08250*.

Zhang, G. and L. Li (2019). Analysis of Markov chain approximation for option pricing and hedging: Grid design and convergence behavior. *Operations Research 67*(2), 407–427.