Optimal Stopping via Randomized Neural Networks

Calypso Herrera[1] Florian Krach[1] Pierre Ruyssen[2]
Josef Teichmann[1]

March 29, 2022

[1]Department of Mathematics, ETH Zurich
[2]Google Brain, Google Zurich

State-of-the-art and Machine Learning Algorithms

- Backward recursion using basis functions
  - Tsitsiklis and Van Roy (2001)
  - Longstaff and Schwartz (2001)
- Backward recursion using neural networks
  - Kohler et al. (2010)
  - Lapeyre and Lelong (2021) and Becker et al. (2020)
  - Becker et al. (2019)
- Reinforcement learning
  - Tsitsiklis and Van Roy (2001) and Li et al. (2009)

American option pricing is an optimal stopping problem

- The price U_n of the discretized American option at time n can be expressed through the Snell envelope

$$U_N := g(X_N),$$
$$U_n := \max\Big(\underbrace{g(X_N)}_{\text{payoff}},\quad \underbrace{\mathbb{E}[\alpha\, U_{n+1} \,|\, X_n]}_{\text{continuation value}}\Big),\quad 0 \le n < N,$$

  where α is the step-wise discounting factor.
- This can equivalently be expressed as the optimal stopping problem

$$U_n = \sup_{\tau \in \mathcal{T}_n} \mathbb{E}[\alpha^{\tau - n} g(X_\tau) \,|\, X_n],$$

  where $\mathcal{T}_n$ is the set of all stopping times $\tau \ge n$. The smallest optimal stopping time is given by

$$\tau_N := N,$$
$$\tau_n := \begin{cases} n, & \text{if } g(X_n) \ge \mathbb{E}[\alpha\, U_{n+1} \,|\, X_n], \\ \tau_{n+1}, & \text{otherwise.} \end{cases}$$

- The price at initial time is

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

Monte Carlo Simulation

- m realizations of the stock paths are sampled, where the i-th realization is denoted by $x^i = (x_0, x_1^i, \ldots, x_N^i)$, with the fixed initial price x_0.
- For each realization i, the cash flow p^i realized by the holder when following the stopping strategy is given by the backward recursion

$$\begin{aligned} p_N^i &= g(x_N^i) \\ p_n^i &= \underbrace{g(x_n^i)}_{\text{payoff}} \underbrace{1_{\{g(x_n^i) \geq c_n(x_n^i)\}}}_{\text{exercise}} + \underbrace{c_n(x_n^i)}_{\text{continuation value}} \underbrace{1_{\{g(x_n^i) < c_n(x_n^i)\}}}_{\text{continue}} . \end{aligned}$$

- As p_1^i are samples of $\alpha^{\tau_1 - 1} g(X_{\tau_1})$, by the strong law of large numbers we have that almost surely

$$U_0 = \max\left(g(X_0), \lim_{m\to\infty} \frac{1}{m} \sum_{i=1}^{m} \alpha p_1^i \right).$$

The continuation value is only used for the stopping decision

- Tsitsiklis and Van Roy (2001)

$$\begin{aligned} p_N^i &= g(x_N^i) \\ p_n^i &= \underbrace{g(x_n^i)}_{\text{payoff}} \underbrace{1_{\{g(x_n^i) \geq c_n(x_n^i)\}}}_{\text{exercise}} + \underbrace{c_n(x_n^i)}_{\text{continuation value}} \underbrace{1_{\{g(x_n^i) < c_n(x_n^i)\}}}_{\text{continue}} \end{aligned}$$

- Longstaff and Schwartz (2001) use the approximated continuation value only for the stopping decision.

$$\begin{aligned} p_N^i &= g(x_N^i) \\ p_n^i &= \underbrace{g(x_n^i)}_{\text{payoff}} \underbrace{1_{\{g(x_n^i) \geq c_n(x_n^i)\}}}_{\text{exercise}} + \underbrace{\alpha p_{n+1}^i}_{\text{discounted future price}} \underbrace{1_{\{g(x_n^i) < c_n(x_n^i)\}}}_{\text{continue}} \end{aligned}$$

The continuation value is approximated by a linear combination of features

- Longstaff and Schwartz (2001)

$$
\begin{aligned}
p_N^i &= g(x_N^i) \\
p_n^i &= \underbrace{g(x_n^i)}_{\text{payoff}} \underbrace{1_{\{g(x_n^i) \geq c_n(x_n^i)\}}}_{\text{exercise}} + \underbrace{\alpha p_{n+1}^i}_{\text{discounted future price}} \underbrace{1_{\{g(x_n^i) < c_n(x_n^i)\}}}_{\text{continue}}
\end{aligned}
$$

- The continuation value is approximated by linear combination of basis functions,

$$c_n(x) = E(\alpha U_{n+1} | x_n = x) \approx \sum_{i=1}^{n} \theta_i f_i(x) = \theta^\top f(x)$$

- where the parameters θ are found by minimizing the loss function

$$\varphi(\theta_n) = \sum_{i=1}^{N} \left(c_{\theta_n}(x_n^i) - \alpha p_{n+1}^i \right)^2$$

- using a linear regression **at each date** $n \in \{N-1, \ldots, 1\}$.

The continuation value is approximated by a neural network

- Kohler et al. (2010) approximate the continuation value in (Tsitsiklis and Van Roy, 2001) by a neural network.

$$\begin{aligned} p_N^i &= g(x_N^i) \\ p_n^i &= \underbrace{g(x_n^i)}_{\text{payoff}} \underbrace{1_{\{g(x_n^i) \geq c_{\theta_n}(x_n^i)\}}}_{\text{exercise}} + \underbrace{c_{\theta_n}(x_n^i)}_{\text{continuation value}} \underbrace{1_{\{g(x_n^i) < c_{\theta_n}(x_n^i)\}}}_{\text{continue}} \end{aligned}$$

- The continuation value is now approximated by a neural network

$$c_n(x) \approx c_{\theta_n}(x)$$

- where the parameters θ_n are found by minimizing the loss function

$$\varphi(\theta_n) = \sum_{i=1}^{N} \left(c_{\theta_n}(x_n^i) - \alpha p_{n+1}^i \right)^2 .$$

- using a gradient descent method at **each date** $n \in \{N-1, \ldots, 1\}$.

The continuation value is approximated by a neural network

- Lapeyre and Lelong (2021) and Becker et al. (2020) approximate the continuation value in (Longstaff and Schwartz, 2001) by a neural network.

$$
\begin{aligned}
p_N^i &= g(x_N^i) \\
p_n^i &= \underbrace{g(x_n^i)}_{\text{payoff}} \underbrace{1_{\{g(x_n^i) \geq c_{\theta_n}(x_n^i)\}}}_{\text{exercise}} + \underbrace{\alpha p_{n+1}^i}_{\text{discounted future price}} \underbrace{1_{\{g(x_n^i) < c_{\theta_n}(x_n^i)\}}}_{\text{continue}}
\end{aligned}
$$

- The continuation value is now approximated by a neural network

$$c_n(x) \approx c_{\theta_n}(x)$$

- where the parameters θ_n are found by minimizing the loss function

$$\varphi(\theta_n) = \sum_{i=1}^{N} \left(c_{\theta_n}(x_n^i) - \alpha p_{n+1}^i \right)^2 .$$

- using a gradient descent method at **each time** $n \in \{N-1, \ldots, 1\}$.

The indicator function is approximated by a neural network

- Becker et al. (2019) approximate the optimal stopping decision in (Longstaff and Schwartz, 2001).

$$\begin{aligned} p_N^i &= g(x_N^i) \\ p_n^i &= \underbrace{g(x_n^i)}_{\text{payoff}} \underbrace{1_{\{g(x_n^i) \geq c_{\theta_n}(x_n^i)\}}}_{f_{\theta_n}(x)} + \underbrace{\alpha p_{n+1}^i}_{\text{discounted future price}} \underbrace{1_{\{g(x_n^i) < c_{\theta_n}(x_n^i)\}}}_{1 - f_{\theta_n}(x)} \end{aligned}$$

- The indicator function is now approximated by a neural network

$$1_{\{g(x_n^i) \geq c_{\theta_n}(x_n^i)\}} \approx f_{\theta_n}(x)$$

- where the parameters θ_n are found by ~~minimizing~~ maximizing the loss function

$$\varphi(\theta_n) = \sum_{i=1}^{N} \alpha p_{n+1}^i \, .$$

- using a gradient descent method at **each time** $n \in \{N-1, \dots, 1\}$.

Randomized Least Squares Monte Carlo (RLSM)

- We propose to use a randomized neural network to approximate the continuation value in (Longstaff and Schwartz, 2001).

$$\begin{aligned} p_N^i &= g(x_N^i) \\ p_n^i &= \underbrace{g(x_n^i)}_{\text{payoff}} \underbrace{\mathbf{1}_{\{g(x_n^i) \geq c_\theta(x_n^i)\}}}_{\text{exercise}} + \underbrace{\alpha p_{n+1}^i}_{\text{discounted future price}} \underbrace{\mathbf{1}_{\{g(x_n^i) < c_\theta(x_n^i)\}}}_{\text{continue}} \end{aligned}$$

- For each date n, going backward, the continuation value is approximated by a neural network

$$c_{\theta_n}(x) = A_n^\top \sigma(Ax + b) + b_n$$

- where the parameters of the hidden layer (A,b) are randomly chosen and not optimized
- and only the parameters of the last layer $\theta_n = (A_n, b_n)$ are optimized by minimizing the loss function

$$\psi_n(\theta_n) = \sum_{i=1}^{m} \left(c_{\theta_n}(x_n^i) - \alpha p_{n+1}^i \right)^2 . \qquad (1)$$

Randomized Least Squares Monte Carlo (RLSM)

Algorithm 1 Optimal stopping via randomized least squares Monte Carlo (RLSM)

```
Input: discount factor $\alpha$, initial value $x_0$
Output: price $p_0$
1: sample a random matrix $A \in \mathbb{R}^{(K-1)\times d}$ and a random vector $b \in \mathbb{R}^{K-1}$
2: simulate $2m$ paths of the underlying process $(x_1^i, \dots, x_N^i)$ for $i \in \{1, \dots, 2m\}$
3: for each path $i \in \{1, \dots, 2m\}$, set $p_N^i = g(x_N^i)$
4: for each date $n \in \{N-1, \dots, 1\}$
   a: for each path $i \in \{1, \dots, 2m\}$, set $\phi(x_n^i) = (\sigma(Ax_n^i + b)^\top, 1)^\top \in \mathbb{R}^K$
   b: set $\theta_n = \alpha \left(\sum_{i=1}^m \phi(x_n^i)\phi^\top(x_n^i)\right)^{-1} \left(\sum_{i=1}^m \phi(x_n^i)p_{n+1}^i\right)$
   c: for each path $i \in \{1, \dots, 2m\}$, set $p_n^i = g(x_n^i)\mathbf{1}_{g(x_n^i)\geq\theta_n^\top\phi(x_n^i)} + \alpha p_{n+1}\mathbf{1}_{g(x_n^i)<\theta_n^\top\phi(x_n^i)}$
5: set $p_0 = \max(g(x_0), \frac{1}{m}\sum_{i=m+1}^{2m} \alpha p_1^i)$
```

Theorem (informal)

As the number of sampled paths m and the number of random basis functions K go to ∞, the price p_0 computed with Algorithm 1 converges to the correct price of the Bermudan option.

Max Call on Black Scholes with RLSM

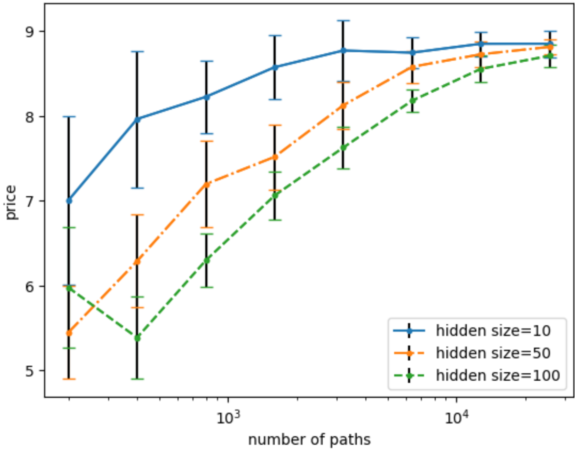

Figure: Mean $\pm$ standard deviation (bars) of the price for a max-call on 5 stocks following the Black Scholes model for RLSM for varying the number of paths m and the number of neurones in the hidden layer K.

An alternative to the backward recursion

- Instead of having a different function approximator for each date, **we use the same function approximator for all dates**

$$C_n(x_n) \approx C_\theta(n, x_n) \ \ \forall n \in \{1, \dots, N-1\}$$

- The algorithm is initialized with a parameter vector θ_0, and **after one sample path the parameter vector is updated** following

$$\theta_{n+1} = \theta_n + \alpha \nabla \varphi(\theta)$$

- This is the principle of **Reinforcement Learning**.
- We first make some random optimal stopping decisions and then we reinforce our learning over the iterations.
- **This is not new!** The first who introduce the reinforcement learning for the optimal stopping were Tsitsiklis and Van Roy (2001).

Fitted Q-learning (FQI)

- Tsitsiklis and Van Roy (2001) use the fitted Q-iteration where the continuation function is approximated by a linear combination of basis functions (Li et al., 2009).

$$\begin{aligned} p_N^i &= g(x_N^i) \\ p_n^i &= \underbrace{g(x_n^i)}_{\text{Payoff}} \underbrace{1_{\{g(x_n^i) \geq c_\theta(n, x_n^i)\}}}_{\text{exercise}} + \underbrace{c_\theta(n, x_n^i)}_{\text{continuation value}} \underbrace{1_{\{g(x_n^i) < c_\theta(n, x_n^i)\}}}_{\text{continue}} \end{aligned}$$

- The continuation value is approximated by a linear combination of basis functions $c_\theta(n, x_n) \approx \theta^T f(n, x)$
- Where the parameters θ are found by minimizing the loss

$$\varphi(\theta) = \sum_{i=1}^{m} \sum_{n=1}^{N} \left(c_\theta(n, x_n^i) - \alpha p_{n+1}^i \right)^2$$

Randomized Fitted Q-Iteration (RFQI)

- We propose to approximate the continuation value in the fitted-Q-iteration by a randomized neural network.

$$\begin{aligned} p_N^i &= g(x_N^i) \\ p_n^i &= \underbrace{g(x_n^i)}_{\text{payoff}} \underbrace{\mathbf{1}_{\{g(x_n^i)\geq c_\theta(x_n^i)\}}}_{\text{exercise}} + \underbrace{c_\theta(x_n^i)}_{\text{continuation value}} \underbrace{\mathbf{1}_{\{g(x_n^i)<c_\theta(x_n^i)\}}}_{\text{continue}} \end{aligned}$$

- As for reinforcement learning algorithms, we first fix the parameters θ^0 and then we update them $\theta^1, \theta^2, \ldots$ through multiples iterations.
- For all dates n, the continuation value is approximated by a neural network

$$c_\theta(\tilde{x}) = A_2^\top \sigma(A\tilde{x} + b) + b_2$$

- where $\tilde{x}_n = (n, x_n)$.
- where the parameters of the hidden layer (A,b) are randomly chosen and not optimized
- and only the parameters of the last layer $\theta = (A_2, b_2)$ are optimized by minimizing the loss function

$$\psi(\theta) = \sum_{n=1}^{N} \sum_{i=1}^{m} \left(c_{\theta_n}(\tilde{x}_n^i) - \alpha p_{n+1}^i \right)^2 . \qquad (2)$$

Algorithm 2 Optimal stopping via randomized fitted Q-Iteration (RFQI)

```
Input: discount factor $\alpha$, initial value $x_0$
Output: price $p_0$
1: sample a random matrix $A \in \mathbb{R}^{(K-1)\times(d+2)}$ and a random vector $b \in \mathbb{R}^{K-1}$
2: simulate $2m$ paths of the underlying process $(x_1^i, \dots, x_N^i)$ for $i \in \{1, \dots, 2m\}$
3: initialize weights $\theta = 0 \in \mathbb{R}^K$
4: for each iteration $\ell$ until convergence of $\theta$
  a: for each path $i \in \{1, \dots, 2m\}$,
    i: set $p_N^i = g(x_N^i)$
    ii: for each date $n \in \{1, \dots, N-1\}$,
          set $\phi(n, x_n^i) = (\sigma(A\tilde{x}_n^i + b), 1) \in \mathbb{R}^K$
          set $p_n^i = \max(g(x_n^i), \phi(n, x_n^i)^\top \theta)$
  b: set $\theta = \alpha \left( \sum_{n=1}^N \sum_{i=1}^m \phi(n, x_n^i)\phi^\top(n, x_n^i) \right)^{-1} \cdot \left( \sum_{n=1}^N \sum_{i=1}^m \phi(n, x_n^i) p_{n+1}^i \right) \in \mathbb{R}^K$
5: set $p_0 = \max(g(x_0), \frac{1}{m} \sum_{i=m+1}^{2m} \alpha p_1^i)$
```

Theorem (informal)

As the number of iterations L, the number of sampled paths m and the number of random basis functions K go to ∞, the price p_0 computed with Algorithm 2 converges to the correct price of the Bermudan option.

Max Call on Black Scholes with RFQI

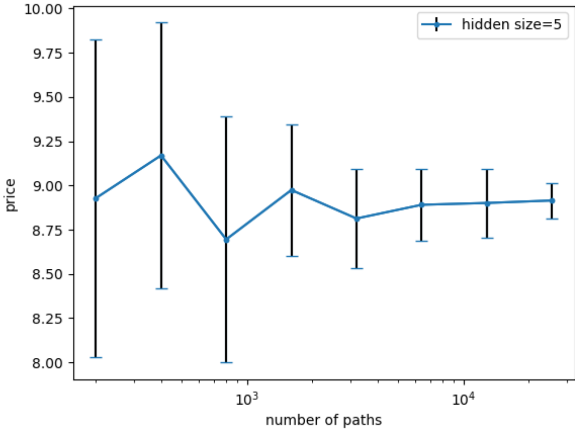

Figure: Mean $\pm$ standard deviation (bars) of the price for a max-call on 5 stocks following the Black Scholes model for RFQI for varying the number of paths m.

Experimental Setup

Besides our algorithms, we also implemented the baselines and provided all of them at `https://github.com/HeKrRuTe/OptStopRandNN`.

- We compare RLSM and RFQI to three backward induction algorithms
  - the state-of-the-art Least Squares Monte Carlo (LSM) (Longstaff and Schwartz, 2001)
  - Neural Least Squares Monte Carlo (NLSM) (Lapeyre and Lelong, 2021)
  - Deep Optimal Stopping (DOS) (Becker et al., 2019)

  and one reinforcement learning approach
  - fitted Q-iteration (FQI) (Tsitsiklis and Van Roy, 1997)
- We used the classical polynomial basis functions up to the second order.
- No Regularization for LSM and FQI

Architecture of Neural Networks 1/2

In order to have a fair comparison in terms of accuracy and in terms of computation time, we use the same number of hidden layers and nodes per layer for all the algorithms.

- As we observed that one hidden layer was sufficient to have a good accuracy (an increase of the number of the hidden layers did not lead to an improvement of the accuracy), we use one hidden layer. Therefore, the NLSM, DOS, and all algorithms that we proposed have only one hidden layer.
- We use 20 nodes for the hidden layer. For RFQI the number of nodes is set to the minimum between 20 and the number of stocks for stability reasons.
- Leaky ReLU is used for RLSM and RFQI and tanh for the randomized recurrent neural network RRLSM(see below). For NLSM and DOS, we use the suggested activation functions, Leaky ReLU for NLSM and ReLU and sigmoid for DOS.

Architecture of Neural Networks 2/2

- The parameters (A, b) of the random neural networks of RLSM and RFQI are sampled using a standard normal distribution with mean 0 and standard deviation 1. Different hyper-parameters were tested, but they didn't have a big influence on the results so we kept the standard choice.
- For the randomized recurrent neural network of RRLSM, we use a standard deviation of 0.0001 for A_x and 0.3 for A_h. Also here different hyper-parameters were tested, and the best performing were chosen and used to present the results. The same holds for tested path-dependent versions of RFQI , however, none of the hyper-parameters performed very well as shown below.
- Some of the reference methods suggest to use the payoff as additional input, while others do not or leave this open. Therefore, we tested using the payoff as input and not using it for each method in each experiment. We came to the conclusion that the backward induction algorithms (LSM, DOS, RLSM) usually work slightly better with, while the reinforcement learning algorithms (FQI, RFQI) usually work slightly better without the payoff. Hence, we show these results.

Max call option on Black–Scholes

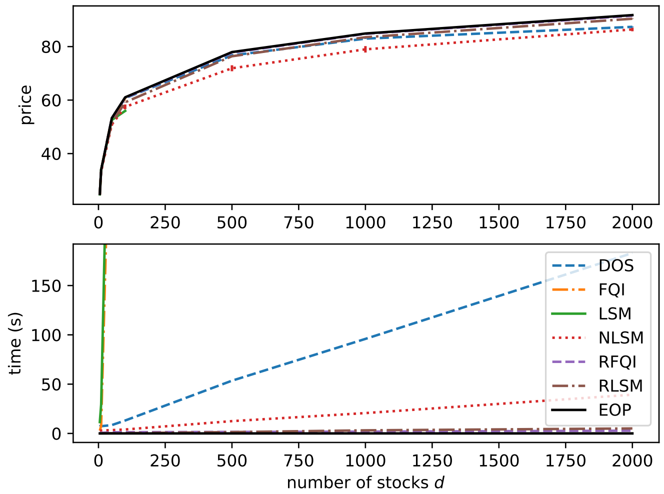

Figure: At the money max call option without dividend on Black–Scholes.

Max call option on Black–Scholes

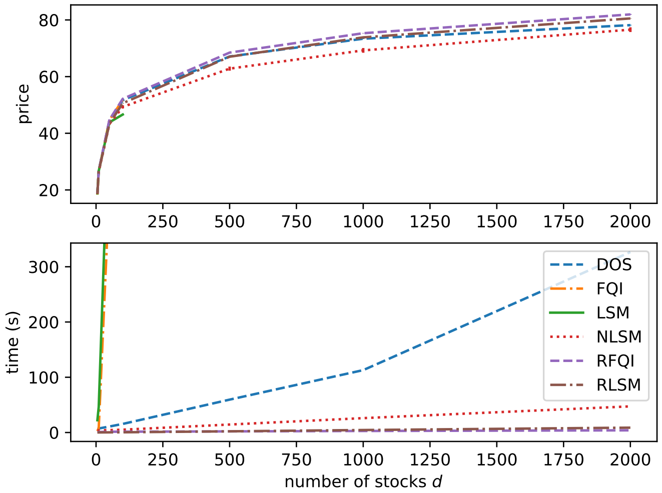

Figure: At the money max call option with dividend on Black–Scholes.

We have very similar results for the other experiments we conducted:

- Heston, Rough Heston
- Min Put, Geometric Put, Basket Call

Moreover, our methods work well to compute Greeks ...

Computation of Greeks 1/2

Most popular Greeks: delta ($\frac{\partial p_0}{\partial x_0}$), gamma ($\frac{\partial^2 p_0}{\partial x_0^2}$), theta ($\frac{\partial p_0}{\partial t}$), rho ($\frac{\partial p_0}{\partial r}$) and vega ($\frac{\partial p_0}{\partial \sigma}$).

- The straight forward method to compute them is via the finite difference (FD) method.
- For theta, rho and vega, the standard forward finite difference method can be used.
- In our experiments we saw, that computing delta works best when using the central finite difference method, where the exercise boundary is frozen to be the one of the central point.
- Moreover, the computation of gamma, as a second derivative, turns out to be unstable when computed with the second order finite difference method, even when using the same technique as for delta. Therefore, we use two alternative ways to circumvent this instability.

Computation of Greeks 2/2

- As reference we use the binomial model with $N = 50'000$ equidistant dates, for which only the finite difference method is used.
- For RLSM the activation function was changed to Softplus, since this worked best, although all other tested activation functions did also yield good results. For NLSM and DOS the computation of theta (and therefore also gamma when using the PDE method), rho and vega was unstable.

		price		delta		gamma		theta	rho	vega
K	algo	FD	regr.	FD	regr.	PDE	regr.			
36	B	0.9192 (–)		-0.1982 (–)		0.0389 (–)		-0.7152 (–)	-0.6188 (–)	10.9106 (–)
36	LSM	0.9032 (0.0093)	0.8993 (0.0061)	-0.1925 (0.0012)	-0.1911 (0.0026)	0.0374 (0.0003)	0.0388 (0.0006)	-0.6803 (0.0063)	-0.6471 (0.0067)	10.7306 (0.0797)
36	RLSM	0.9092 (0.0067)	0.9063 (0.0057)	-0.1952 (0.0027)	-0.1922 (0.0035)	0.0378 (0.0004)	0.0378 (0.0007)	-0.6877 (0.0067)	-0.6491 (0.0126)	10.8385 (0.0863)
36	FQI	0.8533 (0.0108)	0.8622 (0.0081)	-0.1779 (0.0017)	-0.1785 (0.0041)	0.0338 (0.0003)	0.0331 (0.0008)	-0.6037 (0.0064)	-0.8007 (0.0130)	10.4100 (0.0973)
36	RFQI	0.8980 (0.0100)	0.8741 (0.0113)	-0.1953 (0.0085)	-0.1847 (0.0038)	0.0377 (0.0012)	0.0366 (0.0010)	-0.6846 (0.0191)	-0.6354 (0.0354)	10.7590 (0.1674)
36	NLSM	0.8855 (0.0284)	0.8824 (0.0167)	-0.2068 (0.0228)	-0.1898 (0.0027)	–	0.0383 (0.0011)	–	–	–
36	DOS	0.8916 (0.0071)	0.9070 (0.0067)	-0.1926 (0.0028)	-0.1949 (0.0029)	–	0.0379 (0.0008)	–	–	–

- The computation of Greeks is also available on `https://github.com/HeKrRuTe/OptStopRandNN`.

Randomized Recurrent Least Squares Monte Carlo

- We propose to use a randomized recurrent neural network in (Longstaff and Schwartz, 2001).

$$\begin{aligned} p_N^i &= g(x_N^i) \\ p_n^i &= \underbrace{g(x_n^i)}_{\text{payoff}} \underbrace{\mathbf{1}_{\{g(x_n^i) \geq c_\theta(x_n^i)\}}}_{\text{exercise}} + \underbrace{\alpha p_{n+1}^i}_{\text{discounted future price}} \underbrace{\mathbf{1}_{\{g(x_n^i) < c_\theta(x_n^i)\}}}_{\text{continue}} \end{aligned}$$

- The continuation value is approximated by a **recurrent neural network**

$$\begin{cases} h_n &= \sigma(A_x x_n + A_h h_{n-1} + b) \\ c_{\theta_n}(h_n) &= A_n^\top h_n + b_n \end{cases}$$

- where the parameters of the hidden state (A_x, A_h, b) are randomly chosen and not optimized
- and only the parameters of the readout map $\theta_n = (A_n, b_n)$ are optimized by minimizing the loss function

$$\psi_n(\theta_n) = \sum_{i=1}^{m} \left(c_{\theta_n}(x_n^i) - \alpha p_{n+1}^i \right)^2 .$$

- We go forward for computing h_n and backward for computing p_n.

Algorithm 3 Optimal stopping via randomized recurrent neural network (RRLSM)

```
Input: discount factor $\alpha$, initial value $x_0$, initial latent variable $h_{-1} = 0$
Output: price $p_0$
1: sample random matrices $A_x \in \mathbb{R}^{(K-1)\times d}$, $A_h \in \mathbb{R}^{(K-1)\times(K-1)}$ and a
random vector $b \in \mathbb{R}^{K-1}$
2: simulate $2m$ paths of the underlying process $(x_1^i, \dots, x_N^i)$ for $i \in \{1, \dots, 2m\}$
3: for each path $i \in \{1, \dots, 2m\}$, set $p_N^i = g(x_N^i)$
4: for each date $n \in \{0, \dots, N-1\}$
  a: for each path $i \in \{1, \dots, 2m\}$
     set $h_n^i = \sigma(A_x x_n^i + A_h h_{n-1}^i + b)$
5: for each date $n \in \{N-1, \dots, 1\}$
  a: for each path $i \in \{1, \dots, 2m\}$
     set $\phi_n^i = ((h_n^i)^\top, 1)^\top \in \mathbb{R}^K$
  b: set $\theta_n = \alpha \left(\sum_{i=1}^m \phi_n^i (\phi_n^i)^\top\right)^{-1} \left(\sum_{i=1}^m \phi_n^i p_{n+1}^i\right)$
  c: for each path $i \in \{1, \dots, 2m\}$, set $p_n^i = g(x_n^i)\mathbf{1}_{g(x_n^i)\geq\theta_n^\top\phi_n^i} + \alpha p_{n+1}^i \mathbf{1}_{g(x_n^i)<\theta_n^\top\phi_n^i}$
6: set $p_0 = \max(g(x_0), \frac{1}{m}\sum_{i=m+1}^{2m} \alpha p_1^i)$
```

Theorem (informal)

As the number of sampled paths m and the number of random basis functions K go to ∞, the price p_0 computed with Algorithm 3 converges to the correct price of the Bermudan option.

Max Call on Black Scholes with RFQI

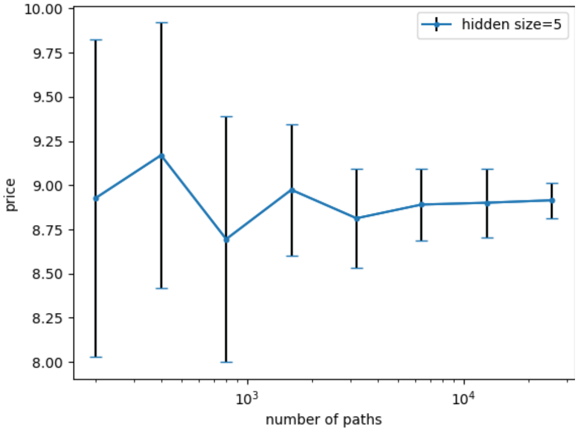

Figure: Mean $\pm$ standard deviation (bars) of the price for a max-call on 5 stocks following the Black Scholes model for RFQI for varying the number of paths m.

Fractional Brownian Motion

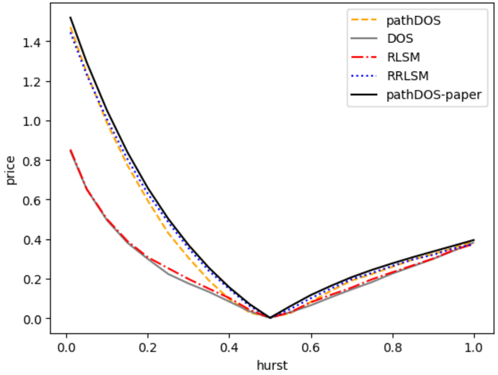

Figure: Payoff identity. Algorithms processing path information outperform.
DOS: Deep Optimal Stopping (Becker et al., 2019)
pathDOS: Deep Optimal Stopping using the entire path as input
pathDOS-paper: values reported in the paper (Becker et al., 2019)
RLSM: Randomized Least Squares Monte Carlo
RRLSM: Randomized Recurrent Least Squares Monte Carlo

Fractional Brownian Motion

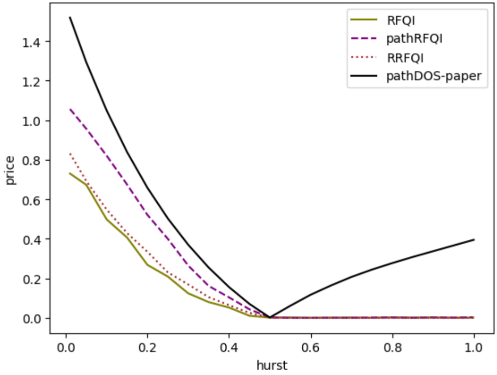

Figure: Reinforcement learning does not work well in non-Markovian case.
RFQI: Randomized Fitted Q-Iteration
pathRFQI: Randomized Fitted Q-Iteration
RRFQI: Randomized Recurrent Fitted Q-Iteration
pathDOS-paper: values reported in the paper (Becker et al., 2019)

Fractional Brownian Motion

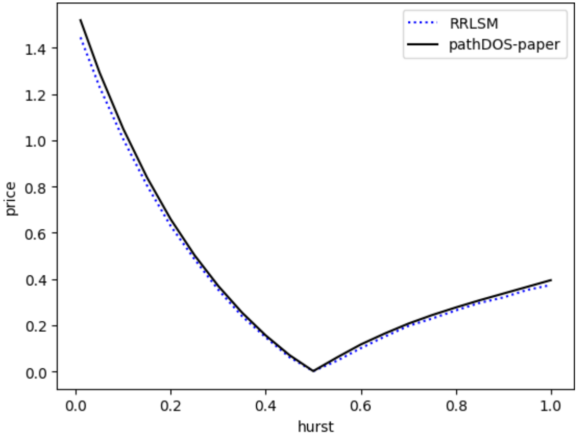

Figure: Randomized Recurrent Least Monte Carlo (RRLSM) achieves similar results as reported in deep optimal stopping, while using only 20K paths instead of 4M for training which took only 4s instead of the reported 430s

Conclusion 1/2

- Based on a broad study of machine learning based approaches to approximate the solution of optimal stopping problems, we introduced two simple and powerful approaches, RLSM and RFQI. As state-of-the-art algorithms, they are very simple to implement and have convergence guarantees. Moreover, similarly to the neural network methods, they are easily scalable to high dimensions and there is no need to choose basis functions by hand. Furthermore, in our empirical study we saw that RLSM and RFQI are considerably faster than existing algorithms for high dimensional problems. In particular, 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.
- In our Markovian experiments, RFQI often achieves the best results and if not, usually is very close to the best performing baseline method under consideration, reconfirming that reinforcement learning methods surpass backward induction methods.
- In contrsat to NLSM and DOS, our methods ensure stable results for the computation of theta, rho and vega.

Conclusion 2/2

- In our non-Markovian experiments on fractional Brownian Motion, our randomized recurrent neural network algorithm RRLSM achieves similar results as the path-version of DOS, while requiring less training data and being much faster. However, this example also brought up the limitations of reinforcement learning based approaches, in particular of RFQI, which do not work well in those non-Markovian experiments.
- In our non-Markovian experiments on rough Heston, we concluded that there is no need of using a recurrent neural network, since RLSM has similar results as RRLSM. This is also the case with DOS and pathDOS.
- Overall, the speed of our algorithms is very promising for applications in high dimensions and with many discretization times, where existing methods might become impractical and where our methods show very reliable performance.
- To summarize, we suggest to use RFQI for Markovian problems, RLSM for non-Markovian processes which do not have a strong path-dependence, as the stock price of rough Heston and finally RRLSM for non-Markovian processes which have a strong path-dependence like fractional Brownian Motion.

Optimal Stopping via Randomized Neural Networks

- The paper is avaible on `https://arxiv.org/abs/2104.13669`
- The code is avaible on `https://github.com/HeKrRuTe/OptStopRandNN`

Thank you

for your attention

Bibliography I

Becker, S., Cheridito, P., and Jentzen, A. (2019). Deep optimal stopping. Journal of Machine Learning Research, 20:74.

Becker, S., Cheridito, P., and Jentzen, A. (2020). Pricing and hedging american-style options with deep learning. Journal of Risk and Financial Management.

Kohler, M., Krzyżak, A., and Todorovic, N. Z. (2010). Pricing of high-dimensional american options by neural networks. Wiley-Blackwell: Mathematical Finance.

Lapeyre, B. and Lelong, J. (2021). Neural network regression for bermudan option pricing. Monte Carlo Methods and Applications.

Li, Y., Szepesvari, C., and Schuurmans, D. (2009). Learning exercise policies for american options. In International Conference on Artificial Intelligence and Statistics.

Longstaff, F. A. and Schwartz, E. S. (2001). Valuing american options by simulation: a simple least-squares approach. Review of Financial Studies.

Bibliography II

Tsitsiklis, J. and Van Roy, B. (1997). Optimal stopping of markov processes: Hilbert space theory, approximation algorithms, and an application to pricing high-dimensional financial derivatives. IEEE Transactions on Automatic Control, 44:1840–1851.

Tsitsiklis, J. and Van Roy, B. (2001). Regression methods for pricing complex american-style options. IEEE Transactions on Neural Networks, 12(4):694–703.