Pricing and calibration with neural networks in finance

Kees Oosterlee[1]

[1]CWI, Amsterdam

BIRS workshop, Banff, 26/9/2019

Agenda

- Derivatives pricing, Feynman-Kac Theorem
- Fourier methods
  - Parameter calibration
  - Initial attempts with neural networks.
  - Overview of NN projects

Agenda

- Derivatives pricing, Feynman-Kac Theorem
- Fourier methods
    - Parameter calibration
    - Initial attempts with neural networks.
    - Overview of NN projects
- Joint work with Shuaiqiang Liu, Sander Bohte, Anastasia Borovykh

Feynman-Kac Theorem

- The linear partial differential equation:

$$\frac{\partial v(t,x)}{\partial t} + \mathcal{L}v(t,x) + g(t,x) = 0, \quad v(T,x) = h(x),$$

with operator

$$\mathcal{L}v(t,x) = \mu(x)Dv(t,x) + \frac{1}{2}\sigma^2(x)D^2v(t,x).$$

Feynman-Kac Theorem

- The linear partial differential equation:

$$\frac{\partial v(t,x)}{\partial t} + \mathcal{L}v(t,x) + g(t,x) = 0, \quad v(T,x) = h(x),$$

with operator

$$\mathcal{L}v(t,x) = \mu(x)Dv(t,x) + \frac{1}{2}\sigma^2(x)D^2v(t,x).$$

Feynman-Kac theorem:

$$v(t,x) = \mathbb{E}\left[\int_t^T g(s,X_s)ds + h(X_T)\right],$$

where X_s is the solution to the FSDE

$$dX_s = \mu(X_s)ds + \sigma(X_s)d\omega_s,\ X_t = x.$$

Feynman-Kac Theorem (option pricing context)

Given the final condition problem

$$\begin{cases} \dfrac{\partial v}{\partial t} + \dfrac{1}{2}\sigma^2 S^2 \dfrac{\partial^2 v}{\partial S^2} + rS\dfrac{\partial v}{\partial S} - rv = 0, \\ v(T,S) = h(S_T) = \text{ given} \end{cases}$$

Then the value, $v(t,S)$, is the unique solution of

$$v(t,S) = e^{-r(T-t)}\mathbb{E}^Q\{v(T,S_T)|\mathcal{F}_t\}$$

with the sum of first derivatives square integrable, and $S = S_t$ satisfies the system of SDEs:

$$dS_t = rS_t dt + \sigma S_t d\omega_t^Q,$$

- Similar relations also hold for (multi-D) SDEs and PDEs!

A pricing approach; European options

$$v(t_0, S_0) = e^{-r(T-t_0)}\mathbb{E}^Q\{h(S_T)|\mathcal{F}_0\}$$

Quadrature:

$$v(t_0, S_0) = e^{-r(T-t_0)}\int_{\mathbb{R}} h(S_T)f(S_T, S_0)dS_T$$

- Trans. PDF, $f(S_T, S_0)$, typically not available, but the characteristic function, $\widehat{f}$, often is.

Motivation Fourier Methods

- Derive pricing methods that
  - are computationally fast
  - should work as long as we have a characteristic function,

$$\widehat{f}(u;x) = \int_{-\infty}^{\infty} e^{iux} f(x) dx;$$

    (available for Lévy processes and affine SDE systems).
  - The characteristic function of a Lévy process is known by means of the celebrated Lévy-Khinchine formula.

Mathematical models for option pricing

- The Black-Scholes asset model,

$$dS_t = rS_t dt + \sqrt{\nu_t} S_t d\omega_t^s, \quad S_{t_0} = S_0,$$

- The Heston model (considering stochastic volatility),

$$\begin{aligned} dS_t &= rS_t dt + \sqrt{\nu_t} S_t d\omega_t^s, \quad S_{t_0} = S_0, \\ d\nu_t &= \kappa(\bar{\nu} - \nu_t)dt + \gamma\sqrt{\nu_t} d\omega_t^\nu, \quad \nu_{t_0} = \nu_0, \\ d\omega_t^s d\omega_t^\nu &= \rho_{x,\nu} dt, \end{aligned}$$

- The Bates model (considering price jumps),

$$\begin{aligned} \frac{dS_t}{S_t} &= \left(r - \lambda_J \mathbb{E}[e^J - 1]\right) dt + \sqrt{\nu_t} d\omega_t^x + \left(e^J - 1\right) dX_t^{\mathcal{P}}, \\ d\nu_t &= \kappa(\bar{\nu} - \nu_t)dt + \gamma\sqrt{\nu_t} d\omega_t^\nu, \quad \nu_{t_0} = \nu_0, \\ d\omega_t^s d\omega_t^\nu &= \rho_{x,\nu} dt, \end{aligned}$$

Heston option valuation PDE

- Calibrating is to fit 5 parameters, correlation coefficient ρ, long term variance $\bar{\nu}$, reversion speed κ, volatility of volatility γ, initial variance ν_0, given market option prices, v_c^{mkt}, v_p^{mkt}.
- The Heston option pricing PDE with these five parameters,

$$\begin{aligned}\frac{\partial v}{\partial t} &+ rS\frac{\partial v}{\partial S} + \kappa(\bar{\nu} - \nu)\frac{\partial v}{\partial \nu} + \frac{1}{2}\nu S^2\frac{\partial^2 v}{\partial S^2} \\ &+ \rho\gamma S\nu\frac{\partial^2 v}{\partial S \partial \nu} + \frac{1}{2}\gamma^2\nu\frac{\partial^2 v}{\partial \nu^2} - rv = 0.\end{aligned}$$

  where $v = v(t, S, \nu; K, T)$ is the option price at time t, with suitable terminal conditions.
- A European option payoff function: $v_c(T, S_T) = (S_T - K)^+$, $v_p(T, S_T) = (K - S_T)^+$, with strike price K.

Fourier-Cosine Expansions, COS Method (with Fang Fang)

- The COS method:
  - Exponential convergence;
  - Greeks (derivatives) are obtained at no additional cost.
- Based on the availability of a characteristic function.
- The basic idea:
  - Replace the density by its Fourier-cosine series expansion;
  - Coefficients have simple relation to characteristic function.

Results, Heston stochastic volatility PDE

- GPU computing: Multiple strikes for parallelism, 21 IC's.

Heston model				
	N	64	128	256
MATLAB	msec	3.850890	7.703350	15.556240
	max.abs.err	6.0991e-04	2.7601e-08	$< 10^{-14}$
GPU	msec	0.177860	0.209093	0.333786

Table 1: Maximum absolute error when pricing a vector of 21 strikes.

- Exponential convergence, Error analysis in our papers.
- Also work with wavelets instead of cosines.

Solving the inverse pricing model function

How to find implied volatility?

The inverse of the BS pricing function BS, $g_\sigma(\cdot)$, is not known in closed-form. A root-finding technique is used to solve the equation:

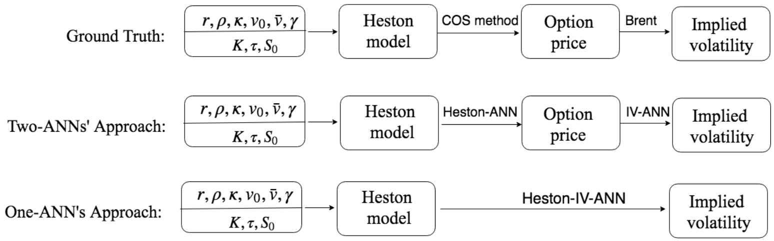
$$BS(\sigma_{impl}, r, T, K, S_0) - v_c^{mkt} = 0.$$

There are many ways to solve this equation, like "Newton-Raphson" or "Brent" iteration [1]. Since the options prices can move very quickly, it is often important to use the most efficient method when calculating implied volatilities.

[1]http://en.wikipedia.org/wiki/Brent's_method

CaNN for option pricing models

- CaNN consists of two stages, a forward pass and a backward pass. For example, Heston-CaNN:
- Forward pass:



“Neural Networks”

- Generate the sample data points for input parameters,
- Calculate the corresponding output with PDE or MC (option price or implied volatility), to form a complete set with in- and outputs,
- Split the above data set into a training and a test part,
- Train the ANN on the training data set,
- Evaluate the ANN on the test data set,
- Replace the original solver by the trained ANN in applications.

Implied volatility

- A gradient squashing technique is used to deal with a gradient in the volatilities wrt. option prices (see [Shuaiqiang et al, 2018]).
  - Obtain a **time value** by subtracting a **intrinsic value**,

$$\hat{V} = V_t^* - \max(S_t - Ke^{-r\tau}, 0)$$

  - Log-scale the intrinsic value, $\log(\hat{V}/K)$

	MSE	MAE	R^2
Input: S, K, τ, r, V/K Output: σ^*	$6.36 \cdot 10^{-4}$	$1.24 \cdot 10^{-2}$	0.97510
Input: S, K,τ, r, $\log(\tilde{V}/K)$ Output: σ^*	$1.55 \cdot 10^{-8}$	$9.73 \cdot 10^{-5}$	0.9999998

ANN-based model calibration

- Calculating IV is the most frequently executed numerical task in practice. The paper [S. Liu et al., 2019] developed a neural network solver to learn the 1D inversion of Black-Scholes.

Iterative algorithm	GPU (sec)	CPU (sec)	Robust
Newton-Raphson	19.68	23.06	No
Brent	52.08	60.67	Yes
Bi-section	337.94	390.91	Yes
IV-ANN	0.20	1.90	Yes

Table 2: The total time over 20,000 different cases. CPU (Intel i5) and GPU (Tesla P100). Robustness means no initial value is required.

Asset model calibration

- The difference between model value Q and market value Q^*,

$$J(\Theta) := \sum_{i=1}^{N} w_i ||Q(\tau_i, m_i; \Theta) - Q^*(\tau_i, m_i)|| + \bar{\lambda}||\Theta||,$$

  where Q could be either an option price or implied volatility (IV), with moneyness $m = S/K$ and time to maturity $\tau = T - t$, N the number of samples, $\bar{\lambda}$ a regularization factor.

- The objective function,

$$\underset{\Theta \in \mathbb{R}^n}{\operatorname{argmin}} J(\Theta),$$

with n the number of parameters to calibrate. For Heston, $\Theta := [\rho, \kappa, \gamma, \bar{\nu}, \nu_0]$; for Bates, $\Theta := [\rho, \kappa, \gamma, \bar{\nu}, \nu_0, \lambda_J, \mu_J, \sigma_J]$; for Black-Scholes, $\Theta := [\sigma]$;

Asset model calibration

- The inverse problem is computationally intensive, and the objective functions are often non-convex and non-linear, especially for high-dimensional model calibration.
- A fast and generic calibration framework should (at least) comprise three components, an efficient solver, a global optimizer and a parallel computing environment.

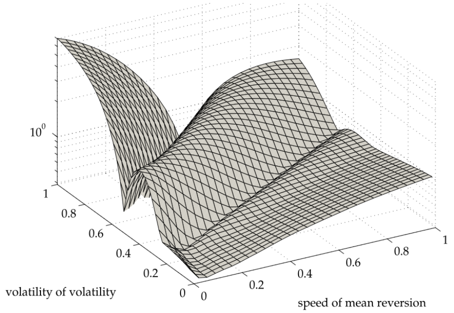
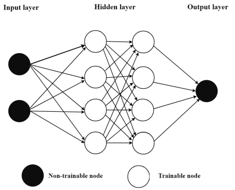


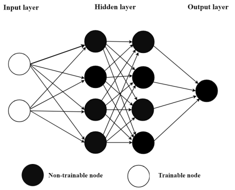
Figure 1: Multiple minima when calibrating Heston [Gilli and Schumann, 2011].

Calibration neural networks

- Training/prediction phases learn the numerical solvers, while the calibration phase inverts the trained ANN.
- The three phases are viewed as a whole, and the difference is just to change the learnable units.



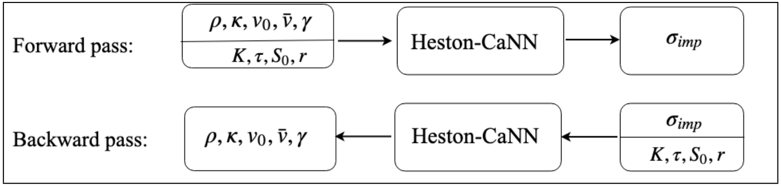
[Training phase (offline)]



[Calibration phase (online)]

CaNN for option pricing models

- Backward pass:



The forward pass

- The training data set with IV being the target quantity:

ANN	Parameters	Value Range	Method
ANN Input	Moneyness, $m = S_0/K$	[0.6, 1.4]	LHS
	Time to maturity, τ	[0.05, 3.0](year)	LHS
	Risk free rate, r	[0.0%, 5%]	LHS
	Correlation, ρ	[-0.90, 0.0]	LHS
	Reversion speed, κ	(0, 3.0]	LHS
	Volatility of volatility, γ	(0.01, 0.8]	LHS
	Long average variance, $\bar{\nu}$	(0.01, 0.5]	LHS
	Initial variance, ν_0	(0.05, 0.5]	LHS
-	European put price, v	$(0, 0.6)$	COS
ANN Output	implied volatility, σ	$(0, 0.76)$	Brent

Table 3: LHS=Latin Hypercube Sampling, COS [Fang and Oosterlee, 2008] to solve Heston, and Brent for implied vol.

- The evaluation result suggests no over-fitting.

Heston-CaNN	MSE	MAE	MAPE	R^2
Training	8.07×10^{-8}	2.15×10^{-4}	5.83×10^{-4}	0.9999936
Testing	1.23×10^{-7}	2.40×10^{-4}	7.20×10^{-4}	0.9999903

The backward pass of the CaNN

- Calibration on 35 samples (7 strike prices and 5 maturity time).
- Heston-CaNN averaged performance over 15,625 test cases.

Deviation from true Θ^*		Averaged Cost/Error	
$\lvert\nu_0^\dagger - \nu_0^*\rvert$	4.39×10^{-4}	CPU time (seconds)	0.85
$\lvert\bar{\nu}^\dagger - \bar{\nu}^*\rvert$	4.54×10^{-3}	GPU time (seconds)	0.48
$\lvert\gamma^\dagger - \gamma^*\rvert$	3.28×10^{-2}	Function evaluations	$193,249$
$\lvert\rho^\dagger - \rho^*\rvert$	4.84×10^{-2}	Data points	35
$\lvert\kappa^\dagger - \kappa^*\rvert$	4.88×10^{-2}	Calibration error $J(\Theta)$	2.52×10^{-6}

parameter	lower	upper	points	CaNN search space
ρ	-0.75	-0.25	5	[-0.85,-0.05]
$\bar{\nu}$	0.15	0.35	5	[0.05, 0.45]
γ	0.3	0.5	5	[0.05, 0.75]
ν_0	0.15	0.35	5	[0.05, 0.45]
κ	0.5	1.0	5	[0.1, 2.0]

Other ANN projects, with Anastasia Borovykh, Sander Bohte

- Conditional time series forecasting with convolutional neural networks based on Google DeepMind's WaveNet. Layers of dilated convolutions applied to the input and multiple conditions, learning trends and relations.
- Generalization in fully-connected neural networks for time series forecasting Use of input and weight Hessians, smoothness of the learned function w.r.t. the input and the width of the minimum in weight space, to quantify the ability to generalize. Control the generalization by means of the training using the learning rate, batch size and number of training iterations.

Other machine learning projects

- Mortgage pre-payment, learn to know your customers
- Estimating model uncertainty for conditional prepayment rate predictions using artificial neural networks with dropout.
- GANs, generative adverserial networks (two NNs competing, a generator and a discriminator)
- Solving PDEs with neural networks, defining loss functions etc.
- Sequential Monte Carlo method for training Neural Networks on non-stationary time series

Summary

- The problem of financial model calibration is converted into a machine learning problem.

- We need robust components (many different parameter sets)!
- The robust and generic framework CaNN rapidly reaches a global solution with ANN's inherent parallelism.

- One neural network solves two problems, e.g., the forwards pass for a numerical solution of models, the backward pass for model calibration and sensitivity analysis. i
- Training is highly efficient with the COS method

References

1. S. Liu, C.W. Oosterlee, S.M. Bohté. (2019). Pricing options and computing implied volatilities using neural networks, Risks, 7(1), 16.
2. S. Liu, A. Borovykh, L.A. Grzelak, C.W. Oosterlee (2019) A neural network-based framework for financial model calibration, arXiv:1904.10523.
3. Justin S. et al.(2018). A deep learning algorithm for solving partial differential equations, Journal of Computational Physics, Volume 375.
4. Gilli M., Schumann E. (2011). Calibrating Option Pricing Models with Heuristics. Natural Computing in Computational Finance, vol 380. Springer, Berlin, Heidelberg.
5. Jarmo Ilonen et al., (2003). Differential Evolution Training Algorithm for Feed-Forward Neural Networks, J. Neural Processing Letters, Volume 17.
6. Cybenko, G. (1989). Approximations by superpositions of sigmoidal functions, Mathematics of Control, Signals and Systems, Volume 2, Issue 4.

When we don't know the characteristic function I?

First discretize!

We can write the Euler, Milstein, and 2.0 weak Taylor discretization schemes in the following general form

$$X^{\Delta}_{m+1} = x + m(x)\Delta t + s(x)\Delta\omega_{m+1} + \kappa(x)(\Delta\omega_{m+1})^2, \quad X^{\Delta}_{m} = x.$$

For the Euler scheme:

$$m(x) = \mu(x), \quad s(x) = \sigma(x), \quad \kappa(x) = 0.$$

For the Milstein scheme:

$$m(x) = \mu(x) - \tfrac{1}{2}\sigma\sigma_x(x), \quad s(x) = \sigma(x), \quad \kappa(x) = \tfrac{1}{2}\sigma\sigma_x(x).$$

For the order 2.0 weak Taylor scheme:

$$\begin{aligned} m(x) &= \mu(x) - \tfrac{1}{2}\sigma\sigma_x(x) + \tfrac{1}{2}\left(\mu\mu_x(x) + \tfrac{1}{2}\mu_{xx}\sigma^2(x)\right)\Delta t, \\ s(x) &= \sigma(x) + \tfrac{1}{2}\left(\mu_x\sigma(x) + \mu\sigma_x(x) + \tfrac{1}{2}\sigma_{xx}\sigma^2(x)\right)\Delta t, \\ \kappa(x) &= \tfrac{1}{2}\sigma\sigma_x(x). \end{aligned}$$

Characteristic function

$$X^{\Delta}_{m+1} = x + m(x)\Delta t + \kappa(x)\left(\Delta\omega_{m+1} + \frac{1}{2}\frac{s(x)}{\kappa(x)}\right)^2 - \frac{1}{4}\frac{s^2(x)}{\kappa(x)}$$
$$\overset{d}{=} x + m(x)\Delta t - \frac{1}{4}\frac{s^2(x)}{\kappa(x)} + \kappa(x)\Delta t\left(U_{m+1} + \sqrt{\lambda(x)}\right)^2,$$

with $\lambda(x) := \frac{1}{4}\frac{s^2(x)}{\kappa^2(x)\Delta t}$, $U_{m+1} \sim \mathcal{N}(0,1)$. $(U_{m+1} + \sqrt{\lambda(x)})^2 \sim \chi_1'^2(\lambda(x))$ non-central chi-squared distributed.

The characteristic function of X^{Δ}_{m+1}, given $X^{\Delta}_m = x$

$$\widehat{f}_{X^{\Delta}_{m+1}}(u|X^{\Delta}_m = x) = \mathbb{E}\left[\exp\left(iuX^{\Delta}_{m+1}\right)\middle|X^{\Delta}_m = x\right]$$
$$= \exp\left(iux + ium(x)\Delta t - \frac{\frac{1}{2}u^2 s^2(x)\Delta t}{1-2iu\kappa(x)\Delta t}\right)(1 - 2iu\kappa(x)\Delta t)^{-1/2}.$$

When we do not know the ChF II? “Taylor expansion”

- In Pascucci’s group (U. Bologna), the adjoint expansion method for the approximation of the ChF in local Lévy models was developed;
- Taylor expansion-based formulas for the ChF possess a structure that allows for the FFT in COS for early-exercise options.

$\Rightarrow$ An efficient second-order accurate Bermudan COS pricing formula results.

- J. Math. Anal. Applic. paper with A. Borovykh, A. Pascucci: “Pricing options under local Lévy models with default”;