Risk management with machine-learning-based algorithms

Simon Fecamp
Électricité de France (EDF)

Joseph Mikael
Électricité de France (EDF)

Xavier Warin
Électricité de France (EDF)

Risk management with machine-learning-based algorithms

Simon FECAMP [*†] Joseph MIKAEL [‡§] Xavier WARIN [¶‖]

February, 2019

Abstract

In this paper, we present several machine-learning-based algorithms to solve hedging problems in incomplete markets. The sources of incompleteness tested are illiquidity, non-tradable risk factors, discrete hedging dates and proportional transaction costs. Strategies of the machine-learning-based algorithms are compared to classical stochastic control techniques on several payoffs using a variance criterion. Some of the proposed algorithms are flexible enough to deal with any criteria. Thus, we compare strategies obtained with different risk criteria.

Keywords. Incomplete markets, transaction costs, deep learning, LSTM

1 Introduction

Despite its desirable properties, the complete market assumption is destroyed as soon as we consider transactions costs, discrete time hedging dates, illiquidity, non-tradable risk factors (e.g. volume risk), ... These properties make the completeness assumption not realistic in most of the financial markets and especially when trading on commodities markets. In an incomplete market, the set of non attainable contingent claims (i.e. contingent claims that cannot be replicated by a self-financing strategy) is not empty and for these, one need a criteria to decide how to share risks between the seller and the buyer.

The literature deals with three families of criteria: quantile hedging, utility functions and moment-based criteria.

Quantile hedging (see Föllmer and Leukert (1999), Bouchard et al. (2017)) aim is to construct a hedging strategy which maximizes the probability of a successful hedge given a constraint on the required cost. Another possibility offered by quantile hedging is to set a shortfall probability ε and minimize the cost in the class of hedging strategies such that the probability of covering the claim is at least $1 - \varepsilon$.

Utility-based-criteria and more precisely utility indifference (see Carmona (2008)) has the favor of academics as it sometimes allows to get analytic prices and hedging strategies. However this approach is not used by practitioners as the associated risk aversion coefficient is hard to define.

The last family, that we use in this paper is based on moments of the distribution of the hedged portfolio. The simplest moment-based-criterion is the variance criterion minimizing the variance of the hedged portfolio and the local variance of the portfolio (see Schweizer (1999) for a survey in continuous time). However quadratic criteria penalizes in the same way losses and gains. This might be seen as a drawback but this however offers the advantage of giving the same price to both buyers and sellers. Gobet et al. (2018) extends the local mean square criterion by introducing an asymmetry in the loss function that penalizes more losses than gains. In the case of a variance criterion or a local variance criterion, continuous time hedging strategies when the assets are modeled using some Levy processes are given for example in Tankov (2003).

Once the criterion has been chosen, one has to compute the trading strategy minimizing the criteria. Specific methods must be developed to deal with the source of incompleteness (whether it is illiquidity, transaction costs, non-tradable risk factor, ...)

Limited availability of hedging products can be dealt in two ways. First, Potters and Bouchaud (2003), Gatheral (2010) or Lehalle and Laruelle (2013) study the price impact of selling or buying an underlying on markets. The impact being greater with the exchanged volume, a seller will tend to limit the amount of volume to sell at one time. A second approach consists in assuming that in practice risk managers are aware of the liquidity constraints of the markets and try to implement strategies taking these into account. In the

*EDF R&D
†simon.fecamp@edf.fr
‡EDF R&D
§joseph.mikael@edf.fr
¶EDF R&D & FiME, Laboratoire de Finance des Marchés de l'Energie
‖xavier.warin@edf.fr

case of a global variance minimization of the hedged portfolio, Warin (To appear in 2020) developed some algorithms based on regression to calculate the hedging strategy taking into account all of these liquidity constraints.
In the literature, transaction costs treatment comes together with discrete hedging. The pioneering work of Leland (1985) proposes to use the Black-Scholes formula with a modified volatility. Kabanov and Safarian (2009) gives replication bound errors to the Leland (1985) model. Toft (1996) uses a mean-variance criteria to analyze the trade off between costs and risks of discretely re balanced option hedges in the presence of transactions costs.
In general when no closed-form-formula for the optimal hedging strategy is available we use some stochastic dynamic programming algorithms that suffers from the curse of dimensionality. To our knowledge, there are no algorithm to define the optimal strategy with arbitrary criteria together with liquidity constraints and transaction costs and robust to high dimensions.
In this article we propose some machines learning algorithms to derive optimal hedging strategy.

- the first set of algorithms try to calculate hedging positions by solving a global risk minimization problem. The hedging strategies are calculated using different types of architectures. The most efficient architecture is easy to implement and can be used with liquidity constraints, general risk criteria and with transaction costs. In the latter transaction costs case, we describe how we can use the algorithm to estimate a Pareto frontier by training the algorithm with random mean-variance combinations. This algorithm is fast enough to be used in high dimensions.
- the second and third algorithms are some machine learning version of the two algorithms described in Warin (To appear in 2020) that can only be used for a variance criterion: a dynamic programming method is used and some minimization problems are solved at each time step in order to calculate the optimal hedging strategy.

In the second section of the article, we describe the hedging problem and set the price model used for the experiments. We present several well-known loss functions and we propose a new one.
In the third section, we detail the different algorithms used.
In the fourth section, we focus on the variance criterion and compare the results obtained by the different algorithms on options involving a variable number of risk factors, be they tradable or not. We take as a reference calculations achieved on high performance computers by the StOpt library Gevret (2016) using the algorithm 2 described in Warin (To appear in 2020). Clearly the first machine learning algorithm appears to be the best out of the three machine learning algorithms developed especially regarding computation time. We then train the first algorithm with the different risk criteria mentioned in the second section, and discuss the impact of these on distribution of the hedged portfolio.
In the fifth section, we introduce transaction costs and show how to estimate a Pareto frontier by training the algorithm with random combinations of mean and variance targets.
Therefore the main result of this article is to show that an effective and flexible machine learning algorithm can solve difficult hedging problems in moderate dimension as effectively as the most effective existing algorithm using regressions but at a far smaller computing cost.

2 Problem description

In the numerical tests, we retain the price modelling used in Warin (To appear in 2020). A short description is done in Section 2.1 and we refer to the original paper for further details.

2.1 Risk factors modelling

We are given a financial market operating in continuous time: we begin with a probability space $(\Omega, \mathcal{F}, \mathbb{P})$, a time horizon $0 < T < \infty$ and a filtration $\mathcal{F} = (\mathcal{F}_t)$, $0 \leq t \leq T$ representing the information available at time t. We consider $d+1$ assets $\hat{F}^0, \ldots, \hat{F}^d$ available for trade. For the sake of simplicity, we suppose a zero interest rate and we assume that there exists a risk free asset $\hat{F}^0$ having a strictly positive price. We then use $\hat{F}^0$ as numeraire and immediately pass to quantities discounted with $\hat{F}^0$. We denote $F^i = \hat{F}^i/\hat{F}^0$, $i = 1, ...d$ the thus discounted quantities and F the vector having the $(F^i)_{i=1..d}$ as coordinates. We consider another non tradable risk factor (the volume risk) denoted $\mathcal{V}$.
The evolution of the $(F^i)_{i=1..d}$ and of $\mathcal{V}$ are respectively described by a diffusion process having values in $\mathbb{R}^d$ and in $\mathbb{R}$.
More precisely, the volume risk $\mathcal{V}_t$ is stochastic and follows the dynamic:

$$\mathcal{V}_t = \hat{\mathcal{V}}_t + \left(\mathcal{V}_u - \hat{\mathcal{V}}_u\right) e^{-a_{\mathcal{V}}(t-u)} + \int_u^t \sigma_{\mathcal{V}} e^{-a_{\mathcal{V}}(t-s)} dW_s^{\mathcal{V}} \tag{1}$$

where $a_{\mathcal{V}}$ is the mean reverting coefficient, $\sigma_{\mathcal{V}} \geq 0$ the volatility, and $W_t^{\mathcal{V}}$ is a Brownian motion on $(\Omega, \mathcal{F}, \mathbb{P})$. $\hat{\mathcal{V}}_u$ is the average load seen on the previous years at the given date u. We suppose that, for $i = 1, \ldots, d$, the prices are martingales and follow the dynamic:

$$
\begin{aligned}
F_t^i &= F_0^i e^{-(\sigma_{i,E})^2 \frac{e^{-2a_{i,E}(T-t)} - e^{-2a_{i,E}T}}{4a_{i,E}} + e^{-a_{i,E}(T-t)} \hat{W}_t^{i,E}}, &(2)\\
\hat{W}_t^{i,E} &= \sigma_{i,E} \int_0^t e^{-a_{i,E}(t-s)} dW_s^i &(3)
\end{aligned}
$$

where F_t^i represents the forward price seen at time t for a delivery at date T which is given once for all and will correspond to the maturity of the considered contracts, $a_{i,E}$ the mean reverting parameter for risk factor i, $\sigma_{i,E}$ the volatility parameter for risk factor i and W_s^i a Brownian motion on $(\Omega, \mathcal{F}, \mathbb{P})$ so that the W_t^i are correlated and also correlate with $W_t^{\mathcal{V}}$. We will denote S_t the vector $(F_t^1, \ldots, F_t^d, \mathcal{V}_t)$.

2.2 Hedging problem

We consider the hedging problem of a contingent claim paying $g(S_T)$ at time T. Without loss of generalities, in the following we consider ourselves as the derivative seller. We consider a finite set of hedging dates $t_0 < t_1 < \ldots < t_{N-1} < \ldots < t_N = T$. The discrete hedging dates bring the first source of incompleteness. At each date, each of the discounted assets F^i can only be bought and sold at a finite quantity l^i giving a second source of incompleteness. The volume risk $\mathcal{V}_t$ cannot be traded and is the third source of incompleteness. A self-financing portfolio is a d-dimensional $(\mathcal{F}_t)$-adapted process Δ_t. Its terminal value at time T is noted X_T^Δ and satisfies:

$$
X_T^\Delta = p + \sum_{i=1}^{d} \sum_{j=0}^{N-1} \Delta_{t_j}^i (F_{t_{j+1}}^i - F_{t_j}^i), \tag{4}
$$

where p will be referred to as the premium. Between two time steps, the change in Δ^i, corresponding to the buy or sell command $C_{j+1}^i := \Delta_{t_{j+1}}^i - \Delta_{t_j}^i$ should not exceed in absolute value the liquidity l^i so that:

$$
|\Delta_{t_0}^i| \leq l^i, \quad |C_j^i| \leq l^i, \quad j = 1, \ldots, N-1, i = 1, \ldots, d. \tag{5}
$$

Given a loss function L, we search for a strategy verifying:

$$
(p^{Opt}, \Delta^{Opt}) = Argmin_{p,\Delta} L(X_T^\Delta - g(S_T)) = Argmin_{p,\Delta} L(Y_T). \tag{6}
$$

We will focus on the following loss functions:

- **Mean Square error** defined by
$$
L(Y) = \mathbb{E}\left[Y^2\right]. \tag{7}
$$
It has been intensively studied for example in Schweizer (1999). It has the drawback of penalizing losses and gain the same way. This also can be seen as an advantage as it gives the same value and strategy for the buyer and for the seller.
- **Asymmetrical loss** defined by:
$$
L^\alpha(Y) = \mathbb{E}\left[(1+\alpha) Y^2 \mathbf{1}_{Y \leq 0} + Y^2 \mathbf{1}_{Y \geq 0}\right]. \tag{8}
$$
When $\alpha > 0$ (resp. $0 < \alpha$) , the losses (resp. gains) are penalized. It will be referred to as the asymmetrical loss. It has been studied for example in Gobet et al. (2018).
- **Loss Moment 2/Moment 4 function** defined by:
$$
L^\alpha(Y) = \mathbb{E}\left[Y^2 \mathbf{1}_{Y \geq 0}\right] + \alpha \mathbb{E}\left[Y^4 \mathbf{1}_{Y \leq 0}\right], \alpha \geq 1. \tag{9}
$$
This criteria is designed to penalize heavy tail on the loss side.

3 Neural-network-based algorithms

Deep neural networks are state-of-the-art tools for approximating functions (see Liang (2017)). We propose to utilize their universal estimator property to approach solutions of hedging problems. In a first set of algorithms that we call global algorithms, we use different neural networks architectures to estimate the hedging portfolio process by a global risk minimization. The second set of algorithms, called local algorithms, are based on stochastic dynamic programming with both residual risk conditional expectations and hedging portfolio process estimated by deep neural networks. Global algorithms can be used with all risk criteria of Section 4.2 while local algorithms can only be used with a variance criterion.

3.1 Feedforward neural network as function approximators

We suppose in this section that the input is in dimension d_0 (the state variable x) and the output is in dimension d_1 (the number of value functions to estimate). The network is characterized by a number of layers $L+1 \in \mathbb{N} \setminus \{1,2\}$ with m_ℓ, $\ell = 0, \ldots, L$, the number of neurons (units or nodes) on each layer: the first layer is the input layer with $m_0 = d$, the last layer is the output layer with $m_L = d_1$, and the $L-1$ layers between are called hidden layers, where we choose for simplicity the same dimension $m_\ell = m$, $\ell = 1, \ldots, L-1$.

A feedforward neural network is a function from $\mathbb{R}^{d_0}$ to $\mathbb{R}^{d_1}$ defined as the composition

$$x \in \mathbb{R}^d \longmapsto A_L \circ \varrho \circ A_{L-1} \circ \ldots \circ \varrho \circ A_1(x) \ \in \ \mathbb{R}. \tag{10}$$

Here A_ℓ, $\ell = 1, \ldots, L$ are affine transformations: A_1 maps from $\mathbb{R}^d$ to $\mathbb{R}^m$, $A_2, \ldots, A_{L-1}$ map from $\mathbb{R}^m$ to $\mathbb{R}^m$, and A_L maps from $\mathbb{R}^m$ to $\mathbb{R}^{d_1}$, represented by

$$A_\ell(x) = \ \mathcal{W}_\ell x + \beta_\ell, \tag{11}$$

for a matrix $\mathcal{W}_\ell$ called weight, and a vector β_ℓ called bias term, $\varrho : \mathbb{R} \to \mathbb{R}$ is a nonlinear function, called activation function, and applied component-wise on the outputs of A_ℓ, i.e., $\varrho(x_1, \ldots, x_m) = (\varrho(x_1), \ldots, \varrho(x_m))$. Standard examples of activation functions are the sigmoid, the ReLu, the Elu, tanh.

All these matrices $\mathcal{W}_\ell$ and vectors β_ℓ, $\ell = 1, \ldots, L$, are the parameters of the neural network, and can be identified with an element $\theta \in \mathbb{R}^{N_m}$, where $N_m = \sum_{\ell=0}^{L-1} m_\ell(1+m_{\ell+1}) = d(1+m)+m(1+m)(L-2)+m(1+d_1)$ is the number of parameters. We denote by Θ_m the set of possible parameters.
The universal approximation theorem of Hornick et al. Hornik et al. (1990) states that set all feedforward approximators making m vary is dense in $L^2(\nu)$ for any finite measure ν on $\mathbb{R}^d$, whenever ϱ is continuous and non-constant.
Assuming the optimal control of Equation (6) is sufficiently smooth, from the universal approximation theorem we do know that the control can be approached with a feedforward neural network having sufficient depth and width. The latter theorem does not tell what are the minimal depth and width so that empirical studies have to be done to know what is the best architecture. The universal approximation theorem does not tell neither how to optimize the neural networks weights but it appears that a stochastic gradient descent shows good results in many cases.

3.2 Recurrent and LSTM neural networks as time-dependent-function approximator

Recurrent neural networks (RNNs) are dynamical systems that make efficient the use of temporal information in the input sequence. For RNNs the input is a times series and in this paper, the output is composed of two vectors: a memory state M_t and an output state C_t. At each time step t, M_{t-1} and C_{t-1} are given together with the time series to a recurrent cell i.e. a neural network which weights are shared across all time steps (see Figure 3). Long short term memory cells (Hochreiter and Schmidhuber (1997)) are powerful for capturing long-range dependence of the data. They are designed to avoid some vanishing gradients effect that basic RNN suffers. In an LSTM cell, structures called gates regulates the flow of information contained in the memory state M_t by adding or removing information to the state. Gates are composed out of a sigmoid neural network layer and a pointwise multiplication operation. Mathematically, the rules inside the t-th cell follows:

$$\Gamma_t^f = \sigma(A_f S_t + U_f C_{t-1} + b_f) \tag{12}$$
$$\Gamma_t^i = \sigma(A_i S_t + U_i C_{t-1} + b_i) \tag{13}$$
$$\Gamma_t^o = \sigma(A_o S_t + U_o C_{t-1} + b_o) \tag{14}$$
$$M_t = \Gamma_t^f \odot M_{t-1} + \Gamma_t^i \odot tanh(A_M S_t + U_M C_{t-1} + b_M), M_0 = 0 \tag{15}$$
$$C_t = \Gamma_t^o \odot tanh(M_t), C_0 = 0 \tag{16}$$

where $\odot$ is the Hadamard product, σ is the sigmoid activation function $\left(\sigma(x) = \frac{1}{1+e^{-x}}\right)$, $A_\bullet \in \mathbb{R}^{h \times d}$, $U_\bullet^{h \times h}$, $b_\bullet \in \mathbb{R}^h$, h being the cell state size. Γ_t^f represents the forget gate. It decides what information needs to be deleted from the memory state. This decision is made by a sigmoid layer called the "forget gate layer". It outputs a number between 0 and 1 and multiply it to each number in the memory state $M_{\text{tâĹŠ1}}$. Γ_t^i is the input gate evaluating what new information needs to be stored in the memory state. The output gate layer Γ_t^f decides what parts of the memory state needs to be outputted. It is based on filtered version of the memory state. The weight matrices and bias vector $(A_\bullet, U_\bullet, b_\bullet)$ are shared through all time steps and are learned during the training process. The output C_t is used as an approximation of the unknown function.

3.3 Global neural network architectures

In this section, at each time step we are given inputs (risk factors realizations, time-to-maturity) and we search for an optimal control to minimize one (single) global loss function. Firstly, we present different

architectures and secondly we compare numerically these architectures on simple options.
In the following, $\tilde{S}_t$ denotes a normalized version of S_t (see Section 3.3.3 for normalization details). Consider a neural network $\mathbb{NN}^\theta$ parameterized by θ and taking as inputs simulations of the S_t's. For each sample of S_t $\mathbb{NN}^\theta(S_t)$ outputs a control i.e. a number of assets to buy or to sell. For a given loss function L, and empirical simulations of S_t the aim of global algorithms is to compute the following:

$$\arg\min_\theta L(\mathbb{NN}^\theta(S_T) - g(S_T)) \tag{17}$$

To find the θ a mini batch stochastic gradient descent is used. Adaptive Moment Estimation (Adam) Kingma and Ba (2014) is a method that computes adaptive learning rates for each parameter. In addition to storing an exponentially decaying average of past squared gradients v_t like AdaDelta Zeiler (2012) and RMSprop (Tieleman and Hinton (2012)) Adam also keeps an exponentially decaying average of past gradients m_t similar to momentum.

Algorithm 1 Forward resolution of global algorithms

α : Stepsize
β_1, $\beta_2 \in [0,1]$, Exponential decay rates for the moment estimates,
N_{iter} number of iterations
N_{batch}, the number of simulations at each gradient descent iteration (batch size).
θ_0 randomly chosen
$m_0 \leftarrow 0$
$v_0 \leftarrow 0$
$t \leftarrow 0$
for $t = 0 \ldots N_{Iter}$ **do**
    $S_u \leftarrow N_{batch}$ samples simulations of S_u, $u = t_0, ..., t_{N-1}, T$
    $t \leftarrow t + 1$
    $g_t = \nabla_\theta L(\mathbb{NN}^{\theta-1}(S_u) - g(S_T))$ (get gradient w.r.t objective function)
    $m_t \leftarrow m_{t-1} + (1-\beta_1).g_t$ (update biased first moment estimate)
    $v_t \leftarrow \beta_2 v_{t-1} + (1-\beta_2)g_t^2$ (update biased second raw moment estimate)
    $\hat{m}_t \leftarrow \frac{m_t}{1-\beta_1^t}$ (computes bias-corrected first moment estimate (β_1^t stands for β_1 to the power of t))
    $\hat{v}_t \leftarrow \frac{v_t}{1-\beta_2^t}$ (computes bias-corrected second raw moment estimate (β_2^t stands for β_1 to the power of t))
    $\theta_t \leftarrow \theta_{t-1} - \alpha \hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon)$ (update parameters)

3.3.1 Feedforward neural networks architectures for the hedging problem

A possible approach to solve the hedging problem described in Section 2.2 consists in training N different feedforward neural networks (one per time steps) as done in Han et al. (2018) for the PDE case and as illustrated in Figure 1 and described in Section 3.3.1.a. This architecture (denominated feedforward control in the following) generates a possibly high number of weights and bias to be estimated ($N * depth * width$). Another possibility is to train one single feedforward neural network fed with the prices and the time to maturity as illustrated in Figure 2 and described in Section 3.3.1.b. This architecture is referred to as feedforward merged control in what follows.

3.3.1.a Feedforward control structure

In the feedforward control network, $N-1$ networks are fed successively with $(\tilde{S}_{t_i})_{i=1 \ldots N-1}$. The feedforwards networks are parameterized by θ (the bias and weights to be estimated). The i-th feedforward neural network provides a d dimensional control $\Delta_{t_i}(\tilde{S}_{t_i}, \theta)$. The first control $\Delta_{t_0}(\tilde{S}_{t_0}, \theta)$ and the premium $p(\theta)$ are trainable variables. The final payoff is given by:

$$X_T(\theta) \quad = \quad p(\theta) + \sum_{i=1}^{d} \sum_{j=0}^{N-1} \Delta_{t_j}^i (S_{t_j}, \theta)(F_{t_{j+1}}^i - F_{t_j}^i). \tag{18}$$

and the problem (6) leads to the following optimization problem:

$$\theta^* = Argmin_\theta L(X_T(\theta) - g(S_T)). \tag{19}$$

3.3.1.b Feedforward merged control structure

In the feedforward merged control structure a single neural network is fed successively with $(\tilde{S}_{t_i})_{i=1 \ldots N-1}$. For each pair (t_i, S_{t_i}) the network provides a control $\Delta(t_i, S_{t_i}, \theta)$ where θ represents the bias and weights to

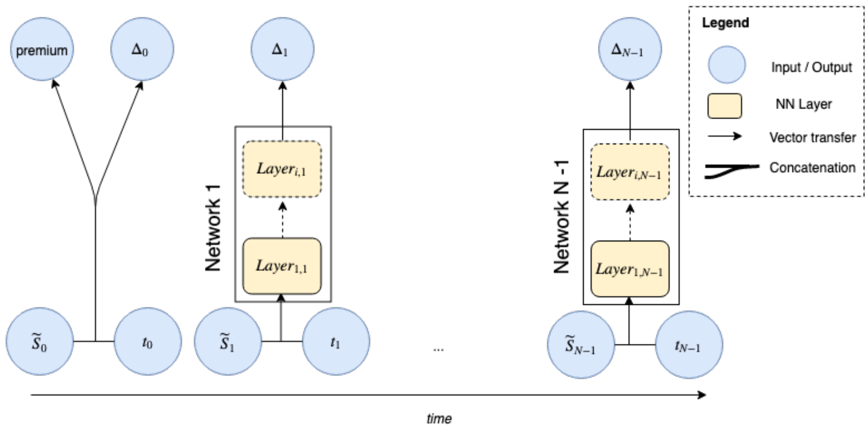


Figure 1: Feedforward basic architecture: At each time step a new feedforward neural network is trained

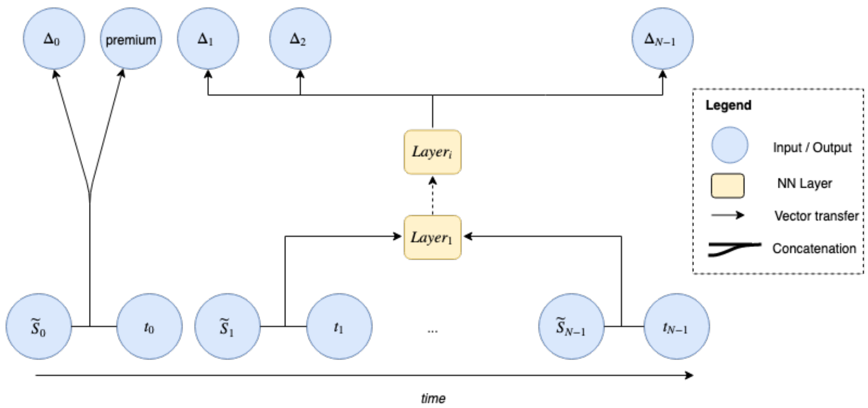


Figure 2: Feedforward merged architecture: a time dimension is added to the input features but the feedforward bias and weights networks are shared within all the timesteps

be estimated. Again, the first control $\Delta(t_0, \tilde{S}_{t_0}, \theta)$ and the premium $p(\theta)$ are trainable variables. The final payoff is given by:

$$X_T(\theta) \quad = \quad p(\theta) + \sum_{i=1}^{d} \sum_{j=0}^{N-1} \Delta^i(t_j, S_{t_j}, \theta)(F^i_{t_{j+1}} - F^i_{t_j}). \tag{20}$$

The problem (6) leads to the following optimization problem:

$$\theta^* = Argmin_\theta L(X_T(\theta) - g(S_T)). \tag{21}$$

Remark 3.1 *As we retain neither the feedforward nor the feedforward merged architectures we don't implement any liquidity management for these two architectures.*

3.3.2 Recurrent networks

The hedging problem sequential nature makes relevant the use of recurrent neural networks (RNN). This kind of networks is used for example in Chan-Wai-Nam et al. (2019) for the PDE numerical resolution problem. As mentioned in Chung et al. (2014), among all RNN architectures, LSTM neural networks (see Hochreiter and Schmidhuber (1997)) present several advantages among which the convergence speed and the memory management. It would allow for example the management of non markovian underlying models.

3.3.2.a Structure of the augmented LSTM network

As mentioned in Section 3.3.4, the classical LSTM cell is not sufficient for our application and we modify it according to Figure 4. The extra ReLu layers allows to describe complex functions of the state and inputs. The recurrent cell is fed with $\tilde{S}_t$. Its recursive calls on a sequence of inputs provides a sequence of underlying positions changes (see Figure 3).

At each date t_j, the recurrent cell produces a d-dimensional output depending on historical events and controls $\hat{C}_j(\theta, (\tilde{S}_{t_s})_{s\leq j}, (\Delta_{t_s})_{s\leq j})$ (denoted simply C_j in in Figure 4) that is not bounded. The strategy Δ's are calculated for $j = 0, \ldots, N-1; i = 1, \ldots, d$

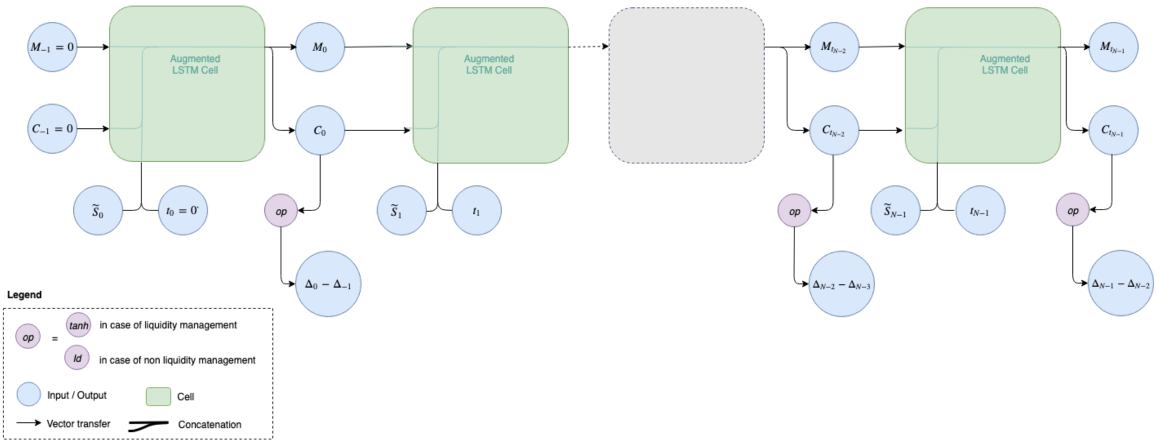


Figure 3: Recurrent architecture. The difference between Basic LSTM and augmented LSTM lies in the use of the augmented LSTM cell

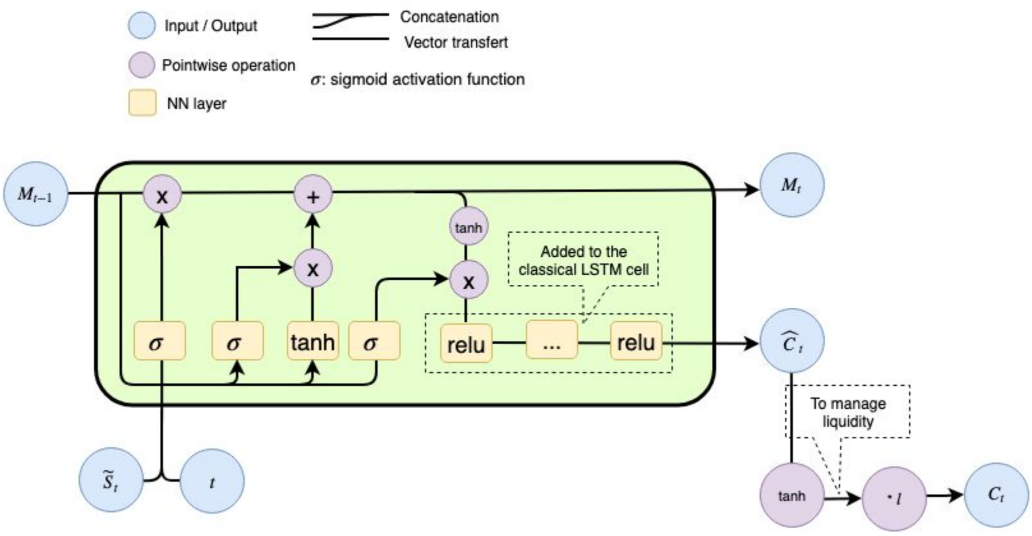


Figure 4: Augmented LSTM Cell (Figure inspired from Olah (2015))

- if there are no liquidity constraints by

$$\Delta^i(t_j, (S_{t_i})_{i\le j}, \theta) \quad = \quad \sum_{k=0}^{j} \hat{C}^i_k((\tilde{S}_{t_s})_{s\le j}, \Delta(t_s, (S_{t_s})_{s\le j}, \theta)). \tag{22}$$

- if there are liquidity constraints by

$$\Delta^i(t_j, (S_{t_i})_{i\le j}, \theta) \quad = \quad l^i \sum_{k=0}^{j} \tanh(\hat{C}^i_k((\tilde{S}_{t_s})_{s\le j}, \Delta(t_s, (S_{t_s})_{s\le j}, \theta))). \tag{23}$$

  By the way, the control difference between two time steps belongs to $[-l_i, l_i]$. The premium $p(\theta)$ is a trainable variable.

The final payoff is then given by:

$$X_T(\theta) \quad = \quad p(\theta) + \sum_{i=1}^{d} \sum_{j=0}^{N-1} \Delta^i(t_j, (S_{t_k})_{k\le j}, \theta)(F^i_{t_{j+1}} - F^i_{t_j}). \tag{24}$$

and the problem (6) leads the following optimization problem:

$$\theta^* = Argmin_\theta L(X_T(\theta) - g(S_T)). \tag{25}$$

Remark 3.2 *In the following the Basic LSTM architecture refers to the very same architecture except that we do not apply extra ReLu layers.*

3.3.3 Global neural networks extra-parameters

The neural networks results depend on some extra parameters listed hereafter. Unless otherwise specified, these parameters are shared for all the test cases.

- The **batch size**, the number of simulations we give at each iteration of the Adam optimizer is equal to 50.
- The Adam initial **learning rate** is equal to 0.001 (default parameter).
- The **number of units in LSTM cells** (dimension of M_t) in the LSTM cell is equal to 50.
- We use 3 ReLU layers and densities 10 for the augmented LSTM cell.
- We use **batch normalization** of the data before they are given to the neural networks. The mean and variance used for the normalization are computed once for all over a subset of 100 000 simulations.
- Unless otherwise specified the **number of iterations** in the gradient descent algorithm is equal to 20 000. Every 1 000 iterations, we keep the neural network state if it gives a better loss on the test set than previously.

In the following, tests are done using TensorFlow (Abadi et al. (2015)).

3.3.4 Numerical comparison of global neural-network architectures

Table 1 compares the Mean square hedging error of Equation (7) obtained with these two architectures and with the augmented LSTM architecture proposed in Section 3.3.2.a for a Black-Scholes call option (with trend μ and volatility σ) with no liquidity constraints. Results obtained with the Black-Scholes Δ are also shown. After 20 000 iterations, the results obtained with the augmented LSTM architecture are better than with the two feedforward networks.

	Mean Square error
Black-Scholes $\Delta(N(d_1))$	1.61e-05
Feedforward delta [10, 10, 10]	1.32e-04
Feedforward delta [10, 15, 30]	1.31e-04
Feedforward merged [10, 10, 10]	1.37e-04
Feedforward merged [10, 15, 30]	1.30e-04
Augmented LSTM 50 units [10, 10, 10]	1.73e-05

Table 1: Mean Square error on a Black-Scholes call option with different neural network architectures. Layer sizes are denoted with a list (e.g. [10, 15, 20] means three hidden layers of sizes respectively 10, 15 and 20). Parameters: $S_0 = K = 1, \Delta t = 1/365, T = 1/12$ years, $\mu = 0, \sigma = 0.2$). The number of iterations is set to 20 000. Activation functions for the feedforward networks are ReLu functions.

In Table 2, we show the Mean Square error of Equation (7) loss derived from a basic LSTM cell on a liquidity-constraints-free vanilla call option and on a 2 market spreads call option (having payoff $(S^1_T - S^2_T - K)^+$). cases (2 and 3 markets). We compare this loss to the loss derived from the augmented LSTM cell defined in Section 3.3.2.a. We can see that for complex payoffs, the augmented LSTM cell gives better results. Moreover the feedforward cells used in the augmented cells allow the use of different activation functions.

	Black Scholes call option	2 markets spread
Basic LSTM Cell	5.73e-05	3.64e-04
Augmented LSTM Cell	3.97-05	1.11E-04

Table 2: Mean Square comparison between, different basic and augmented LSTM architectures. Parameters: Call option: $T = 3/12, \Delta t = 1/360, S^1_0, \mu = 0.02, \sigma^1 = 0.3$ - 2 Markets spread option ($S^1_0 = 1., S^2_0 = 0.5, K = 0.5, \sigma^1 = \sigma^2 = 0.3, \mu^1 = \mu^2 = 2\%, corr(W^1, W^2) = 0.2$).

3.4 Local algorithms

The two other algorithms are local algorithms based on a dynamic programming principle proposed in Warin (To appear in 2020). The objective function to minimize is given by equation (6), (7), so corresponds to a global variance hedging problem. In the original article the author uses some grids for the discretization of the asset level and some regressions to calculate conditional expectations. As previously stated, theses two algorithms are only available to optimize variance problems.
It can be noticed the two local machine learning algorithms proposed can be related to some recent works

in Huré et al. (2018); Bachouch et al. (2018) and Huré et al. (2019).
We introduce the spaces for $\tilde{\Delta}$ in $\mathbb{R}^d$

$$\begin{aligned}
W_i(\tilde{\Delta}) =& \{(V,\Delta) \in \mathbb{R} \times \mathbb{R}^d, \mathcal{F}_{t_i}\text{-adapted with } |\Delta^k - \tilde{\Delta}^k| \leq l^k, \text{ for } k = 1,\dots,d\},\\
\Theta_i(\tilde{\Delta}) =& \{(\Delta_i,\dots,\Delta_{N-1}), \text{ where for } j \geq i, \Delta_j \text{ are } \mathbb{R}^d \text{ valued}\\
& \mathcal{F}_{t_j}\text{-adapted with } |\Delta_i^k - \tilde{\Delta}^k| \leq l^k, |\Delta_{j+1}^k - \Delta_j^k| \leq l^k \text{ for } i \leq j < N-1, k = 1,\dots,d\}\\
\hat{W}_i(\tilde{\Delta}) =& \{(V,\Delta) \text{ where } V \text{ is } \mathbb{R} \text{ valued}, \mathcal{F}_{t_i}\text{-adapted }, \Delta \in \Theta_i(\tilde{\Delta})\}.
\end{aligned}$$

As shown in proposition 3.1 in Warin (To appear in 2020), the problem (6), (7) can be written as

$$\begin{aligned}
(\hat{p},\hat{\Delta}) = \underset{p\in\mathbb{R},\Delta\in\hat{\Theta}_0(0)}{\arg\min} & \sum_{i=2}^{N} \mathbb{E}\left[\left(V_i - \sum_{k=1}^{d} \Delta_{i-1}^k (F_{t_i}^k - F_{t_{i-1}}^k) - V_{i-1}\right)^2\right] + \\
& \mathbb{E}\left[\left(V_1 - \sum_{k=1}^{d} \Delta_0^k (F_{t_1}^k - F_{t_0}^k) - p\right)^2\right],
\end{aligned} \tag{26}$$

where the V_i satisfies:

$$\begin{aligned}
V_N =& g(S_T),\\
V_i =& \mathbb{E}\left[g(S_T) - \sum_{k=1}^{d}\sum_{j=i}^{N-1} \Delta_j^k (F_{t_{j+1}}^k - F_{t_j}^k) \quad |\mathcal{F}_{t_i}\right], \forall i = 1,\dots,N-1,
\end{aligned} \tag{27}$$

3.5 First local algorithm

Equation (26) gives a dynamic programming algorithm: introducing the optimal residual R at date t_i, for current state S_{t_i} and having in portfolio an investment in Δ_{i-1} assets:

$$R(t_i, S_{t_i}, \Delta_{i-1}) = \min_{(V,\Delta)\in\hat{W}_i(\Delta_{i-1})} \mathbb{E}\left[\left(g(S_T) - \sum_{k=1}^{d}\sum_{j=i}^{N-1} \Delta_j^k (F_{t_{j+1}}^k - F_{t_j}^k) - V\right)^2 \quad |\mathcal{F}_{t_i}\right], \tag{28}$$

then equation (26) gives

$$R(t_i, S_{t_i}, \Delta_{i-1}) = \min_{(V,\Delta)\in W_i(\Delta_{i-1})} \mathbb{E}\left[\left(\tilde{V} - \sum_{k=1}^{d} \Delta_i^k (F_{t_{i+1}}^k - F_{t_i}^k) - V\right)^2 + R(t_{i+1}, S_{t_{i+1}}, \Delta_i)|\mathcal{F}_{t_i}\right], \tag{29}$$

where $\tilde{V}$ is the first component of the argmin in equation (28) calculating $R(t_{i+1}, S_{t_{i+1}}, \Delta_i)$.
In the special case where the prices are martingale the $(\tilde{V}, V)$ in (29) are independent of the hedging strategy and given by $(\mathbb{E}[g(S_T)|\mathcal{F}_{t_{i+1}}], \mathbb{E}[g(S_T)|\mathcal{F}_{t_i}])$ and only the hedging strategy is left to calculate by solving the classical local min variance problem:

$$\hat{R}(t_i, S_{t_i}, \Delta_{i-1}) = \min_{\Delta\in\mathbb{R}^d} \mathbb{E}\left[\left(\tilde{V} - \sum_{k=1}^{d} \Delta^k (F_{t_{i+1}}^k - F_{t_i}^k) - V\right)^2 |\mathcal{F}_{t_i}\right], \tag{30}$$

Our goal is then to use a neural network to calculate the V_i functions (so only calculate a conditional expectation) and the optimal control Δ_i both as functions of S_{t_i} at each date t_i by minimizing (30) at each time step by a backward recursion.
At the opposite of the V, the delta have bounded values due to liquidity constraints and $\Delta_j \in [\underline{\Delta}_j, \overline{\Delta}_j]$ where the minimal constraints $\underline{\Delta}_j$ and maximal constraints $\overline{\Delta}_j$ are in $\mathbb{R}^d$.
Normalizing the position in hedging products, we introduce $\hat{\Delta}_j = R_j(\Delta_j) := \frac{\Delta_j - \underline{\Delta}_j}{\overline{\Delta}_j - \underline{\Delta}_j}$ such that $\hat{\Delta}_j \in [0,1]$ and we normalize the S_{t_i} introducing $\hat{S}_{t_i}$ defined by $\hat{S}_{t_i}^k = \frac{S_{t_i}^k - \mathbb{E}[S_{t_i}^k]}{\sqrt{\mathbb{E}[(S_{t_i}^k)^2] - \mathbb{E}[S_{t_i}^k]^2}}$ for $k = 1,\dots,d+1$.
At each time step a Feed Forward Neural Network is used to parametrize the portfolio value and the normalized command as a function of the normalized uncertainties and storage level: $\left(\hat{V}_j(\theta_j; \hat{S}_{t_j}, \hat{\Delta}_j), \hat{C}(\theta_j; \hat{S}_{t_j}, \hat{\Delta}_j)\right)$.

The first algorithm 2 solves in a backward recursion (30). Then at each time step, the resolution of equation (31) is achieved by using a machine learning approach where each functions depends on some normalized variables to ease convergence of the method. The resolution of equation (31) is achieved by using a classical stochastic gradient descent.

Remark 3.3 *We create a single network for $\hat{V}_j$ and $\hat{\Delta}_j$ letting $\hat{V}_j$ depend on $\hat{\Delta}_{t_{j-1}}$ the hedging position at the previous date. In this martingale case it would have been possible to create two networks , the second being used to represent V as a function of $\hat{S}$ only.*

Remark 3.4 *The position x in the hedging position (normalized in $[0,1]^d$) is sampled uniformly in the algorithm. The $\hat{S}_{t_j}$ are sampled according their own empirical laws and the $\hat{S}_{t_{j+1}}$ are sampled conditionally to the $\hat{S}_{t_j}$.*

Remark 3.5 *The output of the Neural network has unbounded values. In order to satisfy the constraints on the hedging positions, a* tanh *transformation of the output of the neural network $\hat{C}(\theta_j; \hat{S}_{t_j}, \hat{\Delta}_j)$ permits to have an output in $[-1,1]^d$.*

Algorithm 2 Backward resolution for first local resolution algorithm (martingale case)

1: $U_N(\hat{S}_{t_N}(\omega), \hat{\Delta}_N) = g(S_T), \quad \forall \hat{\Delta}_N \in [0,1]^d,$
2: **for** $j = N-1, N-2, \dots, 1$ **do**
3: For $x \in U(0,1)^d$

$$\theta_j^* = \arg\min_\theta \mathbb{E}\left[\left(U_{j+1}(\hat{S}_{t_{j+1}}, R_{j+1}(\phi_j(\theta; \hat{S}_{t_j}, x))) - \phi_j(\theta; \hat{S}_{t_j}, x).(F_{t_{j+1}} - F_{t_j}) - \hat{V}_j(\theta; \hat{S}_{t_j}, x)\right)^2 |F_{t_j}\right], \quad (31)$$

where

$$\phi_j(\theta; \hat{S}_{t_j}, x) = \left(R_j^{-1}(x) + l \tanh(\hat{C}_j(\theta, \hat{S}_{t_j}, x))\right)$$

4: $U_j(.,.) = \hat{V}_j(\theta_j^*, .,.)$
5: At last:

$$\arg\min_{p \in \mathbb{R}, \Delta_0 \in [-l,l]} \mathbb{E}\left[(U_1(\hat{S}_{t_1}, R_1(\Delta_0)) - C_0.(F_{t_1} - F(t_0)) - p)^2\right]$$

3.5.1 Second local algorithm

The second algorithm can be seen as a path generalization of the first algorithm where at each time step an optimization is achieved to calculate the value function and the command at the current time step using the previously calculated commands. In this algorithm the gain functional $\bar{R}$ is updated ω by ω. Then $\bar{R}$ satisfies at date t_i with an asset value S_{t_i} for an investment Δ_{i-1} chosen at date t_{i-1}:

$$\begin{aligned}\bar{R}(t_i, S_{t_i}, \Delta_{i-1}) =& H - \sum_{k=1}^{d} \sum_{j=i}^{N-1} \Delta_j^k (F_{t_{j+1}}^k - F_{t_j}^k), \\ &= \bar{R}(t_{i+1}, S_{t_{i+1}}, \Delta_i) - \sum_{k=1}^{d} \Delta_i^k (F_{t_{i+1}}^k - F_{t_i}^k),\end{aligned}$$

and, as shown in Warin (To appear in 2020), at the date t_i the optimal control Δ is associated to the minimization problem:

$$\min_{(V,\Delta) \in \mathbb{R} \times \mathbb{R}^d} \mathbb{E}\left[(\bar{R}(t_{i+1}, S_{t_{i+1}}, \Delta) - \sum_{k=1}^{d} \Delta^k (F_{t_{i+1}}^k - F_{t_i}^k) - V)^2 |\mathcal{F}_{t_i}\right].$$

This leads to the second algorithm 3.

Algorithm 3 Backward resolution for second local resolution algorithm

1: **for** $j = N-1, N-2, \dots, 1$ **do**

2: For $x \in U(0,1)^d$

$$\theta_j^* = \arg\min_{\theta} \mathbb{E}\left[\left(g(S_T) - \sum_{k=j}^{N-1} \Delta_k.(F_{t_{k+1}} - F_{t_k}) - \hat{V}_j(\theta; \hat{S}_{t_j}, x)\right)^2 |S_{t_j}\right], \tag{32}$$

where

$$\begin{aligned} \Delta_j = & \quad \phi_j(\theta; \hat{S}_{t_j}, x) \\ \Delta_{k+1} = & \quad \phi_{k+1}(\theta^*_{k+1}, \hat{S}_{t_{k+1}}, R_{k+1}(\Delta_k)) \text{ for } k \in [j, N-2] \end{aligned}$$

and

$$\phi_k(\theta; \hat{S}_{t_k}, x) = R_k^{-1}(x) + l \tanh(\hat{C}_k(\theta, \hat{S}_{t_k}, x)) \text{ for } k \in [j, N-1]$$

3: At last:

$$\arg\min_{p \in \mathbb{R}, \Delta_0 \in [-l,l]} \mathbb{E}\left[(g(S_T) - \sum_{k=0}^{N-1} \Delta_k.(F_{t_{k+1}} - F_{t_k}) - p)^2\right]$$

Each optimization is achieved using a stochastic gradient descent. Notice that the second algorithm is far more costly than the first one as, at each time step, some command values have to be evaluated from the current time to the maturity of the asset to hedge.

3.5.2 Parameters for the local algorithm

We give the parameters used in the optimization process:

- At each time step, a classical Feed Forward network of **four layers** (so one input layer, 2 hidden layers and one output layer) with **12 neurons** each is used. The three first layers use an ELU activation function while the output layer uses an identity activation function.
- The **batch size**, i.e. the number of simulations we use at each iteration to proceed an Adam gradient update is 2000.
- At each time step the number of iterations used is limited to a number increasing with the dimension of the problem, from 5000 for the 4 dimensions problem to 25000 for problems which dimension strictly exceeds 4.
- The initial learning rate is $1e-3$.

4 Numerical results in the transaction cost-free case and mean square error

In this section, we compare the three machine learning-based algorithms with a stochastic control based tool (Gevret (2016)) using a thin discretization to evaluate the optimal variance.

4.1 Spread options payoff description

We use some spread option problem to compare the three algorithms. The payoff in this section is defined for $M \geq 2$ by:

$$g(S_T) \quad = \quad \mathcal{V}_T \left(F_T^1 - \frac{1}{d-1}\sum_{i=2}^{d} F_T^i - K\right)^+. \tag{33}$$

For all the cases we take the following parameters:

- The maturity in days is equal to $T = 90$ days,
- $K = 10$,
- the number of hedging dates N is taken equal to 14 (but the control on last hedging date is trivial).

- l^i the liquidity (i.e. the maximum quantity we can buy or sell) at each date is taken equal to 0.2 for all underlying,
- $F_0^1 = 40$, $\sigma_{1,E} = 0.004136$, $a_{1,E} = 0.0002$ in days.
- the initial load associated to the option satisfy $\mathcal{V}_0 = 1$.

The three cases take the following parameters:

1. Case 1: $d = 2, \sigma_{\mathcal{V}} = 0$
   This case is a four dimensional case (2 assets and 2 hedging positions) with:
   - $F_0^2 = 30$, $\sigma_{2,E} = 0.003137$, $a_{2,E} = 0.0001$ in days.
   - $\rho_{1,2} = 0.7$ is the correlation between the two assets.
2. Case 2: $d = 2$
   This is a 5 dimensional case, with the same parameters as in the first case but with a varying load with parameters $\sigma_{\mathcal{V}} = 0.02$, $a_{\mathcal{V}} = 0.02$ in days. The correlation between each of the tradeable assets and $\mathcal{V}$ is equal to 0.2.
3. Case 3: $d = 3$, $\sigma_{\mathcal{V}} = 0$
   This is a case in dimension 6 with
   - $F_0^2 = 35$, $\sigma_{2,E} = 0.003137$, $a_{2,E} = 0.0001$ in days.
   - $F_0^3 = 25$, $\sigma_{3,E} = 0.005136$, $a_{3,E} = 0.0001$ in days.
   - The correlation between asset i and j is noted $\rho_{i,j}$ and satisfies: $\rho_{1,2} = 0.7, \rho_{1,3} = 0.3, \rho_{2,3} = 0.5$.

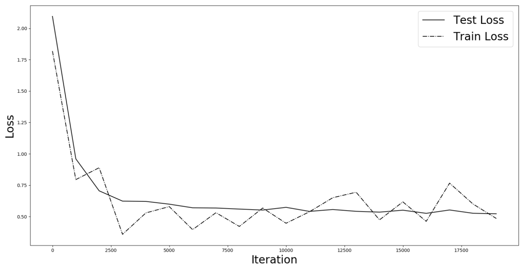
4.1.1 Numerical results

In Table 3, the variance obtained on 100 000 common simulations are given for the 3 algorithms and compared to the variance obtained by the StOpt library. Notice that due to the size of the problem the case 3 is not totally converged with the StOpt library.
For local algorithm 1 and 2, we run the optimization 10 times and take the best variance obtained.
The global algorithm is far more effective in term of computing time than the local algorithm as 10 000 iterations runs in 220 s on the graphic card of a core I3 laptop while algorithm 2 and 3 can take some hours for the case 3.

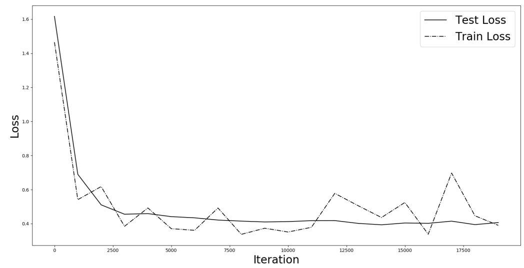
Mean Square Error	**Case 1**	**Case 2**	**Case 3**
Not hedged Portfolio	8.3058	8.5250	10.5960
Hedged with StOpt	0.3931	0.5160	0.4983
Hedged with Global Algo	0.3920	0.5205	0.4852
Hedged with Algo 1	0.3971	0.5168	0.4763
Hedged with Algo 2	0.3912	0.5183	0.4943

Table 3: Mean Square comparison between, NN-based algorithms and stochastic control algorithm

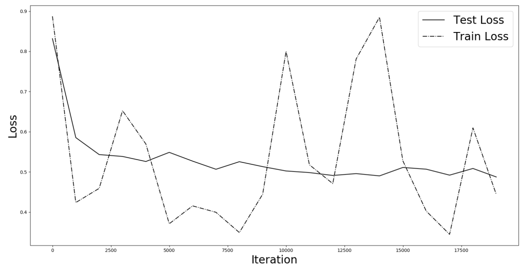
In Figure 5, the losses for the market spread and for the Global NN algorithm are plotted.



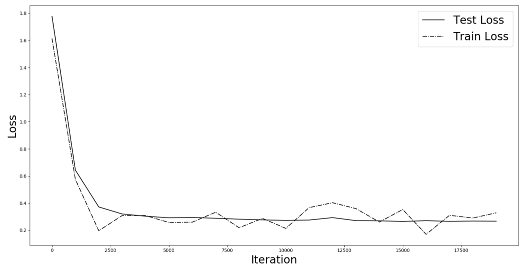
Case 1.



Case 2.



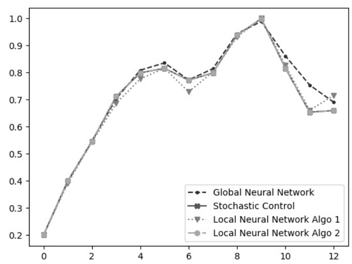
Case 3.



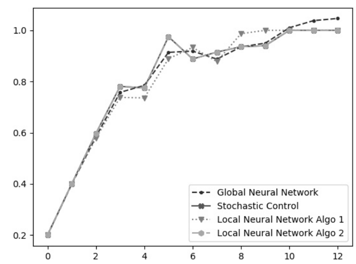
Case 2. with two time more time steps

Figure 5: Loss functions for the Global NN algorithm.

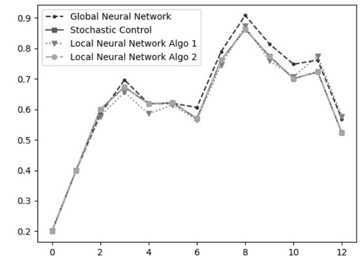
As shown in Figures 6, 7 and 8 the Deltas for the 2 and 3 markets spread follow the same shape for the four algorithms.



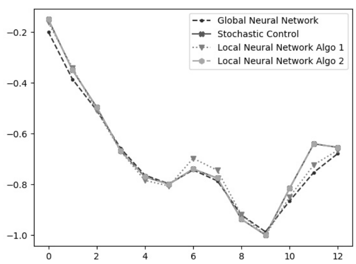
Future 1 Sim Nb 1



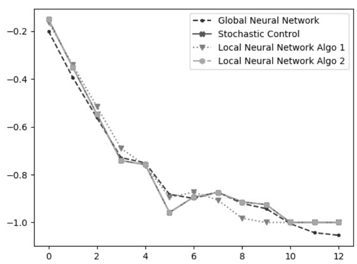
Future 1 Sim Nb 2



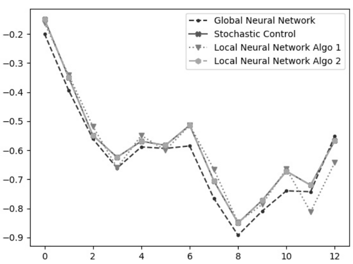
Future 1 Sim Nb 3



Future 2 Sim Nb 1



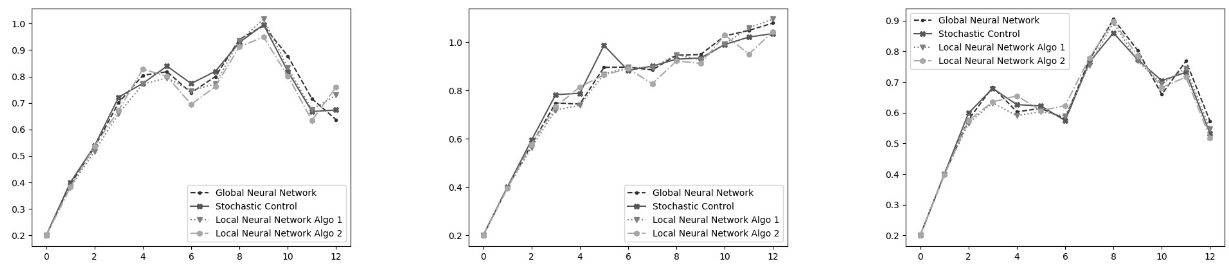
Future 2 Sim Nb 2



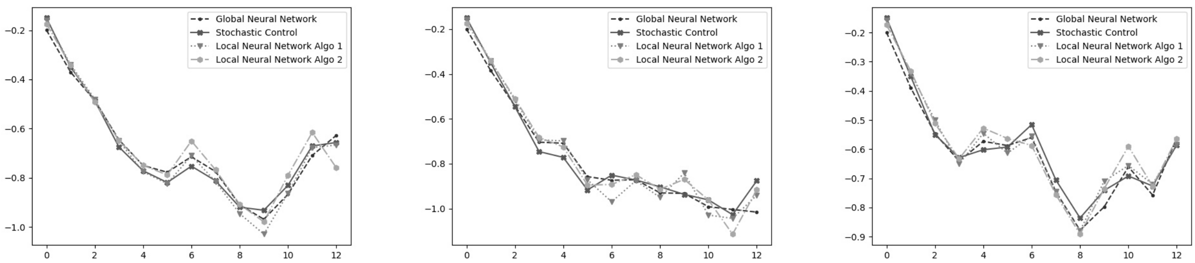
Future 2 Sim Nb 3

Figure 6: Delta for Case 1.

The numerical results indicate that the global algorithm and local algorithm give similar results. We observe that, using 10 runs, the local algorithms gives similar results in the low dimension, but as the dimension increases, the results obtained may differ a lot meaning that the optimizer is often trapped in a local minimum solution far from the result. Besides the number of iterations to use at each step has to be increased a lot with the dimension leading to a non-competitive running time compared to the global algorithm.

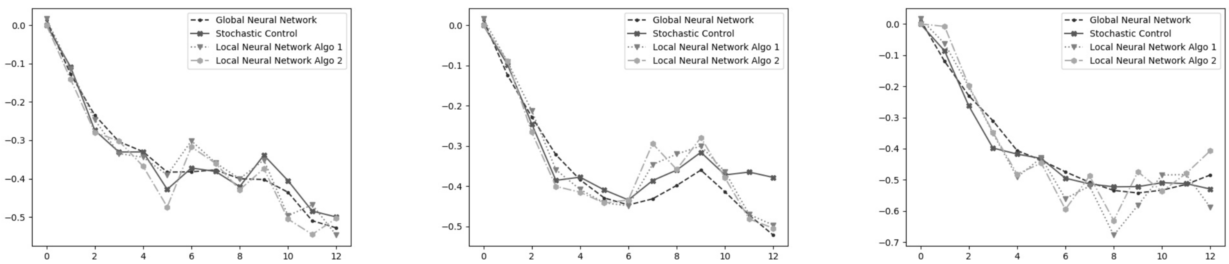


Future 1 Sim Nb 1 Future 1 Sim Nb 2 Future 1 Sim Nb 3

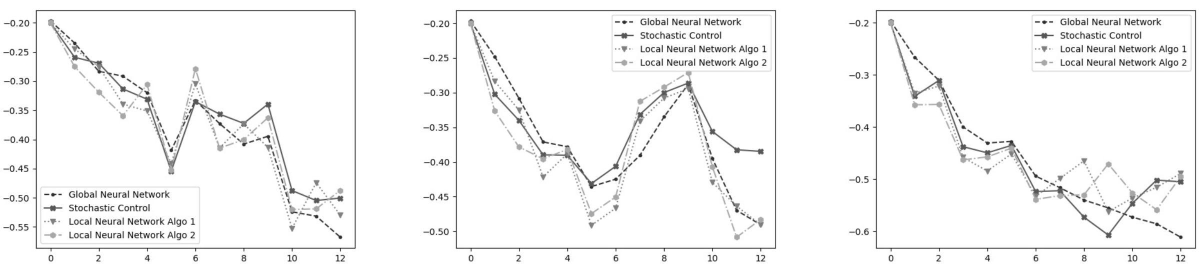


Future 2 Sim Nb 1 Future 2 Sim Nb 2 Future 2 Sim Nb 3

Figure 7: Delta for Case 2.



Future 2 Sim Nb 1 Future 2 Sim Nb 2 Future 2 Sim Nb 3



Future 3 Sim Nb 1 Future 3 Sim Nb 2 Future 3 Sim Nb 3

Figure 8: Delta for Case 3.

The global algorithm is still very effective in dimension 6 and, being able to solve the problem very quickly, is a candidate to give a method solving problems in very high dimensions.
One question that arises is how the three neural-network-based algorithms perform when the number of decisions i.e. the number of hedging dates increases. To increase the number of hedging dates we can increase the maturity T while keeping the same distance between two hedging dates. Due to the mean reverting nature of the chosen models a more complex case consists in keeping $T = 90$ days while increasing the number of hedging dates. In Table 4 we compute the error of the four algorithms with 28 (instead of 14 previously) hedging dates and a liquidity of 0.15 (instead of 0.20) units per date. The three approaches are effective in term of accuracy. The time spent with the local algorithm 2 explodes due to the resimulation at

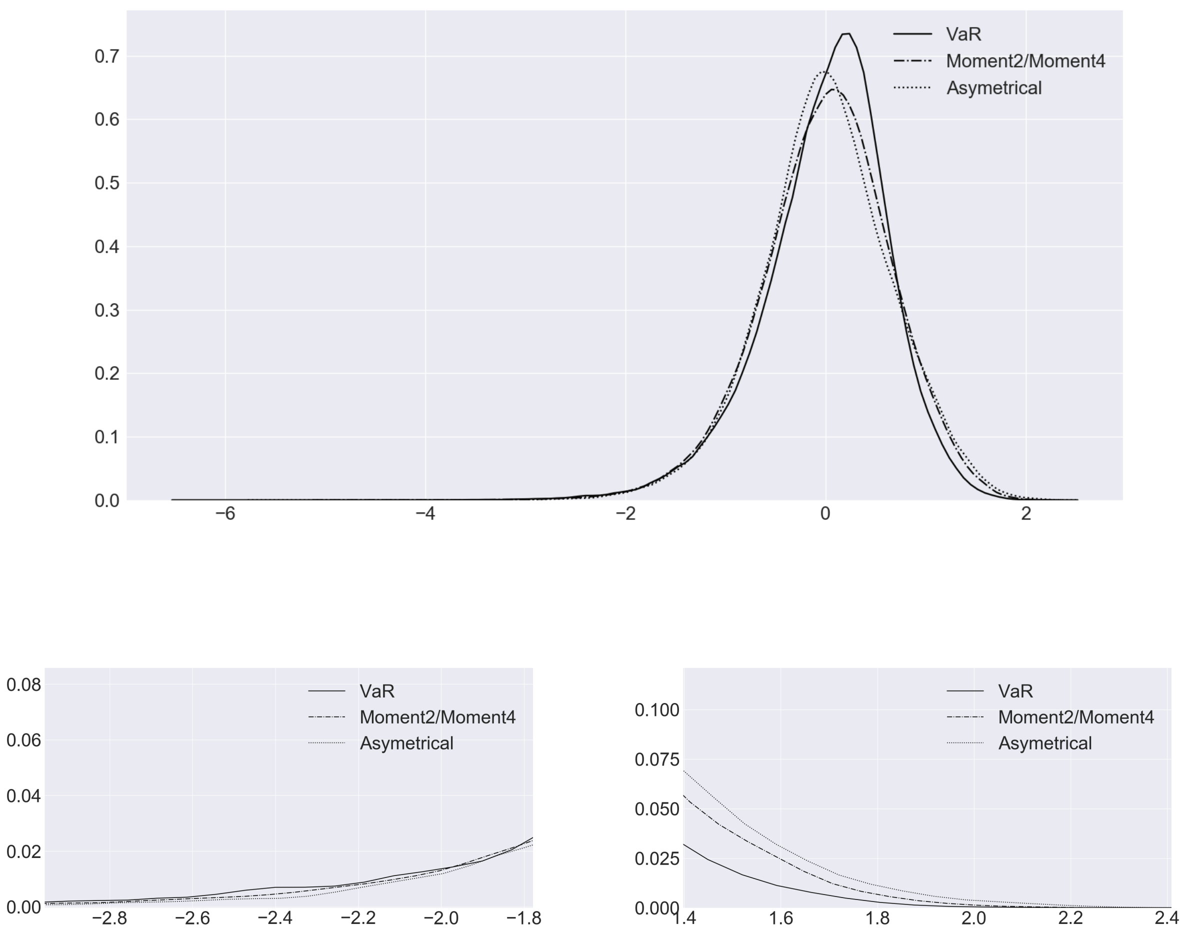
each date of the optimal strategy until maturity.

Stochastic Control	**Global**	**Algo 1**	**Algo 2**
0.271	0.265	0.259	0.262

Table 4: Mean Square error on Case 1 with 24 hedging dates and a liquidity of 0.15.

4.2 Testing different risk criteria

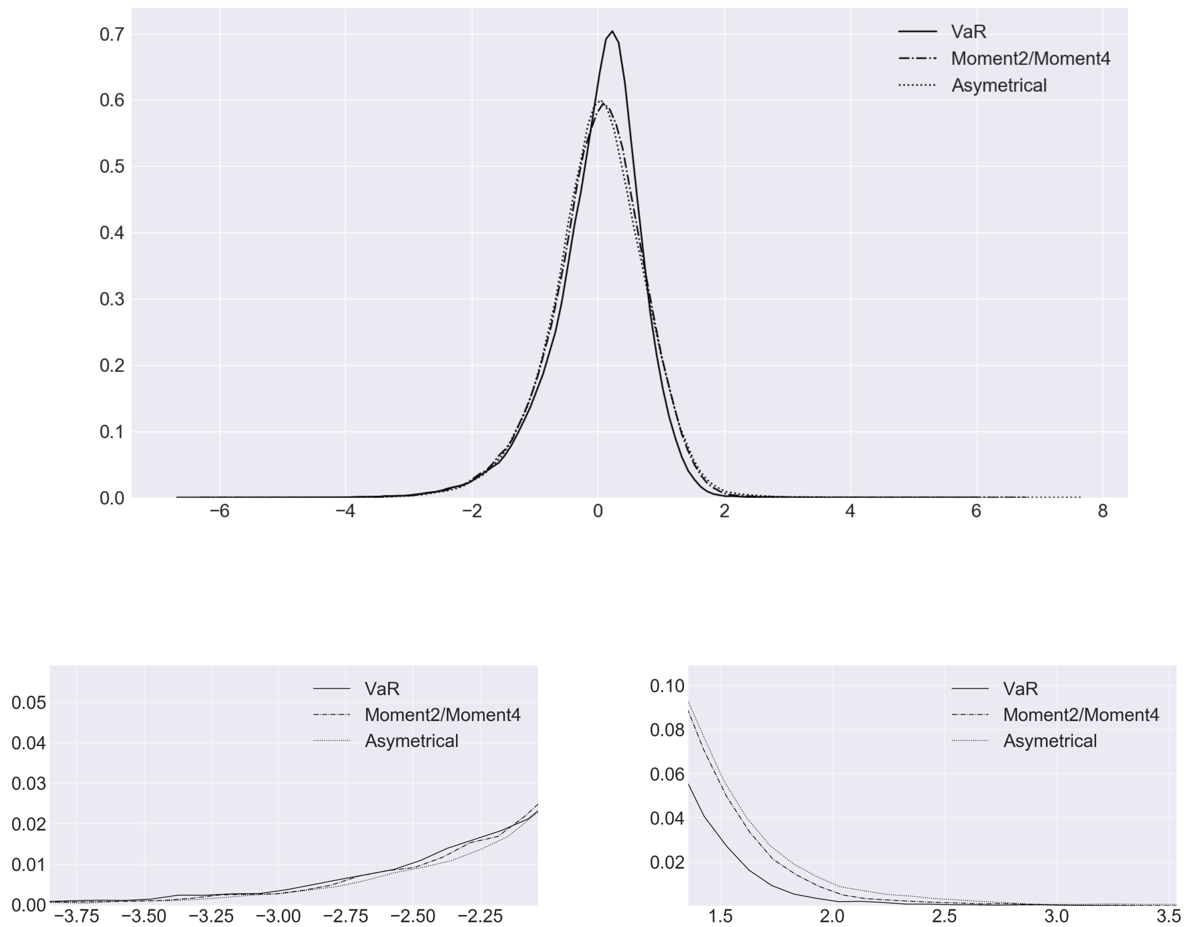
One of the advantage of the global neural network approach is its flexibility. There are no particular limitations on the models (markovian or not, gaussian or not ...) to use and we can chose different loss functions. In this section, we derive the optimal controls from different losses functions.
In Figures 9, 10 and 11, we plot the distribution of the hedged portfolio with the loss functions defined in Equations 7, 8 and 9. In general the non-symmetric losses functions give different shapes for the distributions. On the left hand side, both the asymmetrical loss curve and the Moment 2/Moment 4 loss curve are below the Mean Square loss curve. On the extreme left hand tail represented for example in Figure 11, the Mean Square loss function is the only one which is represented: extreme losses are avoided by Moment 2/Moment 4 and asymmetrical loss functions. This is paid on the average (middle of the distribution): there are more minor losses for the two non-symmetrical loss functions. Some of the distribution mass is deported on the right hand side (the gain side). This is an attractive side effect: compared to Mean Square error, L2/L4 and asymmetrical losses functions tends to favor gains.



Zoom on left hand tail

Zoom on right hand tail

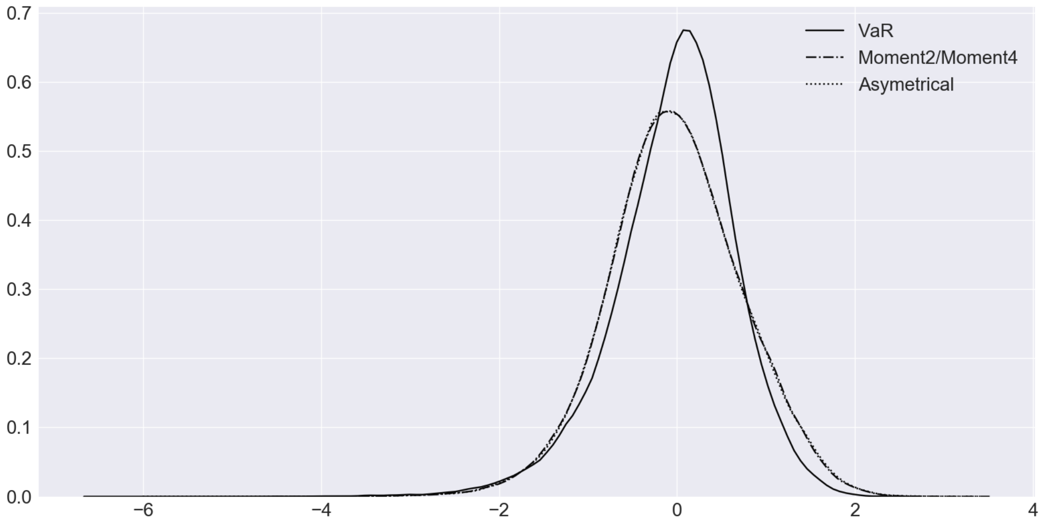
Figure 9: Distribution of the hedged portfolio for Case 1 and different risk criterion - Zoom on the tails

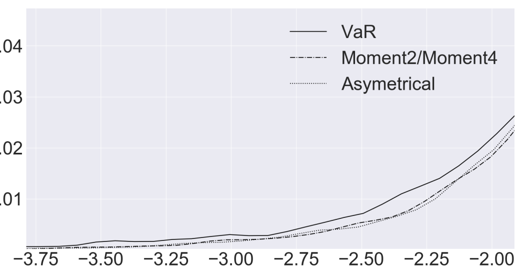


Zoom on left hand tail

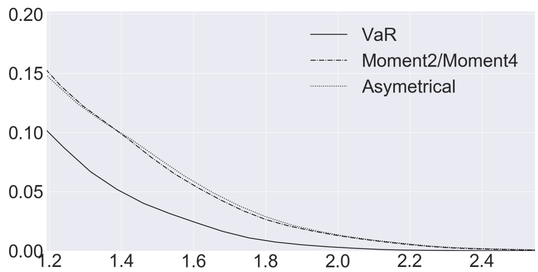
Zoom on right hand tail

Figure 10: Distribution of the hedged portfolio for Case 2 and different risk criterion - Zoom on the tails

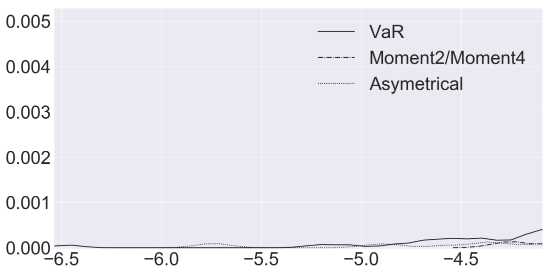




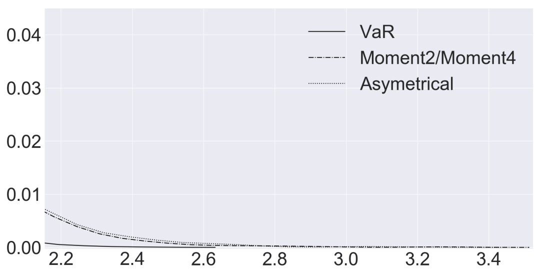
Zoom on left hand tail



Zoom on right hand tail



Zoom on extreme left hand tail



Zoom on extreme right hand tail

Figure 11: Distribution of the hedged portfolio for Case 3 and different risk criterion - Zoom on the tails

5 Numerical results for portfolio management problem with transaction costs

In this section, we investigate the effect of transaction costs when implemented in the global algorithm. We consider that the cost of selling or buying a volume k of F^i is equal to $k.c^i, c^i \geq 0$. As we sell the derivative the terminal wealth of the strategy X_T and associated transaction costs Y_T are equal to:

$$X_T = p + \sum_{j=1}^{d} \sum_{i=1}^{N-1} \Delta^j_{t_i} (S^j_{t_{i+1}} - S^j_{t_i}), \quad (34)$$

$$Y_T = \sum_{j=1}^{d} \sum_{i=1}^{N-1} |\Delta^j_{t_i} - \Delta^j_{t_{i-1}}| c_j. \quad (35)$$

We use the criterion defined by:

$$d^\alpha(X^\Delta, g(S_T)) = (1-\alpha)\mathbb{E}\left[Y_T\right] + \alpha \sqrt{\mathbb{E}\left[(X_T - g(S_T))^2\right]}, \alpha \in [0,1]. \quad (36)$$

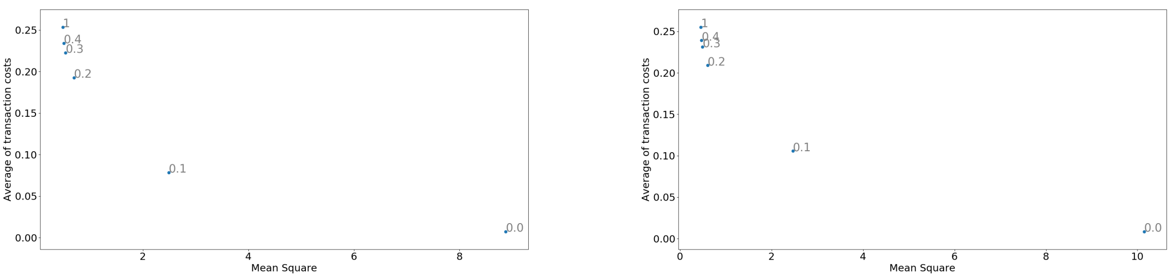
This criteria describes a trade-off between risk-limitation and hedging costs. If $\alpha = 1$, the criterion is equivalent to the variance minimization studied in Section 4.1.1; if $\alpha = 0$, we just minimize transaction costs regardless of risks (which corresponds to doing nothing).
$\alpha \in [0,1]$ is a parametrization of the Pareto frontier of the risk and transaction costs minimization trade-off. This problem is a portfolio management problem, where p is an input (so not optimized) that we take equal to $\mathbb{E}[g(S_T)]$ in our numerical tests.

5.1 Training the Pareto frontier

Instead of training N versions of the neural network for N values of α, we propose to add α to the input variables of the neural network (see Figure 4) and to randomly pick a value of α following a random uniform distribution $\mathcal{U}(0,1)$ at each training iteration. By doing this, we add a dimension to the problem but we obtain the optimal strategy for all $\alpha \in [0,1]$ at once. This goes against traditional algorithms where it is often preferred to evaluate N function defined on $\mathbb{R}^K$ instead of one function defined on $\mathbb{R}^{K+1}$. Getting the whole Pareto frontier is appealing for many reasons as it allows for example to retrieve the α corresponding to an expected transaction cost target budget.
To obtain the Pareto frontier estimate, we increase the width of the neural network (3 hidden layers of 50 - instead of the 10 previously - neurons for the projection part of the LSTM), and run 100 000 iterations of mini-batch gradient descent where 20 000 where sufficient until now. α is generated from a Sobol quasi random generator.

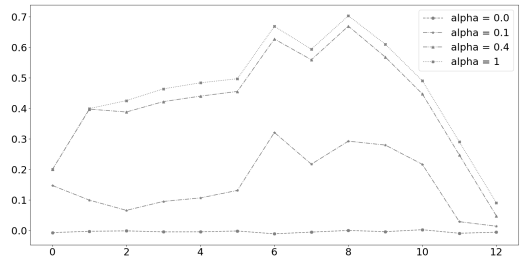
5.2 Numerical results

We consider the markets spreads option of Case 2. and Case 3. described in 4.1. The transaction cost is the same for all tradable risk factors and is set to 0.02 per unit of traded volume. In Figure 12 we plot the resulting average transaction cost and variance of hedged portfolio values for different α. As expected, when $\alpha \sim 1$, the strategy gives similar results to the pure variance minimization of Section 4.1.1; when $\alpha \sim 0$, we obtain results corresponding to a not-hedged portfolio. In Figures 13 and 14, the delta for Case 2 and Case 3 are plotted for some simulations with several α's. For lower α the algorithm prefer to reduce the control amplitude in order to reduce transaction costs.

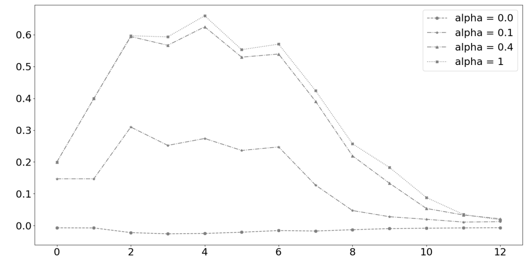


Case 2. Case 3.

Figure 12: Spread Option Mean Square VS Mean of transaction costs for various α and transaction cost of 0.02

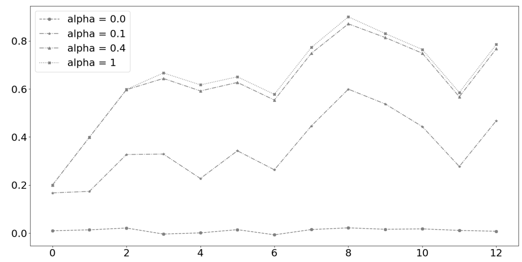


Sim Nb 1

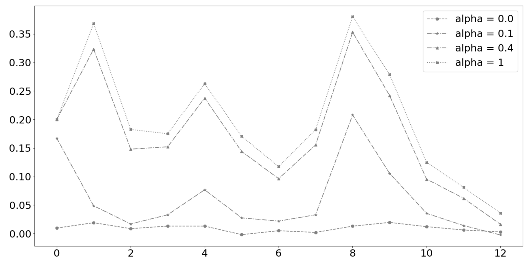


Sim Nb 2

Figure 13: Delta for Future 1 and Case 2. and various α



Sim Nb 1



Sim Nb 2

Figure 14: Delta for Future 1 and Case 3. and various α

6 Conclusion and perspectives

Three neural-network-based algorithms (two local algorithms and one global algorithm) dedicated to the hedging of contingent claim are proposed. The three algorithms show good results compared to stochastic-control-based techniques. In particular, the global algorithm is interesting both in terms of execution speed and flexibility.
The global algorithm is tested with different well known losses function and the use of an LSTM architecture in the global algorithm would allow to use some non-markovian underlying models. Moreover, we propose a methodology to draw a Pareto frontier. We apply this methodology to the trade-off between maximizing mean and minimizing variance in the transaction costs case (parameterized by an *alpha* combining mean and variance in the objective function). The advantage of getting the whole Pareto frontier is threefold:

- it increases inference speed as we do not need to retrain the algorithm with different parameterization;
- it becomes easy to do a retro-engineering (for example to get which α corresponds to a target transaction costs budget);
- it is easier to make sensitivity analysis;

The drawback of the global algorithm when compared to stochastic control-based algorithm is the lack of convergence proof. However, the global algorithm allows the treatment of cases that are not attainable by any other techniques.

References

M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mane, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. ViÃĺgas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

A. Bachouch, C. Huré, N. Langrené, and H. Pham. Deep neural networks algorithms for stochastic control problems on finite horizon, part 2: numerical applications. *arXiv preprint arXiv:1812.05916*, 2018.

B. Bouchard, X. Tan, and X. Warin. Numerical approximation of general lipschitz bsdes with branching processes. *arXiv preprint arXiv:1710.10933*, 2017.

R. Carmona. *Indifference pricing: theory and applications.* Princeton University Press, 2008.

Q. Chan-Wai-Nam, J. Mikael, and X. Warin. Machine learning for semi linear pdes. *Journal of Scientific Computing*, Feb 2019.

J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

H. Föllmer and P. Leukert. Quantile hedging. *Finance and Stochastics*, 3(3):251–273, 1999. ISSN 0949-2984.

J. Gatheral. No-dynamic-arbitrage and market impact. *Quantitative finance*, 10(7):749–759, 2010.

L. J. W. X. Gevret, H. Stopt, an open source library for stochastic control. *https://gitlab.com/stochastic-control/StOpt*, 2016.

E. Gobet, I. Pimentel, and X. Warin. Option valuation and hedging using asymmetric risk function: asymptotic optimality through fully nonlinear partial differential equations. *working paper*, Apr. 2018.

J. Han, A. Jentzen, and W. E. Solving high-dimensional partial differential equations using deep learning. *Proceedings of the National Academy of Sciences*, 115(34):8505–8510, 2018. ISSN 0027-8424.

S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

K. Hornik, M. Stinchcombe, and H. White. Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks. *Neural Networks*, 3(5):551–560, 1990.

C. Huré, H. Pham, A. Bachouch, and N. Langrené. Deep neural networks algorithms for stochastic control problems on finite horizon, part i: convergence analysis. *arXiv preprint arXiv:1812.04300*, 2018.

C. Huré, H. Pham, and X. Warin. Some machine learning schemes for high-dimensional nonlinear pdes. *arXiv preprint arXiv:1902.01599*, 2019.

Y. Kabanov and M. Safarian. *Markets with transaction costs: Mathematical Theory.* Springer Science & Business Media, 2009.

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

C.-A. Lehalle and S. Laruelle. *Market microstructure in practice.* World Scientific, 2013.

H. E. Leland. Option pricing and replication with transactions costs. *The journal of finance*, 40(5):1283–1301, 1985.

S. Liang. Why deep neural networks for function approximation? *arXiv:1610.04161*, 2017.

C. Olah. Understanding lstm networks. *colah's blog*, 2015.

M. Potters and J.-P. Bouchaud. More statistical properties of order books and price impact. *Physica A: Statistical Mechanics and its Applications*, 324(1-2):133–140, 2003.

M. Schweizer. A guided tour through quadratic hedging approaches. Technical report, Discussion Papers, Interdisciplinary Research Project 373: Quantification âĂę, 1999.

P. Tankov. *Financial modelling with jump processes*, volume 2. CRC press, 2003.

T. Tieleman and G. Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012.

K. B. Toft. On the mean-variance tradeoff in option replication with transactions costs. *Journal of Financial and Quantitative Analysis*, 31(2):233–263, 1996.

X. Warin. Variance optimal hedging with application to electricity markets. *Journal of computational finance*, To appear in 2020.

M. D. Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.