Research internship: Variational autoencoders for arbitrage-free implied volatility surfaces generation and completion

Author: Abderrahim Mama
Tutor: Vladimir Lucic
Supervisor: Nizar Touzi

Marex Solutions

March 2022 - August 2022

Abstract

Market data of non-liquid options is often incomplete. This project suggests a method based on variational autoencoders inspired by ergeron et al's research paper [3] to complete missing points on partially observed volatility surfaces without introducing static arbitrage. After drawing a parallel between autoencoders and principal component analysis, we compare our new variational autoencoder architecture to the architecture proposed in [3] and we show its utility in completing and generating arbitrage-free equity volatilities.

Foreword

This document covers my work during **the first two months** of my six-month internship at Marex Solutions in London. My task was to explore Bergeron et al's research paper [3] and try to improve their findings. In this report, I studied in-depth the ability of variational autoencoders to reproduce implied volatility surfaces without creating any static arbitrage. In this work, I suggested a new variational autoencoder architecture dedicated to learning how to generate and complete arbitrage-free implied volatility surfaces.

Section 1: A brief introduction of the evolution of implied volatility models and the emergence of deep learning in finance.

Section 2: A definintion of deterministic and variational autoencoders with a proof that I elaborated to show the relationship between autoencoders and principal component analysis (PCA).

Section 3: A definition of delta implied volatility surfaces and a characterization of static arbitrage in terms of delta instead of usual characterizations in terms of log-forward moneyness.

Section 4: The data interpolation method that I opted for to have a fixed size implied surface from listed options. A presentation of the new architecture that I suggested to improve the results of the cited paper [3].

Section 5: All the numerical results of the new architecture and a comparison to the results yield by Bergeron et al' variational autoencoder architecture on our internal data sets.

Section 6: Our method to complete missing points on partially observed implied volatility surfaces on illiquid options.

Section 7: A summary of our findings and ideas for further work that can be done beyond the scope of this project.

Acknowledgements

First of all, I would like to thank my tutor Vladimir Lucic, head of Marex Solutions Technology & Quants, for his great help and cooperation during the internship.

I also warmly thank Mehdi Mlaiki, head of Trading and Co-head of Derivatives Engine, and Nilesh Jethwa, CEO of Marex Solutions, for giving me the opportunity to carry out a six-month internship within Marex Solutions.

I would like to thank Souleymane Dieye, Quantitative analyst at Marex Solutions, for introducing me to static arbitrage notions and exotic options through the book Exotic Options and Hybrids of Mohamed Bouziba and Adel Osseiran.

A special thanks goes to Ali Brahimi, quantitative analyst at Marex Solutions, for his ideas about variational autoencoders.

I thank, Aylin Chakaroglu and Stephane Chaulet, quantitative analysts at Marex Solutions, for their involvement in the project and the good atmosphere they have created in the team.

Last but not least, I would like to thank my school supervisor Nizar Touzi for his course Stochastic Calculus in Finance at L'École polytechnique which was very helpful for my internship topic.

List of symbols

$\lceil x \rceil$	ceil function of x, smallest integer greater or equal than x
$\lfloor x \rfloor$	floor function of x, greatest integer smaller or equal than x
1_n	$1_n := \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} \in \mathbb{R}^{n \times 1}$
$(x)^+$	$(x)^+ := \max(x, 0)$
$\mathcal{C}^n(\mathbb{R})$	continuous functions $f : \mathbb{R} \to \mathbb{R}$ with continuous derivatives up to order n
$\mathcal{N}(\mu, \gamma)$	normal distribution with mean vector μ and covariance matrix γ
$\mathcal{N}(x)$	cdf of a normal random variable $\mathcal{N}(x) = \int_{-\infty}^{x} \frac{1}{\sqrt{2\pi}} e^{-u^2/2}\, du$
$\omega(K, T)$	total implied variance for a strike K and a maturity T
$\sigma(K, T)$	implied volatility for a strike K and a maturity T
$\sim$	if a random variable $W \sim D$, then W has the distribution D
PCA	principal component analysis
RELU	activation function used in neural networks $\text{RELU}(x) := \max(0, x)$
SOFTPLUS	activation function used in neural networks $\text{SOFTPLUS}(x) := \log(1 + e^x)$
$\tilde{\omega}(\Delta, T)$	total implied variance for delta Δ and expiry T
$\tilde{\sigma}(\Delta, T)$	implied volatility for delta Δ and expiry T
I_d	identity matrix in $\mathbb{R}^{d \times d}$
K	exercise price of an option
$n(x)$	pdf of a normal random variable $n(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2}$
T	expiry date of an option
$tr(X)$	trace of the matrix X
X^T	transpose of the matrix X
$\| X \|_F$	Frobenius norm of the matrix X, $\| X \|_F = \sqrt{tr(X^T X)}$

Contents

1 Introduction **5**

2 Autoencoders **6**
- 2.1 Deterministic Autoencoder (DAE) . . . 6
- 2.2 PCA-Autoencoder relationship . . . 6
- 2.3 Variational autoencoders (VAE) . . . 11
  - 2.3.1 The main idea . . . 11
  - 2.3.2 The practical side . . . 12
  - 2.3.3 The latent variable distribution . . . 13
  - 2.3.4 Loss function . . . 13

3 Implied volatility surfaces **14**
- 3.1 Delta implied volatility surface . . . 15
- 3.2 Static arbitrage . . . 17
  - 3.2.1 Butterfly arbitrage . . . 17
  - 3.2.2 Calendar arbitrage . . . 19

4 Methodology **21**
- 4.1 Data interpolation . . . 21
- 4.2 Feed-forward architecture used in [3] . . . 24
- 4.3 Our architecture . . . 24
  - 4.3.1 Convolutional layers . . . 24
  - 4.3.2 Hybrid architecture . . . 27

5 Numerical results **28**
- 5.1 Strike implied volatility surface . . . 29
- 5.2 Delta implied volatility surfaces . . . 31
  - 5.2.1 A preliminary experiment . . . 31
  - 5.2.2 Latent space effect . . . 32
  - 5.2.3 Cyclic β . . . 33
  - 5.2.4 Modified loss . . . 36
  - 5.2.5 Data standardization . . . 37
  - 5.2.6 Arbitrage-free surfaces . . . 38
  - 5.2.7 Error heatmap . . . 40
  - 5.2.8 Multi-asset training . . . 41

6 Implied volatility surface completion **44**

7 Conclusion **46**

References **47**

Appendix **49**

1 Introduction

In the famous Black-Scholes model [4], the implied volatility of all options on the same underlying must be the same and equal to the historical (realized) volatility (standard deviation of annualized returns) of the underlying.

However, from market-quoted option prices, one observes that the implied volatility is always greater than the historical volatility of the underlying. The implied volatilities of different options on the same underlying depend on their strikes and maturity dates. Options whose underlying is governed by a geometric Brownian Motion should theoretically have a flat and unchanging volatility surface since volatility is constant. However, the implied volatility surfaces for most assets are not flat in practice [23], with the graph of implied volatility plotted against strike prices showing a pronounced **skew**, which for some assets becomes a U-shaped curve known as the volatility **smile**. This has been proven in extensive literature with one example being Rubinstein (1994) [27] and Jackwerth and Rubinstein (1996) [19], who showed that the implied volatilities of stock and stock index behave like a skew and those of foreign currencies exhibit a smile.

Based on options market prices, option traders can estimate the volatility surfaces for a wide variety of underlying assets, where some of the points may be deduced directly as they are the actively traded options, while the remaining can usually be completed through interpolation [8].

The attempts to model the implied volatility surface are numerous. **The implied volatility function** (IVF) model is one of the first attempts which develops a one-factor model for an asset price. In 1994, Derman and Kani [9] and Dupire [10] hypothesized that asset return volatility is a deterministic function of asset price and time, and developed the IVF model. However, Han and Suo (2002) [18] showed that there is no guarantee that the IVF model produces good pricing and hedging for many exotic options.

Parametric models were also developed to model the volatility surface. Gatheral presented **The stochastic volatility inspired** (SVI) in 2004 and called the first version of the model the raw SVI. Then, Gatheral and Jacquier (2013) [15] introduced a simple closed-form representation of arbitrage-free SVI surfaces known as surface stochastic volatility inspired (SSVI).

Leveraging its fast computational time, Deep learning techniques are becoming widely used in the field of mathematical finance. An approach where volatility is assumed to be a product of an existing model and a neural network was introduced by Ackerer, Tagasovska, and Vatter (2020) [1]. Chataigner, Crepey, and Dixon (2020) [7] used Feedforward neural networks (FNNs) [26] to fit European vanilla put option prices given the maturity and the strike. The goal was to use Automatic differentiation (AD) [24] on the network to determine the local volatility based on Dupire formula. Bergeron et al. (2021) [3] trained their neural network without any assumption on the process driving the underlying asset. We will focus on Bergeron et al.'s paper [3] and try to explore the ability of Variational Autoencoders (VAE) [20] to reconstruct, complete and generalize implied volatility surfaces on Equity options.

2 Autoencoders

2.1 Deterministic Autoencoder (DAE)

A deterministic autoencoder (DAE) can be seen as **a generalization of PCA**, where instead of finding a low dimensional hyperplane in which the data lies, it is able to learn a non-linear manifold.

A neural network that is trained to learn the identity function is called an autoencoder. Its output layer has the same number of nodes as the input layer, and the cost function is some measure of the reconstruction error.

Autoencoders are **unsupervised** learning models, and they are often used for the purpose of dimensionality reduction [17]. A simple autoencoder that implements dimensionality reduction is a feedforward autoencoder with at least one layer that has a smaller number of nodes, which functions as a bottleneck. After training the neural network using backpropagation, it is separated into two parts: the layers up to the bottleneck are used as an encoder, and the remaining layers are used as a decoder.

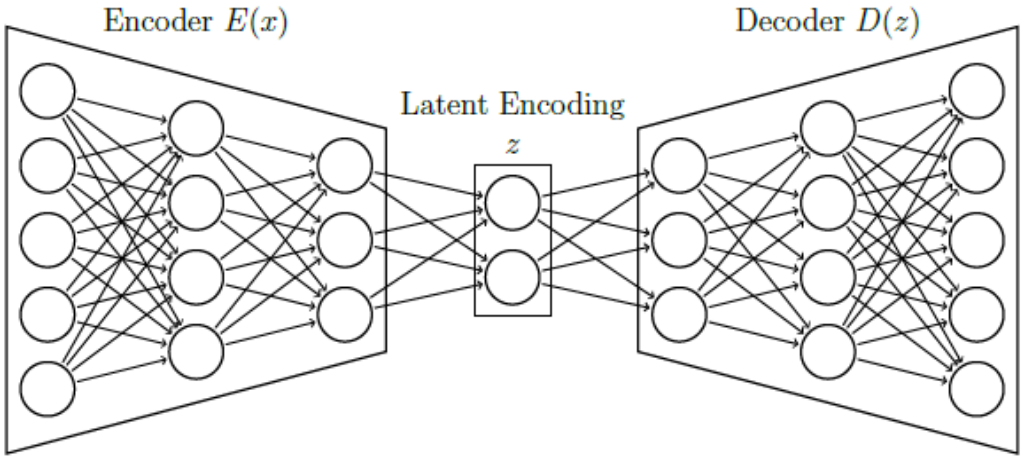


Figure 1: Deterministic autoencoder

2.2 PCA-Autoencoder relationship

Intuition behind PCA-DAE link

In the case of linear autoencoder with one-layer encoder and one-layer decoder, the autoencoder would achieve the same latent representation as ***Principal Component Analysis (PCA)*** *[25].*

Proof

Preliminary tools:

We will use the following properties in the proof

1. *Properties of Moore-Penrose pseudoinverse* [2] : For a matrix $D \in \mathbb{R}^{n\times d}$, we denote its Moore-Penrose pseudinverse by $D^+ \in \mathbb{R}^{d\times n}$.

$$(D^+)^T = (D^T)^+ \tag{i}$$
$$(D^+)^T = (DD^T)^+ D \tag{ii}$$
$$D = DD^T(D^+)^T = (D^+)^+ \tag{iii}$$
$$\text{If } D^TD = I_d \text{ then } D^+ = D^T \tag{iv}$$

$$\begin{cases} \text{For } B \in \mathbb{R}^{n\times k}, DZ = B \text{ has solution(s) if } DD^+B = B. \\ \text{They are all given by } Z = D^+B + (I_d - D^+D)L \\ \text{for any arbitrary } L \in \mathbb{R}^{d\times k}. \end{cases} \tag{v}$$

$$\begin{cases} \text{There exist } V_r \text{ such that } DD^+ = V_rV_r^T \\ \text{where } r \text{ is the column rank of } D \text{ and } V_r \in \mathbb{R}^{n\times r} \\ \text{such that } V_r^TV_r = I_r \end{cases} \tag{vi}$$

A proof of (vi) can be found in the appendix 7.

For the property (v), the solution is unique if and only if D has full column rank, in which case $I_d - D^+D$ is a zero matrix.

2. *SVD*: The Singular value decomposition (SVD) states that any matrix $D \in \mathbb{R}^{n\times d}$ can be written as $D = V\Sigma U^T$ where :

$$\begin{cases} \mathrm{V} := \begin{pmatrix} V_k & V' \end{pmatrix} \in \mathbb{R}^{n\times n},\ V_k \in \mathbb{R}^{n\times k},\ V' \in \mathbb{R}^{n\times(n-k)} \text{ and } V^TV = I_n \\ \mathrm{U} := \begin{pmatrix} U_k & U' \end{pmatrix} \in \mathbb{R}^{d\times d},\ U_k \in \mathbb{R}^{d\times k},\ U' \in \mathbb{R}^{d\times(d-k)} \text{ and } U^TU = I_d \\ \Sigma = \begin{pmatrix} \lambda_1 & & 0 \\ & \ddots & \\ 0 & & \lambda_d \\ \hline & 0_{(n-d)\times d} & \end{pmatrix} := \begin{pmatrix} \Sigma_k & 0_{k\times(d-k)} \\ 0_{(n-k)\times k} & \Sigma' \end{pmatrix} \in \mathbb{R}^{n\times d} \text{ a diagonal matrix} \\ \lambda_1 \geq ... \geq \lambda_d \text{ are the singular values of X, } \Sigma_k \in \mathbb{R}^{k\times k} \text{ and } \Sigma' \in \mathbb{R}^{(n-k)\times(d-k)} \end{cases}$$

3. Eckart–Young–Mirsky lemma [11] :
For any matrix $D \in \mathbb{R}^{n\times d}$, $\min\limits_{\text{rank}(D_k)\leq k} \|D - D_k\|_{\text{F}} = \|D - D^*\|_F$ *where* $D^* = V_k\Sigma_k U_k^T$ *introduced by the singular Value Decomposition theorem (SVD) above.* D^* *is the unique minimizer when* $\lambda_k \neq \lambda_{k+1}$.

We will keep the **same notations** during the rest of the proof.

Core of the proof:

Let $X \in \mathbb{R}^{n\times d}$ be a **centered and full column rank** observation matrix where n is the number of d-dimensional observations. Let us suppose that **there exists** $k \leq d$ **such that** $\lambda_k \neq \lambda_{k+1}$. So the solution of Eckart–Young–Mirsky lemma is unique.

1. **PCA** :
Let k be the number of PCA components.

With the notations above, $X^TX = U \begin{pmatrix} \lambda_1^2 & & 0 \\ & \ddots & \\ 0 & & \lambda_d^2 \end{pmatrix} U^T$

Applying PCA, we consider the first k columns of U, i.e U_k defined above.

Interestingly, U_k solves the following minimization problem

$$\min_{\substack{W\in\mathbb{R}^{d\times k} \\ W^TW=I_k}} ||X - XWW^T||_F \tag{1}$$

In fact,

$$\begin{aligned} XU_kU_k^T &= V\Sigma U^TU_kU_k^T \\ &= V\Sigma \underbrace{\begin{pmatrix} U_k^T \\ U'^T \end{pmatrix} U_k}_{\begin{pmatrix} I_k \\ 0_{d\times k} \end{pmatrix}} U_k^T \\ &= V \begin{pmatrix} \Sigma_k \\ 0_{n\times k} \end{pmatrix} U_k^T \\ &= V_k\Sigma_kU_k^T \end{aligned}$$

By the lemma above we conclude that

$$XU_kU_k^T \text{ is the unique minimizer of } \min_{\substack{X_k\in\mathbb{R}^{n\times d} \\ \text{rank}(X_k)\leq k}} \|X - X_k\|_{\text{F}}$$

But $\mathcal{C} := \{XWW^T | W \in \mathbb{R}^{d\times k}, W^TW = I_k\} \subset \{X_k \in \mathbb{R}^{n\times d}, \text{rank}(X_k) \leq k\}$ and $XU_kU_k^T \in \mathcal{C}$ then the minimizer $XU_kU_k^T$ of $\{X_k \in \mathbb{R}^{n\times d}, \text{rank}(X_k) \leq k\}$ is in $\mathcal{C}$. Consequently, U_k solves (1).

Yet, the minimizer of (1) is not unique: $W = U_k Q$ is also a solution, where $Q \in \mathbb{R}^{k \times k}$ is any orthogonal matrix. Multiplying U_k from the right by Q transforms the first k right eigenvectors of X into a different orthonormal basis for the same subspace.

The set of solutions for (1) is $\{U_k Q,\ Q \in \mathbb{R}^{k \times k} \text{ and } Q^T Q = I_k\}$.

2. **Autoencoder** :
Let us consider an autoencoder with one-linear layer encoder and one-linear layer decoder. Let $W_1 \in \mathbb{R}^{d \times k}$, $W_2 \in \mathbb{R}^{k \times d}$ and $b_1 \in \mathbb{R}^{1 \times k}$, $b_2 \in \mathbb{R}^{1 \times d}$ be the weights and biases associated to the encoder and the decoder respectively.

If we define the error function to optimize as $J := ||f_\theta(X) - X||_F^2$ where f_θ is the autoencoder network and θ its parameters, then the autoencoder on the input matrix X solves :

$$\min_{W_1, W_2, b_1, b_2} J = \min_{W_1, W_2, b_1, b_2} ||X - (XW_1 + 1_n b_1) W_2 - 1_n b_2||_F^2 \tag{2}$$

By setting $\partial_{b_2} J = 0$, we obtain

$$1_n^T (X - (XW_1 + 1_n b_1) W_2 - 1_n b_2) = 0$$

Then

$$b_2 = \frac{1}{n} \left(1_n^T X - 1_n^T X W_1 W_2 - n b_1 W_2\right)$$

But X is a **centered** observation matrix so $1_n^T X = 0_{1 \times d}$. Thus,

$$b_2 = -b_1 W_2$$

When we replace b_2 in (2), the problem (2) becomes :

$$\min_{W_1, W_2} ||X - XW_1 W_2||_F^2 \tag{3}$$

Thus, for any b_1, the optimal b_2 is such that the problem becomes independent of b_1.

By setting $\partial_{W1} ||X - XW_1 W_2||_F^2 = 0$, we obtain :

$$X^T (X - XW_1 W_2) W_2^T = 0$$

Then

$$X^T X W_2^T = X^T X W_1 W_2 W_2^T$$

As X is full column rank, then $(X^T X)^{-1}(X^T X) = I_d$ and we have

$$W_2 W_2^T W_1^T = W_2 \tag{*}$$

Let us define $B := W_2$ and $D := W_2 W_2^T$. The equation (*) becomes

$$DW_1^T = B \tag{**}$$

We notice that

$$\begin{aligned} DD^+B &= W_2 W_2^T \underbrace{(W_2 W_2^T)^+ W_2}_{= (W_2^T)^+ \text{ by(ii)}} \\ &= W_2 W_2^T (W_2^T)^+ \\ &= W_2 \qquad\qquad \text{from (iii)} \\ &= B \end{aligned}$$

From (v), we conclude that $W_1 = (D^+B)^T = ((W_2^T)^+)^T = W_2^+$ (from (i)) is a solution of (**).

Thus, the minimization (3) solved by the autoencoder remains with respect to a single matrix:

$$\min_{W_2 \in \mathbb{R}^{k \times d}} ||X - X W_2^+ W_2||_F^2 \tag{4}$$

which, thanks to the property $(D^+)^+ = D$, can be written as

$$\min_{W_2 \in \mathbb{R}^{d \times k}} ||X - X W_2 W_2^+||_F^2 \tag{5}$$

$W_2 W_2^+$ is the orthogonal projection operator onto the column space of W_2.

We notice also that when W_2 is full column rank, $W_2^T W_2$ is invertible and we obtain the usual formula of the orthogonal projection operator onto the column space, i.e $W_2 W_2^+ = W_2 (W_2^T W_2)^{-1} W_2^T$

Using (vi), the minimization (5) is equivalent to

$$\min_{\substack{W_2 \in \mathbb{R}^{d \times r} \\ r \leq k \\ W_2^T W_2 = I_r}} ||X - X W_2 W_2^T||_F^2 \tag{6}$$

From the uniqueness of the minimizer in Eckart–Young–Mirsky lemma, we conclude that (6) is equivalent to (1) solved by PCA.

As a result, it is possible to solve (1) by first solving the unconstrained problem (5) solved by the autoencoder, then orthonormalizing the columns of the solution, e.g. using the Gram-Schmidt process. However, this does not recover U_k, but rather $U_k Q$ for some unknown orthogonal matrix Q.

QED □

Remark: The linear autoencoder is said to apply PCA to the input data in the sense that its output is a projection of the data onto the low dimensional principal subspace. However, unlike actual PCA, the coordinates of the output of the bottleneck are correlated and are not sorted in descending order of variance.

NB: Full rank observation matrix X and existence of $k \leq n$ such that $\lambda_k \neq \lambda_{k+1}$ are tangible hypothesises because in a realistic environment with noise and finite precision, we can always slightly perturb the conditions so as to fulfill these hypothesises.

2.3 Variational autoencoders (VAE)

2.3.1 The main idea

A variational autoencoder [20] consists of both an encoder and a decoder. Similarly to a DAE, it attempts to learn the identity function but with a small perturbation in the latent space. Instead of encoding an input as a single point in the latent space, the input is encoded as a distribution over the latent space. Then the encoder, samples a point from the learned distribution and decodes it.

Let us consider a set $\{\sigma_i\}_{i=1}^n$ of n implied volatility surfaces. We assume that σ_i are generated by some random process, involving an unobserved continuous random variable z called **latent variable**. The process consists of two steps:

1. a value z_i is generated from some prior distribution $p_{\theta*}(z)$
2. a value σ_i is generated from some conditional distribution $p_{\theta*}(\sigma|z)$

We assume that the prior $p_{\theta*}(z)$ and likelihood $p_{\theta*}(\sigma|z)$ come from parametric families of distributions $p_\theta(z)$ and $p_\theta(\sigma|z)$. As z is unknown, the encoder learns the prior $p_{\theta*}(z)$ by inferring z following the distribution $p_\phi(z|\sigma)$ (ϕ stands for the encoder parameters) which has to be as close as possible from the prior $p_{\theta*}(z)$. The distance used to define this closeness is the **Kulback-Leibler Divergence (D_{KL})** [28] (not a metric because of the asymmetry). It is defined as $D_{KL}(P||Q) = \int P(x) \log\left(\frac{P(x)}{Q(x)}\right) dx$ where P and Q are two strictly positive probability distributions. And the decoder infers σ given a sample of z and learns by maximizing the likelihood $p_\theta(\sigma|z)$ (θ stands for the parameters of the decoder). In this model, θ and ϕ have to be jointly learnt.

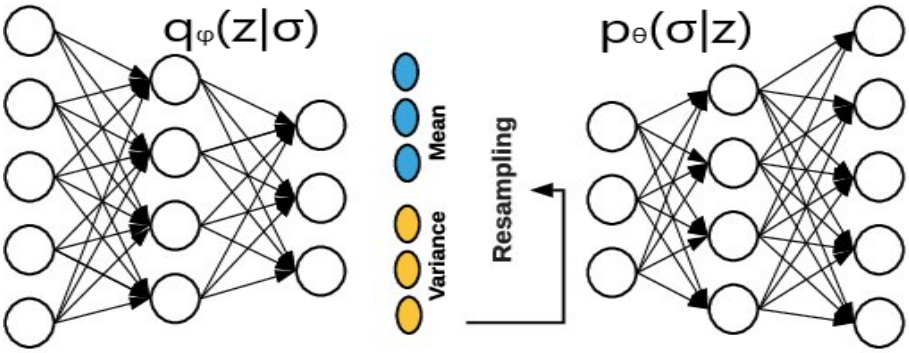


Figure 2: Variational autoencoder

Conditional autoencoder: We will be using a specific type of variational autoencoder called **Conditional autoencoder (CAE)**. The decoder will infer the implied volatility surface based on the latent variable z + some additional information given from outside the network. Namely, we will provide the **maturity** T, the **strike** price K and in some experiments the underlying **spot** price S. So the decoder will maximise the likelihood $p_\theta(\sigma|z, K, T, S)$.

2.3.2 The practical side

We will denote $p_\theta(\sigma|z, K, T, S)$ by $p_\theta(\sigma|z)$ because the other parameters used in the CAE do not have any impact in the following calculation. That is to say that the theoretical study of VAE is quite similar to the CAE one.

The goal is to maximise the log-likelihood

$$\log\left(p_\theta(\sigma)\right) = \log\left(\int p_\theta(\sigma|z)p_\theta(z)dz\right)$$

To avoid the (often) difficult computation of the integral above, the idea behind the VAE is to use Bayes formula. Let us consider a volatility surface σ_i.

$$\begin{aligned}
\log(p_\theta(\sigma_i)) &= \int \log(p_\theta(\sigma_i))q_\phi(z|\sigma_i)dz \qquad \left(\text{because } \int q_\phi(z|\sigma_i)dz = 1\right) \\
&= \int \log\left(\frac{p_\theta(\sigma_i, z)}{p_\theta(z|\sigma_i)}\right) q_\phi(z|\sigma_i)dz \qquad \text{(Bayes formula)} \\
&= \int \log\left(\frac{p_\theta(\sigma_i, z)}{p_\theta(z|\sigma_i)}\frac{q_\phi(z|\sigma_i)}{q_\phi(z|\sigma_i)}\right) q_\phi(z|\sigma_i)dz \\
&= \underbrace{\int \log\left(\frac{q_\phi(z|\sigma_i)}{p_\theta(z|\sigma_i)}\right) q_\phi(z|\sigma_i)dz}_{=D_{KL}\left(q_\phi(z|\sigma_i)||p_\theta(z|\sigma_i)\right)} + \int \log\left(\frac{p_\theta(\sigma_i, z)}{q_\phi(z|\sigma_i)}\right) q_\phi(z|\sigma_i)dz \\
&= D_{KL}\left(q_\phi(z|\sigma_i)||p_\theta(z|\sigma_i)\right) + \int \log\left(\frac{p_\theta(\sigma_i, z)}{q_\phi(z|\sigma_i)}\right) q_\phi(z|\sigma_i)dz
\end{aligned}$$

Because of the intractability of $p_\theta(z|\sigma)$, we will avoid maximizing $\log(p_\theta(\sigma))$. Instead, we try to find a lower bound called Evidence Lower Bound (ELBO).

If we define $\text{ELBO}(\sigma_i) := \int \log\left(\frac{p_\theta(\sigma_i,z)}{q_\phi(z|\sigma_i)}\right) q_\phi(z|\sigma_i)dz$, as $D_{KL} \geq 0$, we have $\log(p_\theta(\sigma_i)) \geq \text{ELBO}(\sigma_i)$. But ELBO can be written as

$$\begin{aligned}\text{ELBO}(\sigma_i) &= \underbrace{\int \log\left(p_\theta(\sigma_i|z)\right) q(z|\sigma_i)dz}_{=-RE(\sigma_i)} - D_{KL}(q_\phi(z|\sigma_i)||p_\theta(z)) \\ &= -\text{RE}(\sigma_i) - D_{KL}(q_\phi(z|\sigma_i)||p_\theta(z))\end{aligned} \tag{7}$$

where RE stands for **reconstruction error**.

The goal of our CAE is to find the parameters θ and ϕ minimising the training loss such that

$$\theta, \phi = \arg\min_{\theta,\phi} -\frac{1}{n}\sum_{i=1}^{n} \text{ELBO}(\sigma_i)$$

2.3.3 The latent variable distribution

We assume that that z is a standard d-dimensional Gaussian variable with d is the latent space dimension. It is a hyper parameter that we set before training the network.

$$z \sim \mathcal{N}(0_d, I_d)$$

In practise, we assume that $q_\phi(z|\sigma) \sim \mathcal{N}(\mu(\sigma), \Gamma(\sigma))$ where $\mu(\sigma) \in \mathbb{R}^d$ and $\Gamma(\sigma) = \begin{pmatrix} \gamma_1^2(\sigma) & & 0 \\ & \ddots & \\ 0 & & \gamma_d^2(\sigma) \end{pmatrix} \in \mathbb{R}^{d\times d}$. The trick is to make the encoder infer $\mu(\sigma)$ and $L(\sigma) := \begin{pmatrix} \gamma_1(\sigma) & & 0 \\ & \ddots & \\ 0 & & \gamma_d(\sigma) \end{pmatrix}$. The goal is to learn ϕ so that $\mu(\sigma) = 0_d$ and $\Gamma(\sigma) = I_d$. A normal latent sample z_i is generated by simulating $\kappa \sim \mathcal{N}(0_d, I_d)$ and setting $z_i = \mu(\sigma_i) + L(\sigma_i)\kappa$

NB: The distribution of the latent variable does not matter a lot since the non-linear decoder can mimic arbitrarily complicated distribution of observations. But we choose normal distribution because sampling from such a distribution and calculating the KL divergence is well known.

2.3.4 Loss function

The loss is $-\frac{1}{n}\sum_{i=1}^{n} \text{ELBO}(\sigma_i)$. In the case of a normal latent variable:

- We approximate $\frac{1}{n}\sum_{i=1}^{n} RE(\sigma_i)$ by $\frac{1}{n}\sum_{i=1}^{n} ||\sigma_i - \widehat{\sigma_i}||_2^2$ where $\widehat{\sigma_i}$ is the output of the decoder based on the input σ_i.

- A simple calculation based on the normality of z and $q_\phi(z|\sigma)$ yields :

$$\text{KL} := \frac{1}{n}\sum_{i=1}^{n} D_{KL}(q_\phi(z|\sigma_i)||p_\theta(z)) = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{2}\sum_{p=1}^{d}\left(-1 - \log(\gamma_p^4(\sigma_i)) + \mu(\sigma_p)^2 + \gamma_p^4(\sigma_i)\right) \tag{8}$$

In practise, we introduce a hyperparameter β to add a regularisation that constrains the capacity of the latent information channel z.

The practical training loss is:

$$\frac{1}{n}\sum_{i=1}^{n}\left(||\sigma_i - \widehat{\sigma_i}||_2^2 + \frac{\beta}{2}\sum_{p=1}^{d}\left(-1 - \log(\gamma_p^4(\sigma_i)) + \mu(\sigma_p)^2 + \gamma_p^4(\sigma_i)\right)\right) \tag{9}$$

When $\beta = 0$, the VAE behaves like a DAE. The extra pressures coming from high β values, however, may create **a trade-off** between reconstruction fidelity and the quality of disentanglement within the learnt latent representations.

3 Implied volatility surfaces

We denote the strike, the maturity and the spot price by K, T and S_0 respectively. For an European put option, let $P_{mkt}(K,T)$ denotes the market price and $P_{BS}(K,T)$ the Black-Scholes price:

$$P_{BS}(K,T,S_0)(\sigma) = -S_0 e^{-qT}\mathcal{N}(-d_1(S_0,K,T)) + Ke^{-rT}\mathcal{N}(-d_2(S_0,K,T))$$

Similarly, we define $C_{mkt}(K,T)$ and $C_{BS}(K,T)$ for a European call option

$$C_{BS}(K,T,S_0)(\sigma) = S_0 e^{-qT}\mathcal{N}(d_1(S_0,K,T)) - Ke^{-rT}\mathcal{N}(d_2(S_0,K,T)) \tag{10}$$

with $\mathcal{N}$ the cumulative distribution function of the standard normal distribution, r the risk-free rate, q the dividend yield,

$$\begin{cases} d_1(S_0,K,T) := \frac{1}{\sigma\sqrt{T}}(\ln(S_0/K) + (r - q + \frac{\sigma^2}{2})T) \\ d_2(S_0,K,T) = d_1 - \sigma\sqrt{T} \end{cases} \tag{11}$$

The implied volatility $\sigma(K,T)$ is defined as the unique solution satisfying:

$$\begin{cases} P_{BS}(K,T,S_0)(\sigma) = P_{mkt}(K,T) \text{ if } S_0 \leq K \\ C_{BS}(K,T,S_0)(\sigma) = C_{mkt}(K,T) \text{ if } S_0 > K \end{cases}$$

When ($K >> S_0$) (out-of-the-money for a call and in-the-money for a put), the put price is significant while the call price is almost null (See figure 3) because traders holding the underlying asset buy **protective puts** to hedge their position against a **downside** change in the asset price. Thus the demand on put options is high which makes the price significant. And when ($K << S_0$), traders buy calls to protect their positions from a big increase in the asset price.

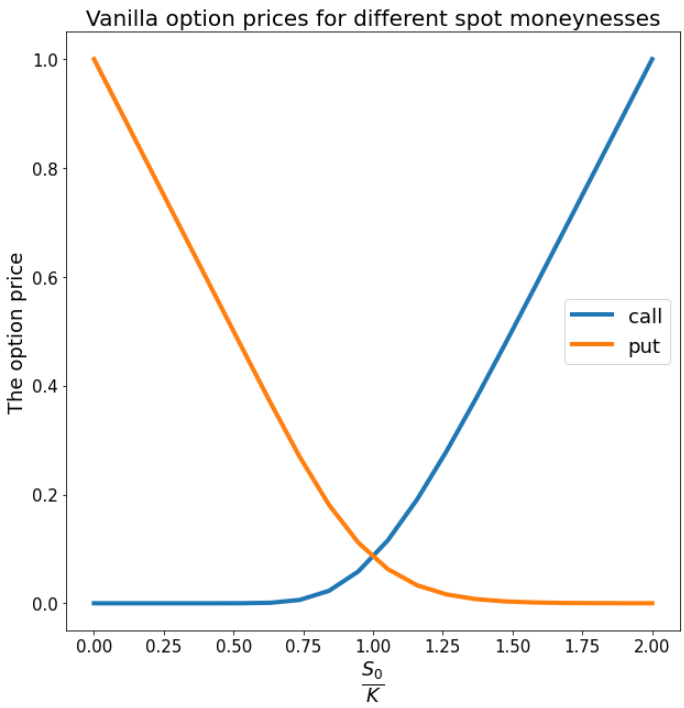


Figure 3: European call and put prices for different spot moneyness values

The liquidity of these vanilla options means that supply and demand indicate that their prices are the market's consensus of the correct price. This means that the volatility extracted from these options is in fact the market's consensus on the forward looking volatility of the asset. This implied volatility incorporates the forward views on all market participants on the asset's volatility.

3.1 Delta implied volatility surface

Delta is a greek introduced to describe the sensitivity of an option price to its underlying asset price. It represents the number of shares to be held at each time in order to perform a perfect (dynamic) hedging of the option according to the model of Black-Scholes.

Let F_T the forward price of an underlying and $x := \log\left(\frac{K}{F_T}\right)$ the corresponding log-forward moneyness. Let $\omega(x,T)$ be the total variance defined as:

$$\omega(x,T) = T\sigma^2(x,T)$$

d_1 and d_2 defined above can be rewritten as:

$$d_{1/2}(x,T) = -\frac{x}{\sqrt{\omega(x,T)}} \pm \frac{\sqrt{\omega(x,T)}}{2} \tag{12}$$

In the Black-Scholes model, the delta of a call and the delta of a put are defined as:

$$\Delta_{call}(x,T) := \mathcal{N}\left(d_1(x,T)\right) \qquad \Delta_{put}(x,T) := \mathcal{N}\left(d_1(x,T)\right) - 1$$

We chose to represent the implied volatility in terms of delta so the implied surface is $\tilde{\sigma}(\Delta, T) := \tilde{\sigma}(\Delta_{call}, T)$ instead of $\sigma(K, T)$ [1]. So the implied surface will be represented for a **fixed** range of deltas and maturities. Even if some values of this range are not necessarily listed in the market, we can use **interpolation** methods to find the corresponding volatility values (see section 4.1).

The reasons behind choosing delta implied surface instead of the usual implied surface (in terms of strike) are:

- Delta is a better indicator (compared to strike) about near the money options. It gives useful and necessary information about options. For short-term options (small maturities), the volatility far from the money is not useful because all the options are near the money as the asset does not have enough time to get far from the money. And interestingly, for short-term maturities, when $K \approx S_0$ we have $\Delta_{call} \in [0.1, 0.9]$ (See figure 4). So looking at $\sigma(K, T)$ is not useful especially for $K >> S_0$ and $K << S_0$ but instead it is more important to consider $\tilde{\sigma}(\Delta, T)$.

- Delta surface is smooth with regard to virtual maturities that are not listed. Time-interpolation at a fixed delta gives a smooth curve (see figure 5 and 6 in section 4.1).

- Delta surface is a multi-variable representation : As Δ_{call} (resp. Δ_{put}) is one-to-one map from log-forward moneyness space $\mathbb{R}$ in $[0, 1]$ (resp. $[-1, 0]$) when the surface is free of butterfly arbitrage (see 3.2.1), we can look at implied volatility as a function of delta or moneyness depending on the appropriate purpose.

- For hedging, one needs delta, not moneyness so it is a natural choice.

[1] In the absence of butterfly arbitrage, d_1 is strictly decreasing as a function of K (See 3.2.1). Then for every $\Delta \in [0, 1]$ we can find a corresponding K. Thus we can define $\tilde{\sigma}(\Delta_{call}, T)$.

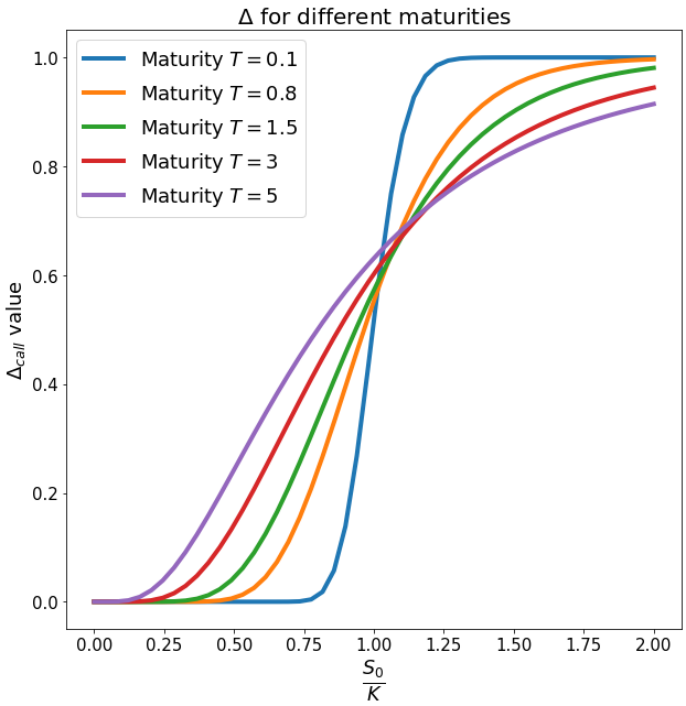


Figure 4: Δ_{call} for different maturities

3.2 Static arbitrage

Volatility surfaces come in many different shapes and levels, which are subject to change over time. However, they do not come in completely arbitrary shapes. Indeed, there are several restrictions on their geometry arising from the absence of (static) arbitrage, that is, the existence of a trading strategy providing instantaneous risk-free profit.

As discussed in 3.1, we consider delta implied volatility surfaces $\tilde{\sigma}(\Delta, T)$ instead of strike volatility surfaces $\sigma(K, T)$. Thus **it is more practical that we check and control static arbitrage in terms of Δ instead of the log-forward moneyness x or the strike K.**

In the following parts, we start by explaining the classical formulas of static arbitrage in terms of the log-forward moneyness x then we express the same conditions in terms of Δ.

3.2.1 Butterfly arbitrage

3.2.1.1 In terms of the log-forward moneyness x

A volatility surface is free of butterfly (strike) arbitrage if and only if each time slice is butterfly-arbitrage free :

$$\forall K > 0 \text{ and } \forall T > 0,\ \partial_K^2 C_{mkt}(K, T) \geq 0 \tag{13}$$

If we suppose $q = 0$ in the equation (10) above and we rewrite it in terms of x we obtain:

$$C_{BS}(x, T) = S_0 \left(\mathcal{N}\left(d_1(x, T)\right) - e^x \mathcal{N}\left(d_2(x, T)\right)\right)$$

Let us define

$$p(x,T) := \frac{\partial^2 C_{BS}}{\partial K^2}\left(\log\left(\frac{K}{F_T}\right), T\right)$$

then

$$p(x,T) = \frac{g(x,T)}{\sqrt{2\pi\omega(x,T)}}\exp\left(-\frac{d_2^2(x,T)}{2}\right)$$

with

$$g(x,T) = \left(1 - \frac{x\partial_x\omega(x,T)}{2\omega(x,T)}\right)^2 - \frac{\partial_x\omega(x,T)^2}{4}\left(\frac{1}{\omega(x,T)} + \frac{1}{4}\right) + \frac{\partial_x^2\omega(x,T)}{2}$$

(13) is equivalent to

$$\forall T > 0, \forall x \in \mathbb{R}, p(x,T) \geq 0 \tag{14}$$

In [15], it is argued that (14) holds if and only if

$$\forall T > 0, \forall x \in \mathbb{R},\ g(x,T) \geq 0 \text{ and } \lim_{x \to +\infty} d_1(x,T) = -\infty$$

In our work, **we will not use this characterization of butterfly arbitrage for two reasons**:

- it uses the second derivative of ω which can be **very slow and memory demanding** when using automatic differentiation (AD).
- it is a characterization in terms of x and not in terms of Δ.

Instead, we will use a characterization introduced by Fukasawa [13]. Let us define the **normalizing volatility transforms**:

$$f_1(x,T) = -d_1(x,T) \qquad f_2(x,T) = -d_2(x,T)$$

Fukasawa [13] showed that the cumulative distribution function (CDF) of $\log\left(\frac{S_T}{F_T}\right)$ is

$$\mu(x,T) = \mathcal{N}\left(f_2\left(x,T\right)\right) + \partial_x\sqrt{\omega}(x,T)n\left(f_2\left(x,T\right)\right)$$

where n is the probability distribution function (pdf) of a standard gaussian variable.

In [13], it is argued that a surface is free of butterfly arbitrage if and only if

$$\forall T > 0, x \mapsto \mu(x,T) \text{ is non-decreasing} \tag{15}$$

3.2.1.2 In terms of Δ

As we are working on $\tilde{\sigma}(\Delta,T)$, we will express the condition 15 in terms of Δ. Let $\tilde{\omega}(\Delta,T)$ be the total variance in terms of Δ defined as:

$$\tilde{\omega}(\Delta,T) = T\tilde{\sigma}^2(\Delta,T)$$

A surface is free of Butterfly arbitrage if and only if

$$\begin{cases} \forall T > 0,\ x \mapsto \Delta(x,T) \text{ is strictly decreasing} \\ \forall T > 0,\ \Delta \mapsto \tilde{\mu}(\Delta,T) \text{ is non-increasing} \end{cases} \tag{16}$$

where [2]

$$\begin{aligned} \tilde{\mu}(\Delta,T) &= \mu\left(d_1^{-1}(\mathcal{N}^{-1}(\Delta),T)\right) \\ &= \mathcal{N}\left(\sqrt{\tilde{\omega}(\Delta,T)} - \mathcal{N}^{-1}(\Delta)\right) - \frac{n\left(\sqrt{\tilde{\omega}(\Delta,T)} - \mathcal{N}^{-1}(\Delta)\right)\partial_\Delta\sqrt{\tilde{\omega}}(\Delta,T)}{\frac{\sqrt{\tilde{\omega}(\Delta,T)}}{n(\mathcal{N}^{-1}(\Delta))}\left(1 - \partial_\Delta\sqrt{\tilde{\omega}}(\Delta,T)n\left(\mathcal{N}^{-1}(\Delta)\right)\right) + \partial_\Delta\sqrt{\tilde{\omega}}(\Delta,T)\mathcal{N}^{-1}(\Delta)} \end{aligned}$$

NB: A proof of (16) can be found in Appendix 7.

Note that $\partial_\Delta\sqrt{\tilde{\omega}(\Delta,T)}$ can be **easily calculated using Automatic differentiation** because it only uses the first derivative of $\tilde{\omega}$ which is simple to calculate using automatic differentiation.

We do not need to express $\partial_\Delta\sqrt{\tilde{\omega}(\Delta,T)}$ in terms of $\partial_\Delta\tilde{\omega}(\Delta,T)$ because automatic differentiation creates the graph of mathematical operations applied on Δ and tracks all the operations involving Δ.

3.2.2 Calendar arbitrage

3.2.2.1 In terms of the log-forward moneyness x

A volatility surface is free of calendar arbitrage if and only if :

$$\forall x \in \mathbb{R} \text{ and } \forall T > 0, \partial_T w(x,T) \geq 0$$

which is equivalent to

$$\forall x \in \mathbb{R} \text{ and } \forall T > 0, \partial_T C_{mkt}(x,T) \geq 0 \text{ (or } \partial_T P_{mkt}(x,T) \geq 0) \tag{17}$$

(17) is an obvious necessary condition for no-arbitrage. Let us suppose that there exist $0 < T_1 < T_2,\ x \in \mathbb{R}$ for which $C_{mkt}(x,T_1) > C_{mkt}(x,T_2)$. Obviously, we can make a risk-free profit:

At $t = 0$: we sell $C_{mkt}(x,T_1)$ and buy $C_{mkt}(x,T_2)$. We make a profit $\epsilon = C_{mkt}(x,T_1) - C_{mkt}(x,T_2) > 0$

[2] d_1^{-1} is the reciprocal function in terms of x.

At $t = T_1$: The portfolio value is:

$$\epsilon - (S_{T_1} - K)^+$$

If $S_{T_1} > K$, we short-sell the asset and we receive K. Otherwise, the portfolio value is ϵ.

At $t = T_2$:

- If $S_{T1} \leq K$, the final profit is $\epsilon + (S_{T_2} - K)^+ > 0$.
- If $S_{T1} \geq K$, if $S_{T_2} \leq K$ we buy the stock by paying S_{T_2} and return the stock that we short sold at T_1 and the final value of the portfolio is $K - S_{T_2} + \epsilon > 0$. If $S_{T_2} \geq K$, we buy the stock by paying the amount K that we had received at T_1, and return the stock that we had short sold at T_1. The final profit is $\epsilon > 0$.

Without any risk, we are making profit which corresponds to an arbitrage strategy.

NB: Gatheral [15] showed that it is important to consider $\partial_T w(x,T) \geq 0$ for a fixed log-forward moneyness line instead of a fixed strike, namely when $r > 0$ or $q > 0$.

3.2.2.2 In terms of Δ

Similarly to butterfly arbitrage, we will express the absence of calendar arbitrage in terms of Δ instead of x or K.

In [22], Lucic justified that:

$$\frac{\partial\sqrt{\omega}}{\partial T}(x,T) = -\sqrt{\omega}(x,T)\frac{\partial d_1}{\partial x}(x,T)\frac{\partial \sigma_1}{\partial T}(\delta,T)\,,\ \delta = d_1(x,T)$$

where $\sigma_1(\delta,T) = \sqrt{\omega}(x_\delta,T)$ such that x_δ is the unique solution of $\mathcal{N}^{-1}(\Delta(x,T)) = \delta$ for a fixed T.

In [13], it is shown that $x \mapsto \mathcal{N}^{-1}(\Delta(x,T)) = d_1(x,T)$ is a non-increasing function in the absence of butterfly arbitrage so $\frac{\partial d_1}{\partial x}(x,T) \leq 0$. Thus, in absence of butterfly arbitrage

$$\frac{\partial\sqrt{\omega}}{\partial T}(x,T) \geq 0 \text{ is equivalent to } \frac{\partial\sqrt{\tilde{\omega}}}{\partial T}(\Delta,T) \geq 0 \tag{18}$$

Thus, in the absence of butterfly arbitrage, a surface is free of Calendar arbitrage if and only if

$$\text{If } 0 < T_1 < T_2 \text{ then } \tilde{\omega}(\Delta,T_1) \leq \tilde{\omega}(\Delta,T_2) \tag{19}$$

for $\Delta = \mathcal{N}\left(d_1(x_1,T_1)\right) = \mathcal{N}\left(d_1(x_2,T_2)\right)$ where x_1 and x_2 are chosen to have this equality for a given Δ.

4 Methodology

4.1 Data interpolation

We chose to work on volatility options on Apple, Microsoft, Google and Banco Bradesco SA stocks from 2011-2019 provided by ICE Data Services. The corresponding ICE tickers are respectively: APL.NQ,MSFT.NQ,GOOGL.NQ and BBD.NQ.

We mainly worked on two implied volatility grid sizes:

- 40 volatility points: formed of 5 maturities (two months, three months, six months, nine months, one year) and 8 deltas (0.1, 0.214, 0.328, 0.442, 0.557,0.671, 0.785, 0.9).
- 140 volatility points: formed of 7 maturities (one month, two months, three months, six months, nine months, one year, three years) and 20 deltas (0.1 , 0.142, 0.184, 0.226, 0.268, 0.311, 0.353, 0.395, 0.437, 0.479, 0.521, 0.563, 0.605, 0.647, 0.689, 0.732, 0.774,0.816,0.858, 0.9).

Maturities have to be numerical thus, we considered $\frac{T-t}{365}$ where $T-t$ is the number of days between the valuation date and the maturity date.

We avoided maturities shorter than one month because the interpolation that ICE Data Services used to fit their volatilities is not accurate on very short-term maturities. So it is noise rather that exploitable data.

ICE Data Services provides **mid implied volatility** (the avergage between the bid and the ask implied volatilities) **in terms of maturities and strikes** for the listed options on **the valuation date (requested date)**. For example, if we request options on 08/08/2022, ICE will provide all mid implied volatilities of the listed options on the day 08/08/2022. Thus, the number of available maturities and strikes may vary from day to day. This format of data is not exploitable for our VAE for two reasons:

- We want to represent the implied volatility in terms of Δ.
- We want a constant input size for our VAE which corresponds to a fixed range of deltas and maturities.

That is where we use **interpolation**.

Data interpolation steps:

For a given valuation date, let us denote the **ordered** set of listed strikes and maturities by $K_{listed} = \{K_i, i \in [1..\mathcal{K}_{listed}]\}$ and $T_{listed} = \{T_i, i \in [1..\mathcal{T}_{listed}]\}$ respectively. And let us consider the ordered target set of deltas and maturities respectively $\Delta_{target} = \{\Delta_i,\ i \in [1..\delta_{target}]\}$ and $T_{target} = \{T_i, i \in [1..\mathcal{T}_{target}]\}$. The interpolation for every valuation date works as follows:

1. For each maturity $T_k \in T_{listed}$, we calculate $\mathcal{X}_{listed} = \{x_i := \log\left(\frac{K_i}{F_{T_k}}\right),\ K_i \in K_{listed}\}$ then we check that $\{d_1(x_i, T_k), x_i \in \mathcal{X}_{listed}\}$ is **a decreasing sequence** in terms of x_i (This is a necessary condition for butterfly arbitrage-free slice).

2. For each maturity $T_k \in T_{listed}$, we fit a Cubic Spline $x \mapsto S_{T_k}(x)$ on $\{(x_1, \sigma(x_1, T_k)) \,, .. \,(x_i, \sigma(x_i, T_k)) \,.. \,(x_{\mathcal{K}_{listed}}, \sigma(x_{\mathcal{K}_{listed}}, T_k))\}$:

$$S_{T_k}(x) = \begin{cases} C_1^{(T_k)}(x),\ x_1 \leq x \leq x_2 \\ ... \\ C_i^{(T_k)}(x),\ x_{i-1} \leq x \leq x_i \\ ... \\ C_{\mathcal{K}_{listed}}^{(T_k)}(x),\ x_{\mathcal{K}_{listed}} - 1 \leq x \leq x_{\mathcal{K}_{listed}} \end{cases}$$

   where $C_i^{(T_K)}(x) = a_i^{(T_K)} + b_i^{(T_K)}x + c_i^{(T_K)}x^2 + d_i^{(T_K)}x^3,\ d_i^{(T_K)} \neq 0,\ S_{T_k} \in \mathcal{C}^2[x_1, x_{\mathcal{K}_{listed}}]$ and $\forall x_i \in \mathcal{X}_{listed},\ S_{T_k}(x_i) = \sigma(x_i, T_k)$

3. For each maturity $T_k \in T_{listed}$, we solve [3] $\mathcal{X}_{target} = \{x_i^{(T_k)},\ \Delta_i = \mathcal{N}\left(d_1\left(x_i^{(T_k)}, T_k\right)\right)$ for $\Delta_i \in \Delta_{target}\}$ using a Newton's algorithm. Then, we calculate the corresponding volatilities using the fitted cubic spline S_{T_k}. We obtain an interpolated volatility slice $\sigma_{target}^{(T_K)} = \{S_{T_K}(x_i^{(T_k)}),\ x_i^{(T_k)} \in \mathcal{X}_{target}\}$.

4. For each $\Delta_k \in \Delta_{target}$, we linearly interpolate the total variance on $\tilde{\omega}_{target}^{(\Delta_k)} = \{\tilde{\omega}(\Delta_k, T_i) := T_i S_{T_i}^2(x_k^{(T_i)}),\ T_i \in T_{listed}\}$. We define $T \mapsto S_{\Delta_k}(T)$ such that :

$$\mathrm{S}_{\Delta_k}(T) = \begin{cases} \frac{T_2 - T}{T_2 - T_1}\tilde{\omega}(\Delta_k, T_1) + \frac{T - T_1}{T_2 - T_1}\tilde{\omega}(\Delta_k, T_2)\ ,\ T_1 < T < T_2 \\ ... \\ \frac{T_{\mathcal{T}_{listed}} - T}{T_{\mathcal{T}_{listed}} - T_{\mathcal{T}_{listed}-1}}\tilde{\omega}(\Delta_k, T_{\mathcal{T}_{listed}-1}) + \frac{T - T_{\mathcal{T}_{listed}-1}}{T_{\mathcal{T}_{listed}} - T_{\mathcal{T}_{listed}-1}}\tilde{\omega}(\Delta_k, T_2)\ ,\ T_{\mathcal{T}_{listed}-1} < T < T_{\mathcal{T}_{listed}} \end{cases}$$

5. For each $\Delta_k \in \Delta_{target}$, we calculate $\{S_{\Delta_k}(T_i),\ T_i \in T_{target}\}$.

6. Thus, we find $\mathcal{Z}_{target} := \{\tilde{\omega}(\Delta_k, T_i),\ (\Delta_k, T_i) \in \Delta_{target} \times T_{target}\}$.

7. We consider $\sigma_{target} := \{\sqrt{\frac{\tilde{\omega}(\Delta_k, T_i)}{T_i}},\ \tilde{\omega}(\Delta_k, T_i) \in \mathcal{Z}_{target}\}$.

8. Finally, we normalize the data (discussed in 5.2.5).

NB: In [22], it is argued that this scheme of interpolation in time does not introduce calendar arbitrage in step (4.) if the initial volatility term structure $\{\sigma_{target}^{(T_K)},\ T_k \in T_{listed}\}$ is free of calendar arbitrage.

We use the available data from 2011-2017 as **a training set** and the rest as **a testing set**.

[3] In step 1, we checked that d_1 is strictly decreasing. Thus $x \mapsto \Delta(x, T_k) = \mathcal{N}(d_1(x, T_k))$ is a one-to-one map, then for every $\Delta \in \Delta_{target}$, we can find x such that $\Delta = \mathcal{N}(d_1(x, T_k))$.

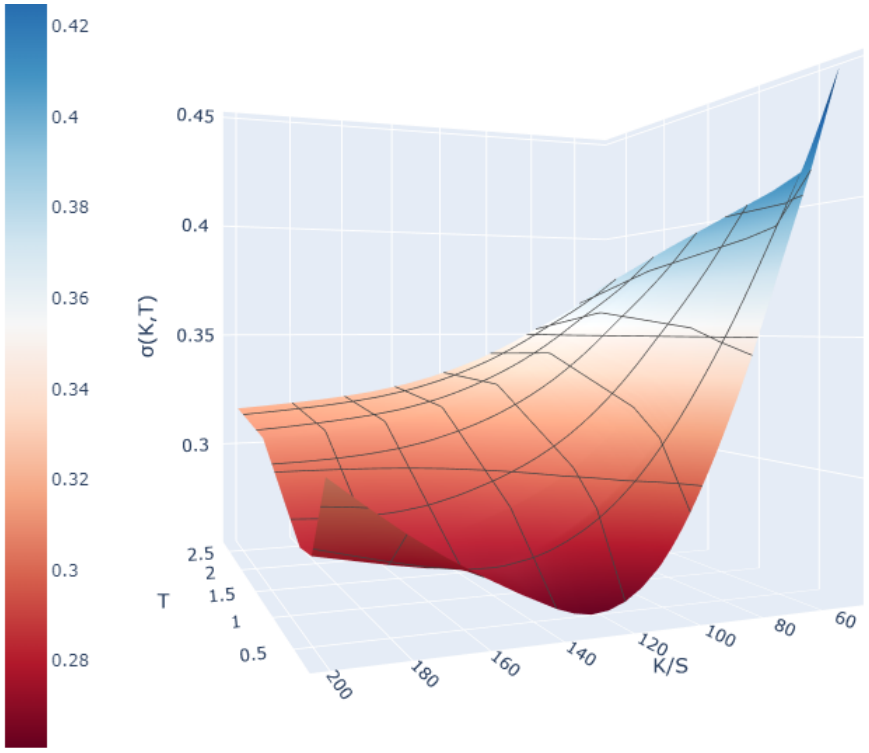


Figure 5: Strike implied volatility

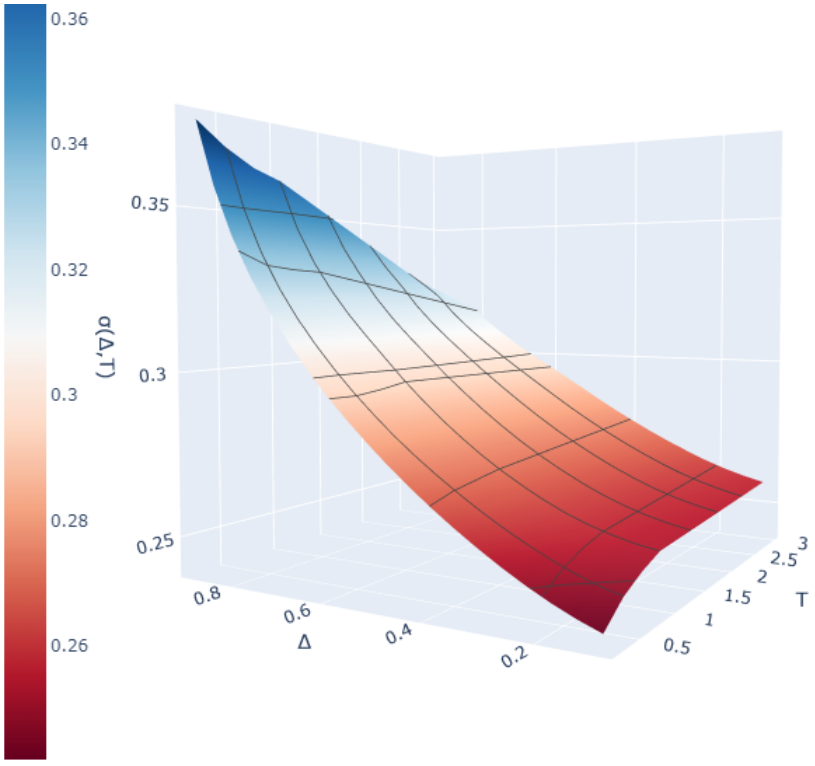


Figure 6: Delta implied volatility

4.2 Feed-forward architecture used in [3]

In [3], the authors opted for a VAE with Feed-Forward layers in both approaches: the **pointwise** approach and the **grid-based** one. In both approaches, the input to the encoder is a volatility surface, sampled at N grid points, which is then flattened into a vector, as shown in 7(a). 7(b) illustrates the grid-based approach, which follows the same architecture as traditional VAEs, where the decoder uses a d-dimensional latent variable to reconstruct the original grid points. Finally, the pointwise approach, as shown in 7(c) is an alternative architecture where the option parameters (moneyness and maturity) are defined explicitly. Concretely, the input for the pointwise decoder is a single option's parameters and the latent variable for the entire surface, and the output is a single point on the volatility surface.

Their network is composed of two 32-neuron fully connected layers in the encoder and two 32-neuron fully connected layers in the decoder. They chose a variable latent space dimension d in their experiments $d \in \{2, 3, 4\}$.

4.3 Our architecture

4.3.1 Convolutional layers

A convolution of two functions f and g is defined as:

$$(f * g)(t) := \int_{\mathbb{R}} f(\tau)g(t - \tau)\ d\tau$$

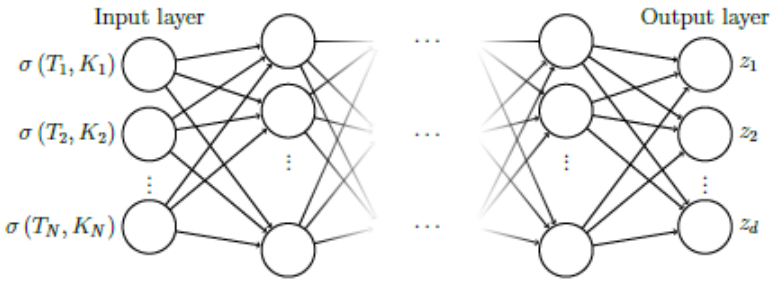
A convolutional neural network performs a moving discrete convolution of **the convolutional filter (kernel)** and **the input given to the network**. In our case, the input is the delta implied surface $\{\tilde{\sigma}(\Delta_i, T_k),\ (\Delta_i, T_k) \in \Delta_{target} \times T_{target}\}$ where Δ_{target} and T_{target} were defined in 4.1. Then the kernel slides a number s (called **stride**) of input entries (pixels for example) and apply a convolution. This process ends when the kernel achieves the bottom right corner of the input (see figure 8).

Sometimes, one may need to include **a padding** p either in the horizontal direction or the vertical direction to control the output shape of the layer. A padding is an artificial extension of the input by adding zeros beyond the boarder of the input.

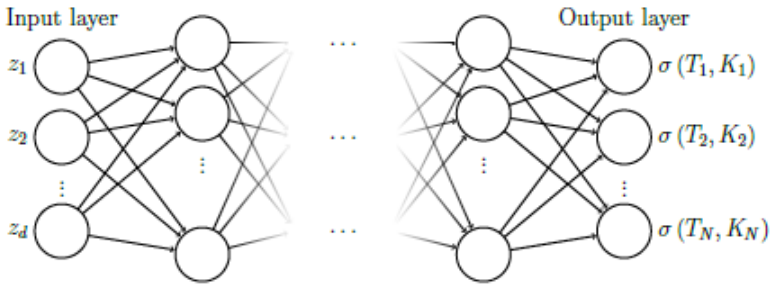
Example: Let us consider a convolutional layer with one neuron that we apply to our delta implied volatility surface $\{\tilde{\sigma}(\Delta_i, T_k),\ (\Delta_i, T_k) \in \Delta_{target} \times T_{target}\}$. Let us rewrite the delta implied volatility surface as a matrix

$$I := \begin{pmatrix} \tilde{\sigma}(\Delta_1, T_1) & \tilde{\sigma}(\Delta_2, T_1) & ... & \tilde{\sigma}(\Delta_{\delta_{target}}, T_1) \\ & \ddots & & \\ \tilde{\sigma}(\Delta_1, T_{\mathcal{T}_{target}}) & \tilde{\sigma}(\Delta_2, T_{\mathcal{T}_{target}}) & ... & \tilde{\sigma}(\Delta_{\delta_{target}}, T_{\mathcal{T}_{target}}) \end{pmatrix}$$

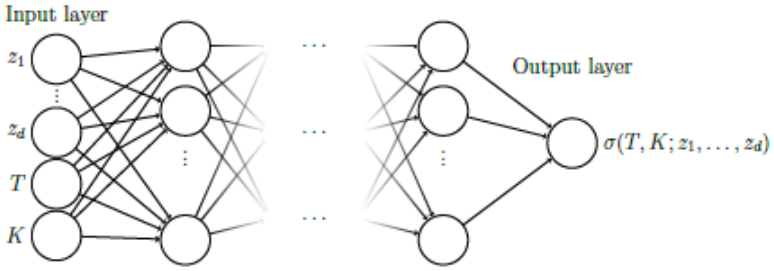
Let us suppose that the neuron is a (2×1) kernel $k := \begin{pmatrix} a \\ b \end{pmatrix}$. If the paddings are null and the vertical and horizontal strides are equal to 1 , the output of the convolutional layer is a



(a) The encoder architecture



(b) The decoder architecture for the grid-based training approach.



(c) The decoder architecture for the pointwise training approach.

Figure 7: An illustration of the grid-based and pointwise feed-forward architectures used in [3].

matrix called **feature map**:

$$\begin{pmatrix} a\tilde{\sigma}(\Delta_1, T_1) + b\tilde{\sigma}(\Delta_1, T_2) & a\tilde{\sigma}(\Delta_2, T_1) + b\tilde{\sigma}(\Delta_2, T_2) & .. & a\tilde{\sigma}(\Delta_{\delta_{target}}, T_1) + b\tilde{\sigma}(\Delta_{\delta_{target}}, T_2) \\ a\tilde{\sigma}(\Delta_1, T_2) + b\tilde{\sigma}(\Delta_1, T_3) & a\tilde{\sigma}(\Delta_2, T_2) + b\tilde{\sigma}(\Delta_2, T_3) & .. & a\tilde{\sigma}(\Delta_{\delta_{target}}, T_2) + b\tilde{\sigma}(\Delta_{\delta_{target}}, T_3) \\ & \vdots & & \\ a\tilde{\sigma}(\Delta_1, T_{\mathcal{T}_{target}-1}) + b\tilde{\sigma}(\Delta_1, T_{\mathcal{T}_{target}}) & & .. & a\tilde{\sigma}\left(\Delta_{\delta_{target}}, T_{\mathcal{T}_{target}-1}\right) \\ & & & + \\ & & & b\tilde{\sigma}\left(\Delta_{\delta_{target}}, T_{\mathcal{T}_{target}}\right) \end{pmatrix}$$

The output of a convolutional layer has **as many channels as** the number of filters (kernels). If we choose **the output channel** to be p then the convolutional layer will apply p filters to the input (See figure 8).

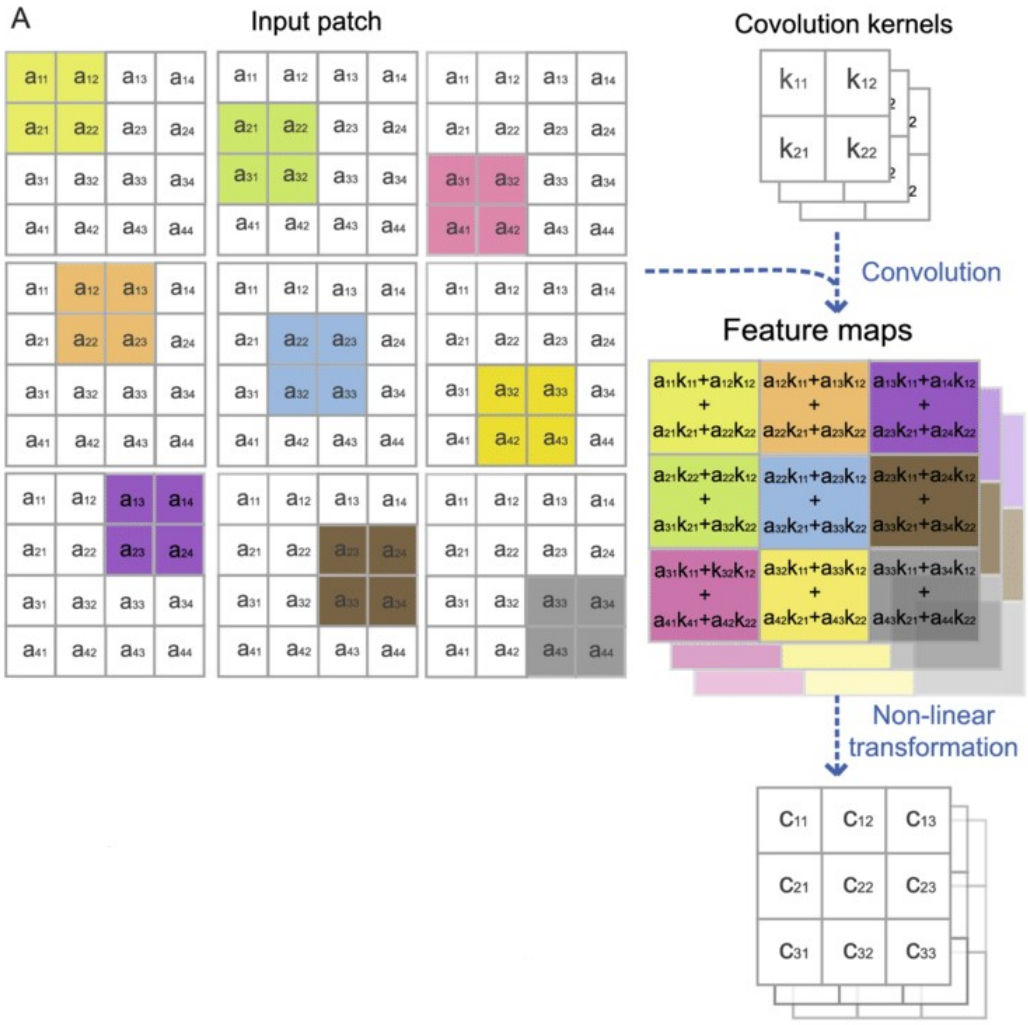


Figure 8: An illustration of a convolutional layer

The main advantages of convolutional layers compared to fully connected layers are:

- Convolutional layers are **spatially invariant**. If a convolution is specialized in a certain task, for example in vertical features detection, no matter the location of the vertical feature in the input the convolution will detect it. This is very useful if we want our neural network to learn **Calendar arbitrage free surfaces**. **By using convolutional layers, the network is more likely to learn that the term structure of the total variance has to be increasing for a fixed Δ.** (See 3.2.2)

- Convolutional layers use less parameters than fully connected layers because they allow **the share of parameters**. In fact, if we flatten the implied surface and we link it to a fully connected layer, we will use $\mathcal{T}_{target} \times \delta_{target}$ parameters per neuron, however for the convolutional neural network used in the example above, we only used 2 parameters per neuron.

- The more we get deeper in the network, the more **specialized** convolutional layers become [21].

4.3.2 Hybrid architecture

For a Grid-based approach, we tested the architecture of a feed-forward VAE used by Bergeron et al. [3] discussed in 4.2.

For a pointwise approach, we opted for **a hybrid architecture formed of a convolutional encoder and a feed-forward (fully connected) decoder**. A fully convolutional VAE is possible in a pointwise approach but it is not the most appropriate approach because the output of the autoencoder has to be a single point of the implied volatility surface and not the whole surface.

Let us denote the horizontal and the vertical output shape of a feature map (output of a convolutional layer) by O_h and O_v respectively. Let s_h, s_v, p_h, p_v, k_h and k_v be respectively the horizontal stride, the vertical stride, the horizontal padding, the vertical padding, the horizontal kernel size and the vertical kernel size. Let I_h and I_v be the horizontal and the vertical input shape of the convolutional layer.

It is easy to see that

$$O_h = \left\lfloor 1 + \frac{I_h + 2p_h - k_h}{s_h} \right\rfloor \tag{20}$$

$$O_v = \left\lfloor 1 + \frac{I_v + 2p_v - k_v}{s_v} \right\rfloor \tag{21}$$

In our experiments, we chose to set $s_h = s_v = p_h = p_v = 1$. Strides of 1 lead to large overlaps which can be beneficial for feature extraction from the volatility surface. For each convolutional layer, we chose the output shape of the feature maps to be half of the input shape i.e $O_h = \left\lfloor \frac{I_h}{2} \right\rfloor$ and $O_v = \left\lfloor \frac{I_v}{2} \right\rfloor$. For example, the output of the **first** convolution of the encoder has a feature map shape

$$O_h = \left\lfloor \frac{\mathcal{T}_{target}}{2} \right\rfloor$$

$$O_v = \left\lfloor \frac{\delta_{target}}{2} \right\rfloor$$

where $\mathcal{T}_{target}$ and δ_{target} are the number of target maturities and target deltas respectively (defined in 4.1).

As $s_h = 1, s_v = 1, p_h = 1, p_v = 1, O_h = \left\lfloor \frac{I_h}{2} \right\rfloor$ and $O_v = \left\lfloor \frac{I_v}{2} \right\rfloor$, the degree of freedoms left in (20) and (21) are k_h and k_v respectively. Thus, from (20) and (21) we have

$$k_h = 3 + \left\lceil \frac{I_h}{2} \right\rceil$$

$$k_v = 3 + \left\lceil \frac{I_v}{2} \right\rceil$$

Thus, if we set the kernel size of the filter $k_h = 3 + \left\lceil \frac{I_h}{2} \right\rceil$ and $k_v = 3 + \left\lceil \frac{I_v}{2} \right\rceil$, we ensure that the feature map produced by the convolutional layer has a shape $O_h = \left\lfloor \frac{I_h}{2} \right\rfloor$ and $O_v = \left\lfloor \frac{I_v}{2} \right\rfloor$.

We left the number of convolutional layers, the number of filters (kernels) per layer, the activation functions, the latent space dimension and the number of conditional variables afforded to the decoder (see the definition of a condition variational autoencoder (CAE) in 2.3.1) to be ***variable*** *in our experiments.*

5 Numerical results

The procedure was to test our architecture for **different hyperparameters**: the number of convolutional layers, the number of filters (kernels) per layer, the activation functions, the latent space dimension, the number of conditional variables afforded to the decoder, the number of asset stocks used during the training, the loss function (9)(See 5.2.4) and the normalization method of the data.

When the question of finding the best hyperparameters was present, we opted for **the cross validation (CV)** scheme to determine the hyperparameters that minimize the loss function :

1. For every combination of parameters, we randomly partition the training set into k equal sized subsamples. Of the k subsamples, a single subsample is retained as the validation data for testing the model, and the remaining $k-1$ subsamples are used as training data. We train the model on the training data and we test the performance on the testing data.

2. The cross-validation process is then repeated k times, with each of the k subsamples used exactly once as the validation data. The k results can then be averaged to produce a single estimation.

The advantage of this method over repeated random sub-sampling is that all observations are used for both training and validation, and each observation is used for validation exactly once.

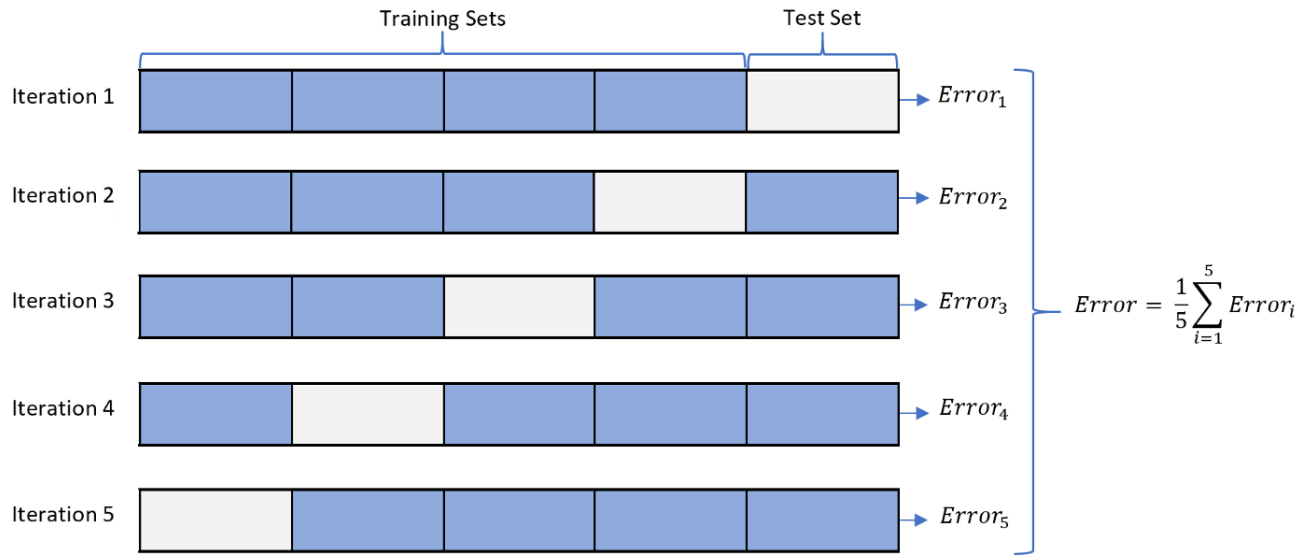


Figure 9: An illustration of a k-fold cross validation scheme with $k = 5$

The metric used to evaluate the performance of the model is **the mean absolute error (MAE)**. Let $S := \{\sigma_k\}_{k\in[1..|S|]}$ be a set of ground truth volatility surfaces (training or testing set) and $S' := \{\widehat{\sigma_k}\}_{k\in[1..|S|]}$ the set of predicted volatility surfaces on the basis of S. We denote the single volatility points constituting the surfaces σ_k and $\widehat{\sigma_k}$ by $\sigma_k^{(i)}$ and $\widehat{\sigma_k}^{(i)}$ respectively where $i \in [1..|A|]$ and $|A|$ is the number of points per volatility surface.

$$\text{MAE} := \frac{1}{|A||S|}\sum_{k=1}^{|S|}\sum_{i=1}^{|A|}|\sigma_k^{(i)} - \widehat{\sigma}_k^{(i)}| \tag{22}$$

All the MAEs will be presented in **basis points (bips)**, MAE in bips is MAE $\times 10^{-4}$.

We used **the KL-divergence** D_{kl} (8) to measure the distance between the distribution of the latent variable and its prior distribution.

Training tricks: During the training phase, the number of training epochs may vary. We chose to stop the training when there is no improvement in the training loss for 10 consecutive epochs. When there is no improvement in the training loss for 5 consecutive epochs, we divide the **learning rate** by 10.

Environment: We used the version 1.12.1 + CUDA 11.3 of Pytorch. All the trainings were achieved on a Tesla gpu $K80$ available on Google Colab.

5.1 Strike implied volatility surface

One of the first experiments we conducted was to train the pointwise feed-forward architecture (See 4.2) on a set of interpolated implied volatility surfaces $\sigma(K, T)$ (See 4.1) in which each volatility point corresponds to a maturity and **a strike**. We compare the feed-forward

VAE to a random model which generates uniform random numbers between the minimal volatility value and the maximal volatility value found in the data set.

The best hyperparameters combination found by the cross validation (**CV**) scheme yields the following results:

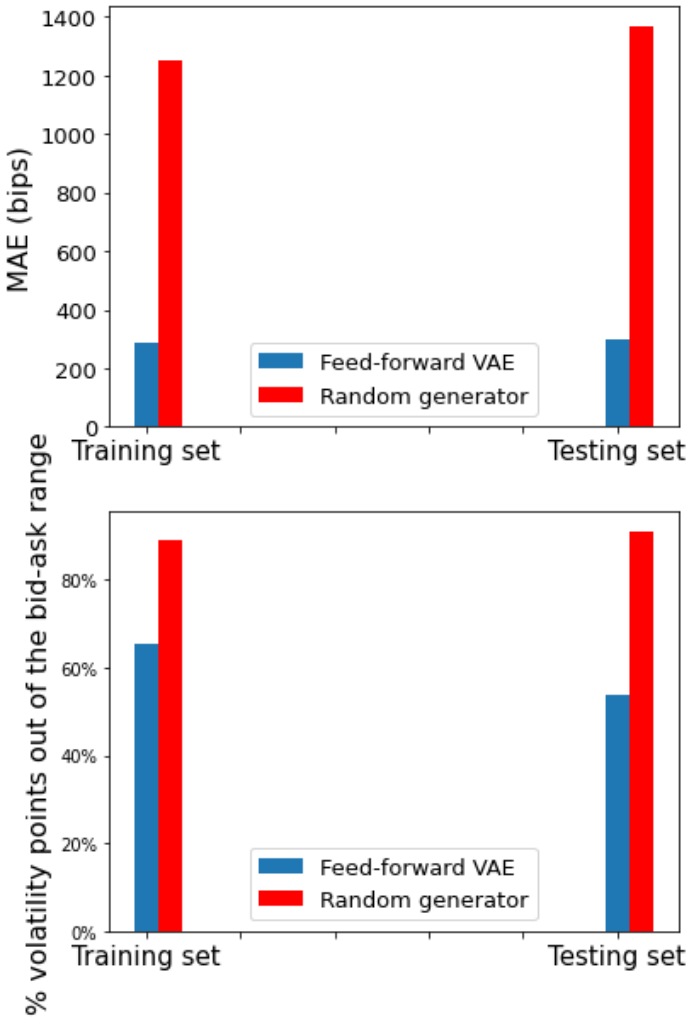


Figure 10: Random generator vs best VAE trained on strike implied volatility surfaces (underlying: Apple stock)

In figure 10, we check that (even if we know that the training loss is decreasing) the trained VAE beats the random generator to ensure that the training was effective. The best VAE (chosen by cross validation) yields a MAE of 301.6 bips which corresponds to **an average relative error of** 11%. Besides, we see that almost all the predicted volatility points **do not fit in the bid-ask range.**

As discussed in 3.1, strike implied surfaces are less smooth than delta implied volatility surfaces. So, a VAE trained on delta implied surface has to yield better results.

5.2 Delta implied volatility surfaces

In the following section, unless otherwise stated, the MAE is calculated on **the test set**, and the data used is the interpolated delta implied volatility surface (See 4.1).

5.2.1 A preliminary experiment

A preliminary experiment was to compare our hybrid architecture to the feed-forward architecture used in [3] discussed in 4.2 for the same hyperparameters.

With $\beta = 10^{-5}$ in the training loss (9), we have the following results:

Model	Properties	number of layers	number of neurons (filters for a convolution)	Activations	latent space dimension	MAE (bips)	Train set	Test set
Feed forward VAE used in [3]	Encoder	3	64 neurons 64 neurons 2×4 neurons	RELU RELU ∅	4		113.2	120.5
	Decoder	3	64 neurons 64 neurons 1 neuron	RELU RELU SOFTPLUS				
Hybrid VAE	Encoder (convolution)	3	32 filters 64 filters Flattening 2×4 neurons	RELU RELU ∅	4		90.3	95.4
	Decoder	3	64 neurons 64 neurons 1 neuron	RELU RELU SOFTPLUS				

Table 1: Feed-forward VAE vs hybrid VAE (underlying: Apple stock)

In table 1, the (2×4) neuron layers correspond to the two layers that predict the mean and the variance of the latent variable (See 2.3.3).

From this first experience, we notice that our hybrid architecture outperforms the feed-forward but even with a MAE of 95.4 bips, the average relative error 3.6% is still high.

NB: We tested different numbers of layers and number of neurons (or filters) in both architectures. We found that this makes slight differences. In all the following sections we will stick to the following architectures:

Model	Properties	number of layers	number of neurons (filters for a convolution)	Activations
Feed forward VAE	Encoder	3	64 neurons 64 neurons $2\times d$ neurons	RELU RELU $\emptyset$
	Decoder	3	64 neurons 64 neurons 1 neuron	RELU RELU SOFTPLUS
Hybrid VAE	Encoder (convolution)	4	8 filters 32 filters 64 filters Flattening $2\times d$ neurons	RELU RELU RELU $\emptyset$
	Decoder	3	64 neurons 64 neurons 1 neuron	RELU RELU SOFTPLUS

Table 2: Architectures used in the following sections (d is the latent space dimension)

5.2.2 Latent space effect

For both architectures defined in table 2, we modify the latent space dimension and we see the impact on the MAE and the D_{KL} (See (8)). For both models and for every latent space dimension, we find the optimal β in (9) and we test the model on the testing set. The results are summarized in the following table:

Metric	Model	Latent space dimension				
		4	6	8	10	14
MAE (bips)	Feed-forward VAE	**44**	46	36	**30**	33
	Hybrid VAE	**44**	**39**	**28**	31	**32**
D_{kl}	Feed-forward VAE	**13**	19	**22**	**28**	30
	Hybrid VAE	17	**18**	24	**28**	**27**

Table 3: Latent space effect on MAE and D_{kl} (underlying: Apple stock). We underlined the minimal number in each metric. All the numbers are rounded to the nearest for readability.

A D_{kl} close to 0 means that the distribution of the latent space is very close of a standard gaussian distribution. We want a trade-off between minimizing D_{kl} and MAE.

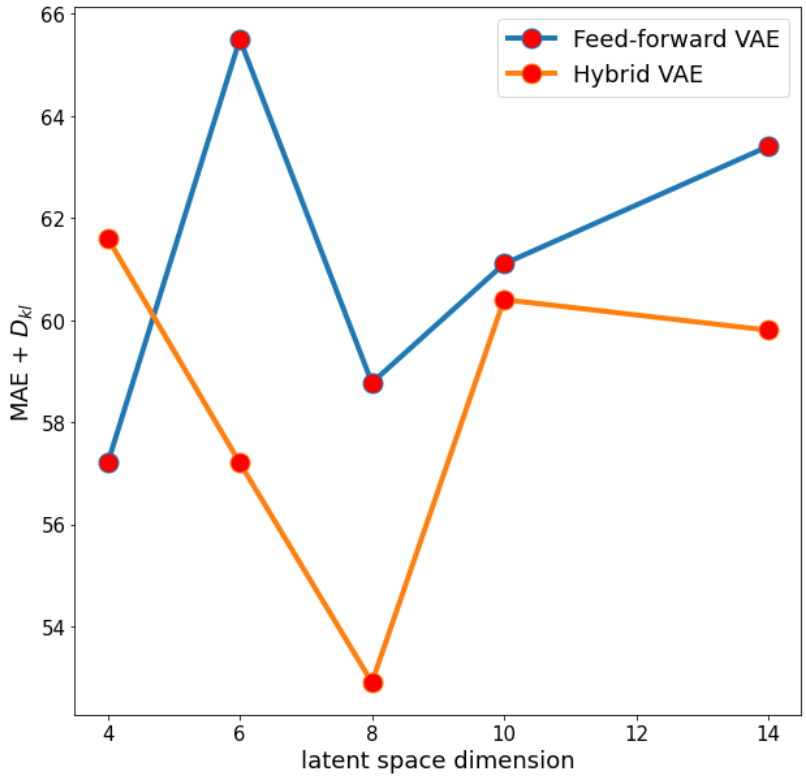


Figure 11: MAE + D_{kl} for the hybrid VAE and the Feed-forward VAE

From the figure 11, we see that the hybrid VAE reaches the best trade-off between D_{kl} and MAE for a latent space dimension $d = 8$. A MAE of 28 corresponds to an average relative error of 1%.

From this experience, we conclude that the performance of both models does not necessarily improve when the dimension of the latent space increases. When the latent space has a higher dimension, it is harder for the VAE to learn an appropriate representation of the prior distribution because of the high correlation between the variables that form the latent space vector, however, when the dimension is very small the latent space cannot encode all the information about the implied volatility. Thus, the latent space dimension has to be moderate.

NB: In the following sections, we will use a latent space dimension $d = 8$ for both architectures.

5.2.3 Cyclic β

During our experiments, we noticed that when $\beta \to 0$, The VAE behaves like a deterministic AE (See 2.1), the model is pushed to only learn the identity function regardless of the latent space distribution. When $\beta > 10^{-4}$, we noticed that the model minimizes well the D_{kl}, however it looses precision in the input reconstruction.

One of the methods to deal with this problem is to consider a β coefficient which is cyclic.

For example, during the 20 first training epochs, we set $\beta := 0$ so the model learns how to reconstruct the input. Then, we gradually increase β until $\beta = \gamma$ is reached (where γ is a constant). In this setting, we do not optimize the proper lower bound in (9) during the early stages of training, but nonetheless improvements on the value of that bound are observed at convergence in previous work [5].

In [12], the authors considered β of the following form:

$$\beta_t = \begin{cases} \zeta(\tau), \ \tau \leq R \\ \gamma, \ \tau > R \end{cases}$$

where $\tau = \frac{(t-1) - \lfloor \frac{(t-1)}{T/M} \rfloor \frac{T}{M}}{T/M}$, t is the iteration number, γ is a constant, T is the total number of training iterations, ζ is a monotonically increasing function, M is the number of cycles and R is the proportion used to increase β within a cycle.

The intuition is that in every cycle, the model learns to minimize the reconstruction error RE and D_{kl} by alternating the learning phases. For example, during a cycle, the model starts by learning to reconstruct the input (minimize the reconstruction error). Then, in the complementary phase of the same cycle the model learns to minimize D_{kl}, hopefully, without losing all the learning made in the previous phase. And from phase to phase, the goal is that the sum $RE + D_{kl}$ decreases more than in case of a constant β.

We opted for two cyclical β schemes:

- Linear cyclical schedule presented in figure 13.
- ON-OFF cyclical schedule presented in figure 12.

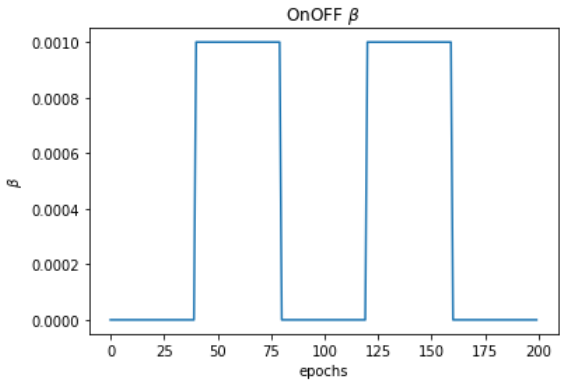


Figure 12: ON-OFF cyclical schedule $R = 0.5$

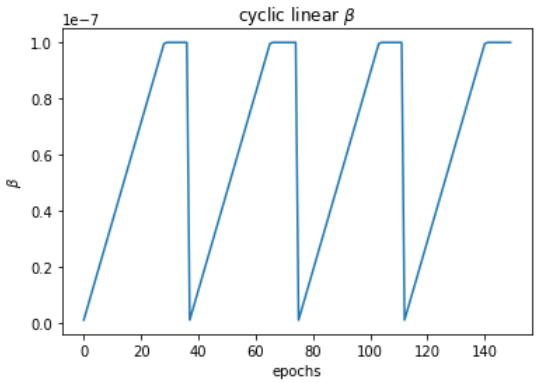


Figure 13: Linear cyclical scedule $R = 0.75$

For the hybrid VAE the KL divergence of the training loss is shown in figure 14 for two cyclical β schemes and for a constant β. When β is cyclically linear, the KL divergence has a decreasing trend with cyclical peaks that decrease from cycle to cycle. The peaks coincide

with the phases where β is minimal. The cyclical parts where the KL is almost flat coincide with the phases where β is maximal. The fact that the peaks keep decreasing from cycle to cycle prove that the network does not lose the information that it learnt during the previous epochs.

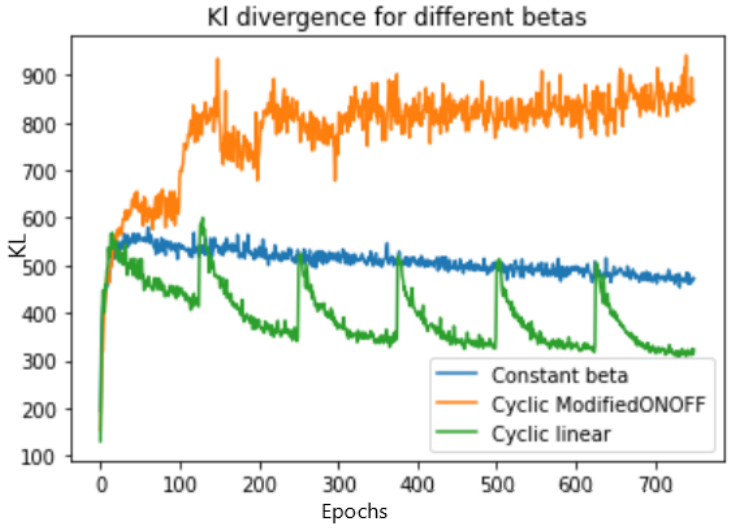


Figure 14: The KL loss D_{kl} of the hybrid VAE for different cyclical β schedules (The D_{kl} presented is the sum of all the D_{kl} of 16-surface batches)

Running the hybrid model and the feed-forward model with the two cyclic schemes on the testing set yields the following results:

Metric	Model	ON-OFF cyclic β	Linear cyclic β
MAE (bips)	Feed-forward VAE	35	30
	Hybrid VAE	**28**	**25**
D_{kl}	Feed-forward VAE	**23**	**25**
	Hybrid VAE	27	26

Table 4: MAE and D_{kl} for both architectures when the cyclical β schedule is used (underlying: Apple stock). We underlined the minimal number in each metric. All the numbers are rounded to the nearest for readability.

From table 4, we notice that the linear cyclic β yields a better trade-off between MAE and D_{kl} than ON-OFF cyclic β. Once again the hybrid architecture beats the feed-forward VAE. We notice that for both architectures, the ON-OFF cyclic β did not decrease the sum MAE + D_{kl} comparing to the results displayed in the table 3 (latent space dimension $d = 8$).

As of this subsection, **we will use the linear cyclic β for both architectures.**

5.2.4 Modified loss

One of the attempts to improve the results was to introduce a range of different target **encoding capacities** in (9) by pressuring the KL divergence to be at a controllable value C [6].

For a training epoch t, (9) becomes:

$$\text{Loss}_t := \frac{1}{n}\sum_{i=1}^{n} ||\sigma_i - \widehat{\sigma_i}||_2^2 + \beta \left| \frac{1}{n}\sum_{i=1}^{n} \left(\frac{1}{2}\sum_{p=1}^{d} \left(-1 - \log(\gamma_p^4(\sigma_i)) + \mu(\sigma_p)^2 + \gamma_p^4(\sigma_i) \right) \right) - C_t \right| \tag{23}$$

where $C_t = \frac{tC_{end}}{T}$ and T is the total number of training epochs and C_{end} is a constant determined by Cross-validation (CV) (See 5). C_t is increased uniformly and linearly during the training phase.

The intuition behind this idea is that during the early epochs of training, the model learns the prior distribution of the latent variable. As the training goes along, the capacity C_t increases and the term $\left| \frac{1}{n}\sum_{i=1}^{n} \left(\frac{1}{2}\sum_{p=1}^{d} \left(-1 - \log(\gamma_p^4(\sigma_i)) + \mu(\sigma_p)^2 + \gamma_p^4(\sigma_i) \right) \right) - C_t \right| \to 0$, the model learns to reconstruct the input. We hope that reconstructing the input is done without losing the information learnt about the latent variable.

This trick yields the following results on the testing set:

Metric	Model	Using loss (9)	Using modified loss (23)
MAE (bips)	Feed-forward VAE	30	43
	Hybrid VAE	**<u>25</u>**	**<u>30</u>**
D_{kl}	Feed-forward VAE	**<u>25</u>**	**<u>21</u>**
	Hybrid VAE	26	33

Table 5: MAE and D_{kl} with and without the capacity term (23) (underlying: Apple stock). We underlined the minimal number in each metric. All the numbers are rounded to the nearest for readability.

Clearly, the modified loss did not improve comparing to the previous section 5.2.3. Such a result is expected because the cyclic β is already doing the same goal as the capacity term introduced in this section. That is why when we use the modified loss (23) and we remove the cyclical β schedule (so we choose β by cross-validation), we roughly obtain the previous section results. This is shown in the following table 6.

Metric	Model	Loss (9) and a linear cyclic β	Modified loss (23) and a constant β
MAE (bips)	Feed-forward VAE	30	32
	Hybrid VAE	**<u>25</u>**	**<u>24</u>**
D_{kl}	Feed-forward VAE	**<u>25</u>**	**<u>22</u>**
	Hybrid VAE	26	28

Table 6: without the capacity term (9) using a cyclic β vs with the capacity term (23) and a constant β (underlying: Apple stock). We underlined the minimal number in each metric. All the numbers are rounded to the nearest for readability.

5.2.5 Data standardization

Let us denote the training set respectively by $S_{train} = \{\tilde{\sigma_i}^{train}(\Delta_k, T_p),\ (i,k,p) \in\ (|S_{train}|, \delta_{target}, \mathcal{T}_{target})\}$ where $|S_{train}|$ is the number of training implied volatility surfaces and $\delta_{target}, \mathcal{T}_{target}$ are defined in 4.1. Similarly, we denote the testing set $S_{test} = \{\tilde{\sigma_i}^{test}(\Delta_k, T_p),\ (i,k,p) \in (|S_{test}|, \delta_{target}, \mathcal{T}_{target})\}$.

Apart from standardizing the training set, we standardize the testing set with the statistics of the training set. It is important to notice that we standardize using the training set statistics only. Otherwise, we will biasing the training phase because we included the testing set statistics.

We tested 3 ways to normalize the input data:

- For a fixed Δ_k, we standardize as follows

$$\tilde{\sigma_i}^{train\ normalized}(\Delta_k, T_p) := \frac{\tilde{\sigma_i}^{train}(\Delta_k, T_p) - \frac{1}{\mathcal{T}_{target}|S_{train}|}\sum_{p=1}^{\mathcal{T}_{target}}\sum_{l=1}^{|S_{train}|}\tilde{\sigma_l}(\Delta_k, T_p)}{std_{\Delta_k}}$$

$$\tilde{\sigma_i}^{test\ normalized}(\Delta_k, T_p) := \frac{\tilde{\sigma_i}^{test}(\Delta_k, T_p) - \frac{1}{\mathcal{T}_{target}|S_{train}|}\sum_{p=1}^{\mathcal{T}_{target}}\sum_{l=1}^{|S_{train}|}\tilde{\sigma_l}(\Delta_k, T_p)}{std_{\Delta_k}}$$

  where std_{Δ_k} is the standard deviation calculated over all the maturities and all the training samples for the fixed Δ_k.

- For a fixed maturity T_p, we standardize as follows

$$\tilde{\sigma_i}^{train\ normalized}(\Delta_k, T_p) := \frac{\tilde{\sigma_i}^{train}(\Delta_k, T_p) - \frac{1}{\delta_{target}|S_{train}|}\sum_{k=1}^{\delta_{target}}\sum_{l=1}^{|S_{train}|}\tilde{\sigma_l}(\Delta_k, T_p)}{std_{T_p}}$$

$$\tilde{\sigma_i}^{test\ normalized}(\Delta_k, T_p) := \frac{\tilde{\sigma_i}^{test}(\Delta_k, T_p) - \frac{1}{\delta_{target}|S_{train}|}\sum_{k=1}^{\delta_{target}}\sum_{l=1}^{|S_{train}|}\tilde{\sigma_l}(\Delta_k, T_p)}{std_{T_p}}$$

  where std_{T_p} is the standard deviation calculated over all the deltas and all the training samples for the fixed maturity T_p.

- The third way is to use the same standardizing statistics for all the maturities and deltas.

$$\tilde{\sigma_i}^{train\ normalized}(\Delta_k, T_p) := \frac{\tilde{\sigma_i}^{train}(\Delta_k,T_p) - \frac{1}{\delta_{target}\mathcal{T}_{target}|S_{train}|}\sum_{p=1}^{\mathcal{T}_{target}}\sum_{k=1}^{\delta_{target}}\sum_{l=1}^{|S_{train}|}\tilde{\sigma}_l(\Delta_k,T_p)}{std}$$

$$\tilde{\sigma_i}^{test\ normalized}(\Delta_k, T_p) := \frac{\tilde{\sigma_i}^{test}(\Delta_k,T_p) - \frac{1}{\delta_{target}\mathcal{T}_{target}|S_{train}|}\sum_{p=1}^{\mathcal{T}_{target}}\sum_{k=1}^{\delta_{target}}\sum_{l=1}^{|S_{train}|}\tilde{\sigma}_l(\Delta_k,T_p)}{std}$$

  where *std* is the standard deviation calculated over all the deltas, all the maturities and all the training samples.

We noticed that the standardization **does not improve** the performance of the models. It just **accelerates** the training phase by reaching the convergence faster. We did not assign a big importance for the standardization process because :

- the training phase does not exceed 10 min on the GPU Tesla $K80$ of Google Colab even without normalization.
- when we use standardization, we have to de-standardize the results to have a correct idea about the MAE (22) because the MAE is not a relative measure.

5.2.6 Arbitrage-free surfaces

The authors of [3] stated that their VAE produces implied volatility surfaces free of static arbitrage (See 3.2). The goal of this part is to check if our hybrid VAE generates arbitrage-free surfaces.

We train the best hybrid VAE (architecture described in table 2) using the loss (9) and the linear cyclical β schedule (See 5.2.3). Then we check the arbitrage on the generated surfaces:

1. We check the butterfly arbitrage based on the property (16). The pointwise derivative in the formula (3.2.1.2) is calculated using the automatic differentiation of Pytorch.
2. Then we check the calendar arbitrage using two approaches which are valid only when there is no butterfly arbitrage (That is why we check the butterfly arbitrage beforehand). The first approach and the second approach are respectively based on the properties (18) and (19).

Metric	Training set	Testing set
% butterlfy arbitrage	0%	0%
% calendar arbitrage 1	27%	43%
% calendar arbitrage 2	0%	0%

Table 7: Percentage of volatility points generated by the best hybrid VAE (underlying: Apple stock). Calendar arbitrage 1: based on the property (18). Calendar arbitrage 2: based on the property (19).

From Table 7, we see that all the generated surfaces are free of butterfly arbitrage. Thus, the calendar arbitrage characterization based on (18) and (19) **are valid** and can be used to check calendar arbitrage.

Interestingly, we see that when we use the derivative of the total variance to check calendar arbitrage (18), we find **arbitraging points** whereas when using the formula based on discrete points without derivative (19), the surface seems to be **free of calendar arbitrage.**

In mathematical terms we have:
$\forall \Delta_k, \{\tilde{\omega}(\Delta_k, T_i)\}_{i \in \mathcal{T}_{target}}$ is a non-decreasing sequence. But $\partial_T \tilde{\omega}(\Delta_k, T_i)$ is not necessarily non-negative for $i \in \mathcal{T}_{target}$.

From this observation, we conclude that the hybrid VAE **learnt how to reproduce the increasing sequences of total variance points given as input**. But, it did not fundamentally learn the arbitrage-free condition (18). Thus, if for a given Δ_k and T_i, we have $\partial_T \tilde{\omega}(\Delta_k, T_i) < 0$ then when we reprice using $\tilde{\sigma}(\Delta_k, T_i + \epsilon)$ for ϵ small enough, we **will not** have arbitrage-free prices.

We suggest 2 possible solutions to solve this problem:

- When interpolating the initial data, we can consider a larger grid of maturities: instead of considering 5 or 7 maturities, we can consider all the maturities needed by traders. This strongly depends on the need.
- We penalize calendar arbitrage during the training loss. We add to (9) a penalization term:

$$\beta_{calendar} \left|\left| \left(\partial_T \widehat{\tilde{\omega}}(\Delta, T) \right)^{-} \right|\right|_F^2 \tag{24}$$

  where $\beta_{calendar}$ is chosen using cross validation and $\widehat{\tilde{\omega}}(\Delta, T)$ is the total variance calculated from the output volatility surface of the hybrid VAE.

We tested the second solution but it did not change anything. Thus we are content with the first solution because if we train the network on all the needed maturities, we are sure that it will produce an increasing sequence $\{\tilde{\omega}(\Delta_k, T_i)\}_{i \in \mathcal{T}_{target}}$ for all Δ_k so **we are sure that the prices calculated based on the generated volatilites will be free of arbitrage.**

It is interesting to mention that it is very easy for the hybrid VAE to learn the property (19). The evolution of the percentage of volatility points where (19) is not respected, during the training process is shown in figure 15. As of the second epoch, $\forall \Delta_k, \{\tilde{\omega}(\Delta_k, T_i)\}_{i \in \mathcal{T}_{target}}$ is a non-decreasing sequence.

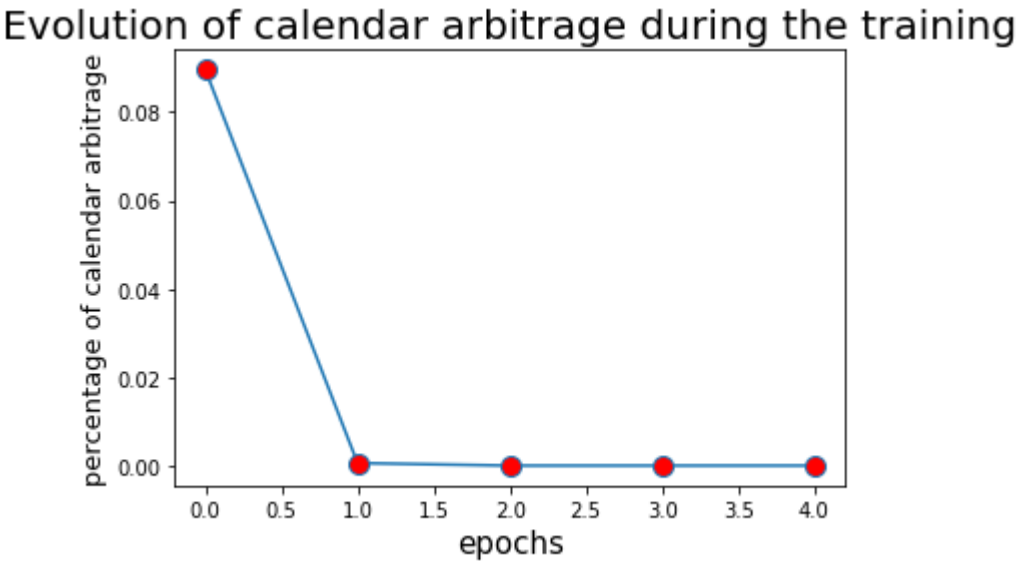


Figure 15: Proportion of calendar arbitrage points in the training of the hybrid VAE

5.2.7 Error heatmap

Given the best hybrid VAE (architecture described in table 2 with a latent space dimension $d = 8$) trained using the loss (9) and the linear cyclical β schedule (See 5.2.3), we display the average distribution of the MAE on the testing set.

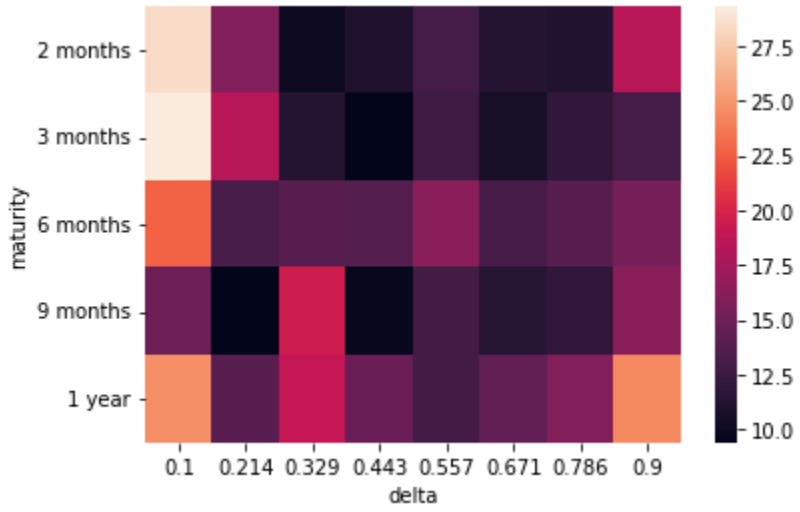


Figure 16: Aveage MAE heatmap over the testing set

From figure 16, we see that the MAE error is concentrated far from the money for small maturities (< 6 months), mainly for $\Delta = 0.1$ ($K > S_0$) and $\Delta = 0.9$ ($K < S_0$). The maximum MAE is for the volatilities at $\Delta = 0.1$ and maturity $= 3$ months. However, the MAE

error at-the-money is small.

This is not problematic for two reasons:

- The Vega[4] of vanilla options are almost null far from the money for the small maturities as it is shown in figure 17. So, the price impact of the error is not considerable.
- The bid-ask range is wider far from the money because the options are more liquid at-the-money. So for deltas far from 0.5/0.6, even if the model is not very precise, it is likely that the predicted values are in the bid-ask range.

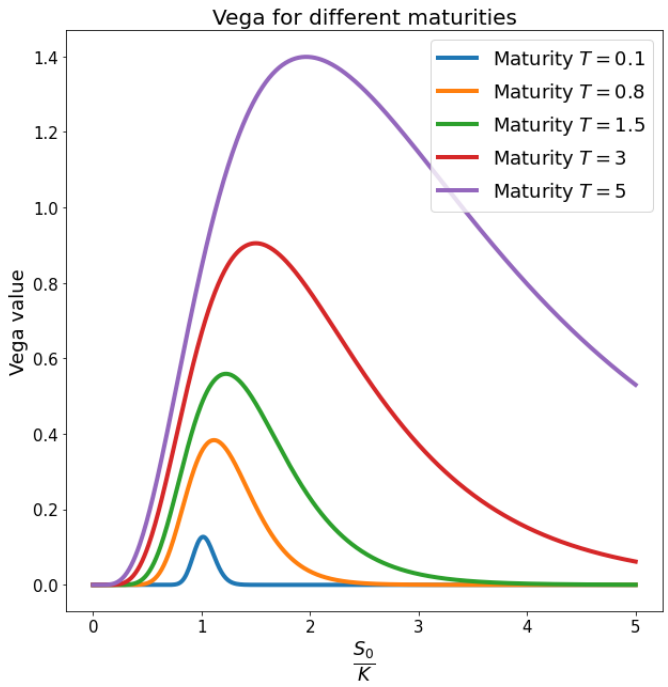


Figure 17: Black-Scholes Vega for different maturities

5.2.8 Multi-asset training

5.2.8.1 Generalization capability

We trained the best hybrid VAE on implied volatility surfaces whose underlying asset is Apple. Then, we tested it on Microsoft and Google volatilities volatilities.

Hybrid VAE trained on Apple volatilities	Metric	Apple testing set	Microsoft testing set	Google testing set
	MAE (bips)	25	23	24
	D_{kl}	26	25	27

[4]The Vega V is the sensitivity of the option price to a movement in the volatility of the underlying asset. For both a call and a put, the Vega $V = S_0 e^{-qT} n(d_1)\sqrt{T}$

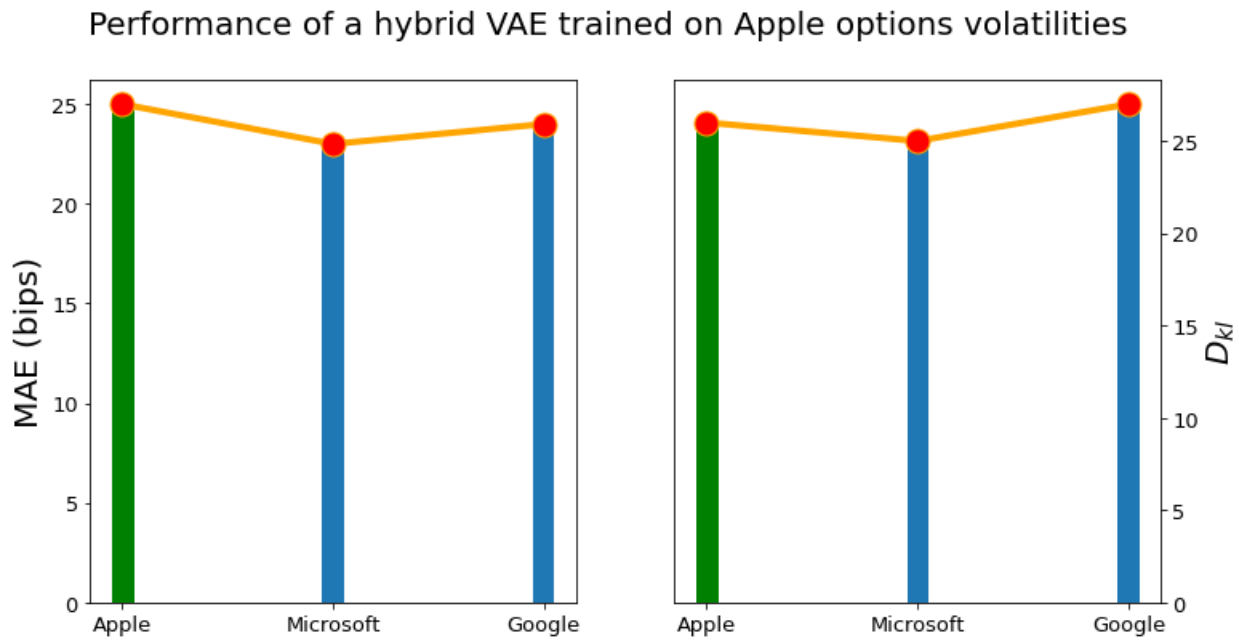


Figure 18: Performance of a hybrid VAE trained on Apple volatilities and tested Microsoft+Google volatilities

We conclude from the figure 18 that the hybrid VAE has the ability to generalize its good performance on similar assets. Trained on Apple volatilities, the hybrid VAE managed to have almost the same results on Microsoft and Google volatilities which are very liquid and have similar properties comparing to Apple options.

5.2.8.2 Labeled conditional VAE

In this section, the idea is to train the hybrid VAE using Apple, Google and Microsoft volatilities together. In order to help the network make the difference between assets, we decided to provide the decoder with **a one-hot encoding label** for each asset and **the spot price of the underlying** (See figure 19).

We do not use a simple enumeration of assets (Microsoft : 1, Google : 2, Apple : 3) to **avoid creating different distances between assets.**. In the one-hot encoding labeling, we assign:

$$[1, 0] \text{ to Apple}$$

$$[0, 1] \text{ to Microsoft}$$

$$[0, 0] \text{ to Google}$$

Note that adding a third dummy variable for Google is useless because it will be highly correlated to the first two dummy variables.

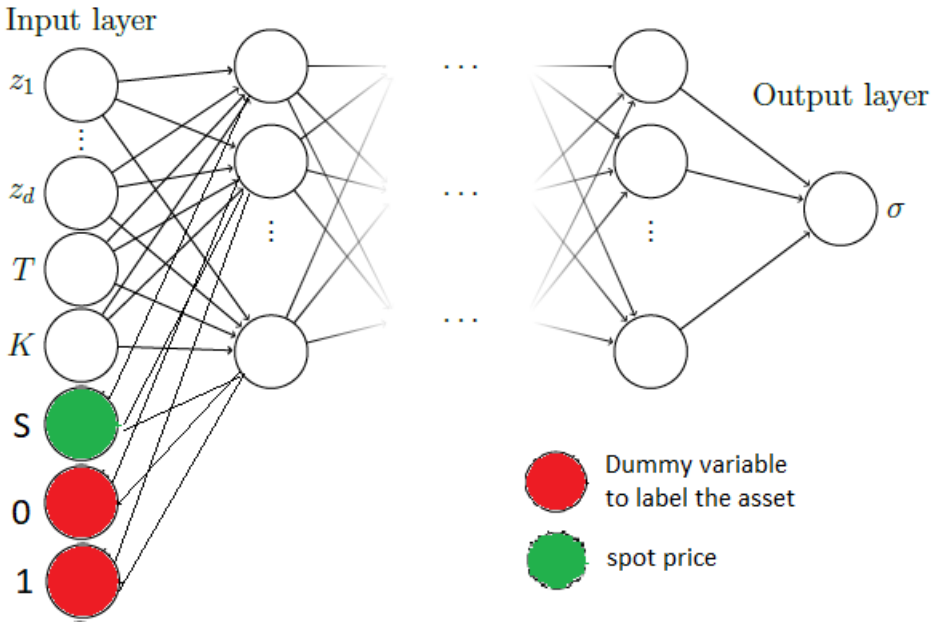


Figure 19: The Decoder of a Hybrid VAE trained on multiple assets

The training is faster and only took 443 sec instead of $\sim$ 10 min. Even if the training set is larger, the training is faster because on every epoch the network has more available data to learn on, which speeds up the convergence.

Hybrid VAE trained using muliple assets	Metric	Apple testing set	Microsoft testing set	Google testing set
	MAE (bips)	25	21	24
	D_{kl}	26	23	23

Table 8: Hybrid VAE trained on multiple assets (underlying: Apple + Google + Microsoft stocks)

From table 8, we see that the model trained on volatilities coming from multiple assets slightly achieves better results than the model trained on one single asset shown in figure 18 of the previous section 5.2.8.1.

Conclusion: From all the previous sections (from 5.2.1 to 5.2.8.2), we conclude an experimental characterization of the best hybrid VAE:

- The best architecture is described in table 2. The latent space dimension $d = 8$.
- The training loss is (9) using the linear cyclical β schedule (See 5.2.3).
- The training set contains Apple+Google+Microsoft volatilities. We provide the decoder with a one-hot encoding and the spot price for each asset (See 5.2.8.2).

6 Implied volatility surface completion

For non-liquid assets such as Banco Bradesco SA (ICE ticker: BBD.NY) and Hello Group Inc (ICE ticker: Hello Group Inc), the volatility surfaces provided by ICE Services may be incomplete. The hybrid VAE may be trained to estimate missing points on partially observed implied volatility surfaces. The main idea is to use liquid assets to train the VAE. Once it is completely trained, we can complete non-liquid volatility surfaces.

To train the hybrid VAE on this specific task , we opted for two methods:

- **Method 1:** On every training epoch (iteration), we randomly sample a specific number of volatility points as an input for the VAE and the task of the decoder is to recover **the exact same volatility points.**
- **Method 2:** On every training epoch (iteration), we randomly sample a specific number of volatility points as an input for the VAE and the task of the decoder is to recover **the whole volatility surface.**

We trained the best hybrid VAE defined in the conclusion of the section 5.2.8.2 on this task using both methods. The initial training grid surfaces have **140 volatility points** (7 maturities and 20 deltas). We trained the VAE using random 40**,** 60 **and** 90 **points**. Then we tested the model on 140−**point testing surfaces**. These testing surfaces contain a mixture of volatilities of the 3 assets: Apple + Google + Microsoft.

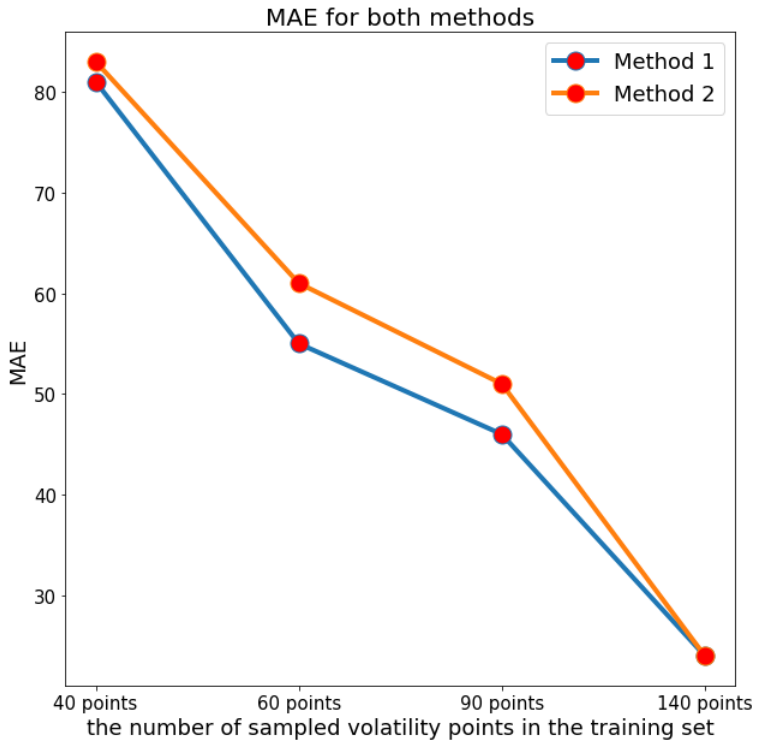


Figure 20: MAE of a hybrid VAE trained on implied volatility surface completion using the two methods introduced above

Expectedly, the MAE on the testing set decreases with the number of sampled points during the training phase to reach MAE = 23.2. We see in figure 20 that **method 1** yields better results than **method 2**.

In our case, when we have missing surface points on some illiquid assets, we can train a hybrid VAE using **method 1** on liquid asset volatility surfaces and use this model to complete the missing points of the illiquid asset. Then we calibrate the internal implied volatility model (of Marex Solutions) on the completed surfaces. Next, we use a local volatility model to price the structured products needed. The local volatility is calculated based on Dupire formula expressed in terms of the implied volatility [14].

7 Conclusion

In this report, I mathematically proved that a linear autoencoder with one-layer encoder and one-layer decoder would achieve the same latent representation as Principal Component Analysis (PCA) (see section 2.2). Besides, I suggested a hybrid variational autoencoder (hybrid VAE) architecture to replicate delta implied volatility surfaces without introducing static arbitrage. To check calendar and butterfly arbitrage, I expressed all the conditions ensuring the absence of arbitrage in terms of Δ (see 3.2). Moreover, I compared the performances of my hybrid VAE architecture to the performance of the feed-forward architecture introduced in Bergeron et al.'s paper [3] (see 5). A hybrid VAE trained on different asset options achieves more accurate results similarly to Bergeron et al.'s paper [3] (see 5.2.8). Finally, I experimentally showed that the hybrid VAE can be trained to complete implied volatility surfaces of illiquid asset options. A hybrid VAE with low KL divergence loss between its latent space and the prior distribution of its latent space (see 2.3) can be used to generate synthetic implied volatility surfaces for stress testing and quantitative investment strategies purposes.

In future works, we might want to capture the evolution of implied volatility surface in time. To achieve this, we can use the Temporal Difference VAE (TD-VAE) designed by Gregor et al. [16] to work with sequential data where each latent code is assumed to be dependent on the previous ones. TD-VAE allows implied volatilities to relate to each other and enables us to simulate the future trend of the market. Applications of TD-VAE include implementing stress tests such as volatility clustering, volatility swaps pricing, and rough volatility model calibration.

References

[1] Ackerer, D., Tagasovska, N., and Vatter, T. Deep smoothing of the implied volatility surface, 2020.

[2] Barata, J. C. A., and Hussein, M. S. The moore–penrose pseudoinverse: A tutorial review of the theory, 2012.

[3] Bergeron, M., Fung, N., Hull, J., Poulos, Z., and Veneris, A. Variational autoencoders: A hands-off approach to volatility, 2022.

[4] Black, F., and Scholes, M. The pricing of options and corporate liabilities, 2019.

[5] Bowman, S. R., Vilnis, L., Vinyals, O., Dai, A. M., Jozefowicz, R., and Bengio, S. Generating sentences from a continuous space, 2015.

[6] Burgess, C. P., Higgins, I., Pal, A., Matthey, L., Watters, N., Desjardins, G., and Lerchner, A. Understanding disentangling in β-vae, 2018.

[7] Chataigner, M., Crépey, S., and Dixon, M. Deep local volatility, 2020.

[8] Daglish, T., Hull, J., and Suo, W. Volatility surfaces: theory, rules of thumb, and empirical evidence, 2007.

[9] Derman, E., Kani, I., Ergener, D., and Bardhan, I. Quantitative strategies research notes, 1995.

[10] Dupire, B., et al. Pricing with a smile, 1994.

[11] Eckart, C., and Young, G. The approximation of one matrix by another of lower rank, 1936.

[12] Fu, H., Li, C., Liu, X., Gao, J., Celikyilmaz, A., and Carin, L. Cyclical annealing schedule: A simple approach to mitigating kl vanishing, 2019.

[13] Fukasawa, M. The normalizing transformation of the implied volatility smile, 2012.

[14] Gatheral, J. The volatility surface: a practitioner's guide, 2011.

[15] Gatheral, J., and Jacquier, A. Arbitrage-free svi volatility surfaces, 2014.

[16] Gregor, K., Papamakarios, G., Besse, F., Buesing, L., and Weber, T. Temporal difference variational auto-encoder, 2018.

[17] Hinton, G. E., and Salakhutdinov, R. R. Reducing the dimensionality of data with neural networks, 2006.

[18] Hull, J., and Suo, W. A methodology for assessing model risk and its application to the implied volatility function model, 2002.

[19] Jackwerth, J. C., and Rubinstein, M. Recovering probability distributions from option prices, 1996.

[20] Kingma, D. P., and Welling, M. Auto-encoding variational bayes, 2013.

[21] LeCun, Y., Kavukcuoglu, K., and Farabet, C. Convolutional networks and applications in vision, 2010.

[22] Lucic, V. It is ok to time-interpolate implied volatility for fixed delta.

[23] Mitra, S. A review of volatility and option pricing, 2009.

[24] Neidinger, R. D. Introduction to automatic differentiation and matlab object-oriented programming, 2010.

[25] Plaut, E. From principal subspaces to principal components with linear autoencoders, 2018.

[26] Rosenblatt, F. The perceptron: a probabilistic model for information storage and organization in the brain., 1958.

[27] Rubinstein, M. Implied binomial trees, 1994.

[28] Shlens, J. Notes on kullback-leibler divergence and likelihood, 2014.

Appendix A

Proof of (16):

In the absence of butterfly arbitrage, $x \mapsto d_1(x,T)$ is strictly decreasing (by Fukazawa [13]) so d_1^{-1} is well defined and non-increasing. From 15, we have $x \mapsto \mu(x,T)$ is non-decreasing. In this case, we can define $\tilde{\mu}(\Delta,T)$ by $\Delta \mapsto \tilde{\mu}(\Delta,T) := \mu(d_1^{-1}(\mathcal{N}^{-1}(\Delta)),T)$. By composition $\tilde{\mu}$ is non increasing.

Now, let us suppose that $x \mapsto d_1(x,T)$ is strictly decreasing (Then d_1^{-1} is well defined) and $\Delta \mapsto \tilde{\mu}(\Delta,T) := \mu(d_1^{-1}(\mathcal{N}^{-1}(\Delta)),T)$ is non-increasing. Thus $x \mapsto \tilde{\mu}(\mathcal{N}(d_1(x,T))) = \mu(x,T)$ is non-decreasing. Then, there is no butterfly arbitrage (by 15).

Thus we proved that a surface is free of Butterfly arbitrage if and only if

$$\begin{cases} \forall T>0,\ x \mapsto \Delta(x,T) \text{ is strictly decreasing} \\ \forall T>0,\ \Delta \mapsto \tilde{\mu}(\Delta,T) \text{ is non-increasing in terms of } \Delta \end{cases}$$

Now, let us calculate $\tilde{\mu}(\Delta,T) := \mu(d_1^{-1}(\mathcal{N}^{-1}(\Delta),T)$ when it is well defined. We have

$$\mu(x,T) = \mathcal{N}\left(f_2\left(x,T\right)\right) + \partial_x\sqrt{\omega}(x,T)n\left(f_2\left(x,T\right)\right)$$

We start by expressing $\partial_x\sqrt{\omega}(x,T)$ in terms of Δ.

For simplicity we work for an arbitrary maturity T, we remove all the dependencies on T. We denote $\sqrt{\tilde{\omega}(\Delta,T)}$, $\sqrt{\omega(x,T)}$ and $d_1(x,T)$ by $\tilde{z}(\Delta)$, $z(x)$ and $d_1(x)$ respectively.

We have

$$\tilde{z}(\Delta) = z\left(d_1^{-1}\left(\mathcal{N}^{-1}\left(\Delta\right)\right)\right)$$

$$z(x) = \tilde{z}\left(\mathcal{N}\left(d_1\left(x\right)\right)\right)$$

$$\Delta = \mathcal{N}(d_1(x))$$

then

$$z'(x) = \tilde{z}'(\Delta)d_1'(x)n(d_1(x)) \tag{25}$$

From the formula of d_1 defined in 3.1, we have

$$z(x)d_1(x) = -x + \frac{z(x)^2}{2}$$

then

$$z'(x)d_1(x) + d_1'(x)z(x) = -1 + z(x)z'(x)$$

We replace $z'(x)$ by its expression (25). We obtain:

$$d_1'(x)\left(z(x) + \tilde{z}'(\Delta)\, d_1(x)\, n(d_1(x)) - z(x)\, n(d_1(x))\, \tilde{z}'(\Delta)\right) = -1$$

Then we replace $d_1(x)$ and $z(x)$ by $\mathcal{N}^{-1}(\Delta)$ and $\tilde{z}(\Delta)$ respectively. After rearranging terms, we obtain:

$$d_1'(x) = \frac{-1}{\tilde{z}(\Delta)\left(1 - \tilde{z}'(\Delta)n(\mathcal{N}^{-1}(\Delta))\right) + \mathcal{N}^{-1}(\Delta)n\left(\mathcal{N}^{-1}(\Delta)\right)\tilde{z}'(\Delta)}$$

We inject this expression in (25), we obtain :

$$z'(x) = \frac{-\tilde{z}'(\Delta)}{\frac{\tilde{z}(\Delta)}{n(\mathcal{N}^{-1}(\Delta))}\left(1 - \tilde{z}'(\Delta)n(\mathcal{N}^{-1}(\Delta))\right) + \mathcal{N}^{-1}(\Delta)\tilde{z}'(\Delta)} \tag{26}$$

Now we have $\partial_x\sqrt{\omega}(x,T)$ in terms of Δ. What is remaining is to express $f_2(x)$ figuring in the expression of $\mu(x,T)$ in terms of Δ.

From the formulas of f_1 and f_2 in section 3.2.1.1, we have $f_2(x) - f_1(x) = z(x) = \tilde{z}(\Delta)$ and $f_1(x) = -d_1(x)$. Then

$$f_2(x) = f_1(x) + \tilde{z}(\Delta)$$
$$f_2(x) = -d_1(x) + \tilde{z}(\Delta)$$

$$f_2(x) = -\mathcal{N}^{-1}(\Delta) + \tilde{z}(\Delta) \tag{27}$$

After injecting (27) and (26) in the expression of $\mu(x,T)$ defined in section 3.2.1.1, we obtain the expression of $\tilde{\mu}(\Delta,T)$

$$\mathcal{N}\left(\sqrt{\tilde{\omega}(\Delta,T)} - \mathcal{N}^{-1}(\Delta)\right) - \frac{n\left(\sqrt{\tilde{\omega}(\Delta,T)} - \mathcal{N}^{-1}(\Delta)\right)\partial_\Delta\sqrt{\tilde{\omega}}(\Delta,T)}{\frac{\sqrt{\tilde{\omega}(\Delta,T)}}{n(\mathcal{N}^{-1}(\Delta))}\left(1 - \partial_\Delta\sqrt{\tilde{\omega}}(\Delta,T)n\left(\mathcal{N}^{-1}(\Delta)\right)\right) + \partial_\Delta\sqrt{\tilde{\omega}}(\Delta,T)\mathcal{N}^{-1}(\Delta)}$$

Proof of (vi):

Let r be the rank of D, then by the SVD we have :

$$D = \begin{pmatrix} V_r & V' \end{pmatrix}\begin{pmatrix} \Sigma_r & 0_{r\times(d-r)} \\ 0_{(n-r)\times r} & 0_{(n-r)\times(d-r)} \end{pmatrix}\begin{pmatrix} U_r & U' \end{pmatrix}^T$$

$$D^+ = \begin{pmatrix} U_r & U' \end{pmatrix} \begin{pmatrix} \Sigma_r^{-1} & 0_{r\times(n-r)} \\ 0_{(d-r)\times r} & 0_{(d-r)\times(n-r)} \end{pmatrix} \begin{pmatrix} V_r & V' \end{pmatrix}^T$$

Then

$$DD^+ = V_r V_r^T$$

and it is clear that $V_r^T V_r = I_r$