Optimal Stopping of exotic American option by Deep Learning

Qinghao Lin

August 2020

Abstract

Our REU project mainly focuses on deep learning method for optimal stopping problems which directly learns the optimal stopping rule from Monte Carlo samples. The introduction to the theory, mathematical proof and appliance of Deep Learning will be discussed. We test the approach on the pricing of a Bermudan max-call option and Asian option. Also, deep learning on Longstaff Schwartz algorithm's regression bases will be mentioned.

1 Introduction

An optimal stopping problem uses 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 $X = (X_n)_{n=0}^N$ be an R^d-valued discrete-time Markov process on a probability space $(\Omega, \mathcal{F}, P)$, where N and d are positive integers. We denote by $\mathcal{F}_n$ the σ-algebra generated by $X_0, X_1, \ldots, X_n$ and call a random variable $\tau \colon \Omega \to 0, 1, \ldots, N$ an X-stopping time if the event $\tau = n$ belongs to $\mathcal{F}_n$ for all $n \in \{0, 1, \ldots, N\}$. In this context, mathematically we say is an X-stopping time if = n F, n 0, 1, ..., N. The notion of reward we look to maximize is given by V = $\sup_{\tau \in \mathcal{T}} E\, g(\tau, X_\tau)$,

Theoretically optimal stopping problems with finite 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 reward from stopping exceeds the expected reward from 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 dimensions of Markov process X is large. This limits the types of optimization problems that can be solved via traditional methods. 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.
The whole project is mainly based on Dr. Patrick Cheridito's work [1] and the discussion on the second method using the Longstaff-Schwartz method as well with [2] and [3].

2 Background

2.1 Deep Learning and Neural Network

Deep learning is an artificial intelligence function that imitates the workings of the human brain in processing data and creating patterns for use in decision making. Deep learning is a subset of machine learning in artificial intelligence (AI) that has networks capable of learning unsupervised from data that is unstructured or unlabeled. The specific strategy that we use in modeling is neural network.

A neural network is a computational system modeled 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.

To apply neural network, we need to functionalize our problem into a mathematical format which is easy to demonstrate and model. So we may rewrite the original statement of the question to a decision to exercise or keep the option: 0 or 1

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

2.2 Expressing Stopping Times as a Series of Stopping Decisions

The first method is to model the stopping decision directly. An intuitive idea to transform the X-stopping problem is to set the stopping decision as a sequence of binary digits in order to meet the above criteria. In principle, the decision whether to stop the process at time n if it has not been stopped before, can be made based on the whole evolution of X from time 0 until n. But to optimally stop the Markov process X, it is enough to make stopping decisions according to $f_n(X_n)$ for measurable functions $f_n \colon R^d \to 0, 1$, $n = 0, 1, \ldots, N$.

Consider the auxiliary stopping problems

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

for $n = 0, 1, \ldots, N$, where $\mathcal{T}_n$ is the set of all X-stopping times satisfying $n \leq \tau \leq N$. Obviously, $\mathcal{T}_N$ consists of the unique element $\tau_N \equiv N$, and one can

write $\tau_N = N f_N(X_N)$ for the constant function $f_N \equiv 1$. Moreover, for given $n \in 0, 1, \ldots, N$ and a sequence of measurable functions $f_n, f_{n+1}, \ldots, f_N \colon R^d \to 0, 1$ with $f_N \equiv 1$,

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

2.3 Neural Network Approximation

Now as we have already got a a sequence of measurable functions, it becomes an intuitive idea to to approximate $f_n \colon R^d \to 0, 1$, $n = 0, 1, \ldots, N-1$, by a neural network $f^\theta \colon R^d \to 0, 1$ with parameter $\theta \in R^q$. We do this by starting with the terminal stopping decision $f_N \equiv 1$ and proceeding by backward induction. More precisely, let $n \in 0, 1, \ldots, N-1$, and assume parameter values $\theta_{n+1}, \theta_{n+2}, \ldots, \theta_N \in R^q$ have been found such that $f^{\theta_N} \equiv 1$ and the stopping time

$$\tau_{n+1} = \sum_{m=n+1}^{N} m f^{\theta_m}(X_m) \prod_{j=n+1}^{m-1} (1 - f^{\theta_j}(X_j))$$

produces an expected value $E\, g(\tau_{n+1}, X_{\tau_{n+1}})$ close to the optimum V_{n+1}. Since f^θ takes values in $0, 1$, it does not directly lend itself to a gradient-based optimization method. So, as an intermediate step, we introduce a feedforward neural network $F^\theta \colon R^d \to (0, 1)$ of the form

$$F^\theta = \psi \circ a_I^\theta \circ \varphi_{q_{I-1}} \circ a_{I-1}^\theta \circ \cdots \circ \varphi_{q_1} \circ a_1^\theta,$$

where

- $I, q_1, q_2, \ldots, q_{I-1}$ are positive integers specifying the depth of the network and the number of nodes in the hidden layers (if there are any),
- $a_1^\theta \colon R^d \to R^{q_1}, \ldots, a_{I-1}^\theta \colon R^{q_{I-2}} \to R^{q_{I-1}}$ and $a_I^\theta \colon R^{q_{I-1}} \to R$ are affine functions,
- for $j \in N$, $\varphi_j \colon R^j \to R^j$ is the component-wise ReLU activation function given by $\varphi_j(x_1, \ldots, x_j) = (x_1^+, \ldots, x_j^+)$
- $\psi \colon R \to (0, 1)$ is the standard logistic function $\psi(x) = e^x/(1 + e^x) = 1/(1 + e^{-x})$.

2.4 Longstaff-Schwartz simulation

There is another way to carry out the estimation by applying Longstaff-Schwartz method to estimate the decision. Due to the size of the project, we focus on the first method discussed above but Longstaff -Schwartz method is a popular and widely-used strategy in derivatives pricing[4]. This approach uses a regression

technique to approximate the continuation value of the option. A comparison is made between a polynomial and spline basis to fit the regression.
One can apply the following variant in the use of neural networks instead of linear combinations of basis functions. In addition, the sum in the equation is over all simulated paths, only in-the-money paths are considered to save computational effort. While it is enough to use in-the-money paths to determine a candidate optimal stopping rule, we need accurate approximate continuation values for all $x \in^d$ to construct good hedging strategies of the Longstaff–Schwartz algorithm:

Simulate[1] paths $(x_n^k)_{n=0}^N$, $k = 1, \ldots, K$, of the underlying process $(X_n)_{n=0}^N$.

Set $s_N^k \equiv N$ for all k.

For $1 \leq n \leq N-1$, approximate $EG_{\tau_{n+1}} \mid X_n$ with $c^{\theta_n}(X_n)$ by minimizing the sum $\sum_{k=1}^K g(s_{n+1}^k, x_{s_{n+1}^k}^k) - c^\theta(x_n^k)^2$ over θ. $Sets_n^k := \begin{cases} n & \text{if } g(n, x_n^k) \geq c^{\theta_n}(x_n^k) \\ s_{n+1}^k & \text{otherwise.} \end{cases}$

Define $\theta_0 := \frac{1}{K} \sum_{k=1}^K g(s_1^k, x_{s_1^k}^k)$, and set c^{θ_0} constantly equal to θ_0.

In this paper we specify c^θ as a feedforward neural network, which in general, is of the form

$$a_I^\theta \circ \varphi_{q_{I-1}} \circ a_{I-1}^\theta \circ \cdots \circ \varphi_{q_1} \circ a_1^\theta,$$

where

- $I \geq 1$ denotes the depth and $q_0, q_1, \ldots, q_I$ the numbers of nodes in the different layers
- $a_1^\theta \colon R^{q_0} \to R^{q_1}, \ldots, a_I^\theta \colon R^{q_{I-1}} \to R^{q_I}$ are affine functions,
- For $j \in N$, $\varphi_j \colon R^j \to R^j$ is of the form $\varphi_j(x_1, \ldots, x_j) = (\varphi(x_1), \ldots, \varphi(x_j))$ for a given activation function $\varphi \colon \to$.

The components of the parameter θ consist of the entries of the matrices $A_1, \ldots, A_I$ and vectors $b_1, \ldots, b_I$ appearing in the representation of the affine functions $a_i^\theta x = A_i x + b_i$, $i = 1, \ldots, I$. So, θ lives in ^{q} for $q = \sum_{i=1}^I q_i(q_{i-1}+1)$. To minimize the loss, we choose a network with $q_I = 1$ and employ a stochastic gradient descent method.

3 Mathematical Framework

Here are two main mathematical problems that we need to solve on the path from the set-up to our network model. Firstly, we need to show the reason why we could use the format of 0-1 function to represent the optimal stopping. Secondly, the effectiveness and correctness of our model above have to be shown.

[1] As usual, we simulate the paths (x_n^k), $k = 1, \ldots, K$, independently of each other.

3.1 Proof of optimal choice

We need to show that the optimal stopping time can be computed as a function of the series of 0-1 stopping decisions before applying Neural Network. So we need to show that the function above describing the stopping time defines the optimal one. The following theorem proves this:

Theorem 1 *For a given $n \in 0, 1, \ldots, N-1$, let τ_{n+1} be a stopping time in $\mathcal{T}_{n+1}$ of the form*

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

for measurable functions $f_{n+1}, \ldots, f_N \colon R^d \to 0, 1$ with $f_N \equiv 1$. Then there exists a measurable function $f_n \colon R^d \to 0, 1$ such that the stopping time $\tau_n \in \mathcal{T}_n$ satisfies

$$E\, g(\tau_n, X_{\tau_n}) \geq V_n - \left(V_{n+1} - E\, g(\tau_{n+1}, X_{\tau_{n+1}})\right),$$

where V_n and V_{n+1} are the optimal values.

Proof We begin by setting an arbitrary stopping time $\tau \in \mathcal{T}_n$ and letting $\varepsilon = V_{n+1} - E\, g(\tau_{n+1}, X_{\tau_{n+1}})$, By the Doob–Dynkin lemma there exists a measurable function $h_n \colon R^d \to R$ such that

$$\mathrm{h}_n(X_n) = Eg(\tau_{n+1}, X_{\tau_{n+1}}) \mid X_n.$$

Now as we previously mentioned, there is another version of is a measurable function of $X_{n+1}, \ldots, X_N$.

$$g(\tau_{n+1}, X_{\tau_{n+1}}) = \sum_{m=n+1}^{N} g(m, X_m) 1_{\tau_{n+1}=m} = \sum_{m=n+1}^{N} g(m, X_m) 1_{f_m(X_m) \prod_{j=n+1}^{m-1}(1-f_j(X_j))=1}$$

So it follows from the Markov property of X that

$$\mathrm{h}_n(X_n) = Eg(\tau_{n+1}, X_{\tau_{n+1}}) \mid \mathcal{F}_n$$

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

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

$\tau_n = n1_D + \tau_{n+1}1_{D^c}$ belongs to $\mathcal{T}_n$ and $\tilde{\tau} = \tau_{n+1}1_E + \tau 1_{E^c}$ to $\mathcal{T}_{n+1}$. It follows from the definitions of V_{n+1} and ε that $E\, g(\tau_{n+1}, X_{\tau_{n+1}}) = V_{n+1} - \varepsilon \geq E\, g(\tilde{\tau}, X_{\tilde{\tau}}) - \varepsilon$. Hence,

$$Eg(\tau_{n+1}, X_{\tau_{n+1}})1_{E^c} \geq Eg(\tilde{\tau}, X_{\tilde{\tau}})1_{E^c} - \varepsilon = Eg(\tau, X_\tau)1_{E^c} - \varepsilon,$$

from which one obtains

$$E\,g(\tau_n, X_{\tau_n}) = Eg(n, X_n)I_D + g(\tau_{n+1}, X_{\tau_{n+1}})I_{D^c} = Eg(n, X_n)I_D + h_n(X_n)I_{D^c}$$
$$\geq Eg(n, X_n)I_E + h_n(X_n)I_{E^c} = Eg(n, X_n)I_E + g(\tau_{n+1}, X_{\tau_{n+1}})I_{E^c}$$
$$\geq Eg(n, X_n)I_E + g(\tau, X_\tau)I_{E^c} - \varepsilon = E\,g(\tau, X_\tau) - \varepsilon.$$

Since $\tau \in \mathcal{T}_n$ was arbitrary, this shows that $E\,g(\tau_n, X_{\tau_n}) \geq V_n - \varepsilon$. Lastly, we show that the n we found here is the same as previously declared, for the function $f_n \colon R^d \to 0, 1$ given by

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

Therefore,

$$\tau_n = nf_n(X_n) + \tau_{n+1}(1 - f_n(X_n)) = \sum_{m=n}^{N} mf_m(X_m) \prod_{j=n}^{m-1} (1 - f_j(X_j)),$$

as required.

3.2 Proof of model effectiveness

Let $n \in 0, 1, \ldots, N-1$ and fix a stopping time $\tau_{n+1} \in \mathcal{T}_{n+1}$. Then, for every depth $I \geq 2$ and constant $\varepsilon > 0$, there exist positive integers $q_1, \ldots, q_{I-1}$ such that

$$\sup_{\theta \in R^q} Eg(n, X_n)f^\theta(X_n) + g(\tau_{n+1}, X_{\tau_{n+1}})(1 - f^\theta(X_n))$$
$$\geq \sup_{f \in \mathcal{D}} Eg(n, X_n)f(X_n) + g(\tau_{n+1}, X_{\tau_{n+1}})(1 - f(X_n)) - \varepsilon,$$

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

proof Fix $\varepsilon > 0$. It follows from the integrability condition (**??**) that there exists a measurable function $\tilde{f} \colon R^d \to 0, 1$ such that

$$Eg(n, X_n)\tilde{f}(X_n) + g(\tau_{n+1}, X_{\tau_{n+1}})(1 - \tilde{f}(X_n))$$
$$\geq \sup_{f \in \mathcal{D}} Eg(n, X_n)f(X_n) + g(\tau_{n+1}, X_{\tau_{n+1}})(1 - f(X_n)) - \varepsilon/4.$$

$\tilde{f}$ can be written as $\tilde{f} = 1_A$ for the Borel set $A = \{x \in R^d : \tilde{f}(x) = 1\}$. Moreover,

$$B \mapsto E|g(n, X_n)|1_B(X_n) \quad \text{and} \quad B \mapsto E|g(\tau_{n+1}, X_{\tau_{n+1}})|1_B(X_n)$$

define finite Borel measures on R^d. Since every finite Borel measure on R^d is tight (see e.g., [1]), there exists a compact (possibly empty) subset $K \subseteq A$ such that

$$[Eg(n, X_n)1_K(X_n) + g(\tau_{n+1}, X_{\tau_{n+1}})(1 - 1_K(X_n))$$

$$\geq Eg(n, X_n)\tilde{f}(X_n) + g(\tau_{n+1}, X_{\tau_{n+1}})(1 - \tilde{f}(X_n)) - \varepsilon/4.$$

Let $\rho_K \colon R^d \to [0, \infty]$ be the distance function given by $\rho_K(x) = \inf_{y \in K} \|x - y\|_2$. Then

$$k_j(x) = \max 1 - j\rho_K(x), -1, \quad j \in N,$$

defines a sequence of continuous functions $k_j \colon R^d \to [-1, 1]$ that converge pointwise to $1_K - 1_{K^c}$. So it follows from Lebesgue's dominated convergence theorem that there exists a $j \in N$ such that

$$\begin{aligned} &Eg(n, X_n)\, 1_{k_j(X_n) \geq 0} + g(\tau_{n+1}, X_{\tau_{n+1}})(1 - 1_{k_j(X_n) \geq 0}) \\ &\geq Eg(n, X_n) 1_K(X_n) + g(\tau_{n+1}, X_{\tau_{n+1}})(1 - 1_K(X_n)) - \varepsilon/4. \end{aligned}$$

By Theorem 1,k_j can be approximated uniformly on compacts by functions of the form

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

for $r, s \in N$, $v_1, \ldots, v_r, w_1, \ldots, w_s \in R^d$ and $c_1, \ldots, c_r, d_1, \ldots, d_s \in R$. So there exists a function $h \colon R^d \to R$ expressible as in (**??**) such that

$$\begin{aligned} &Eg(n, X_n)\, 1_{h(X_n) \geq 0} + g(\tau_{n+1}, X_{\tau_{n+1}})(1 - 1_{h(X_n) \geq 0}) \\ &\geq Eg(n, X_n)\, 1_{k_j(X_n) \geq 0} + g(\tau_{n+1}, X_{\tau_{n+1}})(1 - 1_{k_j(X_n) \geq 0}) - \varepsilon/4. \end{aligned}$$

Now note that for any integer $I \geq 2$, the composite mapping $1_{[0,\infty)} \circ h$ can be written as a neural net f^θ of the form (4) with depth I for suitable integers $q_1, \ldots, q_{I-1}$ and parameter value $\theta \in R^q$. Hence,we get

$$\begin{aligned} &Eg(n, X_n)\, f^\theta(X_n) + g(\tau_{n+1}, X_{\tau_{n+1}})(1 - f^\theta(X_n)) \\ &\geq \sup_{f \in \mathcal{D}} Eg(n, X_n) f(X_n) + g(\tau_{n+1}, X_{\tau_{n+1}})(1 - f(X_n)) - \varepsilon, \end{aligned}$$

and the proof is complete.

4 Neural Network Model

Now with our proof above and Neural Network for the problem, we are able to calculate the estimation of key statistics of our estimated decision. Our numerical method for problem consists in iteratively approximating optimal stopping to simulate the decisions $f_n \colon R^d \to 0, 1,\ n = 0, 1, \ldots, N-1$, by a neural network $f^\theta \colon R^d \to 0, 1$ with parameter $\theta \in R^q$.

$$\tau_{n+1} = \sum_{m=n+1}^{N} m f^{\theta_m}(X_m) \prod_{j=n+1}^{m-1} (1 - f^{\theta_j}(X_j))$$

Based on the equation, we introduce a feed-forward neural network $F^\theta : R^d \to (0,1)$ of the form

$$F^\theta = \psi \circ a_I^\theta \circ \varphi_{q_{I-1}} \circ a_{I-1}^\theta \circ \cdots \circ \varphi_{q_1} \circ a_1^\theta,$$

and calculate the targeted estimator, $f^{\theta_n} : R^d \to 0,1$ by $\mathrm{f}^{\theta_n} = 1_{[0,\infty)} \circ a_I^{\theta_n} \circ \varphi_{q_{I-1}} \circ a_{I-1}^{\theta_n} \circ \cdots \circ \varphi_{q_1} \circ a_1^{\theta_n}, where 1_{[0,\infty)} : R \to 0,1$ is the indicator function of $[0,\infty)$.

4.1 Geometric Brownian motion simulation

Geometric Brownian Motion (GBM), which is technically a Markov process, meaning that the stock price follows a random walk and that is one of the fundamental hypotheses of Black-Sholes Model, on which we build the whole model. So it is essential and necessary to simulate the path of GBM, which will be used in training and testing. The formula for GBM is found below:

$$\frac{\Delta S}{S} = \mu \Delta t + \sigma \epsilon \sqrt{\Delta t}$$

$where:$

$S =$ the stock price

$\Delta S =$ the change in stock price

$\mu =$ the expected return

$\sigma =$ the standard deviation of returns

$\epsilon =$ the random variable

$\Delta t =$ the elapsed time period

To simulate a path of a Brownian motion in a forward way, we can use the independent and stationary Gaussian increments property, or apply Brownian bridge in a backward simulation. a Brownian bridge X is a continuous Gaussian process with $X_0 = X_1 = 0$, and with mean of

$$E(X_t) = 0 \ for \ t \in [0,1]$$

and covariance of

$$cov(X_s, X_t) = min\{s,t\} - st \ for \ s,t \in [0,1]$$

And then we may use Cholesky's method to extend the simulation from one dimension to n-dimension. The details are argued in [5]. As this is not the major question that we deal with, the rigid proof is omitted but we may adopt NumPy easily in real simulation as follows.

```
def stock_sim_path(S, alpha, delta, sigma, T, N, n):
    """Simulates geometric Brownian motion."""
    h = T/n
    # uncomment below for deterministic trend. or, can pass it in
                                        as alpha as an array
```

```
    alpha = alpha # + np.linspace(0, 0.1, 500).reshape((n,N))
    mean = (alpha - delta - .5*sigma**2)*h
    vol = sigma * h**.5
    return S*np.exp((mean + vol*np.random.randn(n,N)).cumsum(axis =
                                                    0))
```

4.2 Lower bound

To estimate the lower bound of our decision, we simulate a new set of independent realizations $(y_n^k)_{n=0}^N$, $k = 1, 2, \ldots, K_L$, of $(X_n)_{n=0}^N$. τ^Θ is of the form $\tau^\Theta = l(X_0, \ldots, X_{N-1})$ for a measurable function $l\colon R^{dN} \to 0, 1, \ldots, N$. Denote $l^k = l(y_0^k, \ldots, y_{N-1}^k)$. The Monte Carlo approximation

$$\hat{L} = \frac{1}{K_L} \sum_{k=1}^{K_L} g(l^k, y_{l^k}^k)$$

gives an unbiased estimate of the lower bound L, and by the law of large numbers, $\hat{L}$ converges to L for $K_L \to \infty$.

4.3 Upper bound

Estimation of upper bound is much more complicated and requires for multiple paths of geometric Brownian Motion. Here we may omit the rigorous proof and estimate the upper bound as

$$U = E \max_{0 \le n \le N} g(n, X_n) - M_n^\Theta - \varepsilon_n,$$

4.4 Point estimate and confidence intervals

Our point estimate of V_0 is the average

$$\frac{\hat{L} + \hat{U}}{2}.$$

To derive confidence intervals, we assume that $g(n, X_n)$ is square-integrable for all n. Then

$$g(\tau^\theta, X_{\tau^\Theta}) \quad \text{and} \quad \max_{0 \le n \le N} g(n, X_n) - M_n^\Theta - \varepsilon_n$$

are square-integrable too. Hence, one obtains from the central limit theorem that for large K_L, $\hat{L}$ is approximately normally distributed with mean L and variance $\hat{\sigma}_L^2 / K_L$ for

$$\hat{\sigma}_L^2 = \frac{1}{K_L - 1} \sum_{k=1}^{K_L} g(l^k, y_{l^k}^k) - \hat{L}^2.$$

So, every $\alpha \in (0, 1]$,

$$\left[\hat{L} - z_{\alpha/2}\frac{\hat{\sigma}_L}{\sqrt{K}_L}\,,\, \hat{U} + z_{\alpha/2}\frac{\hat{\sigma}_U}{\sqrt{K}_U}\right]$$

is an asymptotically valid $1 - \alpha$ confidence interval for V_0.

5 Example and Extension

I will explore two examples based on Neural Network model on American-style option. The first one is the traditional American option stopping strategy, which is discussed in the paper without simulation code. I will share the crucial part of my implementation, and I will extend this implementation to another popular and widely traded option, Asian option, by constructing a new function of option payoff.

5.1 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:

$$X_t^i = (r - \gamma_i)dt + \sigma_i dW_i^{ti}$$

where

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

The corresponding payoff function of the option at time t is of the form and hence its price is given by

$$\sup_\tau E\left[e^{-r\tau}\left(\max_{1\le i\le d} S_\tau^i - K\right)^+\right],$$

where the supremum is over all S-stopping times taking values in $t_0, t_1, \ldots, t_N$ Denote $X_n^i = S_{t_n}^i$, $n = 0, 1, \ldots, N$, and let $\mathcal{T}$ be the set of X-stopping times. Then the price can be written as $\sup_{\tau\in\mathcal{T}} E\, g(\tau, X_\tau)$ for

$$g(n, x) = e^{-rt_n}\max_{1\le i\le d} x^i - K^+,$$

and it is straight-forward to simulate $(X_n)_{n=0}^N$.

We also assume an equidistant time grid, meaning $t_n = nT/N$, $n = 0, 1, \ldots, N$, for a maturity $T > 0$ and $N + 1$ equidistant exercise dates. In our example, for $i = 1, .., d$ we take

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

Thus we simulate asset prices according to Black-Sholes Model

$$x_m^{i,k,j} = x_n^{0,i} \exp\left[r - \delta_i - \sigma_i^2/2\right](m-n)\Delta t + \sigma_i[v_{n+1}^{i,k,j} + \cdots + v_m^{i,k,j}]$$

In this section of the report we will 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

```
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
```

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 the loss expectation in proof 2, $E[g(n, Xn)f^\theta(Xn) + g(\tau_n + 1, X_{\tau_n+1})(1 - f^\theta(Xn))]$.

```
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())
```

We also define a function in python, NNtime, to find the optimal parameters, n, for the time n neural network, $f^{\theta n}$. 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 $F^{\theta n}$ using the θn set in the previous epoch.

d	x_0	Std. Error	Point est.	95% Confidence Interval
2	90	0.23	7.49	$[7.26, 7.72]$
3	90	0.34	11.10	$[10.75, 11.45]$
5	90	0.32	16.81	$[16.41, 17.12]$
10	90	0.29	23.80	$[23.49, 24.11]$
20	90	0.4	37.25	$[36.83, 37.65]$

Table 1: Bermudan Max-call option Simulation

```
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
```

The result is shown below.

The result is shown in the table above.

5.2 Asian Max-call Option

An Asian option (or average value option) is a special type of option contract. For Asian options the payoff is determined by the average underlying price over some pre-set period of time. This is different from the case of the usual European option and American option, where the payoff of the option contract depends on the price of the underlying instrument at exercise; Asian options are thus one of the basic forms of exotic options. There are two types of Asian options: fixed strike, where averaging price is used in place of underlying price; and fixed price, where averaging price is used in place of strike. Fixed strike (also known as an average rate) Asian call payout

$$C(T) = \max\left(A(0,T) - K, 0\right)$$

where A(0,T) represents the average of underlying price from time 0 to T.

In addition, as our model is not able to deal with time-continuous situation, we used the same model dealing with Bermudan call option with 9 stopping points. The only difference between Bermudan call option and Asian option now is the payoff function, intuitively that is the only part of code needs to be replaced.

However, a seperate and independent variable needs to be set to store the stock price before we can calculate the average. This variable is essential in both

d	x_0	Std. Error	Point est.	95% Confidence Interval
2	90	0.49	6.23	[6.21, 6.24]
3	90	0.44	10.20	[10.18, 10.21]
5	90	0.31	13.72	[13.71, 13.73]
10	90	0.46	18.95	[18.93, 18.99]
20	90	0.58	31.29	[31.25, 33.32]

Table 2: Asian call option Simulation

training and testing, so we make a bottom modification on the class of stock, the payoff function to be specific. The original payoff function is

```
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]))
```

Now we add a new variable a to store the stock price and calculate the average before carrying out the maximum in the loss function. We rewrite the stock class as following:

```
class stock:
    def __init__(self, T, K, sigma, delta, So, r, N, M, d):
        self.T = T                          # ending time step
        self.K=K                            # price purchased
        self.sigma=sigma *np.ones(d)        # asset volatility
        self.delta=delta                    # dividend yield
        self.So=So*np.ones(d)               # price at time 0
        self.r=r                            # marker risk free return
        self.N=N                            # number of time steps
        self.M=M                            # number of sample paths
        self.d=d                            # number of assets
        self.a=a                            # value of stock
        self.i=0                            # number of stopping

    def g(self,n,m,X):
        self.a = self.a + X
        self.i = self.i + 1
        Y = self.a/self.i

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

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

The result is shown as the above table.

References

[1] Becker S, Cheridito P, Jentzen A. Deep optimal stopping[J]. Journal of Machine Learning Research, 2019, 20: 74.

[2] Becker S, Cheridito P, Jentzen A. Pricing and hedging American-style options with deep learning[J]. Journal of Risk and Financial Management, 2020, 13(7): 158.

[3] Cheridito P, Jentzen A, Rossmannek F. Efficient approximation of high-dimensional functions with deep neural networks[J]. arXiv preprint arXiv:1912.04310, 2019. 1222–1234.

[4] Longstaff F A, Schwartz E S. Valuing American options by simulation: a simple least-squares approach[J]. The review of financial studies, 2001, 14(1): 113-147.

[5] Karl S, Simulating Brownian motion and geometric Brownian motion