Data-related innovations in finance: Deep pricing project

Instructors: Samuel STEPHAN / Jiang PU

2023

Introduction

The purpose of this project is to compare classical pricing methods with new ones in terms of accuracy, robustness, and running time complexity. In the first part, you're going to implement an option valuation using the Monte Carlo method. In the second part, you will be asked to implement a deep neural network in PyTorch using the differential machine learning framework seen in the course. In the third part, you will compare the results and give your conclusions.

0.1 Evaluation

You must send a report in PDF format and a jupyter notebook before the 07th January 2024 00:00. The project must be done by groups of 5 students (please make sure that the name of all the contributors is on the final report). The project (report + notebook) must be zipped with the following name: Group_X.zip where X is the number of your group indicated in the shared excel file (link: here).

The PDF report must contain:

- The answers to all questions (including code)
- All relevant information to appreciate your work

The python notebook must contain all the codes used to answer the questions. Please make sure that all the code is useful and follows these general guidelines:

- The default random number generator is seeded by the sequence 123
- The only authorized packages are PyTorch, and matplotlib
- The code answers the questions in the right order
- The results returned by your code are consistent with the PDF report
- All unnecessary code is removed

You will be evaluated on your capacity to implement and interpret the results. Be as explicit as possible and precise to demonstrate your understanding of the problem.

Presentation of the pricing problem

Price diffusion

In this project, we suppose that the evolution of a single underlying asset S_t is given by the Heston model i.e.

$$\begin{cases} dS_t = rS_t dt + S_t\sqrt{V_t}dW_t^S \\ dV_t = \kappa(\bar{v} - V_t)dt + \gamma\sqrt{V_t}dW_t^V \\ dW_t^S dW_t^V = \rho_{S,V}dt \end{cases}$$

Where dW_t^S and dW_t^V are two independent Brownian motions A. The process V_t is known as Cox–Ingersoll–Ross (CIR) process and has been presented first to model the dynamics of interest rates [1]. It has been discussed after to model the variance in the Heston model. It follows that the Heston model is a stochastic volatility model for which there is no closed-form formula contrary to the black and Scholes model. Thus, to price options under this model, we usually need to perform Monte Carlo simulations.

We consider that this model has already been calibrated and the optimized parameters are:

- Long run average variance $\bar{v} = 0.5$
- $\kappa = 0.1$
- Vol-of-vol $\gamma = 0.1$
- Correlation of the two Brownian processes $\rho_{S,V} = -0.9$
- Risk-free interest rate $r = 0.02$
- Initial asset price $S_0 = 100$
- Initial variance $V_0 = 0.1$

From now, you will use these default parameters in all of this project. Note that the specific condition $2\kappa\bar{v} > \gamma^2$ is known as the Feller condition and ensures that the variance process cannot reach zero. In practice, this is not always true when we calibrate the model, so practitioners often use truncated schemes, reflecting schemes, or exact simulations of the CIR process by using the distribution of V_t.

Derivative instrument

Barrier options are financial derivatives whose payoffs depend on the crossing of a certain predefined barrier level by the underlying asset price process $(S_t)_t \in [0,T]$. Let the discounted payoff function be a down-and-out barrier call option defined as

$$g = \exp(-rT)\max(0, S_T - K)\mathbf{1}_{\{\min_{0\leq t\leq T} S_t > B\}}.$$

Concretely, this means that the option is exercisable if the price of the underlying has not broken the barrier. Thus, a down-and-out barrier call is a path-dependent option since if the underlying asset reaches the barrier during the option's life, then the option is terminated and will never come back into existence.

We consider that the option has the following characteristics:

- Strike $K = 100$
- Barrier $B = 90$
- Maturity $T = 1$

We want to compute the expected value of this call option that is $C = \boldsymbol{E}[g(S_t)]$ and the Greeks corresponding to the first and second derivatives of C with respect to various parameters: $\delta = \frac{\partial C}{\partial S_0}, \gamma = \frac{\partial C^2}{\partial S_0^2}, \rho = \frac{\partial C}{\partial r}, \theta = \frac{\partial C}{\partial T}$.

1 Pricing and hedging by Monte Carlo

1.1 Finite differences approach

1. Implement a standard Euler-Maruyama approximation to the Heston model. Define a function called "GeneratePathsHestonEuler()" whose arguments are:
   - The seed
   - The model parameters
   - The number of paths
   - The number of steps

   This function returns a matrix of size (NbofPaths, NbofSteps)
2. Implement a function "Payoff" that returns the payoff of the down-and-out barrier call option defined above. This function takes the option parameters and the asset prices in argument and returns the payoff of the option.
3. Implement a function "MC_Pricing()" which takes the seed, the number of paths, the number of steps, the model parameters, and the option parameters in argument and return the price of the option computed by Monte Carlo. This function must call "GeneratePathsHestonEuler()" and "Payoff() " to ensure code readability.
4. Implement three functions "DeltaFD()", "GammaFD()", "RhoFD()" which estimate the delta, the gamma, and the rho of the option by using first-order finite differences (i.e. "bumping").
5. For the three sensitivities set the number of paths to 10000 and the number of steps to 100. Plot how the variance of the estimator changes with the bump size, and comment on the reasons for this.
6. Implement a function "StandardError()" which takes the number of paths, and the payoff vector as arguments and returns the standard error of one pricing by Monte-Carlo. Plot the 95% confidence interval for different values of the number of paths. Interpret the results.
7. (Bonus) The vega is the sensibility of the option with respect to the volatility. Since the volatility is stochastic in the Heston model, propose a method for estimating the vega and justify the approach.

1.2 Automatic adjoint differentiation approach

In this section, you will benefit from the PyTorch implementation. Indeed, PyTorch gives you automatically the computation graph which makes AAD transparent for you. Therefore, you need to work with tensors instead of variables. Don't forget to add "requires_grad=True" to the parameters for which you want to compute the Greeks. Note that you may need to adapt your code of the Euler-Maruyama approximation of the Heston model to be able to backpropagate the derivatives into the computation graph. Hint: the ".clone()" method should be useful to prevent overwriting the tensors.

1. Apply the AAD approach to the computation of the Greeks and compare it to the results obtained by bumping. Note that in order to calculate the gamma, it requires remaking the computation graph for the delta.
2. (Bonus) Can you notice a strong difference for the vega computation? What do you conclude?

2 Pricing and hedging by differential deep learning

2.1 Dataset generation

In this section we are going to generate a dataset *a la* LSM [4] composed of the initial states, the payoffs, and the differentials of the initial states with respect to the payoffs.

We denote X the training samples (i.e. the initial states), Y the labels (i.e. the payoffs), and dYdX the pathwise differentials computed by AAD.

1. Implement a function "HestonLSM()" which takes the seed, the number of samples to generate, the number of paths, the number of steps, the model parameters, and the option parameters in arguments and returns the initial states, the payoffs and the differentials of the payoff wrt the inputs computed by AAD. The initial value of the asset denoted S_0 is taken into an equally spaced range from 10 to 200.
2. Implement a function "normalize_data()" which takes the X, Y, and dYdX as inputs and returns the normalized X, the mean of X, the std of X, the normalized Y, the mean of Y, the std of Y, the normalized dY_dX and the value lambda_j computed as $\lambda_j = \frac{1}{\sqrt{\frac{1}{N}\sum dy\_dx^2}}$ where dy_dx is the normalized version of $dYdX$.

   The value λ_j is called the differential weights of the cost function. Each differential with respect to input parameter j is also scaled by the average magnitude of the normalized differential with respect to j in the training set so as to let each differential have a similar magnitude in the loss function.

   Explain why it is important to work with normalized data in machine learning.

2.2 Implementation of a twin network

In this section, you will implement a twin network [3] as defined in the last lecture. This particular neural network architecture allows learning accurate pricing functions as well as sensitivities.

1. Implement a class "Twin_Network()" which inherits from the torch.nn.Module class. This way, the backward method will be inferred automatically. This deep neural network has a

feedforward architecture with 4 hidden layers of 20 neurons. These neurons are activated using the ReLu activation function. The output layer is made of one neuron which should not be activated (remember that we are in a regression problem). Define properly the "init" (number of inputs, number of hidden layers, number of neurons) and the "forward" methods to take into account this architecture. Use He initialization [2] (already implemented in PyTorch) to prevent the gradient vanishing problem.

2. Implement a method "predict_price()" of the "Twin_Network" class which takes as argument the input X to predict, the mean of X, the std of X, the mean of Y, and the std of Y. In this method, first, you normalize X, then you predict the values of Y given X, and finally, you unscale the predicted values Y.

3. Implement a method "predict_price_and_diffs()" of the "Twin_Network" class which takes as argument the input X to predict, the mean of X, the std of X, the mean of Y, the std of Y. In this method, first, you normalize X, second, you predict the values of Y given X, third, you compute the gradient of the outputs wrt the inputs (hint: you should use the torch.autograd method) and finally, you unscale the predicted values Y and dY_dX.

2.3 Neural network training

1. Implement a function "training()" which takes as argument the "Neural_Network()" class instantiated, the normalized training samples X, the normalized training labels Y, the normalized differential training labels dY_dX, lambda_j, and the number of epochs necessary to train the model.

   We want this function to be able to test two scenarios: (1) training the neural network only on the training samples (that is without the differential labels) and (2) training the neural network on the training samples and the differentials. Be careful, the loss must change according to the scenario we consider. The cost functions for both scenarios are given in the appendix B as a reminder.

   The sequential organization of this function is as follows:

   1. If we are in scenario (2) compute $alpha = \frac{1}{1+N}$, where N is the number of inputs.
   2. Define MSE as the cost function
   3. Define the Adam optimizer with a fix learning rate of 0.1. Keep the other default parameters.
   4. Enter the optimization loop: compute the predictions and the cost according to the scenario you are in, backpropagate the gradients to optimize the network weights and biases, and store the cost for each epoch.

2.4 Model comparison

In this section, we want to compare the standard deep neural network and the deep differential neural network.

1. Create a training set consisting of 1000 samples using the "HestonLSM()" function.

2. Normalize the data using the "normalize_data()" function on the generated dataset.

3. Instantiate two networks with the "Twin_Network()" class having the following architecture: 4 hidden layers consisting of 20 neurons each. Don't forget the seed.

4. Fix the number of epochs to 100. Run the classical neural network and the differential neural network. Plot in the same figure the cost of both neural networks for all epochs. Interpret this graph.

5. Can you imagine a simple procedure to estimate the uncertainty of the results given by a deep neural network? Implement it and give the results for both neural networks (hint: you are allowed to change the seed to answer this question).

6. (Bonus) Increase the number of samples generated by the HestonLSM() function for example : [1000, 3000, 5000, 8000,10000], and plot the corresponding cost at the end of training for each sample size. Interpret the results.

3 Conclusion

1. What is the interest of having "seeded" all the results?

2. What is the advantage of using a neural network to do pricing compared to a Monte-Carlo pricer?

3. Create a table that summarises the results of all approaches: for all models (FD, AAD, Neural network), report data generation time, pricing and Greeks computation time, convergence given the number of observations/ convergence given the scheme (delta time), the uncertainty. What is the best pricer? Why?

4. Gives the advantages and drawbacks of all the techniques presented above.

References

[1] Cox, J. C., Ingersoll, J. E., Ross, S. A. A theory of the term structure of interest rates. *Econometrica*, 53:385–407, 1985.

[2] He K., Zhang X., Ren S., Sun J. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. 2015.

[3] Huge, B., Savine, A. Differential machine learning. 2020.

[4] Longstaff, F. A., Schwartz, E. S. Valuing american options by simulation: a simple least-squares approach. *The review of financial studies*, 14:113–147, 2001.

A Independent Brownian motion

In this subject, it is demanded to simulate two correlated Brownian motions. Here is a general example of how you can simulate it:

Let Z_t and W_t^X be two independent Brownian motions where $Z_t \sim \mathcal{N}(0, \sqrt{dt})$ and $W_t^X \sim \mathcal{N}(0, \sqrt{dt})$.

You can then create a BM W_t^S of correlation parameter ρ with W_t^X using this simple formula:

$$W_t^S = \sqrt{1-\rho^2} Z_t + \rho W_t^X.$$

You can convince yourself that the relation is true by computing $\boldsymbol{V}[W_t^X]$, $\boldsymbol{V}[W_t^S]$ and $corr(W_t^X, W_t^S)$.

B Cost functions

The cost function C of a standard feed-forward network used for regression is the MSE. It is defined as:

$$C_{standard} = \frac{1}{M}\sum_{i=1}^{M}(y^{(i)}_{true} - y^{(i)}_{pred})^2,$$

where M is the total number of observations in the training set, $y^{(i)}_{true}$ is the true payoff value of the option and $y^{(i)}_{pred}$ its prediction by the model.

The cost function C of a differential feed-forward network used for regression is the MSE augmented with the MSE of the differentials scaled by the parameter λ_j. It is defined as:

$$C_{Differential} = \frac{\alpha}{M}\sum_{i=1}^{M}(y^{(i)}_{true} - y^{(i)}_{pred})^2 + \frac{(1-\alpha)}{M}\sum_{i=1}^{M}\sum_{j=1}^{N}\lambda_j(z^{(i)}_{true}[j] - z^{(i)}_{pred}[j])^2,$$

where $\alpha = \frac{1}{1+N}$, $\lambda_j = \frac{1}{||z^{(i)}_{true}[j]||^2}$ and $z^{(i)}_{true}[j]$ is the true (resp. predicted) differential wrt the input parameter j.