Monte-Carlo for high-dimensional problems in quantitative finance

H. Mete Soner, ORFE, Princeton

joint work with Max Reppen, Boston University

6th Asian Quantitative Finance Seminar

January 23, 2021

Outline

Recent Results

Two relatively recent papers use quite classical Monte-Carlo type method together deep neural networks to study high dimensional optimal control problems.

Deep learning approximation for stochastic control problems, by
Han, J. and E, W., published in
Deep Reinforcement Learning Workshop, NIPS, 2016.

Solving high-dimensional partial differential equations using deep learning, by
Han, J., Jentzen, A. and E, W., published in
Proceedings of the National Academy of Sciences, 115/34, 8505–8510, 2018.

This method was used successfully in several papers that I will outline next.

The method will be described afterwards.

Deep optimal stopping, by Becker, S., Cheridito, P. and Jentzen, A., published in *Journal of Machine Learning Research*, 20/4,1–25, 2019.

An example studied in this paper is the following :

- $X_t \in \mathbb{R}^d$, with $dX_t^{(i)} = X_t^{(i)}\ [rdt + \sigma dW_t^{(i)}]$, where $W^{(i)}$'s are independent Brownian motions.
- One can stop at N many time points and collect

$$\varphi(t, X_t) = e^{-rt}\left(\max_i X_t^{(i)} - K\right)^+.$$

- $S_0 = 110, K = 100, \sigma = 0.2, r = 0.05$, $N = 9$ and dimensions d up to 500.
- Using the dual upper bound, the 95% confidence interval in 500 dimensions is [136.521 , 137.064].
- This paper has theoretical guarantees and studies several other examples as well.

Deep hedging, by Buehler, H., Gonon, L., Teichmann, J. and Wood, B., published in *Quantitative Finance*, 19/8, 1271–1291, 2019.

While the paper does not contain guarantees, it reports numerous successful, numerical results in high dimensions. An example is the indifference price of a European Call option with 30 time points, with transaction costs and Heston dynamics in dimension 10.

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

Deep neural networks algorithms for stochastic control problems on finite horizon, part 2 : Numerical applications, by the same group, 2018.

They study the control problem related to a semi-linear PDE. . Numerical studies in dimensions 1, 10, 100 with $N = 20$ and a training data size of $10,000$. In these problems exact solution is known and they report very small error.

A convergence analysis is given.

There are many other recent papers using the recent computational techniques. Here is a partial list.

- *Statistical learning for probability-constrained stochastic optimal control*, by Balata, Ludkovski, Maheshwari & Palczewski.
- *Hedging with Neural Networks*, by Ruf & Wang.
- *Asset Pricing with General Transaction Costs : Theory and Numerics*, by Gonon, Muhle-Karbe & Shi.
- *Deep learning for discrete-time hedging in incomplete markets*, by Fecamp, Mikael & Warin.
- *Machine learning for semi linear PDEs*, by Chan-Wai-Nam, Mikael & Warin.
- *Deep backward multistep schemes for nonlinear PDEs and approximation error analysis*, by Germain, Pham & Warin.
- *Learning a functional control for high-frequency finance*, by Leal, Laurière & Lehalle.

- There is a flurry of activity for high dimensional problems.
- We seem to have efficient algorithms for high dimensional control problems.
- We need better guarantees and not only convergence results.
- The problems studied up to now, are build on specific models and the training data is simulated.
- If the available data is large, then one can have a completely data driven approach.
- Neural networks approximate the optimal control directly.
- In all studies, a separate network is employed for each time point.

These are exciting developments bringing interesting and highly applicable questions.

Deep Monte Carlo Optimization

Outline

We consider the following classical problem

$$\text{minimize} \quad a \to v(a) := \mathbb{E}_\nu \left[\sum_{t \in \mathcal{T}} \psi(t, X_t^a, a(t, X_t^a, Z_t)) + \varphi(X_T^a) \right],$$

where a is a feedback control or action, controlled state X^a is given by

$$X_{t+1}^a = f(t, X_t^a, Z_{t+1}, a(t, X_t^a, Z_t)), \quad t \in \mathcal{T} := \{0, 1, \ldots, T-1\}.$$

- $Z = (Z_1, Z_2, \ldots, Z_T)$ is an exogenous, observable random process with $Z_0 = 0$.
- ν is the distribution of Z is ν; it does not have to be Markov.
- The state process is a deterministic function of the random trajectory, i.e., $X_t^a = x_t^a(Z)$ for some deterministic function x^a.

Summarizing : the pay-off associated feedback control or action $a \in \mathcal{C}$ is given by,

$$v(a) := \mathbb{E}_\nu \left[\ell(a, Z)\right] = \mathbb{E}_\nu \left[\sum_{t \in \mathcal{T}} \psi(t, X_t^a, a(t, X_t^a, Z_t)) + \varphi(X_T^a)\right], \quad \text{where}$$
$$\ell(a, z) := \sum_{t \in \mathcal{T}} \psi(t, x_t^a(z), a(t, x_t^a(z), z_t)) + \varphi(x_T^a(z)).$$

and the deterministic function $x_t^a(z)$ is defined previously using the dynamics of the problem.

The *dynamic stochastic decision problem* is to minimize $v(a)$ over all admissible controls $a \in \mathcal{C}$.

The *optimal value* is given by,

$$v^* := \inf_{a \in \mathcal{C}} v(a).$$

Outline

- The *training set* is a collection of n observations of trajectories :

$$\mathcal{L}_n = \left\{ Z^{(1)}, Z^{(2)}, \ldots, Z^{(n)} \right\} \quad \text{where} \quad Z^{(i)} = (Z_1^{(i)}, Z_2^{(i)}, \ldots, Z_T^{(i)}).$$

- The *loss function* is simply the empirical average,

$$L(a; \mathcal{L}_n) := \frac{1}{n} \sum_{i=1}^{n} \ell(a, Z^{(i)}) \approx v(a) = \mathbb{E}_\nu \left[\ell(a, Z) \right].$$

- The set of neural networks is given abstractly by

$$\mathcal{N}_k := \left\{ \Phi(\cdot; \theta) \ : \ \theta \in \mathcal{O}_k \right\}$$

where for each parameter $\theta \in \mathcal{O}_k$, a neural network is a feedback control, i.e., a continuous function

$$\Phi(\cdot; \theta) : \mathcal{T} \times \mathcal{X} \times \mathcal{Z} \to \mathcal{A};$$

- The *compact* parameter sets $\mathcal{O}_k \subset \mathcal{R}^{d(k)}$ have increasing dimensions $d(k)$, and we assume that the sequence $\{\mathcal{N}_k\}_{k=1,2,\ldots}$ have the approximation capability.

We fix the training set $\mathcal{L}_n$ and a set of neural networks $\mathcal{N}_k$, and

$$\text{minimize} \quad \theta \in \mathcal{O}_k \;\mapsto\; L(\Phi(\cdot;\theta);\mathcal{L}_n).$$

As L is continuous and $\mathcal{O}_k$ is compact, there exists a minimizer $\theta^*_{k,n} \in \mathcal{O}^*_k$. Then,

$$\Phi^*_{k,n}(t,x,z) := \Phi(t,x,z;\theta^*_{k,n}), \qquad t \in \mathcal{T}, x \in \mathcal{X}, z \in \mathcal{Z},$$

is the *optimal feedback action* that could be constructed by the neural network $\mathcal{N}_k$ using $\mathcal{L}_n$.

Results

- a^* is the optimal control ;
- $v^* := v(a^*) = \inf_{a \in \mathcal{C}} v(a)$;
- $\mathcal{N}_k = \{\Phi(\cdot;\theta)\}_{\theta \in \mathcal{O}_k}$ is the neural networks ;
- $\mathcal{L}_n$ is the training set ;
- $L(\Phi(\cdot;\theta);\mathcal{L}_n)$ is the loss function ;
- $\Phi^*_{k,n} = argmin_{\Phi \in \mathcal{N}_k} L(\Phi;\mathcal{L}_n)$;
- $L(\Phi^*_{k,n};\mathcal{L}_n) = \inf_{\Phi \in \mathcal{N}_k} L(\Phi;\mathcal{L}_n)$;
- $v^*_{k,n} := v(\Phi^*_{k,n})$;
- $v^*_k := \min_{\Phi \in \mathcal{N}_k} v(\Phi)$;
- $\hat{\mathcal{L}}_n$ is the data set chosen identically and independently of $\mathcal{L}_n$.

We have proved :

- Convergence :

$$v^*_k = \lim_{n \to \infty} v^*_{k,n} = \lim_{n \to \infty} L(\Phi^*_{k,n};\mathcal{L}_n).$$

$$v^* = \lim_{k \to \infty} v^*_k = \lim_{k \to \infty} \lim_{n \to \infty} L(\Phi^*_{k,n};\mathcal{L}_n).$$

- Over-Learning : "generically",

$$\lim_{n \to \infty} \lim_{k \to \infty} L(\Phi^*_{k,n};\mathcal{L}_n) < v^*.$$

- non-asymptotic estimates : with $1-\delta$ probability

$$\sup_{\Phi \in \mathcal{N}_k} |L(\Phi;\mathcal{L}_n) - L(\Phi;\hat{\mathcal{L}}_n)| \leq C(\mathcal{N}_k, n, \delta),$$

where the constant $C(\mathcal{N}_k, n, \delta)$ is given by the Rademacher complexity.

Outline

- $S_t \in \mathcal{R}_+^d$ is the stock price process, interest rate is taken to be zero and the amount of money to be invested in the stock, $\pi_t = (\pi_t^1, \ldots, \pi_t^d) \in \mathcal{R}^d$, is the control variable.
- We consider feedback controls $\pi_t = a(t, X_t^a, Z_t)$ and X^a is the corresponding wealth process :

$$X_{t+1}^a = X_t + a(t, X_t^a, Z_t) \cdot Z_{t+1}, \qquad Z_{t+1} = \frac{S_{t+1} - S_t}{S_t}, \quad t \in \mathcal{T}.$$

- We maximize

$$v(a) := \mathbb{E}[1 - \exp(-X_T^a)].$$

- The certainty equivalent $\mathrm{ce}(v)$ of a utility value $v < 1$ is a more standard way of comparing different utility values :

$$\mathrm{ce}(v) := \ln(1 - v) \quad \Leftrightarrow \quad v = u(\mathrm{ce}(v)).$$

- $X_0 = 0$, $T = 2$. To simplify we fix the initial portfolio to be $\pi_0 = (1, \dots, 1)/d$;
- $X_1 = Z_1 \cdot \pi_0$ with Z_1 is uniform on $[-1/2, 1/2]^d$;
- The only control is $\pi_1 = a(Z_1) \in \mathbb{R}^d$ (as X_1 is a linear function of Z_1, we do not need it);
- For $Z_2 = \zeta\eta$ with $\eta \in \mathbb{R}^d$ is fixed, $\zeta \in \mathbb{R}$ is Gaussian with mean m and variance s, independent of Z_1,

$$X_2^a = X_1 + a(Z_1) \cdot Z_2 = X_1 + [a(Z_1) \cdot \eta]\ \zeta.$$

- problem is

$$\text{maximize } v(a) = \mathbb{E}[(1 - \exp(-X_2^a)].$$

- Explicit solutions are

$$a^* \cdot \eta = \frac{m}{s^2}, \quad v^* = 1 - \exp(-\frac{m^2}{2s^2}) \quad \Leftrightarrow \quad \mathrm{ce}(v^*) = \frac{m^2}{2s^2}.$$

- We use $m = 18\%, s = 0.44\%$ with $a^* \cdot \eta = 0.9297$.

The network tries to learn the function $Z_1 \in \mathbb{R}^d \to a^*(Z_1) \cdot \eta \in \mathbb{R}$.
We use a fully connected neural network with

- 3 hidden layers of width 10 and ReLU activation functions ($10d + 241$ parameters);
- in most cases, a training data of size of $n = 100,000$.
- stochastic gradient descent with mini-batches of size 32.
- employ commonly used conservative stopping rules such as :
  - after 4 epochs stop when the in-sample-performance increases;
  - after 4 epochs stop when the test-performance increases.
- Unless forced, the optimization is finished in 4-5 epochs in most cases.

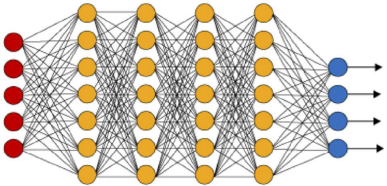

Figure 1: An illustration of a (fully connected) deep neural network. The red neurons represent the inputs to the network and the blue neurons denote the output layer. They are connected by hidden layers with yellow neurons. Each hidden unit (neuron) is connected by affine linear maps between units in different layers and then with nonlinear (scalar) activation functions within units.

Here $\Phi(\cdot;\theta) = (\sigma \circ C_4 \circ \sigma \circ C_3 \circ \sigma \circ C_2 \circ \sigma \circ C_1)(\cdot)$, where $\sigma(a) = a^+$ and C_k's are affine functions with

$$C_1 : \mathbb{R}^d \to \mathbb{R}^{10}, \quad C_2, C_3 : \mathbb{R}^{10} \to \mathbb{R}^{10}, \quad C_4 : \mathbb{R}^{10} \to \mathbb{R}^1.$$

The parameters are the coefficients of the affine functions.

- $p_{in} := \frac{\text{nn}_{\text{in-sample}} - \text{true}_{\text{in-sample}}}{\text{true}_{\text{in-sample}}}$,
- $p_{out} := \frac{\text{nn}_{\text{out-of-sample}} - \text{true}_{\text{out-of-sample}}}{\text{true}_{\text{out-of-sample}}}$,
- $\text{nn}_{\text{in-sample}} := \text{ce}(L(\Phi_{k,n}; \mathcal{L}_n))$,
- $\text{true}_{\text{in-sample}} := \text{ce}(L(a^*; \mathcal{L}_n))$,
- $\text{nn}_{\text{out-of-sample}} := \text{ce}(L(\Phi_{k,n}; \hat{\mathcal{L}}_n))$,
- $\text{true}_{\text{out-of-sample}} := \text{ce}(L(a^*; \hat{\mathcal{L}}_n))$,
- a^* is the known optimal control,
- $\Phi_{k,n}$ is the trained network, (note $\neq \Phi^*_{k,n}$),
- $\mathcal{L}_n$ is the training set used to compute $\Phi_{k,n}$,
- $\hat{\mathcal{L}}_n$ is the independent test date set.

d	p_{in} (%)	$p_{in} - p_{out}$ (%)
100	10.12820	23.67080
85	8.38061	20.16440
70	7.32720	15.62060
55	5.05783	10.93950
40	3.74648	7.91105
25	2.11501	4.58954
10	0.53982	1.46138

Table 1 – Average relative in-sample performance, and its comparison to the out-of-sample performance with the above described conservative stopping rule. Everything is in % with training size of $n = 100,000$ and three hidden layers of width 10.

- We have proved that "generically"

$$\lim_{n\to\infty} \lim_{k\to\infty} L(\Phi^*_{k,n}; \mathcal{L}_n) < v^*.$$

  And the numerical results also show this.
- The trained feedback controls perform better than the optimal : hence it is not "adapted" on the training set.
- Although the returns Z_1, Z_2 are independent, networks *try to learn* Z_2 as a function of Z_1.
- Learning becomes easier in higher dimensions.
- This is the classical bias-variance trade-off in this context.

Keeping all other parameters fixed : (3 hidden layers of width 10, $d = 100$, $n = 10^5$), we ran the stochastic gradient descent algorithm for a long time.

This resulted in very substantial over-learning.

epochs	p_{in} (%)		$p_{in} - p_{out}$ (%)	
	μ	σ	μ	σ
200	30.3161	2.46850	315.875	540.4750
100	25.8374	1.72027	111.553	41.5841

Table 2 – Longer iterations performance in 100 dimensions. Based on 15 runs. Especially the 200 epoch runs show signs of a heavy tail, as expected with high degrees of overlearning. All other parameters as in Table 1.

sample size	dimension	p_{in} (%)		$p_{in} - p_{out}$ (%)	
		μ	σ	μ	σ
2,000,000	100	0.49597	0.11849	1.12846	0.32184
1,000,000	100	1.14094	0.14640	2.39532	0.27415
500,000	100	2.36352	0.20154	5.24018	0.81235
250,000	100	4.41388	0.37928	10.02040	1.45355

Table 3 – Performance for larger sample sizes. Based on 15 runs. All other parameters as before.

This illustrates the convergence as the training data gets larger.

Outline

- Time discretization error : well studied but might be a computational difficulty. Here we simply take the problem to be already discretized.
- Law-of-Large-Numbers type error : this is the discrepancy between

$$L(a; \mathcal{L}_n) = \frac{1}{n} \sum_{i=1}^{n} \ell(a; Z^{(i)}) \qquad \text{and} \qquad v(a) = \mathbb{E}[\ell(a, Z)].$$

  We use (Radamacher type) complexity estimates to quantify it.
- Network approximation error : this is how well the finite neural network $\mathcal{N}_k$ could approximate a^* or other near-minimizers. This is a better understood analytical question.
- Optimization error : this is how well the stochastic gradient descent (or the optimization algorithm used) is doing. Extremely important and not studied here. Also it is not clear whether we should run the optimization to its end.

- Set $v_k^* := \inf_{\theta \in \mathcal{O}_k} v(\Phi(\cdot;\theta))$. As the neural networks have approximation capability,

$$\lim_{k\to\infty} v_k^* = v^* = \inf_{a\in\mathcal{C}} v(a).$$

- The following holds with at least $1-\delta$ probability for every $\theta \in \mathcal{O}_k$,

$$\begin{aligned}\left|L(\Phi(\cdot;\theta);\mathcal{L}_n) - L(\Phi(\cdot;\theta);\hat{\mathcal{L}}_n))\right| \leq& |v(\Phi(\cdot,\theta)) - L(\Phi(\cdot;\theta);\mathcal{L}_n))| + \left| v(\Phi(\cdot,\theta)) - L(\Phi(\cdot;\theta);\hat{\mathcal{L}}_n)\right| \\ \leq& 2c_\nu(\mathcal{N}_k, n, \delta/2),\end{aligned}$$

  where $c_\nu(\mathcal{N}_k, n, \delta/2)$ (it will be discussed in the next slide) is related to the Rademacher complexity $r_\nu(\ell(\mathcal{N}_k), n)$ which converges to zero as n tends to zero.

- Then, the following holds with probability one,

$$\lim_{n\to\infty} L(A_{k,n}^*;\mathcal{L}_n) = \lim_{n\to\infty} L(A_{k,n}^*;\hat{\mathcal{L}}_n) = \lim_{n\to\infty} v(A_{k,n}^*) = v_k^* := \inf_{\theta\in\mathcal{O}_k} v(\Phi(\cdot;\theta)).$$

- $\mathcal{G}$ is a hypothesis class of a set of real-valued functions defined on the set of trajectories.
- The *empirical Rademacher complexity* of $\mathcal{G}$ on the training set $\mathcal{L}_n$ is

$$R_e(\mathcal{G}, \mathcal{L}_n) := \mathbb{E}\left[\sup_{g \in \mathcal{G}} \frac{1}{n} \sum_{i=1}^{n} \sigma_i g(Z^{(i)})\right],$$

  where the expectation is over the Rademacher variables σ_i's, which are identically and independently distributed taking values ± 1 with equal probability.
- For a probability measure ν on $\mathcal{Z}^T$, the Rademacher complexity of $\mathcal{G}$ for the distribution ν is

$$r_\nu(\mathcal{G}, n) := \mathbb{E}_\nu\left[R_e(\mathcal{G}, \mathcal{L}_n)\right],$$

  where the expectation is over the random training set $\mathcal{L}_n = \{Z^{(1)}, Z^{(2)}, \ldots, Z^{(n)}\}$ whose elements are independently and identically drawn from ν.

- Recall that $v(a) := \mathbb{E}[\ell(a, Z)]$.
- For a neural network $\mathcal{N}_k$ and a training set $\mathcal{L}_n$, set

$$G_\nu(\mathcal{N}_k, \mathcal{L}_n) := \sup_{\theta \in \mathcal{O}_k} |v(\Phi(\cdot;\theta)) - L(\Phi(\cdot;\theta); \mathcal{L}_n)|.$$

- $\mathcal{L}_n$ is drawn from ν independently and identically.
- We assume that $|\ell(a, \cdot)| \leq c^*$ for every $a \in \mathcal{C}$.
- Then, for a given $\delta \in (0, 1)$, with probability at least $1 - \delta$ the following estimates hold,

$$\boxed{G_\nu(\mathcal{N}_k, \mathcal{L}_n) \leq c_\nu(\mathcal{N}_k, n, \delta) \leq C_e(\mathcal{N}_k, \mathcal{L}_n, \delta),}$$

where with $\ell(\mathcal{N}_k) = \{\ell(\Phi(\cdot;\theta)) | \theta \in \mathcal{O}_k\}$,

$$c_\nu(\mathcal{N}_k, n, \delta) := 2r_\nu(\ell(\mathcal{N}_k); n) + 2c^* \sqrt{\frac{\ln(2/\delta)}{2n}},$$

$$C_e(\mathcal{N}_k, \mathcal{L}_n, \delta) := 2R_e(\ell(\mathcal{N}_k); \mathcal{L}_n) + 6c^* \sqrt{\frac{\ln(2/\delta)}{n}}.$$

- Deep Monte-Carlo optimization is an effective and a flexible tool.
- It can handle details of the markets as well as complicated and general dynamics with ease.
- One has to be careful with the complexity of the networks and the size of the training data.
- Optimization step is the least understood part.

THANK YOU FOR YOUR ATTENTION.

joint work with

Max Reppen of Boston University

Bias-Variance Trade-off and Overlearning in Dynamic Decision Problems

https ://arxiv.org/abs/2011.09349

- Deep Monte-Carlo optimization is an effective and a flexible tool.
- It can handle details of the markets as well as complicated and general dynamics with ease.
- One has to be careful with the complexity of the networks and the size of the training data.
- Optimization step is the least understood part.

THANK YOU FOR YOUR ATTENTION.

joint work with

Max Reppen of Boston University

Bias-Variance Trade-off and Overlearning in Dynamic Decision Problems

https ://arxiv.org/abs/2011.09349