On using artificial neural networks for calibrating tempered stable Lévy processes to probabilities of crossing absorbing barriers

O Kudryavtsev1 and V Rodochenko2

1 Russian Customs Academy Rostov branch, Budennovskiy 20, Rostov-on-Don 344002, Russia
2 Institute for Mathematics, Mechanics, and Computer Science in the name of I.I. Vorovich (Southern Federal University), Milchakova str. 8a, Rostov-on-Don 344090, Russia
E-mail: koe@donrta.ru

Abstract. We propose a new method for calibrating tempered stable Lévy processes based on an artificial neural network (ANN), which takes probabilities of crossing a number of fixed barriers by a random walk as input data, and demonstrate its performance for the widely used CGMY model. To train the network we use real historical data and a synthetic dataset. We download and prepare the former to create a sequence of histograms with historical probabilities of crossing the set of barriers by log-returns of the underlying asset. To construct the synthetic dataset, we generate the values of the CGMY model’s parameters and calculate the respective probabilities of crossing the barriers as prices of synthetic one-touch-digital options by means of an effective numerical method, which is based on the fast Wiener-Hopf factorization technique. After that, we become able to calibrate the parameters for this model by means of the trained ANN, using the probabilities as input data. As the result, we obtain a fast method to calibrate the CGMY Lévy model, which can be used to solve risk management problems on financial markets – especially for the case where the asset under consideration is highly liquid and highly volatile at the same time (e.g. cryptocurrencies).

1. Introduction
To maintain everyday activity of financial markets, a tremendous amount of computationally expensive problems should be solved. Among them are computing trading indicators, pricing derivatives, calculating hedge coefficients, and many others. To construct effective numerical methods for solving those, a heavy use of modern instruments of probability theory and computational mathematics is required. However, the methods for calibrating models with jumps and/or stochastic volatility still require a huge amount of computational resources, because accurate derivative prices calculation is relatively slow for such models, while there is a strong need to perform those calculations constantly and repeatedly [1].

The rapid progress we observe in the areas of machine learning and artificial intelligence influences the methods researchers and engineers use in this area on a daily basis. The goal of this paper is to apply an artificial neural networks (ANN) for solving one of the classic problems of risk management – the evaluation of a probability of crossing a fixed barrier by an asset price (which essentially can be described as a random process). We restrict ourselves to the case where we can make an assumption of an adequate model, which can be used to describe the...
random walk properties, while keeping in mind that such choice is itself a non-trivial and may heavily affect the choice of suitable instruments (see e.g. [1]).

In computational finance, the problem of this kind emerges when evaluating a risk-neutral price of a certain financial derivative – an option. The most interesting case for us is the case of “one touch digital” option – a contract, which pays $1 when the price $S$ of an underlying asset crosses a known fixed barrier $K$ during the contract’s lifetime $[0, T]$, and becomes worthless otherwise. Its risk-neutral price at time 0 can be interpreted as a probability of crossing the barrier $K$ by a random walk associated with $S$ within $[0, T]$. This observation allows to evaluate synthetic prices for such options on financial markets where derivative trading is not developed enough to rely on real market trading data.

The idea that artificial neural networks can be efficiently used to solve calibration problems for models, which describes asset prices behavior, is presented, e.g. in [2]. The approach proposed is as follows. After the appropriate model has been chosen, the calibration problem essentially becomes a search for optimal model parameters, in a sense that the optimal parameters can be used to minimize the value of an error functional, which is interpreted as a distance between the historical market prices and the prices generated by the model. With respect to ANN, the idea is to train the ANN using the available historical data (and synthetic dataset, if needed), replace the existing calibration method with the trained ANN and compare the results in terms of speed and accuracy. The calibration speed ANN demonstrates can, in some cases, be faster then the speed of traditional methods by an order of magnitude [2, 3]. Being computationally efficient, this approach has its significant drawbacks, which are described in [2, 3] in more detail. Among them there are low accuracy when the training dataset is not large enough and does not represent important market patterns, the ability to generate jumps in continuous derivative pricing models, and the need to specially handle non-arbitrage conditions.

Significant attention in financial applications is given to Lévy processes, which models random walk as a combination of a linear trend, an infinite number of small magnitude jumps, and a number of rare jumps of large magnitude. One good example of those is the class of tempered stable Lévy processes (TSL) [4].

Let us recall that the Lévy process is the stochastically continuous process with independent increments (see e.g. [4] for details). It is well-known that the Lévy process $X_t$ can be completely defined by its characteristic exponent $\psi$, which can be found using the following representation of its characteristic function:

$$E[e^{i\xi X_t}] = e^{-t\psi(\xi)}.$$  

The characteristic exponent can be written using the Lévy-Khinchine formula:

$$\psi(\xi) = \frac{\sigma^2}{2} \xi^2 - i\mu \xi + \int_{-\infty}^{+\infty} (1 - e^{i\xi y} + i\xi y 1_{|y|\leq 1}) \Pi(dy),$$

where $\sigma^2$ is the dispersion of the Gaussian component, and Lévy measure $\Pi(dy)$ satisfies the condition

$$\int_{\mathbb{R}\setminus\{0\}} \min\{1, y^2\} \Pi(dy) < +\infty.$$  

If $\Pi(dx) = \pi(x) dx$, then $\pi(x)$ is a Lévy density.

The characteristic exponent for TSL processes is defined by a formula:

$$\psi(\xi) = -i\mu \xi + c_+ \Gamma(-\nu_+) [(\lambda_+ - i\xi)^{\nu_+} - (\lambda_+ + i\xi)^{\nu_+}] + c_- \Gamma(-\nu_-) [(\lambda_- - i\xi)^{\nu_-} - (\lambda_- + i\xi)^{\nu_-}],$$

where

$$\nu_+, \nu_- \in (0, 2), \nu_+ \neq 1, c_+, c_- > 0, \mu \in \mathbb{R},$$
and $\lambda_+ < -1 < 0 < \lambda_+$. In this case $\pi(x)$ is defined as:

$$
\pi(x) = c_+ e^{\lambda_+ x}|x|^{-\nu_+ - 1}1_{\{x<0\}} + c_- e^{\lambda_- x}|x|^{-\nu_- - 1}1_{\{x>0\}}.
$$

If $c_- = c_+ = c$ and $\nu_- = \nu_+ = \nu$, then we have the KoBoL (CGMY [5]) model. With the CGMY parametrization, we have $C = c, Y = \nu, G = \lambda_+, M = -\lambda_-.$

The goal of the present paper is to propose a new method for calibrating TSL processes using a dataset, which includes the history of crossing a set of fixed barriers by a random walk associated with financial asset price. The implementation is restricted to the case of CGMY model. To prepare synthetic historical data we use a technique which allows to generate CGMY model parameters and calculate the probabilities of crossing barriers as prices of one-touch-digital options in this model. It allows to avoid the commonly used approach based on simulating random walk trajectories, and significantly decrease the computational time required to prepare synthetic data. The choice of the model and the technique of input data preparation, makes the approach significantly different from other similar researches in the area (e.g. [3, 6, 7]). As the result, we obtain a calibration scheme, which allows to evaluate risks of price movements above and below certain levels. It should also be mentioned that we don’t train the ANN to predict price movements itself (in computational finance the models are mostly used to calculate derivative prices and manage the associated risks).

The rest of the paper is organized as follows. In section 2 an overview of materials and methods used is provided. Subsection 2.1 describes the architecture of the ANN we use and provides important theorems and considerations. Also, the structure of the CGMY model is discussed in more detail. Subsection 2.2 describes the mechanism of market data preparation and the technique we use to get from OHLC format the exchange provides to the histograms our ANN takes as input data. The constraints on parameter values and the motivation behind them are demonstrated. In subsection 2.3 we discuss the approach we took to generate synthetic data and the positive effects of this choice. Subsection 2.4 describes the role of activation function in more detail and provide a brief overview on activation functions used in ANN. Subsection 2.5 provides the details of training the ANN and the results of numerical experiments, along with the discussion on ANN performance. Section 3 concludes.

2. Materials and methods

2.1. Network architecture and activation function

Artificial neural networks are networks of artificial neurons [8]. The links between neurons are modeled using weights. A neuron can take certain amount of signals (represented by vector $x = [x_1, x_2, ..., x_{d_0}], x \in R^{d_0}$), which are then being multiplied by weights $\omega_j = [\omega_{1,j}, ..., \omega_{d_0,j}], \omega_{i,j} \in R^{d_0}$, which corresponds to a hidden layer $j$ and then summarized as a linear combination. After that, the activation function is applied to define the value of an output signal $o_j = y, y \in R^{d_1}$. In the absence of an activation function, the neural network can be used only for linear regression problems, which seriously affects its ability to learn complex patterns. An illustration of a structure of a simple neuron and a trivial network architecture example can be found on figures 1 and 2, respectively (the illustrations are borrowed from [3]).

From practical standpoint, it is often important that the result generated by the output layer should belong to $C^2$. Because of that, we use non-recurrent network with no loops and cycles, by analogy to the one considered in [3]. For such ANN, the Hornik theorem [9] can be applied. The theorem essentially says that the activation function must belong to the class we expect the output to belong to, and is formulated as follows:

**Theorem 1.** Let $N_{d_0,d_1}^{\sigma}$ be a set of ANNs with activation function $\sigma : R \rightarrow R$, with a dimension $d_0 \in N$ of input vector and $d_1 \in N$ for output vector. Let $F \in C^n$ and $F_{ANN} : R^{d_0} \rightarrow R$. Then if the activation function $\sigma \in C^n(R)$ and non-constant, then $N_{d_0,d_1}^{\sigma}$ approximates $F$ and its derivatives up to order $n$. 

3
Let us mention that the variety of ANN architectures used nowadays is quite wide. An overview of most commonly used ones can be found in [6, 7]. In paper [10] recurrent LSTM networks are introduced, which are successfully applied to solve regression problems and make time series prediction, for example, in [11]. In [6] a special ANN is developed to predict Lévy process parameters. Also, the methods to increase the volume of training dataset are discussed. In papers [2, 7] ANNs are used for option prices evaluation in Black-Scholes and Heston models.

Let us now consider the 4-parametric CGMY model as an example. It is assumed that the drift parameter $\mu$ is fixed by non-arbitrage conditions, and the parameter set $\{C, G, M, Y\}$ is a function of 5 parameters, $S, K, T, r$ and $P$, which have the following economical interpretation:

- $S$ – the underlying asset price,
- $K$ – European option strike price,
- $T$ – contract expiration date,
- $r$ – risk-free interest rate,
- $P$ – the option price.

A naive neural network, which may be used as a proxy for such model, would have $d_0 = 5$, $d_1 = 4$, and work as an approximator, which acts $\mathbb{R}^5 \rightarrow \mathbb{R}^4$. More accurate approximation would include some historical information on price change for the underlying asset and option prices, which, in turn, would increase the dimensionality of the input vector.

The progress in computational power and open-access availability of special libraries like Keras and Tensorflow made it possible to solve high-dimensional problems if this kind numerically in a reasonable amount of time.

### 2.2. Historical data preparation

As the data source we used the history of bitcoin (more specifically, BTC/USD rate) trades, which has been collected using 5-minutes data from GDAX platform using its API for years 2017 and 2018. There is a very convenient property for automatic analysis of this pair on GDAX, that the trades happened everyday and 24/7, except for several days in a couple of years. It allows to obtain at least some data for almost each and every time period and minimize the amount of artificial data in the historical dataset. Since there was still a some amount of minutes without trades, we took 5-minutes intervals as a base. In case there were no deals during the 5-minutes interval under consideration, we assumed that the price remained the same.

Traditionally the data is offered in $OHLC$ format (open, high, low, close), and those fields have the following interpretations:
open – the price of the first deal during the time period,
• high – the highest deal price during the time period,
• low – the lowest deal price during the time period,
• close – the price of the last deal during the time period,
• volume – the volume of all deals made during the time period.

To reduce dimensions and avoid analyzing OHLC patterns, we calculate a weighted average price for each 5-minutes period as $S_w^m = \frac{S_i \cdot V_i}{\sum_i V_i}, i = 1, 2, ..., 5, m \in N$, where among $V_i, i = 1, ..., 5$ at least one is non-zero. To further avoid the effect associated with the specific asset, we move from $S$ to a sequence of log-returns:

$$x_0 = 0, \quad x_{m+1} = \ln S_{m+1}^w - \ln S$$

Then we set a sequence of barriers $h^+_k = 10^{-2} \cdot k, k = 1, 2, ..., 30$ and $h^-_k = 10^{-2} \cdot k, k = 1, 2, ..., 30$, and start registering the events of crossing those by log-returns within 1 trading day. If the barrier, which is further from zero, gets crossed, then all barriers between zero and this barrier are also considered crossed. This way we are able to construct a histogram of historical (statistical) probabilities of crossing barriers using the collected data. Due to the fact that those probabilities can be interpreted as prices of one touch digital options at time 0, we also automatically get the estimations for the values of a set of synthetic one touch digital options on this asset, with strike prices corresponding to the respective barriers. One example of such histogram is provided on figure 3.

![Distribution histogram sample, may 2019](image)

**Figure 3.** Input data (histogram) sample.

The problem of CGMY model parameters calibration is then solved as follows. For a given arbitrarily chosen set of parameters we calculate the values of one touch digital options for all strikes $K = S \exp(h^+_k)$. Then we get the value of error function, which we set to be a mean square distance between the prices obtained and the synthetic one touch digital option prices derived...
from the market data. After that, we use the Nelder-Mead algorithm with randomly chosen simplex to minimize the value of the error function. To lower the chance that the algorithm gets stuck in local minima, we repeat our calculation for a number of different starting values of parameters picked on a Sobol cube. Being sub-optimal in a sense of computational performance, this scheme, however, allows to cover all range of parameters values pretty well and approach the global minimum of the error function. In the latter part of the paper we will refer to this calibration mechanism as “traditional”, for brevity.

In paper [12] we conducted an analysis of cryptocurrency market in the framework of the CGMY model. We were able to reduce the range of parameters within the range of trading days under consideration to the following:

- \( \nu \in (0.5, 1.4) \),
- \( \lambda_+ \in (2, 100) \),
- \( \lambda_- \in (2, 100) \),
- \( c \in (0.1, 5) \).

The calibration results using a traditional method also has shown that the values of the CGMY model are within the specified range.

To reduce the amount of computations and let Nelder-Mead algorithm work within the specified range, the loss function can be penalized by the following constraints:

\[
L_\nu = L_\nu + L_{\lambda_+} + L_{\lambda_-} + L_c,
\]

where

\[
L_\nu = \begin{cases} 
|\nu - \nu_l|, & \nu < \nu_l; \\
|\nu - \nu_r|, & \nu < \nu_r; \\
0, & \text{otherwise};
\end{cases}
\]

\[
L_{\lambda_+} = \begin{cases} 
|\lambda_+ - \lambda_{+l}|, & \lambda_+ < \lambda_{+l}; \\
|\lambda_+ - \lambda_{+r}|, & \lambda_+ < \lambda_{+r}; \\
0, & \text{otherwise};
\end{cases}
\]

\[
L_{\lambda_-} = \begin{cases} 
|\lambda_- - \lambda_{-l}|, & \lambda_- < \lambda_{-l}; \\
|\lambda_- - \lambda_{-r}|, & \lambda_- < \lambda_{-r}; \\
0, & \text{otherwise};
\end{cases}
\]

\[
L_c = \begin{cases} 
|c - c_l|, & c < c_l; \\
|c - c_r|, & c < c_r; \\
0, & \text{otherwise};
\end{cases}
\]

and the bounds \( \nu_l, \nu_r, \lambda_{+l}, \lambda_{+r}, \lambda_{-l}, \lambda_{-r}, c_l, c_r \) and \( c_l, c_r \) are chosen from market observations. This allows to speed up convergence speed and keep the number of iterations it takes to reach the global minimum on low.

2.3. Synthetic data

The maximum amount of data we can achieve using the approach described above is about 700 daily histogram samples (and about 500 if we restrict ourselves to days where there sufficiently large number of deals took place). To train an ANN we should be able to retrieve sufficiently larger data set.

To increase a number of samples it is natural to use synthetic data. Unlike, for example, in [6], we don’t need to generate the trajectories of a random walk using the model parameters.
Instead, we can directly pick sufficiently large number of parameter values on a Sobol cube in $d_1$-dimensional space and calculate the probabilities of barrier crossing events as prices of one touch digital options using the efficient Wiener-Hopf factorization method proposed in [13]. This approach can significantly reduce the time required to generate data, as well as memory consumption, while maintaining the same volume of a data set. We put the number of samples to 30000, which made it possible to train the neural network.

There is a common practice in machine learning, which is known as cross-validation. The technique is developed to minimize the influence of training dataset to overall model performance, The idea is to split the original dataset into 3 subsets, which are then used as a training, testing and validation datasets, respectively. The training dataset is exposed to the network and the learning algorithm iterates through the data during multiple epochs to fit to the features presented. At this point we end up with a trained model and we can measure its performance against the data the network was trained on (in-sample performance). Then, when the values of error function approaches some satisfactory level, the performance of the model is evaluated against the testing data set. At this point, hyperparameters can be adjusted. Then the performance is measured on a validation data set, to control overfitting, which may occur because of hyperparameters setup (out-of-sample performance).

2.4. The choice of the activation function

As it has been pointed up earlier, in the absence of activation function an ANN’s ability to learn complex patterns is seriously decreased. In paper [7] an overview if the most frequently used activation functions is provided. Among them there are Relu:

$$\phi(x) = \max(x, 0),$$

the Sigmoid function:

$$\phi(x) = \frac{1}{1 + e^{-x}},$$

and Leaky Relu:

$$\phi(x) = \max(x, \alpha x), \quad 0 < \alpha < 1,$$

where $\alpha$ is a hyperparameter. Another good option can be ELU

$$R(z) = \begin{cases} 
z, & z > 0, \\
\alpha(e^z - 1), & z \leq 0,
\end{cases}$$

with $\alpha$ as a hyperparameter – in particular, because if has a zero-mean distribution which speeds up learning. Nevertheless, its first derivative is non-continuous in zero, and, being implemented in the output layer it does not guarantee to generate positive values.

To avoid this problem, in [3] a modified function MELU is introduced:

$$R(z) = \begin{cases} 
\frac{z^2 + az}{z + b}, & z > 0, \\
\alpha(e^z - 1), & z \leq 0,
\end{cases} \quad a = 1 - 2a, \quad b = -2 + \frac{1}{\alpha}.$$  

We use Relu, Leaky Relu and MELU in the hidden layers of our network, and set up hyperparameter values experimentally.
2.5. Training the network

The number of hidden layers, (and the values of $\alpha$ variable in activation functions) are referred to as hyperparameters – the variables, whose values are set by a researcher. Following the framework introduced in [6, 3, 7] we included 4 hidden layers, put $\alpha = 0.5$ and set activation functions in the following order:

- Relu,
- Melu, $\alpha = 0.5$,
- Melu, $\alpha = 0.5$,
- Leaky Relu, $\alpha = 10^{-3}$.

Then for 7 epochs we trained the ANN on the training dataset using the RMSprop stochastic gradient descent algorithm from Keras library. We used the generated CGMY parameters values as benchmarks for synthetic data, whereas for the market data we used the ones obtained by means of a traditional calibration.

We defined the error function as the normalized mean square distance between the values of the CGMY model’s parameters predicted by the ANN and the values of the same parameters calculated using the traditional calibration scheme (for the market data) or the parameters we generated (for the synthetic data). Figures 4 and 5 demonstrates the comparison between the error values obtained by means of traditional calibration and the results obtained using a trained ANN in 100 samples of test and validation data sets, respectively.

![Figure 4. In-sample error.](image)

The results of numerical experiments shows that, for the testing and validation data sets we had, the method proposed can significantly decrease the computational time required to perform a calibration. The mean time of a traditional calibration was 870.010 seconds per sample, whereas for the ANN it took 1.383 seconds on average to generate a set of parameters (the numerical experiment has been conducted for OS Windows 10 64-bit, Intel Core i5-3470 CPU, 8 GB RAM). The error values does not exceed 5 % in-sample and 6 % out-sample, which can be interpreted as a good error level and demonstrates that the effects of overfitting are
almost negligible. To further increase of the quality of the calibration we can run an exhaustive calibration over hyperparameters and try to increase the amount of data feed at a time.

3. Results and discussions

The calibration of the CGMY model can be performed fast and without a significant decrease of accuracy, by means of a trained artificial neural network. The average speed of the method can be faster than the speed of traditional calibration schemes by orders of magnitude, while maintaining comparable accuracy. The resulting model can be used then to evaluate the risks of crossing a barrier by a random process. A trained network demonstrates good performance both on test (in-sample) and validation (out-sample) datasets. The approach presented can be used in financial applications – in particular, in cryptocurrency options evaluation and in derivative pricing for other highly volatile and highly liquid financial assets.

To increase the volume of a training dataset, it is convenient to generate the values of the CGMY model and build distribution histograms for historical probabilities of events of crossing barriers by a random process, which is associated with the behavior of a financial asset. There is an important fact that such probabilities can be interpreted as prices of synthetic one-touch-digital options. It allows to use the fast Wiener-Hopf factorization method presented in [13] to calculate those from parameter values and avoid additional generation of random walk trajectories, which, in turn, can decrease the amount of computational time required to generate the training set. However one should keep in mind that to achieve good accuracy it is absolutely necessary to simulate all possible ranges of parameter values within the validation dataset – the performance becomes worse as we are getting close to the boundaries of the range of parameters of the training set.

The method proposed can be used to solve a vast variety of problems, which can be associated with analysis of probabilities of crossing a set of barriers by a suitable random walk.

Acknowledgments

The reported study was funded by RFBR according to the research project No 18-01-00910.
References

[1] De Spiegeleer J, Madan D B, Reynolds S and Schoutens W 2018 Machine Learning for Quantitative Finance: Fast Derivative Pricing, Hedging and Fitting Quantitative Finance 18 (10) 1–9
[2] Horvath B, Muguruza A and Tomas M 2019 Deep learning volatility a deep neural network perspective on pricing and calibration in (rough) volatility models SSRN Electronic Journal 3322085
[3] Itkin A 2019 Deep learning calibration of option pricing models: some pitfalls and solutions Preprint arXiv:1906.03507v1
[4] Cont R and Tankov P 2008 Financial modelling with jump processes (Boca Raton: Chapman & Hall/CRC Financial Mathematics Series)
[5] Carr P, Geman H, Madan D and Yor M 2002 The fine structure of asset returns: An empirical investigation The Journal of Business 75 (2) 305–33
[6] Huh J 2019 Pricing Options with Exponential Lévy Neural Network Expert Systems with Applications 127
[7] Liu S, Borovykh A, Grzelak L and Oosterlee C W 2019 A neural network-based framework for financial model calibration Journal of Mathematics in Industry 9(1):9
[8] Bishop C M 1995 Neural Networks for Pattern Recognition (New York: Oxford University Press Inc)
[9] Hornik K, Stinchcombe M and White H 1990 Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks Neural Networks 3 551—60
[10] Schmidhuber J and Hochreiter S 1997 Long short-term memory Neural Computation 9 (8) 1735–80
[11] Kim H Y and Kim T 2019 Forecasting stock prices with a feature fusion LSTM-CNN model using different representations of the same data PLoS ONE 14 (2)
[12] Kudryavtsev O and Grechko A 2018 Statistical methods for cryptocurrency models calibration Accounting and Statistics 52(4) 67–76. (in Russian)
[13] Kudryavtsev O 2016 Advantages of the Laplace transform approach in pricing first touch digital options in Lévy-driven models Boletín de la Sociedad Matemática Mexicana 22 (2) 711–31