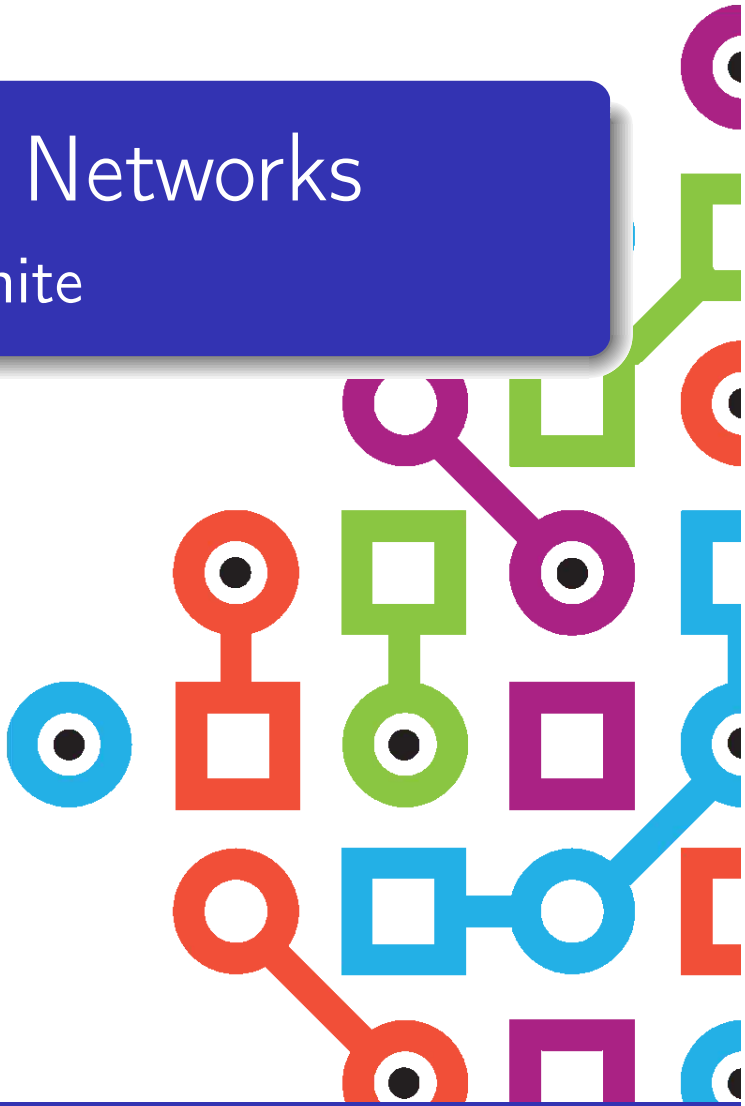
Calibration with Neural Networks

Example with Hull-White

Andres Hernandez

IBM Risk Analytics

July 12, 2016



Motivation

A central consideration of any pricing model is the ability to calibrate that model to market prices. Whether the necessary information, e.g. correlation, can be effectively implied from the data or not is one part of this, but also the speed with which that calibration can be done influences the usability of a model.

Motivation

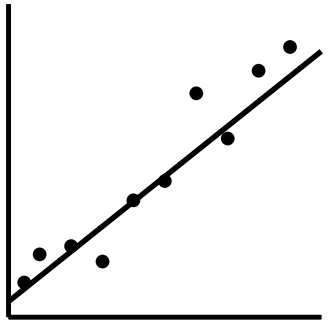
The point of this talk is to provide a method that will perform the calibration significantly faster regardless of the model, hence removing the calibration speed from a model's practicality.

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

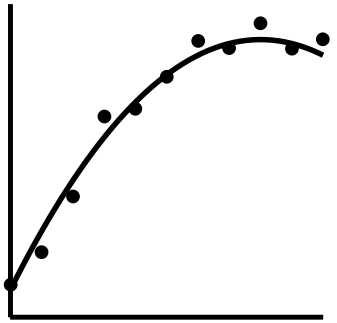
Table of contents

ANN

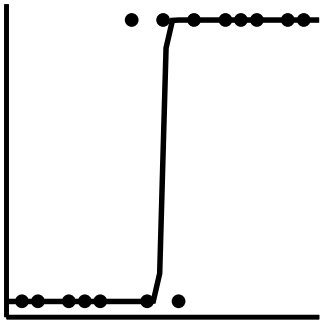
Artificial neural networks (ANN) are a family of machine learning techniques, which are currently used in state-of-the-art solutions for image and speech recognition, and natural language processing.
In general, artificial neural networks are an extension of regression



$aX + b$



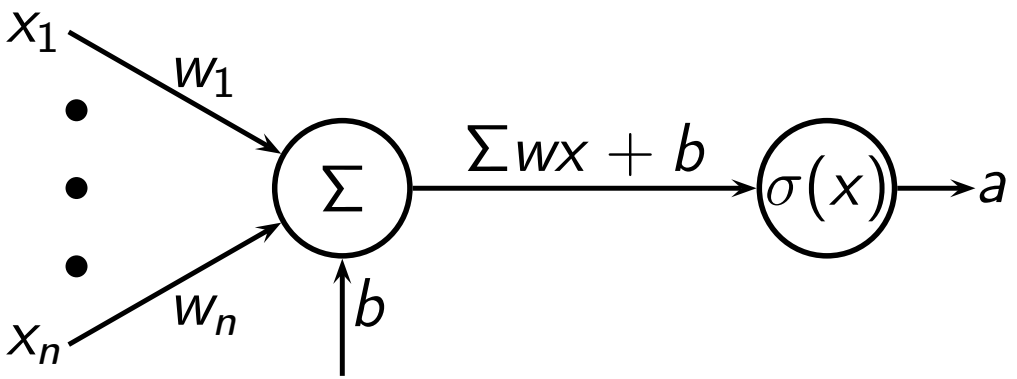
$aX^2 + bX + c$



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

Neuron

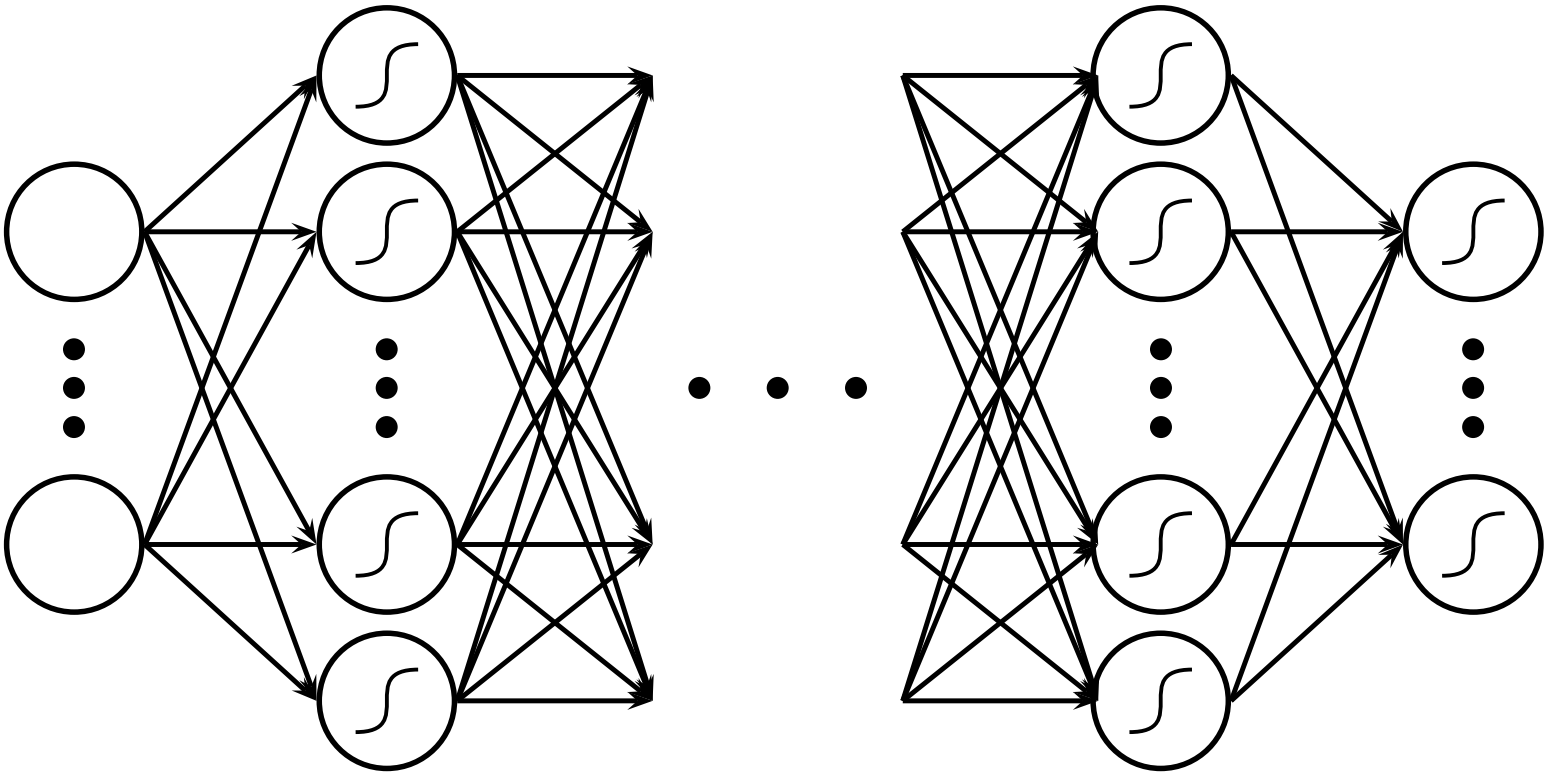
An ANN is simply a network of regression units stacked in a particular configuration. Each regression unit, called a neuron, takes input from the previous layer [1], combines that input according to a rule, and applies a function on the result:



[1] There are more complicated topologies, e.g. Recursive Neural Networks or Restricted Boltzmann machine

ANN

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



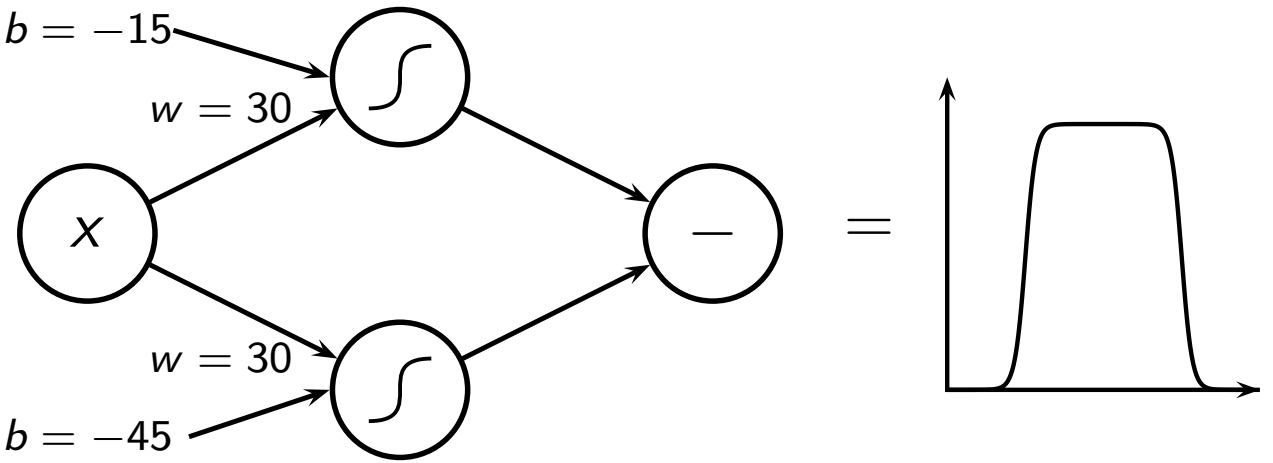
Universal approximation theorem

Neural networks are generally used to approximate a function, usually one with a large number of input parameters.

To justify their use, besides practical results, one relies on the Universal Approximation theorem, which states that a continuous function in a compact domain can be arbitrarily approximated by a finite number of neurons with slight restrictions on the activation function.

Universal approximation theorem

It should be easy to accept it as plausible, considering that a step function can be formed by taking the difference of two sigmoid neurons



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 usually used is some 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.

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 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

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

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

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

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

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

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.

Definition

The calibration problem is then reduced to finding a neural network to approximate Θ. The problem is furthermore 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.

Hull-White Model

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

$$dr_t = (\theta(t) - \alpha r_t)dt + \sigma dW_t$$

with α and σ constant. $\theta(t)$ is normally picked to replicate the current curve $y(t)$.

The problem is then

$$(\alpha, \sigma) = \Theta\left(\{\hat{\mathbf{Q}}\}; \{\tau\}, y(t)\right)$$

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

Generating Training Set

What we still need is a large training set. 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

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

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

Generating the training set:

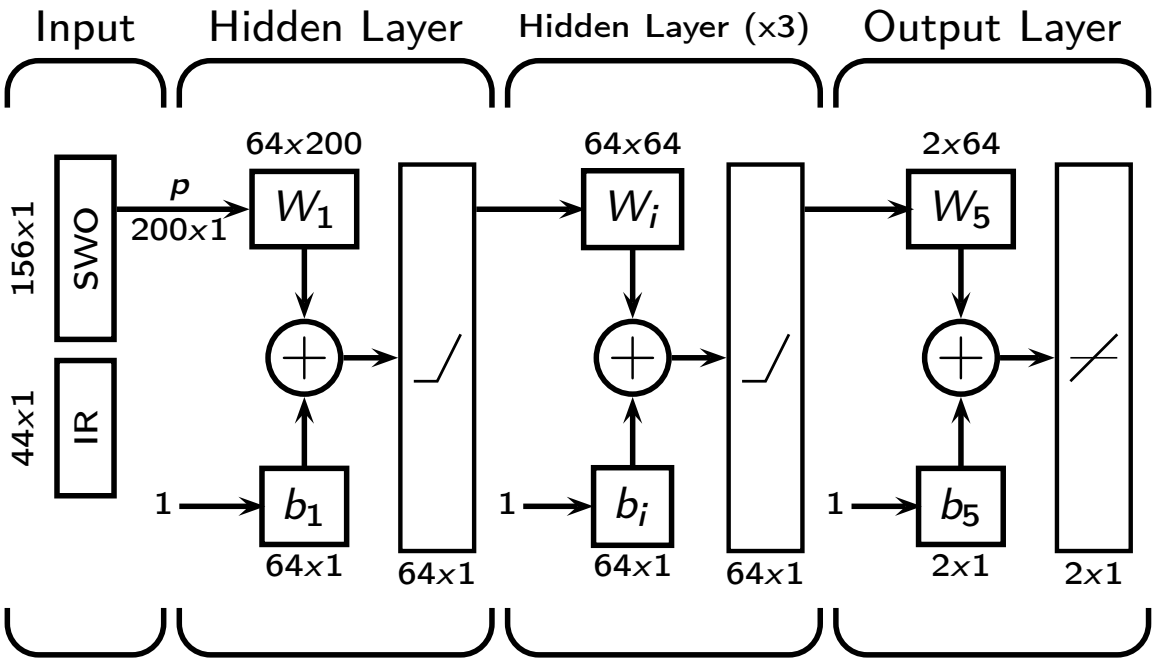
1. Calibrate model for training history
2. Obtain absolute errors for each instrument for each day
3. As parameters are positive, take logarithm on the historical values
4. Rescale yield curves, parameters, and errors to have zero mean and variance 1
5. Apply dimensional reduction via PCA to yield curve,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

Generating Training Set

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

Simple FNN

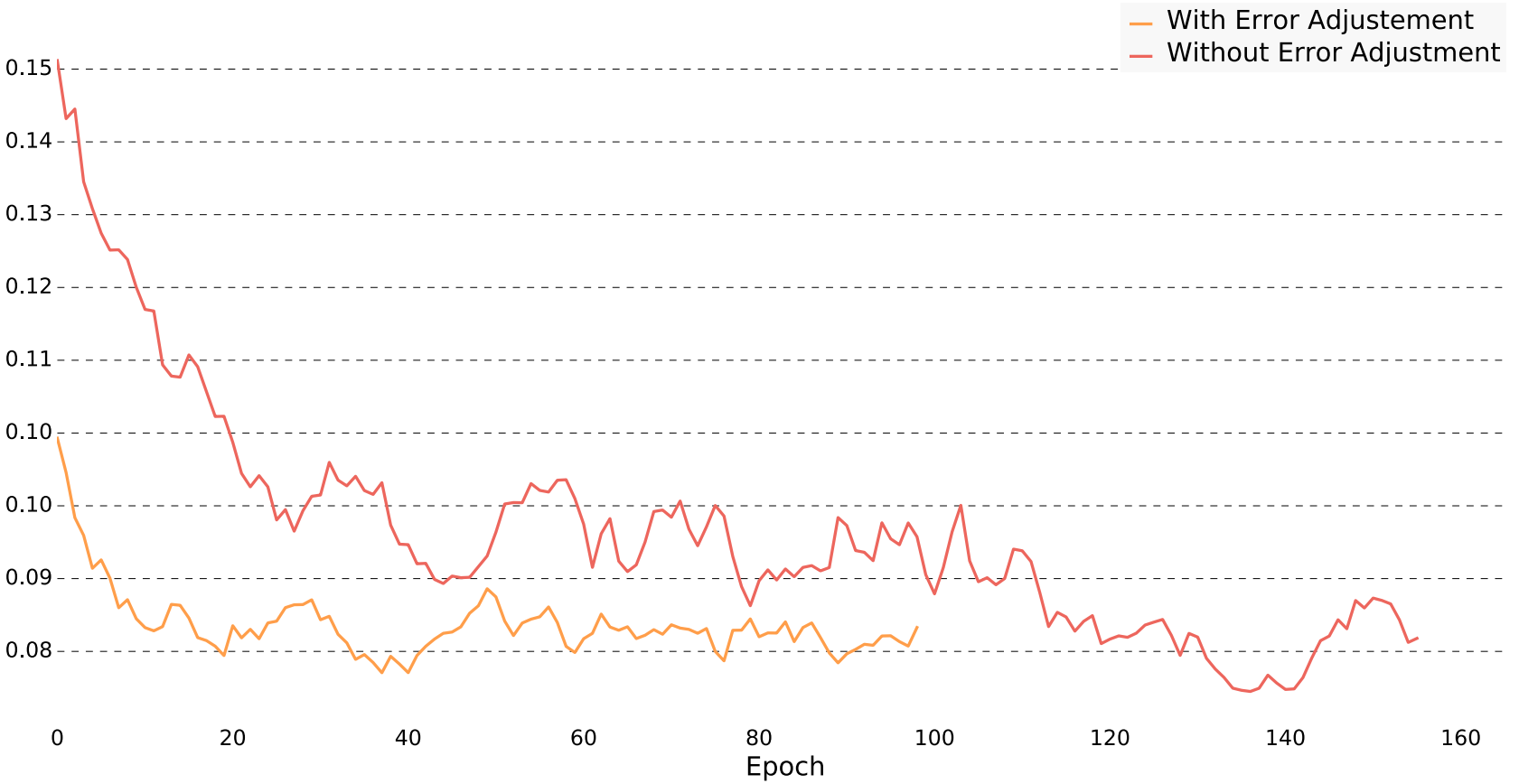
The network used was a 'simple' feed-forward neural network:



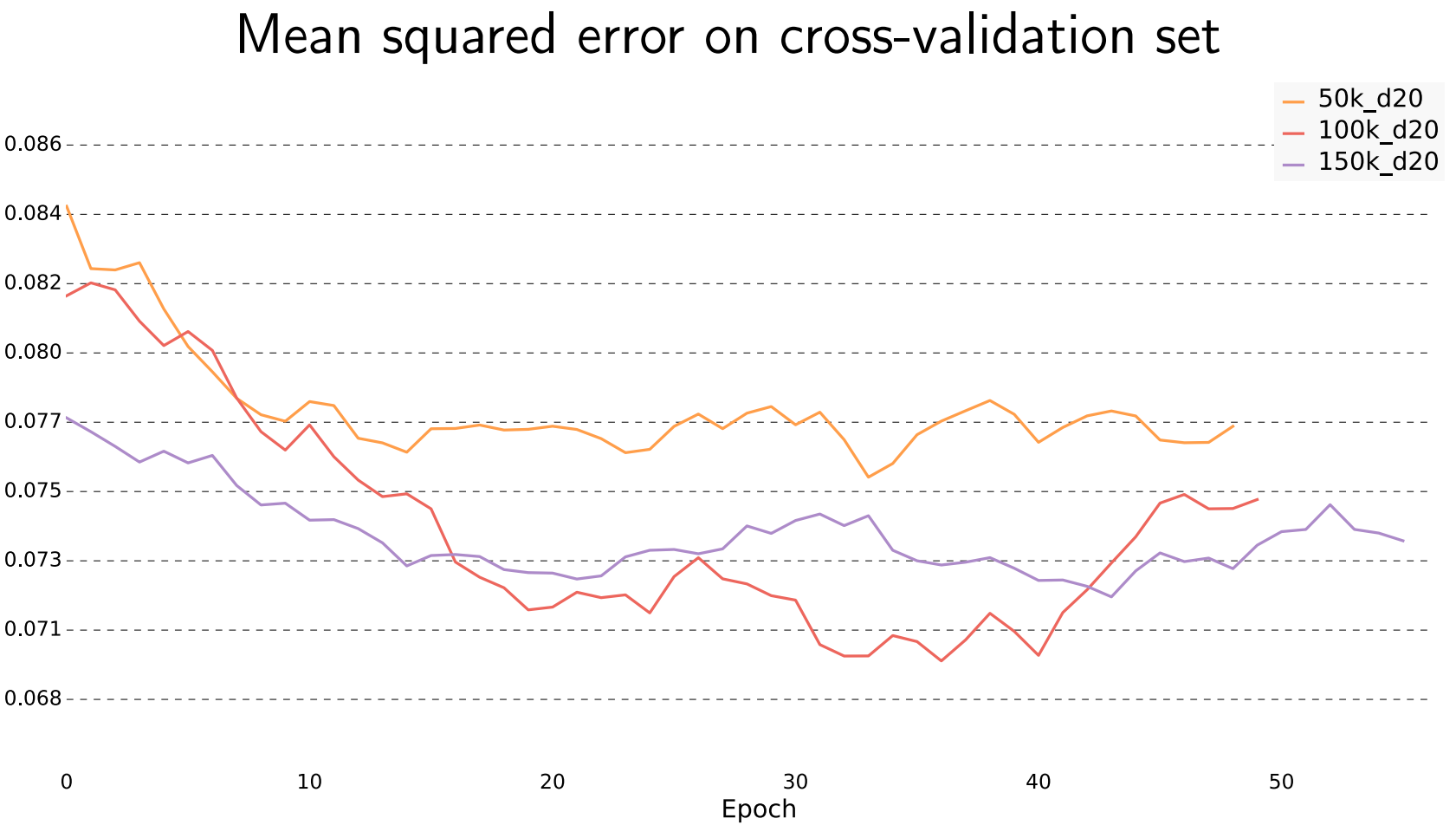
$$a_1 = elu(W_1 p + b_1) \quad a_i = elu(W_i a_{i-1} + b_i) \quad a_5 = W_5 a_4 + b_5$$

Training

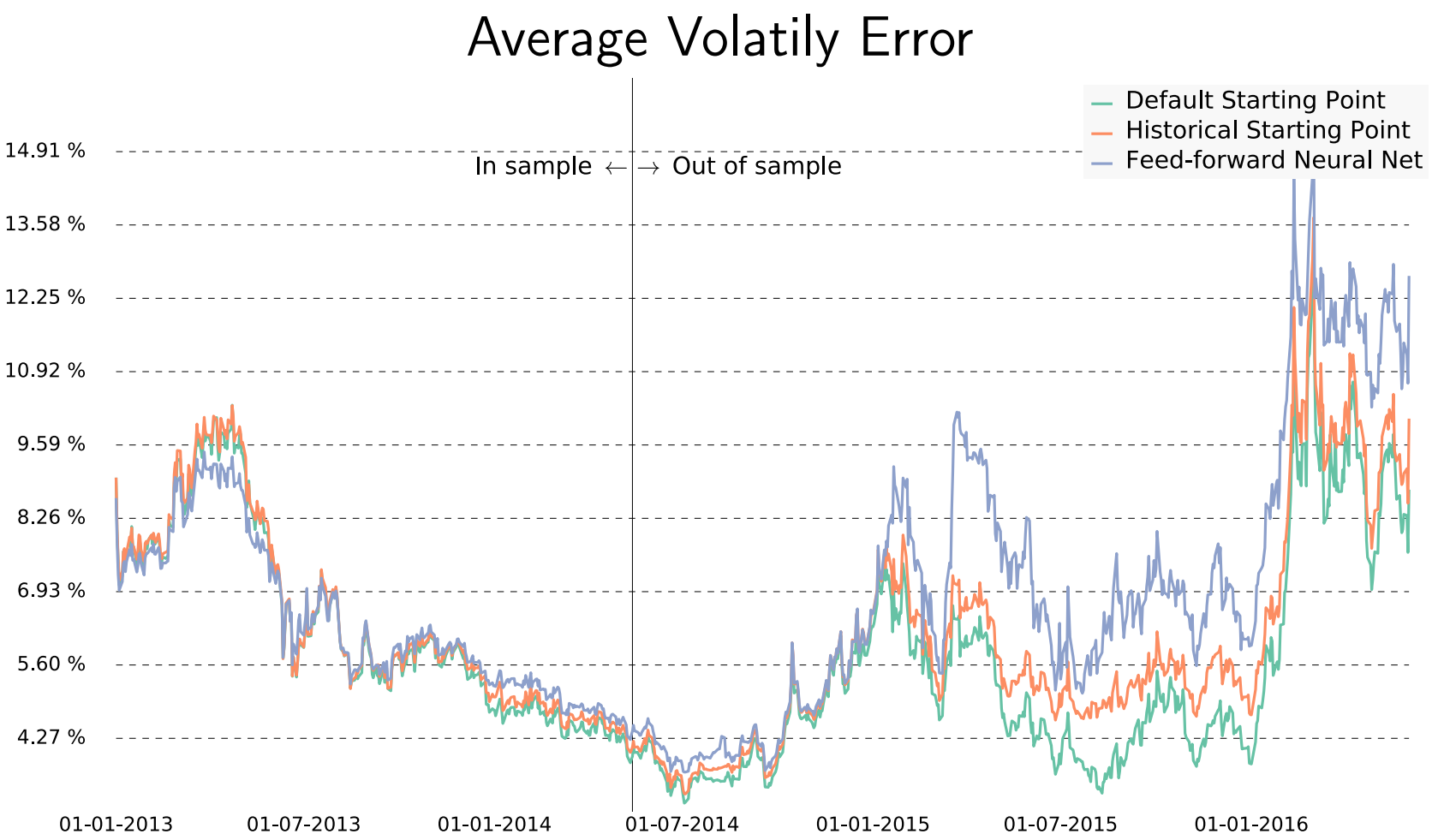
Mean squared error on cross-validation set (with 50k total set)



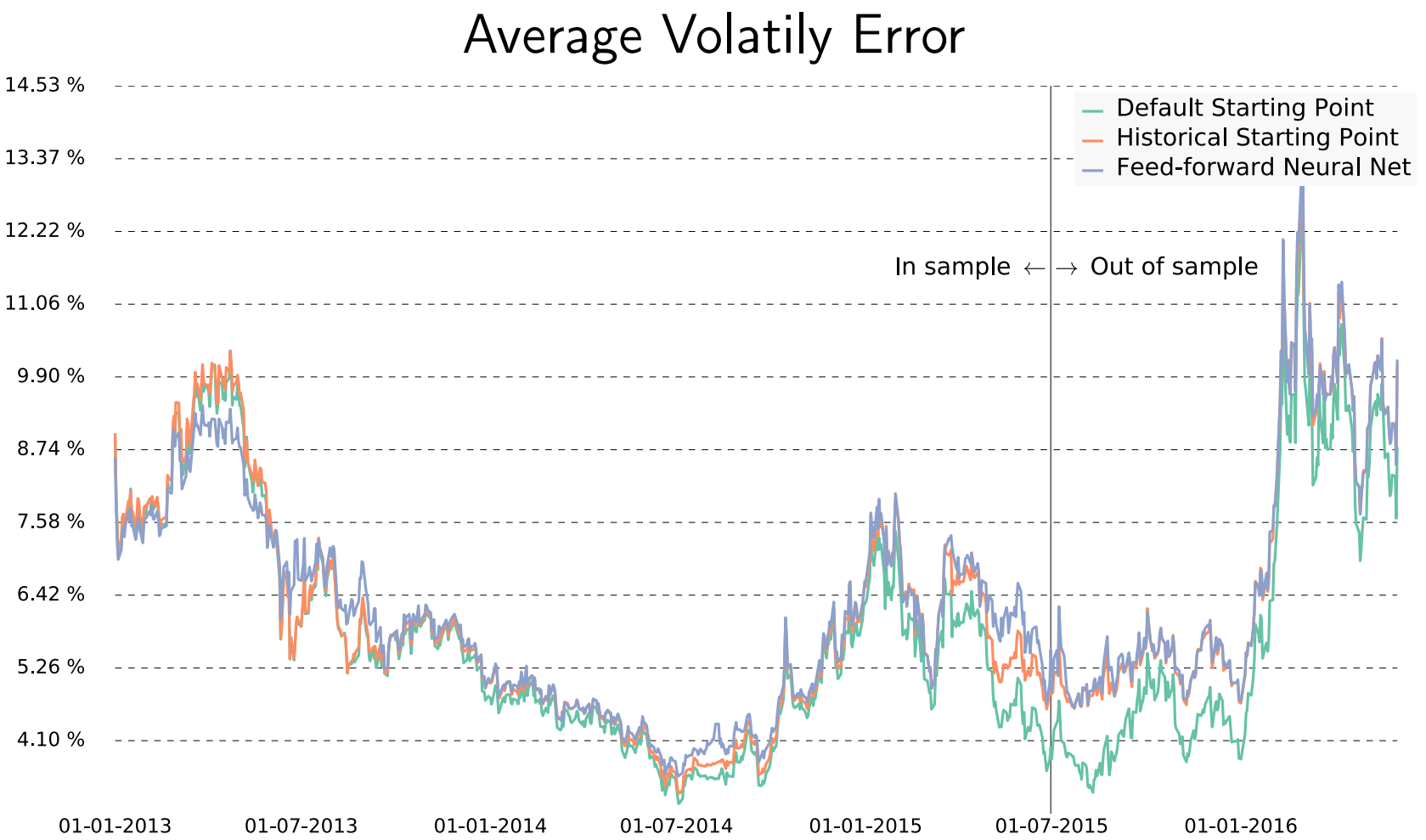
Training


Mean squared error on cross-validation set
50k_d20
100k_d20
150k_d20
0.086
0.084
0.082
0.080
0.077
0.075
0.073
0.071
0.068
0
10
20
30
40
50
Epoch

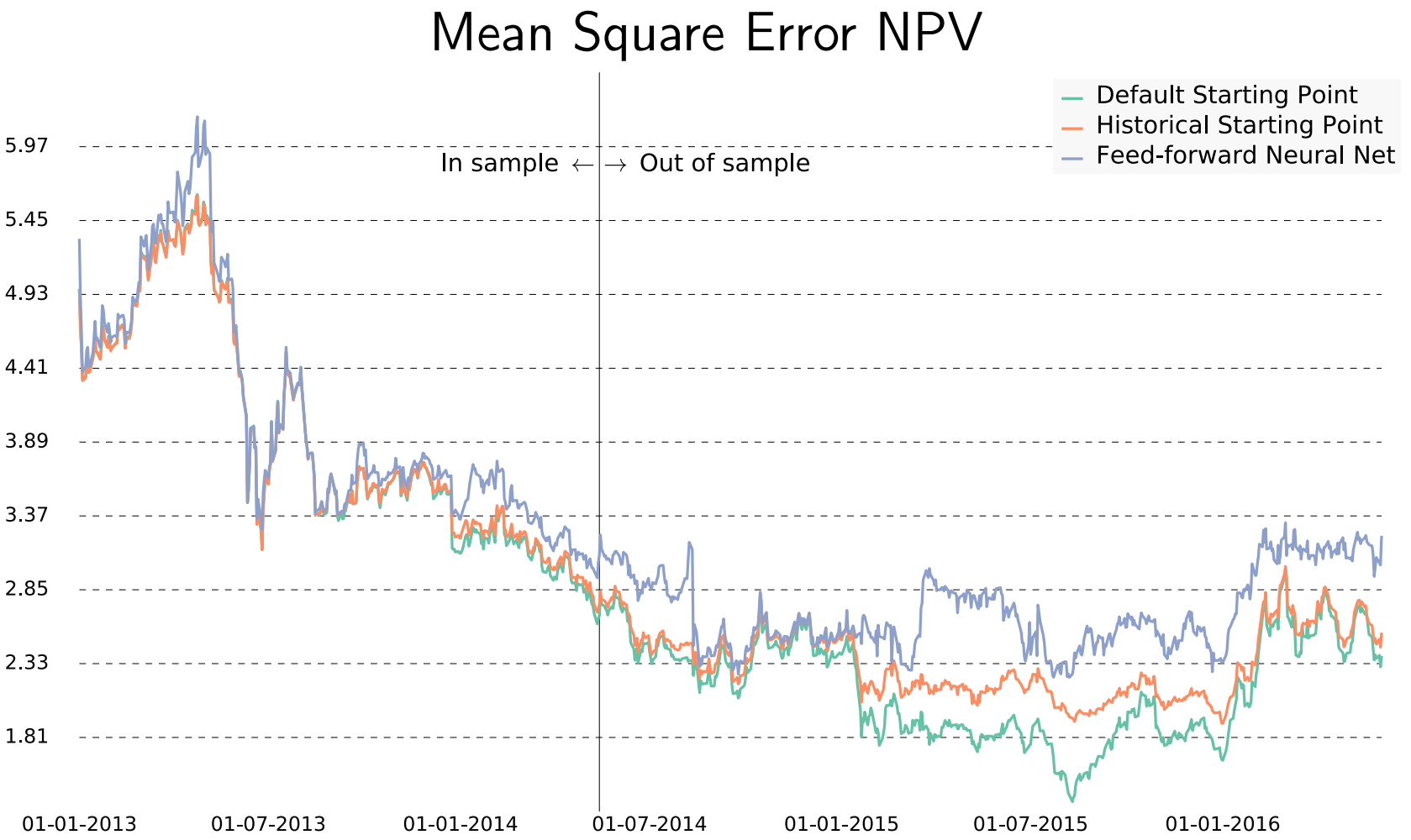
Correlation from 01-2013 to 06-2014



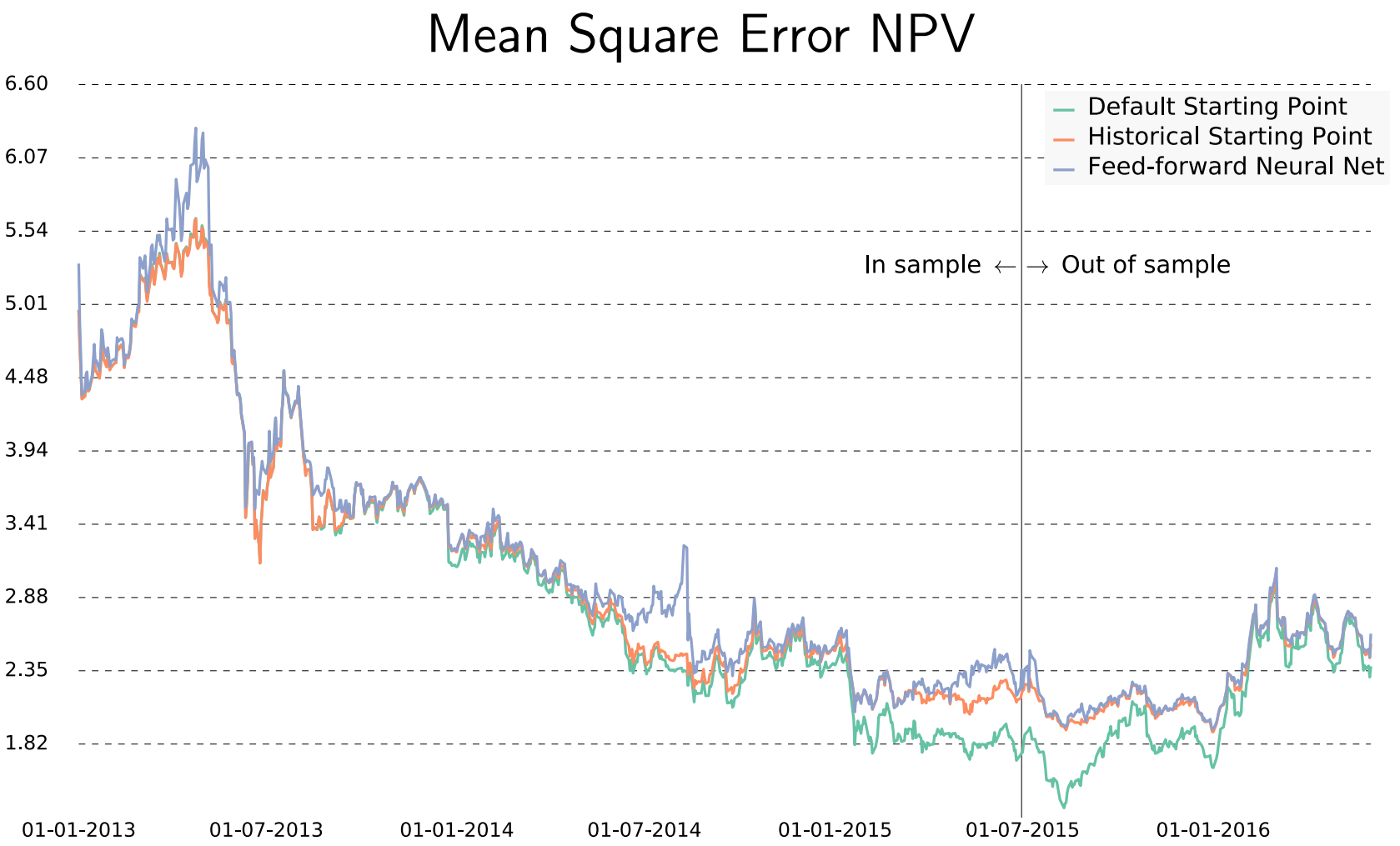
Correlation from 01-2013 to 06-2015



Correlation from 01-2013 to 06-2014

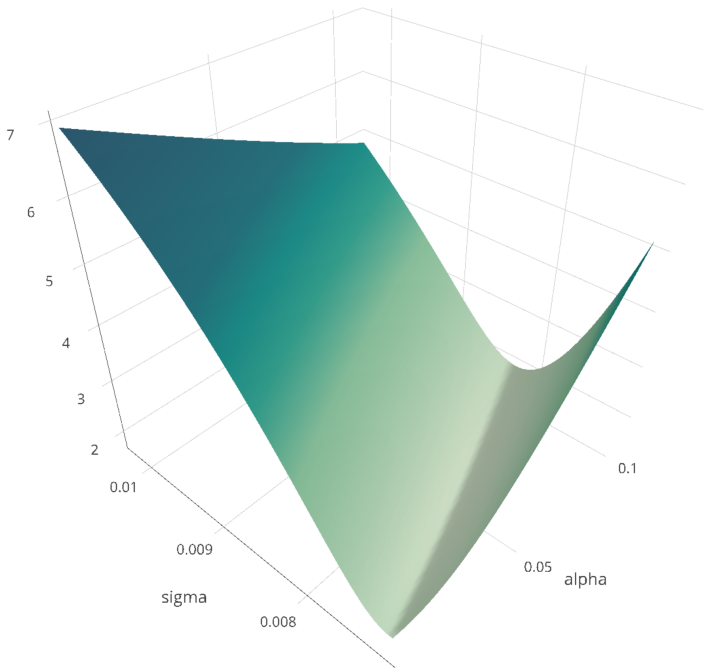


Correlation from 01-2013 to 06-2015

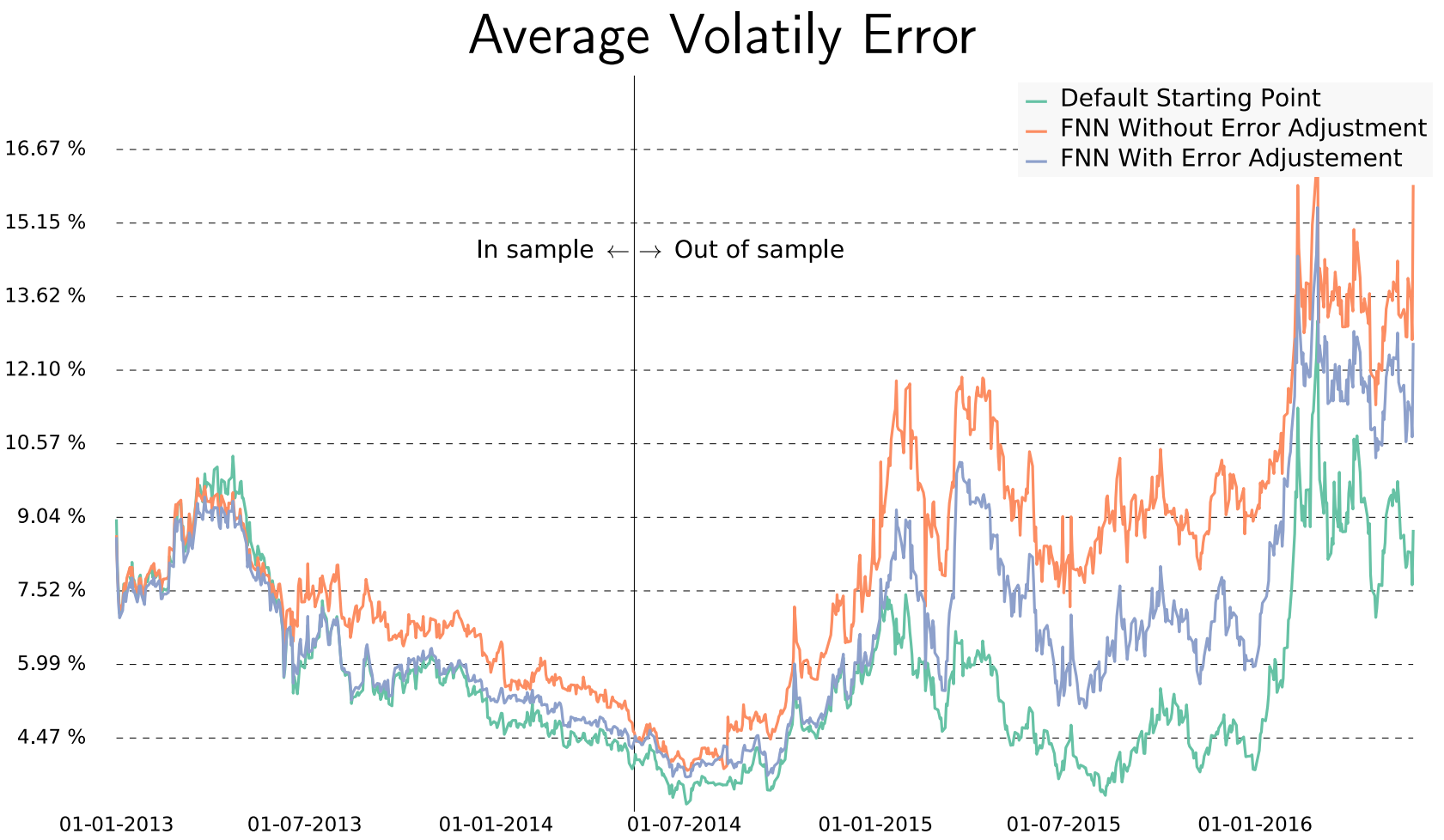


Cost Function on 01-07-2015

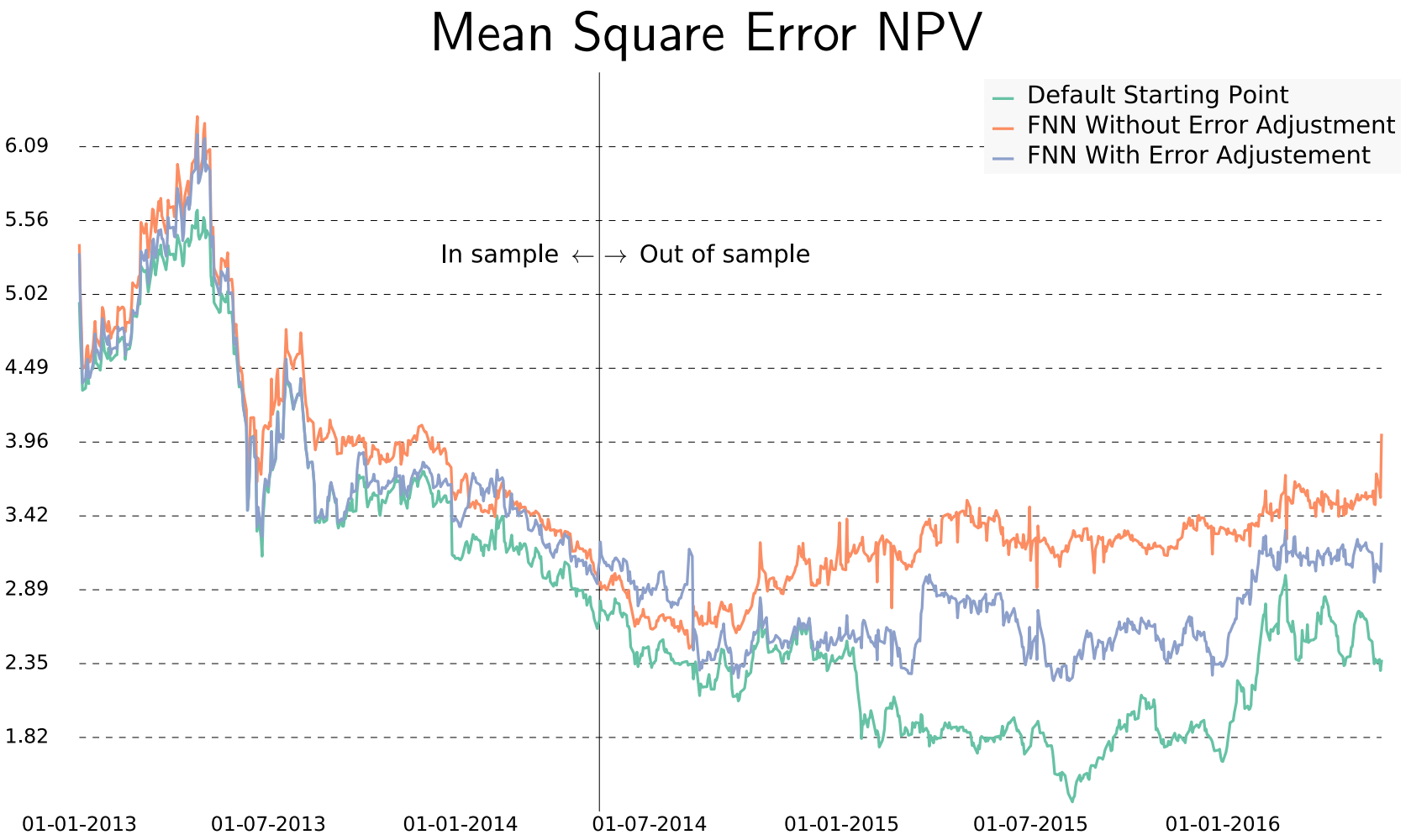
The historical point, lies on the trough. The default starting point ($\alpha = 0.1$, $\sigma = 0.01$) starts up on the side.



FNN


Average Volatily Error
Default Starting Point
FNN Without Error Adjustment
FNN With Error Adjustement
16.67 %
15.15 %
13.62 %
12.10 %
10.57 %
9.04 %
7.52 %
5.99 %
4.47 %
In sample ← → Out of sample
01-01-2013
01-07-2013
01-01-2014
01-07-2014
01-01-2015
01-07-2015
01-01-2016

FNN


Mean Square Error NPV
Default Starting Point
FNN Without Error Adjustment
FNN With Error Adjustement
In sample ← → Out of sample
6.09
5.56
5.02
4.49
3.96
3.42
2.89
2.35
1.82
01-01-2013
01-07-2013
01-01-2014
01-07-2014
01-01-2015
01-07-2015
01-01-2016

Further work

- Sample with parameterized correlation
- Test with equity volatility models. In particular with large volatility surface and use convolutional neural networks
- Test with more complex IR models

Accessing it

The example detailed here can be found in my GitHub account:
`https://github.com/Andres-Hernandez/CalibrationNN`
To run the code, the following python packages are needed:

- QuantLib
- QuantLib Python SWIG bindings
- Typical python numerical packages: numpy, scipy, pandas, sklearn (, matplotlib)
- Keras: a deep-learning python library. Requires Theano or TensorFlow as backend