Deep learning calibration of option pricing models: some pitfalls and solutions

Andrey Itkin considers a classical problem of mathematical finance: the calibration of option pricing models to market data. He highlights some pitfalls in the existing approaches and proposes resolutions that improve both performance and the accuracy of calibration. Itkin also addresses the problem of no-arbitrage pricing when using a trained neural net, which is currently ignored in the literature

Recently, a classical problem of mathematical finance – the calibration of option pricing models to market data – has captured the attention of the financial community in the context of deep learning (DL) and artificial neural networks (ANNs).[1] Among many relevant papers, we mention those by Horvath *et al* (2019) and Liu *et al* (2019), and the references cited therein. The idea of applying DL to solving this problem is inspired by the fact that computing model option prices can be slow, meaning the whole calibration is slow. Instead, the DL calibration assumes this problem can be solved in two steps. The first step is to replace a slow pricer with its approximator using an ANN. This ANN is trained using an in-sample data set, so the weights of the ANN become known after this step is complete. A standard calibration is then used in which the model pricer is replaced with the trained ANN constructed during the previous step. A typical method for constructing and training the ANN is described in detail in Horvath *et al* (2019). For the general theory of using DL for asset pricing, see, for example, the extensive presentation in Elouerkhaoui (2019) as well as the references cited therein.

As well as these technical details (which are certainly very important), the approach advocated in the referenced literature has some internal pitfalls we discuss further in this article. First, because option Greeks are as important as the prices themselves, one needs to make sure that the ANN prices are at least twice differentiable. Second, the ANN prices cannot, by default, guarantee no-arbitrage, neither in-sample nor out-of-sample. Finally, in our opinion the second step of the calibration process (running the global optimiser) could be eliminated, as the same result can be achieved by training the ANN at the first step.

Option pricing using an ANN

In this section we give a short overview of how ANNs are used for option pricing. An ANN is a network of artificial neurons. The connections between the neurons are modelled as weights. All inputs are modified by a weight and summed. This activity is referred to as a linear combination or a transfer function. Finally, an activation function controls the amplitude of the output. Further, we consider only feed-forward ANNs, which are artificial ANNs in which connections between the nodes do not form a cycle. As such, they are different from recurrent neural networks. A simple scheme of a one-layer feed-forward ANN is presented in figure 1, and detailed neuron behaviour is highlighted in figure 2, which is borrowed from Changhau (2017). Here, $x = [x_1, \dots, x_n]$, $x \in \mathbb{R}^n$, are the ANN inputs, $w_j = [w_{i,j}, \dots, w_{1,j}, \dots, w_{n,j}]$, $w_{i,j} \in \mathbb{R}^m$, are the network weights corresponding to the hidden layer j that connect nodes $1, \dots, n$ to the next layer, and $o_j = y$ is the output. The function Σ is often chosen in the form:

$$\Sigma = b_j + \sum_{k=1}^{n} w_{k,j} x_k$$

where $b_j \in \mathbb{R}$ denotes the bias term. A wide range of activation functions are used in practice (see, for example, Changhau 2017).

Suppose we have some function $y = F(x)$ and need to construct a feed-forward ANN, where $y = F_{\text{ann}}(x, w)$ approximates this function in some optimal sense. To do this, the ANN should use a set of input data created by running an optimisation. This optimisation takes as an input a sequence of training examples $(x_1, y_1), \dots, (x_m, y_m)$, $m \in \mathbb{Z}$, and produces a sequence of weights $w_{i,j}$ that are optimal in the sense that for the given training they minimise the difference $\sum_i \|F(x_i) - F_{\text{ann}}(x_i)\|$ taken under some appropriate norm, eg, L^2. We now want to apply this ANN approximation to the option pricing problem. Consider a financial model $\mathcal{M}$ with parameters $p_1, \dots, p_n$, $p_i \in \mathbb{R}$, $n \in \mathbb{Z}$, $n \geq 1$, which provides prices for some financial instruments given the input data $\theta_1, \dots, \theta_l$, $l \in \mathbb{Z}$, $l \geq 1$. For example, the celebrated Black-Scholes model has one model parameter σ, the volatility of the stock, and five input parameters: S, K, T, r, q, which are the stock price, the strike, the time to maturity, the free interest rate and the continuous dividend, respectively. The DL approach for option pricing basically assumes the ANN can be used as a universal approximator $\mathbb{R}^{n+l} \to \mathbb{R}$, ie, given a vector of the input data $\boldsymbol{\theta}$ and a vector of the values of the model parameters $\boldsymbol{p}$, it provides a unique option price, eg, the call option price $C(\boldsymbol{\theta}, \boldsymbol{p})$.

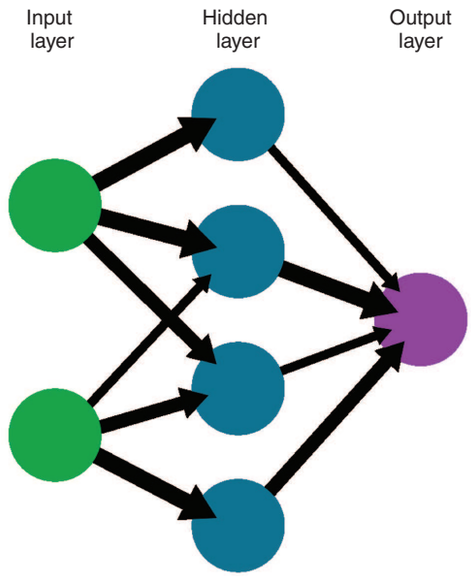
It makes sense to combine the market inputs and the parameters of the model into a single vector $\boldsymbol{\xi} = [\boldsymbol{\theta}, \boldsymbol{p}]$, and to train the ANN as $\boldsymbol{\xi} \to C_{\text{ann}}(\boldsymbol{\xi})$. This requires more intensive computational work but eliminates any retraining. Hence, the ANN pricer would need to be trained once and once only. Thus, in 'old school language', the ANN pricer could be treated as a tensor lookup table created by using nonlinear multi-dimensional regressions.

ANNs and option Greeks

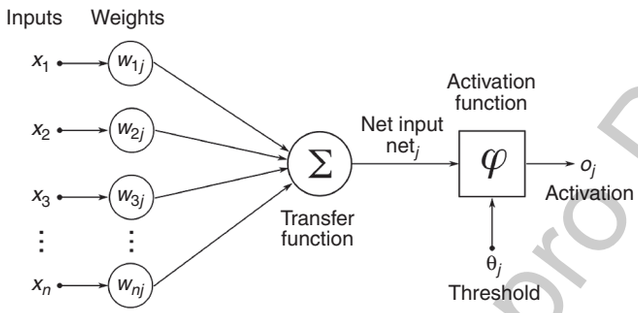
In addition to option prices, traders and front office and risk management quants also need option Greeks. Therefore, the ANN prices must be at least $\mathcal{C}^1$, or, better, $\mathcal{C}^2$. From this perspective, Horvath *et al* (2019) cite an important theorem of Hornik.

[1] *As has been mentioned by Igor Halperin, it is overkill to use non-parametric methods (ANNs) to calibrate parametric models. Indeed, if one resorts to ANNs, why deal with a parametric model in the first place, since if one is given the market data, the ANN could be trained directly to the data with no model in between (Halperin 2017).*

1 An example of a one-layer feed-forward ANN



2 The behaviour of the neuron and the activation function



THEOREM 1 (Hornik *et al* 1990) *Let $\mathcal{N}^{\sigma}_{d_0,d_1}$ be the set of ANNs with activation function $\sigma : \mathbb{R} \mapsto \mathbb{R}$, input dimension $d_0 \in \mathbb{N}$ and output dimension $d_1 \in \mathbb{N}$. Let $F \in \mathcal{C}^n$ and let $F_{\text{ann}} : \mathbb{R}^{d_0} \to \mathbb{R}$. Then, if the (non-constant) activation function is $\sigma \in \mathcal{C}^n(\mathbb{R})$, $\mathcal{N}^{\sigma}_{d_0,d_1}$ arbitrarily approximates F and all its derivatives up to order n.*

Differentiability of the activation function is therefore important for option pricing.

If the activation function is not applied at all, the output signal becomes a simple linear function. Linear functions are only single-grade polynomials, therefore a non-activated neural network will act as a linear regression with limited learning power. As ANNs are designed as universal function approximators, they are intended to learn any function. Thanks to non-linear activation functions, greater learning in networks can be achieved. However, there is no activation function which both has good properties and for which the first two derivatives exist. For instance, the rectified linear activation unit (ReLu) could be a good candidate, but it suffers from the 'dying problem' in neutral networks. One could decide to use the leaky rectified linear activation unit (LeakyReLU) as a solution, but it is not $\mathcal{C}^1$ at the origin. The choice of the activation function could therefore be critical. From this perspective, the exponential linear unit (ELU) activation function:

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

with the hyperparameter $\alpha = 1$ could be a good choice, so the first derivative of the ELU is smooth. However, the second derivative jumps at $z = 0$. This function has two additional important properties: (i) it produces a zero-centred distribution, which can make training faster; and (ii) it provides one-sided saturation, which leads to better convergence (see, for example, Kathuria 2018). However, if the output of the last ANN layer is the option price, the ELU cannot be used as it does not guarantee the positivity of the price. Therefore, one either has to use another activation function for the last layer or one has to scale the price, so the scaled price could become negative.

We therefore propose the following activation functions for the ANN built of four layers:

(1) LeakyReLU with $\alpha = 1$. This function is then $\mathcal{C}^2$.

(2) A custom activation function that is a modified ELU (MELU):

$$R(z) = \begin{cases} (\frac{1}{2}z^2 + az)/(z + b), & z > 0 \\ \alpha(\mathrm{e}^z - 1), & z \leqslant 0 \end{cases}$$

$$a = 1 - 2\alpha, \quad b = -2 + (1/\alpha)$$

It can be verified that $R(z) \in \mathcal{C}^2$ and $R'(0) = R''(0) = \alpha$.

(3) The same as in (2).

(4) The softplus function minus 0.5. This function is also $\mathcal{C}^2$.

All the activation functions in this ANN are therefore $\mathcal{C}^2$, so the entire ANN approximation is $\mathcal{C}^2$.

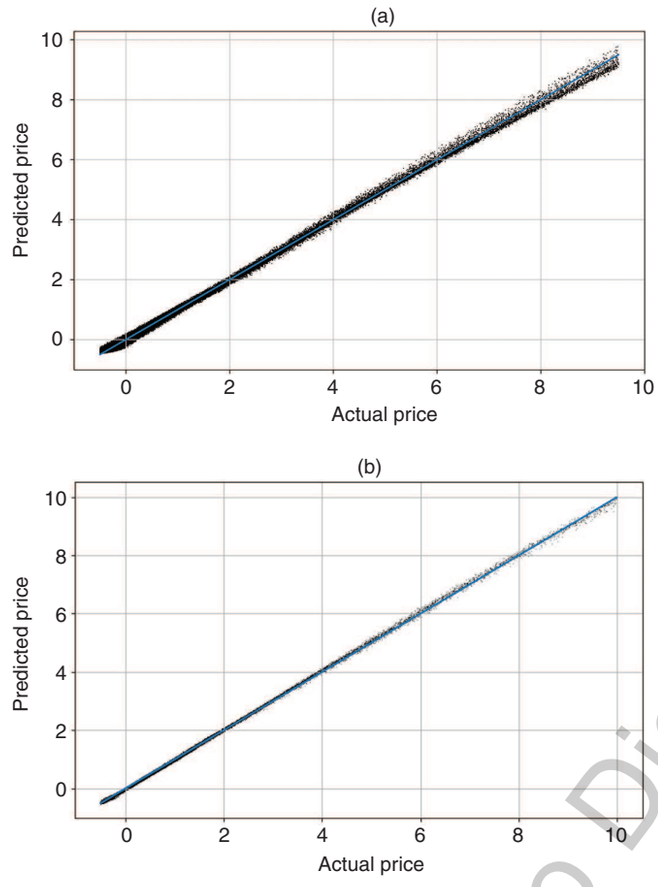
The results of a test where such an ANN is trained to the Black-Scholes call option prices are presented in figure 3 for in-sample and out-of-sample inputs. The plots of the density of the difference $\delta = y - y_{\text{ann}}$ can also be found in Itkin (2019). In this test we choose $\alpha = 0.49$, and therefore $\text{MELU}(x) > -0.5$. Accordingly, we scale the inputs: $S \mapsto S/K$, $C \mapsto C/K - \min_{\boldsymbol{\xi}}(C/K) - 0.5$.[2] We generated 300,000 random vectors $\boldsymbol{\xi} = [S, K, T, r, q, \sigma]$, computed the option prices and then left only those that obey $0.001 \leqslant C \leqslant 10$, which provided 259,012 samples. Eighty percent of them were used as a training set and the remaining 20% as a prediction set. We built a feed-forward ANN with 128 nodes and four layers, so the total number of trainable parameters was 33,921. For training the ANN we used either Rmsprop or the Adam optimiser with the mean square error (MSE) loss function, 15 epochs and batch size 64. The code ran on a PC with an Intel i7-4790 CPU (8 Cores, 3.6 GHz), and one epoch took about 15 seconds. The MSE both in-sample and out-of-sample is 9.7 basis points, and the mean percentage error is 0.1 after 15 epochs.

If the predicted price exactly matches the actual price, all scattering points on the price graphs should coincide with the line $y(x) = x$, which is also plotted in the figures as a solid line. It seems like slightly larger differences occur at larger prices, but the percentage difference at high prices is similar to the difference at lower prices.

Various packages, eg, tensorflow, also make it possible to obtain derivatives of the option prices with respect to all input and model parameters in $\boldsymbol{\xi}$. Therefore, option Greeks of the first and second order are available

[2] *We underline that since the scaled price C/K is shifted by -0.5 in the last layer, the scaled option prices in figure 3 start from -0.5.*

3 (a) In-sample and (b) out-of-sample ANN Black-Scholes scaled call option price versus the Black-Scholes scaled call option price



on the ANN, and they are continuous according to the construction of the activation functions.

ANN option prices and no-arbitrage

It is known from option pricing theory (see Itkin & Lipton (2018) and references therein) that the necessary and sufficient conditions for the call option prices $C = C(S, K, T, r, q)$ to be arbitrage-free are $C \geqslant 0$, $\partial C/\partial T > 0$, $\partial C/\partial K < 0$, $\partial^2 C/\partial K^2 > 0$. Given three call option prices $C(K_1)$, $C(K_2)$, $C(K_3)$ for three strikes $K_1 < K_2 < K_3$, and with all other parameters being the same, a discrete version of these conditions reads:

$$C(K_3) > 0, \quad C(T_2) > C(T_1), \quad C(K_2) > C(K_3) \qquad (1)$$
$$(K_3 - K_2)C(K_1) - (K_3 - K_1)C(K_2) + (K_2 - K_1)C(K_3) > 0$$

The second condition represents a so-called calendar spread, the third one a vertical spread, and the last one a butterfly spread.

Suppose that the call option prices are computed using the trained ANN, similar to how this is done in the previous section. By default we cannot guarantee that the conditions in (1) are satisfied (although if the ANN was well trained they could be almost, or even fully, satisfied). This is because when the ANN is trained, the optimiser knows nothing about these conditions. Thus, if we want the in-sample ANN prices to obey the no-arbitrage conditions, we need to replace unconditional optimisation with its conditional version. This is similar, for instance, to how a no-arbitrage parametric implied volatility surface is constructed by running a global constrained optimisation (see, for example, Itkin 2015).

As applied to the ANN, this approach can be implemented as follows. Any constraints (including the no-arbitrage constraints) can be imposed as soft constraints by adding new (penalty) terms to the loss (objective) function that is minimised during training (see Smith & Coit (1996) for a general introduction to penalty functions). It is known that this approach has two drawbacks. First, it makes it necessary to carefully choose the relative importance of the different terms in the loss function in order for the optimiser to converge. Sometimes this might not be easy. Second, in contrast to hard constraints, soft constraints do not fully guarantee that the corresponding hard constraints are satisfied, only to some degree of accuracy.

However, as Marquez-Neila *et al* (2017) surprisingly observed, imposing soft constraints instead of hard ones yields even better results while being far less computationally demanding when applied in the framework of DL. Therefore, in this article we discuss only the soft constraints, while further investigation into this topic would definitely be encouraged.

With this argument in mind, suppose that at every layer j, training of the ANN weights w_j – where w_j is the set of the node weights for the layer j in the ANN – is provided by using the MSE loss function:

$$L = \arg\min_{w} \sum_i \sum_j [C(\boldsymbol{\xi}_i) - C_{\text{ann}}(\boldsymbol{\xi}_i, w_j)]^2 \qquad (2)$$

This loss function can be penalised by the following normalised constraints:

$$\left.\begin{aligned} L_c &= \arg\min_{w} \sum_i \sum_j [C(\boldsymbol{\xi}_i) - C_{\text{ann}}(\boldsymbol{\xi}_i, w_{i,j})]^2 + \mathcal{P} \\ \mathcal{P} &= \sum_i \sum_j \left\{ \Phi_{\lambda_1,m_1}\left(- K^2 \frac{\partial^2 C_{\text{ann}}(\boldsymbol{\xi}_i, w_{i,j})}{\partial K^2} \right) \right. \\ &\quad + \Phi_{\lambda_2,m_2}\left(- T \frac{\partial C_{\text{ann}}(\boldsymbol{\xi}_i, w_{i,j})}{\partial T} \right) \\ &\quad \left. + \Phi_{\lambda_3,m_3}\left(K \frac{\partial C_{\text{ann}}(\boldsymbol{\xi}_i, w_{i,j})}{\partial K} \right) \right\} \\ \Phi_{\lambda,m}(x) &= \begin{cases} 0, & x < 0 \\ \lambda x^m, & x \geqslant 0 \end{cases} \end{aligned}\right\} \qquad (3)$$

where $T, K \in \boldsymbol{\xi}_i$, and $\lambda \in \mathbb{R}$, $m \in \mathbb{Z}$ are some positive constants to be appropriately chosen. These constants control how strongly the constraint will be enforced. The penalty functions modify the original objective function, so that if any inequality constraint is violated, a penalty is invoked. And if all constraints are satisfied, there is no penalty.

By construction, the penalty function $\Phi_{\lambda,m}(x)$ is $m - 1$ times differentiable. Therefore, choosing $m > 2$ would be helpful as it should not cause any trouble for the optimisation algorithm, which relies on the first or second derivatives. Obviously, if the constants λ are too large, the optimisation problem transforms to minimising the constraints while the loss function is almost ignored. The opposite case is when these constants are small. In that case they have to be treated carefully, so as not to make the penalty terms either negligible or too significant.

Technically, this can be implemented as a custom loss function, in Keras for example, which provides an appropriate interface. The necessary gradients could also be obtained inside this custom loss function. Therefore, implementation of L_c defined in (3) is fully supported.

A. Penalty term $\mathcal{P}_{1,0}$ when predicting call option prices using the ANN with soft constraints

λ_1	λ_2	λ_3	In-sample	MSE (bp)	Mean percentage error	Out-of-sample	MSE (bp)
0	0	0	1,008,550	6.50	0.091	248 432	6.48
1	1	1	421	10.32	0.115	101	10.38
10	10	10	105	11.95	0.124	25	12.15
50	50	50	36	12.17	0.125	9	12.34
100	100	100	25	11.26	0.120	6	11.43

To test this approach we again use the example considered in the previous section. As a measure of arbitrage we compute the entire penalty term $\mathcal{P}$ in (3) assuming $\lambda_1 = \lambda_2 = \lambda_3 = 1$ and $m_1 = m_2 = m_3 = 0$, which is further denoted as $\mathcal{P}_{1,0}$. Thus, in the case of no-arbitrage, $\mathcal{P} = 0$. The higher the value of the term, the more arbitrage is contained in the prices predicted by the ANN. In our experiments we choose $m_1 = m_2 = m_3 = 4$ and vary λ_i, $i = 1, 2, 3$. The results are presented in table A. The case $\lambda_1 = \lambda_2 = \lambda_3 = 0$ corresponds to no penalty.

It can be seen that training without no-arbitrage constraints produces high values of the penalty function (and therefore high arbitrage) both in-sample and out-of-sample.[3] This justifies our hypothesis that, by default, the ANN does not support no-arbitrage of the option prices. Increasing the penalty barrier by increasing all coefficients λ decreases the arbitrage, almost eliminating it. This is, however, obtained at the cost of a small increase in the MSE, which is often the case for the constraint optimisation. The results for the last line in table A are also presented graphically in Itkin (2019). The plots demonstrate that the no-arbitrage conditions cause the deep in-the-money prices predicted by the ANN to deviate from the actual prices, but they do not practically affect the prices for the other moneynesses. As we are using soft constraints, even at $\lambda = 100$ the arbitrage is not fully eliminated (in contrast to the hard constraints), but it is reduced to almost a negligible level.

The elapsed time necessary to train this network certainly increases: for example, for the last line in table A it becomes 40–60 seconds per epoch. We also need more epochs to achieve better convergence. In this test the number of epochs was therefore increased to 30. This is the cost of doing the constrained optimisation.

As a result, ANNs trained in this way guarantee no-arbitrage prices in-sample. However, they do not guarantee arbitrage-free prices out-of-sample. We leave a general solution of this problem as an open question for the time being. However, in some particular cases it is possible to provide a solution. For more details, see Itkin (2019).

Calibration of option pricing models using ANNs

A derivatives pricing model is said to be calibrated to a set of benchmark instruments if the values of those instruments, computed in the model, correspond to their market prices. Model calibration is the procedure of selecting model parameters in order to verify the calibration condition. Model calibration can be viewed as the inverse problem associated with the pricing of derivatives. In the theoretical situation where prices of call options are available for all strikes and maturities, the calibration problem can be explicitly solved using an inversion formula. In real situations, given a finite (and often sparse) set of derivative prices, model calibration is an ill-posed problem whose solution often requires a regularisation method.

Mathematically, this is equivalent to solving an optimisation problem. Suppose we are given a set of call option market prices C_{M} for a set of maturities T_i, $i = 1, \ldots, n$, and strikes $K_{i,j}$, $j = 1, \ldots, m_i$, corresponding to the known market input data, such as S, r, q, etc. This notation implies that the number of traded strikes $K_{i,j}$ could differ for each maturity T_i. Suppose we calibrate the model $\mathcal{M}$ described earlier, which has the model parameters $\boldsymbol{p}$ and the input data parameters $\boldsymbol{\theta}$. We differentiate between $\boldsymbol{p}$ and $\boldsymbol{\theta}$ as follows: parameters $\boldsymbol{p}$ have to be found by calibration, while the input data $\boldsymbol{\theta}$ represents some observable (or known) market values, like S, r, q, T, K.[4]

Given the values of $\boldsymbol{\theta}$ and $\boldsymbol{p}$, the model $\mathcal{M}$ can generate the theoretical (model) call option prices C. The following optimisation problem then needs to be solved to obtain the calibrated parameters of the model:

$$\arg\min_{\boldsymbol{p}} \sum_{i=1}^{n} \sum_{j=1}^{m_i} \omega_{i,j} \| C_{\mathrm{M}}(\boldsymbol{\theta}_{i,j}) - C(\boldsymbol{\theta}_{i,j}, \boldsymbol{p}) \| \tag{4}$$

where $\omega_{i,j}$ is a set of weights and a typical norm $\| \cdot \|$ used in (4) is L^2. In general, solving this problem for the non-linear function $C(\boldsymbol{\theta}_{i,j}, \boldsymbol{p})$ requires using algorithms of global optimisation, as the multi-dimensional objective (loss) function might have a lot of local minima. Also, this problem could be constrained, which is typically the case when, for example, the implied volatility surface is constructed based on market quotes (again see Itkin (2015) and the references cited therein).

Since computing the prices $C(\boldsymbol{\theta}_{i,j}, \boldsymbol{p})$ can be slow, eg, when using Monte Carlo simulation, solving this optimisation problem can be extremely time consuming, while frequent recalibration would be desirable for front offices and trading desks. Therefore, with the significant recent progress of DL and ANNs in mind, it has been proposed to replace the prices $C(\boldsymbol{\theta}_{i,j}, \boldsymbol{p})$ with their ANN approximation, as was discussed above. A nice survey of this approach can be found in Horvath *et al* (2019) and Liu *et al* (2019). Since the ANNs can be trained offline, and because getting the prices from the trained ANNs is very fast, the first bottleneck in solving the optimisation problem disappears. However, in both Horvath *et al* (2019) and Liu *et al* (2019), the global optimisation problem is still solved using traditional numerical methods.

In our opinion this second step (global optimisation) could be fully eliminated by using the inverse map approach. This approach was considered in Hernandez (2017), among other places, where the network was trained to directly return the calibrated parameters of the stochastic model. It was reported, however, that such an approach is unstable or does not converge. Therefore, in Horvath *et al* (2019), for example, the choice is made to use a two-step process, ie, to calibrate the model by running a global optimiser, but instead of using a real pricer, one uses its ANN approximation. We propose another DL approach to calibrate the model with no second step (global optimisation) below. This approach is constructed based on the inverse map but differs in its last steps.

■ **The inverse map approach.** As the first example consider again a simple Black-Scholes framework in which the only parameter of the model is the implied volatility σ. Earlier, we generated random vectors $\boldsymbol{\xi} =$

[3] *The in-sample arbitrage is bigger because the number of in-sample inputs is 182,016, while the number of out-of-sample inputs is four times smaller.*

[4] *Here, as an example, we discuss just the simplest case. In more complex models, r, q could be stochastic variables with their own models that are characterised by an additional set of parameters $\boldsymbol{p}$.*

$[S, K, T, r, q, \sigma]$ and then computed the call option price C using the Black-Scholes model. In other words, by doing so we constructed a map $\mathcal{M}_d : [\boldsymbol{p}, \boldsymbol{\theta}] \mapsto C$. However, once the input $\boldsymbol{\xi} = [S, K, T, r, q, \sigma]$ and the output $C(S, K, T, r, q, \sigma)$ are generated, these variables can be rearranged to produce the inverse map $\mathcal{M}_i : [S, K, T, r, q, C] \mapsto \sigma$, or, in general, $\mathcal{M}_i : [\boldsymbol{\theta}, C] \mapsto \boldsymbol{p}$. Then, using the same procedure described in the section titled 'ANNs and option Greeks', we build a feed-forward ANN. Given a vector $\boldsymbol{\xi} = [S, K, T, r, q, C]$, this ANN returns the implied volatility σ.

A similar approach (albeit, perhaps, a more general one) is discussed in Ardizzone *et al* (2018), where a particular class of neural networks – so-called invertible neural networks (INNs) – is considered to be well suited to this task. The authors verify experimentally, on artificial data and on real-world problems from astrophysics and medicine, that INNs are a powerful analytical tool for finding multi-modalities in parameter space, for uncovering parameter correlations, and for identifying unrecoverable parameters. Therefore, applying INNs to the calibration problem in finance could be an alternative approach.

The inverse map $\mathcal{M}_i : [\boldsymbol{\theta}, C] \mapsto \boldsymbol{p}$ can be used for calibration of a given model. However, it cannot be used directly. Indeed, suppose again we are given a set of call options market prices C_{M} for a set of maturities T_i, $i = 1, \ldots, n$, and strikes $K_{i,j}$, $j = 1, \ldots, m_i$, corresponding to the known market input data $\boldsymbol{\theta} = [S, r, q]$. Then, for each particular set $\boldsymbol{\xi}_{i,j} = [\boldsymbol{\theta}, T_i, K_{i,j}, C(\boldsymbol{\theta}, T_i, K_{i,j}, \sigma_{i,j})]$ being used as an input to the ANN, the latter will return the unique value $\sigma_{\mathrm{ann}}(\boldsymbol{\xi}_{i,j})$. Thus, the trained ANN does not solve the calibration problem, which requires a single value of the parameter σ to solve (4).

Therefore, it is useful to reformulate (4) in the following form:

$$\arg\min_{\sigma_{\mathcal{M}}} \sum_{i=1}^{n} \sum_{j=1}^{m_i} \bar{\omega}_{i,j} \| \sigma_{\mathrm{ann}}(\boldsymbol{\xi}_{i,j}) - \sigma_{\mathcal{M}} \| \tag{5}$$

where $\sigma_{\mathcal{M}}$ denotes the calibrated value of the implied volatility and $\bar{w}$ are some weights. In a more general case of the model with N parameters $\boldsymbol{p} = p_1, \ldots, p_N$, this could be rewritten as:

$$\arg\min_{\boldsymbol{p}} \sum_{i=1}^{n} \sum_{j=1}^{m_i} \sum_{k=1}^{N} \bar{\omega}_{i,j,k} \| p_{k,\mathrm{ann}}(\boldsymbol{\xi}_{i,j}) - p_k \| \tag{6}$$

This is also an optimisation problem, but under the L^2 norm it can be solved analytically. The obvious solution reads:

$$p_k = \frac{\sum_{i=1}^{n} \sum_{j=1}^{m_i} \bar{\omega}_{i,j,k}\, p_{k,\mathrm{ann}}(\boldsymbol{\xi}_{i,j})}{\sum_{i=1}^{n} \sum_{j=1}^{m_i} \bar{\omega}_{i,j,k}} \tag{7}$$

Thus, the algorithm for finding the calibrated values of all the parameters $\boldsymbol{p}$ also consists of two steps. In the first we use the inverse map $\mathcal{M}_i$ and create a trained feed-forward ANN. In the second step we substitute the market data into the ANN to obtain $\boldsymbol{p}_{\mathrm{ann}}(\boldsymbol{\xi}_{i,j})$, and then we use (6) to find the calibrated values of $\boldsymbol{p}$. However, it does not require solving any optimisation problem as this problem is already solved analytically in (7).

It would be a valid question to ask how the objective function in (6) differs from that in (4). Do we solve the same problem, or is this statement of the calibration problem quite different? To answer this, first observe that the approach in Horvath *et al* (2019) and Liu *et al* (2019) replaces (4) with another problem:

$$\arg\min_{\boldsymbol{p}} \sum_{i=1}^{n} \sum_{j=1}^{m_i} \omega_{i,j} \| C_{\mathrm{M}}(\boldsymbol{\theta}_{i,j}) - C_{\mathrm{ann}}(\boldsymbol{\theta}_{i,j}, \boldsymbol{p}) \| \tag{8}$$

Based on the direct map $\mathcal{M}_d : [\boldsymbol{p}, \boldsymbol{\theta}] \mapsto C$ generated by the trained ANN we find (Itkin 2019) that to the first order of approximation on each $\Delta p_{i,j,k} = p_{k,\mathrm{ann}}(\boldsymbol{\xi}_{i,j}) - p_k$ under the norm $|\cdot|$, we obtain:

$$\omega_{i,j} |C_{\mathrm{M}}(\boldsymbol{\theta}_{i,j}) - C_{\mathrm{ann}}(\boldsymbol{\theta}_{i,j}, \boldsymbol{p})| \leqslant \sum_{k=1}^{N} \bar{\omega}_{i,j,k} |p_{k,\mathrm{ann}}(\boldsymbol{\xi}_{i,j}) - p_k| \tag{9}$$

where:

$$\bar{\omega}_{i,j,k} = \omega_{i,j} \left| \frac{\partial C_{\mathrm{ann}}(\boldsymbol{\theta}_{i,j}, \boldsymbol{p}_{i,j})}{\partial p_{k,\mathrm{ann}}(\boldsymbol{\xi}_{i,j})} \right|$$

All sums on the right-hand side of (9) are non-negative and depend on the same set of parameters $\boldsymbol{p}$. Therefore, minimisation $\arg\min_{\boldsymbol{p}}$ of the right-hand side of (9) also solves the problem (8) assuming that all $\bar{\omega}_{i,j,k}$ exist and are finite. Since $|x| = \sqrt{x^2}$, this solution (given by (7)) also solves (8) up to $O(\sum_{k,i,j} (\Delta p_{i,j,k})^2)$.

The derivatives of the call option price on each parameter, as defined in the right-hand side of (9), are computed by the ANN automatically (by using adjoint algorithmic differentiation), and are therefore known. Given weights $w_{i,j}$, therefore, all weights $\bar{w}_{i,j,k}$ can be obtained at each node of the ANN for free.

It is important to underline that, in order to be consistent when switching from (8) to (9), we use the following sequence of steps:

(1) Given a set of inputs $\boldsymbol{\xi}$ we use the model pricer to generate a set of corresponding model outputs $C(\boldsymbol{\xi})$.

(2) We use all pairs $\{\boldsymbol{\xi}, C(\boldsymbol{\xi})\}$ to generate a map $\mathcal{M}_d : \boldsymbol{\xi} \mapsto C_{\mathrm{ann}}(\boldsymbol{\xi})$. This map is produced by training the ANN, ie, by solving a constraint optimisation problem, and therefore takes into account no-arbitrage conditions in-sample.

(3) We now rearrange variables to produce the inverse map $\mathcal{M}_{i,\mathrm{ann}}$: $\{\boldsymbol{\theta}, C_{\mathrm{ann}}(\boldsymbol{\xi})\} \mapsto \boldsymbol{p}$. It is worth mentioning that, in general, this map differs from the map $\mathcal{M}_i : \{\boldsymbol{\theta}, C(\boldsymbol{\xi})\} \mapsto \boldsymbol{p}$.

(4) Having the inverse map, we use it together with the market data to generate the outputs $p_{k,\mathrm{ann}}(\boldsymbol{\xi}_{i,j})$, and then we use (7) together with the last line of (9).

In a more sophisticated scenario, the model could be calibrated not just to a current snapshot of the market data at time t, but to some historical data at time $\tau < t$, where the stock price $S_\tau \neq S_t$. This then extends a set of indexes i, j to reflect the dependence of the call option price on S, but this extension is straightforward.

■ **Stability and convergence.** Here again we consider the simple example described earlier that uses the Black-Scholes model. We now want to build an inverse map and solve the problem of calibration of the implied volatility σ given the market option prices. The only difference compared with the direct approach is that after all training sets are generated, we switch the call price C and σ, thus producing a map $\mathcal{M}_i : [S, K, T, r, q, C] \mapsto \sigma$. If we then train the ANN using this data as inputs and outputs (using the same architecture of the ANN and even the same code as in the section titled 'ANNs and option Greeks'), the results obtained look discouraging. Training is definitely unstable and of poor quality even for in-sample data, in agreement with what was found by Hernandez (2017).

This, however, might be easily explained. As the option price C is now an input parameter and σ is an output, consider the gradient $\partial\sigma/\partial C =$

1/Vega, where Vega is the option Vega (see, for example, figure 11 in Itkin (2019)). Obviously, Vega is a bell-shaped function. When the product $\kappa = \sigma\sqrt{T}$ is small, the inverse gradient is very high at both high and low moneyness M, and is small at-the-money. This creates a problem for training the ANN that is known in the literature as the exploding gradients problem. This problem has already caught the attention of the industry: see, for example, Goldberg (2017). Various technical methods to deal with it have been proposed, including reducing the batch size, changing the architecture of the ANN by switching to a long short-term memory network, having fewer hidden layers in the network, and so on. Also, if exploding gradients occur, one can check for and limit the size of the gradients during the training of the ANN. This is called gradient clipping. As per Goldberg (2017), dealing with the exploding gradients has a simple but very effective solution: clipping gradients if their norm exceeds a given threshold. In particular, the Keras library provides an interface for setting the 'clipnorm' or 'clipvalue' arguments on your favourite optimiser before training. The choice of clipnorm is discussed in more detail in Itkin (2019).

Another important thing is scaling. Scaling the inputs first makes sense if one reveals some symmetries (as in the Black-Scholes model for European vanilla options). In this case, scaling allows a reduction in the number of input variables. Second, good scaling of both inputs and outputs helps with changing the norm of the gradients. Therefore, it also helps with solving the exploding gradients problem. In particular, as applied to our test, we replace the output variable σ with:

$$\sigma \mapsto N\left(\frac{\log(S/K)}{\sigma\sqrt{T}}\right) \equiv \mathcal{N}(\sigma)$$

where $N(x)$ is the normal cumulative distribution function. Thus, all outputs are in [0, 1]. This scaling is chosen for the reasons mentioned above and also because the inverse map $\mathcal{N}(\sigma) \mapsto \sigma$ can be done analytically. Other functions can also be proposed and work well, eg:

$$\sigma \mapsto \frac{1}{2}\left[1 + \tanh\left(\frac{\log(S/K)}{\sigma\sqrt{T}}\right)\right]$$

In a test we use the same ANN design as in the section titled 'ANNs and option Greeks' with the Adam optimiser. The results of this test for in-sample and out-of-sample inputs, including the density of the difference $\delta = y - y_{\text{ann}}$, can be found in Itkin (2019). The MSE both in-sample and out-of-sample is 1.5bp, and the mean percentage error is −0.3 after 30 epochs. Our experiments show that the training of the ANN is stable and converges. Still, some improvements could be desirable at both very small and high implied volatilities (at the tails of the distributions discussed above), where, as is known, the call option price reaches its intrinsic value. Therefore, the inverse map $\mathcal{M}_i : C \mapsto \sigma$ is not well defined at the fixed machine arithmetic.

Further applications of the method include numerical experiments with more sophisticated models with a number of model parameters (outputs). These results will be reported elsewhere. ■

Andrey Itkin is an adjunct professor at NYU's Department of Risk and Financial Engineering and director, senior research associate at Bank of America in New York. He thanks Peter Carr, Igor Halperin, Brian Huge, Antoine Savine, Mehdi Tomas and participants of the QuantMinds 2019 International Conference for useful discussions, and he thanks two anonymous referees for helpful comments. Email: aitkin@nyu.edu.

REFERENCES

Ardizzone L, J Kruse, S Wirkert, D Rahner, EW Pellegrini, RS Klessen, L Maier-Hein, C Rother and U Kothe, 2018
Analyzing inverse problems with invertible neural networks
Preprint (September), arXiv:1808.04730

Changhau I, 2017
Activation functions in neural networks
Available at https://isaacchanghau.github.io/post/activation_functions/

Elouerkhaoui Y, 2019
Derivatives pricing with a deep learning approach
QuantMinds International Conference (May)

Goldberg Y, 2017
Neural network methods in natural language processing
In *Synthesis Lectures on Human Language Technologies*, G Hirst (ed)
Morgan & Claypool

Halperin I, 2017
QLBS: Q-learner in the Black–Scholes(–Merton) worlds
Preprint (December), arXiv:1712.04609

Hernandez A, 2017
Model calibration with neural networks
Risk June, pages 133–137

Hornik K, M Stinchcombe and H White, 1990
Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks
Neural Networks 3(5), pages 551–560

Horvath B, A Muguruza and M Tomas, 2019
Deep learning volatility: a deep neural network perspective on pricing and calibration in (rough) volatility models
Preprint (February), SSRN, 3322085

Itkin A, 2015
To sigmoid-based functional description of the volatility smile
North American Journal of Economics and Finance 31, pages 264–291

Itkin A, 2019
Deep learning calibration of option pricing models: some pitfalls and solutions
Preprint (June), arXiv:1906.03507

Itkin A and A Lipton, 2018
Filling the gaps smoothly
Journal of Computational Sciences 24, pages 195–208

Kathuria A, 2018
Intro to optimization in deep learning: vanishing gradients and choosing the right activation function
Blog post, Optimization Series (July), available at https://blog.paperspace.com/vanishinggradients-activation-function/

Liu S, A Borovykh, LA Grzelak and CW Oosterlee, 2019
A neural network-based framework for financial model calibration
Preprint (April), arXiv:1904.10523

Marquez-Neila P, M Salzmann and P Fua, 2017
Imposing hard constraints on deep networks: promises and limitations
Preprint (June), arXiv:1706.02025

Smith AE and DW Coit, 1996
Penalty functions
In *Handbook of Evolutionary Computation*, T Baeck, D Fogel and Z Michalewicz (eds)
Oxford University Press/Institute of Physics Publishing