Rapport semestriel S8

Adil Dinia, Léna Ezzine, Mohamed Hamdouche, Yassine Abbahaddou

La valorisation des options américaines par le Deep Learning

Encadrants :
Ludovic Goudenège
Gautier Viaud

04/06/2019

Résumé
Dans ce rapport, nous faisons état du travail que nous avons fourni pendant ce deuxième semestre en liaison avec les résultats obtenus en premier semestre.

CentraleSupélec

Table des matières

Introduction

Dans de nombreux articles récemment publiés, ont été présentés des résultats optimistes de réseaux de neurones résolvant des équations aux dérivées partielles (EDP). Nous souhaitons nous-même évaluer la pertinence et la fiabilité d'une telle technique en l'appliquant au problème du pricing des options américaines.

L'idée d'user de deep learning pour la résolution d'EDP provient du fait que celles-ci nécessitent des méthodes complexes et souvent coûteuses en calcul, d'autant plus quand la dimension devient élevée. Ces méthodes sont alors très limitées par la capacité de calcul des ordinateurs. Les réseaux de neurones pourraient pallier à ce problème : une fois entraînés, ils devraient donner un résultat beaucoup plus rapidement. Le problème est de déterminer à quel point un réseau de neurones est pertinent et performant pour la résolution. En effet, une fois construit et entraîné, un réseau de neurones fournit une solution sans justification. On peut évaluer sa précision pour des équations dont on connaît la solution, mais celles-ci ne seront pas toujours calculables par les moyens traditionnels (et a priori fiables).

C'est pourquoi nous tentons d'évaluer la faisabilité de la résolution avec un réseau de neurones pour une équation à petite dimension. Nous nous intéressons en particulier à l'équation de Black-Scholes pour les options américaines. L'intérêt de ce problème est que l'on dispose d'une grande base de données de prix réels sur laquelle nous pourrons entraîner le réseau de neurones.

Les codes de nos algorithmes se trouvent sur le git :

https ://github.com/damianib/DLforPDE ?fbclid=IwAR0sTAmx2Cb821aUMaxy$_P z5$
$qJl7N9yqgxrbEL7bmXM9qRbfDq3u2ta4tbs$

1 Les options

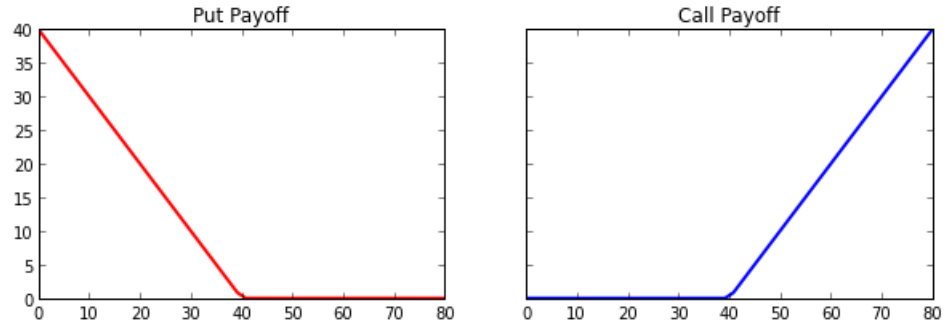
1.1 Définitions

Une option est un contrat par lequel le détenteur a le droit, et non l'obligation, d'acheter (option d'achat : call) ou de vendre (option de vente : put) une quantité donnée de l'actif sous-jacent 1 à un cours convenu à l'avance, appelé prix d'exercice (Strike) à (ou jusqu'à) une date déterminée dite d'échéance de l'option moyennant le paiement immédiat d'une prime (premium) qui est le prix de l'option.

On distingue deux grandes catégories d'options : les options européennes et les options américaines. La différence tient au fait de pouvoir exercer ou non l'option avant l'échéance. Dans le cas d'une option américaine, l'acheteur peut exercer son option à tout moment entre t_0 (prise de position) et T (échéance). Une option européenne ne peut être exercée avant l'échéance T. Dans ce projet nous allons nous intéresser uniquement aux options européennes.

En l'absence d'une couverture spécifique et dans le cas le plus défavorable, l'acheteur d'une option aura une perte limitée à la prime qu'il aura payée. Son gain maximum est en revanche illimité s'il a acquis une option d'achat et limité au prix d'exercice diminué de la prime pour une option de vente.

Symétriquement, le vendeur d'une option voit son gain maximum limité à la prime qu'il reçoit. Sa perte peut être illimitée (vendeur d'un call) ou limitée (vendeur d'un put). Il s'agit d'une stratégie spéculative très risquée.



1.2 Les caractéristiques d'une option

Une option dépend essentiellement des paramètres suivants :

1. S_0 : le prix de l'actif sous jacent à l'instant initial t_0.
2. T : La date de maturité ou d'échéance, l'instant auquel l'option pourra être exercée.
3. K : Le prix d'exercice d'une option, prix auquel l'acheteur d'une option peut acheter (dans le cas d'une option d'achat) ou vendre (dans le cas d'une option de vente) l'actif sous-jacent.
4. r : Le taux d'intérêt sans risque.
5. σ : La volatilité du sous-jacent.

1.3 Étude théorique des options américaines

Une option américaine peut être exercée à tout instant t entre 0 et T. Nous nous contenterons ici d'étudier les options bermudiennes, qui peuvent être exercées à des instants discrets $0 = t_0 < t_1 < ... < t_N = T$. Ces options approchent bien les options américaines lorsque N devient grand.

Une telle option est définie par une suite (Z_n) positive et adaptée à la filtration $\mathcal{F}_n$, représentant le profit que donnerait l'exercice de l'option à l'instant n. Nous notons $\mathcal{Z}_n$ la valeur actualisée du profit.

L'enveloppe de Snell du Pay-off :

Pour définir la valeur de l'option à l'instant t_n, il faut raisonner par récurrence à partir de l'échéance t_N . La valeur UN d'une option de vente 'a l'échéance est, comme pour l'option européenne, $U_N = Z_N = (K - S_N)_+$.
A tout instant $n-1$, le détenteur de l'option a le choix entre :

— exercer son option et en tirer immédiatement un profit Z_{n-1}

— conserver son option, et détenir à l'instant suivant la valeur U_n, dont la valeur actualisée à l'instant $n-1$ est $U_n B(n-1,n)$. L'espérance de cette valeur, connaissant l'histoire jusqu'à l'instant $n-1$, est $B(n-1,n)\mathbb{E}^*(Un|F_{n-1})$

La valeur U_{n-1} de l'option à l'instant t_{n-1} s'en déduit :

$$U_{n-1} = \max[Z_{n-1}, B(n-1,n)\mathbb{E}^*(Un|F_{n-1})]$$

Le système vérifié par U_n est donc :

$$\begin{cases} U_N = Z_N \\ U_{n-1} = \max[Z_{n-1}, B(n-1,n)\mathbb{E}^*(Un|F_{n-1})] \end{cases}$$

En considérant les quantités actualisées :

$$\begin{cases} \mathcal{U}_N = \mathcal{Z}_N \\ \mathcal{U}_{n-1} = \max[\mathcal{Z}_{n-1}, \mathbb{E}^*(\mathcal{U}_n|F_{n-1})] \end{cases}$$

Temps d'arrêt optimal

L'option va être exercée à un instant $\tau = t_n$ entre l'origine et l'échéance, bornes incluses, et elle va rapporter un pay-off Z_n. τ est un temps d'arrêt, puisque la décision d'exercer l'option à un instant n se fera en connaissance du seul cours de l'actif sur la période $[0...t_n]$.

Définissons τ_0 comme le premier instant où $U_{\tau_0} = Z_{\tau_0}$. C'est évidemment un temps d'arrêt, à valeur dans $0...N$. Pour tout $t_n < \tau_0$, on a $U_n > Z_n$, donc la propriété de martingale. On en déduit le prix de l'option :

$$\mathcal{U}_0 = \mathbb{E}^*(\mathcal{U}_{\tau_0}|F_0) = \mathbb{E}^*(\mathcal{Z}_{\tau_0}|F_0) = \mathbb{E}^*(e^{-r\tau}Z_{\tau_0}|F_0) = \mathbb{E}^*(e^{-r\tau_0}Z_{\tau_{00}})$$

Par définition, un temps d'arrêt τ^* est optimal s'il réalise le sup sur tous les temps d'arrêt du pay-off actualisé : $\mathcal{Z}_{\tau^*} = \sup_\tau \mathcal{Z}_\tau$
Le temps d'arrêt τ_0 est optimal, puisque :

$$\begin{cases} \mathcal{U}_0 = \mathbb{E}^*(\mathcal{Z}_{\tau_0|F_0)} \\ \mathcal{U}_0 \geq \mathbb{E}^*(\mathcal{U}_{\tau_0}|F_0) \geq \mathbb{E}^*(\mathcal{Z}_{\tau_0}|F_0) \end{cases}$$

La valeur U_0 de l'option à l'instant initial est donc le sup sur $T_{[0,N]}$ de l'espérance de pay-off actualisé :

$$U_0 = \mathcal{U}_0 = \sup_\tau \mathbb{E}^*(\mathcal{Z}_\tau) = \sup_\tau \mathbb{E}^*(e^{-r\tau}Z_\tau)$$

Remarque : Dans le cas particulier de l'option call américaine sur un actif ne versant pas de dividende, comme la valeur temps est toujours positive, il n'est jamais optimal d'exercer l'option avant la maturité, et donc, **le prix d'un call américain est égal au prix d'un call européen.**
Dans ce qui suit, nous nous s'intéresseront qu'aux options de type put puiqu'il existe déja une formule pour les options américaines de type call.

2 Implémentation de l'algorithme de rétropropagation

2.1 Le modèle adopté

La valorisation des options américaines par l'algorithme de rétropropagation est très utilisée dans les marchés financiers, cette méthode est connue sous le nom d'arbre binomial. Car dans cette algorithme, on travaille avec le modèle d'un marché binomial au lieu de travailler avec le modèle de Black and Scholes qui est un peu très compliqué pour la valorisation de ces options. En effet, le modèle de Black and Scholes peut être vu comme la limite d'une suite de processus de prix qui suivent un modèle binomal. Soit N un entier non nul, $(\epsilon_k)_{k\in\mathbb{N}}$ une suite de variables aléatoires indépendantes qui prennent $+\sigma\sqrt{T}$ ou $-\sigma\sqrt{T}$ avec une probabilité de $\frac{1}{2}$, on considère le modèle binomial à N périodes suivant :

$$S_{k+1}^{(N)} = S_k^{(N)}(1 + \frac{\epsilon_k}{\sqrt{N}} + \frac{rT}{N})$$

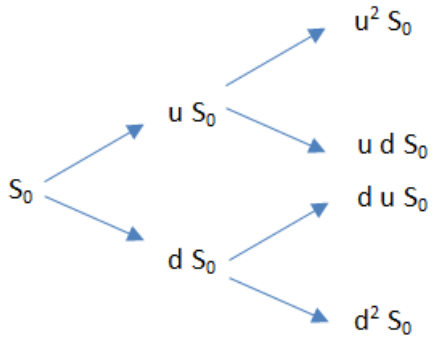
Pour N très grand, ce modèle rapproche le modèle de Black and Scholes, et les paramètres de ce modèle peuvent se mettre sous la forme suivante :

$$\begin{aligned} u &= 1 + \frac{\sigma\sqrt{T}}{\sqrt{N}} + \frac{rT}{N} \\ &\simeq exp(\sigma\sqrt{\frac{T}{N}}) \end{aligned}$$

$$\begin{aligned} d &= 1 - \frac{\sigma\sqrt{T}}{\sqrt{N}} + \frac{rT}{N} \\ &\simeq exp(-\sigma\sqrt{\frac{T}{N}}) \end{aligned}$$

Dans le modèle binomial la mesure de probabilité Q à risque neutre est définit par :

$$\mathbb{Q}(\frac{S_{k+1}^{(N)}}{S_k^{(N)}} = u) = \frac{exp(\frac{rT}{N}) - d}{u - d} = q, \quad \mathbb{Q}(\frac{S_{k+1}^{(N)}}{S_k^{(N)}} = d) = \frac{u - exp(\frac{rT}{N})}{u - d} = 1 - q$$



2.2 Algorithme

Dans le cas d'une option portant sur un sous-jacent de payoff $f(S_N)$, l'algorithme s'écrit sous la forme suivant :
Pour tout $k \in \{0, ..., N-1\}$,

$$\begin{cases} V_k = \frac{1}{exp(\frac{rT}{N})}(qV_{k+1}(S_k u) + (1-q)V_{k+1}(S_k d)) \\ V_N = f(S_N) \end{cases}$$

Le pseudo-code de l'algorithme de valorisation par rétropropagation est donné ci-dessous :

Compute of model parameters : u,d,q
Creation of an empty P
for $k = 0, ..., N$ **do**
  Add $f(S_0 u^{N-k} d^k)$ to P
end
for $k = 0, ..., N-1$ **do**
  for $i = 0, ..., N-k-1$ **do**
    $P[i] \leftarrow max(f(S_0 u^{N-k-1-i} d^i, \frac{qP[i]+(1-q)P[i+1]}{exp(\frac{rT}{N})})$
  end
  return P[0]
end

3 Choix du module d'implémentation des réseaux de neurones

Le langage de programmation de choix pour les réseaux de neurones est Python, pour lequel on distingue trois modules majeurs pour le Machine Learning :

Scikit-learn un module composé de nombreuses fonctions "boîte noire". Il propose des solutions à notre problème de régression mais très peu de marge de manoeuvre pour en modifier la structure ou les paramètres.

Tensorflow permet de construire un réseau de neurones simplement en spécifiant l'intégralité des couches. Le module fournit même des outils pour re-créer un réseau de neurones à partir de "rien", permettant de changer n'importe quel élément de sa structure.

PyTorch dans la même philosophie que Tensorflow, mais avec une syntaxe se rapprochant plus de celle rencontrée habituellement en Python.

Pour la suite de notre projet, nous avons rapidement écarté Scikit-learn en raison de son manque de flexibilité, et avons finalement privilégié Tensorflow à PyTorch en raison du plus grand nombre de ressources disponibles sur Internet pour ce module, celui-ci étant plus connu et plus largement utilisé.

4 Implémentation de l'algorithme de Longstaff Shwartz

4.1 Le modèle adopté

Le modèle utilisé dans cet algorithme est le modèle de**Black-Scholes** qui est un modèle mathématique du marché pour une action, dans lequel le prix de l'action est un processus stochastique en temps continu.

Le prix de l'actif sous-jacent $S_t$suit un mouvement brownien géométrique avec une volatilité σ constante et une dérive r constante.

$$dS_t = rS_t dt + \sigma S_t dW_t$$

Sous ce modèle, le prix de l'action à l'instant t s'écrit :

$$S_t = S_0 e^{(r-\frac{\sigma^2}{2})t+\sigma W_t}$$

4.2 Algorithme

On se place ici dans le cas d'un put, mais l'algorithme peut également être adapté aux calls. La méthode de Longstaff-Schwarz (abréviation LS), telle qu'elle est définie dans

l'article fondateur [1], propose de résoudre le problème des options américaines par la méthode des moindres carrés et la méthode de Monte Carlo. On définit un nombre n d'évolutions de prix à simuler et un entier d correspondant au nombre d'instants considérés entre $t = 0$ et $t = T$ la maturité. Une première étape consiste à utiliser les équations de Black-Scholes en temps discret, qui permettent de créer aléatoirement une évolution des n prix pour les d instants. La deuxième partie de l'algorithme est récursive en partant de la maturité. On calcule quel est le payoff pour le vendeur s'il exerce l'option à la maturité pour toutes les simulations de prix et l'on retient celles qui donnent un payoff positif, que l'on stocke dans un tableau Y. En considérant la valeur de l'option à l'instant avant la maturité, que l'on stocke dans X, on souhaite estimer la valeur de Y sachant X. Pour ce faire, on fixe un degré m, et on cherche à donner une estimation de la fonction f telle que $Y = f(X)$ dans la base $(1, X, ..., X^m)$ en minimisant la distance entre la fonction estimée et les points déterminés à partir de la simulation (méthode des moindres carrés). D'autres bases de polynômes, comme les polynômes de Laguerre peuvent bien sûr être considérées.

A partir de cette régression, on peut donc obtenir le bénéfice du vendeur si il exerce avant la maturité ainsi que l'estimation de son gain si il attend au-delà, appelé continuation. Notons qu'il est nécessaire de prendre en compte le taux d'intérêt dans ce calcul, puisque si le vendeur vend un instant avant il récoltera plus d'intérêts sur l'argent gagné par rapport aux instants d'après. Ainsi on connaît la décision du vendeur à l'instant considéré qui tentera nécessairement de maximiser son gain.

En répétant le même processus de régression récursivement pour les instants précédents, on construit une matrice dite "cash-flow" contenant le gain du vendeur et à quel instant pour chacune des n simulations. En n'oubliant pas d'appliquer les taux d'intérêts, on en déduit par une moyenne une estimation du gain moyen du vendeur, donc du prix à donner à l'option (méthode de Monte-Carlo).

Le pseudo-code de l'algorithme est écrit ci-dessous (on note g la fonction payoff)

Générer l'évolution des prix $S_i(t), t = 0, t_1, ..., t_d, i = 1, 2, ..., n$;
Mettre $P_i \leftarrow g(S_i(t_d))$ pour tout i ;
for t *allant de* t_{d-1} *à* t_1 **do**
  Trouver les $(i_1, ..., i_k)$ tels que $g(S_i(t)) > 0$;
  Mettre $aRegarder \leftarrow (i_1, ..., i_k)$;
  Mettre $x_i \leftarrow S_i(t)$ et $y_i \leftarrow e^{-r\Delta t} P_i$ pour tout $i \in aRegarder$;
  Trouver les coefficients de régression sur x, y;
  Estimer la valeur de la continuation $\hat{C}(S_i(t))$ et la valeur de l'exercice $g(S_i(t))$ pour tout $i \in aRegarder$;
  for i *allant de* 1 *à* n **do**
    if $i \in aRegarder$ *et* $g(S_i(t)) > \hat{C}(S_i(t))$ **then**
      $P_i \leftarrow g(S_i(t))$;
    else
      $P_i \leftarrow e^{-r\Delta t} P_i$;
    end
  end
end
Mettre $prix \leftarrow \frac{1}{n} \sum_{i=1}^{n} e^{-r\Delta t} P_i$;

Nous avons vérifié que l'algorithme était correctement implémenté en comparant ces résultats à ceux du logiciel Premia.

5 Option sur multi sous-jacents

On commence par comparer les résultats obtenus par les différents algorithmes qu'on avait implémenté. Pour faire cette comparaison, on considère trois options : Put En dehors de la monnaie, Put à la monnaie et Put Dans la monnaie. Ci-dessous un récapitulatif des résultats obtenus En considérant le jeu de paramètres suivant :

$$\begin{aligned} N &= 100 \\ \sigma &= 0.2 \\ S_0 &= 1 \\ r &= 0.01 \\ T &= 1 \end{aligned}$$

et K prend les valeurs suivantes :
- Dans la monnaie : K = 1.1
- à la monnaie : K = 1
- En dehors de la monnaie : K = 0.9

	Backpropagation	Longstaff Schwartz
Put En dehors de la monnaie	0.03337234807690873	0.03442638159475607
Put à la monnaie	0.07498604149689439	0.07425299404409078
Put Dans la monnaie	0.13703334054347352	0.136274355498526

Pour évaluer la performance des nos algorithmes, nous allons essayer dans la suite de cette section d'augmenter la dimension de notre problème. En effet, au lieu de travailler sur des options portant sur un seul sous-jacent, nous envisageons de travailler avec des options sur plusieurs sous-jacents, et le nombre de sous-jacents représente la dimension de notre problème. Voici ci-dessous les payoffs adoptés pour chaque dimension :

- dimension 1 : $f(S) = (K - S)^+$
- dimension 2 : $f(S_1, S_2) = (K - \frac{(S_1+S_2)}{2})^+$
- dimension 3 : $f(S_1, S_2, S_3) = (K - \frac{(S_1+S_2+S_3)}{3})^+$

Ci-dessous un tableau qui montre le temps d'exécution pour les deux algorithmes et pour chaque dimension :

	dimension 1	dimension 2	dimension 3
Backpropagation algorithm	0.01901	2.66389	366.80669
Longstaff Schwartz algorithm	2.29222	7.4224	11.39403

Fléau de dimension dans l'algorithme de rétro-propagation
En dimension r, il faut connaître à chaque instant n la valeur en tous les r-uplets $\{(S^p(1 + a_p)^{k_p}(1 + a_p)^{n-k_p}, 0 \leq p \leq r), 0 \leq k_1, .., k_r \leq n\}$, soit $(n + 1)^r$ valeurs 'a calculer et stocker. L'algorithme de rétro-propagation a donc un coût exponentiel, ce qui empêche son utilisation pratique sur des dimensions grandes.

Fléau de dimension dans l'algorithme de Longstaff Schwarz
Pour avoir une bonne précision, la cardinalité de la base doit être plus importante en multi-dimension : il y a par exemple 10 polynêmes de degré inférieur à 2 en dimension 3 : $\{1, X, Y, Z, X^2, Y^2, Z^2, XY, YZ, ZX\}$

A d donné (d représente le degré des polynomes), le nombre de fonctions à considérer est de l'ordre de r^d, le nombre de tirages est rM. Le coût de cet algorithme est finalement en r^d.

6 Régression de LSM avec réseaux de neurones

Comme expliqué plus haut, la partie régression polynomiale de Longstaff Schwartz étant trop coûteuse quand on augmente le nombre de données, une alternative à cela est de la remplacer par un réseau de neurones qu'on entraîne à faire la régression.

Pour cela, notre base de données est constituée des couples (points, image de ces points par un polynôme de degré 4 écrit dans la base canonique, dont les coefficients sont dans $[-1, 1]$, et qui est évalué entre -0.25 et 0.25 pour assurer que ses valeurs soient entre -1 et 1 (normalisation) . Ces polynômes sont bruités pour éviter l'overfitting, et une fois notre base de données prête on alimente le réseau de neurones avec les couples précédents.

On obtient les résultats suivants :

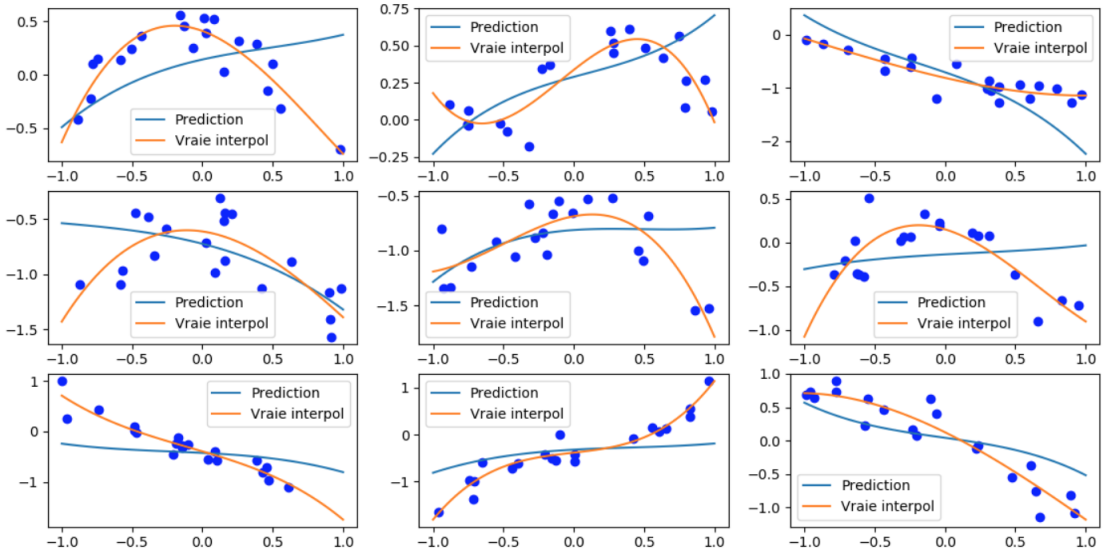


FIGURE 1 – Prédictions du polynôme prédit par le réseau de neurones vs. vraie interpolation

Qui ne sont pas illogiques, mais qui peuvent parfois s'éloigner du vrai polynôme (Voir figure au milieu à droite).

Nous allons tenter de remplacer la partie régression par un réseau de neurones comme suit : On se donne m évolutions du sous-jacent entre les temps 0 et T.

t = 0	...	t = j	...	t = T
X_0^0	$\vdots$	X_0^j	$\vdots$	...
X_1^0	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\vdots$	$\vdots$	X_m^j	$\vdots$	$\vdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
X_M^0	...	X_M^j	...	$\vdots$

À chaque étape, un réseau de neurone effectue une régression pour prédire :

$$P(X_m^j) = B(j, \tau_m^{j+1})\phi(X_m^{\tau_m^{j+1}})$$

où B désigne le taux d'intérêt qui dépend de la durée Δt (comme dans l'expression $exp(-r\Delta t)$) et donc de l'instant de départ ici j et de fin τ_m^{j+1}, qui est le temps auquel on compte exercer en ayant parcouru les valeurs après l'instant $j+1$ (voir l'expression ci-après au temps j). Et ϕ est le payoff à l'instant τ_m^{j+}, i.e. $(K - X_m^{\tau_m^{j+1}})$.

À partir de la valeur prédite, à l'étape j (incrémentation négative), on peut fixer la valeur du temps d'exercice prédit :

$$\tau_m^j = \begin{cases} j & \text{si } \phi(X_m^j) > P(X_m^j) \\ \tau_m^{j+1} & \text{sinon} \end{cases}$$

Le but à la fin de la *backward induction* est d'obtenir τ_m^0.

6.1 Application

Implémentons cet algorithme pour 10 000 données, avec un prix initial de 1 et un strike de 1, sur une période de maturité d'un an, un taux d'intérêt de 0.01 et une volatilité de 0.2. À chaque étape, un réseau de neurones à 4 couches de respectivement 1, 60, 60 et 1 neurone s'entraîne sur 1000 données puis fait la prédiction sur toutes les données.

On obtient les résultats suivants :

FIGURE 2 – Valeur de continuation à l'instant t+1 en fonction du prix à l'instant t pour les réseaux de neurones de notre algorithme.

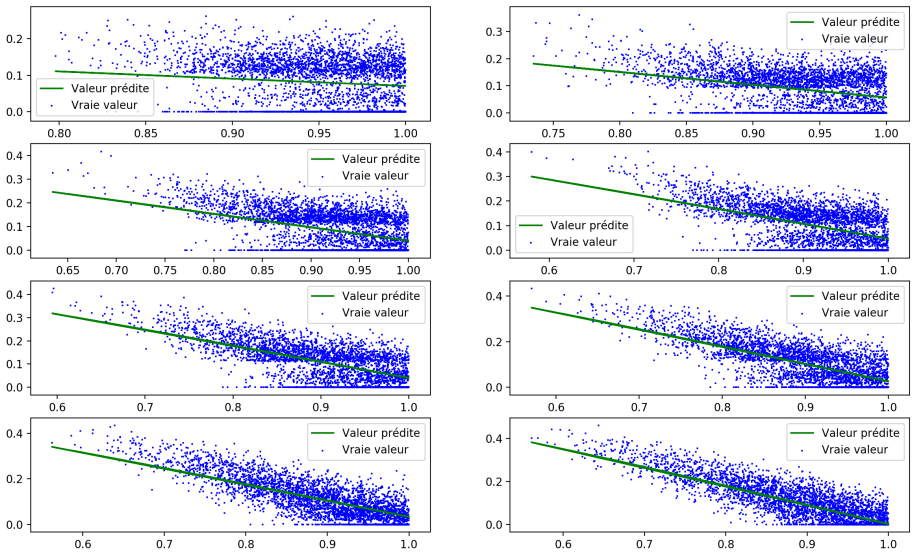
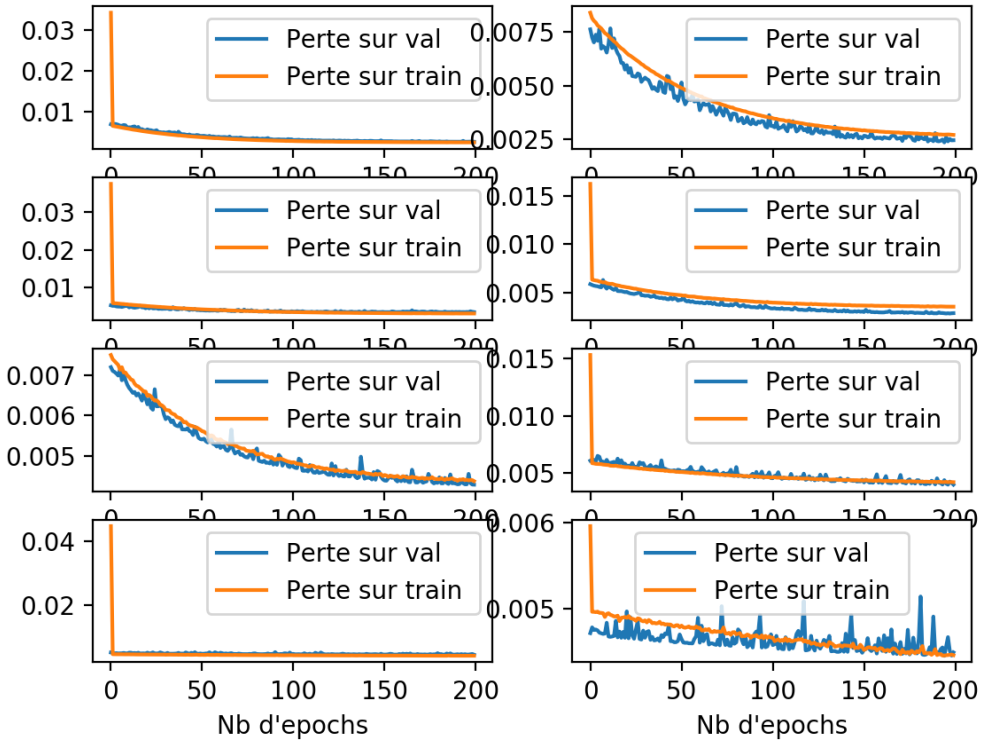


FIGURE 3 – Erreurs des réseaux de neurones à chaque instant



On remarque que même en s'entraînant sur 1/10 des données seulement, le réseau de neurones est très performant et donne le même ordre de grandeur d'erreur qu'un Longstaff-Schwartz classique entraîné sur toutes les données.

Comparaison des3 méthodes :

	Backpropagation	Longstaff Schwartz	LSM + NN
Put En dehors de la monnaie	0.03337234807690873	0.03442638159475607	0.03116485070953004
Put à la monnaie	0.07498604149689439	0.07425299404409078	0.0718975719336913
Put Dans la monnaie	0.13703334054347352	0.136274355498526	0.13057425026340017

7 Réseaux de neurones appliqués aux problèmes de contrôle optimal

Les algorithmes de contrôle optimal sont des types d'algorithmes largement utilisés et naturellement adaptés pour des problèmes d'optimisation stochastique.

Soit $(X_n)_n$ un processus stochastique à valeurs dans $\mathbb{R}^d$ dont l'évolution est caractérisée par la formule suivante :

$$X_{n+1} = F(X_n, \alpha_n, \varepsilon_{n+1}), n = 0, ..., N-1, X_0 = x_0 \in \mathbb{R}^d,$$

avec $(\varepsilon_n)_n$, une suite de variables aléatoires i.i.d de bruit générant une filtration $\mathbb{F} = (\mathcal{F}_n)_n$, $(\alpha_n)_n$ un processus dit processus de contrôle , à valeurs dans un ensemble de décision $\mathbb{A}$,adapté à la filtration $\mathbb{F}$.
Le processus $(\alpha_n)_n$ dans notre cas est directement lié au temps d'arrêt optimal τ.

Les variables de contrôle permettent de définir une fonction de coût à l'aide des fonctionnelles f et g, définie par l'expression suivante :

$$J(\alpha) = \mathbb{E}[\sum_{n=0}^{N-1} f(X_n, \alpha_n) + g(X_N)]$$

L'algorithme de contrôle optimal cherche à minimiser la fonction du coût , c'est-à-dire trouver :

$$V_0(x_0) := \inf_{\alpha \in \mathcal{A}} J(\alpha)$$

,associé à la variable de contrôle optimal α^*.
Le document propose une solution à ce problème, par une programmation dynamique formulé comme suit :

$$V_N(x) = g(x), x \in \mathbb{R}^d,$$
$$V_n(x) = \inf_{a \in \mathbb{A}} Q_n(x, a), \ (*)$$
$$\text{avec } Q_n(x, a) = f(x, a) + \mathbb{E}[V_{n+1}(X_{n+1})|X_n = x, \alpha_n = a](x, a) \in \mathbb{R}^d \times \mathbb{A},$$

De plus, si l'optimum dans la formule de programmation dynamique est atteint au temps n en $a_n^*(x) \in arg\min_{a \in \mathbb{A}} Q_n(x, a)$, cela nous définit un processus de contrôle optimal $\alpha^* = (a_n^*(X_n^*))_n$, avec X^* le processus markovien défini par :

$$X_{n+1}^* = F(X_n^*, a_n^*(X_n^*), \varepsilon_{n+1}), \ n = 0, ..., N-1, \ X_0^* = x_0$$

7.1 Formulation

Essayons d'appliquer cet algorithme au problème du pricing d'options américaines.
La formulation de l'évolution du processus $(X_n)_n$ est bien adaptée au modèle de Black and Scholes .

En notant τ le temps d'arrêt de l'exercice, la fonctionnelle de gain à optimiser s'exprime en fonction de τ :

$$J(\tau) = \mathbb{E}[exp(\frac{-rT\tau}{N})(K - X_\tau)^+]$$

Développons cette expression :

$J(\tau) = \mathbb{E}[\sum_{n=0}^{N} exp(\frac{-rTn}{N})(K - X_n)^+ \mathbb{1}_{\tau=n}]$

On introduit le processus de contrôle $(\alpha_n)_n$, et le processus $(\phi_n)_n$ défini par :

$$\phi_0 = 1$$

$$\phi_{n+1} = \phi_n(1 - \alpha_n)$$

Le processus ϕ sert de mémoire pour indiquer que l'option n'a pas encore été exercée. Quant au processus α,c'est un processus indépendant qui sert de variable de décision : exercice immédiat ou conservation de l'option.
Nous pouvons donc poser

$$\mathbb{1}_{\tau=n} = \alpha_n\phi_n$$

Nous avons alors :$J(\alpha) = \mathbb{E}[\sum_{n=0}^{N} exp(\frac{-rTn}{N})(K - X_n)^+\alpha_n\phi_n]$

$$= \sum_{n=0}^{N-1} \mathbb{E}[exp(\frac{-rTn}{N})(K - X_n)^+\alpha_n\phi_n] + \mathbb{E}[exp(-rT)(K - X_N)^+\phi_N]$$

Nous pouvons omettre la variable α_N, car il n'y a plus de décision au temps N. Nous pouvons reconnaître les expression des fonctionnelles f et g, en posant :

$$\forall n = 0, ..., N-1, f(X_n, \alpha_n) = -exp(\frac{-rTn}{N})(K - X_n)^+\alpha_n\phi_n$$

$$,\text{et } g(X_N) = -exp(-rT)(K - X_N)^+\phi_N$$

Le rajout du signe - est justifié car nous cherchons à maximiser l'expression $J(\alpha)$.
La phase suivante consiste à appliquer des versions de l'algorithme de programmation dynamique, par simulation Monte de Carlo sur un grand nombre de trajectoires possibles de $(X_n)_n$.

7.2 Algorithme NNContPI :

Input : the training distributions $(\mu_n)_{n=0}^{N-1}$;
Output : estimates of the optimal strategy $(\hat{a}_n)_{n=0}^{N-1}$;
for $n = N-1, ..., 0$ **do**
Compute :

$$\hat{\beta}_n \in argmin_\beta \mathbb{E}[f(X_n, A(X_n;\beta)) + \sum_{k=n+1}^{N-1} f(X_n^\beta, \hat{a}_n(X_k^\beta))]$$

Where $X_n \sim \mu_n$ and where $(X_k^\beta)_{k=n+1}^N$ is defined by induction as :

$$\begin{cases} X_{n+1}^\beta = F(X_n, A(X_n;\beta), \epsilon_{n+1}) \\ X_{k+1}^\beta = F(X_k^\beta, \hat{a}_k(X_k^\beta;\beta), \epsilon_{k+1}) \text{ for k=n+1,...,N-1.} \end{cases}$$

Set $\hat{a}_n = A(., \hat{\beta}_n)$
$\hat{a}_n$ is the estimate of the optimal policy at time n.
end

Cet algorithme est implémenté par des réseaux de neurones successifs indexés par le temps n, dont les paramètres d'entrée sont une simulation des prix du sous-jacent à la période n.
A chaque étape n ,l'algorithme garde en mémoire les stratégies optimales $(\hat{a}_n)_{k=n}^{N-1}$ et prédit la stratégie optimale au temps n en minimisant $\mathbb{E}[f(X_n, A(X_n;\beta)) + \sum_{k=n+1}^{N-1} f(X_n^\beta, \hat{a}_n(X_k^\beta))]$
Cette expression comprend deux termes : la fonctionnelle de coût correspondant à la période n, et les fonctionnelles prévisionnelles des temps ultérieurs conditionnées par la variable X_n ainsi que les stratégies optimales $(\hat{a}_k)_{k=n}^{N-1}$.
A la fin,la connaissance des décisions optimales $(\hat{a}_n)_{n=0}^{N-1}$ nous permet d'évaluer le processus optimal α^* pour ensuite en déduire le prix de l'option.

7.3 Algorithme Regress Now (Hybrid-Now) :

Input : the training distributions $(\mu_n)_{n=0}^{N-1}$;
Output :
-estimates of the optimal strategy $(\hat{a}_n)_{n=0}^{N-1}$;
-estimate of the value function $(\hat{V}_n)_{n=0}^{N-1}$;
Set $\hat{V}_N = g$ **for** $n = N-1, ..., 0$ **do**

Compute :

$$\hat{\beta}_n \in argmin_\beta \mathbb{E}[f(X_n, A(X_n;\beta)) + \hat{V}_{n+1}(X_{n+1}^\beta)]$$

Where $X_n \sim \mu_n$ and $X_{n+1}^\beta = F(X_n, A(X_n;\beta), \epsilon_{n+1})$
Set $\hat{a}_n = A(., \hat{\beta}_n)$
$\hat{a}_n$ is the estimate of the optimal policy at time n.
Compute :

$$\hat{\theta}_n \in argmin_\theta \mathbb{E}[(f(X_n, \hat{a}_n(X_n)) + \hat{V}_{n+1}(X_{n+1}^{\hat{\beta}_n}) - \Phi(X_n, \theta))^2]$$

Set $\hat{V}_n = \Phi(., \hat{\theta}_n)$
$\hat{V}_n$ is the estimate of the value function at time n.
end

Cet algorithme est implémenté par deux types de réseaux chacun indexé par le temps n, dont les paramètres d'entrée sont une simulation des prix du sous-jacent à la période n. Le premier réseau de neurones prédit les stratégies optimales, tandis que le deuxième calcule les fonctions de valeur de l'option.
A chaque étape n ,l'algorithme garde en mémoire les stratégies optimales $(\hat{a}_n)_{k=n+1}^{N-1}$, ainsi que les fonctions de valeurs actualisées $(\hat{V}_k)_{k=n+1}^{N}$, puis le premier réseau de neurones prédit la stratégie optimale au temps n, $(\hat{a}_n) = A(X_n;\beta)$ en minimisant $\mathbb{E}[f(X_n, A(X_n;\beta)) + \hat{V}_{n+1}(X_{n+1}^\beta)]$
Cette expression comprend deux termes : la fonctionnelle de coût correspondant à la période n, ainsi que la valeur actualisée prévisionnelle au temps n+1, conditionné par la variable X_n.
En second lieu, le deuxième réseau de neurones calcule la fonction valeur au temps n, $\hat{V}_n = \Phi(X_n, \theta)$ en minimisant l'erreur quadratique moyenne : $\mathbb{E}[(f(X_n, \hat{a}_n(X_n)) + \hat{V}_{n+1}(X_{n+1}^{\hat{\beta}_n}) - \Phi(X_n, \theta))^2]$
A la fin,la connaissance des décisions optimales $(\hat{a}_n)_{n=0}^{N-1}$ nous permet d'évaluer le processus optimal α^* pour ensuite en déduire le prix de l'option.

7.4 Résultats

Comparaison des 5 méthodes :

	CRR	LSM	LSM + NN	NNContPI	Regress Now
$S_0 > K$	0.0333	0.0344	0.0311	0.03	0.045
$S_0 = K$	0.0749	0.0742	0.0718	0.063	0.0652
$S_0 < K$	0.1370	0.1362	0.1305	0.1273	0.1299

On remarque que les 2 nouveaux algorithmes donnent des valeurs très proches aux prix donnés par les algorithmes classiques (CRR et LSM).

8 DataChallenge : Exotic pricing with multidimensional non-linear interpolation

8.1 Description du challenge

Le data challenge, fourni par Natixis, consiste en l'utilisation d'un ensemble d'apprentissage de 1 million de prix d'options pour apprendre à pricer un type spécifique d'options décrit par 23 paramètres. Tous les paramètres ont été normalisés entre 0 et 1 :

- S_1, S_2, s_3 : sous-jacents 1, 2 et 3
- μ_1, μ_2, μ_3 : taux d'intérêt moyen des sous-jacents 1, 2 et 3
- $\sigma_1, \sigma_2, \sigma_3$: volatitilé des sous)jacents 1, 2 et 3
- $\rho_{12}, \rho_{23}, \rho_{13}$: corrélation sous-jacents 1-2, 2-3 et 1-3
- Bonus : coupon payé à la fin de chaque période jusqu'à ce que l'accord soit terminé ou rappelé
- Yetibarrier : barrière sur le minimum des 3 sous-jacents au-dessous duquel le coupon Yeti est payé
- Yeticoupon : coupon Yeti
- PhoenixBarrier : barrière sur le minimum des 3 sous-jacents qui rappelle l'accord
- Phoenixcoupon : coupon payé quand le rappel survient pour compenser la perte des coupons Yeti et Bonus
- PDIBarrier : barrière du putdown dans le options putdown and in si aucun recall n'est arrivé
- PDIGearing : nombre d'options PDI pour une unité notionnelle du deal
- PDIType : -1 pour un put, +1 pour un call
- Maturity : maturité de l'accord
- NbDates : nombre de dates de réinitialisation de l'autocall

  Et l'output est le payoff des options, défini comme suit :

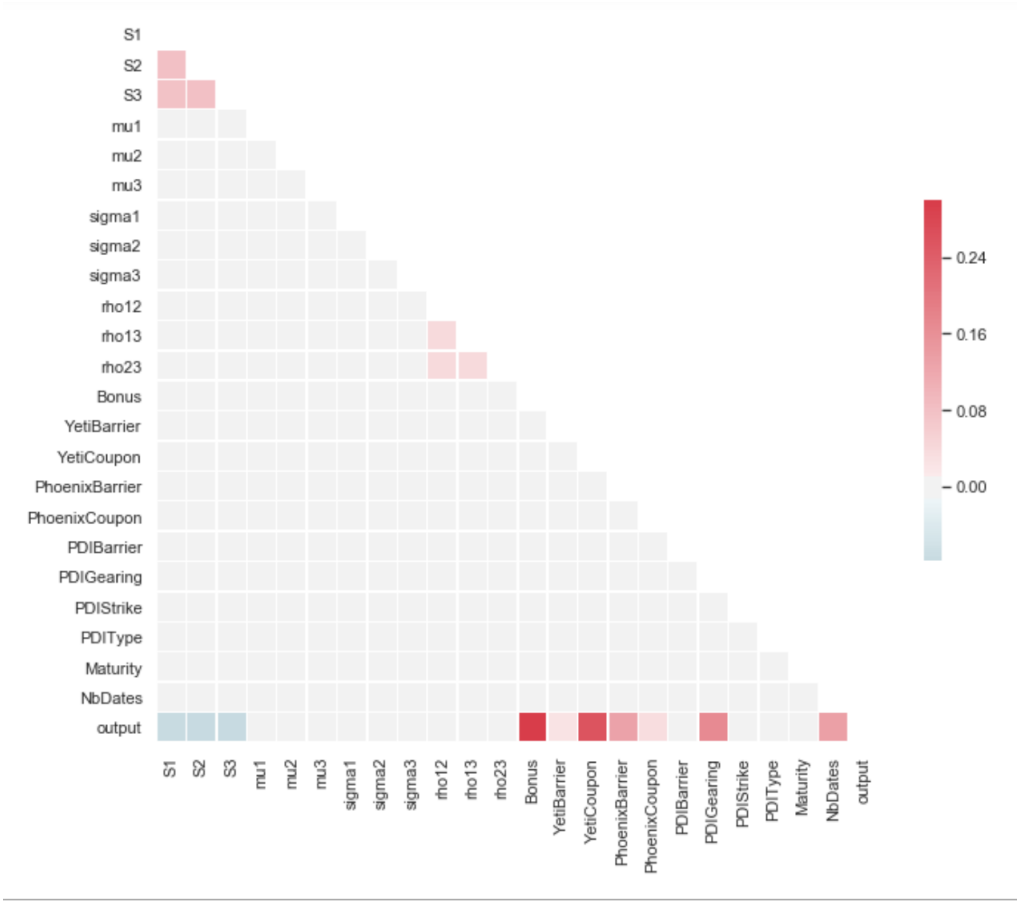
$$Payoff(t_i) = (BO_i + YE_i.\mathbf{1}_{YB_i<P_i} + (PH_i).\mathbf{1}_{AB_i<P_i}).\prod_{j=1}^{i-1}\mathbf{1}_{P_j<AB_i} +$$

$$(Gearing(\delta_{type}.(P_n - Strike)^+))(1 - \prod_{k=0}^{n-1}\mathbf{1}_{DB_k<P_k})(\prod_{i=0}^{n}\mathbf{1}_{P_i<AB_i})$$

Avec $P_i = min(S_1(t_i), S_2(t_i), S_3(t_i))$

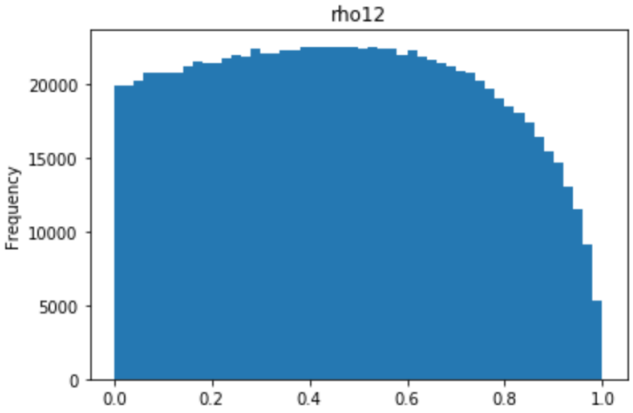
Et BO_i le Bonus coupon, YE_i le Yéti coupon, PH_i le Phoenix coupon.

L'ensemble de test contient 400 000 données, et la métrique utilisée est la somme des moindre carrés sur ces données. Ci-dessous un graphique qui représenta la corrélation entre les paramètres (l'output est le prix de l'option) :



8.2 Exploration de la base de données

Comme les prix sont normalisés, la plupart des prix sont autour de 0.5. Toutes les autres variables, mis à part $NbDates$ et $rho12, rho13, rho23$ sont équiprobablement répartis entre 0 et 1.



Il n'y a pas de données manquantes dans les bases de données. La matrice de corrélation pour chacune des options doit être semi-définie positive ; pourtant nous trouvons 2 pour cent de données aberrantes. Nous choisissons d'omettre celles-ci pour la phase d'apprentissage. De plus, comme les options peuvent être des put ou des call, nous émettons l'hypothèse que les options à $PDIType > 0.5$ sont des put et celles à $PDIType < 0.5$ sont des call. Cette hypothèse est fondée par la distribution équiprobable.

8.3 Phases d'entraînement et de prédiction

Nous entraînons un réseau de neurones sur les put et un autre sur les call.

Les deux réseaux sont denses et ont la même structure : La couche d'entrée a 22 neurones pour les 22 features. la couche 1 a 512 neurones, la couche 2 128, la couche 3 32, et la couche 4 1. On rajoute un dropout de 20 pour cent et des batchnomalizations.

Nous obtenons une erreur de 26 en soumettant le résultat sur le site.

8.4 Conclusion

Pour aller plus loin, il faudrait essayer une nouvelle structure de réseaux de neurones avec méta-apprentissage ; et tenter de nouvelles méthodes de feature selection plus efficaces que celle avec la matrice de corrélation, comme les Wrapper methods.

Références

[1] Francis A. Longstaff and Eduardo S. Schwartz. Valuing american options by simulation : A simple least-squares approach. *The Review of Financial Studies*, pages Pages 113 – 147, 2001.