Deep Optimal Stopping

Vanessa Pizante

University of Toronto

March 26, 2019

Optimal Stopping Problem

- $X = (X_n)_{n=0}^N$ is an Markov process in $\mathbb{R}^d$ on $(\Omega, \mathcal{F}, (\mathcal{F})_{n=0}^N, \mathbb{P})$.
- $\mathcal{T}$ is the set of all X-stopping times.
  - Recall: τ is an X-stopping time if $\{\tau = n\} \in \mathcal{F}_n, \ \forall n \in \{0, 1, ..., N\}$
- $g : \{0, 1, ..., N\} \times \mathbb{R}^d \to \mathbb{R}$ is a measurable, integrable function.

Objective

Find $V = \sup_{\tau \in \mathcal{T}} \mathbb{E} g(\tau, X_\tau)$.

Introduction

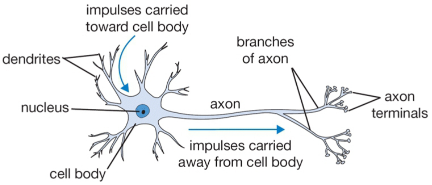
Primary Ressource

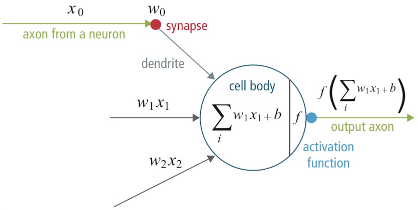
Becker, S., Cheridito, P., and Jentzen, A.: *Deep Optimal Stopping*. (2018).

- Optimal stopping problems with finitely many stopping times can be solved exactly.
  - Optimal V given by Snell Envelope.
- Difficult to approximate numerically in higher dimensions.
- [Becker et al., 2018] proposes decomposing τ into a sequence of $N+1$ 0-1 stopping decisions.
- Each decision is learned via a deep feedforward neural network.

What is a Neural Network?

- A neural network is a computational system modelled loosely after the human brain: a neuron receives input from other neurons or from an external source which it uses to generate output.
- Each input has a weight quantifying its relative significance.
- You can think of a neuron's output as its relative firing rate.

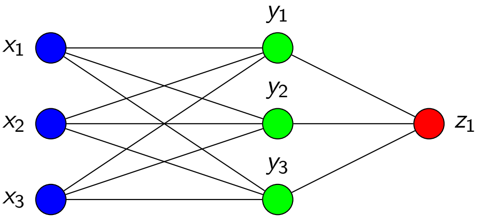




Source: [Hijazi et al., 2015]

Feedforward Neural Network

- **Input Layer**: $x = (x_1, x_2, x_3)$.
- **Hidden Layer**: Takes x, and outputs $y = (y_1, y_2, y_3)$ where $y_i = f_y(w_i x + b_i)$ and $w_i = (w_{i1}, w_{i2}, w_{i3})$ for $i = 1, 2, 3$.
- **Output Layer**: Take y and outputs $z_1 = f_z(w_z y + b_z)$ where $w_z = (w_{z1}, w_{z2}, w_{z3})$



- f_y and f_z are called **activation functions**.

Training a Neural Network

- The **training data** can consist of input and target pairs $(x^m, z_1^m)_{m=1}^M$ or just inputs $(x^m)_{m=1}^M$.

- **Training** the network refers to setting the **weight** w_i and **bias** b_i terms using the training data.
    - $\theta = \{w_1, w_2, w_3, w_z, b_1, b_2, b_3, b_z\}$ is called the model **parameters**.

- Training is done by minimizing a **loss function**, C, w.r.t θ.
    - e.g. mean squared-error loss function

$$C(\theta) = \frac{1}{2M} \sum_{m=1}^{M} \|z_1^\theta(x^m) - z^m\|^2$$

Training a Neural Network

- The optimal parameters are found by minimizing the loss function via a gradient descent algorithm.
  - e.g. Vanilla gradient descent with learning rate η update step:

$$\theta_{t+1} \leftarrow \theta^t - \eta \cdot \nabla_\theta C(\theta^t)$$

- $\nabla_\theta C(\theta^t)$ is computed using **backpropagation**.
  - Method for efficiently moving backwards through the neural network graph to compute each $\frac{\partial C}{\partial \theta_i}$ via the chain rule.
  - Special case of reverse-mode autodifferentiation.

- After training, we run the algorithm using a new dataset, this is the **testing data**.

- Note: Minimizing a loss function via gradient descent is equivalent to maximizing a reward function via gradient ascent.

Reinforcement Learning (Optional)

Problem set-up:

- Agent receives reward $r_t = r(s_t, a_t)$ depending on its actions a_t and the state s_t.
- Environment is a Markov Decison Process with *unknown* transition probabilites $p(s_{t+1}|s_t, a_t)$.

Reinforce:

- Agents aim to learn a policy $\pi_\theta(s_t, a_t)$ so they interact with their environment to maximize some cummulative reward function.
- Samples different rollouts $\rho = (s_1, a_1, ..., s_T, a_T)$ to obtain a policy where rollouts corresponding to a high reward are more likely to be chosen.

- *Model-free* approach to policy iteration.

Q-Learning (Optional)

- Q-function (a.k.a. action-value function): Expected rewards (discounted by factor γ) if you take action a then follow your policy:

$$Q^\pi(s,a) = \mathbb{E}\left[\sum_{i=0}^{\infty} \gamma_i r_{t+i} \middle| s_t = s, a_t = a\right]$$

- Q-learning is an iterative algorithm based on the recursive formula for Q derived via Bellman's equation:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r(s_t, a_t) + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t))$$

- Note $r(s_t, a_t) + \gamma \max_a Q(s_t, a) - Q(s_t, a_t)$ is the Bellman error $\implies$ Q-Learning is equivalent to minimizing the Bellman error.

Optimal Stopping and Q-Learning (Optional)

Yu, H. and Bertsekas, D.P.: *Q-learning Algorithms for Optimal Stopping Based on Least Squares.* (2007).

- Given current state x_t we have two options:
  - Stop and incur *cost* $f(x_t)$.
  - Continue and incur *cost* $h(x_t, x_{t+1})$.
- Let $Q^*(x_t) = \min_a Q(x_t, a)$ where a is either to stop or continue.
- Optimal policy is to stop as soon as (X_t) enters the set: $\mathcal{D} = \{x_t | f(x_t) \leq Q^*(x_t)\}$

Optimal Stopping and Q-Learning (Optional)

- Consider approximations for Q that use a linear projection of transformed x_t [Yu and Bertsekas, 2007]:

$$Q^*(x_t) \equiv \phi(X_t)'\beta_t$$

- The corresponding projected Bellman error is then minimized w.r.t. β_t via a least-squares approximation based on the Q-learning algorithm as follows:

$$\beta_{t+1} = \beta_t + \alpha(\hat{\beta}_{t+1} - \beta_t)$$

where

$$\hat{\beta}_t = \arg\min_{\beta} \sum_{k=0}^{t} \left(\phi(x_k)'\beta - h(x_k, x_{k+1}) - \gamma \min\{f(x_{k+1}), \phi(x_k)'\beta_t\}\right)^2$$

Least-Squares Monte Carlo (LSMC)

Longstaff, F. and Schwartz E.: *Valuing American Options by Simulation: A Simple Least-Squares Approach*, (2001).

- Simulate M stock price paths $(x_n^m)_{n=0}^N$.
- Set $V_N^m = g(N, x_N^m), \forall m$ and then for $n = N-1, ..., 0$:
  - Approximate continuation value q_n via $\hat{q}_n(x) = \sum_{k=0}^K \beta_k x^k$ where β minimizes the least-squares error

$$\sum_{m=1}^{M} (e^{-rt_n} V_{n+1}^m - q_n(x_n^m))^2$$

  - Set $V_n^m = \begin{cases} g(n, x_n^m) & \text{if } q_n(x_n^m) < g(n, x_n^m) \\ e^{-rn\frac{T}{N}} V_{n+1}^m & \text{otherwise} \end{cases}$

f

Least-Squares Neural Network Approach

Kohler, Krzyzak and Todorovic: *Pricing of High-Dimensional American Options by Neural Networks*, (2006).

- Neural network applied to Longstaff-Schwartz algorithm to price higher dimensional American options.
- Replaces least-squares regression with neural network of the form:

$$q^{\theta_n} = a_2^{\theta_n} \circ \sigma \circ a_1^{\theta_n}$$

  where

  - $a_1 : \mathbb{R}^d \to \mathbb{R}^K$, $a_2 : \mathbb{R}^K \to \mathbb{R}$: $a_i(x) = W_i x + b_i$ and $\sum_{k=0}^{K} |w_{2,k}| \leq C_n$.
  - $\sigma : \mathbb{R}^k \to [0,1]^K$: $\sigma_k(y_k) = 1/(1 + e^{-y_k})$.

- Algorithm backwards recursively creates training data $(x_n^m, e^{-rt_n} \cdot V_{n+1}^m)$ and has a mean squared loss function.

Extension: More Complex Neural Networks

Hu, R.: *Deep Learning for Ranking Response Surfaces with Applications to Optimal Stopping Problems*, (2019).

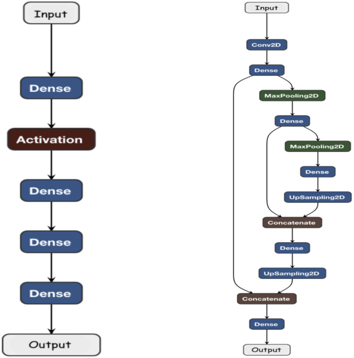


Figure 1: Left: Feed-forward NN, similar to NN applied in [Becker et al., 2018], Right: UNET, applied in [Hu, 2019].

Source: [Hu, 2019]

Ranking Response Surfaces

- Optimal stopping problem can be reframed as an image classification problem via surface ranking.
- Surface ranking problem: Extract index of minimal surface for each input x and treat it as a class label.
  - For optimal stopping labels are just stop or continue (at each time step).
- Can now **use convolution NNs** (e.g. UNET), which can be more computationally efficient [Hu, 2019].
- *Disadvantage*: Convergence theory has yet to have been completely developed for fully convolutional NN.

Expressing Stopping Times as a Series of 0-1 Decisions

Goal

Show optimal decision to stop $(X_n)_{n=0}^N$ can be made according to $\{f_n(X_n)\}_{n=0}^N$ where $f_n : \mathbb{R}^d \to \{0, 1\}$.

- Write time n stopping time, τ_n, as a function of $\{f_k(X_k)\}_{k=n}^N$.
- Let $\mathcal{T}_n$ be the set of all X-stopping times τ s.t. $n \leq \tau \leq N$.
- Clearly $\mathcal{T}_N = \{N\}$, so let $\tau_N = N \cdot f_N(X_N)$ where $f_N \equiv 1$.
- Now, for $n = N - 1, ..., 0$, we can define:

$$\tau_n = \sum_{k=n}^{N} k f_k(X_k) \prod_{j=n}^{k-1} (1 - f_j(X_j)) \in \mathcal{T}_n \tag{1}$$

Theorem 1

For $n \in \{0, ..., N-1\}$, let $\tau_{n+1} \in \mathcal{T}_{n+1}$ be of the form:

$$\tau_{n+1} = \sum_{k=n+1}^{N} k f_k(X_k) \prod_{j=n+1}^{k-1} (1 - f_j(X_j)) \tag{2}$$

Then there exists a measurable function $f_n : \mathbb{R}^d \to \{0, 1\}$ such that $\tau_n \in \mathcal{T}_n$ satisfies

$$\mathbb{E} g(\tau_n, X_{\tau_N}) \geq V_n - (V_{n+1} - \mathbb{E} g(\tau_{n+1}, X_{\tau_{n+1}})) \tag{3}$$

where V_n and V_{n+1} satisfy

$$V_n = \sup_{\tau \in \mathcal{T}_n} \mathbb{E} g(\tau, X_\tau) \tag{4}$$

Sketch of Theorem 1 Proof

- Fix stopping time $\tau \in \mathcal{T}_n$ and let $\epsilon = V_{n+1} - \mathbb{E}g(\tau_{n+1}, X_{\tau_{n+1}})$.
- Show $g(\tau_{n+1}, X_{\tau_{n+1}})$ is meas. using (2).
  $\implies \exists h_n$ meas. and Markovian s.t. $h_n(X_n) = \mathbb{E}[g(\tau_{n+1}, X_{\tau_{n+1}})|X_n]$.
- Define $D = \{g(n, X_n) \geq h_n(X_n)\}$, $E = \{\tau = n\} \in \mathcal{F}_n$ and:
  - $\tau_n = nI_D + \tau_{n+1} I_{D^c} \ \in \mathcal{T}_n$
  - $\tilde{\tau} = \tau_{n+1} I_E + \tau I_{E^c} \ \in \mathcal{T}_{n+1}$
- So $\mathbb{E}g(\tau, X_{\tau_{n+1}}) \geq \mathbb{E}g(\tilde{\tau}, X_{\tilde{\tau}}) - \epsilon \implies \mathbb{E}[g(\tau, X_{\tau_{n+1}}) I_{E^c}] \geq \mathbb{E}[g(\tau, X_\tau) I_{E^c}] - \epsilon$.
- *Exercise:* Show $\mathbb{E}g(\tau_n, X_{\tau_n}) \geq \mathbb{E}g(\tau, X_\tau) - \epsilon$.
- Since τ is arbitrary, we have $\mathbb{E}g(\tau_n, X_{\tau_n}) \geq V_n - \epsilon$, satisfying (3).
- Proof is completed by showing τ_n here satisfies (1).

What have we proven?

- Theorem 1 shows that τ_n from (1) can adequately be used to compute V_n.
  - Meaning τ_n is the time n optimal stopping time.
  - Follows from (3) due to the backwards recursive way τ_n is computed.

- Optimal stopping time corresponding to $V = \sup_{\tau \in \mathcal{T}} \mathbb{E} g(\tau, X_\tau)$ is:

$$\tau = \sum_{n=1}^{N} n f_n(X_n) \prod_{k=0}^{n-1} (1 - f_k(X_k))$$

- ...But how do we find the sequence $\{f_n\}_{n=0}^{N}$?

Introducing a Neural Network

- Using $f_N \equiv 1$ we construct a sequence of neural networks, $f^{\theta_n} : \mathbb{R}^d \to \{0, 1\}$, to approximate f_n for $n = N - 1, ..., 0$.
- We can then approximate τ_{n+1} via

$$\sum_{k=n}^{N} k \cdot f^{\theta_k}(X_k) \prod_{j=n}^{k-1} (1 - f^{\theta_j}(X_j)) \tag{5}$$

Problem:

- f^{θ_n} produces binary output $\implies$ it is not continuous w.r.t. θ.
- Need continuous output to train θ_n via a gradient-based optimization algorithm.

Neural Network with Continuous Output

Introduce $F^\theta : \mathbb{R}^d \to (0,1)$, a two-layer, feed-forward neural network of the form

$$F^\theta = \psi \circ a_3^\theta \circ \phi_{q_2} \circ a_2^\theta \circ \phi_{q_1} \circ a_1^\theta \tag{6}$$

where

- q_1 and q_2 are the number of nodes in the hidden layers.
- $a_1^\theta : \mathbb{R}^d \to \mathbb{R}^{q_1}$, $a_2^\theta : \mathbb{R}^{q_1} \to \mathbb{R}^{q_2}$, $a_3^\theta : \mathbb{R}^{q_2} \to \mathbb{R}$ satisfy

$$a_i^\theta(x) = W_i x + b_i$$

- $\phi_{q_i} : \mathbb{R}^{q_i} \to \mathbb{R}^{q_i}$ are ReLU activation function, $\phi_{q_i}(x_1, ..., x_{q_i}) = (x_1^+, ..., x_{q_i}^+)$.
- $\psi : \mathbb{R} \to \mathbb{R}$ is the logistic sigmoid function, $\psi(x) = 1/(1 + e^{-x})$.

Return to Binary Decisions

- Parameters are $\theta = \{(A_i, b_i)_{i=1}^3\} \in \mathbb{R}^q$, where $q = q_1(d + q_2 + 1) + 2q_2 + 1$.
- Can now use F^θ to find optimal θ_n via gradient descent for $n = N - 1, ..., 0$.
- After, we can compute $f^{\theta_n} : \mathbb{R}^d \to \{0, 1\}$ using:

$$f^{\theta_n} = I_{[0,\infty)} \circ a_3^{\theta_n} \circ \phi_{q_2} \circ a_2^{\theta_n} \circ \phi_{q_1} \circ a_1^{\theta_n} \tag{7}$$

- Note [Becker et al., 2018] does not provide formal proof that using F^θ to optimize θ will provide optimal parameters for f^θ.
- However, it does make sense intuitively when we consider $F^{\theta_n}(X_n)$ to be the probability that stopping is the optimal decision at time n (given X_n).

Selecting a Reward Function

Main Idea

Want to define a reward function that yields a stopping decision at time n that will maximize our expected future payoff.

If at time n we:

- Stop $\implies$ $f_n(X_n) = 1$ and we will receive payoff $g(n, X_n)$
- Continue and after proceed optimally $\implies$ $f_n(X_n) = 0$ and we will eventually receive payoff of $g(\tau_{n+1}, X_{\tau_{n+1}})$.

So we want our reward function at time n to approximate

$$\sup_{f \in \mathcal{D}} \mathbb{E}[g(n, X_n) f(X_n) + g(\tau_{n+1}, X_{\tau_{n+1}})(1 - f(X_n))] \quad (8)$$

where $\mathcal{D}$ is the set of all $f : \mathbb{R}^d \to \{0, 1\}$ measurable.

- ...But can we even replace f in (8) with neural network f^θ?

Theorem 2

Let $n \in \{0, ..., N-1\}$ and fix a stopping time $\theta_{n+1} \in \mathcal{T}_{n+1}$, then $\forall \epsilon > 0$, there exists $q_1, q_2 \in \mathbb{N}^+$ such that

$$\begin{aligned} &\sup_{\theta \in \mathbb{R}^q} \mathbb{E}[g(n, X_n) f^\theta(X_n) + g(\tau_{n+1}, X_{\tau_{n+1}})(1 - f^\theta(X_n))] \\ \geq &\sup_{f \in \mathcal{D}} \mathbb{E}[g(n, X_n) f(X_n) + g(\tau_{n+1}, X_{\tau_{n+1}})(1 - f(X_n))] - \epsilon \end{aligned} \tag{9}$$

where $\mathcal{D}$ is the set of all measurable functions $f : \mathbb{R}^d \to \{0, 1\}$.

Corrallary 1

For any $\epsilon > 0$, there exists $q_1, q_2 \in \mathbb{N}^+$ and neural network functions of the form (7) such that $f^{\theta_N} \equiv 1$ and the stopping time

$$\hat{\tau} = \sum_{n=1}^{N} n f^{\theta_n}(X_n) \prod_{k=0}^{n-1} (1 - f^{\theta_k}(X_k)) \tag{10}$$

satisfies $\mathbb{E} g(\hat{\tau}, X_{\hat{\tau}}) \geq \sup_{\tau \in \mathcal{T}} \mathbb{E} g(\tau, X_\tau) - \epsilon$.

- This means we can approximate the sequence of optimal stopping decisions $\{f_n\}_{n=0}^N$ with the sequence of optimized neural networks $\{f^{\theta_n}\}_{n=0}^N$.

Sketch of Theorem 2 Proof (Optional)

- By integrability of g, $\exists \tilde{f} : \mathbb{R}^d \to \{0,1\}$ meas. s.t.

$$\begin{aligned} &\mathbb{E}[g(n, X_n)\tilde{f}^\theta(X_n) + g(\tau_{n+1}, X_{\tau_{n+1}})(1 - \tilde{f}^\theta(X_n))] \\ \geq\ &\sup_{f \in \mathcal{D}} \mathbb{E}[g(n, X_n)f(X_n) + g(\tau_{n+1}, X_{\tau_{n+1}})(1 - f(X_n))] - \epsilon/4 \end{aligned} \tag{11}$$

- Let $\tilde{f} = I_A$, $A = \{x \in \mathbb{R}^d | \tilde{f} = 1\}$ and note

$$B \mapsto \mathbb{E}[|g(n, X_n)| I_B(X_n)] \qquad \text{and} \qquad B \mapsto \mathbb{E}[|g(\tau_{n+1}, X_{\tau_{n+1}})| I_B(X_n)]$$

  are finite Borel measures on $\mathbb{R}^d$.

- So $\exists K \subseteq A$ compact s.t.

$$\begin{aligned} &\mathbb{E}[g(n, X_n)I_K(X_n) + g(\tau_{n+1}, X_{\tau_{n+1}})(1 - I_K(X_n))] \\ \geq\ &\mathbb{E}[g(n, X_n)\tilde{f}^\theta(X_n) + g(\tau_{n+1}, X_{\tau_{n+1}})(1 - \tilde{f}^\theta(X_n))] - \epsilon/4 \end{aligned} \tag{12}$$

Sketch of Theorem 2 Proof Cont. (Optional)

- Let $\rho_K(x) = \inf_{y \in K} \|x - y\|_2$ and note $k_j(x) = \max\{1 - j \cdot \rho_K(x),\ -1\}$, $j \in \mathbb{N}$ converges pointwise to $I_K - I_{K^c}$. So by DCT:

$$\begin{aligned} &\mathbb{E}[g(n, X_n) I_{k_j(X_n) \geq 0} + g(\tau_{n+1}, X_{\tau_{n+1}})(1 - I_{k_j(X_n) \geq 0})] \\ &\geq \mathbb{E}[g(n, X_n) I_K(X_n) + g(\tau_{n+1}, X_{\tau_{n+1}})(1 - I_K(X_n))] - \epsilon/4 \end{aligned} \tag{13}$$

- By **[Leshno et al., 1993]** since k_j can be approx. uniformly on compact sets by $h(x) = \sum_{i=1}^{r} (v_i^T x + c_i)^+ - \sum_{i=1}^{s} (w_i^T x + d_i)^+$ and thus:

$$\begin{aligned} &\mathbb{E}[g(n, X_n) I_{h(X_n) \geq 0} + g(\tau_{n+1}, X_{\tau_{n+1}})(1 - I_{h(X_n) \geq 0})] \\ &\geq \mathbb{E}[g(n, X_n) I_{k_j(X_n) \geq 0} + g(\tau_{n+1}, X_{\tau_{n+1}})(1 - I_{k_j(X_n) \geq 0})] - \epsilon/4 \end{aligned} \tag{14}$$

- By considering $I_{[0,\infty)} \circ h$ as an NN of the form f^θ, eq.(14) becomes:

$$\begin{aligned} &\mathbb{E}[g(n, X_n) f^\theta(X_n) + g(\tau_{n+1}, X_{\tau_{n+1}})(1 - f^\theta(X_n))] \\ &\geq \mathbb{E}[g(n, X_n) I_{k_j(X_n) \geq 0} + g(\tau_{n+1}, X_{\tau_{n+1}})(1 - I_{k_j(X_n) \geq 0})] - \epsilon/4 \end{aligned} \tag{15}$$

- Combining (11), (12), (13) and (15) we obtain (9), as required.

Computing Estimates via the Neural Network

- Simulate M independent paths of Markov process, $(x_n^m)_{n=0}^N$.
- θ_N is chosen so $f^{\theta_N} \equiv 1$ and for $n = N-1, ..., 0$:
  - Use $\theta_{n+1}, ..., \theta_N$ to compute $\bar{\tau}_{n+1}^m$ along each of the m paths via

$$\bar{\tau}_{n+1}^m = \sum_{k=n+1}^{N} k f^{\theta_k}(x_n^m) \prod_{j=n+1}^{k-1} (1 - f^{\theta_j}(x_j^m))$$

  - At time n along path m, if we stop w.p. $F^{\theta_n}(x_n^m)$ and then adhere to $\{f^{\theta_k}(x_k^m)\}_{k=n+1}^N$, the realized reward is

$$r_n^m(\theta_n) = g(n, x_n^m) F^{\theta}(x_n^m) + g(\bar{\tau}_{n+1}^m, x_{\bar{\tau}_{n+1}^m}^m)$$

  - For sufficiently large M

$$\frac{1}{M} \sum_{m=1}^{M} r_n^m(\theta_n) \tag{16}$$

approximates $\mathbb{E}[g(n, X_n) F^{\theta_n}(X_n) + g(\tau_{n+1}, X_{\tau_{n+1}})(1 - F^{\theta_n}(X_n))]$.

Computing Estimates via the Neural Network

- Note (16) acts as the reward function we want to maximize w.r.t. θ_n via a gradient ascent algorithm.
- Next translate $F^{\theta_n}(x_n^m)$ into 0-1 stopping decisions, $f^{\theta_n}(x_n^m)$.

- Now generate our testing sample paths $(y_n^m)_{n=0}^N$ for $m = 1, ..., M$.
- For $n = N-1, ..., 0$, using the $\{\theta_n\}$ found during training, compute

$$\tilde{\tau}_{n+1}^m = \sum_{k=n+1}^{N} k f^{\theta_k}(y_k^m) \prod_{j=n+1}^{k-1} (1 - f^{\theta_j}(y_j^m))$$

along each of the m sample paths.

Computing Estimates via the Neural Network

- Overall optimal stopping time (estimate of $\hat{\tau}$ from (10)):

$$\tilde{\tau}^m = \sum_{n=1}^{N} k f^{\theta_n}(y_n^m) \prod_{k=1}^{n-1} (1 - f^{\theta_k}(y_k^m))$$

- Corresponding Monte Carlo estimate of $V = g(\hat{\tau}, X_{\hat{\tau}})$:

$$\hat{V} = \frac{1}{M} \sum_{m=1}^{M} g(\tilde{\tau}_m, y_{\tilde{\tau}_m}^m) \tag{17}$$

  - By CLT a $1 - \alpha$, $\alpha \in (0, 1)$ confidence interval for V is:

$$\left[\hat{V} - z_{\alpha/2} \frac{\hat{\sigma}}{\sqrt{M}}, \hat{V} + z_{\alpha/2} \frac{\hat{\sigma}}{\sqrt{M}}\right]$$

  where $z_{\alpha/2}$ is the $1 - \alpha/2$ quantile of $\mathcal{N}(0, 1)$ and

$$\hat{\sigma} = \frac{1}{M-1} \sum_{m=1}^{M} \left(g(\tilde{\tau}_m, y_{\tilde{\tau}_m}^m) - \hat{V} \right)^2$$

Application: Bermudan Max Call Option

Example

Bermudan max-call option expiring at time T with strike price K written on d assets, $X^1, ..., X^d$, with $N+1$ equidistant exercise times $t_n = nT/N,\ n = 0, 1, ..., N$.

- Payoff function time t is: $\left(\max_{i \in \{1,\dots,d\}} X_t^i - K\right)^+$.
- So its price at time t is $\sup_\tau \mathbb{E}\left[e^{-r\tau}\left(\max_{i \in \{1,\dots,d\}} X_\tau^i - K\right)^+\right]$.
- Frame this as an optimal stopping problem $\sup_{\tau \in \mathcal{T}} \mathbb{E} g(\tau, X_\tau)$ where

$$g(n, x) = e^{-rt_n}\left(\max_{i \in \{1,\dots,d\}} x^i - K\right)^+ \tag{18}$$

Application: Bermudan Max Call Option

- Assume a Black-Scholes market model with d uncorrelated assets.
- For $i = 1, ..., d$ set:

$$x_0^i = 90, \quad K = 100, \quad \sigma_i = 0.2, \quad \delta_i = 0.1, \quad r = 0.05, \quad T = 3, \quad N = 9$$

- Asset price paths can be simulated via

$$x_{n,i}^m = x_{0,i} \cdot \exp\left\{ \sum_{k=0}^{n} \left((r - \delta_i - \sigma_i^2/2)\Delta t + \sigma_i \sqrt{\Delta t} \cdot Z_{k,i}^m \right) \right\} \tag{19}$$

where $\Delta t = T/N$ and $Z_{k,i}^m \sim \mathcal{N}(0,1)$.

Building Neural Network using PyTorch

Constructing a neural network of the form F^{θ} from (6) is simple using `Pytorch`!

```
import torch.nn as nn

class NeuralNet(torch.nn.Module):
    def __init__(self, d, q1, q2):
        super(NeuralNet, self).__init__()
        self.a1 = nn.Linear(d, q1)
        self.relu = nn.ReLU()
        self.a2 = nn.Linear(q1, q2)
        self.a3 = nn.Linear(q2, 1)
        self.sigmoid=nn.Sigmoid()

    def forward(self, x):
        out = self.a1(x)
        out = self.relu(out)
        out = self.a2(out)
        out = self.relu(out)
        out = self.a3(out)
        out = self.sigmoid(out)

        return out
```

Training the Network in Pytorch

```
def loss(y_pred,s, x, n, tau):
    r_n=torch.zeros((s.M))
    for m in range(0,s.M):
        r_n[m]=-s.g(n,m,x)*y_pred[m] - s.g(tau[m],m,x)*(1-y_pred[m])
    return(r_n.mean())

def NN(n,x,s, tau_n_plus_1):
    epochs=50
    model=NeuralNet(s.d,s.d+40,s.d+40)
    optimizer = torch.optim.Adam(model.parameters(), lr = 0.0001)

    for epoch in range(epochs):
        F = model.forward(X[n])
        optimizer.zero_grad()
        criterion = loss(F,S,X,n,tau_n_plus_1)
        criterion.backward()
        optimizer.step()

    return F,model
```

- `s.g` is g, as defined in equation (18).

Adaptive Moment Estimation (Adam) Optimization

Input: α, $r(\theta)$, θ^0

Set $\beta_1 = 0.9; \beta_2 = 0.99; \epsilon = 10^{-8}$

$m_0 = v_0 = 0;\ t = 0$

while $r(\theta_t)$ *not minimized* **do**

- $t = t + 1$
- $G_t = \nabla_\theta r(\theta_{t-1})$
- $m_t = \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot G_t$
- $v_t = \beta_2 \cdot v_{t-1} + \beta_2 \cdot (G_t)^2$
- $\hat{m}_t = m_t / (1 - (\beta_1)^t)$
- $\hat{v}_t = v_t / (1 - (\beta_2)^t)$
- $\theta_t = \theta_{t-1} - \alpha \cdot \hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon)$

Output: θ^t

- Adaptive: Performs smaller updates (lower learning rate) for frequently occurring features.
- Moment Estimation: Stores exponentially decaying average of gradient's mean (m_t) and uncentered variance (v_t).

Results

d	Actual	Estimate	Standard Error	95 % Confidence Interval
2	8.04	7.50	0.23	(7.05, 7.95)
5	16.64	16.80	0.32	(16.18,17.43)
10	26.20	23.83	0.29	(23.25, 24.40)

Table 1: Tabulated estimates for V. Results computed using $M = 5000$ sample paths.

Note: Actual estimate for $d = 2$ is from [Hu, 2019] and for $d = 5, 10$ are from [Becker et al., 2018].

Optimally Stopping a Fractional Brownian Motion (Optional)

- Fractional Brownian motion with Hurst Parameter H is a Gaussian process with mean zero and covariance structure

$$\mathbb{E}[W_t^H W_s^H] = \frac{1}{2}\left(t^{2H} + s^{2H} - |t-s|^{2H}\right) \tag{20}$$

- Standard Brownian Motion is a fractional Brownian motion with $H = 1/2$.

Objective

We want to evaluate $\sup_{0\leq\tau\leq 1} \mathbb{E} W_\tau^H$.

Challenge of Optimally Stopping FBM (Optional)

Optional Stopping Theorem (OST)

Let $(X_t)_{t\geq 0}$ be a martingale and τ be a stopping time with respect to the filtration $(\mathcal{F}_t)_{t\geq 0}$, then if either of the following hold:

- τ is bounded
- $X_{\tau\wedge t}$ is bounded

it follows that $\mathbb{E}X\tau = \mathbb{E}X_0$.

- So by OST $\mathbb{E}W_\tau^{1/2} = 0$ for any stopping time τ s.t. $W_{\tau\wedge t}^{1/2}$ is bounded above by a constant.
- However if $H \neq 1/2$, W^H is not a martingale, so the OST does not apply.

Optimally Stopping a Fractional Brownian Motion (Optional)

- Discretize the time interval $[0, 1]$ into time steps, $t_n = \frac{n}{100}$.
- Create a 100-dimensional Markov process, $(X_n)_{n=0}^{100}$ to describe $(W_{t_n}^H)_{n=0}^{100}$:

$$\begin{aligned} X_0 &= (0, 0, ..., 0) \\ X_1 &= (W_{t_1}^H, 0, ..., 0) \\ X_2 &= (W_{t_1}^H, W_{t_2}^H, ..., 0) \\ &\vdots \\ X_{100} &= (W_{t_{100}}^H, W_{t_{99}}^H, ..., W_{t_1}^H) \end{aligned}$$

- Letting $g : \mathbb{R}^{100} \to \mathbb{R}$ be $g(x_1, ..., x_{100}) = x_1$, we now aim to compute

$$\sup_{\tau \in \mathcal{T}} \mathbb{E} g(X_\tau) \tag{21}$$

where τ is the set of all X-stopping times.

Optimally Stopping a Fractional Brownian Motion (Optional)

- [Becker et al., 2018] simulated $(X_n)_{n=0}^{100}$ by defining $Y_n^H = W_{t_n}^H - W_{t_{n-1}}^H$ for each $n = 1, ..., N$, which forms a stationary Gaussian process with autocovariance

$$\mathbb{E}[Y_n^H Y_{n+k}^H] = \frac{|k+1|^{2H} - |k|^{2H} + |k-1|^{2H}}{2(100^{2H})}$$

- Sample paths of (X_n) are then computed via $W_{t_n}^H = \sum_{k=1}^{n} Y_k^H$.

- Then train neural networks of the form (7) with $d = 100$, $q_1 = 110$ and $q_2 = 55$ and approximate (21) via Monte Carlo.

Optimally Stopping a Fractional Brownian Motion (Optional)

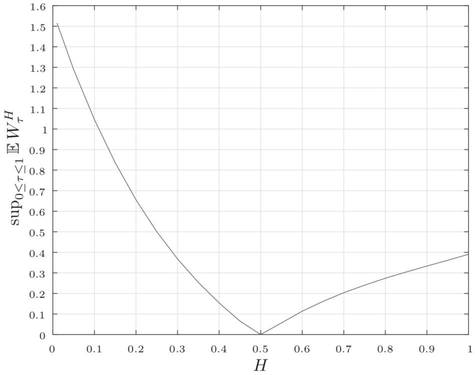


Figure 2: Results of [Becker et al., 2018]. Expected value of optimally stopped FBM w.r.t. its Hurst parameter H.

Source: [Becker et al., 2018]

Concluding Remarks

- Neural networks (and machine learning algorithms in general) can aid in solving the optimal stopping problem in higher dimensions.
- New area of research:
  - Neural Network employed by [Becker et al., 2018] has one of the simplest architectures.
  - More complex architectures lack fully developed convergence theory [Hu, 2019].

Questions for further research:

- Can we strategically select the number of nodes per hidden layer and/or the number of epochs for this algorithm?
- What other neural network architectures might be worth trying?
- What other problems in mathematical finance can be approximated in higher dimensions by applying neural networks?

References I

Becker, S., Cheridito, P., and Jentzen, A. (2018).
Deep optimal stopping.
Technical report.

Hijazi, S., Kumar, R., and Rowen, C. (2015).
Using convolutional neural networks for image recognition.

Hu, R. (2019).
Deep learning for ranking response surfaces with applications to optimal stopping problems.
arXiv e-prints, page arXiv:1901.03478.

Kingma, D. P. and Ba, J. (2014).
Adam: A method for stochastic optimization.
CoRR.

Kohler, M., Krzyżak, A., and Todorovic, N. (2010).
Pricing of high-dimensional american options by neural networks.
Mathematical Finance, 20(3):383–410.

References II

Leshno, M., Lin, V. Y., Pinkus, A., and Schocken, S. (1993).
Multilayer feedforward networks with a nonpolynomial activation function can approximate any function.
Neural Networks, 6:861–867.

Yu, H. and Bertsekas, D. P. (2007).
Q-learning algorithms for optimal stopping based on least squares.
2007 European Control Conference (ECC), pages 2368–2375.