Optimal Stopping via Randomized Neural Networks

علیرضا عظیمی‌نیا 401 205 061
کیانا عسگری 473 100 97

۱ مقدمه

چالش پیدا کردن بهترین زمان برای توقف یک فرایند به منظور بیشینه‌کردن پاداش یا کمینه‌کردن هزینه، به مسئله یافتن زمان توقف بهینه (optimal stopping time) معروف است. این مسئله یکی از پرکاربردترین مسائل در آمار، اقتصاد و ریاضیات مالی است. با وجود اینکه الگوریتم‌های کنونی برای این مسئله مانند برنامه‌نویسی پویا (dynamic programming) یا استفاده از یادگیری تقویتی (reinforcement learning) عملکرد خوبی از خود نشان داده‌اند، اما در ابعاد بالا عملکرد خوبی ندارند.

یکی از مهم‌ترین کاربردهای مسئله‌ی توقف بهینه در پیدا کردن بهترین زمان اجرای یک قرارداد اختیار آمریکایی (American option) است. مقاله‌ی اصلی، دو الگوریتم بر مبنای شبکه‌های عصبی برای حل مسئله توقف بهینه معرفی می‌کنند که تنها پارامترهای لایه آخر آن‌ها یاد گرفته می‌شوند. ثابت نگه داشتن پارامترهای پنهان در طول فرایند آموزش مسئله‌ی بهینه‌سازی را از حالت غیر محدب به حالت محدب تبدیل می‌کند. توابع پنهان در شبکه می‌توانند نقش توابع پایه‌ای در رگرسیون را بازی کنند. مقاله‌ی اصلی در ادامه، الگوریتم‌های خود را به صورت خاص برای تعیین قیمت قراردادهای اختیار آمریکایی تشریح می‌کند.

۲ تاریخچه

دو ایده‌ی معروف برای حل مسئله‌ی توقف بهینه، استقرای بازگشتی یا به عبارتی برنامه ریزی پویا [۱] و استفاده از یادگیری تقویتی [۲] است. هر دوی این روش‌ها، بر اساس تخمین کمترین مربعات هستند

[1] J. N. Tsitsiklis and B. Van Roy. Regression methods for pricing complex american-style options. IEEE Transactions on Neural Networks, 12(4), 2001.

[2] J. N. Tsitsiklis and B. Van Roy. Optimal stopping of markov processes: Hilbert space theory, approximation algorithms, and an application to pricing high-dimensional financial derivatives. IEEE Transactions on Automatic Control, 44(10), 1999.

و نیاز به انتخاب مجموعه‌ای از توابع پایه دارند. علاوه بر اینکه انتخاب این مجموعه دشوار است، با افزایش ابعاد مسئله تعداد توابع پایه مورد نیاز به صورت چند جمله‌ای یا نمایی افزایش می‌یابد که استفاده از این ایده‌ها را در ابعاد بالا غیرممکن می‌سازد.

ایده‌ای نسبتاً جدید، استفاده از شبکه‌های عصبی به عنوان جایگزین توابع پایه است. شبکه‌های عصبی در فضای توابع $L^p(\mu)$ برای $1 \leq p < \infty$ و اندازه متناهی μ چگال هستند و در اکثر مسائل به نفرین ابعاد (curse of dimensionality) دچار نمی‌شوند. در مقابل، شبکه‌های عصبی توابع غیرمحدبی هستند و روش‌های بر پایه‌ی کاهش گرادیان لزوماً به مینیمم سراسری همگرا نخواهند بود.

مقاله‌ی اصلی تلاش می‌کند هم به مشکل نفرین ابعاد و هم به مشکل همگرایی اثبات‌پذیر غلبه کند.

۳ قرارداد اختیار خرید آمریکایی

چنین قراردادی به دارنده‌ی قرارداد اختیار خرید یک دارایی با قیمتی معین را در هر لحظه‌ای تا زمان سررسید می‌دهد. چنین قراردادی را می‌توان با یک قرارداد برمودان (Bermudan Option) تخمین زد؛ به این شکل که دارنده‌ی قرارداد تنها امکان اجرای آن را در زمان‌های گسسته $t_0 < t_1 < \cdots < t_N$ خواهد داشت. تعیین قیمت این نوع قراردادها یکی از کاربردهای مسئله‌ی توقف بهینه است.

فرض می‌کنیم عایدی (payoff) قرارداد که آن را با g نشان می‌دهیم تنها به قیمت فعلی سهام بستگی دارد: $g = g(X_t)$. به عنوان مثال $g(X) = \max\{0, X - K\}$.

فرض می‌کنیم $(X_t)_{t\geq 0}$ یک فرایند مارکوف باشد که قیمت سهام‌ها را توصیف می‌کند. چنین فرایندی می‌تواند توسط یک معادله دیفرانسیل، مانند مدل Black-Scholes، توصیف شود:

$$\begin{aligned} dX_t &= X_t(r\,dt + \sigma\,dW_t), \\ X_0 &= x_0 \end{aligned}$$

معادله بالا یک حرکت براونی هندسی را توصیف می‌کند:

$$X_t = x_0 \exp\left[\left(\mu - \frac{\sigma^2}{2}\right)t + \sigma W_t\right]$$

تقریبی گسسته از قرارداد اختیار خرید آمریکایی در نظر می‌گیریم: بازه‌ی زمانی $[0, T]$ را به N قسمت مساوی تقسیم می‌کنیم، و برای سادگی، زمان‌ها را با $0, 1, \ldots, N$ نشان می‌دهیم. در این صورت، قیمت قرارداد اختیار خرید در زمان n بصورت زیر است:

$$U_n = \sup_{t\geq n} E\left(\alpha^{t-n} g(X_t) \mid X_n\right)$$

که α نرخ تعدیل است. معادلاً

$$\begin{aligned} U_N &= g(X_N), \\ U_n &= \max\left\{g(X_n), E(\alpha U_{n+1} \mid X_n)\right\}, \quad 0 \leq n < N \end{aligned} \tag{۱}$$

در هر لحظه، عایدی قرارداد $g(X_n)$ در آن لحظه با مقدار $c_n(X_n) = E(\alpha U_{n+1} \mid X_n)$ مقایسه می‌شود و بر اساس آن زمان توقف بهینه τ_n با شروع از زمان n بدست می‌آید:

$$\begin{aligned} &\tau_N = N, \\ &\tau_n = n \text{ if } g(X_n) \geq c_n(X_n), \text{ otherwise } \tau_n = \tau_{n+1} \end{aligned} \tag{۲}$$

به خصوص

$$U_0 = \max\left\{ g(X_0), E\left(\alpha^{\tau_1} g(X_{\tau_1})\right) \right\}$$

۴ شبیه‌سازی Monte Carlo

فرض می‌کنیم روشی برای شبیه‌سازی X_t داریم. هر شبیه‌سازی یک مسیر از قیمت سهام‌ها با شروع از x_0 می‌دهد. m بار قیمت سهام‌ها را شبیه‌سازی می‌کنیم و آنها را به صورت $x_0, x_1^i, \dots, x_N^i$ نشان می‌دهیم. اندیس بالا برای مسیر، و اندیس پایین برای زمان است. با توجه به (۱) اگر استراتژی توقف (۲) را دنبال کنیم، جریان نقدی که نصیبمان می‌شود برابر است با

$$\begin{aligned} &p_N^i = g(x_N^i), \\ &p_n^i = g(x_n^i) \text{ if } g(x_n^i) \geq c_n(x_n^i), \text{ otherwise } p_n^i = \alpha p_{n+1}^i \end{aligned} \tag{۳}$$

به عبارتی دیگر، $\{p_n^i\}_{i=1}^m$ نمونه‌هایی از $\alpha^{\tau_n - n} g(X_n)$ هستند و می‌توان از آنها برای تقریب U_n استفاده کرد. به خصوص با توجه به قانون اعداد بزرگ می‌توان نوشت:

$$U_0 \approx \max\left\{ g(x_0), \frac{1}{m} \sum_{1}^{m} \alpha p_1^i \right\} \tag{۴}$$

۵ تقریب توابع c_n با استفاده از شبکه عصبی

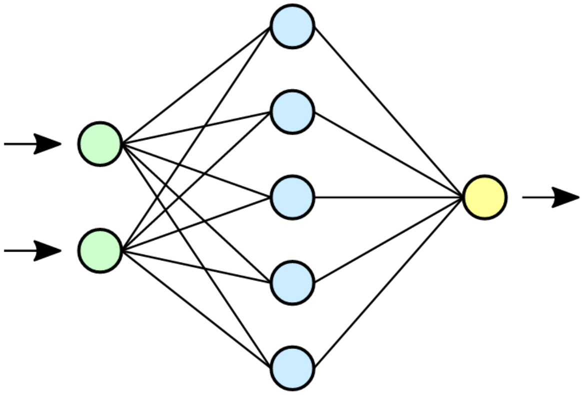
در هر زمان n، مقدار $c_n(x)$ برابر میانگین قیمت تعدیل‌شده قرارداد αU_{n+1} است در صورتی که قیمت فعلی سهام‌ها x باشد و قرارداد را تا زمان $n + 1$ اجرا نکنیم.

برای تقریب هر کدام از c_nها از یک شبکه عصبی دولایه که فقط پارامترهای لایه‌ی آخر یادگرفته شدند استفاده می‌کنیم. (شکل)

تابع فعال‌سازی (activation function) را با σ نشان می‌دهیم. مثلاً leaky ReLU:

$$\sigma(x) = \max(0, x) - \max(0, -ax)$$

فرض کنید d سهام به عنوان ورودی و k گره در لایه میانی داشته باشیم. ماتریس $A \in \mathbb{R}^{k \times d}$ و بردار $b \in \mathbb{R}^k$ که وزن‌ها و بایاس لایه‌ی اول هستند را بصورت تصادفی انتخاب می‌کنیم. A, b برای تمام



c_nها مشترک است. برای هر c_n، بردار $A_n \in \mathbb{R}^k$ و عدد $b_n \in \mathbb{R}$ که وزن و بایاس لایه‌ی آخر هستند باید یاد گرفته شوند. اکنون تقریبی از c_n که با $\widetilde{c}_n$ نشان می‌دهیم در دسترس است:

$$\widetilde{c}_n(x) = A_n \cdot \sigma(Ax + b) + b_n \tag{۵}$$

از طرفی $c_n(x_n^i) = \alpha p_{n+1}^i$. پس A_n و b_n بهینه از طریق حل مسئله رگرسیون خطی

$$A_n \cdot \sigma(Ax_n^i + b) + b_n \approx \alpha p_{n+1}^i, \quad i = 1, \dots, m \tag{۶}$$

بدست می‌آیند. برای این کار، می‌توان از روش مینیمم‌کردن خطای کمترین مربعات استفاده کرد:

$$\text{MSE} = \frac{1}{m} \sum_1^m \left(\widetilde{c}_n(x_n^i) - \alpha p_{n+1}^i\right)^2$$

از آنجایی که p_N^i مشخص است، با حل (۶) می‌توان A_{N-1} و b_{N-1} بهینه را پیدا کرد. اکنون با محاسبه $\widetilde{c}_{N-1}$ از (۵) با توجه به (۳) می‌توان p_{N-1}^i را حساب کرد. سپس از آن می‌توان برای محاسبه $\widetilde{c}_{N-2}$ و p_{N-2}^i استفاده کرد. به همین ترتیب به شکل بازگشتی می‌توان p_n^i را برای هر زمانی حساب کرد. در نهایت با داشتن p_1^iها، از رابطه‌ی (۴) می‌توان تقریبی از قیمت قرارداد در لحظه صفر بدست آورد.

۶ تقریب توابع c_n با یادگیری تقویتی

در قسمت قبل، هر تابع c_n را مجزا تقریب زدیم و از $N-1$ شبکه عصبی برای این کار استفاده کردیم. در این قسمت از یک شبکه عصبی برای تقریب همه‌ی c_nها استفاده می‌کنیم. چنین تقریبی فرمی مشابه (۵) دارد:

$$\widetilde{c}(n, x) = \widetilde{A} \cdot \sigma(A\widetilde{x} + b) + \widetilde{b} \tag{۷}$$

که $A \in \mathbb{R}^{k\times(d+2)}$ و $b \in \mathbb{R}^k$ پارامترهای لایه‌ی اول هستند که به طور تصادفی انتخاب می‌شوند، $\widetilde{x} = (n, N-n; x)$ برداری به طول $d+2$ است، و $\widetilde{A}$ و $\widetilde{b}$ مانند قسمت قبل پارامترهای لایه‌ی آخر هستند و به طور بهینه انتخاب می‌شوند. برای آموزش چنین شبکه‌ای از روش Q-learning استفاده می‌کنیم.

فرض کنید π یک استراتژی توقف باشد. تابع ارزش برای این استراتژی به صورت زیر است:

$$\begin{aligned} Q^{\pi}(t, X_t; \text{stop}) &= g(X_t) \\ Q^{\pi}(t, X_t; \text{cont}) &= \alpha \sum_{a\in\{\text{stop},\text{cont}\}} \pi(a \mid t+1, X_{t+1})\, Q^{\pi}(t+1, X_{t+1}; a) \end{aligned}$$

برای استراتژی بهینه (۲)، داریم:

$$\begin{aligned} Q^{*}(t, X_t; \text{stop}) &= g(X_t) \\ Q^{*}(t, X_t; \text{cont}) &= \alpha \max_{a\in\{\text{stop},\text{cont}\}} Q^{*}(t+1, X_{t+1}; a) \end{aligned} \tag{۸}$$

در زمان N اکشن cont غیرمجاز است. در هر وضعیت (t, X_t) با مقایسه‌ی دو مقدار $Q^*(t, X_t; \text{stop})$ و $Q^*(t, X_t; \text{cont})$ اکشن بهینه در آن وضعیت مشخص می‌شود. چون Q^* تابع ارزش استراتژی توقف بهینه (۲) بود، نتیجه می‌شود: $Q^*(n, X_n; \text{cont}) = c_n(X_n)$. برای تقریب Q^* از تابعی به فرم (۷) استفاده می‌کنیم.

با یک حدس اولیه $\widetilde{c}_0$ شروع کنید. با داشتن $\widetilde{c}_j$، برای $i \in \{1, \dots, m\}$ و زمان $n \in \{1, \dots, N-1\}$ قرار دهید:

$$p_n^i = \max\left\{\widetilde{c}_j(n, x_n^i), g(x_n^i)\right\}$$

توجه کنید که $p_N^i = g(x_N^i)$. اکنون از روش مینیمم‌کردن خطای کمترین مربعات، $\widetilde{c}_{j+1}$ را تعیین کنید:

$$\text{MSE} = \frac{1}{mN} \sum_{n=1}^{N-1} \sum_{i=1}^{m} \left(\widetilde{c}_j(n, x_n^i) - \alpha p_{n+1}^i\right)^2$$

می‌توان ثابت کرد [3] که دنباله $\widetilde{c}_j$ همگراست به تقریبی بهینه به فرم (۷). اکنون می‌توان از (۳) و (۴) قیمت قرارداد را در زمان صفر حساب کرد.

۷ الگوریتم

تعداد $2m$ مسیر از فرایند تصادفی X_t شبیه‌سازی می‌کنیم. در تاریخ سررسید، قیمت قرارداد برابر عایدی آن است: $p_N^i = g(x_N^i)$.

[3] John Tsitsiklis and Benjamin Van Roy. Regression methods for pricing complex american-style options. IEEE Transactions on Neural Networks, 12(4):694–703, 2001. Section 6

در الگوریتم بازگشتی Monte Carlo، ابتدا با استفاده از m مسیر اول و روشی که در بخش مربوطه توضیح داده شد، توابع $\widetilde{c}_n$ را بدست می‌آوریم. سپس با استفاده از m مسیر دوم و رابطه (۴) قیمت تقریبی قرارداد را در زمان صفر حساب می‌کنیم.

صورت کامل این الگوریتم، randomized least squares Monte Carlo (RLSM):

۱: ماتریس $A \in \mathbb{R}^{k\times d}$ و بردار $b \in \mathbb{R}^k$ را به طور تصادفی تولید کنید.

۲: تعداد $2m$ مسیر از فرایند تصادفی X_t (قیمت سهام‌ها در طول زمان) تولید کنید: $x_1^i, \dots, x_N^i$

۳: برای هر مسیر $i \in \{1, \dots, 2m\}$ قرار دهید: $p_N^i = g(x_N^i)$.

۴: برای هر زمان $n \in \{N-1, \dots, 1\}$:

آ: برای هر مسیر $i \in \{1, \dots, 2m\}$ قرار دهید: $\phi(x_n^i) = (\sigma(Ax_n^i + b), 1) \in \mathbb{R}^{k+1}$

ب: جواب مسئله کمترین مربعات

$$\sum_{i=1}^{m} \left(\phi(x_n^i) \cdot x - \alpha p_{n+1}^i\right)^2$$

را θ_n بنامید: $\theta_n \in \mathbb{R}^{k+1}$.

ج: برای هر مسیر $i \in \{1, \dots, 2m\}$: اگر $g(x_n^i) \geq \theta_n \cdot \phi(x_n^i)$، آنگاه قرار دهید $p_n^i = g(x_n^i)$، در غیر این صورت قرار دهید $p_n^i = \alpha p_{n+1}$.

۵: قرار دهید $p_0 = \max\left\{ g(x_0), m^{-1} \sum_{m+1}^{2m} \alpha p_1^i \right\}$.

روند مشابهی را برای الگوریتم یادگیری تقویتی در پیش می‌گیریم. با استفاده از m مسیر اول پارامترهای تابع $\widetilde{c}$ را تقریب می‌زنیم، سپس با m مسیر دوم قیمت اولیه قرارداد را محاسبه می‌کنیم.

صورت کامل این الگوریتم، randomized fitted Q-iteration (RFQI):

۱: ماتریس $A \in \mathbb{R}^{k\times(d+2)}$ و بردار $b \in \mathbb{R}^k$ را به طور تصادفی تولید کنید.

۲: تعداد $2m$ مسیر از فرایند تصادفی X_t تولید کنید.

۳: قرار دهید $\theta_0 = 0 \in \mathbb{R}^{k+1}$ ، $j = 0$.

۴: تا همگرایی θ_j فرایند زیر را تکرار کنید:

آ: برای هر مسیر $i \in \{1, \dots, 2m\}$ قرار دهید: $p_N^i = g(x_N^i)$.

ب: برای هر زمان $n \in \{1, \dots, N-1\}$ و هر مسیر $i \in \{1, \dots, 2m\}$ قرار دهید:

$$\phi(n, x_n^i) = \left(\sigma(A\widetilde{x}_n^i + b), 1\right) \in \mathbb{R}^{k+1}$$
$$p_n^i = \max\left\{ g(x_n^i), \phi(n, x_n^i) \cdot \theta_j \right\}$$

که $\widetilde{x}_n^i = (n, N-n; x_n^i)$.

ج: جواب مسئله کمترین مربعات

$$\sum_{n=1}^{N-1} \sum_{i=1}^{m} \left(\phi(n, x_n^i) \cdot x - \alpha p_{n+1}^i\right)^2$$

را θ_j بنامید: $\theta_j \in \mathbb{R}^{k+1}$.

د: قرار دهید: $j \leftarrow j+1$

۵: از روی

$$p_N^i = g(x_N^i),$$
$$p_n^i = g(x_n^i) \text{ if } g(x_n^i) \geq \phi(n, x_n^i) \cdot \theta, \text{ otherwise } p_n^i = \alpha p_{n+1}^i$$

مقادیر p_1^i را حساب کنید.

۶: قرار دهید $p_0 = \max\left\{ g(x_0), m^{-1} \sum_{m+1}^{2m} \alpha p_1^i \right\}$.

۸ شبیه‌سازی

کد پیاده‌سازی الگوریتم‌های بالا در فایل `simulation.py` قرار دارد. در پیاده‌سازی و شبیه‌سازی این الگوریتم‌ها ملاحظات زیر در دستور کار بوده است:

- از مدل Black-Scholes برای شبیه‌سازی X_t استفاده شده است. برای تولید یک مسیر X_t از الگوریتم زیر استفاده کردیم:

  ۱: گام زمانی dt را به شکل مناسب انتخاب کنید: $dt = T/N$.

  ۲: بردارهای تصادفی $Z_t \in \mathbb{R}^d$، $t = 1, \dots, N$، که عناصر آن‌ها از توزیع استاندارد نرمال آمده‌اند تولید کنید.

۳: برای هر $t = 1, \ldots, N$ قرار دهید

$$X_t = x_0 \prod_{i=1}^{t} \exp\left(\left(\mu - \frac{\sigma^2}{2}\right) dt + \sigma \sqrt{dt}\, Z_i\right)$$

منظور این است که عبارت بالا را درایه به درایه برای Z_i حساب کنید و حاصل را که یه بردار با ابعادی برابر ابعاد Z_t است X_t بنامید.

- درایه‌های ماتریس A و بردار b به صورت تصادفی از توزیع استاندارد نرمال انتخاب شده‌اند.
- تابع عایدی همان $g(x) = (\max(x_1, \ldots, x_d) - K)_+$ است به ازای $K = 100$.
- تابع فعال‌سازی leaky ReLU است: $\sigma(x) = \max(0, x) - \max(0, -ax)$ به ازای $a = 0.5$.
- هایپرپارامترهای الگوریتم به این شکل انتخاب شده‌اند: $T = 1$، $N = 10$، $m = 10\,000$، $k = 20$، $\alpha = 1$، $\mu = 0$، $\sigma = 0.2$.
- تعداد سهام‌ها d از بین $\{5, 10, 50, 100, 500, 1\,000\}$، و قیمت اولیه سهام‌ها x_0 از بین مقادیر $\{80, 100, 120\}$ انتخاب شده‌اند.

نتایج شبیه‌سازی‌ها در دو جدول زیر آمده است. می‌توان دید که اعداد بدست آمده بسیار به نتایج شبیه‌سازی مقاله‌ی اصلی نزدیک است.

۹ نتایج

مقاله‌ی اصلی مقایسه‌ی وسیعی بین دو الگوریتم RLSM و RFQI و الگوریتم‌های مشابه این حوزه از حیث سرعت اجرا و دقت قیمت خروجی انجام داده است. گزاره‌های زیر را مستقیم از مقاله اصلی نقل می‌کنیم:

> In all cases RLSM and RFQI are the fastest algorithms while achieving at least similar prices to the best performing baselines. Their biggest strength are high dimensional problems $d \geq 500$, where this speed-up becomes substantial.
>
> In these high dimensional problems, RLSM outperforms all baselines in terms of prices, even tough RLSM has much less trainable

		price	
d	x_0	RLSM	RFQI
	80	4.64 (0.13)	4.98 (0.06)
5	100	23.94 (0.20)	24.53 (0.09)
	120	48.32 (0.21)	49.25 (0.33)
	80	8.24 (0.08)	8.81 (0.08)
10	100	32.81 (0.16)	33.61 (0.14)
	120	59.30 (0.13)	60.36 (0.17)
	80	21.81 (0.06)	22.84 (0.07)
50	100	52.26 (0.09)	53.51 (0.13)
	120	82.76 (0.11)	84.12 (0.17)
	80	28.40 (0.08)	29.26 (0.05)
100	100	60.47 (0.08)	61.6 (0.08)
	120	92.57 (0.15)	93.81 (0.19)
	80	43.07 (0.06)	43.64 (0.07)
500	100	78.81 (0.10)	79.55 (0.12)
	120	114.55 (0.08)	115.42 (0.15)
	80	49.12 (0.09)	49.63 (0.05)
1000	100	86.37 (0.06)	87.05 (0.11)
	120	123.66 (0.12)	124.42 (0.14)

Table 1: Max call option on Black–Scholes for different number of stocks d and varying initial stock price x_0.

		price	
d	N	RLSM	RFQI
	10	32.73 (0.14)	33.63 (0.20)
10	50	31.54 (0.17)	34.05 (0.13)
	100	31.04 (0.22)	34.23 (0.12)
	10	52.28 (0.09)	53.57 (0.11)
50	50	50.62 (0.10)	53.88 (0.14)
	100	50.15 (0.11)	54.01 (0.10)
	10	60.55 (0.08)	61.59 (0.11)
100	50	58.77 (0.07)	61.93 (0.14)
	100	58.26 (0.09)	62.06 (0.15)

Table 2: Max call option on Black–Scholes for different number of stocks d and higher number of exercise dates N.

parameters. RFQI achieves the highest prices there, and therefore works best, while having considerably less trainable parameters.

RLSM and RFQI are considerably faster than existing algorithms for high dimensional problems. For high dimensions $d \geq 500$, RLSM is about 8 times faster than the fastest baseline NLSM (Neural Least Square Monte Carlo) and about 30 times faster than DOS (Deep Optimal Stopping).

RLSM and RFQI are up to 2400 (and 4800) times faster than LSM (and FQI respectively) with basis functions of order 2, 5 to 16 times faster than NLSM and 20 to 66 times faster than DOS.

When increasing the number of exercise dates for the maxcall option on Black–Scholes from $N = 10$ to $N \in \{50, 100\}$ the Bermudan option price should become closer to the American option price. RFQI is more than 30 times faster than DOS for high dimensions. Increasing the number of dates further, the computation time can become a limiting factor for DOS, while this is not the case for RFQI.

در مورد مقایسه بین دو الگوریتم RLSM و RFQI:

Reinforcement learning techniques usually outperform the backward induction in the Markovian setting. RFQI, achieving similar prices as FQI, therefore naturally outperforms RLSM which achieves similar prices as LSM.

A possible explanation for the outperformance of the reinforcement learning algorithm is the following: The backward induction algorithms have approximately N times the number of trainable parameters used in the reinforcement learning algorithms, since a different network is trained for each discretisation date. Moreover, for the backward induction algorithms, a different continuation value function is approximated for each date, hence, only the data of this date is used to learn the parameters. In contrast, the reinforcement learning methods train their parameters using the data of all dates. Hence, the reinforcement learning methods use N times the number of data to train $1/N$ times the number of parameters, which seems to lead to better approximations.