Nowcasting Networks[†]

Marc Chataigner[1], Stéphane Crépey[2], and Jiang Pu[3]

December 9, 2020

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 or surfaces (also applicable in higher dimensions, at least in theory). In particular, we introduce an original architecture amenable to the treatment of data defined at variable grid nodes (by far the most common situation in financial nowcasting applications, so that PCA or classical autoencoder methods are not applicable). This is illustrated by three case studies on real data sets. First, we introduce our approach on repo curves data (with moving time-to-maturity as calendar time passes). Second, we show that our approach outperforms elementary interpolation benchmarks on an equity derivative surfaces data set (with moving time-to-maturity again). We also obtain a satisfying performance for outlier detection and surface completion. Third, we benchmark our approach against PCA on at-the-money swaption surfaces redefined at constant expiry/tenor grid nodes. Our approach is then shown to perform as well as (even if not obviously better than) the PCA (which, however, is not be applicable to the native, raw data defined on a moving time-to-expiry grid).

Keywords: data compression, data completion, outliers, neural networks, autoencoders, equity derivative Black–Scholes implied volatilities, swaption implied normal volatilities, repo rates.

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

JEL Classification: C450, G170, G120.

[1] *LaMME, Université de Paris-Saclay*

[2] *LPSM, Université de Paris*

[3] *Léonard de Vinci Pôle Universitaire, Research Center, Paris La Défense*

[†] *This article has been accepted for publication in Journal of Computational Finance, published by Incisive Media Ltd.*

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. All data sets in our case studies have been put at our disposal by HSBC. 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. The authors also thank Maud Thomas for her bibliographical assistance regarding the outlier detection literature review of Section 2.3.

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 or surfaces. 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.

As such our approach is not bound to vectors and matrices. For generality and notational convenience it is presented in the methodological part on arbitrary tensors (but we do not address the strong aspect of dimension reduction that typically comes with genuine tensors as opposed to matrices and vectors).

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 can handle moving grids, which corresponds to the vast majority of applications in financial nowcasting applications (unless the data have been transformed in a preprocessing stage to make them fit a fixed grid, entailing an undesirable layer of approximation). 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. Kondratyev (2018) predicts a new curve given a shock on a curve. The neural network is trained for shocks applied to a particular location. Hence, to consider a new shock, the model needs to be retrained. In contrast, our convolutional network has a latent structure capturing interdependencies between all points in the grid. This is even more obvious in the case of our functional approach, where extra variables can be provided as direct inputs to the model.

Autoencoders (hence, unsupervised learning) are also considered in 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, 6 and 7 present repo curves, equity derivative implied volatility surfaces and at-the-money swaption implied volatility surfaces case studies on real data sets. Section 8 concludes and discusses 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 Λ 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 cube (curve or surface in our case studies, but the methodology is generic) 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, i.e. the number of grid nodes in each tensor (here assumed constant across observations ω, see Section 3.3 regarding the variant of the functional approach with a possibly variable m_ω). 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, where $f \ll m$ is the number of factors. 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 7). Last, some (monotonicity, convexity,...) patterns are often apparent (e.g. the well-known volatility smile in equity derivative, and some similar features in interest rate swaption implied volatility surfaces, cf. Figure 7.1).

Both maps E and D are sought within classes of neural networks with respective parameters ε and δ, collectively denoted by θ. The motivation for using neural networks in this context is their nonparametric (or, at least, very expressive) and nonlinear features. Gaussian processes for instance would be much less flexible, with only a few, e.g. two, kernel hyperparameters for squared exponential kernel to calibrate a full data set of thousands of tensors.

We include into θ weights, biases, as well as any variable calibrated during the compression stage. Denoting $E = E_\varepsilon$ 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 Ω stands for the training data set (cf. Section 4).

Certain additional properties are desirable for D and E. The parameterization θ 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" in such way as to preserve distance in the subspace.

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 ω (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 (higher 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. 4, 15, or 8 in our repo, equity index derivative and interest-rate swaption 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 (as outlined by the reconstruction error from our neural networks, 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.

Literature Review The literature on completion primarily deals with data structured on a fixed grid. This means that columns in the data set refer to the same feature (in our case: financial instrument). This is not consistent with most financial nowcasting applications, for which, in particular, the time-to-maturity decreases with calendar time. To the best of our knowledge, only naive interpolation methods on a given tensor, without possible exploitation of a data set, are available in the case of a moving grid.

The standard completion framework relies on a low rank representation of the data set (see Nguyen, Kim, and Shim (2019)). Along this line (but on a possibly moving grid), we compress each observation in a code which can be seen as a latent vector. However, in contrast with methods such as SVD, alternating least squares (see Hastie, Mazumder, Lee, and Zadeh (2015)), or denoising autoencoders (see Strub and Mary (2015)), which learn a user matrix, we do not consider the interaction between the observations (i.e. the dynamics): at this stage at least, we focus on the interaction between the variables (instruments).

Finally, standard completion methods in recommender systems assume missing completely at random (MCAR) values dispersed throughout the whole data set. In our case studies missing values are located completely at random but only for the current observation.

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 participants, or failure from computation processes (for instance, implicitation of volatility surface from option prices). It can be either a 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 data sets, 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. The functional variant of our method works on an unstructured grid.

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 and Mac Kay (2003). That is, a compression/decompression setup provides a natural anomaly detection tool.

Specifically, we identify an outlier as an observation whose reconstruction error (cf. (1)) is above a predefined threshold.

Some key practical questions in outlier detection are how a threshold for outlier detection should be chosen or how one can validate the method. In principle this can only be addressed by human expertise. An expert would gradually diminish the threshold until the newly detected 'outliers' are no longer considered such by the expert. The method is valid and performs well if the outlier detection in a validation set is consistent with the expert view (so that, in particular, the threshold is stable through time and does not need to be reassessed too frequently).

However, our compression methodology also provides a validation tool for the quality of our outlier detection method. Namely, one can corrupt some of the data (manually or in an automated fashion) and check whether the outlier detection procedure identifies the corrupted data.

Our approach also provides guidance to a human expert for anomaly correction. Currently experts only rely on naive heuristics, such as interpolation between different points of a surface, who cannot automatically exploit the overall data set of surfaces. In the outlier detection validation framework of the previous paragraph, one can also check whether the correction that our approach provides is closer to the true data than to the corrupted ones.

Literature Review Among many related references on outlier detection:

- Patcha and Park (2007), Chandola, Banerjee, and Kumar (2009), Omar, Ngadi, and Jebur (2013), or Anandakrishnan, Kumar, Statnikov, Faruquie, and Xu (2018) provide surveys, the last one specialized in finance and the next-to-last one on machine learning techniques;

- Lakhina, Joseph, and Verma (2010) use PCA, An and Cho (2015) variational autoencoders, Schlegl, Seeböck, Waldstein, Langs, and Schmidt-Erfurth (2019) generative adversarial networks, Lakhina, Joseph, and Verma (2010) and Cappozzo, Greselin, and Murphy (2020) semi-supervised learning. Chaloner and Brant (1988) and Cansado and Soto (2008) resort to Bayesian methodologies;

- Ro, Zou, Wang, and Yin (2015) is about high-dimensional data, Anandakrishnan, Kumar, Statnikov, Faruquie, and Xu (2018) about high dimensional big data, Rocke and Woodruff (1996) about multivariate data, Goix, Sabourin, and Clémençon (2017) and Goix, Sabourin, and Clémençon (2015) about detection of anomalies among extremes.

3 Models

The main innovation of this work is the functional approach. The description of the PCA and linear projection methods are mainly provided so that the reader can compare carefully both frameworks. The PCA and linear projection methods are also used for benchmarking purposes in the swaption case study of Section 7 for which both approaches are available. The base PCA method is of course standard. The linear projection variant of it is detailed in Section 3.2. The description is short because the method falls directly under the umbrella of sections 2.1 and 2.2 (as opposed to the functional approach of section 3.3, which requires a specific development).

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, with an overall bottleneck structure (to enforce compression in the middle).

Convolutional layers 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 (whereas dense layers bind any output unit to all input units). 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 7.3). 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 Chapter 14 in Bengio, Goodfellow, and Courville (2017)). 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.

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 δ 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 ω, 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 δ, 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 δ. 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 ω 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 5.1). 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 δ 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 7.3). 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.

3.4 Synthesis

To conclude this section, Tables 3.1 and 3.2 summarize and put into perspective the different approaches referred to in the above.

Also note that, from a numerical complexity point of view, the functional approach is less sensitive to the dimension than, say, a classical autoencoder on a fixed grid (including our convolutional approach), for which the size of the grid typically grows exponentially with the dimension.

Encoder	Implicit and non-linear $\hat{c} = \underset{c}{\operatorname{argmin}} \sum_{(n,y)\in\omega} \left(y - D_\delta(c,n)\right)^2$
Decoder	Analytic and non-linear $\hat{y} = D_\delta(c,n)$
Compression (training) step	Optimization w.r.t. (δ, c) $\min_{\delta,C} \sum_{\omega\in\Omega} \sum_{(n,y)\in\omega} \left(y - D_\delta(C_\omega,n)\right)^2$
Reconstructed surface Reconstruction	Implicit $\hat{y} = D\left(\underset{c}{\operatorname{argmin}} \sum_{(n,y)\in\omega} \left(y - D_\delta(c,n)\right)^2, n\right)$
Completed surface	$\hat{y} = D\left(\underset{c}{\operatorname{argmin}} \sum\limits_{(n,y)\in\omega} \left(y - D_\delta(c,n)\right)^2, n\right)$

Table 3.1: The functional approach.

	PCA	Convolutional
Encoder	Analytic and Linear $\hat{c} = E_\varepsilon(y)$	Analytic and non-linear $\hat{c} = E_\varepsilon(y)$
Decoder	Analytic and linear $\hat{y} = D_\delta(c)$	Analytic and non-linear $\hat{y} = D_\delta(c)$
Compression (training) step	Optimization w.r.t. (δ, ε) $\min_{\theta=(\delta,\varepsilon)} \sum_{\omega\in\Omega} \sum_{(n,y)\in\omega} \left(y - \left(D_\delta(E_\varepsilon(\omega))\right)_n\right)^2$	Optimization w.r.t. (δ, ε) $\min_{\theta=(\delta,\varepsilon)} \sum_{\omega\in\Omega} \sum_{(n,y)\in\omega} \left(y - \left(D_\delta(E_\varepsilon(\omega))\right)_n\right)^2$
Reconstructed surface Reconstruction	Explicit/analytic $\hat{y} = D\left(E(y)\right)$	Explicit/analytic $\hat{y} = D\left(E(y)\right)$
Completed surface	$\hat{y} = D\left(\underset{c}{\operatorname{argmin}} \sum\limits_{(n,y)\in\omega} \left(y - D_\delta(c)\right)^2\right)$	$\hat{y} = D\left(\underset{c}{\operatorname{argmin}} \sum\limits_{(n,y)\in\omega} \left(y - D_\delta(c)\right)^2\right)$

Table 3.2: PCA and convolutional approaches.

4 Experimental Methodology and Setting

The data and code for the equity and repo case studies can be found on a public github repository `https://github.com/mChataign/smileCompletion` (the data and code for the swaption case study are proprietary).

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. With the help of automatic adjoint differentiation, it provides fast training for most neural networks architectures. 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. Mini-batch learning is used in the repo and equity index derivative case studies, whereas batch-learning is employed with swaption volatilities. 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 tensorflow package in the swaption case study and pytorch in the two others. Note that all hyperparameters are chosen manually, rather than by grid search or random search techniques. Grid search is not possible because we have too many hyperparameters. Exploring different neural net architectures would be too demanding computationally. However, some of the hyperparameters can be fixed based on human expertise. For instance, 15 factors in our case study of Section 6 is the number of factors that equity derivative traders commonly use in PCAs (after interpolation on a fixed grid, as they are faced with moving grids).

4.1 Performance Metrics

We want to assess, for each approach, the performance 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 Ω' (root mean square error $RMSE_\omega$ between the values at the nodes of the tensor ω and their reconstructed counterparts, i.e.

$$\sqrt{\frac{1}{m}\sum_{(n,y)\in\omega}\left(y-\Big(D_{\delta^\star}\left(E_{\varepsilon^\star}(\omega)\right)\Big)_n\right)^2} \tag{5}$$

(or the analogous quantities with m_ω and $D_{\delta^\star}\left(C^\star_\omega, n\right)$ instead of m and $\Big(D_{\delta^\star}\left(E_{\varepsilon^\star}(\omega)\right)\Big)_n$, as relevant). In the case of the functional approach the encoder E is implicit and its definition is detailed in table 3.1. We refer to (5) as the reconstruction loss in the compression stage of our case studies given that ω is a complete surface.

In constrast with (5),

$$\sqrt{\frac{1}{m}\sum_{(n,y)\in\omega}\left(y-\Big(D_{\delta^\star}\left(\hat{c}\right)\Big)_n\right)^2} \tag{6}$$

(or m_ω rather than m in the case of the functional approach) is called the completion loss when we compare the complete original observation with the completed observation. This completed observation is given by the decoder for code values $\hat{c}$ which are calibrated on the incomplete view provided by ω.

In the case of interpolation benchmarks, there is no compression stage and no code is involved at the completion stage: the completion loss is then defined by the

RMSE between the interpolated surface (from an incomplete ω) and the original complete ω.

We provide a focus on the observation ω 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 Ω', 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 Ω'.

4.2 Introduction to the Case Studies

We provide numerical results on three daily time series of real financial data: repurchase agreement yield rates, equity implied volatility surfaces and at-the-money swaption implied volatilities. However, the swaption implied volatilities have been preprocessed by our data provider to fit a fixed grid (whereas the native, raw data had a moving time-to-expiry). A preprocessing entails an unquantifiable bias and our recommendation would be to apply the functional approach to the original data (whenever available). The main motivation for the third example is that one can then benchmark the functional approach against PCA and the convolutional approach.

The advantage of working with yield rates or implied volatilities, instead of the corresponding 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. The ensuing arbitrage issue is discussed in the next subsection.

4.3 Discussion of the Arbitrage Issue

Arbitrage constraints can be expressed naturally in terms of options prices using calendar spread and butterfly. But in terms of implied volatility, they are nontrivial, even in the simplest case of equity derivatives (for which they are fully stated in Roper (2010)). No compression/completion method applied to implied volatility surfaces provides a way to deal with those constraints without coming back inherently to option prices. In order to circumvent that problem, one could apply our approach to the coefficients of a (e.g. local vol) model, from which non arbitrable prices and implied volatilities could be derived in a second step. However, we do not choose this route because:

- the market practitioners, who play both the roles of human experts and users, have built intuitions over decades on implied volatilities. They think of option prices directly in terms of implied volatilities. Providing them with a good recommendation tool in terms of a quantity that is familiar to them is of great value and the primary purpose of our approach;

- most of the times, the starting point for calibrating a model (e.g. Dupire) is nothing else than the implied volatilities. Therefore the trader must correct the anomalies *before* the implied volatility surface can be plugged as an input to model calibration. Hence one of the requirements of our proposed approach is that it should be model-free;
- Having said this, if one assumes that, on the one hand, most of the surfaces in our database are arbitrage-free and, on the other hand, a more regular surface is less prone to arbitrage opportunities, then one concludes that our model should tend to remove part of the arbitrages present in the data. This can actually be seen empirically on some of the examples in Section 6. This is a natural by-product of anomaly correction and it also eases the calibration process.

Similar comments apply on most markets (beyond equity implied volatility), including the ones of our three case studies, i.e. repo contracts, handled by traders in terms of yield curves, and equity index derivatives and swaptions, which are handled in terms of implied volatility.

5 Repo Curves

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

The grid of nodes in the data is 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 5.2. 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 poses issues of its own) 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 5.1).

5.1 Functional Network Architecture

Our 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 5.1). Hyperbolic tangent activation is applied to each but the output layer for the same reasons as above (and the output layer is linear).

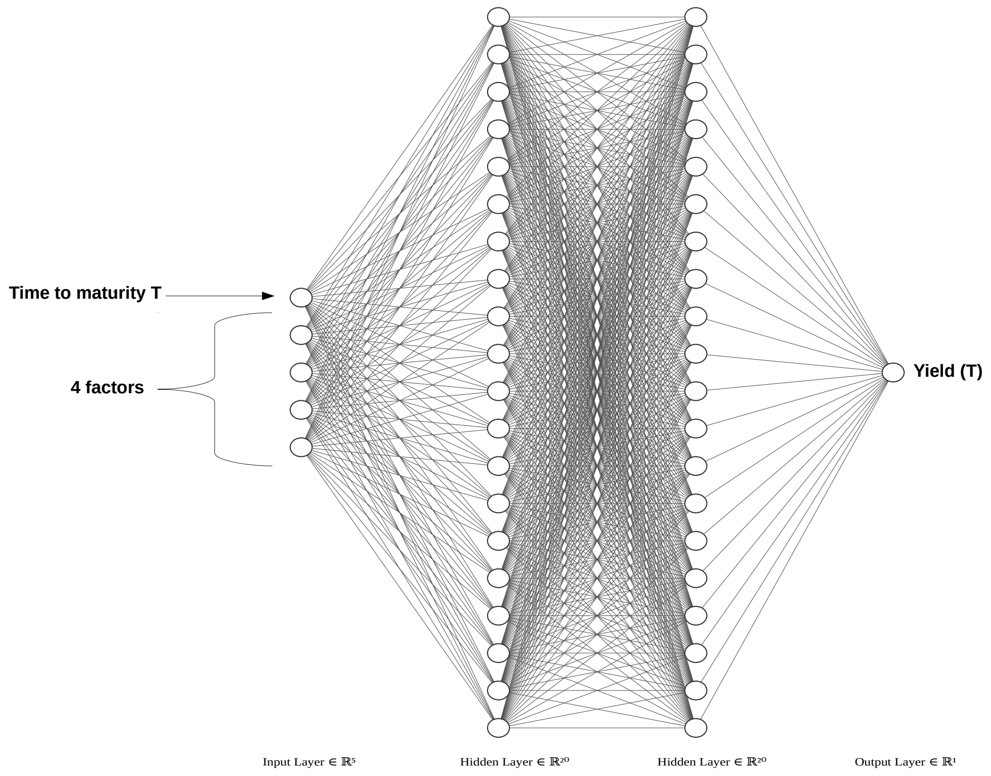


Figure 5.1: Network of the functional approach used in the repo case study. Here and in Figures 6.1 and 7.2 below, the graphs have been produced using the style FCNN of the NN-SVG software: the units and the connections between them are represented by circles and edges.

5.2 Numerical Results

As the bottom panels of Figure 5.2 illustrate, 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).

The lower panels of Figure 5.2 show the gap between the observed data points and the reconstructed ones. The upper left panel spots the outliers at a 0.035 absolute RMSE threshold. The upper right panel gives an example of outlier correction.

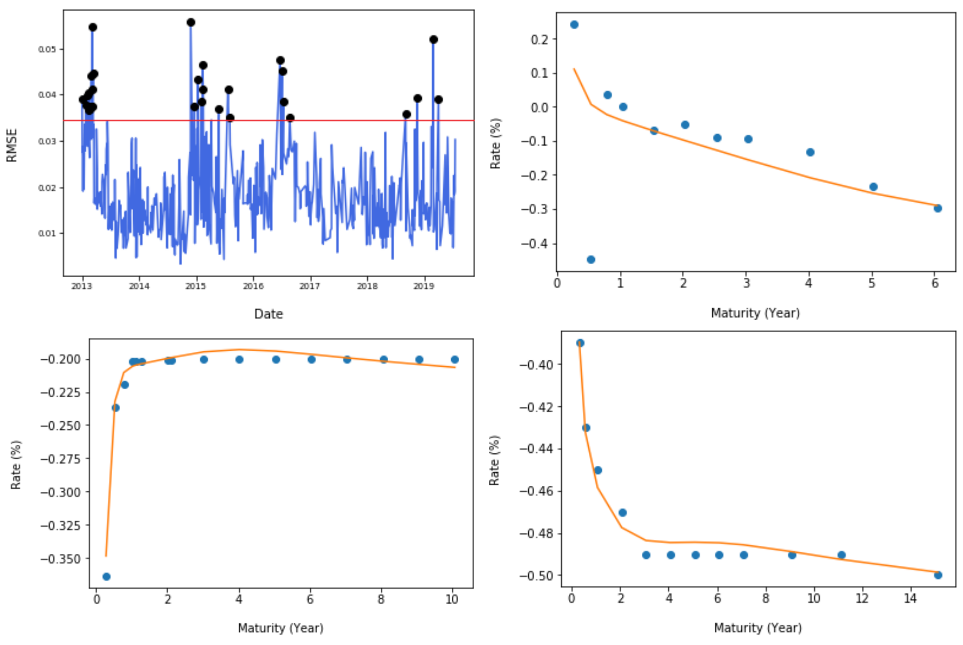


Figure 5.2: *(Bottom)* Interpolation of two inlier repo curves; *(Top left)* Time series of the (absolute) RMSEs on the repo data and 0.035 RMSE threshold; the spotted values correspond to the outliers at the chosen threshold. *(Top right)* Interpolation of an outlier repo curve.

6 Equity Derivative Implied Volatility Surfaces

As a second experiment, we apply our functional approach to Black–Scholes implied volatilities surfaces of equity index derivatives. The corresponding volatilities price options on the Nikkei 225 index from 2015 to 2018 (included), corresponding to 1544 observable surfaces. The order of magnitude of implied volatilities fluctuates between 0.15 and 1.2. We include the forward rate as an exogenous variable that can be plugged into the functional network (5.1) along with log-maturity and log-moneyness.

As in the repo case study, the grid of nodes in the data is unstructured, in the sense that the corresponding dates (time-to-maturities of equity index options with standardized maturity dates) But, again, this is the situation the functional approach is tailored for (cf. Figure 5.1). The corresponding architecture of the functional approach is then similar to the one used for repo curves in the previous section, except that the log-time-to-maturity and the log-moneyness are used as the (two dimensional) localization inputs, and that 15 latent variables are used (instead of only 4 previously): see Figure 6.1. Moreover, one can also easily incorporate the forwards as exogenous variables. For taking them into account, it suffices to add to the network of Figure 6.1 an additional feature (input unit) containing the level of the forward swap rate with maturity T. Hence, the units for the maturity T indicate the common location of the corresponding volatilities and forward rates.

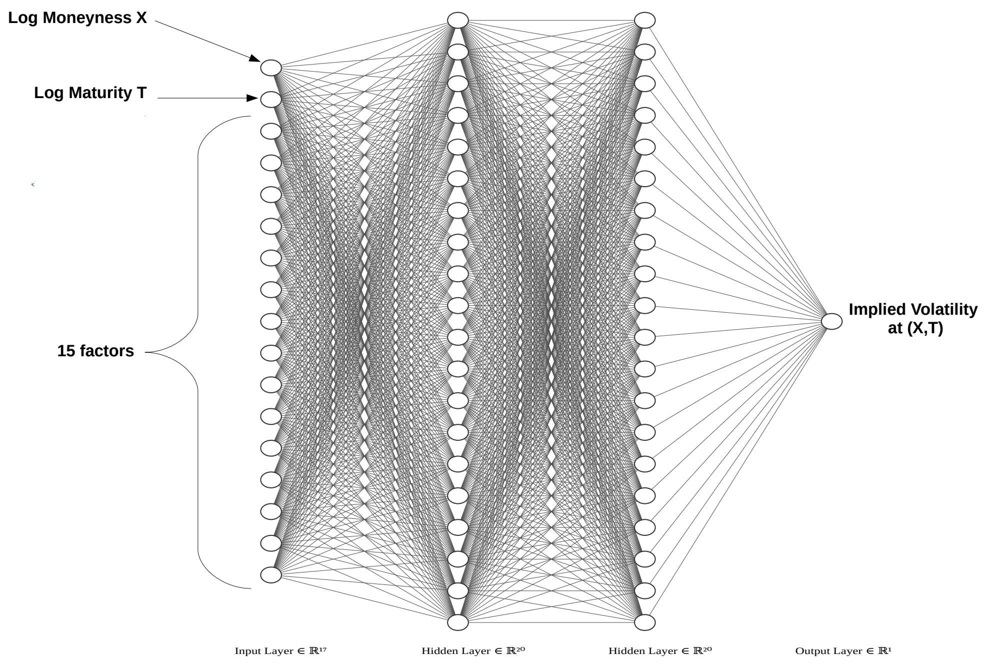


Figure 6.1: Network of the functional approach used in the equity case study (style FCNN of the NN-SVG software, cf. Figure 5.1).

6.1 Compression

We first calibrate our functional approach with the compression stage. Toward this end, we execute the optimization (1) on the training set and then calibrate codes with (2) for each observation in both testing and training data sets. The quality of the compression is assessed through the reconstruction errors reported in Table 6.1. By reconstruction error we mean the gap between the original surface and the surface induced by the code calibrated from (2).

We emphasize the difference between a reconstucted surface (as above) and a completed surface (considered later): the code leading to the completed surface is calibrated from an incomplete surface whereas the one for the reconstructed surface is obtained from a complete real surface.

	Functional	Functional with Forward
Training set	0.0070	0.0063
Testing set	0.0058	0.0064

Table 6.1: RMSEs for reconstructed implied volatilities.

In all four cases, the RMSEs in Table 6.1 are very small compared to the order of magnitude of implied volatilities (between 0.15 and 1.2). The results show no sign of overfitting (the reconstructions error are similar on the training set and the testing set). Moreover the comparison between the two columns of the table indicates that there is no benefit in including the forward price as an exogenous variable in our network.

Another way to assess the performance of the compression stage is to consider the worst compression, i.e. the surface yielding the highest reconstruction error. This worst reconstruction corresponds to a RMSE of 0.0096. It is represented in Figure 6.2, with the real surface on the top-left corner, the reconstructed couterpart on the top-right corner and the pointwise absolute difference between the two at the bottom.

We notice that the errors are concentrated on the upper tail (deep in the money call options) and for short maturities, which corresponds to illiquid options.

A bad reconstruction of a surface can also be used for qualifying it as an outlier. For instance, Figure 6.3 shows the implied volatility values corresponding to the most extreme strikes in Figure 6.2: original data points as dots and curves from the reconstructed surface. The left panel corresponding to the illiquid upper tail shows around the maturity 1.5 year a very low point that an expert would indeed qualify as an anomaly. The correction (i.e. the reconstructed surface) ignores this anomaly and has a more reasonable shape from a practitioner of view.

The left part of Figure 6.5 shows that the corrected surface is not prone to calendar arbitrage: the sensitivity to the maturity of the corresponding implied total variance is positive for every maturity T.[1] Sensitivity is computed thanks to adjoint automatic differentiation from neural network.

The above example shows that the functional neural network is indeed apt to learn from the compression stage a low-dimensional representation of likely observations. The low-dimensional representation gives large reconstruction errors to the surfaces of the testing set atypical with respect to the past observations (the training set in our experiments) and their latent structure.

[1]Regarding butterfly arbitrages, Durrleman's condition on the density (involving sensitivity with respect to forward log-moneyness, cf. Roper (2010)) can unfortunately not be checked for lack of data regarding dividends and discounting.

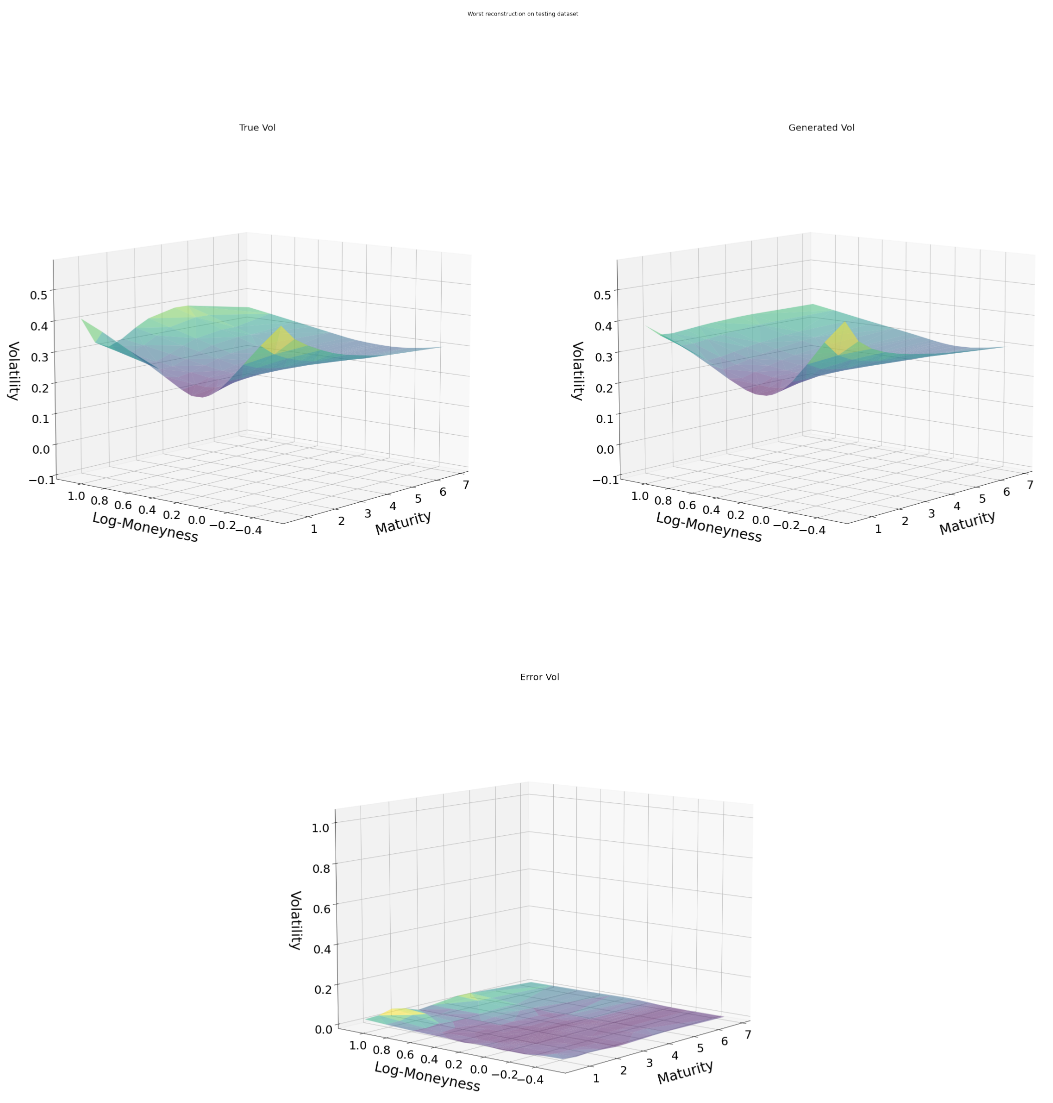


Figure 6.2: Original surface vs compressed surface yielding worst RMSE.

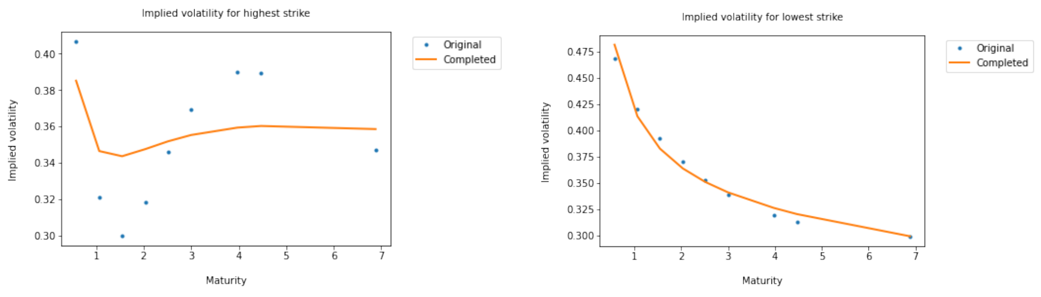


Figure 6.3: Tails of compressed surface vs original implied volatilities.

6.2 Outlier Detection and Correction

To confirm our views on outliers, we propose the following sanity check. An observation (first volatility surface in the testing set) is chosen and articially corrupted by doubling the values on four randomly chosen points: see the top-left corner in Figure 6.4.

Then we run the optimization (2) on this corrupted surface and obtain recalibrated codes. These code produce with the decoder the reconstructed surface (called correction) on the top-right corner. The correction is a smooth surface in which the corrupted values have been overwritten by values close to the original (non corrupted) ones. The bottom-left panel shows that only the corrupted values have been modifed significantly by the correction stage. The bottom-right figure indicates that the corrected surface is very close to the original one. The RMSE between the corrupted and the corrected surface is 0.0446 whereas the one between the correction and the original surface is 0.0151.

Note that the calendar arbitrage condition is still respected (see figure 6.5) for the correction, which exhibits a positive sensitivity of the implied total variance with respect to the maturity of the option.

This experiment confirms that a high reconstruction error is a good indicator of an outlier. The calibrated latent structure of the functional network smoothes the corresponding surface by identifying and correcting its anomalous points.

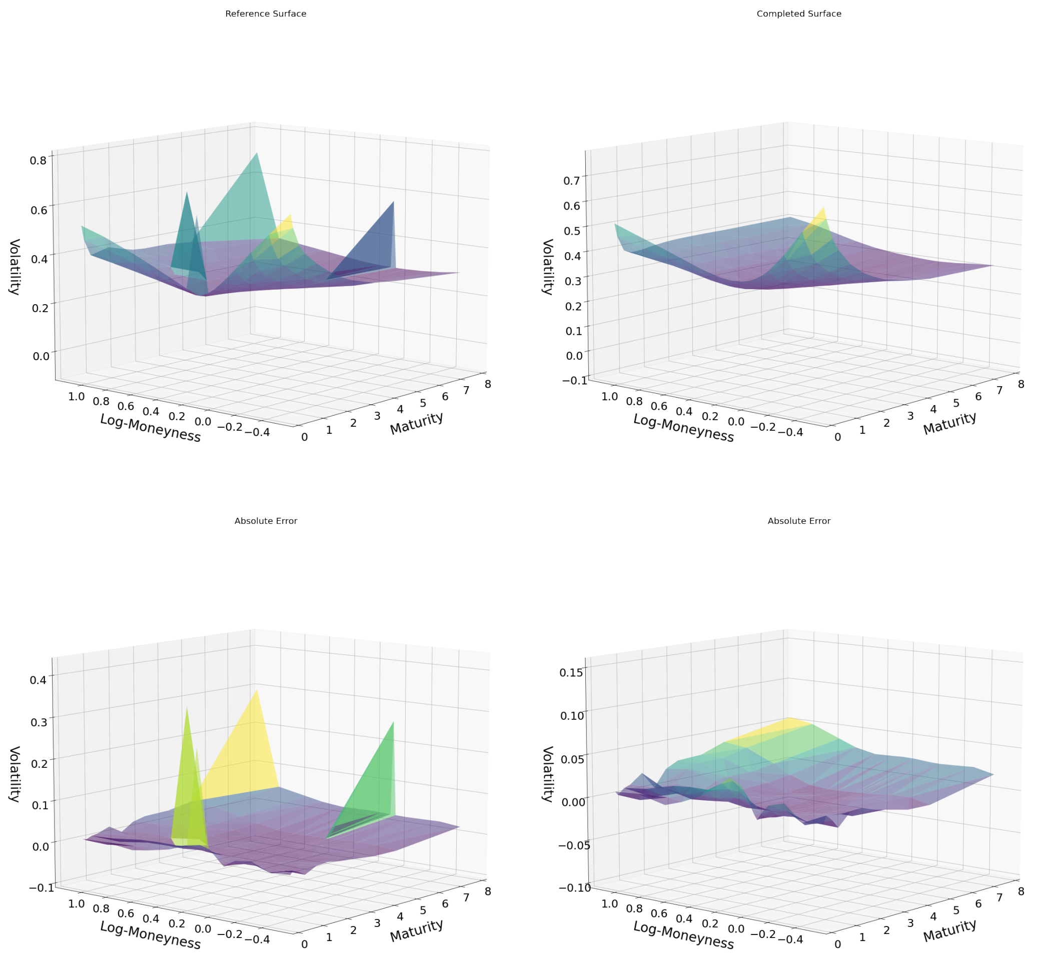
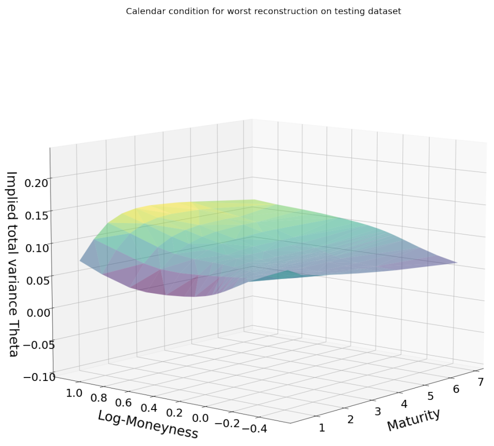
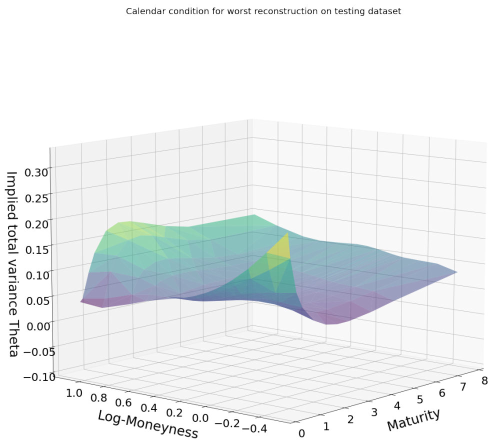


Figure 6.4: Outlier correction : Corrupted surface (Top-left), Corrected surface (Top-right), absolute error between corruption and correction (bottom-left), absolute error between correction and original surface before corruption (bottom-right)





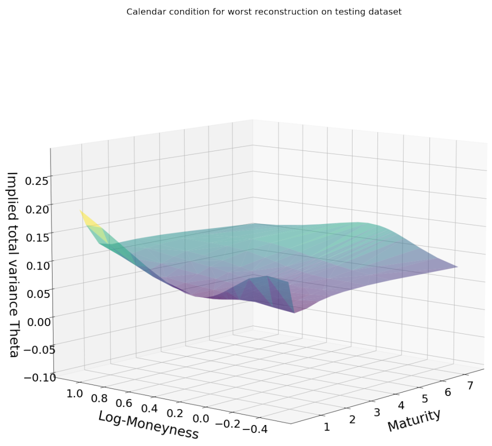


Figure 6.5: Implied total variance theta for worst reconstruction on top-left, outlier correction on top-right and worst completion at the bottom.

6.3 Completion

We now want to leverage on the calibrated low-dimensional latent structure of the functional network to recover a complete surface from partial information. Our hope is that this procedure will generate likely surfaces while approaching the available values (including on moving grids).

For each observation (surface) in the testing set, we select 40 points among the 255 points and remove all the others. Then we calibrate the latent variables by solving numerically the problem (2) with loss corresponding to these 40 points.

In order to benchmark the functional approach and assess the contribution of the historical data to the performance of the method, we report average completion errors [2] on the testing set for standard interpolation procedures (within each given surface, without exploitation of the information provided by the others):

1. Linear interpolation: given a triangulation of the 2D maturity and log-moneyness space base on the locations of the 40 available points, the interpolated value is taken as the barycenter on each triangle;

2. Spline interpolation: uses in each triangle as above a piecewise cubic interpolating Bezier polynomial (see Alfeld (1984) and the scipy documentation of the CloughTocher2DInterpolator method);

3. Gaussian process regression and squared exponential kernel: denoting by X the observed locations (maturity and log-moneyness), by Y the observed log-normal volatilities at locations X, by $X^\star$ the locations without values and by $Y^\star$ the unknown (looked for) implied volatilities, a Gaussian process regression assumes a Gaussian distribution

$$(Y, Y^\star) \sim \mathcal{N}(0, \begin{pmatrix} K(X,X) & K(X,X^\star) \\ K(X^\star,X) & K(X^\star,X^\star) \end{pmatrix}) \text{ with } K(X,X^\star)_{ij} = \sigma \exp\left(-\frac{\|x_i - x_j\|^2}{l^2}\right), \tag{7}$$

where σ and l are two hyperparameters calibrated by log-likelihood to the available values. In (7),

$$\|x_i - x_j\|^2 = (T_i - T_j)^2 + (\ln(m_i) - \ln(m_j))^2 \,,$$

where T denotes a maturity and $\ln(m)$ a log-moneyness;

4. Gaussian process regression with flat extrapolation; similar to 3, except that the Gaussian process predictor is only used for interpolation purposes; extrapolation whenever required is performed by the nearest neighbour method.

Again, a major difference between our functional (or neural net more generally) approach and these interpolation benchmarks is that, in order to interpolate a given surface, the neural network takes into account the information contained in all the surfaces of the data set, which is used as training set at the compression stage. In contrast, the above interpolation benchmarks only use the information provided by the available points of the currently interpolated surface, without consideration of the other surfaces in the data set. In particular, by Gaussian process regression in 3.

[2] Gap between the original surface and the completed one.

and 4., we just mean interpolation within a given surface, using the available points in this surface as training set (unrelated to the potential use of Gaussian processes as an alternative to neural networks in our compression/completion approaches, which would be unrealistic as discussed in Subsection 2.1).

Accordingly, the functional approach exhibits significantly smaller completion errors. In Table 6.2, we reported these errors for two different choices of the 40 visible points :

- Less correlated points, i.e. locations for which the implied volatilities are the less correlated;
- Uniformly spread points, i.e.a random selection of at least 2 points per maturity. The lowest maturity can be assigned 3 visible points in order to reach a total number of 40 points.

As the loss in (2) is now computed on much fewer points (partial information in this sense), the compression errors of the functional approach are obviously higher than the reconstruction errors from Table 6.1. Smaller error are reported in the second case above because less correlated points are rather located on short maturities, so that, in the first case little information, is available for the long maturities.

	Functional	Functional with Forward	Linear interpolation	Spline interpolation	Gaussian process no extrapolation	Gaussian process flat extrapolation
Less correlated points	0.0262	0.0265	0.0632	0.0462	0.0555	0.0459
Uniformly spread points	0.0076	0.0091	0.0211	0.0168	0.0201	0.0208

Table 6.2: RMSEs for completed implied volatilities.

All the completion results reported hereafter correspond to the case of uniformly spread visible points.

The completion method provided by the functional approach is also robust: even the worst completion does not produce an outlier, i.e.

- the completed surface is smooth,
- the completed surface has a shape similar to the one of the original surface (the pointwise errors between the original and the completed surfaces are uniformly distributed),
- the implied total variance sensitivity with respect to the maturity is still positive (see Figure 6.5), inducing no calendar arbitrage opportunity,
- tails are consistent with the original points (see Figure 6.7) and not irregular.

Such robustness is not provided by the interpolation benchmarks. For instance, in the case of the worst completion with the spline interpolation, the completed surface (top-right corner of Figure 6.8) is irregular in the tails.

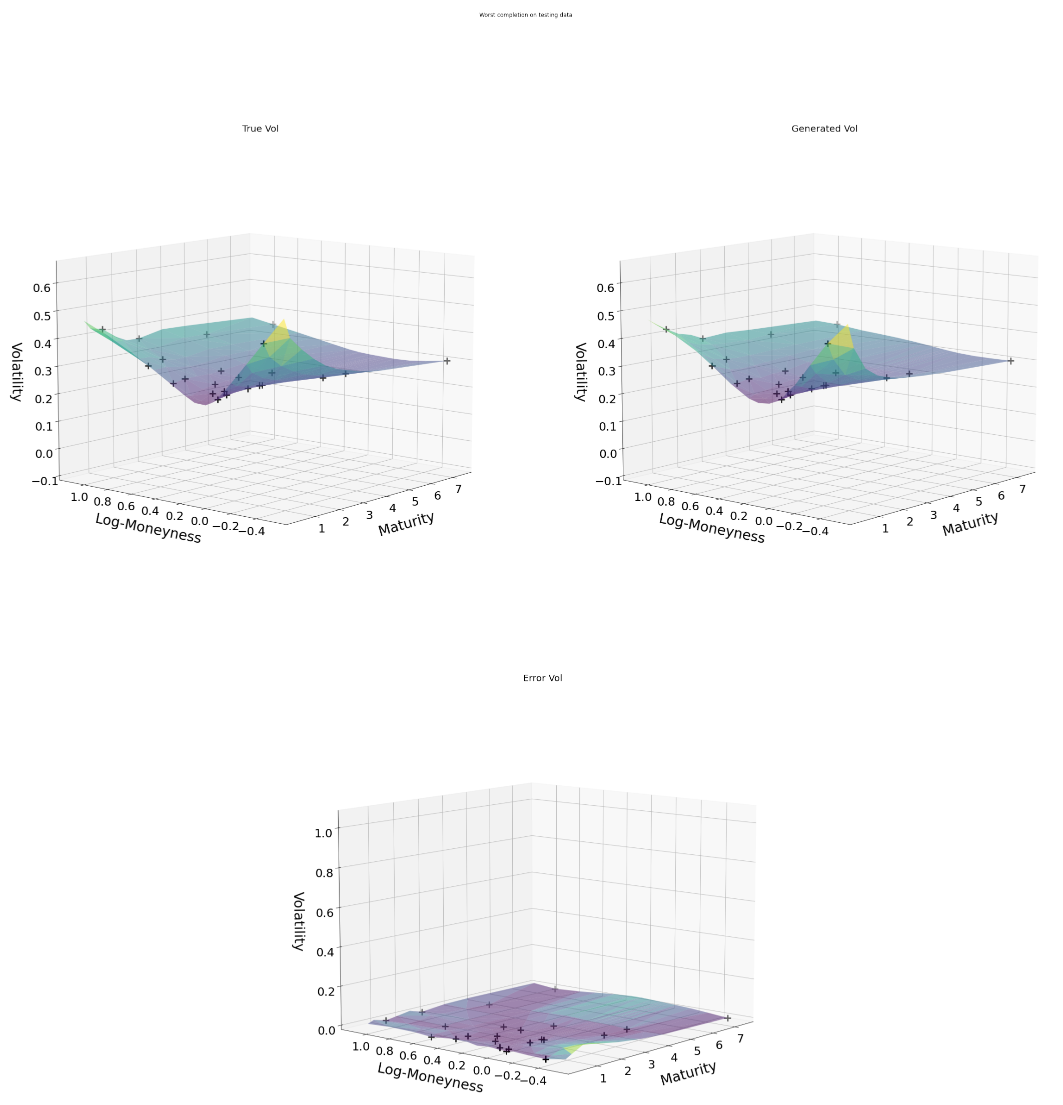


Figure 6.6: Original surface vs. completed surface yielding the worst RMSE. Black crosses mark visible points.

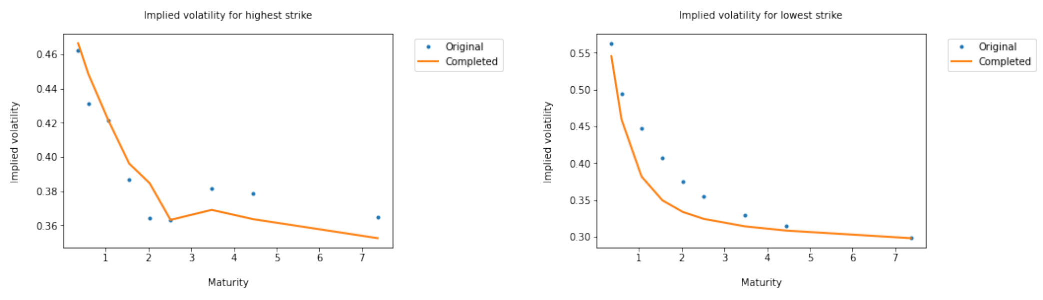


Figure 6.7: Tails of completed surface vs. original implied volatilities.

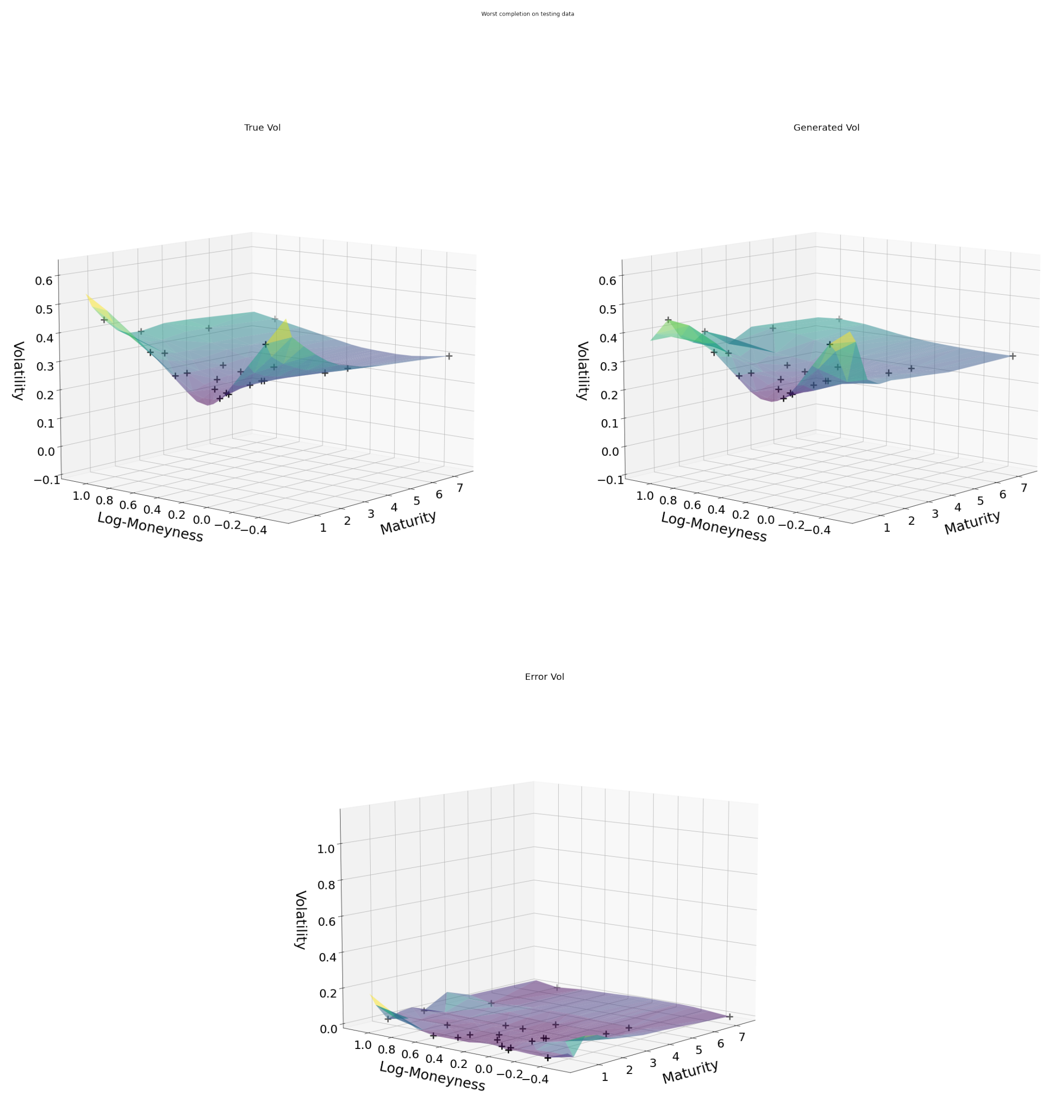


Figure 6.8: Original surface vs. completed surface yielding the worst RMSE with spline interpolation. Black crosses mark visible points.

7 At-the-Money Swaption Surfaces

The previous section was showing a case where the functional approach outperforms elementary interpolation benchmarks in an situation (in fact, the most common in the context of financial nowcasting applications) involving a moving grid.

We now consider an application where the grid is constant (after a preprocessing by our data provider) so that PCA or more classical autoencoder approaches are also available. The results show that the functional approach then performs as well as these classical benchmarks (which, however, would not be available on the original data with variable time-to-maturity).

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.

By contrast, very few works are dealing with the swaption implied volatility as a function of the expiry and tenor parameters (see Figure 7.1). One exception is Trolle

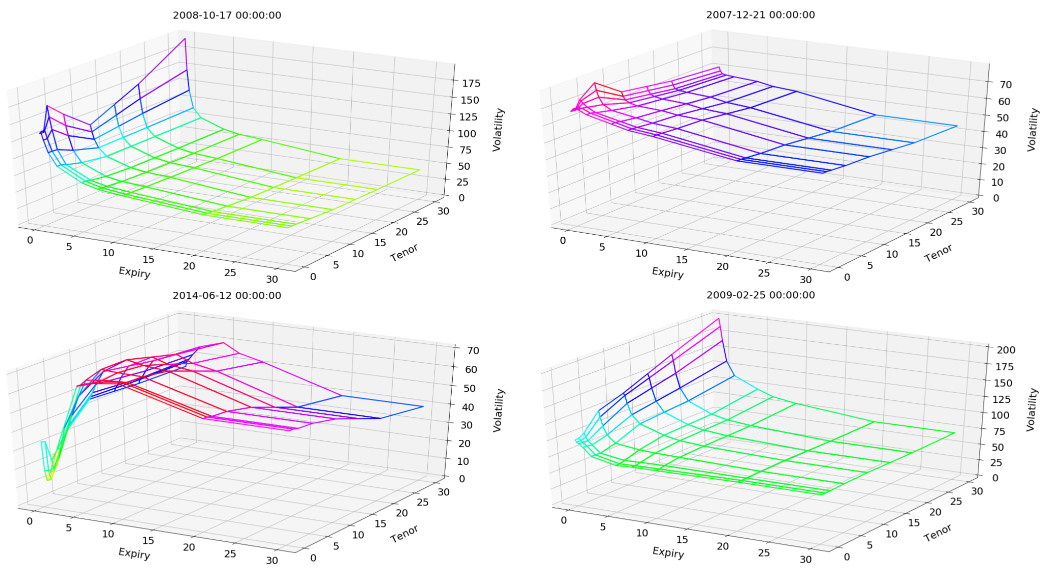


Figure 7.1: Different patterns of at-the-money swaption volatility surface.

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 a daily database of monocurrency (euro) ATM swaption normal[3] implied volatilities, covering 2400 business days corresponding to the period from 2007 to 2017. The training calibration and validation set Ω covers the 2007 to 2014 sub-period (1900 first observation days of the data set), whereas the test set Ω' ranges from 2015 to 2017 (500 subsequent ones). The data have been preprocessed by our provider 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 Ω', only keeping the volatility points corresponding to the grid nodes (U, T) in

$$\begin{aligned}&(1M, 3M), (1M, 10Y), (1M, 30Y), (6M, 2Y),\\ &(6M, 15Y), (5Y, 1Y), (5Y, 20Y), (10Y, 5Y).\end{aligned} \tag{8}$$

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.

7.1 Network Architectures

The corresponding architecture of the functional approach is then similar to the one used for equity derivatives in Section 5.1, except that the expiry U and tenor T are used as the localization inputs, and only 8 latent variables are used (instead of 15 previously): see Figure 7.2. Moreover, one can also incorporate the forward swap rates 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 7.2 an additional feature (input unit) containing the level 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.

The 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

[3]rather than Black–Scholes, because of the negative rates environment.

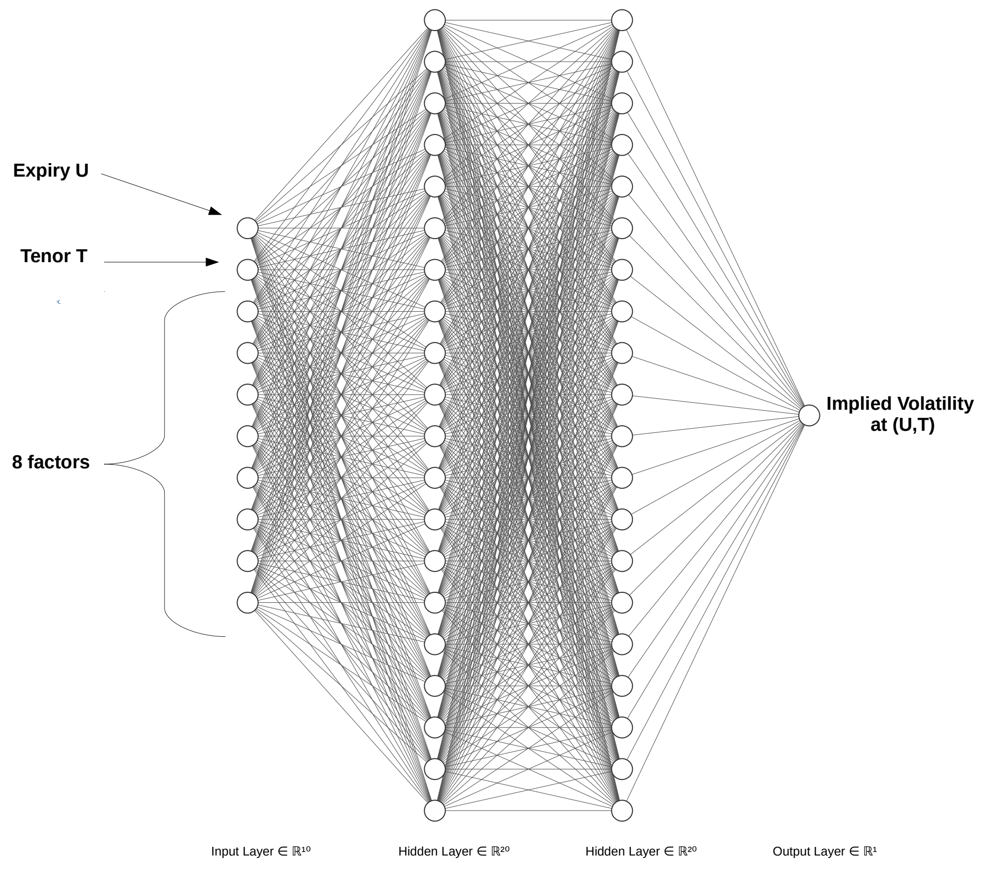


Figure 7.2: Network of the functional approach used in the swaption case study (style FCNN of the NN-SVG software, cf. Figure 5.1).

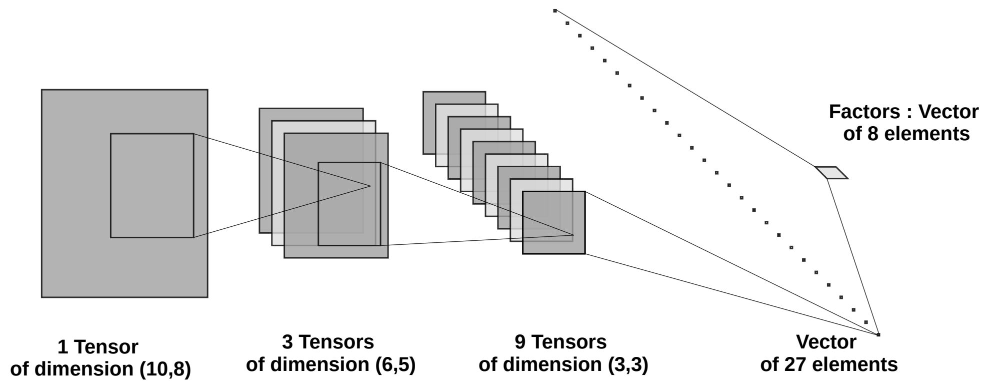

Factors : Vector of 8 elements
1 Tensor of dimension (10,8)
3 Tensors of dimension (6,5)
9 Tensors of dimension (3,3)
Vector of 27 elements

Figure 7.3: 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 7.3) 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 intermediate 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.

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.

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.

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.

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.

7.2 Numerical Results

Table 7.1 is a report on the errors of all our approaches (cf. Section 4.2). It is based on the absolute daily RMSEs (cf. (5) and (6)).

The last row of Table 7.1 displays 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.

At the completion stage, we take as initial factor values the volatility encoding of the previous day. Figure 7.4 shows the stability through calendar time of the codes obtained by the linear projection approach.

	Standard PCA	Linear projection	Convolutional autoencoder	Functional approach	Functional approach with forward rate
Average compression error on Ω	1.23	1.58	1.97	1.85	2.29
Average compression error on Ω'	3.71	3.54	6.19	3.77	3.02
Worst compression error on Ω [day] ([day])	4.15 [2008-12-03]	3.98 [2008-12-09]	7.18 [2008-12-08]	8.32 [2008-10-09]	6.93 [2008-10-10]
Worst compression error on Ω' [day] ([day])	5.76 [2016-04-28]	5.18 [2016-04-28]	12.0 [2015-07-07]	6.34 [2015-12-21	5.16 [2015-12-18]
Average completion error on Ω'	6.19	4.07	5.03	6.41	5.19
Worst completion error on Ω' [day] ([day])	12.6 [2015-06-30]	6.50 [2015-07-10]	9.89 [2015-07-10]	12.8 [2015-03-09]	9.09 [2016-01-14]
Training time in seconds	$\emptyset$	9	411	1287	276

Table 7.1: RMSEs in the sense of (5) and (6)

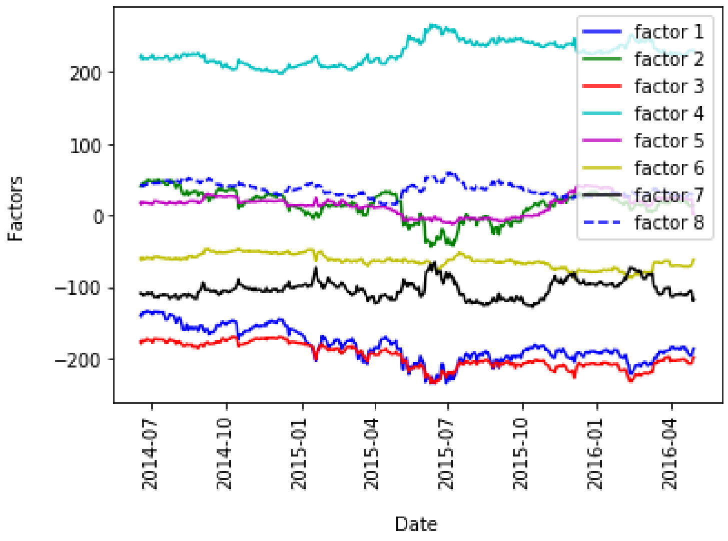


Figure 7.4: Time series of the factors obtained by encoding of the training observations under the linear projection approach.

As shown by Figure 7.5 in the case of the linear projection approach (but this is also true of the nonlinear approaches), the dominant errors are concentrated on

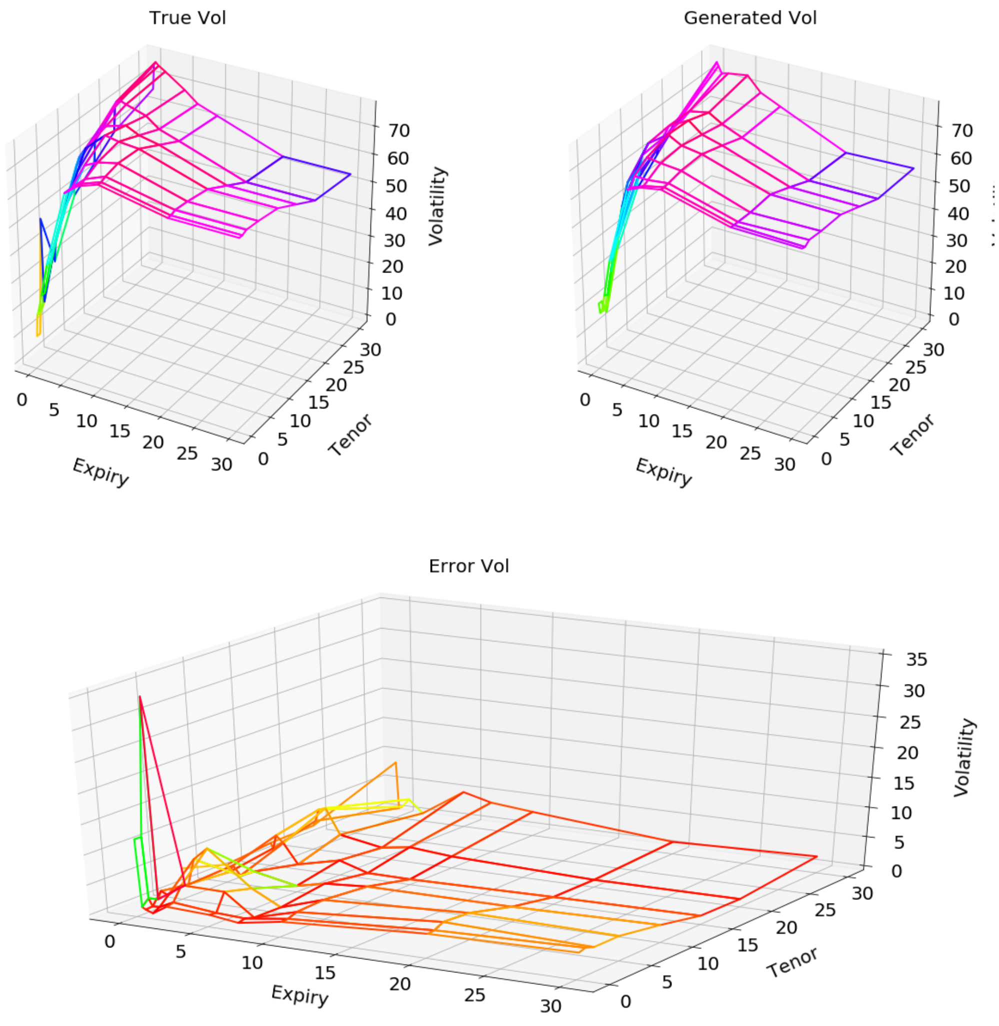


Figure 7.5: Linear projection approach: *(Top left)* Original (full) tensor; *(Top right)* Tensor $D_{\delta^\star}(c^\star)$ completed based on the 8 points of the latter given by (8); *(Bottom)* Pointwise absolute error between the two, for the worst observation in Ω'.

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 Table shows that this only has a minor positive impact.

The linear approaches are as accurate as the nonlinear ones and the convolutional approach is typically outperformed by at least the linear projection or the functional approach.

Figure 7.7 illustrates that the functional approach enables to interpolate smoothly the surface over an arbitrarily fine grid, in this case 10^4 points obtained by the corresponding interpolation of the tensor of Figure 7.6.

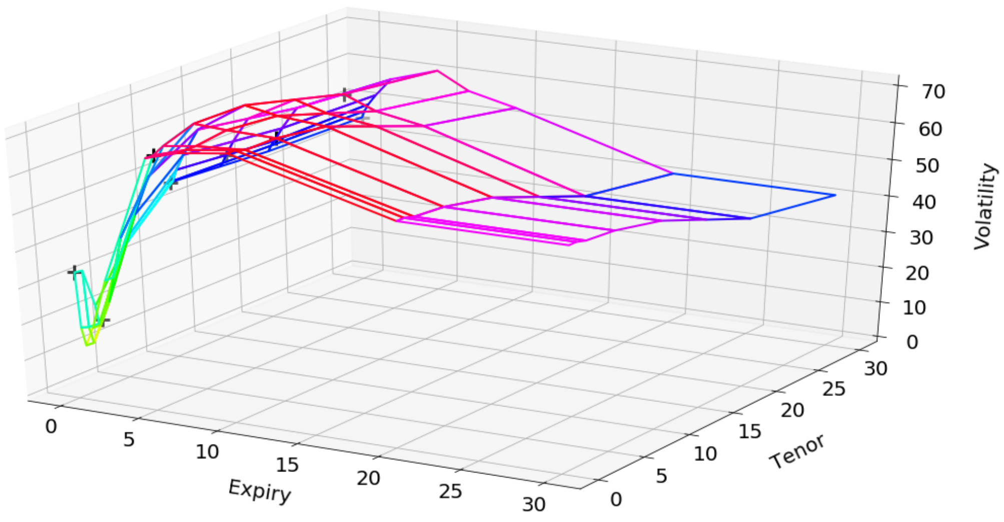


Figure 7.6: Complete tensor corresponding to the first observation in Ω'. The black crosses designate the "available points", specified by (8), that are used in the completion exercise.

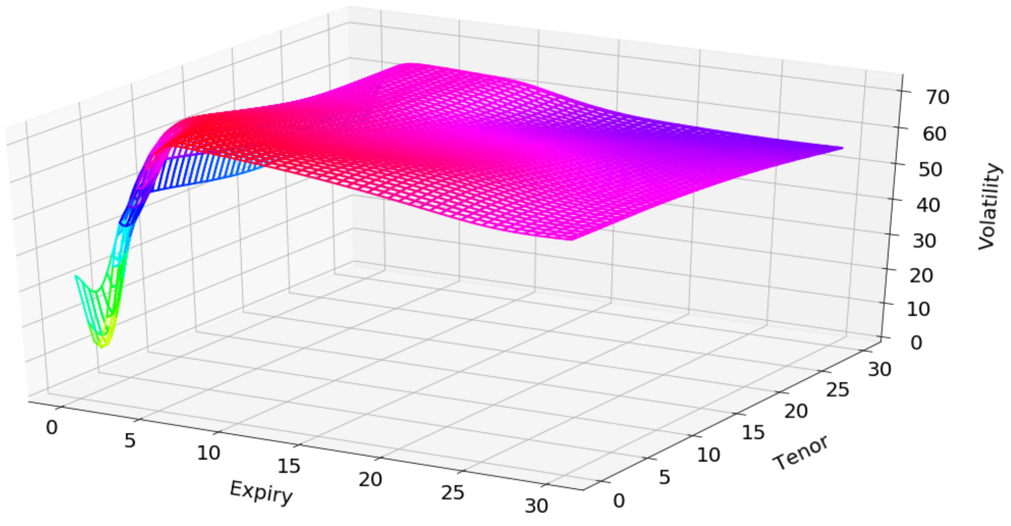


Figure 7.7: Surface with 10^4 points obtained by the functional approach applied to the first observation in Ω'.

8 Conclusions and Perspectives

We have devised a generic neural network based curve or surface (or more general tensor) compression/completion methodology, for which we propose two concrete specifications: the functional approach, amenable to the treatment of unstructured data with varying grid nodes (as natively the case in most financial nowcasting applications), and a convolutional autoencoder approach, including PCA or PCA-like projections as linear special cases, applicable in the special case of a constant grid (natively or possibly after some preprocessing). 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 the data, starting from about 10% of known values. But the functional approach is the only one that is able to directly deal (without preprocessing) with the most common situation of unstructured tensors. The only alternative is then naive interpolation benchmarks that do not exploit the data set, and which the functional approach is shown to outperform in our equity derivative case study.

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)), application of the method to the whole swaption volatility cube, strike dimension included (cf. Trolle and Schwartz (2010)), or specification of dynamics on the factors (for instance by Kalman filters).

References

Alfeld, P. (1984). A trivariate clough—tocher scheme for tetrahedral data. *Computer Aided Geometric Design 1*(2), 169–181.

An, J. and S. Cho (2015). Variational autoencoder based anomaly detection using reconstruction probability. *Special Lecture on IE 2*(1), 1–18.

Anandakrishnan, A., S. Kumar, A. Statnikov, T. Faruquie, and D. Xu (2018). Anomaly detection in finance: editors' introduction. In *KDD 2017 Workshop on Anomaly Detection in Finance*, pp. 1–7.

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.

Cansado, A. and A. Soto (2008). Unsupervised anomaly detection in large databases using Bayesian networks. *Applied Artificial Intelligence 22*(4), 309–330.

Cappozzo, A., F. Greselin, and T. B. Murphy (2020). Anomaly and novelty detection for robust semi-supervised learning. *Statistics and Computing*, 1–27.

Chaloner, K. and R. Brant (1988). A Bayesian approach to outlier detection and residual analysis. *Biometrika 75*(4), 651–659.

Chandola, V., A. Banerjee, and V. Kumar (2009). Anomaly detection: A survey. *ACM computing surveys (CSUR) 41*(3), 1–58.

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.

Goix, N., A. Sabourin, and S. Clémençon (2015). On anomaly ranking and excess-mass curves. In *Artificial Intelligence and Statistics*, pp. 287–295.

Goix, N., A. Sabourin, and S. Clémençon (2017). Sparse representation of multivariate extremes with applications to anomaly detection. *Journal of Multivariate Analysis 161*, 12–31.

Hastie, T., R. Mazumder, J. D. Lee, and R. Zadeh (2015). Matrix completion and low-rank svd via fast alternating least squares. *The Journal of Machine Learning Research 16*(1), 3367–3402.

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.

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

Kiran, B. R., D. M. 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.

Lakhina, S., S. Joseph, and B. Verma (2010). Feature reduction using principal component analysis for effective anomaly–based intrusion detection on NSL-KDD.

MacKay, D. J. and D. J. Mac Kay (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.

Nguyen, L. T., J. Kim, and B. Shim (2019). Low-rank matrix completion: A contemporary survey. *IEEE Access 7*, 94215–94237.

Omar, S., A. Ngadi, and H. H. Jebur (2013). Machine learning techniques for anomaly detection: an overview. *International Journal of Computer Applications 79*(2).

Patcha, A. and J.-M. Park (2007). An overview of anomaly detection techniques: Existing solutions and latest technological trends. *Computer networks 51*(12), 3448–3470.

Ro, K., C. Zou, Z. Wang, and G. Yin (2015). Outlier detection for high-dimensional data. *Biometrika 102*(3), 589–599.

Rocke, D. M. and D. L. Woodruff (1996). Identification of outliers in multivariate data. *Journal of the American Statistical Association 91*(435), 1047–1061.

Roper, M. (2010). Arbitrage free implied volatility surfaces. Preprint University of Sydney available at https://talus.maths.usyd.edu.au/u/pubs/publist/preprints/2010/roper-9.pdf.

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

Schlegl, T., P. Seeböck, S. M. Waldstein, G. Langs, and U. Schmidt-Erfurth (2019). f-anogan: Fast unsupervised anomaly detection with generative adversarial networks. *Medical image analysis 54*, 30–44.

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.

Strub, F. and J. Mary (2015). Collaborative filtering with stacked denoising autoencoders and sparse inputs. In *NIPS workshop on machine learning for eCommerce.*

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*.