# Nowcasting Networks

Marc Chataigner[1], Stéphane Crépey[1], and Jiang Pu[2]

December 29, 2019

**Abstract**

We devise a neural network based compression/completion methodology for financial nowcasting. The latter is meant in a broad sense encompassing completion of gridded values, interpolation, or outlier detection, in the context of financial time series of curves, surfaces, or more general tensors. In particular, we introduce an original architecture amenable to the treatment of data defined at variable grid nodes (e.g. bonds quotations, whose time-to-maturity decreases with calendar time). This is illustrated by two case studies on real data sets. First, we validate our approach on at-the-money swaption surfaces, defined at constant expiry/tenor grid nodes. The neural network approaches are then shown to perform as well as, but no better than, PCA, which is then available as an alternative (and is also much quicker). Second, we illustrate the flexibility of our approach on repo curves data, on which PCA or even classical autoencoder methods are no longer applicable.

**Keywords:** data compression, data completion, outliers, neural networks, autoencoders, swaption implied volatilities, repo rates.

**Mathematics Subject Classification:** 62M45, 62P05, 62M40.

**JEL Classification:** C450, G170, G120.

[1] *LaMME, Univ Evry, CNRS, Université Paris-Saclay*

[2] *Europlace Institute of Finance, Paris*

*Acknowledgement:* This research has been conducted with the support of the Research Initiative Modélisation des marchés actions, obligations et dérivés financed by HSBC France under the aegis of the Europlace Institute of Finance. The views and opinions expressed in this presentation are those of the author alone and do not necessarily reflect the views or policies of HSBC Investment Bank, its subsidiaries or affiliates. The research of Marc Chataigner is co-supported by a public grant as part of investissement d'avenir project, reference ANR-11-LABX-0056-LLH LabEx LMH. The authors would like to thank Daniel Girard, Nicolas Grandchamp des Raux, Hugo Lebrun, and Guillaume Macey, for their advice and encouragement during the conduct of this study.

# 1 Introduction

In this paper, we devise a neural network based methodology for financial nowcasting. The latter is meant in a broad sense encompassing completion of gridded values, interpolation, or outlier detection, in the context of financial time series of curves, surfaces, or more general tensors. Toward this end we develop a generic two-step methodology, whereby a pre-processing compression stage is followed by a completion stage. Moreover, we detail two variations along this baseline, corresponding to two slightly different perspectives and significantly distinct neural network architectures.

Under the so called convolutional approach, which is of the autoencoder type, we assume that the information contained in an observed tensor can be encoded into a reduced set of variables, dubbed factors. Conversely, given the factors, we can reconstruct the whole tensor with a decoder. As a limiting case, we obtain a linear, principal component analysis (PCA) kind of approach, but one itself implemented in the optimization training mode, as an autoencoder with linear activation functions (as opposed to spectral decomposition in the classical PCA case).

Under the so called functional approach, factors are rather used as a way to adjust a map taking as input a location (coordinates that may be part or not of the original tensor nodes) and returning the corresponding reconstructed value.

The convolutional approach is more particularly dedicated to completion of values on a fixed grid of coordinates, whereas the functional approach has the flavor of an interpolation tool—but one with the "memory" of a data set. Moreover, in the functional approach, including additional variables is straightfoward.

The use of autoencoders as a nonlinear extension of the PCA can be traced back to the 1980s (see Chapter 14 in Bengio, Goodfellow, and Courville (2017) for a survey of autoencoder-based learning). Autoencoders have also already been used in data completion (see Kiran, Thomas, and Parakkal (2018), Strub, Gaudel, and Mary (2016)). In contrast, the neural network architecture of our functional approach is new to the best of our knowledge.

At the intersection between neural networks and finance, the related paper by Kondratyev (2018) is more about forecasting. Accordingly, we work in a mostly unsupervised setting, whereas Kondratyev (2018) is in a mostly supervised setting. With respect to the neural network architecture reflected by the equations (4)–(5) in Kondratyev (2018), our convolutional approach has the advantage (irrespective of the different forecasting versus nowcasting focuses) of a latent structure, which does not need to be retrained simply because, for instance, the index $j$ (curve pillar of the shock of interest) changes in these equations. This is even more obvious in the case of the functional approach, where extra variables can be provided as direct inputs to the model.

Autoencoders (hence, unsupervised learning) are also considered Section 5.4 in Kondratyev (2018). However, this is then with a focus on curve regularization on a fixed grid, which can be done directly by decoding. The completion problem that we are dealing in this work is more general and it requires one additional layer of numerical optimization. Moreover, Kondratyev (2018) only deals with the univariate case of curves, for which spatial regularity is a much less challenging issue.

The paper is outlined as follows. Sections 2 and 3 introduce the problems and models. By the latter, we mean different algorithmic strategies and neural network architectures that can be used for addressing the former. Section 4 lays an experimental setup putting the different models on comparable grounds. Sections 5 and 6 present at-the-money swaption surfaces and repo curves case studies on HSBC real data sets. Section 7 concludes and discussses further research perspectives in connection with the quantitative finance and machine learning literatures.

Any notation of the form $\min_x \Lambda(x, y)$ means that we minimize in $x$ a loss $\Lambda$ given the value $y$ of additional parameters; $x^\star$ then refers to a numerical minimizer of $\Lambda(x, y)$ (which is typically nonconvex in $x$), for this given $y$.

# 2 Problems

We consider a data set consisting of a time series of observations, each consisting of $m$ points, or features, structured as a multivariate tensor. By the latter, we mean a discretized hypercube of values of homogenous quantities, such as rates of different terms, implied volatilities of different strikes and maturities, etc., defined at each tensor grid node.

## 2.1 Compression

The compression problem is mainly a pre-processing stage that aims at reducing the dimensionality $m$ of a feature space. Assume that each observation takes its values in (a subset of) $\mathbb{R}^m$. We call encoder $E$ any injective map from a relevant subset $\mathcal{S}$ of $\mathbb{R}^m$ to a space $\mathbb{R}^f$ of factors, with $f \ll m$. Conversely, one would like to be able to reconstruct the $m$ values of a tensor from any set of factors, or code, thanks to a map, called decoder, $D : \mathbb{R}^f \to \mathcal{S}$. The compression challenge is to build $D$ and $E$ such that $D \circ E : \mathcal{S} \to \mathcal{S}$ is bijective and "as close as possible to identity" (cf. Bengio, Goodfellow, and Courville (2017, Chapter 14)).

The inspection of common financial time series of tensors suggests that, in their case, this challenge is somehow not unreasonable. Indeed, structural constraints often exist between the values at different tensor nodes, e.g. arbitrage pricing relationships throughout the option chain. Moreover, usual financial tensors exhibit some spatial regularity, in the sense that values at grid nodes vary smoothly with respect to node location (think of interest rates with respect to their term or implied volatilities with respect to the maturity and strike of an option). In addition, some coordinates may have a regularizing effect. For instance, in the region of large expiries, the at-the-money swaption implied volatility surface is mostly affected by translation moves (and not so much by steepening, etc.) as time passes (see Section 5). Last, some (monotonicity, convexity,...) patterns are often apparent (cf. Figure 5.1).

Both maps $E$ and $D$ are sought for within classes of neural networks with respective parameters $\delta$ and $\varepsilon$, collectively denoted by $\theta$. We include into the latter weights, biases, as well as any variable calibrated during the compression stage. Renoting $E = E_\varepsilon$ and $D = D_\delta$ in reference to this parameterization, the compression stage is the training of the neural networks according to the following

optimization problem:

$$\min_{\theta=(\delta,\varepsilon)} \sum_{\omega\in\Omega} \sum_{(n,y)\in\omega} \Big(y - \Big(D_\delta\big(E_\varepsilon(\Omega)\big)\Big)_n\Big)^2, \tag{1}$$

where $\Omega$ stands for the training data set (cf. Section 4).

Certain additional properties are desirable for $D$ and $E$. The parameterization $\theta$ should allow for a robust and fast numerical solution to the problem (1). This may be harder to achieve for some deep neural networks too sensitive to the initialization of their parameters. In particular, two similar tensors should give rise to similar codes and vice versa, i.e. we want $D$ and $E$ to be "sufficiently smooth".

## 2.2 Completion

Having found a parametrization $\theta^\star = (\delta^\star, \varepsilon^\star)$ that ensures a satisfying reconstruction loss in (1), the completion task consists in the exploitation of $D_{\delta^\star}$ in order to find the missing values of an incomplete observation $\omega$ (of the current day, say, to be completed based on the complete observations of the previous days, used as training set).

Toward this end, we introduce the following optimization problem:

$$\min_{c} \sum_{(n,y)\in\omega} \Big(y - \big(D_\delta(c)\big)_n\Big)^2, \tag{2}$$

considered for $\delta = \delta^\star$. The completed tensor is then defined as the image $D_{\delta^\star}(c^\star)$ of the code $c^\star$ by the decoder $D_{\delta^\star}$. Obviously, the more missing values, the harder the completion task (enhanced overfitting risk, unless some appropriate regularization is used).

Note that, thanks to the compression step, the number of variables to estimate is drastically reduced in (2), to some reference number, i.e. the dimensionality of $c$ (e.g. 8 or 4 in our case studies), independent of the number of unknowns in the native, "uncompressed" completion problem (such as the number of missing implied volatility values in a to-be-completed surface). Moreover, a factorial representation with $f \ll m$ filters out the unlikely tensors (cf. Section 2.3) that could otherwise arise from a decoding due to the ill-posedness of large-scale arg-minimization problems. The regularity of the map $D_{\delta^\star}$ can sometimes be exploited to ease the completion, by initializing the numerical solution of (2) with the encoding of the last fully observed (e.g. already completed) tensor.

## 2.3 Outlier Detection

Hawkins (1980) defines outliers as "observation which deviates so much from other observations as to arouse suspicion it was generated by a different mechanism". Outlier detection is of course a crucial issue in finance. For instance, investment banks receive market information from a data provider. Sometimes, the data can be polluted with errors of various sources, or "different mechanism", whether it is data feed bugs, fat finger of other market participant, or failure from computation processes (for instance, implicitation of volatility surface from option prices). It

can be either punctual outlier, i.e. a single value of the tensor is too far away from what it should be, or the whole tensor may have a shape that is very unlikely.

To detect the punctual outliers, many simple methods are available, based on smoothness metrics or on historical percentile ranges of the values. To detect shape outliers, some criteria can be checked for very specific datasets, e.g. non-arbitrage butterfly/calendar spread conditions in the case of option prices. Here we propose a general method to detect both punctual outliers and shape aberration.

To say that the tensors generated from the "normal mechanism" is of a certain form is equivalent to say that the mechanism generates values that lie in a sub-manifold $\mathcal{S}$ of the initial feature space (cf. Section 2.1). Finding this sub-manifold is equivalent to detecting anomalies. From this point of view, anomaly detection and compression/decompression are two sides of the same coin. Indeed, from an information theory point of view, there is an equivalence between being an anomaly and being hard to reconstruct (large reconstruction error in a lossy data compression setup, low or even negative compression rate in a lossless data compression setup): See the seminal paper by Shannon (1948) or Chapter 4 in MacKay (2003). That is, a compression/decompression setup provides a natural anomaly detection tool. We will see some details and application in Section 6.

# 3 Models

## 3.1 The Convolutional (Autoencoder) Approach

Typical autoencoder architectures are composed of two successive feedforward neural networks $E$ and $D$, the encoder and the decoder. Both networks can be constituted of several layers, intermediated by nonlinear activation functions. Generally these layers have a symmetrical (and often bottleneck) structure. The layers are regrouped by pairs and, within each pair, the number of inputs neurons of one layer equals the number of output neurons of the other layer.

Convolutional networks have been introduced for image processing and, more generally, any data structure represented as a tensor. These networks aim to model the interactions between close points, which is not possible when one "flattens" the data in a vector (univariate tensor). Spatial regularity properties are handled by a convolutional structure of the neural network architectures, whereby the only (non-zero) connections are between units corresponding to adjacent (in a suitable sense) grid nodes (cf. Figure 5.2). The network then also uses fewer parameters, which reduces the complexity of the corresponding compression problem. For implementation details such as kernels and padding, we refer to Chapter 9 in Bengio, Goodfellow, and Courville (2017).

## 3.2 The Linear Projection Approach

It is well known that an autoencoder with linear activation functions and an $L_2$ reconstruction error is equivalent to a PCA (see Section 1). As a limiting case of the above, we consider a linear, PCA kind of benchmark, but one itself implemented as an autoencoder with linear activation functions (as opposed to spectral decomposition for classical PCA implementation). With respect to classical PCA (which

will also be included in our case studies), this approach involves an additional bias parameter. Moreover, it allows benefiting from the implicit regularization provided by early stopping in the related training procedure (see Section 4), as opposed to a regularization provided by truncation of the lowest eigenvalues in spectral decomposition based PCA implementation. These smallest eigenvalues play a significant role during stress periods. Hence, in compression terms, extreme events are less well handled by classical PCA than by the above approach, which we dub linear projection.

### 3.3 The Functional Approach

We introduce a variant of the above, especially suited to interpolation purposes (without reference to a fixed grid of nodes). This approach relies on a parameterized function $D = D_\delta(c, n)$ of a code $c$ and a node location $n$, where the latter no longer needs belong to a pre-determined grid. Here $\delta$ corresponds to the parameters of the decoder $D$, whereas the approach does not entail any encoder (at least, not explicitly).

The compression is written as (compare with (1), using a similar notation as well as $C = (C_\omega)_{\omega\in\Omega}$)

$$\min_{\delta,C} \sum_{\omega\in\Omega} \sum_{(n,y)\in\omega} \Big(y - D_\delta\big(C_\omega, n\big)\Big)^2. \tag{3}$$

Then, given a single, possibly partial observation $\omega$, the completion is given as (similar to (2))

$$\min_{c} \sum_{(n,y)\in\omega} \Big(y - D_\delta\big(c, n\big)\Big)^2, \tag{4}$$

considered for $\omega = \omega^\star$ and $\delta = \delta^\star$. Importantly, for each given $\delta$, the minimization (3) decouples into one (full observation) minimization (4) for each $\omega \in \Omega$. Hence, the larger compression problem (3) can be solved numerically as a succession of smaller problems (4), in conjunction with gradient iterations in the direction of $\delta$. This ensures the scalability of the approach. It also makes it amenable to online learning. The above observation also shows the consistency between (3) and (4) in the sense that, if a full observation $\omega$ is used in (4), it should yield $c^\star = C^\star_\omega$ (assuming global and unique minima to all problems for the sake of the argument).

Under this approach, dubbed functional, the decoder takes as input the location $n$ of the point, in addition to the factors $c$ (see Figure 5.3). It rebuilds each point individually, as per $n \to D_\delta(c, n)$. The network is thus able to interpolate between the nodes of the data grid. The concept of neighborhood intervenes through the argument $n$ of $D$, but the parameterization $\delta$ as well as the code $c$ are common to all locations $n$. The compression (3) can also accommodate incomplete data or discretization changes, i.e. varying grids in the training data. This feature allows training the functional network with "missing completely at random data" (MCAR, in the statistical missing data terminology).

By comparison, under the convolutional approach of Section 3.1, the concept of neighborhood intervenes through $\theta = (\delta, \varepsilon)$, since each point of the grid is only sensitive to a subset of connections (the convolutional architecture only connects

neighbouring points, cf. Figure 5.2). The encoding $c$ is obtained directly thanks to $E$, when the observation is complete, or by numerical completion (as always under the functional approach) otherwise.

# 4 Experimental Methodology and Setting

In this section, we devise an experimental methodology and the learning procedures, so that all models are set on comparable grounds.

All the optimization (compression or completion) problems are solved with the Adam adaptive learning rate stochastic gradient algorithms of Kingma and Ba (2015). The output of a neural network is by construction non-convex with respect to its parameters. So are therefore all our loss functions. The Adam algorithm has proven its robustness in non-convex optimization context. It provides fast training for most neural networks architectures thanks to automatic adjoint differentiation, which is an efficient way of producing the sensitivities of a loss function with respect to its arguments. However, no convergence is guaranteed theoretically.

For the compression stage, we make a $80 : 20$ split of a full data set into a training set and a test set. The split is chronological in order to avoid look-ahead bias (cf. Ruf and Wang (2019)). The training set is further split into a calibration and a validation data set. The former is used for computing the gradients driving the numerical optimization in the training problem, whereas the latter is used for determining an early stopping rule that provides implicit regularization, as detailed below.

The learning rate of the Adam optimizer is set to 0.001. Each iteration leads to the computation of the loss gradient on the whole calibration data set. Indeed, given the relatively small size of our data sets, full gradient evaluation is not an issue in practice. Moreover, mini-batch would require that each batch sample has approximately the same distribution, which is notoriously violated in the case of (non-stationary) financial time series. The gradient descent is driven by the loss computed on the calibration set, but the validation error is the loss function computed on the validation data set. The learning procedure is stopped when we do not observe any decrease of the validation error during a certain number of iterations, called patience. The parametrization returned by the compression is the one that minimizes the validation error. Early stopping in this sense limits the generalization error (cf. Engl et al. (1996)), i.e. the gap between the reconstruction errors computed on the calibration data set and a new, unobserved data set, the role of which is played by the test set. Sometimes, as detailed later, a penalization term is added to the compression loss function in order to provide a more regular and stable minimization. A maximum number of iterations is fixed to $10^4$ at compression stage and $10^3$ at completion stage, in order to cap the length of the optimizations.

All approaches are implemented in Python using the standard Anaconda and Tensorflow packages. Note that all hyperparameters are chosen manually, rather than by grid search or random search techniques.

### 4.1 Performance Metrics

We want to assess, for each approach, the robustness of the corresponding compression and completion procedures, as well as the behavior (distribution and dynamics) of the resulting factors.

For the compression, we consider the average root mean square reconstruction error $RMSE_\omega$ on the test set $\Omega'$ (root mean square error $RMSE_\omega$ between the values at the nodes of the tensor $\omega$ and their reconstructed counterparts, where errors will be considered in the sense of absolute and relative differences as detailed in Section 5.2). We provide a focus on the observation $\omega$ leading to the worst $RMSE_\omega$ over the test set, in order to identify the locations that are less well handled (e.g. short option maturities). In addition, we display the time series of the codes. A good compression should exploit each factor in the code (we should not observe factors stuck at zero).

The quality of the completion is assessed by a backtest on the test set. Each day of $\Omega'$, we solve the problem (2) or (4), initialiazing the factors with the fully informed encoding of the previous day. We then mask 90 % of the points in each tensor of the test set. For each such observation $\omega \in \Omega'$, we check the reconstruction $RMSE_\omega$ between the completed surface and the true one. Like for compression, we plot the worst completion obtained on the test set $\Omega'$.

### 4.2 Introduction to the Case Studies

We provide numerical results on two daily time series of real financial data: repurchase agreement yield rates and at-the-money swaption implied volatilities.

The advantage of working with yield rates or implied volatilities, instead of the corresponding bond or option prices, is that these are scaled quantities, exempt from first order dependence on contract characteristics such as nominal, time-to-maturity, actual level of the underlying in at-the-money option data, etc., which should otherwise be added to the set of explanatory variables in all learning procedures. On the other hand, no-arbitrage pricing relationships are less explicit in these scales, but our bet is that the data in such formats, which are of standard use on the trading floor, should still be sufficiently structured for any relevant no-arbitrage relationships to be exploitable by the learning algorithms.

We perform our data analysis in terms of both variations and levels, the implementation in levels being the only one amenable to actual (out-of-grid) interpolation in the functional approach, fault of interpolated values in the tensor of the previous day otherwise (under implementation in variations, one can only interpolate the variations themselves). Moreover, levels might be better for capturing structural properties of the data that may be related to, in particular, no arbitrage constraints. See Section 7 for further considerations on the no arbitrage issue.

## 5 At-the-Money Swaption Surfaces

A swaption is a financial contract allowing a client to enter into an interest rate swap with some strike $K$ at some future expiry date $U$, for some tenor length $T$. A large body of literature deals with the swaption implied volatility as a function

of the strike parameter. Also related to machine learning, Horvath, Muguruza, and Tomas (2019) consider the acceleration of the calibration of stochastic volatility models, by offline learning of the corresponding vanilla options pricing function.

By contrast, very few works are dealing with the swaption implied volatility as a function of the expiry and tenor parameters (see Figure 5.1). One exception is Trolle

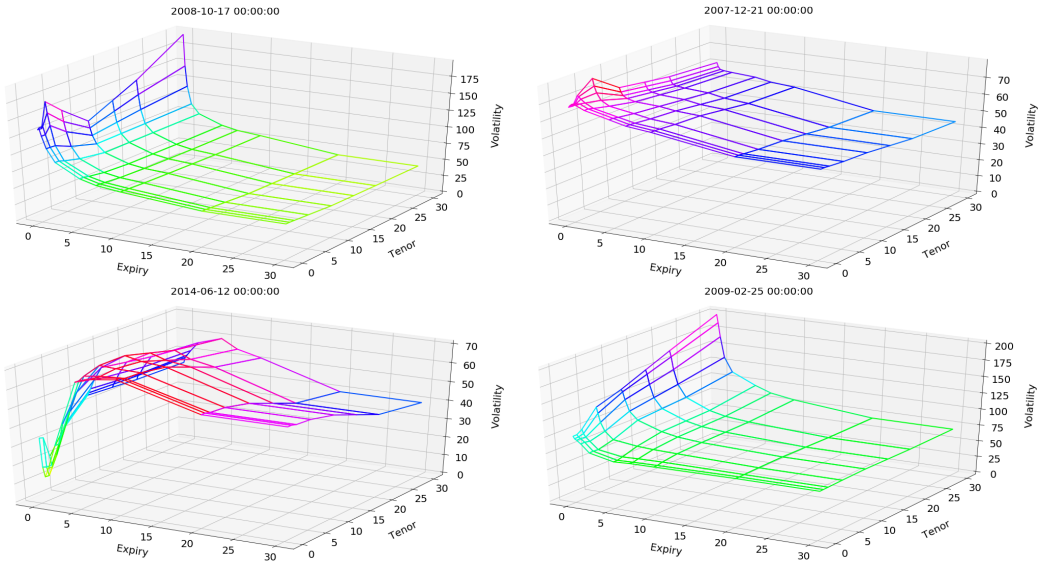


Figure 5.1: Different patterns of at-the-money swaption volatility surface.

and Schwartz (2010), who, based on a time series of swaption cubes, investigate how the conditional moments of the underlying swap rate distributions vary with expiry, tenor, and calendar time. One possible reason for this relative lack of literature may be that swaption arbitrage pricing relationships are mainly known along the strike direction. Across expiries and tenors, one only has "statistical arbitrage" relations, reflecting the overlap between the cash flow streams of the underlying swaps.

In the following case study, we focus on at-the-money (ATM, which are also the most liquid) swaption implied volatilities as a function of $U$ and $T$. The approach is model free in the sense that we do not formulate or use any hypothesis on the underlying forward swap rate processes.

Our study is conducted on an HSBC daily database of monocurrency (euro) ATM swaption implied volatilities, covering 2400 business days corresponding to the period from 2007 to 2017. The training calibration and validation set $\Omega$ covers the 2007 to 2014 sub-period (1900 first observation days of the data set), whereas the test set $\Omega'$ ranges from 2015 to 2017 (500 subsequent ones). The data have been cleaned so that all the ATM implied volatility surfaces are defined on a common rectangular grid of eighty $(U, T)$ nodes, without missing implied volatility values at any day or node, corresponding to the ten expiries (with M for month and Y for year)

$$U \in (1M, 3M, 6M, 1Y, 2Y, 5Y, 7Y, 10Y, 20Y, 30Y)$$

and the eight tenors

$$T \in (3M, 1Y, 2Y, 5Y, 10Y, 15Y, 20Y, 30Y).$$

For testing our completion approach, we mask 90% of the points in each surface of the test set $\Omega'$, only keeping the volatility points corresponding to the grid nodes $(U, T)$ in

$$\begin{aligned}&(1M, 3M), (1M, 10Y), (1M, 30Y), (6M, 2Y),\\&(6M, 15Y), (5Y, 1Y), (5Y, 20Y), (10Y, 5Y).\end{aligned} \tag{5}$$

Such specification is in line with the reality of a market where the shortest expiries are the most liquidly traded ones (as well as the most volatile). Hence, our completion exercise corresponds to the intraday situation of a swaption trader facing mostly short expiry ATM implied volatility data, and left with the task of guessing the "most likely values" of the remaining implied volatilities.

### 5.1 Fine-Tuning of the Network Architectures

We detail the neural network architectures used in the swaption case study.

Our convolutional autoencoders use feed-forward neural networks for the encoder and the decoder, with four hidden layers each: one dense layer is applied on top of three convolutional layers for the encoder and, symmetrically, three deconvolutional layers are built on top of one dense layer. The data set is reshaped as a $(10, 8)$ tensor per day. The convolution layers are built with the respective kernels (used for specifying the localization of the weights) $(5, 4)$, $(4, 3)$, and $(3, 3)$. Each

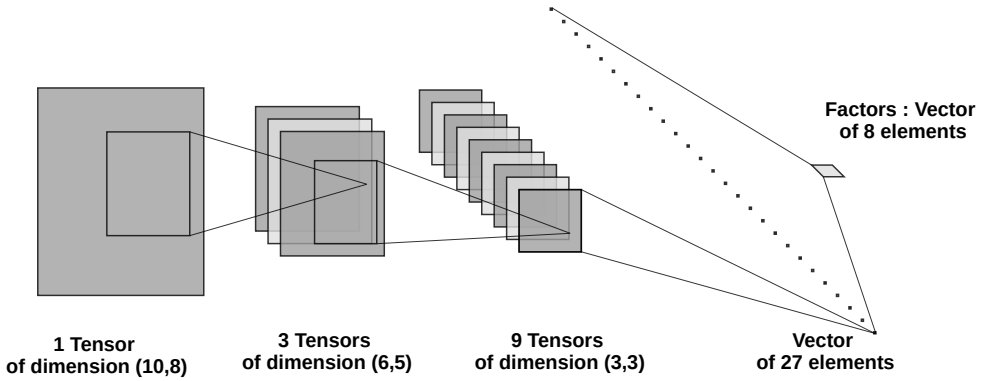


Figure 5.2: Architecture of the convolutional encoder used in our ATM swaption case study. Graph produced using the style LeNet of the NN-SVG software: Each of the four layers is represented by a triangle; The inputs of each of the three convolutional layers are displayed as collections of tensors; The ones of the last, dense layer are represented as a series of dots.

convolution layer produces 3 channels (see Figure 5.2) and, symmetrically, each deconvolution layer has in input 3 times more channels than in output. Padding is set as VALID in order to reduce the size of the hidden units after each convolution layer. As output of the three convolution layers, we have a hidden layer of 27 units, corresponding to 27 channels of size $(1, 1)$. A softplus (regularized ReLU) activation function is chosen after each convolution layer. This results in sparsity

of the calibrated network (the compression stage sets very negative biases on the intermediates units that the neural network wants to ignore, cf. Bengio (2012)), as well as positivity and regularity of the ensuing implied volatility surface. The dense layers between the factors and the (de)convolution layers are linear. Hence, the convolution layers can be seen as a kernel that linearly separates the features.

The functional approach is implemented by a single feed-forward neural network composed of three fully-connected layers with 20, 20 and 1 units (see Figure 5.3). A softplus activation is applied to each but the output layer for the same reasons

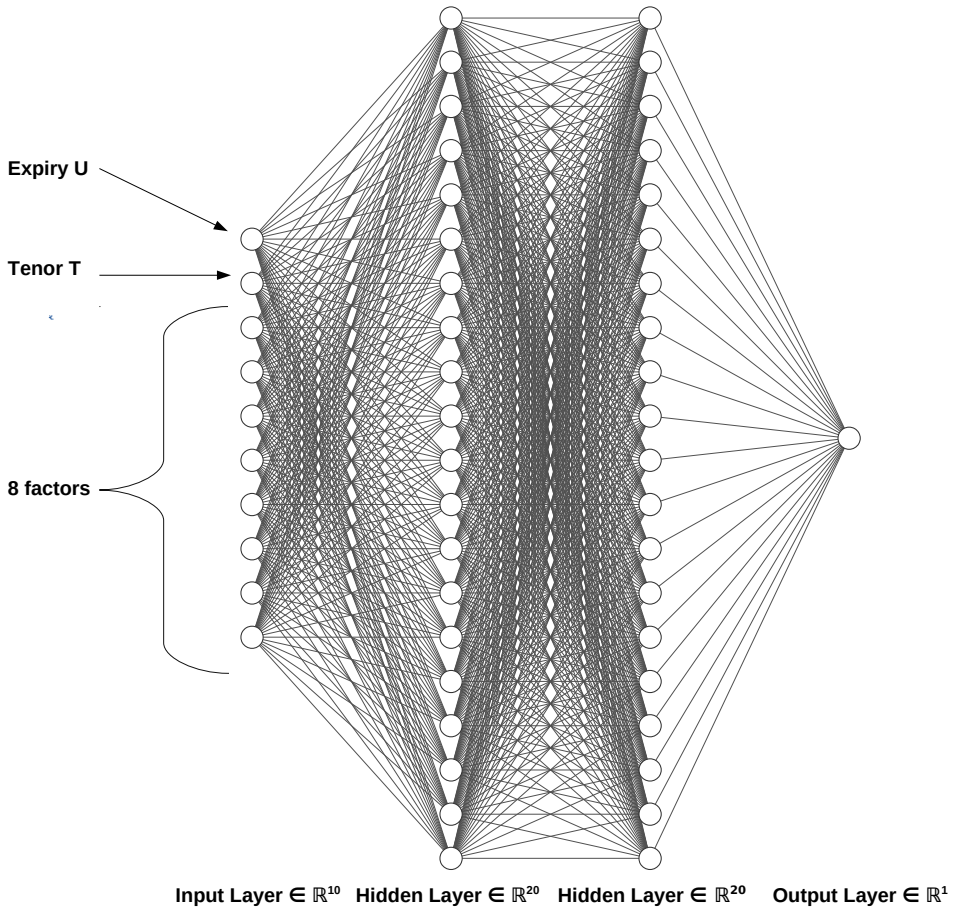


Figure 5.3: Network of the functional approach used in the ATM swaption case study. Graph produced using the style FCNN of the NN-SVG software: The units and the connections between them are represented by circles and edges.

as above (and the output layer is linear). Under this approach, we can also easily incorporate the forward swap rates (or their variations) as exogenous variables. These are the underlyings of the swaptions and they are structured similarly to the ATM implied volatilities of the latter, located by an expiry and a tenor. For taking

them into account, it suffices to add to the network of Figure 5.3 an 11th feature (input unit) containing the level (or variation) of the forward swap rate with expiry $U$ and tenor $T$. Hence, the units for the expiry $U$ and the tenor $T$ indicate the common location of the corresponding ATM volatilities and forward swap rates.

Penalization is used at the compression stage for regularizing the calibrated parameters. More precisely, ridge regularization is used for the kernel weights of the fully-connected layers of the convolutional and of the functional approaches, with a penalization coefficient of 0.1 intended to balance the reconstruction loss and the penalization term at the minimum.

In the case of the fully connected networks that are used in the linear projection and in the functional approaches, we use the Glorot and Bengio (2010) initialization rule for the weights, with a centered normal distribution of standard deviation equal to $\sqrt{\frac{4}{n_{inputs}+n_{outputs}}}$. In the case of the convolutional layers we use a truncated normal distribution with 0.1 standard deviation. All biases are initialized to zero.

Following a divide-and-conquer, sequential training strategy, we train the convolutional layers by pairs, from the most outer to the most inner ones, i.e. the layers surrounding the latent variables (greedy layer-wise pre-training as per Hinton, Osindero, and Teh (2006) and Bengio, Lamblin, Popovici, and Larochelle (2007)). A final optimization fine-tunes the weights of all the layers together. This also allows exploiting any hierarchical structure of the data (cf. Masci, Meier, Cireşan, and Schmidhuber (2011)): The outer layers detect the greatest patterns, while inner layers detect the finest ones.

## 5.2 Variations Better Than Levels for Most but Not All Purposes

Table 5.1 is a report on the errors regarding the implementation in levels of all our approaches (cf. Section 4.2). It is based on absolute and relative daily $\text{RMSE}_\omega$s, in the sense of

$$\sqrt{\frac{1}{m}\sum_{(n,y)\in\omega}\left(y-\left(D_{\delta^\star}\left(E_{\varepsilon^\star}(\omega)\right)\right)_n\right)^2} \text{ and } \sqrt{\frac{1}{m}\sum_{(n,y)\in\omega}\left(\frac{y-\left(D_{\delta^\star}\left(E_{\varepsilon^\star}(\omega)\right)\right)_n}{y}\right)^2} \tag{6}$$

(or the analogous quantities with $D_{\delta^\star}\left(C^\star_\omega, n\right)$ instead of $\left(D_{\delta^\star}\left(E_{\varepsilon^\star}(\omega)\right)\right)_n$, as relevant). Table 5.2 shows the analogous results regarding the implementations in variations, based on absolute and relative $\text{RMSE}_\omega$s in the sense of

$$\sqrt{\frac{1}{m}\sum_{(n,y)\in\omega}\left(\Delta y-\left(D_{\delta^\star}\left(E_{\varepsilon^\star}(\omega)\right)\right)_n\right)^2} \text{ and } \sqrt{\frac{1}{m}\sum_{(n,y)\in\omega}\left(\frac{\Delta y-\left(D_{\delta^\star}\left(E_{\varepsilon^\star}(\omega)\right)\right)_n}{y}\right)^2} \tag{7}$$

(or $D_{\delta^\star}\left(C^\star_\omega, n\right)$ instead of $\left(D_{\delta^\star}\left(E_{\varepsilon^\star}(\omega)\right)\right)_n$), where $\Delta y$ refers to the difference between the current $y$ in $\omega$ and the corresponding one in the observation preceding $\omega$ in the data set.

| | Standard PCA | Linear projection | Convolutional autoencoder | Functional approach | Functional approach with forward rate |
|---|---|---|---|---|---|
| Average compression error on $\Omega$ | 1.23 (1.87%) | 1.58 (2.36%) | 1.97 (3.06%) | 1.85 (2.75%) | 2.29 (3.59%) |
| Average compression error on $\Omega'$ | 3.71 (11.6%) | 3.54 (8.83%) | 6.19 (20.7%) | 3.77 (9.40%) | 3.02 (8.42%) |
| Worst compression error on $\Omega$ [day] ([day]) | 4.15 (7.87%) [2008-12-03] ([ 2014-06-10]) | 3.98 (6.92%) [2008-12-09] ([ 2014-06-10]) | 7.18 (11.5%) [2008-12-08] ([ 2014-06-10]) | 8.32 (15.1%) [2008-10-09] ([ 2014-06-06]) | 6.93 (12.2%) [2008-10-10] ([ 2014-06-13]) |
| Worst compression error on $\Omega'$ [day] ([day]) | 5.76 (23.5%) [2016-04-28] ([ 2016-04-28]) | 5.18 (22.6%) [2016-04-28] ([ 2016-04-28]) | 12.0 (55.1%) [2015-07-07] ([ 2016-04-28]) | 6.34 (16.6%) [2015-12-21 ([ 2014-10-09]) | 5.16 (16.4%) [2015-12-18] ([ 2014-07-23]) |
| Average completion error on $\Omega'$ | 6.19 (16.9%) | 4.07 (10.4%) | 5.03 (13.5%) | 6.41 (15.5%) | 5.19 (13.8%) |
| Worst completion error on $\Omega'$ [day] ([day]) | 12.6 (32.8%) [2015-06-30] ([ 2015-06-30]) | 6.50 (19.2%) [2015-07-10] ([ 2016-04-28]) | 9.89 (33.2%) [2015-07-10] ([ 2015-07-10]) | 12.8 (29.3%) [2015-03-09] ([ 2014-08-12]) | 9.09 (25.0%) [2016-01-14] ([ 2014-08-11]) |
| Training time in seconds | $\emptyset$ | 9 | 411 | 1287 | 276 |

Table 5.1: Errors regarding the implementations in levels, based on RMSEs on absolute (relative) differences in the sense of (6).

| | Standard PCA | Linear projection | Convolutional autoencoder | Functional approach | Functional approach with forward rate |
|---|---|---|---|---|---|
| Average compression error on $\Omega$ | 0.38 (0.59%) | 0.31 (0.48%) | 0.40 (0.61%) | 0.38 (0.58%) | 0.36 (0.55%) |
| Average compression error on $\Omega'$ | 0.36 (1.08%) | 0.28 (0.71%) | 0.37 (1.05%) | 0.31 (0.87%) | 0.31 (0.88%) |
| Worst compression error on $\Omega$ [day] ([day]) | 3.44 (6.22%) [2008-11-14] ([ 2014-06-05]) | 3.04 (4.57%) [2008-12-03] ([ 2014-06-09]) | 3.46 (5.83%) [2008-12-03] ([ 2008-12-03]) | 3.79 (6.41%) [2008-11-14] ([ 2014-06-05]) | 3.33 (6.23%) [2008-11-14] ([ 2014-06-05]) |
| Worst compression error on $\Omega'$ [day] ([day]) | 3.15 (29.6%) [2016-04-28] ([ 2016-04-28]) | 1.74 (8.38%) [2016-01-21] ([ 2016-04-28]) | 3.06 (31.1%) [2016-04-28] ([ 2016-04-28]) | 3.35 (30.3%) [2016-04-28] ([ 2016-04-28]) | 3.05 (26.7%) [2016-04-28] ([ 2016-04-28]) |
| Average completion error on $\Omega'$ | 0.88 (2.14%) | 0.95 (2.26% ) | 1.10 (2.46%) | 0.44 (1.29%) | 0.45 (1.33% ) |
| Worst completion error on $\Omega'$ [day] ([day]) | 5.47 (42.8%) [2015-12-07] ([ 2016-04-28]) | 5.34 (45.6%) [2015-12-07] ([ 2016-04-28]) | 5.68 (48.0%) [2015-12-07] ([ 2016-04-28]) | 4.65 (24.6%) [2016-04-28] ([ 2016-04-28]) | 4.40 (21.7%) [2016-04-28] ([ 2016-04-28]) |
| Training time in seconds | $\emptyset$ | 113 | 1623 | 707 | 974 |

Table 5.2: Errors regarding the implementations in variations, based on RMSEs on absolute (relative) differences in the sense of (7).

Note that the corresponding $D$ terms estimate the $y$ terms in (6) (which regards implementations in levels), but the $\Delta y$ terms in (7) (which regards implementations in variations). In particular, the relative error numbers are comparable between Tables 5.1 and 5.2: The smaller any relative error number, the better the corresponding method/implementation.

The last rows of Tables 5.1 and 5.2 display the corresponding training times for all but the standard PCA approach, which involves no training and is in fact much faster than all the others (as it essentially reduces to the inversion of an $m \times m$ matrix, with $m = 80$). The dates in brackets in the tables identify the observations corresponding to the worst errors.

The results of Tables 5.1 and 5.2 show that, for any approach, its implementation in variations leads to significantly smaller compression and completion errors than its implementation in levels. This is in line with the fact that the average order of magnitude of (ATM swaption implied) volatility variations is of about 1%, whereas the corresponding volatility levels fluctuate between 10% and 180%.

In the case of implementations in variations, the testing reconstruction errors are even inferior to the training reconstruction errors (compare the first two rows in Table 5.2), which may seem inconsistent with the notion of generalization error. However, looking at the reconstruction error time series (under any of the approaches, see e.g. Figure 5.4 in the case of the linear projection approach), we

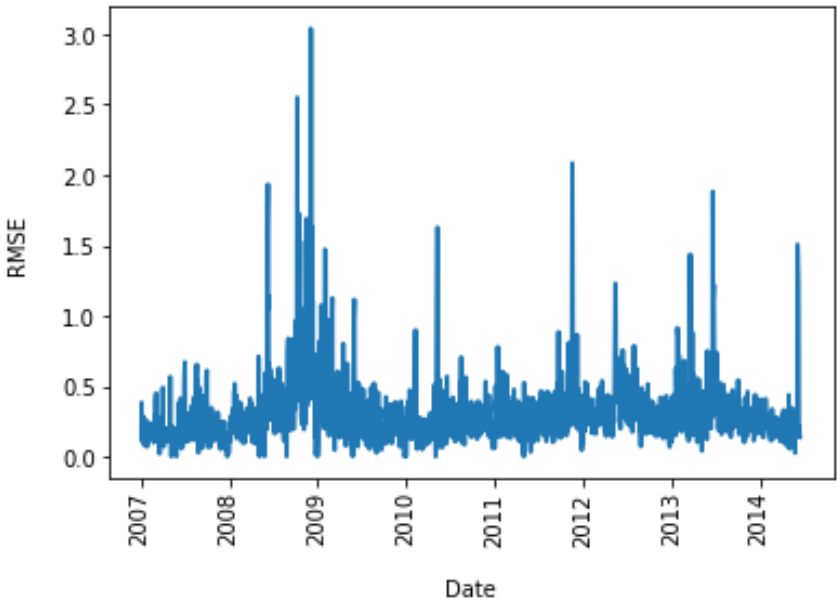


Figure 5.4: Reconstruction error for each day of the calibration set under the linear projection approach.

notice some peaks during stress periods such as the 2008 financial crisis. A possible interpretation is that, during stressed periods, eight factors, as we do here, is not sufficient to summarize all the information contained in a volatility surface. The fact that there are more violent and numerous stressed periods in the training set than in the test set can then explain the above-mentioned phenomenon. This advocates for the use of our approaches for the detection of outliers, which could be defined

as observations for which the daily error exceeds a given threshold (cf. Section 2.3).

The error gain obtained in switching from levels to variations is less significant for completion, probably because variations are less spatially correlated than levels (the surface in Figure 5.5 oscillates less than the one in Figure 5.6). Moreover,

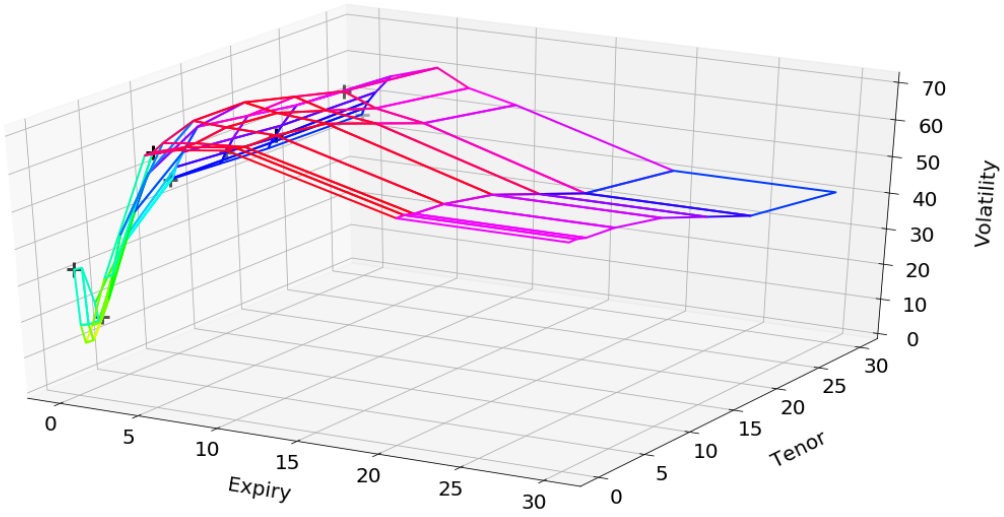


Figure 5.5: An ATM swaption volatility surface.

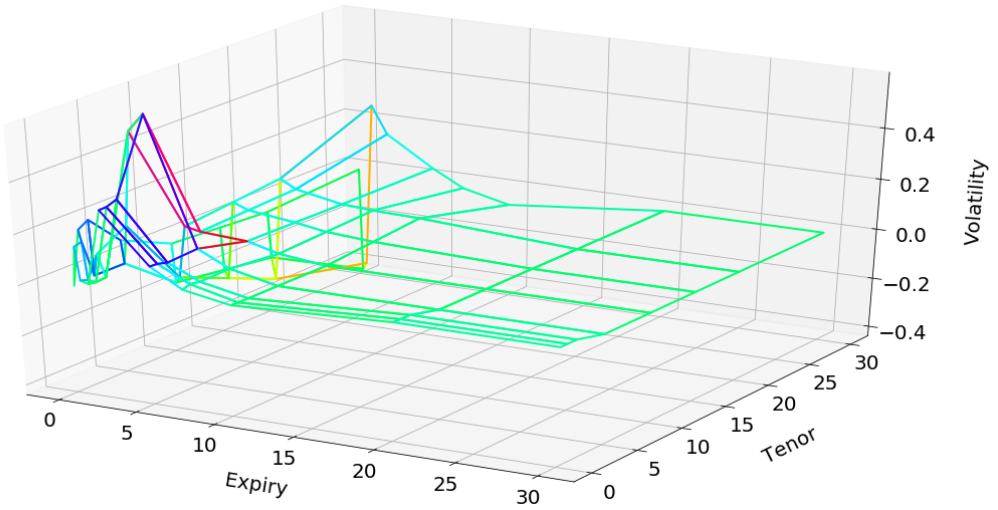


Figure 5.6: Variations between the volatility surface of Figure 5.5 and the one of the previous day.

at the completion stage, we take as initial factor values the volatility encoding of the previous day. But the factors of volatility variations are much more stable than the ones of volatility levels. In fact, the former look essentially like white noise: compare the two panels in Figures 5.7. As visible in the last rows of Tables 5.1 and 5.2, training in levels also happens to be much quicker than training in variations.

As shown by Figure 5.8 in the case of the linear projection approach (but this

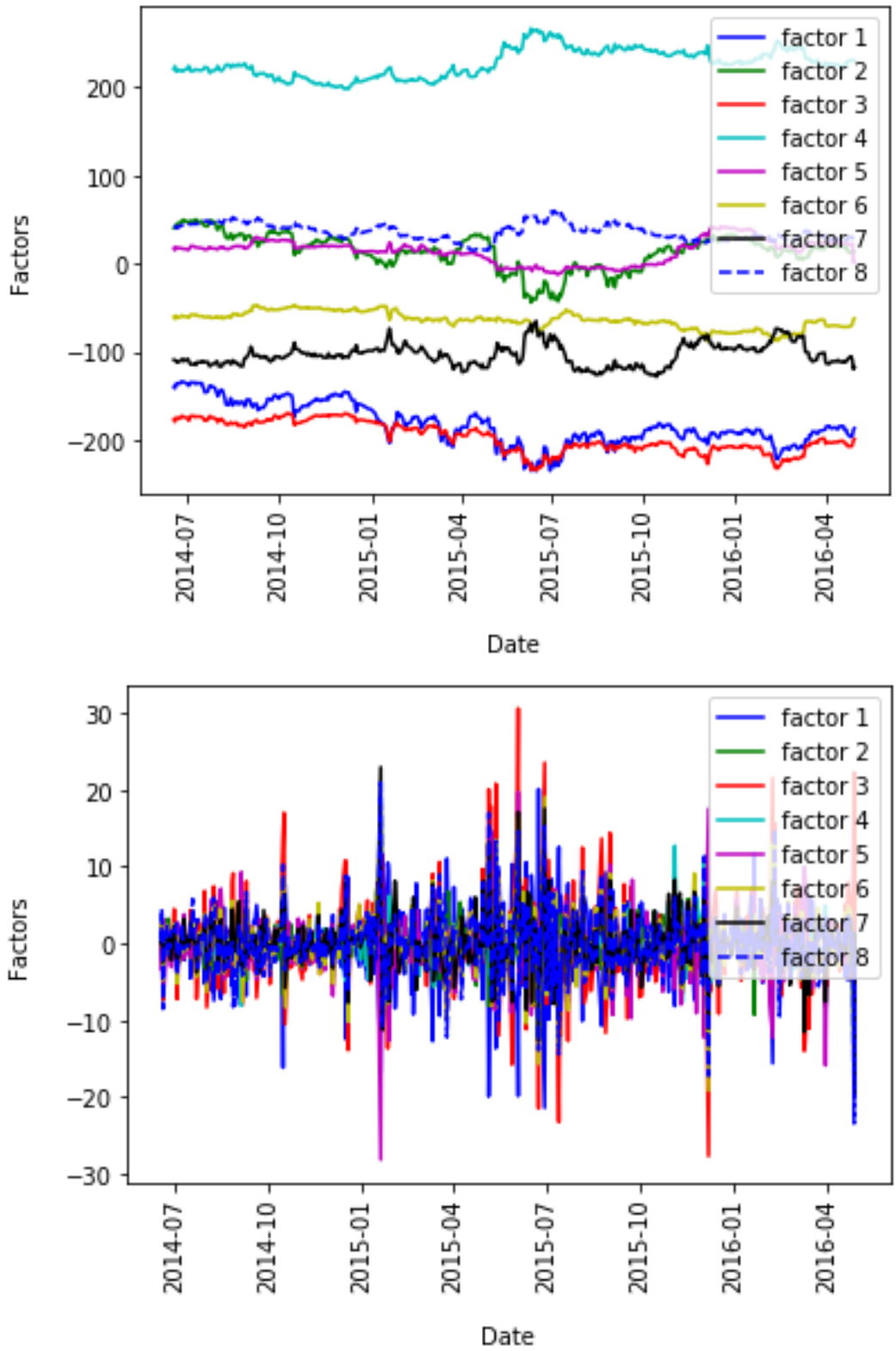


Figure 5.7: Time series of the factors obtained by encoding of the training observations under the linear projection approach. *(Top)* Implementation in levels; *(Bottom)* Implementation in variations.

is also true of the nonlinear approaches), the dominant errors are concentrated on

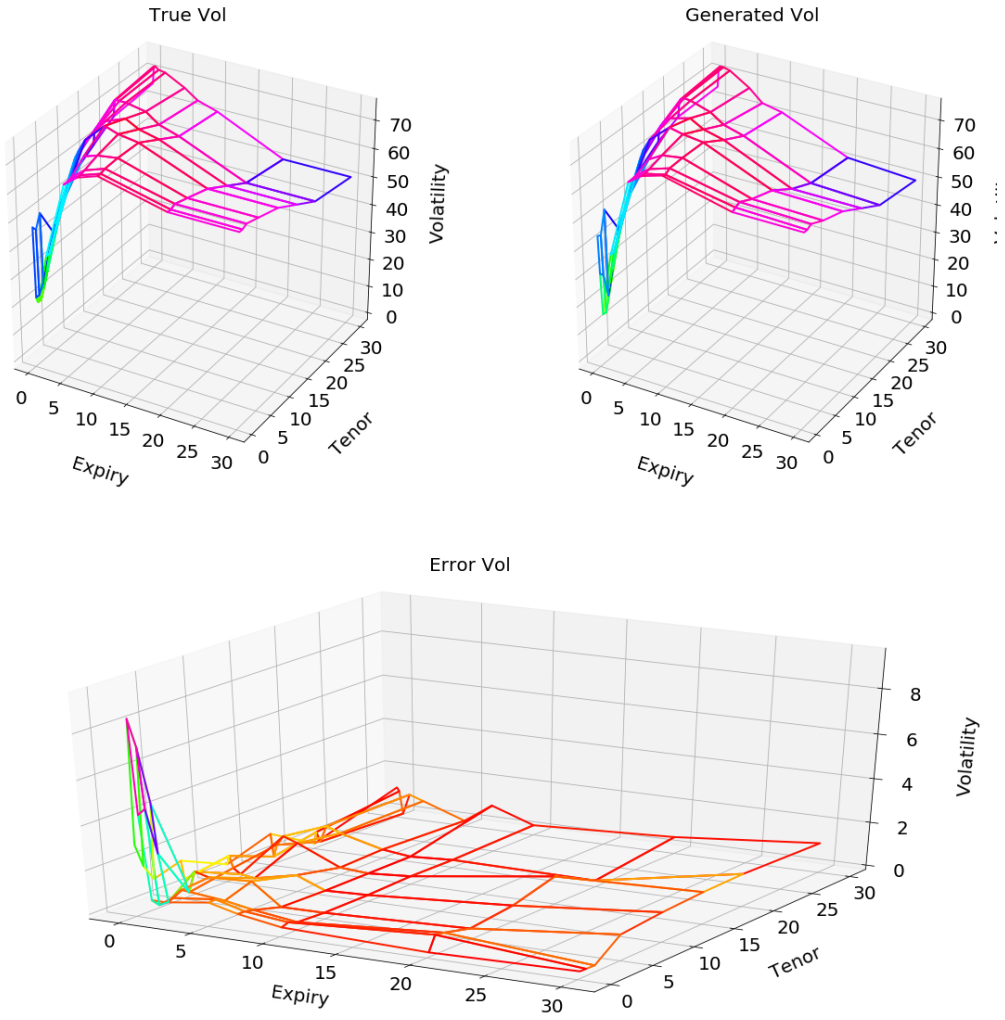


Figure 5.8: Linear projection approach: *(Top left)* Original (full) tensor; *(Top right)* Tensor $D_{\delta^\star}(c^\star)$ completed based on the 8 points of the latter given by (5); *(Bottom)* Pointwise absolute error between the two, for the worst observation in $\Omega'$.

the shortest expiries. This is because the implied volatilities corresponding to these shortest expiries are the more volatile. Hence, their spatial dependence structure is less informative. To recover these points better, one could think of providing extra information through exogenous variables, such as the level of the underlying forward swap rates. Under the functional approach, this can easily be done in the way explained in Section 5.1. However, the last columns in Tables 5.1 and 5.2 show that this only has a minor positive impact.

### 5.3 Linear Approaches Good Enough for Several but Not All Purposes

Except for completion on volatility variations, for which the functional approach performs better, the linear approaches are as accurate as the nonlinear ones. In particular, the convolutional approach is typically outperformed by at least the linear projection or the functional approach.

For implementations in variations, the linear projection approach has a slightly lower average compression error and a slightly higher average completion error than standard PCA. But the former has a much worse completion error than the latter (which is also much quicker). Our interpretation thereof is that the factors suppressed by the truncation of the smallest eigenvalues, under the classical PCA regularization scheme, would be required in distressed market conditions, when the worst completion errors arise.

As visible from the learning curves displayed in Figure 5.11, the reconstruction errors fluctuate more under the nonlinear approaches: the complexity of the nonlinear neural networks makes their training more difficult (it may also lead to overfitting of some market regimes). This could explain why the functional and the convolutional methods may even lead to higher worst case compression errors than the linear approaches (at least, under implementations in levels).

However, the functional approach is the best one in completion (at least, when implemented in variations). This is because the linear approaches do not take into account the spatial structure of the volatility points (whereas the convolutional approach does not compress well enough in the first place).

Interpolation of the volatility surface with the functional approach is in fact only possible with implementation in levels (otherwise, yesterday's interpolated value is not available). Figure 5.10 thus shows a surface of $10^4$ points obtained by the corresponding interpolation of the tensor of Figure 5.9.

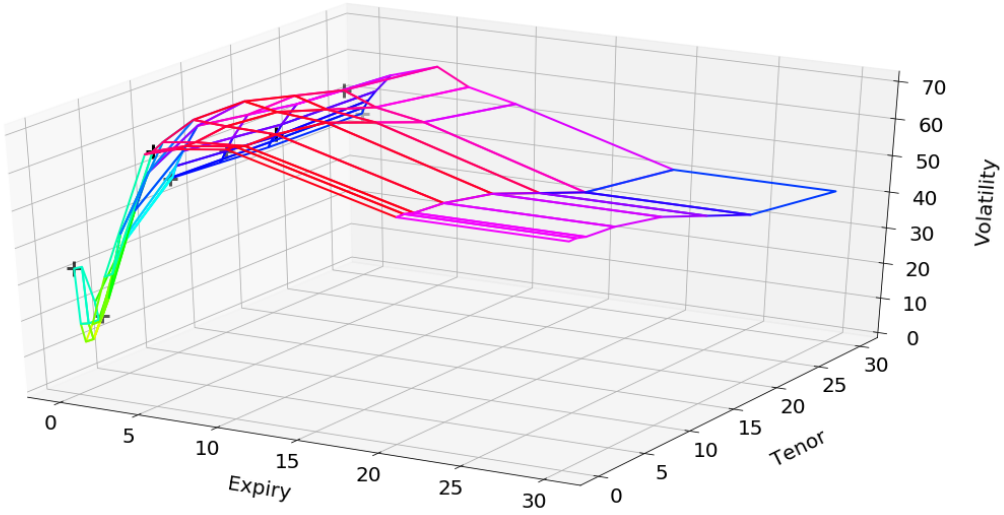


Figure 5.9: Complete tensor corresponding to the first observation in $\Omega'$. The black crosses designate the "non-missing points", specified by (5), that are used in the completion exercise.

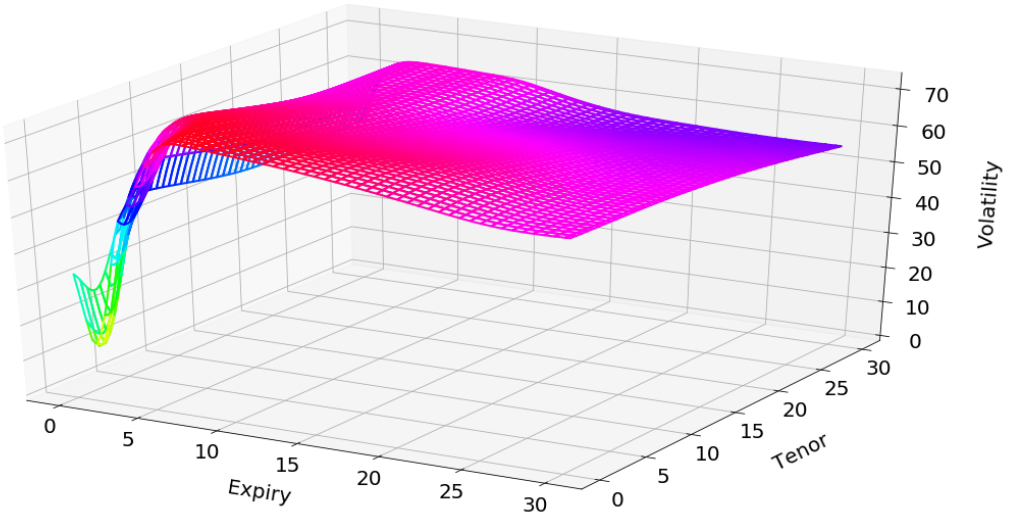


Figure 5.10: Surface with $10^4$ points obtained by the functional approach implemented in levels applied to the first observation in $\Omega'$.

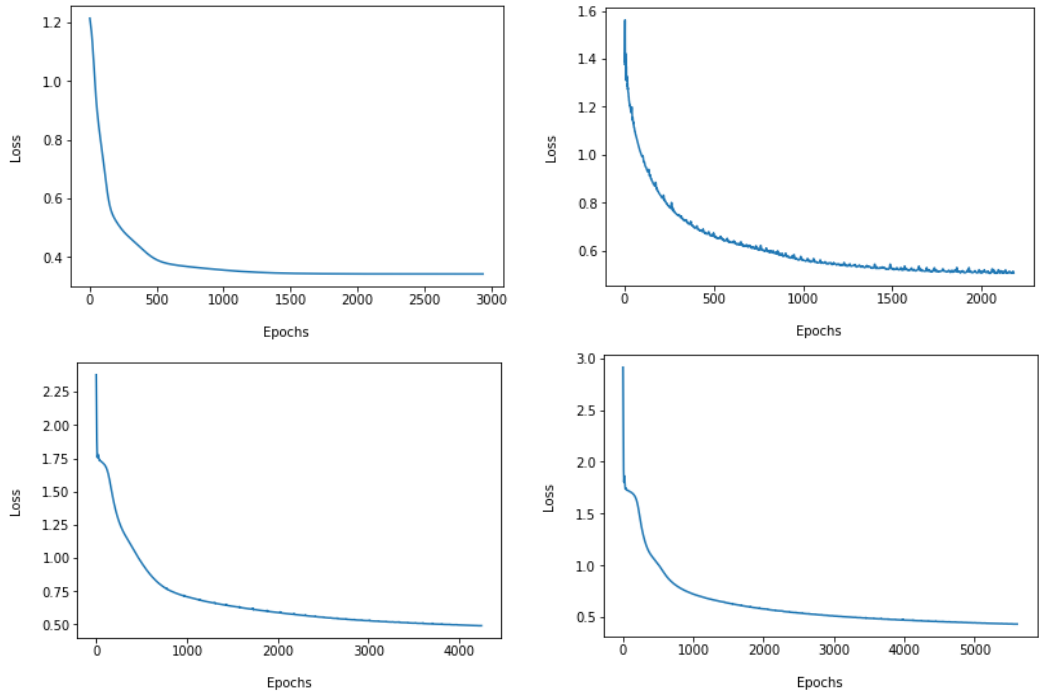


Figure 5.11: Implementations in variations: Loss on the testing data set, penalization included, after each epoch of the optimization. *(Top left)* Linear projection; *(Top right)* Convolutional autoencoder; *(Bottom left)* Functional approach; *(Bottom right)* Functional approach including forward rate variations as exogenous extra input variables.

# 6 Functional Approach Also Workable for Repo Curves

Our second case study bears on the nowcasting of repo rates, based on an HSBC 2013–2019 daily time series of repo yield curves (repo rates, where repo is a shorthand for repurchase agreement).

A major difference with the previous case is that the grid of nodes in the data is now unstructured, in the sense that the corresponding dates (time-to-maturities of bonds with standardized maturity dates) vary, in both number and location, from day to day (with as little as two or three points on particularly idle days), see e.g. Figure 6.1. Indeed, as the expiration dates used to compute the repo curve are fixed, and the variable of interest for the repo curve shape is rather time to expiry, the latter decreases as the expiry date approaches. For a given repo curve, the times to expiry for which the repo value is available is not known in advance for that reason. Therefore, there is no canonical way to have a systematic representation of repo curves on a fixed grid, one would need to introduce artificial time to expiry of interest and interpolate/extrapolate (which leads to other obstacles) the repo curve to get the values, and then working on transformed data. This is the situation the functional approach is tailored for. By not making any assumptions on the domain of input (time to expiry), the functional approach enables to handle unaltered data, by treating the time-to-maturity of a transaction as an input value (cf. Figure 5.3).

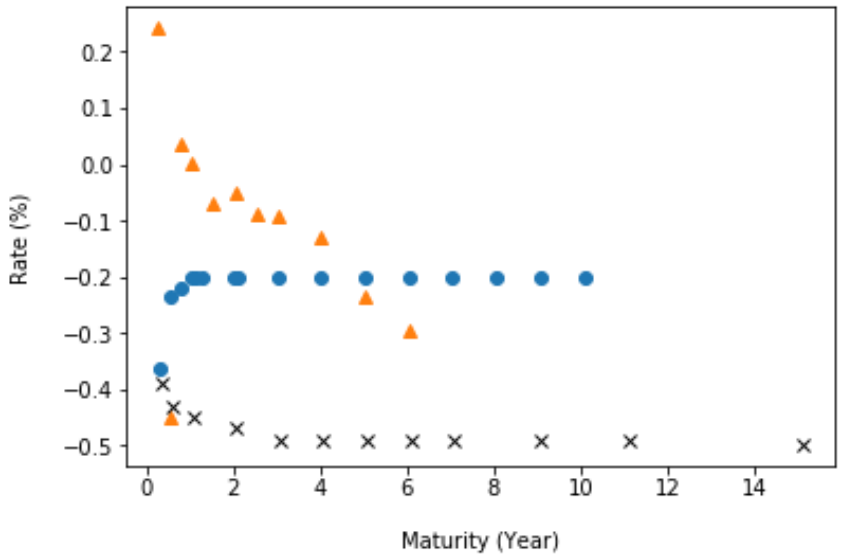


Figure 6.1: Original repo points for the three days considered further in Figures 6.2 and 6.4.

The corresponding architecture of the functional approach is then similar to the one used for swaption volatilities in Section 5.1, except that the time to maturity is used as the (one dimensional) localization input, and only 4 latent variables are used (instead of 8 previously). As Figure 6.2 illustrates, the parameterization is flexible and can accomodate different curve shapes or node localizations.

As explained in Section 2.3, the compression stage can be used for detecting an abnormal curve and correcting it with a more likely one. The distinction between inliers and outliers is determined by a threshold on the reconstruction error. A

bad reconstruction is taken as a signal that the codebook is not able to explain the corresponding observation. We then conclude that the latter does not lie in the manifold $\mathcal{S}$ of the "usual" curves, hence we classify it as an outlier (see Section 2.3). We can then correct (replace) these data by the curve reconstructed from the decoder with the factors calibrated on the current values, i.e. by the output of the corresponding completion (4).

Figure 6.3 shows the gap between the observed data points and the reconstructed ones, and spots the outliers at a 0.035 absolute RMSE threshold. Figure 6.4 gives an example of outlier correction.

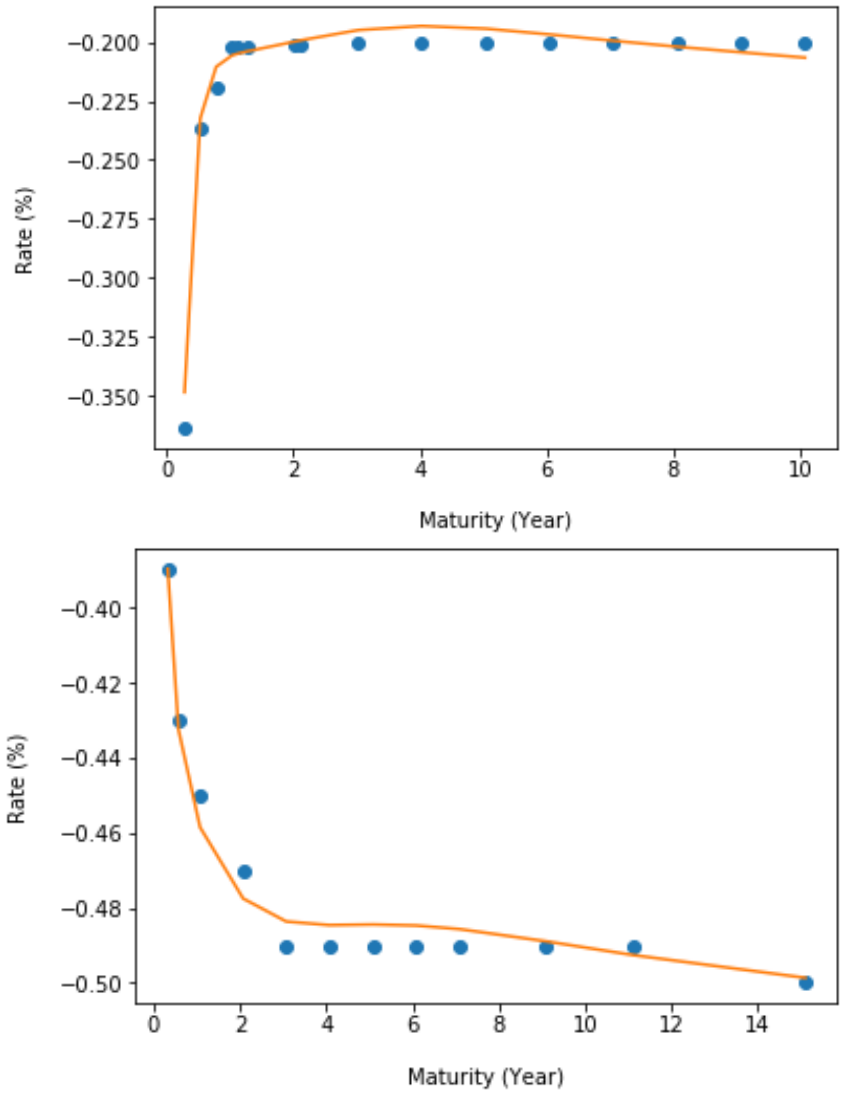


Figure 6.2: Interpolation of two inlier repo curves.

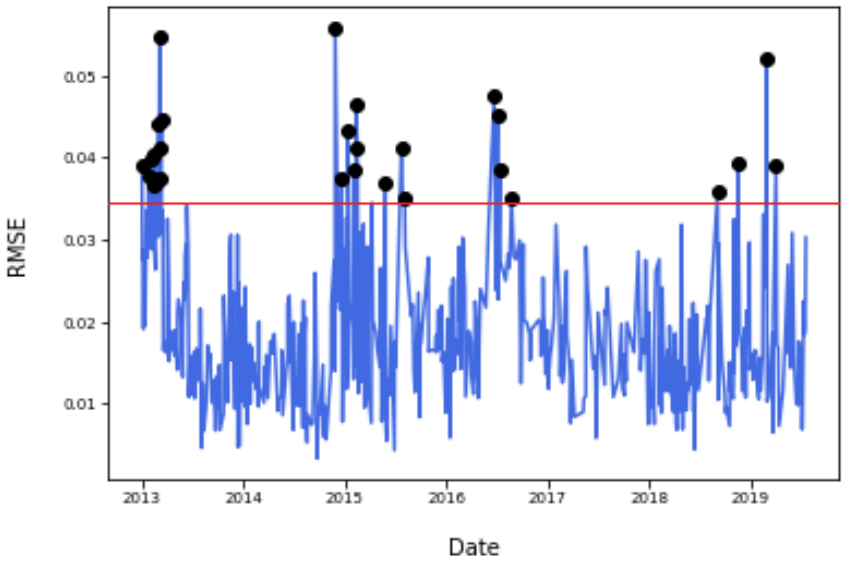


Figure 6.3: Time series of the (absolute) RMSEs on the repo data. The spotted values correspond to the outliers at the 0.035 RMSE threshold.

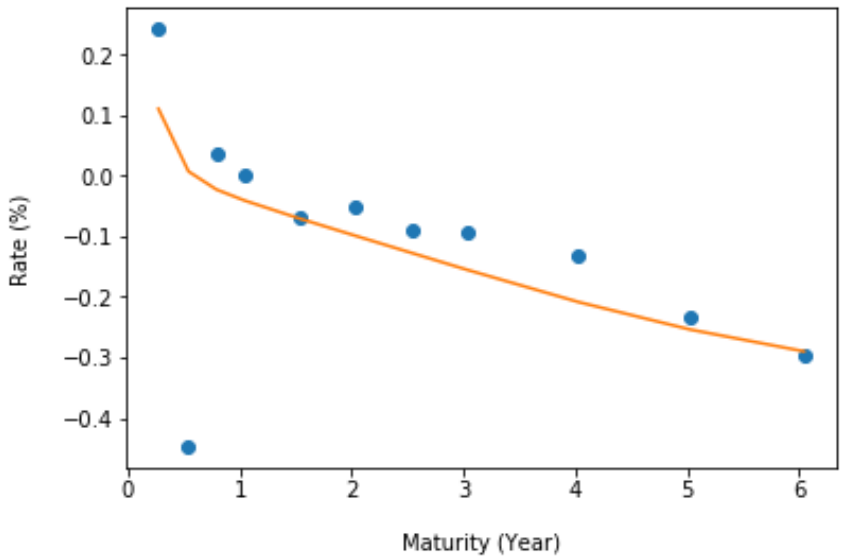


Figure 6.4: Interpolation of an outlier repo curve.

# 7 Conclusions and Perspectives

We have devised a generic neural network based tensor compression/completion methodology, for which we propose two concrete specifications: a convolutional autoencoder approach, including PCA or PCA-like projections as linear special cases, and our so called functional approach, also amenable to the treatment of unstructured data with varying grid nodes. The compression stage also allows for outlier detection and correction by generating surfaces or curves in line with training samples.

The analysis of the corresponding reconstruction errors suggests that linear methods are sufficient to compress structured tensors, corresponding to a constant grid of nodes, into few factors coefficients. The completion stage allows recovering with success about 90% values of a tensor, starting from about 10% of known values. But, again, the functional approach is the only one that is able to deal with unstructured tensors with nodes varying across calendar time, as for instance with repo curves data, for which maturities are standardized dates (hence time to maturity diminishes as calendar time increases).

All approaches suffer from non-stationarities occurring during extreme events or change of market regimes. This can be seen as an advantage with respect to anomaly detection. For other purposes, it would plead in favor of further modeling of the factor dynamics, whether this relies on times series machine learning or Markov chain Monte Carlo (filtering) techniques. More generally, it would be interesting to extend this study in several directions, such as the introduction of backtesting hedging criteria (cf. Garcia and Gençay (2000)), scenario simulation in a context of variational networks (see Tschannen, Bachem, and Lucic (2018)), or application of the method to the whole swaption volatility cube, strike dimension included (cf. Trolle and Schwartz (2010)), as opposed to expiry and tenor only in our at-the-money swaption case study.

As a final word, just like with PCA or its nonlinear extensions such as in Kondratyev (2018), there is no theoretical guarantee that such methodologies never lead to an arbitrage opportunity (static across a tensor, or dynamic upon repeated use of the method day after day in a production context). One may argue that, for the trading applications targeted in this work, one should not hinder the quality of forecasting (or nowcasting alike) by adding constraints to the optimizations meant to prevent arbitrage strategies that stay mostly theoretical anyway. Still, this question of no arbitrage may be left as a question for future research, along possible lines such as Kratsios and Hyndman (2019) or Cuchiero, Fontana, and Gnoatto (2019).

# References


Bengio, Y. (2012). Practical recommendations for gradient-based training of deep architectures. In *Neural networks: Tricks of the trade*, pp. 437–478. Springer.

Bengio, Y., I. Goodfellow, and A. Courville (2017). *Deep Learning*, Volume 1. Citeseer.

Bengio, Y., P. Lamblin, D. Popovici, and H. Larochelle (2007). Greedy layer-wise training of deep networks. In *Advances in neural information processing*

*systems*, pp. 153–160.

Cuchiero, C., C. Fontana, and A. Gnoatto (2019). Learning multiple yield curve models. Working paper (presented at the ICIAM 2019 and QMF 2019 conferences).

Engl, H., M. Hanke, and A. Neubauer (1996). *Regularization of Inverse Problems*. Kluwer.

Garcia, R. and R. Gençay (2000). Pricing and hedging derivative securities with neural networks and a homogeneity hint. *Journal of Econometrics 94*(1-2), 93–115.

Glorot, X. and Y. Bengio (2010). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 249–256.

Hawkins, D. M. (1980). *Identification of outliers*, Volume 11. Springer.

Hinton, G. E., S. Osindero, and Y.-W. Teh (2006). A fast learning algorithm for deep belief nets. *Neural computation 18*(7), 1527–1554.

Horvath, B., A. Muguruza, and M. Tomas (2019). Deep learning volatility. *Available at SSRN 3322085*.

Kingma, D. P. and J. Ba (2015). Adam: A method for stochastic optimization. In *International Conference on Learning Representations*.

Kiran, B., D. Thomas, and R. Parakkal (2018). An overview of deep learning based methods for unsupervised and semi-supervised anomaly detection in videos. *Journal of Imaging 4*(2), 36.

Kondratyev, A. (2018). Curve dynamics with artificial neural networks. *Risk Magazine*, May. Preprint version available at https://ssrn.com/abstract=3041232.

Kratsios, A. and C. Hyndman (2019). Deep learning in a generalized HJM-type framework through arbitrage-free regularization. arXiv:1710.05114v4.

MacKay, D. (2003). *Information theory, inference and learning algorithms*. Cambridge university press.

Masci, J., U. Meier, D. Cireşan, and J. Schmidhuber (2011). Stacked convolutional auto-encoders for hierarchical feature extraction. In *International Conference on Artificial Neural Networks*, pp. 52–59. Springer.

Ruf, J. and W. Wang (2019). Neural networks for option pricing and hedging: a literature review. arXiv:1911.05620.

Shannon, C. E. (1948). A mathematical theory of communication. *Bell system technical journal 27*(3), 379–423.

Strub, F., R. Gaudel, and J. Mary (2016). Hybrid recommender system based on autoencoders. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*, pp. 11–16. ACM.

Trolle, A. B. and E. S. Schwartz (2010, November). An empirical analysis of the swaption cube. Working Paper 16549, National Bureau of Economic Research.

Tschannen, M., O. Bachem, and M. Lucic (2018). Recent advances in autoencoder-based representation learning. *arXiv preprint arXiv:1812.05069*.