Deep Optimal Stopping

Jackson Cornell

May 3, 2023

1 Summary

Optimal stopping is a stochastic control problem where given a d-dimensional stochastic process $\mathcal{X} = \{X_n\}_{n=0}^N$ with $X_n \in \mathbb{R}^d$, we must estimate the timestep $\tau \in \{0, ..., N\}$ that maximizes the reward up to our chosen τ. To do this, we develop an optimal policy of the form

$$V = \sup_\tau \mathbb{E}[g(\tau, X_\tau)] \tag{1}$$

where $g(\cdot, \cdot)$ is a measurable function of the Markov process and it's current index. Becker et al. [1] propose using a family of d-dimensional indicator functions $\{f_0, f_1, ..., f_N\}$ of the form $f_n(X_n) : \mathbb{R}^d \to \mathcal{U}$ that map the process to a stopping decision $\mathcal{U} = \{0, 1\}$ to estimate equation 1 at each timestep n. The optimal stopping time is then found by putting the indicator functions into the form

$$\tau_n = \sum_{m=n}^{N} m f_m(X_m) \prod_{j=n}^{m-1} (1 - f_j(X_j)) \tag{2}$$

where n is the timestep from which we start, and $f_N \equiv 1$. A family of deep neural networks (DNN) $\{f_0^\theta, f_1^\theta, ..., f_N^\theta\}$ that map from the input process to the interval $[0, 1]$ are then trained to maximize the reward, given below:

$$r_n^m(\theta) = g(n, x_n^m) f^\theta(x_n^m) + g(\tau_{n+1}, x_{\tau_{n+1}}^m)(1 - f^\theta(x_n^m)) \tag{3}$$

where $m \in \{1, ..., M\}$ is the simulation paths for a large integer M. The deep learning framework, compared to other methods of optimal stopping, has the benefit of being applicable to very high dimensional problems, a major bottleneck in many stochastic control problems. In this paper, the approach is applied to the problem of Bermudan Max-Call Options (BMCO) where a trader must decide when to sell their d assets to maximize profit. A Monte Carlo simulation is performed and the results of Becker et al. [1] are repeated. We additionally discuss using deep learning to solve optimal stopping problems in conjunction with Q-Learning, Zap Q-Learning, and Actor-Critic methods, using the algorithm proposed by Meyn et al. [2] as a reference.

2 Critique

The paper [1] extends the stochastic control problem to the non-linear case, as well as to the problem of optimal stopping. Optimal stopping is unique as the policy space $\mathcal{U}$ consists of only two possible values: 0 and 1. Another unique aspect is the time horizon for BMCOs are quite short, with our simulations having a maximum number of samples of only $N = 9$. This is not enough iterations for an algorithm like stochastic gradient descent (SGD) to converge, so instead a very large number of simulations M must be run to get good estimates for the stopping time τ. In this section we use the analysis of Meyn et al. [2] to view the problem from the viewpoint of reinforcement learning.

One major difference between the two papers is the design of the policy function ϕ. Instead of a single function dependent on Q^θ as in [2], the paper [1] uses a family of functions $\phi^\theta = \{f_0^\theta, ..., f_N^\theta\}$, one for each timestep, to find the optimal stopping time. A major downside to this approach is, for increasing N, that problem becomes exponentially more complex to solve. An alternate deep learning method for estimating the policy is presented in the Extensions section.

A major upside to deep learning approaches, when compared to the linear approach uses in [2], is the Universal Approximation Theorem for neural networks. This theorem states that, given a sufficiently large network, it can approximate any function. This can greatly simplify the problem setup, as there is not a need to set up a family of basis functions ψ_i. Instead, a sufficiently complex neural network can approximate these basis functions. Though this makes the problem setup for specific stochastic models and their value function easier, it does increase the computational complexity for estimating ϕ.

One last point worth addressing is the derivation of an upper bound $\hat{U}$ estimate for the returned value in [1], given below:

$$\hat{U} = \frac{1}{K_U} \sum_{k=1}^{K_u} \max_n \{g(n, x_n^k), \sum_{m=1}^{n} \Delta M_m^k\} \tag{4}$$

where the Martingale process ΔM_m^k is estimated as

$$\Delta M_m^k = f^{\theta_n}(x_m^k) g(n, x_m^k) + (1 - f^{\theta_n}(x_m^k)) C_m^k - C_{m-1}^k \tag{5}$$

$$C_m^k = \frac{1}{J} \sum_{j=1}^{J} g(\tau_{m+1}^{k,j}, x_{\tau_{m+1}^{k,j}}^{k,j}) \tag{6}$$

Our point of contention concerns the estimate C_m^k, which is meant to estimate the process

$$\begin{aligned} C_n^\theta &= \mathbb{E}[g(\tau_{n+1}^\theta, X_{\tau_{n+1}^\theta}) | \mathcal{F}_n] \\ &= \mathbb{E}[g(\tau_{n+1}^\theta(x), X_{\tau_{n+1}^\theta}) | X_n = x] \\ &= \int_y P(x, dy) g(\tau_{n+1}^\theta(y), X_{\tau_{n+1}^\theta}) \\ &\approx \sum_y P(x, y) g(\tau_{n+1}^\theta(y), X_{\tau_{n+1}^\theta}) \end{aligned} \tag{7}$$

with the equation for τ_n^θ being given in equation 2. From this reformulation, we can see that the estimate given in equation 6 assumes a uniform distribution, which is unlikely to be the case. A more accurate estimate would be to take into account the entire distribution and perform a weighted sum based on said distribution.

3 Extensions

By tweaking the setup of our optimal control problem, we can apply reinforcement learning techniques such as Q-Learning, Actor-Critic, and even Zap. We begin by generalizing our family of policies $\{f_0, f_1, ..., f_N\}$ into a single policy function $\phi : \mathcal{X} \to \mathcal{U}$ that can be applied at all timesteps. In addition, we use a discount factor $\beta \in (0, 1)$ in conjunction with our expected reward, which is formulated as follows:

$$\mathbb{E}[\sum_{n=0}^{\tau-1} \beta^n c(X_n) + \beta^\tau c_s(X_\tau)] = -\mathbb{E}[\beta^\tau g(X_\tau)] \tag{8}$$

which is found by setting the non-terminal cost $c(\cdot) = 0$ and the terminal cost $c_s(\cdot) = -g(\cdot)$ [2]. That is to say, we are limiting our scope of analysis to pricing functions that provide rewards only at the stopping time, and we will see that this greatly simplifies analysis and algorithm formulation. As an example, for our Bermudan Max Call Options problem used in the experiments section, $\beta = e^{-r/T}$ and $g(x) = (\max\{x_1, ..., x_d\} - K)^+$ [1].

3.1 Q-Learning with Function Approximation

The optimal value function and Q-function associated with equation 8 is given below [2]:

$$h^*(x) = \inf_\tau - \mathbb{E}[\beta^\tau g(X_\tau)|X_0 = x] \tag{9}$$

$$\begin{aligned} Q^*(x) &= \beta\mathbb{E}[h^*(X_1) \mid X_0 = x] \\ &= \mathbb{E}[\min\{-g(X_1),\ Q^*(X_1)\} \mid X_0 = x] \\ &= \mathbb{E}[\max\{g(X_1),\ Q^*(X_1)\} \mid X_0 = x] \end{aligned} \tag{10}$$

To estimate the Q-function, we parameterize it with a neural network, resulting in our estimate $Q^\theta : \mathbb{R}^d \to \mathbb{R}$, and its gradient $\nabla_\theta Q^\theta$ being known (most machine learning frameworks such as Tensorflow and PyTorch provide gradients of the constructed model). To estimate Q^θ we use temporal difference (TD) learning which, when paired with stochastic gradient descent (SGD) and Polyak-Ruppert (PR) averaging, will give us an optimal estimate. The TD term is given below [2]:

$$\mathcal{D}_{n+1} = \beta\max\{g(X_{n+1}),\ Q^{\theta_n}(X_{n+1})\} - Q^{\theta_n}(X_n) \tag{11}$$

Using the TD(λ) algorithm, our algorithm to estimate our optimal policy for a given simulation is as follows:

Algorithm 1 Deep Q Learning for Optimal Stopping

Initialize:
θ_0, ζ_0
for n = 1, ..., N **do**
$\mathcal{D}_{n+1} = \beta\max\{g(X_{n+1}),\ Q^{\theta_n}(X_{n+1})\} - Q^{\theta_n}(X_n)$
$\zeta_{n+1} = \lambda\beta\zeta_n + \nabla_{\theta_n} Q^{\theta_n}(X_{n+1})$
$\theta_{n+1} = \theta_n + \alpha_{n+1}\zeta_n\mathcal{D}_{n+1}$
end for

Here, β and λ are hyperparameters valued between 0 and 1. The step size α_n can be some sequence that converges to 0 yet has a sum of infinity, such as $\alpha_n = 1/n$. Since we are working with such a small time horizon, we propose using PR averaging after each iteration of n to get θ_n. It is also worth mentioning that since we are maximizing a reward rather than minimizing a cost, we are technically performing gradient *ascent*, though this is as simple as making a sign change in algorithm 1 for the update of θ_n.

3.2 Zap Stochastic Approximation

Zap Stochastic Approximation uses a matrix gain to approximate Newton-Raphson flow, which under assumptions of a function to approximate f, has ideal transient behavior and optimal asymptotic variance [4]. The matrix gain $\hat{A}_{n+1}$ is recursively updated with the following equation:

$$\hat{A}_{n+1} = \hat{A}_n + \delta_{n+1}[A_{n+1} - \hat{A}_n] \tag{12}$$

where $A_{n+1} = \partial_\theta f_{n+1}(\theta_n)$. The update using SGD for our Q-function parameters θ is then:

$$\theta_{n+1} = \theta_n + \alpha_{n+1} G_{n+1} f_{n+1}(\theta_n) \tag{13}$$

where $G_{n+1} = \hat{A}^\dagger_{n+1}$ and $\dagger$ denotes the psuedo-inverse [4]. Coming up with A_{n+1} is non-trivial for the non-linear case, and so an exact solution is not given, but a general approach one may use is proposed. The problem can essentially be boiled down to how do we estimate the expression $A_{n+1} = \partial_\theta f_{n+1}(\theta_n, X_n) = \partial_\theta \zeta_n \mathcal{D}_{n+1}$ for each timestep n [3]. By applying the chain rule to the aforementioned equation, we can simplify into the following expression:

$$A_{n+1} = \zeta_n[\beta\max\{0,\ \partial_\theta Q^{\theta_n}(X_{n+1})\} - \partial_\theta Q^{\theta_n}(X_n)]^T + [\partial_\theta \zeta_n]\mathcal{D}^T_{n+1} \tag{14}$$

In some cases, we can even ignore the $[\partial_\theta \zeta_n]\mathcal{D}_{n+1}^T$ term, greatly reducing complexity (especially for the case where TD($\lambda \neq 0$)) [3]. With this in mind, we can use a two-time-scale to estimate θ^* and $A(\theta)$ in conjunction, resulting in an algorithm with better convergence characteristics at the cost of increased complexity.

Algorithm 2 Deep Zap Q Learning for Optimal Stopping

Initialize:
$\theta_0, \zeta_0, \hat{A}_0$
for n = 1, ..., N **do**
$\mathcal{D}_{n+1} = \beta \max\{g(X_{n+1}),\ Q^{\theta_n}(X_{n+1})\} - Q^{\theta_n}(X_n)$
$A_{n+1} = \zeta_n[\beta \max\{0,\ \partial_\theta Q^{\theta_n}(X_{n+1})\} - \partial_\theta Q^{\theta_n}(X_n)]^T$
$\hat{A}_{n+1} = \hat{A}_n + \delta_{n+1}[A_{n+1} - \hat{A}_n]$
$\zeta_{n+1} = \lambda\beta\zeta_n + \nabla_{\theta_n} Q^{\theta_n}(X_{n+1})$
$\theta_{n+1} = \theta_n + \alpha_{n+1} G_{n+1} \zeta_n \mathcal{D}_{n+1}$
end for

It is imperative that the step size sequences α_{n+1} and δ_{n+1} be updated in a manner in which the step size updating $A(\theta)$ converges faster than that updating Q^θ. A simple scheme is given as follows:

$$\alpha_n = \frac{1}{n}, \quad \delta_n = \frac{1}{n^\rho} \tag{15}$$

with $\rho \in (0.5, 1)$ [2]. We again use PR averaging in conjunction with repeated runs to obtain a better estimate of θ^*.

3.3 Actor-Critic Method

Actor-Critic methods, in addition to estimating the Q-function, also estimates the policy ϕ^ω. One common example of a policy is Gibb's policy:

$$\phi^\omega_{\text{Gibb}}(u \mid x) = \exp\left(\frac{F^\omega(x,u)}{\epsilon}\right) / \sum_u \exp\left(\frac{F^\omega(x,u)}{\epsilon}\right) \tag{16}$$

where $F^\omega : \mathcal{X} \times \mathcal{U} \to \mathbb{R}$, $\epsilon \in \mathbb{R}$ is the "temperature" term vital for numerical stability, and the bottom summation is used to normalized the policy to ensure $\phi^\omega_{\text{Gibb}}$ is a valid distribution [4]. We can then parameterize F^ω as a neural network, and by placing a softmax activation function on the output, construct the Gibb's policy function. Because we are using sampling to define our policy, Actor-Critic methods are an example of off-policy learning, where the algorithm "searches" the policy space to help develop its policy.

Algorithm 3 Deep Actor-Critic Method for Optimal Stopping

Initialize:
$\omega_0, \theta_0, \zeta_0$
for n = 1, ..., N **do**
$U_n \sim \phi^\omega(\cdot \mid X_n)$
if $U_n \neq 1$ **then**
$\mathcal{D}_{n+1} = \beta \max\{(1 - U_n) g(X_{n+1}),\ Q^{\theta_n}(X_{n+1})\} - Q^{\theta_n}(X_n)$
$\omega_{n+1} = \omega_n + \delta_{n+1} \nabla_\omega \log \phi^\omega(U_n \mid X_n) Q^{\theta_n}(X_n)$
$\zeta_{n+1} = \lambda\zeta_n + \nabla_{\theta_n} Q^{\theta_n}(X_{n+1})$
$\theta_{n+1} = \theta_n + \alpha_{n+1} \zeta_n \mathcal{D}_{n+1}$
end if
end for

With this algorithm, we use two deep neural networks, one to estimate the policy and the other to estimate the Q-function (as before). The practitioner must ensure that the step size sequence associated with the update of the Q-function α_n is larger than that of the policy function δ_n, as done in the Zap Stochastic Approximation section, to ensure stability [4]. We choose to ignore the estimate of η as the non-terminal cost function is always 0. Due to the added estimate of ϕ^ω, it'll likely take much more simulation runs to achieve good estimates.

4 Numerical Experiments

4.1 Procedure

We begin by simulating Bermudan Max-Call Options (BMCO) which have a pricing depending on the maximum of d assets of the time of stopping. They follow the following stochastic model:

$$S_t^i = s_0^i \exp\left([r - \delta_i - \sigma_i^2]t/2 + \sigma_i W_t^i\right) \tag{17}$$

for $i = 1, ..., d$ assets and a Wiener process W_t^i [1]. $M = 5000$ Monte Carlo simulations are run and the value $g(\cdot)$ found for each time index off each simulation. An example of a batch of values is given in figure 1. In addition, we obtain an estimate of equation 1, given in figure 2. This estimate was vital in interpreting optimal stopping times, which is given by the maximum of its trace. The value function for BMCOs is given in equation 18.

$$g(x) = (\max_i\{x_i\} - K)^+, \quad \beta = e^{-r/T} \tag{18}$$

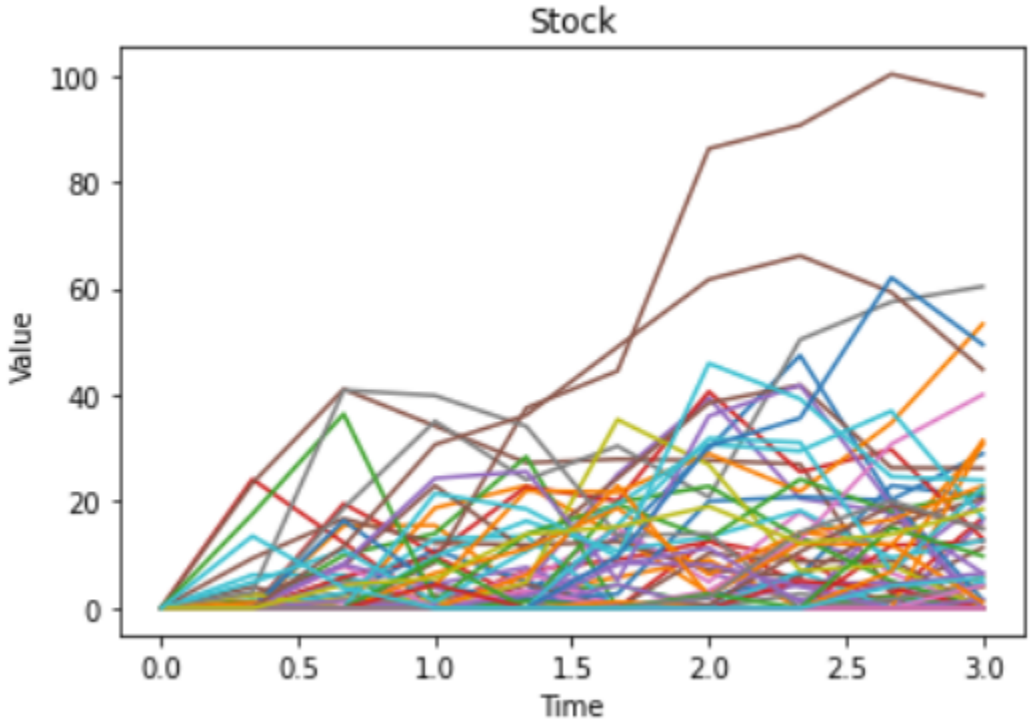

Figure 1: Stock return over multiple runs for $d = 2, s_0 = 90$

To develop the deep neural networks, Keras was used in conjunction with a custom training loop. For each time index n, a neural network f_n^θ is constructed. As in the paper [1], each network is 2 layers deep with $d + 40$ units in each layer. I found that Xavier initialization (as used in the paper) yielded inconsistent results, and so instead always initialized with all zeros to allow more deterministic starting conditions. An Adam optimizer with learning rate of 0.1 was used to update the weights. The loss function used was simply the negative of equation 3. For training, we start with $f_N^\theta \equiv 1$, and recursively update the remaining networks working backwards. Each network was trained using 50 epochs, and the outputs rounded to either 0 or 1. The stopping times τ_m for each simulation was found by finding the first occurrence of 1. The stopping times τ_m and their corresponding values $X_{\tau_m}^m$ were then plugged into equation 18 and the results averaged to find the lower bound of our returned value found by our trained models.

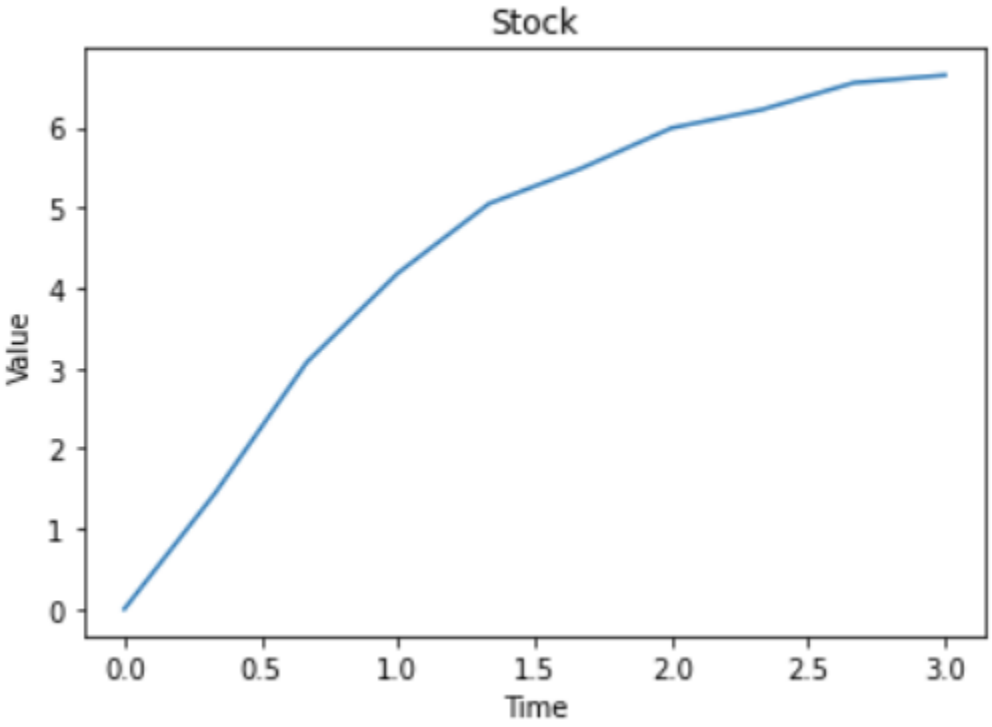

Figure 2: Expected returns for $d = 2, s_0 = 90$

4.2 Results

Simulation results are illustrated in table 1. The estimate for the value using averaging is a lower bound, hence it is denoted as $\hat{L}$. We can see that that the results in our paper match those in the original [1].

d	s_0	Our $\hat{L}$	Original $\hat{L}$
2	90	6.936	8.072
2	100	11.239	13.895
5	90	14.805	16.648
5	100	22.997	26.156
20	90	35.782	37.701
20	100	49.292	51.571
50	90	52.398	53.903
50	100	67.843	69.582
200	90	78.022	78.993
200	100	96.357	97.405

Table 1: Simulation results for estimated lower bound $\hat{L}$

Learning curves are shown in figure 3. Early stopping was used to reduce training time. In this case, $\tau = N$ is the optimal stopping time, so the networks learn to push the outputs to zero, resulting in each model having the same rewards being generated. In cases where the stopping time was $\tau \neq N$, one will observe all traces for $n \leq \tau$ to reach the maximum reward, and traces for $n > \tau$ reach a reward

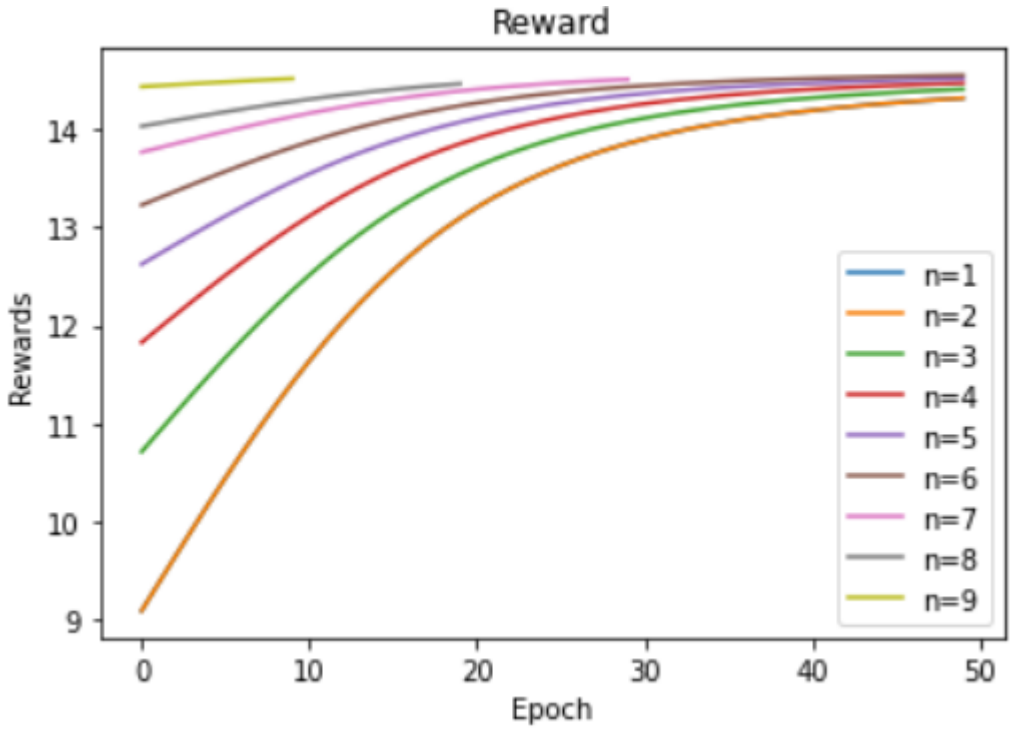

Figure 3: Learning Curves for $d = 5, s_0 = 90$

below that of τ. One may notice that the curves are very smooth, this is because all M samples are being considered for the gradient's estimate, rather than in SGD where the gradient is computed on a sample-by-sample basis. One reason for not using SGD here is because the time horizon we are working with is very small ($N = 9$) and SGD would not be able to converge to any meaningful value in this time. As a quick sanity check of the results for $d = 2, s_0 = 90$, we can see its value $\hat{L} = 6.936$ matches the maximum value found in figure 2. This was done for each of the cases, as well as checking which time index the first $f_n^\theta = 1$ was being found on average. Overall, we found that that method outlined above provided accurate estimates for the optimal stopping time in every case that was tested.

5 Appendix

A link to the repository with all the code used for this paper can be found here. [1]

Symbol	Variable	Value
s_0	Starting value	90, 100
d	Asset count	2, 5, 20, 50, 200
r	Risk-free interest rate	0.05
δ_i	Dividend yields	0.10
σ_i	Volatility	0.20
T	Time length	3
N	Sample count	9
M	Simulations count	5000
K	Strike price	100

Table 2: List of symbols

[1] https://github.com/jcornell616/DeepOptimalStopping

References

[1] Sebastian Becker, Patrick Cheridito, and Arnulf Jentzen. "Deep optimal stopping". In: *The Journal of Machine Learning Research* 20.1 (2019), pp. 2712–2736.

[2] Shuhang Chen et al. "Zap Q-Learning for optimal stopping". In: *2020 American Control Conference (ACC)*. IEEE. 2020, pp. 3920–3925.

[3] Shuhang Chen et al. "Zap Q-learning with nonlinear function approximation". In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 16879–16890.

[4] Sean Meyn. *Control systems and reinforcement learning*. Cambridge University Press, 2022.