www.pwc.de

Model Calibration with Neural Networks

Andres Hernandez

pwc

Motivation

The point of this talk is to provide a method that will perform the calibration significantly faster regardless of the model, hence removing the calibration speed from a model's practicality.

As an added benefit, but not addressed here, neural networks, as they are fully differentiable, could provide model parameters sensitivities to market prices, informing when a model should be recalibrated

While examples of calibrating a Hull-White model are used, they are not intended to showcase best practice in calibrating them or selecting the market instruments.

Table of contents

Background

Definition

Model calibration is the process by which model parameters are adjusted to 'best' describe/fit known observations. For a given model $\mathbb{M}$, an instrument's theoretical quote is obtained

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

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. interest rate curve.

Definition

The calibration problem consists then in finding the parameters θ, which best match a set of quotes

$$\theta = \underset{\theta^* \in S \subseteq \mathbb{R}^n}{\arg\min}\ \text{Cost}\left(\theta^*, \{\hat{\mathbf{Q}}\}; \{\tau\}, \phi\right) = \mathbf{\Theta}\left(\{\hat{\mathbf{Q}}\}; \{\tau\}, \phi\right),$$

where $\{\tau\}$ is the set of instrument properties and $\{\hat{\mathbf{Q}}\}$ is the set of relevant market quotes

$$\{\hat{\mathbf{Q}}\} = \{\hat{\mathbf{Q}}_i | i = 1 \ldots N\}, \quad \{\tau\} = \{\tau_i | i = 1 \ldots N\}$$

The cost can vary, but is usually some sort of weighted average of all the errors

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

Definition

The calibration problem can be seen as a function with N inputs and n outputs

$$\boldsymbol{\Theta} : \mathbb{R}^N \mapsto \mathbb{R}^n$$

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 N can often be quite large, this presents a good use case for a neural network.

Hull-White Model

As examples, the single-factor Hull-White model and two-factor model calibrated to 156 GBP ATM swaptions will be used

$$dr_t = (\theta(t) - \alpha r_t)dt + \sigma dW_t \qquad dr_t = (\theta(t) + u_t - \alpha r_t)\,dt + \sigma_1 dW_t^1$$
$$du_t = -bu_t dt + \sigma_2 dW_t^2$$

with $dW_t^1 dW_t^2 = \rho dt$. All parameters, α, σ, σ_1, σ_2, and b are positive, and shared across all option maturities. $\rho \in [-1, 1]$. $\theta(t)$ is picked to replicate the current yield curve $y(t)$.

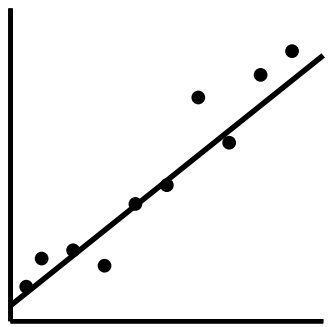
The related calibration problems are then

$$(\alpha, \sigma) = \mathbf{\Theta}_{1F}\left(\{\hat{\mathbf{Q}}\}; \{\tau\}, y(t)\right) \qquad (\alpha, \sigma_1, \sigma_2, b, \rho) = \mathbf{\Theta}_{2F}\left(\{\hat{\mathbf{Q}}\}; \{\tau\}, y(t)\right)$$

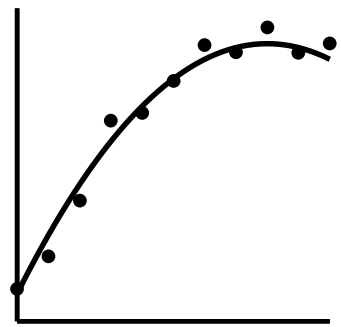
This is a problem shown in QuantLib's BermudanSwaption example, available both in c++ and Python.

Artificial neural networks

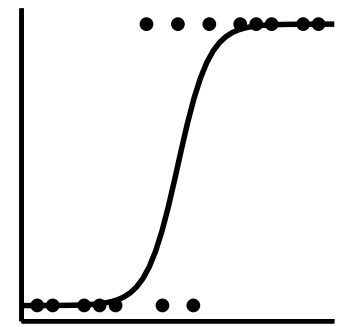
Artificial neural networks are a family of machine learning techniques, which are currently used in state-of-the-art solutions for image and speech recognition, and natural language processing.
In general, artificial neural networks are an extension of regression



$aX + b$



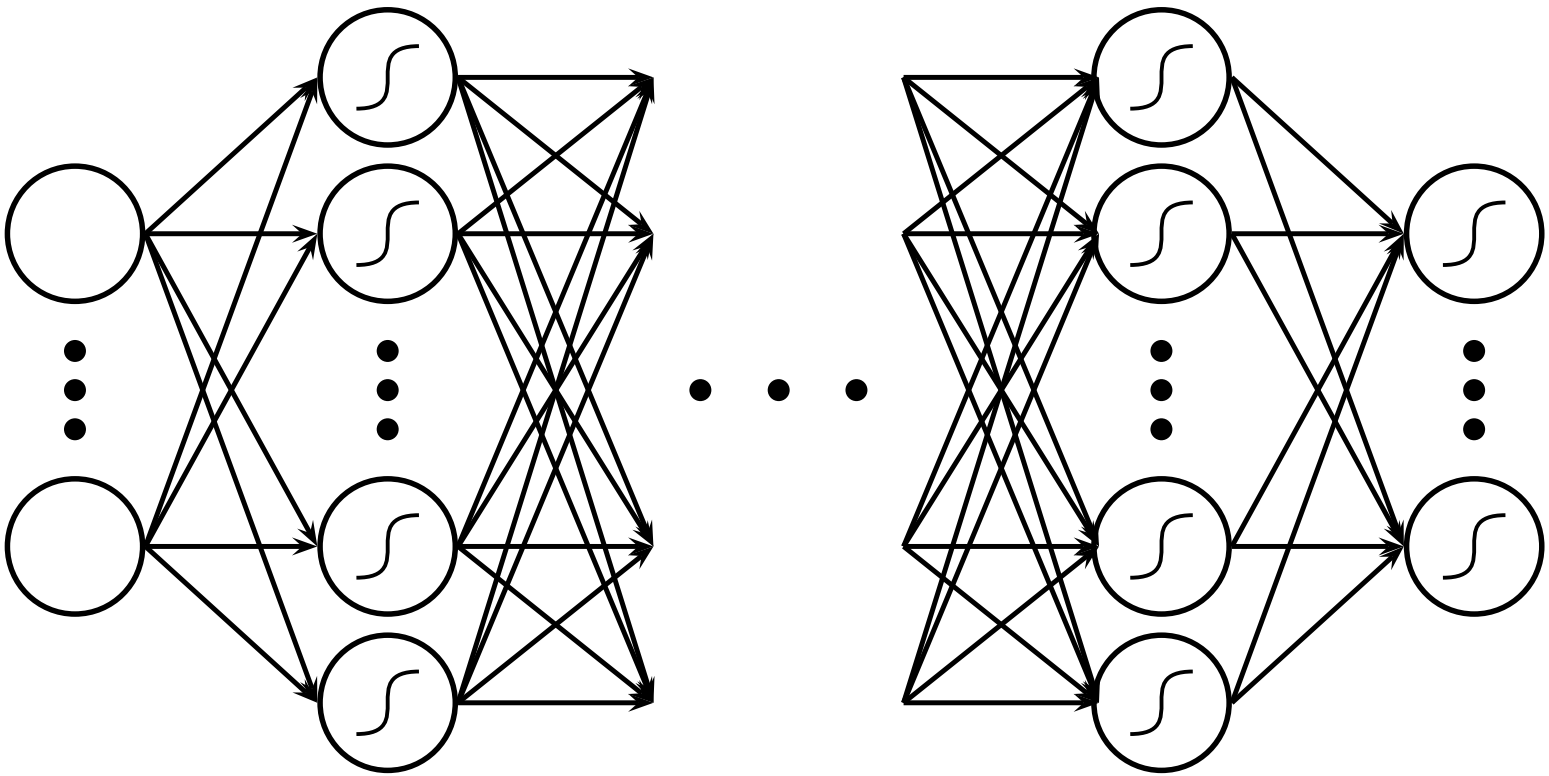
$aX^2 + bX + c$



$\frac{1}{1+\exp(-a(X-b))}$

Neural Networks

In neural networks, independent regression units are stacked together in layers, with layers stacked on top of each other



Supervised Training Approach

Calibration through neural networks

The calibration problem can been reduced to finding a neural network to approximate Θ. The problem is split into two: a training phase, which would normally be done offline, and the evaluation, which gives the model parameters for a given input

Training phase:

1. Collect large training set of calibrated examples
2. Propose neural network
3. Train, validate, and test it

Calibration of a model would then proceed simply by applying the previously trained Neural Network on the new input.

Supervised Training

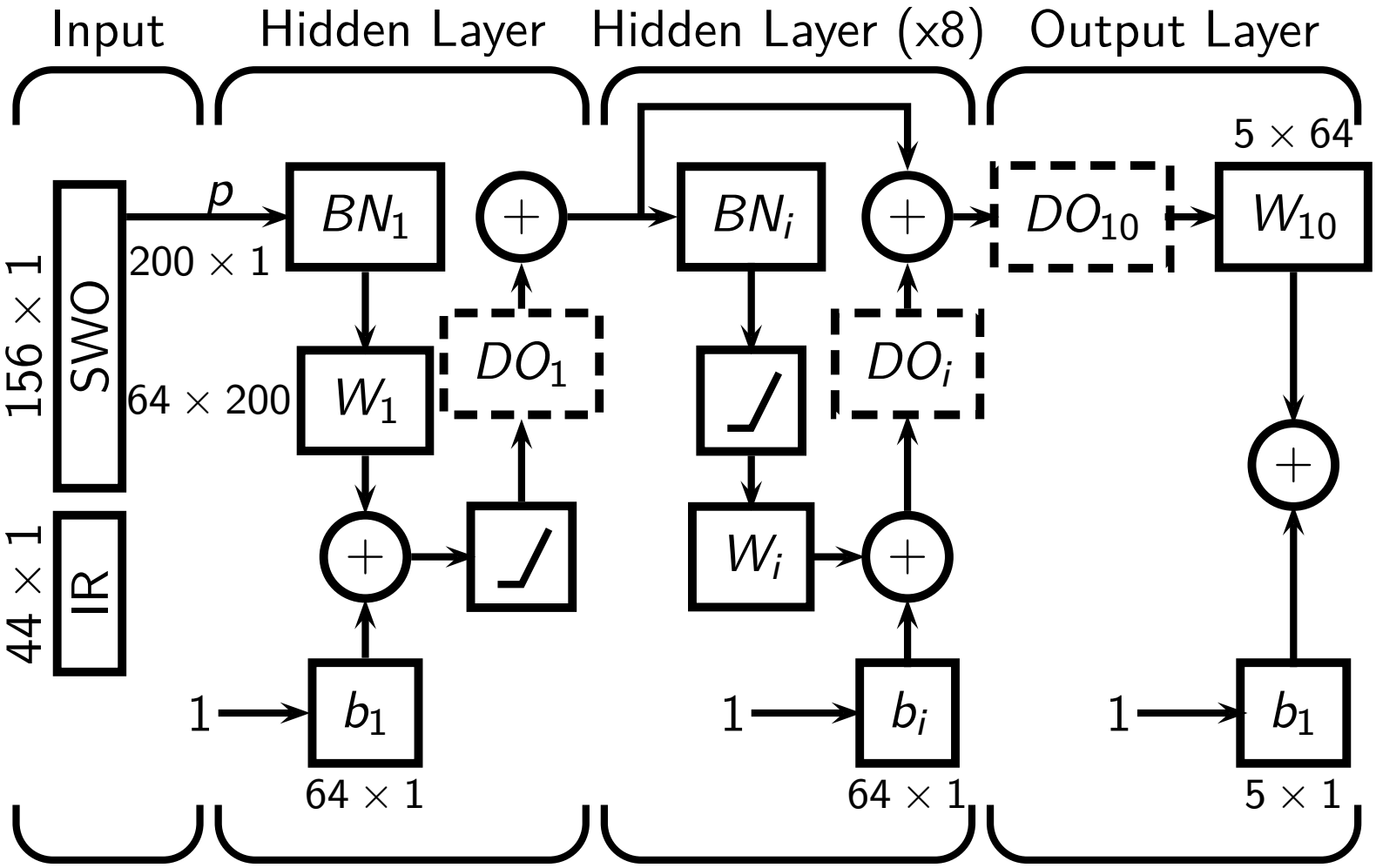
If one is provided with a set of associated input and output samples, one can 'train' the neural network's to best be able to reproduce the desired output given the known inputs.

The most common training method are variations of gradient descent. It consists of calculating the gradient, and moving along in the opposite direction. At each iteration, the current position is $\mathbf{x}_m$ is updated so

$$\mathbf{x}_{m+1} = \mathbf{x}_m - \gamma \nabla F(\mathbf{x}_m),$$

with γ called learning rate. What is used in practice is a form of stochastic gradient descent, where the parameters are not updated after calculating the gradient for all samples, but only for a random small subsample.

Feed-forward neural network for 2-factor HW



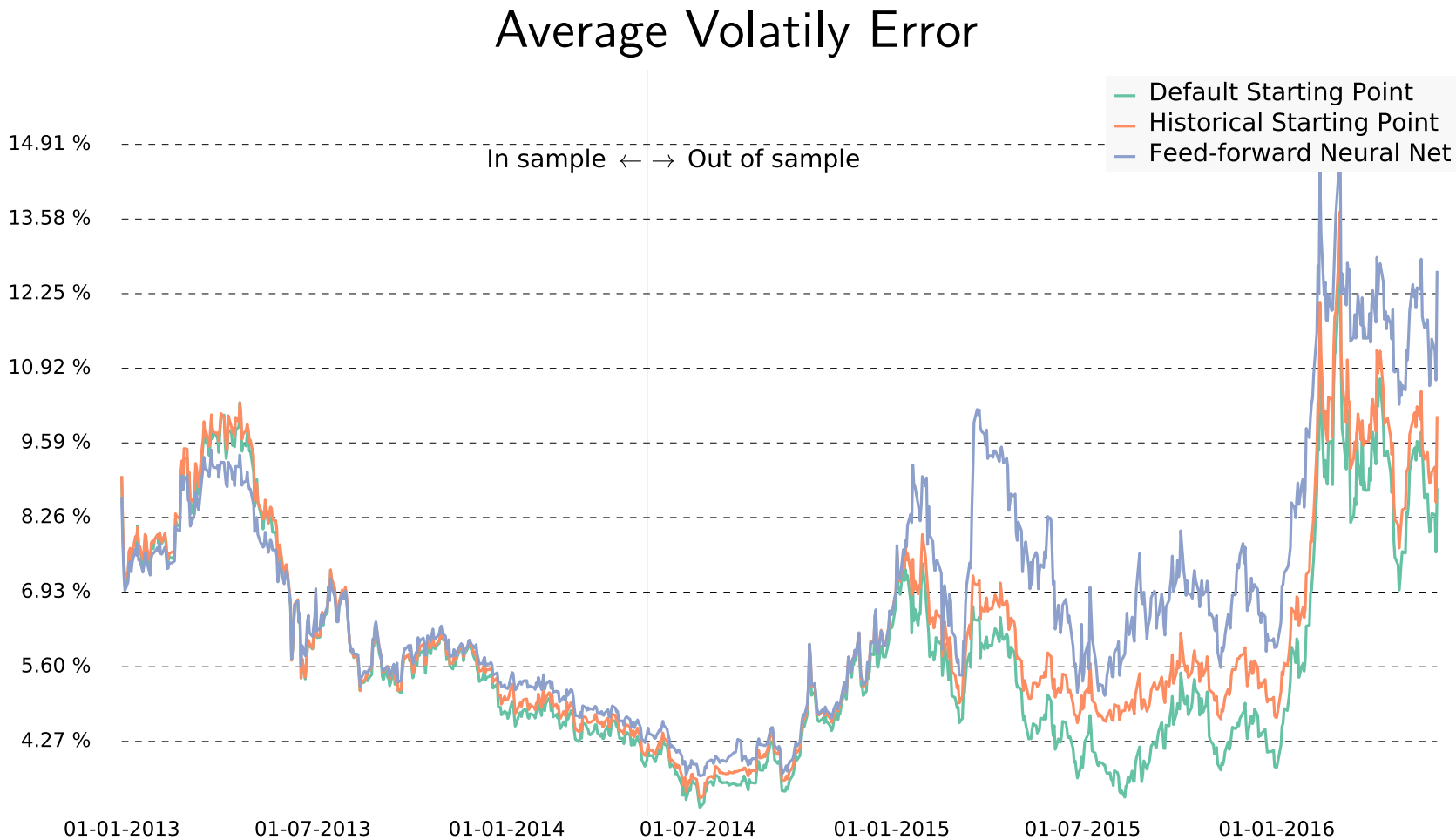
$a_1 = elu(W_1 \cdot BN(p) + b_1)$

$a_i = a_{i-1} + elu(W_i \cdot BN(a_{i-1}) + b_i)$

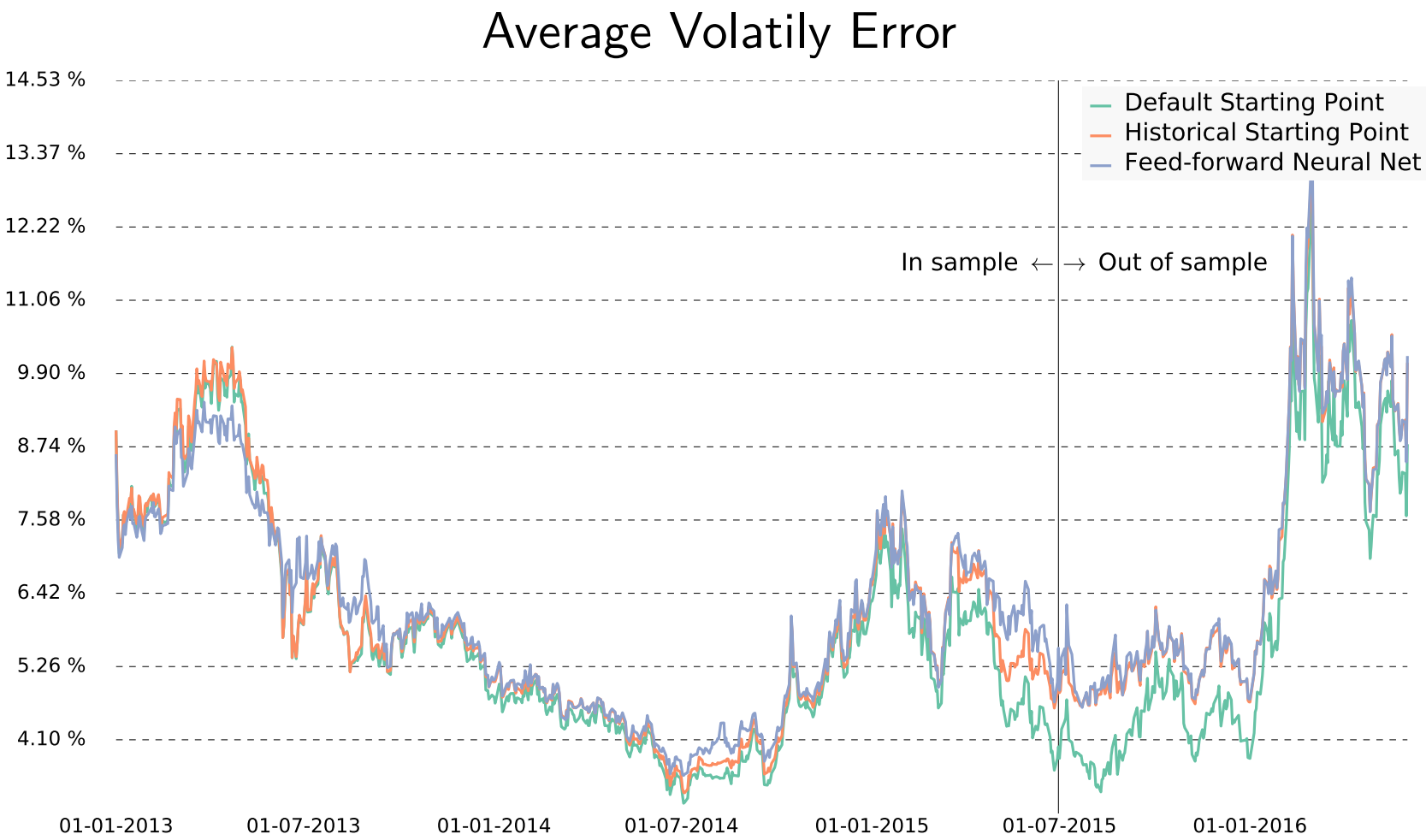
$a_{10} = W_{10} \cdot a_9 + b_{10}$

Hull-White 1-Factor: train from 01-2013 to 06-2014

Sample set created from historical examples from January 2013 to June 2014

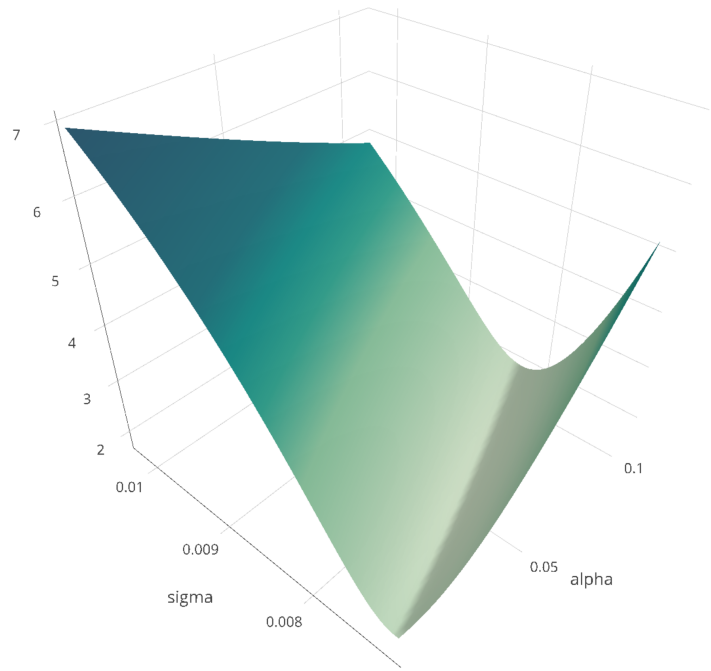


Hull-White 1-Factor: train from 01-2013 to 06-2015



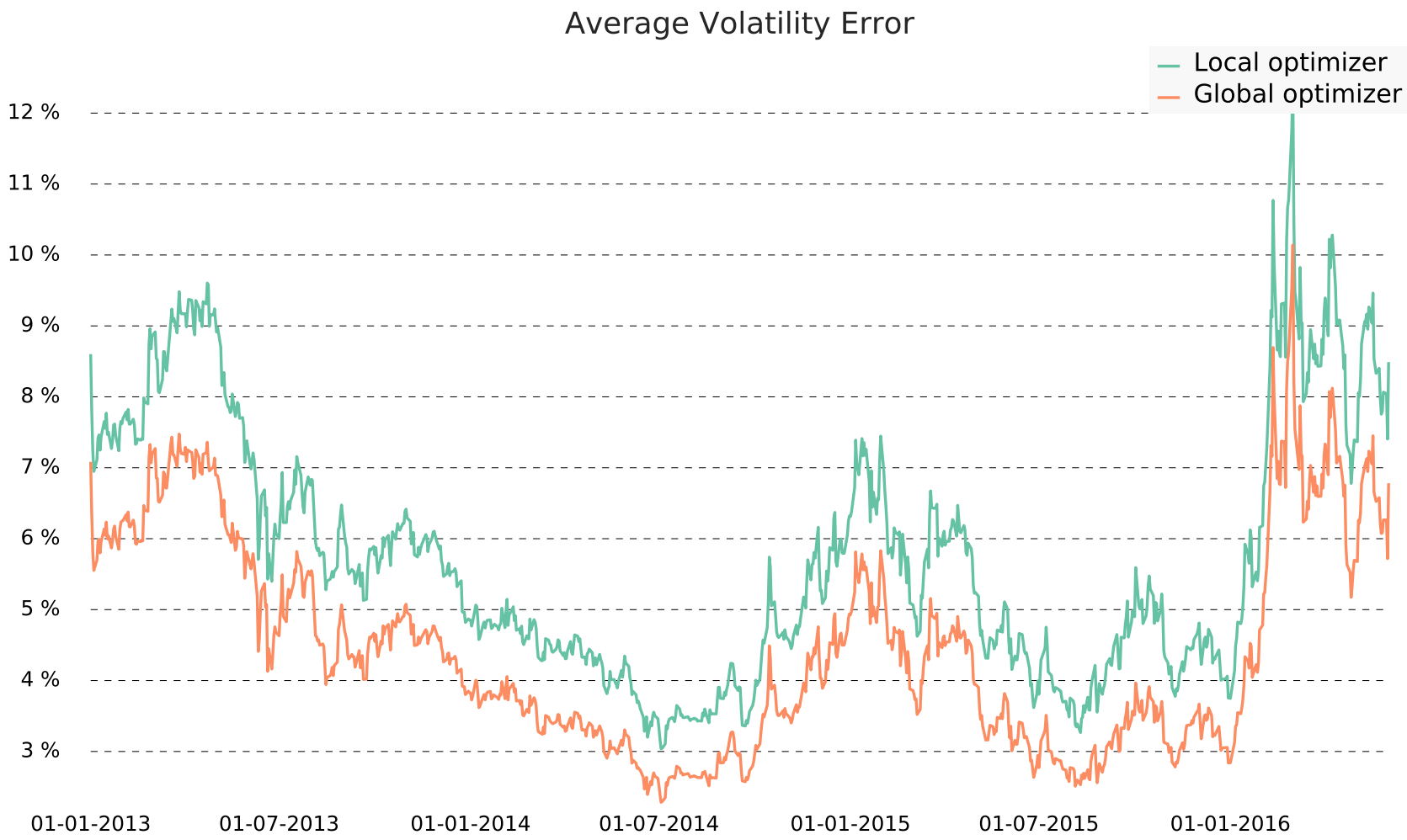
Cost Function on 01-07-2015

The historical point, lies on the trough. The default starting point ($\alpha = 0.1$, $\sigma = 0.01$) starts up on the side.

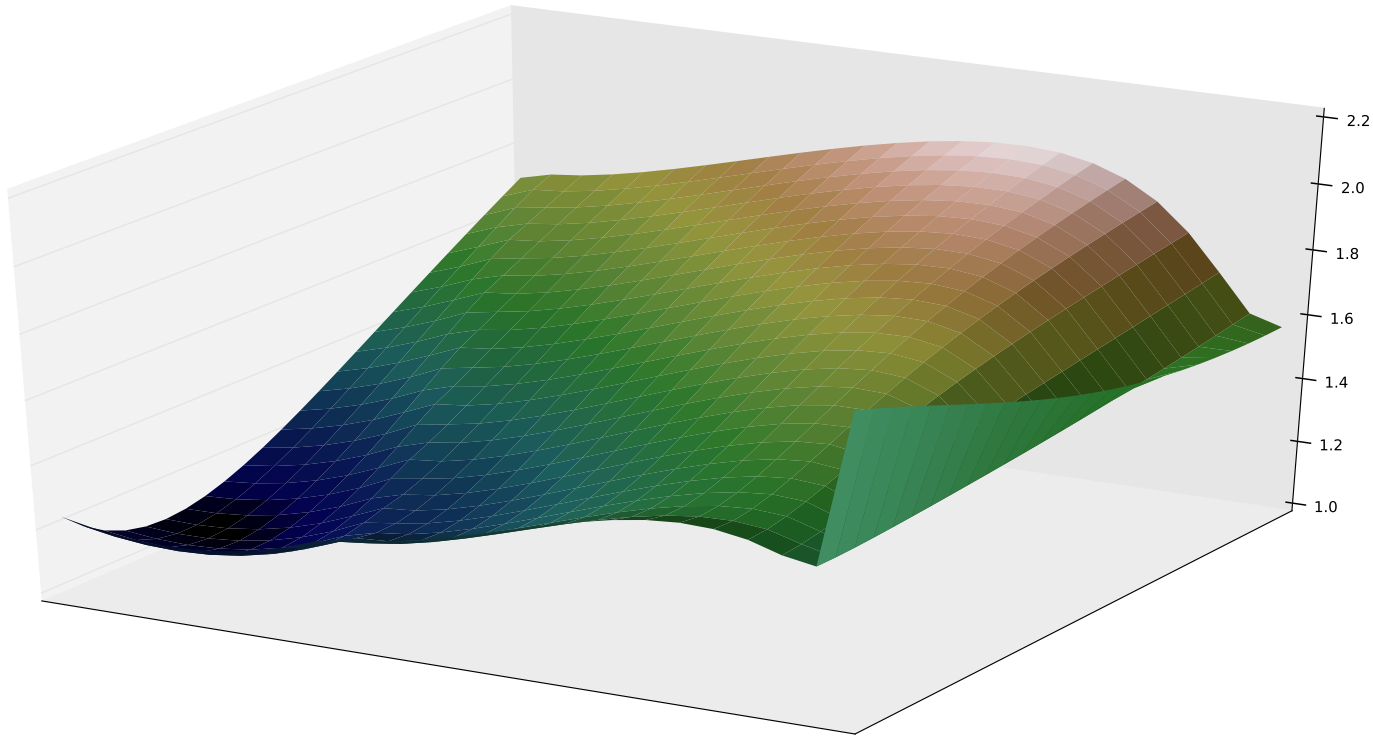


Hull-White 2-Factor

Comparison of local optimizer against global optimizer



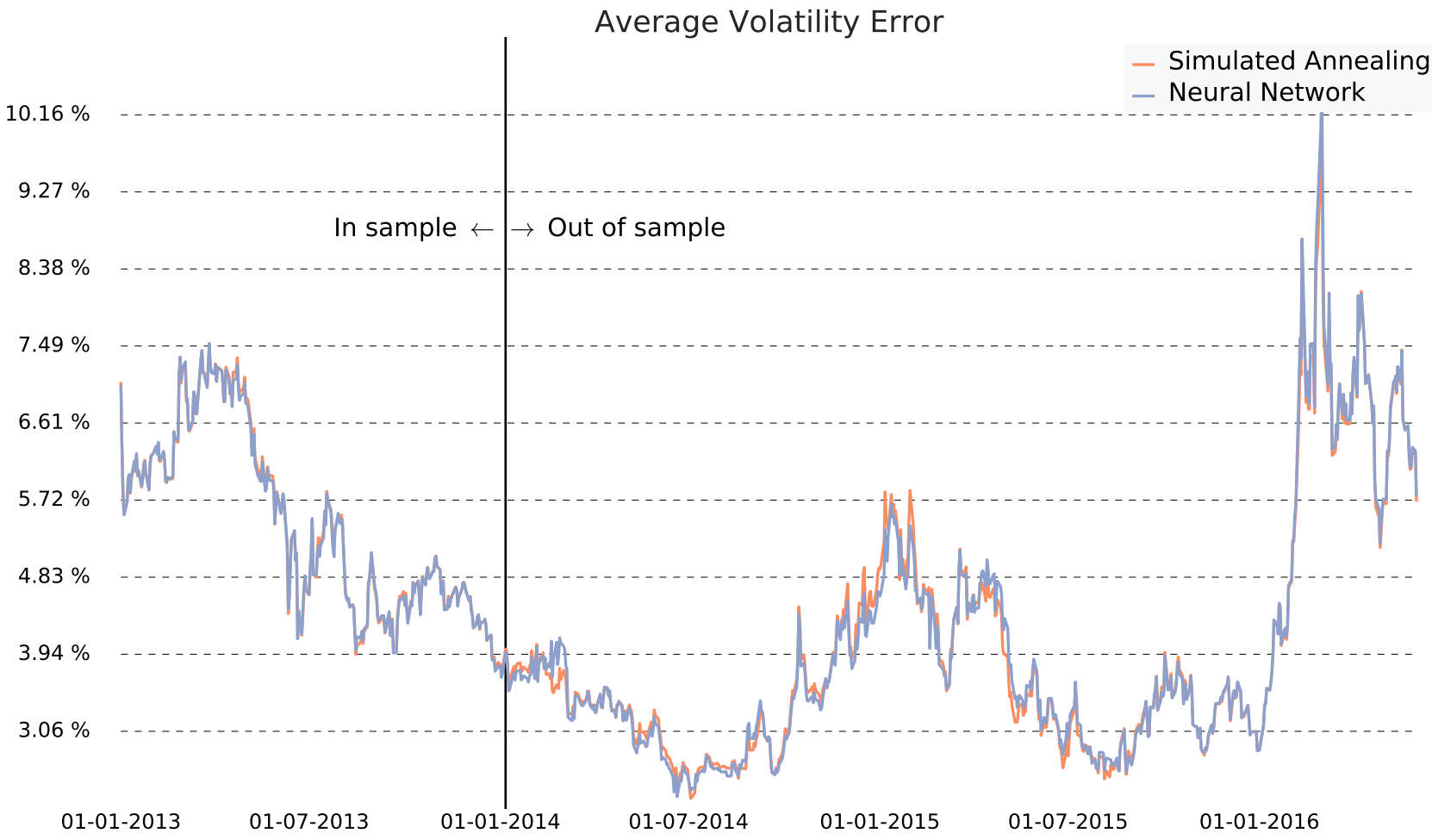
Hull-White 2-Factor - Global vs local optimizer



The above shows the plane defined by the global minimum, the local minimum, and the default starting point.

Hull-White 2-Factor - retrained every 2 months

To train, a 1-year rolling window is used.



Generating Training Set

The large training set has not yet been discussed. Taking all historical values and calibrating could be a possibility. However, the inverse of $\mathbf{\Theta}$ is known, it is simply the regular valuation of the instruments under a given set of parameters

$$\{\mathbf{Q}\} = \mathbf{\Theta}^{-1}(\alpha, \sigma; \{\tau\}, y(t))$$

This means that we can generate new examples by simply generating random parameters α and σ. There are some complications, e.g. examples of $y(t)$ also need to be generated, and the parameters and $y(t)$ need to be correlated properly for it to be meaningful.

Generating Training Set

The intention is to collect historical examples, and imply some kind of statistical model from them, and then draw from that distribution.

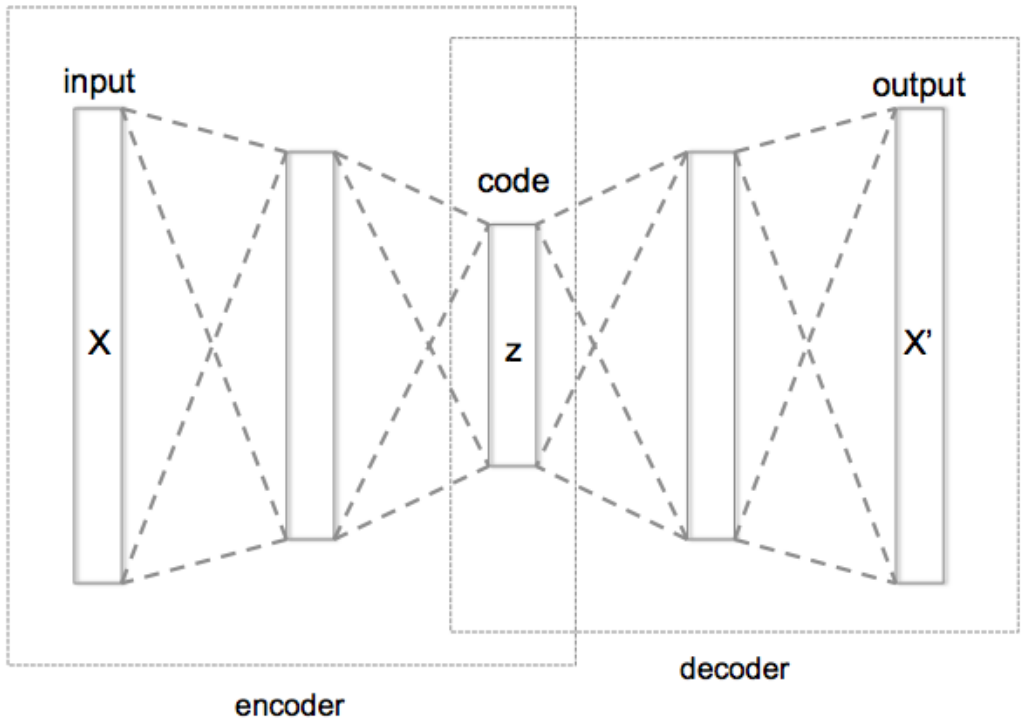
1. Calibrate model for training history
2. Obtain errors for each instrument for each day
3. As parameters are positive, take logarithm on the historical values
4. Rescale yield curves, parameters, and errors to have zero mean and variance 1
5. Apply dimensional reduction via PCA to yield curve, and keep parameters for given explained variance (e.g. 99.5%)

Generating Training Set - From normal distribution

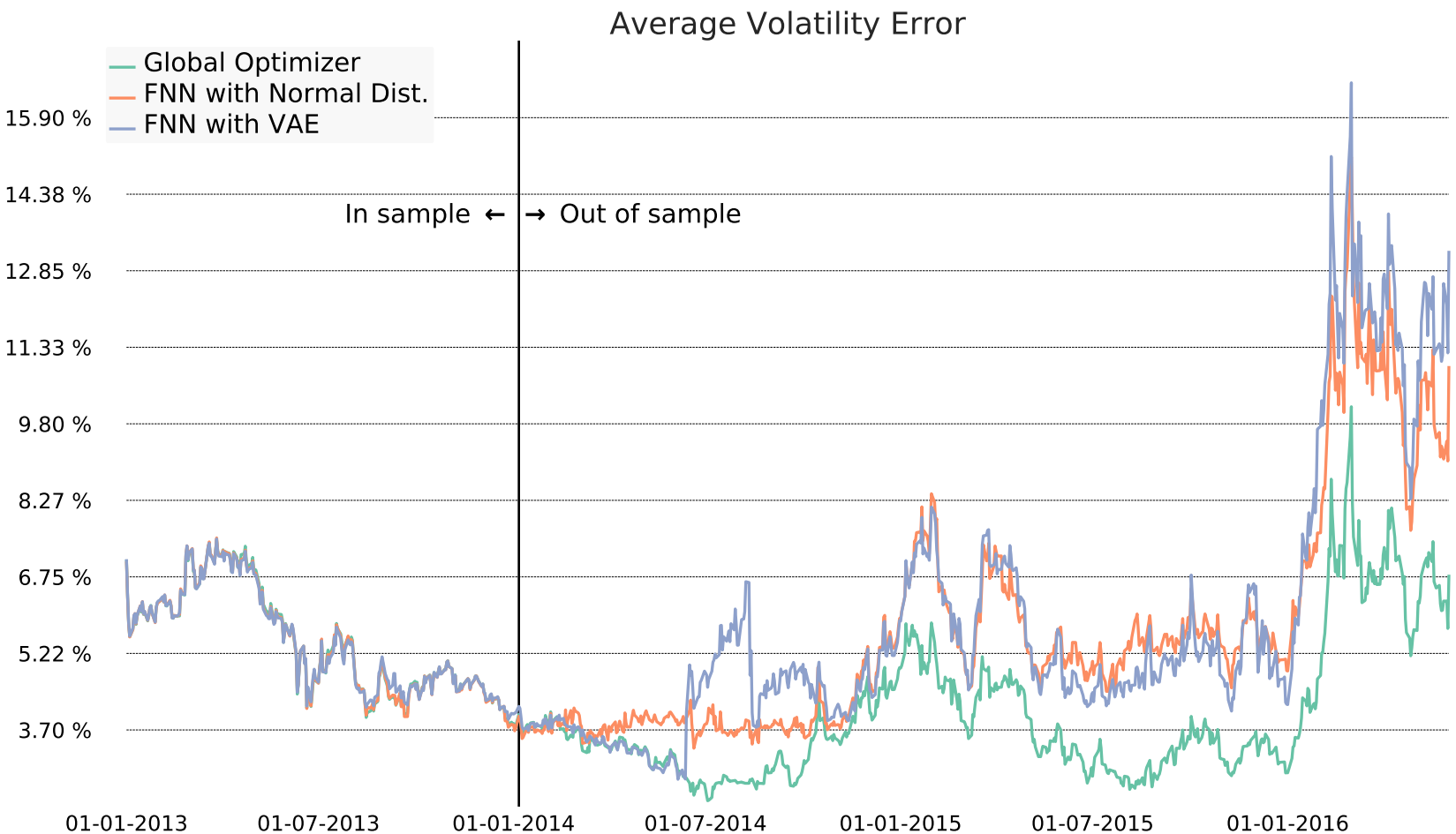
6. Calculate covariance of rescaled log-parameters, PCA yield curve values, and errors
7. Generate random normally distributed vectors consistent with given covariance
8. Apply inverse transformations: rescale to original mean, variance, and dimensionality, and take exponential of parameters
9. Select reference date randomly
10. Obtain implied volatility for all swaptions, and apply random errors

Generating Training Set - Variational autoencoder

Variational autoencoders learn a latent variable model that parametrizes a probability distribution of the output contingent on the input.



Normal distribution vs variational autoencoder (no retraining)

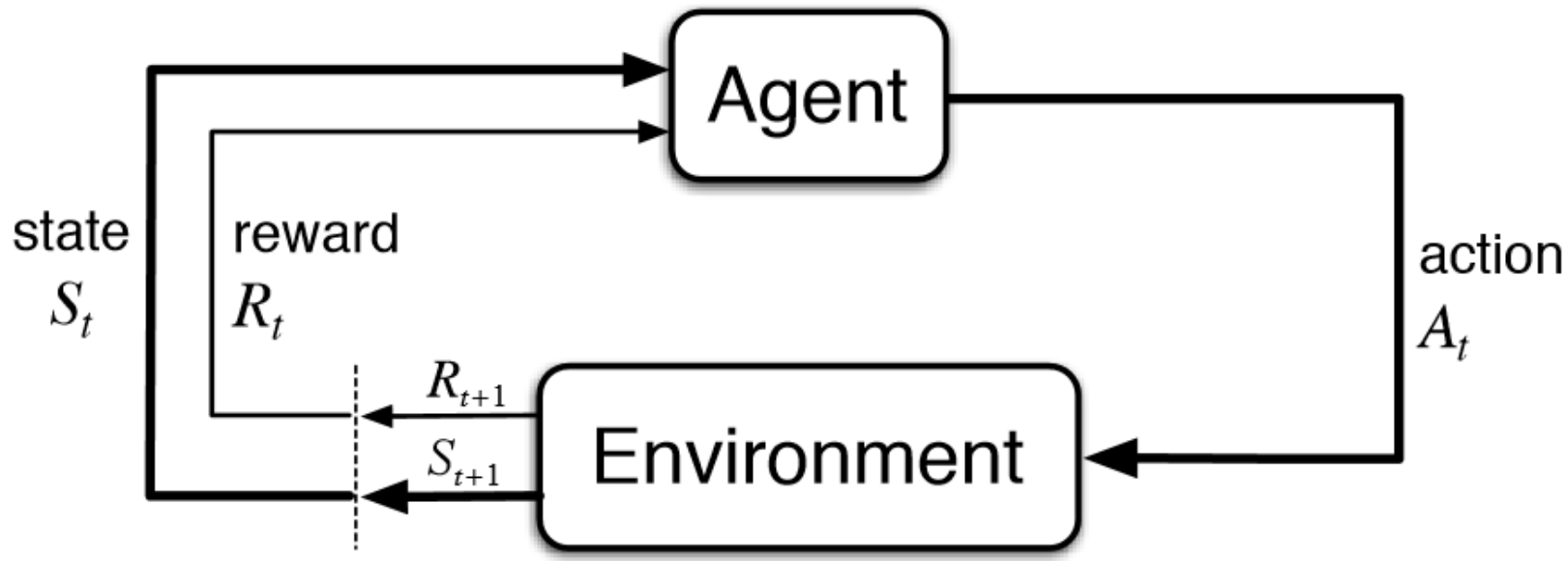


Unsupervised Training Approach

Bespoke optimizer

But what about the case where one doesn't have a long time-series?

Reinforcement learning can be used to create better bespoke optimizers than the traditional local or global optimization procedures.



Deep-q learning

A common approach for reinforcement learning with a large possibility of actions and states is called Q-Learning:

An agent's behaviour is defined by a policy π, which maps states to a probability distribution over the actions $\pi : S \rightarrow \mathcal{P}(\mathcal{A})$.

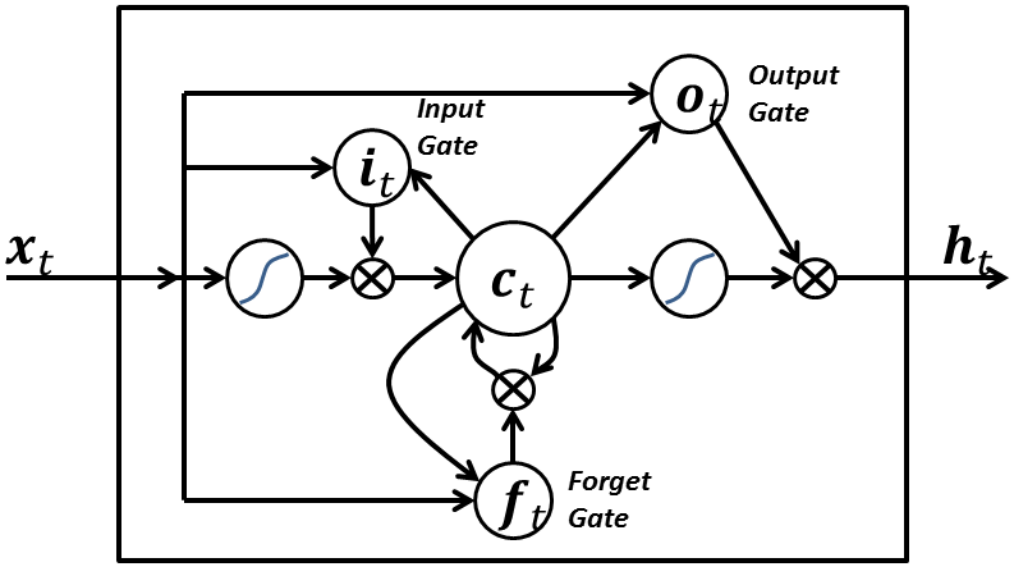
The return R_t from an action is defined as the sum of discounted future rewards $R_t = \sum_{i=t}^{T} \gamma^{i-t} r(s_i, a_i)$.

The quality of an action is the expected return of an action a_t in state s_t

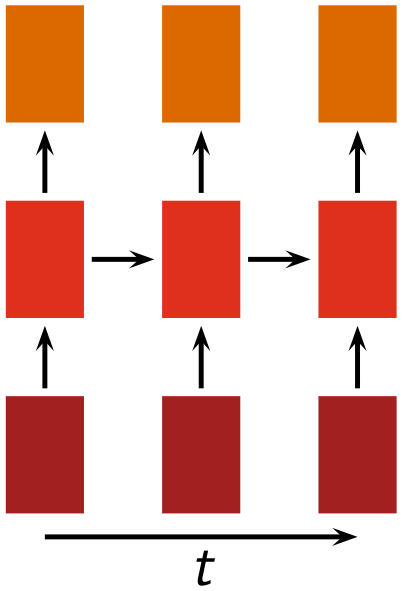
$$Q^{\pi}(a_t, s_t) = \mathbb{E}_{r_i \geq t, s_i > t, a_i > t} \left[R_t | s_t, a_t \right]$$

Learning to learn without gradient descent with gradient descent

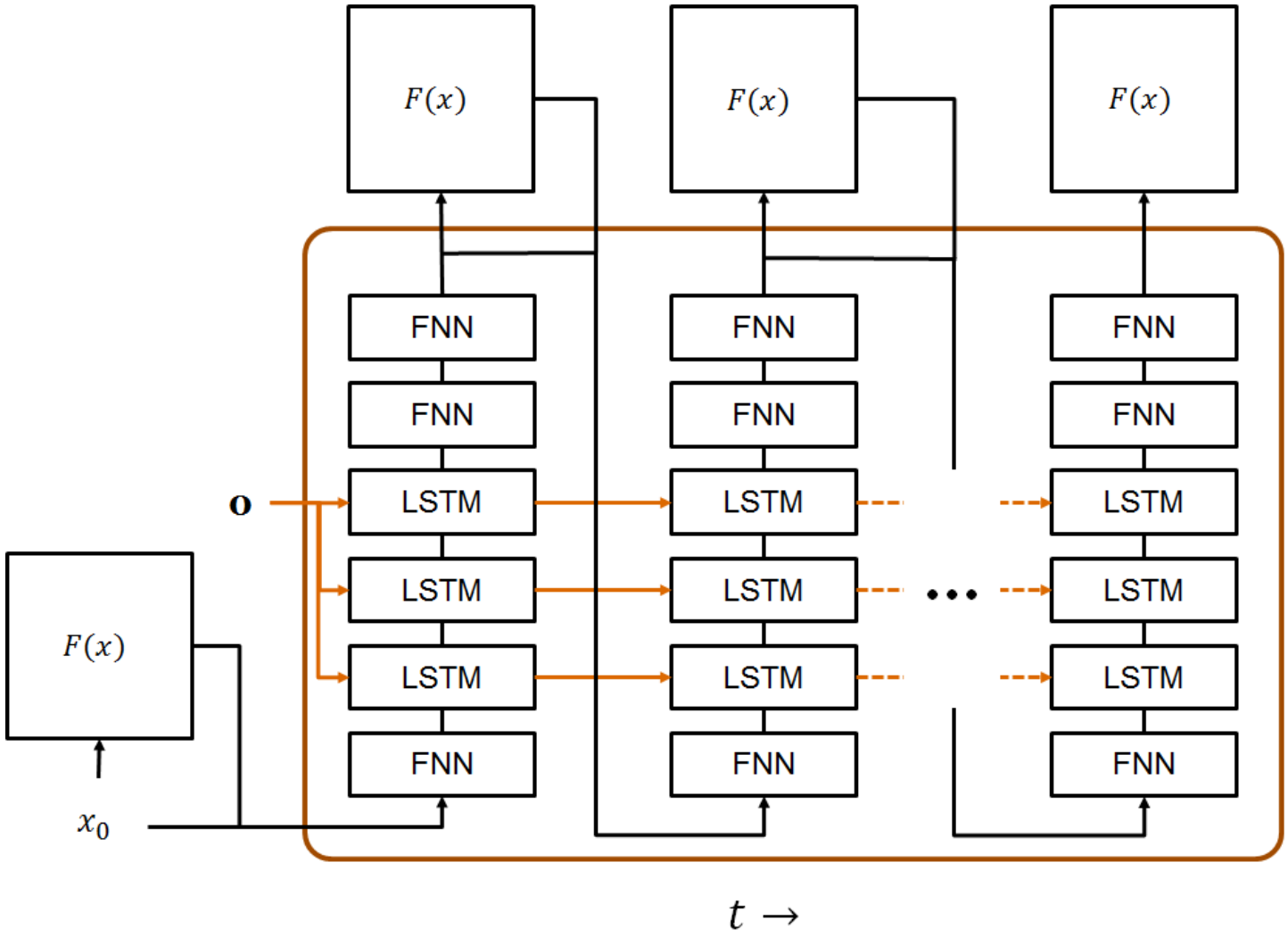
A long-short-term memory (LSTM) architecture was used to simplify represent the whole agent. The standard LSTM block is composed of several gates with an internal state:



In the current case, 100 LSTM blocks were used per layer, and 3 layers were stacked on top of each other

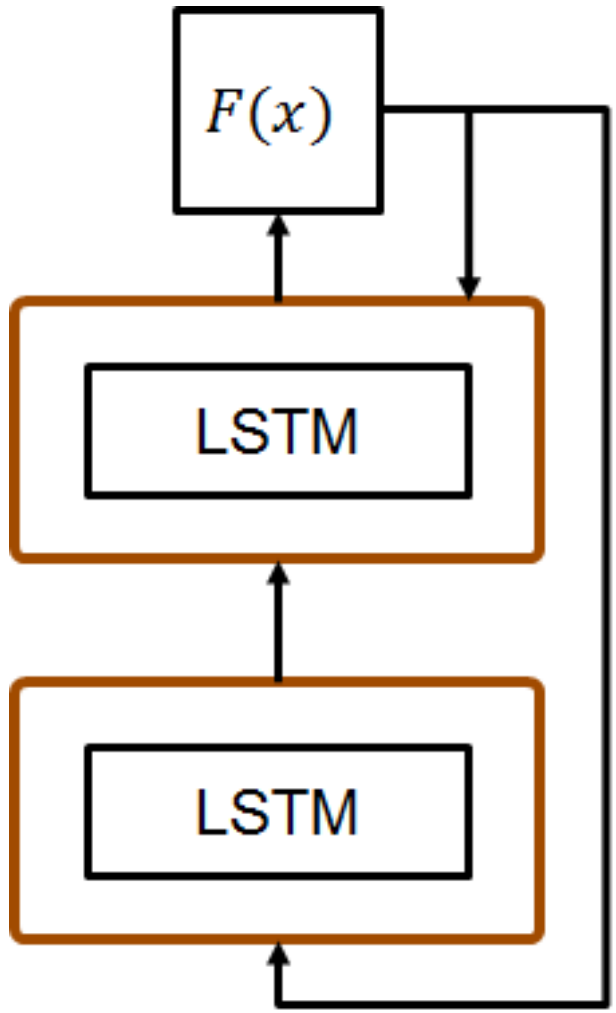


Unrolled recurrent network

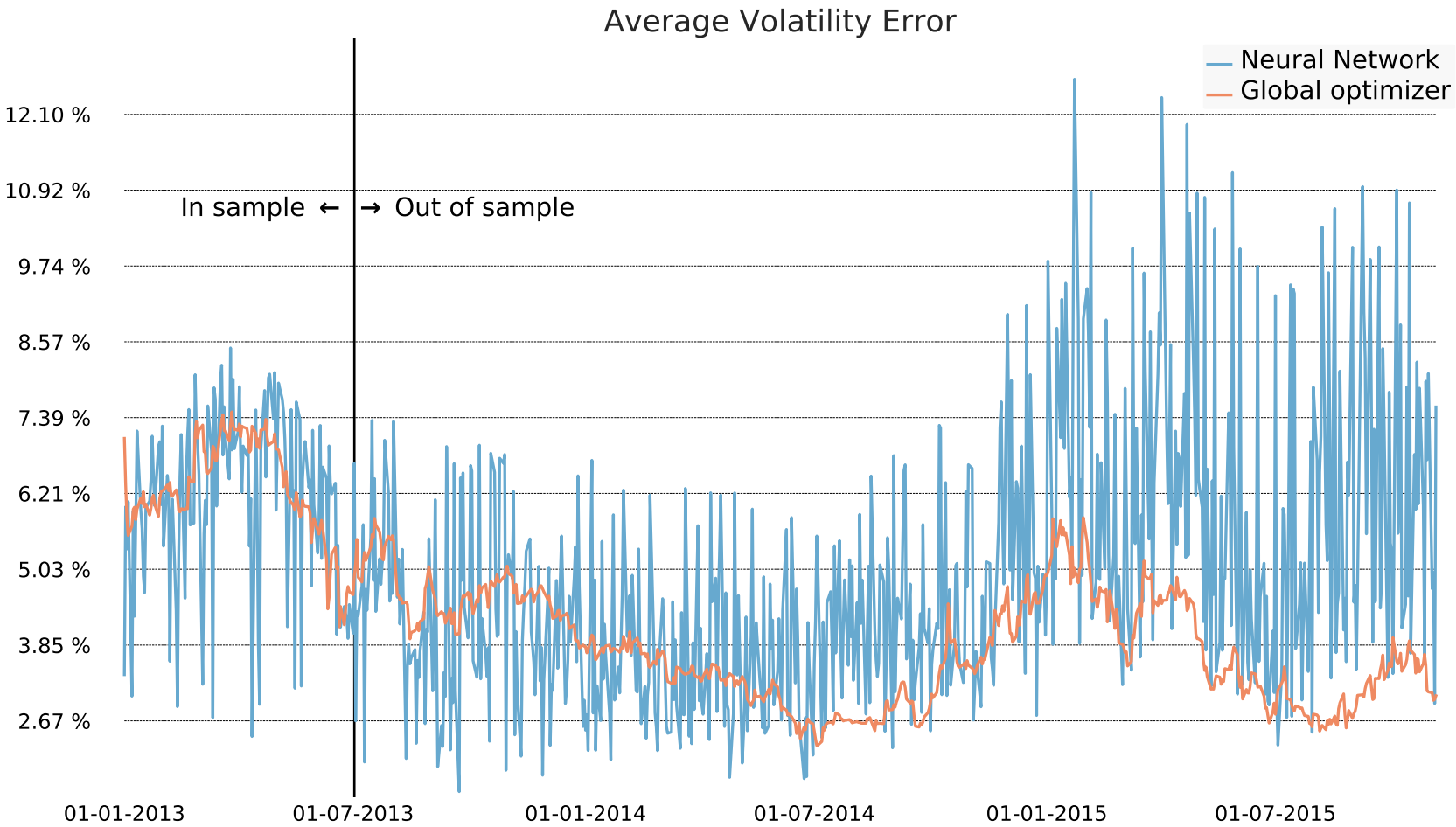


Train the optimizer

- Train it with approximation of $F(x)$, whose gradient is available
  - Advantage: training proceeds fast
  - Disadvantage: potentially will not reach full possibility
- Train it with non-gradient based optimizer
  - Local optimizer: generally requires a number of evaluations $\sim$ to number of dimensions to take next step
  - Global optimizer: very hard to set hyperparameters
- Train a second NN to train first NN



Bespoke optimizer



References

- `https://github.com/Andres-Hernandez/CalibrationNN`
- A. Hernandez, *Model calibration with neural networks*, Risk, June 2017
- A. Hernandez, *Model Calibration: Global Optimizer vs. Neural Network*, SSRN abstract id=2996930
- Y. Chen, et al *Learning to Learn without Gradient Descent by Gradient Descent*, arXiv:1611.03824

Future work

- Calibration of local stochastic volatility model. Work is being undertaken in collaboration with Professors J. Teichmann from the ETH Zürich, and C. Cuchiero in University of Wien, and W. Khosrawi-Sardroudi from the University of Freiburg.
- Improvement of bespoke optimizers, in particular train with more random environment: different currencies, constituents, etc.
- Use of bespoke optimizer as large-dimensional PDE solver