Pricing and calibration with neural networks in finance

Kees Oosterlee\textsuperscript{1}

\textsuperscript{1}CWI, Amsterdam

TopQuants, 28/11/2019
 Agenda

- Derivatives pricing, Feynman-Kac Theorem
- Fourier method
  - Parameter calibration
  - Attempt with neural networks.
  - Overview of NN projects
- Advertisement: New Book, with Lech Grzelak!
Derivatives pricing, **Feynman-Kac Theorem**

Fourier method
- Parameter calibration
- Attempt with neural networks.
- Overview of NN projects

Advertisement: New Book, with Lech Grzelak!

Joint work with Shuaiqiang Liu, Sander Bohte, Anastasia Borovykh, Lech Grzelak
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

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

$$dX_s = \mu(X_s)ds + \sigma(X_s)d\omega_s, \quad X_t = x.$$
Feynman-Kac Theorem (option pricing context)

Given the final condition problem

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

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, \( \hat{f} \), often is.
Motivation Fourier Methods

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

\[
\hat{f}(u; x) = \int_{-\infty}^{\infty} e^{iux} f(x)dx;
\]

(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,
  \[
  dS_t = rS_t \, dt + \sqrt{\nu_t} S_t \, d\omega_t^s, \quad S_t = S_0,
  \]

- The Heston model (considering stochastic volatility),
  \[
  dS_t = rS_t \, dt + \sqrt{\nu_t} S_t \, d\omega_t^s, \quad S_t = S_0,
  
  d\nu_t = \kappa (\bar{\nu} - \nu_t) \, dt + \gamma \sqrt{\nu_t} \, d\omega_t^\nu, \quad \nu_t = \nu_0,
  
  d\omega_t^s \, d\omega_t^\nu = \rho_{\omega^s,\omega^\nu} \, dt,
  \]

- The Bates model (considering price jumps),
  \[
  \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^p,
  
  d\nu_t = \kappa (\bar{\nu} - \nu_t) \, dt + \gamma \sqrt{\nu_t} \, d\omega_t^\nu, \quad \nu_t = \nu_0,
  
  d\omega_t^s \, d\omega_t^\nu = \rho_{\omega^s,\omega^\nu} \, dt,
  \]
Calibrating is to fit 5 parameters, correlation coefficient $\rho$, long term variance $\bar{\nu}$, reversion speed $\kappa$, volatility of volatility $\gamma$, initial variance $\nu_0$, given market option prices, $\nu_{c}^{mkt}$, $\nu_{p}^{mkt}$.

The Heston option pricing PDE with these five parameters,

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

where $\nu = \nu(t, S, \nu; K, T)$ is the option price at time $t$, with suitable terminal conditions.

A European option payoff function: $\nu_{c}(T, S_T) = (S_T - K)^+$, $\nu_{p}(T, S_T) = (K - S_T)^+$, with strike price $K$. 

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

<table>
<thead>
<tr>
<th></th>
<th>N</th>
<th>64</th>
<th>128</th>
<th>256</th>
</tr>
</thead>
<tbody>
<tr>
<td>MATLAB</td>
<td>msec</td>
<td>3.850890</td>
<td>7.703350</td>
<td>15.556240</td>
</tr>
<tr>
<td></td>
<td>max.abs.err</td>
<td>6.0991e-04</td>
<td>2.7601e-08</td>
<td>&lt; $10^{-14}$</td>
</tr>
<tr>
<td>GPU</td>
<td>msec</td>
<td>0.177860</td>
<td>0.209093</td>
<td>0.333786</td>
</tr>
</tbody>
</table>

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.
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:

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

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:

  Ground Truth: $\rho, \kappa, \nu_0, \bar{v}, \gamma \quad K, \tau, S_0, r$ → Heston model → COS method → Option price → Brent → Implied volatility
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.
A gradient squashing technique is used to deal with a steep gradient in the volatilities wrt. option prices (see [Shuaiqiang et al, 2018]).

- Obtain a **time value** by subtracting the **intrinsic value**,

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

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

<table>
<thead>
<tr>
<th>Input: (S, K, \tau, r, V/K)</th>
<th>Output: (\sigma^*)</th>
<th>MSE</th>
<th>MAE</th>
<th>(R^2)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>6.36 ( \cdot 10^{-4})</td>
<td>1.24 ( \cdot 10^{-2})</td>
<td>0.97510</td>
</tr>
<tr>
<td>Input: (S, K, \tau, r, \log(\hat{V}/K))</td>
<td>Output: (\sigma^*)</td>
<td>1.55 ( \cdot 10^{-8})</td>
<td>9.73 ( \cdot 10^{-5})</td>
<td>0.9999998</td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>Iterative algorithm</th>
<th>GPU (sec)</th>
<th>CPU (sec)</th>
<th>Robust</th>
</tr>
</thead>
<tbody>
<tr>
<td>Newton-Raphson</td>
<td>19.68</td>
<td>23.06</td>
<td>No</td>
</tr>
<tr>
<td>Brent</td>
<td>52.08</td>
<td>60.67</td>
<td>Yes</td>
</tr>
<tr>
<td>Bi-section</td>
<td>337.94</td>
<td>390.91</td>
<td>Yes</td>
</tr>
<tr>
<td>IV-ANN</td>
<td>0.20</td>
<td>1.90</td>
<td>Yes</td>
</tr>
</tbody>
</table>

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,

$$ \arg\min_{\Theta \in \mathbb{R}^n} J(\Theta), $$

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:

  Forward pass: \( \rho, \kappa, v_0, \bar{v}, \gamma \rightarrow K, \tau, S_0, r \rightarrow \text{Heston-CaNN} \rightarrow \sigma_{imp} \)

  Backward pass: \( \rho, \kappa, v_0, \bar{v}, \gamma \rightarrow \text{Heston-CaNN} \rightarrow \sigma_{imp} \rightarrow K, \tau, S_0, r \)
The forward pass

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

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Value Range</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Moneyness, $m = S_0 / K$</td>
<td>[0.6, 1.4]</td>
<td>LHS</td>
</tr>
<tr>
<td>Time to maturity, $\tau$</td>
<td>[0.05, 3.0] (year)</td>
<td>LHS</td>
</tr>
<tr>
<td>Risk free rate, $r$</td>
<td>[0.0%, 5%]</td>
<td>LHS</td>
</tr>
<tr>
<td>Correlation, $\rho$</td>
<td>[-0.90, 0.0]</td>
<td>LHS</td>
</tr>
<tr>
<td>Reversion speed, $\kappa$</td>
<td>(0, 3.0)</td>
<td>LHS</td>
</tr>
<tr>
<td>Volatility of volatility, $\gamma$</td>
<td>(0.01, 0.8)</td>
<td>LHS</td>
</tr>
<tr>
<td>Long average variance, $\bar{\nu}$</td>
<td>(0.01, 0.5)</td>
<td>LHS</td>
</tr>
<tr>
<td>Initial variance, $\nu_0$</td>
<td>(0.05, 0.5)</td>
<td>LHS</td>
</tr>
<tr>
<td>European put price, $v$</td>
<td>(0, 0.6)</td>
<td>COS</td>
</tr>
<tr>
<td>implied volatility, $\sigma$</td>
<td>(0, 0.76)</td>
<td>Brent</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Heston-CaNN</th>
<th>MSE</th>
<th>MAE</th>
<th>MAPE</th>
<th>$R^2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Training</td>
<td>$8.07 \times 10^{-8}$</td>
<td>$2.15 \times 10^{-4}$</td>
<td>$5.83 \times 10^{-4}$</td>
<td>0.99999936</td>
</tr>
<tr>
<td>Testing</td>
<td>$1.23 \times 10^{-7}$</td>
<td>$2.40 \times 10^{-4}$</td>
<td>$7.20 \times 10^{-4}$</td>
<td>0.99999903</td>
</tr>
</tbody>
</table>
Optimization algorithms within CaNN

⇒ Gradient-Based methods:

- **Stochastic Gradient Descent (SGD)** algorithm to update the weights and biases (training phase),

\[
\begin{align*}
W^{(i+1)} &\leftarrow W^{(i)} - \eta(i) \frac{\partial L}{\partial W}, \\
b^{(i+1)} &\leftarrow b^{(i)} - \eta(i) \frac{\partial L}{\partial b},
\end{align*}
\]

\(\eta\) learning rate, \(L\) loss function, \(i = 0, 1, 2, \ldots\)

- Several SGD variants, like Adam, RMSprop

⇒ Gradient-free methods: **Differential Evolution method (DE)** a global optimizer used to train ANNs [Jarmo, 2003] (calibration phase).
Differential Evolution

1 Initialization: Generate a population with \( N_p \) individuals, \((\theta_1, \theta_2, \ldots, \theta_{N_p})\);

2 Mutation: Add a randomly sampled difference to “best” individual,
\[
\theta'_{i} = \theta_a + F \cdot (\theta_b - \theta_c).
\]

3 Crossover: Filter out samples by a crossover probability, \( Cr \in [0, 1] \),
\[
\theta''_{i} = \begin{cases} 
\theta'_{i}, & \text{if } p_i \leq Cr \\
\theta_i, & \text{otherwise}
\end{cases}
\]

4 Selection: Compare each new candidate with the corresponding target individual on the objective function,
\[
\theta_i \leftarrow \begin{cases} 
\theta''_{i}, & \text{if } g(\theta''_{i}) \leq g(\theta_i) \\
\theta_i, & \text{otherwise}
\end{cases}
\]
Accelerate CaNN

- CaNN with a conventional global searcher is not fast enough!
- Develop a parallel version of DE incorporate it within ANN.
An example of acceleration in CaNN

Assuming there are 40 market data samples for calibration.

- The setting of DE,

<table>
<thead>
<tr>
<th>Parameter</th>
<th>option</th>
</tr>
</thead>
<tbody>
<tr>
<td>Population size</td>
<td>50</td>
</tr>
<tr>
<td>Strategy</td>
<td>best1bin</td>
</tr>
<tr>
<td>Mutation</td>
<td>(0.5, 1.0)</td>
</tr>
<tr>
<td>Crossover recombination</td>
<td>0.7</td>
</tr>
</tbody>
</table>

- Each generation contains $50 \times 40 = 2000$ input samples, which traditionally are computed individually.

- Speed-up: a whole population (all 2000 samples) within one generation is computed by the ANN solver simultaneously.
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.

<table>
<thead>
<tr>
<th>Deviation from true $\Theta^*$</th>
<th>Averaged Cost/Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>\nu_0^\dagger - \nu_0^*</td>
</tr>
<tr>
<td>$</td>
<td>\bar{\nu}^\dagger - \bar{\nu}^*</td>
</tr>
<tr>
<td>$</td>
<td>\gamma^\dagger - \gamma^*</td>
</tr>
<tr>
<td>$</td>
<td>\rho^\dagger - \rho^*</td>
</tr>
<tr>
<td>$</td>
<td>\kappa^\dagger - \kappa^*</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>parameter</th>
<th>lower</th>
<th>upper</th>
<th>points</th>
<th>CaNN search space</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\rho$</td>
<td>-0.75</td>
<td>-0.25</td>
<td>5</td>
<td>[-0.85,-0.05]</td>
</tr>
<tr>
<td>$\bar{\nu}$</td>
<td>0.15</td>
<td>0.35</td>
<td>5</td>
<td>[0.05, 0.45]</td>
</tr>
<tr>
<td>$\gamma$</td>
<td>0.3</td>
<td>0.5</td>
<td>5</td>
<td>[0.05, 0.75]</td>
</tr>
<tr>
<td>$\nu_0$</td>
<td>0.15</td>
<td>0.35</td>
<td>5</td>
<td>[0.05, 0.45]</td>
</tr>
<tr>
<td>$\kappa$</td>
<td>0.5</td>
<td>1.0</td>
<td>5</td>
<td>[0.1, 2.0]</td>
</tr>
</tbody>
</table>
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 adversarial 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
• Point of departure BS equation
• Many asset price models!
• Equity, interest rate, FX derivatives
• Fourier and MC methods
⇒ Including exercises, Python and Matlab codes!

https://worldscientific.com/worldscibooks/10.1142/q0236
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 forward pass for a numerical solution of models, the backward pass for model calibration and sensitivity analysis.

Training is highly efficient with the COS method.


