Model Calibration with Neural Networks

Andres Hernandez
IBM Risk Analytics
andres.hernandez@gmx.net

July 20, 2016

Abstract

A central consideration for the use of any pricing model is the ability to calibrate that model to market or historical prices. Whether the information needed by the model can be effectively implied from the data or not is one part of the calibration problem. However, in many applications, the speed with which that calibration can be performed influences the usability of that model.

In the following a method is presented to calibrate models using artificial neural networks, which can perform the calibration significantly faster regardless of the model, hence removing the calibration speed from consideration for a model's usability.

1 Introduction

There are several aspects that need to be taken into account when considering the usability of a pricing model, and among those, one of central practical importance is the ability to calibrate the model to current market information.

On the one hand, there is the consideration of whether there are sufficient sources of information available from which the required information that a model requires could be implied, or at the very least proxied from. This is a vital point for the successful use of the model, and one whose scope is basically in finance.

In many applications, when considering the applicability of a pricing model, a second important point is the speed with which it can be calibrated. This is a purely technical matter. A model might be deemed preferable to another on financial grounds, but if it takes too long to calibrate (too long being dependent on the application), then the model becomes impractical.

Of course, the decision might normally not be so clear cut. In some instances, the model could be acceptable from a technical point, but only after compromises are made on the finance side, e.g. by limiting the number

of instruments used during calibration or by using approximations which can lead to inaccuracies for certain parameter regions.

The purpose of this work is to provide an alternative calibration method, which will remove completely from consideration the time constraint imposed on a model. By using Artificial Neural Networks, the vast majority of the calculation workload can be offloaded to a separate training process, which by its very nature is allowed to run offline, while the actual online calibration runs very fast, equivalent to multiplying a few small-sized matrices.

As will be seen in Section 4, the splitting of the calibration problem into a training phase and an evaluation phase is possible, with the fulcrum of the exercise being the generation of a large training set, such that the neural network can generalize from it.

As an added benefit, but not addressed here in this work, neural networks, being fully differentiable functions, can also provide model parameter sensitivities to market prices, potentially informing when a model should be recalibrated.

This paper is structured as follows. Section 2 lays out the calibration process, such that the problem that needs to be solved is made explicit. Section 3 details how neural networks can be used to approach the calibration problem. Section 4 provides an example use of the previously discussed methodology, and applies it to the simple calibration of a Hull-White model with constant parameters. Section 5 provides some closing comments and perspectives for future work.

2 Calibration Problem

For a given model $\mathbf{M}$, an instrument's theoretical quote will be denoted by

$$Q(\tau) = \mathbf{M}(\theta; \tau, \phi), \tag{1}$$

where θ represents the model parameters, τ represents the identifying properties of the particular instrument, e.g. maturity, day-count convention, etc., and ϕ represents other exogenous factors used for pricing, e.g. an interest rate curve. Eq. (1) can represent expectation under a risk-neutral measure, or some other similar procedure, the nature of which will not be relevant for this discussion.

The model $\mathbf{M}$ has n parameters that need to be calibrated, and some or all of them might have constraints

$$\theta \in S \subseteq \mathbb{R}^n \tag{2}$$

Model calibration is the process by which the model parameters are adjusted so as to 'best' describe a set of N relevant market quotes

$$\{Q^{mkt}\} = \{Q_i^{mkt} | i = 1 \dots N\}, \quad \{\tau\} = \{\tau_i | i = 1 \dots N\}. \tag{3}$$

In general, the question of which instruments should be considered is not straightforward, and arguments can often be made for the inclusion or exclusion of a particular quote, e.g. liquidity concerns or to avoid over-fitting to a particular maturity region.

However, it is also possible that the set of instruments is reduced, not for financial and mathematical considerations, but for the practical concern of reducing the calibration time. For the current exercise, this concern should be completely discarded, and instead the set is chosen purely on financial and mathematical grounds, although weights could be adjusted as deemed necessary.

Previously it was mentioned that the model parameters were chosen to best fit the given quotes. The concept of best is measured by a cost function, which will be particular to a problem, and like the decision of which quotes to include, there are different choices for it. And like in the quote set, the decision of which cost function to adopt, can, in general, be advised by the practical concern of reducing calibration time, but once again this concern should be dropped.

There are different types of cost functions in use, and we will not argue here for one over another, but it usually takes the form of a weighted average of all the differences between a market provided indicator and the indicator as calculated via the model

$$\text{Cost}\left(\theta^*, \{Q^{mkt}\}; \{\tau\}, \phi\right) = \sum_{i=1}^{N} w_i (Q(\tau_i) - Q^{mkt}(\tau_i))^2. \tag{4}$$

The calibration problem consists then in finding the parameters θ, which minimize the cost function

$$\theta = \underset{\theta^* \in S \subseteq \mathbb{R}^n}{\text{argmin}}\ \text{Cost}\left(\theta^*, \{Q^{mkt}\}; \{\tau\}, \phi\right) = \mathbf{\Theta}\left(\{Q^{mkt}\}; \{\tau\}, \phi\right). \tag{5}$$

The calibration problem is now seen as a function with N arguments and n outputs

$$\mathbf{\Theta} : \mathbb{R}^N \mapsto S \in \mathbb{R}^n \tag{6}$$

It is this function, $\mathbf{\Theta}$, which will be approximated by a neural network.

The different models and cost functions used in quantitative finance, present a great variety of conditions, from convex cost functions, which can be solved in a straightforward manner by use of local optimizers, to more complex structures, with multiple local optima present, and for which a global optimizer might be required in order to reach an acceptable solution. Like in almost every other aspect of model calibration, the choice of optimizer and the acceptability of an optimization procedure, can be informed by financial considerations, e.g. the error is below a financially acceptable threshold, but it can also be influenced by computation time. Just like in

the model selection, the set of calibration instruments, and the form of the cost function, the consideration of computation time should be discarded in the current exercise.

Now, the optimization problem might be ill defined, or result in parameters that vary wildly from calibration to calibration. This could certainly be grounds for concern, but in the following they will not be addressed as they are not particular to the approach laid out here, nor is it relevant to purpose of this work, which, once again, is to allow for moving away from obtaining the calibrated parameters via an optimization, possibly very time consuming, and replace it with a single evaluation of a neural network, which regardless of the model being represented, should be a very fast operation.

Now, while the cost function might even present a very chaotic outlook, but still have a well defined global minimum, this should not, a priori, present difficulties for the current approach, as the neural network will not be trained to learn the shape of the cost function, but of $\boldsymbol{\Theta}$.

It need not be everywhere smooth, and may in fact contain a few discontinuities, either in the function itself, or on its derivatives, but in general it is expected to be continuous and smooth almost everywhere. As it will be mentioned in Section 3, the mathematical basis for the representation of a function by a neural network, is the continuity of such a function [6]. However, it can still be that in practice a neural network is able to approximate the function, but in the case that it is not so, then a system with multiple networks could be set up, with one neural network trained to reproduce the function in each regime, while a separate neural network is trained to classify the current regime, and so selecting the predictor to apply to obtain the model parameters.

3 Neural Network

Artificial neural networks have now become ubiquitous in image and speech recognition tasks, and while the use to which they are put are now very diverse, they are, in the end, simply approximating a function with many inputs and some outputs. When used to classify the input into a finite number of categories, the function transforms a continuous input into a discrete output, but still is attempting to approximate that function.

There are, of course, other approaches available to approximate a function. There are other approaches even within machine learning, and they can be successful. However, neural networks have been very successful in approximating functions of many inputs. In terms of Eq.(6), when N is large. This is where neural networks excel, and where other methods can falter, e.g. interpolation tables.

We will be very pragmatic, and not attempt to find the set of financial models, whose $\boldsymbol{\Theta}$ function could be approximated via a neural network.

Instead, with the confidence bestowed by the Universal Approximation Theorem [6], we will assume that it can in fact be done, and then attempt to do it.

It is expected that different models will best be proxied by different neural networks, and in particular the number of layers and of neurons per layer can certainly be expected to change from application to application. However, regardless of the topology being used, once a suitable neural network has been found, the calibration problem will be split into two phases: a supervised training phase, which will be performed offline; and the evaluation phase, which gives the model parameters for a given input, and which will be extremely fast.

What is gained by calibrating a model with neural networks is the offloading of the computationally intensive part into an offline process, which for financial use cases can take from hours to days, depending on available resources and the model itself. The calculations performed live, i.e. when the calibrated parameters are required, is akin to multiplying a few matrices, which any computer will be able to handle in an expeditious manner.

For the supervised training phase, the task is again split in two: collecting a large enough training set; and the actual training, validation and testing. In particular, the preparation of the training set will be the crucial part of this whole exercise. The training set needs to be large enough, so as to allow the learning of peculiarities of the model, i.e. not to overfit. The size and quality of the model will directly affect the neural network's ability to generalize the obviously limited set of examples, into new situations.

Taking all historical values available and calibrating them would give a set of examples, where the output is known for a given input, and which can then be used for training. The disadvantage of this approach is two fold, first the training set is unlikely to be large enough, and second, the possibility of backtesting to historical data will be compromised.

As a very good approximation for the inverse of $\mathbf{\Theta}$ is known, it is simply the regular valuation of the instruments under a given set of parameters, it is the model itself, which will serve as the basis of the training set

$$\mathbf{\Theta}^{-1}\left(\theta;\{\tau\},\phi\right) \approx \mathbf{M}\left(\theta;\{\tau\},\phi\right) = \{Q\}. \tag{7}$$

This means that we can generate new examples simply by generating random parameters θ, albeit with some complications that need to be considered. The first is that if the model requires exogenous factors ϕ, then they will also need to be generated randomly and very likely will need to be correlated properly with the parameters for the example set to be meaningful.

A second complication can come from the inability of a model to perfectly fit all relevant market quotes. In the example presented in Section 4, it was observed that generating random noise, properly correlated with the exogenous ϕ and parameters θ, improved the ability of the neural network to learn and generalize.

The historical data then does need to be collected. A portion of it will be used purely to serve as an out-of-sample backtest, and ultimately as basis for an acceptability judgement, but another part will be needed to estimate the correlation between the parameters, and if needed the exogenous factors and the errors.

The process as described above already assumes that a neural network topology has been proposed. In principle, it is difficult a priori to determine what is a good topology. Instead the usual procedure is to first collect a large training set, and then to try out different configurations.

Even just limiting the discussion to feed-forward (FNN) and convolutional neural networks (CNN), one will have a large number of options: number of layers, neurons per layer, regularization procedure, optimization method, activation function, number of training iterations, etc. Setting these so called hyper-parameters judiciously is important because they can have a strong impact on the learning process.

The process of hyper-parameter optimization is a recurring problem in neural networks, and there are standard approaches that can give good results. The simplest examples are grid search and manual search. In the former a multi-dimensional grid of parameters is prepared and the configuration with the best test results is picked.

In the current exercise a mixture of grid and manual search was used. As each set of hyper-parameters requires that a new neural network be trained, which is time-consuming, a grid search can spend significant amount of time pursuing avenues which are very unlikely to bear fruit. After a limited grid search, a manual inspection selected the areas on which a further grid search should concentrate. This was repeated in an iterative process until no further improvement was found.

There exist, of course, more developed approaches, e.g. random search [2] or more formal search strategies [3]. For the example in Section 4, the combination of grid and manual search mentioned above produced good results, but the authors are aware that in future problems, specially ones with bigger inputs where CNNs should have a bigger impact, these more sophisticated approaches could pay off.

For the example in Section 4, the input will be composed of two elements, a 13×12 swaption ATM matrix, and an IR curve. Due to the success of CNNs in image recognition problems, it was natural to try them, but for the small 'image' size, it did not provide much improvement over simpler FNNs, while increasing the learning time significantly.

4 Example: Hull-White

As an example, the single-factor Hull-White model calibrated to 156 GBP ATM swaptions will be used

$$dr_t = (\theta(t) - \alpha r_t)dt + \sigma dW_t \tag{8}$$

with α and σ constant, and shared across all option maturities. $\theta(t)$ is picked to replicate the current yield curve $y(t)$.

The problem is then

$$(\alpha, \sigma) = \mathbf{\Theta}\left(\{Q^{mkt}\}; \{\tau\}, y(t)\right) \tag{9}$$

This is a problem shown in QuantLib's BermudanSwaption example [1], available both in c++ and Python.

The collected historical data comprises log-normal ATM volatility quotes for GBP from January 2nd, 2013 to June 1st, 2016. The option maturities are 1 to 10 years, plus 15 and 20 years. The swap terms from 1 to 10 years, plus 15, 20, and 25 years. Giving a total of 156 swaptions to be used equally weighted as calibration instruments.

For the yield curve, the 6M tenor Libor curve was used: bootstrapped using a monotonic cubic spline interpolation of the zero curve, and built on top of the OIS curve. Only FRAs and swaps were used to bootstrap the curve. When serving as input to the neural network, the yield curve will be discretely sampled on 44 points: 0, 1, 2, 7, 14 days; 1 to 24 months; 3 to 10 years; plus 12, 15, 20, 25, 30, 40, and 50 years.

A Levenberg–Marquardt local optimizer was used to minimize the equally-weighted average of squared NPV differences[1]. The calibration was done twice, with different starting points: a default a starting point with $\alpha = 0.1$ and $\sigma = 0.01$; and as a second starting point, the calibrated parameters from the previous day using the default starting point were also used. The results for the default calibration are shown in Fig. (1).

An interesting effect is seen in Fig.(1), where α seems to switch regimes in Q3 2013. This could be interesting in other models, but for this exercise will not be pursued further.

Once the calibration history is available, the generation of the training set can proceed:

1. Obtain errors for each calibration instrument for each day
2. As parameters are positive, take the natural logarithm on the calibrated parameters

[1] This is the default calibration objective in QuantLib, but it can easily be changed to have different weights or different value indicator

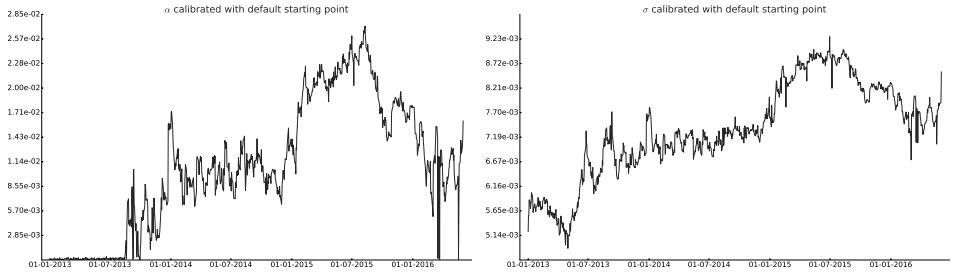

Figure 1: Calibration using default starting point

3. Rescale yield curves, parameters, and errors to have zero mean and variance 1

4. Apply dimensional reduction via PCA to yield curve, and keep parameters for given explained variance (99.5%)

5. Calculate covariance of rescaled log-parameters, PCA yield curve values, and errors

6. Generate random normally distributed vectors consistent with given covariance

7. Apply inverse transformations: rescale to original mean, variance, and dimensionality, and take exponential of parameters

8. Select reference date randomly from set used for covariance estimation

9. Obtain implied volatility for all calibration instruments and apply random errors to results

For the hyper-parameter optimization, the sample set was divided into three parts: 60% served as the training set, 20% served as the cross-validation set, and the last 20% served as a testing set. As usual, the training set was used during the actual intra-epoch training. The cross-validation set was used to measure the inter-epoch improvement, including the possibility of stopping early. And the testing set was used to pick the best configuration. Note that the real testing set will be the history, so the fitting of the hyper-parameters to this fictitious testing set, does not imply any problems for the actual testing done later.

As mentioned above, the optimization was a mixture of grid and manual search, with a truncated grid search over the number of layers, neurons per layer[2] (with all hidden layers having the same number of neurons), the learning rate for a Root Mean Square Propagation (RMSProp) optimizer,

[2]The number of neurons per layer was varied in multiples of 2, with the following values tried 2^6, 2^8, 2^{10}, 2^{12}, and 2^{14}

and a dropout rate, that was set equal for all layers, including the output layer. A hyperbolic-tangent activation function was used for all neurons on all tests.

Once the best of these was selected, different activation units were tried, including a radial basis function, an rectified linear unit (ReLU), and an exponential linear unit (ELU)[4]. Each one provided improvement over the hyperbolic tangent, and in increasing order, with the ELU providing the most improvement. Finally, an Adam optimizer with Nesterov Momentum incorporated[5] (Nadam) was tried, and the learning rate was again shifted, which found the original value of 0.001 being the best out of the tried valued. It was observed that the Nadam optimizer converged faster than the RPSProp optimizer.

The dropout rate, i.e. the amount of instruments that will be randomly picked and dropped at each update, was surprisingly found to be quite low, with the best results found with 1/5 from among the tried values of 1/5, 1/4, 1/3, and 1/2. After the selection of the best configuration, activation unit, and optimizer, a last experiment shifted the dropout rate on only the last layer, but it was observed that the lower value still provided the best results.

With a sample set of 150 thousand samples, the hyper-parameter optimization led to the feed-forward neural network detailed in Fig.(2). It is a FNN with 4 hidden layers, each layer with 64 neurons using an ELU activation function. The input is all put into a flat vector, as a FNN is fully connected and therefore has no concept of locality. A standard 500 epochs was used for training, but an early stopping strategy was applied, whereby if no improvement on the cross-validation loss objective was detected for 50 epochs, then learning would be stopped. A dropout rate of 1/5 was used in all layers, and a learning rate of 0.001 for the Nadam optimizer was used.

Different variations of CNNs were tried, but as the input matrix of 13×12 is relatively small, no significant improvement was obtained over the best FNN, while significantly increasing training time. It is expected that for other problems, e.g. involving an equity volatility surface with several hundreds or even thousands of calibration instruments, CNNs could become useful.

Two different neural networks were trained using a sample set produced with a covariance matrix estimated with historical data from January 2013 to June 2014 (40% of the historical data), and a second sample set using data from January 2013 to June 2015 ($\sim$73% of the historical data). For training, the sample set was split into 80% training set and 20% cross-validation. The testing set will be the historical data itself, which will then serve as backtesting for the model.

The comparison between the model and the calibrations with an optimizer and the two starting points are presented in Fig. (3-6). As is readily apparent the covariance matrix used in the production of the training set had

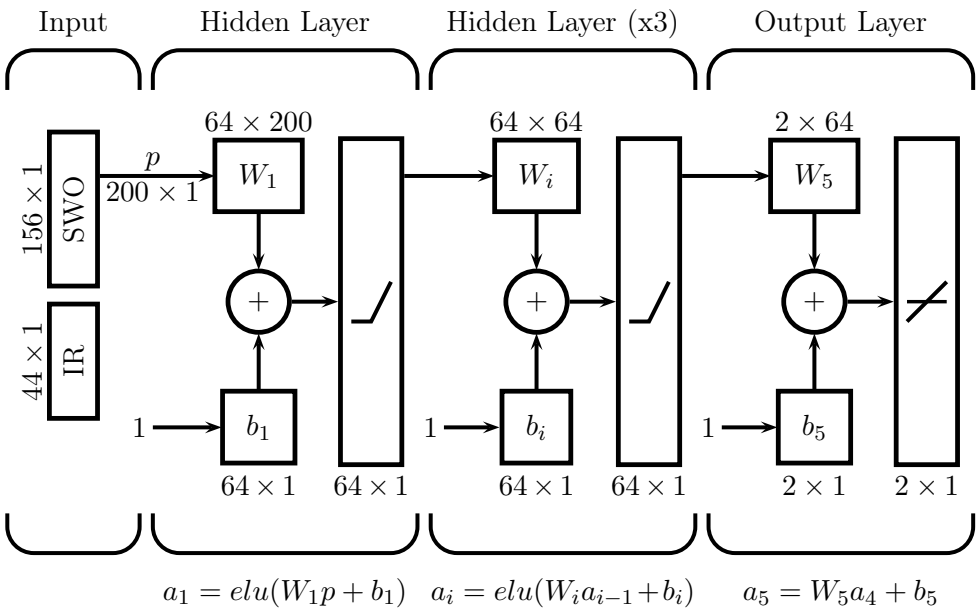

Figure 2: Feed-forward neural network used for Hull-White calibration

important consequences during backtesting. It was observed that a trained model would behave well only for a period of 6 months to 1 year beyond the period which was used for the covariance estimation.

5 Conclusions

The use of pricing models in quantitative finance generally necessitates their calibration to relevant market data in an optimization process which can be time consuming. While the choice of model is informed by many considerations, the time required to perform the calibration is certainly one of them. This is not to imply, that all model choices were driven by this consideration, but in certain settings, like trading desks, calibration time can certainly be important.

We propose a novel approach for calibration of financial models, which can offer significant time savings during the actual calibration over the traditional optimization based approach. The improvement in performance is achieved by utilizing neural networks to approximate the calibration problem, off-loading the bulk of the calculations to a training phase, which is done offline and at a convenient time.

As an example, the approach was tried out with a simple model, the Hull-White model with constant parameters calibrated to 156 swaptions. The results are shown in Fig.(3-6). As can be seen, the out-of-sample back-

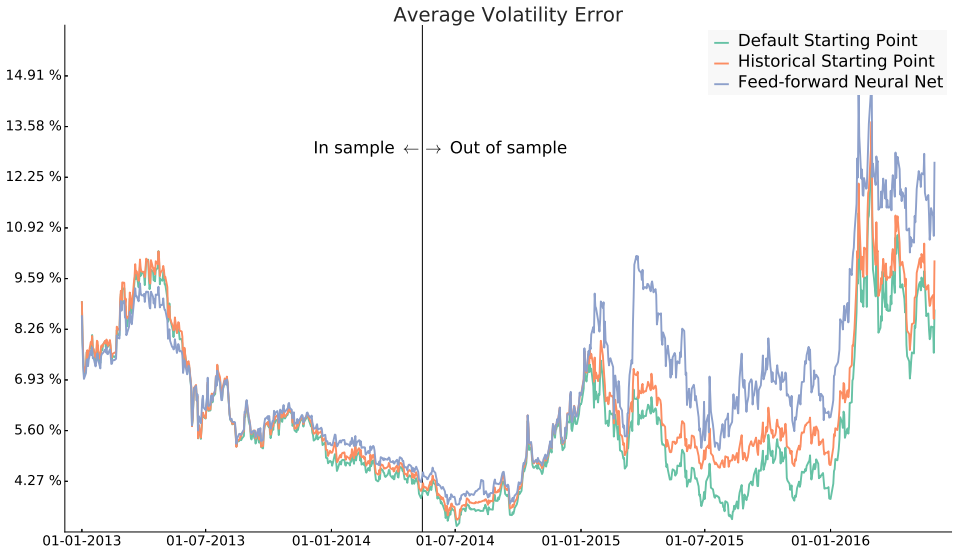

Figure 3: Correlation up to June 2014

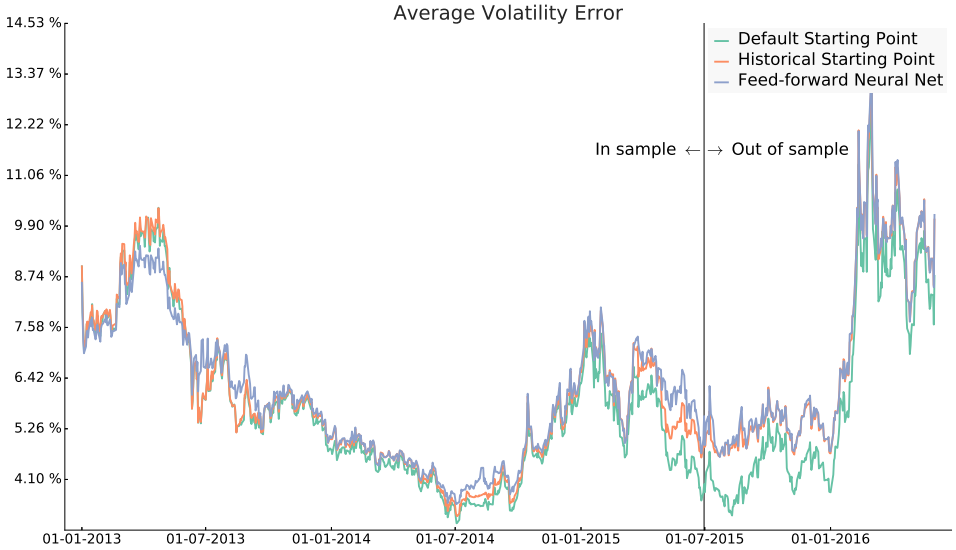

Figure 4: Correlation up to June 2015

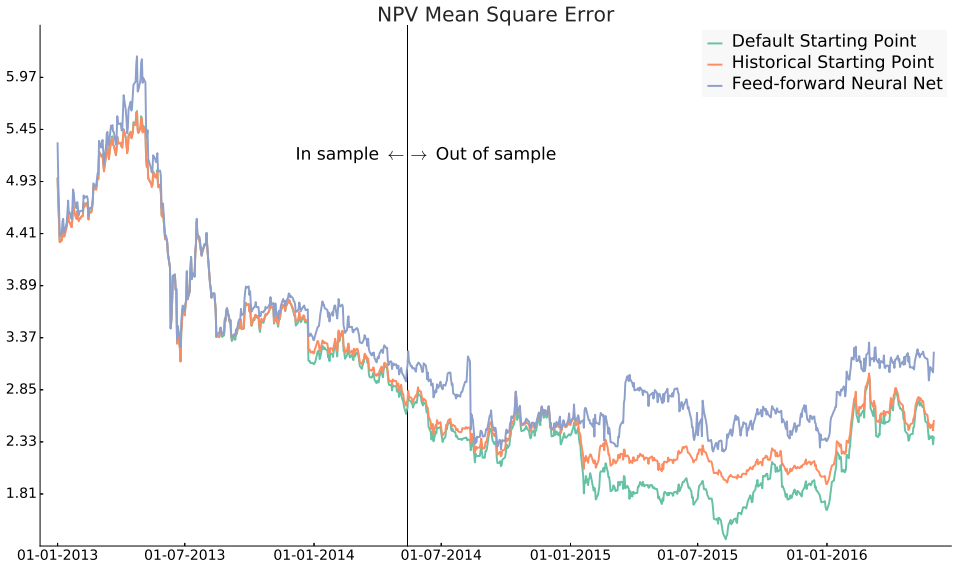

Figure 5: Correlation up to June 2014

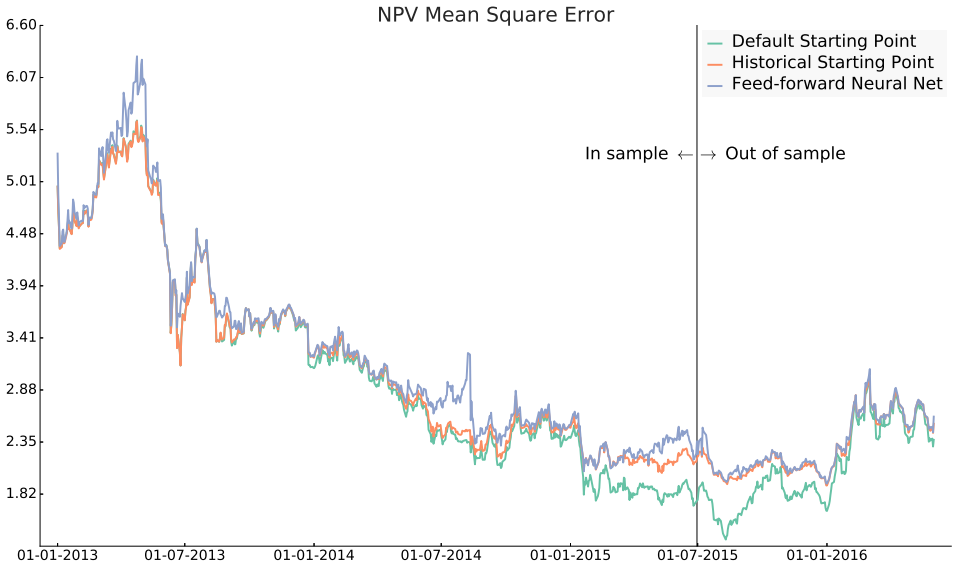

Figure 6: Correlation up to June 2015

testing shows good behaviour for between 6 months and 1 year after the end of the period used in the correlation estimation. After that, a significant degradation of the calibration performance of the neural network was observed.

The performance degradation after 6 months or 1 year is not problematic, as one could simply retrain the model every 2 or 3 months, and remain within acceptable bounds. Also, in critical environments, safeguards could be set up to monitor the reliability of the solution. For example, a periodic parallel calibration using the traditional approach could be used to ascertain the reliability of the neural network vis-a-vis the current market state.

An example with a more complicated calibration procedure, requiring the use of a global optimizer, will be produced shortly in a separate work. Further points of study involve using a parametrized correlation. Such a possibility would allow for training the neural network with states that do not only reflect the current market environment, but also possible future developments. It is important to mention at this point that unlike in Monte-Carlo calculations, it is not necessary for the sampling set to correctly be able to predict how the future market conditions will statistically look like, but rather simply to contain samples in sufficient quantities that mimic such possible conditions. Hence, sampling from a parametrized correlation structure, could extend the life-span of a trained model.

References

[1] QuantLib A free/open-source library for quantitative finance, `http://www.quantlib.org`

[2] James Bergstra, and Yoshua Bengio, *Random Search for Hyper-Parameter Optimization*, Journal of Machine Learning Research, 13:281-305, 2012

[3] Jasper Snoek, Hugo Larochelle, and Ryan Prescott Adams, *Practical Bayesian Optimization of Machine Learning Algorithms*, Neural Information Processing Systems, 2012

[4] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter, *Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)*, arXiv:1511.07289 [cs.LG]

[5] Timothy Dozat, *Incorporating Nesterov Momentum into Adam*, `http://cs229.stanford.edu/proj2015/054_report.pdf`

[6] Kurt Hornik, *Approximation capabilities of multilayer feedforward networks*, Neural Networks, Volume 4 Issue 2: 251-257, 1991