Neural Network for CVA: Learning Future Values

Jian-Huang She*, Dan Grecu†

Revised version: Nov. 6, 2018
First version: Nov. 6, 2018

Abstract

A new challenge to quantitative finance after the recent financial crisis is the study of credit valuation adjustment (CVA), which requires modeling of the future values of a portfolio. In this paper, following recent work in [2, 3], we apply deep learning to attack this problem. The future values are parameterized by neural networks, and the parameters are then determined through optimization. Two concrete products are studied: Bermudan swaption and Mark-to-Market cross-currency swap. We obtain their expected positive/negative exposures, and further study the resulting functional form of future values. Such an approach represents a new framework for modeling XVA, and it also sheds new lights on other methods like American Monte Carlo.

1 Introduction

The 2008 financial crisis has brought to central stage counterparty credit risk. Thereafter it has become a crucial task for banks to evaluate the Credit Valuation Adjustment (CVA), or more broadly XVA, including in addition Debit Valuation Adjustment(DVA), Funding Value Adjustment (FVA), Margin Value Adjustment (MVA) etc. The key concept for XVA is the future value of a portfolio. Henceforth, in addition to time-zero price, we now need to model the distribution of prices in the future. Furthermore for collateralized trades, we need to model the joint distribution of prices at different times in the future. For MVA, we need to model derivatives of future values (future greeks).

In fact, in the (time-zero) pricing of products with early exercise features, e.g. Bermudan swaptions, one already faces the problem of obtaining prices at future time points: the holding values at the exercise dates. This points to the

*Corporate Model Risk, Wells Fargo Bank, email contact: Jian-huang.She@wellsfargo.com
†Corporate Model Risk, Wells Fargo Bank

close relationship between XVA and callable products. For callable products, while partial differntial equations (PDE) and tree methods work well in low dimensions, when dimension becomes high, e.g. for Libor Market Model, the only resort is still the Monte Carlo method.

To draw inspiration, let us examine more closely the use of Monte Carlo method for callable products. It is clear that conventional Monte Carlo method does not work here, since each future value requires a separate Monte Carlo pricer, and such nested Monte Carlo method is too costly. The so-called American Monte Carlo (AMC) is used instead [1]. To proceed, one starts with the seemingly obvious observation that future value is determined by the information available till that date. More formally, future value is the conditional expectation value filtered on information available at the corresponding time in the future. The direct consequence of this observation is that future value is a function of the historical time series of all the risk factors on a single Monte Carlo path. Without any barrier or Asian feature, it is further simplified that future value is simply a function of all the risk factors on that particular date and for that particular path. That is, for all the risk factors X_i with $i = 1, \cdots, d$, the future value is a function of these d-variables: $V(T_n) = f(X_1(T_n), \cdots, X_d(T_n))$. Note that this function is generally highly nonlinear, and it is defined in high dimensions.

One way to determine such a function is through AMC, which proceeds with the further observations that (1) given a Monte Carlo path for the whole life of the trade, one can obtain the trade value at a given date for the particular path from discounted cash-flows; (2) the value thus obtained represents a sampling of the unknown function $f(X_1, \cdots, X_d)$. This way one can obtain a large number of samplings for a single function, and statistical methods can then be used to infer this function. In practice, linear regression is usually used due to its efficiency. The linear regression method assumes fair amount of prior knowledge of the problem at hand, in the form of basis functions. And the performance of this method depends crucially on the intelligent choice of these base functions. For example the prices of the more liquid products that are used to hedge this product are usually included in the set of base functions.

Here we explore the possibility of determining the functional form of the future value without using as much prior knowledge of the products. More concretely we seek to to find a "universal approximator" for the high dimensional, non-linear function $f(X_1, \cdots, X_d)$. We follow closely the seminal work of Wei-Nan E and collaborators (see [2, 3]) to use neural network (NN) as such a "universal approximator". This approach is largely inspired by recent success in applying NN to various areas such as computer vision, speech recognition, machine translation, playing board games and medical diagnosis (see [4, 5, 6, 7]). Two attractive features of the neural network are (1) the power to represent high-dimensional non-linear functions [8, 9], (2) the easiness to determine the involved parameters using efficient optimization algorithms [10].

NN approach has been applied to CVA in [11] using a dual formulation of stochastic control problems, and to time-zero pricing of Bermudan swaption using Libor-Market Model in [12]. A new algorithm has been proposed in [13] to

directly parameterize the future value with NN, exploring the power of automatic differentiation.

The rest of this paper is organized as follows. In section 2, we review the basic concepts of CVA/DVA, and then lay out the neural network approach to model future values and compute CVA/DVA. In section 3 and 4, two concrete products, i.e. Bermudan swaptions and Mark-to-Market cross currency swaps, are studied. Their EPE/ENE are computed and the functional form of future values are studied. In section 5, we summarize our approach, and comment on future directions.

2 The formalism

2.1 CVA/DVA

CVA and DVA are the risk-neutral prices of counterparty risk. They are defined as (see e.g. [14, 15]):

$$CVA = \int_0^T E_0^Q \left[(1 - R_C(t)) D(0,t) V^+(t) 1_{t \le \tau_C < t+dt} \right], \quad (2.1)$$

$$DVA = \int_0^T E_0^Q \left[(1 - R_B(t)) D(0,t) V^-(t) 1_{t \le \tau_B < t+dt} \right], \quad (2.2)$$

with discount factor $D(0,t)$, future value of the portfolio $V(t)$, recovery rate R_C and R_B, default time τ_C and τ_B for the counterparty (C) and bank (B). Here $V^+ \equiv max(0,V)$, $V^- \equiv min(0,V)$. The indicator functions represent the conditions that counterparty/bank defaults in the time interval $[t, t+dt)$. The expectation is taken under risk-neutral measure conditioned on information at time zero, i.e. valuation time.

Under the assumption that default events and future prices are independent, CVA/DVA can be separated into the market part and credit part:

$$CVA = \int_0^T (1 - R_C(t)) EPE(t) dPr_C(t), \quad (2.3)$$

$$DVA = \int_0^T (1 - R_B(t)) ENE(t) dPr_B(t). \quad (2.4)$$

The credit part is encoded in the recovery rates $R_{C/B}$ and default probabilities $dPr_C(t) \equiv Pr(t \le \tau_C < t+dt)$, and $dPr_B(t) \equiv Pr(t \le \tau_B < t+dt)$. The market part is encoded in the expected positive exposure (EPE) and expected negative exposure (ENE):

$$EPE(t) = E_0^Q \left[D(0,t) V^+(t) \right], \quad (2.5)$$

$$ENE(t) = E_0^Q \left[D(0,t) V^-(t) \right]. \quad (2.6)$$

Note that EPE represents a call option on the portfolio with strike zero, and ENE a put option. In this paper, we will focus on EPE and ENE, which involve the modeling of future values $V(t)$.

2.2 Neural network for future values

We invoke deep learning to model future values. In this subsection we outline the formulation of the problem. While the approach of [2, 3] starts with PDE, and then converts them to backward stochastic differential equations (BSDE [16, 17, 18]), we find it more convenient to start direcly in the framework of BSDE. We start with the risk factor dynamics. We consider d risk factors X_i with $i = 1, \cdots, d$ (think for example the Libor forward rates), and $\mathbf{X}_t \equiv (X_1(t), \cdots, X_d(t))$. Their dynamics reads:

$$dX_i(t) = \mu_i(t, \mathbf{X}_t)\, dt + \sum_j \sigma_{ij}(t, \mathbf{X}_t)\, dW_j(t), \tag{2.7}$$

with the correlation $< dW_j(t), dW_k(t) >= \rho_{jk} dt$, and $j, k = 1, \cdots, K$ from a K dimensional Brownian motion.

Consider then the future value $V(t)$, which is defined as the conditional expectation filtered on information available at time t:

$$V(t) = B(t)\, E\left[\sum_n \frac{CF(T_n)}{B(T_n)} | \mathcal{F}_t\right], \tag{2.8}$$

with cashflow CF and numeraire B. The natural variable for BSDE of future value is the relative value to the numeraire $B(t)$, i.e. $\tilde{V}(t) \equiv V(t)/B(t)$, since it is a martingale under the corresponding measure. The resulting dynamics involves no drift but only diffusion:

$$d\tilde{V}(t) = \sum_{ij} \frac{\partial \tilde{V}}{\partial X_i}(t, \mathbf{X}_t)\, \sigma_{ij}(t, \mathbf{X}_t)\, dW_j(t). \tag{2.9}$$

If needed, one can then obtain the BSDE for $V(t)$ from the above equation.

Of central importance is the unknown vector function $\frac{\partial \tilde{V}}{\partial X_i}(t, \mathbf{X}_t)$, which in financial terms represents Delta for the corresponding risk factors. We discretize the time direction. At a given time step T_n, each Delta is a function of all the risk factors at T_n:

$$\frac{\partial \tilde{V}}{\partial X_i}(T_n) = f_i^{(n)}(X_1, \cdots, X_d), \tag{2.10}$$

where the functional form $f_i^{(n)}$ is unkown. The insight of [2, 3] is to parameterize this function by a neural network:

$$\frac{\partial \tilde{V}}{\partial X_i}(T_n) \simeq \mathcal{F}_i^{(n)}(X_1, \cdots, X_d), \tag{2.11}$$

and then obtain the functional form through optimization.

Consider a fully-connected neural network, which is formed by repeatedly applying two simple operations:

- linear combination of all input variables, i.e. $z_j = \sum_i w_{ji} x_i$.

- nonlinear mapping of a single variable, i.e. $y_j = \phi(z_j)$, where $\phi(z)$ can be $\tanh(z)$, $\max\{z, 0\}$ (relu), $1/(1+e^{-z})$ (sigmoid).

We can compare the neural network representation of a function with that of a polynomial representation: the crucial difference is that neural network can have multiple layers. With enough layers, the neural network can essentially represent arbitrarily complicated non-linear functions. The power of this approach really comes from the existence of a fast optimization algorithm [10].

Once the functional forms of the Delta's are given, the evolution of the discounted future values is determined:

$$\tilde{V}(T_{n+1}) = \tilde{V}(T_n) + \sum_{ij} \mathcal{F}_i^{(n)}(\mathbf{X}_n)\, \sigma_{ij}(T_n, \mathbf{X}_n) \left[W_j(T_{n+1}) - W_j(T_n)\right]. \quad (2.12)$$

In addition, when there are cashflows or option exercises, the portfolio values can jump. We use $T_n^{\pm}$ to represent times immediately after and before date T_n. At the cashflow dates, one has the jump condition:

$$V\left(T_n^-\right) - V\left(T_n^+\right) = CF(T_n). \quad (2.13)$$

At the option exercise dates, one has the jump condition:

$$V\left(T_n^-\right) = \max\left\{V\left(T_n^+\right), U_n(T_n)\right\}, \quad (2.14)$$

where $U_n(T_n)$ denotes the exercise value. Note the asymmetry in time for the option exercise condition: given the exercise value $U_n(T_n)$, we can determine $V(T_n^-)$ from $V(T_n^+)$, but we can not determine $V(T_n^+)$ from $V(T_n^-)$. Mathematically it is because the max function does not have an inverse function. Financially it is because fair values are determined by expectations of the future. The consequence is that for products involving early exercise, one has to evolve the portfolio value backward in time:

$$\tilde{V}(T_n) = \tilde{V}(T_{n+1}) - \sum_{ij} \mathcal{F}_i^{(n)}(\mathbf{X}_n)\, \sigma_{ij}(T_n, \mathbf{X}_n) \left[W_j(T_{n+1}) - W_j(T_n)\right]. \quad (2.15)$$

This applies to all formalisms including PDE, trees, AMC, and also NN method (see [12]).

Given the equations governing the evolution of future value, to obtain its full history, we still need the boundary conditions. The final condition is that right after the maturity date T_N, the portfolio value should be zero:

$$V\left(T_N^+\right) = 0. \quad (2.16)$$

The initial condition is unknown, and will be parameterized and obtained through optimization. We parameterize the initial value $V(0) = V_0$, and the initial Delta's $\frac{\partial V}{\partial X_i}(0) = Z_i^{(0)}$.

Given (1) the Monte Carlo paths of the risk factors $X_i(T_n, \omega_p)$, where ω_p denotes a Monte Carlo path, (2) initial condition for future values, in terms of

the parameters V_0 and $Z_i^{(0)}$, (3) BSDE for future values, with neural network parameters $w_i^{(n)}$, we can obtain the "tentative" history of all the future values $V(T_n, \omega_p)$. From such a history, we can construct a loss function that represents the deviation of the involved trial parameters $\left(V_0, Z_i^{(0)}, w_i^{(n)}\right)$ from their "real" values. One possible choice for the loss function is the mean squared error of the future values from the target values

$$\mathcal{L}\left(V_0, Z_i^{(0)}, w_i^{(n)}\right) = \frac{1}{\mathcal{A}} \sum_{Mp} \left[V(T_M, \omega_p) - V_{\text{target}}(T_M)\right]^2, \tag{2.17}$$

with a normalization factor $\mathcal{A}$, e.g. number of paths. If forward induction is used, we can use the deviation at the maturity date, i.e. $T_M = T_N$, where $V_{\text{target}} = 0$. If backward induction is used, e.g. with early exercise, we can use the deviation at the valuation date $T_M = T_0$, where $V_{\text{target}} = V_0$.

With the constructed loss function, one applies optimization to train the neural network. We use Adam optimization algorithm [19], which has been widely adopted for recent deep learning applications in computer vision and natural language processing.

2.3 Exposure calculation

Once the future values are known, the EPE and ENE can be computed from Eq.(2.5) and (2.6). As we already have the Monte Carlo set up, this step is relatively straightforward for linear products. We evolve the future values one more time with the already trained parameters, and in this process, compute EPE and ENE according to Eq.(2.5) and (2.6).

The exposure calculation for options is more involved. We still evolve the future values one more time with the trained parameters, but in this process, we also need to keep track of the exercise time τ_p for each Monte Carlo path ω_p. If the option has not been exercised, the portfolio value is the value of the option. If the option has been exercised, there are two different cases that need to be treated separately: for cash-settled option, the portfolio value is zero; for physically-settled option, the portfolio value is the value of the underlying. To summarize, one has

$$V(t, X_p) = \begin{cases} V_{\text{option}}(t, X_p) & \text{for } \tau_p > t, \\ 0 & \text{for } \tau_p \leq t \text{ and cash-settled}, \\ V_{\text{underlying}}(t, X_p) & \text{for } \tau_p \leq t \text{ and physically-settled}. \end{cases} \tag{2.18}$$

3 Bermudan Swaption

3.1 The algorithm

As a concrete example, we consider in this section Bermudan swaption. A Bermudan swaption is an option to enter into a swap contract on a given set of dates (exercise dates). For simplicity, we consider the cash-settled case, for

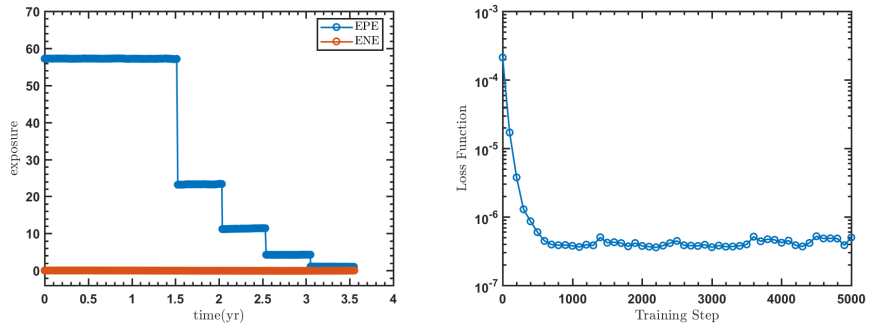

Figure 3.1: Left: EPE and ENE of cash-settled Bermudan swaption. Right: Evolution of loss function with training steps.

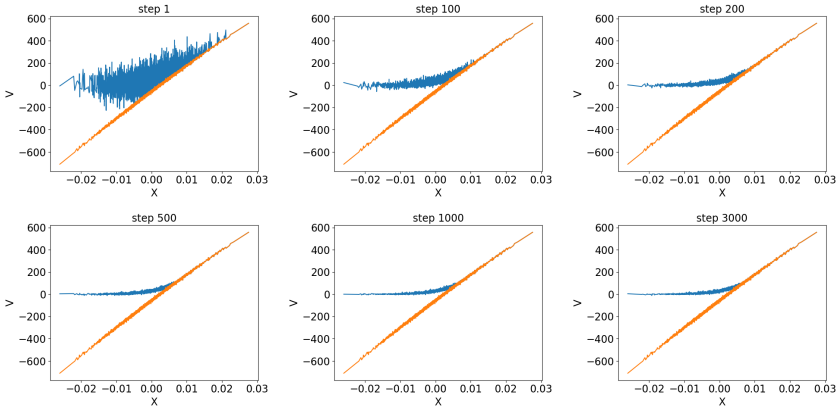

Figure 3.2: Evolution of future values of Bermudan swaption with training (1st exercise date). Blue lines represent the portfolio value, and orange lines the exercise value.

which the exposure vanishes after the option is exercised. The calculation of the exposure of a Bermudan swaption involves three stages. The first stage (pre-training) uses forward induction to generate all quantities that do not require knowledge of the neural network. In this stage, the following operations are carried out:

- Forward evolve the risk factors, and store them in a tensor X_{ipn}, with dimension index i, path index p, and time index n.
- Compute the exercise values, and store them in a tensor U_{pm}, with path index p, time index m, where T_m represent the exercise dates.

The second stage is to construct and train the neural network, which uses backward induction. In this stage, the following operations are carried out:

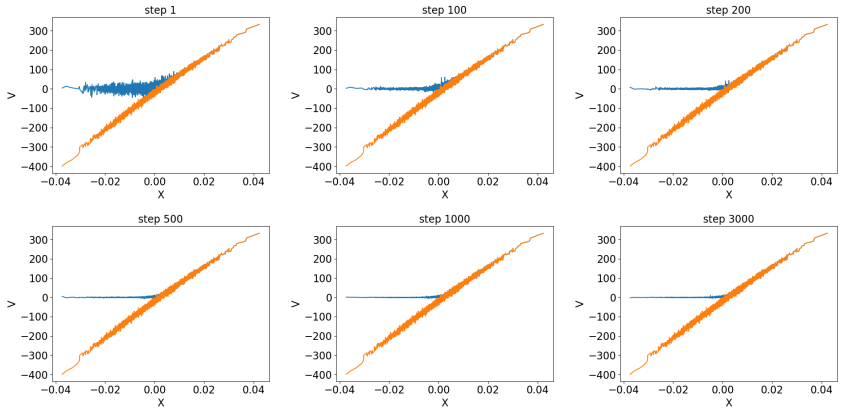

Figure 3.3: Evolution of future values of Bermudan swaption with training (4th exercise date).

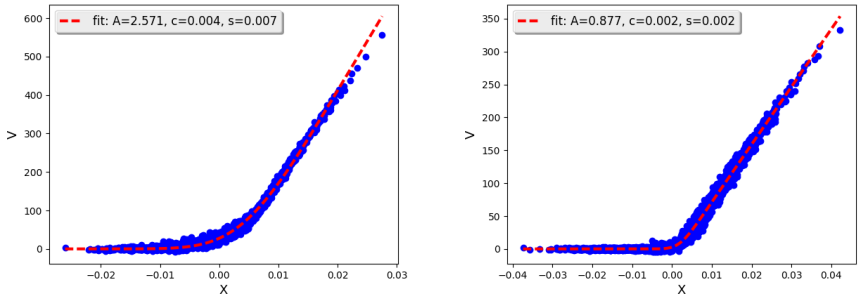

Figure 3.4: Fitting future values of Bermudan swaptions to Bachelier formula.

- Build the neural network. At each time step T_n, construct a neural network for the Delta's, i.e. $\frac{\partial V}{\partial X_i} \equiv Z_i$. Each neural network is in the form of a function that maps a vector $\mathbf{X}_{pn} \equiv (X_{1pn}, \cdots, X_{dpn})$ to a vector $\mathbf{Z}_{pn} \equiv (Z_{1pn}, \cdots, Z_{dpn})$, i.e. $\mathcal{F}^{(n)} : \mathbf{X}_{pn} \to \mathbf{Z}_{pn}$.
- Backward induction. Evolve the future value according to the diffusion equation (2.15) in backward form and the jump condition (2.14) for early exercise. The results are stored in a tensor V_{pn}, with path index p, and time index n.
- Construct the loss function from V_{pn}. The loss function is of the form

$$\mathcal{L}\left(V_0, Z_i^{(0)}, w_i^{(n)}\right) = \frac{1}{\mathcal{A}} \sum_p \left(V_{p0} - V_0\right)^2 . \tag{3.1}$$

- Train the neural network, using for example the Adam optimizer.

- Store the future values in the last run in a tensor V_{pn}. In addition, for the exposure calculation, we also need to extract the exercise indicator η_{pm} at each exercise date T_m for each path ω_p, where

$$\eta_{pm} = \begin{cases} 0 & \text{if exercised at } T_m, \\ 1 & \text{otherwise.} \end{cases} \tag{3.2}$$

The third stage (post-training) is to use the stored future values and exercise indicators to compute the exposure. The following operations are carried out:

- Forward induction. The purpose is to obtain an indicator at each credit time T_n to represent whether the option has been exercised on this path. This can be achieved by simply multiplying all the stored exercise indicators before T_n, i.e. $\tilde{\eta}_{pn} = \prod_{m<n} \eta_{pm}$. Here $\tilde{\eta}_{pn} = 0$ if the option has been exercised before time T_n, and $\tilde{\eta}_{pn} = 1$ otherwise.

- Averaging. EPE/ENE at each credit date is obtained by averaging the positive/negative part of the future value for paths where the option has not been exercised:

$$EPE\left(T_n\right) = E\left[\left(\tilde{\eta}_{pn} V_{pn}\right)^+\right], \tag{3.3}$$

$$ENE\left(T_n\right) = E\left[\left(\tilde{\eta}_{pn} V_{pn}\right)^-\right]. \tag{3.4}$$

3.2 Hull-White one factor model

The above algorithm is quite general, and applies to different models. Below we present a concrete example using Hull-White one factor model. The advantage of a one-factor model is that one can easily visualize the resulting functional form of the future value. We follow the notation of [20]. The short rate can be written as $r\left(t\right) = x\left(t\right) + f\left(0,t\right)$, with the initial forward rate $f\left(0,t\right)$, and the stochastic part

$$dx\left(t\right) = \left[y\left(t\right) - \kappa x\left(t\right)\right] dt + \sigma_r\left(t\right) dW\left(t\right), \tag{3.5}$$

with $x(0) = 0$, and

$$y\left(t\right) = \int_0^t e^{-2\kappa(t-u)} \sigma_r(u)^2 du. \tag{3.6}$$

Here the mean-reversion speed κ is set to be a constant, while the volatility $\sigma_r(t)$ retains term structure.

With a single risk factor $x(t)$, the neural network is in a simple form. For example, with 2 hidden layers, it reads explicitly

$$\mathcal{F}\left(x\right) = \sum_j w_j^C \phi_B \left(\sum_i w_{ji}^B \phi_A \left(w_i^A x\right)\right), \tag{3.7}$$

at a given time T_n, with the parameters w_i^A, w_{ji}^B, w_j^C, where $i, j = 1, \cdots, 1 + \tilde{d}$, and the nonlinear scalar functions ϕ_A, ϕ_B.

We consider a sample trade of cash-settled Bermudan swaption, which can be exercised semi-annually starting from 1.5 years to 3.5 years. The underlying is a standard fixed-for-floating swap indexed to 3M Libor rate, with notional 10000, and fixed rate 0.028. For the Hull-White model, the mean-reversion parameter is chosen to be $\kappa = 0.01$, and the model is calibrated to market data on 1/18/2018. We choose a fully connected neural network with 2 hidden layers and $\tilde{d} = 10$. The EPE and ENE results are shown in Figure 3.1. ENE vanishes as the portfolio value is never negative. EPE decreases with time, displays jumps at the exercise dates, and has a convex envelope function, as the exposure on the path vanishes if the option gets exercised. We can see from the evolution of the loss function with training that the optimization procedure converges with around 500 training steps.

We further show in Figures (3.2), and (3.3) the evolution of the future values with training at two exercise dates. For comparison, the exercise values are displayed. The exercise value $U_n(x)$ does not depend on the neural network, and hence does not change with training. It is a linear function of x. The functional form of the portfolio value $V_n(x)$ evolves with training. As we started with the randomly chosen parameters $\left(V_0, Z_i^{(0)}, w_i^{(n)}\right)$, $V_n(x)$ starts out quite noisy. As training proceeds, $V_n(x)$ converges to a smooth function at about 500 training steps. The resulting function increases monotonically with x, and interpolates between the x-axis and the exercise value. Such a functional form is typical for an option price: when the interest rate is high, the underlying swap is deep in the money, the swaption will be exercised, and the swaption value will approach the exercise value; when the interest rate is low, the swap is deep out of the money, and the swaption will not be exercised, and the swaption value will be close to zero.

We can fit the resulting function to a Bachelier-type option price formula:

$$V_{Bach}(x) = A\left[(x-c)\,\Phi\left(\frac{x-c}{s}\right) + s\phi\left(\frac{x-c}{s}\right)\right], \tag{3.8}$$

where $\Phi(\cdot)$ is the standard Gaussian cumulative distribution function, and $\phi(\cdot)$ the corresponding probability density function. As $x \to \infty$, $V_{Bach}(x) \to A(x-c)$. So A basically corresponds to the slope of the exercise value, and c its intersection with x-axis. s represents an effective volatility, and larger s gives more rounded interpolation between the x-axis and the exercise value. The results are shown in Figure 3.4.

Being able to visualize the functional form of the future value can shed new light on other approaches. For AMC, a difficulty is the choice of basis functions. As the future value of Bermudan swaption is of the form of an option price, while the exercise value is essentially linear, merely using powers of the exercise value as basis functions seems not an optimal choice, and related European option prices should also be included. Furthermore, with the known functional form of the future value, one can try to replace the linear regression step in AMC by

directly fitting the inferred function, e.g. Bachelier-type formula for Bermudan swaption. Note that it is important to constrain the parameters to be in the financially meaningful regimes when carrying out the fitting, as the result is a local minimum, and not a global minimum.

4 MtM cross-currency swap

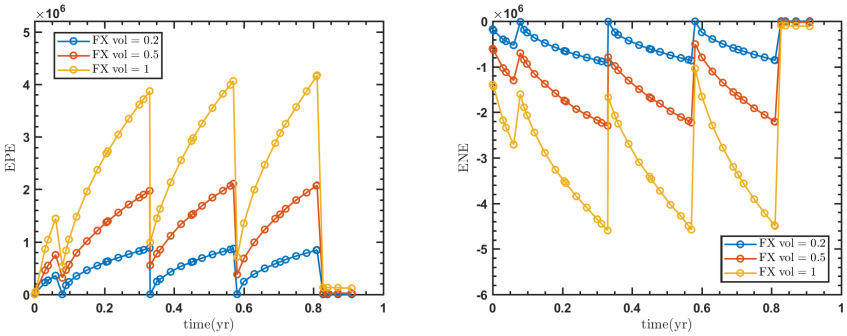

Figure 4.1: Exposure of MtM XCCY swap with $(\sigma_0, \sigma_1, \eta_1) = (0.001, 0.001, 0.2), (0.1, 0.1, 0.5), (0.2, 0.2, 1)$.

The advantage of the neural network approach lies in its easiness to generalize to high dimensions, i.e. multi-factor models. In this section, we consider as example a Mark-to-Market (MtM) cross-currency (XCCY) swap. The exposure of cross-currency swap involves risk factors from multiple asset classes, i.e. interest rate (IR) and foreign exchange (FX), hence it serves as a prototype for cross-asset CVA modeling. In addition, MtM swaps reset the notional periodically. Consequently in general their future values do not have analytic expressions, and they serve as testing ground for the neural network method.

4.1 The model

We consider the correlated dynamics of both IR and FX rates. The IR short rates are again decomposed into the initial curve part and the stochastic part: $r_i(t) = x_i(t) + f_i(0,t)$. Under the domestic risk-neutral measure, the risk factor evolution is as follows:

$$dx_0(t) = [y_0(t) - \kappa_0 x_0(t)]\,dt + \sigma_0(t)\,dW_0(t), \tag{4.1}$$

$$dx_i(t) = [y_i(t) - \kappa_i x_i(t) - \rho_{i,i+N}\sigma_i(t)\,\eta_i(t)]\,dt + \sigma_i(t)\,dW_i(t), \tag{4.2}$$

$$d\ln S(t) = \left[r_0(t) - r_i(t) - \frac{1}{2}\eta_i(t)^2\right]dt + \eta_i(t)\,dW_{i+N}(t), \tag{4.3}$$

with $< dW_i(t), dW_j(t) >= \rho_{ij}dt$, where $i, j = 0, \cdots, 2N$, and N is the number of foreign currencies. Note that the $\rho\sigma\eta$ cross term comes from change of mea-

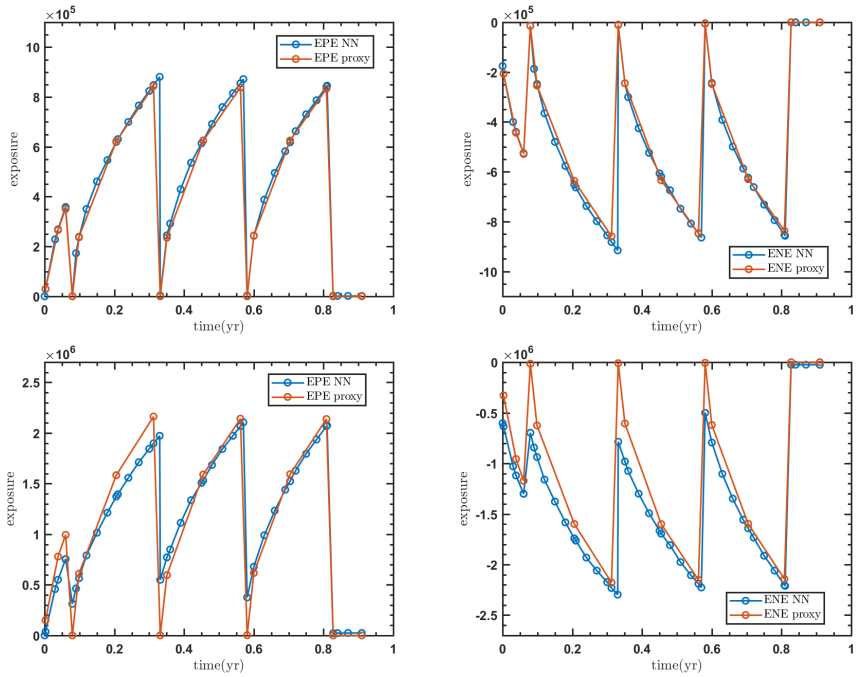

Figure 4.2: Comparison of exposure from NN method and proxy method for MtM XCCY swap for different volatilities. Up: $(\sigma_0, \sigma_1, \eta_1) = (0.001, 0.001, 0.2)$. Down: $(\sigma_0, \sigma_1, \eta_1) = (0.2, 0.2, 1)$.

sure from the foreign risk-neutral measure to the domestic risk-neutral measure. In the domestic risk-neutral measure, the numeraire is $B(t) = \exp\left[\int_0^t r_0(u)\,du\right]$.

To set the stage, we consider first a standard XCCY swap with one floating leg in domestic currency and one fixed leg in foreign currency. Standard XCCY swaps are linear products, and their future values can be determined analytically from the information available at credit time t. The future value of the fixed leg is

$$V_{fxd}(t) = N_f S(t) \sum_n K_n \tau_n P_D\left(t, T_n^P\right), \tag{4.4}$$

with notional N_f, fixed coupon rate K_n, day counting for each accrual period τ_n, the foreign exchange rate $S(t)$, and the discount factor $P_D\left(t, T_n^P\right)$ from t to future payment time T_n^P. The future value of the floating leg is

$$V_{flt}(t) = N_f S(0) \sum_n \left[\alpha_n L_n(t) + \beta_n\right] \tau_n P_D\left(t, T_n^P\right), \tag{4.5}$$

with multiplier α_n, spread β_n, and the forward rate $L_n(t)$. There are typically notional exchanges at maturity, which can be included by replacing $K_n\tau_n \to 1 + K_n\tau_n$, and $\beta_n\tau_n \to 1 + \beta_n\tau_n$ for the last period.

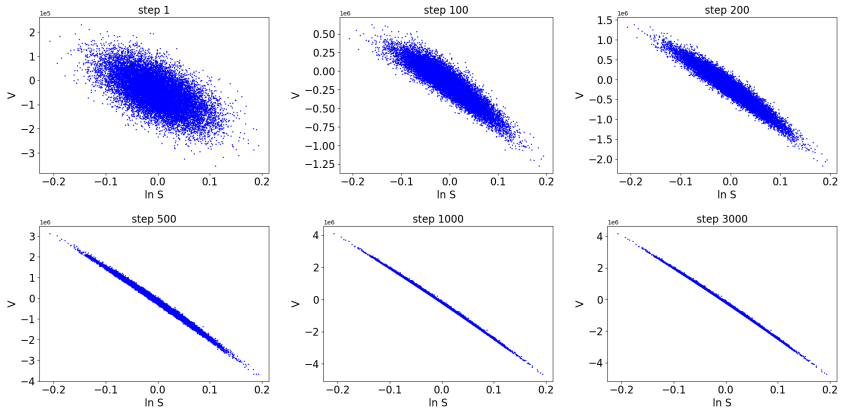

Figure 4.3: Evolution of future values of MtM XCCY swap with training. Here $(\sigma_0, \sigma_1, \eta_1) = (0.001, 0.001, 0.2)$.

The story is quite different for cross-currency swaps with MtM legs. MtM XCCY swaps reset the notional at the beginning of each accural period. The MtM leg includes two payments: (1) notional payment: $N_f \left[S\left(T_n\right) - S\left(T_{n+1}\right)\right]$, (2) rate payment: $N_f S\left(T_n\right) L_n \tau_n$, which in addition to having a floating rate L_n, effectively also has a stochastic notional $N_f S\left(T_n\right)$. Hence the future value of the MtM leg is

$$V_{mtm}\left(t\right) = E_t^Q \left[N_f \sum_n P_D\left(t, T_n^P\right) \left[S\left(T_n\right)\left(1 + L_n \tau_n\right) - S\left(T_{n+1}\right)\right] \right]. \tag{4.6}$$

One can see that the future value of MtM leg depends on the FX rates at all future reset times, i.e. $S\left(T_n\right)$, which are unknown at time t. No analytical expression for the future value of a MtM XCCY swap with correlated factors is currently known.

One can consider a proxy model. Assuming the decoupling of the IR part and the FX part, the expectation value of the FX rates then can be computed directly using the relation

$$E_t^Q \left[S\left(T_n\right)\right] = S\left(t\right) \frac{P_f\left(t, T_n\right)}{P_d\left(t, T_n\right)}, \tag{4.7}$$

with the domestic and foreign discount factors $P_d\left(t, T_n\right), P_f\left(t, T_n\right)$. Such a proxy method produces an analytic expression for the future value, but fails to capture the convexity adjustment due to the cross-asset correlation.

4.2 Neural network approach

We will attack this problem in the NN approach. Let us start by formulating the problem in the BSDE framework. Consider a XCCY swap with one MtM

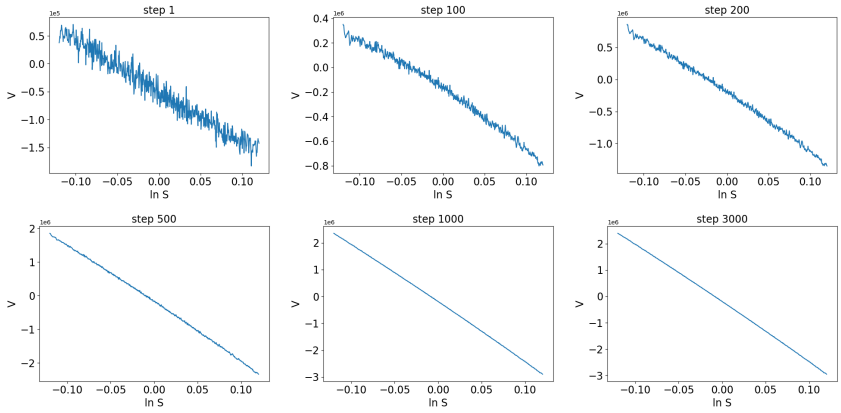

Figure 4.4: Evolution of future values of MtM XCCY swap with training (projected to $x_0 = 0$, $x_1 = 0$). Here $(\sigma_0, \sigma_1, \eta_1) = (0.001, 0.001, 0.2)$.

leg and one floating leg. The future value is $V_{XCCY}(t) = V_{mtm}(t) - V_{flt}(t)$. In the domestic risk-neutral measure, it grows with rate $r_0(t)$, and hence follows the BSDE:

$$dV_{\text{XCCY}}(t) = r_0(t) V_{\text{XCCY}}(t) dt + \sum_i \frac{\partial V_{\text{XCCY}}}{\partial X_i}(t, \mathbf{X}_t) \sigma_i(t) dW_i(t). \quad (4.8)$$

Here we group the risk factors in a vector $\mathbf{X} \equiv (x_0, x_1, \ln S)$, and the corresponding volatilities in a vector $\sigma \equiv (\sigma_0, \sigma_1, \eta_1)$. We also include the jump condition at the cashflow dates

$$V_{\text{XCCY}}(T_n^+) = V_{\text{XCCY}}(T_n^-) - CF(T_n), \quad (4.9)$$

and the boundary condition at maturity

$$V_{\text{XCCY}}(T_N^+) = 0. \quad (4.10)$$

We observe that while the cashflow $CF(T_n)$ is fully determined by information available at time T_n, the Delta's $\frac{\partial V_{\text{XCCY}}}{\partial X_i}(t, \mathbf{X}_t)$ depend on information beyond time t, in particular the future FX rates. We proceed by parameterizing the Delta's of the future value by fully connected neural networks:

$$\frac{\partial V_{\text{XCCY}}}{\partial X_i}(T_n, \mathbf{X}_n) \simeq \mathcal{F}_i^{(n)}(X_1, \cdots, X_d). \quad (4.11)$$

Below we present the algorithm for the exposure calculation in the NN approach.

First stage (pre-training):

- Forward evolve the risk factors, and store them in a tensor X_{ipn}, with dimension index i, path index p, and time index n.

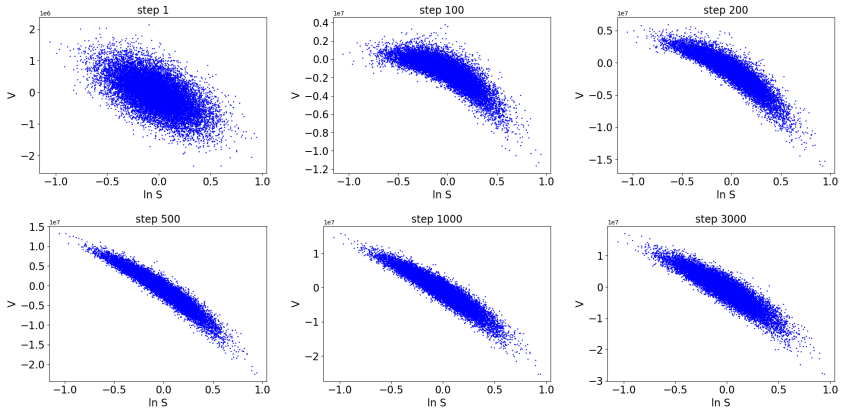

Figure 4.5: Evolution of future values of MtM XCCY swap with training. Here $(\sigma_0, \sigma_1, \eta_1) = (0.2, 0.2, 1)$.

Second stage:

- Build the neural network. At each time step, construct a neural network for the Delta's, i.e. $\frac{\partial V_{\text{XCCY}}}{\partial X_i} \equiv Z_i$.
- Forward induction. Evolve the future value according to the diffusion equation (4.8) in forward form and the jump condition (4.9) for cashflow. The results are stored in a tensor V_{pn}, with path index p, and time index n.
- Construct the loss function from V_{pn}. The loss function is of the form

$$\mathcal{L}\left(V_0, Z_i^{(0)}, w_i^{(n)}\right) = \frac{1}{\mathcal{A}} \sum_p V\left(T_N^+, \omega_p\right)^2. \tag{4.12}$$

- Train the neural network.

Third stage (post-training):

- Averaging. Compute EPE/ENE at each credit date by averaging the positive/negative part of the future values.

Since no early exercise is involved in the current problem, we use forward induction to evolve the future value. The exposure calculation is also more straightforward.

We consider a MtM XCCY swap with currency pair CAD/USD, maturity 0.83 year, quarterly paid and reset. The initial FX rate is 0.76. The Hull-White model has mean-reversion $\kappa = 0.01$ for both currencies, and flat yield curve $f_{\text{USD}}(0,t) = 0.01$, $f_{\text{CAD}}(0,t) = 0.02$. The correlation parameters are: $\rho_{\text{USD,CAD}} = 0.149$, $\rho_{\text{USD,CAD/USD}} = 0.139$, $\rho_{\text{CAD,CAD/USD}} = 0.676$. We choose

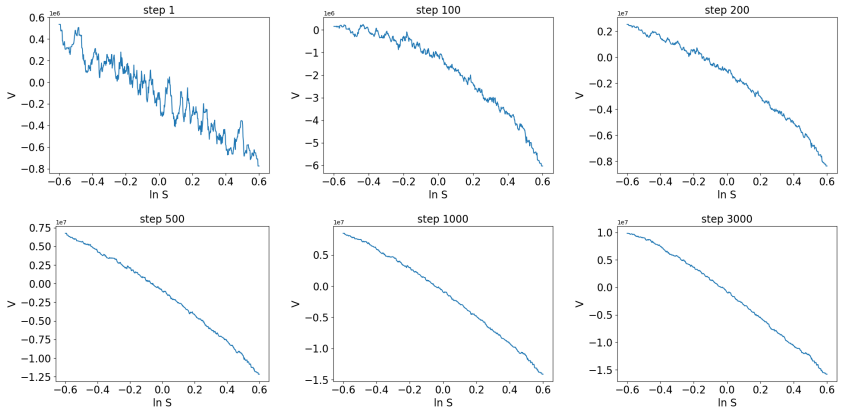

Figure 4.6: Evolution of future values of MtM XCCY swap with training (projected to $x_0 = 0$, $x_1 = 0$). Here $(\sigma_0, \sigma_1, \eta_1) = (0.2, 0.2, 1)$.

a fully connected neural network with 2 hidden layers of dimension $d + \tilde{d}$, where $d = 3$ (number of risk factors), and $\tilde{d} = 10$.

The resulting EPE and ENE for several different parameter sets of volatilities are shown in Figure 4.1. One can see that when the volatilities are small, the exposure vanishes at the MtM dates. The practice of Mark-to-Market reduces the long-term risk to short term (here three months). As the volatilities increase, the exposure on those dates increases to finite values. We further compare in Figure 4.2 the results from NN approach with those from the proxy method where cross-asset correlation is ignored. We note that proxy method always gives vanishing exposure at the MtM dates, and the NN approach properly captures the convexity adjustment missing in the proxy method. While such effects are mild in normal conditions, they can become significant in stress conditions.

We then study the functional form of future values. To have an intuitive understanding of the functional form, we try to visualize it. While it is easy to visualize the function for a single risk factor, it is more involved for multiple risk factors. As FX rate is the dominant risk factor, we study the functional relation between future value and FX rate. We present two types of plots here. The first type plots $(\ln S, V)$ for random (x_0, x_1) as obtained from the simulation paths. The second type plots $(\ln S, V)$ projected to given x_0, x_1. The projection is carried out using k-nearest neighbor regression, which is a non-parametric method without assuming any particular relationship among the different variables.

The results for two sets of volatility parameters are shown in Figures 4.3, 4.4, 4.5 and 4.6. When the IR volatilies are small (here $\sigma_0 = \sigma_1 = 0.001$, and $\eta_1 = 0.2$), the dynamics is dominated by the FX part, and the problem is essentially one dimensional. Consequently $(\ln S, V)$ pairs converge to one-dimensional lines in both plots. When both IR and FX volatilities are large (here $\sigma_0 = \sigma_1 = 0.2$, and $\eta_1 = 1$), the problem is intrinsically high dimensional.

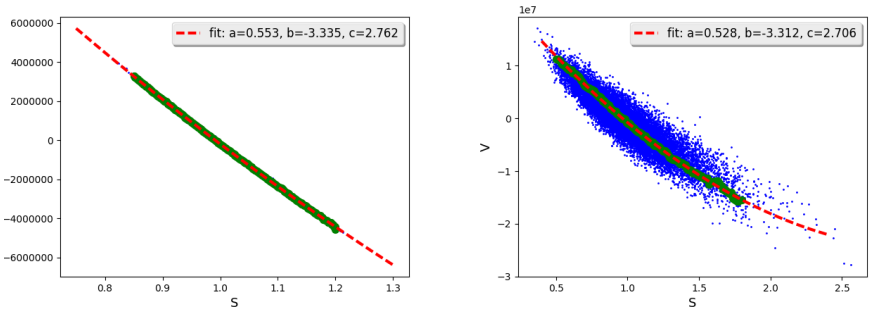

Figure 4.7: Regression of future value for MtM XCCY swap, fitting to function $V = 10^7 \left(aS^2 + bS + c\right)$. Blue dots are raw data of (S, V) with random x_0, x_1, green lines are (S, V) projected to $x_0 = 0, x_1 = 0$, red lines are fitted function of the projected data.

In the random-x plot, $(\ln S, V)$ pairs still form two dimensional areas after training. Such broadening effect (analogous to self-energy effect in physics) signals the important role played by IR rates in determining the functional form of future value. In the projected-x plot, $(\ln S, V)$ pairs converge to one-dimensional curves. These projected curves give a glimpse of the high dimensional function $V_{\text{XCCY}}\left(x_0, x_1, S\right)$.

We examine further the resulting functional dependence of future value on the FX rate S. In the proxy method, future value is linear in S. For the NN approach, to see the effect of convexity adjustment, we fit the future value to a quadratic function of S (see Figure 4.7). We consider (S, V) pairs projected to given x_0, x_1. From the fitted coefficients (in particular the ratio a/b), one can see that convexity effect clearly exists for both small and large volatilities, though the effect is less visible for small volatilities.

5 Conclusions

We have explored a new approach to model the future value and compute CVA/DVA, employing neural network as a universal approximator. The core idea is common in artificial intelligence: to convert a complicated problem to a search problem. For the present approach, it can be summarized as parameterize and optimize. The gradients of the future values are parameterized by neural network, and then efficient optimization algorithms are used to determine the involved parameters. Immediate future directions include (1) generalizing the models to cover more risk factors, e.g. using Libor-market model ([12]), (2) exploring different ways of parameterization, e.g. directly parameterizing the future values ([13]), (3) exploring different types of neural networks, e.g. convolutional neural network (CNN [21]), generative adversarial network (GAN [22]), (4) using this approach to model future greeks and compute MVA.

The idea of randomization is crucial here. In some sense, the present approach is a natural generalization of American Monte Carlo. As the NN approach is more general, and does not assume prior knowledge as in AMC, it can not compete with AMC in speed. We regard this approach as first a benchmark model for AMC. What is more interesting is to combine the two approaches to form a new strategy for exotics/XVA modeling, which takes two steps: (1) use the NN method to infer the functional form of future values for each class of products, (2) then use the learned knowledge to compute the future values in a faster method like AMC or its variant. The NN step is run infrequently, and hence requirement on its speed is not stringent. The AMC step is spared of expert input. We regard this approach as a more practical way of applying deeping learning to security pricing.

References

[1] Francis A. Longstaff, and Eduardo S. Schwartz. *Valuing American Options by Simulation: A Simple Least-Squares Approach*, The Review of Financial Studies, Volume 14, Issue 1, 1 January 2001, Pages 113–147.

[2] Weinan E, Jiequn Han, and Arnulf Jentzen. *Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations*, A. Commun. Math. Stat. (2017) 5: 349.

[3] Jiequn Han, Arnulf Jentzen, and Weinan E. *Overcoming the curse of dimensionality: Solving high-dimensional partial differential equations using deep learning*, arXiv:1707.02568.

[4] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep Learning*, MIT Press, 2016. http://www.deeplearningbook.org.

[5] Geoffrey Hinton, Li Deng, Dong Yu, George Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior et al. *Deep neural networks for acoustic modeling in speech recognition*, Signal Processing Magazine 29 (2012), 82–97.

[6] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. *Imagenet classification with deep convolutional neural networks*, Advances in Neural Information Processing Systems 25 (2012), 1097–1105.

[7] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. *Deep learning*, Nature 521 (2015), 436–444.

[8] George Cybenko. *Approximation by superpositions of a sigmoidal function*, Mathematics of control, signals and systems 2, no. 4 (1989): 303-314.

[9] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. *Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks*, Neural networks 3, no. 5 (1990): 551-560.

[10] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. *Learning representations by back-propagating errors*, Nature (1986), 323, 533-536.

[11] Pierre Henry-Labordere. *Deep Primal-Dual Algorithm for BSDEs: Applications of Machine Learning to CVA and IM*, (November 15, 2017). Available at SSRN: https://ssrn.com/abstract=3071506 or http://dx.doi.org/10.2139/ssrn.3071506

[12] Haojie Wang, Han Chen, Agus Sudjianto, Richard Liu, and Qi Shen. *Deep Learning-Based BSDE Solver for Libor Market Model with Application to Bermudan Swaption Pricing and Hedging*, arXiv:1807.06622.

[13] Maziar Raissi. *Forward-Backward Stochastic Neural Networks: Deep Learning of High-dimensional Partial Differential Equations*, arXiv:1804.07010.

[14] Damiano Brigo, Massimo Morini, and Andrea Pallavicini *Counterparty credit risk, collateral and funding: with pricing cases for all asset classes*, John Wiley & Sons, Mar 5, 2013.

[15] Jon Gregory. *The xVA Challenge: Counterparty Credit Risk, Funding, Collateral and Capital*, John Wiley & Sons, Oct 26, 2015.

[16] N. El Karoui, S. G. Peng and M. C. Quenez. *Backward Stochastic Differential Equations in Finance*, Mathematical Finance, Vol. 7, No. 1 (January 1997), 1-71.

[17] E. Pardoux and S. G. Peng. *Adapted solution of a backward stochastic differential equation*, Systems & Control Letters 14 (1990) 55-61.

[18] E. Pardoux and S. G. Peng. *Backward stochastic differential equations and quasilinear parabolic partial differential equations*, Stochastic Partial Differential Equations and Their Applications pp 200-217.

[19] Diederik P. Kingma, and Jimmy Ba. *Adam: A method for stochastic optimization*, arXiv:1412.6980. 2014 Dec 22.

[20] Leif B. G. Andersen and Vladimir V. Piterbarg. *Interest Rate Modeling. Volume 2: Term Structure Models*, Atlantic Financial Press, 2010.

[21] Yann LeCun, Bernhard Boser, John S. Denker, Donnie Henderson, Richard E. Howard, Wayne Hubbard, and Lawrence D. Jackel. *Backpropagation applied to handwritten zip code recognition*, Neural computation 1.4 (1989), 541-551.

[22] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. *Generative adversarial nets*. In Advances in neural information processing systems 2014 (pp. 2672-2680).