Imperial College
London

IMPERIAL COLLEGE LONDON

DEPARTMENT OF MATHEMATICS

Deep intensity-based CVA with Wrong Way Risk

Author:
Wiam El Mouden (CID: 01948802)

A thesis submitted for the degree of

MSc in Mathematics and Finance, 2020-2021

Declaration

The work contained in this thesis is my own work unless otherwise stated.

Abstract

Wrong Way Risk (WWR) is a heavily model-dependent calculation where we price an option with random maturity - namely the counterparty CVA - on the positive exposure of a portfolio netting set when there can be correlation between the triggering credit event and the portfolio risk factors. This results in very slow simulations. Our purpose is to see whether Deep Learning can learn the simulation results and accelerate dramatically the calculation. This thesis studies CVA for at-the-money European call options under stochastic default intensity models. CVA in presence of Wrong Way Risk (WWR) is embedded in the correlation between risk factors and default intensity. We develop a Monte Carlo framework to simulate CVA and then train a neural network, using the data simulated, that will be able to calculate CVA quickly.

Keywords: Credit Value Adjustment, Wrong Way risk, Numerical simulation, Cox-Ingersoll-Ross process, Intensity models, Universal Approximation Theorem.

To my parents, Khadija & Aziz
To my sisters, Ikram & Fatima Ezzahra

Acknowledgements

I would like to thank Dr Damiano Brigo, my thesis supervisor, and excellent lecturer in Interest Rates Models. Thank you for guiding me and providing me with the tools needed to complete my thesis.

My sincere gratitude goes to Hafsae Tabti, Mohamed Taik and Ivan Schonenberger for their precious help and advice this year.

Finally, I would like to thank my parents and my two sisters Ikram and Fatima Ezzahra. I wouldn't have made it this far without you. Thank you for always believing in me and supporting my choices. I will always be grateful to have you in my life.

Contents

List of Figures

List of Tables

1 Introduction

Before the financial crisis of 2008, the aspect of counterparty credit risk in financial contracts wasn't popular and had often been overlooked. However, due to the high number of default events experienced by a lot of financial institutions and the credit quality deterioration, counterparty credit risk has proven to be one of the main driving force of the credit crisis. Several firms had huge mark-to-market (MtM) losses in their trading books. In 2008, Lehman Brothers, one of the largest investment banks declared bankruptcy, which was the largest in the U.S. history. At the time of the default, Lehman Brothers had around half a million derivative contracts with close to 8,000 different counterparties. The majority of these counterparties most likely never considered a scenario where Lehman Brothers would default.[13]

As a consequence of the financial crisis, new financial regulation was beginning to take shape around the practices of banks and it is now common practice to take the counterparty credit risk into account when valuing some derivatives. This adjustment of the price is known as the Credit Value Adjustment (CVA). In other words, CVA represents the discount to the standard derivative value that a buyer would offer after taking into account the possibility of a counterparty's default. [1]

It has been proven that CVA was a key risk driver behind the losses experienced during the crisis. The report from BIS (2011)[2] notices that "Under Basel II, the risk of counterparty default and credit migration risk were addressed but mark-to-market losses due to credit valuation adjustments (CVA) were not. During the financial crisis, however, roughly two-thirds of losses attributed to counterparty credit risk were due to CVA losses and only about one-third were due to actual defaults." This emphasises the crucial value of proper CVA valuation.

Computing CVA has become of great importance in the financial industry. One difficulty in pricing CVA concerns the dependency between exposure and counterparty credit quality, which is known as Wrong/Right Way Risk (WWR). When computing CVA, the correct inclusion of WWR is still a major concern. Many different approaches have been proposed to assess WWR. We focus in our work on the dependence between the counterparty default and general market risk factors using stochastic intensity models. (see [26] for the Firm Value models approach)

In this paper, we study CVA calulation for at-the-money European call options under stochastic default intensity models, i.e., the CIR++ process which provides a natural and effective framework to handle the correlation between the underlying asset and the default. We develop an efficient Monte Carlo framework for pricing CVA. Then, we study how CVA computation can be accelerated using Deep Learning. We will build a neural network to learn the impact of the correlation and other parameters of the intensity model on the CVA value.

[1]`https://www.risk.net/definition/credit-valuation-adjustment-cva`

The outline of this thesis is as follows. In Chapter 2, we describe CVA valuation problem in general terms. Chapter 3 describes the stochastic intensity model and underlying asset driven process. In Chapter 4, we present the general framework for CVA pricing based on Monte Carlo simulation. In Chapter 5, we present the simulation results. Chapter 6 is an introduction to Deep learning. In Chapter 7, we build step by step a Neural Network (NN) to calculate WWR CVA. In Chapter 8, we present NN results. Conclusion is summarized in Chapter 9.

2 Credit Value Adjustment

In this chapter, we will introduce the mathematical concepts of CVA needed for our study.

2.1 Credit valuation adjustment

Credit valuation adjustment is one of the most important counterparty credit risk measures. Indeed, according to Basel III, banks are required to hold regulatory capital based on CVA charges against each of their counterparties [11].

CVA can be defined at time t, as the difference of the default risk-free price $\Pi(t;X)$ and the true price $\tilde{\Pi}(t;X)$ of a financial contract X, thus

$$\text{CVA} = \Pi(t;X) - \tilde{\Pi}(t;X)$$

A better or more technical definition of CVA is in terms of an expected future loss due to default of a counterparty. It is important to point out that since CVA is concerning valuation rather than risk measuring, the calculations are done under the risk-neutral measure following the standard valuation principles.[16]

To calculate the CVA, we follow the derivation in Brigo et al. (2013) [6]. Lets consider a fixed time horizon $T > 0$ and a probability space $(\Omega, \mathcal{G}, \mathcal{G}_t, \mathbb{Q})$ where Ω is the outcome space. $\mathcal{G}_t$ represents the flow of information on whether default occurred before t and if so at what time exactly. Furthermore, we denote by $\mathcal{F}_t$ the filtration of default-free market variables, i.e $\mathcal{F}$ is a σ-algebra such that the filtration $\mathcal{F}_t$ contains the same information as $\mathcal{G}_t$ but the default events. Formally, we assume that :

$$\mathcal{G}_t = \mathcal{F}_t \vee \sigma(\{\tau \leq u\}, 0 \leq u \leq t)$$

where τ is the default time of the counterparty.

In this setup, $\mathbb{Q}$ corresponds to the risk-neutral measure such that under the risk-free bank account numeraire B_t following the process :

$$dB_t = B_t r_t dt, B_0 = 1 \ ,$$

with r_t the risk free interest rate and all discounted tradeable assets (i.e. assets divided by B_t) are martingales.

Lets consider a portfolio of derivative contracts with a risky counterparty up to a maturity T, which risk-free value is denoted by $V(t,T)$. Suppose that the counterparty defaults at time $\tau < T$ and we can recover Rec percent of our exposure (Rec is the recovery rate). Then at $t = 0$, we have a discounted loss L of :

$$\begin{aligned} L &= 1_{\{\tau \leq T\}}(1 - Rec)\frac{B_0}{B_\tau}(V(\tau,T))^+ \\ &= 1_{\{\tau \leq T\}}(1 - Rec)D(0,\tau)(V(\tau,T))^+ \end{aligned}$$

where $D(t,T)$ is the discount factor B_t/B_T. CVA is then the risk-neutral expectation of this loss. Finally, we get that the CVA at time $t=0$ can be expressed as follows:

$$\text{CVA} = \mathbb{E}^{\mathbb{Q}}\left[1_{\{\tau\leq T\}}(1-Rec)D(0,\tau)(V(\tau,T))^{+} \mid \mathcal{G}_0\right]. \tag{1}$$

We can see that CVA is in fact an option on the residual value of a portfolio, with a random maturity given by the default time of the counterparty τ. Hence, CVA is a strictly positive value which is subtracted from the price. This is very intuitive, since entering into an agreement with a more risky counterparty would lead to a larger CVA and thus a lower price.

Computing CVA is challenging. Counterparty risk adds a level of optionality to the payoff. In particular, model independent products become model dependent also in the underlying market [3]. Hence, calculating CVA of a product does require a model.

A widely adopted assumption is that credit exposure, V and the counterparty's default time, τ are independent. In this case, we can express the CVA in terms of the density function f of τ as follows:

$$\begin{aligned}\text{CVA} &= \int_0^T \mathbb{E}^{\mathbb{Q}}\left[(1-Rec)D(0,\tau)(V(\tau,T))^{+} \mid \tau=t\right] f(t)dt \\ &= \int_0^T \mathbb{E}^{\mathbb{Q}}\left[(1-Rec)D(0,t)(V(t,T))^{+}\right] f(t)dt\end{aligned}$$

where the last equality follows from the independence of V and τ. In practice, a counterpary's default time distribution is approximated from counterparty credit spreads observed in the market. Monte Carlo simulation is used then to estimate independent CVA by estimating $\mathbb{E}^{\mathbb{Q}}[D_t V_t^{+}]$, based on a discrete time grid. However the efficacity of the independent CVA is limited, since there a several important practical cases, where credit exposure and the counterparty's default time are correlated (see [14]).

2.2 Wrong Way Risk

When credit exposure is negatively correlated with a counterparty's credit quality, the exposure and its associated risk measures are said to be wrong way. In other terms, Wrong-way risk (WWR) is when the exposure to a counterparty increases with the risk of default of the counterparty. Wrong Way CVA refers to CVA in presence of wrong way risk.

A basic example of wrong way risk is when an investor has taken a long position in a put option on the underlying stock S_C. Suppose the counterparty of the option is the same company as of the underlying stock. We name it company C. Lets consider a scenario where company C runs into some financial difficulties. The stock value is likely to drop, and as a consequence the investor's put option will increase in value.

Due to company C's financial difficulties, the risk that they won't be able pay for the put option at maturity is increasing. Therfore, from the investor's point of view, both the exposure and risk to counterparty C has increased significantly [16]. This is known as wrong way risk.

Similarly, when credit exposure is positively correlated with a counterparty's credit quality, the exposure and its associated risk measures are said to be right way. In order to capture WWR/RWR it is crutial to have a model that models the dependence between underlying market risk factors and credit risk factors. For instance in the example above, we can try to capture the dependence between the default risk and the stock price of company C.

3 CVA calculation under stochastic intensity models

In this chapter, we will present the credit and default correlation models, and we will derive a formula for the CVA of an at-the-money call option in this set-up. We choose to work with an at-the-money call option so that the CVA term would be relevant. In order to keep the computation tractable, we consider a square root diffusion model driving the intensity of the investor and counterparty.

3.1 Credit spread model

We model the stochastic intensity λ_t as follows:

$$\lambda_t = y_t + \psi(t;\beta), \quad t \geq 0 \tag{2}$$

where the intensity has a random component y and a deterministic component ψ to fit the CDS term structure.

We take each y to be a Cox Ingersoll Ross (CIR) process:

$$dy_t = \kappa(\mu - y_t)\,dt + \nu\sqrt{y_t}dZ_t \tag{3}$$

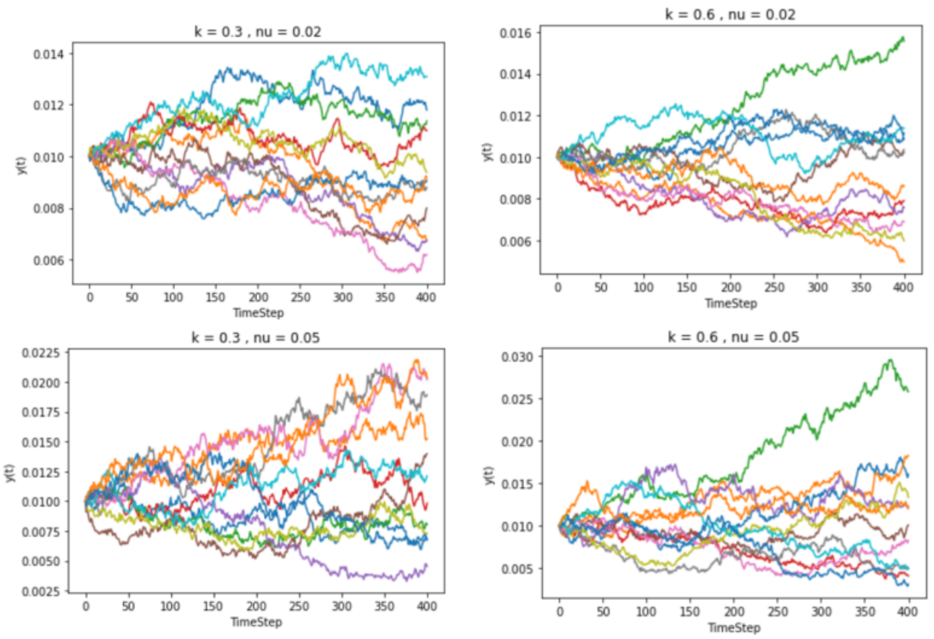


Figure 1: Sample paths of the CIR process y , with $\mu = 0.03, y_0 = 0.01$

where the parameter vector is $\beta = (\kappa, \mu, \nu, y_0)$ and each parameter is a positive deterministic constant. κ corresponds to the speed of mean reversion of the process,

μ the long term mean reversion level and ν the volatility. As usual, Z is a standard Brownian motion process under the risk neutral measure. We choose the CIR process to model the intensity (instead of a Varsicek model for instance) because it has the important proprety : $y > 0$ as must be for an intensity model.

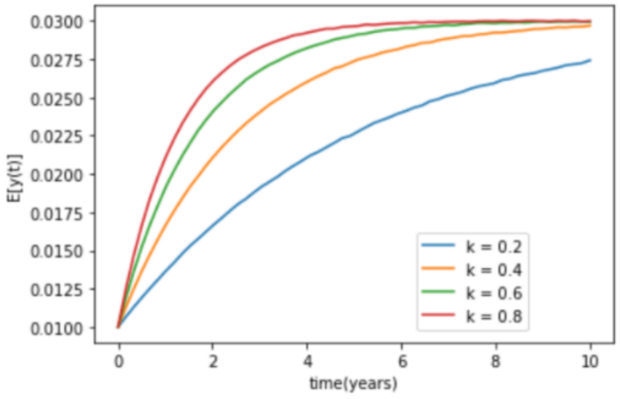


Figure 2: $E[y(t)]$, with $\mu = 0.03, y_0 = 0.01$

We also define the integrated quantities :

$$\Lambda(t) := \int_0^t \lambda_s ds, \quad Y(t) := \int_0^t y_s ds, \quad \Psi(t,\beta) := \int_0^t \psi(s,\beta)\, ds.$$

The default time τ is defined as the inverse of the cumulative intensity on an exponential random variable ξ with mean 1 and independent of λ :

$$\tau = \Lambda^{-1}(\xi) \tag{4}$$

Calibration to CDS Market Quotes
If we can read from the market some implied risk-neutral default probabilities, and associate to them implied hazard functions Γ^{Mkt}, we may wish our stochastic intensity model to agree with them [3]. Since the survival probabilities in our model are given by :

$$\begin{aligned}\mathbb{Q}(\tau > t)_{\text{model}} &= \mathbb{E}_0\left[e^{-\Lambda(t)}\right] \\ &= \mathbb{E}_0\left[\exp\left(-\Psi(t,\beta) - Y(t)\right)\right] \\ &= \exp\left(-\Psi(t,\beta)\right)\mathbb{E}_0\left[\exp\left(-Y(t)\right)\right]\end{aligned}$$

For the model to agree with the market, we need to guarantee that :

$$\mathbb{Q}(\tau > t)_{\text{CDSmkt}} = \exp\left(-\Gamma^{\text{Mkt}}(t)\right) = \exp(-\Psi(t,\beta))\mathbb{E}_0\left[\exp\left(-Y(t)\right)\right] \tag{5}$$

Now notice that $\mathbb{E}_0\left[\exp(-Y(t))\right] = \mathbb{E}_0\left[e^{-\int_0^t y_s ds}\right]$ is simply the bond price for a CIR interest rate model with short rate given by y, so that it is known analytically. We denote it by $P^y(0,t,y_0,\beta)$.

Hence, λ is calibrated to the market implied hazard function Γ^{Mkt} easily if we set :

$$\Psi(t,\beta) := \Gamma^{\mathrm{Mkt}}(t) + \ln\left(P^y\left(0,t,y_0;\beta\right)\right) \tag{6}$$

where the parameters β are chosen so that we have a positive function ψ (i.e. a non-decreasing Ψ) [5].

3.2 Stock price model

In order to keep the computation tractable, we assume that the risk free interest rate r is constant and we consider the Black-Scholes framework: a market with one risk free bond and one risky asset with its price following a geometric Brownian motion

$$dS_t = rS_t dt + \sigma S_t dW_t \tag{7}$$

where σ is volatility of risky asset and W standard Brownian motion process under the risk neutral measure.

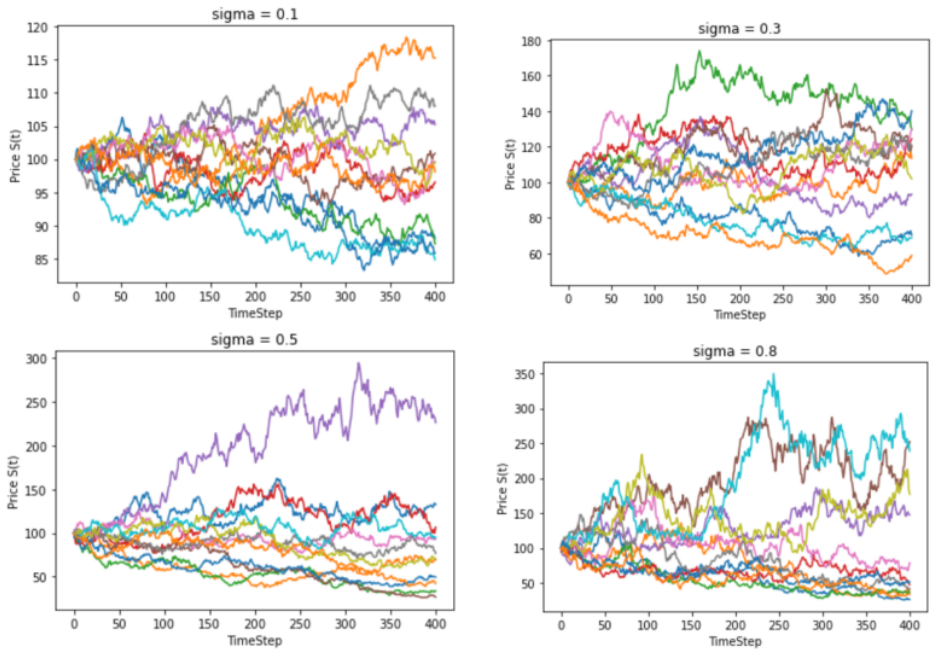


Figure 3: Sample paths of the stock price S

We recall that the Black-Scholes closed formula for European call option price at time 0 with strike K and maturity T in the Black-Scholes framework is given by:

$$C\left(S_0,K,r,\sigma,T\right) = S_0\Phi\left(d_1\right) - Ke^{-rT}\Phi\left(d_2\right) \tag{8}$$

where $d_1 = \frac{\log(S_0/K)+\left(r+\sigma^2/2\right)T}{\sigma\sqrt{T}}$, $d_2 = d_1 - \sigma\sqrt{T}$ and $\Phi(\cdot)$ is the standard normal cdf.

Finally, we assume that the stock price process S is correlated to the stochastic intensity λ via the brownian motion processes as follows :

$$dW_t dZ_t = \rho dt, \rho \in [-1,1] \tag{9}$$

In this setup, the default intensity λ is driven by a Brownian motion Z correlated to W such that :

$$W_t = \rho Z_t + \sqrt{1-\rho^2} Z_t^{\perp}, \rho \in [-1,1] \tag{10}$$

with $Z^{\perp}$ a Brownian motion independent of Z.

3.3 CVA

The CVA of the call option with final maturity T in this set-up is given by :

$$CVA = (1-\text{Rec})E_0\left[1_{\{\tau<T\}}D(0,\tau)(\text{NPV}(\tau))^+\right] \tag{11}$$

where $NPV(\tau)$ is the residual net present value (NPV) of the call at the default time τ of the counterparty. The residual NPV in our case is the expected value at time τ of the discounted payoff of the call, which is $\text{NPV}(\tau) = E_\tau\left[D(\tau,T)(S_T-K)^+\right]$

$$\begin{aligned}
CVA &= (1-\text{Rec})E_0\left[1_{\{\tau<T\}}D(0,\tau)\left(E_\tau\left[D(\tau,T)(S_T-K)^+\right]\right)^+\right] \\
&= e^{-rT}(1-\text{Rec})E_0\left[1_{\{\tau<T\}}E_\tau\left[(S_T-K)^+\right]\right] \\
&= e^{-rT}(1-\text{Rec})E_0\left[E_\tau\left[1_{\{\tau<T\}}(S_T-K)^+\right]\right] \\
&= e^{-rT}(1-\text{Rec})E_0\left[1_{\{\tau<T\}}(S_T-K)^+\right]
\end{aligned}$$

where we use the fact that $1_{\{\tau<T\}}$ is measurable for E_τ and we apply the tower property of conditional expectation. We also use that in case the risk free interest rate is constant, we have $D(0,t) = e^{-rt}$.

Now, recalling that $\tau = \Lambda^{-1}(\xi)$, we have :

$$\begin{aligned}
CVA &= e^{-rT}(1-\text{Rec})E_0\left[1_{\{\Lambda^{-1}(\xi)<T\}}(S_T-K)^+\right] \\
&= e^{-rT}(1-\text{Rec})E_0\left[E\left[1_{\{\xi<\Lambda(T)\}}(S_T-K)^+ | \Lambda, S\right]\right] \\
&= e^{-rT}(1-\text{Rec})E_0\left[(1-\exp(-\Lambda(T)))(S_T-K)^+\right]
\end{aligned}$$

Here again, we use the tower property and the fact that ξ follows an exponential distribution with mean 1.

Finally, we get :

$$CVA = (1-\text{Rec})(C(S_0,K,r,\sigma,T) - e^{-rT}E_0\left[\exp(-\Lambda(T))(S_T-K)^+\right]) \tag{12}$$

4 Numerical Evaluation

We use the standard Monte Carlo method to simulate the CVA of a call option. We fix the following parameters: $S_0 = 100$, $K = 100$, $T = 1$, $y_0 = 0.01$, $\mu = 0.03$ and $\Gamma^{\text{Mkt}} = 0.03$. For the sake of simplicity, the recovery rate Rec and the risk free interest rate r are assumed to be constant and set to zero to put the focus on the credit-exposure dependency.

4.1 Simulation procedure and results

To evaluate the CVA of an European call option with maturity T and exercise price K, we divide the time interval $[0, T]$ into n equal subintervals with grid points $t_0 < t_1 < \ldots < t_n$ satisfying $t_i = ih, i = 0, \ldots, n$, and $h = T/n$. We then generate simultaneously sample paths of S and y by first generating $2n$ independent $N(0,1)$ variables $Z_1, \ldots, Z_n$ and $Z_1^{\perp}, \ldots, Z_n^{\perp}$, and then computing :

$$S_{t_{i+1}} = S_{t_i} + rS_{t_i}h + \sigma S_{t_i}\sqrt{h}Z_{t_{i+1}}$$

$$y_{t_{i+1}} = y_{t_i} + \kappa\left(\mu - y_{t_i}\right)h + \nu\sqrt{y_{t_i}h}(\rho Z_{t_{i+1}} + \sqrt{1-\rho^2}Z_{t_{i+1}}^{\perp})$$

$$Y_{t_{i+1}} = Y_{t_i} + y_{t_i}h$$

for $i = 0, \ldots, n-1$, where $S_{t_0} = S_0$ and $y_{t_0} = y_0$ are respectively the initial asset price at time $t_0 = 0$ and the initial value of the CIR process y. This simulation procedure is called the explicit Euler-Maruyama scheme.

Finally, adding the shift : $\Lambda_n = Y_n + \Phi_n$ we compute :

$$CVA = C(S_0, K, r, \sigma, T) - e^{-\Lambda_n}(S_n - K)^+$$

which gives the value of CVA of a call option for this particular realization of the sample path. This completes one simulation. If we repeat it M times (M is a large number) and compute the average, we will get the CVA with Monte Carlo method. The table bellow displays the simulations results :

n	M	CVA	Standard error (std error)	time
10	10000	0.55288	0.196263	0.00728
100	10000	0.50830	0.200837	0.05526
10	50000	0.43660	0.088561	0.03716
100	50000	0.48732	0.089587	0.34122
10	100000	0.44439	0.062415	0.07893
100	100000	0.40429	0.064113	0.90955

Table 1: The results of MC simulation of the CVA for $\rho = 0.2$, $\sigma = 0.3$, $\kappa = 0.4$, $\nu = 0.04$

For the next simulations, we fix $n = 100$ and $M = 100.000$. Choosing a higher value for the sample size M can reduce the standard error of the simulation but is computationally costly, $M = 100.000$ is a good compromise. We also fix the seed the simulation to ensure we get the same results every time.

4.2 Numerical schemes

It is important to note that the explicit Euler-Maruyama scheme :

$$y_{t_{i+1}} = y_{t_i} + \kappa\left(\mu - y_{t_i}\right)h + \nu\sqrt{y_{t_i}h}(\rho Z_{t_{i+1}} + \sqrt{1-\rho^2}Z^{\perp}_{t_{i+1}})$$

can lead to negative values since the Gaussian increment is not bounded from below. Hence, this scheme is not well defined. To deal with this issue, we will introduce several discretization schemes. This section is based on the paper : "On the discretization schemes for the CIR (and Bessel squared) processes" by Aurélien Alfonsi [1]. To explain the different schemes, lets consider a process (X_t) following the CIR process :

$$dX_t = k(\theta - X_t)dt + \sigma\sqrt{X_t}dW_t \tag{13}$$

Fisrt, we consider the following scheme proposed by Deelstra and Delbaen [9]:

$$X_{t_{i+1}} = X_{t_i} + \frac{kT}{n}\left(\theta - X_{t_i}\right) + \sigma\sqrt{X_{t_i}1_{X_{t_i}>0}}(W_{t_{i+1}} - W_{t_i})$$

where we consider again the regular grid $t_i = \frac{iT}{n}$
Another scheme is the one introduced by Diop [10]:

$$X_{t_{i+1}} = |X_{t_i} + \frac{kT}{n}\left(\theta - X_{t_i}\right) + \sigma\sqrt{X_{t_i}}(W_{t_{i+1}} - W_{t_i})|$$

On the other hand, Brigo and Alfonsi [4] proposed an implicit scheme to assure the positivity of a CIR process. Rewriting the CIR process with the stochastic integral and using the fact that $d\langle\sqrt{X}, W\rangle_s = \frac{\sigma}{2}ds$, we get :

$$\begin{aligned}
X_t =& x_0 + \int_0^t k(\theta - X_s)\,ds + \sigma\int_0^t \sqrt{X_s}dW_s \\
=& x_0 + \lim_{n\to\infty}\left\{\sum_{i;t_i<t} k(\theta - X_{t_{i+1}})\frac{T}{n} + \sigma\sum_{i;t_i<t}\sqrt{X_{t_{i+1}}}\left(W_{t_{i+1}} - W_{t_i}\right)\right. \\
&\left. -\sigma\sum_{i;t_i<t}\left(\sqrt{X_{t_{i+1}}} - \sqrt{X_{t_i}}\right)\left(W_{t_{i+1}} - W_{t_i}\right)\right\} \\
=& x_0 + \lim_{n\to\infty}\left\{\sum_{i;t_i<t}\left(k\theta - \frac{\sigma^2}{2} - kX_{t_{i+1}}\right)\frac{T}{n} + \sigma\sum_{i;t_i<t}\sqrt{X_{t_{i+1}}}\left(W_{t_{i+1}} - W_{t_i}\right)\right\}
\end{aligned}$$

Following this derivation, it is natural to consider the following implicit scheme :

$$\hat{X}_{t_{i+1}} = \hat{X}_{t_i} + \left(k - \frac{\sigma^2}{2} - k\hat{X}_{t_{i+1}}\right)\frac{T}{n} + \sigma\sqrt{\hat{X}_{t_{i+1}}\left(W_{t_{i+1}} - W_{t_i}\right)}$$

This scheme is well defined under the hypothesis $2k\theta > \sigma^2$ at least when the time step is small enough. Indeed, when $\hat{X}_{t_i} \geq 0$ and $\frac{T}{n} \leq 1/k^-$ (with $k^- = \max(-k, 0)$), we can choose $\sqrt{\hat{X}_{t_{i+1}}}$ as the unique positive root of the second-degree polynomial

$$P(x) = \left(1 + k\frac{T}{n}\right)x^2 - \sigma\left(W_{t_{i+1}} - W_{t_i}\right)x - \left(\hat{X}^n_{t_i} + \left(k\theta - \frac{\sigma^2}{2}\right)\frac{T}{n}\right)$$

since $2k\theta > \sigma^2$and $P(0) < 0$. Thus we get

$$\hat{X}_{t_{i+1}} = \left(\frac{\sigma\left(W_{t_{i+1}} - W_{t_i}\right) + \sqrt{\sigma^2\left(W_{t_{i+1}} - W_{t_i}\right)^2 + 4\left(\hat{X}_{t_i} + \left(k\theta - \frac{\sigma^2}{2}\right)\frac{T}{n}\right)\left(1 + k\frac{T}{n}\right)}}{2\left(1 + k\frac{T}{n}\right)}\right)^2 \quad (14)$$

This scheme is well defined and we can easily check that the monotonicity property of the CIR process is still satisfied, i.e if $x_0 < x'_0$ are two initial conditions, then the scheme satisfies $\hat{X}_{t_i} < \hat{X}'_{t_i}$.

We can also derive another scheme if we consider the SDE of the square-root of (X_t) :

$$d\sqrt{X_t} = \frac{k\theta - \sigma^2/4}{2\sqrt{X_t}}dt - \frac{k}{2}\sqrt{X_t}dt + \frac{\sigma}{2}dW_t$$

By impliciting the drift and following a similar derivation to the previous example, we get again a second-degree equation in $\sqrt{\hat{X}_{t_{i+1}}}$:

$$\left(1 + \frac{kT}{2n}\right)\hat{X}_{t_{i+1}} - \left[\frac{\sigma}{2}\left(W_{t_{i+1}} - W_{t_i}\right) + \sqrt{\hat{X}_{t_i}}\right]\sqrt{\hat{X}_{t_{i+1}}} - \frac{k\theta - \sigma^2/4}{2}\frac{T}{n} = 0$$

This equation also has only one positive root when $\sigma^2 < 4k\theta$ and $\frac{T}{n} < 2/k^-$. Finally, we get:

$$\hat{X}_{t_{i+1}} = \left(\frac{\frac{\sigma}{2}\left(W_{t_{i+1}} - W_{t_i}\right) + \sqrt{\hat{X}_{t_i}} + \sqrt{\left(\frac{\sigma}{2}\left(W_{t_{i+1}} - W_{t_i}\right) + \sqrt{\hat{X}_{t_i}}\right)^2 + 4\left(1 + \frac{kT}{2n}\right)\frac{k\theta - \sigma^2/4}{2}\frac{T}{n}}}{2\left(1 + \frac{kT}{2n}\right)}\right)^2 \quad (15)$$

Again, we can check that the motonicity property of the CIR process is still satisfied.

If we consider the SDEs that derives X^α for $\alpha \in \mathbb{R}$, it is clear that the only two values of α that give a second-degree equation are 1 and 1/2. Hence,we cannot get other schemes looking at the SDE satisfied by X^α since other powers different than 1 and 1/2 don't lead to analytical formulas and require a numerical resolution.

We will run the CVA simulation using the previous schemes : Scheme (14) $\sigma^2 < 2k\theta$, Scheme (15) $\sigma^2 < 4k\theta$, Diop and Deelstra Delbaen. The convergence of these schemes are proven in [1]. The table below displays the average standard error of on 1000 simulations when using different schemes.

Deelstra Delbaen	Diop	Scheme (14) $\sigma^2 < 2k\theta$	Scheme (15) $\sigma^2 < 4k\theta$
0.05	0.07	0.05	0.06

Table 2: std error of the simulation using different schemes

In our work, we choose to work with the Deelstra Delbaen scheme, i.e we will simulate simultanuously sample path of the stock price process S and the stochastic intensity process y as follows :

$$S_{t_{i+1}} = S_{t_i} + rS_{t_i}h + \sigma S_{t_i}\sqrt{h}Z_{t_{i+1}}$$

$$y_{t_{i+1}} = y_{t_i} + \kappa\left(\mu - y_{t_i}\right)h + \nu\sqrt{y_{t_i}1_{y_{t_i}>0}h}(\rho Z_{t_{i+1}} + \sqrt{1-\rho^2}Z^{\perp}_{t_{i+1}})$$

$$Y_{t_{i+1}} = Y_{t_i} + y_{t_i}h$$

for $i = 0,\dots,n-1$, where we keep the same notation as in the previous section. Finally, we add the shift and compute the average of the CVA realizations.

4.3 Variance reduction

The efficiency of Monte Carlo methods may be improved if we can reduce the variance of random sequence. Two common variance reduction methods are antithetic variate method and control variate method :

Antithetic variate method
The idea of this method is very simple and can be explained as follows: assume $Z \sim N(0,1)$ and we want to compute $E[f(Z)]$. Since $-Z \sim N(0,1)$, this is equivalent to computing

$$E[f(Z)] = \frac{E[f(Z)] + E[f(-Z)]}{2} = E\left[\frac{f(Z)+f(-Z)}{2}\right] =: E[X]$$

We hope to have $\mathrm{Var}(X) < \mathrm{Var}(f(Z))$. Under some conditions, this will be true. The antithetic variate method is supported by the following general result: Let f be a monotone function and X be any rv, then we have

$$\mathrm{Cov}(f(X), f(1-X)) \le 0 \text{ and } \mathrm{Cov}(f(X), f(-X)) \le 0$$

The standard normal variables are ideal in reducing variance due to their symmetric properties.
We apply the anthetic variate methode to the Monte Carlo simulation to compute the CVA. The table bellow displays the simulation results.

σ	ρ	κ	ν	CVA	std
0.1	-0.5	0.4	0.02	0.11	0.01
0.1	-0.5	0.4	0.08	0.14	0.01
0.1	-0.5	0.8	0.02	0.11	0.01
0.1	-0.5	0.8	0.08	0.13	0.01
0.1	0.3	0.4	0.02	0.13	0.01
0.1	0.3	0.4	0.08	0.17	0.01
0.1	0.3	0.8	0.02	0.13	0.01
0.1	0.3	0.8	0.08	0.16	0.01
0.1	0.9	0.4	0.02	0.15	0.01
0.1	0.9	0.4	0.08	0.18	0.01
0.1	0.9	0.8	0.02	0.17	0.01
0.1	0.9	0.8	0.08	0.16	0.01
0.5	-0.5	0.4	0.02	0.65	0.07
0.5	-0.5	0.4	0.08	0.61	0.07
0.5	-0.5	0.8	0.02	0.56	0.07
0.5	-0.5	0.8	0.08	0.51	0.07
0.5	0.3	0.4	0.02	0.61	0.07
0.5	0.3	0.4	0.08	0.69	0.07
0.5	0.3	0.8	0.02	0.61	0.07
0.5	0.3	0.8	0.08	0.69	0.07
0.5	0.9	0.4	0.02	0.73	0.07
0.5	0.9	0.4	0.08	0.82	0.07
0.5	0.9	0.8	0.02	0.75	0.07
0.5	0.9	0.8	0.08	0.81	0.07
0.8	-0.5	0.4	0.02	1.02	0.15
0.8	-0.5	0.4	0.08	0.97	0.15
0.8	-0.5	0.8	0.02	1.01	0.15
0.8	-0.5	0.8	0.08	1.05	0.15
0.8	0.3	0.4	0.02	1.13	0.15
0.8	0.3	0.4	0.08	1.21	0.15
0.8	0.3	0.8	0.02	1.04	0.15
0.8	0.3	0.8	0.08	1.21	0.15
0.8	0.9	0.4	0.02	1.13	0.15
0.8	0.9	0.4	0.08	1.29	0.15
0.8	0.9	0.8	0.02	1.10	0.15
0.8	0.9	0.8	0.08	1.32	0.15

Table 3: The results of MC simulation of the CVA with Antithetic variate method

Control Variate

We recall that the Control Variate method consists on selecting an alternative payoff Π^{an}, which we know how to evaluate analytically, i.e: $E[\Pi^{an}] = \pi^{an}$ is known. Suppose we want to compute the expectation of a payoff Π. When we simulate our original payoff Π, we now simulate as well the analytical payoff π^{an} as a function of the same scenarios for the underlying variables [3]. We define a new control-variate estimator for $E[\Pi]$ as :

$$\widehat{\Pi}_c\left(\gamma; n_p\right) := \frac{\sum_{j=1}^{n_p} \Pi^j}{n_p} + \gamma\left(\frac{\sum_{j=1}^{n_p} \Pi^{\mathrm{an},j}}{n_p} - \pi^{\mathrm{an}}\right)$$

with n_p the number of scenarios, γ a constant to be determined and Π^j and $\Pi^{\mathrm{an},j}$ iid copies of Π and Π^{an} respectively. It is easy to see that estimator above is unbiased. Lets consider now the random variable :

$$\Pi_c(\gamma) := \Pi + \gamma\left(\Pi^{\mathrm{an}} - \pi^{\mathrm{an}}\right)$$

whose expectation is the $E(\Pi)$ we are estimating, we have :

$$\mathrm{Var}\left(\Pi_c(\gamma)\right) = \mathrm{Var}(\Pi) + \gamma^2\,\mathrm{Var}\left(\Pi^{\mathrm{an}}\right) + 2\gamma\,\mathrm{Corr}\left(\Pi, \Pi^{\mathrm{an}}\right)\mathrm{Std}(\Pi)\,\mathrm{Std}\left(\Pi^{\mathrm{an}}\right)$$

We may minimize this function of γ by differentiating and setting the first derivative to zero. We can easily show that the variance is minimized by the following value of

$\gamma : \gamma^* := -\operatorname{Corr}(\Pi, \Pi^{\text{an}}) \operatorname{Std}(\Pi)/\operatorname{Std}(\Pi^{\text{an}})$. We plug $\gamma = \gamma^*$ into the above expression and we get:

$$\operatorname{Var}(\Pi_c(\gamma^*)) = \operatorname{Var}(\Pi)\left(1 - \operatorname{Corr}(\Pi, \Pi^{\text{an}})^2\right)$$

from which we see that $\Pi_c(\gamma^*)$ has a smaller variance than the original Π. This variance decreases when the correlation between Π and Π^{an} increases (in absolute value). Accordingly, when moving to simulated quantities, we set :

$$\widehat{\operatorname{Std}}\left(\Pi_c(\gamma^*); n_p\right) = \widehat{\operatorname{Std}}\left(\Pi; n_p\right)\left(1 - \widehat{\operatorname{Corr}}\left(\Pi, \Pi^{\text{an}}; n_p\right)^2\right)^{1/2}$$

where $\operatorname{Cor}\left(\Pi, \Pi^{\text{an}}; n_p\right)$ is the sample correlation

$$\widehat{\operatorname{Cor}}\left(\Pi, \Pi^{\text{an}}; n_p\right) = \frac{\widehat{\operatorname{Cov}}\left(\Pi, \Pi^{\text{an}}; n_p\right)}{\widehat{\operatorname{Std}}\left(\Pi; n_p\right)\widehat{\operatorname{Std}}\left(\Pi^{\text{an}}; n_p\right)}$$

Hence, we choose :

$$\gamma^* := -\frac{\widehat{\operatorname{Cor}}(\Pi, \Pi^{\text{an}})}{\widehat{\operatorname{Var}}(\Pi^{\text{an}})} \tag{16}$$

For our CVA simulation, we consider the following control variable :

$$\Pi^{an} = 1_{\{\xi < \Lambda(T)\}}(S_T - K)^+ \tag{17}$$

such that the stock price process S and the intensity process λ are not correlated. In this case, using the fact that $r = 0$, the expectation π^{an} is know analytically and equal to :

$$\pi^{an} = \mathbb{Q}(\tau < T)C(S_0, K, r, \sigma, T) \tag{18}$$

with $\mathbb{Q}(\tau < T)$ the market risk-neutral default probability.
Finally, taking :

$$\Pi = \exp(-\Lambda(T))(S_T - K)^+ \tag{19}$$

$$\Pi_c = \Pi - \frac{\widehat{\operatorname{Cor}}(\Pi, \Pi^{\text{an}})}{\widehat{\operatorname{Var}}(\Pi^{\text{an}})}(\Pi^{an} - \pi^{an}) \tag{20}$$

we get the following CVA estimator :

$$\widehat{CVA} = C(S_0, K, r, \sigma, T) - E[\Pi_c] \tag{21}$$

We apply the control variate method to the Monte Carlo simulation to compute the CVA. The table bellow displays the simulation results.

σ	ρ	κ	ν	CVA	std error
0.1	-0.5	0.4	0.02	0.124235	0.018766
0.1	-0.5	0.4	0.08	0.113267	0.018825
0.1	-0.5	0.8	0.02	0.124295	0.018765
0.1	-0.5	0.8	0.08	0.113682	0.018823
0.1	0.3	0.4	0.02	0.131032	0.018726
0.1	0.3	0.4	0.08	0.138989	0.018686
0.1	0.3	0.8	0.02	0.130935	0.018726
0.1	0.3	0.8	0.08	0.138579	0.018687
0.1	0.9	0.4	0.02	0.136054	0.018698
0.1	0.9	0.4	0.08	0.161356	0.018547
0.1	0.9	0.8	0.02	0.135981	0.018698
0.1	0.9	0.8	0.08	0.159978	0.018556
0.5	-0.5	0.4	0.02	0.587033	0.049702
0.5	-0.5	0.4	0.08	0.523515	0.040144
0.5	-0.5	0.8	0.02	0.587438	0.049698
0.5	-0.5	0.8	0.08	0.525335	0.040137
0.5	0.3	0.4	0.02	0.629641	0.049381
0.5	0.3	0.4	0.08	0.676339	0.041910
0.5	0.3	0.8	0.02	0.629134	0.041938
0.5	0.3	0.8	0.08	0.674131	0.041911
0.5	0.9	0.4	0.02	0.659058	0.041917
0.5	0.9	0.4	0.08	0.818958	0.041797
0.5	0.9	0.8	0.02	0.658814	0.041917
0.5	0.9	0.8	0.08	0.809908	0.041805
0.8	-0.5	0.4	0.02	0.877284	0.073479
0.8	-0.5	0.4	0.08	0.765224	0.083577
0.8	-0.5	0.8	0.02	0.878246	0.083478
0.8	-0.5	0.8	0.08	0.767839	0.083576
0.8	0.3	0.4	0.02	0.957674	0.084046
0.8	0.3	0.4	0.08	1.037982	0.083481
0.8	0.3	0.8	0.02	0.956815	0.084054
0.8	0.3	0.8	0.08	1.034519	0.083500
0.8	0.9	0.4	0.02	1.009239	0.083590
0.8	0.9	0.4	0.08	1.308031	0.080834
0.8	0.9	0.8	0.02	1.008968	0.083606
0.8	0.9	0.8	0.08	1.290402	0.081030

Table 4: The results of MC simulation of the CVA with Control Variate method

As Control Variate method gave better results, we will use this technique to simulate CVA values from now onwards.

5 Data Visualisation

We fix the parameters S_0, K, μ, r, Rec, T and Γ^{Mkt}, and we simulate 200.000 CVAs. We will give the fixed parameters the same values as in the section Numerical evaluation, i.e :

Parameter	Value
S_0	100
K	100
y_0	0.01
μ	0.03
r	0
Rec	0
T	1 year
Γ^{Mkt}	0.03

Table 5: Parameter values

The table below summarises the results of the simulation.

Parameter	Count	Min	25%	50%	75%	Max
σ	10	0.1	0.3	0.55	0.8	1
ρ	20	-1	-0.525	0	0.525	1
κ	20	0.1	0.35	0.5	0.7	1
ν	50	0.01	0.032	0.055	0.078	0.1
CVA	200.000	0.11204	0.39824	0.69641	0.92544	1.46692

Table 6: CVA simulation description

Next we plot the CVA as function of the different parameters : σ , ρ, κ and ν.

5.1 Impact of the volatility σ

We plot the CVA as a function of σ for different values of ρ, κ and ν. We notice that the CVA is always increasing with σ, which is what we expected. Indeed, when σ increases the call option price increases and therefore the credit value adjustment as well. We also notice that when the correlation ρ is bigger (in absolute value) the volatility ν has greater impact on the CVA.

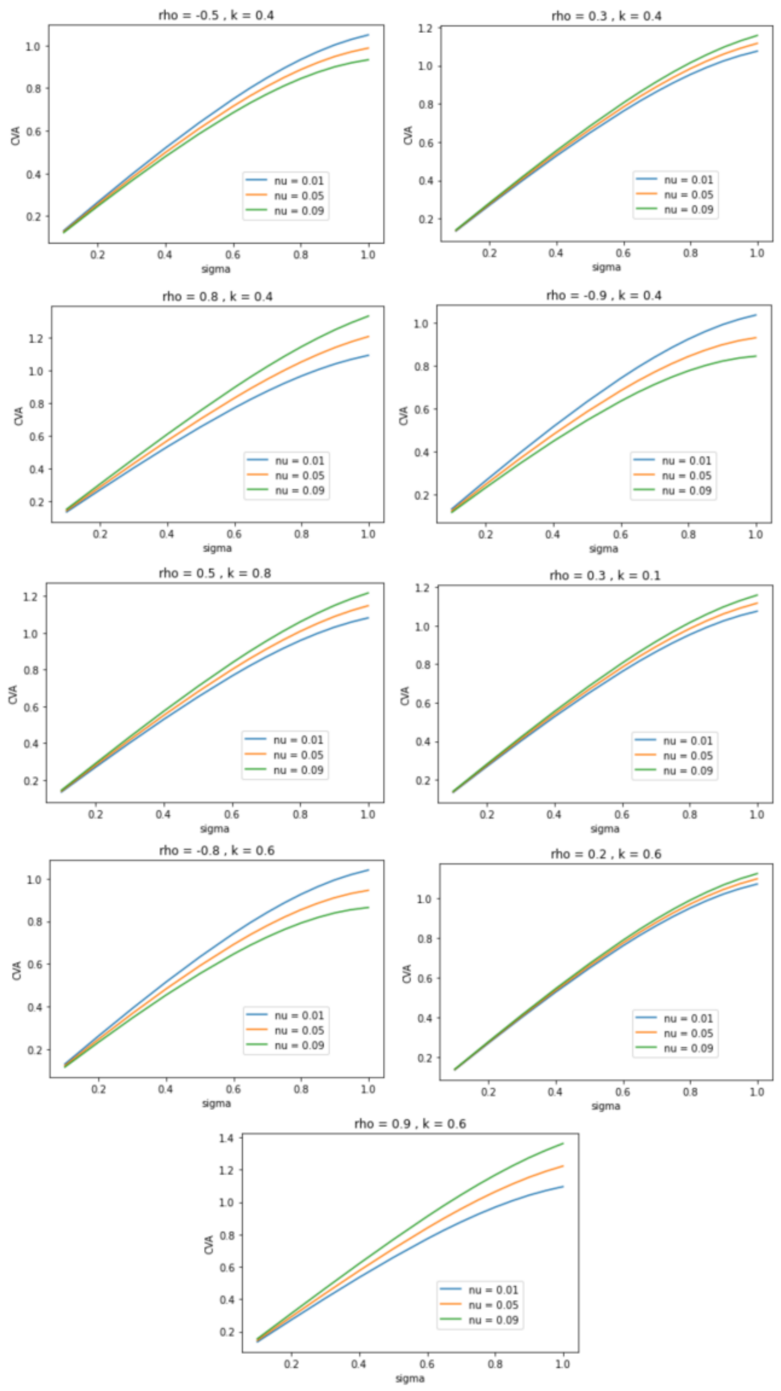


Figure 4: CVA(sigma)

5.2 Impact of the correlation ρ

The figure bellow shows the plots of the CVA as a function of ρ, the correlation between the default intensity and the stock price S. The plots show that the CVA is linearly increasing with ρ. We notice that as the default dependence grows, the CVA increases. This can be explained by the fact that, when the intensity goes up, the probability of the default increases, and when the correlation is larger, the equity goes up as well and the option becomes more in money. Hence, it has more value and the CVA is larger.

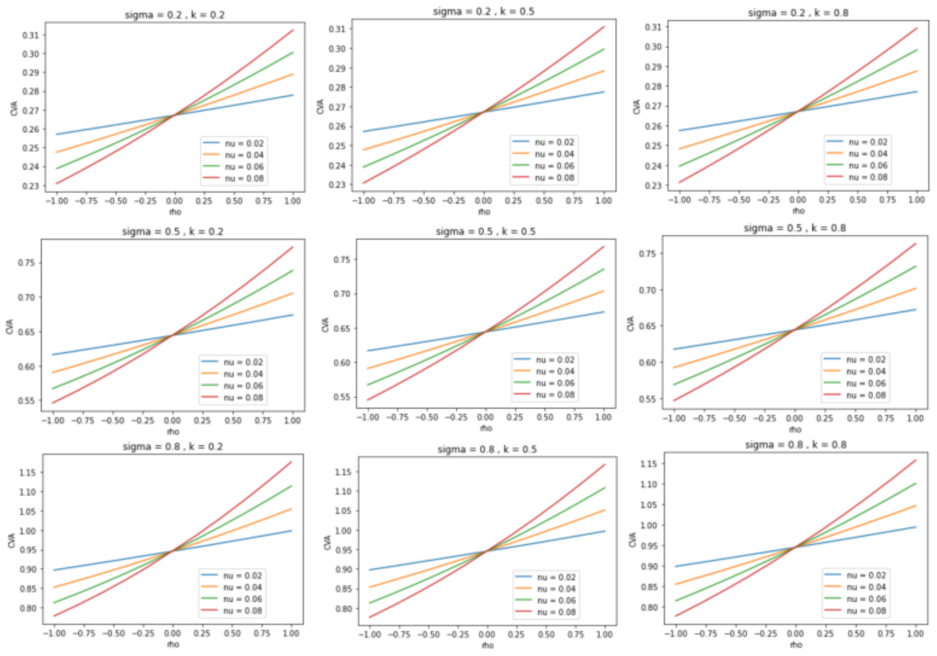


Figure 5: CVA(rho)

5.3 Impact of the speed of mean reversion κ

We look now at the impact of κ on the CVA results. We notice that CVA values are more less constant when we change κ and fix the other parameters. This is because for small maturity (1 year), the speed of mean reversion κ has a low impact on the CVA. Howerver, the figures show a clear pattern : a decreasing tendency of the CVA as we increase κ for positive ρ and an increasing tendency of the CVA as we increase κ for negative ρ.

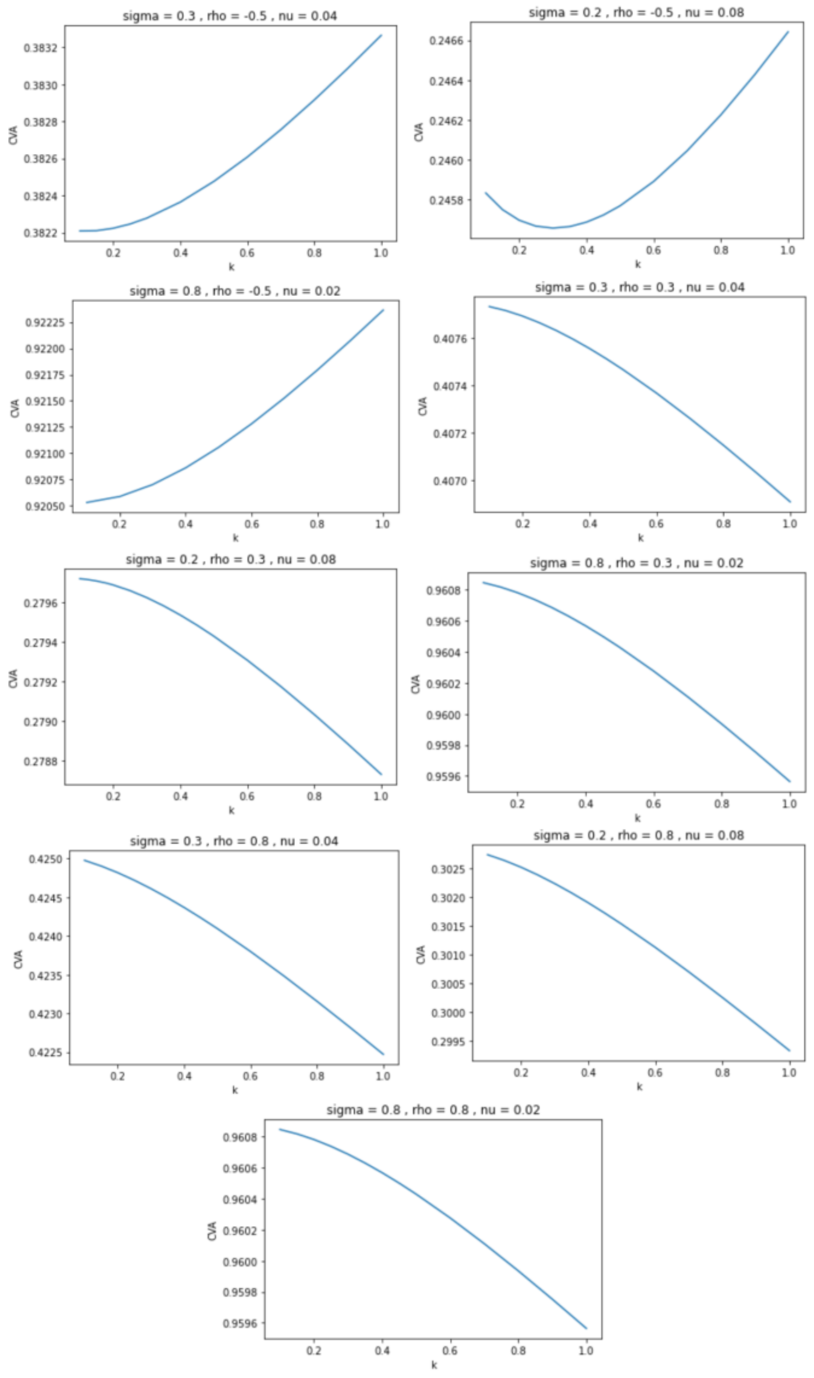


Figure 6: CVA(κ)

5.4 Impact of the volatility ν

The plots of the CVA as a function of ν show that the CVA increases slightly with ν when ρ is positive and inversely when ρ is negative. We notice that κ and ν have opposite effects on the CVA. Indeed, in the CIR process, κ and ν fight each other. κ is the speed of mean reversion, so if we increase κ, the paths will converge to the long time mean faster and that means there is less uncertainty in the system because there is faster convergence to the long term mean. Hence, it is like lowering the volatility in a sense. On the other hand, if we increase ν, we increase the randomness. Therefore, the two parameters have obviously opposite effects.

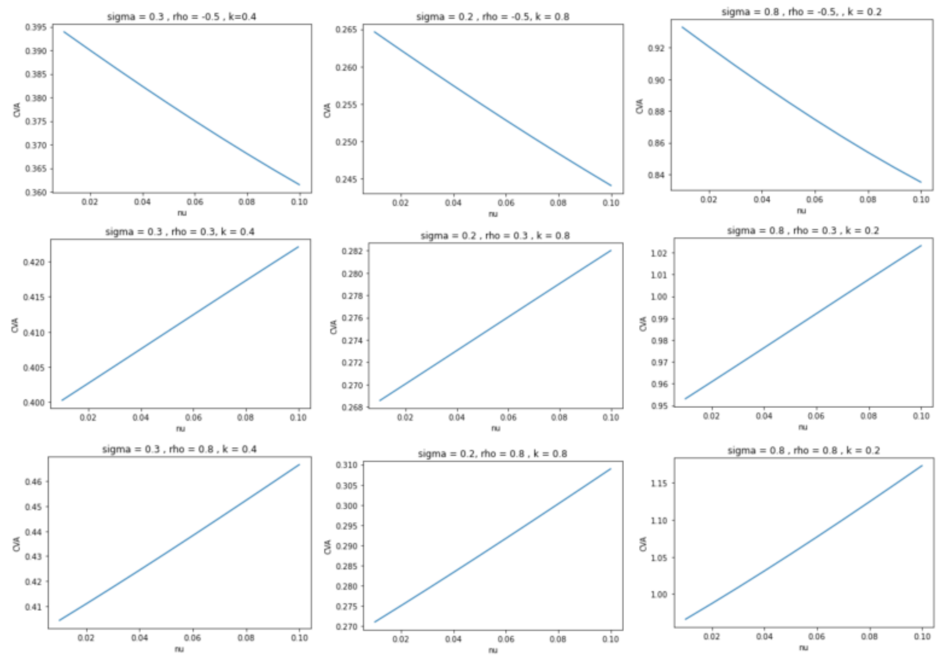


Figure 7: CVA(ν)

5.5 Linear Regression

Although the CVA graphs seem linear in certain cases, a linear regression fails to map well the CVA, especially for large volatilities and correlations. The figures bellow illustrate this issue.

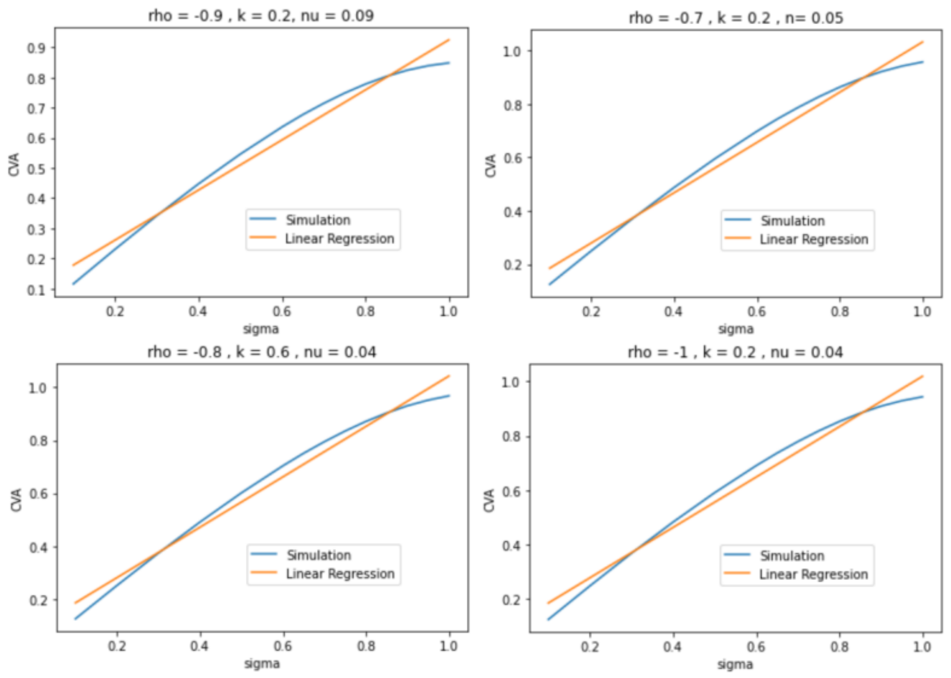


Figure 8: Linear regression

To capture the non-linearity of the CVA, we use a Neural Network approach. We will discuss the different Deep learning techniques in the following sections.

6 Artificial Neural Network

In this chapter, we will present an introduction to deep learning and develop an understanding of the properties of Neural networks and how they are trained. This chapter is inspired of Deep Learning Lecture notes by Mikko Pakkanen (2020)[27], ”Artificial Neural Networks for SABR model calibration hedging” by Hugues Thorin [33] and ”Deep Learning: An Introduction for Applied Mathematicians” (2018) [17]. Illustrations in this chapter without mentioned sources are extracted from Deep Learning Lectures Notes.

6.1 Introduction

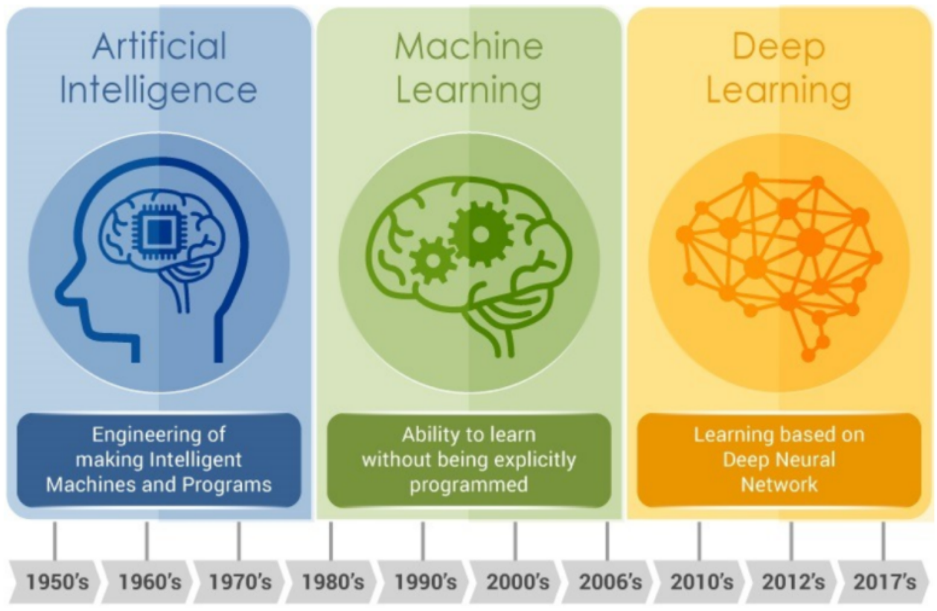

Artificial Intelligence
Machine Learning
Deep Learning
Engineering of making Intelligent Machines and Programs
Ability to learn without being explicitly programmed
Learning based on Deep Neural Network
1950's
1960's
1970's
1980's
1990's
2000's
2006's
2010's
2012's
2017's

Figure 9: History of Deep Learning

In the last decades, deep learning was a topic of great interest to many researchers. Today, deep learning has become extremely popular in a vast range of application fields, from image recognition, speech recognition and natural language processing to targeted advertising and drug discovery. Deep learning has achieved many outstanding breakthroughs, perhaps the most notable of which has been the development of *AlphaGo* [31], a computer program developed by Google DeepMind in London in 2015. This program has been able to beat the best human players in the classical Chinese board game Go, a game that was thought to be far too difficult to learn and master by Artificial Intelligence. [2]

[2]Image source : `https://www.slideshare.net/linagora/deep-learning-in-practice-speech-recognition-a`

The financial industry has become increasingly interested in the new Deep Learning methodology. In the past, different techniques of Machine Learning has been applied in retail banking to detect fraud, machine read cheques and perform credit scoring. Today, the industry focuses more in the automation of different tasks using Artificial Intelligence, ranging from customer service in retail banking to trading and portfolio construction in investment banking and asset management,respectively.

In very broad terms, the problem deep learning aims to solve is finding (creating) a function, most often non-linear,

$$\boldsymbol{f} : \mathbb{R}^I \to \mathbb{R}^O$$

that turns $I \in \mathbb{N}$ inputs

$$x_1, \ldots, x_I$$

into $O \in \mathbb{N}$ outputs

$$\boldsymbol{f}_1(x_1, \ldots, x_I), \ldots, \boldsymbol{f}_O(x_1, \ldots, x_I)$$

in an optimal way.

6.2 History of Deep Learning

The history of Deep Learning can be traced back to 1943, when Walter Pitts and Warren McCulloch [24] created a computer model based on the neural networks of the human brain. Both researchers were working in neuroscience and they aimed to develop a mathematical model of nervous activity, founded in logic. The artificial neuron model they developed can be expressed in terms of real functions as

$$f(\boldsymbol{x}) := \begin{cases} 1, & \sum_{i=1}^{I} x_i \geq a \\ 0, & \sum_{i=1}^{I} x_i < a \end{cases}$$

where $\boldsymbol{x} = (x_1, \ldots, x_I) \in \{0, 1\}^I$ is a vector of binary inputs and $a \in \mathbb{R}$ is an activation threshold parameter. This model is able to represent some basic logical operations such as AND and OR. Although this McCulloch Pitts Neuron has very limited capability and has no learning mechanism, it will lay the foundation for Artificial Neural Network Deep Learning.

In 1957, Frank Rosenblatt [28] refined the model of McCulloch and Pitts by introducing the concept of perceptrons in his paper: "The Perceptron: A Perceiving and Recognizing Automaton". This augmented the artifical euron model by implementing weighting of inputs, which no longer had to be binary, as follows :

$$f(\boldsymbol{x}; \boldsymbol{w}, b) := H(\boldsymbol{w}'\boldsymbol{x} + b)$$

where $\boldsymbol{x} \in \mathbb{R}^I$ is the input vector, $\boldsymbol{w} \in \mathbb{R}^I$ is a vector of weights and $b \in \mathbb{R}$ is a bias term, mapped through the Heaviside function

$$H(x) := \begin{cases} 0, & x < 0 \\ 1, & x \geq 0 \end{cases}$$

'Perceptron' had true learning capabilities to do binary classification and Frank Rosenblatt's work inspired the revolution in research of shallow neural network for years to come. However, in 1969, Marvin Minsky and Seymour Papert published the book "Perceptrons" [25] in which they show that Rosenblatt's perceptron cannot solve complicated functions like XOR. For such function perceptrons should be placed in multiple hidden layers which compromises perceptron learning algorithm. This setback triggers the so-called AI winter, a period of reduced funding of AI research in the US and UK. The AI winter was also due to initial unrealistic expectations of what AI could achieve and subsequent disappointments.

Nevertheless, research into neural networks did not die out completely during the AI winter. In fact, in the 1970s key ideas behind backpropagation were discovered by Seppo Linnainmaa . The researcher published general method for automatic differentiation for backpropagation and also implemented backpropagation in computer code. Althought research in backpropagation had come very far in the 1970s, yet it would not be implemented in neural network till next decade. In 1986, Geoffrey Hinton, Rumelhart, and Williams in their paper "Learning Representations by backpropagating errors" [29] show the successful implementation of backpropagation in the neural network. It opened gates for training complex deep neural network easily which was the main obstacle in earlier days of research in this area. More precisely, The conclusion of their paper is that backpropagation helps learn useful internal representations of data — i.e., transformations of the data in the hidden layers of the neural network, which facilitate the task the network is aiming to accomplish(classification for example). The idea of internal representations is central to many developments in Deep Learning, this why the paper is often considered as the foundation of current Deep Learning research.

Despite the several discoveries and important advances in Deep Learning, neural networks were still not a popular research topic in the 1990s and it was until the late 2000s and early 2010s that Deep Learning entered a new era, after breakthroughs in speech and image recognition (e.g., Hinton at al. (2012) and Krizhevsky at al. (2012) [20], respectively) using deep neural networks.

6.3 Properties of neural networks

6.3.1 General construction

Definition 1
Let $I, O, r \in \mathbb{N}$. A function $\boldsymbol{f} : \mathbb{R}^I \to \mathbb{R}^O$ is a feedforward neural network (FNN) with $r-1 \in \{0,1,\ldots\}$ hidden layers, where there are $d_i \in \mathbb{N}$ units in the i -th hidden layer for any $i = 1,\ldots,r-1$, and activation functions $\boldsymbol{\sigma}_i : \mathbb{R}^{d_i} \to \mathbb{R}^{d_i}, i = 1,\ldots,r$, where $d_r := O$, if

$$\boldsymbol{f} = \boldsymbol{\sigma}_r \circ \boldsymbol{L}_r \circ \cdots \circ \boldsymbol{\sigma}_1 \circ \boldsymbol{L}_1, \tag{22}$$

where $\boldsymbol{L}_i : \mathbb{R}^{d_{i-1}} \to \mathbb{R}^{d_i}$, for any $i = 1,\ldots,r$, is an affine function

$$\boldsymbol{L}_i(\boldsymbol{x}) := W^i \boldsymbol{x} + \boldsymbol{b}^i, \quad \boldsymbol{x} \in \mathbb{R}^{d_{i-1}} \tag{23}$$

parameterised by weight matrix $W^i = \left[W^i_{j,k}\right]_{j=1,\dots,d_i,k=1,\dots,d_{i-1}} \in \mathbb{R}^{d_i \times d_{i-1}}$ *and bias vector* $\boldsymbol{b}^i = \left(b^i_1,\dots,b^i_{d_i}\right) \in \mathbb{R}^{d_i}$*, with* $d_0 := I$*. We shall denote the class of such functions* f *by*

$$\mathcal{N}_r(I, d_1, \dots, d_{r-1}, O; \boldsymbol{\sigma}_1, \dots, \boldsymbol{\sigma}_r) \tag{24}$$

If $\boldsymbol{\sigma}_i(\boldsymbol{x}) = \left(g(x_1), \dots, g\left(x_{d_i}\right)\right), \boldsymbol{x} = \left(x_1, \dots, x_{d_i}\right) \in \mathbb{R}^{d_i}$*, for some* $g : \mathbb{R} \to \mathbb{R}$*, we write* g *in place of* σ_i*.*

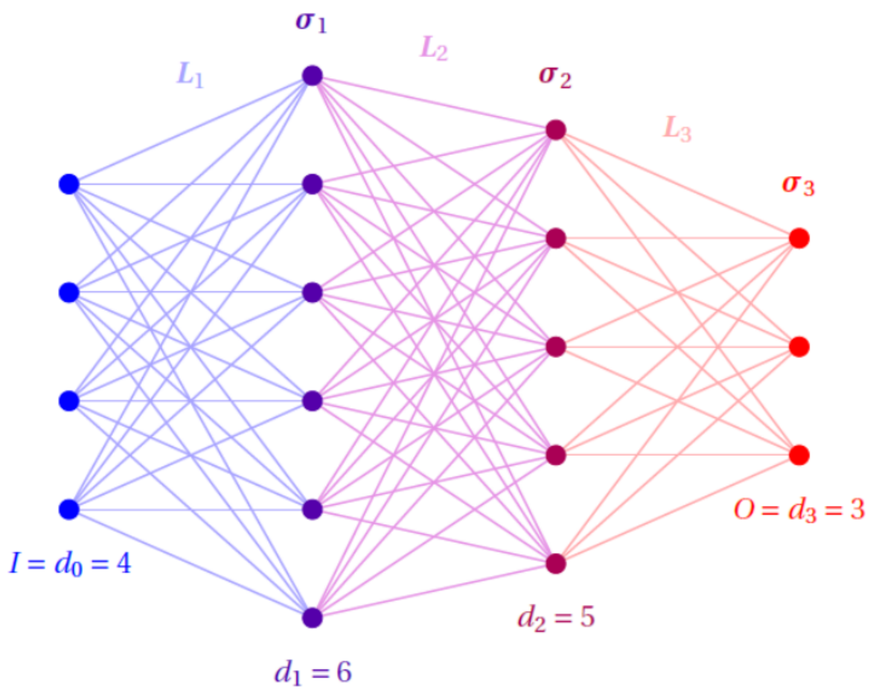


Figure 10: Graphical representation of a neural network with r = 3, $I = d_0 = 4$, $d_1 = 6$, $d_2 = 5$ and $O = d_3 = 3$.

The architecture of a neural network is fully determined by :

- r and $d_1, \dots, d_{r-1}$ which are called the hyper-parameters of the network.
- the weights $W^1, \dots, W^r$ and biases $b^1, \dots, b^r$ which are the parameters of the network
- the activation functions $\sigma^1, \dots, \sigma^r$

Building functions by composition is essential in Neural Networks to generate non-linearities. By composing affine functions, it produces high-order polynomials exponentially fast.

Proposition 1
The number of parameters that characterise $f \in \mathcal{N}_r(I, d_1, \dots, d_{r-1}, O; \sigma_1, \dots, \sigma_r)$ is (assuming that $\sigma_1, \dots, \sigma_r$ involve no additional parameters)

$$\sum_{i=1}^{r} (d_{i-1} + 1) d_i \tag{25}$$

6.3.2 Activation function

An activation function σ, as introduced in the previous section, is a function that is added into an artificial neural network, in order to add some kind of non-linear property to the network and help the network learn complex patterns in the learning data. Without the activation functions, the neural network could perform only linear mappings from inputs x to the outputs y. When comparing with a neuron-based model in our brains, the activation function decides what is to be fired to the next neuron. That is exactly what an activation function does in an artificial neural network as well. It takes in the output signal from the previous cell and converts it into some form that can be taken as input to the next cell [18]. The figure below illustrates this comparison.

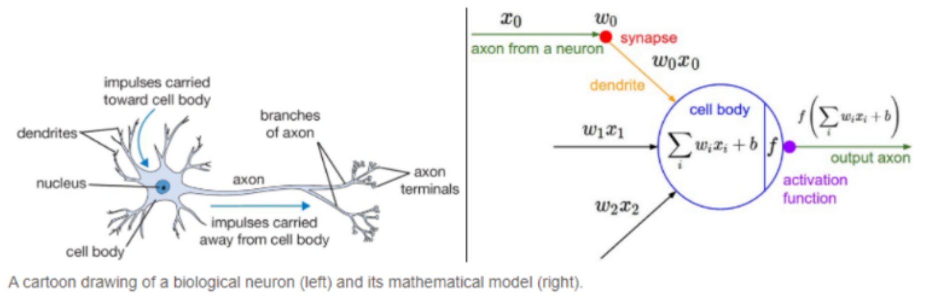


Figure 11: Biological neuron and its mathematical model, Source: cs231n by Stanford

We can distinguish two types of activation functions used in Neural Networks : one-dimensional activation functions, applied component wise and multi-dimensional activation functions.

We join in the appendix examples of one-dimensional activation functions and multi-dimensional activation functions and give their key properties.

6.3.3 Universal Approximation Property

The use of Neural Networks to model complex functional relationships can be justified by the universal approximation theorem, that stipulates that any "reasonable" function can be approximated by a suitable Neural Network.
To measure the precision of approximation by a neural network, we define two norms for functions. Let $K \subset \mathbb{R}^I$ be compact[3]. For any measurable $f : \mathbb{R}^I \to \mathbb{R}$, we introduce the sup norm

$$\|f\|_{\sup, K} := \sup_{\boldsymbol{x} \in K} |f(\boldsymbol{x})|$$

and, for any $p \geq 1$, the L^p norm

$$\|f\|_{L^p(K)} := \left(\int_K |f(\boldsymbol{x})|^p \, \mathrm{d}\boldsymbol{x} \right)^{\frac{1}{p}}$$

Moreover, we denote by $L^p(K, \mathbb{R})$ the class of measurable functions $f : K \to \mathbb{R}$ such that $\|f\|_{L^p(K)} < \infty$. The following is a slight reformulation of Leshno et al.[21] (1993, Theorem 1 and Proposition 1).

Theorem 1
(Universal approximation property)
Let $g : \mathbb{R} \to \mathbb{R}$ be a measurable function such that :

- *(a) g is not a polynomial function,*
- *(b) g is bounded on any finite interval,*
- *(c) the closure of the set of all discontinuity points of g in $\mathbb{R}$ has zero Lebesgue measure.*

Moreover, let $K \subset \mathbb{R}^I$ be compact and $\varepsilon > 0$. We have :

- *(i) For any $u \in C(K, \mathbb{R})$, there exist $d \in \mathbb{N}$ and $f \in \mathcal{N}_2(I, d, 1; g, Id)$ such that*

$$\|u - f\|_{sup, K} < \varepsilon$$

- *(ii) Let $p \geq 1$. For any $v \in L^p(K, \mathbb{R})$, there exist $d' \in \mathbb{N}$ and $h \in \mathcal{N}_2(I, d', 1; g, Id)$ such that*

$$\|v - h\|_{L^p(K)} < \varepsilon$$

While the previous theorom holds for networks with a single hidden layer, we usually extend the universal approximation property to deeper networks in practice.

However, the theorem doesn't tell us how to construct f and h and how to choose the architecture of the network and its parameters and hyper-parameters. The theorem only guarantees the existence of a Neural Network.

[3] $K \subset \mathbb{R}^I$ is compact if it is both closed and bounded.

6.4 Training Neural Networks

6.4.1 Loss function

A Deep Learning neural network learns to map a set of inputs to a set of outputs from training data. In a Deep learning problem, we may seek to maximize or minimize the objective function, i.e we look for a candidate solution that has the highest or lowest score respectively.

Typically, with artificial neural networks, we seek to minimize the error. The objective function is often referred to as a cost function or a loss function and the value calculated by the loss function is referred to as simply “loss”.

Definition 2
A loss function is

$$\ell : \mathbb{R}^O \times \mathbb{R}^O \to \mathbb{R} \tag{26}$$

If x and y are the realisation of a joint random vector (X, Y), with X corresponding to the random input vector and Y the random output vector, the goal is to find the optimal function f that minimises the loss : $\ell(f(X), Y)$. If the joint distribution of (X, Y) is known, we seek optimal f by minimising:

$$\mathbb{E}[\ell(f(X), Y)] \tag{27}$$

Nevertheless, the joint distribution of (X, Y) is usually unknown in practice, and we need to generate samples $x^1, \dots, x^N$ of the input x and $y^1, \dots, y^N$ of the ouput y, and minimise the *the empirical risk* :

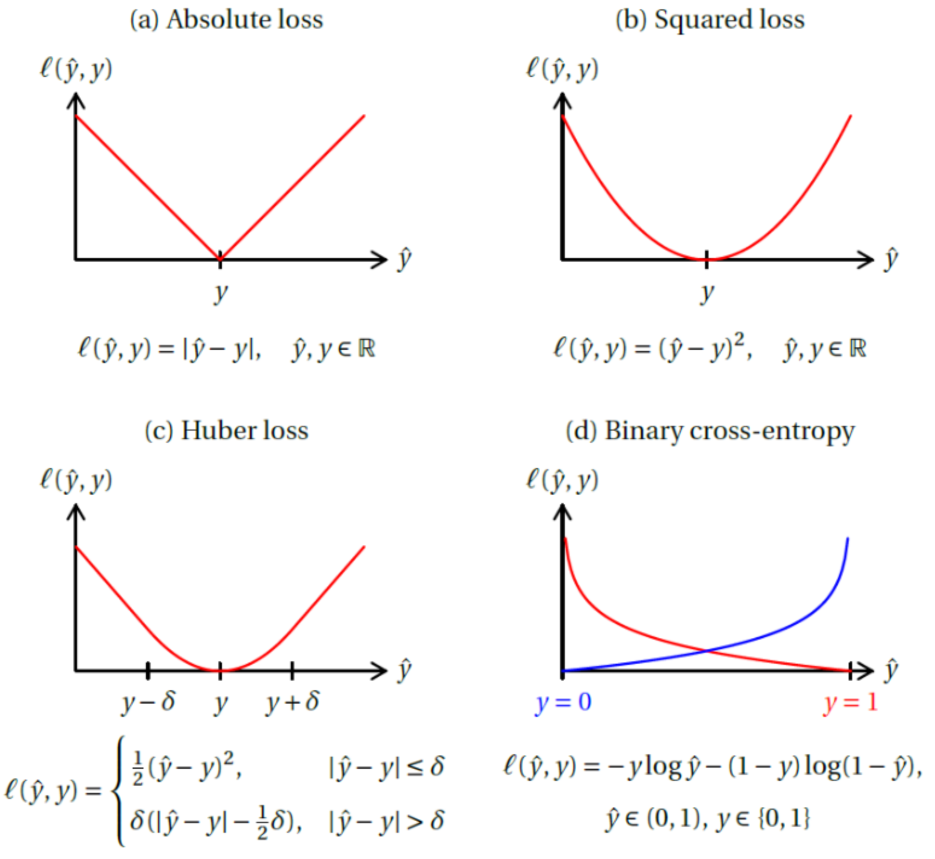
$$\mathcal{L}(f) = \frac{1}{N} \sum_{i=1}^{N} \ell\left(f\left(x^i\right), y^i\right) \tag{28}$$

The minimisation problem is :

$$\min_{\hat{y} \in \mathbb{R}} \mathbf{E}[\ell(\hat{y}, Y)] \tag{29}$$

The choice of the loss function ℓ is important in deep learning and we shall study briefly the different loss functions presented in the figure bellow:

- **Absolute loss**: The unique solution of the minimisation problem is the *median* of Y. Thus training with absolute loss targets the *median*.
- **Squared loss**: The unique solution of the minimisation problem is the *mean* of Y. Thus training with squared loss targets the *mean*. The major shortcoming of the square loss is that, because of the quadratic growth, a prediction $\hat{y}$ far from the true value y is more penalized than the one close to the true value, which amplifies the impact of outliers on the training.
- **Huber loss**: To mitigate the shortcoming of the squared loss, we can use the Hubert loss which keeps the quadratic behaviour of $\hat{y} \to \ell(\hat{y}, y)$ in a δ-neighbourhood (for small $\delta > 0$) of y, and applying a *linear extrapolation* elsewhere.



$$\ell(\hat{y}, y) = |\hat{y} - y|, \quad \hat{y}, y \in \mathbb{R}$$

$$\ell(\hat{y}, y) = (\hat{y} - y)^2, \quad \hat{y}, y \in \mathbb{R}$$

$$\ell(\hat{y}, y) = \begin{cases} \frac{1}{2}(\hat{y} - y)^2, & |\hat{y} - y| \leq \delta \\ \delta(|\hat{y} - y| - \frac{1}{2}\delta), & |\hat{y} - y| > \delta \end{cases}$$

$$\ell(\hat{y}, y) = -y \log \hat{y} - (1-y)\log(1-\hat{y}), \quad \hat{y} \in (0,1),\ y \in \{0,1\}$$

Figure 12: Loss functions examples

- **Binary cross-entropy**: It is the loss function used for binary classification

When the output is multidimensional, we can define the loss function as a weighted sum of one-dimensional losses.

6.5 Minibactch

We introduce the new notion minibach in Deep learning, which is a primordial concept in loss minimisation.

Let $f_\theta \in \mathcal{N}_r(I, d_1, \dots, d_{r-1}, O, \boldsymbol{\sigma}_1, \dots, \boldsymbol{\sigma}_r)$ be the function approximated by a Neural Network with parameters $\theta = \left(W^1, \dots, W^r; \boldsymbol{b}^1, \dots, \boldsymbol{b}^r\right)$. Instead of considering the empirical risk,

$$\mathcal{L}(f) = \frac{1}{N}\sum_{i=1}^{N} \ell\left(f\left(x^i\right), y^i\right)$$

we introduce the minibatch risk :

$$\mathcal{L}_B(\boldsymbol{\theta}) = \frac{1}{\#B}\sum_{i\in B} \ell\left(\boldsymbol{f}_{\boldsymbol{\theta}}\left(\boldsymbol{x}^i\right), \boldsymbol{y}^i\right) \tag{30}$$

where the error is the average over different subsets, minibatches: $B \in \{1, \ldots, N\}$ of samples. The size of a minibatch is called the batch size.

6.6 Epoch

This another important concept in deep learning. In order to train a neural network, the same dataset goes through the network multiple times. We call an epoch the passage of the whole data set through the Neural Network. Multiple epochs are needed to train a network, each epoch is divided into smaller subsets (minibatches). We must choose the number of epochs meticulously. Indeed, a small number of epochs will produce under-fitting, a large number of epochs will produce over-fitting. There is no rules to help us choose the right number of epochs or minibatches, it depends on the dataset. We will dive in the tuning of these parameters in the following chapter.

6.7 Stochastic Gradient Descent (SGD)

Neural Networks are trained by minimising empirical risk using a common method called *stochastic gradient descent (SGD)*.

6.7.1 Gradient Descent

”Gradient descent is a first-order iterative optimization algorithm for finding a local minimum of a differentiable function. The idea is to take repeated steps in the opposite direction of the gradient (or approximate gradient) of the function at the current point, because this is the direction of steepest descent”[4]. The speed it converges to the minima is determined by the learning rate η in the context of deep learning.

Lets consider problem of minimising a generic differentiable objective function $F : \mathbb{R}^d \to \mathbb{R}$. The usual approach would be to solve $\nabla F(\boldsymbol{x}) = 0$ for $\boldsymbol{x} \in \mathbb{R}^d$, where ∇F is the gradient of F. However, ∇F may be hard to compute or $\nabla F(\boldsymbol{x}) = 0$ hard to solve in practice. In this case, we consider the differential equation

$$\frac{\mathrm{d}\boldsymbol{x}(t)}{\mathrm{d}t} = -\nabla F(\boldsymbol{x}(t)), \quad t > 0 \tag{31}$$

[4]Source : `https://en.wikipedia.org/wiki/Gradient_descent`

with initial condition $\boldsymbol{x}(0) \in \mathbb{R}^d$. This equation, if solvable, defines the so-called *gradient flow* $(\boldsymbol{x}(t))_{t\geq 0}$ of F (see Gradient Flows: an overview, Santambrogio, 2016)[30]. Under certain assumptions on F, which guarantee the existence of a unique minimiser and are stronger than mere convexity, it can be shown that $\boldsymbol{x}(t)$ tends to the minimiser as $t \to \infty$ [30]. However, we need to discretise the previous equation by doing the following approximation:

$$\frac{\boldsymbol{x}(t+\eta) - \boldsymbol{x}(t)}{\eta} \approx -\nabla F(\boldsymbol{x}(t))$$

with η the learning rate. The equation can be rewritten as :

$$\boldsymbol{x}(t+\eta) \approx \boldsymbol{x}(t) - \eta \nabla F(\boldsymbol{x}(t))$$

This Euler approximation motivates the gradient descent process, which is characterised by the following iterative algorithm that progressively look for a minimiser with gradient updates :

$$\boldsymbol{x}_{new} := \boldsymbol{x}_{old} - \eta \nabla F(\boldsymbol{x}_{old}),$$

given some initial condition x_0.

6.7.2 Stochastic Gradient Descent

Let f_θ be the function approximated by a Neural Network with parameters $\theta = \left(W^1, \ldots, W^r; \boldsymbol{b}^1, \ldots, \boldsymbol{b}^r\right)$. In practice, computing the gradient of the empirical risk $\mathcal{L}(f_\theta)$ may be computationally costly with large N (sample size). More, gradient descent applied to $\mathcal{L}(f_\theta)$ can also cause overfitting and lead to an overfitted network f_θ. Hence, we apply stochastic gradient descent (SGD) to train neural networks.

In SGD, we first randomly split the training data into minibatches. The parameters of the Neural Network are updated after each minibatch, and the outputs are inputs to the following minibatch. This procedure is repeated on multiple epochs, with new minibatches, while initialising θ with the last value of the previous epoch.

6.8 Backpropagation

In the Stochastic gradient descent process, we need to be able to compute the gradient $\nabla_\theta \mathcal{L}_B(\theta)$ of minibatch empirical risk $\mathcal{L}_B(\theta)$. The idea is to use finite-difference approximation to approximate the derivative :

$$F'(x) \approx \frac{F(x + \frac{1}{2}\Delta) - F(x - \frac{1}{2}\Delta)}{\Delta}$$

for small $\Delta > 0$. We can use other alternative methods such as symbolic differentiation or algorithmic differentiation, when F is highly non-linear since finite-difference approximation gives bad results in this case.

Backpropogation is a special case of algorithmic differentiation, which consists on the the computation of the gradient of empirical risk for a Neural Network. We study briefly the main ideas behind backpropagation.

We suppose that we are training a neural network $f_\theta \in \mathcal{N}_r(I, d_1, \dots, d_{r-1}, O, \boldsymbol{\sigma}_1, \dots, \boldsymbol{\sigma}_r)$ by SGD, , where we assume, for simplicity, that σ_i is the component-wise application of a one-dimensional activation function $g_i : \mathbb{R}^d \to \mathbb{R}$, for any $i = 1, \dots, r$.

We seek to minimise:

$$\nabla_{\boldsymbol{\theta}} \mathcal{L}_B(\boldsymbol{\theta}) = \frac{1}{\#B} \sum_{i \in B} \nabla \ell\left(\boldsymbol{f}_{\boldsymbol{\theta}}\left(\boldsymbol{x}^i\right), \boldsymbol{y}^i\right)$$

Therefore, we will only need to know how to compute :

$$\nabla \ell\left(\boldsymbol{f}_{\boldsymbol{\theta}}\left(\boldsymbol{x}^i\right), \boldsymbol{y}^i\right)$$

Following the notations in Deep Learning Lecture Notes, we introduce the recursive notations. For $x \in \mathbb{R}^I$

$$\begin{aligned} z^i = \left(z_1^i, \dots, z_{d_i}^i\right) &:= \boldsymbol{L}_i\left(\boldsymbol{a}^{i-1}\right) = W^i \boldsymbol{a}^{i-1} + \boldsymbol{b}^i, \quad & i = 1, \dots, r \\ \boldsymbol{a}^i = \left(a_1^i, \dots, a_{d_i}^i\right) &:= \boldsymbol{g}_i\left(\boldsymbol{z}^i\right), & i = 1, \dots, r \\ \boldsymbol{a}^0 &:= \boldsymbol{x} \in \mathbb{R}^I, & \end{aligned}$$

hence: $\boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{x}) = \boldsymbol{a}^r$ and $\ell\left(\boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{x}), \boldsymbol{y}\right) = \ell\left(\boldsymbol{a}^r, \boldsymbol{y}\right)$. We define the adjoint by: $\boldsymbol{\delta}^i = \left(\delta_1^i, \dots, \delta_{d_i}^i\right) \in \mathbb{R}^{d_i}$

$$\delta_j^i := \frac{\partial \ell}{\partial z_j^i}, \quad j = 1, \dots, d_i, \quad i = 1, \dots, r$$

We recall the chain rule, which is the key idea in Backpropagation. For differentiable $G : \mathbb{R}^d \to \mathbb{R}$ and $\boldsymbol{F} = (F_1, \dots, F_d) : \mathbb{R}^{d'} \to \mathbb{R}^d$, , define $H(x) = G(y)$ with $y = (y_1, \dots, y_d) = F(x)$, that is, $H = G \circ F : \mathbb{R}^{d'} \to \mathbb{R}^d$. Then

$$\frac{\partial H}{\partial x_i}(x) = \sum_{j=1^d} \frac{\partial G}{\partial y_j}(y) \frac{\partial F_j}{\partial x_i}(x), \quad x = (x_1, \dots, x_{d'})$$

Using the chain rule, we find the following results :
Proposition

$$\begin{aligned} \boldsymbol{\delta}^r &= \boldsymbol{g}_r'\left(\boldsymbol{z}^r\right) \odot \nabla_{\hat{y}} \ell\left(\boldsymbol{a}^r, \boldsymbol{y}\right) & \\ \boldsymbol{\delta}^i &= \boldsymbol{g}_i'\left(\boldsymbol{z}^i\right) \odot \left(W^{i+1}\right)' \boldsymbol{\delta}^{i+1}, & \\ \frac{\partial \ell}{\partial b_j^i} &= \delta_j^i, & i = 1, \dots, r-1 \\ \frac{\partial \ell}{\partial W_{j,k}^i} &= \delta_j^i a_k^{i-1}, & i = 1, \dots, r, j = 1, \dots, d_i \end{aligned}$$

where $\odot$ stands for the component-wise Hadamard product of vectors.

7 Creating Deep Learning-Artificial Neural Networks(ANN) model to calculate the CVA

In this chapter, we will build a feed-forward neural network to predict the WWR CVA value of a call option, and will choose its architecture step by step.
The inputs (features) of the neural network are :

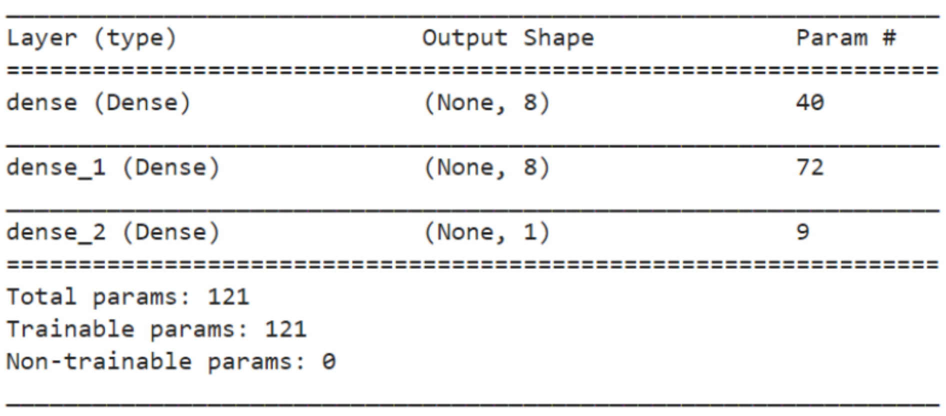
- σ : the volatilty of the stock price process S
- κ : the speed of mean reversion of the stochastic intensity process y
- ν : the volatility of the stochastic intensity process y
- ρ : the correlation between the stock price process and stochastic intensity process

The output of the neural network is the CVA.
First, we center and scale the features of the data set generated previously (200.000 samples). We will train the neural network using 70% of the data and will test it on the remaining data.

7.1 Architecture choice

There is no analytical formula or empirical rule to calculate the appropriate number of hidden layers and units to build the neural network. Tuning the parameters of the model requires brute-force search. This tuning is sometimes called ”black art”.

Choosing the number of units for the input layer and output is obvious : The input layer has 4 units and the output layer has one unit. After testing several architectures we choose to keep 2 hidden layers with 8 units each.



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 8)                 40
_________________________________________________________________
dense_1 (Dense)              (None, 8)                 72
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 9
=================================================================
Total params: 121
Trainable params: 121
Non-trainable params: 0
_________________________________________________________________
```

Figure 13: Total number of parameters of the NN

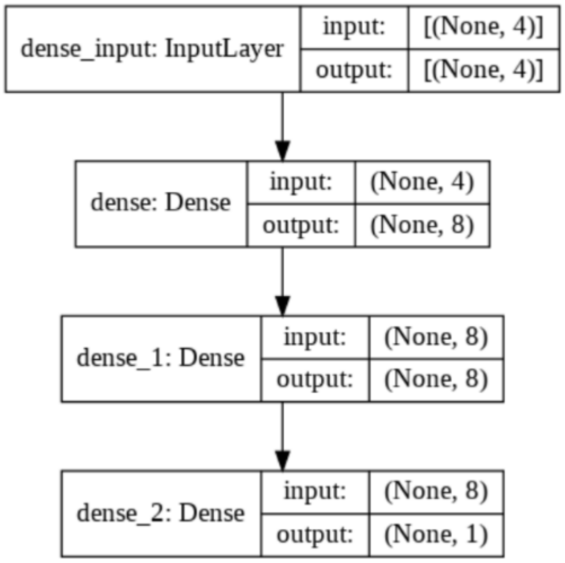


Figure 14: NN Architecture

7.2 Activation function

In hidden layers, the activation function ReLU has become the default activation function for many types of neural networks. This activation function was first introduced by Hahnloser et al. in 2000 with biological motivations and mathematical justifications, and popularised by Glorot et al. (2011)[12].

$$\begin{aligned} \mathrm{ReLU}(x) &= \begin{cases} x \text{ if } x \geq 0 \\ 0 \text{ else} \end{cases} \\ \mathrm{ReLU}'(x) &= \begin{cases} 1 \text{ if } x \geq 0 \\ 0 \text{ else} \end{cases} \end{aligned} \tag{32}$$

The activation function ReLU is popular for many reasons. ReLU and its derivative are mathematically very simple, making them numerically efficient. More, ReLU speeds up the convergence, it is proven that convergence happens six times faster using a ReLU activation function as compared to using a sigmoid or a Tanh activation function.
However, ReLU has a serious limitation. The gradient is either 0 or 1, which can cause the dying ReLU problem studied in ReLU and Initialization: Theory and Numerical Examples (2019) [22]. The dying ReLU refers to the problem when ReLU neurons become inactive and only output 0 for any input, which may freeze gradient based learning algorithms. Indeed, during training, if the output of a certain ReLU neuron is 0 after an improper update of the parameters, then the gradient of the neuron's own parameters will always be 0, and it will never be activated in the

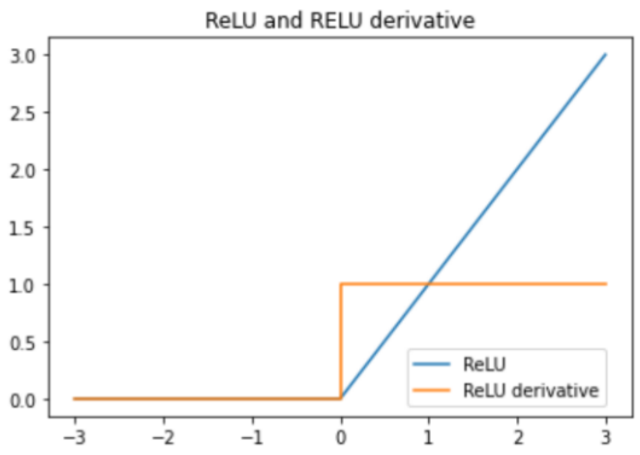


Figure 15: Relu activation function

subsequent training process. The neuron can't improve and is considered as 'dead'. If too many neurons are dead, the learning process will be very slow.

To alleviate the dying ReLU, Maas et al. (2013)[23] and He et al. (2015) [15] proposed the leaky ReLU (LReLU) and parametric ReLU (PReLU), respectively. Both of them have a slight slope in the negative range to prevent the dying ReLU issue, letting them thereby "leak" through the function.

- **Leaky ReLU (LReLU)**: makes the negative part really small but non zero by multiplying by a close to zero constant

$$\text{LeakyRelu}\,(x) = \begin{cases} x \text{ if } x \geq 0 \\ \alpha x \text{ else} \end{cases} \tag{33}$$

- **Parametric ReLU (PReLU)** is a type of leaky ReLU that, instead of having a predetermined slope α, makes it a parameter for the neural network to figure out itself: $y = \alpha x$ when $x < 0$.

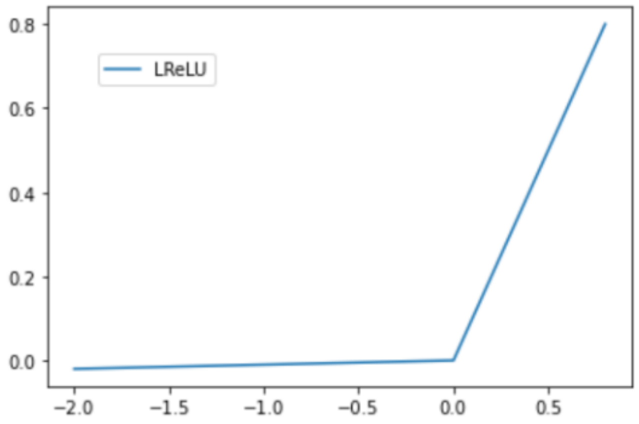


Figure 16: LReLU activation function $\alpha = 0.01$

However these two subtitute activation functions can lead to some issues. Clevert et al in "Fast and accurate deep network learning by exponential linear units", (2015) [8] explains that even though these previous activation functions solve the dying ReLU problem, they have no "noise-robust deactivation state". Therefore, Clevert et al proposed a new substitute of ReLU, the exponential linear unit (ELU) :

$$Elu(x) = \begin{cases} x \text{ if } x \geq 0 \\ \alpha\left(e^x - 1\right) \text{ else} \end{cases} \tag{34}$$

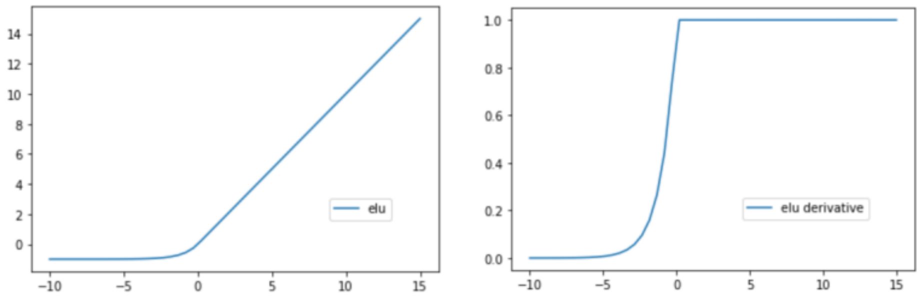


Figure 17: Elu activation function $\alpha = 1$

This new activation function has several advantages :

- deletes the dying ReLU problem
- has a quicker learning process
- it saturates to $-\alpha$, making it robust to noise

Hence, We choose to use Elu as the activation function for the two hidden layers of the neural network. We reached the following architecture for our neural network :

$$\widehat{\text{CVA}} \in \mathcal{N}_3(4, 8, 8, 1; Elu, Elu, Id)$$

7.3 Loss function

Because we are dealing with a regression problem, Mean Absolute Loss Function or Mean Square Loss function are adequate loss functions to train the ANN. We choose to work with the Mean Square Loss function though the two loss functions didn't change our parameters and hyper-parameters calibration results.

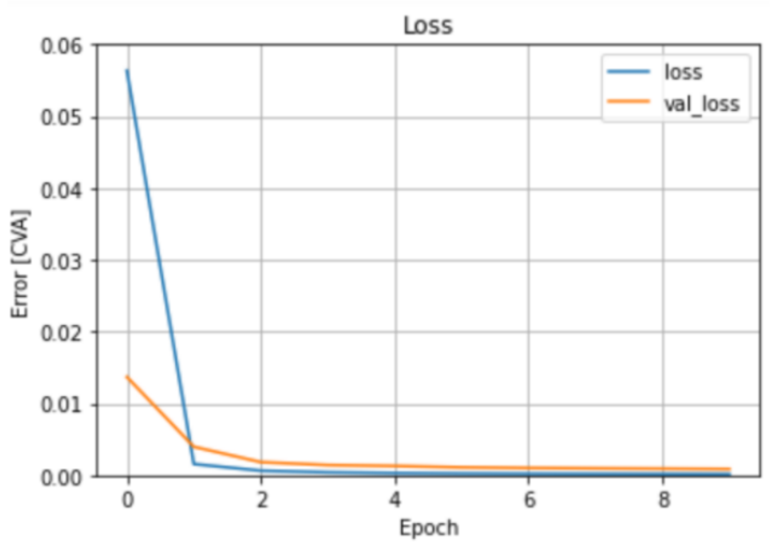


Figure 18: Training loss VS validation loss

7.4 Hyper-parameter tuning of ANN

Finding the best values for batch size and epoch is very important as it directly affects the model performance. Bad values can lead to overfitting or underfitting. Again, there is no thumb rule which can help decide the value of the hyper-parameters of neurons. We will try different parameters and choose the combination which produces the lowest error. To do so, we use the manual grid search method. This is a simple for loop based approach that calculates the training error for all the combination of the batch size and epoch. The table bellow displays the hyper-parameter trial results for the ANN. We choose to take a batch size of 50 and 10 epochs.

Epochs	Batch size	Loss
5	5	0.00045
5	10	0.00029
5	20	0.00006
5	50	0.00018
5	100	0.00004
10	5	0.00037
10	10	0.00029
10	20	0.00006
10	50	0.00003
10	100	0.00005
15	5	0.00035
15	10	0.00017
15	20	0.00006
15	50	0.00006
15	100	0.00005
20	5	0.00056
20	10	0.00005
20	20	0.00004
20	50	0.00004
20	100	0.00006

Table 7: hyper-parameter trial results

7.5 Optimisation Algorithm

Before we can train the neural network, we need to specify our optimisation algorithm. We choose to use the Adam optimizer which stands for ADAptive Moment estimation.

Adam was introduced by Diederik Kingma from OpenAI and Jimmy Ba from the University of Toronto in their 2015 ICLR paper titled "Adam: A Method for Stochastic Optimization"[19]. It is an optimization algorithm that can be used instead of the classical stochastic gradient descent procedure to update network weights iterative based in training data.

Adam combines the advantages of two other extensions of stochastic gradient descent :

- **Adaptive Gradient Algorithm (AdaGrad)** that "maintains a per-parameter learning rate that improves performance on problems with sparse gradients"[5].
- **Root Mean Square Propagation (RMSProp)** [6] that also "maintains per-parameter learning rates that are adapted based on the average of recent magnitudes of

[5] Source: `https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/`
[6] `https://keras.io/api/optimizers/rmsprop`

the gradients for the weight (e.g. how quickly it is changing). This means the algorithm does well on online and non-stationary problems (e.g. noisy)"[5].

Adam optimiser has several attractive advantages. We chose to work with it because it requires little memory, is straightforward to implement and is computationally efficient for small change in hyper-parameters.

7.6 Learning rate

During the training process, we should give an important attention to the learning rate η because it determines the speed of convergence. A low learning rate will calibrate precisely but the training will progress very slowly. A high learning rate calibrates quickly but risk to cause drastic updates that lead to divergent behaviours. To solve this issue and find a compromise, the idea is to have a dynamic learning rate[7], starting from a high and decreasing it progressively.

7.7 Dropout layer

To avoid overfitting, we can apply a regularisation method. One widely used regularisation method is "dropout", introduced by Srivastava at al. (2014) [32]. The idea behind dropout is to add a dropout layer after a hidden layer, where each input gets randomly replaced during the training process of the neural network by zero value with fixed probability $p \in [0,1]$.

In the neural network we built, we introduce a dropout layer with $p = 0.2$ after each hidden layer, and we train the new neural network other settings unchanged. We notice that the validation loss ends up being close to training loss, which is a sign that the overfitting problem has been mitigated.

	Training Loss	Validation Loss
without dropout layers	0.00001	0.000033
with dropout layers	0.00002	0.000031

Table 8: Training and validation Loss

7.8 Neural Network implementation using Keras

We will now dive into to the process of implementing a neural network on Python. TensorFlow is arguably the most popular deep learning software library at the moment (alongside PyTorch). The library was developed by Google Brain for internal use at Google initially, then was released to the public as open source in 2015.

Keras, developed by Google engineer François Chollet [7], is a high-level deep learning library that makes the implementation and training of neural networks easy and

[7]https://keras.io/api/callbacks/reduce_lr_on_plateau/

fast. It is part of the TensorFlow core library. A detailed introduction to Keras is given in Chollet (2018) [7]. We will use this library to implement our neural network

Similarly to the numerical simulation of the CVA, we need to fix the randomness of the neural network in order to get the same results at each training. To do that, we fix the seed in keras[8] in order to have the same result every time.

[8]`https://www.tensorflow.org/api_docs/python/tf/random/set_seed`

8 Results

We use the neural network built in the previous chapter to compute CVA for different parameters. The figures bellow gives the CVA results with Deep Learning.

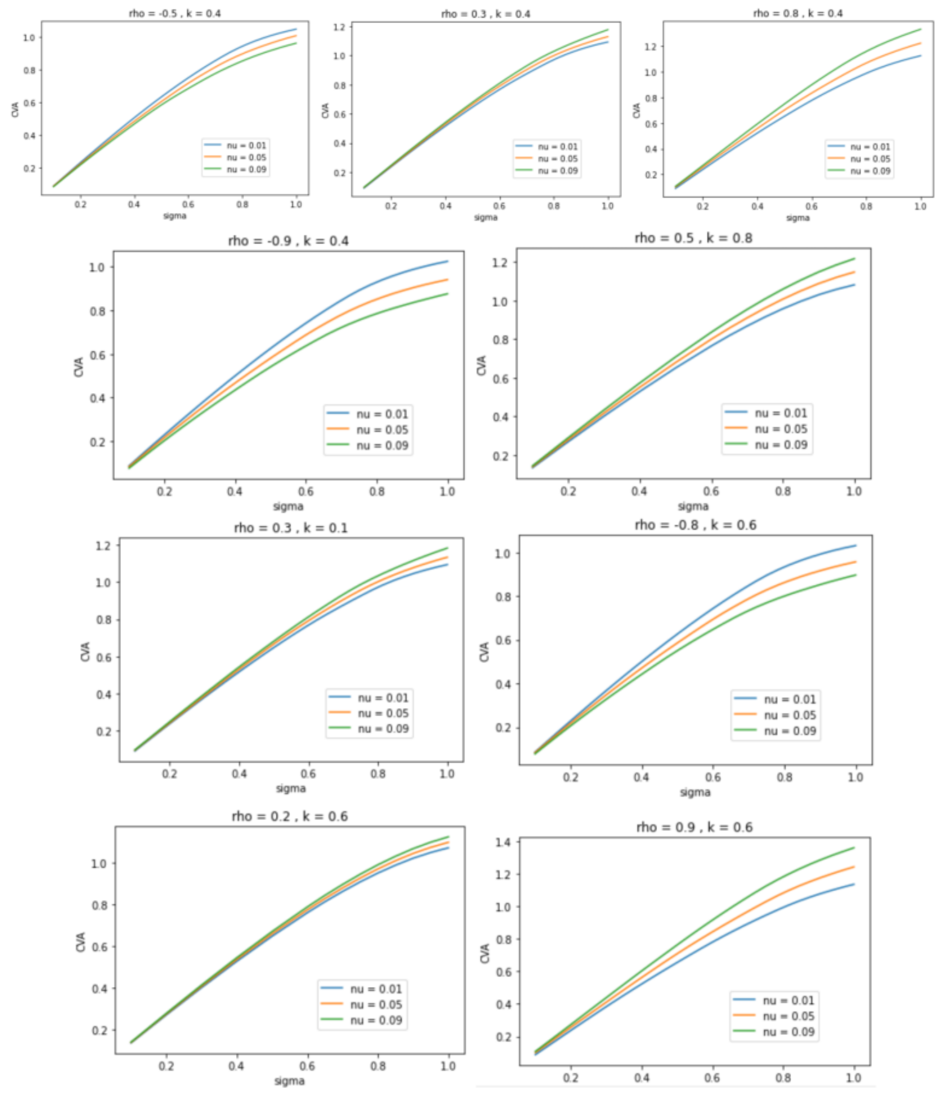


Figure 19: CVA(sigma)

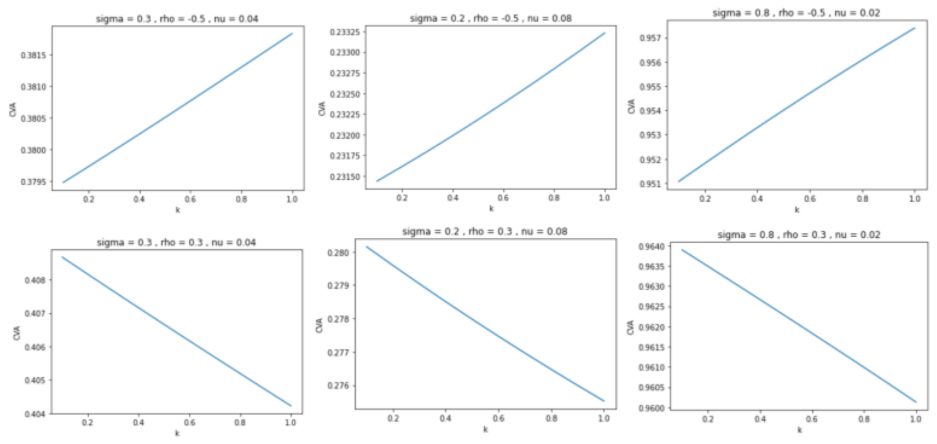


Figure 20: CVA(κ)

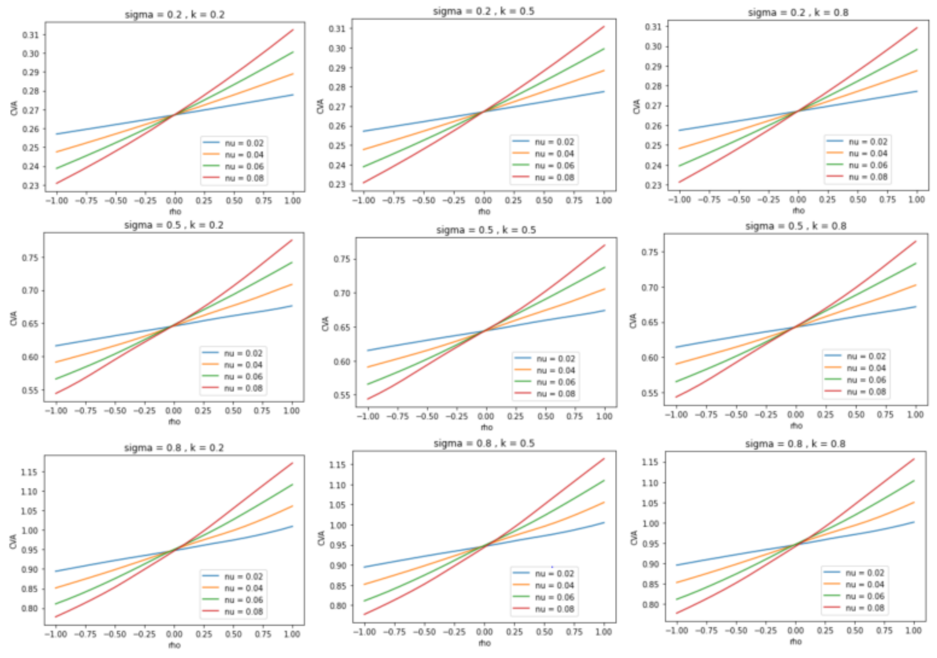


Figure 21: CVA(rho)

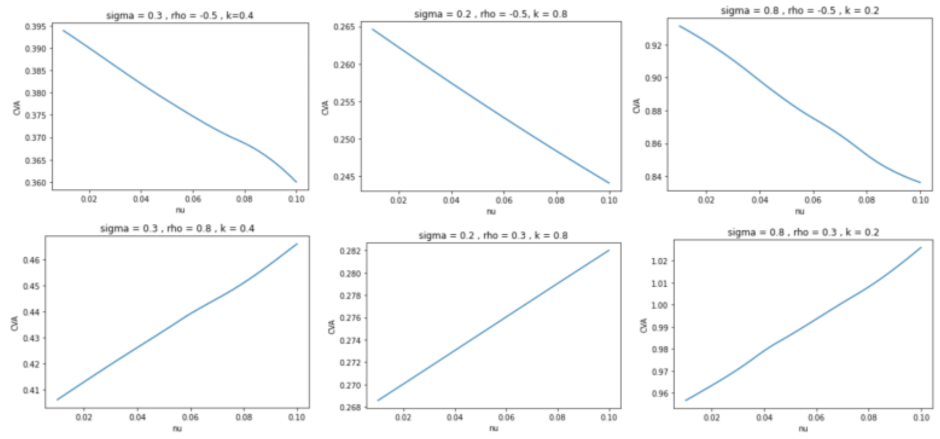


Figure 22: CVA(ν)

We notice that, with Deep Learning, CVA results are given instantaneously. The neural network was able to map well the CVA values. We have similar shapes and values as in the graphs given by simulation. However, we can see a slight difference between the shape of the plots of $CVA(\kappa)$ for the simulation and for the neural network. This is because κ has a low impact on the CVA values when the maturity is small. Finally, CVA values and errors given by the neural network are very stable on the out-of sample data.

9 Conclusion

In this thesis, we investigated the computation of Credit Value Adjustment, when taking into account the correlation between credit and market risk. This calculation requires a model as counterparty risk adds a level of optionality to the payoff. To derive a CVA formula, we used intensity models. We focused our study on computing the CVA of at the money call options. In order to keep the computation tractable, we considered a square root diffusion model driving the intensity of the investor and counterparty. More, we modelled the underlying stock using the Black-Scholes framework. Wrong Way Risk (WWR) is embedded in the correlation between the stock price and default intensity.

Secondly, we simulated CVA values of call options using Monte Carlo Simulation. We chose to fix the following parameters : the time to maturity of the call option T (1 year), the initial underlying stock price S_0 and strike price K (100), the initial value of the intensity process y_0 (0.01), the mean reversion level of the intensity process μ (0.03), the implied hazard functions Γ^{Mkt} (0.03). In order to improve the Monte Carlo Simulation, we used two common variance reduction methods : antithetic variate method and control variate method. Finally, we simulated 200.000 samples of CVA under intensity models for in the money call options using the control variate method.

We then studied the impact of the four parameters we haven't fixed, on the CVA value: the correlation ρ, the volatility of the stock price process σ, the volatility of the intensity process ν and the speed of mean reversion of the intensity process κ. We noticed that the CVA is an increasing function of σ and ρ. More, we saw that the parameters κ and ν fight each other. For negative correlations, the CVA is increasing with κ and decreasing with ν . On the other hand, for positive correlations, the CVA is increasing with ν and decreasing with κ.

Afterwards, we tried to apply Deep Learning techniques to compute CVA values faster. To do so, we built a feed-forward neural network and trained it using the 200.000 samples generated via the Monte Carlo simulation. The trained neural network tried to map the CVA value for different correlations, volatilities and speed of mean reversion of the stochastic intensity process. The results showed that Deep Learning was able to accelerate the computation of CVA. We conclude that it is plausible to think that neural networks could be used in CVA computation to accelerate the calculation of Wrong way risk, which if done with simulation would take a very long time.

As an extension to this work, we could study the interpretability of the neural network. A future goal of this thesis would also be to use longer maturities of the call options and learn their impact on CVA. We could also use Deep Learning approach to compute WWR CVA of other financial products (swaptions, caps ...)

A Appendix

	Activation	Definition	Derivative	Range	Smoothness
	Identity (Id)	$g(x) = x$	$g'(x) = 1$	$\mathbb{R}$	C^∞
	Sigmoid (Logistic, σ)	$g(x) = \frac{1}{1+e^{-x}}$	$g'(x) = g(x)(1-g(x))$	$(0,1)$	C^∞
	Heaviside (H)	$g(x) = \begin{cases} 0, & x < 0 \\ 1, & x \geq 0 \end{cases}$	$g'(x) = 0,\ x \neq 0$	$\{0,1\}$	none
	Hyperbolic tangent (tanh)	$g(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$	$g'(x) = 1 - g(x)^2$	$(-1,1)$	C^∞
	Rectified linear unit (ReLU)	$g(x) = \max\{x, 0\}$	$g'(x) = \begin{cases} 0, & x < 0 \\ 1, & x > 0 \end{cases}$	$[0,\infty)$	C
	Parametric rectified linear unit (PReLU)	$g(x) = \begin{cases} \alpha x, & x < 0 \\ x, & x \geq 0 \end{cases}$ $\alpha > 0$	$g'(x) = \begin{cases} \alpha, & x < 0 \\ 1, & x > 0 \end{cases}$	$\mathbb{R}$	C
	Exponential linear unit (ELU)	$g(x) = \begin{cases} \alpha(e^x - 1), & x < 0 \\ x, & x \geq 0 \end{cases}$ $\alpha > 0$	$g'(x) = \begin{cases} g(x) + \alpha, & x < 0 \\ 1, & x > 0 \end{cases}$	$(-\alpha,\infty)$	$C^1,\ \alpha = 1$ $C,\ \alpha \neq 1$
	Softplus	$g(x) = \log(1 + e^x)$	$g'(x) = \frac{1}{1+e^{-x}}$	$(0,\infty)$	C^∞
	Gaussian	$g(x) = e^{-x^2}$	$g'(x) = -2xg(x)$	$(0,1]$	C^∞

Figure 23: Common one-dimensional activation functions and their properties. Adapted from Wikipedia (2019).

Activation	Definition	Derivative	Range	Smoothness
Softmax	$g_i(\boldsymbol{x}) = \frac{e^{x_i}}{\sum_{j=1}^{d} e^{x_j}},\ i = 1,\ldots,d$	$\frac{\partial g_i(\boldsymbol{x})}{\partial x_j} = \begin{cases} g_i(\boldsymbol{x})(1 - g_i(\boldsymbol{x})), & i = j \\ -g_i(\boldsymbol{x})g_j(\boldsymbol{x}), & i \neq j \end{cases}$	$(0,1)^d$	C^∞
Maxout	$g(\boldsymbol{x}) = \max\{x_1,\ldots,x_d\}$	$\frac{\partial g(\boldsymbol{x})}{\partial x_i} = \begin{cases} 1, & x_i > \max_{j \neq i} x_j \\ 0, & x_i < \max_{j \neq i} x_j \end{cases}$	$\mathbb{R}$	C

Figure 24: Common multi-dimensional activation functions and their properties. Adapted from Wikipedia (2019).

References

[1] Aurélien Alfonsi. "On the discretization schemes for the CIR (and Bessel squared) processes". In: (2005).

[2] BIS Basel Committee on Banking Supervision. "Basel Committee finalises capital treatment for bilateral counterparty credit risk". In: (2011).

[3] Damiano Brigo. "Interest Rate Models with Credit Risk, Collateral,Funding Liquidity Risk and Multiple Curves lecture notes". In: (2020).

[4] Damiano Brigo and Aurélien Alfonsi. "Credit default swap calibration and derivatives pricing with the SSRD stochastic intensity model". In: *Finance and stochastics* 9.1 (2005), pp. 29–42.

[5] Damiano Brigo, Agostino Capponi, and Andrea Pallavicini. "Arbitrage-free bilateral counterparty risk valuation under collateralization and application to credit default swaps". In: *Mathematical Finance: An International Journal of Mathematics, Statistics and Financial Economics* 24.1 (2014), pp. 125–146.

[6] Damiano Brigo, Massimo Morini, and Andrea Pallavicini. *Counterparty credit risk, collateral and funding: with pricing cases for all asset classes*. Vol. 478. John Wiley & Sons, 2013.

[7] Francois Chollet. *Deep learning with Python*. Simon and Schuster, 2017.

[8] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. "Fast and accurate deep network learning by exponential linear units (elus)". In: *arXiv preprint arXiv:1511.07289* (2015).

[9] Griselda Deelstra, Freddy Delbaen, et al. "Convergence of discretized stochastic (interest rate) processes with stochastic drift term". In: *Applied stochastic models and data analysis* 14.1 (1998), pp. 77–84.

[10] Awa Diop. "Sur la discrétisation et le comportement à petit bruit d'EDS unidimensionnelles dont les coefficients sont à dérivées singulières". PhD thesis. Nice, 2003.

[11] Samim Ghamami and Lisa R Goldberg. "Stochastic intensity models of wrong way risk: wrong way CVA need not exceed independent CVA". In: *The Journal of Derivatives* 21.3 (2014), pp. 24–35.

[12] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. "Deep sparse rectifier neural networks". In: *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings. 2011, pp. 315–323.

[13] Jon Gregory. *Counterparty credit risk and credit value adjustment: A continuing challenge for global financial markets*. John Wiley & Sons, 2012.

[14] Jon Gregory. *Counterparty credit risk and credit value adjustment: A continuing challenge for global financial markets*. John Wiley & Sons, 2012.

[15] Kaiming He et al. "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification". In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 1026–1034.

[16] Martin Hellander. *Credit Value Adjustment: The Aspects of Pricing Counterparty Credit Risk on Interest Rate Swaps*. 2015.

[17] Catherine F Higham and Desmond J Higham. “Deep learning: An introduction for applied mathematicians”. In: *Siam review* 61.4 (2019), pp. 860–891.

[18] Vandit Jain. “Everything you need to know about “Activation Functions” in Deep learning models”. In: (Dec. 2019). URL: `https://towardsdatascience.com/everything-you-need-to-know-about-activation-functions-in-deep-learning-models-84ba9f82c253`.

[19] Diederik P Kingma and Jimmy Ba. “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980* (2014).

[20] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “ImageNet classification with deep convolutional neural networks”. In: *Advances in neural information processing systems* 25 (2012), pp. 1097–1105.

[21] Moshe Leshno et al. “Multilayer feedforward networks with a nonpolynomial activation function can approximate any function”. In: *Neural networks* 6.6 (1993), pp. 861–867.

[22] Lu Lu et al. “Dying relu and initialization: Theory and numerical examples”. In: *arXiv preprint arXiv:1903.06733* (2019).

[23] Andrew L Maas, Awni Y Hannun, Andrew Y Ng, et al. “Rectifier nonlinearities improve neural network acoustic models”. In: *Proc. icml*. Vol. 30. 1. Citeseer. 2013, p. 3.

[24] Warren S McCulloch and Walter Pitts. “A logical calculus of the ideas immanent in nervous activity”. In: *The bulletin of mathematical biophysics* 5.4 (1943), pp. 115–133.

[25] Marvin Minsky and Seymour A Papert. *Perceptrons: An introduction to computational geometry*. MIT press, 2017.

[26] Marianne Jocelyn Toscano Montoya. “Deep intensity-based CVA with Wrong Way Risk”. In: (2021).

[27] Mikko Pakkanen. “Deep learning lecture notes”. In: (2020).

[28] Frank Rosenblatt. “The perceptron: a probabilistic model for information storage and organization in the brain.” In: *Psychological review* 65.6 (1958), p. 386.

[29] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. “Learning representations by back-propagating errors”. In: *nature* 323.6088 (1986), pp. 533–536.

[30] Filippo Santambrogio. “{Euclidean, metric, and Wasserstein} gradient flows: an overview”. In: *Bulletin of Mathematical Sciences* 7.1 (2017), pp. 87–154.

[31] David Silver et al. “Mastering the game of Go with deep neural networks and tree search”. In: *nature* 529.7587 (2016), pp. 484–489.

[32] Nitish Srivastava et al. “Dropout: a simple way to prevent neural networks from overfitting”. In: *The journal of machine learning research* 15.1 (2014), pp. 1929–1958.

[33] Hugues Thorin. “Artificial Neural Networks for SABR model calibration hedging”. In: (2019).

EL_MOUDEN_Wiam_01948802

PAGE 21

PAGE 22

PAGE 23

PAGE 24

PAGE 25

PAGE 26

PAGE 27

PAGE 28

PAGE 29

PAGE 30

PAGE 31

PAGE 32

PAGE 33

PAGE 34

PAGE 35

PAGE 36

PAGE 37

PAGE 38

PAGE 39

PAGE 40

PAGE 41

PAGE 42

PAGE 43

PAGE 44

PAGE 45

PAGE 46

PAGE 47

PAGE 48

PAGE 49

PAGE 50

PAGE 51

PAGE 52

PAGE 53

PAGE 54

PAGE 55

PAGE 56

PAGE 57

PAGE 58