POLITECNICO DI TORINO

Corso di Laurea Magistrale
in Ingegneria Matematica

Tesi di Laurea Magistrale

Differential Machine Learning – A key tool for practical risk management

Relatore

prof. Paolo Brandimarte

Candidato

Corrado Costanzi

Anno Accademico 2020-2021

Abstract

Differential machine learning combines automatic adjoin differentiation (AAD) with modern machine learning (ML) in the context of financial risk management. This work aims to resolve computational bottlenecks of derivatives risk reports and capital calculation by introducing novel algorithms for training fast, accurate pricing and risk approximation, online, in real time, with convergence guarantees. Differential ML is a general extension of supervised learning: the model is not only trained on examples of inputs and labels but also on differentials of labels with respect to inputs. It is also applicable in many situations outside finance, where high quality first-order derivatives with respect to training inputs are available. AAD computes *pathwise differentials* with remarkable efficacy so differential ML algorithm provides extremely effective pricing and risk approximation. The algorithm can produce fast analytics in model too complex for closed form solutions, extract the risk factor of complex transactions and trading book, and effectively computed risk management metrics. In particular, three practical cases have been tested: an European call option, a Basket option and a Down-and-Out barrier option, with remarkable results in each case.

Acknowledgements

Non sono molto bravo a scrivere i ringraziamenti, non è nemmeno una cosa che amo particolarmente fare. Mi sembra sempre di scrivere frasi inflazionate e mielose, in più credo di non avere abbastanza talento comunicativo per riuscire ad esprimere efficacemente l'importanza che avete avuto voi in questi anni. Ho pensato di sfruttare le conoscenze maturate in questi sei anni tra numeri e formule per esprimere la ragione per cui vi sono grato associandovi a un'equazione che penso possa riassumere il motivo della mia gratitudine, ma inizio subito con un GRAZIE che va a tutti, d'ora in poi consideratelo come implicito in ogni frase così evito di ripetermi; esistono troppo pochi sinonimi per una parola così semplice.

Vorrei ringraziare in primis il prof. Brandimarte per tutti i consigli, le idee e per il supporto ricevuto. La sua simpatia ed il suo genio sono stati un'ispirazione in questi anni.

Agli amici "*torinesi*" collego lo sviluppo in serie di Fourier: $f(n) = \frac{a_0}{2} + \sum_{n=i}^{N}[a_n \cos(nt) + b_n \operatorname{sen}(nt)]$. Come ogni funzione periodica questi anni felici dell'università sono passati tra una ciclica alternanza di gioie, delusioni, momenti di allegria e di sconforto, orgoglio e sconsolazione. Se scomponessi questa esperienza vedrei che le componenti fondamentali siete stati voi.

Alla mia famiglia, mio padre Paolo e mia madre Anna associo il teorema di Bayes: $P(A|B) = \frac{P(B|A) * P(B)}{P(A)}$. La formula lega la probabilità di un evento rispetto ad una causa nota. Ecco, sicuramente tutto ciò che nel mio piccolo sono riuscito a fare è condizionato a tutti gli sforzi, il supporto ed il fatto che non mi avete mai fatto mancare niente per farmi arrivare qui.

Ad Alessandro, mio fratello, ed ai miei cugini Antonio e Riccardo volevo associare la forza di gravità per come vincola le stelle tra loro nonostante la lontananza: $F = -G\frac{m*M}{r^2}$. Come ben sapete l'intensità della forza decresce con l'aumentare della distanza ma se con voi questo non accade deve essere dovuto alla costante che ci lega, ben più grande di G.

A Camilla, voglio associare la teoria lineare del moto ondoso $\varphi = \frac{ag}{\omega}\frac{\cosh(k(h+z))}{\cosh(kh)}\sin(kx) - wt$, perché non scorderò mai le parole che mi dedicasti 6 anni fa "Non c'è sfiorare senza mutare. E nel mutare sorridere. […] Non temiamo il destino. Non ci tireremo indietro. Prima di essere schiuma saremo indomabili onde.". Dopo 6 anni rileggo ancora questa poesia, e sono sempre più convinto che tutto questo non sarebbe mai stato possibile senza di te, sono orgoglioso di averti al mio fianco.

A mia nonna Iole, la vera forza della famiglia, ed ai miei nonni Antonio, Elda e Corrado che sarebbero stati molto felici di leggere questo ringraziamento voglio dedicare invece questo momento, la corona di alloro, tutte le foto ed i sorrisi. Perché è solo grazie ai lori sforzi che tutto questo è stato possibile.

Ai miei Zii, Franco, Marina, Leondina e Sergio voglio dedicare l'analisi alle componenti principali (PCA): lo scopo della tecnica è quello di ridurre il numero più o meno elevato di variabili che descrivono un insieme di dati alle componenti principali, ossia variabili latenti che spiegano il fenomeno. Ecco se dovessi applicare questa tecnica per spiegare il mio percorso, voi sareste sicuramente una delle componenti principali.

Tommaso, Nicolas, Federico ed Alberto voi siete le equazioni di Maxwell: $\nabla * E = \frac{\rho}{\varepsilon}$, $\nabla * B = 0$, $\nabla\, x\, E = \frac{-dB}{dt}$, $\nabla\, x\, B = \mu * J + \mu\varepsilon\frac{dE}{dt}$. Queste equazioni descrivono come il campo elettrico accompagni il campo magnetico e viceversa, di come si influenzano e di come evolvano insieme; proprio come voi che siete sempre stati presenti nelle mie avventure avete contaminato i miei interessi e le mie idee. Siete da sempre un motivo di ispirazione.

A Giammarco, Kevin, Lorenzo e a tutti gli altri amici che in questi anni hanno condiviso con me una parte di questo percorso dedico l'elasticità: la proprietà che permette ad un corpo di deformarsi sotto l'azione di una forza esterna e di riacquisire la sua forma originale al venir meno della causa sollecitante. Ecco la nostra amicizia è affetta da questa proprietà e nonostante le distanze e gli impegni della vita è capace di rimanere sempre uguale. Avete la capacità di rendere la vita più leggera.

Contents

The fool doth think he is wise, but the wise man knows himself to be a fool.

[WILLIAM SHAKESPEARE, As You Like It]

1 Introduction

Pricing function approximation is crucial for derivatives risk management, where the value and risk of transactions and portfolios must be computed rapidly. Exact closed-form formulas *a la* Black-Scholes-Merton are only available for simple instruments and simple models. More realistic stochastic models and more complicated exotic transactions require numerical pricing by finite difference methods (FDM) or Monte-Carlo (MC), which is too slow for many practical applications. Researchers experimented with e.g. moment matching approximation for Asian and Basket options, or Taylor expansions for stochastic volatility model. Iconic results like the Hagan's SABR formula were derived in 1990s, and allowed the deployment of sophisticated models on trading desks. New results are being published regularly.

Although pricing approximation was traditionally derived by hand, automated techniques borrowed from the fields of Artificial Intelligence (AI) and Machine Learning (ML) got traction in recent years. Standard ML trains neural networks (NN) and other supervised ML models on punctual examples. The general format is the classic supervised learning: approximate asset pricing functions $f(x)$ of a set of inputs x, with a function $\hat{f}(x; w)$ subject to a collection of adjustable weights w, learned from a training set of m examples of inputs $x^{(i)}$(each a vector of dimension n) paired with labels $y^{(i)}$(typically real numbers), by minimization of a cost function (often the mean square error between predictions and labels). The training sets included a vast number of examples labeled by ground truth prices, computed by numerical methods. This approach essentially interpolates prices in parameter space. ML models generally learn approximations from training data alone, without additional knowledge of the generative simulation model or financial instrument. Although performance may be considerably improved on a case-by-case basis with contextual information such as the nature of the transaction, the most powerful and most widely applicable ML implementations achieve accurate approximations from data alone. Neural networks, in particular, are capable of learning accurate approximations from data. Trained NN computes prices and risks with near analytic speed. Inference is as fast as a few matrix by vector products in limited dimension, and differentiation is performed in similar time by backpropagation.

This work aims to implement and extend the main idea of Antoine Savine and Brian Huge [1]. In chapter 2 (Artificial Intelligence and Differential Machine Learning) an introduction to Neural networks have been shown together with the Differential Machine Learning context. Basic features of the financial framework are presented in chapter 3 together with the Annex. Finally, in section 4 main results and scraps of code have been published.

2 Artificial Intelligence and Differential Machine Learning

In computer science, the term artificial intelligence (AI) refers to any human-like intelligence exhibited by a computer, robot, or other machines. In popular usage, artificial intelligence refers to the ability of a computer or machine to mimic the capabilities of the human mind—learning from examples and experience, recognizing objects, understanding and responding to language, making decisions, solving problems—and combining these and other capabilities to perform functions a human might perform, such as greeting a hotel guest or driving a car.

After decades of being relegated to science fiction, today, AI is part of our everyday lives. The surge in AI development is made possible by the sudden availability of large amounts of data and the corresponding development and wide availability of computer systems that can process all that data faster and more accurately than humans can. AI is completing our words as we type them, providing driving directions when we ask, vacuuming our floors, and recommending what we should buy or binge-watch next. In addition, its driving applications—such as medical image analysis—helps skilled professionals do important work faster and with greater success.

An Artificial Neural Network is based on a collection of connected units or nodes called artificial neurons, which model the neurons in a biological brain. Each connection, like the synapses in a biological brain, can transmit a signal to other neurons. The "signal" at a connection is a real number, and the output of each neuron is computed by some non-linear function of the sum of its inputs. The connections are called edges. Neurons and edges typically have a weight that adjusts as learning proceeds. The weight increases or decreases the strength of the signal at a connection. Typically, neurons are aggregated into layers. Different layers may perform different transformations on their inputs. Signals travel from the first layer (the input layer), to the last layer (the output layer), possibly after traversing the layers multiple times. Because of their ability to reproduce and model nonlinear processes, artificial neural networks have found applications in many disciplines.

Let us first introduce notations for the description of feedforward networks. Define the input (row) vector $x \in R^n$ and the predicted value $y \in R$. For every layer $l = 1, \dots, L$ in the network define a scalar 'activation' function $g_{l-1} : R \to R$. Popular choice are *relu*, *elu* and *softplus*, with the convention $g_0(x) = x$ is the identity. The notation $g_{l-1}(x)$ denote elementwise application. We denote $w_l \in R^{n_{l-1} * n_l}$, $b_l \in R^{n_l}$ the weights and the biases of layer l.

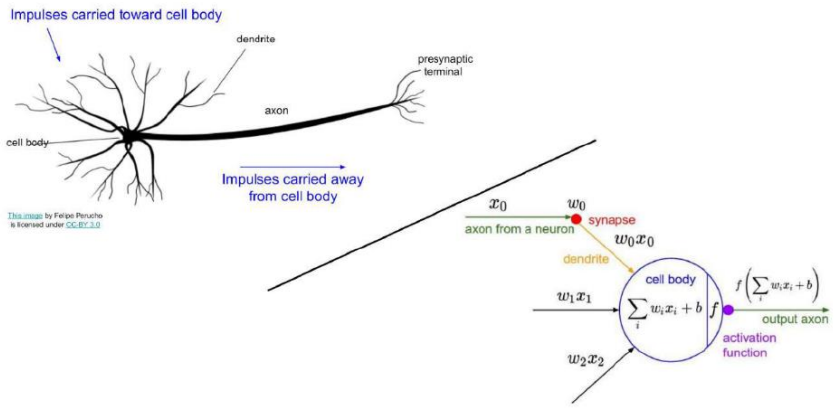


Figure 1 – How artificial Neurons works.

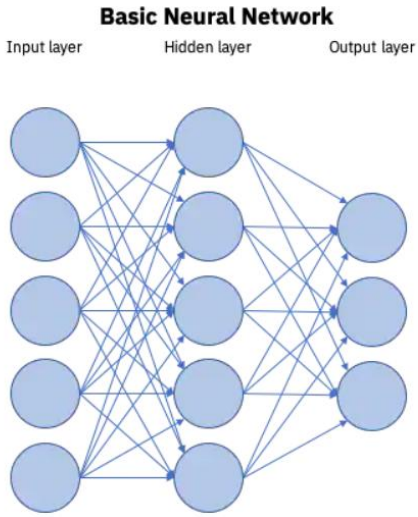


Figure 2 – Structure of a Neural Network, where each dot is a neuron.

The network is defined by its feedforward equations:

$$z_0 = x$$

$$z_l = g_{l-1}(z_{l-1}) * w_l + b_l \qquad l = 1, \dots, L \tag{1}$$

$$y = z_L$$

where $z_l \in R^{n_l}$ is the row vector containing the n_l pre-activation values, also called *units* or *neurons*, in layer l. Figure 3 illustrates a feedforward network with $L = 3$ and $n = n_0 = 3, n_1 = 5, n_2 = 3, n_3 = 1$, together with backpropagation.

In machine learning, backpropagation is a widely used algorithm for training feedforward neural networks. The backpropagation algorithm works by computing the gradient of the loss function with respect to each weight by the chain rule, computing the gradient one layer at a time, iterating

backward from the last layer to avoid redundant calculations of intermediate terms in the chain rule. In fitting a neural network, backpropagation computes the gradient efficiently and this efficiency makes it feasible to use gradient methods for training multilayer networks, updating weights to minimize loss.

As customary in modern deep learning, the training set is traversed in mini-batches, where the cost function is minimized with the best practice ADAM [2] algorithm. Defining a cost function and its gradient with respect to weights and biases

$$L = \frac{1}{m}\sum_{i=1}^{m}\left(Y_i - \widehat{Y_i}\right)^2$$

$$\mathrm{dL} = \nabla_{(W,b)} L$$

Then ADAM works in a sequence of steps:

$$V_{dL} = \beta_1 V_{dL} + (1-\beta_1)\mathrm{dL}$$

$$S_{dL} = \beta_2 S_{dL} + (1-\beta_2)(\mathrm{dL})^2$$

$$Vcorr_{dL} = \frac{V_{dL}}{(1-\beta_1)^t}$$

$$Scorr_{dL} = \frac{S_{dL}}{(1-\beta_2)^t}$$

$$for\ t = 1, \ldots, Num\ Iterations$$

$$W = W - \searrow \frac{Vcorr_{dL}}{\sqrt{Scorr_{dL}} + \varepsilon}$$

Weights are initialized with TensorFlow's *variance_scaling_initializer*, implementing the particularly effective Xavier/Glorot [3] initialization strategy, a best practice in modern deep learning. A correct initialization is a key ingredient in an effective practical implementation of deep learning.

Feedforward network are efficiently differentiated by backpropagation, which is generally applied to compute the derivatives of some cost function wrt the weights and biases for optimization. For now, we are not interested in those differentials, but in the differentials of the *predicted* value $y = z_L$ wrt the inputs $x = z_0$. Recall that inputs are states and predictions are prices, this suggests a way to evaluate the differentials of the predicted value with respect to the inputs, these differentials are predicted risk sensitivities (*Greeks*), obtained by differentiation of lines in (1), in the reverse order:

$$\bar{z}_L = \bar{y} = 1$$

$$\bar{z}_{l-1} = \left(\bar{z}_l w_l^T\right) o\ g'_{l-1}(z_{l-1}) \qquad l = 1, \ldots, L \tag{2}$$

$$\bar{x} = \bar{z}_0$$

with the adjoint notation $\bar{x} = \frac{dy}{dx}, \bar{z}_l = \frac{dy}{dz_l}, \bar{y} = \frac{dy}{dy} = 1$ and o is the elementwise product.

Notice, the similarity between (1) and (2). In fact, backpropagation defines a second feedforward network with inputs $\bar{y}, z_0, \ldots, z_L$ and output $\bar{x} \in R^n$, where the weights are shared with the first network and the units in the second network are the adjoints of the corresponding units in the original network.

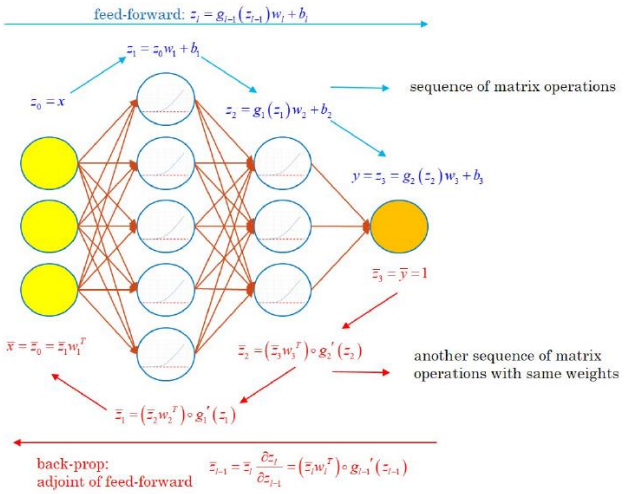


Figure 3 – Feedforward Neural Network with backpropagation

2.1 Twin Networks

We can combine feedforward (1) and backpropagation (2) equations into a single network representation, or twin network, corresponding to the computation of a prediction (approximate price) together with its differentials w.r.t. inputs (approximate risk sensitivities).The first half of the twin network (Figure 4) is the original network, traversed with feedforward induction to predict a value. The second half is computed with the backpropagation equations to predict risk sensitivities. It is the mirror image of the first half, with shared connection weights. A mathematical description of the twin network is simply obtained by concatenation of equations (1) and (2).

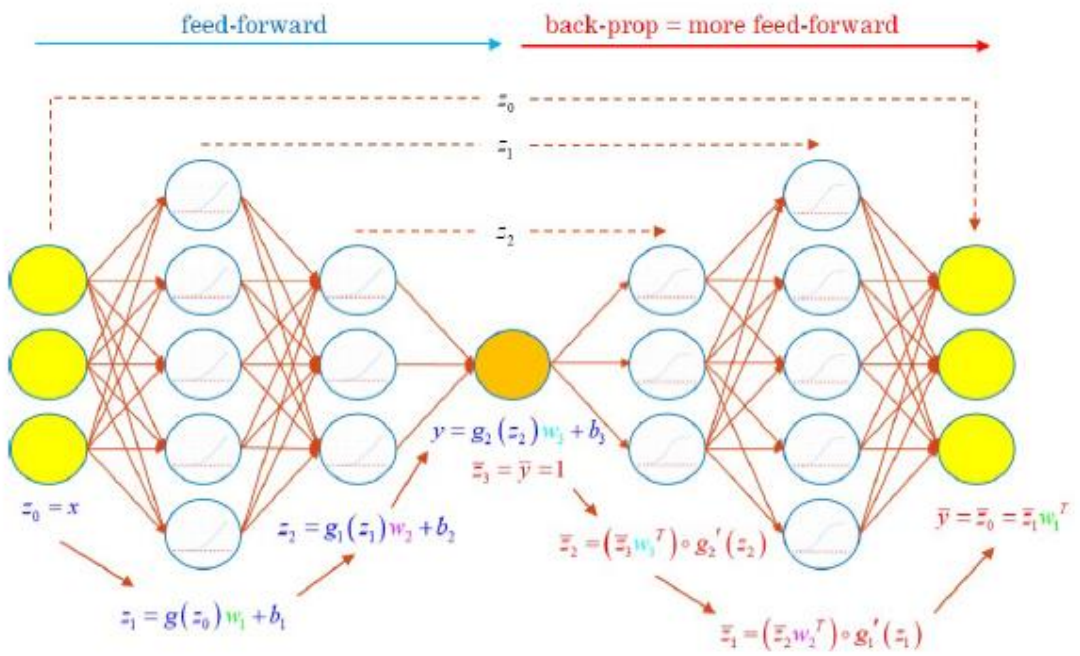


Figure 4 – Twin Network diagram

The evaluation of the twin network returns a predicted value y, and its differentials $\bar{x}$ wrt the $n_o = n$ inputs. The combined computation evaluates a feedforward network of twice the initial depth. Like feedforward induction, backpropagation computes a sequence of matrix by vector products. The twin network, therefore, predicts prices and risk sensitivities for twice the computation complexity of value prediction alone, irrespective of the number of risks. Hence, a trained twin net approximates prices and risk sensitivities, wrt potentially many states, in a particularly efficient manner. Note from (2) that the units of the second half are activated with the differentials g'_l of the original activations g_l. If we are going to backpropagate through the twin network, we need continuous activation throughout. Hence, the initial activation must be C^1, ruling out, e.g. ReLU.

Considering the final release of the *Python* code [4], after testing some activation functions the best results in terms of both pricing and derivatives approximation is given by using: $g_l = softplus(\quad)$ and its derivative $g'_l = sigmoid(\quad)$. In particular,

$$g_l = sofplus = \ln(1 + e^x)$$

$$g'_l = \frac{dg_l}{dx} = \frac{e^x}{(1 + e^x)} = \frac{1}{(1 + e^{-x})} = sigmoid.$$

2.2 Classical training and training with differential labels

The purpose of the twin network is to estimate the correct pricing function $f(x)$ by an approximate function $\hat{f}\left(x, \{w_l, b_l\}_{l=1,\dots,L}\right)$. It learns the optimal weights and biases from an augmented training set $\left(x^{(i)}, y^{(i)}, \bar{x}^{(i)}\right)$, where $\bar{x}^{(i)} = \frac{dy^{(i)}}{dx^{(i)}}$ are the differential labels.

Here, we describe the mechanism of differential training and discuss its effectiveness. As is customary with ML, we stack data in matrices with example in rows and units in columns:

$$X = \begin{bmatrix} x^{(1)} \\ \vdots \\ x^{(m)} \end{bmatrix} \in R^{m\,x\,n} \qquad Y = \begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(m)} \end{bmatrix} \in R^{m} \qquad \bar{X} = \begin{bmatrix} \bar{x}^{(1)} \\ \vdots \\ \bar{x}^{(m)} \end{bmatrix} \in R^{m\,x\,n}$$

Notice, the equations (1) and (2) identically apply to matrices and row vectors. Hence, the evaluation of the twin network computes the matrices:

$$Z_l = \begin{bmatrix} z_l^{(1)} \\ \vdots \\ z_l^{(m)} \end{bmatrix} \in R^{m\,x\,n_l} \qquad Y = \begin{bmatrix} \bar{z}_l^{(1)} \\ \vdots \\ \bar{z}_l^{(m)} \end{bmatrix} \in R^{m\,x\,n_l}$$

respectively in the first and second half of its structure. Training consists in finding weights and biases minimizing some loss function $L: \{w_l, b_l\}_{l=1,\dots,L} = argmin\, L\left(\{w_l, b_l\}_{l=1,\dots,L}\right)$.

Let us first recall classical deep learning. We have seen that the approximation obtained by global minimization of the MSE converges to the correct pricing function, hence, referring to ADAM optimizer in section 2:

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

The second part of the network does not affect cost, hence, training is performed by backpropagation through the standard feedforward network alone. Let us change gear and train with differentials $\bar{x}^{(i)}$ instead of payoff $y^{(i)}$, by minimization of the MSE (here in after denoted $\overline{MSE}$) between the differential labels and predicted differentials:

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

Here we must evaluate the twin network in full to compute $\bar{Z}_0$, effectively doubling the cost of training. Gradient-based methods, like ADAM, minimize $\overline{MSE}$ by backpropagation through the twin network, effectively accumulating second order differentials in its second half. A deep learning framework, like *TensorFlow*, performs this computation seamlessly. As we have seen,

the second half of the twin network represent backpropagation, in the end, this is just another sequence of matrix operations, easily differentiated by another round of backpropagation, carried out silently, behind the scenes. *TensorFlow*, by implementation, automatically invokes the necessary operations, evaluating the feedforward network when minimizing MSE and the twin network when minimizing $\overline{MSE}$. Let us now discuss what it *means* to train approximations by minimization of the $\overline{MSE}$ between pathwise differentials $\bar{x}^{(i)} = \frac{dy^{(i)}}{dx^{(i)}}$ and predicted risk $\frac{d\hat{f}(x^{(i)})}{dx^{(i)}}$. Given appropriate smoothing[1], expectation and differentiation commute so the (true) risk sensitivities are expectation of pathwise differentials:

$$\frac{df(x)}{dx} = \frac{dE[Y|X=x]}{dx} = E\left[\frac{dY}{dX}\middle|X=x\right]$$

It follows that pathwise differentials are unbiased estimates of risk sensitivities, and approximations trained by minimization of the $\overline{MSE}$ converges (modulo finite capacity bias) to a function with correct differentials, hence, the right pricing function, modulo an additive constant. Therefore, we can choose to train by minimization of value or derivative errors, and converge near the correct pricing function all the same. This consideration is, however, an asymptotic one. Training with differentials converges near the same approximation, but it converges much faster, and as seen in numerical results of chapter 4. The main reasons are the following:

- **The effective size of the dataset is much larger** evidently, with m training examples we have mn differentials (n being the dimensions of the input $x^{(i)}$). With AAD, we effectively simulate a much larger dataset for a minimal additional cost, especially in high dimensions (see Bachelier model).
- **The neural nets pick up the *shape* of the pricing function** learning from slopes rather than points, resulting in much more stable and potent learning, even with few examples.
- **The neural approximator learns to produce correct Greeks** by construction, not only correct values. By learning the correct shape, the ML approximation also correctly orders values in different scenarios, which is critical in applications like value at risk (V@R) or expected loss (EL), including for FRTB.

The best numerical results are obtained by combining values and derivatives errors in the cost function:

$$L\left(\{w_l, b_l\}_{l=1,\dots,L}\right) = \alpha * MSE + \beta * \overline{MSE}$$

[1] Pathwise differentials of discontinuous payoff like barrier or digitals are not well defined, and it follows that the risk sensitivities of these instruments cannot be reliably computed with Monte-Carlo, with AAD or otherwise. This is well known problem in the industry, generally resolved by *smoothing,* i.e. the approximation of discontinuous cash flows with close continuous ones, like soft barrier in place of hard barriers.

which is the one implemented in the demonstration notebook. Notice the similarity with classical regularization of the form $L = MSE + \lambda\, penalty$. Ridge (Tikhonov) and Lasso regularizations impose a penalty for large weights (respectively in L^2 and L^1 metrics), effectively preventing overfitting small datasets by stopping attempts to fit noisy labels. Differential training also stops attempts to fit noisy labels, with a penalty for wrong differentials. It is therefore, a form of regularization, but a different kind. It does not introduce bias, since, as it is shown in section 4, training on differentials alone converges to the correct approximations too. We computed the coefficients α and β for balancing cost between values and derivatives in a straightforward manner:

$$\alpha = \frac{1}{1 + \lambda * n} \quad and \quad \beta = \frac{\lambda * n}{1 + \lambda * n}$$

where n is the number of inputs, so an error on a derivative has a weight similar to a value error, and λ is a hyperparameter left to 1, safe for debugging.

2.3 Data Normalization

The practical performance of neural networks strongly depends on implementation details, like weights initialization and optimization. Another crucial practicality is the normalization of training data. We refer to deep learning textbooks for a discussion of the importance of normalization. One reason is that we need hyperparameters like the learning rate schedule to remain constant over datasets. If notional was to be increased by factor 1 million all things equal, gradients would be multiplied by 1 million too and learning rates would have to be divided by 1 million to keep things similar. Normalizing data avoids manual tinkering of hyperparameters for different datasets. All the examples in the paper: the Gaussian basket, the plain vanilla and the barrier option, were all approximated with exact the same hyperparameters. This is only possible with normalized datasets. We implement a basic normalization strategy, where the training inputs and labels are normalized by mean and standard deviation, with differentials normalized accordingly. The differential weights in the cost function γ_j divide costs by the norm of the normalized differentials, keeping similar the magnitude of all the components of the cost.

$$L\left(\{w_l, b_l\}_{l=1,\dots,L}\right) = \alpha * MSE + \beta * \overline{MSE} = \alpha * c_{val} + \beta * c_{diff}$$

Where c_{val} is the classic mean square error (MSE) of predictions to labels and c_{diff} is the cost of wrong derivatives:

$$c_{diff} = \frac{\sum_{inputs\ j} {\gamma_j}^2 * \overline{MSE_j}}{m}$$

Where $\overline{MSE_j}$ is the mean squared error of derivatives to input j and the weights γ_j normalize derivatives so all the components of the cost have similar magnitudes.

Note that the prediction of values and derivatives must be adjusted accordingly: prediction inputs must be normalized, and resulting predictions must be 'un-normalized'. We implement basic preprocessing:

$$\tilde{Y}^{(i)} = \frac{Y^{(i)} - \mu_Y}{\sigma_Y} \quad and \quad \tilde{X}^{(i)} = \frac{X^{(i)} - \mu_X}{\sigma_X}$$

$$\mu_Y = \frac{1}{m}\sum_{i=1}^{m} Y^{(i)} \quad and \quad \mu_X = \frac{1}{m}\sum_{i=1}^{m} X^{(i)}$$

similarly for standard deviation σ_Y and σ_X. The differentials computed by the prediction model (e.g. the twin network) are:

$$\frac{d\tilde{Y}}{d\tilde{X}} = \frac{\sigma_X}{\sigma_Y}\frac{dY}{dX}$$

hence, we adjust differential labels accordingly:

$$\tilde{Z}^{(i)} = \frac{\sigma_X}{\sigma_Y} Z^{(i)}$$

therefore, the weights γ_j becomes:

$$\gamma_j = \frac{1}{||\tilde{Z}_j||^2}$$

We implements a simple feedforward network with 20 neurons per layer and 4 hidden layers.

3 Introduction to Financial Derivatives

This section aims to introduce the lector to financial derivatives. After a brief discussion about stochastic process, the basic statements of the Black-Scholes-Merton formula are presented, together with analytical results for a European call option (3.1). In the next two sections, the formulas of Bachelier's model for pricing a basket option (3.2) and the analytical approximation of a Barrier option (3.3) were shown. In the last section 3.4, a brief introduction on derivatives risk managements have been presented. Main relevant proofs could be found in the Annex.

3.1 Stochastic process and Black-Scholes-Merton formula

Any variable whose value uncertainly changes over time is said to follow a *stochastic process*. A Markov process is a particular type of stochastic process where only the current value of a variable is relevant for predicting the future. The history of the variable and the way that the present has emerged from the past are irrelevant. Stock prices are usually assumed to follow a Markov process. Suppose that the price of a stock is $100 now. If the stock price follows a Markov process, our predictions for the future should be unaffected by the price one week ago, one month ago, or one year ago. The only relevant piece of information is that the price is now $100. Predictions for the future are uncertain and must be expressed in terms of probability distributions. The Markov property implies that the probability distribution of the price at any particular future time is not dependent on the particular path followed by the price in the past; this is consistent with the weak form of market efficiency. This states that the present price of a stock impounds all the information contained in a record of past prices. If the weak form of market efficiency were not true, technical analysts could make above-average returns by interpreting charts of the history of stock prices. There is very little evidence that they are in fact able to do this. A particular type of Markov continuous-time stochastic process is the Wiener process. Defined as:

$$W(t)\ for\ \ t \geq 0\ such\ that\ W(t+s) - W(t) \sim N\ (0, s), s > 0$$

it is a stochastic process with independent and normally distributed increments, with mean 0 and a variance s. It has been used in physics to describe the motion of a particle that is subject to a large number of small molecular shocks and is sometimes referred to as Brownian motion. A *generalized Wiener process* is then defined as:

$$dS_t = \mu * dt + \sigma * dW_t$$

A further type of stochastic process, known as an Ito's process, can be defined. This is a generalized Wiener process in which the parameters μ and σ are functions of the value of the underlying variable S and time t. An Ito's process can therefore be written as:

$$dS_t = \mu(S_t\,, t) * dt + \sigma(S_t\,, t) * dW_t$$

A particular case is the Geometric Brownian Motion (hereinafter GBM):

$$dS_t = \mu * S_t * dt + \sigma * S_t * dW_t$$

The equation above is the most widely used model of stock price behavior. The variable μ is the stock's expected rate of return. The variable σ is the volatility of the stock price. The variable σ^2 is referred to as its variance rate. The model in the equation represents the stock price process in the real world. In a risk-neutral world, μ equals the risk-free rate r. Using Ito's lemma in deriving the process followed by $\ln(S)$ when S follows a GBM we obtained that:

$$\ln(S_T) - \ln(S_0) \sim N\left(\left(\mu - \frac{\sigma^2}{2}\right)T, \sigma^2 T\right)$$

In the early 1970s, Fischer Black, Myron Scholes, and Robert Merton achieved a major breakthrough in the pricing of European stock options. This was the development of what has become known as the Black–Scholes–Merton model. The model has had a huge influence on the way that traders price and hedge derivatives. In 1997, the importance of the model was recognized when Robert Merton and Myron Scholes were awarded the Nobel prize for economics. Sadly, Fischer Black died in 1995; otherwise, he too would undoubtedly have been one of the recipients of this prize.

An European call option is an option that gives the holder the right to buy the underlying asset by a certain date T for a certain price K defined here and now at time $t = 0$ and its payoff is:

$$f(S_T, T) = \max\{S_T - K, 0\}$$

The Black-Scholes-Merton formula is, in risk neutral world:

$$C_t = S_t * CDF(d_1) - K * e^{-r(T-t)} * CDF(d_2)$$

Where CDF is the cumulative distribution function for the standard normal and:

$$d_1 = \frac{\ln\left({}^{S_t}/_{K}\right) + \left(r + \frac{\sigma^2}{2}\right)(T-t)}{\sigma\sqrt{T-t}}$$

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

A sketch of the proof could be found in the Annex.

3.2 Basket Option

The second example is a Gaussian Basket option, an option written on a portfolio of n correlated assets with the assumption of a lognormal forward process:

$$dF_t = \sigma * dW_t$$

Applying Ito's lemma to $f(t, F_t)$ we derive the process in terms of S_t:

$$dS_t = \mu * S_t * dt + \sigma * e^{\mu(t-T)} * dW_t$$

As said before, S_t is a portfolio composed of n assets and its dynamics depends on correlations between assets at a specific time step t. Practical implementation has been treated in chapter 4.2. Under the forward model, the call option price with drift is obtained from the standard Bachelier option price, in a risk-free world, like (see. Annex):

$$C_t = e^{-r(T-t)}\sigma\sqrt{T-t}\,[D * CDF(D) + PDF(D)]$$

Where CDF is the cumulative distribution function for the standard normal, PDF is its probability distribution function and:

$$D = \frac{S_t e^{r(T-t)} - K}{\sigma\sqrt{T-t}}$$

3.3 Barrier Option

Another kind of option was also tested, the European Barrier Option. Barrier options are options where the payoff depends on whether the underlying asset's price reaches a certain level during a certain period. Many different types of barrier options regularly trade in the over-the-counter market. They are attractive to some market participants because they are less expensive than the corresponding regular options. These barrier options can be classified as either knock-out options or knock-in options. A knock-out option ceases to exist when the underlying asset price reaches a certain barrier; a knock-in option comes into existence only when the underlying asset price reaches a barrier. A down-and-out call is one type of knock-out option. It is a regular call option that ceases to exist if the asset price reaches a certain barrier level H. The barrier level is below the initial asset price. We only consider in detail the case where the lower barrier is set below the option's strike price $K > H$. Those kinds of options are particularly interesting for our purpose because of their non-linearity.

Under the usual Black–Scholes assumptions, there is an explicit formula for the fair value of this option. The corresponding knock-in option is a down-and-in call. This is a regular call that comes into existence only if the asset price reaches the barrier level. Referring to [5] the value of a down-and-in call at time zero is:

$$c_{t=0}^{di} = S_0 e^{-rT} * \left(\frac{H}{S_0}\right)^{2\varphi} * CDF(y) - Ke^{-rT}\left(\frac{H}{S_0}\right)^{2\varphi-2} * CDF\left(y - \sigma\sqrt{T}\right)$$

where:

$$\varphi = \frac{r + \frac{\sigma^2}{2}}{\sigma^2}$$

$$y = \frac{\ln\left(\frac{H^2}{S_0 K}\right)}{\sigma\sqrt{T}} + \varphi\sigma\sqrt{T}$$

Because the value of a regular call equals the value of a down-and-in call plus the value of a down-and-out call, the value of a down-and-out call is given by:

$$c_{t=0}^{do} = c - c_{t=0}^{di}$$

In other terms:

$$c_t^{do}(S, T, K, H, r, \sigma) = c_t^{vanilla}(S_t, T, K, r, \sigma) - \left(\frac{H}{S_t}\right)^{2\varphi-2} c_t^{vanilla}\left(\frac{H^2}{S_t}, T, K, r, \sigma\right)$$

A sketch of the proof is presented in the Annex.

3.4 Derivatives Risk Management and Greeks

Considering the *BSM* price of a vanilla European-style option, it depends on five factors:

- Current price S_t of the underlying asset
- Volatility σ
- Time-to-maturity $T - t$
- Risk-free rate r
- Strike price K

For risk management purposes, we should evaluate the sensitivity of the option price with respect to each of these risk factors. These sensitivities are collectively known as the options Greeks, and are essential in hedging and risk management applications.

The option Delta (Δ) is the first order sensitivity of the option price wrt the current price of the underlying asset. For a vanilla call at time t under the usual BSM assumption the delta is:

$$\Delta_{vanilla} = \frac{dc_t^{vanilla}}{dS_t} = CDF(d_1)$$

The delta is given by the CDF of the standard normal distribution, which is a probability. Hence, the call delta is in the interval $[0,1]$. It should be the number of stock shares that the option writer should be long for each call option. The more the option is in-the-money, the closer this number is to 1. Similar formulas have been found analytically also of the Basket Option and for the Down-And-Out option. For now on the Delta is sufficient to test the goodness of the twin-net approach. As said in 2.1 the exercise is to learn from $x^{(i)}$, the spot price sampled on some present or future date $T_1 \geq 0$ called *exposure date*, the labels $y^{(i)}$ that would be the payoff of a call expiring on a later date $T_2 > T_1$, sampled on the same path number i. In this case, the differential labels $\frac{dy^{(i)}}{dx^{(i)}}$ are the *pathwise derivatives* of the payoff at T_2 wrt the state at T_1 on path number i. In the BSM vanilla:

$$\frac{dy^{(i)}}{dx^{(i)}} = \frac{d\left(S_{T_2}^{(i)} - K\right)^+}{dS_{T_1}^{(i)}} = \frac{d\left(S_{T_2}^{(i)} - K\right)^+}{dS_{T_2}^{(i)}} \frac{dS_{T_2}^{(i)}}{dS_{T_1}^{(i)}} = 1_{\left\{S_{T_2}^{(i)} > K\right\}} \frac{S_{T_2}^{(i)}}{S_{T_1}^{(i)}}$$

This simple exercise exhibits some general properties of *pathwise differentials*.

First, we computed the *BSM pathwise* derivative analytically, with an application of the chain rule. The resulting formula is computationally efficient: the derivative is computed together with the payoff along the path, there is no need to regenerate the path, contrarily to e.g. differentiation by finite difference. This efficacy is not limited to European calls in Black and Scholes: *pathwise differentials* are always efficiently computable by a systematic application of the chain rule, also known as adjoint differentiation or *AD*. Furthermore, automated implementations of *AD*, or *AAD*, perform those computations by themselves, behind the scenes.

Secondly, $\frac{dY}{dX}$ is a T_2 measurable random variable, and its T_1 expectation is $CDF(d_1)$, the Black and Scholes delta. This property too is general: assuming appropriate smoothing of discontinuous cash-flows, expectation and differentiation commute so risk sensitivities are expectations of pathwise differentials. Turning it upside down, pathwise differentials are unbiased (noisy) estimates of ground truth Greeks. Therefore, we can compute pathwise differentials efficiently and use them for training as unbiased estimates of ground truth risks, irrespective of the transaction or trading book, and irrespective of the stochastic simulation model. Learning from ground truth labels is slow, but the learned function is reusable in many contexts *AAD* is closely related to backpropagation, which powers modern deep learning and has largely contributed to its recent success.

4 Numerical results

Let us now review some numerical results and compare the performance of differential and the simplest feedforward network. We picked three examples from relevant textbooks and real-world situations, where neural networks learn pricing and risk approximations from small datasets. We kept neural architecture constant in all the examples, with four hidden layers of 20 softplus-activated units. We train neural networks on mini-batches of normalized data, with the ADAM optimizer and we applied the recent one-cycle learning rate schedule of Leslie Smith [7] and found that it considerably accelerates and stabilizes the training of neural networks. 100 epochs are more than sufficient in most practical cases.

4.1 BSM

The first example is the European style option, the task is to learn the pricing function of a $1y$ call with strike $K = 110$ the initial asset price $S_0 = 100$, volatility=20% and risk-free rate $r = 0$. We allow raising volatility by a factor $volMult = 1{,}5$ between now ($t = 0$) and the pricing date T_1 to get more samples on the wings and better learn asymptotics. We generate two antithetic paths from time T_1 until time T_2 in order to reduce the variance of the Monte-Carlo simulation. The code below should be self-explanatory.

```
def __init__(self,
             vol=0.2,
             T1=1,
             T2=2,
             K=1.10,
             volMult=1.5):

    self.spot = 1
    self.vol = vol
    self.T1 = T1
    self.T2 = T2
    self.K = K
    self.volMult = volMult

# training set: returns S1 (mx1), C2 (mx1) and dC2/dS1 (mx1)
def trainingSet(self, m, anti=True, seed=None):

    np.random.seed(seed)

    # 2 sets of normal returns
    returns = np.random.normal(size=[m, 2])

    # SDE
    vol0 = self.vol * self.volMult
    R1 = np.exp(-0.5*vol0*vol0*self.T1 + vol0*np.sqrt(self.T1)*returns[:,0])
    R2 = np.exp(-0.5*self.vol*self.vol*(self.T2-self.T1) \
               + self.vol*np.sqrt(self.T2-self.T1)*returns[:,1])
    S1 = self.spot * R1
    S2 = S1 * R2

    # payoff
    pay = np.maximum(0, S2 - self.K)

    # two antithetic paths
    if anti:

        R2a = np.exp(-0.5*self.vol*self.vol*(self.T2-self.T1) \
                - self.vol*np.sqrt(self.T2-self.T1)*returns[:,1])
        S2a = S1 * R2a
        paya = np.maximum(0, S2a - self.K)

        X = S1
        Y = 0.5 * (pay + paya)

        # differentials
        Z1 =  np.where(S2 > self.K, R2, 0.0).reshape((-1,1))
        Z2 =  np.where(S2a > self.K, R2a, 0.0).reshape((-1,1))
        Z = 0.5 * (Z1 + Z2)

    # standard
    else:

        X = S1
        Y = pay

        # differentials
        Z =  np.where(S2 > self.K, R2, 0.0).reshape((-1,1))

    return X.reshape([-1,1]), Y.reshape([-1,1]), Z.reshape([-1,1])
```

Figure 5 – Monte-Carlo simulation for European Option price and delta

We have trained neural networks on 1024 (1k) and 65536 (64k) paths and predicted values and derivatives on 1024 independent test scenarios, with initial spot values on the horizontal axis and option prices/delta on the vertical axis compared with the correct results computed with Black-Scholes-Merton formula. The twin network with 1k examples performs better than the classical net with 64k examples for values, and a lot better for derivatives. In particular, it learned that the option price and deltas are a fixed function of the underlying asset price, as evidenced by the thinness of the approximation curve. The classical network doesn't learn this property well, even with 64k examples.

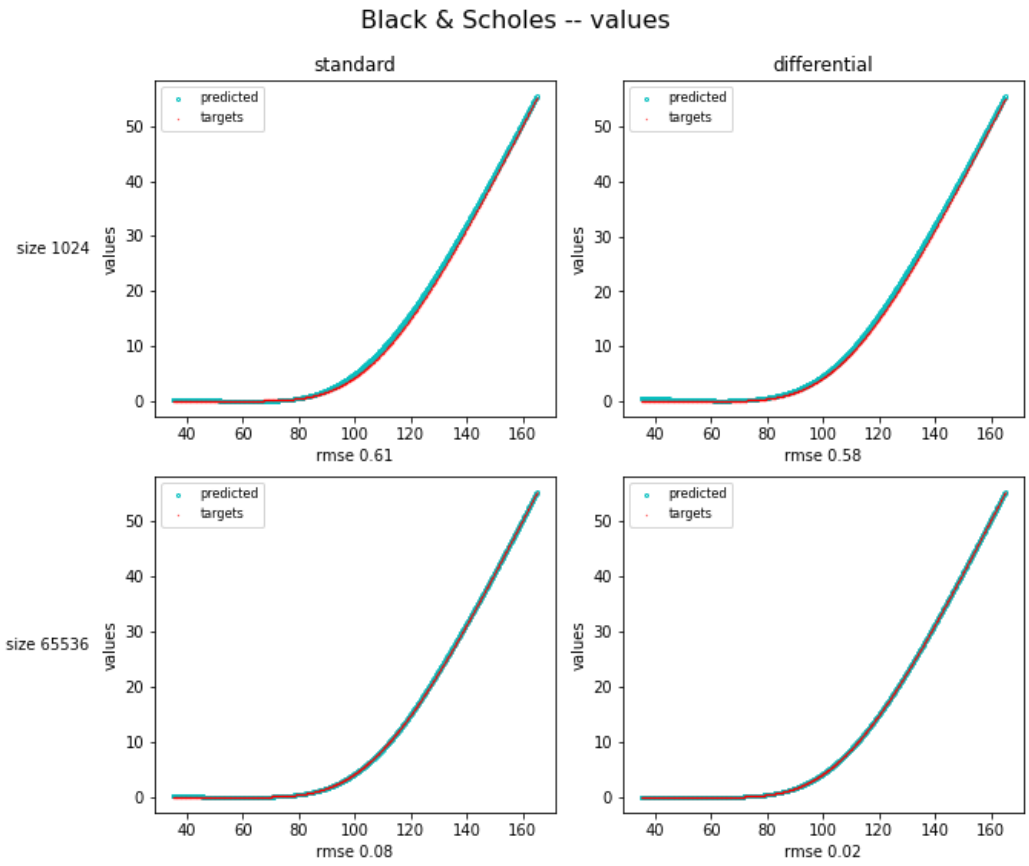


Figure 6 – Price of an European style call option using the standard Neural Network and the Twin network.

Networks perform better, at least with respect to the options prices, when the size of the training net is increased, this suggests that we avoid overfitting. Looking at the delta in the figure below it is clear that the classical neural network is not sufficient for risk management purposes even if it is approximating well the option price. The twin net, instead, results in well-behaved deltas together with a better approximation of the option price. This Neural Network works as fast as the classical one and so we have improved results without additional effort in terms of computational complexity.

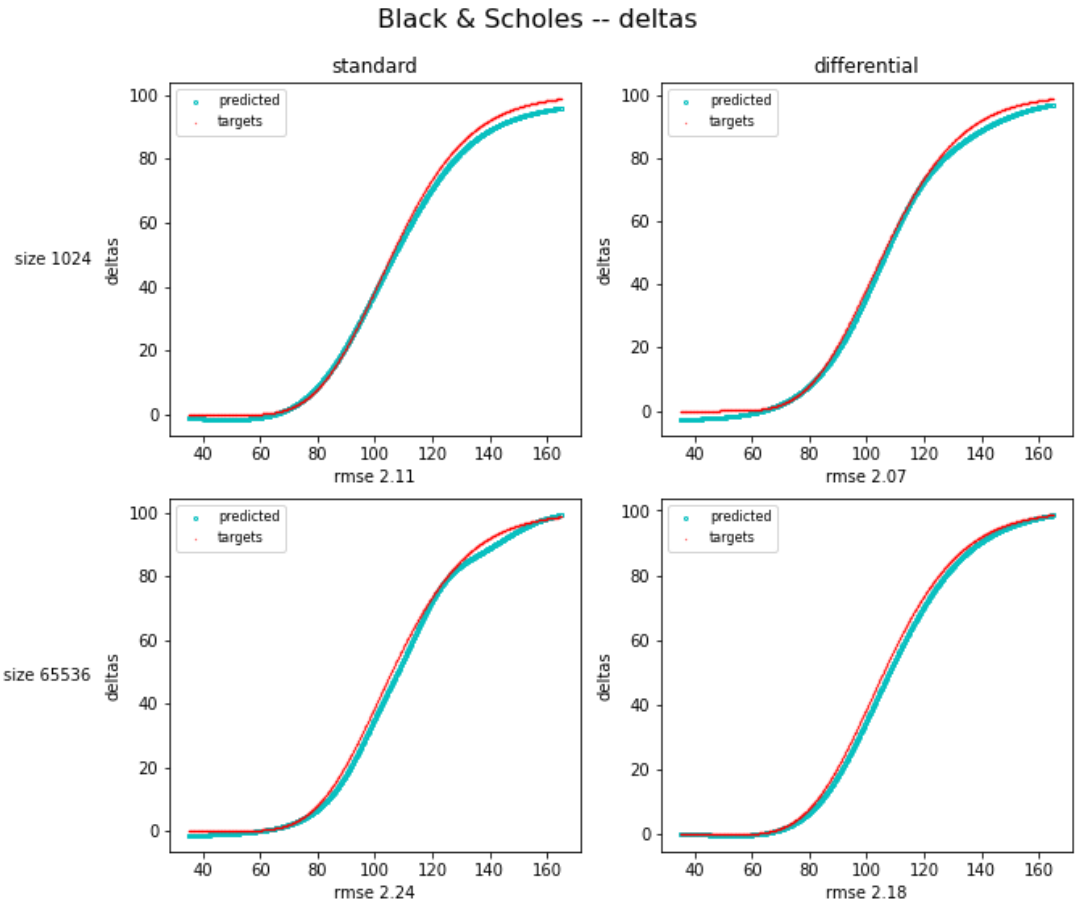


Figure 7 - Delta of an European style call option using the standard Neural Network and the Twin network.

4.2 Bachelier

The recent success of deep learning is largely due to its ability to break the long-standing curse of dimensionality that breaks classic regression models in high dimension. Contrary to classic linear models, neural networks do not regress on a fixed set of basis functions. They learn from data a relevant regression basis in their hidden layers, embedding a powerful dimension reduction capability in their structure. This is why deep learning succeeded in such high dimensional tasks as computer vision, where the dimension of the inputs is the number of pixels in a picture. Convolutional nets effectively learn the low dimensional features that matter to e.g. image recognition, something a classic regression model could not do. Differential machine learning also shines in high dimensions, where differential labels help identify relevant features more effectively. In fact, the additional performance from differential training exponentially increases with dimensions. This is why twin networks are particularly effective for learning values of complex transactions or trading books as functions of a high dimensional state.

To illustrate this ability in a simple context, we extend the Black & Scholes example to a basket option written on n correlated stocks, with $n = 1,7\ and\ 20$. In place of the Black & Scholes model, the stocks are simulated in Bachelier's Gaussian model, where the true price of the basket

option is known in closed form, and given by Bachelier's formula applied to the basket at T_1. Therefore, we can monitor the performance of our approximators by comparison to the correct, analytic prices and deltas. In addition, the example is particularly interesting because the price is really a non-linear function of the one-dimensional basket. We expect the machine to learn that from data, and twin networks achieve this a lot better than feedforward networks.

As in the BSM case, the task is to learn the pricing function of a $1y$ call with strike $K = 110$ the initial asset prices $S_0^i = 100 \ \forall\, i = 1, \dots, n$, volatility-Basket=20% and risk-free rate $r = 0$. We allow raising volatility by a factor $volMult = 1{,}5$ between now ($t = 0$) and the pricing date T_1 to get more samples on the wings and better learn asymptotics. Correlations, assets volatilities and basket weights are re-generated randomly on every run, allowing verifying performance in multiple configurations. We normalize these volatilities with respect to a given volatility of the basket and then we apply the Cholesky factorization for time simulation. The idea is that we generate 2 sets of standard normal returns of dimension n and then we correlate it. Two antithetic paths were generated as a variance reduction method. The code below should be self-explanatory.

```
class Bachelier:

    def __init__(self,
                 n,
                 T1=1,
                 T2=2,
                 K=1.10,
                 volMult=1.5):

        self.n = n
        self.T1 = T1
        self.T2 = T2
        self.K = K
        self.volMult = volMult

    # training set: returns S1 (mxn), C2 (mx1) and dC2/dS1 (mxn)
    def trainingSet(self, m, anti=True, seed=None, bktVol=0.2):

        np.random.seed(seed)

        # spots all currently 1, without loss of generality
        self.S0 = np.repeat(1., self.n)
        # random correl
        self.corr = genCorrel(self.n)

        # random weights
        self.a = np.random.uniform(low=1., high=10., size=self.n)
        self.a /= np.sum(self.a)
        # random vols
        vols = np.random.uniform(low=5., high = 50., size = self.n)
        # normalize vols for a given volatility of basket,
        # helps with charts without loss of generality
        avols = (self.a * vols).reshape((-1,1))
        v = np.sqrt(np.linalg.multi_dot([avols.T, self.corr, avols]).reshape(1))
        self.vols = vols * bktVol / v
        self.bktVol = bktVol

        # Choleski etc. for simulation
        diagv = np.diag(self.vols)
        self.cov = np.linalg.multi_dot([diagv, self.corr, diagv])
        self.chol = np.linalg.cholesky(self.cov) * np.sqrt(self.T2 - self.T1)
        # increase vols for simulation of X so we have more samples in the wings
        self.chol0 = self.chol * self.volMult * np.sqrt(self.T1 / (self.T2 - self.T1))
        # simulations
        normals = np.random.normal(size=[2, m, self.n])
        inc0 = normals[0, :, :] @ self.chol0.T
        inc1 = normals[1, :, :] @ self.chol.T

        S1 = self.S0 + inc0

        S2 = S1 + inc1
        bkt2 = np.dot(S2, self.a)
        pay = np.maximum(0, bkt2 - self.K)

        # two antithetic paths
        if anti:

            S2a = S1 - inc1
            bkt2a = np.dot(S2a, self.a)
            paya = np.maximum(0, bkt2a - self.K)

            X = S1
            Y = 0.5 * (pay + paya)

            # differentials
            Z1 =  np.where(bkt2 > self.K, 1.0, 0.0).reshape((-1,1)) * self.a.reshape((1,-1))
            Z2 =  np.where(bkt2a > self.K, 1.0, 0.0).reshape((-1,1)) * self.a.reshape((1,-1))
            Z = 0.5 * (Z1 + Z2)

        # standard
        else:

            X = S1
            Y = pay

            # differentials
            Z =  np.where(bkt2 > self.K, 1.0, 0.0).reshape((-1,1)) * self.a.reshape((1,-1))

        return X, Y.reshape(-1,1), Z
```

Figure 8 – Monte-Carlo simulation for Bachelier Basket Option price and delta

The training set sizes are 4096, 8192 and 16384 except for the case in which the dimension of the basket is equal to 1. In next figures, prices and delta of the Basket Option were shown for different basket and training size.

- $Basket\ dimension = 1$

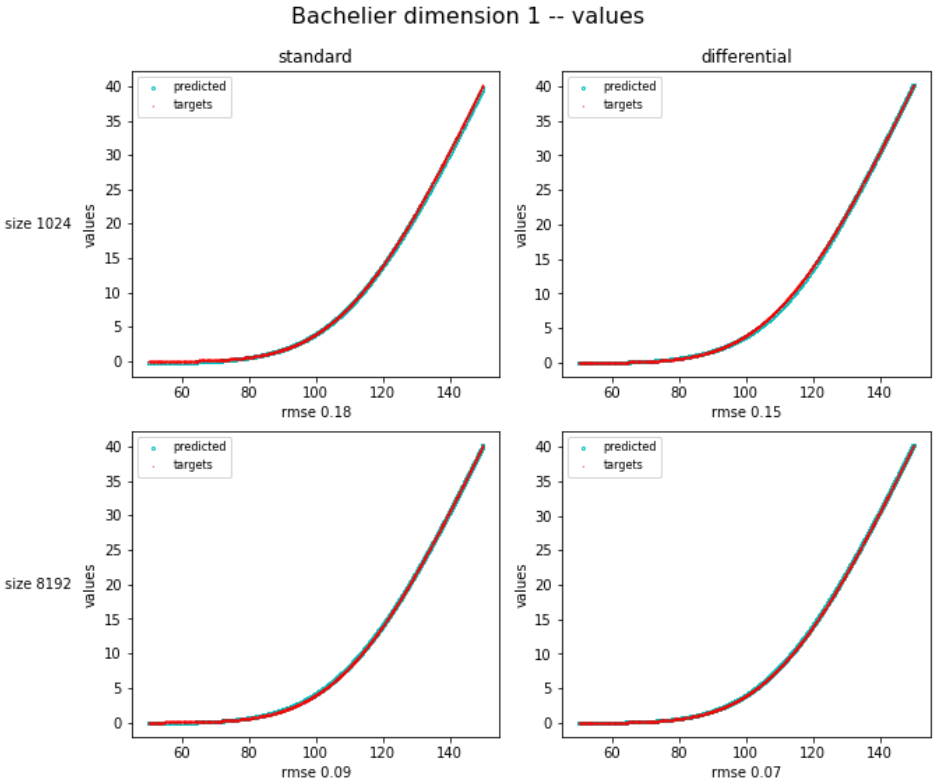


Figure 9 - Price of a Basket call option using the standard Neural Network and the Twin network.

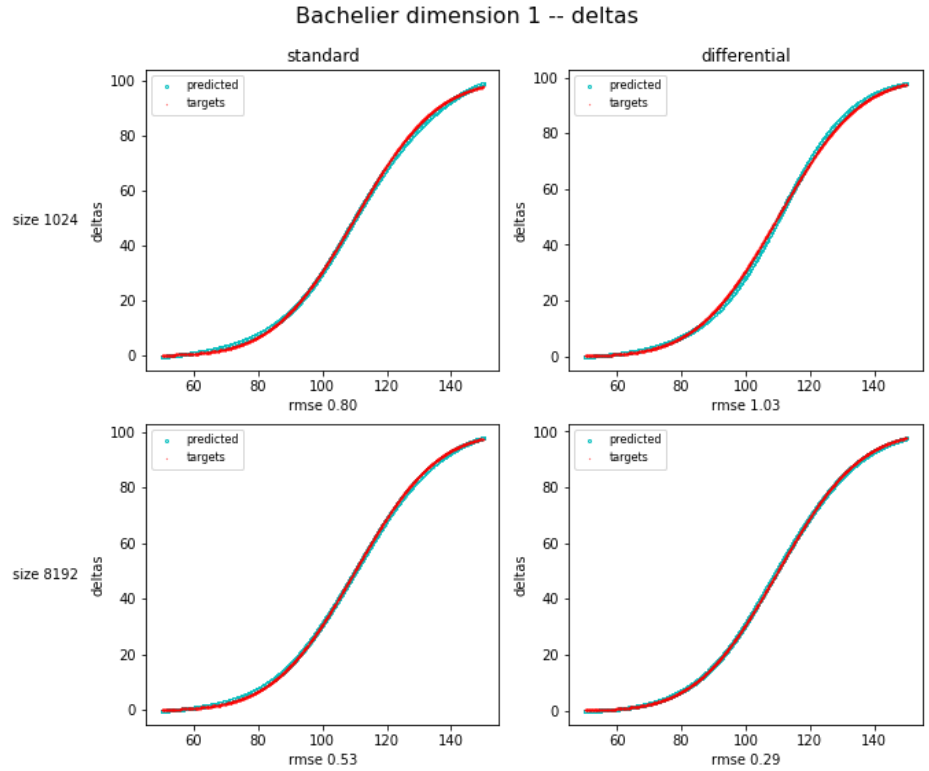


Figure 10 - Delta of a Basket call option using the standard Neural Network and the Twin network

Moving on to dimension 7, we display predictions and correct values as a function of the underlying basket at T_1. The thickness of the plot measures the ability to learn from data that the value is a fixed function of the current underlying basket. A thin curve reflects that this property is correctly learned. A thick line means that the approximator predicts different values for different sets of stocks corresponding to the same basket, hence, failing to learn the pricing function correctly.

- $Basket\ dimension = 7$

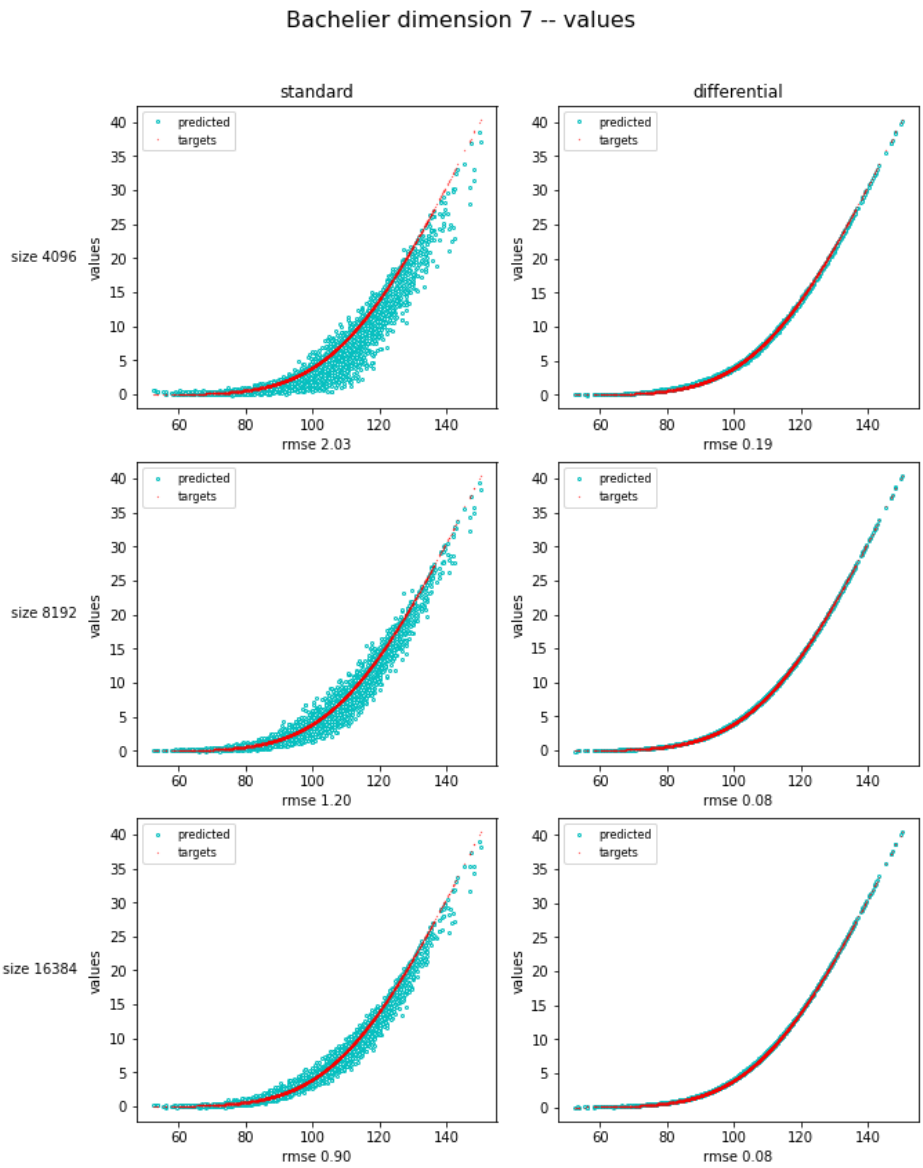


Figure 11 - Price of a Basket call option using the standard Neural Network and the Twin network.

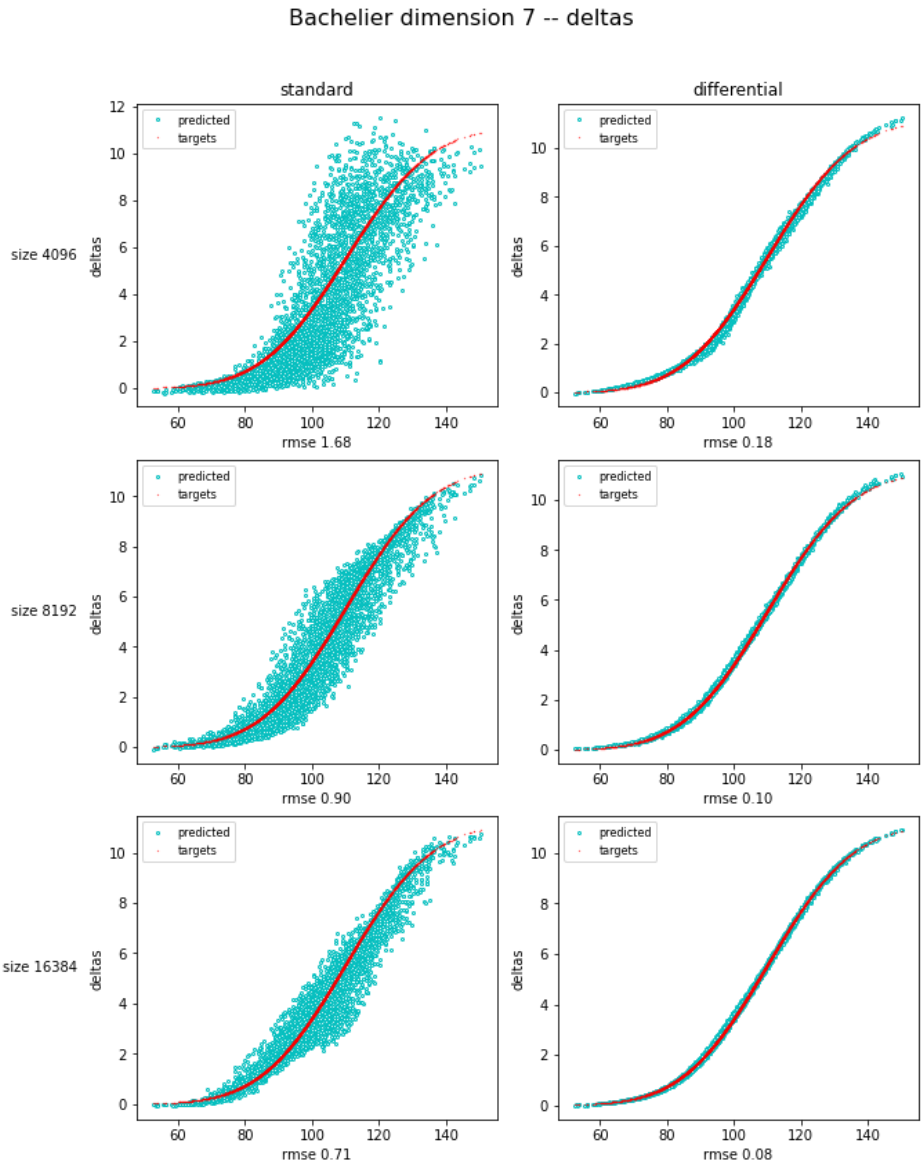


Figure 12 - Delta of a Basket call option using the standard Neural Network and the Twin network

Both networks converge to a correct approximation, but differential training gets their orders of magnitude faster, and especially outperforms on smaller training sets. This is what makes it so particularly relevant for the risk management of financial Derivatives. Below, we test dimension 20. Notice that learning time is virtually unaffected by dimension, and that the performance of the twin network is resilient to high dimensionality, where the standard network starts to struggle.

- $Basket\ dimension = 20$

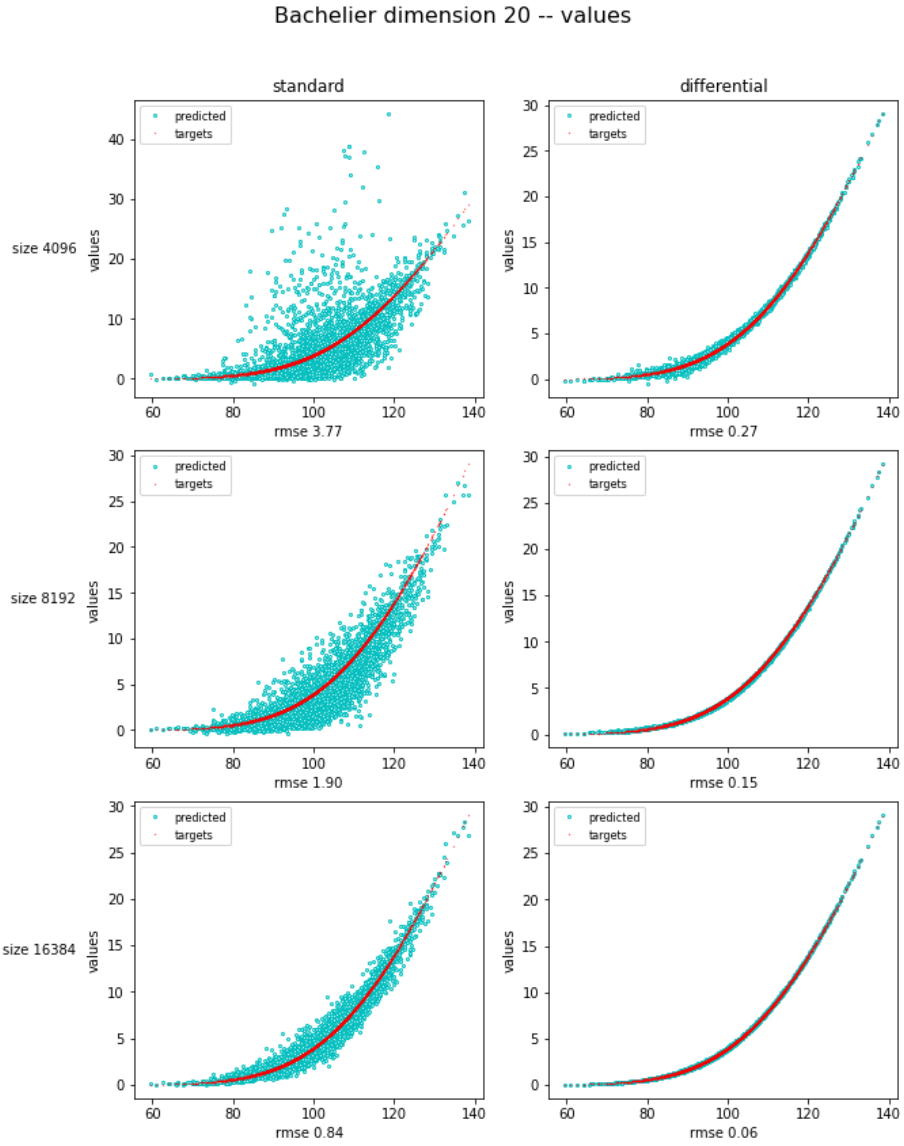


Figure 13 - Price of a Basket call option using the standard Neural Network and the Twin network.

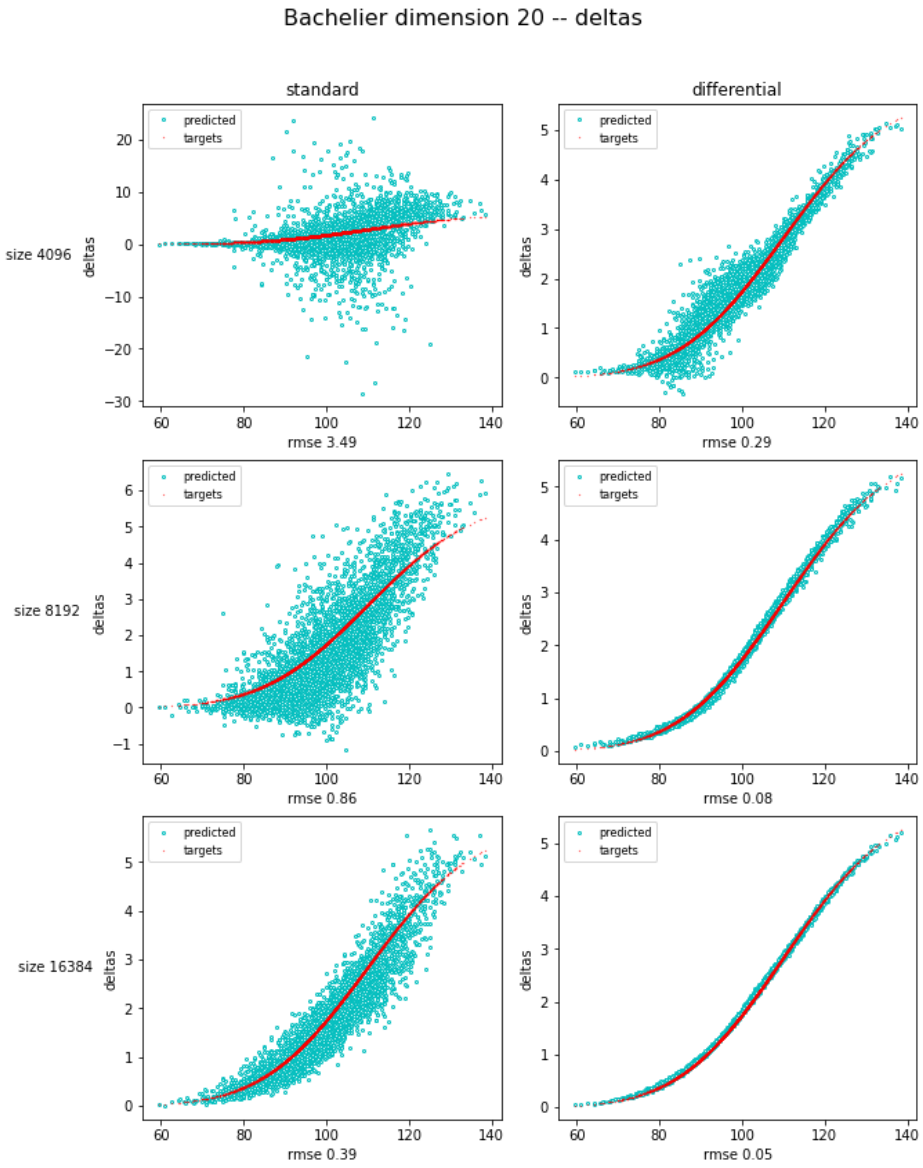


Figura 14 - Delta of a Basket call option using the standard Neural Network and the Twin network

The twin network with 4k examples performs better than the classical net with 16k examples for values, and a lot better for derivatives. In particular, it learned that the option price and deltas are a fixed function of the basket, as evidenced by the thinness of the approximation curve. The classical network doesn't learn this property well, even with 16k examples. It overfits training data and predicts different values or deltas for various scenarios on the seven assets with virtually identical baskets.

4.3 Barrier

The last example is the European style Barrier option, that kind of option is particularly interesting because of its non-linearity. In fact, the payoff is a discontinuous function of the asset price. The example is also relevant because discontinuous payoff produces unstable risks with Monte-Carlo. In order to avoid this problem, the industry developed a simple and surprisingly effective method that consists of replacing the non-linear payoff with closed continuous ones. The benefit of this method is that we work directly on the payoff and no work is required on the model. The problem is that to identify and smooth all discontinuities we need to know exactly how the payoff is calculated.

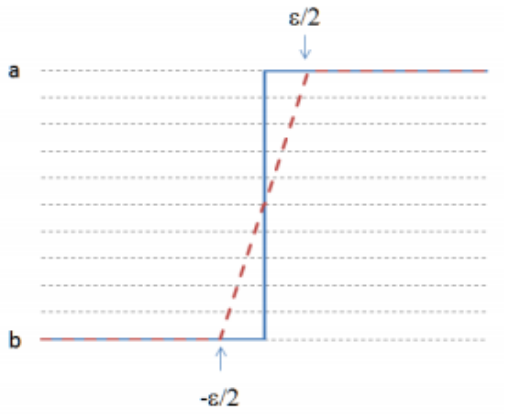


Figure 15 – Example of a smooth barrier

The task is to learn the pricing function of a $1y$ down-and-out with strike $K = 110$ the initial asset price $S_0 = 100$, barrier $H = 80$, volatility=20% and risk-free rate $r = 0$. We allow raising volatility by a factor $volMult = 1{,}5$ between now ($t = 0$) and the pricing date T_1 to get more samples on the wings and better learn asymptotics. Appropriate smoothing is applied for backpropagation. We test the Twin Network in two different scenarios, one with discontinuous payoff and delta and one with the relaxed ones. We want to understand the ability of the network in using the fuzzy logic automatically behind the scenes increasing its efficiency. AAD could not differentiate something that is not differentiable and so in using non-linear payoff the network should be able to approximate it with a continuous function. We use the analytical formula presented in section 3.3 for generating the test dataset. Both networks are trained on 10000 (10k) and 100000 (100k) examples. All the following results are reproduced on the online TensorFlow notebook [4].

- **Hard Barrier**

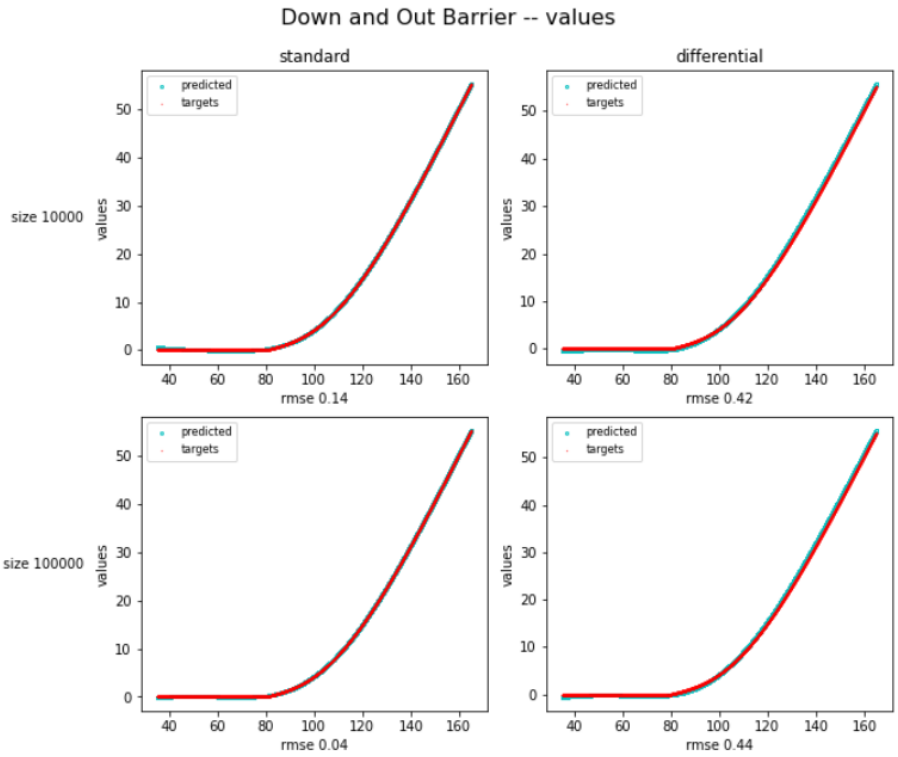


Figure 17 - Price of a Down and Out call option with Hard Barrier using the standard Neural Network and the Twin network.

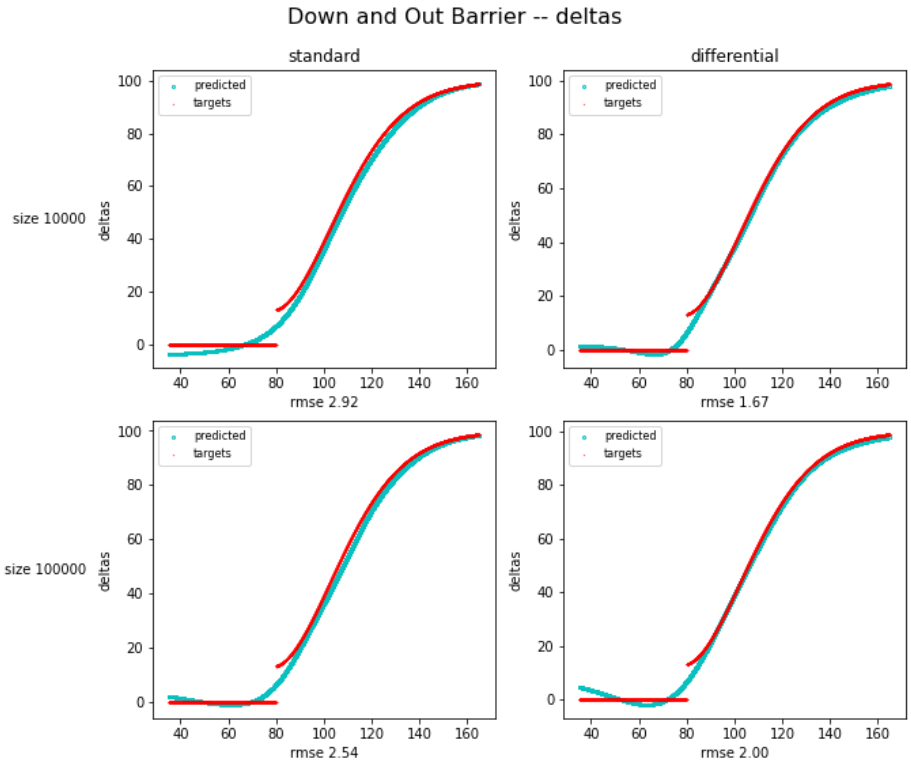


Figure 16 - Delta of a Down and Out call option with Hard Barrier using the standard Neural Network and the Twin network.

- **Soft Barrier**

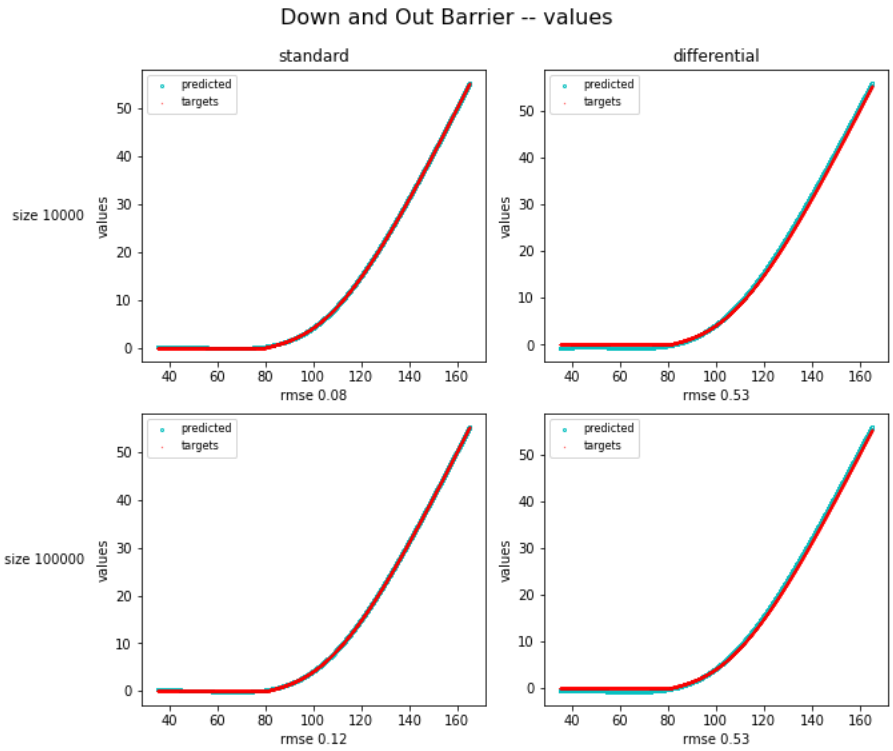


Figure 18 - Price of a Down and Out call option with Soft Barrier using the standard Neural Network and the Twin network

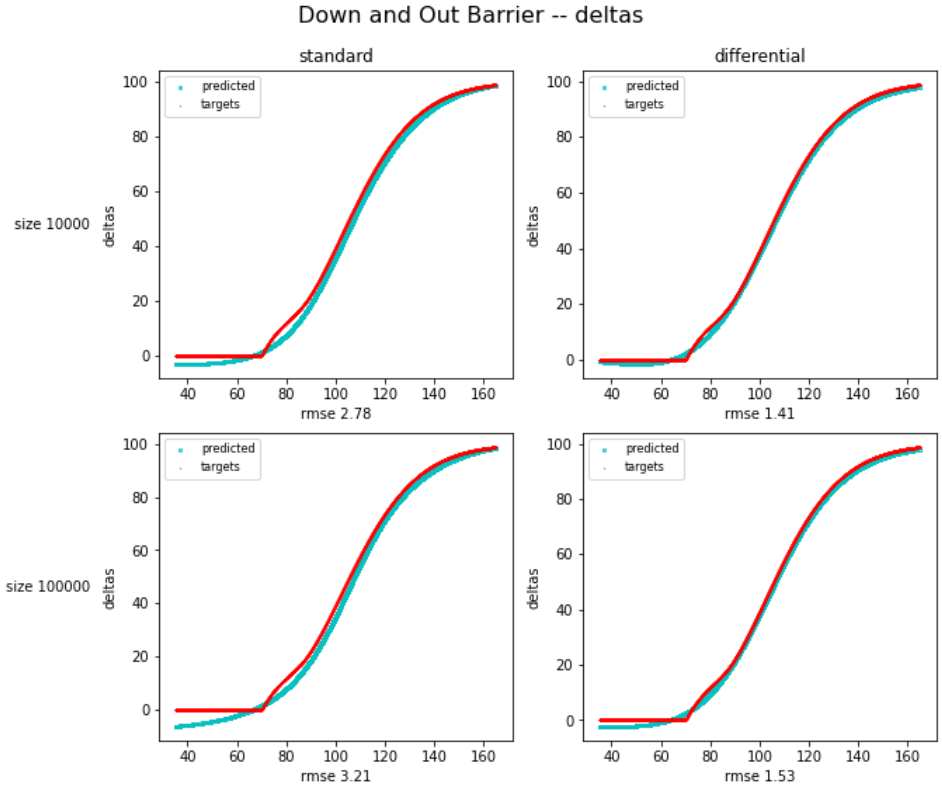


Figure 19 - Delta of a Down and Out call option with Soft Barrier using the standard Neural Network and the Twin network

In the first case, the one with the hard barrier, the network tries to approximate the barrier but the non-linearity observed realize in an instability of the results, with respect to the delta estimation. Relaxing the barrier, we observe a substantial improvement in the results especially concerning the first order derivative. The approximation works fast and in analytical time suggesting that the approximator works well even with non-linear payoff or complex transaction. In a practical context that network should be trained offline using a nested and sophisticated Monte-Carlo simulation, but once trained the network is able in working with online data and it can retrain itself using online data.

The rest of this sub-section aims to test the goodness of the network in some particular cases tested for each kind of option: Vanilla, Bachelier and Barrier. Those cases were taken from literature and consider an At-The-Money call option with a residual time to maturity equal to 1 day; classical methods like finite difference tends to be unstable in the case just considered, due to the fact that the delta tends to a nonlinear function of S_t. In fact, considering an ATM vanilla call option with a residual time to maturity equal to 1 day, if $S_{t+1} > K$ than the delta will be equal to one and the option writer should buy 1 unit of asset, on the other hand, if $S_{t+1} < K$ than the option writer should get a delta to 0. This has a practical implication for delta hedging close to maturity, since when delta gets more "nervous", delta hedging may become difficult and expensive.

- Vanilla call option: $T_1 = 1, T_2 = 1.001, K = 1.1, S_1 = 1.1, \sigma = 0.2 \; and \; r = 0$

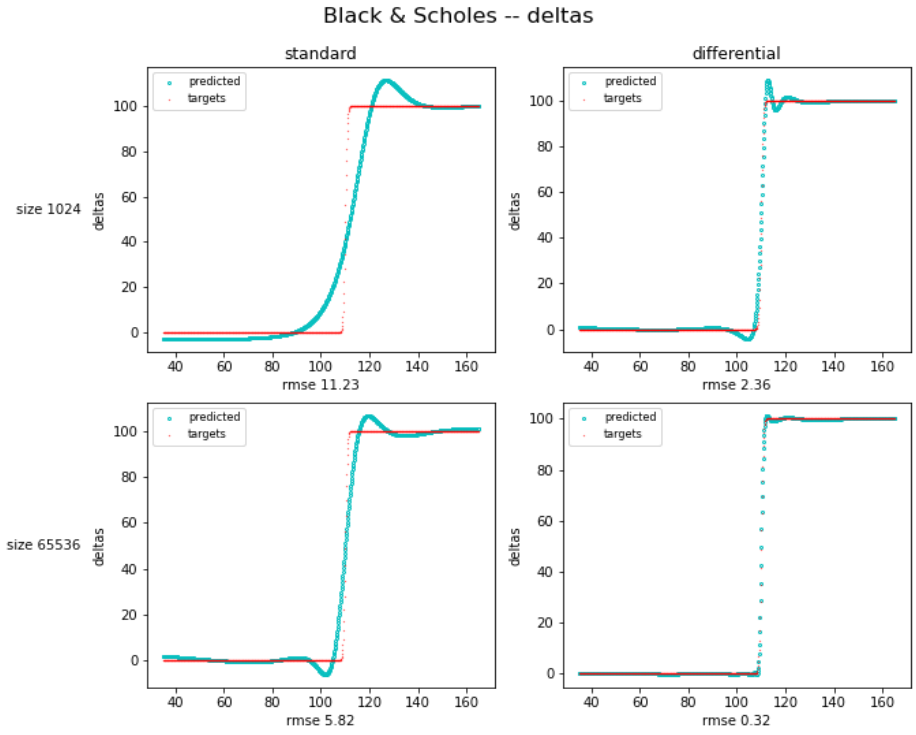


Figure 20 – Delta of an ATM call option.

As is clear from the figure above using the twin net the approximation tends to be tighter to the real delta also with a small dataset. Results obtained with 65k samples suggests that the network is robust also considering some limit case.

- Basket call option: $T_1 = 1, T_2 = 1.001, K = 1.1, S_1 = 1.1 \; and \; r = 0$

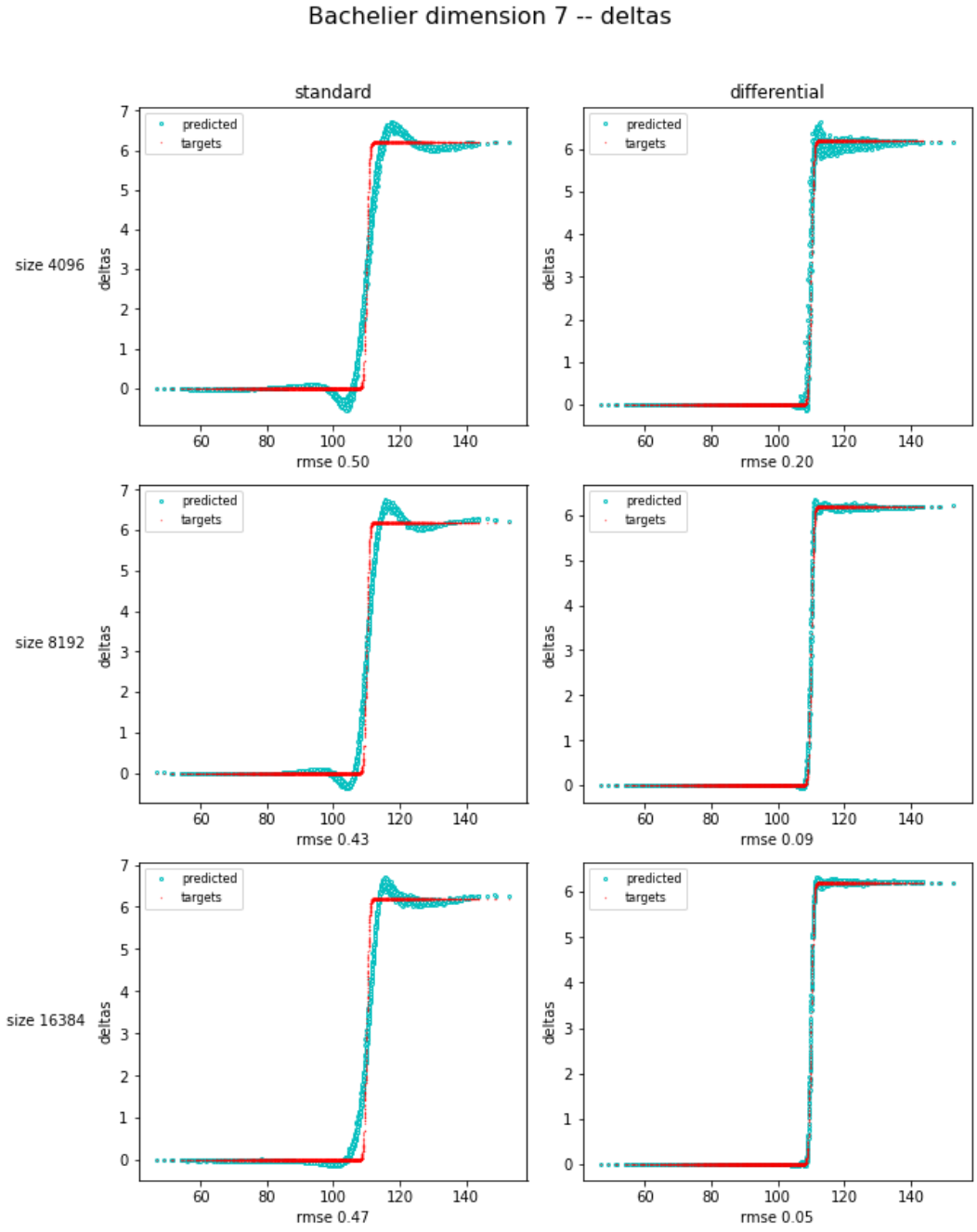


Figure 21 - Delta ATM Basket Option, with basket dimension equal to 7.

With dimension 7 the estimated delta tends to be stable with respect to the underlying value of the basket, considering the randomness in basket weights and in underlings volatility. In the figure below the same conclusion could be taken also for a basket dimension equal to 20. Looking at the

figure at the bottom right of the panel below is observable that the delta is affected by the randomness of the weights realizing in idiosyncratic instabilities that could now be predicted by the neural network. Apart from these considerations, the delta is well behaved and the approximation works at analytical speed considering the Twin-Net results.

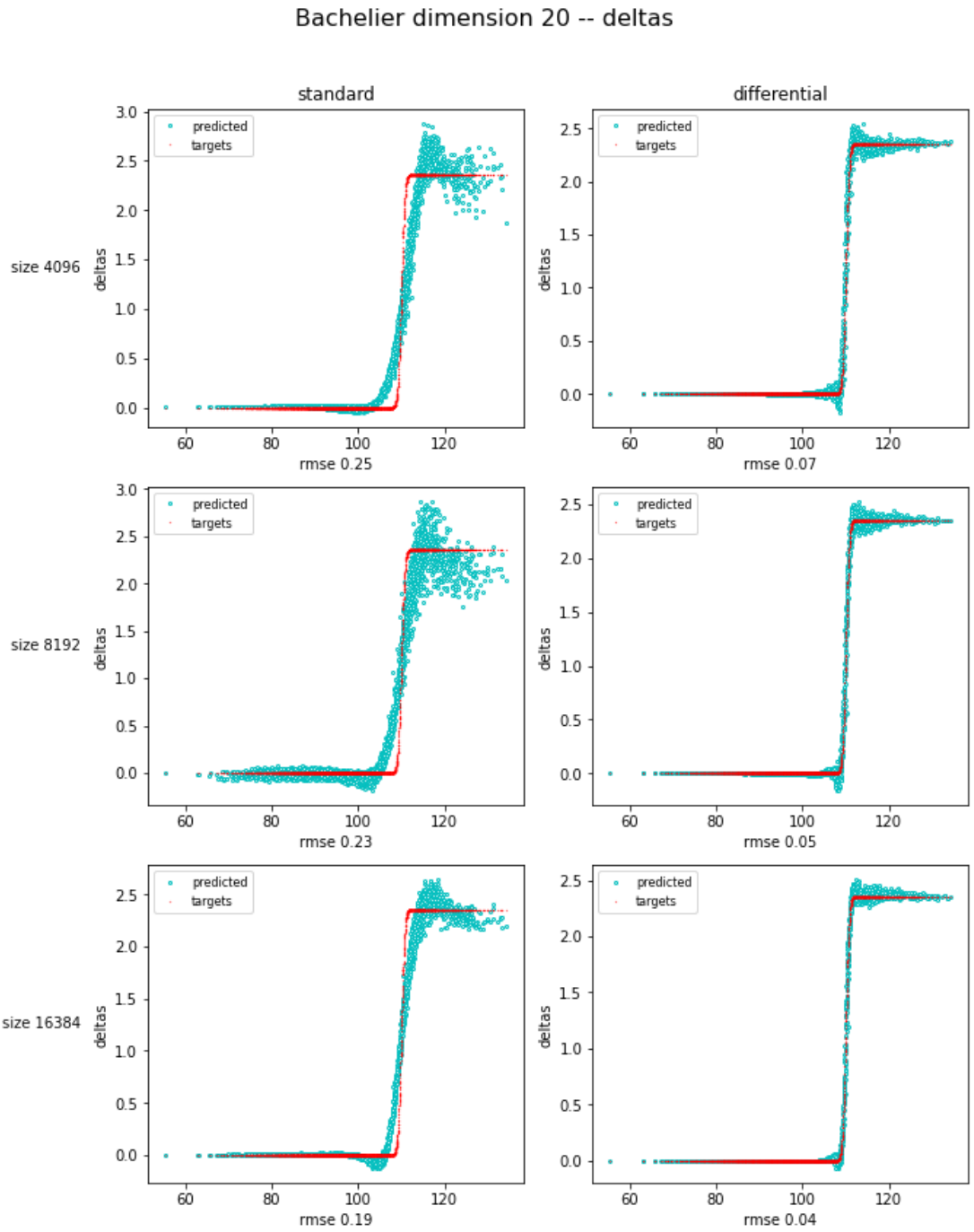


Figure 22- Delta ATM Basket Option, with basket dimension equal to 20

The barrier case is the most interesting one since the non-linearity appears twice, ones approximately at the strike (K) like in the previous kind of option, and ones at the barrier. We use the same fuzzy logic as in previous chapter 4.3 for relaxing the barrier. We expect that the Twin-

Net is able in predicting the delta of an out of the money barrier option. It is an OTM since the underlying $S_1 = H$ and the strike $K = 1.1$ is above S_1.

- Down-and-out call option: $T_1 = 1,\ T_2 = 1.001, K = 1.1,\ S_1 = 0.8\ , H = 0.8, \sigma = 0.2\ and\ r = 0$

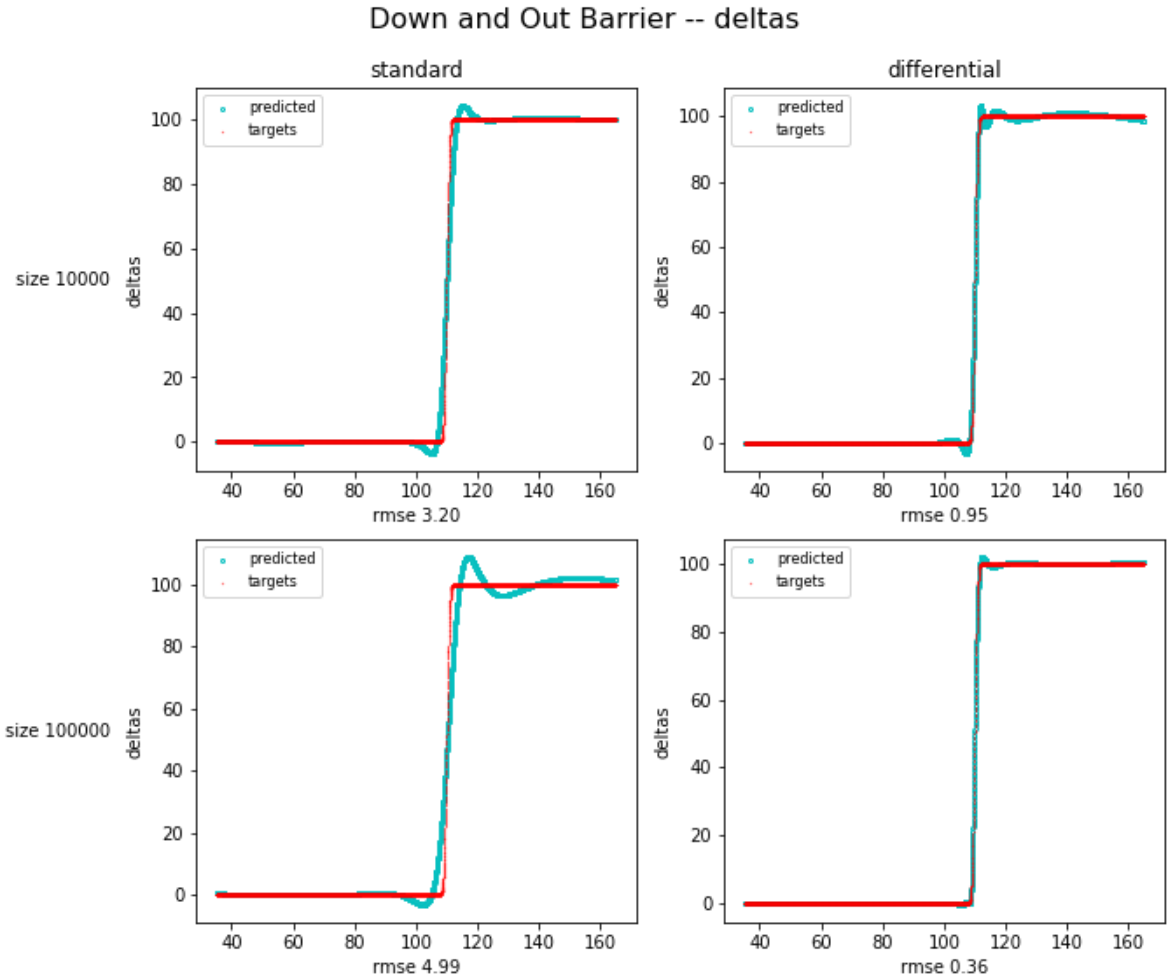


Figure 23 - Delta ATM Basket Option, with basket dimension equal to 20

The approximation works fast and produces an accurate approximation of the true delta. The results obtained below suggests that the Twin-Net approach works better than the classical feedforward network also considering some limit cases. The Delta obtained is well behaved and observing the same is clear how difficult could be approximating that kind of function.

5 Conclusion

Throughout our analysis, we have seen that 'learning the correct shape' from differentials is crucial to the performance of regression models, including neural networks, in such complex computational tasks as the pricing and risk approximation of arbitrary Derivatives trading books. The unreasonable effectiveness of what we called 'differential machine learning' permits us to accurately train ML models on a small number of simulated payoffs, in real-time, suitable for online learning. Differential networks apply to real-world problems, including regulations and risk reports with multiple scenarios. Twin networks predict prices and Greeks with almost analytic speed, and their empirical test error remains of comparable magnitude to nested Monte-Carlo.

Our machinery learns from data alone and applies in very general situations, with arbitrary schedules of cash flows, scripted or not, and arbitrary simulation models. Differential ML also applies to many families of approximations, including classic linear combinations of fixed basis functions, and neural networks of arbitrary complex architecture. Differential training consumes differentials of labels wrt inputs and requires clients to somehow provide high-quality first-order derivatives. In finance, they are obtained with AAD, in the same way, we compute Monte-Carlo risk reports, with analytic accuracy and very little computation cost. One of the main benefits of twin networks is their ability to learn effectively from small datasets. Differentials inject meaningful additional information, eventually resulting in better results with small datasets. Learning effectively from small datasets is critical in the context of e.g. regulations, where the pricing approximation must be learned quickly, and the expense of a large training set cannot be afforded. The penalty enforced for wrong differentials in the cost function also acts as a very effective regularize, superior to classical forms of regularization like Ridge, Lasso or Dropout, which enforce arbitrary penalties to mitigate overfitting, whereas differentials meaningfully augment data. Standard regularizes are very sensitive to the regularization strength λ, a manually tweaked hyperparameter. Differential training is virtually insensitive to λ because, even with infinite regularization, we train on derivatives alone and still converge to the correct approximation, modulo an additive constant. Differential training also appears to stabilize the training of neural networks, and improved resilience to hyperparameters like network architecture, seeding of weights or learning rate schedule was consistently observed, although to explain exactly why is a topic for further research. Standard machine learning may often be considerably improved with contextual information not contained in data, such as the nature of the relevant features from the knowledge of the transaction and the simulation model. For example, we know that the continuation value of a Bermudan option on some call date mainly depends on the swap rate to maturity and the discount rate to the next call. We can learn pricing functions much more effectively with hand-engineered features. But it has to be done manually,

on a case by case basis, depending on the transaction and the simulation model. If the Bermudan model is upgraded with stochastic volatility, the volatility state becomes an additional feature that cannot be ignored, and hand-engineered features must be updated. Differential machine learning learns just as well, or better, from data alone, the vast amount of information contained in pathwise differentials playing a role similar, and sometimes more effectively, to manual adjustments from contextual information. Differential machine learning is similar to data augmentation in computer vision, a technique consistently applied in that field with documented success, where multiple labeled images are produced from a single one, by cropping, zooming, rotation or recoloring. In addition to extending the training set for a negligible cost, data augmentation encourages the ML model to learn important invariances. Similarly, derivatives labels, not only increase the amount of information in the training set, but also encourage the model to learn the shape of the pricing function.

6 Annex

• Black-Scholes-Merton PDE

Here, we apply stochastic calculus and derive the celebrated and controversial Black-Scholes-Merton (BSM) pricing formula. The approach relies on a GBM model for the stock price $dS_t = \mu S_t dt + \sigma S_t dW_t$: Let us take the viewpoint of the writer of a vanilla, European-style call option, written on a stock share that does not pay any dividend. Let $f(S_t, t)$ be the fair option price at time t, when the underlying asset price is S_t Using Ito's lemma, we may write a stochastic differential equation for $f(\cdot,\cdot)$:

$$df = \frac{df}{dt}dt + \frac{df}{dS_t}dS_t + \frac{1}{2}\sigma^2 S_t^2 \frac{d^2 f}{dS_t^2}dt$$

Just as in the binomial case, we know is the option value at maturity,

$$f(S_T, T) = \max\{S_T - K, 0\}$$

Consider again the hedging problem for the option writer, who should take a position in Δ stock shares, so that the value of the hedged portfolio at time t is:

$$\pi(S_t, t) = -f(S_t, t) + \Delta S_t$$

Unlike previous applications of Ito's lemma, we do not know the function $f(\cdot,\cdot)$. We can hedge risk away, by eliminating the dependence of π_t on random variations in S_t. This may be accomplished by choosing:

$$\Delta = \frac{df}{dS_t}$$

To see this, let us differentiate the portfolio value π and take advantage of our choice of Δ:

$$d\pi = -df + \Delta S_t = \left(-\frac{df}{dS_t} + \Delta\right)dS_t - \left(\frac{df}{dt} + \frac{1}{2}\sigma^2 S_t^2 \frac{d^2 f}{dS_t^2}\right)dt = -\left(\frac{df}{dt} + \frac{1}{2}\sigma^2 S_t^2 \frac{d^2 f}{dS_t^2}\right)dt$$

Thanks to the choice of Δ, the term multiplying the random increment dS_t vanishes, so that the portfolio is riskless. Then, by no-arbitrage arguments, it must earn the risk-free interest rate r:

$$d\pi = r\pi dt$$

Substituting $d\pi$ we obtain:

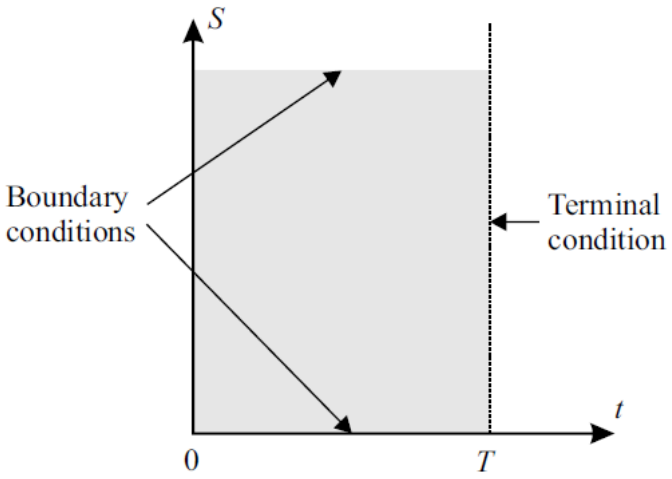
$$-\left(\frac{df}{dt} + \frac{1}{2}\sigma^2 S_t^2 \frac{d^2 f}{dS_t^2}\right)dt = r\left(-f + \frac{df}{dS_t}S_t\right)dt$$

which can be simplified by eliminating dt and rearranged as:

$$\frac{df}{dt} + rS_t \frac{df}{dS_t} + \frac{1}{2}\sigma^2 S_t^2 \frac{d^2 f}{dS_t^2} = rf$$

This is the **Black-Scholes-Merton equation (BSM equation for short),** which must be solved, subject to suitable boundary conditions. Pricing the option requires solving the above partial differential equation (PDE), and additional conditions are needed to pinpoint a specific solution. In fact, the BSM equation is fairly generic and, for instance, it does not discriminate between a call and a put option. The domain in which we have to solve the equation is an unbounded strip, for $t \in [0, T]$ and $S_t \in [0, +\infty)$. The strip is bounded in time but unbounded in price. Considering the specific case of a vanilla call, we get a terminal condition related to the payoff: $C_T^e = \max\{S_T - K, 0\}$. We get the following terminal conditions:

$$\lim_{S_t \to 0} C_t^e = 0 \ \ and \ \lim_{S_t \to +\infty} C_t^e = S_t - Ke^{-r(T-t)}$$



Solving the PDE leads to the celebrated **Black-Scholes-Merton equation.**

- **Bachelier Formula**

Referring to chapter 3.2, here we solve and derive the Bachelier model for the basket option. Starting from the forward model: $dF_t = \sigma_t dW_t$ integrating between $[0,T]$, $F_T = F_0 + \sigma W_T$ and so $F_T \sim N(F_0, \sigma^2 T)$. If we assume that the risk-free interest rate $r = 0$, then the spot price moves like the forward price and so: $S_T \sim N(S_0, \sigma^2 T)$.

We need to calculate:

$$C_0 = E_0^Q[\max\{S_T - K, 0\}] = \int_K^\infty (x-K)\, f(x)\, dx = \int_K^\infty (x-K)\, \frac{1}{\sqrt{2\pi T}\,\sigma} e^{-\frac{1}{2}\left(\frac{x-S_0}{\sigma\sqrt{T}}\right)^2} dx$$

By substituting $y = \frac{x-S_0}{\sigma\sqrt{T}}$ we get $dy = \frac{dx}{\sigma\sqrt{T}}$ and so $x = S_0 + \sigma\sqrt{T}y$:

$$C_0 = \int_{\frac{K-S_0}{\sigma\sqrt{T}}}^\infty (x-K)\, \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}(y)^2} dy = \int_{\frac{K-S_0}{\sigma\sqrt{T}}}^\infty \left(S_0 + \sigma\sqrt{T}y - K\right) \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}(y)^2} dy =$$

$$= \int_{\frac{K-S_0}{\sigma\sqrt{T}}}^\infty \sigma\sqrt{T}\, (y\, \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}(y)^2})\, dy + \int_{\frac{K-S_0}{\sigma\sqrt{T}}}^\infty (S_0 - K)\, \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}(y)^2} dy$$

Solving those integrals, we get the formulas used in 3.2.

- **Barrier Option**

Under the usual Black–Scholes assumptions, there is an explicit formula for the fair value of this option. We only consider in detail the case where the lower barrier is set below the option's strike price, $K > H$. In so doing, we see that there is a neat shortcut, which allows us to do many more complicated cases with little effort. Suppose that we are above the barrier, at asset value $S_t > H$ and time t, and we hold the down-and-out call. The next timestep, being infinitesimal, will not take us to the barrier. We can therefore apply the usual Black–Scholes hedging analysis, to show that the value of the option $C_{d|o}(S_t, t)$ satisfies the Black–Scholes equation:

$$\frac{df}{dt} + rS_t\frac{df}{dS_t} + \frac{1}{2}\sigma^2 S_t^2\frac{d^2f}{dS_t^2} = rf$$

Of course, this equation only holds for $H < S_t < +\infty$. The option does not exist for $S < H$. As before, the final condition for the equation above is $C_{d|o}(S_T, T) = \max\{S_T - K, 0\}$ but again only for $H < S_t < +\infty$.. As S becomes large the likelihood of the barrier being activated becomes negligible and so:

$$C_{d|o}(S_t, t) \approx S_t - K * e^{-r(T-t)} \text{ as } \quad S_t \to \infty.$$

We now see the most conspicuous way in which this valuation problem differs from that for a vanilla call. There, S_t runs from $0\ to + \infty$. Here, the second 'spatial' boundary condition is applied at $S = H$ rather than at $S = 0$. If S ever reaches H then the option expires worthless; this financial condition translates into the mathematical condition that on $S = H$ the value of the option is zero:

$$C_{d|o}(S_t, t) = 0$$

This completes the formulation of the problem; we now find the explicit solution, using a reduction to the heat equation and using a reflection principle to the BSM equation. See [9] for more references.

Bibliography

[1] B. Huge, A. Savine, Differential Machine Learning. arXiv, page arXiv:2005.02347v4, 2020.

[2] D. P. Kingma, J. Lei Ba, Adam: A Method for Stochastic Optimization, arXiv, page arXiv:1412.6980, 2017

[3] https://www.deeplearning.ai/ai-notes/initialization/

[4] DifferentialML.ipynb - Colaboratory (google.com)

[5] John C. Hull, Options, Futures, and Other Derivatives, Tenth Edition, New York: Pearson Education, 2018.

[6] P. Brandimarte, An Introduction to Financial Markets: A Quantitative approach, Wiley, 2018.

[7] Leslie N. Smith, A disciplined approach to neural network hyper-parameters: Part 1 -- learning rate, batch size, momentum, and weight decay, arXiv, page arXiv:1803.09820, 2018

[8] A. Savine. (2016). Fuzzy Logic for financial derivatives. 10.13140/RG.2.2.21293.54244.

[9] Prof. S. Howison, Oxford Mathematical Institute, barriers.pdf (ox.ac.uk).