TUM Data Innovation Lab

Munich Data Science Institute (MDSI)

Technical University of Munich

&

KPMG AG Wirtschaftsprüfungsgesellschaft

Final report of project:

Project: Deep xVA - speeding-up derivative pricing

Authors	Tim Emmert, Tobias Lausser, Xiaolan Liu, Leonie Wagner
Mentor(s)	Yannick Limmer (M.Sc.)
Project Lead	Dr. Ricardo Acevedo Cabra (MDSI)
Supervisor	Prof. Dr. Massimo Fornasier (MDSI)

Aug 2022

Abstract

This project focuses on implementing and evaluating novel machine learning approaches to speed up the computation of (incremental) xVAs, in particular focusing on CVA, DVA and FVA.

Valuation Adjustments (xVAs) are adjustments made to the value of a derivative portfolio obtained from risk-neutral pricing, that account for costs and benefits that are not captured by pricing under the risk neutral measure. There are many xVAs considering different effects that might influence the price of the portfolio, with the most important ones being Credit Valuation Adjustment (accounting for counterparty credit risk), Debit Valuation Adjustment (accounting for the own credit risk of the entity holding the portfolio) and Funding Valuation Adjustment (accounting for the funding costs arising in context with the portfolio).

Doing those adjustments is essential to every player in the over-the-counter derivatives market to be able to properly reflect the value and risks of the derivatives held. This is necessary for accounting purposes and portfolio and risk management under regulatory requirements.

At the moment, compute-expensive nested Monte Carlo simulations are used to calculate xVAs. Due to those high requirements on the computation time, they can often not be computed intraday, and banks have to fall back to inaccurate approximate solutions. This project proposes a machine learning-based approach to speed up the computation of xVAs, namely Credit Valuation Adjustment, Debit Valuation Adjustment and Funding Valuation Adjustment, for European basket and Bermudan options.

It was found, that a technique called differential machine learning can be used to more accurately compute xVAs, outperforming standard neural network and kNN-based approaches, i.e., provided the same amount of training instances, differential machine learning outperforms the other data-driven approaches by up to 60%. The use of differential machine learning also enables accurate xVA computations in a fraction of the time needed using Monte Carlo simulations. More precisely, in the scenarios evaluated here, differential machine learning performs inference 57 times faster than a Monte Carlo simulation of comparable performance.

Contents

1 Introduction

"In my view, derivatives are financial weapons of mass destruction, carrying dangers that, while now latent, are potentially lethal", is a quote by Warren Buffett, CEO of Berkshire Hathaway and highly recognized investor. Unfortunately, he has been proven right as the financial markets crashed in 2008/09, resulting in the most severe financial crises since the "Great Depression" in 1929. In posterior, it has been determined that derivatives played a substantial role in causing this crisis, as it is for instance elaborated in [3].
As a reaction, authorities implemented numerous new regulations on the derivatives market, which forced banks to adjust their pricing and risk modelling accordingly to prevent another crash. These valuation adjustments (xVA) inter alia include credit valuation adjustments (CVA), debit valuation adjustment (DVA), and funding valuation adjustments (FVA); and constitute a major challenge for the affiliated institutions. This is particularly due to the fact that it requires tedious computations of high computational complexity involving every single derivative position of the institution, what therefore amounts to tremendously intense computational effort.
The objective of this project is to reduce the computational complexity of a xVA computation by incorporating advancements in the field of deep learning. Currently, accurate xVA computations rely on compute-heavy Monte Carlo simulations, which cannot be performed on a daily basis and, thus, require corporations to rely solely on extremely rough estimations. Since this is an obvious source of additional risk, a speed-up would result in a significant advantage for market participants when it comes to valuation of portfolios and risk management. Hopefully, improvements in this matter will cause even critics – such as Warren Buffett – to re-evaluate their opinion on the derivative market.
In this project, the use of a technique called differential machine learning enables accurate xVA computations in a fraction of the time needed using Monte Carlo simulations. More precisely, in the scenarios evaluated here, differential machine learning performs inference 57 times faster than a Monte Carlo simulation of comparable accuracy computing xVAs.
The remainder of this report is structured as follows. First, we will provide basic definitions and motivate xVAs and computational methodologies; and we will discuss possible bottlenecks of the prevailing approaches. This is accompanied by an introduction to the relevant financial products, and we derive properties of their expected exposures. Second, we outline the machine learning tools involved, as for instance neural networks and differential machine learning. Moreover, we explain the concept of Bayesian optimization that is used for hyperparameter optimization. To this end, we discuss the implementation in detail, with a focus on the data generation methods, the model building process and training of the model. Eventually, we provide numeric results for our approach and benchmark it with conventional techniques.

2 Financial background

2.1 Valuation Adjustments

Derivatives can be traded between two market participants in two different ways: Either via an intermediary, a so-called exchange, that provides credit security, liquidity and efficient price discovery, or directly between each other. The latter trades are called

over-the-counter (OTC), they are typically less standardized and in contrast to exchange traded derivatives the parties in a trade are not necessarily required to post so-called *collateral* for counterparty credit risk mitigation. Counterparty credit risk of a trade is the risk that the counterparty can't fulfil all payment obligations that come along with the financial contract. Exchanges deal with that risk by usually requiring both parties of a derivative trade to post securities in so-called margin accounts with the exchange, which can be used to settle the outstanding obligations in case of a default.
Such mechanisms are not standard in OTC trades, even though the market got quite heavily regulated as a reaction to the financial crisis 2008/09. The counterparty credit risk is therefore taken directly by each party with respect to the other party in the trade and since many players don't have a strong credit quality, nor are they able to post collateral to reduce the counterparty risk, this risk is an unavoidable consequence of OTC markets. Notably, few players dominate the OTC market: generally these are large and highly interconnected, and are generally viewed as being "too big to fail". [6] To understand why this is a big risk for the overall financial system one simply has to look at the sheer size of the OTC market: The outstanding nominals in the market in 2013 were \$693 trillion while exchange traded derivatives were at \$70 trillion. [13]

OTC markets were considered to be catalysts within the financial crisis in 2008/09 which is why authorities were focussed on regulating those markets as a reaction to that. Even though financial institutions were already aware of counterparty credit risk before the crisis and few investment banks already had a very basic form of CVA implemented from 2007/08 on [9] those new regulations and experiences forced the market participants to react to them and can be understood as the starting point of xVA considerations.
In particular, two major changes are relevant for understanding xVA [8]:
Firstly, during the crisis, concerns about banks' creditworthiness led to an almost complete breakdown of the interbank funding market. After the crisis interbank lending rates were more volatile and traded at increased spreads reflecting the corrected market view on bank credit risk, increasing the funding costs for banks activities.
Secondly, the crisis showed that many financial institutions, even the ones "too big to fail", used to expose themselves to excessive risks they weren't capitalized for. To prevent this, the new regularization introduced a central clearing mandate (similar to the procedures on exchanges), additional capital requirements and bilateral posting of collateral aiming to counterparty risk mitigation and control.

The impact this had was significant, forced market participants to incorporate those changes in pricing [6] and required their trading desks to manage their credit and funding exposures more actively [9]. In particular, xVAs attempt to reflect the associated costs that have become more significant through changes adequately [8], e.g. CVA directly aims at increased counterparty risk capital requirements [6]. Those adjustments are then added or distracted from the value of the portfolio obtained from risk-neutral pricing. There exist many xVA considerations, most of them are heavily linked to each other, but not all of them are common to the same extent in the industry. In Figure 1 we illustrate some of them but in the project and the following we focus on the three most common xVAs, namely CVA, DVA and FVA.

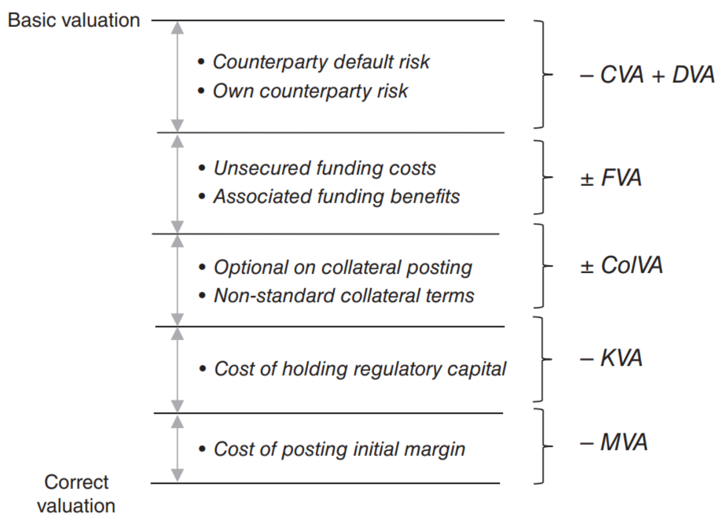

Basic valuation
• Counterparty default risk
• Own counterparty risk
– CVA + DVA
• Unsecured funding costs
• Associated funding benefits
± FVA
• Optional on collateral posting
• Non-standard collateral terms
± ColVA
• Cost of holding regulatory capital
– KVA
• Cost of posting initial margin
– MVA
Correct valuation

Figure 1: Illustration of the most common xVAs. A negative sign indicates that the adjustment is considered a cost and decreases the value of a portfolio, a positive sign indicates a benefit. [6]

2.1.1 Credit Valuation Adjustment

Credit Valuation Adjustment (CVA) is the most widely known Valuation Adjustment, it is a key topic for market participants because of the volatility of credit spreads and the associated accounting and capital requirements by Basel III. It is therefore material for every significant OTC derivative user and shouldn't be ignored [6].
Economically, it represents a compensation for the counterparty credit risk taken by a party in the trade and can therefore be interpreted as the market price of counterparty risk. It is by definition the difference between the risk-free portfolio value (assuming all future payments are made) and the true portfolio value that takes into account the possibility of a counterparty's default, i.e. strictly speaking it is a cost diminishing the value of a portfolio in comparison to the risk-free portfolio value.

Since it quantifies credit risk it inherits that risk's challenges: It varies substantially with the counterparty and the transaction and in contrast to the risk faced in loans the amount at risk is usually at uncertainty and both parties carry the risk. This bilateral nature is it that makes quantification of counterparty risk particularly difficult. [6]
Additionally, when considering the amount at risk, one has to take into account the different methods of credit risk mitigation. These are usually clarified in a so-called Credit Support Annex (CSA) agreement between the two parties, and in particular consist of netting and collateralization. For the sake of simplicity and because this can be very individual, it is not taken into account in the further considerations.

A quite simplifying (assuming independence of the probability of default and the exposure and looking at the CVA isolated from other xVAs) but well known formula for CVA is given by [13]

$$CVA = (1 - R_{cpty}) \sum_{k=1}^{N} EPE(t_k) PD_{cpty}(t_{k-1}, t_k) \tag{1}$$

where $EPE(t)$ is the so-called *expected positive exposure* of the portfolio at time point t, $PD_{cpty}(t, t+s)$ is the marginal of the default probability of the counterparty between times t and $t+s$ and R_{cpty} the so-called *recovery rate* of the counterparty. All terms will be explained in more detail below.
In this example one can already see the general structure of computation of most of the xVA's: There is one term that captures the magnitude of the mark-to-market of the considered product over time, in this case the expected positive exposure, and some other terms assigning some cost to the mark-to-market value of the portfolio, here recovery rate and the probability of default.
The expected exposure simulation is the computationally expensive part of the valuation and since we want to focus on its computation later we will describe it detailed in section 2.2.

The cost components in the CVA computation directly depend on the counterpart the trade is done with. R_{cpty} is the rate of recovery and describes which percentage of the owed amount is expected to be covered in case of default of the counterparty. The term $(1 - R)$ can therefore be interpreted as the loss given default, i.e. the percentage amount of the owed amount to be lost if the counterparty defaults. These amounts obviously depend on seniority of the claim, which is usually pari passu with senior unsecured debt such as credit default swap (CDS) contracts. [6] In practice, it is often taken as constant with $R = 0.4$, which is also adapted by the implementations in this project. [13]

The evaluation of the probability of default term requires an estimation of the marginal default probability within the time interval $[t_{i-1}, t_i]$. The benefit of the default entering indirectly via the probability of default only is that within the sampling framework it is not necessary to simulate (rare) default events, which ultimately saves computational time. [6]
To compute the probability of default, one usually uses risk neutral default probabilities (which are derived from market data like CDS) instead of real-world default probabilities (derived from historical default data); this is underlined also by accounting standards and Basel III. [6] Several market observables could be used to define the probability of default, calculate from and calibrate to, but the CDS market is the most obvious clean and directly available quote. Nevertheless, it is often problematic because of illiquid or non-existent CDS markets. In that case, a hybrid solution of blending historic and risk-neutral probabilities is used.

Assuming the CDS spread is observable in the market, the computation in the project is as follows: [7]
Making the market standard model assumption that the credit event is the first event of a Poisson counting process τ which occurs at some time t with probability $\mathbb{P}(\tau < t + dt | \tau \geq t) = \lambda(t)dt$ with λ a deterministic function called the hazard rate, then the probability to

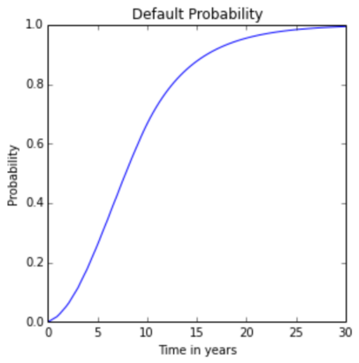


Figure 2: The implemented generic default probability over time. [4]

default in the interval $[t, T]$ can be computed as

$$PD(t,T) = 1 - \exp(-\int_t^T \lambda(s)ds) \tag{2}$$

This means we only need to determine $\lambda(s)$. That can be done by assuming it to be constant between time points where the CDS is quoted, observing the credit spread of the CDS of the regarding entity for that time interval and then sequentially for different time points solving for the hazard rate that equates both legs of the CDS. This methodology is called bootstrapping.
Having found $\lambda(t)$ we can then compute

$$PD(t_{i-1}, t_i) = \exp(-\int_{t_{i-1}}^T \lambda(s)ds) - \exp(-\int_{t_i}^T \lambda(s)ds) \tag{3}$$

Since we are not focusing on optimizing the bootstrapping method in the course of our project and since no market data was given we simply assume a constant hazard rate of $\lambda(t) \equiv 0.02$. Given some counterparty market data, we could implement the probability of default either as described above if a liquid CDS for the entity would be quoted for enough maturity points such that the term structure of the credit spreads is observable. Else we could use some different source of credit spread information such as other direct observables or single name proxies.

2.1.2 Debit Valuation Adjustment

"[DVA is] a counter-intuitive but powerful accounting effect that means banks book a paper profit when their own credit quality declines". This is how the *efinancialnews* criticized usage of the Debit Valuation Adjustment, on October 26th 2011, in the article "Papering over the great Wall St Massacre". This gives an idea of how controversial the DVA is in the industry. [6]

To understand this better, we first have a look at how the DVA is actually defined. It is basically CVA from the counterparty's perspective: So far, when considering CVA, we made the assumption that the party doing the computation cannot default at all. This is now taken into account by computing the DVA. The idea is that if the party doing the calculations defaults, it may make a "gain" due to not entirely paid obligations in case it owes money to its counterparty when taking the mark-to-market at the time of default. So it is strictly speaking a benefit increasing the value of a portfolio by taking into account the possibility of an own default.

There are a lot of pro's and contra's to such considerations when valuating an own portfolio. The situation is clear from an accounting point of view as they actually require taking into account the own default when valuating the own liabilities and therefore also require the DVA to be taken into account. This is why it is necessary for market participants to have an efficient machinery to compute the DVA.

It is in a simplifying setting defined as

$$DVA = (1 - R_{own}) \sum_{k=1}^{N} ENE(t_k) PD_{own}(t_{k-1}, t_k) \tag{4}$$

where the ENE is the so-called expected negative exposure, which can be understood as the expected positive exposure from the counterparty's point of view. Just as the expected exposure, its simulation is explained in detail down below. The other notation and computation is just as in the CVA case above, with the subtle difference that here the recovery rate and probability of default of the party doing the computations is considered.

Looking at the CVA and DVA in two separate computations is again a usual but oversimplifying thing to do: When computing the CVA we are incorrectly not conditioning on the own survival until the counterparty's default, there is no consideration of default correlation and netting and close out assumptions are still ignored while being more crucial since the party doing the computations is not risk-free anymore (it could default itself on the default of the counterparty).

2.1.3 Funding Valuation Adjustment

Funding valuation adjustment (FVA) is quantifying the funding costs or benefits of uncollateralised derivatives compared to the risk-free rate. It represents the costs and benefits of hedging an uncollateralised trade with a collateralised one in the interbank market.
Before the financial crisis, the funding costs were close to the risk-free rate. Thus, the funding cost were not considered by financial institutions. [6] However, after the financial crisis the funding becomes increasingly costly, and the interest paid on collateral no longer offsets the increased cost. The evaluation and controlling of funding costs has therefore become a critical part of the risk management strategy of financial institutions.

FVA can be expressed as the difference between the funding cost and funding benefit. The formula for FVA can be divided into two parts: the funding cost adjustment (FCA), arising when the derivatives business requires funding, and the funding benefit adjustment (FBA), arising when the derivatives business generates funding[14]. The formula is as follows:

$$FVA = \sum_{i=1}^{N} EPE\left(t_k\right) FS_B\left(t_k\right) \Delta_{t_{k-1},t_k} - \sum_{i=1}^{N} ENE\left(t_k\right) FS_L\left(t_k\right) \Delta_{t_{k-1},t_k} \tag{5}$$

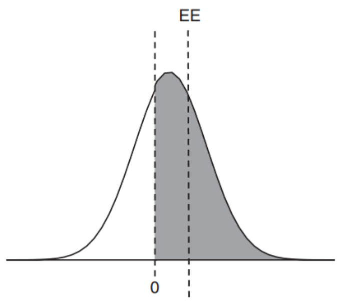


Figure 3: Illustration of the computation of the expected positive exposure at a certain time point t given the distribution of the value of the portfolio at that time point. Note that only the gray area is considered. [6]

where Δ_{t_{k-1},t_k} is the time step for accruing. EPE and ENE are the expected positive exposures and expected negative exposure respectively. $FS_B(t)$ is the funding spread of borrowing money at time point t, $FS_L(t)$ is funding spread when lending money at time point t. All terms will be explained in more detail below.

Determination of funding spread is quite subjective as in the credit curve case.[6] In practice, it is prevalent to construct the funding spread over a reference curve, which is funding forward - reference curve. For simplicity, we assumed in the course of the project that the spreads are constant over time.

2.2 Expected exposures & their simulation via nested Monte Carlo simulations

As already mentioned, the project focused on the simulation of exposures in the speed-up process, since they are considered the bottleneck of the xVA computations we focused on. As we will see, computing the expected exposure (this is in the following used as a generic term for expected positive and negative exposure) is far more sophisticated than pricing the underlying product. [6]
The expected positive exposure of a portfolio is given by

$$EPE(t) = E[max(V(t), 0)] \tag{6}$$

where $V(t)$ is the value of the portfolio at time point $t \in \mathbb{R}$. Note that this portfolio value is a random variable. The illustration 3 shows the schematic computation of the EPE given the distribution of the random variable.

The reason why this is the quantity under risk to be considered when computing the CVA is that assuming the party doing the computation owes money to a defaulted counterparty after netting and close out transactions, then they are still obliged to settle that debt. This default doesn't change anything about the cash flows connected to the portfolio since after the settlement with the defaulted counterparty one can simply set up the same portfolio with another counterparty, resulting in the same mark-to-market as before. On the other hand, if there is a positive exposure at the time of default of the counterparty, one is left with a claim on the remaining amount. It is not reasonable to assume that this claim will be completely settled, but only to some extent.

Hence, the party doing the computation will only lose money if they are owed money at counterparty's default and therefore the expected positive exposure is considered for CVA calculations.

The expected negative exposure is given by

$$ENE(t) = E[max(-V(t), 0)] \tag{7}$$

where the notation is the same as above. This reflects, similar as above, the fact that the counterparty will only lose money if they are owed something at the time of the default of the party doing the computations. Therefore, the expected negative exposure is the quantity to be considered for DVA computations.

Since in an actual case of default there will always be the chance of dispute over amounts and in general the netting and collateral computations are individually set in the CSA agreement, we will neglect those subtleties and will only focus on the mark-to-market value of the portfolio under consideration.

The crucial question is now how to simulate the expected exposure to be able to use it for the xVA computations afterwards. Since we tried to work with a very general simulation framework to later allow for extensions of our results (to portfolio level, netting and collateralisation or path-dependence) we used a Monte Carlo approach to do this. This is also the industry standard across all counterparties and products.

The methodology can be split up into 4 steps: [6]

1. Define all relevant risk factors and their corresponding models.

   Both should be realistic and parsimonious since we need many Monte Carlo simulations for the training data generation later, and we want to reuse the model to some extent. Simplicity also allows involving co-dependencies (within risk-factors but in extensions later also for netting in portfolio effects). In our case we have as risk factors generally the prices of the underlyings which are modeled as geometric Brownian motions. Note that the simulation has to be done in the risk-neutral framework (i.e. the parameters are market-implied and justified by hedging and arbitrage considerations) since we are aiming at a pricing application in the xVA computation.

2. Generate scenarios along the time grid of the simulation, i.e. produce joint realizations of the different risk factors.

   Therefore, first choose a grid of time points for the simulations which is reasonably large to capture the main details of the exposure, but that isn't too large from a computational point of view. It should be spanning from the current time point to the last payment date within the simulated portfolio. It is important to make sure to include critical points (e.g. where for example payoffs happen) into the grid to avoid missing hotspots or huge discontinuities. Note that grids can be non time-homogeneous.

   A crucial question that arises is if the simulation should happen pathwise or in a direct way, as illustrated in Figure 4. We use both approaches at different points in the project, as explained later.

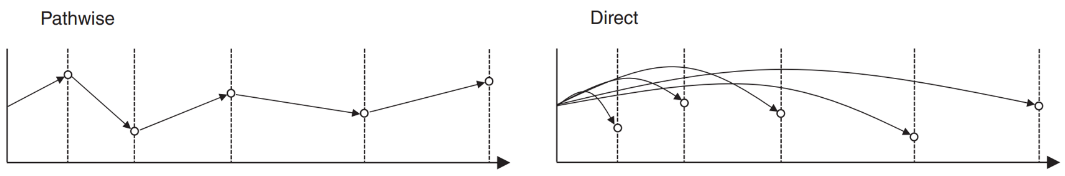


Figure 4: Path-wise simulation where we always simulate from one time point to the next one vs. direct simulation where we jump from the current time point to the ones we want to simulate. [6]

3. Revaluation of the portfolio on the time grid.

   Given a certain realization of the risk factors at a given point on the time grid, one has to evaluate the portfolio in this market scenario. If no closed valuation formula is given, this has to be done by a (nested inside the scenario generation) Monte Carlo simulation. This in combination with the scenario generation is clearly a bottleneck of the xVA computation. As an illustration of this fact, look at the case where we want to simulate for a single counterparty a portfolio consisting of 20 transactions at 100 simulation time steps with 10,000 scenarios. This already results in 20 million evaluations of financial products!

4. Aggregation of the simulation results.

   In order to compute the CVA we have to aggregate the evaluations on netting set levels (i.e. over the different products priced at the different time point in multiple simulation), apply then the given risk mitigations (e.g. possibly pathwise collaterals) and are then able to report the distribution of the portfolio exposure by averaging over all simulations for a certain point in time. Applying the formulas above allows us then to compute the expected positive and negative exposure.

2.3 Financial products & their expected exposure

In order to simplify all considerations we always assume that the risk-free rate is constantly 0 and hence discounting can be neglected.

2.3.1 European call options

A *(European) call option* gives a buyer the right to buy the underlying asset by a certain date, called *maturity*, for a certain price, called *strike price*. It can only be exercised at maturity and is traded on exchanges or OTC.
The payoff of a call option at maturity is given by

$$C(T) = \max\{S(T) - K; 0\} \tag{8}$$

where T is the maturity, $S(t)$ is the price of the underlying at the time point $t \in \mathbb{R}$ and K is the strike price.

Following the principle of risk-neutral valuation, the value of a call option at the time point $0 \leq t \leq T$ is given by

$$V_C(t) = \mathbb{E}_{\mathbb{Q}}\left[C(T) \mid \mathcal{F}_t\right] \tag{9}$$

where Q is an equivalent martingale measure with respect to the physical measure $\mathbb{P}$ and $(\mathcal{F}_t)_{t\geq 0}$ is the filtration in the model. In particular, the value of a call option is by monotonicity of the conditional expectation always non-negative. This formulation will be used later to price the call options by Monte Carlo simulation.
Finally, computing the expected positive exposure for a call option gives

$$EPE(t) = \mathbb{E}_{\mathbb{Q}}[\max\{V_C(t); 0\}] = \mathbb{E}_{\mathbb{Q}}[\mathbb{E}_{\mathbb{Q}}[C(T)|\mathcal{F}_t]] = \mathbb{E}_{\mathbb{Q}}[C(T)] = V_C(0) \tag{10}$$

where the second equality holds because the value of a call option is always non-negative, and the third by the tower property of conditional expectations. In particular, the expected positive exposure of a call option is constant and equal to its spot price.

2.3.2 Basket options

A *(European) basket option* is an option whose payoff at maturity depends can depend on the value of more than one asset. The payoff of the considered basket option is given by:

$$D(T) = \max\{\min_{i\in[d]}\{S_i(T)\} - K; 0\} \tag{11}$$

Here K is again a predetermined strike price and $(S_i(T))_{t\geq 0}$ is the price of asset $i = 1, ..., d$ at maturity T. Note that the prices of the underlyings $(S_i)_{i\in[d]}$ are usually stochastically dependent.
By exactly the same arguments as above and using again the same notation the value of a basket option at time $t \leq T$ is given by $V_D(t) = \mathbb{E}_{\mathbb{Q}}[D(T)|\mathcal{F}_t]$ which is non-negative, which further implies as before that the expected positive exposure of a basket option is constant and equal to its initial value.

2.3.3 Bermudan options

Bermudan call options follow the same concept as their European counterpart with the only difference that they cannot only be exercised at maturity but instead have a finite set of time points $0 \leq t_1 \leq ... \leq t_n = T$ where exercise is possible. However, note that exercise is only possible once.
The hypothetical payoff at time point t_i is then given by

$$C(t_i) = \max\{S(t_i) - K; 0\} \tag{12}$$

Pricing a Bermudan call option is particularly more difficult than pricing a European call option, where it is a priori clear that a potential exercise can only happen at maturity. Analytically, one would consider a procedure known as *backwards induction.* The starting point is that the value $V_C(t_n)$ at exercise time $t_n = T$ is equal to the payoff $C(t_n)$ if the option wasn't exercised before already. Looking at an arbitrary exercise time t_{i-1} the question arises if exercising the option at this point in time (this is called an early exercise) would make sense if the option wasn't exercised yet. An intuitive answer to this is, "yes, if the payoff of an early exercise exceeds the expectation for the value of the option at the next time point given all information up to the current time point". This consideration leads to the value of the option at exercise time t_{n-1} given by

$$V_C(t_{i-1}) = \max\{C(t_{i-1}); \mathbb{E}_{\mathbb{Q}}[V_C(t_i)|\mathcal{F}_{t_{i-1}}]\} \tag{13}$$

where $\mathcal{F}_{t_{i-1}}$ is the sigma algebra in the filtration that formalizes the idea of the information collected up until time point t_{i-1}. Doing this backwards in time one can find the value of the option at all potential payoff dates inductively.
This methodology can be utilized to construct a Monte Carlo approach for pricing and later computing the expected positive exposure of a Bermudan call option. W.l.o.g. we set $C(t) \equiv 0$ if $t \notin \{t_1, ..., t_N\}$ to simplify the notation. In particular, the recursive formula above holds then for every $t \leq T$ and the price of a Bermudan call option at time point $t \in [t_i, t_{i+1})$ can then be computed as

$$\begin{aligned} V_C(t) &= \max\{C(t); \mathbb{E}_{\mathbb{Q}}[V_C(t_{i+1})|\mathcal{F}_t]\} && (14)\\ &= \max\{C(t); \mathbb{E}_{\mathbb{Q}}[\max\{C(t_{i+1}); \mathbb{E}_{\mathbb{Q}}[V_C(t_{i+2})|\mathcal{F}_{t_{i+1}}]\}|\mathcal{F}_t]\} && (15)\\ &= \max\{C(t); \max\{\mathbb{E}_{\mathbb{Q}}[C(t_{i+1})|\mathcal{F}_t]; \mathbb{E}_{\mathbb{Q}}[V_C(t_{i+2})|\mathcal{F}_t]\}\} && (16)\\ &= \max_{s\in\{t_i:t_i\geq t\}\cup\{t\}} \{\mathbb{E}_{\mathbb{Q}}[C(s)|\mathcal{F}_t]\} && (17)\end{aligned}$$

where the third equality follows by the tower property. Phrased differently, the value of the Bermudian call option is the maximum of the expected payoffs at all payoff dates from the presence to maturity, given all information up to the presence.
The value of a Bermudian call option is obviously non-negative, since all future payoffs are non-negative.

2.3.4 Forwards

A *forward contract* is an agreement to sell or buy an underlying asset at a certain time in the future, again called maturity, for a certain price, called the *forward price.* One party takes the so-called *long position* in the trade, which corresponds to agreeing to buy the asset at maturity, while the other party then takes the *short position* and agrees to sell the underlying. Forward contracts are in contrast to future contracts, usually OTC traded and less standardized.
In the course of the project we always assumed to take the long position which corresponds to a payoff of the derivative given by

$$F(T) = S(T) - K(0, T) \tag{18}$$

where $S(T)$ is the price of the underlying at maturity T and K is the forward price of the forward contract with maturity T initiated at time point 0. This forward price is usually set in a way such that the initial value of a forward at time point 0 is equal to 0. In the case of a 0 risk-free rate, the forward price is simply given by the spot price of the underlying at the starting point of the contract, i.e. $K(0, T) = S(0)$.
The payoff structure leads to the fact that it is possible to have a non-zero expected positive exposure as well as a non-zero expected negative exposure when trading forwards. In that way, forwards extend the set of different financial contracts.

2.3.5 Interest Rate Swaps

An interest rate swap is a contract between two parties to exchange one stream of interest payments for another over a specified period of time. Swaps are derivative contracts and are traded over the counter.

Interest swaps are the exchange of a fixed interest rate payments for floating rate payments and the swap price is given by the difference between the expected floating leg and fixed leg cash flows[11]:

$$s\left(t,X_t\right)=\sum_{k=n+1}^{N}\Delta_k E_t^{Q_k^T}\left[L\left(T_{k-1},T_k\right)\right]P\left(t,T_k\right)-R\Delta t_{k+1},t_kP\left(t,T_k\right) \tag{19}$$

Here $T_1,..,T_N$ denote the settlement dates of the swap, L is the floating rate, R is fixed rate. $\Delta_k=T_{k+1}-T_k$, $P\left(t,T_k\right)$ is the discount rate for both fixed leg and floating leg. We can get the discount curve from Option-Adjusted Spread (OSA) and the float rate from Secured Overnight Financing Rate (SOFR).
A usual assumption is that the interest rate follow the Vasicek Model. In the Vasicek Model, the short rate follows the stochastic differential equation

$$dr\left(t\right)=k\left(\theta-r\left(t\right)\right)dt+\sigma dW\left(t\right) \tag{20}$$

where k and σ are considered piecewise constant, and W is a Brownian motion under the risk-neutral measure.
The expected exposure can be calculated as described in section 2.2. A usual plot of simulations of the exposure over time for a payer swap can be seen in the graphic below.

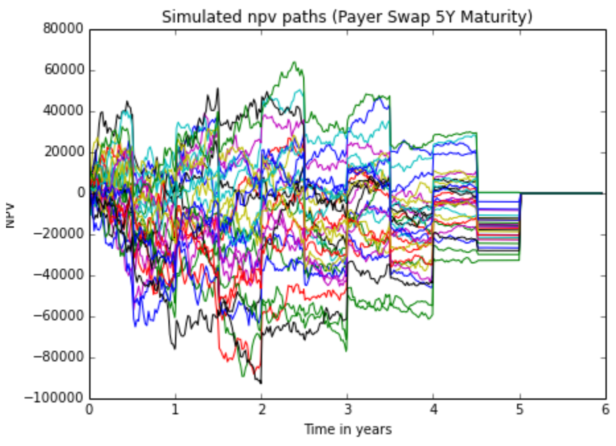


Figure 5: expected exposure of Interest rate swap

3 Technical background

3.1 General Approach

The goal of the methodologies explained in the following is to use Neural Networks to accurately being able to approximate the result of xVA computations. For that, Monte

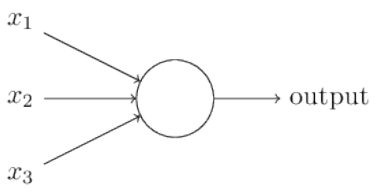


Figure 6: Single perceptron with three input nodes

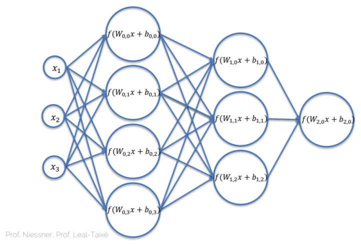


Figure 7: Network with one input (3 neurons), two hidden (4 and 3 neurons) and one output layer

Carlo simulations are used to compute noisy estimates of expected exposures over time given volatilities and spot prices of financial products. Machine learning models are then trained in order to be able to estimate those exposures accurately. Then, the performance of the model is evaluated on test data comprised of results of Monte Carlo simulations averaging over many paths, yielding compute-intensive but accurate benchmarking results for the models.
Note, that the machine learning models output exposure estimates. This was chosen as an intermediate step to provide further explainability to the approach. In order to compute xVAs, these exposures still need to be fed into easy-to-compute formulas introduced in subsection 2.1.

3.2 Neural Networks

Neural Networks have attracted considerable attention in the last years. Additionally, it is a rapidly expanding field, with new applications found every day. In this chapter, the underlying fundamentals and functioning will be explained.
Neural Networks, also called Artificial Neural Networks (ANN), are a type of machine learning algorithm that is inspired by the human brain. Neural networks are made up of nodes, which can be seen as neurons. These nodes are connected to each other, and they have weights associated with them.
Artificial neurons: The simplest type of artificial neurons is a perceptron which takes several binary inputs $\bar{X} = x_1, x_2, ...$, as an input and produces a single binary output y (see fig. 6. The weights of the perceptron $\bar{W} = w_1, w_2, ...$, define the output of the neuron: If the weighted sum $\sum_j w_j x_j$ is less than a threshold value, the output is 0 and 1, if the weighted sum is greater than the threshold value.
By gradually changing the weights of the neurons, the network is able to produce the desired output. To observe learning, small changes in the input shall result in small changes in the output. Using binary outputs like in the perceptron, it is not possible to observe a small change in the output as it is either 0 or 1. To overcome this problem, sigmoid neurons are used. Instead of using a threshold function, a sigmoid function is used to compute the output: $\sigma(x) \equiv \frac{1}{1+e^{-x}}$.
In advanced neural networks, there are other activation functions such as tanH ($\tanh(x)$), ReLU ($\max(0, x)$) or Leaky ReLU ($\max(0.1x, x)$).
Architecture: Together, those neurons form layers. As seen in fig. 7, a Neural Network is made of the input layer, the hidden layers and the output layer. The output layer size

is the number of classes. The graph is fully connected, meaning that the previous layer's nodes are connected to all the next layer's nodes. In feedforward networks, the output from one layer is used as input to the next layer, resulting in no loops. In recurrent neural networks, feedback loops are possible. [10] **Training:** For training a dataset of input and correct output data is need. It works by feeding the input data set $\bar{X}$ into the network to create a prediction $\hat{y}$. The goal is to minimize the error between the prediction and the corresponding desired output $E(\bar{X}) = (y - \hat{y})$. In each epoch, the weights are updated according to

$$\bar{W} \leftarrow \bar{W} + \alpha E(\bar{X})\bar{X}, \tag{21}$$

where α is the learning rate. This is called gradient-descent update. To save computational time, the update is often done on randomly chosen training points and is then called stochastic gradient-descent. To quantify how close the network comes to the desired output, a loss functions needs to be defined. The choice of loss function depends on the application:

- Least-squares regression with continuous output requires a simple squared loss of the form $L = (y - \hat{y})^2$
- If the observed value is in the range -1, 1 and the prediction is a numerical value, logistic regression is used: $L = \log(1 + \exp(-y \cdot \hat{y}))$

Backpropagation: In a single-layer network, the training is simple as the error/loss function can be computed as a direct function of the weights, which enables to compute easy gradient computation. In multi-layer networks, the loss functions is a complicated function of the weights of also earlier layers. However, this gradient can be computed via backpropagation. Using dynamic programming, the backpropagation is divided into the forward and backward phase:

1. In the forward phase, the training input is fed into the neural network. The particular outputs, including the final output, are computed across the layers with the current set of weights. The final predicted output is compared to desired outputs of the training data. The derivative of the loss function with respect to the output is computed.

2. In the backward phase, the gradient of the loss function w.r.t. the different weights in all layers is computed. Using the chain rule, gradients are computed from the output node to the input nodes and used to update the weights, see [1].

3.3 Differential Machine Learning

Differential Machine Learning (DML) is an extension of Vanilla Machine Learning. Models are not only trained on input and output values, but also on differentials of the outputs with regard to the inputs. This leads to the benefit, that the model can learn with knowing the shape of the target function given by the differentials instead of just knowing punctual examples. Especially on small datasets in large dimensions, this will improve the performance of learning.

3.3.1 Adjoint Differentiation

The dataset needed for DML is an augmented version of the dataset for ML: In addition to the input $x^{(i)}$ and the output $y^{(i)}$, the differential $\frac{\partial y^{(i)}}{\partial x^{(i)}}$ is used. Those differentials can be computed via finite difference methods or adjoint differentiation (AD) efficiently. Finite difference is slow and computational costly in comparison to AD. For AD, the derivatives are taken analytically pathwise by application of the chain rule. The method is computationally efficient as the derivative is computed together with the output value itself, so that the path does not need to be re-generated afterwards.
Every function $y = f(\vec{x})$ can be displayed as computation graphs which is a sequence of operations (few basic mathematical operations such as matrix-product, activation function). By knowing the differential of all operations w.r.t. the inputs

$$\bar{y} = \frac{\partial c}{\partial \vec{y}} \tag{22}$$

and using the chain rule, the total differential (the adjoint equation A) can be computed with

$$\frac{\partial c}{\partial \vec{x}} = \frac{\partial \vec{y}}{\partial \vec{x}} \frac{\partial c}{\partial \vec{y}} \quad \rightarrow A : \bar{x} = \varphi_x \bar{y}, \tag{23}$$

where $\partial \vec{y}/\partial \vec{x}$ is the Jacobian matrix. While feedforwarding the graph compute outputs from inputs from left to right, adjoint differentiation differentials to inputs from differentials to outputs and flows right to left. Putting more operations together, all differentials are computed in on traversal of the graph, resulting in backpropagation. Using automatic adjoint differentiation (AAD), the whole process can be automatized. The compiler records the order of execution of the ops, called tape, resulting in a graph in the correct execution order. Backpropagation adjoint equations right to left result in all pathwise differentials. Repeating that for more simulations and averaging converges to true model risks. Modern packages as Tensorflow consist of an implementation of AAD/backpropagation.

3.3.2 Twin network

Given the input, output, and differentials w.r.t. the inputs, DML can be implemented. The model consists of two parts, the feedforward and backpropagation part. The feedforward part is defined by the equations:

$$\begin{aligned} \text{input: } & z_0 = x \\ \text{neurons of layer l: } & z_l = g_{l-1}\left(z_{l-1}\right) w_l + b_l \quad , l = 1, \ldots, L \\ \text{output: } & y = z_L \end{aligned} \tag{24}$$

w_l and b_l are the weights and biases of layer l. The activation function of the layer is denoted by g_{l-1}. In figure 8, a feedforward layer with $L = 3$ can be seen.
In the backpropagation part, we compute the differentials of the predicted value $y = z_L$

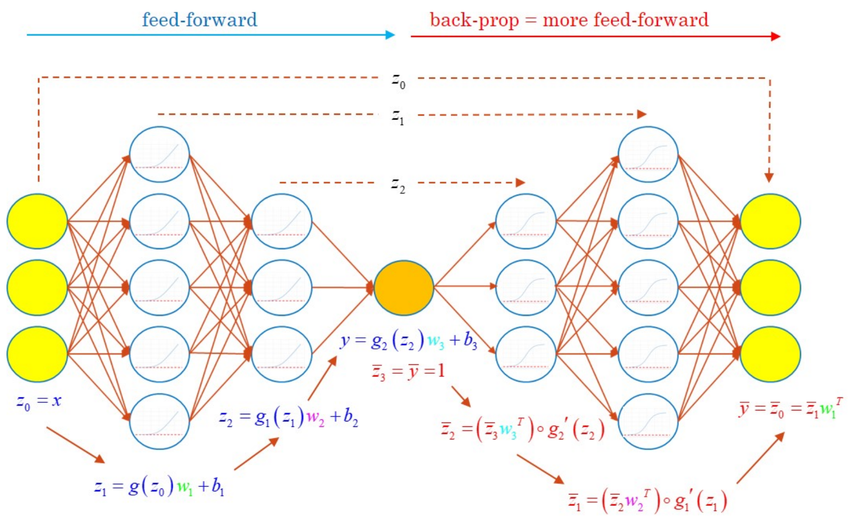

feed-forward
back-prop = more feed-forward
z_0
z_1
z_2
$z_0 = x$
$z_1 = g(z_0) w_1 + b_1$
$z_2 = g_1(z_1) w_2 + b_2$
$y = g_2(z_2) w_3 + b_3$
$\bar{z}_3 = \bar{y} = 1$
$\bar{z}_2 = (\bar{z}_3 w_3^T) \circ g_2'(z_2)$
$\bar{z}_1 = (\bar{z}_2 w_2^T) \circ g_1'(z_1)$
$\bar{y} = \bar{z}_0 = \bar{z}_1 w_1^T$

Figure 8: Twin network [5]

w.r.t. the inputs $x = z_0$. Hence, equations 24 are differentiated in reverse order:

$$\begin{aligned}
\bar{z}_L &= \frac{\partial y}{\partial y} = \bar{y} = 1 \\
\bar{z}_{l-1} &= \frac{\partial y}{\partial z_{l-1}} = \left(\bar{z}_l w_l^T\right) \circ g_{l-1}'\left(z_{l-1}\right) \quad , l = L, \ldots, 1 \\
\bar{x} &= \frac{\partial y}{\partial x} = \bar{z}_0
\end{aligned} \tag{25}$$

Comparing the feedforward and backpropagation network, weights are shared and the neurons $\bar{z}_l$ are the adjoints of the corresponding neurons z_l in the feedforward layer. Combining the feedforward and the backpropagation part leads to the DML twin network in figure 8. In the first part the prediction is computed and in the second part its differentials w.r.t. inputs. Both parts are connected by shared weights z_l. In the middle the networks are connected by combining equations 24 and 25.

3.3.3 Training

As our inputs, outputs and differentials are multidimensional, training data is stacked into matrices, where n is the dimension of the feature space and m the number of neurons in the particular layer, respectively the training examples:

$$X = \begin{bmatrix} x^{(1)} \\ \vdots \\ x^{(m)} \end{bmatrix} \in \mathbb{R}^{m \times n} \quad Y = \begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(m)} \end{bmatrix} \in \mathbb{R}^{m} \quad \bar{X} = \begin{bmatrix} \bar{x}^{(1)} \\ \vdots \\ \bar{x}^{(m)} \end{bmatrix} \in \mathbb{R}^{m \times n}, \tag{26}$$

Feedforward computation (left part) results in Z_l and back-propagating (right part) in $\bar{Z}_l$:

$$Z_l = \begin{bmatrix} z_l^{(1)} \\ \vdots \\ z_l^{(m)} \end{bmatrix} \in \mathbb{R}^{m \times n_l} \quad \text{and} \quad \bar{Z}_l = \begin{bmatrix} \bar{z}_l^{(1)} \\ \vdots \\ \bar{z}_l^{(m)} \end{bmatrix} \in \mathbb{R}^{m \times n_l} \tag{27}$$

In the first part the mean squared error (MSE) between the output Y and the predicted output shall be minimized:

$$C\left(\{w_l, b_l\}_{l=1,\dots,L}\right) = MSE = \frac{1}{m}\left(Z_L - Y\right)^T\left(Z_L - Y\right) \tag{28}$$

In the second part, the network is trained with pathwise differentials $\bar{X}$ instead of the output Y and thus the error $M\bar{S}E$ shall be minimized:

$$C\left(\{w_l, b_l\}_{l=1,\dots,L}\right) = \overline{MSE} = \frac{1}{m}\operatorname{tr}\left[\left(\bar{Z}_0 - \bar{X}\right)^T\left(\bar{Z}_0 - \bar{X}\right)\right] \tag{29}$$

In total training, both errors are combined, where λ is a hyperparameter:

$$C = MSE + \lambda\overline{MSE} \tag{30}$$

3.3.4 Benefits

Differential Machine Learning has several benefits compared to Vanilla Machine Learning:

1. **Larger dataset size:** With additionally considering the differentials in the training set, the size has increased significantly. AAD produces a much larger dataset (nm additional differentials) without a lot of computational extra costs.

2. **Learn the shape of the pricing function:** By learning from the gradients, instead of just punctual examples, the shape of the function can be learning resulting in a more robust learning.

3. **Production of correct Greeks :**Learning the shape of the pricing function results in a correct construction of the Greeks, which is important for many financial applications.

4. **Bias-free regularization:** Equation 30 is similar to classic regularization, which penalizes large weights and thus avoids overfitting. In comparison to classic regularization, there is no bias introduced here as just learning with the differentials would also converge to the true approximation. Thus, differential regularization is more similar to data augmentation, as there is more data available, reducing the variance without adding bias. [5]

3.4 Hyperparameter tuning with Bayesian Optimization

Hyperparameter search for neural networks is a non-trivial task. In order to obtain suitable hyperparameters that allow the neural network to generalize well, techniques such as grid search employing a brute-force approach to finding suitable hyperparameters may be used. However, hyperparameter search can also be formulated as an optimization problem that tries to find the optimum of an expensive-to-evaluate, multi-optimum cost function, i.e., optimizing the validation error of the neural network with respect to its hyperparameters.

An optimization technique called bayesian optimization is well suited for optimizing expensive black box functions with many local optima like neural network training. This

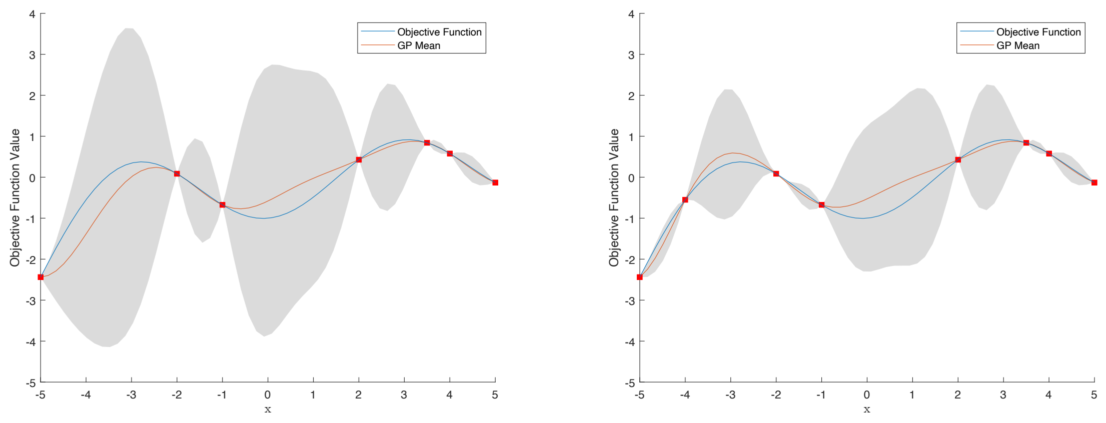


Figure 9: A GP models an unknown objective function, providing an estimated mean and a covariance at every point x in the optimization space. On the left side, a Gaussian process after seven objective function evaluations is displayed. The right-hand side displays the same Gaussian process after sampling the objective function at $x = 4$ and updating the GP. The GP on the left side is updated using the Bayes rule given an objective function sample to obtain the GP displayed on the right side. With more objective function evaluations, the Gaussian process is able to model the objective function more accurately. In the figure, this can be observed as the certainty of the GP around the newly evaluated point increases.

approach is used to find well-suited hyperparameters for all neural networks trained in this project. This section is strongly based on [2], [12] and [15] in structure and content.

In Bayesian optimization, Gaussian processes (GP) are used to model the black-box objective function via a mean and covariance, effectively using previous evaluations of the neural network training process to quantify uncertainty and expectation of the performance of the neural network for all hyperparameters. An example of such a GP is visualized in 9. Given a Gaussian process modeling an objective function, an acquisition function can be defined. Acquisition functions are used to score each point in the optimization space. This score provides a heuristic of where to evaluate the objective function next. It provides a trade-off between exploring the objective function and finding optimal values.

It is then possible to use the acquisition function to determine where to evaluate the objective function next, i.e., determine with which hyperparameters to re-train and validate the neural network. After evaluating the objective function, the GP is updated with the newly obtained data point, making it more accurate. This is done multiple times, leading to finding better hyperparameters over multiple iterations.

4 Methods

4.1 Data Generation

Simulating geometric Brownian motions efficiently is the backbone of the expected exposure simulation as described in section 2.2 and therefore vital for generating large amounts of training data and being able to benchmark the models that have been created. For each kind of the financial contract from section 2.3. this generally splits up into two parts (even though it slightly differs depending on the contract): the scenario generation of the risk factors and the (nested) Monte Carlo simulations for pricing the contracts in the regarding scenario.

4.1.1 Efficient Simulation of Geometric Brownian Motions for scenario generation

In the following, n_p denotes the number of paths that are simulated, indexed by i, n_t denotes the number of time steps of the simulation time grid, indexed by j and n_u denotes the number of underlying stocks, each indexed by k.

The simulation of a single path of the Brownian motion over time is done in the following way: The vector $S^{(i)}(t_j) \in \mathbb{R}^{n_u}$ containing the n_u stock values at time point $t = t_j$ is simulated using a discrete version of the multidimensional geometric Brownian motion

$$S_k^{(i)}(t_j) = \prod_{j'=1}^{j} r_{j'k}^{(i)} \tag{31}$$

for $k \in [n_u]$, where $r_{jk}^{(i)} \in \mathbb{R}$ contains the evolution of the k-th element of the multidimensional geometric brownian motion in the time range $\Delta t_j = t_j - t_{j-1}$. Furthermore,

$$\hat{R}^{(i)} := \begin{bmatrix} S^{(i)}(0) \\ R^{(i)} \end{bmatrix} \in \mathbb{R}^{(n_t+1)\times n_u} \tag{32}$$

where $S^{(i)}(0) \in \mathbb{R}^{n_u}$ is the vector of the initial spot prices of the underlyings and the matrix

$$R^{(i)} = \begin{bmatrix} r_{11}^{(i)} & ... & r_{1n_u}^{(i)} \\ \vdots & ... & \vdots \\ r_{(n_t-1)1}^{(i)} & ... & r_{(n_t-1)n_u}^{(i)} \end{bmatrix} \in \mathbb{R}^{(n_t)\times n_u} \tag{33}$$

is computed efficiently through

$$R^{(i)} = \exp(-0.5 \cdot \Delta t \cdot diag(\sigma^{(i)} \cdot (\sigma^{(i)})^T) + (\sqrt{\Delta t} \circ W^{(i)}) \cdot Chol(\Sigma^{(i)})) \tag{34}$$

where $\circ$ denotes the Hadamard product, $W^{(i)} \in \mathbb{R}^{(n_t-1)\times n_u}$ is a matrix of iid standard normally distributed random numbers.
The covariance matrix of the stocks $\Sigma^{(i)} = (\sigma^{(i)} \cdot (\sigma^{(i)})^T) \circ C^{(i)}$ captures the volatilities $\sigma^{(i)} \in \mathbb{R}^{n_u}$ of the individual underlying stocks as well as their correlations $C^{(i)} \in \mathbb{R}^{n_u \times n_u}$. Finally, "Chol" denotes that the Cholesky decomposition of the covariance matrix is computed.

The matrix Δt capturing the simulation time grid is given by

$$\Delta t = \begin{bmatrix} \Delta t_1 & ... & \Delta t_1 \\ \vdots & & \vdots \\ \Delta t_{n_t} & ... & \Delta t_{n_t} \end{bmatrix} \in \mathbb{R}^{(n_t)\times n_u} \tag{35}$$

is built using the sizes $\Delta t_j = t_j - t_{j-1}$ of the individual time steps where $t_0 = 0$ denotes the starting time of the simulation.

The tensor $S \in \mathbb{R}^{n_p \times n_t \times n_u}$ which contains the evolution of n_u underlying stocks over n_t time steps for a total of n_p simulated paths with different volatilities and spot prices can then be computed efficiently without loops on a GPU. This can be identified to follow the pathwise simulation procedure mentioned in section 2.2.

Note that in order to generate a training set of size, n_p one would use this procedure to generate one scenario per time step for each of the samples in the training set.

4.1.2 Geometric Brownian Motion Hyper Rectangle for Efficient Nested Monte Carlo Simulations

Using the simulation routine of the geometric Brownian motion we just described, we are now in the situation that we have for each training sample a 2 dimensional matrix of realizations of prices of the underlying. The first dimension describes the realizations for different time steps, and the second dimension for the different underlyings.
The price of a derivative clearly depends on the scenario we are in and the realization of the risk factors we are looking at respectively. In principle, one could now apply a closed form solution to derive the value of the derivative depending on the realized price of the underlyings at each time step (if such a closed form solution exists) to come up with the exposure of the derivative in the respective scenario. Since we want to have a general approach to be able to extend the methodology later to more complex situations, we aim to do the pricing with a nested Monte Carlo simulation.
To do this efficiently, the concept of geometric Brownian motion cubes is introduced. It generally follows the idea of direct simulation described in section 2.2.

In order to compute the value of a derivative at a certain time step using a Monte Carlo approach, the underlyings are simulated n_n times until maturity, the derivative's payoff is evaluated and the average over those n_n simulations is used as an estimate for the value of the derivative (since we are assuming the interest rate is 0 there is no need for discounting).
In the particular case we are in we want to simulate for each point on the time grid n_n realizations of each underlying until maturity using the same procedure as described before with the only difference that the time differences are substituted by

$$\Delta t' = \begin{bmatrix} T - t_0 & \dots & T - t_0 \\ \vdots & \dots & \vdots \\ T - t_{(n_t-1)} & \dots & T - t_{(n_t-1)} \\ 0 & \dots & 0 \end{bmatrix}. \tag{36}$$

In that way we directly simulate the evolution of the underlyings from a certain time point to maturity without looking at the intermediate steps.
Multiplying those resulting $n_n \times n_t \times n_u$ realizations now suitably with the corresponding realizations of the spot prices from the scenario generation, we get $n_n \times n_t \times n_u$ realizations for the prices of the underlyings at maturity corresponding to the realized scenarios at the different time steps. Computing then the payoff of the derivative we want to price over

the n_u axis for each time step and for each simulated path, we get an array of n_n different derivative payoffs for all n_t different time steps. Averaging afterwards over the n_n payoffs leaves us with an estimate for the derivative's price at each time point, depending on the scenario we generated for this time point in the previous step. Taking then at each time step the maximum of these estimates and 0 leaves us with an estimate for the expected positive exposure of the derivative at the regarding time point. Taking the maximum of the negative of the initial estimate we found and 0 we get an estimate of the expected negative exposure.
We can execute this procedure in parallel for each training data sample to achieve generation of the training data on a GPU without the use of loops.
There are different subtleties to mention for the different derivatives we were looking at. First, for an European call option we only need to simulate one underlying at a time, hence $n_u = 1$. The same holds for forward contracts.

The most notable difference is there for the simulation of the expected exposure of Bermudan call options, since their value does not only depend on the simulated payoff at maturity but at several exercise dates. On the other hand, they also only depend on a single underlying asset, which simplifies the evaluation.
As seen in section 2.3. the value of such an option at a certain time point t can be found by simulating the expected payoff for all future exercise dates, i.e. the ones that are between the current time point and maturity. To reuse the geometric Brownian motion cube we constructed before, we include an additional dimension containing simulations from the current time point to all upcoming exercise dates. Therefore, we are again in a 3 dimensional setting for each training sample: one dimension over n_t describing the point on the time grid we are currently at, one over n_n describing the path we are sampling and one over n_d describing the payoff date we are simulating to. Note that the last dimension has fewer entries the further we are on the simulation time grid for the scenario generation because there are less future exercise dates coming up. Having simulated the price of the underlying over those 3 dimensions, the payoff at each simulated point is computed, and the average is built regarding the second dimension. In that way, we get for every point in the time grid the expected payoff for all future payoff dates. Taking now the maximum along the n_d axis, we find the value of the derivative at each time point t by virtue of the result from section 2.3. Since those values are all positive, this already serves as an estimate for the expected positive exposure of one training sample of the Bermudan option over time. Parallelization of the different training samples is again possible as before.

4.2 Building the model

The Differential Machine Learning model introduced in chapter 3.3 was adapted to fit our problem. A visualization of the model can be seen in figure 10. The input layer consists of two neurons matching the input dimension of two (volatility and initial spot price). The model consists of n hidden feed-forward and n hidden backpropagation layers. Each hidden layer has m neurons, and feedforward and backpropagation layer share weights. n and m are hyperparameters are tuned during the validation phase. The feed-forward neurons have softplus as an activation function, while the backpropagation have sigmoid.The

model computes the expected exposure for 10 time steps, therefore the output in the

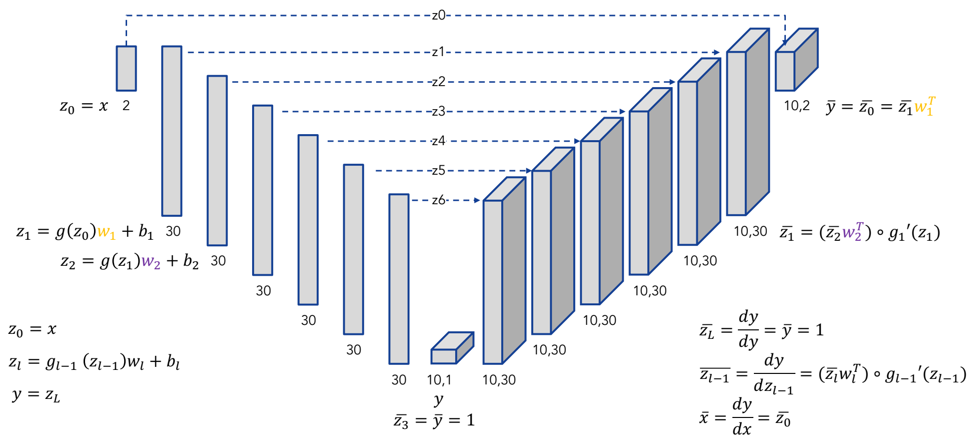

z0
z1
z2
z3
z4
z5
z6
$z_0 = x$
2
$z_1 = g(z_0)w_1 + b_1$
30
$z_2 = g(z_1)w_2 + b_2$
30
30
30
30
30
$z_0 = x$
$z_l = g_{l-1}(z_{l-1})w_l + b_l$
$y = z_L$
10,1
y
$\bar{z}_3 = \bar{y} = 1$
10,30
10,30
10,30
10,30
10,30
10,30
10,2
$\bar{y} = \bar{z}_0 = \bar{z}_1 w_1^T$
$\bar{z}_1 = (\bar{z}_2 w_2^T) \circ g_1'(z_1)$
$\bar{z}_L = \frac{dy}{dy} = \bar{y} = 1$
$\overline{z_{l-1}} = \frac{dy}{dz_{l-1}} = (\bar{z}_l w_l^T) \circ g_{l-1}'(z_{l-1})$
$\bar{x} = \frac{dy}{dx} = \bar{z}_0$

Figure 10: Visualization of the used Differential Machine Learning network with n = 6 hidden layers and a hidden layer size of m = 30

middle of the network is of the shape [10,1]. So the following backpropagation layers also need one dimension more, as there is a differential at every time step, resulting in a differential output at the end in the shape of [10,2].

4.3 Training the model

4.3.1 Training procedure

First, the input and output are standardized by subtracting the mean and dividing by the standard deviation. To normalize the differentials dx/dy, they are divided by the standard deviation of y and multiplied with the standard deviation of x.
Before training the neural network, the data is split into a training set (66%) and a validation set (34%). The model is trained with the training set and hyperparameters are tuned with the validation set. The model was trained in 100 epochs using the NADAM optimizer (found to be optimal in the hyperparameter optimization). Learning rate and batch size are also hyperparameters (see next subsection).

4.3.2 Hyperparameter tuning

Using Bayesian optimization, explained in chapter 3.4, hyperparameters are tuned on the validation set. For every training set, different hyperparameters were found. Table 1 in the appendix shows some exemplary resulting hyperparameters found in the optimization.

4.4 Evaluating the model

After training the neural networks with the best hyperparameters obtained in Bayesian optimization, their performance is evaluated on a test set.

This test set consists of 400 instances of the financial product the model is built for and contains the input volatilities and spot prices as inputs as well as accurate exposures as outputs. The exposures have been generated using efficient Monte Carlo simulations outlined in 4.1.1. In order to be useful for benchmarking, the Monte Carlo simulations

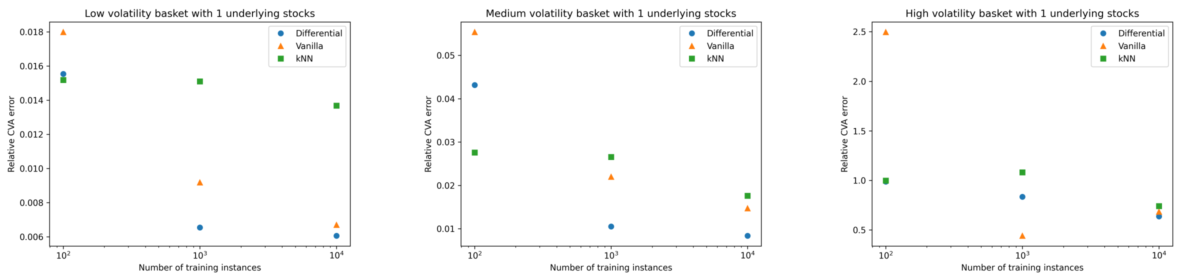


Figure 11: Basket option with one underlying

for the same starting values are run multiple times and the average over the results are computed, reducing the variance of the exposure values and making them suitable for benchmarking the machine learning model.
To evaluate the machine learning models, the exposure given the test set inputs is computed. CVA is then computed using those exposure values, it is denoted as CVA_{pred}. The CVA computed from the test set exposures are denoted as CVA_{test}.
The key measure to benchmark the performance of a xVA computation method used in the following results section is then computed as the mean squared error (MSE) over the relative difference of the predicted and test CVAs as

$$Relative\ Error = MSE(\frac{CVA_{pred} - CVA_{test}}{CVA_{test}}). \tag{37}$$

5 Results and Discussion

In order to evaluate the performance of the vanilla neural network (ML) approach and the differential machine learning (DML) approach, they are benchmarked on different data sets representing different numbers of training instances, different data generation ranges and different types of financial products. The results are then compared to the performance of a simple k-nearest-neighbor-based (kNN) inference approach. Simple approximations through kNNs are motivated by practice in the industry, where intraday the CVA is approximated based on interpolation of CVA values in a grid of parameters (possibly populated by front-office systems) and then recalculated overnight by doing the expensive Monte Carlo simulations.

5.1 Comparison between DML, ML and kNN

Our final results show two key findings:

(A) Growing amount of training instances lead to an increasingly better performance of DML over kNN.

(B) Small volatility range leads to better results for DML, ML and kNN.

Regarding insight (A), figure 11 shows the relative error of DML, ML and kNN. It can be seen that DML is up to 60% more accurate than kNN and up to 56% better than vanilla

Neural Networks. Relative error of other financial options can be seen in the appendix, but they show the same behavior.
With respect to the key insight (B), smaller volatility ranges lead to better results for all three pricing methods, so this is not particular to DML. It is found that this behavior is mitigated by providing a larger number of training samples. As discussed in the previous paragraphs, neural network-based approaches perform particularly well when a larger number of training instances are provided, resulting in another advantage of DML. Results of the influence of volatility ranges on the performance in detail can be found in the appendix.
To summarize, differential machine learning possesses multiple advantages over vanilla neural networks and the kNN-based approach, justifying the higher implementation effort.

5.2 Impact on execution time

Monte Carlo simulations can compute exposure estimates up to arbitrary precision, depending on the time allocated for simulation. As seen in the previous sections, the performance of machine learning based approaches depends on the amount of data provided for training. After training, however, estimating exposure up to a high precision only requires one compute-inexpensive forward pass of the neural network.
To directly compare the valuation time of Monte Carlo based approaches to inference using neural networks, expected exposures for 400 basket options with five underlyings are computed using a Monte Carlo-based approach as described in subsection 4.1.1. A differential machine learning model is trained on 1000 training instances and then evaluated to achieve an average relative CVA error of 0.7% in $0.4s$ on the test data. When evaluating the execution time of a Monte-Carlo-simulation to achieve the same relative CVA error, $22.7s$ are needed. Monte-Carlo simulation and differential machine learning model both were implemented in tensorflow and executed on the same hardware to guarantee comparability.
Differential machine learning models achieving similar results are therefore around 57 times faster than Monte Carlo simulations in this setting. This makes machine learning based approaches explored in this project more suited for intraday valuation than Monte Carlo simulations, and constitutes the second key insight of this project.

6 Conclusion

Over the course of the project, a framework for running efficient Monte Carlo simulations for the computation of exposure values was implemented that allows computing the derivative of the outputs of the simulations with respect to the inputs. Different machine learning-based strategies to build parametric models for accurately and efficiently computing the exposures were implemented and evaluated against each other. It was found that a technique called differential machine learning can be used to significantly improve model performance and decrease the need for expensive data generation. Data-driven approaches lead to a significant execution-time improvements over Monte Carlo simulations, which can compute high-precision option valuation with large computational effort.
Future research may evaluate the performance of the aforementioned xVA computation

techniques using deep learning on more complex financial products with more underlyings and compute different xVAs, for example margin valuation adjustments (MVA) or credit valuation adjustments (KVA). Furthermore, the proximity to reality of the xVA computations may be increased by considering risk mitigation measures like netting and collateral.

References

[1] Charu C. Aggarwal. *Neural Networks and Deep Learning*. Cham: Springer International Publishing, 2018. ISBN: 978-3-319-94462-3. DOI: 10.1007/978-3-319-94463-0.

[2] E. Brochu, V. M. Cora, and N. de Freitas. "A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning". In: (Dec. 2010).

[3] James Crotty. "Structural causes of the global financial crisis: a critical assessment of the â˜new financial architectureâ™". In: *Cambridge journal of economics* 33.4 (2009), pp. 563–580.

[4] Matthias Groncki. *CVA Calculation with QuantLib and Python*. 2015.

[5] Brian Huge and Antoine Savine. *Differential Machine Learning*. URL: http://arxiv.org/pdf/2005.02347v4.

[6] Jon Gregory. *The xVA Challenge*. Wiley, 2015.

[7] Le Nezet, Bertrand. *Credit Curve Bootstrapping*. R library, 2020.

[8] Yura Mahindroo, Barron, Ewan, and Michael Codling. *xVA explained: Valuation adjustments and their impact on the banking sector*. 2015.

[9] Steven Marshall. *A Brief History Of XVA*. LinkedIn, 2019.

[10] Michael A. Nielsen. *Neural networks and deep learning*. Vol. 25. Determination press San Francisco, CA, USA, 2015.

[11] Pascal Pierrot. *How to use Machine Learning for efficient xVA calculations: The case of Credit Valuation Adjustment (CVA)*.

[12] C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006.

[13] Matthias Scherer and Sebastian Walter. "CVA für Kontrahenten-Ausfallrisiken: Bewertungsadjustierungen". In: *Risiko Manager* 2015.15.-16. 2015 ().

[14] Stuart Nield. *FVA - time to go asymmetric?* 2018. URL: https://ihsmarkit.com/research-analysis/fva-time-to-go-asymmetric.html.

[15] Tim Emmert. "Parameter Identification and Design of Experiments for Truck Trailer Combinations using Bayesian Optimization". Bachelor thesis. Erlangen-Nürnberg: Friedrich-Alexander-Universität, 2021-06-01.

Appendix

Exemplary Hyperparameter Values

Table 1: Exemplary values found in the hyperparameter optimization for Differential Machine Basket option with 1 and 5 underlyings and parameter range

Hyperparameter	**Values for DML with 1 underlying**	**Values for DML with 5 underlyings**	**Range**
Optimiser	NADAM	ADAM	ADAM or NADAM
Number of layers n	5	6	(1, 10)
Layer size m	25	27	(10, 30)
Batch size	448	9	(8, 512)
Learning rate	0.056	0.055	(0.00001, 0.1)

Relative error of DML, ML and kNN on different financial products

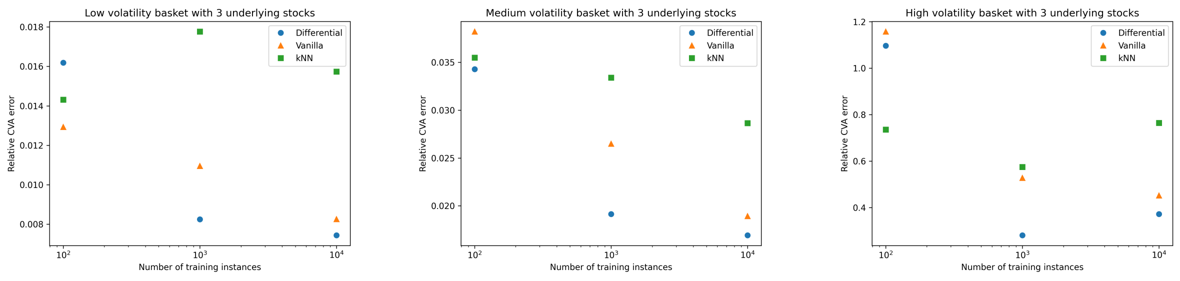


Figure 12: Basket option with three underlyings

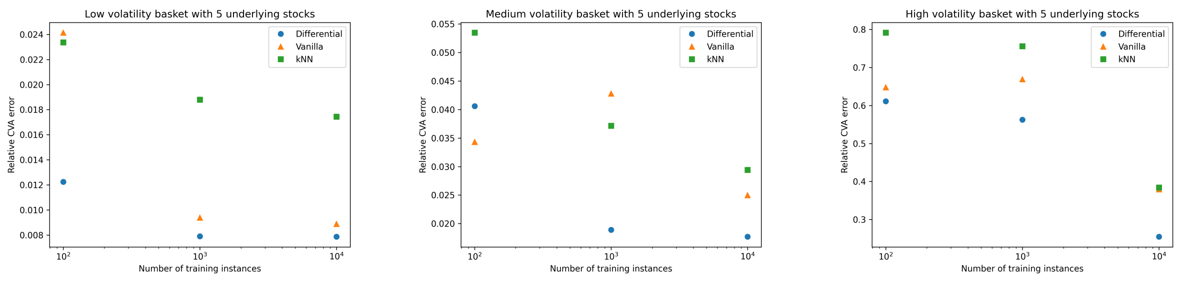


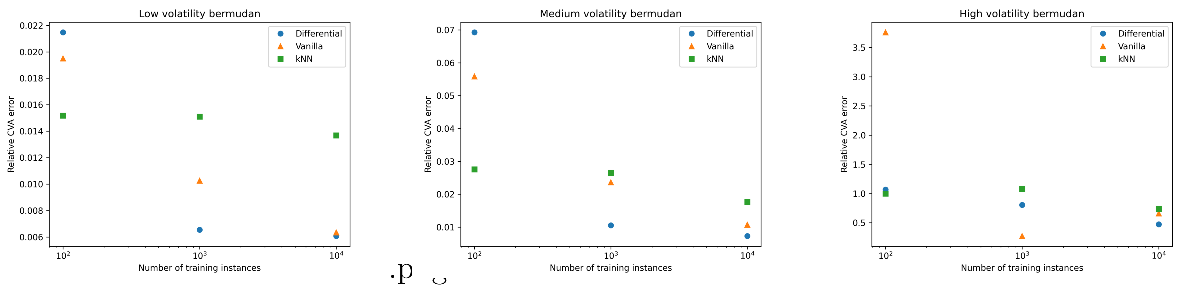
Figure 13: Basket option with five underlyings

Financial option	**Volatility range**	**# simulated paths**	**# underlying stocks**	**Differential error**	**Vanilla error**	**kNN error**	**Relative improvement Differential to kNN**
bermudan	medium	100	1	0.0692	0.0559	0.0276	-1.5099
basket	medium	100	1	0.0431	0.0554	0.0276	-0.5641
basket	large	100	3	1.0966	1.1575	0.7352	-0.4915
bermudan	small	100	1	0.0215	0.0195	0.0152	-0.4142
basket	small	100	3	0.0162	0.0129	0.0143	-0.1311
bermudan	large	100	1	1.0677	3.7588	0.9989	-0.0688
basket	small	100	1	0.0155	0.0180	0.0152	-0.0238
basket	large	100	1	0.9884	2.4963	0.9989	0.0105
basket	medium	100	3	0.0343	0.0382	0.0355	0.0339
basket	large	10000	1	0.6377	0.6842	0.7412	0.1396
basket	large	100	5	0.6113	0.6481	0.7916	0.2278
basket	large	1000	1	0.8348	0.4410	1.0824	0.2288
basket	medium	100	5	0.0406	0.0343	0.0535	0.2406
bermudan	large	1000	1	0.8069	0.2722	1.0824	0.2546
basket	large	1000	5	0.5624	0.6695	0.7558	0.2559
basket	large	10000	5	0.2547	0.3796	0.3842	0.3371
bermudan	large	10000	1	0.4729	0.6603	0.7412	0.3619
basket	medium	10000	5	0.0177	0.0250	0.0294	0.3984

Table 2: Overview over relative error of Differential Machine Learning, Vanilla Machine Learning and kNN (sorted by relative improvement of Differential Machine Learning compared to kNN (part 1)

Financial option	**Volatility range**	**# simulated paths**	**# underlying stocks**	**Differential error**	**Vanilla error**	**kNN error**	**Relative improvement Differential to kNN**
basket	medium	10000	3	0.0169	0.0189	0.0286	0.4092
basket	medium	1000	3	0.0191	0.0265	0.0334	0.4272
basket	small	100	5	0.0122	0.0241	0.0234	0.4758
basket	medium	1000	5	0.0189	0.0428	0.0372	0.4908
large	basket	1000	3	0.2802	0.5280	0.5743	0.5121
large	basket	10000	3	0.3719	0.4521	0.7643	0.5134
basket	medium	10000	1	0.0084	0.0147	0.0176	0.5247
basket	small	10000	3	0.0074	0.0083	0.0157	0.5274
basket	small	1000	3	0.0083	0.0110	0.0178	0.5354
basket	small	10000	5	0.0079	0.0089	0.0174	0.5491
bermudan	small	10000	1	0.0061	0.0064	0.0137	0.5558
basket	small	10000	1	0.0061	0.0067	0.0137	0.5570
bermudan	small	1000	1	0.0066	0.0103	0.0151	0.5656
basket	small	1000	1	0.0065	0.0092	0.0151	0.5666
basket	small	1000	5	0.0079	0.0094	0.0188	0.5799
bermudan	medium	10000	1	0.0073	0.0107	0.0176	0.5858
bermudan	medium	1000	1	0.0105	0.0237	0.0265	0.6029
basket	medium	1000	1	0.0105	0.0220	0.0265	0.6046

Table 3: Overview over relative error of Differential Machine Learning, Vanilla Machine Learning and kNN (sorted by relative improvement of Differential Machine Learning compared to kNN (part 2)



.p

Figure 14: Bermudan option

Volatility range	**Relative DML error**	**Relative ML error**	**Relative kNN error**
Small	0.0102	0.0120	0.0163
Medium	0.0247	0.0307	0.0301
Large	0.6655	1.0123	0.8042
Average	**0.2335**	**0.3517**	**0.2835**

Table 4: Relative errors of DML, ML and kNN for different volatility ranges