

Mémoire présenté devant l'ENSAE Paris
pour l'obtention du diplôme de la filière Actuariat
et l'admission à l'Institut des Actuaires
le 14/03/2021

Par : **Eya Ben Slama**

Titre : **Deep pricing and deep calibration:
applications in finance and insurance**

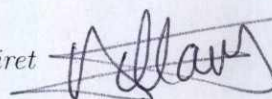
Confidentialité : NON OUI (Durée : 1 an 2 ans)

Les signataires s'engagent à respecter la confidentialité indiquée ci-dessus

Membres présents du jury de la filière

Entreprise : Mazars

Nom : Caroline Hillairet



Signature :

david.dagane

Signé numériquement par
david.dagane
DN: cn=david.dagane, ou=Users,
email=david.dagane@mazars.fr
Date: 2022.02.22 13:59:03
+01'00'

Membres présents du jury de l'Institut
des Actuaires

Nicolas Zec

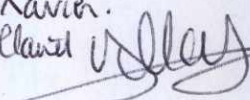
Directeur du mémoire en entreprise :
David Dagane

Nom :

Signature : **david.dagane**

Signé numériquement par
david.dagane
DN: cn=david.dagane, ou=Users,
email=david.dagane@mazars.fr
Date: 2022.02.22 13:58:11
+01'00'

NEGRI François Xavier.
(visio) p/c Hillairet



Autorisation de publication et de
mise en ligne sur un site de
diffusion de documents actuariels
(après expiration de l'éventuel délai de
confidentialité)

Signature du responsable entreprise

Secrétariat:

david.dagane

Signé numériquement par
david.dagane
DN: cn=david.dagane, ou=Users,
email=david.dagane@mazars.fr
Date: 2022.02.16 22:32:43
+01'00'

Bibliothèque:

Signature du candidat



Acknowledgements

To my family, for their love and unwavering encouragement.

I would like to thank the PIF team for their warm welcome and their trust especially David, Thomas and Christophe.

Special thanks to my academic supervisor Caroline Hillairet for her support, valuable advice and her accessibility. I extend my gratitude to the entire teaching staff of ENSAE.

I would also like to thank all the people in Mazars Actuariat for the enriching discussions and the excellent moments of sharing.

Abstract

Regulators are imposing increasingly demanding requirements on banks and insurance companies for the monitoring of their key risk parameters. For instance, the Fundamental Review of The Trading Book (FRTB) under the internal model approach (IMA), now requires the calculation of expected shortfall (ES) and stressed ES with multiple liquidity horizons for each risk category, compared to a single liquidity horizon and calculations of value at risk (VaR) and stressed VaR under Basel III.

These requirements increase by more than tenfold the computational requirements to assess market risk capital and make the management of portfolios with highly complex and structured products a challenging task. Therefore, exploring new approaches adaptable for high-dimensional problems that challenge Monte Carlo simulations accuracy and outperform their computational efficiency becomes a crucial goal for the industry.

In this report, we use neural networks as universal approximators, to replicate a valuation function (model) on a given parameter space of interest. We first focus on the case of financial options valuation using the Black-Scholes model and a stochastic volatility model (Heston). We prove the methodology constructed first on European vanilla options and then generalize it to more complex and exotic instruments (Asian options, lookback options, etc.).

We then present two approaches to differently calibrate commonly used valuation models, taking the example of the Black-Scholes and Heston models. These two approaches capitalize on the trained neural network (deep calibration). Finally, we present various interesting applications of these approaches, in particular for risk management, and highlight their computational efficiency compared to traditional simulation methods.

The last two chapters focus on evaluating the validity of the approach deep pricing to the valuation of annuity variables (VA). These are unit-linked life insurance contracts that offer several guarantees providing exposure to the financial markets while protecting against market downturns. The valuation of these increasingly complex contracts (combining different guarantees) requires the consideration of all underlying risks (e.g. modeling of surrender behavior, longevity risk, market risk, etc.) and involves nested simulation approaches. These approaches are computationally intensive, hence the appeal of deep pricing methods.

Keywords:

Deep pricing, Deep calibration, Black-Scholes, Heston, Variable Annuities, GMxB

Résumé

Les régulateurs imposent des exigences de plus en plus strictes aux banques et aux compagnies d'assurance pour le suivi de leurs principaux paramètres de risque. Par exemple, la revue fondamentale du trading book (FRTB) exige désormais dans le cadre de l'approche par modèle interne, le calcul de l'Expected Shortfall (ES) et de l'ES stressée sur plusieurs horizons de liquidité pour chaque catégorie de risque. Ce calcul consistait en un calcul de Value at Risk (VaR) et de VaR stressée sur un seul horizon de liquidité sous Bâle III. Ces exigences multiplient par plus de dix les besoins de calcul pour évaluer le capital réglementaire de risque de marché et complexifient la gestion des portefeuilles contenant des produits structurés. Par conséquent, l'exploration de nouvelles approches plus efficaces que les simulations de Monte Carlo (MC) devient un objectif crucial pour l'industrie.

Dans ce rapport, nous utilisons les réseaux de neurones comme approximateurs universels, pour répliquer une fonction de valorisation (modèle) sur un espace de paramètres d'intérêt donné. Nous nous intéressons d'abord au cas de valorisation des options financières en utilisant le modèle de Black-Scholes et un modèle de volatilité stochastique (Heston). Nous prouvons la méthodologie construite d'abord sur des options européennes vanilles et la généralisons ensuite à des instruments plus complexes et exotiques (options asiatiques, options lookback, etc.).

Nous présentons ensuite deux approches pour aborder différemment la calibration des modèles de valorisation couramment utilisés, en prenant l'exemple des modèles Black-Scholes et Heston. Ces deux approches capitalisent sur le réseau de neurones entraîné (deep calibration). Nous présentons enfin différentes applications intéressantes de ces approches en particulier pour la gestion des risques et en soulignons l'intérêt.

Nous consacrons les deux derniers chapitres à l'évaluation d'une potentielle application du deep pricing à la valorisation des variables annuités (VA). Il s'agit de contrats d'assurance-vie en unités de compte dont les garanties permettant de bénéficier de l'exposition aux marchés financiers notamment aux marchés actions, obligataire et immobilier tout en étant protégé en cas de chute des marchés. La valorisation de ces contrats de plus en plus complexes (combinant différentes garanties) nécessite la prise en compte de l'ensemble des risques sous-jacents (par exemple la modélisation du comportement de rachat, le risque de longévité, le risque de marché, etc.) et fait intervenir des approches de simulations dans les simulations (Nested simulations). Ces approches sont computationnellement intensives, d'où l'intérêt de l'approche du deep pricing.

Mots-clés: Deep pricing, Deep calibration, Black-scholes, Heston, Variable Annuities, GMxB

Contents

1	Neural networks fundamentals	8
1.1	Feedforward neural networks	9
1.1.1	Architecture	9
1.1.2	Theoretical basis of neural networks	10
1.1.3	More sophisticated architectures	10
1.2	Training phase	11
1.2.1	Training, validation and test datasets	11
1.2.2	Trainable parameters initialization	12
1.2.3	The learning algorithm: Gradient descent	12
1.2.4	Efficient calculations of gradients: the backpropagation algorithm	15
1.2.5	Hyper-parameters tuning	15
2	Deep pricing (learning model price function)	17
2.1	Litterature review	17
2.2	Option Pricing models	18
2.3	Supervised approach	20
2.3.1	Data generation	20
2.3.2	Numerical results	22
2.3.3	Limits of the neural network approach	30
2.4	Unsupervised approach	31
2.5	Deep pricing conclusion	31
3	Deep calibration: option pricing models calibration	33
3.1	Deep calibration paradigm	33
3.1.1	Problem overview	33
3.1.2	Problem's reformulation	34
3.1.3	Numerical implementation	35
3.1.4	Methodology description	36
3.2	Numerical results	39
3.2.1	Results	39
3.2.2	Calibration results on market data (CAC 40)	40
3.2.3	Critics and further improvements	41
3.3	Contributions of the neural network approach from a risk management point of view	42
3.3.1	More stringent regulatory constraints:the Fundamental Review of the Trading Book	42
3.3.2	Model validation and stress-tests	44

4 Pricing of Variable Annuity contracts	45
4.1 Variable Annuities market: Context, trends, and challenges	45
4.1.1 A changing retirement context	45
4.1.2 Variable Annuities market trends and challenges	47
4.2 Variable Annuities: presentation and definitions	49
4.2.1 Presentation of Variable Annuities contracts	49
4.2.2 Useful definitions	50
4.3 Variable Annuities valuation	52
4.3.1 The guaranteed minimum death benefit (GMDB)	52
4.3.2 Guaranteed Minimum Accumulation Benefit (GMAB)	53
4.3.3 Guaranteed Minimum Income Benefit (GMIB)	54
4.3.4 Guaranteed Minimum Withdrawal Benefit (GMWB)	55
4.4 Challenges associated with VA pricing and hedging	56
4.4.1 Pricing of Variable Annuity contracts in practice	56
4.4.2 Overview of the problems and commonly used solutions	58
5 Variable Annuities deep pricing approach	60
5.1 Training dataset	60
5.1.1 Gan and Valdez Synthetic portfolio of VA	61
5.1.2 Data Enrichment	64
5.2 Results and further improvements	67
5.2.1 Numerical results	67
5.2.2 Further improvements and applications	69
Conclusion	72
Annexes	74
Bibliography	74
Executive summary	79

List of Figures

1.1	Example of a two-hidden layer neural network	10
1.2	Training phase example	12
2.1	Samples generated according to different distributions	21
2.2	Hidden layers= [100,100], batch size= 1024, epochs= 50, batch normaliza- tion= no, learning rate= 10^{-4}	23
2.3	Scatter plot of the predicted and target prices by moneyness	24
2.4	Normalized error distribution by maturity and volatility	25
2.5	Train and validation loss vanilla option (unoptimized model, no fixed grid) .	25
2.6	Comparison of tuned and untuned model predictions	26
2.7	Impact of the learning rate on the loss convergence	27
2.8	Train and validation loss, Asian option, fixed moneyness grid	28
2.9	Average relative error on the test dataset (Heston call price)	28
2.10	Neural network approximation on a random test sample (Heston)	29
3.1	Synthetic data	36
3.2	Average relative errors on the test dataset	36
3.3	Average relative errors on the test dataset	37
3.4	Calibration results	39
3.5	Calibration results	40
3.6	IVS of the CAC 40 (01/06/2020) (source: Bloomberg: OVDV)	41
3.7	FRTB regulatory overview	43
3.8	Market risk capital components under FRTB	43
4.1	Evolution of the age pyramid by 2050 according to the United Nations pro- jections of 2008 (median scenario)	46
4.2	Hierarchy of French pension schemes	46
4.3	Overview of life-insurance products	47
4.4	U.S. Individual Annuities Sales Trends between 1986-2020 (\$ Billions), LIMRA.	48
4.5	Projected population aged 60+ in Asia, AXA GIE, 2017	49
4.6	Overview of Variable Annuities guarantees according to customer needs . . .	52
4.7	GMDB example	54
4.8	GMAB example	54
4.9	GMIB example	55
4.10	GMWB example	56
4.11	Financial and non financial risk drivers	57
4.12	A general framework of metamodeling approaches	59
5.1	Risk neutral scenarios for Large Cap Equity index value	63
5.2	US Treasury yield curve	65

5.3	US yield curve principal components	66
5.4	Equity indices and investment funds covariance matrices	67
5.5	Main variables correlations	68

"Deep learning will transform every single industry."

Andrew Ng

Introduction

To strengthen the soundness of the financial system, regulators are imposing stricter requirements on both banks and insurance companies in monitoring their key risk metrics. For instance, the Fundamental Review of The Trading Book (FRTB) under the internal model approach (IMA), now requires the calculation of expected shortfall (ES) and stressed ES with multiple liquidity horizons for each risk category, compared to a single liquidity horizon and calculations of value at risk (VaR) and stressed VaR under Basel III.

These requirements increase by more than tenfold the computational requirements to assess market risk capital and make the management of portfolios with highly complex and structured products a challenging task. Therefore, exploring new approaches adaptable for high-dimensional problems that challenge Monte Carlo simulations accuracy and outperform their computational efficiency becomes a crucial goal for the industry.

Recently, with the increasing applications of artificial intelligence to financial and insurance problematics, deep neural networks were considered to approximate the pricing and calibration models as an alternative to the traditional Monte Carlo simulations. In fact, neural networks offer efficient approximations with no curse of dimensionality. Although the idea to use neural networks in finance was already researched in the 90s, its applications only thrived in recent years.

Considerable calculations speedups compared to the Monte Carlo approach can be achieved by trading off compute time for training with inference time for pricing and thus induce drastic improvements in both the financial and insurance sectors.

For instance, this technique can make live exotic option pricing a realistic goal and model calibration an easier task especially for products that require intraday valuations of their key risk metrics. This can typically be useful for insurers managing and hedging the risks associated with Variable Annuity (VA) contracts. Although Monte Carlo (MC) simulations are widely adopted by insurance companies for the valuation of their VA portfolios, the complex structure of these contracts makes their accurate valuation a computationally demanding task.

We will start by introducing the deep pricing paradigm in the case of options pricing using different models: the Black-Scholes model and a stochastic volatility model (the Heston model). We will start with a proof of concept on European vanilla options pricing and generalize the methodology to more complex exotics (Asian options, lookback options, etc..).

In the deep pricing paradigm, option prices are a function of the derivative parameters (the initial stock price, the strike, the time to maturity, the dividends, etc) and model parameters (the stock prices volatility in the case of the Black-Scholes model). Deep pricing aims to learn the solution for all parameters simultaneously. The idea is to train a neural network on synthetic samples properly generated on a given parameters space of

interest. After introducing the fundamental notions and principles of deep learning in **the first chapter**, we formalize this approach in **the second chapter**. We assess its accuracy under the Black-Scholes and Heston models, compare different approaches to generate the training data, and highlight the importance of this step. We finally show the limits of this approach and suggest avenues for improvement. At the end of the chapter, we introduce the unsupervised approach which was briefly tested during the internship but not retained or developed. Therefore, we do not dwell on it.

Correctly learning the pricing function of a given model is not enough. Being able to calibrate the model to market data is our crucial next step. Calibration is one of the most important inverse problems in finance and insurance: we search for models which explain current price structures in the market. We introduce in **the third chapter** two calibration approaches using the previously learned pricing map to find, given a price/volatility surface, the model parameters that are most likely to generate it. We compare the two approaches, highlight their limits and the next steps. We conclude this chapter with a summary of some challenging regulatory requirements that the presented approaches can help alleviate.

The remaining two chapters introduce Variable Annuity policies, their valuation framework and recall some of the challenges associated with managing a large portfolio of them. Finally, we examine the validity of the deep pricing approach on the valuation of these equity-linked insurance policies.

1

Neural networks fundamentals

The term "deep learning" is used to describe deep neural networks, in other words, neural networks with many layers and neurons and hence with a very large number of trainable parameters. Calibrating a large number of parameters is very challenging in classical statistical approaches.

Neural networks appeared in the 1943 pioneering work by Warren McCulloch and Walter Pitts[26] which simulates neurons in the brain, aiming to develop artificial intelligence.

The huge success of deep learning relies on three pillars;

- The application of backpropagation: In fact, neural networks (NNs) support high-speed training. They are calibrated by minimizing some loss function. Therefore optimization of this function is required. The bigger the NN, the larger the number of parameters to fit. For large NNs, taking numerical gradients for optimization is computationally infeasible. Luckily, we can do so analytically rather than numerically, using the backpropagation algorithm.
- Advancements in computational capacity and processor speed: Decades ago neural networks were at best able to handle two layers. Nowadays, networks with as high as 40 layers are trainable thanks to increases in computational speed due to the development of graphics processing units (GPUs). Indeed, the calculations in neural networks optimization involve high dimensional matrix calculations also known as tensors calculations, highly optimized on GPUs.
- The availability of big data: This is a significant driver of the success of deep learning in general, as data is today much more available than it has ever been.

In this chapter, we first introduce the main concepts and definitions used in deep learning to understand the approaches we will be developing in the following chapters. We first present the notion of a feedforward neural network. This is the most fundamental network architecture and it already works quite well.

1.1 Feedforward neural networks

Neural networks are inspired by the functioning of the neurons in the brain and they tend to mathematically mimic it, *i.e.* by taking a wide range of inputs, and passing it on through layers of neurons to learn "by experience" the mapping of the input to the output.

1.1.1 Architecture

A feedforward neural network (FFNN) is a parametrized function $f(\cdot; \theta) : \mathbb{R}^{N_0} \rightarrow \mathbb{R}^{N_L}$ such that:

$$f(\cdot; \theta) := F^{(L)} \circ F^{(L-1)} \circ \dots \circ F^{(1)} \quad \text{where } F^{(l)} : \begin{array}{l} \mathbb{R}^{N_{l-1}} \rightarrow \mathbb{R}^{N_l} \\ x \rightarrow \sigma_l(W^{(l)}x + b^{(l)}) \end{array} \quad (1.1)$$

The number of **hidden layers** is $L - 1$. The first layer is the input layer while the layer L is the output layer, $\sigma_l : \mathbb{R} \rightarrow \mathbb{R}$ is a non linear function called **activation function** of layer l applied component-wise, $W^{(l)} \in \mathcal{M}_{N_l, N_{l-1}}(\mathbb{R})$ is the layer l **weight matrix** (an $N_l \times N_{l-1}$ matrix) and $b^{(l)} \in \mathbb{R}^{N_l}$ is the layer l **bias**. The parameter $\theta \in \mathbb{R}^q$ gathers all the weight matrices and biases.

In other words, this is a **nonlinear regression**. The neurons connect via their inputs and outputs in different ways. In the case of a feedforward network, the data moves only in one direction through a series of layers, each with at least one **neuron (node)**. The first layer represents the original inputs while the last layer the outputs (see fig. 1.1). The dimensions of both the input and the output are fixed.

The output layer can be unidimensional meaning that the neural network predicts a unique value. This is typically the case in our first pricing model where given the model and option inputs the neural network predicts the option model price. The output can also be multidimensional, meaning that the neural network returns a set of values. This is typically the case when predicting a volatility surface or price surface for a set of input parameters. Layers between the input and output are called hidden layers. Each of these neurons receives each output from the layer preceding it as input, and passes its own output to every neuron in the following layer.

A feed forward neural network is therefore defined as follows:

$$F : \mathbb{R}^{n_0} \longrightarrow \mathbb{R}^{n_L} \\ x \mapsto F_L \circ F_{L-1} \circ \dots \circ F_1(x)$$

Different activation functions mentioned in the literature are already implemented in different deep learning modules such as *Pytorch*¹ and *Tensorflow*² and can be tested directly, just to cite a few:

- The logistic function (Sigmoid) defined as $h(x) = \frac{1}{1+\exp(-x)}$ and generally used to return a probability for classification purposes or normalised outputs.

¹<https://pytorch.org>

²<https://www.tensorflow.org>

- The exponential linear unit function (ELU) defined as $h(x) = x, x \geq 0$, else $h(x) = \alpha(\exp(x) - 1)$
- The Rectified Linear Unit function (ReLU) where $h(x) = \max(x, 0)$ The gradient is therefore zero in the region where x is negative, and the neuron is not active.
- The leaky reLU function, where $h(x) = x, x \geq 0$ else $h(x) = \alpha x, \alpha > 0$. In this case, the function allows a weak but not zero gradient, when the neuron is not active (α is small).

Other activation functions were considered. For instance, we used an exponential activation function for the output layer to ensure that the predicted prices are positive. Moreover, some activation functions mentioned in the literature[19] but not predefined in the existing packages were implemented and tested.

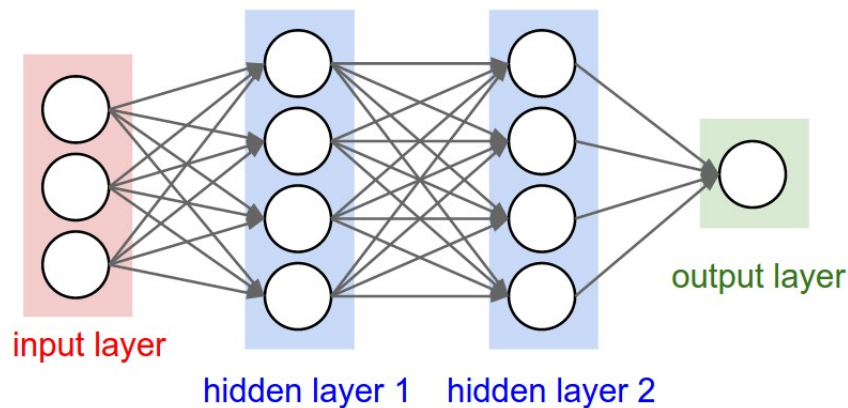


Figure 1.1 – Example of feedforward NN with 3 inputs, 2 hidden layers, and one output

1.1.2 Theoretical basis of neural networks

The fundamental theoretical bases for neural networks are the universal approximation theorems (George Cybenko, Kurt Hornik, et al.)[6][31]. They state that a neural network with a hidden layer containing a large enough but finite number of neurons can approximate any function to any given level of accuracy, under certain conditions on the activation functions.³

1.1.3 More sophisticated architectures

One of the architectures frequently mentioned in the financial literature is recurrent neural networks (RNNs). This architecture is useful to capture temporal dynamic behaviors due to the connection between its different layers RNNs. In fact, this architect allows previous outputs to be used as inputs in the next layers. Therefore they can take into account historical information and manage variable-length sequences of inputs. It can be typically used to predict stock market prices.

However, RNNs are not very good at capturing long-term dependencies. This particular point can be solved by using long short-term memory (LSTM) networks. Such networks

³A sufficient but not necessary condition is the activation functions need to be continuous, bounded, and non-constant.

are similar to RNNs but where hidden layer updates are replaced by memory cells. This difference makes them better at capturing long-term dependencies of the input data.

These architectures do not seem relevant to vanilla options pricing. But can be more suited for path-dependent option pricing or insurance policies pricing where the input is a time series. We may note that in our code changing the network architecture is easy and makes the code modifiable for different situations and problems formulations.

1.2 Training phase

We suppose that our samples (x, y) are drawn independently from an unknown distribution \mathbb{P} . We aim to minimize the generalization error which is simply the expected average error the neural network \mathcal{N}_θ would have on samples drawn from this same distribution \mathbb{P} through updates of the parameter set θ :

$$J(\theta) = E_{\mathbb{P}}(\mathcal{L}(x, y, \theta)) \quad (1.2)$$

$\mathcal{L}(x, y, \theta)$ needs to reflect the performances of the network and thus the error on each sample (per sample loss function). As our problem is a regression problem, we use the mean squared error (MSE) as criterion.

One reasonable choice to solve this problem is to minimize the empirical error. As we assume that samples are independent and drawn from the same distribution the law of large numbers guarantees that almost surely the empirical error converges to the generalization error.

1.2.1 Training, validation and test datasets

We aim to minimize the generalization error over all samples drawn from the distribution \mathbb{P} but in practice, we only have a limited number of observations, two main problems may arise: overfitting and underfitting.

- Overfitting occurs when the neural network sticks too much to the training samples and fails to generalize. In practice, the neural network error on a testing dataset (not used for the training) is much higher than the training error (error on the training samples).
- Underfitting means the network fails to learn the mapping function accurately. For example, this can be the case when the generalization error is stuck at a local minimum much higher than the global one. Such a problem shows when the neural network error exceeds the average error on a benchmark of models.

Therefore before training our model, we keep (20% of the dataset) as a testing set, while the remaining dataset is split into a training dataset (90% of the latter) and a validation dataset. The validation dataset is used to track the loss during the training phase and compare it to the loss on the training samples. The model is saved only if the loss on the validation dataset has improved (fig. 1.2).

The test dataset is used in a final step to assess the model's accuracy. We used mainly two types of metrics absolute errors (typically the root mean square error (RMSE) and the

mean absolute error (MAE)) and relative errors (typically the root mean square relative error (RMSRE) and the Mean absolute relative error (MARE)).

$$\text{RMSE} = \sqrt{\frac{1}{n} \sum (y_i - \hat{y}_i)^2} \qquad \text{RMSRE} = \sqrt{\frac{1}{n} \sum \left(\frac{y_i - \hat{y}_i}{y_i}\right)^2}$$

$$\text{MAE} = \frac{1}{n} \sum |y_i - \hat{y}_i| \qquad \text{MARE} = \frac{1}{n} \sum \left| \frac{y_i - \hat{y}_i}{y_i} \right|$$

Epoch 20	Training Loss: 0.0013381984809206593	Validation Loss: 0.08688029646873474
Epoch 21	Training Loss: 0.0012495236264334784	Validation Loss: 0.0838819220662117
Validation Loss Decreased(0.423051---->0.419410)	Saving The Model	
Epoch 22	Training Loss: 0.001172062591649592	Validation Loss: 0.06843466311693192
Validation Loss Decreased(0.419410---->0.342173)	Saving The Model	
Epoch 23	Training Loss: 0.0011011488321754667	Validation Loss: 0.0668438971042633
Validation Loss Decreased(0.342173---->0.334219)	Saving The Model	
Epoch 24	Training Loss: 0.001039964234870341	Validation Loss: 0.07659891247749329
Epoch 25	Training Loss: 0.0009857354025977353	Validation Loss: 0.07797355204820633
Epoch 26	Training Loss: 0.0009360998998292618	Validation Loss: 0.060805611312389374
Validation Loss Decreased(0.334219---->0.304028)	Saving The Model	
Epoch 27	Training Loss: 0.0008913599536754191	Validation Loss: 0.05725258216261864
Validation Loss Decreased(0.304028---->0.286263)	Saving The Model	
Epoch 28	Training Loss: 0.000851828610110614	Validation Loss: 0.05835379287600517
Epoch 29	Training Loss: 0.0008157245633709762	Validation Loss: 0.04991427809000015
Validation Loss Decreased(0.286263---->0.249571)	Saving The Model	
Epoch 30	Training Loss: 0.0007828021266808112	Validation Loss: 0.04231270030140877
Validation Loss Decreased(0.249571---->0.211564)	Saving The Model	

Figure 1.2 – Training phase example

1.2.2 Trainable parameters initialization

During the training, the trainable variables θ (weights and biases) need to be randomly initialized and then updated to minimize the error.

Different initialization methods have been discussed in the literature, the most popular are:

- Random normal initialization (standard normal variables)
- Random uniform initialization
- Xavier/Glorot normal initialization with variance $\sigma = \sqrt{\frac{2}{d_{input} + d_{output}}}$, where d_{input}, d_{output} are respectively the dimension of the input and the dimension of the output.

We used the third method in our implementation. The idea behind this approach is to avoid vanishing or exploding gradients by imposing that the variance of the output of each layer is equal to the variance of the inputs.

1.2.3 The learning algorithm: Gradient descent

The goal of the training phase is to find the generalization error's global minimum. To do so, we use the gradient descent algorithm.

Gradient descent is a line search method⁴ where the step direction is the gradient $\nabla_{\theta} \mathcal{J}(\theta)$ and the step size is lr_n , called the learning rate:

⁴Given a real-valued function f and an iterate x_k line search methods define x_{k+1} as : $x_{k+1} = x_k + \alpha_k p_k$, where p_k and α_k are respectively the step direction and size.

Algorithm 1 Gradient descent algorithm

Require: learning rate $(lr_n)_{n \geq 0} > 0$

Initialise θ

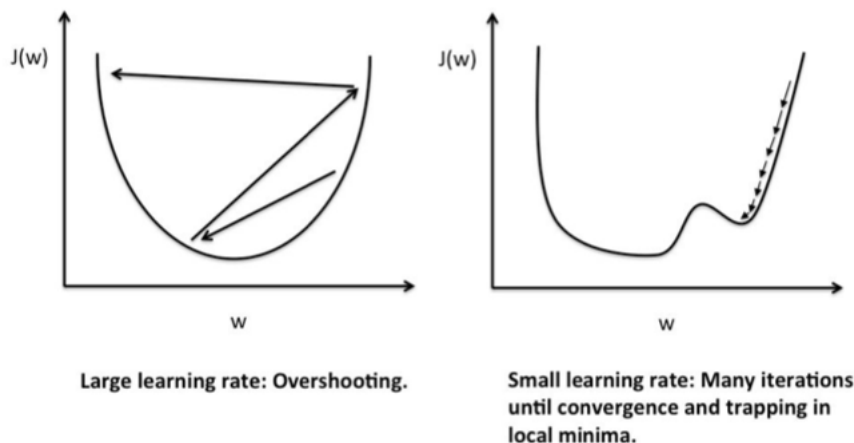
repeat

 Compute the gradient $\nabla_{\theta} \mathcal{L}(x, y, \theta)$

 Update $\theta \leftarrow \theta - lr_n \nabla_{\theta} \mathcal{L}(\theta)$

until Convergence

We compute the objective function gradients for each neuron with respect to its weights, and then the weights are translated in the opposite direction (the direction that should lower the loss function), by a factor lr_n (the learning rate). This is an important hyperparameter that has to be adjusted to prevent both overshooting and slow convergence. Overshooting occurs when the learning rate is too large that it overshoots the minimum and the loss can end up diverging. When the learning rate is too small the neural network will take too long to converge and may end up stuck in a local minimum.



In practice, the algorithms used for the training, although based on the same general principle, differ in the definition of the empirical error in particular the amount of data used to calculate the gradients and hence update the parameters. The most popular versions are:

- (Batch) gradient descent: in this version, the parameters are updated (this is also known as completing a training "epoch"⁵) only after calculating the errors for all the training samples.

$$\nabla_{\theta} \mathcal{L}(x, y, \theta) = \frac{1}{N} \nabla_{\theta} \sum_{i=1}^N \mathcal{L}(x^{(i)}, y^{(i)}, \theta)$$

where N is the total number of samples.

This approach is memory-hungry as it requires the whole training dataset to be in memory. Even though this approach may seem more stable, it doesn't necessarily yield the best results in terms of convergence (local minimum convergence/ slow convergence). Assuming that the losses are independent and identically distributed (iid),

⁵An epoch is when the network completes processing the entire training dataset.

the strong law of large numbers implies that the mean converges to the true value in \sqrt{N} . Consequently, an x -fold accuracy improvement requires x^2 more observations. Therefore, computing several noisier gradients is preferred to using a single, more accurate gradient estimate.

- Stochastic gradient descent: the parameters are updated for each training example, i.e. each observation. On the one hand, this approach allows better tracking of the loss improvement. On the other hand, it is more computationally intensive.
- Mini-batch gradient descent: Unlike the previous version, the gradient is estimated on randomly sampled mini-batches of size m ⁶.

$$\nabla_{\theta} \mathcal{L}(x, y, \theta) = \frac{1}{m} \nabla_{\theta} \sum_{i=1}^m \mathcal{L}(x^{(i)}, y^{(i)}, \theta)$$

Common mini-batch sizes range between 32 and 512. But, there is no clear rule to define the optimal batch size. It depends on the application, the architecture, and the dataset. Using small batches may induce noisy gradients resulting in a less regular error rate decrease and manifesting jumps. Still, this is the most used algorithm since it's a compromise between computational efficiency and estimations robustness.

$$\nabla_w \mathcal{L}(x, y, w) = \frac{1}{N} \nabla_w \sum_{i=1}^N \mathcal{L}(x^{(i)}, y^{(i)}, w)$$

The mini-batch gradient descent algorithm detailed just above is summarised below:

Algorithm 2 mini-batch gradient descent algorithm

Require: learning rate $(lr_n)_{n \geq 0} > 0$, dataset of input variables \mathcal{X} (of size N), N_{batch} , N_{epoch}

Initialise θ

for each of N_{epoch} epochs **do**

Set $\mathcal{X} \leftarrow \mathcal{X}$

while \mathcal{X} is not empty **do**

Sample N_{batch} instances from \mathcal{X} under a uniform distribution to get χ

Update $\mathcal{X} \leftarrow \mathcal{X} \setminus \chi$ (equivalent to sampling without replacement)

Simulate a loss function approximation $\hat{\mathcal{L}}(\theta; \chi)$ from χ

Compute the gradient $\nabla_{\theta} \hat{\mathcal{L}}(\theta; \chi)$ (using back-propagation)

Update $\theta \leftarrow \theta - lr_n \nabla_{\theta} \hat{\mathcal{L}}(\theta; \chi)$

end while

end for

In our implementation, we used the adam optimizer [23] which is an improved version of the classical stochastic gradient descent algorithm. The major difference is that instead of maintaining a single learning rate for all parameters, a learning rate is maintained for each network parameter and separately adapted as learning unfolds.

In fact, the algorithm computes individual adaptive learning rates for each parameter based

⁶The sampling procedure for mini-batches is usually uniform without replacement

on estimates of first (the mean) and second moments (the uncentred variance) of the gradients as follows:

$$\begin{aligned}
 g_t &:= \nabla_{\theta} L(x, y, \theta_t) \\
 m_{t+1} &:= \beta_1 m_t + (1 - \beta_1) g_t \\
 v_{t+1} &:= \beta_2 v_t + (1 - \beta_2) g_t^2 \\
 \theta_{t+1} &:= \theta_t - \alpha \frac{m_{t+1}}{1 - \beta_1^{t+1}} \frac{1}{\sqrt{v_t / (1 - \beta_2^{t+1})} + \epsilon}
 \end{aligned}$$

where $0 \leq \beta_1, \beta_2 < 1$, ϵ and α default parameters and as initialization $m_0 = 0$, $v_0 = 0$. The next step is computing the gradients.

1.2.4 Efficient calculations of gradients: the backpropagation algorithm

The success of neural networks stems from the way these gradients are efficiently computed using the backpropagation principle.

In fact, the backpropagation algorithm is simply a recursive application of chain rule. Its effectiveness arises from the fact that a feedforward network is a large nested function, hence the gradient for each neuron can be calculated via the chain rule.

Backpropagation is an analytical approach that allows the computation of every trainable parameters' loss gradients in only one pass, from the output layer back to the input nodes. On the other hand, numerical methods would need to compute each loss gradient individually. A computationally intensive task that would have made solving this optimization problem infeasible.

1.2.5 Hyper-parameters tuning

Training the neural network involves choosing and tuning its parameters (number of layers, nodes per layer, activation functions, number of training, epochs, batch size, learning rate, etc..) to prevent both overfitting and underfitting.

Choosing the architecture is challenging. We experimentally found that increasing the number of hidden layers beyond 2 or 3 layers does not significantly impact the neural network performance metrics on the test dataset. Consequently, we considered a 2-hidden-layers network. To choose the other hyperparameters, we used a **grid search technique**. This technique considers a pre-defined grid and explores all possible candidates in a brute-force way. We reported some of the used settings below:

Parameters	Options
Activation functions	ReLU, ELU, Sigmoid
Dropout rate	[0.0, 0.1, 0.2, 0.3]
Batch normalization	Yes/no
Number of neurons	[20, 40, 50, 100, 120, 150]
Batch size	[32, 64, 128, 256, 320, 512, 640, 1024]

Table 1.1 – Grid search settings for hyper-parameters optimization (Case of Black-Scholes pricing)

This method is computationally expensive. Therefore, a common approach is to use only a subset of the training data (30%) to find the optimal parameters and then train the tuned neural network on the entire training dataset.[24]

We used a cross-validation method with three folds to compare the trained models. We describe this algorithm below:

Algorithm 3 k -fold cross validation

- Initialize the set of hyper-parameters.
 - Split the training data set into k (we fixed $k = 3$) subsets.
 - Choose one as the validation dataset, and train the remaining datasets.
 - Assess the neural network performance by calculating the metric on the validation set.
 - Continue the above steps on all subsets.
 - The final metric is the average over the k cases.
 - Explore the next set of hyper-parameters and rank the models according to the previous final metric.
-

More efficient algorithms were discussed in the literature typically the random search algorithm introduced by Bergstra and Bengio (2012)[3] and the Bayesian hyper-parameter optimization algorithm[28]. These approaches were not implemented as we focused on building a coherent methodology more than a perfectly tuned model.

2

Deep pricing (learning model price function)

In this chapter, we start with a short review of the application of deep pricing in the literature. We clearly define both the methodology and the data simulation process.

Machine learning techniques have never been used to price products at Mazars Actuariat. Hence, a preliminary step of the project was a proof of the concept on vanilla products (call and put options), to see if this approach can be considered for more complex products (exotics).

2.1 Litterature review

Application of neural networks to option pricing has been researched since 1990, and one of the founder works was Malliaris and Salchenberger's [25][22]. They used a neural network to estimate the close price of S&P 100 options using historical transaction data.

Hutchinson et al.(1994) [18] were the first to attempt to approximate the market option pricing function for put and call options using neural networks.

This approach is hence **model-free**. The NN architecture was quite simple with only a hidden layer and four nodes. Yet, it outperformed the Black-Scholes model in half the test cases.

Several other works used neural networks as universal approximators to approach model prices, especially when no closed-form solution is available. In this case, the pricing task is mainly done using Monte Carlo simulations or numerical schemes. We identified two main approaches in the literature:

- A supervised approach based on minimizing the difference between the predicted prices and the target prices (model prices or market prices depending on the application). We typically use the mean square error (MSE) as a per-sample loss function.
- An unsupervised approach consisting of minimizing a loss function based on a partial differential equation (PDE) verified by the option prices.

In early work, the option pricing and volatility estimation problems were tackled as supervised problems and the feedforward networks represented quite a suitable tool. Recent

approaches extended to more complex architectures (up to 4 hidden layers with 400 neurons each) and rethought the pricing problem as unsupervised, based on the associated PDE. We should mention that feedforward neural networks are still the reference in the literature.

Unlike the market approaches mentioned before, (see for example, Hutchinson et al.(1994) [18]; Yao et al. (2000) [31]; Gencay and Qi (2001)[13]) **our work will focus on model pricing**. We aim to train the NN to learn a model price function on a given input space. This choice is justified by the fact that we do not wish to introduce a “black box” model to predict market prices. But we rather wish to stick to a model and challenge the current methods mainly Monte Carlo approaches, for example, by making live exotics pricing a possible task. Finally, we would like to highlight that the implemented NN can be recycled to a model-free approach. We tested the model on predicting market prices (Mid-price) based on a historical training *S&P* 500 dataset, and the mean relative error was less than 5%.

2.2 Option Pricing models

In this section, we briefly present two widely used option pricing models, the Black-Scholes (BS) (1973) [9] model and the Heston stochastic volatility model (1993) [16].

Option pricing models start from assuming a specific diffusion for the stock price, namely a geometric Brownian motion with constant volatility in the case of the Black-Scholes model and a strictly positive process driven by a Brownian motion and stochastic volatility for the Heston model.

Stochastic volatility models were introduced to better approximate the reality. Since in practice, Black-Scholes’ assumptions are violated. In fact, the stock’s volatility is not constant. The volatility dynamics can be better reproduced by stochastic volatility models for instance the Heston model.

Black Scholes model

Let $(\Omega, \mathcal{F}, \mathbb{P})$ a probability space endowed with the filtration $(\mathcal{F}_t)_{t \geq 0}$.

In the Black-Scholes model, the asset price (S_t) follows a geometric Brownian process, expressed under \mathbb{P} as follows:

$$dS_t = \mu S_t dt + \sigma S_t dW_t \quad (2.1)$$

where W is a \mathbb{P} -Brownian motion, σ is a noise factor called **volatility**, and μ denotes the **drift**. Under the risk-neutral measure \mathbb{Q} , the drift becomes the interest rate and W is now a \mathbb{Q} -Brownian motion.

For a call option with strike K , current stock price S , annualized volatility σ (the standard deviation of the return on the stock), time to maturity T , risk-free rate r , and annualized dividend q the price BS_{call} is given by the following equation:

$$BS_{call} = Se^{-qT} N(d_1) - Ke^{-rT} N(d_2) \quad (2.2)$$

$$d_1 = \frac{\ln(S/K) + (r + \frac{1}{2}\sigma^2) T}{\sigma\sqrt{T}}$$

$$d_2 = d_1 - \sigma\sqrt{T}$$

Using a replicating portfolio¹ and the Itô's lemma to the option price, we derive a partial differential equation (PDE) for European option prices V expressed as a function of time t and the stock price S_t with a terminal condition. Typically, $V(t = T, S) = (S_T - K)_+$ for a European call option.

$$\frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + (r - q)S \frac{\partial V}{\partial S} - rV = 0 \quad (2.3)$$

Heston model

The Heston model came to correct the restrictive assumption of constant volatility made by the Black Scholes model. Volatility is rather modeled by a stochastic process namely a mean-reverting Cox-Ingersoll-Ross (CIR) process.

Let $(\Omega, \mathcal{F}, (\mathcal{F}_t)_{t \geq 0}, \mathbb{P})$ a filtered probability space. In the Heston framework, the diffusions of the stock price and its volatility under the physical probability measure \mathbb{P} are given by the following system of equations:

$$\begin{cases} dS_t = \mu_t S_t dt + \sqrt{v_t} S_t dW_t^s & (2.4) \\ dv_t = \kappa(\theta - v_t) + \sigma \sqrt{v_t} dW_t^v & (2.5) \\ d\langle W^{(s)}, W^{(v)} \rangle_t = \rho dt & (2.6) \end{cases}$$

S_t is the underlying price (stock price) with drift μ and instantaneous variance v_t . θ is the long-run variance (the mean level of the variance) to which v_t reverts with a speed of κ . θ controls the overall level of skew. The variance of the variance is given by σ . It controls the convexity of the smile, the larger σ the more convex the function and hence the more significant the smile. Finally, v_0 denotes the initial variance and W^s and W^v are two correlated \mathbb{P} -Brownian motions with correlation ρ .

The dynamics of S and v under the risk-neutral measure \mathbb{Q} are invariant apart from the drift term μ which becomes $r - q$, where r is the interest rate and q the annual dividend yield. In practice, these parameters are calibrated to market quotes of vanilla options for their liquidity using numerical optimizers and Heston's semi-analytic formula (2.7).

The Feller condition The volatility process v_t is strictly positive if the model's parameters obey the following condition known as the Feller condition:

$$2\kappa\theta > \sigma^2$$

This condition is crucial for data generation in the pricing and calibration phases.

One of the advantages that made the Heston model particularly popular is the fact that it provides a semi-analytical formula for European call option prices.

¹A replicating portfolio for some asset or cash flows is a portfolio of assets with the same cash flows. A static replication refers to a portfolio with the same cash flows as the reference asset at any future time, while dynamic replication, refers to a portfolio that does not have the same cash flows, but rather has the same "Greeks" as the reference asset. The reference asset and replicating portfolio are only assumed to behave similarly at a single point.

The price of a European call option with strike K , maturity T at time t $C(t, K, T)$ can be calculated through Fourier transform as follows [8]:

$$C(t, K, T) = \frac{1}{2}S(t) + \frac{e^{-r(T-t)}}{\pi} \int_0^\infty \operatorname{Re} \left[\frac{K^{-i\phi} f(i\phi + 1)}{i\phi} \right] d\phi - K e^{-r(T-t)} \left(\frac{1}{2} + \frac{1}{\pi} \int_0^\infty \operatorname{Re} \left[\frac{K^{-i\phi} f(i\phi)}{i\phi} \right] d\phi \right) \quad (2.7)$$

where $f(i\phi)$ denotes the characteristic function of the Heston model (see Annexe1).

As in the Black Scholes case we can derive a PDE verified by European option prices [16].

$$\frac{\partial V}{\partial t} + \frac{\partial V}{\partial v} [\kappa(\theta - v_t) - \vartheta(S, v, t)] + (r - q)S \frac{\partial V}{\partial S} + \frac{1}{2}\sigma^2 v \frac{\partial^2 V}{\partial v^2} + \rho\sigma v S \frac{\partial^2 V}{\partial S \partial v} + \frac{1}{2}S^2 v \frac{\partial^2 V}{\partial S^2} = rV \quad (2.8)$$

The $\vartheta(S, v, t)$ represents the "price volatility risk" i.e, the risk of asset price movement due to a change in volatility, usually approximated by the option's greek Vega.

2.3 Supervised approach

The neural network is trained on a simulated dataset. For a given option type we calculate model prices for a set of parameters generated randomly on an interval of interest according to a uniform or normal distribution. Our aim is for the neural network to learn the pricing function for a given derivative and hence to predict as closely as possible the price for any random point of the space of interest. We will start with a proof of concept on vanilla options. After explaining the problems encountered and how we solved them, we test the approach on exotic options with the example of Asian options.

We highlight that our implementation has no dimensional restrictions and therefore can be adapted to the pricing of options in high dimensions. Theoretically, the neural network should yield comparable results. In this section, we focus on learning the price function in both the Black-Scholes and Heston frameworks. The model can also be used to learn implied volatility surfaces as we will see in the next chapter along with the model calibration. Once adequately trained, the neural network should be able to predict instantaneously the price of a given product for different market settings on a given parameters space of interest. This approach will be called "fast surface".

2.3.1 Data generation

Data generation is a crucial step of the training. For each financial model we have two types of inputs: the "observables parameters" and the "model parameters".

Model	Observable parameters	Model parameters
Black and Scholes	(S_0, T, r, q)	σ
Heston	(S_0, T, r, q)	$(\kappa, \theta, \sigma, \rho, v_0)$

Our first approach was randomly choosing the parameters in the space of interest we defined and calculating the associated prices (closed formula for vanilla options and Monte Carlo simulations using $N = 10\,000$ sample paths with 1000 steps each, for exotics). The `data_generation` function can generate Black Scholes and Heston prices for a range of European options (vanilla options, barrier calls and puts, lookback calls and puts, Asian calls and puts, etc). Although for some of these exotics a closed-form formula may exist, we defined a generic pricer based on Monte Carlo simulations. The range of parameters used to generate BS call option prices is shown in the following table.

Parameter	Range
Stock price (S_0)	10- 500
Strike price (K)	7- 650
Maturity in years (T)	0.25- 3
Risk free rate (r)	0%- 3 %
Annualized dividend rate (q)	0%- 3%
Annualized volatility (σ)	5%- 90 %

It should be noted that some generated observations are purely theoretical and do not have a financial sense (for instance, when the strike lies far from the vicinity of the stock price). Thus, **the data was extremely unbalanced** and the observations' frequency were not representative of real market prices.

The following figure (fig.2.1) shows three different generated samples according to three different distributions (uniform, normal (truncated), and lognormal (truncated)).

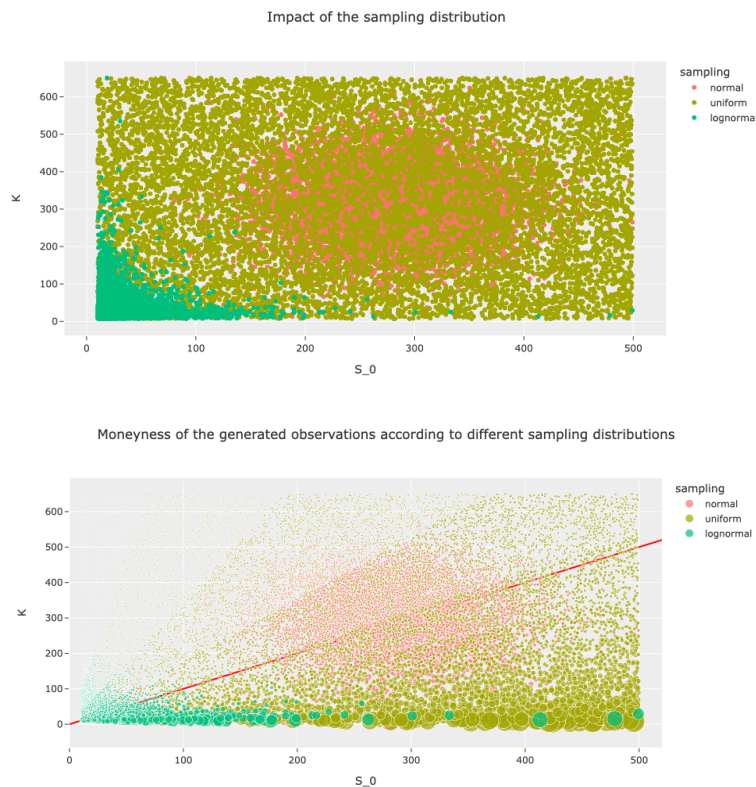


Figure 2.1 – Randomly generated N samples ($N = 10\,000$ according to different distributions

In the second figure, the marker’s size is proportional to the option’s moneyness (S_0/K). It can be noticed, that the uniform distribution yielded many extremes observations (both deeply in and out of the money) compared to the normal distribution where the samples are more concentrated around the money. Yet, the normal distribution seems to shrink the space of interest and the accuracy of the model dropped in the regions that were not well-sampled and represented in the training dataset.

Using the uniform distribution, induced the model’s error being highly correlated to the moneyness of the option. Therefore we built different models by moneyness class to improve the model’s performances. With more hindsight, this seems problematic as it favors overfitting and undermines the approach’s generality.

Modifying the uniform distribution used to generate the parameters samples to a truncated normal distribution globally improved the model’s accuracy. Finally, fixing the moneyness of the generated samples to a given grid, yielded more financially consistent samples and drastically improved the model’s performance.

We divided the generated dataset (typically 100 000 samples) into 3 random sets, training, validation, and test dataset as described in the previous chapter.

Preprocessing

Before passing the prices to the network, we exploited the fact that the pricing function is homogenous in (S, K) :

$$Call(S, K)/K = Call(S/K, 1) \tag{2.9}$$

Accordingly, we modified our data by dividing both stock prices and call prices by the strike. This normalized data was then fed to a two-hidden-layer FFNN to fit the input variables to the output prices. Prices below 10^{-3} were also rounded to this threshold. The input was also normalized before being fed to the neural network. The inverse transformation has been applied to the last layer’s output.

2.3.2 Numerical results

We used mainly two types of metrics absolute errors (typically the root mean square error (RMSE) and the mean absolute error (MAE)) and relative errors (typically the root mean square relative error (RMSRE) and the Mean absolute relative error (MARE)). Report to the previous chapter for the metrics definitions.

Black-Scholes call prices results: (dataset generated according to a uniform distribution without a fixed grid for the option’s moneyness)

First, we consider European call options pricing with a simulated dataset according to a uniform distribution, without a fixed grid for the option’s moneyness.

The first plot shows the train and validation losses per epoch. On the second and third plots, we respectively represent the training loss and the parameters gradient norm on the training batches (fig.2.2).

There seems to be no overfitting, as the validation loss is below the training loss. Both

parameters' gradient norm and the training loss seem to converge.

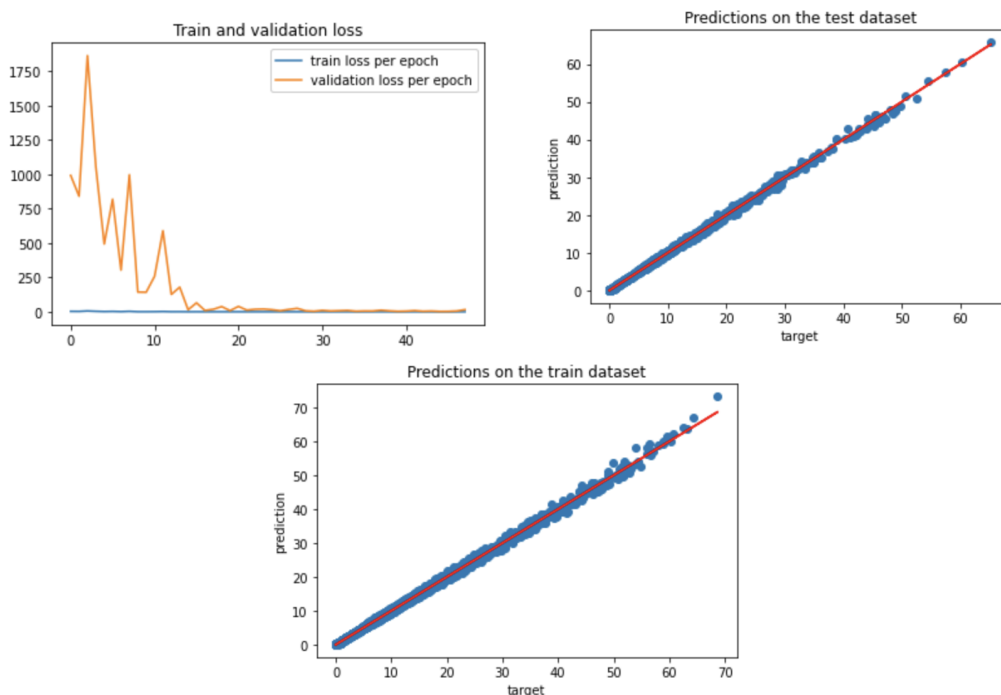


Figure 2.2 – Hidden layers= [100,100], batch size= 1024, epochs= 50, batch normalization=no, learning rate= 10^{-4}

We report the main metrics on both the training and test datasets in the table below (table 2.1).

While the normalized absolute errors are acceptable with an average error of 6% of the strike, the relative errors (average errors divided by the price) are too large (60% on both the training and test datasets). This is mainly due to the unbalanced training dataset, where out-of-the-money (OTM) options² are over-represented. In fact, as OTM option prices tend to be very close to zero, relative errors explode and even tend to be infinite.

Metric	MAE	RMSE	MARE	RMRSE
Test dataset	0.064	0.12	45%	60%
Train dataset	0.065	0.13	45%	60%

Table 2.1 – Cross-validation metrics on the training and test datasets (BS call prices, data generated according to a uniform distribution)

Models by moneyness

To correct this problem, models were constructed by the moneyness class. We defined five classes based on the option's moneyness. Beyond the commonly used in-the-money, out-of-the-money, and at-the-money classes. We introduced deep-in-the-money (DITM)

²An out-of-the-money option is an option that has a strike significantly above (for a call) or below (for a put) the underlying asset price.

and deep-out-of-the-money (DOTM) classes, mainly to distinguish samples that were quite problematic and purely theoretical. A DITM option is an option that has a strike significantly below (for a call) or above (for a put) the underlying asset price.

These options can be contrasted with DOTM options, which have no intrinsic value and also minimal extrinsic value (price close to zero).

To still have enough samples in each class we used quantiles of the training dataset moneyness to define the limits of each class as below, where q denotes the quantile function:

Class	Moneyness ($m = K/S_0$)
ITM	$q_{0.1}(m) < m < 0.95$
OTM	$1.05 < m < qu_{0.9}(m)$
DITM	$m < q_{0.1}(m)$
DOTM	$m > q_{0.9}(m)$
ATM	$m \in [0.95, 1.05]$

The model’s performance is correlated to the option moneyness which suggests building models accordingly (fig. 2.3). The global model performs the best on deep-in-the-money options and in-the-money options. The predictions are more dispersed for out-of-the-money (OTM) and deep out of the money options (DOTM). We also checked that there is no relevant correlation with the maturity or volatility of the option by considering the scatter plots of errors with respect to both the volatility and the maturity globally and for each moneyness class (see example fig.2.4).

Although the models by class of moneyness yielded smaller errors than the global model (3% of the strike as a global mean absolute error on the test dataset), **this approach has not been retained as it favors overfitting and undermines the model’s generality.** It also depends on the definition of the different classes.



Figure 2.3 – Scatter plot of the predicted and target prices by moneyness

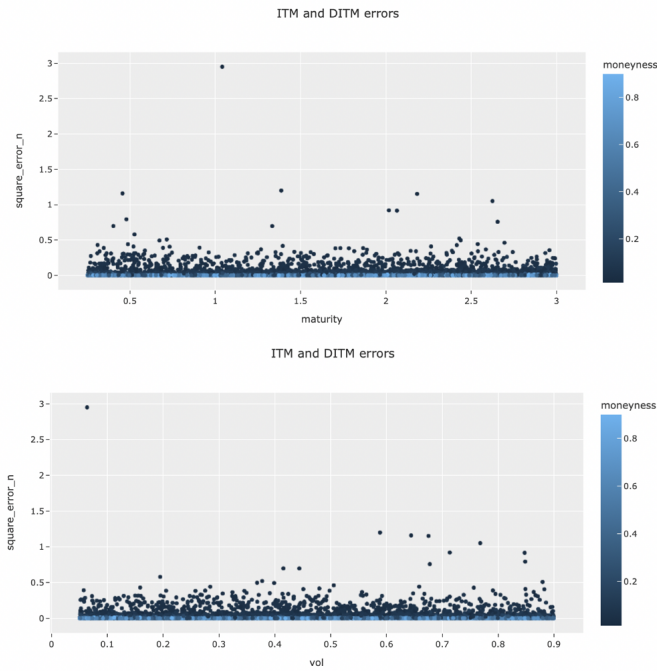


Figure 2.4 – Normalized error distribution by maturity and volatility

Black-Scholes call prices results: (dataset generated according to a fixed grid for the option’s moneyness)

Both the training and parameters’ gradient norm decrease efficiently (fig. 2.5).

The neural network predictions on both the training and test datasets are more accurate

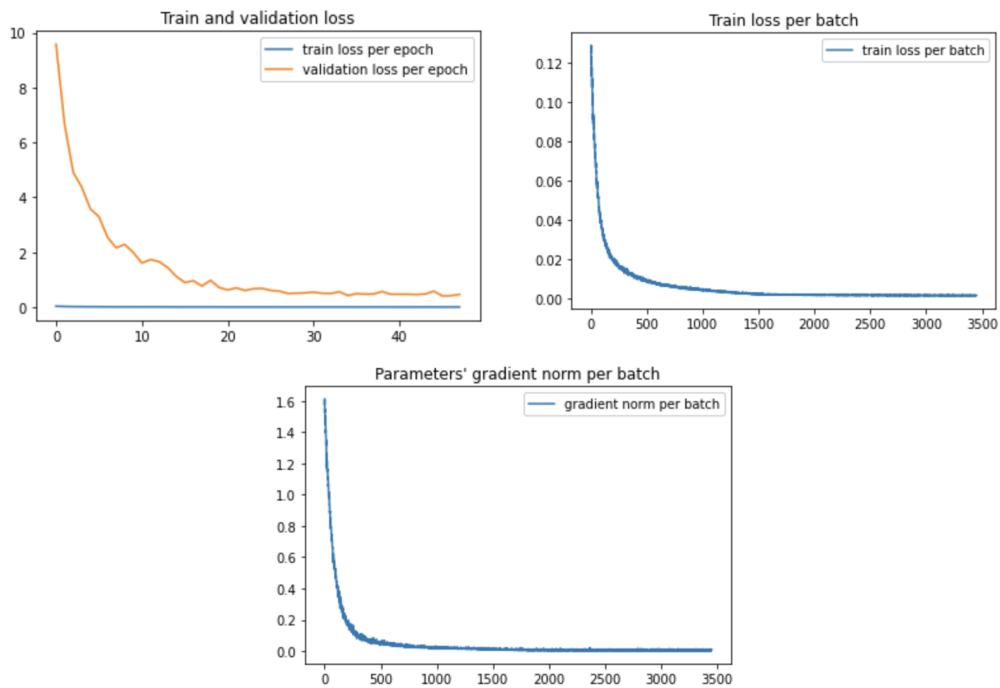


Figure 2.5 – Hidden layers= [100,100], batch size= 1024, epochs= 50, batch normalization=no, learning rate= 10^{-3}

when trained with a fixed moneyiness grid. We report the results on an unoptimized model. The average absolute error drops to 3% of the strike while the average relative error decreases to 30% on the test dataset, as reported below (table 2.2). This is because we no longer have aberrant moneyiness values.

Metric	MAE	RMSE	MARE	RMRSE
Test dataset	0.028	0.040	19%	30%
Train dataset	0.028	0.037	18%	28%

Table 2.2 – Cross-validation metrics without hyperparameter tuning (Hidden layers=[100,100],batch size= 512, epochs= 50, batch normalization= no, learning rate= 10^{-3})

The optimized two-hidden-layer neural network has 100 neurons per layer, a batch size of 320, and has been trained on 250 epochs with a learning rate of 10^{-3} and batch normalization (table 2.3). In figure 2.6 we compare the predictions of the NN before and after hyperparameter tuning, to highlight the importance of this step. The average normalized error drops to 0.1% of the strike while the average relative error drops to 6% on the test dataset.

Metric	MAE	RMSE	MARE	RMRSE
Test dataset	10^{-3}	10^{-2}	2%	6%
Train dataset	$9e^{-4}$	10^{-2}	2%	7%

Table 2.3 – Cross-validation metrics on the training and test datasets (BS call prices with fixed moneyiness grid)

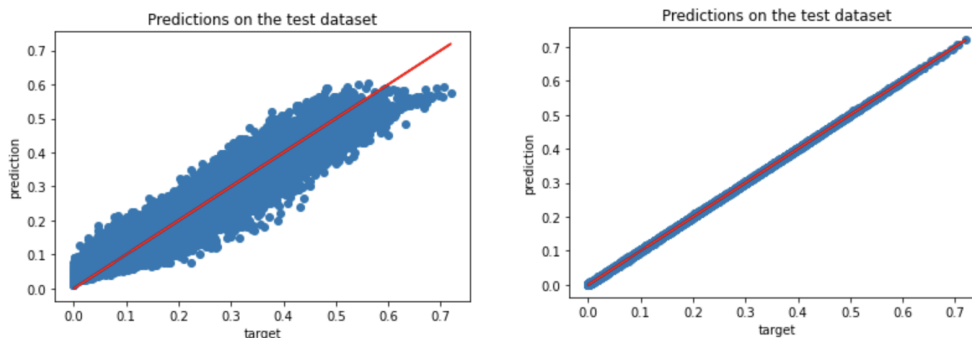


Figure 2.6 – (a) Hidden layers= [40,40], batch size= 1024, epochs= 50,learning rate= 10^{-4}
(b) Hidden layers= [40,40], batch size= 320, epochs= 250,learning rate= 10^{-3}

Remark: To highlight the impact of the learning rate on the loss’s convergence, we reported the train and validation losses as well as the parameters’ gradient norm for three identical NN with the only difference being the learning rate (fig. 2.7). A large learning rate usually causes the loss to drop quickly but prevents its convergence. A smaller learning rate induces a longer time for the NN to converge.

Furthermore, when loss pics are observed at the end of the training, it can be helpful to use a learning rate scheduler to updates the static learning rate used according to a given schedule.

The loss pics are also more common when using smaller batch sizes, or more sophisticated architectures (adding hidden layers).

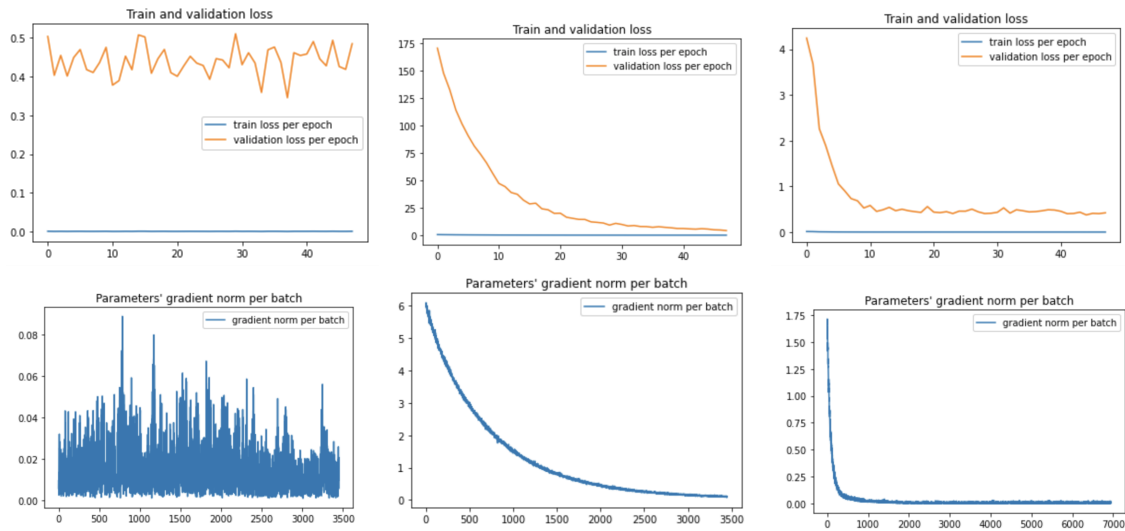


Figure 2.7 – Impact of the learning rate on the convergence of the loss: Hidden layers= [100,100], batch size= 1024, epochs= 50, learning rate: (a) 10^{-2} , (b) 10^{-5} , (c) 10^{-4}

After the neural network has proved to correctly price European vanilla options, we tested the approach on exotics. We report the results for the pricing of an Asian call option.

Black-Scholes Asian call prices prediction

Previously the prices were normalized by the option strike, but not in this case. This explains the scale difference for both the predictions and absolute errors (fig.2.8).

A two-hidden-layer FFNN with 100 neurons per layer trained for 500 epochs with batches of size 320 and a learning rate of 10^{-3} yields an average relative error of 7% on the test samples 3.1.

Metric	MAE	RMSE	MARE	RMRSE
Test dataset	1.47	2.0	8%	7%
Train dataset	1.26	1.8	7%	7%

Table 2.4 – Cross-validation metrics on the training and test datasets (BS asian call prices with fixed moneyness grid)

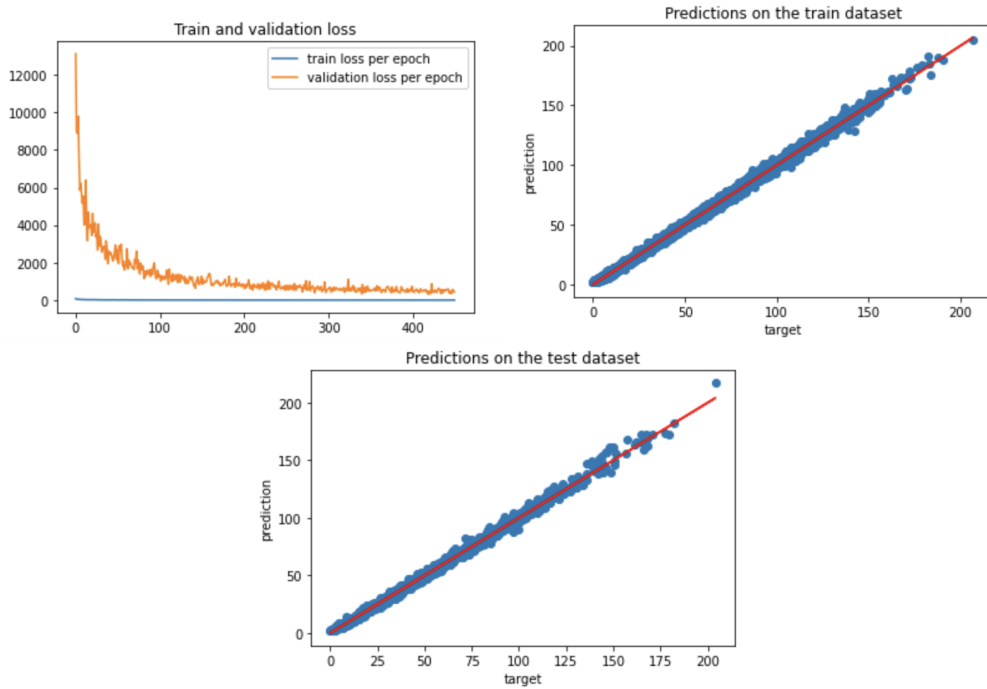


Figure 2.8 – Hidden layers= [100,100], batch size= 320, epochs= 500, batch normalization= no, learning rate= 10^{-3}

Heston surface call price prediction

Predicting Heston call prices, yielded comparable results (average relative error of 8%). Using a fixed moneyness grid significantly improved the model performance. This inspired us to directly map price surfaces on a fixed predefined grid for each given set of parameters (model parameters, S_0, r, q). In fact the NN now returns price surfaces rather than individual prices. We thought this approach, can help the model better capture the price surface regularity. This approach slightly improved the model's global accuracy. If we ignore the aberrant predictions in the short term, ie. for time to maturity inferior to 30 days, the average relative error for Heston prices prediction becomes below 2% on the test dataset.

Average Relative Heston Price Error

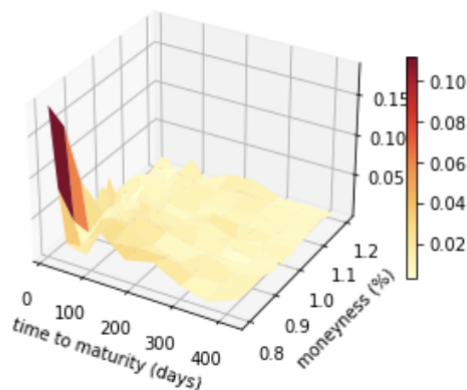


Figure 2.9 – Average relative prediction error on the test dataset (Heston call price)

We compare in the figure below (fig.2.10) the call option price surface computing using *Quantlib*³ and the neural network approximation on a random test sample.

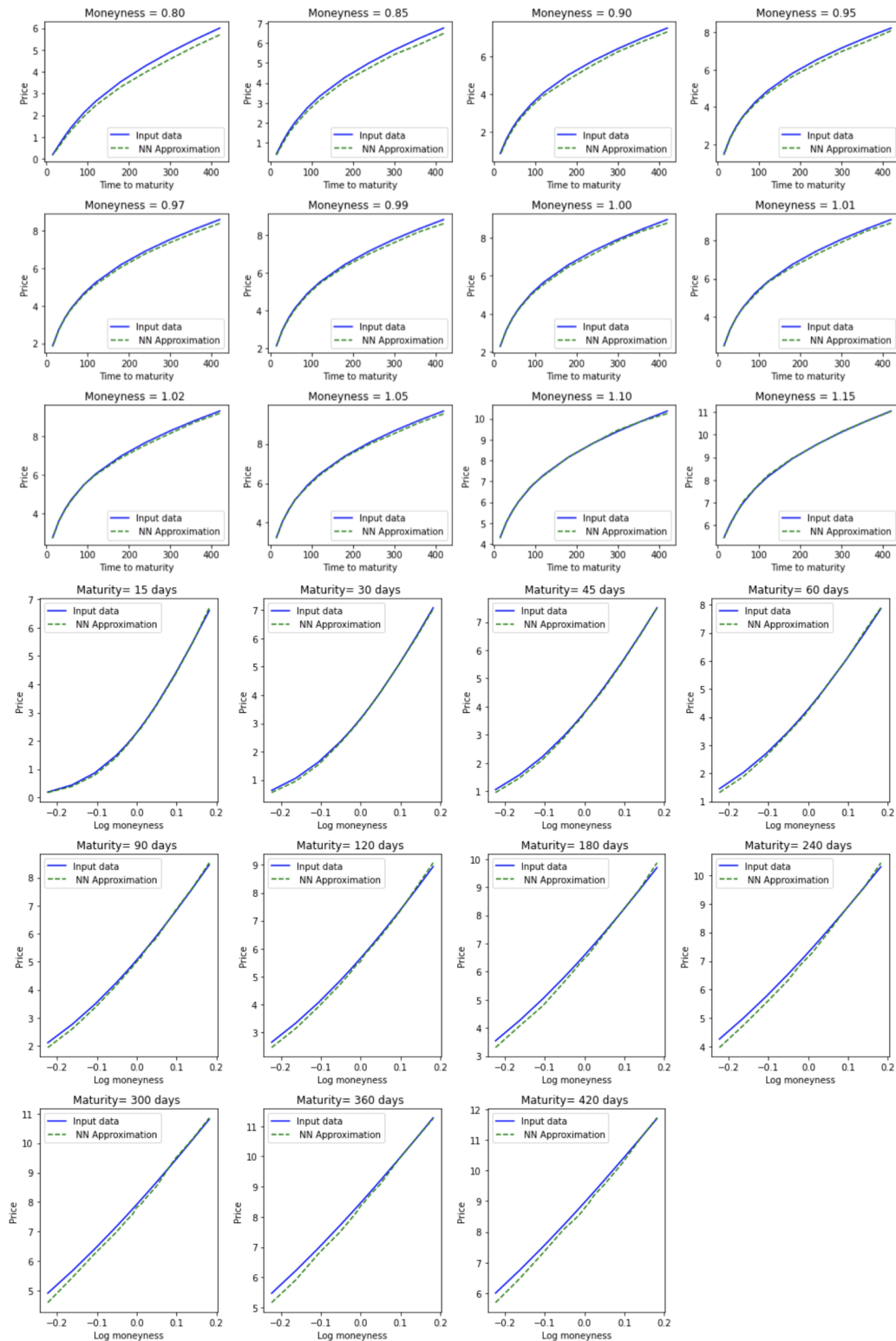


Figure 2.10 – Neural network approximation on a random test sample ($S_0 = 37.8, r = 5.10^{-3}, q = 1.2.10^{-2}, v_0 = 0.6, \kappa = 4.2, \theta = 0.27, \sigma = 0.21, \rho = -0.68$)

³A free open-source library for quantitative finance accessible on www.quantlib.org

Hyperparameter tuning

We used an exponential activation function in the last layer to force the predictions (prices) to be positive. For the hidden layers, we compared Sigmoid and ReLu. Relu yielded better accuracy and was more likely to overcome the gradient vanishing problem and hence induced a better convergence. It is also important to choose a twice differentiable activation function. In fact, option Greeks are as important as the prices themselves. Therefore, ensuring that the neural network prices are at least twice differentiable makes computing greeks a trivial task.

We also used batch normalization which refers to scaling the output of a layer by subtracting the batch mean and dividing by its standard deviation. This technique usually speeds up the training and can be useful for higher dimension problems [24]. However, it did not improve the model accuracy in this regression problem unlike when dealing with sparse features in images where this operation usually yields significant improvements.

2.3.3 Limits of the neural network approach

During the training, we encountered two main problems:

- Non-convergence of the loss function: The loss function did not converge at first, we needed to normalize our inputs. Therefore we compared two normalization methods: transforming the inputs to the same interval typically $[-1, 1]$ and a standard scaling normalization, which has been retained. Moreover, the neural network did not converge to a global minimum. The loss was stuck at a relatively high level. Adjusting the learning rate and using proper random initialization for the weights helped solve the problem. It's important to note that this problem is common for non-convex optimization. In particular, deep neural networks are non-convex. Therefore, stochastic gradient descent does not necessarily converge to a global minimum.
- Overfitting: to avoid overfitting we limited the number of nodes and thus of trainable parameters. We tried adding a dropout function in the hidden layers. This operation deactivates a random set of neurons in the layer, which forces the model to learn more robust features. Moreover, we adopted k-fold cross-validation for hyper-parameters tuning and only saved the model if it yielded an improvement in the validation mean loss.

One of the main critics of this approach is the fact that the model cannot by default, guarantee **no-arbitrage** principle.

Many questions become legit. Is the model coherent with the financial reality and constraints? For example, do we respect the put and call parity for European options? To what extent constraints like the convexity of put and call prices are respected?

Our first idea to improve this point was to predict price surfaces rather than individual prices to help the model capture the regularity of the function. But this does not guarantee no-arbitrage it only ensures more smooth surface predictions. A classical approach consists of regularization. In fact, the option pricing theory provides necessary and sufficient conditions for European options to be arbitrage-free. These conditions, in the case of call options $C = C(S, K, T, r, q)$, translate into:

$$C \geq 0, \quad \partial C / \partial T > 0, \quad \partial C / \partial K < 0, \quad \partial^2 C / \partial K^2 > 0$$

These constraints are added as new penalty terms to the per-sample loss function, which is minimized during the training. In other words, predictions that do not respect the no-arbitrage conditions are penalized with a higher associated loss.

2.4 Unsupervised approach

This approach will be briefly presented, as it has been briefly explored but not retained and developed in this report. Since it was not in line with the next step (deep calibration) The main and common idea in most approaches using NN to solve stochastic differential equations (SDEs) and (PDEs) is that we do not need to compute a target output y and train the NN to map the input parameters to the output. Therefore these approaches are categorized as "unsupervised". They are all based on the minimization of a loss function **conveniently defined** based on the PDE.

The deep Galerkin method

When exploring the different deep pricing paradigms mentioned in the literature, we were intrigued by this approach and tried to implement the deep Galerkin method (DGM)[27] for the pricing of American options.

This algorithm is particularly suited for solving high-dimensional PDEs. The authors choose to apply this general approach to the particular case of American options.⁴ because the associated pricing PDE belongs to the class of high-dimensional free boundary PDEs. This class of PDEs provides a unique opportunity to assess neural networks' accuracy on a class of high-dimensional PDEs with no semi-analytic solutions⁵. Since in this class, the error bounds can be calculated for any approximate solution.

Galerkin methods refer to a wide class of computational methods that seek a reduced-form solution to a given PDE. The reduced-form solution is a linear combination of basis functions. In this paper, the deep Galerkin method (DGM) is based on the same idea but uses a NN instead of a linear combination of basis functions.

The NN is trained to satisfy the differential operator, the initial condition, and boundary conditions based on the PDE, on randomly sampled points of the parameters space of interest.

2.5 Deep pricing conclusion

The deep pricing approach is promising as it can correctly learn and map the pricing function on a given parameters space of interest and instantaneously predict the price surface for different market settings ("fast surface"). Once the NN is trained, computationally expensive MC simulations can be discarded.

We report computing time in a simple MC simulation case with 100 steps and 10 000 scenarios. This table is for illustration purposes only since MC computing time may vary.

⁴An American option is a financial derivative on a set of stocks allowing the holder to exercise their rights at any time before and including the expiration date. The problem's dimension (PDE dimension) equals the number of stocks in the portfolio.

⁵This means that the problem is irreducible (its dimension can't be reduced). In fact, if a semi-analytic solution existed, this means that the PDE has been transformed into a lower-dimensional equation

However, the prediction task boils down to evaluating the neural network function in the input vector with $\mathcal{O}(1)$ complexity.

Method	Asian call BS	Lookback call BS	Asian call Heston	Lookback call Heston
Simple MC	0.27 – 0.3 seconds per price on average		0.72 – 0.75 seconds per price on average	
NN	instantaneous regardless of the size of the vector to be predicted (< 0.1 seconds)			

We should finally highlight that in this approach, the sampling method is crucial. It is the case for both the supervised and unsupervised methods. How the training samples are generated/chosen is the most important factor in determining the accuracy of the method. One could criticize the fact that the training samples, generated randomly, do not reproduce the historical market prices representativity. But this can be an advantage in stressed scenarios for example, where market settings are generally not historical, but rather unforeseen rare scenarios.

3

Deep calibration: option pricing models calibration

3.1 Deep calibration paradigm

3.1.1 Problem overview

Model calibration is an optimization problem. Given a price surface/volatility surface¹, it consists in finding model parameters such that the model prices best approximate the given surface according to an appropriate metric. Often the \mathbb{L}^2 .

In the literature, this problem is generally formulated using the implied volatility surface (IVS) rather than the price surface. The optimization problem is generally solved using iterative optimizers such as Broyden–Fletcher–Goldfarb–Shanno (BFGS), differential evolution (DE), or Levenberg-Marquardt (LM). These optimizers rely on the repetitive evaluation of the cost function (the function to minimize) and therefore can be computationally intensive when for instance, a Monte Carlo simulation is needed for each evaluation. This is typically the case for stochastic volatility models.

As explained in the previous chapter, fully connected neural networks are powerful regression tools able to approximate any continuous real function arbitrarily well. NN, as explained previously, can be used to predict instantaneously and accurately a price or a volatility surface. We call this step "fast surface".

The idea behind deep calibration is to leverage this capability to tackle models calibration differently. In this paradigm, the calibration bottleneck due to the computationally inefficient pricing and assessment of the cost function can easily be lifted.

¹Generally, we consider market volatility surface

3.1.2 Problem's reformulation

Let's first formulate our problem clearly to understand the different approaches to tackle it: We have prices of financial products generated by some financial model parameterized by parameter vector $\theta = (\theta_{observables}, \theta_{model}) \in \Theta$, where Θ is the parameter space or our space of interest.

For example in the Black-Scholes framework, using the previously introduced notations, the parameter vector is $\theta_{BS} = (\theta_{observables} = (S_0, r, q), \theta_{model} = (\sigma))$. Similarly, in the Heston model: $\theta_{he} = (\theta_{observables} = (S_0, r, q), \theta_{model} = (\kappa, \theta, \rho, \sigma, v_0))$.

Given a set of prices (model-simulated prices or market prices), we seek the most appropriate model parameters that would generate such a price surface. "Most appropriate" is defined based on a given loss function typically \mathbb{L}^2 for a regression problem. This loss reflects the distance of the original price structure to the predicted price structure.

$$\text{We minimize: } \theta \mapsto \mathbb{L}_{\text{data}}^2(\theta), \quad \text{for } \theta \in \Theta.$$

The calibration problem, hence consists in finding possible minimizer θ^* and more specifically θ_{model}^* .

Many approaches are to consider in solving this problem **from a machine learning point of view**:

- **Directly learning the map from the data (prices and observables) to the model parameters.**

$$\text{data} \mapsto \theta_{model}^*$$

This is typically the approach used by Andres Hernandez (2017) in [15], a reference paper on the subject. He used a FFNN to calibrate a Hull-White interest rate model by directly learning the inverse map from the data.

$$dr(t) = [\beta(t) - a(t)r(t)]dt + \sigma dW(t)$$

The network is trained with implied volatility surfaces as input and model parameters (a, σ) as output ³.

This approach is quite ambitious. In fact, we do not know if the universal approximation theorem still applies as the inverse map could be a discontinuous function. Therefore, the neural network might learn correctly the map on the given training samples, but can't generalize on out-of-sample data as stated in [10] and [29]. This approach is hence not suitable for stochastic volatility models calibration, because they are generally over-parametrized compared to the Hull-White model which has only two parameters to effectively calibrate.

- **Learning the pricing map $\theta \mapsto \text{data}$** , which is often quite regular, with artificially generated data **and then replacing the pricing functional with the learned "fast surface" to solve the inverse problem** using the classical minimization technics [4].

² $\mathbb{L}^2_{\text{data}}(\theta) = \sum_{i=1}^n (y_{\text{data}} - y_{\text{predicted}(\theta)})^2$

³The term $\beta(t)$ is then deduced for the forward rate yield curve: $\beta(t) = \frac{\partial f(0,t)}{\partial T} + af(0,t) + \frac{\sigma^2}{2a} (1 - e^{-2at})$ where $f(t,T)$ is the instantaneous forward rate.

- **Learning the pricing map ("fast surface") and leveraging the fact that the NN is a nested function of the input parameters whose gradient is known analytically.** The gradient is known not only for the weights and biases but also for the input parameters (model parameters + observables parameters) thanks to the backpropagation principle. Hence we can apply the gradient descent algorithm(1) to minimize the \mathbb{L}^2 loss with respect to the model parameters.

$$\min_{\theta_{model}} (\mathbb{L}^2(\theta))$$

3.1.3 Numerical implementation

Implied volatility

The implied volatility $\sigma_{iv}(K, T, S_0)$ is such that, given the price of a call option (market or model price) $C(S_0, K, T)$ and a zero coupon bond of maturity T , $B_{T,T} = \exp(rT)$, we have the following equation:

$$C(S_0, K, T) = C_{BS}(S_0, K, \sigma_{iv}(K, T; S_0), T)$$

The implied volatility surface is a central notion in the option pricing theory. By definition, it is obtained from market prices of liquid European call and put options on the S&P 500 index. But this notion is easily extended to model prices.

Why do we need to compute implied volatilities ?

Although the approach we are about to develop can be directly applied using the price surface. Generally, when we calibrate a model to market data, we minimize the model options volatilities against market options volatilities. We want the model to generate consistent and robust prices. Therefore, we rely on volatilities. Otherwise, market participants can take advantage by quoting inconsistent prices.

Often the model we try to calibrate computes an option price, not the implied volatility. This is typically the case when it comes to the calibration of a stochastic volatility model or a volatility surface parameterization.

Consequently, a fast method to invert option prices into the associated implied volatility is necessary [1].

Commonly used inversion algorithms such as Newton's algorithm, the differential evolution algorithm, or Brent solver are extremely long. In this implementation, we used an algorithm based on Li and Lee (2011)'s SOR⁴ algorithm and a good initialization introduced by Stefanica and Radoicic (2017) in their paper An Explicit Implied Volatility Formula [10]. "Their formula is simple, fast to compute and results in an implied volatility guess with a relative error of less than 10% " as described in a technical blog⁵ and already implemented on *GitHub*.

⁴An Adaptive Successive Over-Relaxation Method for Computing the Black-Scholes Implied Volatility

⁵<https://chasethedevil.github.io/post/fast-and-accurate-implied-volatility-solver/>

3.1.4 Methodology description

The first step consists in training the "fast surface" using a neural network. For that, we artificially generated the training dataset: the parameters are sampled from uniform distributions whose extremes are defined a priori (This defines our space of interest).

We also used a fixed grid for both option's moneyness ($m = \frac{S_0}{K}$) and time to maturity (ttm). Then, we generate the model price associated with each set of parameters. This step is done by diffusing the price dynamics and calculating the expected payoff using MC methods. In practice, we used the QuantLib Python library[2] which has an efficient MC simulation implementation. Implied volatilities are then retrieved through the Dan Stefanica and Rados Radoicic algorithm.

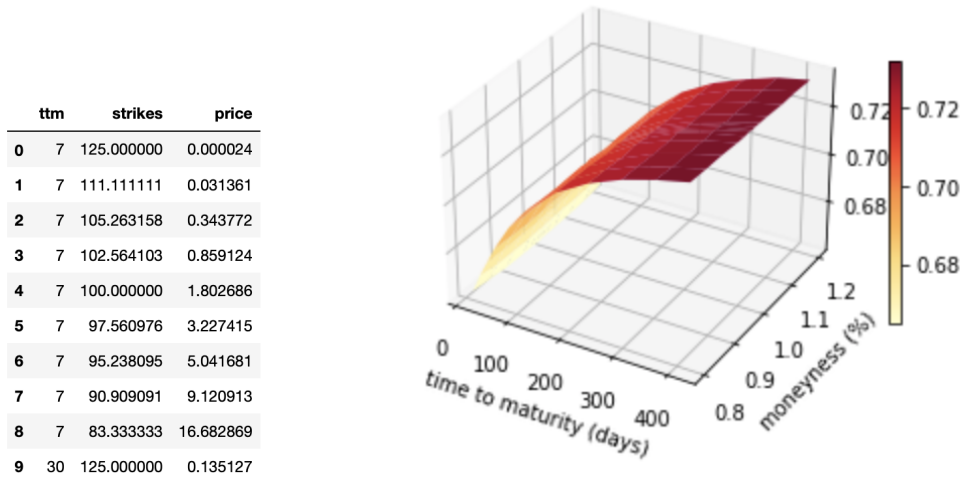


Figure 3.1 – (a) Subset of a fixed-grid generated price surface (b) example of Heston IVS from the training dataset

We then train a NN to map the volatility surface to the input parameters on the given space of interest. The following figures (fig.3.2 and fig.3.3) show some metrics for the relative errors between the neural network predictions and the target values averaged across the test dataset samples. Namely, the average, standard deviation and maximum. The overall performance is uniform, except for short term options which is quite surprising.

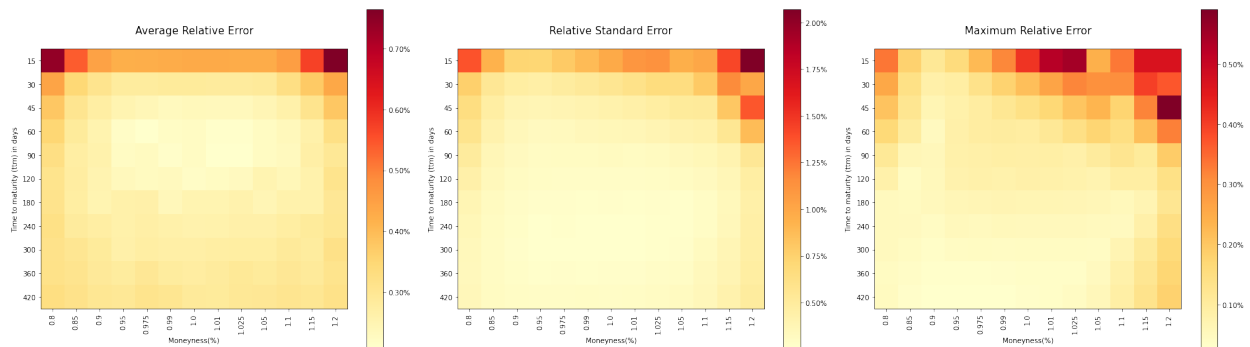


Figure 3.2 – Average relative errors on the test dataset

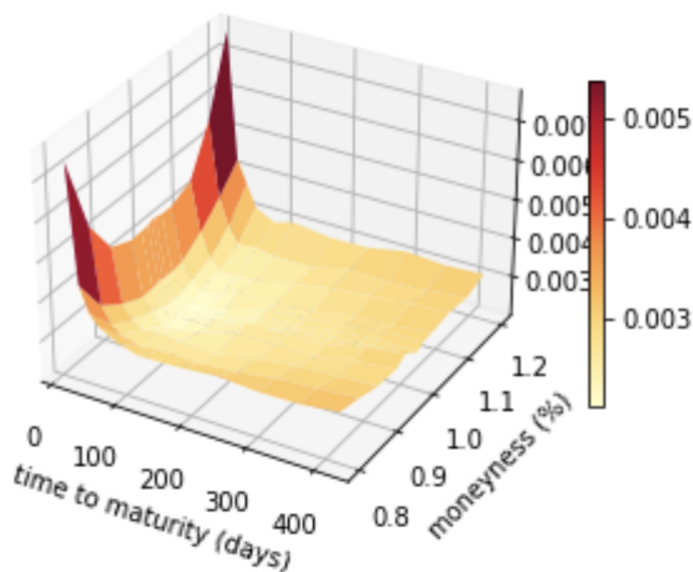


Figure 3.3 – Average relative errors on the test dataset

Once the “fast surface” NN is trained and tuned, we use it to calibrate to model. To test the reliability of the deep calibration approach we generated Heston implied volatility surfaces for dates from 31/12/2020 to 30/01/2121 using *Quantlib* and random parameters. The model parameters are known and saved. We mask this information and apply two different approaches to predict it. We then compare the IVS generated by the predicted parameters and the target IVS to assess the approach’s performance. On the next page, we summarize the two tested approaches (Algorithms 4 and 5).

The second approach consists in learning the pricing map (fast surface) and leveraging the fact that the NN is a nested function of the input parameters whose gradient is known analytically. The gradient is known not only for the weights and biases but also for the input parameters (model parameters + observables parameters) thanks to the backpropagation principle. Hence we can apply the gradient descent algorithm(1) to minimize the \mathbb{L}^2 loss with respect to the model parameters. The approach is described in Algorithm 5 :

$$\min_{\theta_{model}} (\mathbb{L}^2(\theta))$$

Algorithm 4 First approach: Replace the pricing functional with the NN solution when minimizing the loss

- **Construction of "fast surface":**

- Generate: 30 000 samples of $\theta = (\theta_{observables}, \theta_{model})$ (inputs) according to a given sampling method.

- For each sample, define the model to diffuse with parameters θ_{model} . We implemented this approach in *Quantlib* and therefore it can be adapted to other diffusions already implemented in the library, typically local volatility models.

- Compute for each input the price surface (on the grid $m \times ttm$). This step is done using Quantlib's pricing engine based on MC simulations. And the code can easily be adapted to other options by adapting the following model's description and pricing functions.

```
{model: ql.HestonModel;  
  diffusion process: ql.HestonProcess;  
  pricing engine : ql.AnalyticHestonEngine  
  priced option  : ql.Option.Call}
```

- Compute the implied volatilities

- Train a NN network to learn the pricing map (replicate the pricing model).

- **Calibration**

- Define the calibration bounds (an interval slightly larger than the one used to generate the parameters).

- Given a target IVS surface, define the loss function as the difference between the target surface and the surface predicted using the trained network ("fast surface").

- Minimize the loss function using global search optimization algorithms for multivariate functions, typically the differential evolution algorithm⁶, sequential least square quadratic programming (SSLQP) and quasi-Newton methods available in *scipy.optimize*. Consider as an initial guess for the search initialization a random point in the calibration interval or its center.

Algorithm 5 Second approach: leverage the fact that the trained NN is a nested function of the input parameters whose gradient is known analytically to minimize the loss on model parameters

- **Construction of "fast surface":**

- **Calibration**

- Define the calibration bounds (an interval slightly larger than the one used to generate the parameters).

- Given a target IVS surface, define the loss function as the difference between the target surface and the surface predicted using the trained network ('fast surface').

- Minimize the loss function via a gradient descent type algorithm, where the loss's gradient with respect to the input can be expressed as a function of the NN's gradient with respect to the input. Consider as parameter initialization a random point in the calibration interval or its center to compute an initial value for the loss function.

3.2 Numerical results

3.2.1 Results

Both approaches are very quick to calibrate surfaces. We have tried several optimizers. The differential evolution algorithm seemed to globally yield the most accurate results. The original surfaces are shown in pale orange in figure 3.5 and figure 3.4. We represent the calibrated surface using the first approach, on the left. The surface obtained using the second method is represented on the right. The legend details the observable parameters and model parameters (historical or real) as well as the calibrated parameters (fig.3.4 and fig.3.5).

The first approach calibrates well to the target surface. The second method seems to have problems with the calibration of the kappa and more generally when there is a significant difference in scale between the parameters to be calibrated. Opting for different learning rates to update the input parameters in the gradient descent algorithm would probably help. Besides, robustness is definitely an important criterion for the performance of a calibration algorithm. To prevent significant shifts in the calibrated parameters, it is advisable for successive dates to initialize the search with the parameters that have been calibrated on the previous surface. This method's convergence strongly depends on the initialization. For that, it's not very robust. These limitations need to be further analyzed.

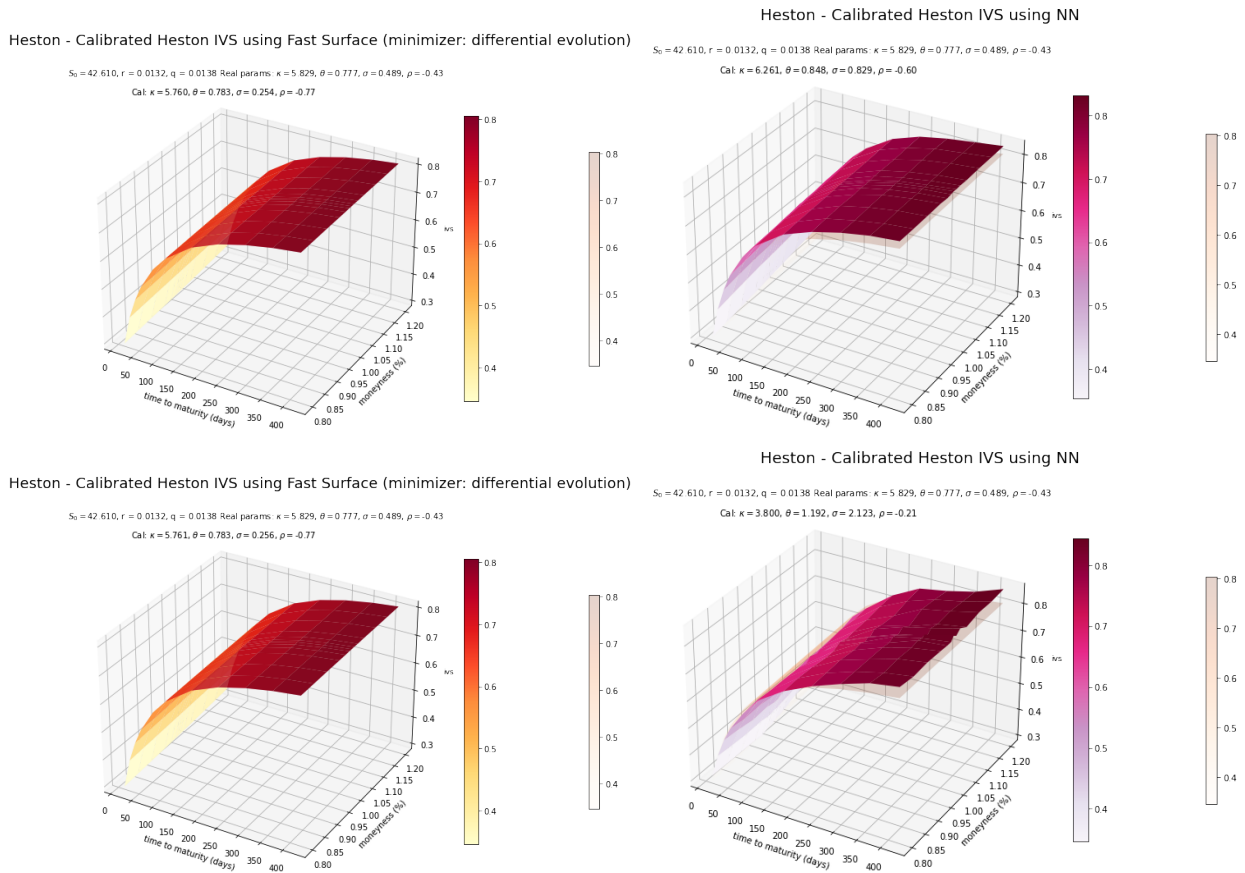


Figure 3.4 – Calibration results

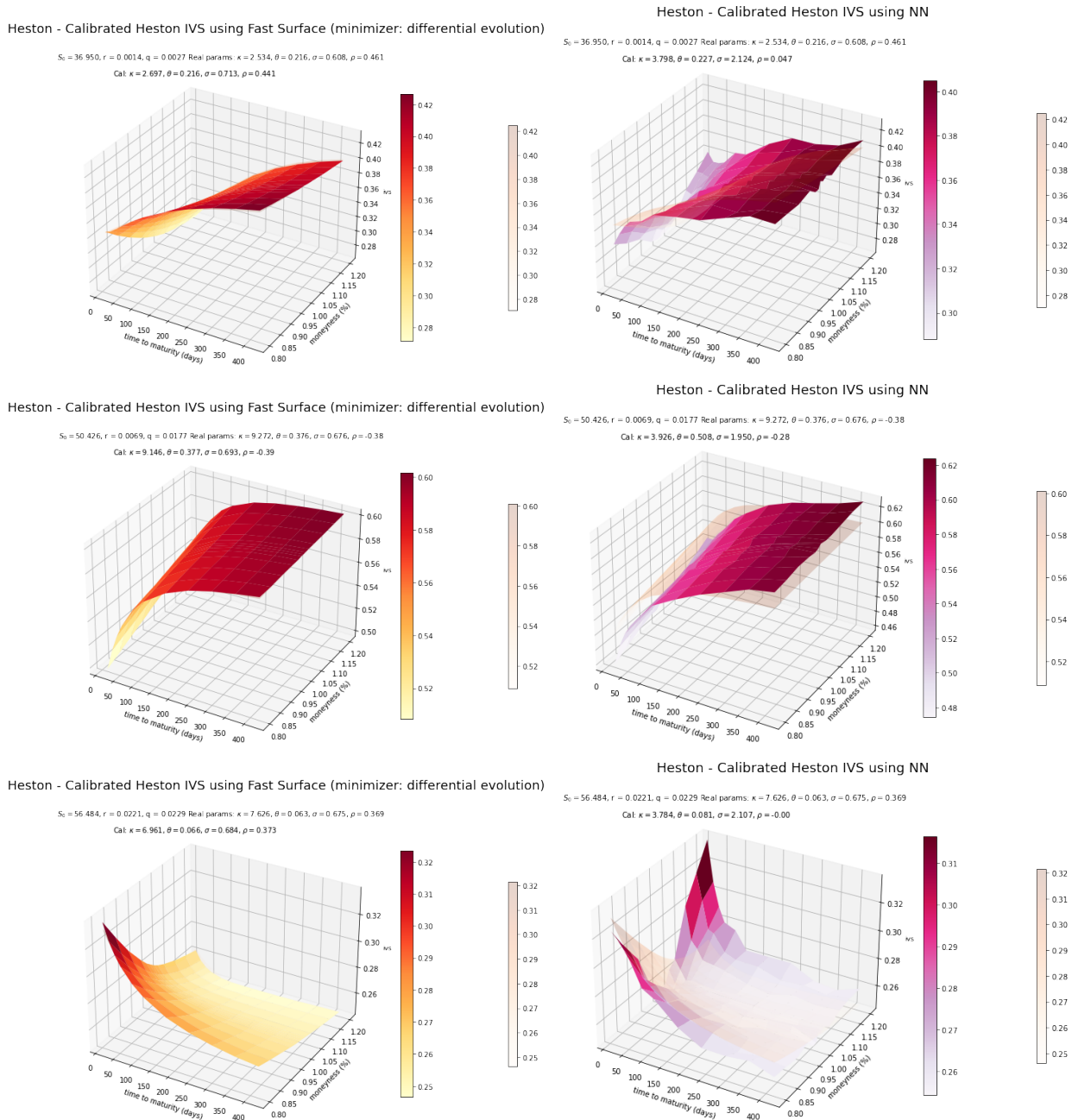


Figure 3.5 – Calibration results

3.2.2 Calibration results on market data (CAC 40)

For the final validation of the calibration algorithm, we used implied volatility surfaces for CAC 40 options available on Bloomberg API via the OVDV function and derived from European call and put prices. Particular attention was paid to the validation of the first approach, which proved to be the most accurate on synthetic data. Let's consider the implied volatility surface of the CAC 40 index on the 1st of June 2020 (fig.3.6) : The first approach yielded a RMRSE of 13% with as initialization the following set of parameters ($v_0 = 0.32, \kappa = 0.5, \theta = 0.05, \sigma = 0.45, \rho = -0.8$).

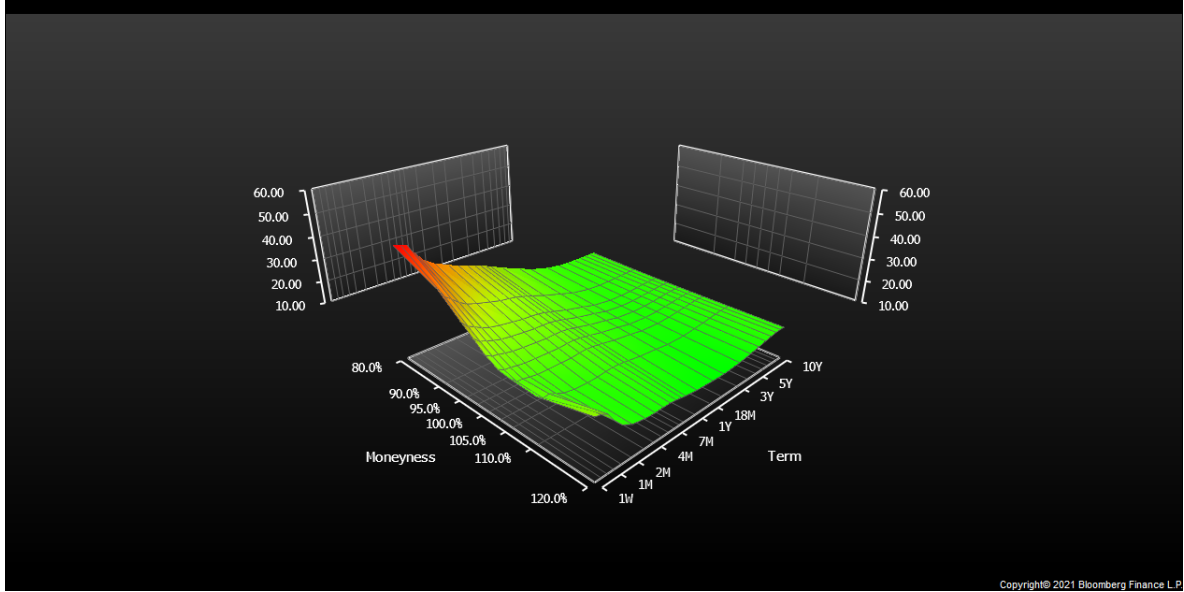


Figure 3.6 – IVS of the CAC 40 (01/06/2020) (source: Bloomberg: OVDV)

v_0	κ	θ	σ	ρ
0.294	2.771	0.086	1.445	-0.689

Table 3.1 – Heston parameters obtained by the second calibration approach (CAC 40 June 2020)

The calibration results seem consistent and show the following:

- A high speed of mean reversion κ and a high level of the volatility's volatility parameter *sigma* revealing a large fluctuations in the stock's variance dynamics.
- A level of mean reversion θ of 8.6% which is equivalent to a volatility of 29.32%.
- A negative correlation between the stock process and the variance process known as leverage effect.
- A relatively high level of volatility due to the Covid 19 crisis and consistent with the market. For instance, the VIX index was 28.23% on the 1st of June).

3.2.3 Critics and further improvements

Our approach has several limitations that can be improved: First, to assess the performance of the different trained models and choose the optimal hyperparameters, we averaged the relative errors on test samples generated according to the same grid points for both the option's moneyness and time to maturity. It would be more interesting to calculate the errors on other surface points. This new error will reflect, in addition to the training error, the model interpolation error.

Regarding the calibration approach, the method using the “fast surface” to optimize the loss function by traditional optimization algorithms is globally more efficient than the second approach, which we tried to formalize. Both methods save a considerable amount of time compared to traditional calibration methods. On average the calibration time of

a one-month history is 1.17 seconds for the first approach and 600ms for the second one) compared to more than 30 minutes for quantlib calibration using the differential evolution algorithm.

The second calibration approach, although attractive, still needs to be researched and improved, in particular by using an adaptive gradient descent method (adapting the learning rates using the order of magnitude of the parameters). Its convergence rate also depends on the initialization point of the search and could be further improved. It would also be interesting to rethink the convergence criterion and to find a compromise between the speed of convergence and the accuracy of the calibration. In our implementation, we used as a stopping criterion a maximum number of iterations limited to 2000 or a loss lower than 10^{-5} .

Within the framework of the internship and due to lack of data accessibility, this approach has been built only on synthetic data. Although it is highly recommended to build this approach on synthetic data ([5]) for the training phase of the NN, the approach should be validated on market data for the calibration phase. Therefore we used implied volatility surfaces for CAC 40 options to validate the approach. Besides, knowing the real parameters that generate the surfaces to be calibrated allowed us to better identify the flaws of each approach. We tried to apply these calibration techniques on a market volatility surface, the results were comparable, with higher accuracy for the first approach. The next step would be to test the approach and assess its performance on more market data.

As mentioned before, a commonly expressed concern would be the fact that the neural network used for the calibration has been trained on synthetic data, which is not necessarily representative of historic market data. We believe that using simulated data, properly sampled on a parameter space of “real interest” and consistent with the market mechanisms allows the model to be more general and exhaustive. In [5] the authors propose a way to improve the sampling method. They suggest a blended approach combining information from historic market data with a synthetic data generator algorithm to produce more market-consistent samples for the training. Furthermore, the model trained on synthetic datasets can easily be adapted to calibrate stress-test scenarios where the market settings are not necessarily historical but simulated.

3.3 Contributions of the neural network approach from a risk management point of view

3.3.1 More stringent regulatory constraints:the Fundamental Review of the Trading Book

Following the 2008 financial market crisis, the Basel Committee on Banking Supervision (BCBS) carried out a thorough review of market risk requirements and introduced several reforms to the Basel II market risk framework. The most urgent deficiencies were addressed by incremental directives while more structural flaws were tackled in the Fundamental Review of the Trading Book (FRTB) (fig.3.8).

The FRTB aims to address the potential systemic risks resulting from unmatched levels of market volatility. It ensures that both the standardized (SA)and internal model approaches (IMA) to market risk modeling, deliver adequate capital requirements estimation.

The ongoing COVID-19 pandemic has certainly emphasized how critical it is for financial institutions to hold sufficient capital buffers to protect against unexpected adverse market movements.

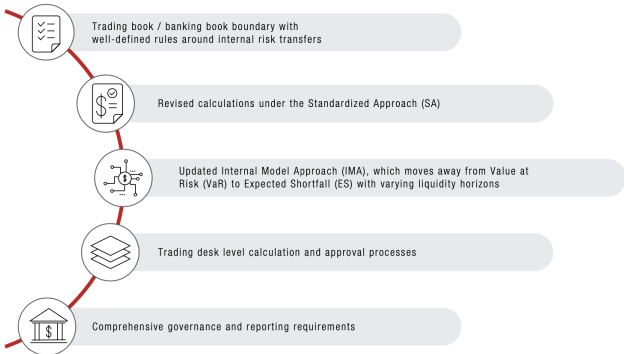


Figure 3.7 – FRTB regulatory overview [20]

Market risk capital components under FRTB

Under Basel II.5/III framework, market risk requires the calculation of VaR and Stressed VaR using a single methodology and liquidity horizon. The FRTB framework under IMA requires **multiple liquidity horizon for each risk category** and calculations of **expected shortfall (ES) and stressed expected shortfall**. This will *increase by more than a ten factor* banks computational requirements to estimate internal model market risk capital. Expected shortfall requires the revaluation of the entire portfolio daily within a window (often a two-year historical simulation). Stressed metrics adds a further level of complexity. These increasingly restrictive regulatory constraints are omnipresent. Without making a

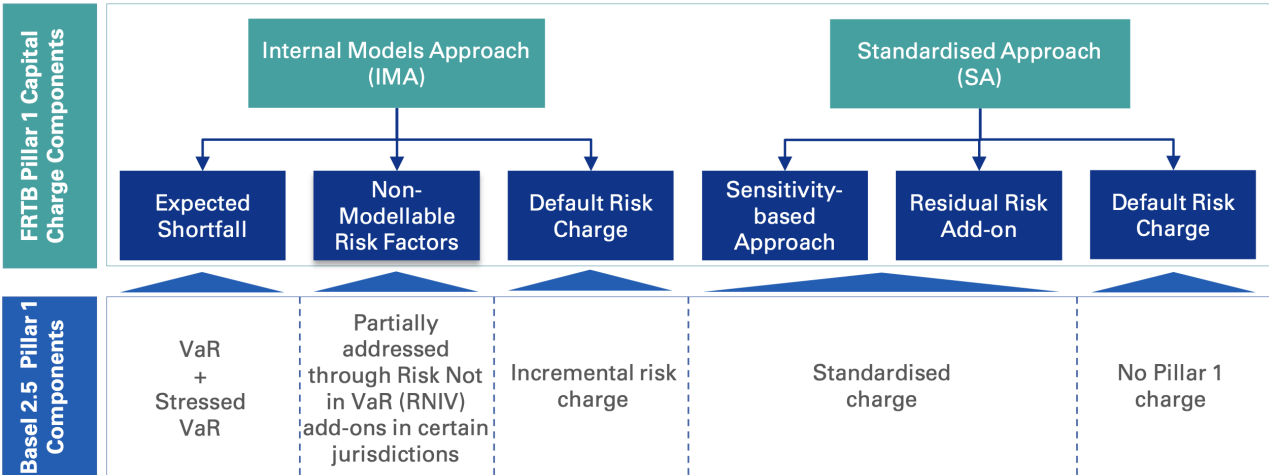


Figure 3.8 – Market risk capital components under FRTB [30]

long list, we should mention that this is still the case for counterparty risk with the Targeted Review of Internal Models (TRIM), where banks are required to have even more accurate models to assess counterparty risk. A more accurate model means more simulations. To sum up, stricter regulatory requirements introduce a significant computational challenge to which

banks need to adapt their current methods. The "fast surface" approach can significantly alleviate this computational burden. In fact, the major advantage of the NN approach over MC simulations lies in the significant time savings. Once the model is trained, evaluating the values of an entire portfolio is practically instantaneous and scales with the size of the network, not the size of the training data.

3.3.2 Model validation and stress-tests

The deep learning approach introduced in this work can also be applied to stress-tests. Since the network is trained on synthetic data, it does not introduce any historical bias to value a portfolio in an extreme market configuration.

Another potential application for the calibration is backtesting and model validation. For instance, for front-end pricers, in the context of P&L assessment, the trained network can learn to replicate the model used. Thus, it can find the calibrated model parameters from the generated data and validate their relevance in a monitoring/audit stage.

Calibrating and testing stochastic models across large databases becomes an easier task. For example, if it takes a second to compute option prices, testing a model that recalibrates daily over a thousand assets and two years of data would have taken a week. With a NN based approach it would take an hour. We finally highlight that we strongly advocate this separation of pricing and calibration in a neural network-based calibration framework. Although, it might be tempting to apply a neural network to directly estimate model parameters from market prices without using a pricing function based on a traditional model. This approach imposes several issues. As explained by Horvath et al.[17] these approaches might not align with the prevailing regulatory requirements due to their lack of interpretability. Furthermore, it is more difficult to prove their stability and robustness as required by regulators.

4

Pricing of Variable Annuity contracts

We can apply the deep pricing approach not only to options pricing but also to complex insurance policies. In particular, it applies to variable annuities (VA), often priced using the option pricing theory.

Variable annuities are long-term insurance policies that can be used by policyholders to accumulate wealth. These products are attractive investments and retirement vehicles because they contain guarantees such as death benefits and living benefits. In this chapter, we give a brief introduction to variable annuities and the computational challenges associated with their valuation as well as an application of the deep pricing approach to mitigate the pricing challenges.

4.1 Variable Annuities market: Context, trends, and challenges

4.1.1 A changing retirement context

The French pension system has operated since 1945 on a pay-as-you-go basis, with retirees' pensions financed by the contributions paid by the working population. However, the transition to baby-boomers retirement since the end of the 2000s has considerably deteriorated the system's sustainability.

The crisis partly arises from the inversion of the age pyramid and the increase in life expectancy, which significantly decrease the ratio of active contributors to retirees. As a result, the pay-as-you-go pension system has become in deficit. Projections for the coming decades are not optimistic as stated in the 2017 report from the Conseil d'Orientation des Retraites (COR).

This problem is not unique to France but affects a large number of countries. The age pyramids below show projections of the population age structure by 2050 in several countries according to the United Nations median scenario published in 2008. The top of the pyramid will be strongly lifted by longer life expectancy. While the United States and France have

maintained a relatively broad base, Italy and Germany bases have significantly shrunk, giving their pyramid a top shape.

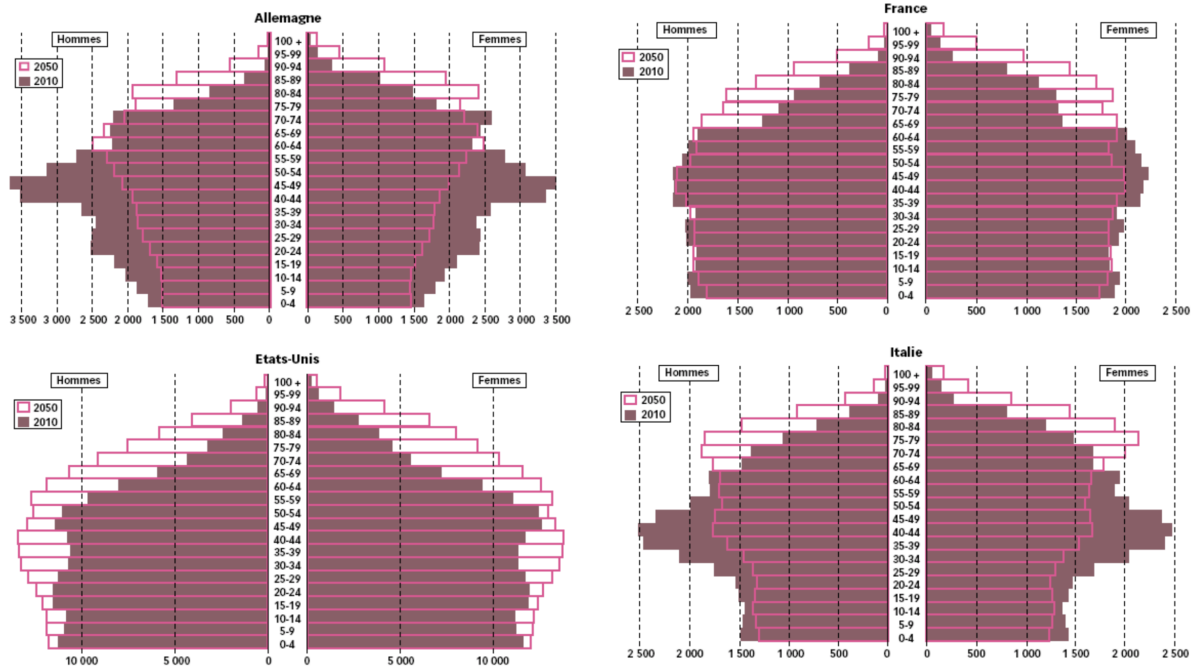


Figure 4.1 – Evolution of the age pyramid by 2050 according to the United Nations projections of 2008 (median scenario)

In France, several reforms were considered to introduce an additional part of income while reducing the compulsory benefit plans/Social Security to meet the changing needs. The new system follows the principle of capitalization, whereby contributions are saved in an individual account which will be used to finance the contributor’s retirement. Today, the French pension system is divided into three parts: the general compulsory basic scheme, the compulsory complementary scheme, and the supplementary scheme.

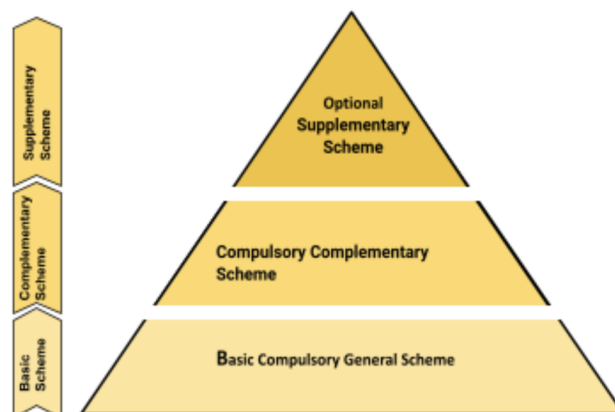


Figure 4.2 – Hierarchy of French pension schemes

Workers have become aware of the urge to build up their own supplementary retirement income, and life insurance is an interesting solution. It offers a range of products depending on the insured’s risk appetite. These products vary from conservative traditional risk-free

annuities with a relatively low guaranteed rate to risky mutual funds that might be quite profitable but with an important risk of capital alienation in the case of a market’s downturn.

The increased reliance on individual savings plans and the increasing life expectancy are leading to a growing need for longevity risk insurance (the risk of one surviving beyond their capital depletion) as well as return for retirees. In fact, a 1% greater return means roughly 10 extra years of spending. Retirees can not afford to depend on fixed-income (outing annuities) because it results in capital alienation. For instance, a 2% inflation rate over 25 years leads to a 40% reduction in purchasing power. Therefore, retirees need equity markets investments with guaranteed income.

In this context, Variable Annuities were introduced as a reasonable mid-point between mutual funds with both high expected performance and high risk and fixed annuities which are almost risk-free but with relatively low expected performance. In fact, variable annuities offer equity market participation with attractive protection against downside market risk. They hence make appealing tax-deferred retirement vehicles.

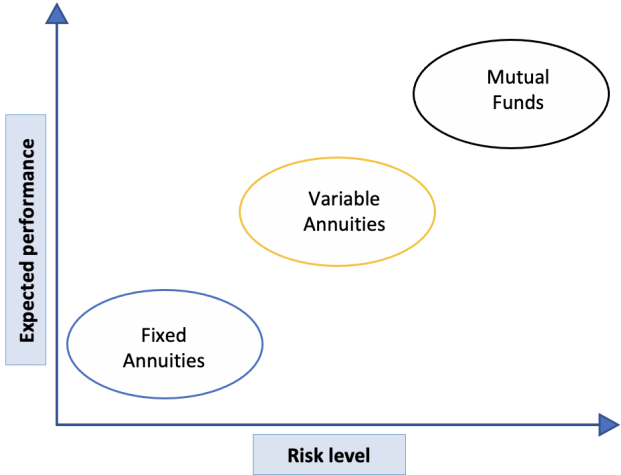


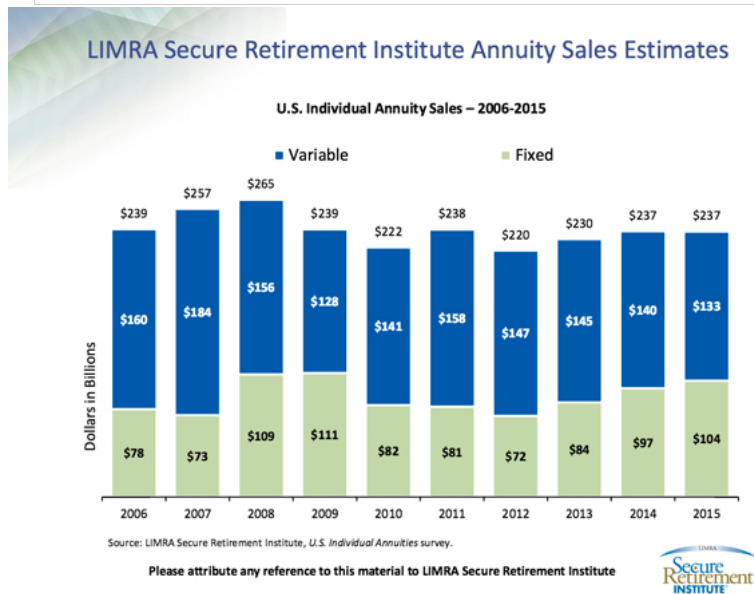
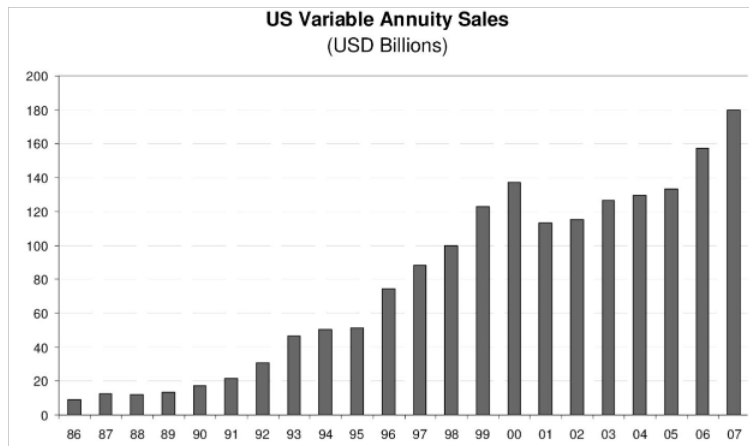
Figure 4.3 – Overview of life-insurance products

4.1.2 Variable Annuities market trends and challenges

Introduced in the early 1990s in the United States, variable annuities have known a huge success at the end of 1990s and early 2000s not only in the United States but also in several markets around the world, mainly Japan, Canada, and the UK.

While the 2001 crisis had an accelerating effect on Variable Annuities with investors looking for guarantees against financial markets downturn risk, the 2008 financial crisis harmed sales and generated significant losses for insurers. In 2008, the Life, Savings, and Pensions business results fell sharply, due to the increase in Variable Annuities hedging costs.

Since the economic crisis of 2008, their popularity has been fluctuating and trending downward. But sales are slowly recovering as shown in the following graphic and confirmed in the Life Insurance and Market Research Association (LIMRA) latest report. VA sales in the U.S reached \$93.4 billion in the first three quarters of 2021, 32% higher than the previous year. And the third-quarter sales represented 49% of the total annuity market in the U.S, the highest level since 2018. We should highlight that this product line represents a significant business for insurers. For example, it stands for \$1.7 trillion assets under management in



Year	Variable	Fixed	Total	
			Amount	Percent change from prior year
2016	\$104.7	\$117.4	\$222.1	5.8%
2017	98.2	105.3	203.5	-8.4
2018	100.2	133.6	233.8	14.9
2019	101.9	139.8	241.7	3.4
2020	98.6	120.5	219.1	-9.4

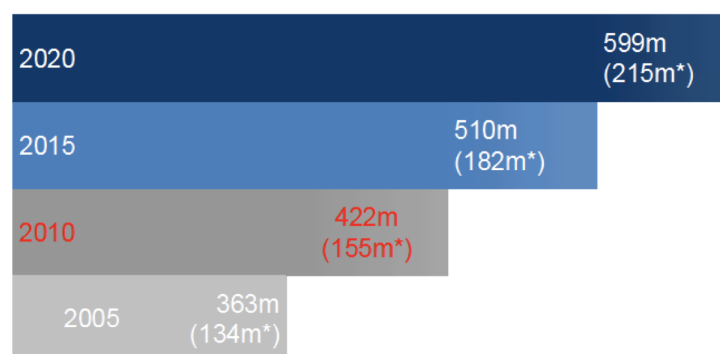
Figure 4.4 – U.S. Individual Annuities Sales Trends between 1986-2020 (\$ Billions), LIMRA.

2017 in the U.S, the world’s largest Variable Annuity market (Insured Retirement Institute 2018). This is comparable to the size of the hedge fund industry.

Although, the market is still dawning in Non-Japan Asia and Europe, it represents a huge business growth opportunity. The region presents high levels of economic activity thus an increasing level of personal wealth and propensity to save. Moreover, population aged beyond 60 years structure evolutions, reveal a significant and growing retirement opportunity as shown on the following chart.

Although Variable Annuities offer advantageous guarantees, they didn’t meet success in

Projected Population aged 60+ in Asia:
rapidly changing, wealthier, smaller and older households



*excluding China and India

Figure 4.5 – Projected population aged 60+ in Asia, AXA GIE, 2017

France. Two main factors explain policyholders' lack of appetite for these policies. First, as presented in the first section, the French Social Security system requires contributions to the basic pension general scheme (1st pillar), to the complementary schemes AGIRC, ARRCO, and company pension plans (2nd pillar). This leaves few workers who can afford to subscribe to Variable Annuities, as these products require a considerable investment with premiums typically of several tens of thousands of euros.

Second, in the 2000s, when variable annuities were booming in the U.S and Japan, euro funds still offered high guaranteed rates and attractive returns.

To sum up, the limited target group on the French market, the high cost of the guarantees, the complexity of managing large VA portfolios, and the numerous tax-efficient contracts already on the market are the main reasons behind the non-development of the market in France. In addition, Axa, who was the first to propose these policies in the U.S, suffered large losses after the subprime crisis. The complexity of the underlying guarantees and the limitations of the commonly used methods to efficiently manage large VA portfolios hindered successful sales and discouraged the competitors to enter the market. Moreover, most reinsurance companies exited the VA market after the crisis, and insurers were required to increase VA reserves. In fact, hedging programs have been strengthened after the crisis, leading to a growing need for computational efficiency.

4.2 Variable Annuities: presentation and definitions

4.2.1 Presentation of Variable Annuities contracts

Variable Annuities (VA) are capital guarantees that provide benefits in the event of life or death on contracts invested in Units of Account (UA) regardless of market trends. Different guarantees can be added to protect the investment. The policyholder pays a single premium at the subscription of the policy of at least several thousand euros or makes regular payments. The insurer suggests a diverse combination of unit-linked investments, typically mutual funds that invest in (equities, bonds, money market instruments, etc..) depending on the policyholder's risk aversion.

Variable Annuities are considered as tax-deferred retirement products allowing the policyholder to receive a complementary income at retirement. It means one pays no taxes on the gains from the contract until withdrawal, where ordinary income tax applies. These policies are hence taxed in the same way as traditional life insurance policies.¹ But differ in two main ways:

- **Surrenderability:** in contrast to other pension contracts, variable annuities can be surrendered at any time. If the surrender is made before the date defined in the contract, guarantees do not apply.
- **Fees:** Due to their guarantees, the costs of these contracts are much higher than traditional life insurance policies. They are almost always subject to a 3 to 5% fee on payments.

The accumulation phase and the payment phase are the two phases of a variable annuity contract.

During the accumulation phase, the client's savings follow the performance of financial markets. The capital invested is accessible at any time, but subject to a redemption fee. During this phase, the contract's guarantees, specifically the minimum amounts the policyholder should be receiving during the payment phase are revalued according to different revaluation mechanisms stated in the contract. These mechanisms will be detailed in the next section.

During the payout phase, the revalued savings are distributed to the beneficiary of the contract in the form of a single capital (lump sum) or in the form of annuities according to underlying guarantees and the performance of the selected funds. For instance, in France, the guarantee offered for classic unit-linked contracts is generally the floor guarantee. During the payment phase, the beneficiary of the contract receives the maximum between the current value of the savings and the floor amount. The latter is the sum of the payments made by the policyholder net of withdrawals.

After understanding the life of variable annuity contracts, we will introduce some fundamental definitions. These definitions are useful to understand the underlying guarantees and to correctly assess the different engagements.

4.2.2 Useful definitions

We introduce two fundamental notions that are useful for a good understanding of the products the Account Value (AV) and the Benefit Base (BB). These two notions are to be differentiated.

- The Account Value (AV) is the amount of savings accumulated at a given date. This value, invested in investment funds, evolves according to the quotation of the assets on the financial markets. Fees for the management of the contract and the guarantee are periodically deducted from this account.
- The Benefit Base (BB), is the basis for calculating the minimum amounts that will be paid to the insured during the payout phase. This base is revalued periodically according to the roll-up or ratchet mechanisms described below.

¹Provided the contract has been in existence for at least eight years, only the interest portion of this income will be subject to the 12.3% social security levy and a flat-rate tax of 7.5%

Revaluation options for the Benefit Base (BB): Roll-up and Ratchet mechanisms

- The roll-up reset: The Benefit Base BB_{t_n} is capitalised annually at a given rate r_{RU} , defined in the contract. The Benefit Base amount at time t_{n+1} is given by :

$$BB_{t_{n+1}} = (1 + r_{RU})BB_{t_n}$$

The roll-up rate is generally between 0% and 5% depending on macroeconomic and financial factors, typically the level of interest rates and inflation of the currency in which the contract is denominated.

- The ratchet reset: The Benefit Base is at all times the maximum value reached by the policyholder's savings on a given set of dates fixed in the contract (generally the policy anniversary dates). Obviously, this is an expensive guarantee, because if the financial markets rise during the life of the contract, the minimum amounts paid to policyholders will be higher no matter how drastically markets might fall afterward. The Benefit Base at time t_{n+1} verifies the following expression:

$$BB_{t_{n+1}} = \max(AV_{t_{n+1}}, BB_{t_n}) = \max(AV_{t_0}, AV_{t_1}, \dots, AV_{t_{n+1}})$$

where t_0, t_1, \dots, t_{n+1} are the policy anniversary dates.

We present the fundamental and commonly used revaluation techniques, more complex mechanisms can also apply. Moreover, variable annuities contracts not only come with guarantees (GMxB: Guaranteed Minimum x Benefit ²) but also with the possibility to combine different guarantees in the same product to meet the policyholder's specific needs and savings goals as shown in the following graphic (fig. 4.6). This rapidly increases the complexity of the products and the challenges associated with their management.

We can broadly divide the guarantees into two categories:

- The guaranteed minimum death benefit (GMDB). A GMDB provides a guaranteed minimum benefit to the beneficiary upon the death of the policyholder.
- The guaranteed minimum living benefit (GMLB). This class of GMxB includes several guarantees which will be detailed later. It offers protection against the possibility that, after retirement, one outlive their assets.

Most of the theoretical papers examine elementary Variable Annuities. These products as we will see in the next sections can be priced using option pricing theory after some simplifying assumptions. Accounting for the full panel of risks (for instance Modelling policyholder's surrender behavior) makes the traditional methods purely theoretical and inefficient. Furthermore, the current marketplace offers a wide variety of increasingly complex forms of guarantees on VA products. As a consequence, more innovative methods such as the deep pricing approach are needed to improve the management of these products.

²GMxB is a generic notation to the following guarantees: GMDB, GMAB, GMIB, GMWB, and GLWB. Where D, A, I, W, and L respectively stand for Death, Accumulation, Income, Withdrawal, and Living

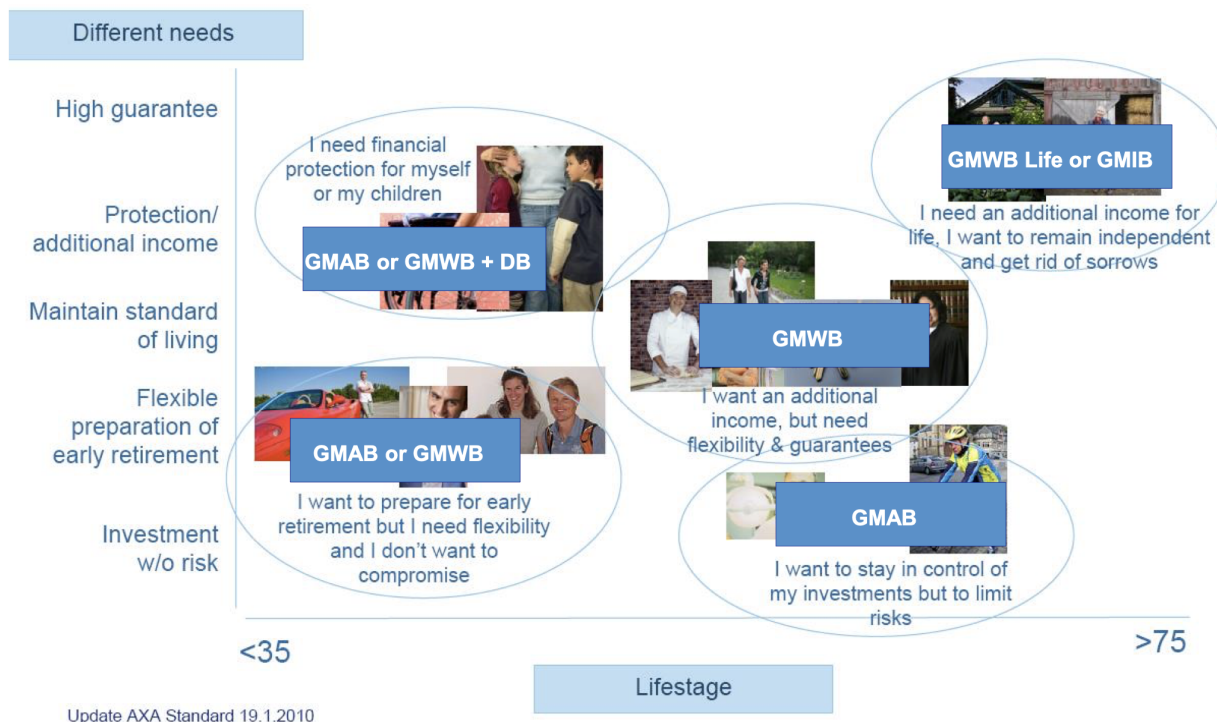


Figure 4.6 – Overview of Variable Annuities guarantees according to customer needs, source [21]

4.3 Variable Annuities valuation

4.3.1 The guaranteed minimum death benefit (GMDB)

A GMDB contract guarantees that if the policyholder dies before the contract's maturity date T , the beneficiary of the contract designated at subscription receives the higher of the account value and the benefit base adjusted for withdrawals. In traditional unit-linked life-insurance contracts, the benefit base is the initial premium (or total premiums in the case of regular payments). For a guarantee with a roll-up respectively ratchet revaluation mechanism, it corresponds to the premium accumulated at a specified interest rate, respectively the maximum account value at any anniversary of the account.

The beneficiary receives:

$$\max(AV_{t_{death}}; BB_{t_{death}}) = AV_{t_{death}} + (BB_{t_{death}} - AV_{t_{death}})^+ \quad (4.1)$$

where t_{death} is the date of death of the policyholder.

The minimum guaranteed amount limits losses in the event of a downturn in the financial markets, while offering the possibility of profiting from an upturn through the revaluation mechanisms of the benefit base.

In the case of a premium return or a roll-up guarantee, the death payment is similar to an American put option held by the beneficiary on the investment fund. $BB_{t_{death}}$ is the strike price, T the maturity and t_{death} the exercise date. However, this is not a classic American option, since the strike is not known in advance.

If we fix t_{death} , the guarantee becomes a European put option with maturity t_{death} and can be easily priced using the Black-Scholes formula.

Let's take an example. Let S_t the value of the fund at time t . For simplicity we take $S_0 = 1$. The undiscounted insurer's engagement for a premium return GMDB at time t_{death} , $V_{t_{death}}$ can be expressed as following:

$$\begin{aligned} E_x [V_{t_{death}}] &= E_x [\max(1 - S_{t_{death}}, 0)] = E_x [E [\max(1 - S_t, 0) | t_{death} = t]] \\ &= E_x [\text{Put}(1, 1, T)] = \int_0^\infty \text{Put}(1, 1, t) dF_x(t) = \int_0^\infty \text{Put}(1, 1, t) f(t) dt \end{aligned} \quad (4.2)$$

where F_x is the mortality cumulative distribution function and $f(t)$ its probability density. The GMDB guarantee is one of the simplest guarantees to price and explicit closed formulas can be derived for the different revaluation options given particular choices for the mortality function, typically the exponential mortality. (See Annexe 3). In fact, two approaches are possible to estimate the integral:

- Fit the mortality function with a continuous parametrical model
- Discretize the time step and approach the mortality curve from a discrete perspective.

A major drawback of the first method is that generally, a one-parameter model such as the exponential model fits poorly the data while two or more parameter models have a better fit but the time integral has no longer an analytical solution. The second approach is straightforward to implement but a time step must be chosen to be small enough not to distort the continuous property of mortality. Normally this time step is smaller than a year. Hence an interpolation hypothesis must be considered to adapt yearly mortality tables.

A commonly cited pricing approach in the literature is the weighted puts method where the guarantee value is approached by the sum of European put options maturing at regular intervals (generally monthly) weighted by the probability of death at each date.

When is a GMDB profitable for the contract's beneficiary?

In the following, we will focus on guarantees with ratchet revaluation options, which are usually more complex to price.

A GMDB is profitable if the policyholder dies during the life of the contract ($t_{death} < T$), and the guaranteed amount is higher than the account. If the GMDB is payable at the end of the month of death, the expected present value of the benefits can be expressed as follows:

$$EPV_{GMDB} = \sum_{k=1}^N q_x^k \cdot E \left[\exp \left(- \int_0^{k/12} r_t dt \right) \cdot \max (BB_{k/12} - AV_{k/12}, 0) \right] \quad (4.3)$$

where q_x^k is the probability of a policyholder age x dying in month k , N is the number of months until the expiration of the contract, and $BB_{k/12}$ is the benefit base at the date of death of the policyholder.

4.3.2 Guaranteed Minimum Accumulation Benefit (GMAB)

The Minimum Accumulation Benefit is similar to the GMDB but rather "in case of life" of the policyholder. It guarantees a flat or contractually increasing (rollup option) accumulated lump sum amount if the policyholder is alive by the end of the accumulation phase. The policyholder should then receive the greater of their account value and some guaranteed

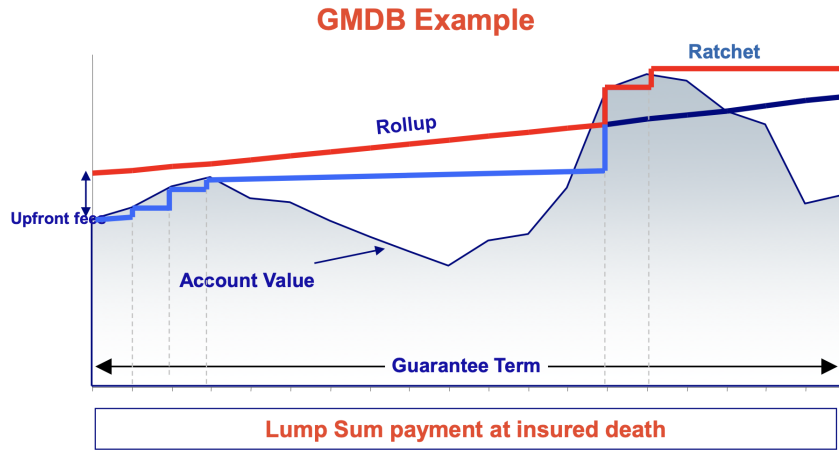


Figure 4.7 – GMDB example, source [21]

amount specified in the contract.

When is a GMAB profitable for the contract's beneficiary?

A GMAB pays off if the policyholder is alive at the end of the accumulation phase and the account value is less than the guaranteed value (benefit base). The expected present value of the benefits can be written as:

$$EPV_{GMAB} = p_x^T \cdot E \left[\exp \left(- \int_0^T r_t dt \right) \cdot \max(BB_T - AV_T, 0) \right] \quad (4.4)$$

where T is the time until the expiration of the contract and p_x^T is the probability of a policyholder age x surviving T years. Note that $p_x^T = 1 - q_x^T$

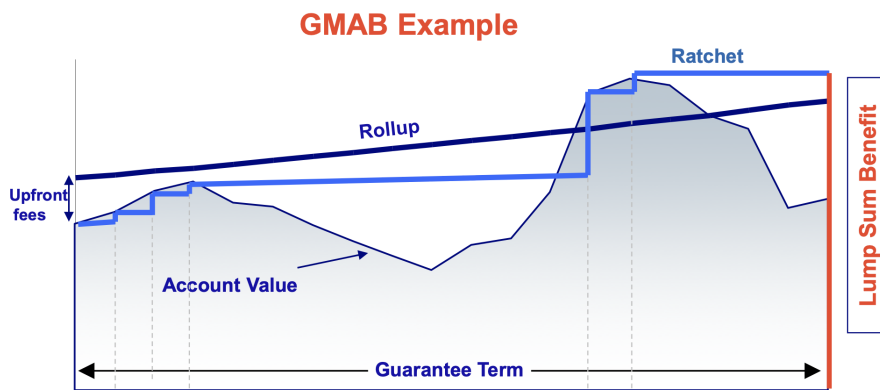


Figure 4.8 – GMAB example, source [21]

4.3.3 Guaranteed Minimum Income Benefit (GMIB)

The Guaranteed Minimum Income Benefit guarantees minimum annuity if the policyholder desires to annuitize their contract. At maturity they have two options: either receive the accumulated account value and/or annuitize it at market conversion rate (a_T) or annuitize

it at a guaranteed conversion rate (a_g).

This benefit hence depends on both the asset’s performance and the interest rate level at the time of conversion. It is the right, to buy a fixed immediate life annuity, for a deterministic strike price during the life of the contract. Consequently, it can be seen as a call option on annuity purchase factors.

From the insurer’s perspective, they promise to guarantee an option on two underlying stochastic variables; future interest rates and future mortality rates.

The GMIB guarantees that the annuitant will receive payouts whose present value is equivalent to the greater of the annuitization and the account value. The guaranteed benefit is an annuitization factor multiplied by the initial account value reevaluated according to a given revaluation mechanism. In the case of a life-contingent annuity, longevity risk also adds to financial risk. We express the guarantee’s payoff at time t_j as follows:

$$\text{Payoff}_{t_j} = \begin{cases} 0, & \text{if } t_j < T \\ \max \left\{ 0, BB_{t_j} \frac{a_T}{a_g} - AV_{t_j} \right\}, & \text{if } t_j = T \end{cases} \quad (4.5)$$

where a_g and a_T are respectively the guaranteed purchase price and the market price of an annuity paying 1 unit per year. a_g is modeled assuming an interest rate model. A simple example is:

$$a_g = \sum_{n=0}^{\infty} np_x e^{-nr}$$

where the interest r is typically assumed to be around 5% and np_x denotes the probability that an annuitant aged exactly x lives for another n years.

When is a GMIB profitable for the contract’s beneficiary?

The guarantee is profitable when the guaranteed benefit exceeds what Account Value could purchase at the current interest environment.

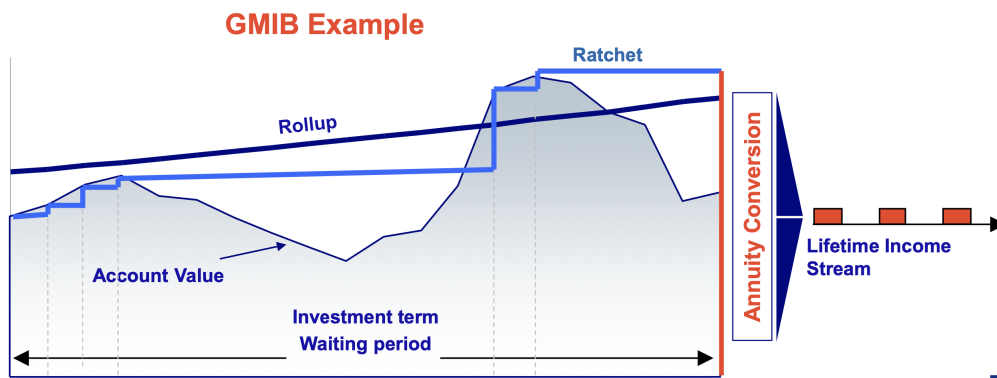


Figure 4.9 – GMIB example,source [21]

4.3.4 Guaranteed Minimum Withdrawal Benefit (GMWB)

The Guaranteed Minimum Withdrawal Benefit (GMWB) is the most complex of the GMxBs. Not only the basic payment structure is strongly related to Asian options, but the guarantee value is very sensitive to the policyholder’s behavior. That is, in order to price this

guarantee it is important to take into account the surrender behavior of the policyholder. A variation of this product is the GMLB, Guaranteed Minimum Withdrawal Benefit for Life which corresponds to a lifetime guarantee.

The GMWB promises the annuitant the possibility to yearly withdraw some fixed amount from their account until the benefit base is entirely depleted, even if the account value is zero. The benefit base is typically set equal to the initial premium and declines with withdrawals. At maturity, the remaining withdrawal balance is returned to the policyholder.

We suppose no lapses and a static withdrawal strategy, where the policyholder can withdraw up to a fixed annual withdrawal rate (r_W) of the guaranteed withdrawal base only on the policy's anniversary dates. the expected present value of the guarantee can be expressed as follows:

$$EPV_{GMWB} = \sum_{n=1}^T E \left[\exp \left(- \int_0^n r_t dt \right) \cdot \max (0.1 \cdot BB_n - AV_n, 0) \right] \quad (4.6)$$

where BB_n is the benefit base, ratcheted in proportion to the account value less withdrawals. Finally, the liability for a contract on a given day is calculated as the expected present value of the benefits minus the expected present value of the fees.

When is a GMWB profitable for the contract's beneficiary?

A GMWB pays off if the account value has been depleted, but the policyholder still has the right to make withdrawals.

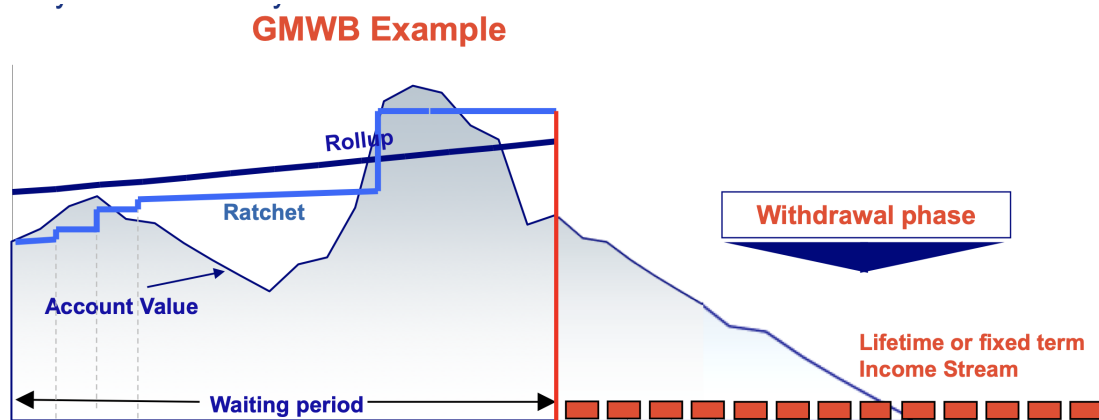


Figure 4.10 – GMWB example,source [21]

4.4 Challenges associated with VA pricing and hedging

4.4.1 Pricing of Variable Annuity contracts in practice

Correctly pricing the different guarantees embedded in variable annuity contracts relies on correctly modeling the underlying risks and their interactions.

On the one hand, the financial risks are modeled in a risk-neutral framework. Recall that the purpose of the risk-neutral valuation is to retrieve the price of traded financial instruments so as to prevent arbitrage. Consequently, scenarios probabilities are implied from the

prices of traded securities whose payoffs depend on those scenarios. Since investors are overall risk-averse, risk-neutral valuation generally implies higher probabilities to unfavorable and extreme scenarios than for favorable ones.

On the other hand, non-financial risks (policyholder behavior, mortality risks, lapses, etc.) are generally real-world scenarios based on historical observations and dynamics. To sum up, the difference between risk-neutral and real-world scenarios does not lie in the individual scenarios themselves; but rather in their occurrence probabilities.

Economic scenario generators are used to simulate movement scenarios of the indices according to an asset model. There are two types of scenarios: risk-neutral and real-world. Risk-neutral scenarios are simulated under the risk-neutral measure; while real-world scenarios are simulated under the real-world measure. Risk-neutral scenarios are used to calculate the fair market values of financial derivatives such as the guarantees embedded in variable annuities. Real-world scenarios are used to calculate solvency capitals or evaluate hedging strategies.

Variable annuity contracts pricing process in practice:

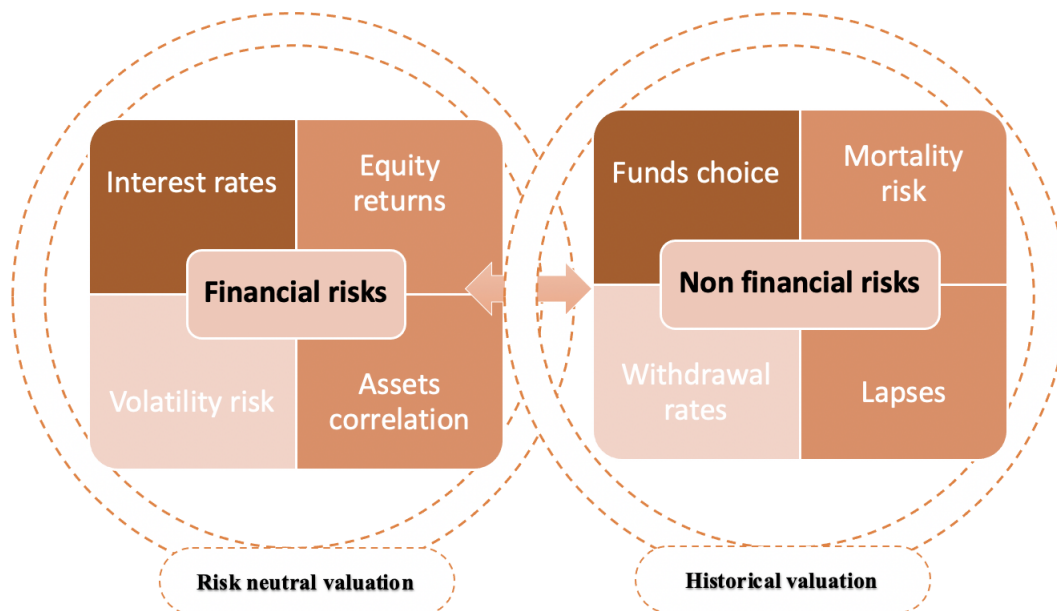


Figure 4.11 – Financial and non financial risk drivers

1. Identify the key financial and non-financial risks impacting the value of the embedded guarantees (Equity movements risk, interest rates risk, longevity risk, etc). (See fig.4.11)
2. Choose an Economic Scenario Generator (ESG) to model the financial risks. This implies making assumptions on the financial risk drivers and diffusing the retained models under risk-neutral probability.
3. Calibrate the ESG according to implied/market values, and historical data.
4. Set assumptions for non-financial risks based on company and/or industry experience (real-world scenarios).

5. Value the guarantees using a sufficient number of Monte Carlo simulations to ensure convergence of a risk-neutral valuation.

4.4.2 Overview of the problems and commonly used solutions

As we have seen VA guarantees can be often considered as financial options and consequently option pricing theory is applied for both their pricing and hedging. Many examples in the literature deduced that the restrictive assumptions retained to derive closed-form formulas for the pricing of these guarantees can yield in these products being largely underpriced. These restrictive assumptions include the modeling of the main financial risk factors namely the interest rate, the fund's dynamics, and the demographic risk factors particularly policyholders' surrender behavior and the mortality dynamics.

The commonly used Black Scholes model assumes deterministic interest rates which do not reflect the reality of the market and can introduce real discrepancies, particularly for long-term financial guarantees such as Variable Annuities. The disparities due to these simplifying assumptions widen as the complexity of the products increases. For instance, Milevsky and Salisbury concluded in their 2006 paper that GMWB products were largely underpriced.

Even for some of the more relatively simple types of guarantees, it is often infeasible to derive closed-form pricing formulas without very restrictive assumptions. We saw in the previous section that to derive a closed-form formula for a GMDB guarantee one should assume a one-factor mortality risk model, typically an exponential model which does not capture the real dynamics of mortality rates.

As a result, MC simulations are largely adopted to solve the problem. Although very useful, these approaches yield a significant computational cost for insurers maintaining a large portfolio of VA contracts. Many techniques were considered to reduce this cost. But as guarantees and model complexity increase, the need for efficient techniques to value and hedge VA is becoming crucial for insurance companies.

In the literature, there are three main categories of approaches to reduce the computational cost of MC simulations:

- Scenario reduction approaches: when using Monte Carlo to value a portfolio of variable annuities, the payoffs of the guarantees embedded in each policy are calculated at each time step for a set of risk-neutral scenarios. Reducing the number of scenarios is one way to reduce the runtime of MC simulation.
- Inforce compression approaches: these approaches reduce the runtime by decreasing the number of insurance policies that are valued by MC simulations. In fact, to price the VA portfolio, contracts are not considered independently but rather as groups or model points via clustering techniques.
- Metamodeling approaches: these approaches are similar to inforce compression techniques as both methods reduce runtime by cutting down the number of policies to price. However, the former are more sophisticated. They rely on two major steps: An experimental design method (EDM) is used to select representative policies and a predictive model or metamodel allows interpolation between previously calculated

values and the remaining policies. A wide range of EDM and predictive models were considered in the literature (See Annexe 5.2.2 for an overview).

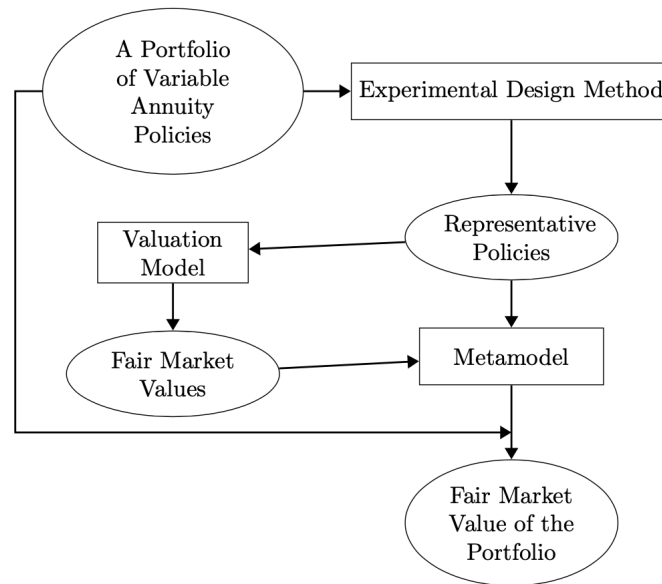


Figure 4.12 – A general framework of metamodeling approaches

5

Variable Annuities deep pricing approach

In recent years, researchers have explored the idea of applying neural networks for the tasks of pricing and/or hedging VA guarantees. Hejazi and Jackson (2017) used neural networks in their approach to estimating the solvency capital requirement (SCR) for a large portfolio of VA policies. They concluded that the neural networks approach yields accurate values while greatly increasing the efficiency of the calculations.

The neural network pricing approach as previously introduced in this report aims to map a set of policy characteristics and different economic settings to the MC estimation of the policy's liabilities or fair market value. Recall this approach is not model-free and we need to make several assumptions on the main risk factors for the valuation.

The neural network can be used to compute guarantees sensitivities, hence once adequately trained, daily MC can be discarded. We will compare the performance of the neural network to the commonly used MC simulations, quantifying the gains in speed, and showing that these gains can be obtained with little sacrifice in accuracy.

5.1 Training dataset

We saw in the second chapter when introducing the deep pricing approach how crucial data quality is for the neural network to correctly learn the pricing function. However, it is difficult to obtain real datasets from insurance companies to train and assess the neural network approach. Moreover, the neural network should be trained on a consequent number of data points for the model to be reliable and accurate. We implemented a simple version of the MC valuation framework described in the previous chapter. We supposed that annuitants can invest in a single fund whose returns are normally distributed (Black-Scholes model). We also assumed a Hull-White diffusion for the short interest rate and a constant mortality rate. Let's suppose that we diffuse 10000 risk-neutral scenarios over 30 years (360 months) for each Monte Carlo valuation. In practice, more than 100 000 paths are used to retrieve a robust valuation. Let's consider a portfolio containing 50000 VA policies and assume as in [12] 200 000 cash flow projections per second. Then it would take the computer to generate

the data approximatively:

$$\frac{1000 \times 360 \times 50000}{200000 \text{projections/second}} = 25 \text{hours}$$

The computational cost of Monte Carlo-based valuation and hedging frameworks will be further discussed at the end of this chapter to highlight the potential gains in speed introduced by a neural network-based approach.

5.1.1 Gan and Valdez Synthetic portfolio of VA

We will be using in this chapter a synthetic portfolio of variable annuity contracts created by Gan and Valdez (2017b) [1] to facilitate the research related to the efficient valuation of large variable annuity portfolios. The public dataset can be freely accessed on their website ¹.

The synthetic portfolio is designed to mirror the major properties of real variable annuity portfolios in the United States. For instance, policyholders have the right to choose their placements among different investment funds provided by the insurer. They can combine various guarantees in the same contract to meet their specific needs. Typically, annuitants looking for financial protection for themselves or their children often combine the GMDB and the GMWB riders. The fees for the combined guarantees are set equal to the sum of the fees of the individual guarantees minus 0.20%.

The authors originally consider 19 variants of variable annuity contracts. These products differ by the embedded guarantees (a single or several guarantees can be included) and the benefit base revaluation mechanism. Gan and Valdez also compute the fair market values and the Greeks of the integrated guarantees using a simple Monte Carlo simulation engine. In this study, we will focus on 17 different products as described in the following table. Refer to the previous chapter for a detailed description of the different guarantees.

Main assumptions

In this section, we present Gan and Valdez's [11] public synthetic portfolio. We focus particularly on its construction and the different assumptions made by the authors. To be fully able to exploit the database and enrich it with relevant variables that capture the economic and demographic parameters used in the simulation task, we focus on an in-depth review of the data generation process (the code used by the authors), which is available on the same website.

The authors assume that each variable annuity contract is financed by a single premium paid at the issue date.

The annuitant chooses a combination of funds in which they wish to invest among 10 funds provided by the insurer. This premium will hence accumulate value through investment returns. In their implementation, the authors use a fund mapping to map 10 investment funds provided by the insurer to a combination of five tradable and liquid indices². The 5 fund indices respectively track US large-cap equity, US small-cap equity, international equity, fixed income, and money market fund.

¹<http://www.math.uconn.edu/~gan/software.html>

²An index fund is an exchange-traded fund (ETF) designed to match the composition and performance of a financial market index.

Product	Description	Rider Fee
DBRP	GMDB with return of premium	0.25 %
DBRU	GMDB with annual roll-up	0.35 %
DBSU	GMDB with annual ratchet	0.35 %
ABRP	GMAB with return of premium	0.50 %
ABRU	GMAB with annual roll-up	0.60 %
ABSU	GMAB with annual ratchet	0.60 %
IBRP	GMIB with return of premium	0.60 %
IBRU	GMIB with annual roll-up	0.70 %
IBSU	GMIB with annual ratchet	0.70 %
WBRP	GMWB with return of premium	0.65 %
WBRU	GMWB with annual roll-up	0.75 %
WBSU	GMWB with annual ratchet	0.75 %
DBAB	GMDB + GMAB with annual ratchet	0.75 %
DBIB	GMDB + GMIB with annual ratchet	0.85 %
DBWB	GMDB + GMWB with annual ratchet	0.90 %

Table 5.1 – Variable annuity contracts in the synthetic dataset (Guojun Gan* and Emiliano A. Valdez) dataset

This approach is convenient for two reasons. First, most of the investment funds are not tradable and can't be used to hedge the contracts' guarantees. In practice derivatives on tradable indices are used via a fund mapping technique. The weights in the mapping function can be estimated by the method of least squares from the historical returns of both the investment fund and the indices. Second, using tradable indices in the ESG model makes the generated scenarios more realistic as they can be calibrated to the market.

The guarantees are financed by periodic risk charges deducted from the policy account. In addition to the general Mortality & Expense (M&E) fee, exist specific rider fees for each guarantee and management fees depending on the selected investment funds. The valuation date of the synthetic portfolio is the 11th of June 2014.

The data generation process is structured around these three key steps :

- **Yield curve generation:** The authors do not use an interest rate model but rather model fixed income indices directly. In this simplified approach they consider US swap rates on the valuation date for 8 different tenors. They exploit Secant method to bootstrap the corresponding discount factors and forward rates. These rates are then interpolated using a loglinear method to get monthly forward rates as shown in this figure.³
- **Scenario generation:** The authors implemented a simple risk-neutral scenario generator. Its inputs are the forward curve bootstrapped from market swap rates, the correlation matrix of the five modeled indices, and their volatilities. After simulating several risk-neutral index scenarios according to a multivariate Black-Scholes model, the fund map is used to allocate returns of the indices to each investment fund according to the corresponding proportion of investment.

³The annualized continuous forward rate for period (t_{j-1}, t_j) is defined as: $f_j = \frac{1}{(t_j - t_{j-1})} \int_{t_{j-1}}^{t_j} r_s ds$

The risk-neutral dynamics of the 5 indices are given by the following equation:

$$\frac{dS_t^{(i)}}{S_t^{(i)}} = r_t dt + \sum_{l=1}^5 \sigma_{il} dW_t^{(l)}, \quad S_0^{(i)} = 1, i = 1, \dots, 5 \quad (5.1)$$

with $W_t^{(1)}, W_t^{(2)}, \dots, W_t^{(5)}$ are independent standard Brownian motions, r_t is the short interest rate, and $\Sigma = (\sigma_{hl})_{1 \leq i, l \leq 5}$ denotes the covariance matrix of the annualized continuous returns of the 5 indices. The solution of the stochastic differential equation can be expressed as:

$$S_t^{(i)} = \exp \left[\left(\int_0^t r_s ds - \frac{t}{2} \sum_{l=1}^5 \sigma_{il}^2 \right) + \sum_{l=1}^5 \sigma_{il} B_t^{(l)} \right], \quad i = 1, \dots, 5 \quad (5.2)$$

Fig. 5.1 shows the 1000 risk-neutral (RN) scenarios used to simulate the Large Cap Equity index value⁴ over 30 years assuming the equity initial value $S_0 = 100$. The mean and covariance matrices of the multivariate geometric Brownian motion were calibrated on historical returns of the considered indices between January 2000 and May 2014. Recall that the valuation date of the portfolio is set to the 1st of June 2014.

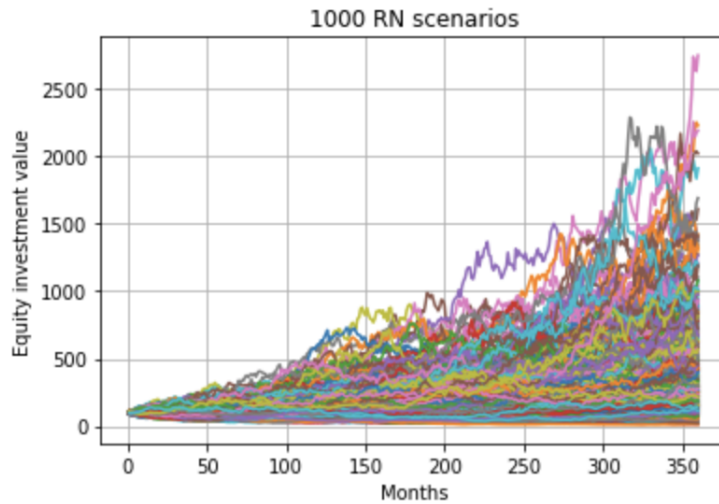


Figure 5.1 – Example of equity investment risk neutral scenarios ($S_0 = 100$)

- **Policy generation:** This part randomly generates a pool of variable annuities based on realistic ranges for the different parameters. The synthetic portfolio consists of annuitants born between 1950 and 1980 with a 40% female share. Policies are issued between 2000 and 2014 with maturities from 15 and up to 30 years and an initial account value between 50000 and 500000.

Investment funds for each policy are also randomly selected among the ten proposed funds and the total account value is equally distributed between them. Fund management fees are 30, 50, 60, 80, 10, 38, 45, 55, 57 and 46 respectively for the ten provided funds. The synthetic portfolio includes 10000 policies of each product listed in the table 5.1.

⁴This index's benchmark is the *S&P* 500 index.

- **Monte Carlo valuation:** This is the last block of code. The cash flows arising from the different guarantees (death benefits+ life benefits and rider fees) are projected according to the different risk-neutral paths. The market-consistent value, also known as **the fair market value (fmv)** of each variable annuity contract guarantees is computed as the discounted value of death and life benefits once risk charges are deducted.

5.1.2 Data Enrichment

We aim to train a neural network to predict the liability (fair market value of the guarantees) based on the main parameters and inputs fed to the simulation. The current database gathers information about both the policyholders and the guarantee riders. After understanding the data generation process and the assumptions made by the authors, we can extend the database with some important features reflecting the dynamics of the financial and demographic processes used in the simulation phase. Report to Annex 5.3 for an insight into the distributions of the main variables.

The table below summarizes the features existing in the original synthetic dataset and the created ones.

Features	Old	Created
Policyholder-related	gender, birth date	age
Policy-related	Issue date, Maturity date, fund values, guarantee fee, Rollup rate, Product type, Guaranteed benefit (<i>gbAmt</i>) Total withdrawal Annual Withdrawal rate Withdrawal balance	Policy age Time to maturity Diversification ratio (number of selected funds/ number of available funds) Account value (the sum of fund values) Total fees $\mathbb{1}_{Ratchet}$ Guarantee moneyness
Demographic	-	q_x
Economic	-	Scaled first three principal components of historical yield curve PCA Price of zero-coupon with the same maturity Equity investment adjusted closing price Equity investment adjusted volatility

The guarantee moneyness (M) is defined as $M = (G - AV)/G$, where AV is the account value. G refers to the guaranteed benefit amount (*gbAmt*) when the policy does not include a withdrawal guarantee. For guarantees with a withdrawal option, G denotes the withdrawal balance. To reflect the demographic conditions used for the valuation task, we created q_x the probability of dying in the next year based on the mortality table used for the simulations⁵.

⁵1996 Individual Annuity Mortality (IAM) in the US, tables 1698 and 1699 available on <https://mort.soa.org>

A major risk factor impacting the valuation of variable annuities is interest rates risk. To study this risk, we collect monthly market yields on U.S. Treasury securities for different maturities (3 months, 6 months, 1 year, 2 years, 3 years, 5 years, 7 years, and 10 years). The historical dataset spans from January 2000 to June 2014 and can be easily accessed via the Quandl API.⁶

Yield curves show many interest or market remuneration rates of debt securities, across different maturities. They are generally expected to be upwards sloping, because of the time value of money. Their levels also depend on the credit quality of the borrower as interest rates tend to increase to compensate for the uncertainty of repayment. Therefore, sovereign governments bonds are used to build the benchmark “risk-free” yield curve applied to discount the expected future cash flows.

After cleaning filtering the data for the relevant information, we plot yield curves for different points in time. Fig 5.2 highlights a decrease in both level and steepness over the years.

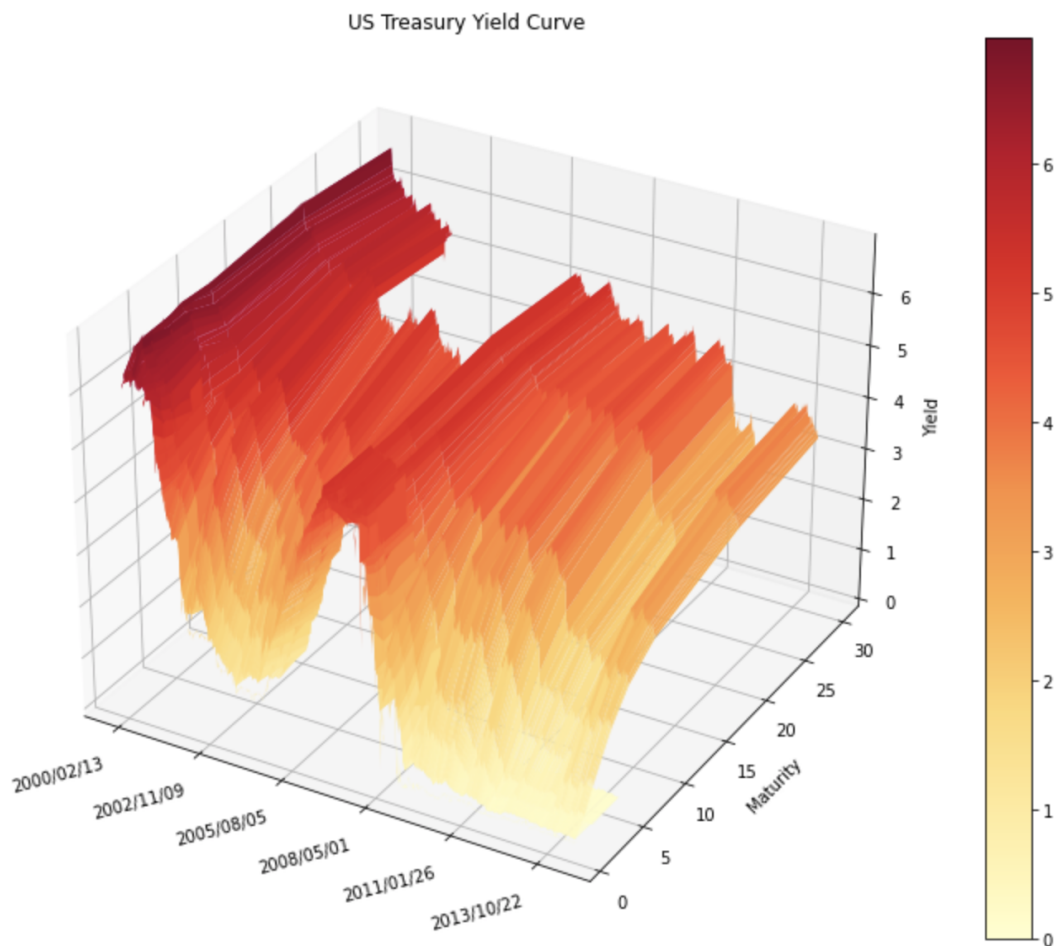


Figure 5.2 – US Treasury yield curve

To reflect the main dynamics of the term structure of interest rates we decompose it into its main drivers. For this, we apply a commonly used technique of dimension reduction: Principal Component Analysis (PCA). This technique reduces the high dimensionality of

⁶Quandl is the API for Nasdaq Data Link, a platform that provides a large panel of financial and economic datasets: [//data.nasdaq.com](https://data.nasdaq.com)

the data by compressing the variability of all features into a limited set of transformed features. We restricted the analysis to observations from 2008 onwards.

Let X the drivers of the interest rates term structure. In our case, each feature of X is a vector of monthly yields for a given maturity. To capture the maximum variability of all maturities, we assess the covariance matrix of X and derive its Eigenvectors. The three most important eigenvalues which are generally interpreted as the height, slope, and curvature of the yield curve; respectively account for 94.22%, 5.38%, and 0.28% of the total variance. Their associated eigenvectors account for the term structure of the yield curve. We scale the eigenvectors with respect to their explained variance proportion.

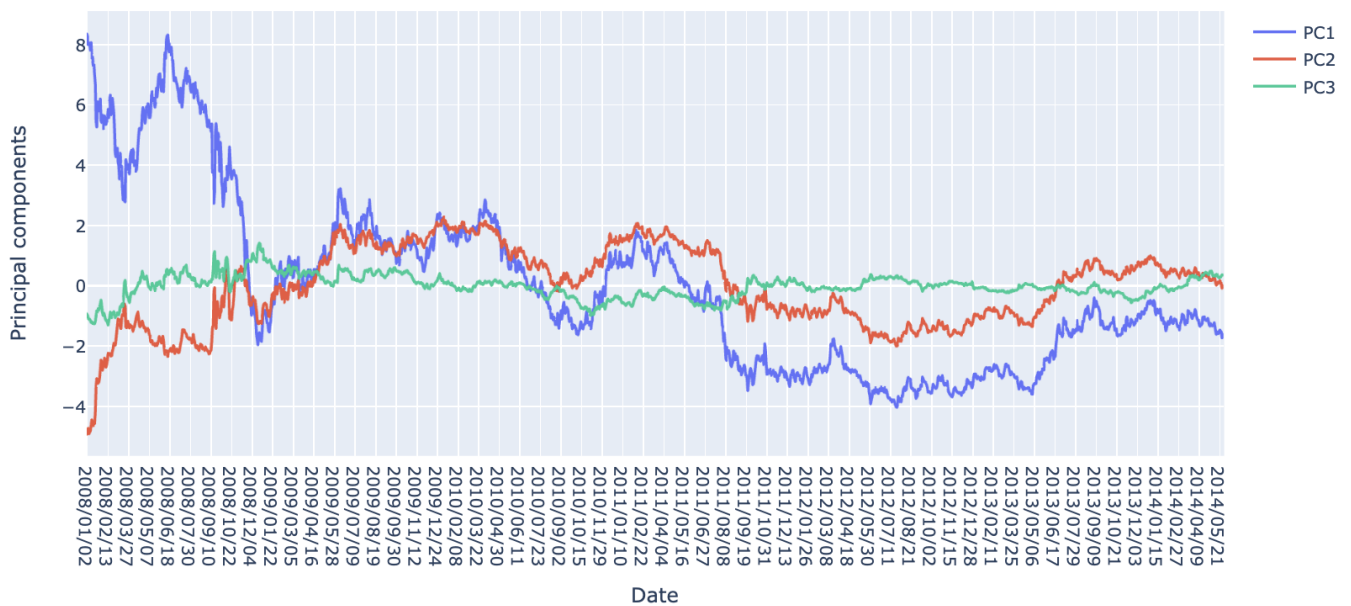


Figure 5.3 – US yield curve principal components

We recall that the multivariate Black Scholes model used in the economic scenario generator is calibrated on the considered five financial indices’ historical returns. The following graph (fig. 5.4) shows the covariance matrix of the annualized continuous returns of the five indices.

We use this matrix and the fund mapping function to define the Equity investment adjusted volatility. This indicator provides an insight into the historical volatility of the investment strategy for each policy. Similarly, the equity-adjusted closing price captures the performance (level) of the investment strategy on the valuation date. We also use the forward curve and discount factors bootstrapped by the authors from the US Swap rates to compute for each policy the price of a zero-coupon bond maturing at the same date.

We should mention that for the eigenvectors as well as the ZC prices an interpolation method (cubic spline interpolation) was applied to fill in the missing maturities with a monthly time step.



Figure 5.4 – Equity indices and investment funds correlation matrices

5.2 Results and further improvements

5.2.1 Numerical results

Preliminary correlation analysis (fig. 5.5) shows that the target variable is positively correlated with the guaranteed benefit, the guarantee’s moneyness, and the roll-up rate. These three variables seem to be strong drivers of the guarantee’s fair market value. The policy’s age is also positively correlated with the guarantee’s fair value, this is consistent with the fact that such policies are used to accumulate wealth. Both withdrawal balance and annual withdrawal rate have a negative impact on the guarantee’s fair market value.

Since the variable annuity contracts are long-term contracts, the guarantees are usually sensitive to the difference between current interest rates and long-term interest rates. This difference in slope is generally captured by the second principal component of the yields curve. In fact, the difference between 30-years rates and 3-months rates, generally interpreted as yields curve slope, is highly correlated with the second principal component (0.98% estimated correlation).

For the training, we used the same architecture as in the previous chapters: a feed-forward network. We emphasize the importance of input normalization. However, target

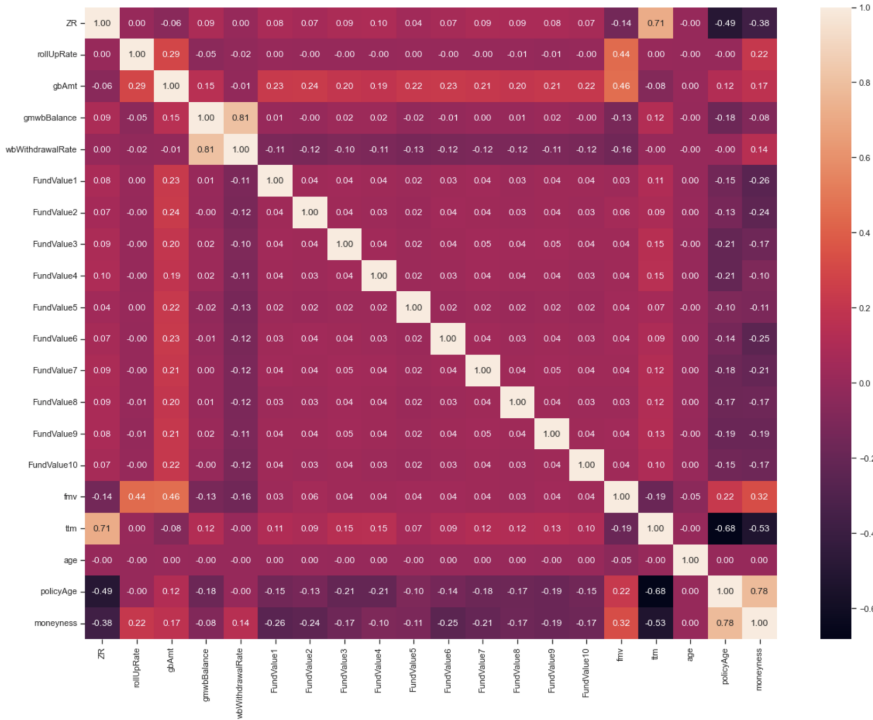


Figure 5.5 – Main variables correlations

normalization did not seem to improve the model’s performance. We used a normalization layer followed by 2 dense layers with 20-50 neurons each and an output layer with a linear activation. Applying an ELU activation function in the hidden layers yielded the best performances. We chose to retain this generic architecture for the different models in this section and tune the following parameters (number of neurons, number of training epochs, batch size, and learning rate) for each trained model. We defined the mean absolute error as the model’s loss function and minimized it using the Adam optimizer. Report to the first chapter for more details about the training and tuning processes.

To assess to performance of the neural network we compare the model’s predictions with Monte Carlo fair market values.

We first trained the model on the entire dataset (100 epoch, batches, and a learning rate of 10^{-3}). The neural network predicted the target given the type of the embedded guarantees (product type), policy characteristics, the mortality context as derived from the probability of death at the age on the valuation date, and economic scenario (equity and bond prices, volatilities, etc..). Although both the training and validation losses decreased significantly, the average relative error remained. The neural network does not seem to learn the mapping function correctly. Increasing the number of neurons in each layer improves the performance of the model on the training set but the results are less stable on the test dataset. This could mean that the map function we try to approach via a neural network is not continuous from one product class to another. The universal approximation theorem may no longer apply.

To improve the predictions we defined 4 classes of products, each class includes guarantees of the same type. For products with several guarantees, we have chosen to include them in the GMWB class if they include a withdrawal option and the GMDB class if they

include a minimum death benefit.

Table 5.2 summarizes some cross-validation metrics computed on the test datasets for several models.

Metrics	The entire dataset	GMWBs	GMDBs	GMABs	GMIBs
R2	0.998	0.999	0.998	0.992	0.996
MAE (test)	1536	490	827	940	589
MARE (test)	7%	3%	5%	4%	4%
$q_{0.95}$ MARE (test)	20%	9%	10.3%	11,7%	9.8%

Table 5.2 – cross-validation metrics on the test datasets

In their paper [7], the authors used R-squared (R^2) to “study the relative effectiveness of using neural networks to approximate MC simulations results for the different guarantees”. The R^2 obtained in this study is equally very high. However, we insist on the fact that this metric does not assess the “goodness” of the neural network model and its capacity to correctly predict the fair market values. It only shows the proportion of the target variable variance that’s explained by the set of independent inputs in the regression model. A low value would mean that the regression model is not valid, but a high value does not necessarily mean high accuracy.

The table above (Table 5.2) shows that the neural network approach performs quite well on average. It substantially yields better results for the different classes of products than for the aggregated model. Both validation and test errors do not suggest any potential overfitting.

Considerable speedups are obtained with a relatively marginal decline in accuracy. However, the method’s real-world applicability requires a deeper study and further investigations to prevent aberrant predictions. In fact, relative errors distribution showed heavy tails (high quantiles) with extreme values despite a preliminary preprocessing of inputs outliers.

5.2.2 Further improvements and applications

The results of the previous section show that the deep pricing approach can be applied to complex variable annuities pricing. At this stage, the model is not applicable in the real world as predictions still show some aberrant values despite a relatively small average error. A deeper investigation of the outlier predictions is required. We suggest aging the portfolio and adding the realized fair market values and realized equity prices to the inputs. Applying a more sophisticated architecture, for instance, a recurrent neural network to capture path dependencies reflected by the new inputs would improve the model’s performances.

We see a significant gain in computational efficiency when compared to the traditional MC pricing methodologies. Computing the fair market values of the synthetic portfolio as well as the greeks took the authors approximately 108 hours, while once trained the evaluation of the neural network on the entire dataset takes no longer than 3 seconds.

The deep pricing framework is an interesting subject to research as it could highly improve several computationally demanding tasks. For instance, it can be used to compute policies sensibilities (greeks) simply via the usual bump method (refer to Annex 6) and

for Solvency capital requirements (SCR) estimations. These tasks generally require nested Monte Carlo simulations with an outer loop spanning a set of real-world scenarios and an inner loop used for the pricing of the guarantees via risk-neutral scenarios.

Policyholder behavior: a key assumption and a significant risk factor

For the sake of simplicity, static assumptions regarding the frequency and magnitude of policy lapse rates and withdrawal rates (in the case of guaranteed withdrawal benefits) were made. No dynamical lapse or withdrawal models were considered. Although simplifying assumptions are usually made by insurers, these modeling hypotheses are critical and shed light on one of the main risk factors in variable annuity modeling: policyholder behavior.

Policyholder behavior risk is the risk that the insurer's expectations in terms of policyholders' benefit utilization (particularly withdrawal) and surrender do not align with annuitants' conduct resulting in a misestimation of the insurer's engagements. Insurers' expectations and assumptions vary according to the type of guarantee and are generally based on internal studies (historical experience) as well as external studies (behavioral economics, etc.).

Although the fair values estimated in this chapter are based on static assumptions, we should mention that, in reality, surrender rates and benefit utilization are dynamic. In fact, policyholders interact with their environment when making their investment decision.

For instance, their rational behavior suggests seeking better investment opportunities when the market is rising (higher lapse rates and/or withdrawals) and passing on the risk to the insurer when the market is declining (lower lapse rates and/or withdrawals). The timing as well as the magnitude of withdrawals and policy surrenders can have a significant impact on the cost of the guaranteed benefits, reserves, and capital. Therefore, shifting to dynamic behavior modeling can improve the understanding of the guarantees underlying risks and yield more market-consistent fair values.

Behavior modeling is nevertheless particularly challenging especially when little sound experience data is available to assess the validity of the modeling assumptions.

We acknowledge that based on our simplifying static assumptions, policyholders' behavior risk has not been properly taken into account, which is a limitation of this approach. However, we stress the importance of the policy-oriented aspect of the deep pricing approach compared to the commonly used aggregated approaches particularly in force compression methods.

Viewing policyholders as individuals rather than averages is a valuable aspect of the deep pricing framework as it can be used to reflect and understand the individual, causal factors of decision making and hence improve the assessment of holders' behavior risk. Such an individual view has only recently become possible thanks to the continuous research in cognitive science and behavioral economics. Using dynamic lapse and withdrawal assumptions in the Monte Carlo simulation engine might enhance the model's real-world applicability by reflecting the individual decision-making dynamics in the guarantees' computed fair values.

The deep pricing approach can also help alleviate the policyholder's behavior risk by averaging the predicted market values of each policy (individually) on a set of static lapse and withdrawal assumptions. This is possible once the neural network is trained and tuned, as the evaluation of the pricing function on different input variables is instantaneous and does

not involve further Monte Carlo simulations.

While the proposed approaches can help better understand and manage holders' behavior risk, they are still insufficient. Product design solutions are often essential. One needs to avoid product designs, that favor crystallization of losses for many policyholders at the same time. In practice, insurers rely on penalty-based solutions. For instance, many insurers would penalize early surrenders and lapses within the first years of policy life pushing policyholders to alter their behaviors to avoid penalties.

Conclusion

The deep pricing framework presented in this study consists in training a neural network to predict the value of a financial instrument or a policy in a given model based on a set of inputs/ parameters. These inputs characterize both the valuation model and the priced policy.

The neural network can be trained to predict a single value (price) or a price surface depending on the application. In the deep calibration framework, the trained neural network ("fast surface") can map relatively accurately the inputs on a given parameter space of interest to the model prices, outperforming Monte Carlo simulations by trading off compute time for training with inference time for pricing .

This fast surface is then used to calibrate a given model according to two different approaches. Either by simply replacing the pricing functional with the learned "fast surface" to solve the inverse problem using classical optimization techniques. Or, by leveraging the fact that the NN is a nested function of the input parameters whose gradient is known analytically to minimize the \mathbb{L}^2 loss with respect to the model parameters. In fact, The gradient is analytically known not only for the weights and biases but also for the input parameters thanks to the backpropagation principle.

We trained and test this methodology on the pricing and calibration of two commonly used option pricing models: the Black and Scholes model and the Heston model. A major strength of both the deep pricing and deep calibration frameworks developed during this internship is their flexibility genericness and capacity to be adjusted to different financial instruments and models.

Throughout this study, we highlighted how these alternatives introduce considerable calculations speedups compared to the Monte Carlo approach and presented different risk management applications in both finance and insurance. In the last chapter we trained a neural network to predict the liability of equity-linked insurance policies, variable annuities. Eventhough this approach significantly reduced the runtime and yielded relatively low errors, the model still needs to be adjusted to prevent aberrant predictions and ensure stability in order to be employed in the real world.

Our work, has many other avenues for improvement and research. Among them, researching more advanced sampling methods for the data generation phase. An approach that combines information from historic market data with a synthetic data generator algorithm would produce more market-consistent samples and induce better performances. For the synthetic portfolio it would be interesting to age the portfolio and integrate realized fair market values and equity scenarios. Additionally, we can further optimize the hyperparameters and consider more complex neural networks architectures. For instance, a recurrent neural network could be more adapted for variable annuities as it helps capture path/time

dependencies.

Improving the performance metrics used for surface mapping is also required. In fact, to assess the performance of the different trained models and to choose the optimal hyperparameters, we averaged the relative errors on test samples generated according to the same grid points for both the option's moneyness and time to maturity. It would be more interesting to calculate the errors on other points of the surface. This new error will reflect, in addition to the training error, the model's interpolation error.

Moreover, the second calibration approach, although theoretically attractive, still needs to be researched and improved. In particular, using an adaptive gradient descent method (adapting the learning rates using the order of magnitude of the parameters) is recommended. Its convergence rate also depends on the initialization point of the search and could be further investigated.

We finally highlight that we strongly advocate this separation of pricing and calibration in a neural network-based calibration framework. Although, it might be tempting to apply a neural network to directly estimate model parameters from market prices without using a pricing function based on a traditional model. This approach imposes several issues. As explained by Horvath et al.[17] these approaches might not align with the prevailing regulatory requirements due to their lack of interpretability. Furthermore, it is more difficult to prove their stability and robustness as required by regulators.

Annex 1: The Heston Model's Characteristic Function

We recall Heston's equations:

$$\begin{aligned} dS_t &= rS_t dt + \rho\sqrt{V_t}S_t dB_t + \sqrt{1-\rho^2}\sqrt{V_t}S_t dW_t \\ dV_t &= k(\theta - V_t) dt + \sigma\sqrt{V_t}dB_t \end{aligned} \quad (5.3)$$

The explicit formula for the Heston model's characteristic function f is obtained by applying Itô's Lemma to $f(t, S_t, V_t)$ and multivariable Taylor series expansion (see [8] for the full proof). We obtain the explicit following formula, where $T - t = \tau$:

$$\begin{aligned} f(i\phi) &= e^{A(\tau)+B(\tau)S_t+C(\tau)V_t+i\phi S_t} \\ A(\tau) &= ri\phi\tau + \frac{k\theta}{\sigma^2} \left[-(\rho\sigma i\phi - k - M)\tau - 2 \ln \left(\frac{1 - Ne^{M\tau}}{1 - N} \right) \right] \\ B(\tau) &= 0 \\ C(\tau) &= \frac{(e^{M\tau} - 1)(\rho\sigma i\phi - k - M)}{\sigma^2(1 - Ne^{M\tau})} \end{aligned} \quad (5.4)$$

Where

$$\begin{aligned} M &= \sqrt{(\rho\sigma i\phi - k)^2 + \sigma^2(i\phi + \phi^2)} \\ N &= \frac{\rho\sigma i\phi - k - M}{\rho\sigma i\phi - k + M} \end{aligned}$$

Annex 2: The free boundary problem: american call option PDE

Let $u(t, x)$ the pricing function of an american call option with underlying x . $u(t, x)$ verifies the following PDE, with \mathcal{L} a differential operator.

$$\begin{aligned} \partial_t u(t, x) + \mathcal{L}u(t, x) &= 0, \quad (t, x) \in [0, T] \times \Omega \\ u(0, x) &= u_0(x), \quad x \in \Omega \\ u(t, x) &= g(t, x), \quad x \in [0, T] \times \partial\Omega \end{aligned} \quad (5.5)$$

where $\partial\Omega$ is the boundary of the domain Ω . We approximate the function $f(t, x)$ by minimising an L^2 error based on the previous pricing PDE. This error is reflected by the following loss function:

$$J(f) = \|\partial_t f + \mathcal{L}f\|_{2,[0,T] \times \Omega}^2 + \|f - g\|_{2,[0,T] \times \partial\Omega}^2 + \|f(0, \cdot) - u_0\|_{2,\Omega}^2 \quad (5.6)$$

Annex 3: Valuation formulas for the G MDB guarantee

Let S_t the value of the fund at time t and α its continuously compounded dividend yield. For simplicity we take $S_0 = 1$. The mortality risk x is a random variable with cumulative

distribution F_x and a probability density $f(t)$.

The undiscounted insurer's engagement for GMDB guarantees at maturity $T = t_{death}$, V_T can be expressed as following:

The premium return:

$$\begin{aligned}
E_x [V_T] &= E_x [\max(1 - S_T, 0)] \\
&= E_x [E [\max(1 - S_t, 0) \mid T = t]] \\
&= E_x [\text{Put}(1, 1, t)] = \int_0^\infty \text{Put}(1, 1, t) dF_x(t) \\
&= \int_0^\infty \text{Put}(1, 1, t) f(t) dt \\
&= e^{-rt} \mathcal{N}(-d_2 \sqrt{t}) - e^{-\alpha t} \mathcal{N}(-d_1 \sqrt{t})
\end{aligned} \tag{5.7}$$

where \mathcal{N} the Gaussian cumulative distribution function and $d_1 = \frac{r - \alpha + \frac{1}{2}\sigma^2}{\sigma}$ and $d_2 = d_1 - \sigma$

The rollup option with rate r_{RU} :

A similar formula can be derived for the rollup option we only need to adapt the strike.

The ratchet option:

For the GMDB guarantee with a ratchet option, the put is replaced by a look-back option :

$$E_x [V_T] = \int_0^\infty GSG(t \mid \sigma, r, \alpha) f(t) dt \tag{5.8}$$

where GSG is the Goldman, Sosin and Gatto look-back option valuation[14].

$$GSG(t \mid \sigma, r, \alpha) = e^{-rt} \mathcal{N}(-\xi_2 \sqrt{t}) - e^{-\alpha t} \mathcal{N}(-\xi_1 \sqrt{t}) - \eta \left(e^{-rt} \mathcal{N}(\xi_3 \sqrt{t}) - e^{-\alpha t} \mathcal{N}(\xi_1 \sqrt{t}) \right)$$

$$\text{where } \eta = \frac{\sigma^2}{2(r-\alpha)} \quad \xi_1 = \frac{r-\alpha+\frac{1}{2}\sigma^2}{\sigma} \quad \xi_2 = \xi_1 - \sigma \quad \xi_3 = \xi_1 - \frac{2(r-\alpha)}{\sigma}$$

Annex 4: Examples of metamodeling approaches for VA pricing in the literature

Publication	EDM	Metamodel
Gan (2013)	Clustering	Kriging
Gan and Lin (2015)	Clustering	Kriging
Gan (2015)	LHS	Kriging
Hejazi and Jackson (2016)	Uniform sampling	Neural network
Gan and Valdez (2016)	Clustering, LHS	GB2 regression
Gan and Valdez (2017a)	Clustering	Gamma regression
Gan and Lin (2017)	LHS	Kriging
Hejazi et al. (2017)	Uniform sampling	Kriging, IDW, RBF
Gan and Huang (2017)	Clustering	Kriging
Xu et al. (2018)	Random sampling	Neural network
Gan and Valdez (2018c)	Clustering	GB2 regression
Gan (2018)	Random sampling	Linear model
Gan et al. (2018)	Clustering	Regression trees
Doyle and Groendyke (2018)	Not disclosed	Neural network

Annex 5: Synthetic portfolio descriptive statistics

	mean	min	25%	50%	75%	max
Rollup rate	0.01	0.00	0.00	0.00	0.05	0.05
Guaranteed benefit	313720.97	50001.72	179977.79	303612.02	427668.27	989145.57
Gmwb Balance	45778.27	0.00	0.00	0.00	34772.29	499708.73
Withdrawal rate	0.01	0.00	0.00	0.00	0.05	0.05
Total withdrawal	27775.21	0.00	0.00	0.00	7614.84	499585.73
Fund 1 Value	25926.72	0.00	0.00	7475.51	38170.53	847749.69
Fund 2 Value	25379.76	0.00	0.00	7546.12	37435.13	844322.70
Fund 3 Value	16985.49	0.00	0.00	4333.06	23510.50	580753.42
Fund 4 Value	14198.31	0.00	0.00	3669.26	20223.12	483936.90
Fund 5 Value	20425.64	0.00	0.00	6419.48	31049.96	494381.61
Fund 6 Value	25899.64	0.00	0.00	7586.89	38076.04	872706.64
Fund 7 Value	20974.33	0.00	0.00	5879.06	30165.74	634819.08
Fund 8 Value	19494.57	0.00	0.00	5522.71	28586.55	562485.37
Fund 9 Value	19304.74	0.00	0.00	5390.37	27400.15	663196.22
Fund 10 Value	20454.52	0.00	0.00	5965.87	30263.87	555121.41
Fair market value	93856.41	-69938.25	758.21	36989.80	104331.70	1784549.09
Time to maturity	14.53	0.59	10.34	14.51	18.76	28.52
Policyholder age	49.50	34.50	42.00	49.50	57.00	64.50
Policy age	7.47	0.50	4.00	7.50	11.01	14.42
Fund fees	0.03	0.00	0.01	0.03	0.04	0.05
qx	0.00	0.00	0.00	0.00	0.00	0.01
Equity adjusted vol	0.49	0.00	0.24	0.49	0.74	0.92
Equity adjusted close	6660.77	0.00	3380.84	6668.45	10241.61	12469.24
Number of funds	-	0.00	3.00	5.00	8.00	10.00
Guarantee moneyiness	0.25	-1.53	-0.01	0.27	0.45	1.00
Ratchet option	0.53	0.00	0.00	1.00	1.00	1.00

Table 5.3 – Synthetic portfolio descriptive statistics

Annex 6: Bump Method

Let Z be a random variable on $(\Omega, \mathcal{F}, \mathbb{R}) \rightarrow (\mathbb{R}, \mathcal{B}(\mathbb{R}))$.

Let F be a function $F : (x, Z) \mapsto F(x, Z)$, $x \in I \subset \mathbb{R}$ and $F(x, Z) \in \mathcal{L}^2(\Omega), \forall x \in I$. Define f as:

$$f(x) = \mathbb{E}[F(x, Z)]$$

The Bump method refers to the following approximation of $f'(x)$.

$$\begin{aligned} f'(x) &\stackrel{\varepsilon \rightarrow 0}{\approx} \frac{f(x + \varepsilon) - f(x - \varepsilon)}{2\varepsilon} \\ &\approx \frac{1}{M} \sum_{i=1}^M \frac{F(x + \varepsilon, Z_i) - F(x - \varepsilon, Z_i)}{2\varepsilon} \end{aligned}$$

Bibliography

- [1] Fast and accurate implied volatility solver. <https://chasethedevil.github.io/post/fast-and-accurate-implied-volatility-solver/>.
- [2] Ferdinando Ametrano, Luigi Ballabio, et al. Quantlib - a free/open-source library for quantitative finance, 2003. Last accessed 16 September 2017.
- [3] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 2012.
- [4] Hans Buehler, Lukas Gonon, Josef Teichmann, and Ben Wood. Deep hedging. *arXiv: Computational Finance*, 2018.
- [5] Patrick Büchel, Michael Kratochwil, et al. Deep calibration of financial models: Turning theory into practice. 2020.
- [6] G. Cybenko. Approximation by superpositions of a sigmoidal function. 1989.
- [7] Daniel Doyle and Chris Groendyke. Using neural networks to price and hedge variable annuity guarantees. *Risks*, 2019.
- [8] R. Dunn, Paloma Hauser, Tom Seibold, and Hugh Gong. Estimating option prices with heston ' s stochastic volatility model. 2014.
- [9] Black Fischer and Myron Scholes. The pricing of options and corporate liabilities. *Journal of Political Economy*, 81(3):637–654, 1973.
- [10] Matteo Gambarara and Josef Teichmann. *Consistent Recalibration Models and Deep Calibration*. 2021.
- [11] Guojun Gan and Emiliano Valdez. Valuation of large variable annuity portfolios: Monte carlo simulation and synthetic datasets. 2017.
- [12] Guojun Gan and Emiliano Valdez. *Computational Problems in Variable Annuities*. 2019.
- [13] R. Gencay and Min Qi. Pricing and hedging derivative securities with neural networks: Bayesian regularization, early stopping, and bagging. *IEEE Transactions on Neural Networks*, 2001.
- [14] M Barry Goldman, Howard B Sosin, and Mary Ann Gatto. Path dependent options: "buy at the low, sell at the high". *Journal of Finance*, 1979.
- [15] Andres Hernandez. *Model calibration with neural networks*. Risk, 2017.

- [16] Steven L Heston. A Closed-Form Solution for Options with Stochastic Volatility with Applications to Bond and Currency Options. *Review of Financial Studies*, 1993.
- [17] Blanka Horvath, Aitor Muguruza, and Mehdi Tomas. Deep learning volatility. 2019.
- [18] James M. Hutchinson, Andrew W. Lo, and Tomaso Poggio. A nonparametric approach to pricing and hedging derivative securities via learning networks. *The Journal of Finance*, 1994.
- [19] A Itkin. Deep learning calibration of option pricing models: some pitfalls and solutions, 2019.
- [20] Owen Jelf, Tej Patel, et al. Frtb, the road to 2023. 2021.
- [21] Aymeric KALIFE. Variable annuities. www.institutdesactuaire.com/global/gene/link.php?doc_id=10043&fg=1, 2017.
- [22] Alexander Ke and Andrew Yang. Option pricing with deep learning. 2019.
- [23] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.
- [24] Shuaiqiang Liu, Cornelis Oosterlee, and Sander Bohte. Pricing options and computing implied volatilities using neural networks. *Risks*, 2019.
- [25] Maryanthe Malliaris and Linda Salchenberger. A neural network model for estimating option prices. *Applied Intelligence*, 2004.
- [26] Warren McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 1943.
- [27] Justin Sirignano and Konstantinos Spiliopoulos. Dgm: A deep learning algorithm for solving partial differential equations. *Journal of Computational Physics*, 2018.
- [28] Jasper Snoek and Hugo Larochelle. Practical bayesian optimization of machine learning algorithms. 2012.
- [29] Mehdi Tomas. Pricing and calibration of stochastic models via neural networks. Master’s thesis, Imperial College London, 2018.
- [30] Simon Topping, Paul McSheaffrey, et al. Implementation of the fundamental review of the trading book in hong kong. 2019.
- [31] Jingtao Yao, Yili Li, and Chew Lim Tan. Option price forecasting using neural networks. *Omega*, 2000.

Executive summary

Introduction

Regulators are imposing more stringent requirements on both banks and insurance companies in monitoring their key risk metrics. For instance, the Fundamental Review of The Trading Book (FRTB) under the internal model approach (IMA) now, requires the calculation of expected shortfall (ES) and stressed ES with multiple liquidity horizons for each risk category, compared to a single liquidity horizon and calculations of value at risk (VaR) and stressed VaR under Basel III.

These requirements increase by more than tenfold the computational requirements to assess market risk capital and make the management of portfolios with highly complex and structured products a challenging task. Therefore, exploring new approaches adaptable for high-dimensional problems that challenge Monte Carlo (MC) simulations accuracy and outperform their computational efficiency becomes a crucial goal for the industry.

In this report, we use neural networks (NN) as universal approximators, to learn the pricing function on a given parameters space of interest. We present two approaches to tackle commonly used option pricing models calibration differently. Considerable calculations speedups compared to the MC approach can be achieved by trading off compute time for training with inference time for pricing and thus induce drastic improvements in both the financial and insurance sectors. For instance, this approach can make live exotic option pricing a realistic goal and model calibration an easier task especially for products that require intra-day valuations of their key risk metrics. This can typically be useful for insurers managing and hedging the risks associated with Variable Annuity (VA) contracts. Although MC simulations are widely adopted by insurance companies for the valuation of their VA portfolios, the complex structure of these contracts makes their accurate valuation a computationally demanding task.

Learning a model price function with a neural network

A preliminary step of the project is a proof of concept on vanilla products (call and put options), to see if this approach can be considered for more complex products. Although, our implementation is flexible and can be adapted to different diffusion models and products. We first analyzed the results for the pricing of European vanilla options in the Black-Scholes model and a stochastic volatility model (Heston). Then extended the approach to Asian options. Unlike market approaches that aim to directly approximate the market prices (model-free approaches), our work will focus on model pricing. We train a neural network to learn a model's price function on a given input space. This choice is justified by the fact that we do not wish to introduce a 'black box' model to predict market prices. But we

rather wish to stick to a model and challenge the current methods mainly MC approaches. Two approaches are mentioned in the literature:

- A supervised approach based on minimizing the difference between the predicted and the target prices.
- An unsupervised approach consisting of minimizing a loss function, conveniently defined, based on a partial differential equation (PDE) verified by the option prices. We do not need to compute a target output y and train the NN to map the input parameters to the output (as in the supervised approach).

We briefly explored the second approach via the example of the deep Galerkin method (DGM). But we focused on the first one, which we developed in this report. Since it was more aligned with the next step (deep calibration).

Data generation was a crucial step. We tried different sampling methods to generate the training data. Using a uniform distribution, caused the training dataset to be very unbalanced, with many samples being purely theoretical. The model's performance was therefore highly correlated to the moneyness of the option. Consequently, we built different models by moneyness class to improve the model's accuracy.

With more hindsight, this seems problematic as it favors overfitting and undermines the approach's generality.

Used to a truncated normal distribution globally improved the model's accuracy. Finally, fixing the moneyness of the generated samples to a given grid, yielded more financially consistent samples and drastically improved the model's performance.

We used two types of metrics to assess the model's performance: averaged absolute errors and relative errors, and used k-fold cross-validation to avoid overfitting. After the tuned NN has proved to correctly map the pricing function for European vanilla options (The average absolute error equals 0.1% of the strike price and the average relative error was 6%), we tested the approach on more complex options for example Asian options.

In light of the models' performances, the deep pricing approach is promising. It can correctly learn and map the pricing function on a given parameters space of interest and instantaneously predict the price surface for different market settings ("fast surface"). Once the NN is trained, computationally expensive MC simulations can be discarded. We should finally highlight that in this approach, the sampling method is crucial. It's the case for both the supervised and unsupervised approaches. One could criticize the fact that the training samples, generated randomly, do not reproduce the historical market prices representativity. But this can be an advantage in stressed scenarios for example, where market settings are generally not historical, but rather unforeseen rare scenarios.

A major limitation of this approach is the fact that the model cannot by default, guarantee no-arbitrage principle.

Predicting price surfaces rather than individual prices to help the model did not guarantee no-arbitrage. It only ensures more smooth surface predictions. A classical approach consists of regularization. In fact, the option pricing theory provides necessary and sufficient conditions for options to be arbitrage-free. These constraints are added as new penalty terms to the loss function.

The deep calibration approach: presentation and limits

In the next step, we focus on solving the inverse problem. Given a price surface/volatility surface, we search for the model parameters such that the model prices best approximate the given surface according to an appropriate metric. Often the \mathbb{L}^2 . Many approaches are to consider in solving this problem from a machine learning point of view.

- Directly learning the map from the data (prices and observables variables) to the model parameters. This method was not retained and we justified this choice based on its limits.
- Learning the pricing map and then replacing the pricing functional with the learned "fast surface" to solve the inverse problem using the classical minimization technics.
- Learning the pricing map ("fast surface") and leveraging the fact that the NN is a nested function of the input parameters whose gradient is known analytically. The gradient is known not only for the weights and biases but also for the input parameters (model parameters + observables parameters) thanks to the backpropagation principle. Hence we can apply the gradient descent algorithm to minimize the \mathbb{L}^2 loss with respect to the model parameters.

We trained a "fast surface" NN. For that, we artificially generated the training dataset: the parameters are sampled from uniform distributions whose extremes are defined a priori (This defines our space of interest). We also used a fixed grid for both option's moneyness ($m = \frac{S_0}{K}$) and time to maturity (ttm). Then, we generate the model price associated with each set of parameters.

Once the "fast surface" NN is trained and tuned, we use it to test the two calibration approaches. We generated Heston implied volatility surfaces for dates from 31/12/2020 to 30/01/2121 using *Quantlib*

The model parameters are known and saved. We mask this information and apply the two previously described approaches to predict them. We then compare the implied volatility surface (IVS) generated using the predicted parameters and the target IVS to assess the approach's performance.

Both approaches are very quick to calibrate surfaces. The first approach calibrates well to the target surface. The second method seems to have problems with the calibration of the kappa and more generally when there is a significant difference in scale between the parameters to be calibrated. Opting for different learning rates to update the input parameters in the gradient descent algorithm would probably help. This method's convergence also depends on the initialization. For that, it's not very robust. These limitations need to be further analyzed.

Our approach has several other limitations that can be improved: First, to assess the performance of the different trained models and to choose the optimal hyperparameters, we averaged the relative errors on a test sample generated according to the same approach and therefore the same grid points for both the option's moneyness and time to maturity. It would be more interesting to calculate the errors on other points of the surface. This new error will reflect, in addition to the training error, the model's interpolation error.

Regarding the calibration approach, the method using the "fast surface" to optimize the

loss function by traditional optimization algorithms is globally more efficient than the second approach, which we tried to formalize. Both approaches save a considerable amount of time compared to traditional calibration methods (On average the calibration time of a one-month history is 1s for the first approach and 600ms for the second one).

The second calibration approach, although attractive, still needs to be researched and improved, in particular by using an adaptive gradient descent method (adapting the learning rates using the order of magnitude of the parameters). Its convergence rate also depends on the initialization point of the search and could be further improved. It would also be interesting to rethink the convergence criterion and to find a compromise between the speed of convergence and the accuracy of the calibration.

Although it is highly recommended to build this approach on synthetic data for the training phase of the NN, the approach should be validated on market data for the calibration phase. Knowing the real parameters that generate the surfaces to be calibrated allowed us to better identify the flaws of each approach. We applied these calibration techniques on CAC 40 market volatility surface, the results were comparable, with higher accuracy for the first approach.

Potential applications of the approach

To sum up, stricter regulatory requirements introduce a significant computational challenge to which banks need to adapt their current methods. The "fast surface" approach can significantly alleviate this computational burden.

The deep learning approach introduced in this work can also be applied to stress tests. Since the network is trained on synthetic data, it does not introduce any historical bias to value a portfolio in an extreme market configuration.

The calibration approach can be applied in the context of model validation/monitoring, for instance, for front-end pricers, in the context of P&L assessment. The trained network learns to replicate the model used in the pricers. Thus, it can find the calibrated model parameters from the generated data and validate their relevance in a monitoring/audit stage. We finally highlight that we strongly advocate this separation of pricing and calibration in a neural network-based calibration framework. Although, it might be tempting to apply a neural network to directly estimate model parameters from market prices without using a pricing function based on a traditional model. This approach imposes several issues. These approaches might not align with the prevailing regulatory requirements due to their lack of interpretability.

Variable Annuities deep pricing application

We Finally adapted the deep pricing approach to the case of insurance contracts with complex guarantees, for instance, variable annuities. We trained a neural network to predict Monte Carlo market fair values from a set of inputs generally fed to the simulations. Although the model seemed to perform quite well with the different types of guarantees we tested (an average relative error below 5%), the error distribution showed relatively fat tails with some extreme values that could not adjust even after preprocessing the outliers. Signif-

icant calculations speedups compared to the Monte Carlo simulations are achievable, once the neural network is correctly trained. This is typically the case for greeks calculations and SCR (Solvency Capital Requirement) estimation which generally involves nested stochastic simulations. We believe training the neural network on different economic scenarios, including previously realized fair market values and trying recurrent neural networks(which are excellent in capturing path dependencies), would help achieve higher model accuracy.