Discussion on Deep Optimal Stopping

Final Project: STA 4246

Vanessa Pizante

University of Toronto
Student ID: 1004874022
v.pizante@mail.utoronto.ca
April 12, 2019

Abstract

In this project, we primarily focus on employing a feed-forward, multi-layer neural network to solve the optimal stopping problem, as proposed by [Becker et al., 2018]. We will also discuss related neural network and machine learning approaches to optimal stopping problems. We highlight how the primary motivation for such approximation techniques is to remedy the issues encountered by traditional numerical methods in higher dimensions. We illustrate this by pricing a Bermudan max-call option written on a large number of assets and by examining how a fractional Brownian motion can be stopped optimally by reframing it as a Markov process by increasing the dimension of the problem.

1 Introduction

An optimal stopping problem involves using sequentially observed random variables to choose the time to take an action in order to maximize some predefined notion of reward (or equivalently, minimize some loss). In this report, we deal with problems where said sequence of random variables is denoted $X = (X_n)_{n=0}^N$ and is an $\mathbb{R}^d$-valued discrete time Markov process on a filtered probability space $(\Omega, \mathcal{F}, (\mathcal{F})_{n=0}^N, \mathbb{P})$. In this context, mathematically we say τ is an X-stopping time if $\{\tau = n\} \in \mathcal{F},\ \forall n \in \{0, 1, ..., N\}$. The notion of reward we look to maximize is given by

$$V = \sup_{\tau \in \mathcal{T}} \mathbb{E} g(\tau, X_\tau) \tag{1}$$

where $\mathcal{T}$ is the set of all X-stopping times and $g : \{0, 1, ..., N\} \times \mathbb{R}^d \to \mathbb{R}$ is a measurable, integrable function.

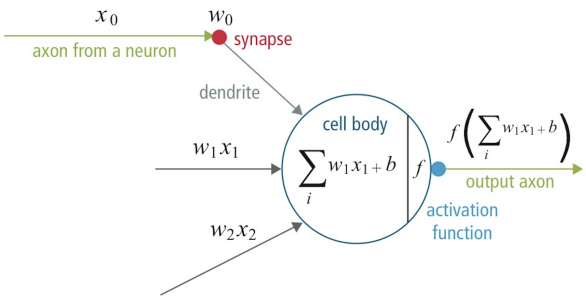
Theoretically optimal stopping problems with finitely many stopping times such as this can be solved exactly, where the optimal V is given by the Snell envelope and the corresponding optimal stopping time is the first time the reward from stopping exceeds the expected reward from continuing (the continuation value). Numerically, these problems are often solved via a dynamic programming based approach. However, more traditional numerical approaches, such as tree-based methods, perform poorly if the dimensionality of the Markov process X is larger than three [Becker et al., 2018]. This limits the types of optimal problems that can be solved. For example being able to work in higher dimensions allows us to price options written on hundreds of underlying assets with many possible exercise times. Furthermore, any process $X = (X_n)_{n=0}^N$ can be made Markov by including any required information about the history of the process in its current state. However, this obviously increases the dimensionality of the problem.

An emerging branch of computational mathematical finance research aims to apply machine learning approaches to optimal stopping problems. Many machine learning algorithms are notoriously strong in very high dimensions and are flexible enough to apply to a wide array of problems. The main focus of this report is the feed-forward, multi-layer neural network applied to the optimal stopping problem by [Becker et al., 2018]. Before delving into their approach, we will first go over some preliminary information on neural networks.

1.1 Neural Networks

A neural network is a computational system modelled loosely after the human brain. Without delving too much into the biological specifics, the basic idea is that a neuron receives input from other neurons or from an external source which it uses to generate output. This output is then either passed onto another neuron as input or treated as the output of the entire system. In computational neural networks these neurons are often referred to as nodes.

The way in which a single node takes input and transforms it to output is illustrated in figure 1. The x_1, x_2 are inputs, which are either the inputs of the overall network if this node is in the first layer or the outputs of the previous layer. Each x_i has a corresponding w_i, called its weight, which quantifies the relative significance of said input. Then, as can be seen in the "cell body" in figure 1, these inputs and weights are then summed with an additional bias term. This summation is then transformed via an activation function f and after it becomes the output of that node. The activation function output can be thought of as the relative firing rate of this node.



Source: [Hijazi et al., 2015]

Figure 1: A single node in a neural network.

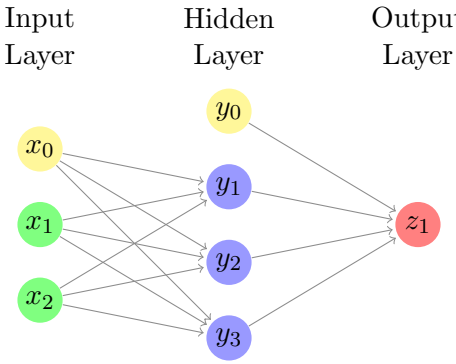


Figure 2: Example of a feed-forward neural network.

Figure 2 illustrates how nodes are linked in a dense, single-layer, feed-forward neural network architecture. The term dense refers to how the network is fully connected, meaning a given node takes input from all nodes in the previous layer and sends its output to all the nodes in the next layer and single-layer simply refers to the single hidden layer featured in the network.

To better understand what the neural network in figure 2 does mathematically, we can describe it as the composite function $f(x) = z_1$, with $x = (x_1, x_2)$ and $y = (y_1, y_2, y_3)$, such that:

$$f^\theta = a_1 \circ \phi_1 \circ a_2 \circ \phi_2$$

where:

- $a_1 : \mathbb{R}^2 \to \mathbb{R}^3$ and $a_2 : \mathbb{R}^3 \to \mathbb{R}^2$ are linear functions of the forms $a_1(x) = W_1x + x_0$ and $a_2(x) = W_yy + y_0$ where $W_1 \in \mathbb{R}^{3\times2}$, $W_2 \in \mathbb{R}^{1\times3}$, $x_0 \in \mathbb{R}^3$, $y_0 \in \mathbb{R}$.
- $\phi_1 : \mathbb{R}^3 \to \mathbb{R}^3$ and $\phi_2 : \mathbb{R} \to \mathbb{R}$ are the activation functions corresponding to the hidden layer and output layer. Common choices for these are the ReLU activation function or the logistic sigmoid.
- The θ part of f^θ refers to the parameters of the model which includes all the model's weight and bias terms, i.e. $\theta = \{W_1, x_0, W_2, y_0\}$.

Once a model's architecture has been established, one then must train the model. Training the model refers to using training data, which can consist of input and target pairs, $(x^m, z_1^m)_{m=1}^M$, or just inputs, $(x^m)_{m=1}^M$, in order to learn the optimal θ. Optimal in this sense refers to minimizing some loss function (or maximizing a reward function), denoted C. A common example of such a function for paired training data is the mean squared-error loss function

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

The minimization of C with respect to θ is typically done through some kind of gradient descent algorithm. The most basic example of this is vanilla gradient descent, which has the update step:

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

where η is the learning rate. We approximate $\nabla_\theta C(\theta^t)$ at each iteration using a graphical differentiation technique called back propagation, which is just a method for efficiently moving backwards through the neural network graph to compute each $\partial C/\partial\theta_i$ via the chain rule. The completion of training occurs when the gradient descent algorithm is complete, meaning $\nabla_\theta C(\theta^t) < \epsilon$ where ϵ is some pre-selected threshold close to 0, since this means $C(\theta^t)$ has reached some kind of extrema. After the network has been trained, we then run the algorithm on a new dataset, which is called the testing data.

Now that we have finished introducing the optimal stopping problem and the basics of neural networks, prior to moving into the methodology of [Becker et al., 2018] we conduct a brief survey of other machine learning techniques that have been applied to the optimal stopping problem.

2 Literature Review

As previously stated, the primary motivation for applying machine learning approaches to the optimal stopping problem is their ability to overcome the curse of dimensionality faced by earlier methods. This is highlighted by [Becker et al., 2018] as a catalyst for their multi-layer, feed forward neural network approach, but there are several other paper that apply related machine learning methods to tackle this same issue. In the following section, we briefly look at the main ideas of these similarly motivated mechanisms.

2.1 Neural Networks Applied to Least-Squares Monte Carlo

A strong illustration of employing a machine learning technique to an earlier approach to the optimal stopping problem is in [Kohler et al., 2010] which applies a neural network to the Longstaff-Schwartz algorithm. The Longstaff-Schwartz algorithm [Longstaff and Schwartz, 2001] uses a least-squares Monte Carlo (LSMC) approach to compute approximate the price of an American option, a benchmark optimal stopping application.

Pricing an American call option written on an asset with price process (X_t), strike price K and expiry time T simply means computing V in (1), where $g(t, X_t) = (X_t - K)^+$ and $\mathcal{T} = \tau | 0 \leq t \leq T$. In both [Longstaff and Schwartz, 2001] and [Kohler et al., 2010] time is discretized by $\{t_n\}_{n=0}^N$, where $t_n = nT/N$, which is then written $\{0, 1, ..., N\}$ for notational simplification. Then, M asset price paths of the form $(x_n^m)_{n=0}^N$ are simulated and after the time N option price, $V_N^m = g(n, x_n^m)$, is computed along each path. Both algorithms then proceed backwards recursively for $n = N-1, ..., 0$ by approximating the continuation value, q_n, at each time step and then exercising if $q_n(x_n^m) < g(n, x_n^m)$, which

translates to setting

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

where r is the market risk-free return rate.

The key way in which the approaches differ is the manner in which they compute $q_n(x)$. The elder Longstaff-Schwartz algorithm employs least squares Monte Carlo in computing the continuation value. This involves approximating q_n via $\hat{q}_n(x) = \sum_{k=0}^{K} \beta_k \phi_k(x)$ where $\phi_k(x)$ is some activation function, typically polynomial (i.e. $\phi_k(x) = x^k$) and $\boldsymbol{\beta} = (\beta_k)$ minimizes the least-squares error function

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

As previously alluded to, this approach for computing the continuation value becomes less reliable for options written on a large number of assets. The alternative proposed by [Kohler et al., 2010] is to build a a single-layer neural network with a least-squares loss function in order to deal with these higher dimensional problems. Specifically they propose a neural network of the form

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

where

- $a_1 : \mathbb{R}^d \to \mathbb{R}^K$ is a linear function of the form $W_1 x + b_1$ where $W_1 \in \mathbb{R}^{K \times d}$, $b_1 \in \mathbb{R}^d$
- $\sigma : \mathbb{R}^k \to [0,1]^K$ is the sigmoid activation function where for each of the k entries, $\sigma_k(y_k) = \frac{1}{1+e^{-y_k}}$
- $a_2 : \mathbb{R}^K \to \mathbb{R}$ is a linear function of the form $W_2 x + b_2$ where $W_2 \in \mathbb{R}^K$, $b_2 \in \mathbb{R}$ and $W_2 = (w_{2,1}, ..., w_{2,K})$ satisfies $\sum_{k=0}^{K} |w_{2,k}| \leq C_n$

An important note about the algorithm is that it creates training pairs, (x_n^m, V_{n+1}^m), backwards recursively by employing a loss function of the form (2), which relies on the output of the previous step backwards, V_{n+1}^m. The results of [Kohler et al., 2010] show that when applied to more complex options (specifically American strangles) in higher dimensions, their algorithm outperforms the Longstaff-Schwartz algorithm, while in lower dimensions for simpler options (an American put), their results are roughly the same. This illustrates how older methods of solving the optimal stopping problem can be modified using a neural network to solve more complex problems. Next, we will look at another approach to the optimal stopping problem that employs a different machine learning technique: Q-Learning.

2.2 Q-Learning Approach

A different branch of machine learning algorithms that have been applied to optimal stopping problems is reinforcement learning. In general, these algorithms are the machine learning approach to Markov Decision Processes. The key difference between reinforcement learning algorithms and more traditional approaches to Markov Decision Processes is that the reinforcement learning approaches are model-free, meaning they do not impose a distribution on the transition probabilites of the Markov Process. The specific approach we are going to examine here is a Q-Learning algorithm from [Yu and Bertsekas, 2007]. It should be noted that these authors also cite the poor performance of dynamic programming in larger state spaces as a key motivation for applying this approach.

Applying a Q-learning approach to an optimal stopping problem involves viewing said problem as a Markov Decision Process governed by the (X_n) where aim to minimize the expected value of the total α-discounted cost ($\alpha \in (0,1)$) by making the optimal sequence of decisions to stop or continue at each step. Instead of trying to minimize an overall cost function, it is more computationally efficient examine the cost of each decision individually from each state - this cost function of each decision at each state is called the Q-function.

Therefore, given we are in state X_n we have two possible decisions: to stop and incur cost $f(X_n)$ or continue and incur cost $h(X_n, X_{n+1})$ (we assume the values of these functions are known from state X_n). So since the Q-function is simply the cost associated with any given action given the state X_n, meaning $Q(X_n, \text{stop}) = f(X_n)$ and $Q(X_n, \text{cont.}) = h(X_n, X_{n+1})$. Thus the optimal Q-function in state X_n is defined as $Q^*(X_n) = \min_{a \in \{\text{stop,cont.}\}} Q(X_n, a)$. It follows that the optimal stopping policy can be reframed as stopping as soon as X_n enters the following set:

$$\mathcal{D} = \{X_n | f(X_n) \leq Q^*(X_n)\}$$

To find this Q^* function, we employ Q-Learning, which is simply an iterative algorithm to approximate the Q-function so that it minimizes the corresponding Bellman error. Specifically, after simulating some unstopped state process (x_n) the Q-Learning algorithm update condition to approximate Q^* is:

$$Q^*(x_n) \leftarrow Q^*(x_n) + \gamma(g(x_n, x_{n+1}) + \alpha \min\{c(x_{n+1}), Q^*(x_{n+1})\} - Q^*(x_n))$$

where $\gamma \in (0,1)$ is the learning rate.

To further the computational efficiency of the algorithm, [Yu and Bertsekas, 2007] considers approximations for Q^* that use a linear projection of a transformation of X_n:

$$Q^*(x_n) \equiv \phi(x_n)'\beta_n$$

The corresponding projected Bellman error is then minimized with respect to β_n via a least-squares approximation based on the Q-learning algorithm as follows:

$$\beta_{n+1} = \beta_n + \alpha(\hat{\beta}_{n+1} - \beta_n)$$

where

$$\hat{\beta}_n = \arg\min_{\beta} \sum_{k=0}^{n} \left(\phi(x_k)'\beta - h(x_k, x_{k+1}) - \gamma \min\{f(x_{k+1}), \phi(x_k)'\beta_k\}\right)^2$$

Note that this is a very slight modification of the Q-learning algorithm, the key difference being it only needs find the minimizing vector of constants β_n, instead of minimizing a possibly complex Q-function.

It is worth noting that there are no numerical examples provided in [Yu and Bertsekas, 2007], so it is difficult to assess the performance of this algorithm in practice. An additional issue I found with the paper is that it does not discuss how to best obtain the continuation value function $h(X_n, X_{n+1})$, as in many optimal stopping problems this value is unknown. However, examining it for our purposes still offers an interesting look into how a branch of machine learning apart from neural networks can also be used to solve the optimal stopping problem.

2.3 Extensions: More Complex Neural Networks

Lastly, we will return to our discussion on the application of neural networks to the optimal stopping problem by looking at a recent extension of [Becker et al., 2018]. The work we look to examine is [Hu, 2019], where they look to recast the optimal stopping problem as an image classification problem. This recasting is done using rank response surfaces, which references the process in which we assign each input x a class label. The primary motivation of transforming the problem in this manner is that it allows us to replace the simple feed-forward network architecture employed in [Becker et al., 2018] and [Kohler et al., 2010] with a Convolution neural network, which can yield a better performance.

As previously alluded to, the first step in applying this method is recasting the problem as a surface ranking problem. This involves extracting the index of the minimal surface for each input x and treating it as a class label. Hence we are segmenting the input space $\mathcal{X}$ into classes. In an optimal stopping problem at time n we only have two labels: stop and continue, so our optimal choice can be reframed as surface ranking problem as follows:

$$\mathcal{C}(n, X_n) = \arg\max_{j \in \{stop,\ cont.\}} \mu_j(n, X_n)$$

where $\mu_{stop}(n, X_n) = g(n, X_n)$ is the reward associated with stopping and $\mu_{cont.}(n, X_n)$ is the reward associated with continuing at time n assuming the optimal path is taken at times $n+1, ..., N$. Note that $\mu_{cont.}(n, X_n)$ does not have a closed form but it can be computed using simulations. The process in which this is done is in fact very similar to the work of **??**, which we examine in detail in later sections of this paper. The key difference is, by employing these rank response surfaces, convolution neural networks can be employed to replace the dense feed-forward neural network propose by [Becker et al., 2018].

Employing a more delicate network architecture such as a CNN can offer increased computational efficiency. This is illustrated by [Hu, 2019] as they demonstrate that their Bermudan option price simulations were computed more quickly and efficiently than in [Becker et al., 2018]. However, they do caution that there is a lack of mathematical convergence theory for fully convolutional neural networks and thus they were unable to confirm mathematically that their method offers reasonable approximations. On the other hand [Becker et al., 2018] is able to employ established convergence theory for dense, feed-forward neural networks (e.g. [Leshno et al., 1993]) to ensure the convergence of their approximations.

3 Mathematical Framework

In this section, we focus on the mathematical framework required to implement a multi-layer, feed-forward neural network to solve an optimal stopping problem of the form (1). There are two key theorems we employ to do this. The first shows that the optimal stopping time, τ, can be expressed as a function of a series of 0-1 stopping decisions. The second shows that we can approximate these 0-1 stopping decisions using a feed-forward, multi-layer neural network. These key results establish mathematically how a neural network can be employed to solve the optimal stopping problem.

3.1 Expressing Stopping Times as a Series of Stopping Decisions

Our first task is to show that the decision to stop the Markov process $(X_n)_{n=0}^N$ can be made according to a sequence of functions $\{f_n(X_n)\}_{n=0}^N$, $f_n : \mathbb{R}^d \to \{0, 1\}$. To do this, we first divide our overall optimal stopping problem in (1) into a sequence of stopping problems. Specifically, consider the time

n stopping problem

$$V_n = \sup_{\tau \in \mathcal{T}_n} \mathbb{E} g(\tau, X_\tau) \tag{3}$$

where $\mathcal{T}_n$ is the set of all X-stopping times such that $n \leq \tau \leq N$. Clearly, since the process must be stopped at time N, $\mathcal{T}_N = \{N\}$ and thus the time N optimal stopping time is $\tau_N = N$. Furthermore, since $\{f_n(X_n)\}_{n=0}^N$ is our sequence of 0-1 stopping decisions, we have write $f_N \equiv 1$ and hence can write $\tau_N = N f_N(X_N)$.

Now that we have written the time N optimal stopping time as a function of the 0-1 stopping decisions $f_N(X_N)$, we aim to do the same for $0 \leq n \leq N-1$. Meaning we aim to write the time n stopping time, τ_n, as function of $\{f_k(X_k)\}_{k=n}^N$. We propose to do so using the following equation:

$$\tau_n = \sum_{k=n}^{N} k f_k(X_k) \prod_{j=n}^{k-1} (1 - f_j(X_j)) \tag{4}$$

where $f_N \equiv 1$. This defines a stopping time in $\mathcal{T}_n$, however we have yet to confirm if this stopping time is optimal. The following theorem shows that τ_n is in fact the time n optimal stopping time.

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

$$\tau_{n+1} = \sum_{k=n+1}^{N} k f_k(X_k) \prod_{j=n+1}^{k-1} (1 - f_j(X_j)) \tag{5}$$

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

$$\mathbb{E} g(\tau_n, X_{\tau_N}) \geq V_n - (V_{n+1} - \mathbb{E} g(\tau_{n+1}, X_{\tau_{n+1}})) \tag{6}$$

where V_n and V_{n+1} satisfy (3)*.*

Proof. We begin by setting an arbitrary stopping time $\tau \in \mathcal{T}_n$ and letting $\epsilon = V_{n+1} - \mathbb{E} g(\tau_{n+1}, X_{\tau_{n+1}})$. By Doob-Dynkin,it follows that since g is integrable, $\exists h_n$ measurable such that

$$h_n(X_n) = \mathbb{E}[g(\tau_{n+1}, X_{\tau_{n+1}}) | X_n]$$

Now by (4), the following is also a measurable function of $X_{n+1}, ..., X_N$

$$g(\tau_{n+1}, X_{\tau_{n+1}}) = \sum_{k=n+1}^{N} g(k, X_k) I_{\tau_{n+1}=k} = \sum_{k=n+1}^{N} g(k, X_k) I_{f_k(X_k) \prod_{j=n}^{k-1}(1-f_j(X_j))=1}$$

This shows $g(\tau_{n+1}, X_{\tau_{n+1}})$ is measurable in $X_{n+1}, ..., X_N$ because it can be rewritten as a sum of measuable functions of $X_{n+1}, ..., X_N$. So since $g(\tau_{n+1}, X_{\tau_{n+1}})$ is measurable in $X_{n+1}, ..., X_N$, it follows that since $(X_n)_{n=0}^N$ is Markovian it follows that

$$h_n(X_n) = \mathbb{E}[g(\tau_{n+1}, X_{\tau_{n+1}}) | X_n] = \mathbb{E}[g(\tau_{n+1}, X_{\tau_{n+1}}) | \mathcal{F}_n]$$

Now we can establish that the following sets are in $\mathcal{F}_n$:

$$D = \{g(n, X_n) \geq h_n(X_n)\} \qquad \text{and} \qquad E = \{\tau = n\}$$

Furthermore, the event $\tau_n = nI_D + \tau_{n+1}I_{D^c}$ is in $\mathcal{T}_n$ and $\tilde{\tau} = \tau_{n+1}I_E + \tau I_{E^c}$ is in $\mathcal{T}_{n+1}$. Hence we have:

$$\begin{aligned}
\mathbb{E}g(\tau, X_{\tau_{n+1}}) =& V_{n+1} - \epsilon \\
=& \sup_{\tau \in \mathcal{T}_n} \mathbb{E}g(\tau, X_\tau) - \epsilon \qquad \text{By definiton of } V_n \\
\geq& \mathbb{E}g(\tilde{\tau}, X_{\tilde{\tau}}) - \epsilon
\end{aligned}$$

It follows

$$\mathbb{E}[g(\tau, X_{\tau_{n+1}})I_{E^c}] \geq \mathbb{E}[g(\tilde{\tau}, X_{\tilde{\tau}})I_{E^c}] - \epsilon = \mathbb{E}[g(\tau, X_\tau)I_{E^c}] - \epsilon \tag{7}$$

Note this holds since $I_{E^c} = 1$ means $\tilde{\tau} = \tau$. Finally, we obtain the following:

$$\begin{aligned}
\mathbb{E}g(\tau_n, X_{\tau_n}) &= \mathbb{E}[g(n, X_n)I_D + g(\tau_{n+1}, X_{\tau_{n+1}})I_{D^c}] && \text{By definition of } \tau_{n+1} \\
&= \mathbb{E}[g(n, X_n)I_D + h_n(X_n)I_{D^c}] && \text{By definition of } h_n(X_n) \\
&\geq \mathbb{E}[g(n, X_n)I_E + h_n(X_n)I_{E^c}] && \text{Since } I_D = 1 \text{ if } g(n, X_n) \geq h_n(X_n) \\
&= \mathbb{E}[g(n, X_n)I_E + g(\tau_{n+1}, X_{\tau_{n+1}})I_{E^c}] && \text{By definition of } h_n(X_n) \\
&\geq \mathbb{E}[g(n, X_n)I_E + g(\tau, X_\tau)I_{E^C}] - \epsilon && \text{By (7)} \\
&= \mathbb{E}g(\tau, X_\tau) - \epsilon
\end{aligned}$$

Since this holds for any τ (as it was fixed arbitrarily), we have $\mathbb{E}g(\tau_n, X_{\tau_n}) \geq V_n - \epsilon$ and thus equation 6 is satisfied.

Lastly, we must show that the τ_n we found here is the same as that in 4. Defining $f_n : \mathbb{R}^d \to \{0, 1\}$ by

$$f_n(x) \begin{cases} 1 & \text{if } g(n, x) \geq h_n(x) \\ 0 & \text{if } g(n, x) < h_n(x) \end{cases}$$

we have that $I_D = f_n(X_n)$. Therefore,

$$\begin{aligned}
\tau_n &= nf_n(X_n) + \tau_{n+1}(1 - f_n(X_n)) && \text{By definition of } \tau_n \\
&= \sum_{k=n}^{N} kf_k(X_k) \prod_{j=n}^{k-1}(1 - f_j(X_j)) && \text{By definition of } \tau_{n+1} \text{ in the theorem statement}
\end{aligned}$$

as required.

□

Remark. Theorem 1 shows that stopping times as given in equation (4) can be used to compute V_n. Namely, the inequality (6) is clearly equivalent to

$$\sup_{\tau \in \mathcal{T}_{n+1}} \mathbb{E}g(\tau, X_\tau) - \mathbb{E}g(\tau_{n+1}, X_{\tau_{n+1}}) \geq \sup_{\tau \in \mathcal{T}_n} \mathbb{E}g(\tau, X_\tau) - g(\tau_n, X_{\tau_N})$$

Thus since we know $\tau_N = Nf(X_N)$, it follows by backwards induction that τ_n will provide a sufficient optimal stopping time.

Remark. It follows by theorem 1 that the overall optimal stopping time corresponding to $V = \sup_{\tau \in \mathcal{T}} \mathbb{E}g(\tau, X_\tau)$ is given by

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

3.2 Introducing a Neural Network

Now that we have shown the optimal stopping time can be computed as a function of the series of 0-1 stopping decisions, $\{f_n\}_{n=0}^N$, we need to find a way to approximate said functions. The method of doing this suggested by [Becker et al., 2018] is to employ a neural network. Specifically, we construct a sequence of neural networks of the form $f^{\theta_n} : \mathbb{R}^d \to \{0,1\}$ with parameters $\theta_n \in \mathbb{R}^q$, to approximate f_n. Then, we can simply approximate τ_{n+1} via

$$\sum_{k=n}^{N} k f^{\theta_k}(X_k) \prod_{j=n}^{k-1} (1 - f^{\theta_j}(X_j)) \tag{8}$$

Note that we are referencing τ_{n+1} at step n, not τ_n, since τ_{n+1} is required to compute the continuation value at time n, $V_{n+1} = \mathbb{E}[g(\tau_{n+1}, X_{\tau_{n+1}})]$. As we will later see, V_{n+1} is a key part of the reward function used to train our neural network.

A problem we must first address is that f^{θ_n} outputs only values in $\{0,1\}$ and is consequently not continuous with respect to θ_n. This is an issue because in training the parameters for a neural network, we typically apply a gradient-based optimization algorithm to maximize some reward (or minimize some loss) function of the network. In order to address this, we instead train the parameters via a multi-layer, feed-forward neural network that outputs probabilities in the interval $(0,1)$. Then after training we can then transform our results into 0-1 stopping decisions. Namely, for $\theta \in \{\theta_0, ..., \theta_N\}$ we introduce $F^\theta : \mathbb{R}^d \to (0,1)$ which is a neural network of the form:

$$F^\theta = \psi \circ a_3^\theta \circ \phi_{q_2} \circ a_2^\theta \circ \phi_{q_1} \circ a_1^\theta \tag{9}$$

where

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

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

  where $A_1 \in \mathbb{R}^{q_1 \times d}, A_2 \in \mathbb{R}^{q_2 \times q_1}, A_1 \in \mathbb{R}^{1 \times q_2}$ are matrices and $b_1 \in \mathbb{R}^{q_1}, b_2 \in \mathbb{R}^{q_2}, b_3 \in \mathbb{R}^{q_3}$ are vectors.
- $\phi_{q_i} : \mathbb{R}^{q_i} \to \mathbb{R}^{q_i}$ is the ReLU activation function, which just takes the maximum of each component and 0, meaning $\phi_{q_i}(x_1, ..., x_{q_i}) = (x_1^+, ..., x_{q_i}^+)$.
- $\psi : \mathbb{R} \to \mathbb{R}$ is the logistic sigmoid function, $\psi(x) = 1/(1 + e^{-x})$.

Note that the parameters we need to tune in this network are $\theta = \{A_1, A_2, A_3, b_1, b_2, b_3\} \in \mathbb{R}^q$, where $q = q_1(d + q_2 + 1) + 2q_2 + 1$. After finding the optimal θ_n by training (9), we then define the network $f^\theta : \mathbb{R}^d \to \{0,1\}$ as follows:

$$f^\theta = I_{[0,\infty)} \circ a_3^\theta \circ \phi_{q_2} \circ a_2^\theta \circ \phi_{q_1} \circ a_1^\theta \tag{10}$$

It is worth noting that [Becker et al., 2018] does not include a formal proof to show this substitution is valid in their paper. However, it does make sense intuitively when F^{θ_n} is thought of as the probability that stopping at time n is optimal.

Now that we have defined a smooth version of our stopping decisions and the manner in which we can transform it back to 0-1 stopping decisions, we must define a loss or reward function is order

to tune the parameters θ_n. At each time step, our objective is to maximize our expected future reward. We know at time n that if we stop ($f_n(X_n) = 1$) we will receive a reward of $g(n, X_n)$ and if we continue ($f_n(X_n) = 0$) and then proceed to behave optimally we will eventually receive a reward of $g(\tau_{n+1}, X_{\tau_{n+1}})$ by theorem 1. Hence at each time step, we are aiming to find the function $f : \mathbb{R}^d \to \{0, 1\}$ that maximizes

$$\mathbb{E}[g(n, X_n)f(X_n) + g(\tau_{n+1}, X_{\tau_{n+1}})(1 - f(X_n))] \tag{11}$$

The following theorem shows that we can replace f in (11) with f^θ. This allows us to maximize (11) with respect to $\theta \in \mathbb{R}^q$, instead of having to find the optimal function $f : \mathbb{R}^d \to \{0, 1\}$.

Theorem 2. *Suppose $n \in \{0, ..., N-1\}$ 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*

$$\begin{aligned} &\sup_{\theta \in \mathbb{R}^q} \mathbb{E}[g(n, X_n)f^\theta(X_n) + g(\tau_{n+1}, X_{\tau_{n+1}})(1 - f^\theta(X_n))] \\ \geq &\sup_{f \in \mathcal{D}} \mathbb{E}[g(n, X_n)f(X_n) + g(\tau_{n+1}, X_{\tau_{n+1}})(1 - f(X_n))] - \epsilon \end{aligned} \tag{12}$$

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

Proof. We begin by fixing $\epsilon > 0$. Recall that since g is presumed integrable, we have $\mathbb{E}|g(n, X_n)| < \infty$, $\forall n \in \{0, ..., N\}$ and hence we can find $\tilde{f} : \mathbb{R}^d \to \{0, 1\}$ measurable such that

$$\begin{aligned} &\mathbb{E}[g(n, X_n)\tilde{f}^\theta(X_n) + g(\tau_{n+1}, X_{\tau_{n+1}})(1 - \tilde{f}^\theta(X_n))] \\ \geq &\sup_{f \in \mathcal{D}} \mathbb{E}[g(n, X_n)f(X_n) + g(\tau_{n+1}, X_{\tau_{n+1}})(1 - f(X_n))] - \epsilon/4 \end{aligned} \tag{13}$$

Next, let $\tilde{f} = I_A$ for $A = \{x \in \mathbb{R}^d | \tilde{f} = 1\}$ and note that A is a Borel set. By the integrability condition of g we can also define the following finite Borel measures on $\mathbb{R}^d$:

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

It follows by the tightness of finite Borel measures on $\mathbb{R}^d$ that there exists $K \subseteq A$ compact such that

$$\begin{aligned} &\mathbb{E}[g(n, X_n)I_K(X_n) + g(\tau_{n+1}, X_{\tau_{n+1}})(1 - I_K(X_n))] \\ \geq &\mathbb{E}[g(n, X_n)\tilde{f}^\theta(X_n) + g(\tau_{n+1}, X_{\tau_{n+1}})(1 - \tilde{f}^\theta(X_n))] - \epsilon/4 \end{aligned} \tag{14}$$

Now we define a distance function $\rho_K : \mathbb{R}^d \to [0, \infty)$ by $\rho_K(x) = \inf_{y \in K} \|x - y\|_2$. We can then define a sequence of continuous functions $k_j : \mathbb{R}^d \to [-1, 1]$, $j \in \mathbb{N}$ by

$$k_j(x) = \max\{1 - j \cdot \rho_K(x), -1\}$$

that converge pointwise to $I_K - I_{K^c}$. It follows by Lebesque's dominated convergence theorem that

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

By [Leshno et al., 1993], since k_j has only discontinuities at points of measure 0 and is bounded, it can be approximated uniformly on compact sets by a function $h : \mathbb{R}^d \to \mathbb{R}$ where

$$h(x) = \sum_{i=1}^{r} (v_i^T x + c_i)^+ - \sum_{i=1}^{s} (w_i^T x + d_i)^+ \tag{16}$$

for $r, s \in \mathbb{N}$, $v_1, ..., v_r, w_1, ..., w_s \in \mathbb{R}^d$ and $c_1, ..., c_r, d_1, ..., d_s \in \mathbb{R}$. Thus it follows that

$$\begin{aligned}&\mathbb{E}[g(n, X_n)I_{h(X_n)\geq 0} + g(\tau_{n+1}, X_{\tau_{n+1}})(1 - I_{h(X_n)\geq 0})]\\ \geq&\mathbb{E}[g(n, X_n)I_{k_j(X_n)\geq 0} + g(\tau_{n+1}, X_{\tau_{n+1}})(1 - I_{k_j(X_n)\geq 0})] - \epsilon/4\end{aligned} \tag{17}$$

Recall that $f^\theta = I_{[0,\infty)} \circ a_3^\theta \circ \phi_{q_2} \circ a_2^\theta \circ \phi_{q_1} \circ a_1^\theta$. Thus we can express $I_{[0,\infty)} \circ h$ as a neural network of the form f^θ for sufficiently large q_1, q_2, meaning (17) becomes:

$$\begin{aligned}&\mathbb{E}[g(n, X_n)f^\theta(X_n) + g(\tau_{n+1}, X_{\tau_{n+1}})(1 - f^\theta(X_n))]\\ \geq&\mathbb{E}[g(n, X_n)I_{k_j(X_n)\geq 0} + g(\tau_{n+1}, X_{\tau_{n+1}})(1 - I_{k_j(X_n)\geq 0})] - \epsilon/4\end{aligned} \tag{18}$$

Thus by combining equations (13), (14), (15) and (18), we have

$$\begin{aligned}&\sup_{\theta\in\mathbb{R}^q} \mathbb{E}[g(n, X_n)f^\theta(X_n) + g(\tau_{n+1}, X_{\tau_{n+1}})(1 - f^\theta(X_n))]\\ \geq&\sup_{f\in\mathcal{D}} \mathbb{E}[g(n, X_n)f(X_n) + g(\tau_{n+1}, X_{\tau_{n+1}})(1 - f(X_n))] - \epsilon\end{aligned} \tag{19}$$

as required.

□

The following result is obtained directly from theorems 1 and 2:

Corollary 1. *For any $\epsilon > 0$ and stopping problem $\sup_{\tau\in\mathcal{T}} \mathbb{E}g(\tau, X_\tau)$, there exists $q_1, q_2 \in \mathbb{N}^+$ and neural network functions of the form (10) such that $f^{\theta_N} \equiv 1$ and the stopping time*

$$\hat{\tau} = \sum_{n=1}^{N} n f^{\theta_n}(X_n) \prod_{k=0}^{n-1} (1 - f^{\theta_k}(X_k)) \tag{20}$$

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

We have now established that we can approximate the solution to problems of the form (1) using a sequence of neural networks $\{f^{\theta_n}\}$, where f^{theta_n} is of the form (10). In the next section, we discuss in detail how we will actually compute this neural network.

3.3 Implementation

In this section, we discuss how we are going to actually approximate V in (1). First, we simulate M independent paths of Markov process $(X_n)_{n=0}^N$. These sample paths are denoted $(x_n^m)_{n=0}^N$ for $m = 1, ..., M$. We then set $f^{\theta_N}(x_N^m) \equiv 1$, $\forall m$, since we must stop at time N. We can now compute θ_n is backwards recursively for $n = N - 1, ..., 0$.

Assume we are at time step n, meaning we have already computed $f^{\theta_{n+1}}, ..., f^{\theta_N}$. We can then compute the time $n + 1$ optimal stopping time for each of the m paths using

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

Notice, at time n along path m, if one stops with probability F^{θ_n} and afterwards adheres to the stopping decisions $f^{\theta_{n+1}}, ..., f^{\theta_N}$, the realized reward is:

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

It follows that for sufficiently large M, $\mathbb{E}[g(n, X_n)f^{\theta}(X_n) + g(\tau_{n+1}, X_{\tau_{n+1}})(1 - f^{\theta}(X_n))]$ can be approximated by

$$\frac{1}{M}\sum_{m=1}^{M} r_n^m(\theta_n) \tag{21}$$

which is a smooth function with respect to θ a.e. by construction. Thus (21) acts as the function we wish to optimize via a gradient descent algorithm with respect to θ_n. We will employ the Adam method in our computational example, which we will later detail. Then, after obtaining the optimal θ_n, we then translate the resulting F^{θ_n} into 0-1 stopping decisions, f^{θ_n}, as described in the previous section.

After repeating this for $n = N-1, ..., 0$, we now generate a new set of sample paths $(y_n^m)_{n=0}^N$ for $m = 1, ..., M$. This will be our testing data. We now, backwards recursively for $n = N-1, ..., 0$, using the parameters $\{\theta_n\}$ we found using our training data, 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.

Once at time zero, we finish by computing the current optimal stopping time

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

which is an estimate of $\hat{\tau}$ from equation (20). The corresponding estimate of $\mathbb{E}g(\hat{\tau}, X_{\hat{\tau}})$ is given by

$$\hat{V} = \frac{1}{M}\sum_{m=1}^{M} g(\tilde{\tau}_m, y_{\tilde{\tau}_m}^m) \tag{22}$$

Note that $\hat{V}$ is a consistent estimator for $\mathbb{E}g(\hat{\tau}, X_{\hat{\tau}})$ by the law of large numbers. In addition, since it is just the standard Monte Carlo estimate for $\mathbb{E}g(\hat{\tau}, X_{\hat{\tau}})$, it is also unbiased. Furthermore, by the central limit theorem, a $1-\alpha$, $\alpha \in (0,1)$ confidence interval for $\mathbb{E}g(\hat{\tau}, X_{\hat{\tau}})$ 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 $\hat{\sigma}$ is the corresponding sample standard deviation which is given by

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

Remark. It is noted in [Becker et al., 2018] that although $\hat{V}$ is an unbiased estimate of $\mathbb{E}g(\hat{\tau}, X_{\hat{\tau}})$, it is not an unbiased estimate of $\sup_{\tau\in\mathcal{T}} \mathbb{E}g(\tau, X_\tau)$. However, they also state that in the examples they looked at, $\hat{V}$ yielded good results.

4 Examples

In the following section we look at some examples of the application of this procedure to solve the optimal stopping problem. The first we will look at is a Bermudan max-call option, which we have implemented in Python ourselves using the procedure discussed in [Becker et al., 2018]. The second example we will look at is the problem of optimally stopping a fractional Brownian motion. Note that we did not implement this method ourselves, so we will be using the computational results of [Becker et al., 2018] to discuss it.

4.1 Bermudan Max-Call Option

A Bermudan max-call option expiring at time T is an option written on d assets, $\{X^i\}_{i=1}^d$, that gives the holder the right, but not the obligation, to purchase one of the d assets at strike price K at any point on the time grid $0 = t_0 < t_1 < ... < t_N = T$.

We assume we are in a Black-Scholes market model, so asset prices follow a geometric Brownian motion:

$$X_t^i = (r - \delta_i)dt + \sigma_i dW_t^i \tag{23}$$

where

- x_0^i is the time 0 stock price
- $r \in [0, \infty)$ is the marker risk free return rate
- $\delta_i \in [0, \infty)$ is the dividend yield
- $\sigma_i \in [0, \infty)$ is the asset volatility
- W_t^i is the i^{th} component of a d-dimensional Brownian motion W. We assume instantaneous correlations of $\rho_{ij} = 0$ between the assets.

The corresponding payoff function of the option at time t is of the form $(\max_{i\in\{1,...,d\}} X_t^i - K)^+$ and hence its price is given by

$$\sup_\tau \mathbb{E}\left[e^{-r\tau}\left(\max_{i\in\{1,...,d\}} X_\tau^i - K\right)^+\right] \tag{24}$$

Now we can apply some simply notation transformations to write this option pricing problem as an optimal stopping problem of the form (1). Since the option can only be exercised at $t_0, t_1, ..., t_N$, we take and thus we can write (24) as $\sup_{\tau\in\mathcal{T}} \mathbb{E}g(\tau, X_\tau)$ where

$$g(n, x) = e^{-rt_n}\left(\max_{i\in\{1,...,d\}} x^i - K\right)^+ \tag{25}$$

We also assume an equidistant time grid, meaning $t_n = n \cdot T/N$ and thus $X_n \equiv X_{t_n}$ for $n = 0, ..., N$.

In our example, for $i = 1, .., d$ we take

$$x_0^i = 90, \qquad K = 100, \qquad \sigma_i = 20\%, \qquad \delta_i = 10\%, \qquad r = 5\%, \qquad T = 3, \qquad N = 9$$

Thus we simulate asset prices according to (23) as follows

$$x_{n,i}^m = x_{0,i} \cdot \exp\left\{\sum_{k=0}^n \left((r - \delta_i - \sigma_i^2/2)\Delta t + \sigma_i\sqrt{\Delta t} \cdot Z_{k,i}^m\right)\right\} \tag{26}$$

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

Note that the entirety of the code used implement this example is available in the appendix. In this section of the report we will just highlight some blocks of code including some key steps in order to properly explain how the example is implemented. To construct the neural network in Python, we use the `Pytorch` package. This allows us to construct the neural network corresponding to equation (9) as follows:

```
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
```

Note that we set $q_1 = q_2 = d + 40$, as per the reccomedation of [Becker et al., 2018]. Note that by building a neural network in `Pytorch`, we are also able to customize the loss function we wish to minimize, which in our case is the negative of (21). This is defined in Python as follows:

```
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())
```

Note that `s.g` in the above code chunk is simply the payoff function given in equation $g(n,x) = e^{-rt_n}\left(\max_{i\in\{1,\dots,d\}} x^i - K\right)^+$.

We also define a function in python, `NNtime`, to find the optimal parameters, θ_n, for the time n neural network, f^{θ_n}. In this function, the feed-forward, back propagation algorithm for selecting the optimal weights is illustrated. For each epoch - meaning for each presentation of the entire training dataset to the algorithm - we first compute F^{θ_n} using the θ_n set in the previous epoch. We then compute the gradient of the corresponding loss function via backpropagation and then update θ_n according to the Adam optimization algorithm. This is carried out in Python below:

```
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
```

As previously alluded to, the optimization algorithm we employ in this case is called the Adam Method, as detailed in [Kingma and Ba, 2014]. The updating scheme itself is detailed below in algorithm 1.

Algorithm 1: Adam Method

Input: α (step size), $r(\theta)$ (objective function), θ^0 (initial parameters)

1 Set $\beta_1 = 0.9; \beta_2 = 0.99; \epsilon = 10^{-8}$ (algorithm defaults)
2 $m_0 = v_0 = 0$
3 $t = 0$
4 **while** $r(\theta_t)$ *not minimized* **do**
5 $\quad t = t + 1$
6 $\quad G_t = \nabla_\theta r(\theta_{t-1})$ (obtained via backprop)
7 $\quad m_t = \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot G_t$ (1^{st} moment estimate)
8 $\quad v_t = \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot (G_t)^2$ (2^{nd} moment estimate)
9 $\quad \hat{m}_t = m_t / (1 - (\beta_1)^t)$ (bias correction)
10 $\quad \hat{v}_t = v_t / (1 - (\beta_2)^t)$ (bias correction)
11 $\quad \theta_t = \theta_{t-1} - \alpha \cdot \hat{m}_t / (\sqrt{\hat{v}_t + \epsilon}$

Output: θ^t (updated parameters)

Adam is shorthand for Adaptive Moment estimation. The adaptive part of the name refers to the algorithm performing smaller updates (meaning the learning rate is lower) for frequently occurring features and larger updates for infrequently occurring features. It does this by scaling the learning rate at each iteration based on all the past gradients that have been computed. This is illustrated by the form of the update condition in algorithm 1 in line 11. The moment portion of the name is evident in lines 7 and 8 of algorithm 1, as m_t and v_t can be regarded as estimates of the gradient's first and second moments.

Note that after computing the optimal θ_n at time step n, we proceed by using this to compute the corresponding optimal stopping time at the next step, as detailed in the previous section of the report. Once we have obtained the parameters for each time step, $\{\theta_n\}$, we then generate a new set of geometric Brownian motions as our testing data and use said $\{\theta_n\}$ to compute $\hat{V}$ in (22).

We ran this algorithm for $d = 2, 5$ and 10 underlying assets using $M = 5,000$ sample paths. Our estimates are given in table 1.

[1] Actual values are estimates from [Hu, 2019] for $d = 2$ and from [Becker et al., 2018] for $d = 5, 10$ each computed with $M = 160,000$ sample paths.

d	Actual[1]	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 were computed using $M = 5,000$ sample paths.

Note that our estimates in table 1 were computed using only $M = 5,000$ sample paths, as our code is not fully optimized in terms of runtime due to both the limited coding ability of myself and the limited computational abilities of my laptop. So for reference, the results of [Hu, 2019] (for $d = 2$) and [Becker et al., 2018] (for $d = 5$ and $d = 10$) are included, as they were computed using $M = 160,000$ sample paths. However, given we only used $M = 5,000$ sample asset price paths, our results are still fairly close to those of [Hu, 2019] and [Becker et al., 2018], which gives us some indication that our implementation is correct.

It is also worth noting that [Becker et al., 2018] included estimates for max-call options written on $d = 500$ assets and still boasted a computational time of only 533.5 seconds[2]. This highlights the computational efficiency and corresponding production of reasonable estimates that becomes feasible when employing a neural network approach to the optimal stopping problem.

4.2 Optimally Stopping a Fractional Brownian Motion

In our final example, we will discuss the work of [Becker et al., 2018] to employ our neural network algorithm to optimally stop a fractional Brownian motion. We will show how we can make a non-Markovian process Markovian by increasing the dimensionality of said process. Using traditional numerical methods, this increase in dimensionality would have made the problem computationally infeasible. However, by employing our neural network approach we are able to approximate the optimally stopped value of this more complex process.

First, we recall that a fractional Brownian motion is a generalization of a standard Brownian motion. Specifically, a fractional Brownian motion with Hurst Parameter H is a Gaussian process with mean zero and covariance structure

$$\mathbb{E}[W_t^H W_s^H] = \frac{1}{2}\left(t^{2H} + s^{2H} - |t-s|^{2H}\right) \tag{27}$$

Note that a standard Brownian motion is simply a fractional Brownian motion with Hurst parameter $H = 1/2$.

We want to estimate the value of an optimally stopped fractional Brownian motion between times 0 and 1. Meaning our objective is to find:

$$\sup_{0\leq\tau\leq 1} \mathbb{E}W_\tau^H \tag{28}$$

First off, recall the optional stopping theorem which states that if $X = (X_t)_{t\geq 0}$ be a martingale and τ an X-stopping time then if either τ or $X_{\tau\wedge t}$ is bounded then $\mathbb{E}X_\tau = \mathbb{E}X_0$. Thus since a standard Brownian motion is a martingale we have that $\sup_{0\leq\tau\leq 1}\mathbb{E}W_\tau^{1/2} = 0$. However for $H \neq 1/2$, W_t^H is

[2]For reference, it only took [Becker et al., 2018] 42.3 seconds to compute their estimate for the $d = 5$ case.

not a martingale, so the optional stopping theorem does not apply.

Thus to approximate (28), we first discretize the time interval $[0,1]$ into 100 time steps, $t_n = \frac{n}{100}$, $n = 0, ..., 100$. We can then create a 100-dimensional Markov process, $(X_n)_{n=0}^{100}$ to describe the $(W_{t_n}^H)_{n=0}^{100}$ as follows:

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

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

$$\sup_{\tau \in \mathcal{T}} \mathbb{E} g(X_\tau) \tag{29}$$

where τ is the set of all X-stopping times, in order to approximate (28).

To approximate (29), [Becker et al., 2018] simulated $(X_n)_{n=0}^{100}$ via the Davies Harte method, which involves 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

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

The sample paths of (X_n) are then computed via $W_{t_n}^H = \sum_{k=1}^n Y_k^H$. They then trained neural networks of the form (10) with $d = 100$, $q_1 = 110$ and $q_2 = 55$ as described in the previous sections and approximated (28) via a Monte Carlo approximation. The approximated optimal stopping solution computed by [Becker et al., 2018] is plotted in figure 3.

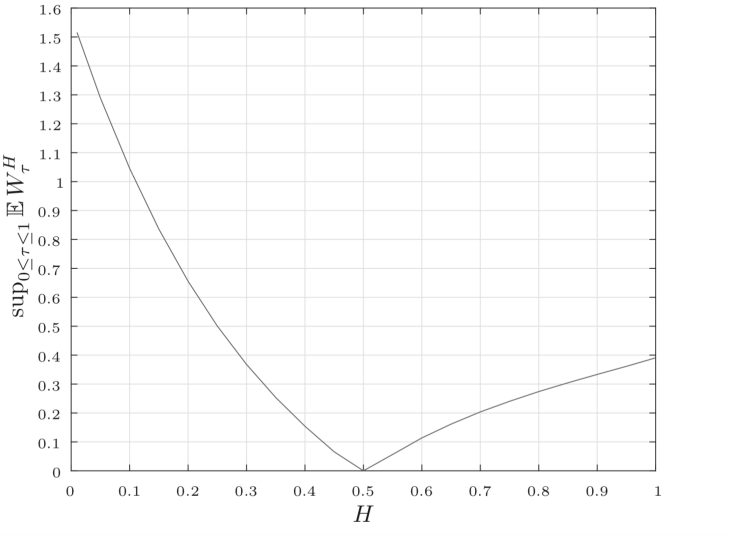


Figure 3: Results of [Becker et al., 2018]. Expected value of an optimally stopped fractional Brownian motion w.r.t. its Hurst parameter H.

Note that the shape of figure 3 makes sense intuitively as well since we expect to see a higher value for the optimally stopped W^H when $H < 1/2$ than for $H > 1/2$. To understand why, in the extreme

case of $H = 1$, the increments of W^H have autocovariance equal to 1, meaning a decrease in W^H at time t indicates a drop in its value at all times after t. Hence, the optimal stopping strategy for $H = 1$ would be to stop only if we see a drop in W^H. Conversely, H close to 0 implies a strong negative autocorrelation and thus indicates a market more prone to large swings. This presents more options in terms of an optimal strategy than the $H = 1$ case, since if the value of W^H falls at time t, it is likely to recover at time $t + s$ for some $s > 0$.

Furthermore, to illustrate the effectiveness of this method of optimally stopping a fractional Brownian motion [Becker et al., 2018] compares their results to [Kulikov and Gusyatnikov, 2016], which employs a traditional numerical discretization approach to approximate (28). The neural network approach yields results of up to five times higher than [Kulikov and Gusyatnikov, 2016] for $H < 1/2$ and up to three times higher for $H > 1/2$.

5 Concluding Remarks

In this report, we have forayed into how the optimal stopping problem can be solved by employing machine learning techniques, with an emphasis on the application of neural networks. As we have repeatedly discussed, the primary motivation for applying such techniques to the optimal stopping problem is that it is an effective cure to the so-called curse of dimensionality. No longer being fearful of optimal stopping problems involving a high dimensional Markov process can be liberating in a number of ways. As we have seen in this report, we are able to find the optimal exercise time for an option written on many assets and we were able to optimally stop a fractional Brownian motion by transforming it into a high-dimensional Markov process. In this sense, an extension of this project of interest would be to see what other types of optimal stopping problems this method can be employed to solve, as the framework we have set up is general enough to apply to any finite optimal stopping problem based on a discretized Markov process.

Our literature review brings up several interesting ideas for manners in which the topic of neural network applications to the optimal stopping problem can be further explored as well. The feed-forward network structure employed in [Becker et al., 2018] is among the simplest and oldest neural network architectures. Hence, the convolutional neural network approach proposed in [Hu, 2019] is intriguing, as it proposes applying a more computationally efficient network architecture to our same problem. However as we previously discussed, there is a lack of developed convergence theory for these architectures and hence they were unable to formally prove that their neural network offered accurate approximations of the optimal stopping solution, as we did in theorem 2. Hence, future research of interest could involve studying the convergence properties of convolutional neural network approximations. Note that the relevant convergence theory we would require for the optimal stopping problem in particular would be a convolutional neural network equivalent of the feed-forward network theorem by [Leshno et al., 1993] employed in the proof of 2.

In addition, [Becker et al., 2018] does not go into much detail about how to choose the number of nodes in the hidden layers, q_1 and q_2 in any of their examples. Hence, further research could involve looking into developing a mathematical framework to select q_1 and q_2 optimally in the context of this optimal stopping problem. It would also be worth looking at the process in which the number of layers employed is chosen, as again [Becker et al., 2018] offered little insight into their choice of a two hidden layer network.

Lastly, an inquiry that came to mind while working on this report is what other mathematical finance problems can perhaps be approximated in higher dimensions by employing a neural network approach. The optimal stopping problem is a classic mathematical finance problem that has been studied at length, yet a paper focused on the application of the simplest neural network structure to it was released in 2018. This could be indicative of a wealth of potential future research topics that involve approximating the solutions to high dimensional mathematical finance problems using neural networks and other machine learning solutions.

6 Appendix

6.1 Python Code

```
import numpy as np
import torch
import torch.nn as nn
np.random.seed(234198)

import scipy.stats

class stock:
    def __init__(self, T, K, sigma, delta, So, r, N, M, d):
        self.T = T
        self.K=K
        self.sigma=sigma *np.ones(d)
        self.delta=delta
        self.So=So*np.ones(d)
        self.r=r
        self.N=N
        self.M=M
        self.d=d

    def GBM(self):

        dt=self.T/self.N
        So_vec=self.So*np.ones((1,S.M, S.d))

        Z=np.random.standard_normal((self.N,self.M, self.d))
        s=self.So*np.exp(np.cumsum((self.r-self.delta-0.5*self.sigma**2)*dt+self.
                                                    sigma*np.sqrt(dt)*Z, axis=0))

        s=np.append(So_vec, s, axis=0)
        return s


    def g(self,n,m,X):
        max1=torch.max(X[int(n),m,:].float()-self.K)

        return np.exp(-self.r*(self.T/self.N)*n)*torch.max(max1,torch.tensor([0.0]))


#%%
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

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())

#%%

S=stock(3,100,0.2,0.1,90,0.05,9,5000,10)

X=torch.from_numpy(S.GBM()).float()
#%%

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

mods=[None]*S.N
tau_mat=np.zeros((S.N+1,S.M))
tau_mat[S.N,:]=S.N

f_mat=np.zeros((S.N+1,S.M))
f_mat[S.N,:]=1

#%%
for n in range(S.N-1,-1,-1):
    probs, mod_temp=NN(n, X, S,torch.from_numpy(tau_mat[n+1]).float())
    mods[n]=mod_temp
    np_probs=probs.detach().numpy().reshape(S.M)
    print(n, ":", np.min(np_probs)," , ", np.max(np_probs))

    f_mat[n,:]=(np_probs > 0.5)*1.0

    tau_mat[n,:]=np.argmax(f_mat, axis=0)

#%%
Y=torch.from_numpy(S.GBM()).float()

tau_mat_test=np.zeros((S.N+1,S.M))
tau_mat_test[S.N,:]=S.N

f_mat_test=np.zeros((S.N+1,S.M))
f_mat_test[S.N,:]=1

V_mat_test=np.zeros((S.N+1,S.M))
V_est_test=np.zeros(S.N+1)
```

```
for m in range(0,S.M):
    V_mat_test[S.N,m]=S.g(S.N,m,Y)

V_est_test[S.N]=np.mean(V_mat_test[S.N,:])



for n in range(S.N-1,-1,-1):
    mod_curr=mods[n]
    probs=mod_curr(Y[n])
    np_probs=probs.detach().numpy().reshape(S.M)

    f_mat_test[n,:]=(np_probs > 0.5)*1.0

    tau_mat_test[n,:]=np.argmax(f_mat_test, axis=0)


    for m in range(0,S.M):
        V_mat_test[n,m]=np.exp((n-tau_mat_test[n,m])*(-S.r*S.T/S.N))*S.g(tau_mat_test
                                                    [n,m],m,X)
        #np.exp((n-tau_mat_test[n,m])*(-S.r*S.T/S.N))*S.g(tau_mat_test[n,m],m,X)

#%%
V_est_test=np.mean(V_mat_test, axis=1)
V_std_test=np.std(V_mat_test, axis=1)
V_se_test=V_std_test/(np.sqrt(S.M))

z=scipy.stats.norm.ppf(0.975)
lower=V_est_test[0] - z*V_se_test[0]
upper=V_est_test[0] + z*V_se_test[0]

print(V_est_test[0])
print(V_se_test[0])
print(lower)
print(upper)
```

References

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

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

[Hu, 2019] Hu, R. (2019). Deep learning for ranking response surfaces with applications to optimal stopping problems. *arXiv e-prints*.

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

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

[Kulikov and Gusyatnikov, 2016] Kulikov, A. V. and Gusyatnikov, P. P. (2016). Stopping times for a fractional brownian motion. In *Computational Management Science*, pages 195–200. Springer International Publishing.

[Leshno et al., 1993] 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.

[Longstaff and Schwartz, 2001] Longstaff, F. A. and Schwartz, E. S. (2001). Valuing american options by simulation: A simple least-squares approach. *Review of Financial Studies*, pages 113–147.

[Yu and Bertsekas, 2007] 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.