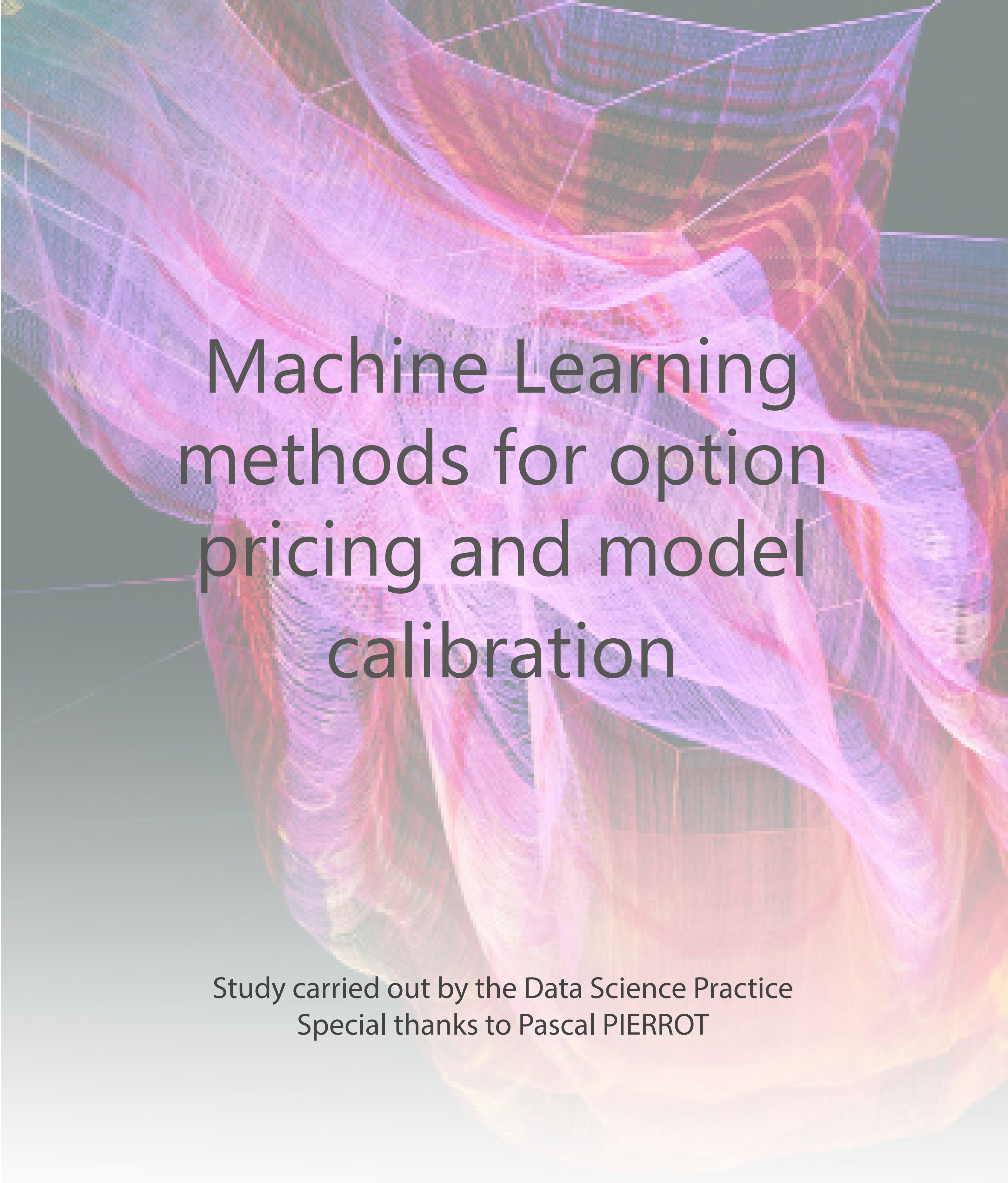


Machine Learning methods for option pricing and model calibration

Study carried out by the Data Science Practice
Special thanks to Pascal PIERROT

coperneec

DECEMBRE 2020

Summary

Introduction

The aim of this article is to present the application of machine learning and deep learning to option pricing and financial model calibration in a data-driven approach. Efficient numerical computation has become increasingly important in finance with real-time risk management or counterparty credit risk. The motivation to use machine learning methods is to save computational cost in comparison with classical numerical methods such as Monte-Carlo, numerical integration or root-finding without loss of precision.

More concretely, the goal of the study is to train a machine learning model to learn the price of an option or IV surface given model parameters or, in the case of calibration, the parameters of the model given observed market prices or implied volatility. The data we use to train machine learning models are generated from classical pricing methods.

In this article, we limit our study to Black Scholes and Heston frameworks for single or multiple underlying European options and American option. We also tackle the problem of calibration and implied volatility surface generation for the Heston model from a machine learning perspective.

We conclude that the machine learning approach can be time efficient and very accurate for these problems.

1 Option pricing frameworks

In this section we briefly present the classical option pricing frameworks such as Black Scholes and Heston stochastic volatility model along with different pricing methods we use in our study to generate synthetic data and compare results

1.1 The Black Scholes framework

In the Black-Scholes framework, a non-dividend paying asset price is assumed to follow a Geometric Brownian process which can be expressed under the risk neutral measure as:

$$dS_t = rS_t dt + \sigma S_t dW_t^{\mathbb{Q}}$$

A PDE for option price can be derived by applying the Ito lemma to the option price expressed as a function of t and S_t, and by using the principle of portfolio replication :

$$\frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial V^2}{\partial S^2} + rS\frac{\partial V}{\partial S} - rV = 0$$

with terminal condition $V(t = T, S) = (S_0 - k)_+$ for an European call option

1.2 Implied volatility in Black Scholes

The Black Scholes implied volatility corresponds to the volatility σ^* in the BS model that matches the BS price with the observed market liquid option prices and can be expressed as the solution of:

$$BS(\sigma^*, S_0, K, T, r) = V_{market} \tag{1}$$

The monotonicity of the BS equation with respect to σ guarantees its existence. It is an important quantity in finance as market convention is to always quote implied volatility using the Black-Scholes model. It is often represented as a function of strikes and maturities (volatility surface).
Because the previous formula is not invertible, we use numerical iterative methods to find the implied volatility.

1.3 The Heston model

The Heston model is one of the multiple extensions of the BS framework to deal with non-constant observed volatility (smile or skew). While the BS assumes the volatility to be constant, the Heston model assumes that the volatility follows a stochastic process correlated with the price of the underlying. It is expressed under the risk neutral measure as :

$$dS_t = rS_t dt + \sqrt{v_t} S_t dW_t^S$$

$$dv_t = \kappa(v - v_t)dt + \gamma\sqrt{v_t} dW_t^v$$

$$d\langle W^S, W^v\rangle_t = \rho_{Sv} dt \quad , \quad S_{t_0} = S_0, \quad v_{t_0} = v_0$$

A PDE for option price can also be derived by a martingale approach where the price depends on t, S_t and v_t. The observed implied volatility shape or market option prices can be reproduced by varying the Heston parameters $\{\kappa, \rho, \gamma, v, v_0\}$. The parameter γ usually impacts the kurtosis of the asset return distribution, and the coefficient ρ controls its asymmetry

1.4 Classical pricing methods

For European plain vanilla options, a closed form solution exists as follow for a one-asset call price:

$$V(t, S_t) = S_t N(d_1) - Ke^{-r(T-t)}N(d_2)$$

with : $d_1 = \frac{\log S/K + (r - 0.5\sigma^2)(T-t)}{\sigma\sqrt{T-t}}$ and $d_2 = d_1 - \sigma\sqrt{T-t}$

When no analytical solution exists, a numerical method is needed to solve the pricing equation. There are three main categories of numerical methods : Finite difference, Monte-Carlo and numerical integration.

While the finite difference method solves the pricing PDE numerically, Monte-Carlo and numerical integration methods rely on the Feyman-Kac representation Theorem which says that the solution of a parabolic PDE can be expressed as a conditional expectation under some measure Q. In the case of European option, the price can be written as the conditional expectation of the discounted terminal payoff under the risk neutral measure:

$$V(t, S_t) = E_t^{\mathbb{Q}}[e^{-\int_t^T r dt} V(T, S_T)] = e^{-r(T-t)} \int_{\mathbb{R}} V(T, S_T) f(S_T|S_t) dS_T$$

The Monte-Carlo method approximates the conditional expectation form by simulating many paths of the underlying, evaluate the payoff $V(T, S_T)$ for each path and then take the mean of discounted payoffs as an approximation of the option price. The convergence rate is in $O(1/\sqrt{N})$ where N is the number of paths. The numerical integration estimates the density function in the second integral form. To generate prices under the Heston model for our study, we use the COS method developed in [7], a variant of Fast Fourier Transform method which consists in replacing the density function in the integral by its Fourier-cosine expansion and use the known characteristic function to evaluate the series coefficients.

For an American option paying $g(\tau)$ if exercised at time $\tau < T$, the risk neutral price is given by the optimal stopping problem (for complete market):

$$V(t) = \sup_{t<\tau<T} E_t^{\mathbb{Q}}[e^{-r(\tau-t)}g(\tau)]$$

and it verifies the following dynamic programming principle:

$$V(t) = max\left[g(t), E_t^{\mathbb{Q}}[e^{-rdt}g(t+dt)]\right]$$

The price can be computed using the least square Monte-Carlo algorithm. This algorithm addresses the dynamic programming problem by simulating many paths of the underlying, then fitting, backward recursively in time, regression models that predict continuation values (values of the option at the next step previously calculated) given the current asset prices with a polynomial basis expansion. More details about the algorithm can be found in ref [8].

1.5 Numerical methods for implied volatility

Classical iterative methods to approximate the solution of (1) include the Newton-Raphson method, the bisection method and the Brent method. In the case of Newton-Raphson, starting with an initial guess σ_0^*, the following iterative update is performed until a criterion is satisfied :

$$\sigma_{k+1}^* = \sigma_k^* - \frac{V(\sigma_k^*) - V_{market}}{\partial V / \partial \sigma_k^*}$$

where the denominator is the Vega of the option.
The Brent's method is another root finding method more robust which combine the bisection method, the secant method and inverse quadratic interpolation. More details can be found in [1].

2 The machine learning approach

Using Machine learning applied to derivative pricing or model calibration has seen increasing interest in recent years, and both practitioners and academics have worked on the subject.
In the machine learning approach for pricing, we approximate the pricing function by a Machine learning model. The model is trained on data generated from classical pricing methods in a supervised learning fashion. More precisely, it consists in the following steps :

- Generating a set of the financial model parameters using random sampling methods
- Use methods of section 1 to compute the price of the option for each parameter
- Split the above dataset into a training sample and a testing sample in a proportion 70% 30%.
- Train the machine learning to map the generated input parameters to the option price using training data
- Use the trained model as the new pricer on the test set

Because we only use the machine learning model in inference mode for pricing, the calculation is expected to be much faster than classical approaches.

In the paper [1], the authors treat the case of European option's price under Black Scholes and Heston with neural networks. We will follow the same way but extend the study to multiple-underlying European options (Baskets) and American options. We also test the Gaussian process regression approach introduced in [5] and compare its performances with neural network. We then focus on the implied volatility computation with neural networks.

2.1 Neural networks

Artificial neural networks (ANN) are models inspired by biological neural networks. It can generally be described by three levels of components: neurons, layers and global architecture. Each neuron of a layer, called unit, is connected to each neuron of the next layer and is associated with a learnable weight, a bias term and an activation function σ. More precisely, in the case of multilayer perceptron architecture (MLP), a neuron consists in the following three operations:

- Calculate the sum of weighted inputs
- Addition of a bias
- Application of the activation function to the precedent calculated quantity

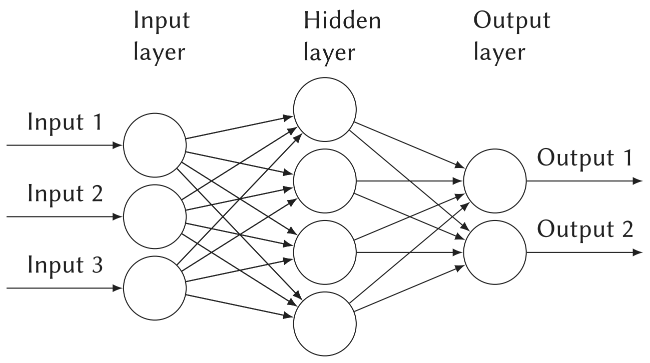


Figure 1: The multi-layer perceptron

Let's denote w_{jk}^l the weight for the connection from the kth neuron in the $(l-1)$th layer to the jth neuron in the lth layer. We have the following relation:

$$a_j^l = \sigma\left(\sum_k w_{jk}^l a_k^{l-1} + b_j^l\right) = \sigma(z_j^l) \qquad (1)$$

where a_j^l is the output of the jth neuron in the lth layer.
The most famous activation functions are the following:

- Relu: $\sigma(x) = max(x, 0) \quad \in [0, +\infty[$
- Sigmoid: $\sigma(x) = \frac{1}{1+e^{-x}} \quad \in [0, 1]$
- Tanh: $\sigma(x) = \frac{1-e^{-2x}}{1+e^{-2x}} \quad \in [-1, 1]$

The use of non-linear activation function allows the model to create complex mappings between the network's inputs and outputs. An activation has to be chosen in order to deal with the following issue:
explosion or vanishing gradient: The bounded and smooth nature of Sigmoid and tanh prevents the gradient to explode but is subject to vanishing for high absolute input values making the learning process slower.
Making clear predictions: by the nature of sigmoid and tanh that tends to separate the outputs.
computaion efficiency: The Relu activation is faster to compute and allows the network to converge faster

Training an ANN: the principle of backpropagation

Backpropagation addresses the problem of how a variation in weight and biases impact the output error.

In order to learn, a neural network needs a routine to update its weights and biases in the direction of minimizing a loss function on the output which is generally a gradient descent algorithm. In supervised regression problem, the output loss is often the MSE: $L = \frac{1}{n}\sum_x | y(x) - a^L(x) |^2$, L being the number of layers. More generally, if we express the output as a function of the input and network parameters: $\hat{y} = f(x, \Theta)$, $\Theta = (W_1, b_1, .., W_L, b_L)$, the training process is the following optimization problem :

$$\text{argmin}_\Theta L(\Theta|(x, y))$$

. The gradient descent is an iterative process that moves toward loss function decrease and update parameters as follows:

$$\begin{cases} W \leftarrow W - \alpha\frac{\partial L}{\partial W} \\ b \leftarrow b - \alpha\frac{\partial L}{\partial b} \end{cases}$$

where α is the learning rate.
The backpropagation algorithm finds a way to compute the partial derivatives $\frac{\partial L}{\partial W}$ and $\frac{\partial L}{\partial b}$. Starting from the output nodes errors, knowing $\frac{\partial L}{\partial a^L}$, it relies on the following chain rule (for a single weight) :

$$\frac{\partial L}{\partial w^l_{jk}} = \frac{\partial L}{\partial z^l_j}\frac{\partial z^l_j}{\partial w^l_{jk}} = \frac{\partial L}{\partial z^l_j}a^{l-1}_k$$

In fact, backpropagation stands only for one training instance (x, y), but the loss function can generally be expressed as the average over losses of individual instances : $L = \frac{1}{n}\sum_{i=1}^n L(x_i)$ where n is the batch size, making possible to update network parameters with many instances.
Finally, training a neural network consists of initialize weights and biases, then repeating for each batch:

- forward propagate: compute the output given an input x using current weight and biases and formula (1)
- compute the output error
- Backpropagate the error
- Update weights and biases using the gradient

The weakness of neural networks is the difficulty to interpret its results. Besides, they are usually sensitive to over-fitting and need a high number of samples to be properly trained.

2.2 Gaussian process regression

Gaussian process regression (GPR) is a nonparametric kernel-based probabilistic model which relies on Gaussian process. A Gaussian process is a collection of random variables whose any finite subset of this collection has a joint Gaussian distribution. It is defined by its mean function $m(w)$ and covariance (kernel) function $k(x, x')$.

Given a dataset $\{(X_i, y_i) \mid i = 1..n\}$, the relation between each input and output for GPR is given by :

$$y_i = f(X_i) + \epsilon_i \qquad \epsilon_i \sim \mathcal{N}(0, \sigma_n^2)$$

Where f is a Gaussian process verifying:

$$f \sim \mathcal{N}(0, K(X, X)) \qquad K(X, X) = \begin{pmatrix} k(x_1, x_1) & .. & k(x_1, x_n) \\ .. & .. & .. \\ k(x_n, x_1) & .. & k(x_n, x_n) \end{pmatrix}$$

The Radial-basis function kernel is the most commonly kernel used and is defined as:

$$k(x, x') = \sigma_f^2 exp(-\frac{d(x, x')^2}{2l^2})$$

where d is the euclidean distance and l the length scale of the kernel. This makes two output points y and y' close in input space highly correlated.

Starting from a prior distribution, training a GPR consists in finding the most appropriate set of hyperparameters $\Theta = \{l, \sigma_n, \sigma_f\}$ using the training data and leads to a posterior distribution. The common way is to maximize the log marginal likelihood function $L(\Theta) = ln(p(y|X, \Theta))$ which is done by a gradient-based optimization method such as L-BFGS.

Given the previous posterior on train dataset, for a new set of observed samples X^*, we infer their unknown values $f^* = f(x_i^*)$ using the following joint distribution for the posterior :

$$\begin{bmatrix} y \\ f^* \end{bmatrix} = \mathcal{N}\left(0, \begin{bmatrix} K(X, X) + \sigma_n^2 I & K(X, X^*) \\ K(X*, X) & K(X^*, X^)\end{bmatrix}\right)$$

which leads to a conditional distribution :

$$f^*|X^*, X, y \sim \mathcal{N}(\mu, \Sigma)$$

where the mean

$$\mu = K(X^*, X)[K(X, X) + \sigma_n^2 I]^{-1} y$$

is the point estimate, and the covariance matrix

$$\Sigma = K(X^*, X^*) - K(X^*, X)[K(X, X) + \sigma_n^2 I]^{-1} K(X, X^*)$$

is used as model uncertainty.

A limit of GPR is that it usually not scale very well for high dimensional datasets.

2.3 Optimization of hyperparameters

Hyperparameters are parameters that control the learning process or define the overall model architecture. For Gaussian Process regression, these parameters are automatically optimized during training process but the parameters of the ANN such as learning rate, batch size or number of neurons and layers need to be chosen in a way to optimize the generalisation performance of the model. Usually, the optimization is performed using a cross-validation score : the training dataset is divided into p partitions (folds) and the model is successively trained on $p-1$ partitions and tested on the remaining one. Then the score is averaged over test folds. Hyperparameters that obtained the best score are used to train the final model.

The optimization of hyper parameters can therefore be formulated as finding the the optimum of a function $f : \mathcal{H} \longrightarrow \mathbb{R}$ mapping a hyperparameter space to the set of cross-validation scores (obtained for each possible hyperparameter combination). Because f is expensive to evaluate (it involves several model trainings), we evaluate f only for some points of $\mathcal{H}$. Here are three approaches to choose these points :

- The grid search approach where we define a discrete range of points for each hyperparameter then test all possible combinations
- The random search approach in which a random draw of hyperparameter combination is performed a certain number of times
- The Bayesian optimization approach which is an iterative approach that uses previously tested points to help reducing the search space of better hyperparameter combination.

In the paper [1] authors use a random search on a small dataset. For our study, we decide to test the Bayesian approach.

As developed in [9], the idea behind this approach is to optimize a model of f or a surrogate that is cheaper to evaluate. The algorithm is an iterative process which consists in performing a numerical optimization on the surrogate function to find the next point where the true function f should be evaluated. Previously evaluated points are used to update the model on f.
The criteron that is optimized to find the next point to evaluate x^* is the following expected improvement:

$$EI(x) = \int_{-\infty}^{y^*} (y^* - y) p_M(y|x) dy$$

where M is a probability representation model of f built using previous observations and y^* a threshold value. It can be interpreted as a trade-off between staying close to the previous optimal point and being in under explorer region.

The Tree-Parzen Estimator algorithm we use in our case models $p(x|y)$ and $p(y)$ by the means of hierarchical processes calibrated using initial input search space defined as probability distributions and previously observed values and then use Bayes rule.

The Bayesian method has a better convergence in comparison with Random search but is subject to local minima.

2.4 Pricing with machine learning

Data generation

The first step is to create a set of combinations of model, option and market parameters (strike, maturities, volatility, risk-free rate, ...) which will be the inputs of the machine learning model. In this study, we use a uniform random sampling on each parameter. Other methods such as Latin hypercube sampling can be used to have better distributed data in the whole parameter space.

For the multi-asset case (basket option), the Scipy implementation of the numerically stable algorithm of Davies & Higham is used to generate random correlation matrices of underlyings.

Once model parameters are generated, we compute the price for each parameter combination using respectively the closed form solution or Monte-Carlo method for European options in the BS model, the COS method for European option in the Heston model and the Least square Monte-Carlo method (LSM) for American put option.

The pricing methods are implemented in Python using the Numpy library. For the least square Monte-Carlo, we use a linear least square regression on polynomial expansion with the PolynomialFeatures method of scikit-learn. The American option is approximated by a Bermuda option with 50 exercise dates and the underlying follows a geometric Brownian.
The Basket option is a call on the index of kind $I_t = \sum a_i S_t^i$ where we use equal weights.

It takes approximately 36s to generate 10 000 samples prices for the basket option (10^5 paths) and about the same time for call prices in the Heston model.

The table below summarizes the input and output data for the supervised machine learning training for the various options and models tested.

	European call option (BS)	
Inputs	Moneyness:	$S_0/K \in \mathcal{U}[0.4, 1.6]$
	Maturity (yearly):	$\tau \in \mathcal{U}[0.2, 1.1]$
	volatility:	$\sigma \in \mathcal{U}[0.01, 1]$
	risk-free rate:	$r \in \mathcal{U}[0.02, 0.1]$
Output	B-S price/K	$[0, 0.85]$
	American put option (BS)	
Inputs	Same τ, σ, r:	
	Initial price :	$S_0 := 1$
	Strike:	$K \in \mathcal{U}[0.4, 1.6]$
Output	LSM price ($N = 10^4$)	$[0, 0.8]$
	European basket call (BS)	
Inputs	Same τ, r	
	S_0^i/K, i=1..d :	$S_0^i/K \in \mathcal{U}[0.2, 1.1]$
	σ^i, i=1..d :	$\sigma^i \in \mathcal{U}[0.01, 1]$
	$\rho^{i,j}$, i,j=1..d :	$\rho^{i,j} \in [-1, 1]$
Output	MC price/K ($N = 10^4$)	$[0, 0.6]$

	European call option (Heston)	
Inputs	same S_0/K, τ, r	
	initial variance:	$v_0 \in \mathcal{U}[0.05, 1]$
	long-term variance:	$v \in \mathcal{U}[0.01, 1]$
	reversion speed:	$\kappa \in \mathcal{U}[0.01, 3]$
	gamma:	$\gamma \in \mathcal{U}[0.01, 0.8]$
	correlation:	$\rho \in \mathcal{U}[-0.9, 0]$
Output	COS price	$[0, 0.6]$

Choice of model and hyperparameters

The Gaussian process regression is trained with a Radial-basis function kernel and L_BFGS optimizer.

The ANN is a multi-layer perceptron (MLP) implemented using Keras, a high level deep learning library implemented on top of TensorFlow.

The weights and biases are initialized using a glorot uniform distribution. The mean squared error is used as loss function. We use Adam as optimization algorithms which is a combination of Momentum and adaptive Learning Rates and which converge faster in general. Inputs are scaled using Min-max scaling: $x_{scaled} = \frac{x - x_{min}}{x_{max} - x_{min}}$ and a linear activation (identity function) is used for the last layer

We decide to tune the following hyperparameters using the Bayesian search for about 20 iterations (through Hyperopt library) with the average mean squared error on 5 validation folds as objective:

- Number of layers : 2 or 3
- Hidden size : Choice in [40, 400]
- batch size : Choice in [32, 64, 128]
- epochs : choice in range [100, 300]
- learning rate : log-uniform distribution $[1.10^{-4}, 1.10^{-3}]$
- Activation function : [elu, relu, tanh, linear, sigmoid]

Then we analyze the points evaluated during the optimization to choose the most appropriate one (trade off between complexity and performance) and retrain the final model on the whole training set.

The models have been trained using different numbers of generated data samples (taking 30% of samples for testing). In a sake of simplicity and limited computer resources, we only optimize MLP hyperparameters for the 10^5 samples case then adapt the model for other cases using the early stopping criterion.

Results

We detail below the results in the case Black Scholes European option on the test set:

Single asset European call option (BS)			
samples	model	MSE (test set)	MAE (test set)
N=10^6	MLP	$5.79 \cdot 10^{-7}$	$5.99 \cdot 10^{-4}$
N=10^5	GPR	$2.70 \cdot 10^{-6}$	$7.48 \cdot 10^{-4}$
	MLP	$1.81 \cdot 10^{-6}$	$1.08 \cdot 10^{-3}$
N=5000	GPR	$3.21 \cdot 10^{-6}$	$8.37 \cdot 10^{-4}$
	MLP	$4.00 \cdot 10^{-6}$	$1.53 \cdot 10^{-3}$

model	hyperparameters		time cost (10^5)
GPR	-		train: 13.47s
			test: 0.65s
MLP	hidden shape:	(280, 280)	train: 65.43s
	activation:	relu	test: 0.162s
	learning rate:	$1 \cdot 10^{-4}$	
	epochs:	230	
	batch size:	64	

The performances are quite satisfying both in term of mean squared error and mean absolute error. We notice a slightly better performance for GPR in the case of smaller dataset suggest that this model can be more appropriate for small datasets. Nevertheless, GPR does not improve its performance and becomes time-consuming for higher sample size while the MLP is still improving with more samples.

An analysis of the absolute errors across moneyness, maturity and volatility range shows a worse performance close to the money and close to each bound of these values. If the former is due to more scattered data, the latter may highlight worse prediction for points close or out of training data space and raises the necessity to train the model on a sufficiently wide input range.

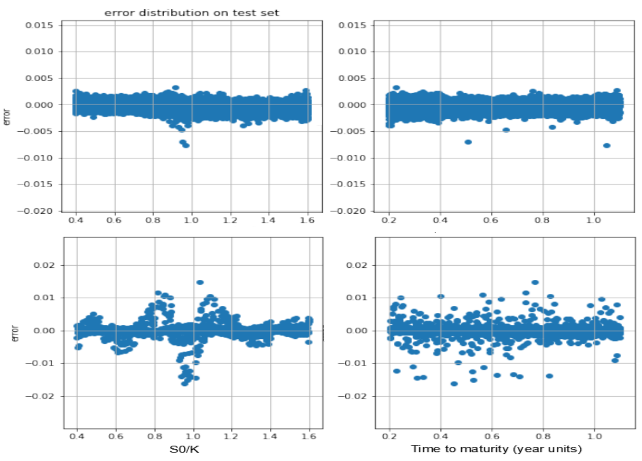


Figure 2: Error across moneyness (right) and maturities (left) for MLP (above) and GPR (below)

The following shows the performance for the other tested options on the test set:

European Basket call on 4 underlyings (BS)			
samples	model	MSE	MAE
N=10^5	GPR	$2.41 \cdot 10^{-5}$	$3.66 \cdot 10^{-3}$
	MLP	$1.48 \cdot 10^{-5}$	$2.91 \cdot 10^{-3}$

American put option			
samples	model	MSE	MAE
N=10^5	GPR	$6.81 \cdot 10^{-6}$	$1.82 \cdot 10^{-3}$
	MLP	$4.15 \cdot 10^{-6}$	$1.42 \cdot 10^{-3}$

European call option (Heston)			
samples	model	MSE	MAE
N=10^5	GPR	$6.05 \cdot 10^{-4}$	$8.1 \cdot 10^{-4}$
	MLP	$3.84 \cdot 10^{-4}$	$3.98 \cdot 10^{-3}$

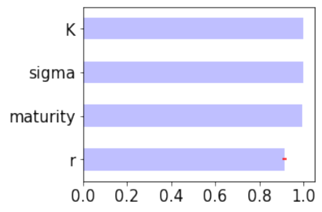
The performances are obtained for the MLP with respectively (170, 170) hidden shape for the Basket option, (280, 280, 110) for the American option and (220, 220, 120) for the call under the Heston model. The training has been performed using between 200 and 250 epochs. The time cost of prediction is still competitive: around 1s for GPR and less than 0.1s for MLP on 3000 test sample. The neural network model tends to be better and less prone to overfitting for these cases.

Feature importance

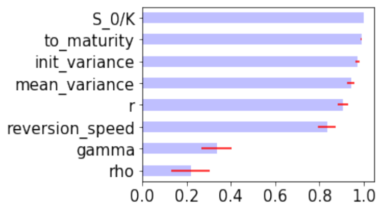
Feature importance can be computed for models using the mean decrease accuracy method : We successively randomly shuffle each input variable and measure the decrease in model performance (mean squared error criterion). Then, compute the importance of each feature in a cross-validation fashion as :

$$Imp_{feat} = -\frac{1}{N_{test\ folds}} \sum_{p \in test\ folds} \frac{MSE^p - MSE^p_{f\ shuffled}}{MSE^p_{f\ shuffled}}$$

The uncertainty of feature importance can also be computed using the standard deviation over test folds and optionally over multiple repeating procedure for different random seeds.



(a) American put option



(b) European call (Heston)

Figure 3: Feature importances obtained by mean decrease accuracy method (MSE metric) over 5 folds for MLP

2.5 IV surface generation with ANN

In this section, we experiment the deep learning approach to BS implied volatility surface generation for the Heston model. The problem can be addressed in different ways. Authors in [1] test the approach with an ANN trained on the output of the ANN used for pricing. In [2], they learn the implied volatility directly from model parameters θ and moneyness/maturities as inputs using uniform sampling. Following the approach of [3], We show that a neural network can generate the complete surface through the learning map :

$$\mathcal{F} : \theta \longrightarrow \{\sigma^{\theta}_{BS}(K_i, T_j)\}_{i,j}$$

where a grid of moneyness / maturities is fixed in advance. Note that with this approach we are free to define the grid as fine as needed.

Input Heston parameters $\theta = \{v_0, v, \kappa, \gamma, \rho\}$ are generated by uniform sampling and the risk-free rate is fixed to zero. Then, the Heston prices are computed using the COS method for each parameter sample and each pairs moneyness - maturities of the grid. Finally, the BS implied volatility is calculated using root-finding method for each pair moneyness - maturity. It takes approximately 0.74 sec to generate one IV surface.

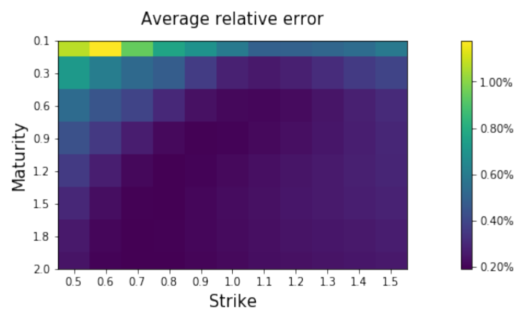
The following table shows inputs and outputs of the neural network :

	Heston IV generation dataset	
Inputs	initial variance:	$v_0 \in \mathcal{U}[0, 0.04]$
	long-term variance:	$v \in \mathcal{U}[0.01, 0.2]$
	reversion speed:	$\kappa \in \mathcal{U}[1, 10]$
	gamma:	$\gamma \in \mathcal{U}[0.01, 1]$
	correlation:	$\rho \in \mathcal{U}[-0.9, -0.1]$
Output	8x11-length vector	$K \in [0.5 - 1.5]$
	of $\sigma_{BS}(K, T)$	$T \in [0.1 - 2.0]$

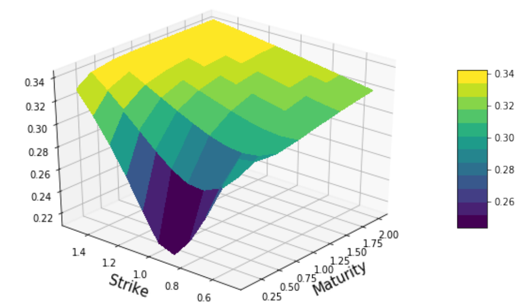
We choose an MLP architecture with 3 layers for the network and optimize hyperparameters using the Bayesian search approach. It gives the following results with a hidden shape (120, 120, 95) and 200 epochs (10^5 training samples and 1800 testing sample) where MSE and MAE are uniform averages over the grid:

train cost	test cost	MSE (test)	MAE (test)
85s	0.05s	$2.09 \cdot 10^{-6}$	$9.50 \cdot 10^{-4}$

The figure below shows the relative error distribution over the IV surface (difference with COS + Root finding values) :



(a) Average relative error in %



(b) IV surface generated by the MLP model ($v_0 = 0.0062$, $\rho = -0.63$, $\gamma = 0.14$, $v = 0.1$, $\kappa = 6.1$)

Figure 4

The table below shows the relative importance of Heston parameters computed with the mean decrease accuracy method.

Parameter	v_0	ρ	γ	κ	v
Feature importance	0.955	0.981	0.984	0.99	0.999

3 Calibration with ANN

In this section, we approach the application of artificial neural networks to the calibration of volatility model. We will focus on the Heston calibration from IV. The cases of Bates model and Rough volatility are treated respectively in [2] and [3]. The author in [4] discusses the calibration of the Heston model from prices.

3.1 Problem formulation

For a stochastic financial model $\mathcal{M}(\Theta)$, the calibration is an inverse problem which consists in finding the most appropriate model parameters θ ($\theta := \{v_0, v, \kappa, \gamma, \rho\}$ in the case of Heston) such as the price generated by the model corresponds exactly to the observed market prices of liquid instruments or the observed BS implied volatility surface. The calibration procedure is essential in finance for risk management purpose and to price OTC exotic derivatives.

More precisely, for a set of M observed market prices with various strikes and maturities $\{V^{mkt}(K_i, T_i), i = 1, .., M\}$ the problem consists in solving the following :

$$\underset{\theta \in \Theta}{\operatorname{argmin}} \sum_{i=1}^{M} | V^{\mathcal{M}(\theta)}(\theta, K_i, T_i) - V^{mkt}(K_i, T_i) |^2$$

Or in the case of observed Implied volatility

$$\underset{\theta \in \Theta}{\operatorname{argmin}} \sum_{i=1}^{m} \sum_{j=1}^{n} | \sigma_{BS}^{\mathcal{M}(\theta)}(K_i, T_j) - \sigma_{BS}^{mkt}(K_i, T_j) |^2$$

Classical methods to solve this problem which is known to be non-linear and non-convex include root-finding, quasi-Newton algorithms or stochastic optimization. Many of them are computer intensive or need to be initialized close to the optimum. The ability of neural networks to learn non-linearity makes it a good candidate to address the problem.

3.2 Calibration from IV surface

The calibration problem can be approached in several ways with ANN. One can learn the inverse mapping :

$$\mathcal{F}^{-1} : \{\sigma_{BS}^{mkt}(K_i, T_j)\}_{i,j} \longrightarrow \hat{\theta}$$

either with market data or synthetic data using a pricer then a root finding method to know in advance the true parameters. Another approach is to use the ANN trained in the previous section which maps the Heston parameters to the IV surface. This last approach is developed in [3]. The principle is, starting with the previous learned map :

$$\mathcal{F} : \theta \longrightarrow \{\sigma_{BS}^{H(\theta)}(K_i, T_j)\}_{i,j}$$

we use the trained model at inference time to solve, for each new observed IV surface $\{\sigma_{BS}^{mkt}(K_i, T_j)\}_{i,j}$ the following optimization problem:

$$\hat{\theta} := \underset{\theta \in \Theta}{\operatorname{argmin}} \sum_{i=1}^{m} \sum_{j=1}^{n} | f(w, \theta, K_i, T_j) - \sigma_{BS}^{mkt}(K_i, T_j) |^2$$

where $f(w, \theta, ., .)$ is the output of the previous MLP on input θ with fixed trained parameters w.

This optimization can be solved by a gradient-based method such as L-BFGS using the Jacobian extracted from the ANN, or gradient-free methods more adapted to non-convex objective function such as differential evolution. The second method might be more appropriate due to the nature of the calibration mapping which is not guaranteed to be $\mathcal{C}^1$ and the objective function can be subject to many-to-one problem: two different sets of parameters may result in the same IV surface leading to potential local minima.

We test this approach using the same test set as in section 2.5. We perform the optimization with differential evolution on 100 samples of the test set and restrict the search space to the initial Heston parameter range. It gives the following error performance for each parameter :

Parameter	v_0	v	κ	γ	ρ
MAE	$9.13 \cdot 10^{-4}$	$8.86 \cdot 10^{-4}$	0.147	$1.55 \cdot 10^{-2}$	$2.98 \cdot 10^{-2}$
MAPE	28.96%	0.81%	2.33%	4.8%	7.8%

The method of the inverse learning map has also been tested (IV as input and Heston parameters as output) with both MLP network and Convolutional network (CNN). We have found this approach to be faster than the previous optimization one with our configuration. In this case, a min-max scaling is also applied on output values because of the high difference across each parameter in terms of range. We use the same dataset as previously (10000 train samples and 1800 test samples) and use the following architectures :

- MLP: hidden shape (100, 100)
- CNN: one convolutional layer with (2,2) kernel size, 32 filters and a fully connected dense layer of size 200

Models are trained using 150 epochs, a batch size of 64 and a learning rate of $5 \cdot 10^{-4}$.
The table below shows the performances on the test set

MLP	v_0	v	κ	γ	ρ
MAE	$7.22 \cdot 10^{-4}$	$1.14 \cdot 10^{-3}$	0.101	$1.27 \cdot 10^{-2}$	$3.69 \cdot 10^{-2}$
MAPE	22.23%	0.99%	1.82%	9.46%	12.53%
CNN	v_0	v	κ	γ	ρ
MAE	$2.75 \cdot 10^{-4}$	$1.53 \cdot 10^{-3}$	0.11	$1.08 \cdot 10^{-2}$	$3.9 \cdot 10^{-2}$
MAPE	9.77%	1.3%	2.38%	7.09%	12.7%
Prediction cost:	MLP: 0.166s		CNN: 0.215s		

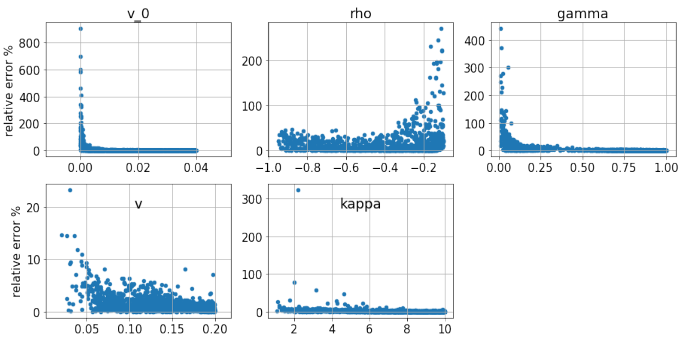


Figure 5: Error distribution across parameter range for CNN

Conclusion

In this paper, we have discussed several applications of machine learning in derivative modelling and found that it is promising in terms of computational aspect and precision Better results were obtained by the use of neural networks on high dimensional datasets for pricing and they turns out to be a good approximation of financial models. All these methods can be easily extended to more complex models such as rough volatility or Levy-based models. Also, more input variables could have been incorporated such as dividends or other market data to bring additional financial knowledge. The Greeks computation can also be done by extracting the gradient information from the neural network. Besides, in this study, machine learning algorithms have been trained on CPU but the computation time cost can be highly improved using GPU which is possible with neural networks.

However, a disadvantage of the machine learning approach might be an absence of confidence interval which is an important quantity when approximating an option price with a numerical algorithm. The interpretability of neural networks might also be questioned in a risk management context.

Our objective was to accelerate the pricing or calibration using model-based synthetic data for training but the use of real market data can also be challenged in a model free setting.

References

[1] Sander M.Bohte Shuaiqiang Liu, Cornelis W. Oosterlee. Pricing options and computing implied volatilities using neural networks. 2019.

[2] Lech A. Grzelak Cornelis W. Oosterlee Shuaiqiang Liu, Anastasia Borovykh. A neural network-based framework for financial model calibration. 2019.

[3] Mehdi Tomas Blanka Horvath, Aitor Muguruza. Deep learning volatility a deep neural network perspective on pricing and calibration in (rough) volatility models. 2019.

[4] Olivier Pironneau. Calibration of heston model with keras. 2019.

[5] Sofie Reyners Wim Schoutens Jan De Spiegeleer, Dilip B. Madan. Machine learning for quantitative finance: Fast derivative pricing, hedging and fitting. 2019.

[6] Michael A. Nielsen. *Neural Network and Deep Learning*. Clarendon Press, 2015.

[7] Fang Fang and Kees Oosterlee. A novel pricing method for european options based on fourier-cosine series expansions. 2009.

[8] F. A. Longstaff and E. S. Schwartz. Valuing american options by simulation: A simple least-squares approach. 2001.

[9] Yoshua Bengio Balazs Kégl James Bergstra, Remi Bardenet. Algorithms for hyper-parameter optimization.

A propos de Coperneec

«From Revolution to Performance»

Coperneec est un cabinet de conseil cross-sectoriel spécialiste de la valorisation de la Data. Nous intervenons sur l'ensemble de la chaîne des savoir-faire autour de la Data Science, la Data Analyse et du Data Management.

Nos méthodes et techniques scientifiques éprouvées permettent de résoudre des problématiques dans tous les secteurs de l'industrie.

Notre vocation : extraire la connaissance à partir des données et pérenniser les avancées technologiques qui en découlent. La R&D est au cœur de notre ADN et les expertises de nos consultants (data scientists, data analysts, data engineers) sont en permanence challengées afin d'accompagner au plus près les révolutions technologiques et scientifiques.

Contactez-nous

Aymeric LISBONNE
Partner
alisbonne@coperneec.com
06 88 69 67 75

coperneec

est une marque de

canopee group