Project report: Machine Learning in Quantitative Finance - Fast Derivatives Pricing and Hedging

Giovanni La Cagnina, Francesco Pettenon, Nicolo' Taroni
Department of Computer Science, EPFL, Switzerland

***Abstract*—This project analyzes the speed-accuracy trade-off in employing Machine Learning methods for pricing vanilla call options. In particular, we exploit the Heston stochastic volatility model and compare the option pricing performance between Neural Networks, Gaussian Process regression, and Random Forests (Task A). Additionally, we investigate how Machine Learning methods can be utilized in i) the estimation of option Greeks and ii) improving the accuracy of option pricing by incorporating the option Greeks in the learning step of the pricing algorithm (Task B).**

I. Introduction

This project aims to investigate the performance of Machine Learning methods for pricing financial derivatives. Given the competitiveness of a market-making environment, the ability to speedily quote option prices consistent with an ever-changing market environment is essential. Thus, the smallest acceleration or improvement over traditional pricing methods is crucial to avoid arbitrage [1]. De Spiegeleer et al. [4] show that one can arrive at pricing speed-ups of several orders of magnitude by deploying Machine Learning techniques based on Gaussian Process Regression (GPR). However, this speed-up is obtained with a certain loss of accuracy. The essential focus of this project is the study of the speed-up accuracy trade-off for pricing vanilla European options under different models such as Gaussian Process Regression, Random Forest, and Deep Neural Networks (Task A). Additionally, the project studies how incorporating information about options Greeks, [7], could lead to improvements in the pricing accuracy for the GPR model (Task B).

A. Heston Stochastic Volatility Model

This model assumes the following risk-neutral dynamics for the underlying price and for the variance of the underlying:

$$\begin{aligned} dS_t &= (r-q)S_t dt + \sqrt{V_t} S_t dW_t^{\mathbb{Q}}, \qquad S_0 = s_0 \\ dV_t &= \kappa(\nu - V_t)dt + \sigma\sqrt{V_t} dB_t^{\mathbb{Q}}, \qquad V_0 = v_0 \\ dB_t^{\mathbb{Q}} dW_t^{\mathbb{Q}} &= \rho dt \end{aligned}$$

where r represent the risk free rate of return, q the dividend yield, κ represents the magnitude of the mean reversion, ν the long term variance of the underlying and σ the volatility of the variance process, ρ the correlation coefficient between the Brownian motion W_t and B_t and v_0, s_0 are the initial values of the processes V_t and S_t. For simplicity in this study we have considered an underlying asset that does not pay any dividend, therefore $q = 0$.

B. European Call options and pricing

Pricing a European call option at time $t = 0$ consists in calculating, under the risk-neutral measure [12], the following expectation:

$$\begin{aligned} C_0 &= \mathbb{E}^{\mathbb{Q}}[e^{-rT}\max(S_T - K^*, 0)|\mathcal{F}_0] \\ &= s_0 \mathbb{E}^{\mathbb{Q}}\left[e^{-rT}\max\left(Y_T - K, 0\right)|\mathcal{F}_0\right] \end{aligned}$$

where T represents the time to maturity, S_T the underlying price at time T, K the strike price, $Y_T = S_T/s_0$, with $Y_0 = 1$, and $K = K^*/s_0$. Therefore considering $s_0 = 1$ the pricing problem depends on the following parameters: r, σ, ν, κ, ρ, v_0, T, K.

II. Methodologies - Task A

This section explains how we produce the training datasets using simulation, the Machine Learning models considered, and our choices.

A. Simulation of Prices

This study's aim is to build a model able to learn the relation between the 8 parameters of the Heston model [5] and the price of the corresponding call option. We build datasets consisting of a feature matrix $\boldsymbol{X} \in \mathbb{R}^{N\times 8}$ where N is the number of observations. We generate each feature matrix by drawing each parameter from a given distribution with specific bounds. The distribution chosen for each parameter, together with the bounds are reported in Table I. As it is well-known in the derivative pricing field, small volatility options are more difficult to price and often they lead to numerical instability. Given this fact, we draw v_0 from an Exponential distribution to increase the number of data points having a small value for v_0. Given the X matrix, we then compute the true option prices implied by the Heston Model, $Y \in \mathbb{R}^N$, deploying the Fast Fourier Transform algorithm as shown by Carr [2]. The speed obtained with this algorithm is $5 \times 10^{-3} s$ per observation. Note that we take into account the Feller condition [5], which guarantees positive values of the variance. The initial value of the underlying price s_0 is fixed to 1, as explained before.

For the training procedure, we generate 5 different training sets with a number of observations equal to $5'000, 10'000, 20'000, 100'000, 1'000'000$.

B. Gaussian Process Regression

Let's first consider the class of Gaussian Process Regression models. A Gaussian process is a collection of random

Parameter	Range	Distribution
K	$[0.4, 1.6]$	Uniform
T	$[1/12, 1.5]$	Uniform
r	$[0.01, 0.05]$	Uniform
ν	$[0.45, 0.75]$	Uniform
κ	$[1.4, 2.6]$	Uniform
ρ	$[-0.85, -0.5]$	Uniform
σ	$[0.01, 0.02]$	Uniform
v_0	$[0.01, 0.2]$	Exponential

TABLE I: Parameters of the Heston model

variables, any finite number of which have a joint Gaussian distribution, [15]. A Gaussian process, $f(x)$, is completely determined by its mean and covariance function:

$$m(x) = \mathbb{E}(f(x))$$
$$k(x, x^*) = \mathbb{E}[(f(x) - m(x))(f(x^*) - m(x^*))^T]$$

Williams and Rasmussen [15] show that using as covariance function the Squared Exponential Kernel, [11], and mean function equal 0, the GPR predicting function corresponds to a Ridge regression model with an infinite number of basis functions. Indeed, calling $K(\cdot, \cdot)$ the kernel function, α the regularization parameter, x^* a data point of the feature space, we have:

$$\mathbb{E}\left(f\left(\boldsymbol{x}_*\right)\right) = K\left(\boldsymbol{x}_*, X\right)\left(K(X, X) + \alpha I\right)^{-1}\boldsymbol{y},$$

Regards the covariance matrix K, we use the Squared Exponential Kernel [11]. It's important to remark that this kernel depends on the length scale parameter ℓ, which is optimized using the log-likelihood. In our case, the predicted prices are the expected values of the conditional Gaussian variables as functions of the test features X_*. Moreover, we can obtain the variance of every price [4], useful to check the robustness of our predictions. Note that this represents the biggest advantage of the GPR model over Ridge regression. Even though our data is not affected by noise, we take $\alpha \neq 0$ to have better stability in the numerical algorithms used to invert the feature matrix. In any case, cross-validation suggests that we take very small α (on the order of 10^{-12}).

C. Parallel Gaussian Process Regression

To scale up Gaussian Process Regression models, and therefore being able to train this model on large datasets without having to invert large kernel matrices, we introduce the Parallel Gaussian Process Regression model. This model exploits the use of parallel computation and stacked regression concepts to efficiently scale up the Gaussian Process model. Given a training set of N observation, the training set is split into M sub-training set, and on each of these datasets, we fit a Gaussian Process Regression model. Therefore we end up with M trained models, where each model has been trained only on a $\frac{N}{M}$ observations. A final prediction is then computed weighting the models based on their accuracy on all the N observations. Given a test set of N^* observations, predicted values are computed as follows: we compute the predicted values with each fitted Gaussian Process Regression model separately, and we weigh the predicted values of each model using the weights computed during the training phase. The implementation of this model is available in the file `ParallelGPR.py`. Note that, since this model enhances the use of large datasets, it is computationally more demanding. Therefore more research can be done to find the optimal trade-off between the size of the training set and the number of models considered, given the computational power available.

D. Random Forest

A Random Forest is an estimator that fits a prespecified number of decision tree classifiers on various sub-samples of the dataset.[13] It returns the average prediction of individual trees to improve the predictive accuracy and control over-fitting. In our case, we use 50 as the number of trees in the forest. Note that the MSE cost function is used to weigh the different models.

E. Neural Networks

The well-known Universal Approximation Theorem shown by Hornik et al. [6] specifies that a Neural Network can approximate any continuous real-valued function on a compact set to an arbitrary accuracy $\epsilon > 0$, provided that specific criteria for activation functions and depth hold. In our study, the objective is to learn the function $CallHeston : (K, T, r, \nu, \kappa, \sigma, \rho, v_0) \rightarrow Price$, using a sufficient number of layers and nodes to achieve a certain degree of accuracy. For this purpose, we construct two fully-connected Neural Networks with the following characteristics:

- Input Layer dimension: 8
- Activation function: LeakyReLU
- Optimizer: Adamax, [8]
- Learning Rate: Adaptive Learning Rate, [3]
- Batch size: 100
- Epochs: 300
- Loss function: Mean Absolute Error
- Output Layer dimension: 1

Regarding networks structure, we consider 4 hidden layers with 128 nodes each, resulting in $50'817$ parameters for the Neural Network Large, and 8 hidden layers with 64 nodes each, resulting in $29'761$ parameters.

Moreover, we train both Networks using the two largest training sets, consisting of $1'000'000$ data points and $100'000$ data points. We report the result of the training phase in Table II.

F. Cost Functions

To investigate the performance of our models we utilize the following cost functions:

$$\text{MAE} = \max_{i=1,\dots,n}\left\{|f_{FFT}(i) - f_{model}(i)|\right\}$$
$$\text{AAE} = \frac{1}{n}\sum_{i=1}^{n}|f_{FFT}(i) - f_{model}(i)|$$

given n the size of the test, f_{FFT} the result generated from the closed form solution and f_{model} the result obtained using

models implemented. This choice has been made to be able to compare our results with De Spiegeleer et al. [4].

III. Results - Task A

In this section, we report the result obtained during our study. We first report the accuracy obtained for the pricing problem, together with the observed speed-up for each model considered, and, secondly, we investigate how the error is distributed for some relevant features. We present our results in Table II and in Table III. Note that we indicate the Random Forest model with RF, the Gaussian Process model with GPR, the Parallel Gaussian Process Regression with P-GPR$_{\text{N. models}}$ and the Neural Network models with NN.

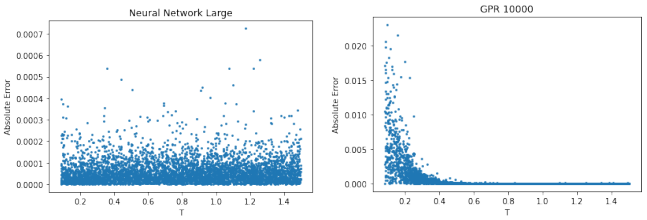
Model	N_{train}	MAE	AAE	Training Time
NN_{Large}	1'000'000	2.21e−3	5.03e−5	170m 10s
NN_{Large}	100'000	1.94e−3	1.56e−4	15m 34s
NN_{Deep}	1'000'000	1.99e−3	7.42e−5	176m 21s
NN_{Deep}	100'000	3.32e−3	2.52e−4	16 37s
GPR	5'000	4.35.0e−3	1.94e−4	1m 53s
GPR	10'000	2.72e−3	1.33e−4	11m 06s
GPR	20'000	7.51e−3	6.75e−4	100m 33s
P-GPR$_{40}$	100'000	1.17e−2	2.77e−4	63m 23s
RF	1'000'000	7.31e−3	6.63e−4	2m 36s
RF	100'000	1.0e−2	1.11e−3	0m 17s

TABLE II: Train Performance

Model	N_{train}	MAE	AAE	Speed-up
NN_{Large}	1'000'000	9.55e−4	5.28e−5	×1114.82
NN_{Large}	100'000	2.67e−3	1.71e−4	×1892.98
NN_{Deep}	1'000'000	8.82e−4	7.68e−5	×1729.48
NN_{Deep}	100'000	2.97e−3	2.65e−4	×2447.27
GPR	5'000	3.04e−2	6.96e−4	×46.25
GPR	10'000	4.57e−2	5.96e−4	×24.84
GPR	20'000	2.24e−2	5.04e−4	×7.33
$P-GPR_{40}$	100'000	1.17e−2	2.80e−4	×4.88
RF	1'000'000	1.90e−2	1.70e−3	×436.64
RF	100'000	2.02e−2	2.77e−3	×979.56

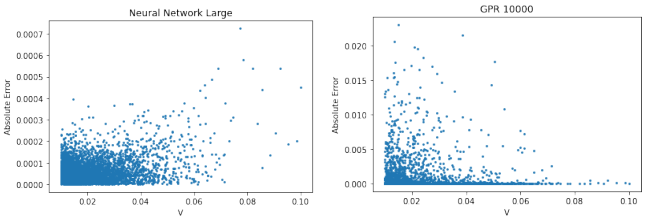
TABLE III: Test Performance

From Table III we can observe that Neural Networks Deep and Large outperform the other models considered. Indeed they achieved the best performance both in terms of AAE and MAE, and together with this, they achieve the highest speed-up. As expected, we observe that the Neural Networks trained on the largest training set available outperform the Neural networks trained on the second largest dataset. We do not observe large differences between the Neural Network Large and the Neural Network Deep: the former achieves the best AAE, while the latter achieves the best MAE. Regarding the class of Gaussian Process models, we observe that even with small-sized training sets they achieve good performance in terms of AAE, while in terms of MAE they seem to suffer high error. Indeed, we can observe in Figure 1 that Gaussian Process models seem to have larger errors for small values of T. This fact is well known in derivative pricing: short maturity options often lead to numerical instability and is more difficult to have clean prices. Notable is the fact that we do not observe larger errors for small values of T in the case of the Neural



(a) Neural Network Large (b) Gaussian Process Regression

Fig. 1: Price Absolute error vs. the maturity parameter T



(a) Neural Network Large (b) Gaussian Process Regression

Fig. 2: Price Absolute error vs. the maturity parameter v_0

Network Large model. Similar effects that we observe for small T are also observed for small values of v_0 (Figure 2), as explained in the *Simulation of Prices* section. This drawing scheme seems to have helped the Gaussian Process model to reduce the error in such area of the feature space but, at the same time, it seems to have challenged the Neural Network Large model. These two observations are a possible explanation for the MAE high values observed for Gaussian Process models. Finally, we observe that the Random Forest is the worst-performing model in terms of MAE.

IV. Methodologies - Task B

This section illustrates how it is possible to use the pricing models to obtain an estimate of the Greeks of a call option, and how introducing information about these quantities can help Gaussian Process Regression models to approximate better the pricing function. Note that in this section we consider GPR models and the Neural Network Large model.

A. Introduction to Greeks

Greeks represent the sensitivity of the call option price with respect to the characteristics of the call option and the parameters of the model. Particularly we consider:

- Delta = $\partial C/\partial S$, sensitivity w.r.t. the underlying price.
- Rho = $\partial C/\partial r$, sensitivity w.r.t. the interest rate r.
- Vega = $\partial C/\partial\sqrt{v_0}$, sensitivity w.r.t. the volatility of the underlying.
- Vega-Alt = $\partial C/\partial\sqrt{\sigma}$, sensitivity w.r.t. the volatility of the volatility process.
- Theta = $\partial C/\partial T$, sensitivity w.r.t. the time to maturity.

Note that we obtain the true value of the Greeks using closed form solution and the FFT algorithm, implemented in the Matlab function *optSensByHestonFFT*, [14], based on results shown in Carr [2].

B. Estimating the Greeks

1) Gaussian Process Regression: Williams and Rasmussen [15] show that, given the linearity of differentiation, a derivative of a Gaussian process is again a Gaussian process. Moreover, the GPR prediction for the derivative with respect to the i^{th} feature can be obtained by differentiating the Kernel, as expressed by the analytical formula:

$$\partial_{x_i^*}\mathbb{E}\left(f\left(x^*\right)\right) = \partial_{x_i^*}K_{x^*,X}\left(K(X,X)+\alpha I\right)^{-1}\boldsymbol{y}.$$

Therefore, thanks to this result we can easily obtain an estimation of the above-mentioned Greeks using the Gaussian Process model trained to price call options. Note that some of the Greeks, such as Delta, Vega, and Vega-Alt, correspond to the derivative of the call price w.r.t. a transformation of one of the features. Hence, we add some additional terms obtained with the chain rule.

2) Neural Network: To estimate the Greeks using the Neural Network model, we deploy the Finite Difference method. More specifically, we implement Forward Difference, Backward Difference, Central Difference, and Double Central Difference Kong et al. [9]. The results reported in Table IV are obtained with Double Central Difference.

C. Optimization of GPR using Greeks

It is also possible to use Greeks to enhance the optimization of the Gaussian Process Regression model. We can do so by expanding the original training set (consisting of N observations) with N more observations of the price derivative w.r.t. a specific feature. Williams and Rasmussen [15] show how to derive the kernel matrix for this particular Gaussian Process Regression model. Note that this kernel matrix depends on a unique length scale parameter, ℓ. Hence, we estimate the optimal ℓ, by maximizing the log-likelihood of this expanded Gaussian Process Regression model. Finally, we can fit a GPR on the N original data points, with parameter ℓ fixed at the optimal value to learn the pricing function. Two models are implemented: DeltaGPR and RhoGPR. The former use Delta to enhance the optimization of ℓ while the latter use Rho. Results are reported in Table V and Table VI.

V. Results - Task B

Let's first consider the estimation of the Greeks. Certain parameter combinations resulted in extremely low Greeks values that could be approximated by 0. Therefore, to correctly measure the relative percentage absolute error, we only consider Greeks observations that are above the respective 5% quantile. Results are reported in Table IV.

Greek	NN_{Large}	GPR 10000	GPR 5000
Theta	4.90%	612.54%	378.86%
Delta	1.35%	1.18%	1.35%
Rho	2.65%	4.73%	5.27%
Vega	5.95%	12.36%	25.96%
Vega-Alt	5.45%	18.82%	27.97 %

TABLE IV: Relative Percentage Absolute Error

From Table IV we can observe that the Neural Network model gives the best accuracy for the majority of the Greeks, and this can be explained by the fact that it is the model that best approximates the pricing function. The GPR model seems to be able to approximate with reasonable accuracy the majority of the Greeks. We believe that the GPR models' high errors, for low values of T, could be the cause of Theta's low accuracy.

Model	AAE	MAE	Speed-up	ℓ
GPR	1.94e−2	1.32e−3	×25.26	0.77
RhoGPR	1.86e−2	1.28e−4	×18.18	0.74
DeltaGPR	2.08e−2	7.22e−4	×18.40	0.17

TABLE V: Comparison GPR with DeltaGPR and RhoGPR. Training size = 1'000

Model	AAE	MAE	Speed-up	ℓ	α
GPR	2.95e−2	1.28e−3	×26.23	0.96	1e−6
RhoGPR	2.81e−2	1.24e−3	×14.44	0.91	1e−6
DeltaGPR	1.62e−2	1.03e−3	×13.86	0.54	1e−10

TABLE VI: Comparison GPR with DeltaGPR and RhoGPR using a validation set for the choice of the regularization parameter based on AAE cost. Training size = 1'000

From results reported in Table V, we observe that introducing the Greeks into the optimization of GPR models can lower the error in the approximation of the pricing function. Better results are found using Rho, compared to using Delta, with a fixed α. We believe that there is room for improvement and further research by optimizing together with ℓ also the regularisation parameter α. An example is given in Table VI.

VI. Implementation

All the results reported in this study are implemented using Python, and the code, together with some useful notebooks are publicly accessible and reproducible, [10]. Note that in the reported notebooks are available more results, such as Greeks volatility surfaces and further investigation plots.

VII. Conclusion

In this study, we showed how the pricing of vanilla call options can be speeded up using different Machine Learning techniques. In terms of speed up and accuracy, Feedforward Neural Networks proved to be the best model class. Promising results are observed for GPRs, given their excellent trade-off between training size and accuracy. This allows us to achieve good performance with a limited number of training observations, compared to Neural Networks. Furthermore, we analyzed how Machine Learning pricing models can be used to produce estimations of Greeks with a reasonable precision level. In summary, we believe that both Neural Networks and GPR have room for future improvements. The former through the fine-tuning procedure and its applications to market calibration, while the latter through a deeper investigation into how Greeks information can contribute to the optimization.

References

[1] David Anderson and Urban Ulrych. Accelerated american option pricing with deep neural networks. *Swiss Finance Institute Research Paper*, (22-03), 2022.

[2] Peter Carr. Option valuation using the fast fourier transform. 1999.

[3] PyTorch Contributors. Reducelronplateau, 2022. URL https://pytorch.org/docs/stable/generated/torch.optim.lr_scheduler.ReduceLROnPlateau.html.

[4] Jan De Spiegeleer, Dilip B Madan, Sofie Reyners, and Wim Schoutens. Machine learning for quantitative finance: Fast derivative pricing, hedging and fitting. *Quantitative Finance*, 18(10):1635–1643, 2018.

[5] Steven 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.

[6] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[7] John Hull. *Options, Futures, and Other Derivatives, Global Edition* . Pearson Deutschland, 2021. ISBN 9781292410654. URL https://elibrary.pearson.de/book/99.150005/9781292410623.

[8] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[9] Qingkai Kong, Timmy Siauw, and Alexandre Bayen. *Python Programming and Numerical Methods: A Guide for Engineers and Scientists*. Academic Press, 2020.

[10] Giovanni La Cagnina, Francesco Pettenon, and Nicolò Taroni. Fast pricing, 2022. URL https://github.com/djoleglc/FastPricing.

[11] Mark JL Orr et al. Introduction to radial basis function networks, 1996.

[12] Miklós Rásonyi. Arbitrage pricing theory and risk-neutral measures. *Decisions in Economics and Finance*, 27(2):109–123, 2004.

[13] Mark R Segal. Machine learning benchmarks and random forest regression. 2004.

[14] Inc. The MathWorks. Heston model, 2022. URL https://ch.mathworks.com/help/fininst/heston-model.html?s_tid=CRUX_lftnav.

[15] Christopher KI Williams and Carl Edward Rasmussen. *Gaussian processes for machine learning*, volume 2. MIT press Cambridge, MA, 2006.