MathematicS

MathsInA.

In Action

Thomas Deschatre & Joseph Mikael

Deep combinatorial optimisation for optimal stopping time problems: application to swing options pricing.

Volume 11 (2022), p. 243-258.

https://doi.org/10.5802/msia.26

CENTRE
MERSENNE

MathematicS In Action *est membre du*
Centre Mersenne pour l'édition scientifique ouverte
http://www.centre-mersenne.org/
e-ISSN : 2102-5754

MathematicS In Action
Vol. 11, 243-258 (2022)

Deep combinatorial optimisation for optimal stopping time problems: application to swing options pricing.

Thomas Deschatre [*]
Joseph Mikael [**]

[*] EDF R&D & FiME, Laboratoire de Finance des Marchés de l'Energie
E-mail address: thomas-t.deschatre@edf.fr
[**] EDF R&D & FiME, Laboratoire de Finance des Marchés de l'Energie
E-mail address: joseph.mikael@edf.fr.

Abstract

A new method for stochastic control based on neural networks and using randomisation of discrete random variables is proposed and applied to optimal stopping time problems. The method models directly the policy and does not need the derivation of a dynamic programming principle nor a backward stochastic differential equation. Unlike continuous optimization where automatic differentiation is used directly, we propose a likelihood ratio method for gradient computation. Numerical tests are done on the pricing of American and swing options. The proposed algorithm succeeds in pricing high dimensional American and swing options in a reasonable computation time, which is not possible with classical algorithms.

1. Introduction

Motivation

Optimal stopping problems are particularly important for risk management as they are involved in the pricing of American options. American-style options are used not only by traditional asset managers but also by energy companies to hedge "optimised assets" by finding optimal decisions to optimise their P&L and find their value. A common modelling of a power plant unit P&L is done using swing options which are options allowing to exercise at most $l \geq 1$ times the option with possibly a constraint on the delay between two exercise dates (see [13] or [32]).

Formally, for $T > 0$, we are given a stochastic processes $(X_t)_{t\geq 0}$ defined on a probability space $(\Omega, \mathcal{F}, \mathbb{F} = (\mathcal{F}_t)_{t\geq 0}, \mathbb{P})$ and we search for an increasing sequence of $\mathbb{F}$ stopping times $\tau = (\tau_1, \tau_2, \dots, \tau_l)$ maximizing the expectation of the objective function

$$\mathbb{E}_{\mathbb{P}}\left(\sum_{i=1}^{l} f(\tau_i, X_{\tau_i})\mathbf{1}_{\tau_i \leq T}\right).$$

Numerical methods to solve the optimal stopping problem when $l = 1, f(t,x) = e^{-rt}g(x)$ and X is Markovian include:

- Dynamic programming equation: the option price P_0 is computed using the following backward discrete scheme over a grid $t_0 = 0 < t_1 < \cdots < t_N = T$:
$$\begin{aligned} P_{t_N} &= g(X_T), \\ P_{t_i} &= \max(g(X_{t_i}), e^{-r(t_{i+1}-t_i)}\mathbb{E}_{\mathbb{P}}(P_{t_{i+1}}|\mathcal{F}_{t_i})),\ i = 0, \dots, N-1. \end{aligned} \tag{1.1}$$
One then needs to perform regression to compute the conditional expectations, see [26] or [11].

Keywords: Optimal stopping, American option, Swing option, Combinatorial optimisation, Neural network, Artificial intelligence.
2020 *Mathematics Subject Classification:* 91G60, 60G40, 90C27, 97R40.

- Partial differential equation (PDE): a variational inequality derived from the Hamilton Jacobi Bellman equation is given by

  $$\min(-(\partial_t + \mathcal{L})v + rv, v - g) = 0,\ v(x,T) = g(x)$$

  where $\mathcal{L}$ is the infinitesimal generator of X [28, Chapter 8, Section 3.3]. A numerical scheme can be applied to solve this PDE and find the option value.

- Reflected Backward Stochastic Differential Equation (BSDE): the value of the American option is the solution of the reflected BSDE [16]:

  $$\begin{gathered} Y_t = g(X_T) - r\int_t^T Y_s \,\mathrm{d}s - \int_t^T Z_s \,\mathrm{d}W_s + K_T - K_t, \\ Y_t \geq g(X_t), \quad 0 \leq t \leq T, \\ \int_0^T (Y_t - g(X_t))\mathrm{d}K_t = 0. \end{gathered}$$

  [10] provides a numerical scheme to solve these equations.

- Policy search: the decision rule or exercise region is parametrized by a vector and the parameters are usually optimised by Monte Carlo methods as in reinforcement learning [19, Chapter 8, Section 2]; [2, 18]. The algorithm proposed in this paper is strongly related to this class of method.

These approaches generalise well for $l \geq 1$, see [13] for dynamic programming principle or [9] for the BSDE method. The non linear case where f is of the form $\phi(\sum_{i=1}^{l} e^{-r\tau_i} g(X_{\tau_i})\mathbf{1}_{\tau_i \leq T})$ is studied by [31]. We refer to [19, Chapter 8] for more exhaustive details on numerical methods for American option pricing. All these algorithms suffer from the curse of dimensionality: the number of underlying is hardly above 5. However energy companies portfolio may trade derivatives involving more that 4 commodities at one time (e.g. swing options indexed on C02, natural gas, electricity, volume, fuel) and traditional numerical methods hardly provide good solutions in a reasonable computing time.

Recently, neural network-based approaches have shown good results regarding stochastic control problems and PDE numerical resolution in high dimension, see [14, 21, 29]. In the following, one describes literature related to optimal stopping time problems using neural networks. [6, 25] use neural networks for regression in the dynamic programming equation (1.1). [3, 5, 22] also use the dynamic programming equation (1.1) but neural networks are used to parameterize the optimal policy. Weights and bias of the neural network(s) minimise at each time step the right hand side of the dynamic programming equation (1.1), going backward. The optimal decision consists in a continuous variable (instead of a discrete one) taking value in $(0, 1)$ modeled by a neural network. [15, 21, 23] use neural networks to solve BSDE's. In [23], the neural networks parameterizing the solution and eventually its gradient minimise the L^2 loss between the left hand-side and the right hand side of the Euler discretisation of the BSDE, going backward from the terminal value. [3] and [23] need to maximise one criteria by time step. The approaches of [21] and [15] are quite different: the neural network allows the parameterisation of the initial value of the BSDE and the gradient at each time step, and it minimises the distance between the terminal value obtained by the neural network and the terminal value of the BSDE, going forward. American put options prices are computed in [23] up to dimension 40 with 160 time steps. Neural networks approaches have also been used in the context of swing options pricing in gas market in [4]. The definition of swing options slightly differs from ours as it considers a continuous control: the option owner buys a certain amount of gas between a minimum and a maximum quantity. It is however related to our problem as in continuous time, this option is bang-bang: it is optimal to exercise at the minimum or the maximum level at each date, that is choosing between two actions. [4] directly models the policy by a neural network and optimises the objective function as in [12, 17].

Contrarily to [3, 6, 15, 21, 22, 23, 25], the goal of this paper is to propose a reinforcement learning algorithm to solve optimal multi-exercise (rather than one single) stopping time problems with constraints on exercise times that does not need to derive a dynamic programming equation nor to find an equivalent BSDE of the problem. The only information needed is the dynamic of the state process X and the objective function. This kind of algorithm is called policy gradient and is well known in the area of reinforcement learning, see [30] for instance. Although continuous control approximation with reinforcement learning shows good results, see [12, 17] for European-style option hedging, the case of optimal stopping times is more difficult as it involves controls taking values in a discrete set of actions. The problem is similar to a combinatorial optimisation one: at each time step, an action belonging to a finite set needs to be taken. One way to solve this problem is to perform a relaxation assuming that the control belongs to a continuous space. For instance, if one needs to price an American option, a decision represented by a value in $\{0,1\}$ and consisting in exercising or not must be taken. Relaxing the problem consists in searching for solutions in $[0,1]$: this relaxation has successfully been applied to a Bermudan option pricing in a high dimensional setting (up to 1000) in [7]. These methods apply well for American-style option pricing but seem to be not flexible enough to be extended to swing options pricing.

Main results

Our approach follows the spirit of [17] and [7]: one directly parameterises the optimal policy by a neural network and maximises the objective function moving forward. We propose an algorithm using reinforcement learning in order to solve optimal stopping times problem seen as an combinatorial optimisation problem. Note that solving combinatorial optimisation problems with neural networks have been considered in [8] in a deterministic framework without dynamics on the state process. The stochastic optimization framework considered in this paper is described in Section 2.

Neural network hardly handles integer outputs which is the main difficulty of the problem addressed in this paper. To encompass this problem, the first step of the algorithm consists in randomizing the optimization variables (that is executing the option or not) and modelling their law by a neural network. The second step consists in computing the gradient of the objective function. It can not be computed as usual by automatic differentiation as the neural network does not output the optimization variables but their law. The use of likelihood ratio method allows to rewrite the gradient as a function of the neural network output gradient that can be computed with automatic differentiation. The algorithm is given in Section 3. Compared to the papers referenced above our approach allows to solve stopping time problems without any knowledge of the dynamic programming equation or of an equivalent BSDE. Furthermore, it presents many advantages as it

- can solve multiple optimal stopping time problems;
- allows to add in a flexible way any constraint on the stopping times;
- can then be associated with the one of [17] considering continuous actions in order to solve stochastic impulse control problems, combining discrete and continuous controls (see [27, Chapter 6] for more information on impulse control problems).

Let us also notice that our method does not take advantage of the linearity (possible inversion between the sum and the expectation) of the considered optimal stopping problems contrarily to [7] and can be applied to non linear problem, making it suitable for impulse control problems. The theoretical convergence study of our algorithm is out of the scope of this paper.

Numerical tests covering Bermudan and swing options are proposed in Section 4 and show good results in the pricing of 10 underlyings Bermudan option and also on 5 underlyings swing options having up to $l = 6$ exercise dates. However, our algorithm gives suboptimal results on one of the considered case.

2. Optimal stopping

2.1. Continuous time modelling

We are given a financial market operating in continuous time. Let $(\Omega, \mathbb{F} = (\mathcal{F}_t)_{t\geq 0}, \mathcal{F}, \mathbb{P})$ a filtered probability space and W a d-dimensional $\mathbb{F}$-Brownian motion. One assumes that $\mathbb{F}$ satisfies the usual conditions of right continuity and completeness. Let $T > 0$ a finite horizon time and $X = (X_1, X_2, \dots, X_d)$ be the unique strong solution of the Stochastic Differential Equation (SDE):

$$X_t = X_0 + \int_0^t \mu(s, X_s)\,\mathrm{d}s + \int_0^t \sigma(s, X_s)\,\mathrm{d}W_s,\ t \in [0, T]\,, \tag{2.1}$$

with $\mu : [0,T] \times \mathbb{R}^d \mapsto \mathbb{R}^d$ and $\sigma : [0,T] \times \mathbb{R}^d \mapsto \mathbb{R}^{d\times d}$ two measurable functions verifying $|\mu(t,x) - \mu(t,y)| + \|\sigma(t,x) - \sigma(t,y)\| \leq K_1|x-y|$ and $|\mu(t,x)| + \|\sigma(t,x)\| \leq K_2(1+|x|)$ for $x, y \in \mathbb{R}^d$ and $t \in [0,T]$ ($|\cdot|$ denotes the Euclidian distance in $\mathbb{R}^d$ and for a matrix $A \in \mathbb{R}^{d\times d}$, $\|A\| = \sqrt{tr(AA^\top)}$) and $K_1, K_2 \in \mathbb{R}$. Using the notations of [13] and with X as defined in (2.1) for $t \in [0,T]$ and $X_t = X_T$ for $t \geq T$, an optimal stopping time problem consists in solving the problem

$$\sup_{\tau\in\mathcal{S}^l} \mathbb{E}_{\mathbb{P}}\left(\sum_{i=1}^{l} f(\tau_i, X_{\tau_i})\mathbf{1}_{\tau_i\leq T}\right) \tag{2.2}$$

where $\mathcal{S}^l$ is the collection of all vectors of increasing stopping times $\tau = (\tau_1, \dots, \tau_l)$ such that for all $i = 2, \dots, l$, $\tau_i - \tau_{i-1} \geq \gamma$ a.s. on the set of events $\{\tau_{i-1} \leq T\}$ and where $f : [0,T] \times \mathbb{R}^d \mapsto \mathbb{R}$ is a measurable function. $l \in \mathbb{N}^* = \mathbb{N} \setminus \{0\}$ corresponds to the number of possible exercises and $\gamma \geq 0$ to the minimum delay between two exercise dates. One wants to find the optimal value (2.2) but also the optimal policy

$$\tau^* \in \arg\max_{\tau\in\mathcal{S}^l} \mathbb{E}_{\mathbb{P}}\left(\sum_{i=1}^{l} f(\tau_i, X_{\tau_i})\mathbf{1}_{\tau_i\leq T}\right). \tag{2.3}$$

2.2. Discrete time modelling

In practice, one only considers optimal stopping on a discrete time grid (for instance, the valuation of a Bermudan option is used as a proxy of the American or swing option). Let us consider $N+1$ exercise dates belonging to a discrete set $\mathcal{D}_N = \{t_0 = 0 < t_1 < \dots < t_N = T\}$, $N \in \mathbb{N}^*$. The problem consists in finding

$$\sup_{\tau\in\mathcal{S}_N^l} \mathbb{E}_{\mathbb{P}}\left(\sum_{i=1}^{l} f(\tau_i, X_{\tau_i})\mathbf{1}_{\tau_i\leq T}\right) \tag{2.4}$$

where S_N^l is the set of stopping times belonging to S^l (set of increasing stopping times vectors including delay constraints) such that $\tau_i \in \mathcal{D}_N$ on $\{\tau_i \leq T\}$, for $i = 1, \dots, l$. This discretisation is needed for our algorithm as it is needed in classical methods such as [26]. Problem (2.4) is equivalent to the following:

$$\sup_{Y} \mathbb{E}_{\mathbb{P}}\left(\sum_{i=0}^{N} Y_i f(t_i, X_{t_i})\right) \tag{2.5}$$

where $(Y_i)_{i=0,\dots,N}$ is a sequence of $(\mathcal{F}_{t_i})_{i=0,\dots,N}$-measurable random variables taking values in $\{0,1\}$ such that

$$\sum_{i=0}^{N} Y_i \leq l \tag{2.6}$$

and

$$D_j \geq \gamma,\ j = 0,\dots,N, \tag{2.7}$$

with

$$D_j = \gamma + t_j - \sum_{i=0}^{j-1} Y_i D_i \tag{2.8}$$

the delay at t_j from the last exercise assuming that we can exercise at time 0 (hence the γ term in (2.8) allowing to start at time 0 with a delay γ) and with the convention $\sum_{i=0}^{-1} \cdot = 0$. Given a solution $(Y_i^*)_{i=0,\dots,N}$ of Problem (2.5), a proxy for the optimal control (2.3) is given on the event $\{\sum_{i=0}^{N} Y_i = m\}$, $m \leq l$, by

$$\tau_k^* = \begin{cases} t_{m_k} & \text{if } k \leq m \\ \infty & \text{otherwise} \end{cases}, \quad k \in \{1,\dots,l\},$$

with $m_k = \min\{j \in \{0,\dots,N\} | \sum_{i=0}^{j} Y_i^* \geq k\}$ the index of the k^{th} exercise.

3. Algorithm description

3.1. Neural network parametrization

As the Y_i's are discrete we cannot assume that they are the output of a neural network which weights are optimised by applying a stochastic gradient descent (SGD). To overcome this difficulty, one can randomize Y and consider that at each time step $t_j, j \in \{1,\dots,N\}$, the discrete variable Y_j is a Bernoulli distributed random variable conditionally on $\mathcal{F}_{t_j} \cup \sigma(Y_i, i \leq j-1)$ (and $\mathcal{F}_{t_0}$ if $j=0$). The Bernoulli distribution parameter depends on the state variable of our control problem. This state variable, denoted S_{t_j}, is

- X_{t_j} in the case of a Bermudan option, that is with only one execution date,
- $(X_{t_j}^1,\dots,X_{t_j}^d, \sum_{i=0}^{j-1} Y_i, D_j)$ when there are constraints (2.6) and (2.7).

Of course, one can adapt the state depending on the constraints (for instance if there is no delay constraint (2.7), the state is $(X_{t_j}^1,\dots,X_{t_j}^d, \sum_{i=0}^{j-1} Y_i))$. In the case where X is a non Markovian process, one needs to consider that the probability for Y_j to be equal to 1 is a function of all the values of X_{t_i} for $i \leq j$ and Y_i for $i \leq j-1$. In this case, one could use a Recurrent Neural Network to parameterise this function but we do not consider this case here. The Bernoulli distribution parameter, which is $\mathbb{P}(Y_j = 1|S_{t_j})$ is parameterised by a neural network $\mathbb{NN}$ defined on $[0,T] \times \mathbb{S} \times \Theta$ and taking values in $\mathbb{R}$ where $\mathbb{S}$ is the state space and Θ represents the sets in which the biases and weights of the neural network lie. The neural network architecture is described in Section 3.3. The parametrization is then the following:

$$\mathbb{P}(Y_j = 1|S_{t_j}) = \operatorname{expit}(C \tanh(\mathbb{NN}(t_j, S_{t_j}, \theta)))c(S_{t_j}),\ j = 0,\dots,N, \tag{3.1}$$

with expit : $\mathbb{R} \mapsto (0,1)$ and $\operatorname{expit}(x) = \frac{1}{1+e^{-x}}$ for $x \in \mathbb{R}$ and $c(S_{t_j})$ is equal to 0 if the constraints are saturated and 1 otherwise. Typically, if there are no constraints, c is always equal to 1, and if there are constraints (2.6) and (2.7),

$$c(S_{t_j}) = \mathbf{1}_{\sum_{i=0}^{j-1} Y_i < l} \mathbf{1}_{D_j \geq \gamma}.$$

Note that the methodology can be extended to any constraint on the policy. $C\tanh(\mathbb{NN}(x,\theta))$ outputs the *logit* (the inverse function of expit) of $\mathbb{P}(Y_j = 1|S_{t_j})$ (when constraints are not saturated). The function *tanh* is not necessary and one could only consider $\mathbb{NN}$ to parameterise the logit of the probability. To reduce the values taken by the *logit*, we bound the output of the neural using *tanh* and choose C such that $\operatorname{expit}(-C) \approx 0$ and $\operatorname{expit}(C) \approx 1$; C is given in Section 3.4.

From now on, $\mathbb{P}$ is replaced by $\mathbb{P}_\theta$ to indicate the dependence of the law of Y with θ. At this step, we still cannot train our neural networks by applying a stochastic gradient descent because of the Y's randomization.

3.2. **Optimization**

To approximate a solution to (2.5) we search for

$$\theta^* \in \underset{\theta\in\Theta}{\arg\max}\, \mathbb{E}_{\mathbb{P}_\theta}\left(\sum_{i=0}^{N} Y_i f(t_i, X_{t_i})\right).$$

Classical neural network parameter optimization consists first in evaluating the objective function

$$\mathbb{E}_{\mathbb{P}_\theta}\left(\sum_{i=0}^{N} Y_i f(t_i, X_{t_i})\right)$$

using Monte Carlo method and replacing it by

$$\frac{1}{N_{\text{batch}}}\sum_{m=1}^{N_{\text{batch}}}\sum_{i=0}^{N} Y_i^m f(t_i, X_{t_i}^m)$$

with $N_{\text{batch}} \in \mathbb{N}^*$ and where X^m and Y^m correspond to one realization of (X,Y) simulated according to (2.1) for X and to $\mathbb{P}_\theta$ for Y. Secondly, a gradient descent is done to update the parameter θ using gradient of the objective function which is computed using backpropagation. However in our case, it is not possible to directly use backpropagation: Y is not a function of θ, but a discrete variable with law depending on θ.

To bypass this problem, we use a likelihood ratio method, see [19, Section 7.3]. Let us consider a random variable $Z : \Omega \mapsto E$ with probability measure $\mathbb{Q}_a$, $a \in \mathbb{R}^d$, absolutely continuous with respect to a measure $\mathbb{Q}$. Let $l_a(x) = \frac{\mathrm{d}\mathbb{Q}_a}{\mathrm{d}\mathbb{Q}}(x)$ be the likelihood function. We have, under some integrability conditions, $\nabla_a \mathbb{E}_{\mathbb{Q}_a}(Z) = \mathbb{E}_{\mathbb{Q}_a}(Z\nabla_a \log(l_a(Z)))$. Using this method and iterative conditioning for probability computation, we find that the gradient of $\mathbb{E}_{\mathbb{P}_\theta}(\sum_{j=0}^{N} Y_j f(t_j, X_{t_j}))$ is given by:

$$\begin{aligned}\nabla_\theta \mathbb{E}_{\mathbb{P}_\theta}&\left(\sum_{j=0}^{N} Y_j f(t_j, X_{t_j})\right)\\ &= \mathbb{E}_{\mathbb{P}_\theta}\left(\sum_{j=0}^{N} Y_j f(t_j, X_{t_j})\left(\sum_{i=0}^{N} Y_i \nabla_\theta \log(\mathbb{P}_\theta\left(Y_i = 1|S_{t_i}\right))\right.\right.\\ &\qquad\left.\left. + (1-Y_i)\nabla_\theta \log(1-\mathbb{P}_\theta(Y_i = 1|S_{t_i}))\right)\right). \end{aligned} \tag{3.2}$$

$\mathbb{P}_\theta(Y_i = 1|S_{t_i})$ which is defined in Equation (3.1) is a continuous function of the neural network: the gradients appearing in Equation (3.2) can be easily computed using backpropagation.

The main drawback of this method is the high variance of the right hand side of (3.2). Several methods exist to bypass this issue, the most used with neural network optimisation being the use of a baseline, see [33].

Every Δ_{test} steps, the objective value is computed over the testing set. The parameters kept at the end are the ones minimising those evaluations. The objective function is finally evaluated on a validation set. While on the training phase actions are sampled from the outputted probability on the training set, they are chosen equal to 1 if the probability is greater than 0.5 and 0 otherwise on the test and validation sets. The algorithm is described in Algorithm 1 with hyperparameters given in Section 3.4.

Remark 3.1. The method does not take any advantage of the fact that we can exchange the sum and the expectation in (2.5). It is then suitable for optimization problems of the form

$$\sup_{Y} \mathbb{E}_{\mathbb{P}}\left(g(Y_1, t_1, X_{t_1}, \ldots, Y_N, t_N, X_{t_N})\right)$$

and could be combined with forward neural network continuous optimization algorithms such as those of [12, 17] to solve impulse control problems.

Algorithm 1 Algorithm for optimal stopping. The lines in blue are the main difference with classical backpropagation.

```
$\alpha$: Learning rate
$\beta_1$, $\beta_2 \in [0,1]$ : Exponential decay rates for the moment estimates,
$N_{\text{iter}}$ : number of iterations
$N_{\text{batch}}$ : number of simulations at each gradient descent iteration (batch size)
$\theta_0$ randomly chosen
$m_0 \leftarrow 0$
$v_0 \leftarrow 0$
for $i_{\text{iter}} = 1 \ldots N_{\text{iter}}$ do
    for $u = 0 \ldots N$ do
        $X_u \leftarrow N_{\text{batch}}$ samples simulations of $X_{t_u}$
        $S_u \leftarrow$ state value
        $p_{\theta_{i_{\text{iter}}-1},u} \leftarrow \text{expit}(C \tanh(\mathbb{NN}(t_u, S_u, \theta_{i_{\text{iter}}-1})))c(S_u)$
        $Y_u \leftarrow N_{\text{batch}}$ of a Bernouilli r.v. with parameter $p_{\theta_{i_{\text{iter}}-1},u}$
    $g_{i_{\text{iter}}} \leftarrow \frac{1}{N_{\text{batch}}} \sum_{n=1}^{N_{\text{batch}}} \left(\sum_{u=0}^{N} Y_u^n f(t_u, X_u^n)\right)$
            $\times \left(\sum_{u=0}^{N} Y_u^n \nabla_\theta \log(p_{\theta_{i_{\text{iter}}-1},u}) + (1 - Y_u^n)\nabla_\theta \log(1 - p_{\theta_{i_{\text{iter}}-1},u})\right)$
    $m_{i_{\text{iter}}} \leftarrow m_{i_{\text{iter}}-1} + (1-\beta_1) g_{i_{\text{iter}}}$ (update biased first moment estimate)
    $v_{i_{\text{iter}}} \leftarrow \beta_2 v_{i_{\text{iter}}-1} + (1-\beta_2) g_{i_{\text{iter}}}^2$ (update biased second raw moment estimate)
    $\widehat{m}_{i_{\text{iter}}} \leftarrow \frac{m_{i_{\text{iter}}}}{1-(\beta_1)^{i_{\text{iter}}}}$ (computes bias-corrected first moment estimate)
    $\widehat{v}_{i_{\text{iter}}} \leftarrow \frac{v_{i_{\text{iter}}}}{1-(\beta_2)^{i_{\text{iter}}}}$ (computes bias-corrected second raw moment estimate)
    $\theta_{i_{\text{iter}}} \leftarrow \theta_{i_{\text{iter}}-1} - \alpha \widehat{m}_{i_{\text{iter}}} / (\sqrt{\widehat{v}_{i_{\text{iter}}}} + \epsilon)$ (update parameters)
```

3.3. Neural network architecture

The neural network architecture is inspired by [14] and consists in one single feed forward neural network which features are the time step t_i and the current state realisation S_{t_i}. Let $t \in [0,T]$ and $x = (x_1, \ldots, x_r)^\top \in \mathbb{S}$ that we assume to be included in $\mathbb{R}^r$. The neural network is defined as follow

$$\mathbb{NN}(t, x, \theta) = A_{L+1} \circ \rho \circ A_L \circ \rho \cdots \circ A_1((t, x_1, \ldots, x_r)^\top)$$

where $A_l(y) = W_l y + b_l$ for $l = 1, \ldots, L+1$, $W_1 \in \mathbb{R}^{m\times(r+1)}$, $W_l \in \mathbb{R}^{m\times m}$ for $l = 2, \ldots, L$, $W_{L+1} \in \mathbb{R}^{1\times m}$, $b_l \in \mathbb{R}^m$ for $l = 1, \ldots, L$ and $b_{L+1} \in \mathbb{R}$. L corresponds to the number of hidden layers and m to the number of neurons per layer (that we assume to be the same for every layer). The $(W_l)_{l=1,\ldots,L+1}$ correspond to the weights and $(b_l)_{l=1,\ldots,L+1}$ to the biases. The function ρ is the activation function and is chosen as the ReLu function, that is $\rho(x) = \max(0, x)$. θ is then equal to $(W_1, \dots, W_{L+1}, b_1, \dots, b_{L+1})$.

3.4. Hyper parameters

- The *batch size* N_{batch} as the *number of iterations* N_{iter} depend on the use case and are specified at a later stage. The training set size is then equal to $N_{\text{iter}} \times N_{\text{batch}}$. As the likelihood ratio estimator of the gradient has high variance, choosing a large batch size (>1000) allows for a better estimation. The drawback is that it tends to slow down the algorithm. To reduce the variance, one could also use a baseline function as in [33].

- The *test set size* is chosen equal to 500 000 and the *validation set size* to 4 096 000 (500 000 and 4 096 000 are chosen high to have very accurate optimisation). The test set is evaluated every $\Delta_{\text{test}} = 100$ steps.

- The *number of hidden layers* L is chosen equal to 3. The *number of neurons* m per layer is constant (but can vary from a case to another).

- The *learning rate* (α in Algorithm 1) is chosen equal to 0.001.

- Since we use the same network at each time step, we use a mean-variance *normalisation* over all the time steps to center all the inputs (t_i, X_{t_i}) for all t_i's with the same coefficients. The scaling and recentering coefficients are estimated on 100 000 pre-simulated data that is just used to this end. The mean and the standard deviation are first computed for every time step over the simulations then averaged over the time steps.

- We use Xavier *initialisation* [20] for the weights and a zero initialisation for the biases.

- The parameter C that bounds the input of the expit function is chosen such that $\text{expit}(-C) \approx 0$ and $\text{expit}(C) \approx 1$. We choose $C = 10$.

- The library used is [1] and the algorithm runs on a laptop with 8 cores of 2.50 GHz, a RAM memory of 15.6 Go and without GPU acceleration.

4. Numerical results

In this section Algorithm 1 is applied to the valuation of Bermudan and swing options. The function $f(s, x)$ is of the form $e^{-rs} g(x)$ where g is the payoff of the option and $r \geq 0$ is the risk free rate. We place ourselves in the Black–Scholes framework: $\mu(s, x) = (r - \delta)x$, with $\delta \geq 0$ corresponding to the dividend rate and $\sigma(s, x) = \text{diag}(x)\Sigma$ with Σ a positive definite matrix. We choose to work with a regular time grid $t_i = \frac{i}{N}$ for $i = 0, \dots, N$. The probability measure corresponds to the risk neutral probability and finding the value of the option consists in solving Problem (2.5).

4.1. Bermudan options

In this section, we assume that $l = 1$ (only one exercise) and we consider different options to price.

Put option. With $d = 1$, payoff $g(x) = (K - x)^+$, $K = 1$, $S_0 = 1$, $r = 0.05$, $\delta = 0$, $\Sigma = 0.2$, $N = 10$, $T = 1$. We consider a batch size equal to $N_{\text{batch}} = 5000$, a neural network with a depth of $L = 3$ hidden layers having $m = 10$ neurons each and $N_{\text{iter}} = 5000$ iterations.

Max-call option. With $d \in \{2, 10\}$, payoff $g(x) = (\max((x_i)_{i=1,\dots,d}) - K)^+$, $K = 100$, $S_0^i = 100$, $i = 1, \dots, d$, $r = 0.05$, $\delta = 0.1$, $\Sigma = 0.2 I_d$ (I_d is the identity matrix with size $d \times d$), $N = 9$, $T = 3$. We consider a batch size equal to $N_{\text{batch}} = 5000$ for $d = 2$ and $N_{\text{batch}} = 12000$ for $d = 10$, a neural network with $L = 3$ hidden layers of size $m = 30$ for $d = 2$ and $m = 70$ for $d = 10$ and $N_{\text{iter}} = 10000$ iterations.

Strangle spread option. With $d = 5$, payoff $g(x) = -(K_1 - \frac{1}{5}\sum_{i=1}^5 x_i)^+ + (K_2 - \frac{1}{5}\sum_{i=1}^5 x_i) + (\frac{1}{5}\sum_{i=1}^5 x_i - K_3)^+ - (\frac{1}{5}\sum_{i=1}^5 x_i - K_4)^+$, $K_1 = 75$, $K_2 = 90$, $K_3 = 110$, $K_4 = 125$, $S_0^i = 100$, $i = 1, \dots, 5$, $r = 0.05$, $\delta = 0$,

$$\Sigma = \begin{pmatrix} 0.3024 & 0.1354 & 0.0722 & 0.1367 & 0.1641 \\ 0.1354 & 0.2270 & 0.0613 & 0.1264 & 0.1610 \\ 0.0722 & 0.0613 & 0.0717 & 0.0884 & 0.0699 \\ 0.1367 & 0.1264 & 0.0884 & 0.2937 & 0.1394 \\ 0.1641 & 0.1610 & 0.0699 & 0.1394 & 0.2535 \end{pmatrix},$$

$N = 48$, $T = 1$. We consider a batch size equal to $N_{\text{batch}} = 8000$, a neural networks with $L = 3$ hidden layers of size $m = 60$ and $N_{\text{iter}} = 10000$ iterations.

Losses and times obtained with Algorithm 1 are given in Table 4.1 for each case and losses are compared to a reference value ([10] for the put option and [7] for the other options). The algorithm succeeds in pricing Bermudan options with a high precision (relative error <1%) in dimension up to $d = 10$ and number of time steps up to $N = 50$. The computing time is more sensitive to the number of time steps than to the dimension: the number of neural network estimation is equal to the number of time steps. The increase of computing time when dimension increases is mostly caused by a need to increase the batch size and a more important simulation time. Algorithm 1 succeeds in pricing Bermudan options and solves problems that are usually hard to solve and very expensive in terms of computation time as they suffer from the curse of dimensionality. The training and testing learning curves are given in Figure 4.1. The testing errors are relatively stable for the put and the max-call options but less stable for the strangle spread. For the put and the 2 dimensional max-call, the testing error converges quickly to the optimal value. The testing error of the 10 dimensional max-call decreases more slowly. The different training errors are all noisy as they are of smaller size.

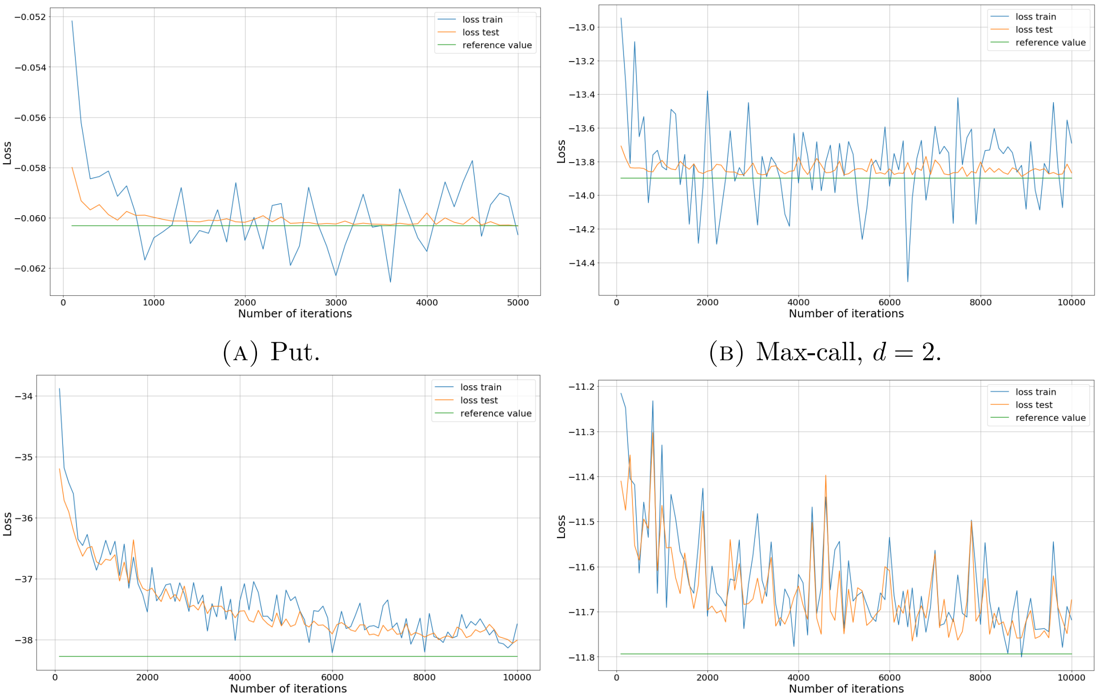
Once trained, the neural network allows to compute the probability to exercise according to the price and time to maturity, then the exercise region in a few seconds, see Figure 4.2 for the Bermudan put option. The probability to exercise has a S-shape with limit values 0 and 1 (do not exercise and exercise) and a small transition region between those two values. As expected, the first value such that the probability becomes 0 increases when time to maturity decreases. This is confirmed in the exercise region in Figure 4.1 with the frontier decreasing with time to maturity. The exercise region is the one below the curve, which is computed using the first value such that the probability of exercise is below 0.5. The frontier is extrapolated from the neural network trained only on a discrete time grid but that allows to use different time to maturity values (even ones not used in the training), which is not possible with classical backward optimisation.

4.2. **Swing options without delay**

In this section, we consider a swing option without delay constraint. We compare in Table 4.2 the results obtained by Algorithm 1 with the results of [24] in the case of a put option with $d = 1$, $g(x) = (K - x)^+$, $K = 40$, $S_0 \in \{35, 40, 45\}$, $r = 0.0488$, $\delta = 0$, $\Sigma = 0.25$, $N = 12$, $T = 0.25$, $l \in \{1, 2, 3, 4, 5, 6\}$ and no delay. We consider a batch size equal to $N_{\text{batch}} = 2000$, a neural networks with $L = 3$ hidden layers with size $m = 10$ and $N_{\text{iter}} = 5000$ iterations. Every case takes around 4 minutes to converge, see Table 4.3. The algorithm gives very accurate results

TABLE 4.1. Results obtained on different Bermudan options pricing with Algorithm 1 with the relative difference between Algorithm 1 and a reference value (given by [11] for the put option and by [7] for the other options). The time in seconds corresponds to the time of training and predicting.

Use case / Method	Algorithm 1	Reference	Difference	Time (s)
Bermudan put	0.0603	0.0603	0.06%	393.9
Max-call, d = 2	13.8787	13.8990	0.15%	1377.8
Max-call, d = 10	38.0347	38.2780	0.64%	5968.7
Strangle spread	11.7681	11.7940	0.22%	12991.3



(A) Put.

(B) Max-call, $d = 2$.

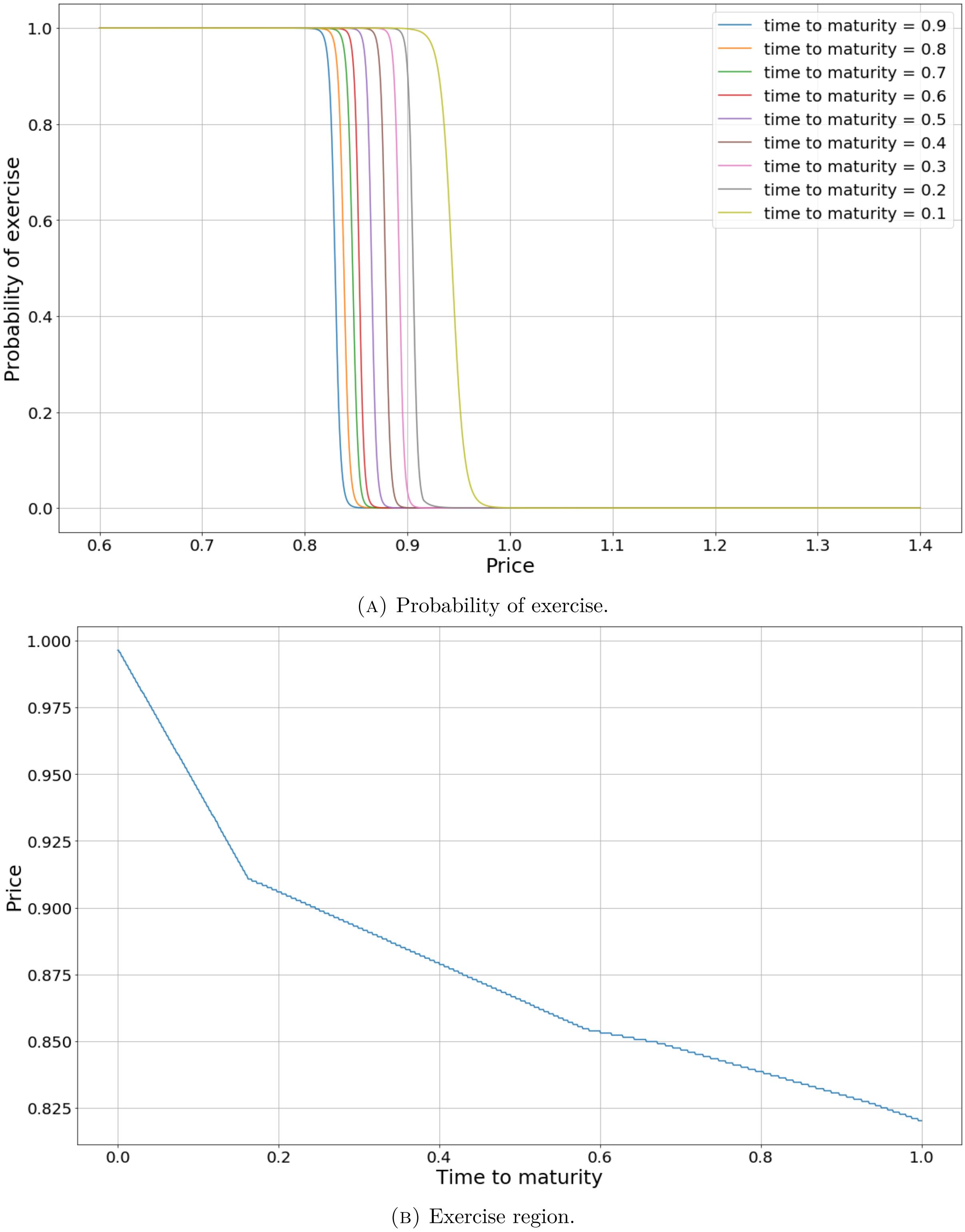
(C) Max-call, $d = 10$.

(D) Strangle spread.

FIGURE 4.1. Learning curves for the different Bermudan options.

in a short period of time for the valuation of the swing options. As for the Bermudan put option, it is possible to compute the probability of exercise and the exercise region for this swing option, see Figure 4.3. Those quantities depend now on the remaining number of exercises. As expected, the first value such that the probability to exercise is 0 increases with the remaining number of exercises. The exercise frontier should increase with the remaining number of exercises which is the case most of the time: the curve for a remaining number of exercises equal to 6 goes below the ones with a remaining number of exercises equal to 5 (resp. 4) when time to maturity is greater than 0.1 (resp. 0.15). To improve the convergence of the case $l = 6$, one could use the results from the case $l = 5$ by for instance constraining the exercise region in Figure 4.3 to be above the one of the case $l = 5$.

To assess the performance of Algorithm 1 in high dimension, let us consider the pricing of the geometrical put option having payoff $g(x) = (K - \prod_{i=1}^{d} x_i)^+$. Let $d = 5$, $K = 40$, $S_0^i = 40^{1/5}$, $r = 0.0488$, $\delta = \frac{4r}{5}$, $\Sigma = \frac{0.25}{\sqrt{5}} I_5$, $N = 12$, $T = 0.25$ and $l \in \{1, 2, 3, 4, 5, 6\}$. Prices dynamic parameters are chosen in order to have an option value equal to the one dimensional case put option value:



(A) Probability of exercise.

(B) Exercise region.

Figure 4.2. Probability of exercise and exercise region (region below the curve) for the Bermudan put option computed from the trained neural network.

the product of the components of X follows a Black–Scholes dynamic with drift parameter equal to 0.0488 and volatility equal to 0.25. It allows to have a reference value (from [24]) while

Table 4.2. Comparison of results obtained by Algorithm 1 with the ones of [24] for different initial values S_0 and different number of executions l. The first value corresponds to the swing option value obtained with Algorithm 1, the second value to the one in [24] and the third value is the relative difference in %.

l / S_0	35	40	45
1	(5.104, 5.114, 0.19%)	(1.776, 1.774, 0.12%)	(0.409, 0.411, 0.42%)
2	(10.165, 10.195, 0.29%)	(3.492, 3.48, 0.34%)	(0.772, 0.772, 0.06%)
3	(15.194, 15.23, 0.24%)	(5.115, 5.111, 0.07%)	(1.09, 1.089, 0.09%)
4	(20.188, 20.23, 0.21%)	(6.658, 6.661, 0.05%)	(1.358, 1.358, 0.0%)
5	(25.19, 25.2, 0.04%)	(8.148, 8.124, 0.3%)	(1.58, 1.582, 0.12%)
6	(30.156, 30.121, 0.12%)	(9.494, 9.502, 0.09%)	(1.764, 1.756, 0.44%)

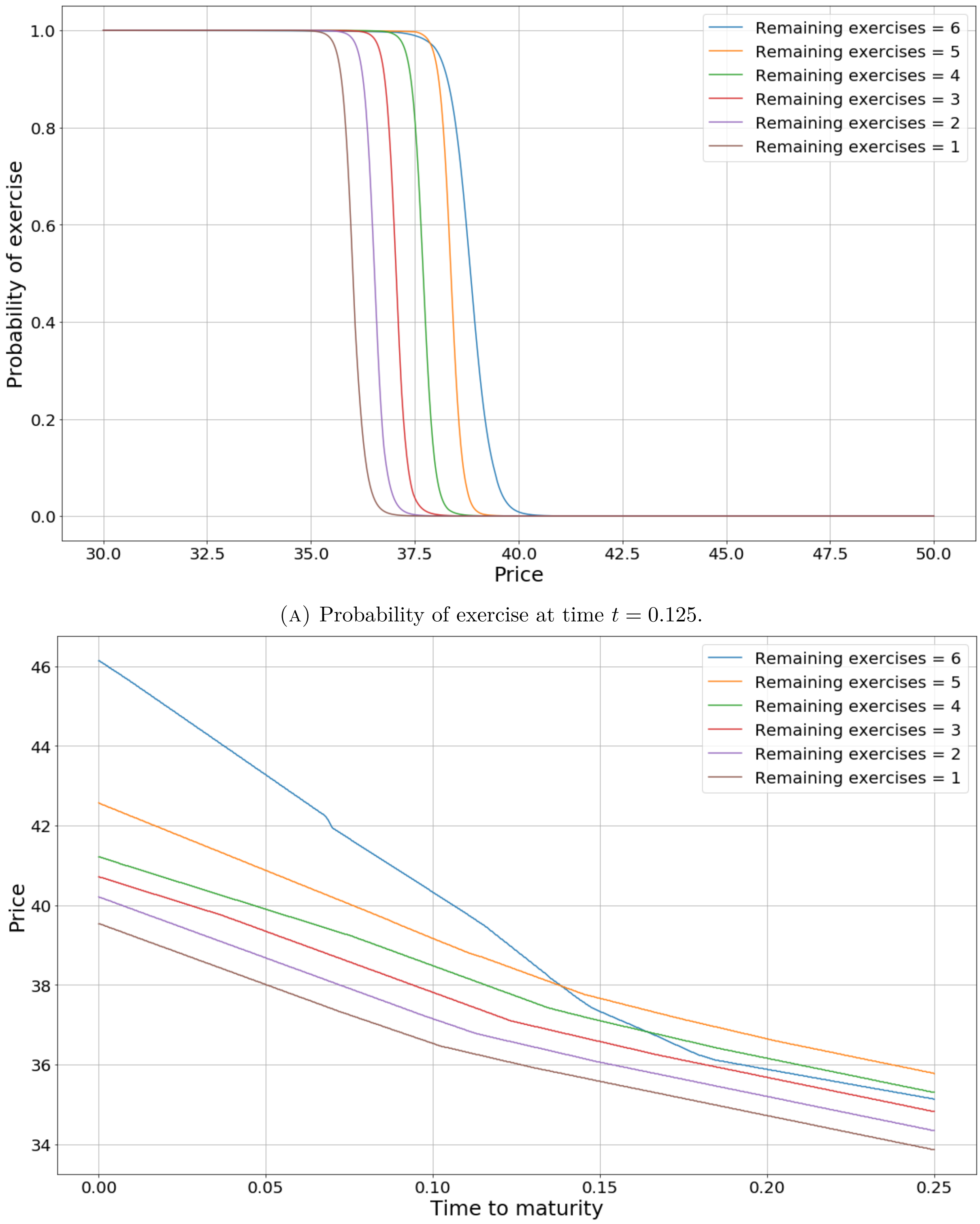
Table 4.3. Time in seconds for training and predicting with Algorithm 1 to price the swing put option for different initial values S_0 and different number of executions l.

l / S_0	35	40	45
1	270.1	278.6	249.9
2	241.6	246.6	236.8
3	240.6	237.7	243.4
4	239.6	242.4	240.0
5	271.3	239.9	235.3
6	243.8	240.9	241.7

considering a high dimensional case. We consider a batch size equal to $N_{\text{batch}} = 8000$, a neural networks with $L = 3$ hidden layers of size $m = 30$ and $N_{\text{iter}} = 5000$ iterations. Results are given in Table 4.4. The algorithm succeeds in pricing this option with 5 underlyings in a reasonable time (less than 30 minutes). By using a less costly hyperparameterization ($m = 20$ neurons density) instead of $m = 30$, $N_{\text{iter}} = 2000$ iterations instead of 5000, $N_{\text{batch}} = 3000$ instead of 8000 we are able to obtain results in less than 4 minutes with a 2% accuracy as shown in Table 4.5.

Table 4.4. Comparison of results obtained by Algorithm 1 for the pricing of a 5 dimensional swing put option with the reference values reported in [24]. The time in seconds corresponds to the time of training and predicting.

Use case / Method	Algorithm 1	Reference	Difference	Time (s)
l = 1	1.767	1.774	0.39%	1814.3
l = 2	3.478	3.480	0.04%	1686.6
l = 3	5.100	5.111	0.22%	1701.5
l = 4	6.639	6.661	0.32%	1661.0
l = 5	8.117	8.124	0.09%	1689.2
l = 6	9.478	9.502	0.25%	1675.7



(A) Probability of exercise at time $t = 0.125$.

(B) Exercise region.

Figure 4.3. Probability of exercise and exercise region (region below the curve) for the swing put option with strike 40 and maximum number of exercises 6.

Table 4.5. Comparison of results obtained by Algorithm 1 with suboptimal hyperparameters for the pricing of a 5 dimensional swing put option with the reference values reported in [24]. The time in seconds corresponds to the time of training and predicting.

Use case / Method	Algorithm 1	Reference	Difference	Time (s)
l = 1	1.733	1.774	2.33%	221.9
l = 2	3.435	3.480	1.29%	222.0
l = 3	5.074	5.111	0.71%	212.2
l = 4	6.591	6.661	1.05%	199.4
l = 5	8.033	8.124	1.13%	197.0
l = 6	9.392	9.502	1.16%	203.0

4.3. Swing options with delay

Let us now consider the case of a put option with $d = 1$, $g(x) = (K - x)^+$, $K = 100$, $S_0 = 100$, $r = 0.05$, $\delta = 0$, $\Sigma = 0.3$, $N = 50$, $T = 1$, $\gamma = 5\frac{T}{N}$ and $l \in \{1, 2, 3, 4, 5\}$. Delay constraint is now present and a higher number of dates is considered. We consider a batch size equal to $N_{\text{batch}} = 5000$, a neural networks with $L = 3$ hidden layers of size $m = 10$ and $N_{\text{iter}} = 10000$ iterations. We compare in Table 4.6 the results obtained with Algorithm 1 to the ones obtained with [13]. The algorithm gives satisfying results but when compared to results of Table 4.5, we notice a loss of performances as l increases. This may be due to the delay constraint that is added in this case. We tried to change the architecture of the neural networks without noticing a significant improvement of the algorithm.

Table 4.6. Comparison of results obtained by Algorithm 1 with the reference values reported in [13]. The time in seconds corresponds to the time of training and predicting.

Use case / Method	Algorithm 1	Reference	Difference	Time (s)
l = 1	9.844	9.85	0.06%	5430.5
l = 2	19.093	19.26	0.87%	6658.0
l = 3	27.827	28.80	3.38%	5518.4
l = 4	36.058	38.48	6.29%	5566.1
l = 5	43.638	48.32	9.69%	5472.1

5. Conclusion and perspectives

A stochastic control algorithm able to deal with (discrete) optimal stopping variables is presented. The different use cases show that the proposed algorithm is able to solve optimal stopping time problems in a reasonable time, even when the dimension is high and also for multi-exercise. The algorithm is simple and allows us to find an optimal policy without any knowledge on the dynamic programming equation. The method presented in this paper avoids a costly backward pass and only needs a forward pass. While the computation time increases a little with dimension, it increases a lot more with the number of time steps and the algorithm can have troubles to converge. To confirm all those results, one should study the theoretical convergence of the algorithm. This algorithm could easily be extended to impulse control if combined with [17] in order to solve problems involving both continuous and discrete controls such as hedging with fixed transaction costs.

Acknowledgements

This research is supported by the department OSIRIS[1] of EDF Lab and FiME[2] Laboratory which are gratefully acknowledged. We would like to thank the referee for constructive comments that helped to improve the quality of the manuscript.

References

[1] Martin Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mane, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems, 2015. Software available from tensorflow.org.

[2] Leif B. G. Andersen. A simple approach to the pricing of Bermudan swaptions in the multi-factor Libor market model, 1999. Available at SSRN 155208.

[3] Achref Bachouch, Côme Huré, Nicolas Langrené, and Huyên Pham. Deep neural networks algorithms for stochastic control problems on finite horizon: numerical applications. *Methodol. Comput. Appl. Probab.*, 24(1):143–178, 2022.

[4] Christophe Barrera-Esteve, Florent Bergeret, Charles Dossal, Emmanuel Gobet, Asma Meziou, Rémi Munos, and Damien Reboul-Salze. Numerical methods for the pricing of swing options: a stochastic control approach. *Methodol. Comput. Appl. Probab.*, 8(4):517–540, 2006.

[5] Sebastian Becker, Patrick Cheridito, and Arnulf Jentzen. Deep Optimal Stopping. *J. Mach. Learn. Res.*, 20(74):1–25, 2019.

[6] Sebastian Becker, Patrick Cheridito, and Arnulf Jentzen. Pricing and hedging American-style options with deep learning. *J. Risk Financ. Manag.*, 13(7):158, 2020.

[7] Sebastian Becker, Patrick Cheridito, Arnulf Jentzen, and Timo Welti. Solving high-dimensional optimal stopping problems using deep learning. *Eur. J. Appl. Math.*, 32(3):470–514, 2021.

[8] Irwan Bello, Hieu Pham, Quoc V Le, Mohammad Norouzi, and Samy Bengio. Neural combinatorial optimization with reinforcement learning, 2017. Workshop Track of the International Conference on Learning Representations.

[9] Marie Bernhart, Huyên Pham, Peter Tankov, and Xavier Warin. Swing options valuation: A bsde with constrained jumps approach. In *Numerical methods in finance*, pages 379–400. Springer, 2012.

[10] Bruno Bouchard and Jean-François Chassagneux. Discrete-time approximation for continuously and discretely reflected BSDEs. *Stochastic Processes Appl.*, 118(12):2269–2293, 2008.

[11] Bruno Bouchard and Xavier Warin. Monte-Carlo valuation of American options: facts and new algorithms to improve existing methods. In *Numerical methods in finance*, pages 215–255. Springer, 2012.

[12] Hans Buehler, Lukas Gonon, Josef Teichmann, and Ben Wood. Deep hedging. *Quant. Finance*, 19(8):1271–1291, 2019.

[13] René Carmona and Nizar Touzi. Optimal multiple stopping and valuation of swing options. *Math. Finance*, 18(2):239–268, 2008.

[14] Quentin Chan-Wai-Nam, Joseph Mikael, and Xavier Warin. Machine learning for semi linear PDEs. *J. Sci. Comput.*, 79(3):1667–1712, 2019.

[1] Optimization, SImulation, RIsk and Statistics for energy markets

[2] Finance des marchés de l’énergie

[15] Weinan E, Jiequn Han, and Arnulf Jentzen. Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations. *Commun. Math. Stat.*, 5(4):349–380, 2017.

[16] Nicole El Karoui, Christophe Kapoudjian, Étienne Pardoux, Shige Peng, Marie-Claire Quenez, et al. Reflected solutions of backward SDE's, and related obstacle problems for PDE's. *Ann. Probab.*, 25(2):702–737, 1997.

[17] Simon Fécamp, Joseph Mikael, and Xavier Warin. Deep learning for discrete-time hedging in incomplete markets. *J. Comput. Finance*, 2020.

[18] Diego Garcıa. Convergence and biases of Monte Carlo estimates of American option prices using a parametric exercise rule. *J. Econ. Dyn. Control*, 27(10):1855–1879, 2003.

[19] Paul Glasserman. *Monte Carlo methods in financial engineering*, volume 53. Springer, 2013.

[20] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.

[21] Jiequn Han, Arnulf Jentzen, and Weinan E. Solving high-dimensional partial differential equations using deep learning. *Proc. Natl. Acad. Sci. USA*, 115(34):8505–8510, 2018.

[22] Côme Huré, Huyên Pham, Achref Bachouch, and Nicolas Langrené. Deep neural networks algorithms for stochastic control problems on finite horizon: convergence analysis. *SIAM J. Numer. Anal.*, 59(1):525–557, 2021.

[23] Côme Huré, Huyên Pham, and Xavier Warin. Deep backward schemes for high-dimensional nonlinear PDEs. *Math. Comput.*, 89(324):1547–1579, 2020.

[24] Alfredo Ibáñez. Valuation by simulation of contingent claims with multiple early exercise opportunities. *Math. Finance*, 14(2):223–248, 2004.

[25] Michael Kohler, Adam Krzyżak, and Nebojsa Todorovic. Pricing of High-Dimensional American Options by Neural Networks. *Math. Finance*, 20(3):383–410, 2010.

[26] Francis A. Longstaff and Eduardo S. Schwartz. Valuing American options by simulation: a simple least-squares approach. *Rev. Financ. Stud.*, 14(1):113–147, 2001.

[27] Bernt Karsten Øksendal and Agnes Sulem. *Applied stochastic control of jump diffusions*, volume 498. Springer, 2005.

[28] Steven E. Shreve. *Stochastic calculus for finance II: Continuous-time models*, volume 11. Springer, 2004.

[29] Justin Sirignano and Konstantinos Spiliopoulos. Dgm: A deep learning algorithm for solving partial differential equations. *J. Comput. Phys.*, 375:1339–1364, 2018.

[30] Richard S. Sutton, David A. McAllester, Satinder P. Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *NIPS'99: Proceedings of the 12th International Conference on Neural Information Processing Systems*, pages 1057–1063. MIT Press, 2000.

[31] Faouzi Trabelsi. Study of undiscounted non-linear optimal multiple stopping problems on unbounded intervals. *Int. J. Math. Oper. Res.*, 5(2):225–254, 2013.

[32] Xavier Warin. Gas storage hedging. In *Numerical methods in finance*, pages 421–445. Springer, 2012.

[33] Tingting Zhao, Hirotaka Hachiya, Gang Niu, and Masashi Sugiyama. Analysis and improvement of policy gradient estimation. In *NIPS'11: Proceedings of the 24th International Conference on Neural Information Processing Systems*, pages 262–270. Curran Associates Inc., 2011.