Robust Operator Learning to Solve PDE

Carl REMLINGER[1,2,3], Joseph MIKAEL[2,3], and Romuald ELIE[1]

[1]Université Gustave Eiffel
[2]EDF Lab
[3]FiME Lab

January 10, 2022

Abstract

A model solving a family of partial differential equations (PDEs) with a single training is proposed. Re-calibrating a risk factor model or re-training a solver every time the market conditions change is costly and unsatisfactory. We therefore want to solve PDEs when the environment is not stationary or for several initial conditions at the same time. By learning operators in a single training, we ensure of the robustness of optimal controls with variations of the models, options or constraints. But, ultimately, we want to generalize by solving the PDE with models or conditions that were not present during training. We confirm the effectiveness of the method with several risk management problems by comparing it with other machine learning approaches. We evaluate our DeepOHedger on option pricing tasks, including local volatility models and option spreads involved in energy markets. Finally, we present a purely data-driven approach to risk hedging, from time series generation to learning optimal controls. Our model then solves a family of parametric PDE from synthetic samples produced by a deep generator previously trained on spot price data from different countries.

1 Introduction

Stochastic control relying on Monte Carlo simulations requires underlying time series modeling assumptions. In practice, there is often uncertainty on the environment. Financial markets, for example, are constantly changing and models can therefore quickly be invalidated. In the energy field, electricity market prices are highly dependent on the weather. The rapid regime changes imposed by unusual weather conditions (cold winters, droughts, windy days influencing renewable production thus prices) require the use of more flexible and robust models for risk management in the energy sector.

Unlike classical models Black and Scholes (1973); Heston (1993); Dupire et al. (1994), an optimal model must involve correlations changes or jumps.

Integrating these phenomena into classical models would make them difficult to calibrate and therefore unstable. Moreover, using miscalibrated models may lead to erratic derivatives pricing. Properly calibrate stochastic models has to be done cautiously and ideally continuously. However, due to the related computational cost and to the need for stability, it is not the always the case.

Moreover, one popular numerical method for risk hedging relies on PDE resolution but struggle in high-dimension and lack flexibility. To answer this issue, some new methods have been developed using machine learning. Deep BSDE approaches is one of the first attempt to tackle the curse of dimensionality Weinan et al. (2017); Han et al. (2018) The models are trained to find the optimal controls when fed with asset prices for instance Weinan et al. (2017); Han et al. (2018). Moreover, these proposals allow specific objective functions and thus can take into account stylized facts Cont (2001) such as the gain-loss asymmetry or classical constraints (illiquidity for instance) FECAMP et al. (2020). However, these deep hedgers are trained for a given risk factor model which, if not accurate, could lead to add approximation errors.

Recent papers propose to use deep parametric PDE methods to learn the solutions of a whole family of equations depending on fixed set of parameters Glau and Wunderlich (2020); Khoo et al. (2021). Instead of conditioning the solution with real-valued parameters, we learn operators associated with the solutions of PDEs, i.e. a mapping from set of parameter functions with the set of solutions.

We therefore propose to solve a stochastic control problem for multiple models in a single training, but which generalizes to unknown models during training. The parameter functions can describe the underlying risk model (in particular its volatility function), characteristics of an option, or constraints. In order to correctly capture the mapping between the functions and the solution, we consider deep operator networks (DeepONets) Lu et al. (2019) in an unsupervised approach. The model architecture consists of two subnetworks, one that learns an approximate input function on a fixed number of sensors and another to represent the state of the underlying process at a given time.

The application focuses on risk hedging. The introduced Deep Operator Hedger (DeepOHedger) is an universal solver for a set of risk factor models or derivative objectives. The parameter functions can describe the underlying risk model (in particular its volatility function), characteristics of an option, or constraints.

Contributions

- We introduce a model based on continuous operator approximation being able to solve nonlinear parametric PDE.
- The model solves a family of parametric PDEs with one single neural network by learning several operators.
- Numerical applications on risk hedging using neural networks are proposed, our DeepOHedger competes with classical machine learning based methods on diverse underlying models.

- A purely data-driven method for risk management is proposed, from time series generation to optimal control learning.

2 Related Works

Most of the current literature in stochastic control addresses the high-dimensional challenge. Due to the inefficiency of classical Monte Carlo methods or PDE approaches in high dimensions, the use of deep neural networks expends over the last few years. In finance, supervised learning approaches were first considered to price derivatives Malliaris and Salchenberger (1993); Hutchinson et al. (1994). Recent literature reviews of neural networks for option pricing and hedging can be found in Ruf and Wang (2020). Although wide research efforts focusing on high-dimension with neural networks, only few methods are available to solve fully nonlinear equations. A first numerical method for solving high dimensional fully nonlinear PDE were introduced Cheridito et al. (2007), then extended with effective schemes developed Fahim et al. (2011); Tan (2013). Despite encouraging results, these approaches could not solve PDEs in dimension greater than 5. To answer this issue, Warin (2018) proposed to design a specific scheme based on nesting Monte Carlo allowing to consider very high-dimension.

Global approach methods, minimizing a terminal objective function to solve fully nonlinear PDEs, were also introduced Beck et al. (2019). The method relies on a second order refinement of the backward stochastic differential equation (BSDE) representation of Cheridito et al. (2007). Other proposals use BSDE for semilinear PDEs Henry-Labordere et al. (2019); Chan-Wai-Nam et al. (2019) or consider to estimate simultaneously the solution and its gradient with deep neural networks Huré et al. (2020). Real fully nonlinear cases were then proposed Weinan et al. (2017); Han et al. (2018) highlighting how machine learning models hold promise for solving large-dimensional PDEs.

A specific related deterministic method is the deep Galerkin approach Sirignano and Spiliopoulos (2018). Solutions are evaluated by automatic differentiation of the network function approximating the solution of the PDE. The adaptability to a large range of PDEs with or without boundary conditions makes this method particularly appealing. First experimented in discrete case Barucci et al. (1997); Meade Jr and Fernandez (1994) , deep Galerkin solvers were extended in continuous time Al-Aradi et al. (2018, 2019); Li et al. (2020). Deep parametric PDE method using the deep Galerkin were also proposed Sirignano and Spiliopoulos (2018); Khoo et al. (2021); Glau and Wunderlich (2020).

Some papers try to solve risk hedging problems in incomplete markets, where illiquidity and transactions costs makes classical approaches inoperative FECAMP et al. (2020). A complete review of neural networks-based algorithms for stochastic control and PDEs in finance is available, presenting several use cases Germain et al. (2021).

Recent proposals focus on robust stochastic control and aim at solving PDEs with uncertainty on the underlying model using neural networks. The uncertainty can be restricted to generalized affine diffusions Lütkebohmert et al. (2021),

but limits the uncertainty to a real-valued parameter estimation in a linear setting. The physical approach of Khoo et al. (2021) lies on solving deterministic parametric PDEs with neural networks from some physical quantities initially randomly chosen. The considered parametric PDE are linked to elliptic homogenisation and nonlinear Schrödinger eigenvalue problem. A recent paper Glau and Wunderlich (2020) introduce a general deep parametric method being able to solve a family of PDEs with one single neural network. To approximate the solution, the network is conditioned with constant values describing the risk factor model or the option. The loss function relies on the least-squares formulation of a PDE as done in Deep Galerkin Sirignano and Spiliopoulos (2018). This deep solver allows wide applications and is tested on multi-asset option pricing, but the numerical results is restricted to the Black-Scholes model Black and Scholes (1973).

3 Background

We introduce below some background on stochastic control and DeepONets.

3.1 Stochastic Control

Let T be a fixed time horizon and d be an integer describing the dimension. Let be $X = (X_t)_{t\in[0,T]}$ a continuous-time controlled diffusion dynamics whose realisations belong to $\mathbb{R}^d$

$$dX_t = \mu(X_t, \alpha_t)dt + \sigma(X_t, \alpha_t)dW_t, \tag{1}$$

where W is a d-dimensional standard Brownian motion on some probability space $(\Omega, \mathcal{F}, \mathbb{P})$ equipped with a filtration $(\mathcal{F}_t)_{t\in[0,T]}$ representing the information available at time t, and X_0 a $\mathcal{F}_0$-measurable random variable valued in $\mathbb{R}^d$. At each time step t, X_t, α_t designate respectively the state of an agent and the control performed by the agent. The functions $\mu : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^d$ describes the drift and $\sigma : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{M}^d$ (the set of $d \times d$ matrices) the volatility. These two functions also depend on time at the expense of heavier notations. Both parameters μ and σ are supposed to satisfy the usual Lipschitz conditions Ikeda and Watanabe (2014) ensuring existence and uniqueness of the solution of Eq.1. The control $\alpha = (\alpha_t)_{t\in[0,T]}$ is a $\mathcal{F}_t$-adapted process on the set $\mathcal{A}$. We seek for an optimal strategy, minimizing the following cost functional over control process α:

$$J(\alpha) = \mathbb{E}\left[\int_0^T f(X_t, \alpha_t)dt + g(X_T))\right],$$

where f is a nonlinear running cost function function defined on $[0, T] \times \mathbb{R}^d \times \mathbb{R}^d$, and g is a terminal function defined on $\mathbb{R}^d$, called payoff. The admissible set of controls $\mathcal{A}$ is the set of controls α which satisfy the usual integrability conditions ensuring that the cost function $J(\alpha)$ is well-defined and finite.

Following Pham (2009), the dynamic programming Bellman equation leads to the partial differential equation:

$$\begin{cases} \partial_t v + H(x, D_x v, D_x^2 v) = 0 & \text{on } [0,T) \times \mathbb{R}^d \\ v(T, .; .) = g & \text{on } \mathbb{R}^d \end{cases} \tag{2}$$

where $H(x,y,z) = \inf_{a \in \mathcal{A}} \left[\mu(x,a)y + \frac{1}{2}\text{Tr}(\sigma\sigma^T(x,a)z) + f(x,a) \right]$ is the so-called Hamiltonian function.

3.2 DeepONets

Deep Operator Network Lu et al. (2021) is a specific network architecture to approximate operators, that is a mapping from a space of functions into another space of functions. Following Lu et al. (2019), we denote an operator G taking a parameter function $u \in \mathcal{U}$ as input and returning a function $G(u)$. We evaluate $G(u)$ on a vector $x \in \mathbb{R}^d$ such that the output $G(u)(x)$ is a real number. The Universal Approximation Theorem for Operator Chen and Chen (1995) indicates that neural networks can learn faithfully any nonlinear continuous operators from data.

Theorem 3.1. ***(Universal Approximation Theorem for Operator)****. Suppose that σ is a continuous non-polynomial function, X is a Banach Space, $K_1 \subset X, K_2 \subset \mathbb{R}^d$ are two compact sets in X and $\mathbb{R}^d$, respectively, V is a compact set in $\mathcal{C}(K_1)$, G is a nonlinear continuous operator, which maps V into $\mathcal{C}(K_2)$. $\mathcal{C}(K)$ indicates Banach space of all continuous functions defined on compact set K with norm $||f||_{\mathcal{C}(K)} = \max_{x \in K} |f(x)|$. Then for any $\varepsilon > 0$, there are positive integers n, p, q, constants $c_i^k, \xi_{ij}^k, \theta_i^k, \kappa_k \in \mathbb{R}, w_k \in \mathbb{R}^d, x_j \in K_1, i = 1, ..., n, k = 1, ..., p, j = 1, ..., q$, such that*

$$\left| G(u)(x) - \sum_{k=1}^{p} \sum_{i=1}^{n} c_i^k \sigma \left(\sum_{j=1}^{q} \xi_{ij}^k u(x_j) + \theta_i^k \right) \sigma(w_k.x + \kappa_k) \right| < \varepsilon$$

holds for all $u \in V$ and $x \in K_2$.

DeepOnets consist in two sub-networks to handle the inputs of the function $u \in \mathcal{U}$ and the vector $x \in \mathbb{R}^d$ independently. The first one is called the branch network and aims at approximating the function u on a given number of scattered values $x_1, ..., x_q$ called *sensors*. The branch network $b : \mathbb{R}^q \to \mathbb{R}^K$ is fed with $[u(x_1), \ldots, u(x_q)]^T$ of the sensors and outputs a vector $[b_1, \ldots, b_K]^T \in \mathbb{R}^K$. The second one, called trunk network, is a function $a : \mathbb{R}^d \to \mathbb{R}^K$ which takes as input the vector x and outputs $[a_1, ..., a_K]^T \in \mathbb{R}^K$. A scheme of the DeepOnet is proposed Figure 1. Both outputs are then concatenated as follows:

$$G(u)(x) \approx \sum_{k=1}^{K} b_k \left(u(x_1), \ldots, u(x_q) \right) a_k(x) + b_0$$

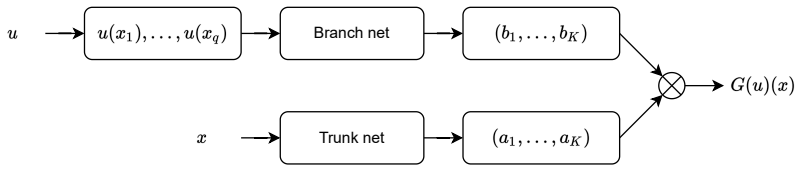


Figure 1: Scheme of the (unstacked) DeepOnet model.
The function u is approximated by q sensors and given as input to the branch net which outputs a real valued vector of dimension K. The trunk net takes as input $x \in \mathbb{R}^d$ and outputs a real valued vector of dimension K. Both outputs are then concatenated to learn the operator $G : u \to G(u)$. This architecture is known as the unstacked DeepONet.

where $b_0 \in \mathbb{R}$ is a variable to be learned. b_0 represents a bias and may increase the performance by reducing the generalization error. Equation (3) thus boils down to a scalar product.

4 Problem Formulation

We aim to solve numerically several PDE involved in the pricing of financial derivatives with a single model and a single training. To do so, a robust model learning operators is considered.

4.1 Methodology

We want to solve the optimisation problem of (2) when the environment is not stationary or for several initial conditions at once. Re-calibrate a risk factor model or re-train a deep hedger each time the market conditions change is costly and unsatisfactory.

For this purpose, we consider deep operator networks to learn the general solution associated with several risk factor models simultaneously. The basic idea of learning operator, that boils down to learn simultaneously the optimal controls and the parameter function, is justified by the desire to solve a family of parametric PDEs with a single training. We design a robust model solving (2) for a set of given underlying models, but, ultimately, we want to generalize by solving the PDE with models or conditions that were not present during training.

By learning the optimal controls α according to a given parameter function u, we also seek to better generalize compared with learning a unique parameter (as done in Glau and Wunderlich (2020)). The parameter function can describe the model parameters, such as the volatility function $\sigma(.)$ of the price model (in Eq.(1)), option parameters (payoff function g, strike, correlation) or constraints. When considering the Itô process terms, μ and σ stay Lipschitz in $x \in \mathbb{R}^d$,

uniformly in $t \in [0, T]$, ensuring the existence of a solution. Typically, uncertainty on volatility is a bound set among the admissible volatility functions.

4.2 Operator Formulation

We consider DeepONets as the architecture is suitable to learn both the parameter function u and the controls α.

Learning with one function The parameter function (i.e. input function) is denoted $u \in \mathcal{U}$, and $v \in \mathcal{V}$ is the corresponding unknown solution of the PDE 2. The problem is solved for a given u with machine learning methods reviewed in Germain et al. (2021). The question is whether one can find a solution $v = v(u) \in \mathcal{V}$ of the PDE 2 for any $u \in \mathcal{U}$ (subject to appropriate initial and boundary conditions). Then, we can define the solution operator $G : \mathcal{U} \to \mathcal{V}$ as

$$G(u) = v(u)$$

The approximation of the function u is decisive in the identification of the underlying dynamics or the option. Thus, the number of sensors $\{x_1, \ldots, x_q\}$ has to be cautiously chosen. We consider a regular grid of $q \in \mathbb{N}$ points of $\mathbb{R}^d$. On each dimension, we construct a fixed interval $[a, b] \subset \mathbb{R}, (a, b) \in \mathbb{R}^2$ where the sensors are defined as $x_j^{(.)} = a + j(b - a)q$ for $j \in \{1, \ldots, q\}$.

Generalizing with a family of functions The introduced method requires to provide an approximation of the function u. By giving a misleading function we could add a model error to the approximation error. As our first objective is being able to solve in a robust manner, we train our model not on one u but on a family of functions. The branch net has to learn various functions u, thus we really take advantage of using neural networks, tackling a high-dimensional problem and functions possibly difficult to estimate.

In the numerical section, we propose to learn a set of volatility functions (linear or not) and data-driven estimations. When uncertain on the market structure, one can provide an approximation of the volatility functions, our model has to be able to properly solve the PDE, whether branch net was trained with the function or not.

4.3 A Global Approach for Deep Hedging

A hedging strategy can be reduced to the stochastic control problem defined in (3.1), by buying or selling some discounted asset at each date. We consider risky assets with price process S and a derivative asset with underlying S and maturity T. In the following, we focus on European call options or option spreads. We are trying to define a self-financing portfolio strategy based on asset S such that wealth at maturity T replicates the option payoff. In case of incomplete market, a perfect replication is not reachable, thus we propose to train an agent by minimizing the residual hedging error. A numerical scheme

for the approximation of the equation (2) is used and the solution is estimated from samples of S.

The samples are supposed to be drawn from a discretization of the continuous time process S, starting from S_0 valued in $\mathbb{R}^d$ and observed on a time grid $\mathcal{T} = \{0 = t_0 < t_1 < ... < t_N = T\}$ for $N > 1$. For the sake of simplicity, in the following, we assume a regular time grid with mesh size $\Delta t = t_{i+1} - t_i$. The discrete model $(S_{t_i})_{i\in\{1,...,N\}}$ is obtained by the following Euler scheme for $i \in \{0, \dots, N-1\}$:

$$\begin{cases} S_{t_i+\Delta t} = S_{t_i} + \mu(t_i, S_{t_i})\Delta t + \sigma(t_i, S_{t_i})\Delta W_{t_i}, \\ S_{t_0} = s_0 \in \mathbb{R}^d \end{cases} \tag{3}$$

where $\Delta W_{t_i} = W_{t_{i_1}} - W_{t_i}$ is a collection of i.i.d. $\mathcal{N}(0, \Delta t I_d)$ random variables.

The learning hedger agent consists in a single model parametrized by θ and trained to find the premium p and the optimal control α_{t_i} at each decision step $t_i \in \mathcal{T}$. The model is fed successively with the time step t_i, the previous state X_{t_i} and the parameter function u to provide a control $\alpha_\theta(t_i, S_{t_i}; u)$ for $i \in \{0, \dots, N-1\}$. The self-financing portfolio of terminal value at time $t_N = T$ is denoted X_T and defined as:

$$X_{T,\theta;u} = p_\theta + \sum_{i=0}^{N-1} \alpha_\theta(t_i, S_{t_i}; u)(S_{t_{i+1}} - S_{t_i}).$$

We give ourselves the opportunity to re-balance the portfolio every time steps. There is no control at the last date.
The global approach for solving (2) leads to the following optimization problem:

$$\min_\theta \mathbb{E}[\ell(X_{T,\theta;u}, g(S_T; u))].$$

In the following, the quadratic loss is considered for $\ell(x, y) = (x - y)^2$. Other objective functions could be considered such as the asymmetrical loss Simon (1972) to take into account stylized facts Cont (2001). The notation $g(X_T; u)$ allows learning the optimal solution for different payoffs.

This approach is very convenient due to its formulation and the simplicity to optimize the parameter θ with machine learning. First introduced by Gobet and Munos (2005), the method consisted in approximating the feedback control at any time, i.e. a function of the state process. Then, the method was extended with deep neural networks Han et al. (2016) or more recently with delay considerations Han and Hu (2021). However, these methods necessitate a solver at each time step, complicating the convergence (the set of parameters being huge) and making difficult to consider a large time horizon. A way to address these issues is to consider a single neural network fed with time step and controls, as we do, and thus gain in stability Fecamp et al. (2020). Other methods are detailed in the literature, including local approaches Germain et al. (2021).

5 Numerical Results

We apply our model on a risk hedging task. We show that DeepOHedger provides better performance compared to several variant of deep hedgers and demonstrates ability to reduce the generalization error even in the easiest linear problem.

Only feed-forward neural networks are considered of 3 layers of 16 neurons each, and use the Adam optimizer with learning rate 0.001. The number of iterations is 10.000 guaranteeing convergence of the training. New samples are generated each iteration from model-based Monte Carlo simulations or deep generative models (in Section 5.4).

5.1 Benchmarks

Benchmarks consist in three variant of deep hedgers solving the problem of (2) with a global approach 4.3. Each model learns to minimise the replication loss at the terminal value T (Eq.4), but differs according to the conditional information given to them during the training process.
The first model, called Misspecified Deep Hedger (MDeepHedger), is trained on a (reasonably) misspecified model, close to the unobserved real model. No information on the model is given to the neural network. The purpose is to quantify the error of using a unique model resulting of a erratic calibration.
Robust Deep Hedger (RDeepHedger) shares the same design as MDeepHedger but is fed with samples of several price models, omitting the real model. Here again, no addition information is provided, thus the robust hedger has to learn the mapping between price realizations and the corresponding model. Conditional Deep Hedger (CDeepHedger) is a Robust Deep Hedger conditioned with the parameter value corresponding to each price model. The parameter boils down to a real value, characterizing for instance the constant σ in the volatility term of a Black-Scholes. CDeepHedger emphasis the need of learning operator $G(u)$ instead of conditioning with additional parameters.

5.2 Robust Hedging With Volatility Function

We apply our method on price European call options. The solver is given approximations of $J \in \mathbb{N}$ volatility functions $\sigma^j(.)$ as well as underlying $S^j = (S_t^j)_{t\in\mathcal{T}}$. The underlying follows the Itô dynamics:

$$dS_t^j = \mu(t, S_t^j)dt + \sigma^j(t, S_t^j)dW_t, \quad \text{for } j \in \{1, ..., J\},$$

where $\mu : [0,T] \times \mathbb{R}^d \to \mathbb{R}^d$, $\sigma^j : [0,T] \times \mathbb{R}^d \to \mathbb{M}_d$ and W is a Brownian motion. Here, $u^j : x \to \sigma^j(x)$ describes the volatility function of the model j, for $j \in \{1, \dots, J\}$. The number of dates is $N = 30$ and there is no control on the last date. We consider European call options, which can be exercised only at the maturity T and at an exercise price K (called strike), and which payoff is given by:

$$g(S_T) = (S_T - K)_+.$$

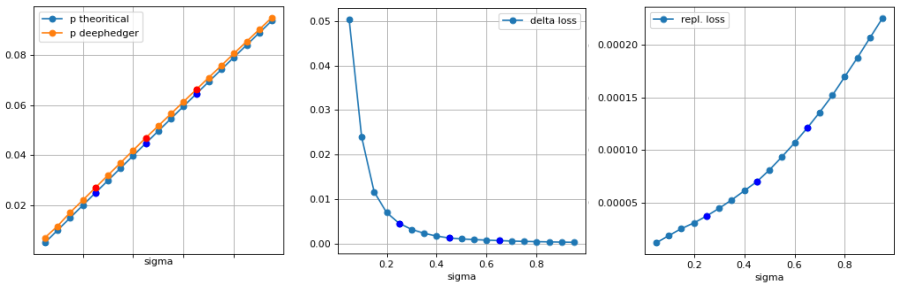


Figure 2: Premium, control and replication losses depending on the volatility value of a Black-Scholes model.

Linear volatility functions The price of the underlying asset follows the Black-Scholes dynamics:

$$dS_t = rS_t dt + \sigma S_t dW_t, \tag{4}$$

where $(r, \sigma) \in \mathbb{R} \times \mathbb{R}_+$ and W is a 1-dimensional Brownian motion. This model benefits of well-known closed formulas for the premium p^* and the optimal controls α^*, allowing us to properly evaluate the accuracy of our approach. The volatility uncertainty is $\sigma^j(x) = \{\sigma x : \sigma \in [\underline{\sigma}^j, \bar{\sigma}^j]\}$, where $\underline{\sigma}^j, \bar{\sigma}^j \in \mathbb{R}_+$. Then, the function u describes the (linear) volatility function $\sigma(t, X_t)$ terms in the process S, then $u : x \to \sigma x$. We approximate u by $q = 10$ sensors on a fixed regular grid $(x_i)_{i \in \{1..q\}}$ valued in $\mathbb{R}$. The drift constant is set at $r = 0$, the maturity is 0.25, the strike is set at $K = 0.5$ as well as the spot $X_0 = 0.5$.

We consider 19 price models varying according to the value of the constant $\sigma \in \{0.05, 0.10, ..., 1.00\}$. The deep hedgers are fed with 16 models during the training process. We omit the values $\sigma \in \{0.25, 0.45, 0.65\}$ in order to evaluate the ability of the model the generalize on unknown parameters.

The outputs of the deep hedgers are compared with theoretical values of the underlying Black-Scholes models. Each loss is the mean squared error (MSE) between theoretical values and deep hedger's outputs. The premium accuracy is given by *p loss*, the optimal controls by *alpha loss* and the replication error by *repl. loss*. DeepOHedger indicates our method, MDeepHedger is only trained on a misspecified model ($\sigma = 0.30$ instead of 0.25), RDeepHedger is fed with the time step and current state from all 16 models, and CDeepHedger is conditioned by the volatility constant of each model thus fed with $x = (t_i, x^j_{t_i}, \sigma^j)$.

Table 1 reports the average performances of the DeepOHedger and the benchmarks on the 16 training models and on the real (unobserved) price model. On average, our model is better able to retrieve the premium than the other models. In particular, learning the volatility function as well as the optimal controls seems beneficial, as CDeepHedger provides a less accurate premium. The two other models return poor results because only one premium value is learned in the MDeepHedger case, and RDeepHedger proposes a unique average

	Training models			Unobserved model ($\sigma = 0.25$)		
	p loss	alpha loss	repl. loss	p loss	alpha loss	repl. loss
DeepOHedger	**6.99e-07**	3.54e-03	**7.20e-05**	**4.29e-07**	1.44e-03	**1.93e-05**
MDeepHedger	2.61e-03	1.35e-02	2.78e-03	2.23e-05	1.30e-03	4.12e-05
RDeepHedger	2.45e-03	1.42e-02	2.62e-03	4.38e-03	2.18e-02	4.55e-03
CDeepHedger	2.69e-06	**1.82e-03**	7.30e-05	4.73e-06	**7.07e-04**	2.31e-05

Table 1: Average losses for premium (p), optimal controls (alpha) and replication (repl.) on the 16 training models and the unobserved model of $\sigma = 0.25$.

premium for all the 16 price models, being unable to distinguish between them. MDeepHedger, which is trained on a $\sigma = 0.30$ samples, provides better results on the real unobserved model with $\sigma = 0.25$ than on training set. Concerning the replication loss, again our model outperforms the others, in a lesser extent for CDeepHedger. The errors are particularly low, which could be close to the discretization approximation. A misleading calibration has a significant impact on the quality of the hedging. The optimal controls however seem to be better learned with the value-conditional deep hedger, on both training and unobserved models. Further analysis could be beneficial to explain this phenomena.

Figure 2 illustrates the losses of DeepOHedger according to each price models. The premium is uniformly retrieved for each volatility constant in the Black-Scholes model. DeepOHedger generalises well when being fed we unknown models, indicated with values at $\sigma \in \{0.25, 0.45, 0.65\}$. The gap between our method and optimal controls decreases when the volatility increases. On the opposite, the replication error increases when the volatility is high, which is an expected behavior: replicating an option when the market is unstable is harder. An illustration of the optimal controls with the ones proposed by our method on unobserved models is illustrated in Figure 4 (in Appendix).

Non linear volatility functions We now consider nonlinear functions u. The price of the underlying process follows the dynamics of a constant elasticity of variance (CEV) model Cox (1996):

$$dS_t = rS_t dt + \sigma S_t^p dW_t \tag{5}$$

where $(r, \sigma, p) \in \mathbb{R} \times \mathbb{R}_+ \times \mathbb{R}$ and W is a 1-dimensional Brownian motion.

In this CEV case, $\sigma^j(x) = \{\sigma x^p : \sigma \in [\underline{\sigma}^j, \bar{\sigma}^j], p \in [\underline{p}^j, \bar{p}^j]\}$, where $\underline{\sigma}^j, \bar{\sigma}^j \in \mathbb{R}_+, \underline{p}^j, \bar{p}^j \in [\frac{1}{2}, 1]$. The function $u : x \to \sigma x^p$ with $p = 0.7$ describes the volatility function characterized by the parameter (σ, p). As done in the linear case, we approximate u by $q = 10$ sensors on a fixed regular grid $(x_i)_{i \in \{1..q\}}$ valued in $\mathbb{R}$. The drift is also set at $r = 0$, the maturity is 0.25, the strike and the spot are set at $K = X_0 = 0.5$. We consider 14 price models varying according to the values $\sigma \in \{0.1, 0.10, ..., 0.80\}$. We train the deep hedgers only with 11 models, omitting the parameters $\sigma \in \{0.25, 0.45, 0.65\}$.

Table 2 reports the performances of each deep hedger on training models and the unobserved real model ($\sigma = 0.25, p = 0.7$). The theoretical values of the

	Average on training models		Real model (unobserved) $\sigma = 0.25, p = 0.7$	
	p loss	repl. loss	p loss	repl. loss
DeepOHedger	6.91e-07	**8.48e-05**	**6.14e-08**	**3.63e-05**
MDeepHedger	8.48e-04	1.15e-03	2.59e-07	3.70e-05
RDeepHedger	2.49e-03	2.68e-03	3.43e-03	3.62e-03
CDeepHedger	**3.77e-07**	9.14e-05	7.36e-08	4.75e-05

Table 2: Price and replication loss of deep hedger on CEV models.

premium for the CEV model is computed as done in Davydov and Linetsky (2001). DeepOHedger outperforms significantly every benchmarks, closely followed by CDeepHedger. When fed with true constants of volatility σ and the order p, CDeepHedger provides less effective hedging than the DeepOHedger, emphasis the need of estimating functions instead of vectors. The cost of mis-calibrating a model, described with the performances of MDeepHedger ($\sigma = 0.30, p = 0.7$)

5.3 Hedging Option Spread for Energy Markets

We propose to hedge an option spread for several underlying models with a single training. An electricity provider may have to buy gas to produce electricity. Then, one can model a power plant as a option spread, where electricity is sold at a given price S_t^e from which the price of gas prices S_t^g and CO2 taxes S_t^{co2} must be subtracted, accordingly to conversion rates β, γ. Mathematical models for commodities markets can be then expressed as a finite factor models. The payoff is:

$$g(S_T^e, S_T^g, S_T^{co2}) = (S_T^e - \beta S_T^g - \gamma S_T^{co2} - K)_+ \tag{6}$$

where K indicates the strike and $\beta, \gamma \in \mathbb{R}$ are the conversion rates.

In order to build a robust hedging strategy, our model is trained with a set of models, from Black-Scholes to Ornstein-Uhlenbeck two usually processes used to describe commodities, and for different correlations. Ornstein-Uhlenbeck is defined by:

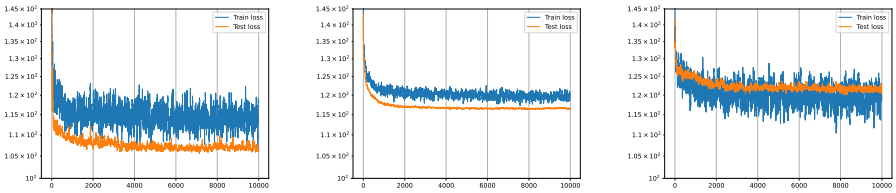
$$dS_t = \theta(\mu - S_t)dt + \sigma dW_t,$$

where $t \in [0, T]$, $(\theta, \mu, \sigma) \in \mathbb{R}^2 \times \mathbb{R}_+$ and W is a Brownian motion. For Black-Scholes processes, the drift constant is set at $r = 0.2$ and $\theta = 1.5, \mu = 1.5$ for Ornstein-Uhlenbeck. The maturity is 0.25, the strike is set at $K = 0.5$ and the spot $(S_0^e, S_0^g, S_0^{co2}) = (0.4, 0.15, 0.15)$. The conversion rates are respectively $\beta = 1.9$ and $\gamma = 0.35$ as usually done.

The function u describes the volatility function $\sigma(t, X_t)$ terms in the process X, then $u : x \to \sigma x$ or $u : x \to \sigma$, and is approximated on $q = 10$ sensors. We consider 36 price models varying according to the nature of the model, the value

Replication loss	Training models	Unobserved models
DeepOHedger	**108.51**	**42.82**
RDeepHedger	1103.50	916.78
CDeepHedger	110.63	43.45
Initial Risk	1006.88	809.16

Table 3: Average Replication loss on training and unobserved models.



(a) DeepOHedger (b) RDeepHedger (c) CDeepHedger

Figure 3: Training and testing losses by iterations.

of the constant $\sigma \in \{0.05, 0.10, ..., 5.00\}$ and the correlation matrices ρ_1, ρ_2. The correlation values for $(S_t^e, S_t^g, S_t^{co2})$ for any t are

$$\rho_1 = \begin{pmatrix} 1.0 & 0.1 & 0.9 \\ 0.1 & 1.0 & 0.5 \\ 0.9 & 0.5 & 1.0 \end{pmatrix}, \quad \rho_2 = \begin{pmatrix} 1.0 & 0.2 & 0.3 \\ 0.3 & 1.0 & 0.4 \\ 0.3 & 0.4 & 1.0 \end{pmatrix}$$

The deep hedgers are fed with 32 models during the training process, omitting $\sigma \in \{0.25, 0.45\}$ for both processes. The misspecified deep hedger MDeepHedger is not considered in this application, as the model itself can vary. For the computation of the replication loss, Monte Carlo simulations are unnormalized with average real prices on the period 2019-2020 from the European Energy markets.

Table 3 reports the replication loss between the real payoff and the one provided by the deep hedgers, as well as the initial risk (i.e the risk of the payoff without hedging $\mathbb{E}[(g(S_T^e, S_T^g, S_T^{co2}))^2]$). The operator hedger outperforms the robust and the conditional hedgers on both training and unobserved models. The gap between DeepOHedger and CDeepHedger is tight, which could be explained by the too basic volatility functions from the BS and OU models. The RDeepHedger is not able to hedge the initial risk, being incapable to distinguish between model samples. However, the global low losses open perspectives to hedge option spread in a robust manner.

5.4 Purely Data-Driven Approach for Risk Hedging

In this section, the deep operator hedger is tested on real data. To do so, data generation is operated to improve the training process of the hedging model. The generative model is trained to provide realistic synthetic prices from historical data. Regarding commodity simulations, most of the literature focus on designing stochastic models Deschatre et al. (2021). However, the selection of the model as well as its calibration are costly. Besides, new market conditions may negate these efforts. In the case of electricity price simulation, a joint simulation with other sources of energy (gas, coal) or weather conditions is also required. To address these issues, new deep generative methods inspired from computer vision Goodfellow et al. (2014); Kingma and Welling (2013) propose to provide realistic synthetic time series Xu et al. (2020); Yoon et al. (2019); Esteban et al. (2017); Mogren (2016). Combined with data-driven approaches for stochastic control, one can thus be able to handle a large class of models Fecamp et al. (2020). The objective is plural:

- full chain of learning models for risk management, from real data to optimal control learning
- joint data-driven generations of commodities
- robust operator hedging depending on the commodity models

For the generative method, we consider a model using a SDE formulation of the time series, whose drift and volatility functions can be given to the DeepOHedger as functions u.

Deep generative method for time series The Conditional loss Euler Generator (CEGEN) Remlinger et al. (2021) relies on Euler scheme of a SDE and on a conditional objective function to produce time series. A Itô structure (Eq.(1)) is proposed to ease the time series construction. The drift and volatility terms of the Itô process are approximated with a neural network of parameter φ, then the sequence is built with the Euler scheme (Eq.(3)) on the discrete grid $\mathcal{T}$. The conditional distance focuses on the distribution accuracy of the generated sequences at each time step. The loss is computed between values whose previous states belong to a common set, allowing thus to learn the distribution between each data point. The considered loss is the Bures-Wasserstein distance, a Gaussian formulation of the Wasserstein-2 ($\mathcal{W}_2$), providing theoretical bounds of the approximation errors of the drift and volatility terms. Unlike most of the works considering $\mathcal{W}_2$ distance Genevay et al. (2018); Xu et al. (2020), here there is no need of regularization. $Y^\varphi = (Y^\varphi)_{t\in\mathcal{T}}$ denotes the synthetic series provided by CEGEN. For a time step t_i, let be I a subset of $\mathbb{R}^d$ covering $\text{Supp}(S_{t_i}) \cup \text{Supp}(Y^\varphi_{t_i})$. The loss function is defined as:

$$\ell(S, Y^\varphi) = \sum_{i=0}^{N-1} \mathcal{W}_2^2(\mathcal{L}(S_{t_{i+1}}|S_{t_i} \in I), \mathcal{L}(Y^\varphi_{t_{i+1}}|Y^\varphi_{t_i} \in I)).$$

The parameter φ is optimized by minimizing

$$\min_{\varphi} \mathbb{E}[\ell(S, Y^{\varphi})].$$

Joint simulation of commodities Electricity prices depend on physical phenomena and other energy sources. As electricity is difficult to stock (excepting hydraulic dams), the offer-demand equilibrium has to be addressed at any time. The electric demand largely depends on weather conditions. For instance, low temperatures increase greater consumption and this affects electricity prices. In addition, the production of electricity is based on other sources of energy, including gas or coal. As these commodities are also traded on the markets, there exists a dependence between gas and electricity prices. Finally, renewable energies generate additional physical hazard due to intermittent production. When the wind is blowing, wind production increases and electricity prices decrease. In order to support the increasing integration of renewable energies and design effectively electricity prices, a joint modeling of risk factors is required. The model CEGEN is trained to generate daily commodities from 2016 to 2020. The time series considered are electric, gas and emission prices on European energy market. The generator is fed with samples of these prices and correlated exogenous variable such as dates (day, month), renewable production (solar, wind) and residual production. Finally, the generator outputs a 3-dimensional synthetic time series which will be dedicated to the training set of the deep operator hedger.

Deep hedging from generations Due to the specific SDE formulation, CEGEN provides drift and volatility functions which can be used as a parameter function u. In a first step, CEGEN is trained to generate new time series from different historical data depending on the location considered. In a second step, the DeepOHedger learns optimal controls from the synthetic data and approximations of the drift and volatility functions provided by the generator:

$$G(u)(x) = \alpha_\theta \begin{pmatrix} \mu^{\varphi}(t_i, x_1) & \dots, & \mu^{\varphi}(t_i, x_n) \\ \sigma^{\varphi}(t_i, x_1) & \dots, & \sigma^{\varphi}(t_i, x_n) \end{pmatrix} (t_i, Y^{\varphi}_{t_i})$$

By doing so, we benefit of a purely data-driven approach, from data generation to robust strategy learning.

Applications to electricity prices We aim to hedge an option spread for several location of spot prices with a single training. The location is France market and its boundary borders (Germany, Belgium, Italy, Swiss and Spain)[1]. Only the electricity price depends on the location, the gas and emission price stays the same. Six CEGEN are trained independently to produce jointly spot, gas and emission prices on the six locations. The payoff is described in Eq.

[1]The data comes from Réseau de transport d'électricité (RTE) datasets and are openly available online

	Training data		Testing data	
	Replication loss	Initial Risk	Replication loss	Initial Risk
French	42.35	694.32	128.83	564.96
German	44.58	659.01	131.68	650.90
Belgium	27.59	503.75	85.95	562.06
Italy	34.09	743.23	116.53	660.95
Swiss	42.41	694.70	135.70	699.01
Spain	54.87	784.83	238.82	564.79
Mean	40.99	679.98	139.59	617.12

Table 4: Average replication loss of DeepOHedger and initial risk on training and testing data.

6. Finally, the deep hedger is evaluated on a testing set, a set of unobserved historical prices during the training of both CEGEN and DeepOHedger.

Table 4 reports the replication loss and the initial risk on each location. The performances between countries are close, as every locations belong to the same electric network in Europe. This could be an expected behavior. A gap is observed between the replication losses of the training and testing set (including only real prices). An explanation lies on the fact that the performances on the testing set combine a model error from CEGEN as well as a generalization error from the DeepOHedger. However, these results compared to a non-hedged portfolio emphasis a good generalization ability of the deep operator hedger. To conclude, this test highlights how unsupervised approaches can address the challenges of probabilistic methods and open new application perspectives in risk management.

Conclusion

A new model solving several PDE in one single training is introduced and applied on risk hedging tasks. Brutal regime changes or unstationary environments are well known phenomenon, especially in finance. Relying on deep operator networks, our model addresses these issues and is able to learn the functions of input parameters simultaneously and thus solving a family of hedging problems. The benefit is double, the deep solver better generalizes with unobserved features and robust to unknown environments than several variant of deep hedgers. We evaluate our model named DeepOHedger several applications including local volatility models and option spread. We show that DeepOHedger reduces the generalization error even in the easiest linear problem. Finally, we present a purely data-driven approach to risk hedging, from time series generation to learning optimal controls. Our model then solves a family of parametric PDEs from synthetic samples produced by a deep generator previously trained on spot price data from different countries. Further works will focus on solving more

general PDE. Extensions with other schemes such as local approach can be considered.

References

A. Al-Aradi, A. Correia, D. Naiff, G. Jardim, and Y. Saporito. Solving nonlinear and high-dimensional partial differential equations via deep learning. *arXiv preprint arXiv:1811.08782*, 2018.

A. Al-Aradi, A. Correia, D. d. F. Naiff, G. Jardim, and Y. Saporito. Applications of the deep galerkin method to solving partial integro-differential and hamilton-jacobi-bellman equations. *arXiv preprint arXiv:1912.01455*, 2019.

E. Barucci, U. Cherubini, and L. Landi. Neural networks for contingent claim pricing via the galerkin method. In *Computational Approaches to Economic Problems*, pages 127–141. Springer, 1997.

C. Beck, E. Weinan, and A. Jentzen. Machine learning approximation algorithms for high-dimensional fully nonlinear partial differential equations and second-order backward stochastic differential equations. *Journal of Nonlinear Science*, 29(4):1563–1619, 2019.

F. Black and M. Scholes. The pricing of options and corporate liabilites. *Journal of Political Economy 81*, 1973.

A. Borji. Pros and cons of gan evaluation measures. *Computer Vision and Image Understanding*, 179:41–65, 2019.

E. Brophy, Z. Wang, Q. She, and T. Ward. Generative adversarial networks in time series: A survey and taxonomy. *arXiv preprint arXiv:2107.11098*, 2021.

Q. Chan-Wai-Nam, J. Mikael, and X. Warin. Machine learning for semi linear pdes. *Journal of Scientific Computing*, 79(3):1667–1712, 2019.

T. Chen and H. Chen. 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, 1995.

P. Cheridito, H. M. Soner, N. Touzi, and N. Victoir. Second-order backward stochastic differential equations and fully nonlinear parabolic pdes. *Communications on Pure and Applied Mathematics: A Journal Issued by the Courant Institute of Mathematical Sciences*, 60(7):1081–1110, 2007.

R. Cont. Empirical properties of asset returns: stylized facts and statistical issues. *Quantitative finance*, 1(2):223, 2001.

J. C. Cox. The constant elasticity of variance option pricing model. *Journal of Portfolio Management*, page 15, 1996.

D. Davydov and V. Linetsky. Pricing and hedging path-dependent options under the cev process. *Management science*, 47(7):949–965, 2001.

T. Deschatre, O. Féron, and P. Gruet. A survey of electricity spot and futures price models for risk management applications. *arXiv preprint arXiv:2103.16918*, 2021.

B. Dupire et al. Pricing with a smile. *Risk*, 7(1):18–20, 1994.

F. Eckerli. Generative adversarial networks in finance: an overview. *Available at SSRN 3864965*, 2021.

C. Esteban, S. L. Hyland, and G. Rätsch. Real-valued (medical) time series generation with recurrent conditional gans. *arXiv preprint arXiv:1706.02633*, 2017.

A. Fahim, N. Touzi, and X. Warin. A probabilistic numerical method for fully nonlinear parabolic pdes. *The Annals of Applied Probability*, 21(4):1322–1364, 2011.

S. Fecamp, J. Mikael, and X. Warin. Deep learning for discrete-time hedging in incomplete markets. *Journal of Computational Finance*, 2020.

S. FECAMP, J. MIKAEL, and X. WARIN. Deep learning for discrete-time hedging in incomplete markets. *Journal of computational Finance*, 2020.

N. Gao, H. Xue, W. Shao, S. Zhao, K. K. Qin, A. Prabowo, M. S. Rahaman, and F. D. Salim. Generative adversarial networks for spatio-temporal data: A survey. *arXiv preprint arXiv:2008.08903*, 2020.

A. Genevay, G. Peyré, and M. Cuturi. Learning generative models with sinkhorn divergences. In *International Conference on Artificial Intelligence and Statistics*, pages 1608–1617, 2018.

M. Germain, H. Pham, and X. Warin. Neural networks-based algorithms for stochastic control and pdes in finance, 2021.

K. Glau and L. Wunderlich. The deep parametric pde method: application to option pricing. *arXiv preprint arXiv:2012.06211*, 2020.

E. Gobet and R. Munos. Sensitivity analysis using itô–malliavin calculus and martingales, and application to stochastic optimal control. *SIAM Journal on control and optimization*, 43(5):1676–1713, 2005.

I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

J. Han and R. Hu. Recurrent neural networks for stochastic control problems with delay. *Mathematics of Control, Signals, and Systems*, 33(4):775–795, 2021.

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

J. Han et al. Deep learning approximation for stochastic control problems. *arXiv preprint arXiv:1611.07422*, 2016.

P. Henry-Labordere, N. Oudjane, X. Tan, N. Touzi, and X. Warin. Branching diffusion representation of semilinear pdes and monte carlo approximation. In *Annales de l'Institut Henri Poincaré, Probabilités et Statistiques*, volume 55, pages 184–210. Institut Henri Poincaré, 2019.

S. L. Heston. 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, 1993.

C. Huré, H. Pham, and X. Warin. Deep backward schemes for high-dimensional nonlinear pdes. *Mathematics of Computation*, 89(324):1547–1579, 2020.

J. M. Hutchinson, A. W. Lo, and T. Poggio. A nonparametric approach to pricing and hedging derivative securities via learning networks. *The journal of Finance*, 49(3):851–889, 1994.

N. Ikeda and S. Watanabe. *Stochastic differential equations and diffusion processes*. Elsevier, 2014.

Y. Khoo, J. Lu, and L. Ying. Solving parametric pde problems with artificial neural networks. *European Journal of Applied Mathematics*, 32(3):421–435, 2021.

D. P. Kingma and M. Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

J. Li, J. Yue, W. Zhang, and W. Duan. The deep learning galerkin method for the general stokes equations. *arXiv preprint arXiv:2009.11701*, 2020.

L. Lu, P. Jin, and G. E. Karniadakis. Deeponet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators. *arXiv preprint arXiv:1910.03193*, 2019.

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

E. Lütkebohmert, T. Schmidt, and J. Sester. Robust deep hedging. *arXiv preprint arXiv:2106.10024*, 2021.

M. Malliaris and L. Salchenberger. A neural network model for estimating option prices. *Applied Intelligence*, 3(3):193–206, 1993.

A. J. Meade Jr and A. A. Fernandez. The numerical solution of linear ordinary differential equations by feedforward neural networks. *Mathematical and Computer Modelling*, 19(12):1–25, 1994.

O. Mogren. C-rnn-gan: Continuous recurrent neural networks with adversarial training. *arXiv preprint arXiv:1611.09904*, 2016.

H. Pham. *Continuous-time stochastic control and optimization with financial applications*, volume 61. Springer Science & Business Media, 2009.

C. Remlinger, J. Mikael, and R. Elie. Conditional loss and deep euler scheme for time series generation, 2021.

J. Ruf and W. Wang. Neural networks for option pricing and hedging: a literature review. *Journal of Computational Finance, Forthcoming*, 2020.

H. A. Simon. Theories of bounded rationality. *Decision and organization*, 1(1): 161–176, 1972.

J. Sirignano and K. Spiliopoulos. Dgm: A deep learning algorithm for solving partial differential equations. *Journal of computational physics*, 375:1339–1364, 2018.

X. Tan. A splitting method for fully nonlinear degenerate parabolic pdes. *Electronic Journal of Probability*, 18:1–24, 2013.

Z. Wang, Q. She, and T. E. Ward. Generative adversarial networks in computer vision: A survey and taxonomy. *ACM Computing Surveys (CSUR)*, 54(2): 1–38, 2021.

X. Warin. Monte carlo for high-dimensional degenerated semi linear and full non linear pdes. *arXiv preprint arXiv:1805.05078*, 2018.

E. Weinan, J. Han, and A. Jentzen. 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, 2017.

T. Xu, L. K. Wenliang, M. Munn, and B. Acciaio. Cot-gan: Generating sequential data via causal optimal transport. *arXiv preprint arXiv:2006.08571*, 2020.

J. Yoon, D. Jarrett, and M. Van der Schaar. Time-series generative adversarial networks. *Advances in Neural Information Processing Systems*, 32, 2019.

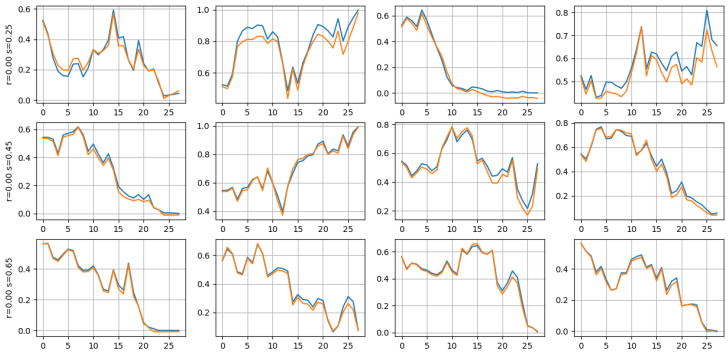


Figure 4: Optimal controls (in blue) compared with DeepOHedger's ones (in orange) for unobserved Black-Scholes models ($\sigma \in \{.25, .45, .65\}$) on 30 dates.

A Supplemental Materials

A.1 Hedging with Volatility Function Parameter

Table 5 reports the theoretical values and outputs from the DeepOHedger for premium and portfolio for all the volatility constants σ. Bold values indicate unobserved volatilities, that is values not present in the training set. The premiums and payoffs provided by our model align very closely with the theoretical ones, however the hedger systematically tends to price a little higher the call options than real values. There is no significant gaps on the performances between the training and unobserved models, emphasis a good generalization ability of the deep operator hedger. Table 6 focuses on the optimal controls and the ones outputted by our model. We compare the average over time of the controls for theoretical formula and DeepOHedger and the Mean Squared Error (MSE) between theoretical values and controls from the deep hedger at each time step. The results align very closely with the volatility values, including on unobserved models. The DeepOHedger is then also able to learn accurately optimal controls without being directly trained to do so.

A.2 Generating Time Series

One well-known challenge in the generation community is the lack of efficient evaluation metrics Borji (2019); Wang et al. (2021). The temporal dependencies of sequences makes even trickier the analysis Eckerli (2021); Gao et al. (2020); Brophy et al. (2021). However, one can provide a set of metrics to better characterizes specific features of the generations. As done in Remlinger et al. (2021), we consider the following metrics:

- Marginal based metrics includes classical statistics (mean, 95% and 5% percentiles denoted respectively **avg**, **q95**, **q05**). We compute the relative mean squared error (MSE) over time of these statistics in percentage.

σ	p	p th.	p loss	X_T	$g(X_T)$	Repl. loss
.05	.006979	.004987	.000004	.007035	.004913	.000012
.10	.011616	.009973	.000003	.011657	.009813	.000020
.15	.017096	.014957	.000005	.017047	.014698	.000026
.20	.021976	.019939	.000004	.021815	.019566	.000031
.25	**.026977**	**.024918**	**.000004**	**.026751**	**.024418**	**.000036**
.30	.031977	.029893	.000004	.031620	.029250	.000043
.35	.036977	.034863	.000004	.036465	.034064	.000050
.40	.041978	.039828	.000005	.041292	.038858	.000058
.45	**.046950**	**.044787**	**.000005**	**.046083**	**.043628**	**.000067**
.50	.051783	.049738	.000004	.050701	.048379	.000077
.55	.056616	.054682	.000004	.055299	.053109	.000090
.60	.061450	.059618	.000003	.059910	.057814	.000104
.65	**.066283**	**.064544**	**.000003**	**.064511**	**.062496**	**.000119**
.70	.071116	.069460	.000003	.069095	.067156	.000133
.75	.075942	.074366	.000002	.073668	.071799	.000150
.80	.080763	.079260	.000002	.078231	.076416	.000169
.85	.085547	.084142	.000002	.082739	.081016	.000189
.90	.090331	.089010	.000002	.087246	.085588	.000210
.95	.095114	.093866	.000002	.091764	.090131	.000233
Mean	.051446	.049622	.000003	.050154	.048058	.000096

Table 5: Payoff and premium of both deep hedger (X_T, p) and theoretical values $(g(X_T)$, p th.) on BS.
$X_0 = K = 0.5, N = 30$ *and maturity is* 0.25. *The bold values indicates price models omitted in the training process of the deep hedger.*

σ	$\overline{\alpha}$	$\overline{\alpha}$ th.	MSE
.05	.577527	.499875	.049805
.10	.582829	.500028	.024109
.15	.581975	.500180	.012088
.20	.569297	.500331	.007321
.25	**.558622**	**.500480**	**.004764**
.30	.549832	.500629	.003292
.35	.542273	.500777	.002350
.40	.535661	.500923	.001705
.45	**.530261**	**.501069**	**.001279**
.50	.527568	.501213	.001051
.55	.525225	.501357	.000894
.60	.523193	.501499	.000772
.65	**.521385**	**.501640**	**.000662**
.70	.519758	.501781	.000562
.75	.518336	.501920	.000484
.80	.517181	.502059	.000426
.85	.516037	.502197	.000370
.90	.515055	.502334	.000323
.95	.514238	.502470	.000286
Mean	.538224	.501198	.005923

Table 6: Controls both deep hedger and theoretical values on BS. $X_0 = 0.5, N = 30, K = 0.5$ *and maturity is 0.25.* $\overline{\alpha}$ *indicates the average over time of controls* α*, MSE is the mean squared error between theoretical values and controls from the deep hedger. The bold values indicates price models omitted in the training process of the deep hedger.*

	Marginal			Temporal	Correlation
	q05	Avg	q95	Qvar	Corr
French	4.77e-04	5.97e-04	2.27e-03	3.87e-02	4.70e-03
German	1.59e-03	8.20e-05	1.06e-03	6.70e-02	1.03e-02
Belgium	8.13e-04	9.11e-06	4.15e-04	5.52e-03	1.08e-02
Italy	2.06e-03	1.88e-04	7.90e-04	6.73e-02	1.61e-02
Swiss	5.44e-04	3.47e-05	3.30e-04	3.29e-02	1.23e-02
Spain	1.52e-03	4.39e-04	3.92e-03	3.95e-02	1.22e-02

Table 7: Evaluation metrics on gas price generation (the lower, the better).

Marginal metrics measures how well the overall envelop of the synthetic sequences fits with the real ones. The metric Qvar focuses on the temporal structure by evaluating the value variations between each time step.

These metrics ensure that the distribution is accurate and quantify the quality of the overall time series envelop.

- The temporal dependencies are measure with the relative quadratic variation (**QVar**) with $QVar(X) = \sum_i |S_{t_{i+1}} - X_{t_i}|^2)$.
- Correlation structure **Corr** is evaluated with the time average MSE between the terms of the covariance matrices of synthetic and reference sequences.