Deep Neural Networks for American Options

Sahenjit Paul

Supervisor: Dr Linus Wunderlich

September 2019

Contents

1 Declaration

I hereby declare that the thesis entitled "Deep Neural Network for American Options" submitted to QMplus is a record of original work done by me under the guidance of Dr Linus Wunderlich. This project work is submitted to fulfil the requirements for the Msc Financial Computing degree run by the School of Mathematical Sciences. The results of this thesis have not been submitted to any other University or institution for the award of any degree of diploma.

2 Abstract

An American option allows the holder (buyer) of the derivative contract the right but not the obligation to exercise the contract at any time during and up-to the lifetime of the option (also known as the expiry date). This extra stipulation creates several layers of difficulty when pricing an American option as there is no complete analytical solution unlike its counterpart the European option.
The aim of this dissertation is to study the properties of American options, numerically, using the Black Scholes Partial Differential Equation. In particular, we will look at classical methodologies such as Finite Difference Methods[4] and modern methodologies like Deep Neural Networks[5] for purposes of pricing the fair value of American options and Multi-Dimensional American options.
To evaluate the accuracy of the Network for the one dimensional case we use Finite Difference Methods as a ground-truth. We analyse the Network's error against the ground truth by evaluating the violation on the free boundary error term (L2) and by increasing the amount of random samples used to evaluate the Network.
Furthermore, to evaluate the price of an American option in a 2-dimensional case we implement the necessary adjustments into the Black Scholes PDE i.e. stock price correlations. To compare the results against some ground truth we can use the Longstaff Schwartz Method[3] or increase the Finite Difference to two dimensions.
Finally, we will discuss the positive and negative effects of Neural Networks and ultimately come to a conclusion suggesting why they should be used.

3 Motivation

Equity Derivatives(Options) are one of the largest traded securities currently in the world. According to Bloomberg, some banks such as JP Morgan have generated roughly generated $ 1.5 billion in revenue by buying and selling these derivatives.[2]
The concept of an Equity Derivative(Options) is so simple yet so complicated. Fundamentally, two counterparties create a legal agreement to buy or sell a certain underlying asset on an agreed future date for a pre - agreed price. The counterparty selling this contract i.e. seller, usually gets paid an upfront premium so that he has some incentive to enter into this agreement.
However, many different scenario's arise from the given idea above. Some scenario's can be:
Scenario 1. How do we price the premium for a derivative?

Scenario 2. What happens if extra stipulations are added to the contract?

Scenario 3. What are the risk management factors that have to be taken into consideration?

The three problems above can be applied to a certain type of derivative known as American options. An American option changes the option's policies such that it can be exercised at any time up to and including a fixed future date. Thus, making it difficult for Scenario 1 to occur due to optimal stopping times.[13] To tackle this, different highly computational methods have be created such as Finite Difference Methods[4] and Monte Carlo Methods[3], while these methods are acceptable they have flaws. For example: Finite Difference Methods become computationally intractable in higher dimensions.[5] This means if we wanted to price an American basket option where the price of the derivative is valued using many underlying stocks(dimensions) we wouldn't be able to given the current method.
Due to a dynamic change in high performance computing and optimization algorithms Deep Neural Networks have taken the center stage for a wide variety of highly complex tasks. Ergo, an idea was proposed, to apply these Deep Neural Networks which use high performance computing and optimization algorithms to price American options. Essentially, we are trying to price derivatives in a way which minimizes the time taken and gives the least error possible.

For risk management purposes this is pivotal since due to the 2008 financial crisis, banks have become highly regulated by financial authorities such as: FCA and Basil Regulations. This lead to banks being more cautious with their liquid assets i.e. cash reserves, leading to higher risk valuation analysis on their portfolio through measures like Value-At-Risk(VAR).
For example: When a counterparty enters into a call option agreement with a bank, there is a huge risk involved since the underlying asset could rapidly jump to a very high value, to combat this risk we could evaluate the option's value at different scenario's i.e. time steps, using the Deep Gelarkin Method.

4 Literature Review

Obtaining the fair value for Options is a very complicated and evolving area of finance. This is because we are trying to set an option premium's while simultaneously minimizing our overall risk exposure. In 1973, Fisher Black and Myron Scholes produced an article called "The Pricing of Options and Corporate Liabilities"[1] that revolutionized the banking industry. The article consists of thorough mathematical explanations and derivations for pricing Options through a generalized yet sophisticated equation known as the Black Scholes Partial Differential Equation. The concept behind this equation was to hedge our position by buying or selling the underlying asset in just the right way so our position is risk free.[12]

The Black Scholes Partial Differential Equation governs the price evolution of an option under the Black Scholes Model. The PDE has no analytical solution certain types of 'non-vanilla options' therefore numerical solutions through heavy computation have been created to price these 'exotic options'. Methods which yielded results for pricing exotic derivatives are called Finite Difference Methods[4] and Monte Carlo simulations[3]. Finite Difference Methods evaluate the Black Scholes PDE by iteratively solving the PDE on a finite mesh grid. This method is commonly used in modern day practices by large investment banks[33] but fails in higher dimensions due to the limitations of a mesh grid.

However, in recent years a new method was discovered for evaluating the Black Scholes PDE through the use of Deep Neural Networks[17].

Recently Deep Learning has revolutionized areas such as speech[31],text[32] and image[34] recognition through the use of Non Linear functions and Back-Propagation. Combining this with Gelarkin Methods, we get the Deep Gelarkin Method. This method takes an alternative approach by solving the free boundary PDE on a 'mesh-free' grid. This 'mesh-free' approach goes far beyond the capabilities of Finite Difference Methods as it is able to solve the Black Scholes PDE in higher dimensions. Furthermore, "the Deep learning algorithm incorporates new computational schemes for the efficient computation of Neural Network gradients arising from the second derivatives of high dimension PDE" [5].

5 Introduction to Equity Derivatives

In this section, we will introduce different types of financial derivatives for equities such as forwards, futures and Options. Also, we will take a closer look into vanilla options such as European call and put options and observe their relation to American options.

5.1 Financial Derivatives

A derivative is a financial contract made between multiple counterparties whose price (premium) depends on the value of some underlying financial asset. An underlying asset can be almost any type of traded financial security. For example: Equity(Shares), Bonds, Foreign currency(FX) or commodities(oil, water, etc). Some common examples of derivatives are forwards, futures, interest rate swaps and vanilla options.
Forward contracts - A forward contract is a legal customized agreement between two counterparties where one counterparty (A) agrees to buy a given asset from another counterparty (B) at a specified future date T for a pre - agreed price K.
Futures contracts - A futures contract is similar to a forward contract however the difference is that futures are highly standardized and traded over an exchange whilst forwards are traded over the counter.
Vanilla options - Similar to a forward/futures contract, it is an agreement between two counterparties where one counterparty (A) has the right but not the obligation to buy/sell a given asset to another counterparty (B) at a specified future date T for a pre - agreed price K.

5.2 European Call options

A European call option is a contractual agreement between two counterparties where counterparty A(the holder/buyer) has the right, but not the obligation to buy a given asset from counterparty B(the writer/seller) at some future date (T) for a pre - agreed price (K). The buyer of a call option profits when the underlying asset's price increases above the pre agreed underlying price.
Figure 1, shows the option's payoff with respect to the evolution of the underlying asset e.g. stock price.
From this we can deduce three scenarios:

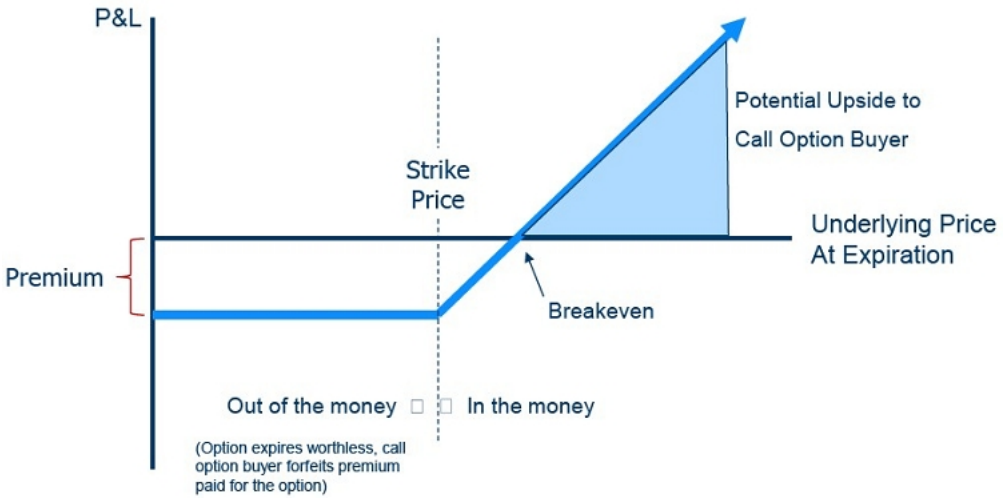


Figure 1: Call Option Payoff Diagram.Ref [6]

Scenario 1: ($S_T < K$) - When the price of the underlying asset (S_T) is less than the strike price (K) at the pre-determined time (T) the option holder makes a loss of the fixed premium amount as the option's payout is negative. However, as the buyer has no obligation to exercise the contract the actual payout is 0. The option is called an out of the money call and essentially worthless.
Scenario 2: ($S_T = K$) - When the price of the underlying asset (S_T) is equal to the strike price (K) at the pre-determined time (T) the option holder makes a loss as the option's payout is 0 but the fixed premium amount has already been paid. The option is called an at the money call and worthless.
Scenario 3: ($S_T > K$) - When the price of the underlying asset (S_T) is greater than the strike price (K) at the pre-determined time (T) the option holder makes a profit *if and only if* the price of the underlying asset is greater than then premium and the strike price. From Figure 1, this is shown as when the underlying asset value passes the break-even point ((S_T-K-premium)=0), the Profit and Loss(PNL) is above 0. The option is called an in the money call.

5.3 European Put options

A European put option is a contractual agreement between two counterparties where counterparty A(the holder/buyer) has the right, but not the obligation to sell a given asset to counterparty B(the writer/seller) at some

future date (T) for a pre - agreed price (K). The buyer of a put option profits when the underlying asset's price decreases below the pre agreed underlying price.

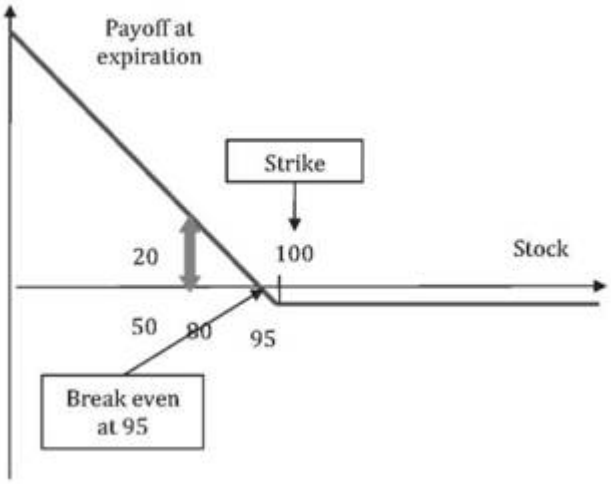


Figure 2: Put Option Payoff Diagram.Ref [7]

Similar to Figure 1, Figure 2 shows the option's payoff with respect to the evolution of the underlying asset e.g. stock price. Hence,we can deduce three scenarios.
Scenario 1: ($S_T < K$) - When the price of the underlying asset (S_T) is less than the strike price (K) at the pre-determined time (T) the option holder makes a profit *if and only if* the price of the underlying asset is less than the premium and strike price ($S_T <$ premium + K). From Figure 2, this is shown as when the underlying asset value decreases below the break-even point ((S_T - K - premium) = 0) the PNL is above 0. The option is called an in the money put.
Scenario 2: ($S_T = K$) - When the price of the underlying asset (S_T) is equal to the strike price (K) at the pre-determined time (T) the option holder makes a loss as the option's payout is 0 but the fixed premium amount has already been paid. The option is known to be an at the money put and worthless.
Scenario 3: ($S_T > K$) - When the price of the underlying asset (S_T) is greater than the strike price (K) at the pre-determined time (T) the option holder makes a loss of the fixed premium amount as the option's payout is negative. However, as the buyer has no obligation to exercise the contract the actual payout is 0. The option is called an out of the money put and essentially worthless.

5.4 American options

Similar to European options, American options have a strike price, stock price and exercise period. However, an American option can be exercised by the holder at any time up to, and including the expiry date T. Hence, American options are harder to price then European options as the optimal time to exercise for the holder is unknown at time of valuation. As a result, pricing an American option analytically is very hard if not impossible. Hence, we use alternative numerical approximations to find the fair value.
Let us consider an American call option on a stock if the option was exercised at some ($t \leq T$) with stock price s the intrinsic value of the option would be max(s-K,0) at a time t. This specifies what the payout will be if the option was exercised at a earlier time when the underlying stock was s.
Due to this added benefit, we can conclude that the fair value of an American option should be atleast equal to if not greater then an European option.

6 Valuation of options using Black Scholes Framework

In this section, we will discuss the stochastic process for which stock prices are modelled upon, the process is known as the geometric Brownian motion. Moreover, we will introduce a new special set of rules needed to differentiate such a process.
Finally we will derive the famous Black Scholes Partial Differential Equation which is crucial in the valuation of European & American options.

6.1 Brownian motion

The Wiener processes *W(t)* often regarded as the Brownian motion is a stochastic process with independent distributed increments. The idea of Brownian motions was first introduced by Robert Brown and was used to explain the random walk of particles in fluids. Additionally, a derived version of this stochastic process can be used to model stock prices for which Options are traded upon as they exhibit the same type of random behaviour.
Theorem: There exists a probability distribution over the set of continuous functions B_i *where* $0 \leq i \leq T$. *The probability distribution then follows these set of rules:*

- $P(B(0)=0)=1$
- *Stationary* $\forall\ 0 \leq s \geq t$ *where* $B(t)-B(s) \sim N(0,t-s)$
- *The increments are independent if interval* $[s_i,t_i]$ *are non-overlapping then* $B(t_i)-B(s_i)$ *are independent.*

Figure 3 below illustrates an example of a Brownian motion. From the graph it can be seen that the following three rules above apply to this stochastic process.

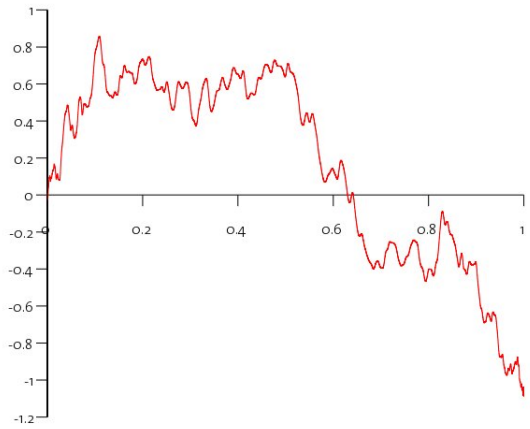


Figure 3: Brownian Motion.Ref [9]

The Brownian motion is a Martingale and Markov process and can be described by the following rules:

- The sample paths are highly irregular, and are not differentiable anywhere.
- The sample path look similar on all scales. However, if we have magnify any part of the sample Brownian path, what is seen is that the apparent behaviour is the same as the whole path.
- The sample path crosses the t - axis infinitely often
- Due to the fractal nature of the sample path, the length of a path between two different finite points is actually infinite.

From the above properties we can conclude the Brownian motion is unlike many other smooth curves such as trigonometric function and polynomials due to its fractal nature.

6.1.1 geometric Brownian motion

The Brownian motion itself is not good for modelling stock prices as there is an equal probability of an 'up' and 'down 'movement.
However, due to our prior knowledge about the share prices having an average increase over time, we can add drift to our existing model.
Additionally, we know that stock prices can never take negative values. As a result we can adjust the Brownian motion to fit these requirements thus we get an improved version known as the geometric Brownian motion.
Definition:
The GBM is defined as:

$$\mathrm{Y(t)} = \mathrm{Y(0)}\ \exp[mt + \sigma W(t)]$$

where the logarithm Y(t) is a Brownian motion with drift m, volatility σ and initial value log Y(0). Due to the logarithmic nature this process cannot have negative values.
A visual representation can be shown below:

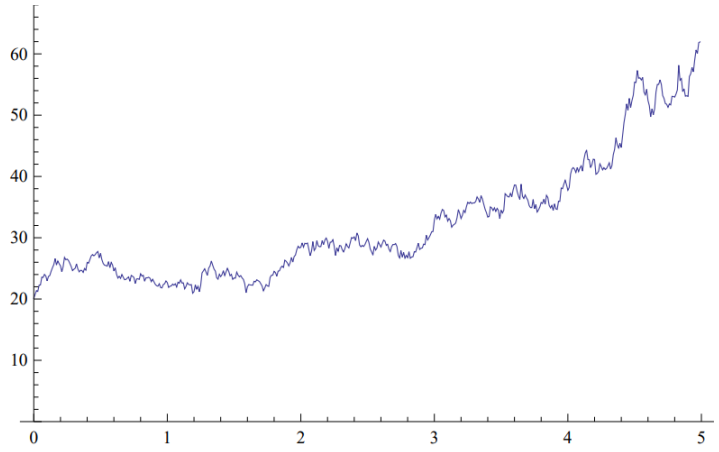


Figure 4: geometric Brownian motion.Ref [8]

6.2 Stochastic Calculus

Due to the volatile nature of Brownian motion the laws of traditional calculus fail. In this section we will look at alternative ways of deriving the Brownian motion with the use of Itô's Lemma.
Itô's Lemma lays the foundation for pricing Options as it gives us a process to differentiate stochastic processes e.g. geometric Brownian motion, which models the underlying asset e.g. stock prices.

6.2.1 Itô's Lemma

Ito's Lemma is an identity use to find the derivative of a time dependent stochastic process such as the geometric Brownian motion. It can be seen as the stochastic calculus version of the chain rule and can be derived using the Taylor series expansion on a smooth differentiable function.
Suppose we want to compute f(B(t) for some smooth function f.
Then using Taylor's expansion we can calculate the derivative of that smooth function df in the following manner:
$f(B_{t+x})$ - $f(B_t) = f'(B_t) * (B_{t+x} - B_t) + f''\frac{Bt}{2} * (dB_t)^2$.
where $f'(B_t)$ is a smooth differentiable function with Brownian motion as input and $(B_{t+x} - B_t)$ is a function that is computable over minuscule time intervals over the Brownian motion.
Quadratic variation tells us $(dB)^2 = dt$. [35]
Therefore we can substitute this term into the Taylor's expansion and get:
$f(B_{t+x})$ - $f(B_t) = f'(B_t) * (B_{t+x} - B_t) + f''\frac{Bt}{2} * (dt)$.
This formula is known as Itô's lemma.
A more sophisticated version of the above formula can be written as follows:
$f(t+d_t,B_t+dB_t)$ - $f(t,B_t) = \frac{df}{dt}\ dt + \frac{df}{dx} dB(t) + \frac{1}{2}\ \frac{d^2f}{dx^2}\ dt$.
which simplified equates to:
$f(t+d_t,B_t+dB_t)$ - $f(t,B_t) = (\frac{df}{dt} + \frac{1}{2}\ \frac{d^2f}{dx^2})dt + \frac{df}{dx} dB(t)$

Extending the formula above, we can find the derivative of the Brownian motion with added drift as follows:
Let a stochastic process X_t with drift be described as follows :
$X_t = \mu t + \sigma B_t$
with derivative:
$dX_t = \mu dt + \sigma B_t$
Then using the sophisticated form of Ito's lemma and substituting in the stochastic process and Stochastic Differential Equation(SDE) we get:
$df = \frac{df}{dt}\ dt + \frac{df}{dx}\ (\mu dt + \sigma B_t) + \frac{1}{2}\ \frac{d^2 f}{dx^2}\ (\mu dt + \sigma B_t)^2$
Which simplified equates to:
$\mathrm{df} = (\frac{df}{dt}\ + \mu\ \frac{df}{dx}\ + \frac{1}{2}\ \sigma^2\ \frac{d^2 f}{dx^2})\ dt + \sigma\ \frac{df}{dx}\ dB(t).$

6.2.2 Deriving the Black Scholes PDE

Suppose some stock price S follows a geometric Brownian motion such that:

$$ds = \mu S dt + \sigma S dw$$

where dw is normally distributed with mean 0 and standard deviation $\sqrt{dt}$, where dt is an interpretation.
To derive the PDE we assume the existence of three financial instruments:

- *A riskless bond B that evolves in accordance with the process dB=rBdt where r is the riskfree rate.*
- *An underlying security which evolves in accordance with the Itô process* $\mathrm{ds} = \mu\ \mathrm{Sdt} + \sigma\ \mathrm{Sdw}$
- *A option V written on the underlying security which, by Ito's Lemma, evolves according to the process* $(\frac{dV}{dt}\ + \mu S\ \frac{dV}{dS}\ + \frac{1}{2}\ \sigma^2 S^2\ \frac{d^2 V}{dS^2})\ dt + \sigma\ \frac{dV}{dS}$ *dB(t).*

The method we will use to derive the PDE will be the hedging argument. To hedge our position we need to set up a self financing portfolio $\prod$ containing $\triangle$ shares such that the portfolio is risk less.
Hence, the portfolio's value and its derivative at time t can be described as:

$$\prod(t) = V(t) + \triangle S(t) \qquad d\prod = dV + \triangle dS$$

Substituting Ito's Lemma as dV and the SDE as dS we get:
$(\frac{dV}{dt}\ + \mu S\ \frac{dV}{dS}\ + \frac{1}{2}\ \sigma^2 S^2\ \frac{d^2 V}{dS^2}\ + \triangle\ \mu S)dt + (\sigma S\ \frac{dV}{dS}\ + \triangle\ \sigma S)dW.$

For this portfolio to be hedged two conditions have to be met:

- The first feature is that it must be riskless, which implies that the second term involving the Brownian motion dW must be zero. For this this occur $\triangle = -\frac{dv}{ds}$. Substituting $\triangle$ into the equation we get:

$$d\prod = (\frac{dV}{dt} + \frac{1}{2}\sigma^2 S^2 \frac{d^2V}{dS^2})dt$$

.

- The second feature is that the portfolio must earn a risk free rate. This implies that the diffusion of the riskless portfolio is $d\prod = r\prod dt$. Therefore, we can write the equation as follows:

$$(\frac{dV}{dt} + \frac{1}{2}\sigma^2 S^2 \frac{d^2V}{dS^2})dt = r(V - \frac{dV}{dS}S)dt$$

.

- Finally, dividing both sides by dt and re-arranging such that all terms are on the left hand side we get the Black Scholes PDE:

$$\frac{dV}{dt} + \frac{1}{2}\sigma^2 S^2 \frac{d^2V}{dS^2} + rS\frac{dV}{dS} - rV = 0$$

.

6.3 Optimal stopping times For American options

European options, unlike American options are much more simpler to price under the given Black Scholes Model as the premium is only evaluated under one time condition which is at expiry(T).
Thus, we can price European options using an analytical solution from the Black Scholes PDE.
However, American options, can be exercised at any time upto and including the expiry date (T) therefore pricing & hedging American options under the Black Scholes framework become more difficult but possible as we have to account for infinitely many possible exercise times(stopping times).

In this section we will discuss the different methodologies for pricing & for American options under the Black Scholes Model, where the underlying asset S(t) pays no dividends and can be modelled using the geometric Brownian motion:

$$S(t) = S(0)\exp[(r - \frac{\sigma^2}{2}) + \sigma W(t)]$$

where r is the riskless rate of return and W(t) is a standard Weiner process under the risk neutral measure Q. For American options on a certain underlying stock, the exercise times must be treated as stopping times with respect to the natural filtration $(\mathrm{F_t})_{0 \le t \le T}$ of the Wiener process W(t). If F(t) is the payoff of the American option exercised when the stock price is s and if T is the expiry date of the option then its value is V(t) at time t $\leq$ T is:

$$V(t) = \sup_{t \le \tau \le T} \mathbb{E}(F(S_t)\exp(-r(\tau - t) \mid F_t)$$

where $\mathbb{E}$ is the risk neutral expectation for which the growth rate of the option is controlled by risk-free rate r. This version of risk neutral valuation provides the extra stipulation of the early exercise boundary.
Consider the early exercise decision at a point (S,t). The value of the payoff at that point is F(S,t). Similarly the expected value of the future exercise is:

$$\hat{V(t)} = \sup_{t \le \tau \le T} \mathbb{E}(F(S_t)\exp(-r(\tau - t) \mid F_t)$$

Therefore, the holder of the option will exercise the option if the payoff $\geq$ option price. Hence, the value of the option must be:
$V(t) = max(F(S,t), \hat{V(t)})$

6.3.1 American call options

Suppose we have an American call option which pays an amount g(s) if exercised at a time S_n = s, where g is convex and g(s) is positive. Then the optimal strategy is to exercise at expiry given the payoff is an in the money call.
Given that under the risk-neutral probability measure Q, the discounted stock price is a martingale and can be expressed by the following:

$$g(Sn) = g(\mathbb{E}^Q\, n[\frac{S(n+1)}{1+R}])$$

Since g is a convex we can re-write the equation in the following way:

$$g(Sn) \leq \mathbb{E}^Q\, n(g[\frac{S(n+1)}{1+R}])$$

and due to g(0) = 0 the right hand side can be written as :

$$\mathbb{E}^Q\, n[\frac{1}{1+R} g(S(n+1))]$$

Accordingly:

$$g(Sn) \leq \mathbb{E}^Q\, n[\frac{1}{1+R} g(S(n+1))]$$

From the proof, we can see that the discounted expected value of the payoff from exercising one time step later is always greater if not the same, therefore, instead of exercising at time n we should always exercise at time n+1 but at time n+1 in order to maximise payoff we would exercise at time n+2.
In conclusion, the most optimal stopping time to exercise the American call option would be only expiry time N.
Consequently, we can price an American call options that pay no dividends in the same way we would price European call options where we use the Black Scholes analytical formula.

6.3.2 American put options

Subsequently, American put options are much more difficult to price when compared to their counterparts American calls since the optimal exercise policy is not to hold until expiration regardless of the scenario. Therefore, the price of an European put option has to be less than or equal to the American put option.

Proposition: if the risk free rate r is positive then for every $t \leq T$

$$V_E(t, S_t) \leq V_A(t, S_t)$$

The optimal policy for put options can be expressed in the following way: For a certain differentiable,monotonically increasing function $s_*(\mathrm{t})$ known as the exercise boundary one should exercise the put at $\tau \leq \mathrm{T}$ such that $S_\tau \leq s_*$. However, the major problem of American options is computing the exercise boundary for which there is no analytical solution. Below is a visual representation of the exercise boundary.

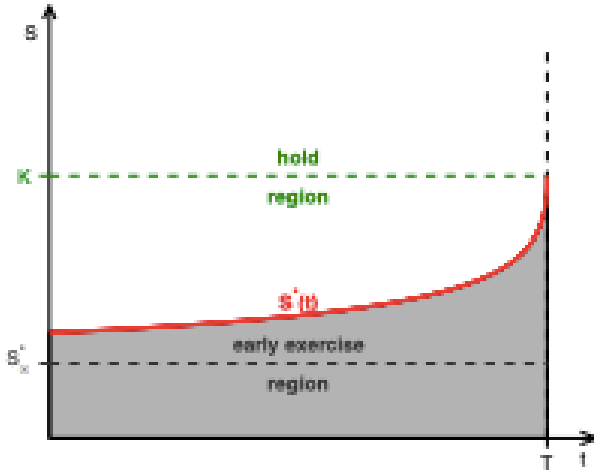


Figure 5: Optimal Exercise Boundary diagram.Ref [15]

From the graph, we can see the red line dividing the two regions $S^*(\mathrm{t})$ is the optimal exercise boundary. The grey dashed line indicates the asymptotic behaviour of the optimal exercise boundary as $\mathrm{T} = \mathrm{t} \to \infty$. The optimal exercise boundary has no analytical solution therefore has to be computed numerically using Linear Complementary Formulation.

Proposition Proof: Consider the following exercise policy for the American put: Exercise at $min(\tau, T)$ where

$$\tau = min(t \leq 0 : S_t \leq K - K \exp r(t - T))$$

If $\tau \leq T$, the put will be exercised at a time τ when

$$S_\tau \leq K - K \exp r(t - T))$$

. The payoff will be minimum $K \exp r(t - T)$. We can invest the amount K into the bank and let in grow for the remaining time giving us an amount K. This is greater then any possible payoff of the European put.
On the other hand if $\tau \geq$ T we can exercise at time T, and the payoff will be the same as the European put. In both cases, the payoff of the American put is the same as the European put and in the first case the payoff of the American put is greater then the European put.

6.3.3 Linear Complementarity Formulation

As stated above, there is no analytical solution for the optimal exercise boundary function. To solve this, we will use a method known as the Linear Complementarity Formulation(LCF). The aims of the LCF are as follows:

- An American option price cannot be less than the payoff.
- For any given time, it is either optimal to exercise the portfolio or it isn't. To decide which, we consider the return on the Black Scholes hedged portfolio which we derived earlier. If the return is equal to the risk free return on the value of the portfolio we keep holding, but if it less we have to tune the portfolio.

Consider the Black Scholes Hedged portfolio:

$$d\prod = (\frac{dV}{dt} + \frac{1}{2}\sigma^2 S^2 \frac{d^2V}{dS^2})dt$$

In the case for an European option we stated that $d\prod = rM_t dt$
as $d\prod < rM_t dt$ and $d\prod > rM_t dt$ lead to arbitrage opportunities.
For an American put option we have the stipulation of early exercise therefore the hedged portfolio has to be changed such that $d\prod \leq rM_t dt$ which leads to the inequality of the Black Scholes PDE

$$\frac{dV}{dt} + \frac{1}{2}\sigma^2 S^2 \frac{d^2V}{dS^2} + rS\frac{dV}{dS} - rV \leq 0$$

.

As stated earlier, there are optimal times to exercise an American put option early therefore there might be times where writing the put option would be detrimental to the seller since $d\prod < rM_t dt$.
Essentially, for an American put option there might be instances where the delta-hedged portfolio($d\prod$) could be under-performing the risk free rate deposit on the value of the portfolio ($rM_t dt$).
Scenario 1:
Suppose that the option's price is greater then the payoff:

$$V_A(t, S_t) \geq P_o(t, S_t)$$

In this scenario it would never be beneficial to exercise the option as you could always sell it for a higher price. If short selling is to be safe then the Black Scholes PDE cannot be less than 0 as it will lead to arbitrage opportunity. As a result, if

$$V_A(t, S_t) \geq P_o(t, S_t)$$

the Black Scholes PDE has to be equal to 0. In this scenario the option is never exercised unless at expiry.
Scenario 2:
Suppose the Black Scholes PDE is less then 0. Then the $\triangle$ hedged portfolio's return is less then the risk free return available by selling the portfolio and putting the money in the bank. Therefore, it would be wise to sell the portfolio and invest the money into the bank to grow at a risk free rate. Consequently, if the Black Scholes PDE is less then 0

$$V_A(t, S_t) = P_o(t, S_t)$$

From the scenario's above we can identify the optimal exercise boundary is where the option's price switches from

$$V_A(t, S_t) > P_o(t, S_t)$$

$$to$$

$$V_A(t, S_t) = P_o(t, S_t)$$

$$\&$$

$$L_{bs}[P_{am}] = 0$$

$$to$$

$$L_{bs}[P_{am}] < 0$$

We can write the LCP formulation as the following:

$$L_{bs}[\,V_{am}] \leq 0,\; V_{am} - P_0 \geq 0$$

$$(\,V_{am} - P_0) * L_{bs}[\,V_{am}] = 0$$

7 Deep Neural Networks

Deep Neural Networks are used to solve a variety of different real life problems which before the 21st century were deemed impossible. However, due to the significant improvements made within technology and the creation of this machine learning technique, such problems are now becoming understood and solved.

7.1 What is a Neural Network?

An Artificial Neural Network(ANN) is a computing system that consists of a collection of interconnected nodes known as artificial neurons. These nodes are organized into different layers i.e. Input layer, hidden layers and output layer. The amount of nodes(width) and hidden layers(depth) are defined by the user. A very simple version of the layout works in the following way:

- *Data gets passed to the nodes within the input layer.*
- *Upon completed procedures on the input layer, updated data is then passed to the nodes of the hidden layers.*
- *Upon completed procedures on the hidden layers, updated data is then passed to the node/nodes of the output layer.*

A visual representation of the description above can be seen below:

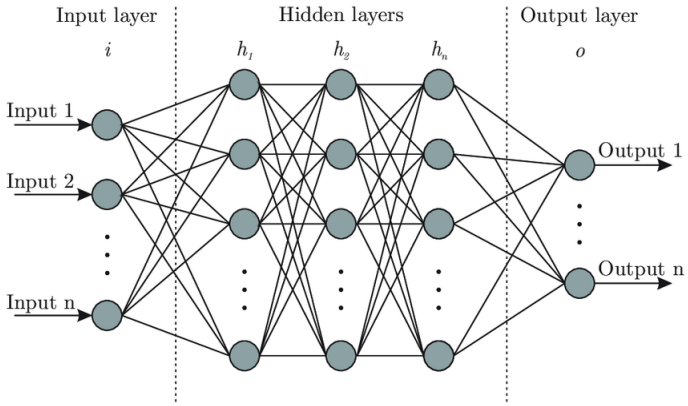


Figure 6: Neural Network structure.Ref [16]

Prior to discussing the Feed Forward & Backward Propagation algorithms, some key notations that should be noted are as follows:

- *L = number of layers in the network*
- *Layers are indexed by l=1,2,...L-1*
- Y_j *= The ground truth for a node j in the output layer L.*
- *Nodes in a given layer l are indexed by j=0,1,...N-1*
- *Nodes in a given layer l-1 are indexed by k=0,1,..N-1*
- $w^{(l)}{}_{jk}$ *= The weight of a single connection that connects node k in layer l-1 to node j in layer l.*
- $w^{(l)}{}_{j}$ *= The vector that contains all weights connected to node j in layer l by each node in layer l-1.*
- $z^{(l)}{}_{j}$ *= The input for node j in layer l.*
- $\sigma^{(l)}{}_{j}$ *= The activation function used for layer l.*
- $a^{(l)}{}_{j}$ *= The activation output of node j in layer l.*

7.2 Forward Propagation

In the previous section we talked about the skeletal structure of a NN. However, what happens within each node in the Neural Network and how do we forward propagate it through the network?
When data is passed to the input layer of a NN, it is multiplied by a default initialized weight and a default bias. This is further wrapped into some kind of an Non - linear activation function such as RELU(Rectified Linear Unit), sigmoid or tanh function. The advantages and disadvantages of using such functions on the weighted data are as follows:
Advantages:

- *The gradient is smooth therefore jumps are prevented in the output values.*
- *Output values are bounded between 0 and 1, this allows for values to be standardised and negative values cannot occur.*
- *The prediction becomes clear, since extreme input values tend to either 0 or 1 when in-putted into the activation function.*

Disadvantages:

- *For very high or low values of X, there is no change in the prediction. This can lead to a exploding gradient or vanishing gradient problem which leads to a network refusing to learn or being to slow to reach a optimal solution. This will be further discussed later in the section.*
- *It can be computationally expensive.*

An example of the inner mechanisms of a node/artificial neuron can be shown below:

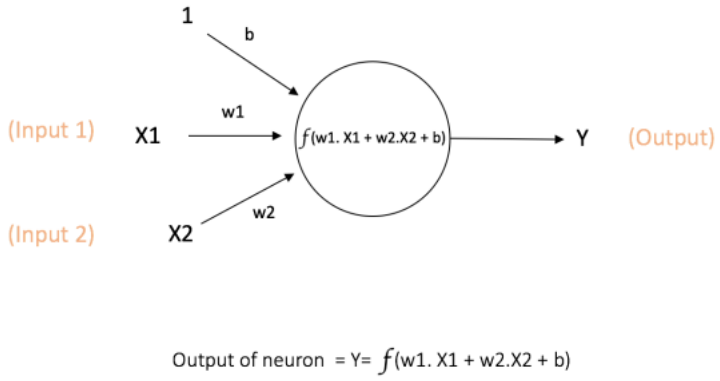


Figure 7: Node structure. Ref[17]

The weights and biases added to the different inputs happens in the edges(connections) that connect the input layer and the hidden layer. Hence, the input to the nodes of the first hidden layer are the summation of multiple weighted outputs of the different input layers. Therefore, the input for a given node j in hidden layer l can be expressed as follows:

$$z^{(l)}{}_j = \sum_{k=0}^{n-1} (w^{(l)}{}_{jk} a^{(l-1)}{}_k)$$

We can further wrap this input into an Non -linear activation function in order to get an activation output for node j in the layer l. We can express the activation output as follows:

$$a^{(l)}{}_j = \sigma_l(z^{(l)}{}_j)$$

The aforementioned process happens within all hidden layers until we reach the output layer where the output gets evaluated against the ground truth. A key note that should be realised within this process is that it moves in one direction therefore the data cannot flow in reverse order, to put it simply the network is acyclic. This process is known as Feed Forward Propagation.

7.3 Stochastic Gradient Descent

Prior to discussing the Backward Propagation algorithm which is used to optimize the Neural Network, we will discuss what is arguably the most fundamental methodology within Back Propagation known as Stochastic Gradient Descent(SGE).
However, preceding SGE we need to understand Gradient Descent. Gradient descent is an optimization algorithm used to minimize some objective function by iteratively moving in the direction of steepest descent. The algorithm can be described as follows:

repeat until convergence

$$\theta_j = \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$$

In machine learning, this optimization tool is used to optimize coefficient terms or weights of machine learning models such as: Neural Networks, Dimensionality Reduction & Regression(Linear, Polynomial & Logistic).A step by step breakdown of the algorithm is as follows:

- *Step 1 - Calculate the cost function for some machine learning model that has been trained. The cost is calculated using the sum of squared error residuals formula which is as follows*

$$C = \frac{1}{N} \sum_{i=1}^{N} (Y_{pred} - Y_{actual})^2$$

  where Y_{pred} contains the weighted coefficient multiplied by an independent variable added to a bias term i.e. $Y = mx+b$ for Linear Regression.

- *Step 2 - Calculate the derivatives of the cost function with respect to each of the parameters. The derivatives tell us the steepness and direction of the descent at that point.*

- *Step 3 - Pick random initial values for the derivatives parameters & update the gradient function by inserting the parameters.*

- *Step 4 - Calculate the step size for the descent. We achieve this multiplying the slope(derivatives output) by some learning rate. This is done to control the rate of descent down the cost function.*

- *Step 5 - Update the parameters of the cost function. This is done by subtracting the old parameters by the step size i.e. New parameter = Old parameter - step size.*
- *Repeat Step 3 - Step 4 until the step size is very small.*

A visual representation of the algorithm can be shown below:

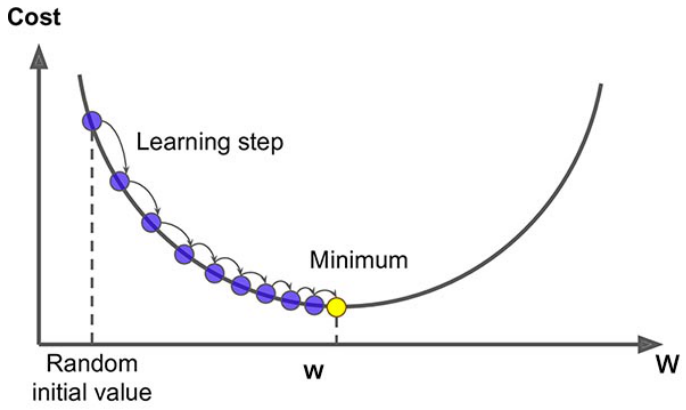


Figure 8: Gradient Descent Diagram. Ref[21]

Gradient Descent is a very good algorithm to optimize parameters. However, for training very large data sets like the ones used for a Neural Network it is very computationally expensive. To prevent this, we use an improved version known as Stochastic Gradient Descent.
Stochastic Gradient Descent randomly picks a data point from the entire data set during each iteration in order to ease the computational burden that normal Gradient descent has. Additionally, we can also randomly choose a small number of data points at each step called a "mini batch" in order to improve accuracy.

7.4 Backward Propagation

Forward propagation allows us to feed forward data into the network until it reaches the output layer. However, what happens when the data is sent to the output layer?
Upon reaching the output layer we obtain the model's output(predicted output) for a given set of inputs. The output node with the highest activation is the one the model believes to be correct for the corresponding inputs. This value is almost guaranteed to be incorrect when compared to the actual output. To fix this, we use Backward Propagation.
Backward Propagation is an algorithm for Neural Networks that uses Stochastic Gradient Descent. In order to use Back Propagation we need to obtain the cost function of a Neural Network. Subsequently, Back Propagation can be used to calculate the gradient of the cost function with respect to the Network's weights. The term 'Backward' is used to describe this process because the calculation of the gradients are executed backwards through the Network, since the gradients of the final layer weights are calculated first and the gradients for the first layer weights are calculated last.
As stated above, we need to be able to define the cost function for the output layer to use Back Propagation.The cost can be expressed in the following way:

$$C_0 = \frac{1}{2N}\sum_{i=1}^{N}({a^{(L)}}_j - y_j)^2$$

The term ${a^{(L)}}_j$ is the activation output in final layer. Regressing, we realise this activation output is calculated by taking a product of the weighted sums of the activation outputs from the previous layer l-1. Therefore, the input for a given node j in layer L can be expressed as follows:

$${z^{(L)}}_j = \sum_{k=0}^{n-1}({w^{(L)}}_{jk}{a^{(L-1)}}_k)$$

Similar to Feed Forward Propagation we can wrap this input into an activation function in order to get an activation output for node j in the layer L. We can express the activation output as follows:

$${a^{(L)}}_j = \sigma_L({z^{(L)}}_j)$$

Observe, the cost function of a single node j is composed of the activation output in the final layer L. The activation output of node j in the output

layer L is composed of the weighted inputs for node j as can seen above. The input for node j can be expressed as a function of all the weights connected to node j. Therefore, we can define C_{0_j} as a composition of functions:

$$C_{0_j} = C_{0_j}(a^{(L)}{}_j(z^{(L)}{}_j(w^{(L)}{}_{jk})))$$

Hence, the derivative of the cost function for a single output node in the output layer can be expressed in the following way:

$$\frac{\partial C_0}{\partial w^{(L)}{}_j} = \frac{\partial C_0}{\partial a^{(L)}{}_j}\frac{\partial a^{(L)}{}_j}{\partial z^{(L)}{}_j}\frac{\partial z^{(L)}{}_j}{\partial w^{(L)}{}_{jk}} \tag{1}$$

The derivatives for each term can be described as follows:

$$\frac{\partial C_0}{\partial a^{(L)}{}_j} = 2(a^{(L)}{}_j - y_j) \qquad \frac{\partial a^{(L)}{}_j}{\partial z^{(L)}{}_j} = \sigma'^{(L)}{}_j z^{(L)}{}_j \qquad \frac{\partial z^{(L)}{}_j}{\partial w^{(L)}{}_{jk}} = a^{(L-1)}{}_j \tag{2}$$

So far we have derived the loss gradient for a single weight for one training sample. To calculate the derivative of the loss with respect for a single weight for all n training samples, we calculate the average derivative of the loss function over all n training samples. We can express this as:

$$\frac{\partial C_0}{\partial w^{(L)}{}_{jk}} = \frac{1}{n}\sum_{i=0}^{n-1}\frac{\partial C_0}{\partial w^{(L)}{}_{jk}}$$

This process would be done on each weight in the network in order to minimize the loss of c with respect to each weight.
The procedure above is done for all of the weights connecting to the different output layers in order to minimize the error of the cost function. Up-till now, we have only talked about the derivative of the loss with some particular weights connecting only to the output layer.
However, what happens when we go a step back to calculate the derivative of the loss with respect to a weight in the hidden layer (L-1)?
We can calculate the derivative of the loss with respect to a weight in the hidden layer L-1 in a similar fashion as we did in the output layer as shown below.

$$\frac{\partial C_0}{\partial w^{(L-1)}{}_k} = \frac{\partial C_0}{\partial a^{(L-1)}{}_k}\frac{\partial a^{(L-1)}{}_k}{\partial z^{(L-1)}{}_k}\frac{\partial z^{(L-1)}{}_k}{\partial w^{(L-1)}{}_k} \tag{3}$$

However, the main difference in this method is that we cannot calculate the first derivative term in the same way we did for the output layer since unlike

the output layer, the loss function isn't composed of the activation output in the final layer.
Nevertheless, as stated previously the activation outputs in the final layer is dependent on the activation outputs from the previous layer and the weights connecting those activation outputs to the output layer. Therefore, a change in the activation output in the previous layer (L-1) will have a direct change in the activation output in the final layer. In essence, since ${z^{(L)}}_j$ is dependent on the activation output ${a^{(L-1)}}_j$ and the following functions can be composed:

$$C_{0_j} = C_{0_j}({a^{(L)}}_j({z^{(L)}}_j({a^{(L-1)}}_k)))$$

The derivatives of the given composed function solved using the chain rule as follows:

$$\frac{\partial C_0}{\partial {a^{(L-1)}}_k} = \sum_{j=0}^{n-1}(\frac{\partial C_0}{\partial {a^{(L)}}_j}\frac{\partial {a^{(L)}}_j}{\partial {z^{(L)}}_j}\frac{\partial {z^{(L)}}_j}{\partial {a^{(L-1)}}_k}) \tag{4}$$

From the derivative above, similarities can be seen when compared to the loss function derivative calculated in the output layer. However, the two significant differences.
The first difference is the current derivative has a summation this is because we are calculating the cost with respect to the activation outputs in hidden layer(L-1), therefore changing one activation output in the previous layer is going to effect the each node j in the final layer L, and so we need to sum up these effects.
The second difference is the third term is evaluated with respect to the activation output and not the weight.
Previously, we calculated ${z^{(L)}}_j$. Substituting this into the third term and expanding we get:

$$\frac{\partial {z^{(L)}}_j}{\partial {a^{(L-1)}}_k} = \frac{\partial}{\partial {a^{(L-1)}}_k}\sum_{k=0}^{n-1}({w^{(L)}}_{jk}{a^{(L-1)}}_k) = {w^{(L)}}_{jk} \tag{5}$$

Henceforth, it can be stated the input for any node j in layer L will respond to a change in the activation output in a previous layer ${a^{(L-1)}}_k$ by an amount equal to the weight connecting the node in the previous layer L-1 to the nodes current layer L.
Now that we solved $\frac{\partial {z^{(L)}}_j}{\partial {a^{(L-1)}}_k}$ we can re combine these terms to get the deriva-

tive of the loss with respect to the previous activation outputs as follows:

$$\frac{\partial C_0}{\partial {a^{(L-1)}}_k} = \sum_{j=0}^{n-1} (2({a^{(L)}}_j - y_j)({\sigma'^{(L)}}_j({z^{(L)}}_j))({w^{(L)}}_{jk})) \tag{6}$$

Additionally, we can use this result to solve the gradient of the loss with respect to any weight connected to any node k in a previous layer L-1 as follows:

$$\frac{\partial C_0}{\partial {w^{(L-1)}}_k} = \sum_{j=0}^{n-1} (2({a^{(L)}}_j - y_j)({\sigma'^{(L)}}_j({z^{(L)}}_j))({w^{(L)}}_{jk}))({\sigma'^{(L-1)}}_k({z^{(L)}}_k))({a^{(L-1)}}_{k-1}) \tag{7}$$

Similar to the output layer, we can find the derivative of the loss function with respect to some particular activation output in the hidden layer over all n training samples by calculating the average derivative of the loss function over all n training samples. We can express this in the following way:

$$\frac{\partial C}{\partial {a^{(L-1)}}_k} = \frac{1}{n} \sum_{i=0}^{n-1} \frac{\partial C_i}{\partial {a^{(L-1)}}_k}$$

This process is done for all weights in the hidden layer L-1 and similarly in every previous layer prior to this until we reach the input layer in order to optimize the weights. This process is done in a backward fashion and is called Backward Propagation.
Consequently an step by step overview of a neural network can be shown in the following way:

- *Step 1 - Input a mini batch of training samples chosen randomly from the set of all training samples.*
- *Step 2 - Set the corresponding activation output* $a^{(x,1)}$.
- *Step 3 - For each l=2,3,....L compute* $z^{(x,l)} = w^l a^{(x,1-1)} + b^l$ *and* $a^{(x,1)} = \sigma(z^{(x,l)})$.
- *Step 4 - Compute the error between the model's output and the actual output* $C_0 = \frac{1}{2N} \sum_{i=1}^{N} ({a^{(L)}}_j - y_j)^2$

- *Step 5 - For each l = L-1,L-2,...L-N back propagate the error* $\frac{\partial C_0}{\partial w^{(L)}{}_j} = \frac{\partial C_0}{\partial a^{(L)}{}_j} \frac{\partial a^{(L)}{}_j}{\partial z^{(L)}{}_j} \frac{\partial z^{(L)}{}_j}{\partial w^{(L)}{}_{jk}}$.

- *Step 6 - For each l= L,L-1,....L-N,update the weights according to the SDE rule* $w_j = w_j - \alpha \frac{\partial C_0}{\partial w^{(L)}{}_j}$ *where L=0.*

- *Step 7 - Repeat step 5 and 6 until convergence.*

7.4.1 Vanishing Gradients

A Deep Neural Network is simply an Artificial Neural Network with many hidden layers. The benefits of having many hidden layers is that alot more internal calculations occur i.e. more weights,more activation outputs.
Therefore, the Network is learning better and will give results with more accuracy and less error.
On the other hand, the disadvantages of implementing a Deep Neural Network is the Vanishing Gradient Problem. The Vanishing Gradient Problem occurs because of 2 reasons:
The first reason is due to the non - linear activation functions being sigmoid or tanh function. While these functions are very good at allowing the network to learn, the downside is the derivative outputs of these functions are very small. For example, the derivative of a sigmoid is between $0 < \sigma'(z_x) < 0.25$.
The second reason comes when the weights are being updated throughout all layers up until the input layer. During the Back Propagation process we realise the gradients which get calculated last have a minuscule impact on the weight updates. This is because the product of all activation outputs are between very small quantities.
This impacts a Deep Neural Network more then an Artificial Neural Network because a Deep Neural Network has more layers to propagate through.
To combat this flaw within Deep Learning we introduce a technique called Long Short Term Memory(LSTM).

7.4.2 Recurrent Neural Networks and LSTM's

Recurrent Neural Networks are similar to Feed-forward Neural Networks but with one key difference, the Network has internal memory at each step. The recurrence nature of the networks allows it to perform the same function for

every input of data while the output of the current input depends on the previous computations. Upon producing the output, it is copied and sent back into the Recurrent Network. Due to this added feature, Recurrent Neural Networks are a great tool for predicting sequential data such as: Time series data, Sentiment analysis and Text data. Below is a visual representation of a Recurrent Neural Network.

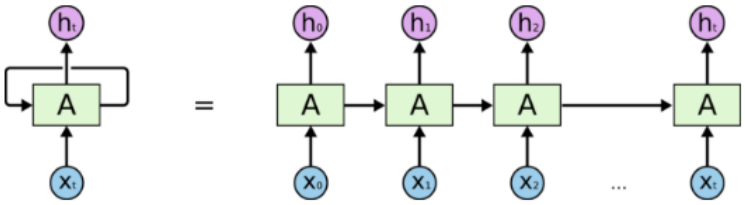


Figure 9: Recurrent Neural Network. Ref[22]

A breakdown of the diagram above can be explained in the following way:

- Step 1 - Create some initial hidden state h_0 which is initialized as 0 and we have some input x_t. We can express the current state as

  $h_t = f(h_{t-1}, x_t)$

- Step 2 - Apply the Activation function to the current state we get:

  $h_t = tanh(W_{hh}h_{t-1} + W_{xh}x_t)$ where W is the weight, h is the single hidden vector, W_{hh} is the weight added to the previous hidden state, W_{hx} is the weight added to the current input state and tanh is a Non-Linear Activation function

- Step 3 - Repeat step 2 until all hidden states have been found.

- Step 4 - Create an output state which contains the final hidden state with a weight. $Y_t = W_{hy}h_y$.

While RNN's are very good at processing sequential data going forward, they face a similar problem to Artificial Neural Networks which is the Vanishing Gradient problem. For RNN'S, this is a huge problem as it loses its memory during the Back Propagation algorithm. In order to tackle this Long Short Term Memory(LSTM) was created which are a modified version of RNN'S. This process allows past memory to be remembered which resolves the vanishing gradient problem for both RNN'S and ANN's.

LSTM are essentially an added layer of control to the hidden states as it allows you keep only relevant information for predictions and lets you forget irrelevant data. The architecture of an LSTM was created for better gradient flow properties.
An LSTM occurs between two hidden states at each and every time step. While maintaining a hidden state h_t within the LSTM architecture we introduce a new vector called c_t known as the cell state. This is an internal vector that is strictly kept within the LTSM.
In an LSTM we take three inputs h_{t-1},x_t,c_{t-1} and use them to compute four new gates. Upon computing these gates we can update our cell state c_t which in turn updates our hidden state.
A visual representation of a LSTM can be shown below:

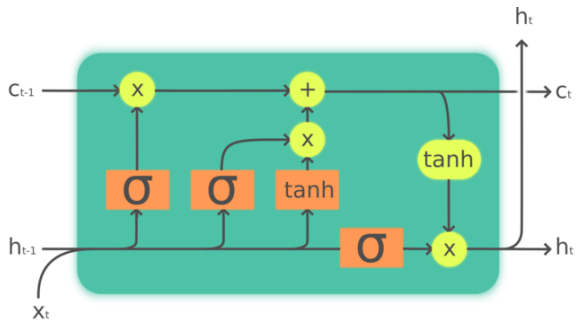


Figure 10: Long Short Term Memory cell. Ref[24]

As stated above, there are four gates within an LSTM. In the figure above, these gates can be shown by the highlighted red cells and can described in the following manner:

- Forget gate(f) - This gate controls the existence of a cell and can be described using the following equation $f = \sigma(w(h_{t-1}) + wx_t)$. In the LSTM diagram it is the first highlighted cell from the left.

- Input gate(i) - This gate controls whether to write to a cell and can be described using the following equation $i = \sigma(w(h_{t-1}) + wx_t)$. In the LSTM diagram this is the second highlighted cell from the left.

- Candidate layer(g) - This gate controls how to much to write to a cell and can be described using the following equation $g = tanh(w(h_{t-1}) +$

wx_t). In the LSTM diagram this is the third highlighted cell from the left.

- Output layer(o) - This gate controls how much to reveal of a cell and can be described using the following equaiton $o = \sigma(w(h_{t-1}) + wx_t)$. In the LSTM diagram this is the fourth highlighted cell from the left.

Upon computing the four gates we can solve the cell state and hidden state for the next LSTM cell using element wise matrix multiplication. The formula's for the cell and hidden states can be described as follows:
$c_t = f \odot c_{t-1} + i \odot g$ $\qquad$ $h_t = f \odot tanh(c_t)$

The LSTM network looks visually quite complicated. To explain the inner workings, lets simplify the problem by imagining the gates as having extreme binary values(0 or 1) of to the activation functions output. Then we can think of the forget gate as being a vector of zeros and ones. Taking this into consideration and looking at the first part of the cell state equation we can see the forget gate is being multiplied element wise by the cell state from the previous time. This multiplication tells us to forget the element of the cell state in the previous time step in the case the forget gate was 0 or to remember the element if the forget gate was 1.
Once this is evaluated we have the second term of the cell state which is the element-wise product of i and g. Similiary i can be seen as a vector of zeros and ones. This tells us if we want to write into that element of the cell state in the case i is 1 or if we do not want to write into the cell state if i is 0. Expanding, the Candidate layer which has a tanh activation function ranges the binary values from -1 to 1, so the element wise multiplication tells us to consider writing each element of the cell state at each time step.
To summarize, we can either forget or remember the cell state and increment or decrement up to one at each time step. Upon computing the cell state, we insert the cell state into an tanh activation function and element-wise multiply it by an output gate which is also consisted of zeros and ones. This tells us whether or not we want to reveal the element of out cell state at each time step.
This allows for more efficient back-propagation because the forget gate is now an element wise multiplication instead of a full matrix multiplication. This leads to a lot more control over the element flow of the matrix since the element wise multiplication will multiply by a forget gate at each time step.

7.4.3 Highway Networks

Highway Networks were introduced in the last decade as a means to optimize Deep Neural Networks by reducing the Vanishing Gradient problem. It was created to be used for Feed Forward versions of LSTM networks therefore shares many similarities to it in-terms of structure. As a result, Highway Networks have a gating mechanism which regulates the flow of information. These paths are called information highways.
In a Feed Forward Neural Network we have one transformation function i.e $H(x, W_H)$. However, in a Highway Networks we have three transformation function including the initial one from the Feed forward Neural Net. These are known as the transform gate $T(W_T, x)$ and the carry gate $C(W_C, x) = 1 - T(W_T, x)$.
We can then write the output of a Highway Network layer as:

$$y = H(x, W_H) \cdot T(x, W_T) + x \cdot (1 - T(x, W_T)$$

Similar to LSTM lets look at the extreme values of the gates(0 and 1).
Substituting $T(x, W_T) = 0$ into the equation above, we get y = x. This allows information to flow through the network without diminishing therefore increasing the overall flow of the network. On the other hand, if $T(x, W_T) = 1$ we get the non linear activation transformed input as the output. In summary, depending on the output of the transform gates the Highway Network can smoothly vary its behaviour between that on a non - linear activation function and that of one that passes inputs as outputs.
A visual representation of the Highway Network can be shown below:

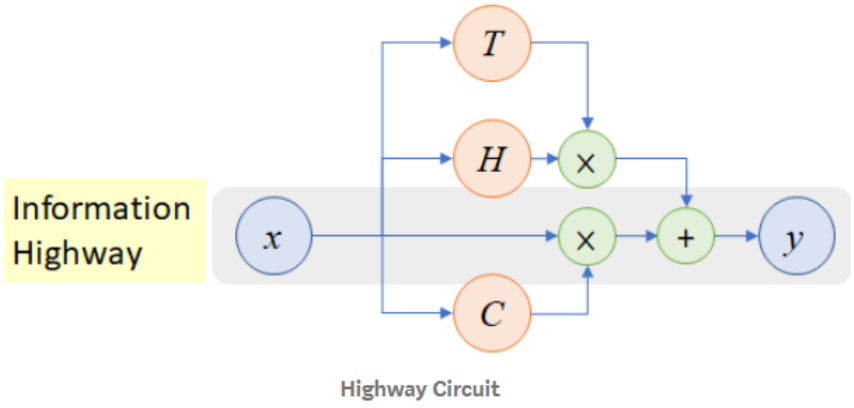


Figure 11: Highway Network Diagram. Ref[26]

8 Numerical Methods for American options

From Valuations of options, we concluded that there is no analytical solution for American put options and we can use the Black Scholes analytical formula to find the fair value of an American call option which pays no dividends. In this section, we will discuss the implementation and results for different numerical methods for pricing American options. The Lattice Method & Monte Carlo Simulation will be briefly discussed since the methods tend to have a higher error rate or high time complexity while pricing American options due to the designs of the algorithm. On the other hand, Finite Difference Methods & Deep Gelarkin Methods are better for pricing American options as there is less error with comparable computational effort. We will price American options by evaluating the Black Scholes PDE using Deep Neural Networks and comparing the results to the Finite Difference method. Additionally, we will exploit the heavy computational power associated with Deep Neural Networks to price Multi-Dimensional American options where the number of dimensions are the amount of different stocks within a given portfolio.

8.1 Lattice Method

The Lattice(tree) Method also known as the Binomial Option Pricing Method, is a discrete time model that is used to find the current fair value of any type option. While the model is simple in nature, it lays the groundwork for every other complex numerical technique. The valuation works in a three step procedure:

- Step 1: Generate the Binomial Pricing Tree

  We have our initial stock price S_0 at T = 0 which we believe will move up or down by a multiplicative factor (u *or* d) at T = 1 where $u \geq 1$ *and* $0 < d \leq 1$.

  The 'up' & 'down' factors are calculated using the underlying volatility (σ) and the time duration step (t), in the following way: u = $\exp[(\sigma\sqrt{t})]$ & d = $\exp[(-\sigma\sqrt{t})]$.

  Suppose, we have a current Stock Price $S_0 = 100$ at T = 0, given u = 1.2 and d = 0.9. The price of the stock S_1 at T = 1 can be ${S^u}_1$ = 120(100*1.2) or ${S^d}_1$ = 90(100*0.9). Proceeding, at T = 2 we get ${S^{uu}}_2$ = 144(120*1.2), ${S^{ud}}_2$ = 108 and ${S^{dd}}_2$ = 81.

The Binomial Tree can be expressed in the following way:

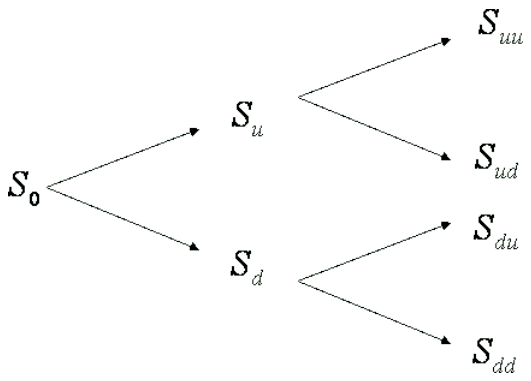


Figure 12: Binomial Pricing Tree. Ref[27]

- Step 2: Find the option value of the underlying asset at the final node for each path.

  We evaluate the option's payoff on the final time step. If the payoff is a call option we evaluate the payoff as max(S_t - K , 0) else if the payoff is a put option we use max(K - S_t , 0).

  In our example above, our final time step was T = 2, given the option was of type American call and had strike price = 105, we would get ${V^{uu}}_2$ = max(39(144 - 105) , 0) in the upmost node. The middle node will be ${V^{ud}}_2$ = max(3(108 - 105) , 0). Finally, the lowest node will be ${V^{dd}}_2$ = max(81-108 , 0) since 81 - 108 is a negative number our payoff is 0.

- Step 3: Find the option's value at earlier nodes.

  We take an inductive step backwards in order to calculate the fair value of the derivative at each time step using risk neutral assumption. Under this assumption it is presumed that the current fair value of a derivative is equal to the future expected value of it's payoff discounted by the risk free rate. The expected value is calculated by using the option's values(payoffs) at two future nodes from its current node and weighting them by their respective probabilities. This value is then discounted at the risk free rate. In the case of American options we take the maximum of the payoff of that node or the option's price (discounted expected value) else we would make a loss. Essentially, if the payoff is greater then the option's price at a certain time step the holder could choose to exercise and make a profit. To tackle this, we take the maximum of the two values to remove arbitrage opportunities.

From the examples above, we have ${V^{uu}}_2 = 39,\ {V^{ud}}_2 = 3,\ {V^{dd}}_2 = 0,$ $S_1^u = 120,\ S_1^d = 90,\ S_0 = 100$ & strike price = 105.

Adding to the example: Let the risk free rate r = 0.06, then the respective probabilities

$$qu = \frac{1 + R - d}{u - d} = \frac{1 + 0.06 - 0.9}{1.2 - 0.9} = \frac{8}{15}$$

$$\&$$

$$qd = 1 - qu = 1 - \frac{8}{15} = \frac{7}{15}$$

Then the option's price at

$${V^u}_1 = \frac{1}{1+R}[qu * {V^{uu}}_2 + qd * {V^{ud}}_2] = \frac{1}{1+0.06}[\frac{8}{15} * 39 + \frac{7}{15} * 3]$$

$$= max(20.9433, 15(120 - 105)) = 20.9433$$

$$\&$$

$${V^d}_1 = \frac{1}{1+R}[qu * {V^{ud}}_2 + qd * {V^{dd}}_2] = \frac{1}{1+0.06}[\frac{8}{15} * 3 + \frac{7}{15} * 0]$$

$$= max(1.5094, 0(90 - 105)) = 1.5094$$

Finally, the option's price at

$$V_0 = \frac{1}{1+R}[qu * {V^u}_1 + qd * {V^d}_1] = \frac{1}{1+0.06}[\frac{8}{15} * 20.9433 + \frac{7}{15} * 1.5094]$$

$$= max(10.761, 5) = 10.761$$

8.2 Monte Carlo Simulation

Monte Carlo Simulation's can be used to model the different probabilistic outcomes of a stochastic process numerically. The uses of such a technique can be to understand the risk and uncertainty in predictive modelling. Due to these reasons, it is used in pricing all types of options since the discounted expected future values of an option are used to find its current value.
For pricing vanilla options the Monte Carlo is as follows:

- *Generate a large number of sample paths for the underlying asset. In order to generate each path we discretize the analytical solution of the GBM such that :*

$$S(\triangle_t) = S_0 exp[(\mu - \frac{\sigma^2}{2})\triangle_t + (\sigma\sqrt{(}\triangle_t)Z_i]$$

  .

- *Calculate the option's payoff at the end of each path using the option's exercise condition*

- *Find the average of these payoffs then discount that average by the risk free rate.*

For an American option the optimal exercise time is dependent on an average over future events therefore Monte Carlo simulations for this derivative has a "Monte Carlo on Monte Carlo" feature which increases the computational complexity. In order to overcome this, the Least Sqaures Monte Carlo Method was proposed.

8.2.1 Least squares Monte Carlo

The Least Squares Monte Carlo was proposed by Longstaff & Schwartz in order to optimize the Monte Carlo Simulation for American options, by solving the stopping times that maximises the value of the option at each point along the different paths.This is accomplished by approximating the continuation value(the option's value the buyer would receive if he held onto the option at that time step) using least squares regression over all paths. At each time step, the continuation value can be formulated as follows:

$$c(x, t_i) = \mathbb{E}[V(St_{i+1})|(St_i)] = \sum_{k=0}^{n} \beta_k X_k(x)$$

where β_k are coefficient's obtained by fitting the least squares regression model to the discounted values of the option at the next time step and X_k are the independent stock values at that current time step. The Algorithm goes as follows:

- *Generate a large number of sample paths for the underlying asset from $t = 0$ to $t = N$.*
- *Upon maturity of each path, set the option value as the payoff $V_{(T)} = h(S_T)$*
- *For each path, start from maturity $i = T$ and go backwards in time such that $i = i$ - 1.*
- *For each path, discount the price by multiplying the payoff one time step ahead with the discounted interest rate.*

  $$V(t_i) = \exp(r \triangle t)V(t_{(i+1)})$$

  .
- *Solve the regression coefficients β_k using least squares fit on the discounted option values $V(t_i) = \exp(r \triangle t)V(t_{(i+1)})$.*
- *For each path, compute the continuation value by inputting the current stock values X_k into the regression equation.*
- *For each path, compare the option's payoff $h(S_T)$ with the continuation value C_t. The maximum of the two values will be the value of the option.*

- *Go back to step 3, until i=1*
- *For each path, discount the price*

$$V(t_0) = \exp(r \triangle t) V(t_{(1)})$$

.

- *Hence, today's price is the average over all paths:*

$$V(S_0) = \frac{1}{no\ of\ paths} \sum_{j}^{no\ of\ paths} V(t_0)$$

The main advantage of this technique is that it reduces the time complexity of a standard Monte Carlo simulation for pricing American options. Thus, when it comes to pricing higher dimensional American options the algorithm in question will perform far more better.

8.3 Finite Difference Method

The Lattice method & Least Squares Monte Carlo method are very intuitive techniques which provide reasonable approximations for the fair value of an American option. However, in terms of accuracy Finite Difference Methods are seen as the superior methodology[4] for pricing one dimensional American options.
Moreover, when American options are priced in a professional environment e.g. an investment bank, Finite Difference Methods are the most commonly used methods[33].

8.3.1 What are Finite Difference Methods?

Finite Difference Methods are discretizations used for solving differential equations i.e. Ordinary & Partial Differential Equations (ODE's & PDE's). We accomplish this by using difference equations (recurrence relations) in order to approximate the differential equation. Essentially, we reduce the problem from continuous time (differential equations) to discrete time (difference equations) in order to make the problem possible to solve.
Finite Difference Methods convert non linear PDE's into a system of equations that can be solved using matrix algebra. The conversion from a differential equation to a system of algebraic equations makes the problem of finding a numerical solution for a given PDE feasible.
Moreover,it cannot be stressed enough that Partial Differential Equations are very hard, if not impossible to solve analytically under certain circumstances due to the increased complexity that comes by introducing new independent variables and boundary conditions. For pricing American options, the Black Scholes PDE derived earlier is an example of one of these impossible solutions.‘

8.3.2 Finite Difference Methodologies

Let us consider the parabolic partial differential equation:

$$\frac{dv(t,x)}{dt} = a(t,x)\frac{d^2v(t,x)}{dx^2} + b(t,x)\frac{dv(t,x)}{dx} + c(t,x)v(t,x) + d(t,x)$$

To solve the equation above, we need to apply some constraints so we may create a discrete grid for where the solution lies. The boundary conditions are as follows:

$$v(T, x) = f(x)$$
$$v(t, x_l) = f(l_t)$$
$$v(t, x_u) = f(u_t)$$

where f is the terminal condition, $f(l_t)$ is the lower boundary condition, $f(u_t)$ is the upper boundary condition.
For derivatives pricing, we have a very similar PDE which governs the evolution of an option's price known as the Black Scholes PDE which we derived earlier.

$$\frac{dV}{dt} + \frac{1}{2}\sigma^2 S^2 \frac{d^2V}{dS^2} + rS\frac{dV}{dS} - rV = 0$$

We can write the Black Scholes coefficients as:

$$a(t, S) = -\frac{1}{2}\sigma^2 S^2 \;\; b(t, S) = -rS \;\; c(t, S) = r \;\; d(t, S) = 0$$

Our objective is to find a solution for the PDE $u(t, z)$ with respect to the boundary conditions. The boundary conditions for each type of option differs since the contractual agreement for every option is somewhat unique. For example the boundary conditions for a European put option are

$$u(T, z) = h(z)$$
$$u(t, z_l) = \exp(-r(T - t))K$$
$$u(t, z_u) = 0$$

where $u(T, z)$ is the payoff (terminal condition),$u(t, z_u)$ is the upper bound since the stock price is higher then the strike price the option is worthless therefore upper bound is 0, $u(t, z_l)$ is the lower bound since the stock price is close to zero therefore the payoff will be an amount close to K.

A visual description of the grid can be described as the following:

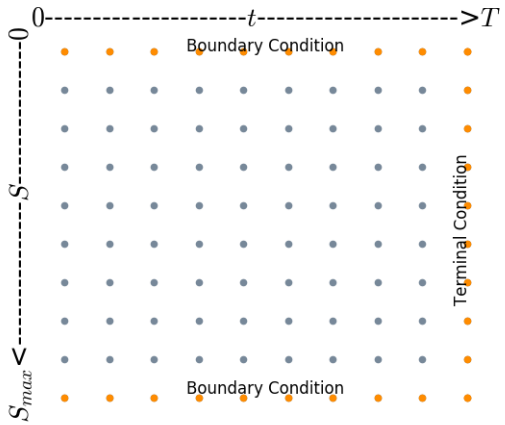


Figure 13: Boundary diagram. Ref[28]

We can solve the PDE numerically on the rectangular grid above using three different methodologies i.e. Explicit Method, Implicit Method, Crank-Nicolson Method.
For all three methods we begin by discretizing the independent variables within the PDE(stock and time values).

$$\triangle t = \frac{T}{i_{max}} \qquad t_i = i \triangle t \qquad i = 0....i_{max}$$

$$\triangle S = \frac{x_u - x_l}{j_{max}} \qquad x_j = x_l + j \triangle x \qquad j = 0....j_{max}$$

8.3.3 Explicit Method

To numerically compute $v(t_i, x_j)$ we use Taylor's expansion to approximate the derivatives in the Black Scholes PDE in the following way:

$$\frac{dv(t_i, x_j)}{dt} \approx \frac{v_{i,j} - v_{i-1,j}}{\triangle t}$$

$$\frac{dv(t_i, x_j)}{dx} \approx \frac{v_{i,j+1} - v_{i,j-1}}{2 \triangle x}$$

$$\frac{d^2v(t_i, x_j)}{dx^2} \approx \frac{v_{i,j+1} - 2v_{i,j} - v_{i,j-1}}{(\triangle x)^2}$$

Substituting these approximations into the Black Scholes PDE we get the following difference equation:

$$\frac{v_{i,j} - v_{i-1,j}}{\triangle t} = a_{ij}\frac{v_{i,j+1} - 2v_{i,j} - v_{i,j-1}}{(\triangle x)^2} + b_{ij}\frac{v_{i,j+1} - v_{i,j-1}}{2 \triangle x} + c_{ij}v_{ij} + d_{ij}$$

The equation above looks very complicated to compute. Hence, simplifying and re-arranging the equation we get:

$$v_{i-1,j} = A_{i,j}v_{i,j-i} + B_{i,j}v_{i,j} + C_{i,j}v_{i,j+1} + D_{i,j}$$

$$where$$

$$A_{i,j} = \frac{\triangle t}{\triangle x}(\frac{b_{i,j}}{2} - \frac{a_{i,j}}{\triangle x}) \quad B_{i,j} = 1 - \triangle t c_{i,j} + \frac{2 \triangle t a_{i,j}}{(\triangle x^2)}$$

$$C_{i,j} = -\frac{\triangle t}{\triangle x}(\frac{b_{i,j}}{2} + \frac{a_{i,j}}{\triangle x}) \qquad D_{i,j} = - \triangle t d_{i,j}$$

A visual representation of the above can be shown by the following:

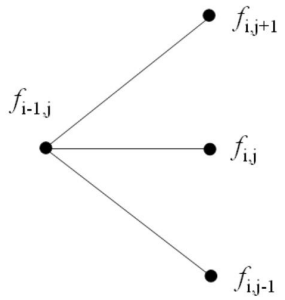


Figure 14: Explicit Method Diagram. Ref[29]

Moreover, we can calculate the extreme boundaries of the rectangular grid using the boundary conditions:

- Right side of the grid will be the payoff: $v_{imax,j} = f_j$
- Top area of the grid will be the Upper Boundary Condition: $v_{i,jmax} = f_{u,i}$
- Bottom area of the grid will be the Lower Boundary Condition: $v_{i,0} = f_{l,i}$

Suppose we have a European put option with stock price $S_o = 100$, strike price = 150. The upper boundary will then be 0, lower bound will be the strike price and the terminal condition will be the payoff.
A visual representation can be shown below:

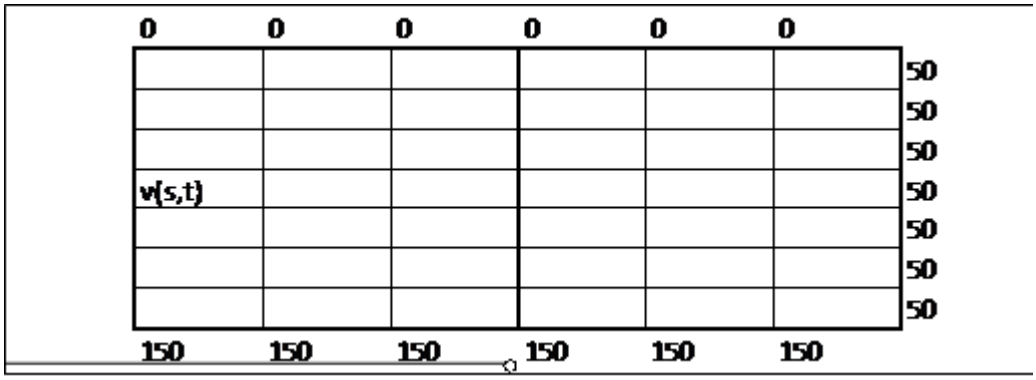


Figure 15: Explicit Boundary example

Consequently, the PDE can be solved in the following manner:

- *Populate points on the right hand column for* $j = 0, ..., j_{imax}$ *using the terminal boundary condition.*
- *Populate points on the top and bottom rows for* $i = 0, ..., i_{max} - 1$ *using the option's respective boundary conditions.*
- *Repeat the following step* i_{max} *times, starting with* i=i_{max} *and finishing when* $i = 1$. *For each* $j = 1, ...j_{max} - 1$, *compute* $v_{i-1,j}$ *using*

$$v_{i-1,j} = A_{i,j}v_{i,j-i} + B_{i,j}v_{i,j} + C_{i,j}v_{i,j+1} + D_{i,j}$$

- *Price of the option today is given by v(0,*S_o*)* using Linear Interpolation since (0,S_o) might be a point on the rectangular grid.

For convenience, we can express the explicit method with the following matrix notations.

$$v_{i-1} = A_i v_i + w_i$$

$$where$$

$$v_i = \begin{pmatrix} v_{i,1} \\ \cdot \\ \cdot \\ \cdot \\ v_{i,j_{max}-1} \end{pmatrix} \qquad w_i = \begin{pmatrix} D_{i,1} + A_{i,1} f_{l,1} \\ D_{1,2} \\ \cdot \\ \cdot \\ D_{i,j_{max}-1} \\ D_{i,j_{max}-1} + C_{i,j_{max}-1} f_{u_i} \end{pmatrix}$$

$$A_i = \begin{pmatrix} B_{i,1} & C_{i,1} & 0 & & 0 & \cdots & 0 \\ A_{i,2} & B_{i,2} & C_{1,2} & & 0 & \cdots & 0 \\ 0 & A_{i,3} & B_{i,3} & & C_{i,3} & \ddots & \vdots \\ 0 & 0 & \ddots & & \ddots & \ddots & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \ddots & C_{i,j_{max}-2} \\ 0 & 0 & \cdots & 0 & A_{i,j_{max}-1} & & B_{i,j_{max}-1} \end{pmatrix}$$

Figure 16: Explicit Matrix

8.3.4 Extension to American options

In chapter 4, we discussed the Linear Complementary Problem for pricing American put options.

$$Lv(t,x) \leq 0, V(t,x) - g(t,x) \geq 0$$
$$(Lv(t,x))(v(t,x) - g(t,x)) = 0$$

where $Lv(t,x)$ is the Black Scholes PDE and $g(t,x)$ is the payoff. To solve American options using Finite Difference Method we inherently need to use a combination of the Linear Complementary Problem along with the Explicit method.

Similar to the Explicit Method we need look for a solution on the rectangular grid [0,T] with boundary conditions x_u & x_l. Additionally, an extra condition needs to be added to the boundary conditions such that the if the payoff is greater then the option price, we use the payoff to define the boundaries. Hence, the boundary conditions can be defined by the following:

$$v(T,x) = max(f(x), g(T,x))$$
$$v(t,x_l) = max(f_l(t), g(t,x_l))$$
$$v(t,x_u) = max(f_u(t), g(t,x_u))$$

To find the fair value of the option v(t,x), either $Lv(t,x) = 0$ meaning the PDE is satisfied or $v(t,x) = g(t,x)$.Therefore, an added boundary condition for American options will be the payoff $g(t,x)$. This is known as the free boundary condition.

For the purposes of solving American options we discretize $Lv(t,x)$ using Explicit difference equations to obtain

$$Lv(t_i,x_j) = v_{i-1,j} - (A_{i,j}v_{i,j-1} + B_{i,j}v_{i,j} + C_{i,j}v_{i,j+1} + D_{i,j})$$

The coefficients $A_{i,j}, B_{i,j}, C_{i,j}, D_{i,j}$ are given in the explicit section. Substituting $Lv(t,x)$ into the Linear Complementary Formulation we get:

$$v_{i-1,j} - (A_{i,j}v_{i,j-1} + B_{i,j}v_{i,j} + C_{i,j}v_{i,j+1} + D_{i,j}) \geq 0, \quad v_{i-1,j} - g_{i-1,j} \geq 0$$
$$(v_{i-1,j} - (A_{i,j}v_{i,j-1} + B_{i,j}v_{i,j} + C_{i,j}v_{i,j+1} + D_{i,j}))(v_{i-1,j} - g_{i-1,j}) = 0$$

With boundary conditions:

$$v_{i_{\max},j} = \max(f_j, g_{i_{\max},j})$$
$$v_{i-1,0} = \max(f_{l,i-1}, g_{i-1,0})$$
$$v_{i-1,j_{\max}} = \max(f_{u,i-1}, g_{i-1,j_{\max}})$$

We can now solve the problem explicitly using the following procedure:

- *Populate points on the right hand column for* $j = 0, ..., j_{imax}$ *using the terminal boundary condition* $(v_{i_{max},j}) = max(f_j, g_{i_{max},j})$.
- *Populate points on the bottom and top rows for* $i = 0, ..., i_{max} - 1$ *using the option's respective boundary conditions* $v_{i-1,0} = max(f_{l,i-1}, g_{i-1,0})$ $v_{i-1,j_{max}} = max(f_{u,i-1}, g_{i-1,j_{max}})$.
- *Repeat the following step* i_{max} *times, starting with* i=i_{max} *and finishing when* $i = 1$. *For each* $j = 1, ...j_{max} - 1$, *compute* $v_{i-1,j}$ *using*

$$v_{i-1,j} = max[A_{i,j}v_{i,j-i} + B_{i,j}v_{i,j} + C_{i,j}v_{i,j+1} + D_{i,j}, g_{i-1,j}]$$

- *Price of the option today is given by v(0,*S_o*)* using Linear Interpolation since (0,S_o) might be a point on the rectangular grid.

Below is a 3D graph for American put options evaluated using the Finite Difference method with different stock prices and time steps with the same expiry date and underlying asset: The parameters are as follows:

$$S0 = 0, 0.025, 0.05, ...1, r = 0.05, \sigma = 0.25, T = 1, 0.888, 0.777, ...0,$$

$$K = 0.5, UB = 2 * S0, LB = 0., imax = 3000, jmax = 200$$

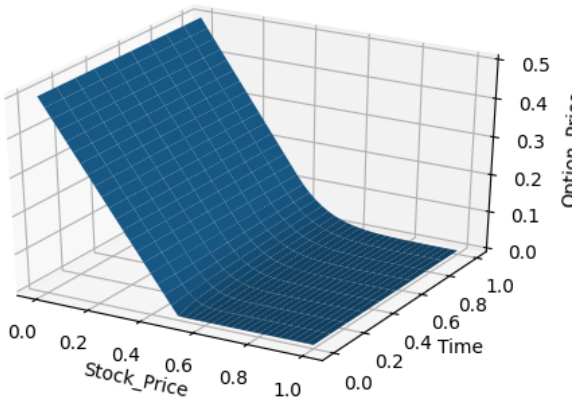


Figure 17: American Put Option

8.4 Deep Gelarkin Method

In this section, we will discuss a new paradigm for solving PDE's, in particular pricing American options under the Black Scholes PDE. The combined efforts of Gelarkin Methods and Deep Neural Networks ushers in this new occurrence. The Deep Gelarkin Method consists of Deep Neural Networks which evaluates the Black Scholes PDE through heavy computational procedures, therefore allowing the pricing for not only one dimensional American options but also multi dimensional American options.

8.4.1 Gelarkin Methods

In finite difference methods, we discretize the Black Scholes PDE on a temporal and spacial grid with boundary conditions. We solve the PDE by inductively walking backwards in time, until we reach the present day where the PDE is valuated to price the fair value of the option.
On the other hand, Gelarkin Methods take an alternative approach to solving the PDE. We are given a linear set of basis functions on the same domain as the PDE with our goal being to find a linear combination of these basis functions that allows us to approximate the PDE in question.
To put it simply, we are trying to solve the equation $F(x) = y$ for x where x and y are members of spaces of functions X and Y respectively. Also suppose ϕ_i and ψ_i are independent bases on X and Y. Ref[30] Then, according to Gelarkin's method, an approximation for x could be given by

$$x_n = \sum_{i=0}^{n} a_i \phi_i$$

Given a_i satisfies the conditions of the equations we can solve the equation by substituting the linear combination into some non linear function to solve for y in the following way:

$$F(\sum_{i=0}^{n} a_i \phi_i), \psi_i) = (y, \psi_i)$$

The above process lays the foundations for which the Deep Gelarkin Method is built upon as it shares many similarities to the inner mechanisms of Neural Networks i.e weights and non linear activation functions.

8.4.2 Free Boundary PDE

As stated above, our previous way for solving the PDE's was through a discretized mesh grid. This method while good becomes computationally intractable as the spacial dimensions of the of the PDE increases, i.e. the amount of underlying assets for an option increases. Therefore, we propose a new way for solving PDE's by changing the grid from a mesh-formed grid to a mesh-free grid.

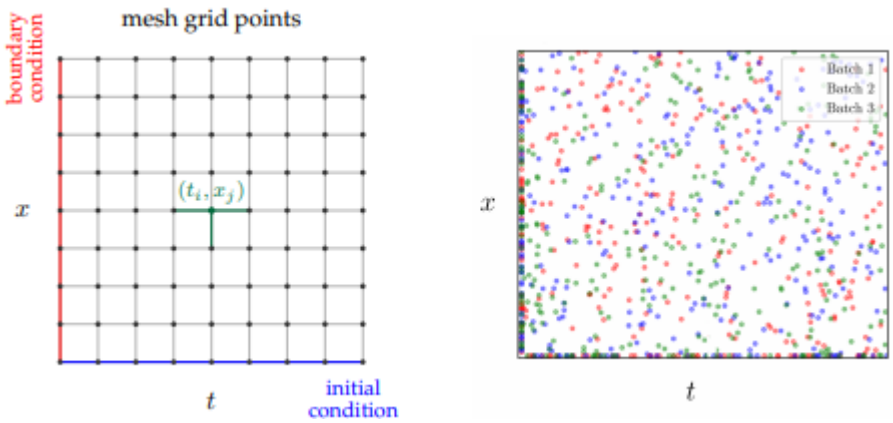


Figure 18: Different mesh grids. Ref[30]

From the figure above, we can see the grid on the left has a mesh while the grid of the right is mesh-free.
To solve the PDE on a mesh free grid we use a Deep Neural Network to train sampled time and space points at random. We start by feed forwarding these random points into the Neural Network until we reach the output layer where the loss function of the Deep Neural Network is evaluated using the differential operator, initial condition and boundary condition. We minimize the loss using stochastic gradient descent at randomly sampled spacial points in order to approximate a solution v for the given PDE.

The fair value $v(t,x)$ of an American option can be satisfied using the differential operator, boundary condition and initial condition.

- The differential operator is given by the Black Scholes PDE $\frac{dV}{dt} + \frac{1}{2}\sigma^2 S^2 \frac{d^2V}{dS^2} + rS\frac{dV}{dS} - rV = 0$.
- The boundary condition is given by $v(t,x) \geq g(t,x)$ where $v(t,x)$ is the option's price and $g(t,x)$ is the option's payoff throughout the lifetime of the option
- The initial condition is given by $v(T,x) = g(T,x)$ where $v(T,x)$ and $g(T,x)$ are the option's price and payoff evaluated at the expiration date.

"The free boundary set F = $(v(t,x) = g(t,x))$ where $v(t,x)$ satisfies the PDE above the free boundary set F and $v(t,x) = g(t,x)$ satisfies the PDE below the free boundary set F. To solve the PDE using the Deep Learning algorithm we simulate points above and below the free boundary and use an iterative method to address the free boundary. The method entails minimizing the following objective function to address the free boundary." Ref[5]

$$J(f;\theta,\hat{\theta}) = || \frac{df}{dt}(t,x;\theta) + \frac{1}{2}\sigma^2 S^2 \frac{d^2 f}{dx^2}(t,x;\theta) + rS\frac{df}{dx}(t,x;\theta) - rf(t,x;\theta) ||^2$$

$$+ || max(g(x) - f(t,x;\theta), 0) ||^2$$

$$+ || f(T,x;\theta) - g(x) ||^2$$

8.4.3 Implementation of DGM

Given the objective function we can implement the deep learning algorithm for pricing American options as follows:

- "Step 1 - Generate a random batch of spacial and temporal points $\mathrm{B}^1 = (t_m, x_m)_{m=1}^M$ from some probability distribution where the range is from 0,...,T. Select the points $\hat{B^1}$ =$[(t,x) \in B^1 : f(t,x;\theta_n) > g(x)]$". Ref[5]

- "Step 2 - Generate a random batch of spacial and temporal points $\mathrm{B}^2 = (t_m, z_m)_{m=1}^M$ from some probability distribution where the range is from 0,....,T. Select the points $\hat{B^2}$ =$[(\tau,z) \in B^2 : f(\tau,z;\theta_n) \leq g(z)]$". Ref[5]

- "Step 3 - Generate a random batch of spacial and temporal points $\mathrm{B}^3 = (w_m)_{m=1}^M$ from some probability distribution where the range is from 0,....,T." Ref[5]

- "Step 4 - Approximate $J(f;\theta,\hat{\theta})$ as $J(f;\theta,S_n)$ at the randomly sampled points where $S_n = [\hat{B^1},\hat{B^2},\hat{B^3}]$:" Ref[5]

$$J(f;\theta,S_n) = \frac{1}{\mid \hat{B^1} \mid} \sum_{t_m,x_m \in \hat{B^1}} (\frac{df}{dt}(t,x;\theta) + \frac{1}{2}\sigma^2 S^2 \frac{d^2 f}{dx^2}(t,x;\theta) + rS\frac{df}{dx}(t,x;\theta) - rf(t,x;\theta))^2$$

$$+ \frac{1}{\mid \hat{B^2} \mid} \sum_{t_m,x_m \in \hat{B^2}} max(g(z_m - f(\tau_m,z_m;\theta_n),0)^2$$

$$+ \frac{1}{\mid \hat{B^3} \mid} \sum_{w_m \in \hat{B^3}} (f(T,w_m;\theta) - g(w_m))^2$$

- Step 5 - Take a gradient descent step for the random batch S_n:

$$\theta_{n+1} = \theta_n - \alpha_n \nabla_\theta J(f;\theta,S_n)$$

- Step 6 - Repeat until convergence criteria is satisfied.

8.4.4 Architecture of DGM

The architecture for the hidden layers of the Deep Gelarkin Method are similar to that of an LSTM. However, the Deep Gelarkin method is not used for sequential data therefore a recurrence 'hidden' state does not exist. To put it simply, the network's overall architecture is that of a Feed Forward Neural Network where the hidden layers internal architecture consists of a combination between LSTM and Highway Networks.
A visual representation of the overall architecture can be seen as follows:

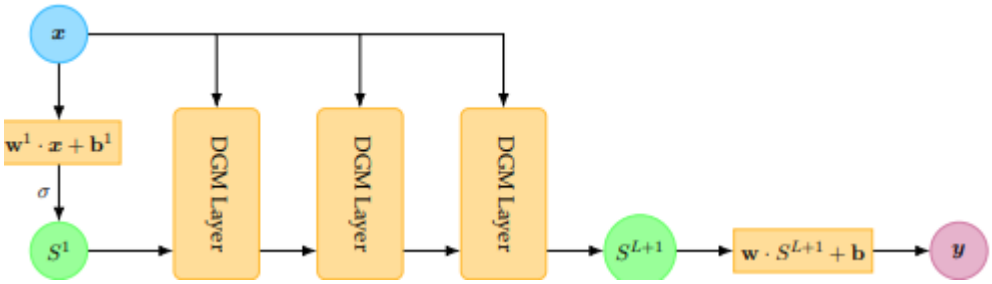


Figure 19: DGM whole architecture. Ref[30]

In each DGM layer, a mini batch of inputs along with the outputs of the previous layer are transformed through a series of element wise operations closely resembling LSTM and Highway Networks . Below are the equations which are used within each DGM layer:

$$S^1 = \sigma(w^1 \cdot x + b^1)$$

$$Z^l = \sigma(u^{z,l} \cdot x + w^{z,l} \cdot S^l + b^{z,l})$$
$$G^l = \sigma(u^{g,l} \cdot x + w^{g,l} \cdot S^l + b^{g,l})$$
$$R^l = \sigma(u^{r,l} \cdot x + w^{r,l} \cdot S^l + b^{r,l})$$
$$H^l = \sigma(u^{h,l} \cdot x + w^{h,l} \cdot (S^l \odot R^l) + b^{h,l})$$
$$S^{l+1} = (1 - G^l) \odot H^l + Z^l \odot S^l$$
$$f(t, x, \theta) = w.S^{l+1} + b$$

We use this internal architecture to solve the vanishing gradient problem by determining how much information gets passed onto the next layer through transformation gates and forget gates. Also, "the repeated element wise multiplication of non - linear activation functions allows for sharp turns which is required due to the final condition $u(T, x) = max(g(x), 0)$" Ref[5] along with the PDE changing in certain spacial regions.
A visual representation of the internal architecture can be shown below:

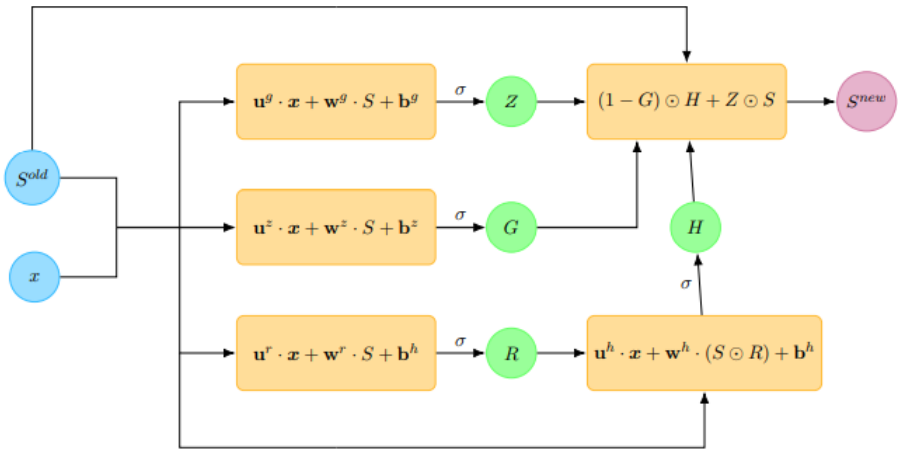


Figure 20: DGM Internal architecture. Ref[30]

A key observation to note is that the LSTM internal architecture doesn't used a 'cell state' but rather replaces it with another stock/time point.

8.5 Numerical Results

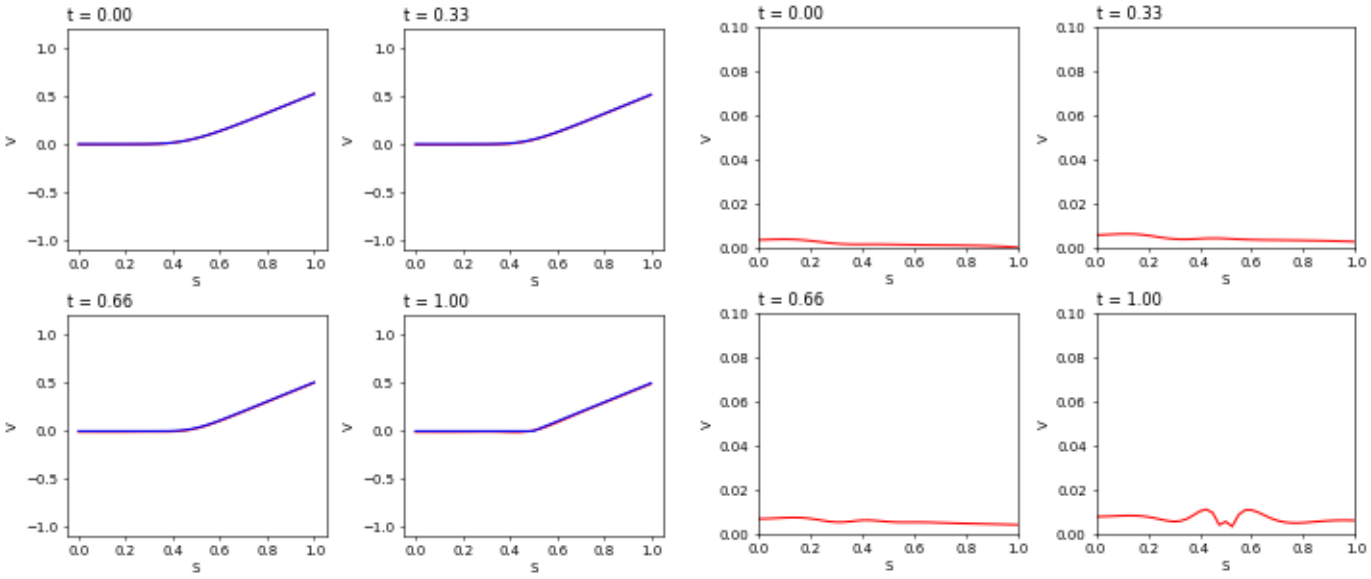
We will now analyse the various different option prices from the Neural Network evaluated at different time steps by comparing our Neural Network's results to that of the Finite Difference Method for American put options. American call options obey the same rules as an European call options so we can use the Black Scholes analytical formula as a ground truth to evaluates the Network.
Also, we will analyse how the violation of the boundary condition effect the option's price.
In addition, we will evaluate the performance of the Neural Network when more samples are entered into the model.

8.5.1 American call option Results

To price American call options we used the following parameters for the Neural Network:

- S0=0,0.025,0.05,...1, r=0.05 ,σ=0.25, T=0,0.111,0.222,...1
- K=0.5, no of layers=3, no-of-nodes-per-layer=50, learning-rate=0.001
- Learning-rate-decay-steps=10000, learning-decay-rate=0.9, steps-per-sample=10
- S-interior=6500, S-initial=6500,no-of-epochs=1000

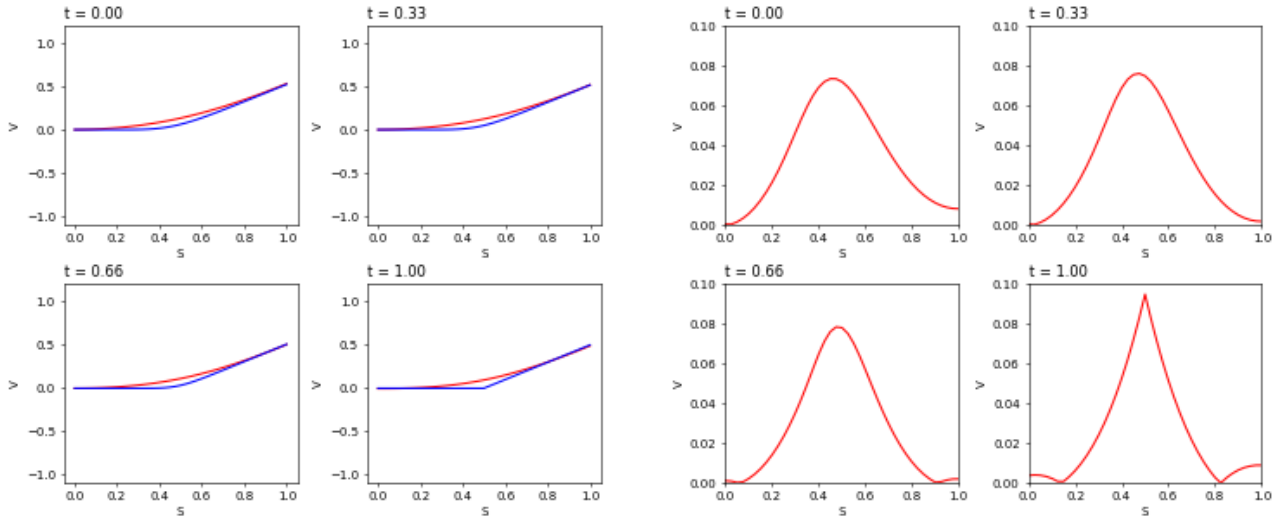
We train the Neural Network 5 times in order to avoid converging at a local minima. Also, we multiplied the free boundary condition by a factor of 1, 100 and 1000 and see how well the Network evaluates the option price.
The two diagrams below show the Neural Network's results when all loss terms are evaluated equally against the Black Scholes formula and measures the absolute error of Network's results at four discrete time steps. It should be noted that in the Results Comparison diagram the blue line represents the Black Scholes analytical formula results for American calls and the Finite Difference results for American puts while the red line represents the Neural Network results.



(a) Results Comparison (b) Absolute Error Comparison

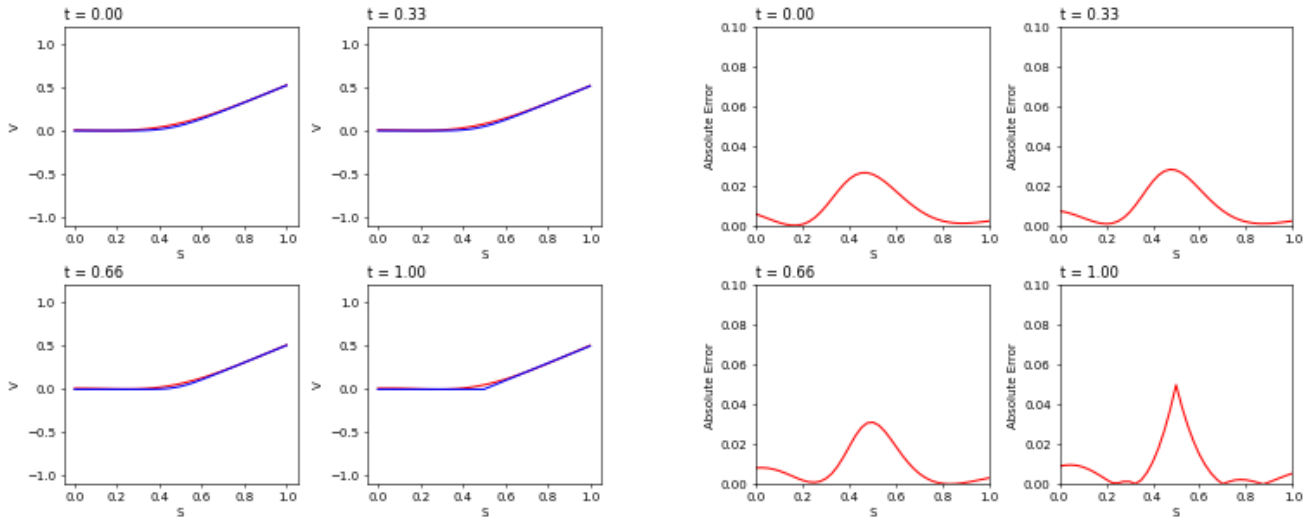
Figure 21: Comparison and Error Figures

From the results above, we can see when all loss terms are evaluated equally for an American Call we get very minimal error. At worst the error slightly increase around at the money when T=1.
However, when the boundary condition is given more emphasis, the Network performs worse as shown below:



(a) Results Comparison (L2·100) (b) Absolute Error Comparison (L2·100)

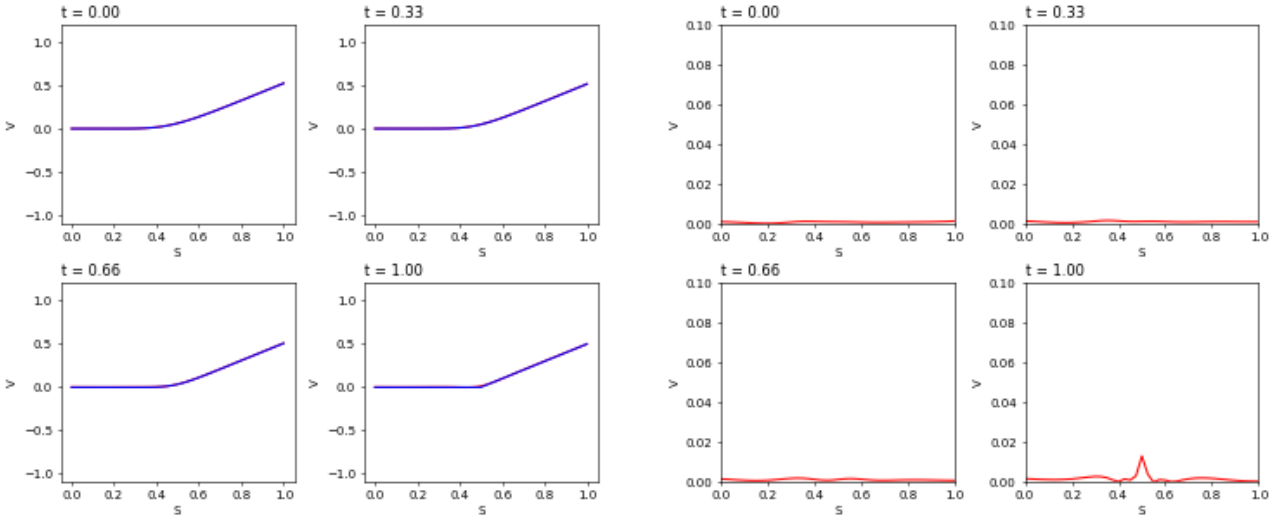
Figure 22: Comparison and Error Figures (L2·100)



(a) Results Comparison (L2·1000) (b) Absolute Error Comparison L2·1000

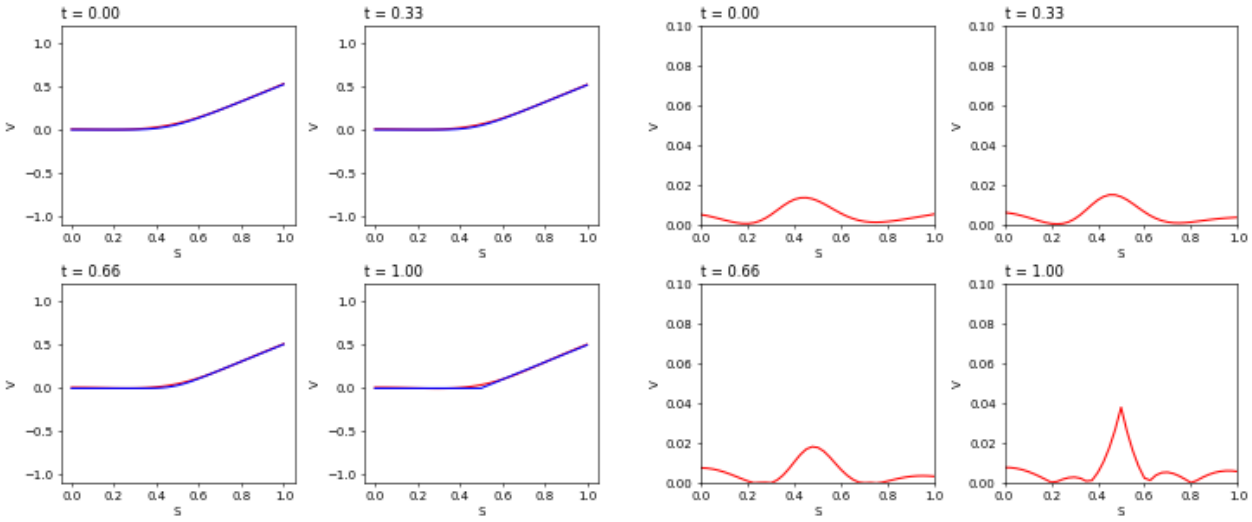
Figure 23: Comparison and Error Figures (L2·1000)

This makes intuitive sense since American call options have almost no violation of the free boundary condition since the stock price is always increasing on average, therefore the best time to exercise the option is at expiry. Thus adding extra weight onto a meaningless loss term will yield worse results. We will now re-evaluate the Network by increasing the samples from 6500 to 15000 and observe the results.



(a) Results Comparison (L2·1) (b) Absolute Error Comparison (L2·1)

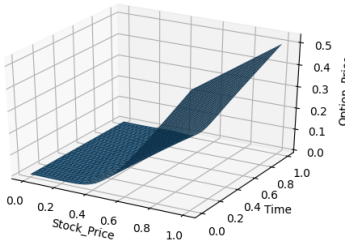
Figure 24: Comparison and Error Figures (L2·1)



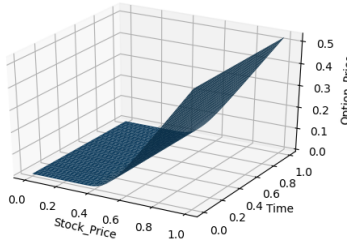
(a) Results Comparison (L2·100) (b) Absolute Error Comparison (L2·100)

Figure 25: Comparison and Error Figures (L2·100)

Analysing the error graph at (L2*1) we can see that increasing the sample size decreases the error at T=1 ever so slightly yet that tiny peak at the money still exists.
Moreover, the error graph at (L2*100) shows a tremendous drop in the error as the sample size increases.
Hence, a reasonable assessment we can make is that increasing the amount of samples decreases the error for American Call options.
Below are two 3D figures for American call option values. The leftmost figure was created by evaluating the Neural Network at 10 different time steps. Whereas, the rightmost figure was created by evaluating the Black Scholes formula at 10 different time steps. Both figures are trained with equal weights on the loss terms and 15k samples.



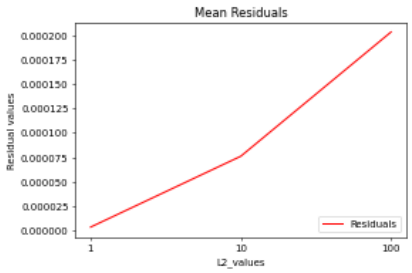
(a) 3D Neural Neural Network Call Option



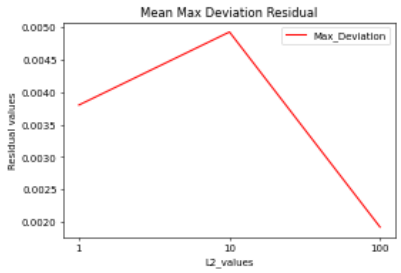
(b) 3D Analytical Solution Call Option

Figure 26: 3D graphs (L2·100)

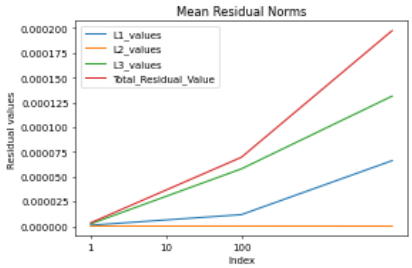
Below are some mean summary statistics regarding the loss terms. These results were summarised after running the Neural Network 5 times.



(a) Total Mean Residual values



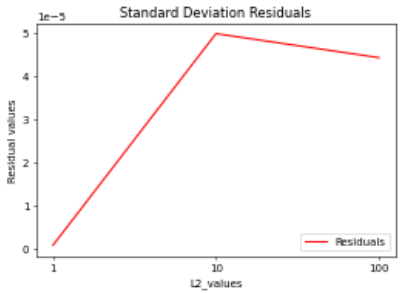
(b) Mean Max L2 Residual value



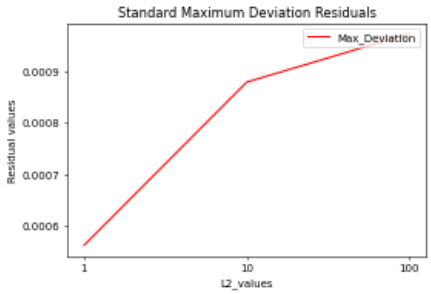
(c) Individual & Total Residual values

Figure 27: Call Option Mean Residual Statistics

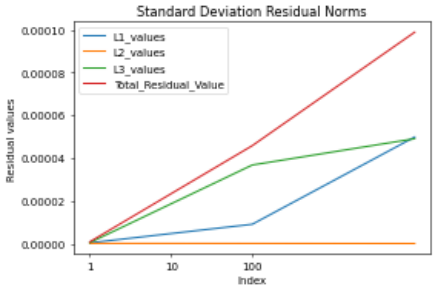
Upon analysing the Total Mean Residual value, we can see that as more weight is put on one loss term the slower the network takes to converge to a minima. Hence, an increase in the residuals value.
However, mean max deviation values takes a massive decrease as the free-boundary term(L2-norm) is multiplied by 1000. A reason for this could be due to the fact the L2 norm being given alot of significance, resulting in a very low chance of being violated.
Additionally, the last graph shows us that while the L2 norm is converging at a faster rate the differential operator and the initial condition are converging much slower. Thus, reducing the performance of the Network as a whole which in turn leads to higher error.
We have also analysed the standard deviations of the Neural Network's performance



(a) Total Standard Deviation Residual values



(b) Standard Max Deviation L2 Residual value



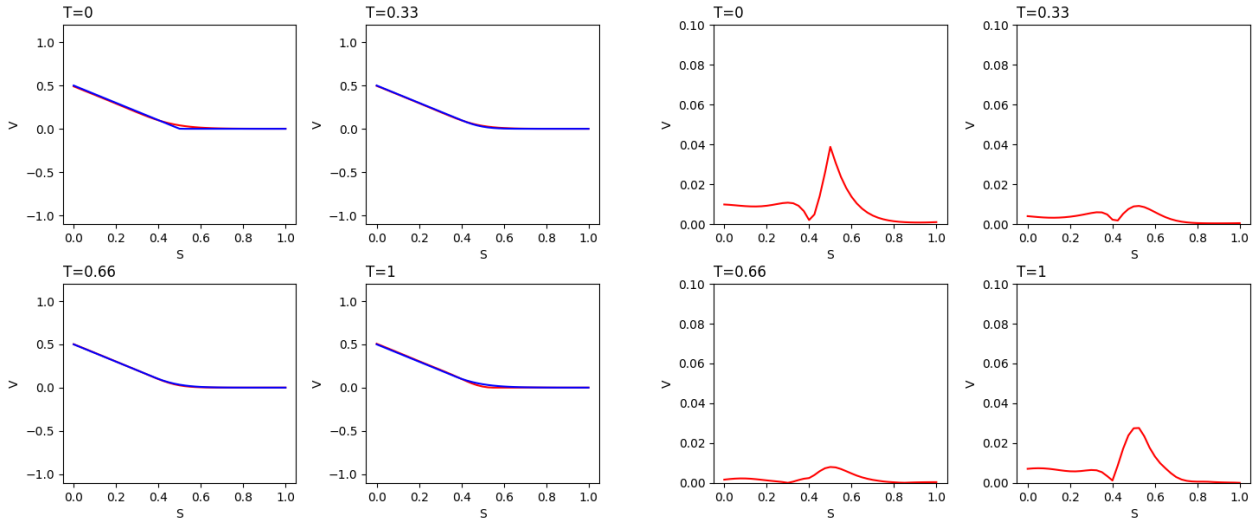
(c) Individual & Total Standard Deviation Residual values

Figure 28: Call Option Standard Deviation Residual Statistics)

The Total Standard Deviation residual graph shows us that by adding extra weight to a single loss term we get more deviation from the Network. Also, Standard Max Deviation increases for the L2 term as more weight is added. Since more weight is added to the L2 error term, more outliers will occur for that individual term.
Finally, as the deviation of the L2 error term drops the deviation of the other two error terms increase resulting in more network errors for American calls.

8.5.2 American Put option Results

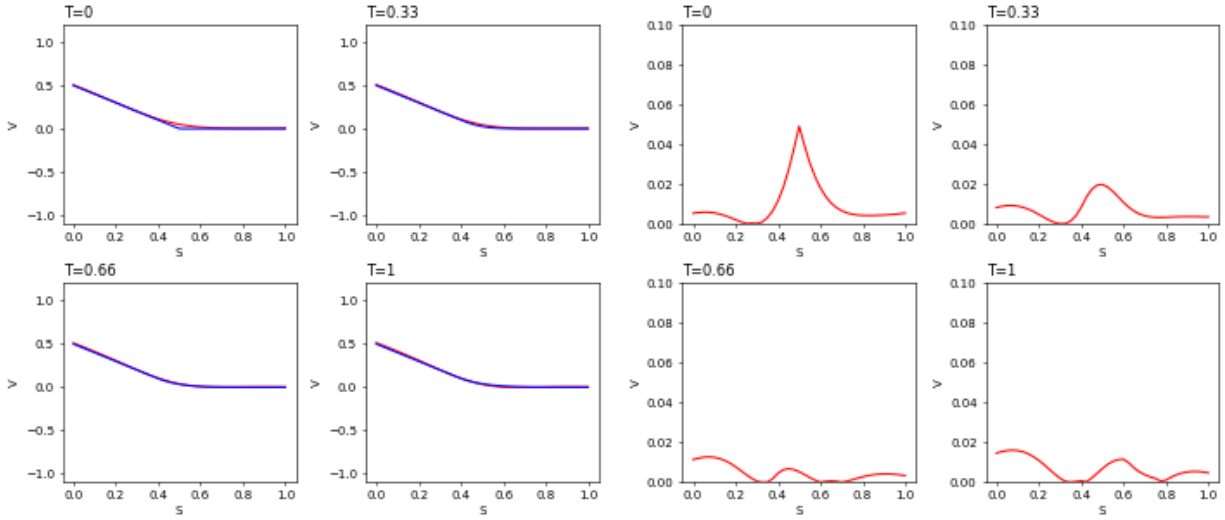
Similar to the American call options we will train the Neural Network to price American put options using 6500 samples. However, to evaluate the Network's performance we will use the Finite Difference Method as the ground truth instead of the Black Scholes analytical formula since American put options have no analytical solution.
The two diagrams below show the Network's performance when all loss terms are weighted equally against the ground truth.



(a) Results comparison (b) Absolute Error Graph

Figure 29: Comparison and Error graphs

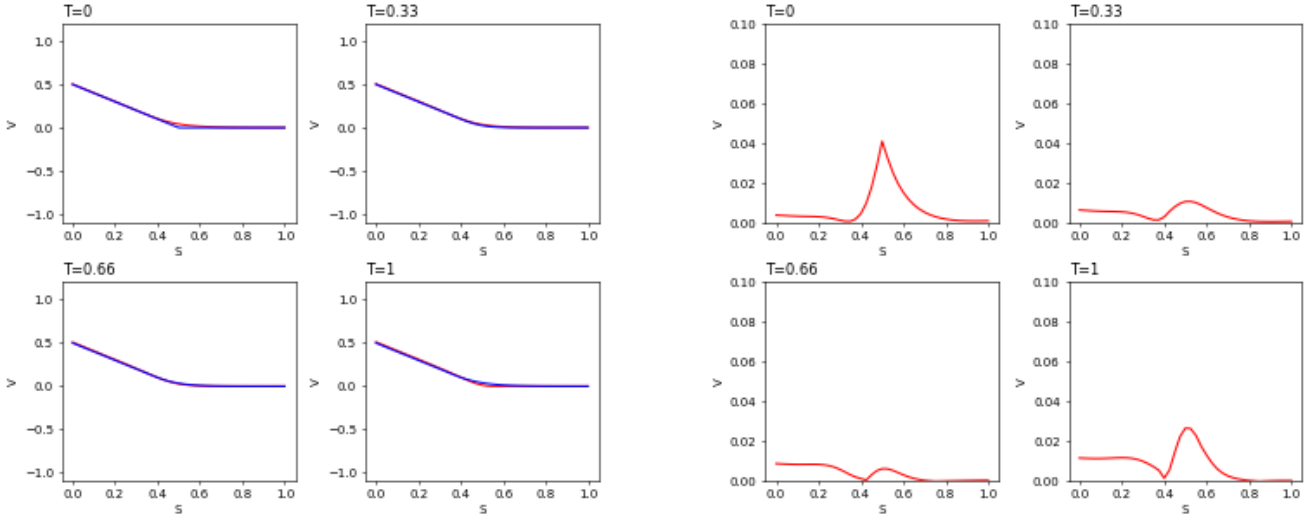
From the results above, we can see when the Neural Network is evaluated against another numerical method instead of an analytical one it tends to perform a little worse, this is to be expected as the boundary condition get violated more often therefore resulting in a higher error also neither method yields the exact solution. However, the overall error is still somewhat low.
For American call options we summarised that adding extra weight on the free boundary increases the error.
However, for an American put option adding some extra weight on the free boundary term reduces the overall error at later time steps. As can be seen below:



(a) Results comparison L2·100 (b) Absolute Error Graph L2·100

Figure 30: Comparison and Error graphs

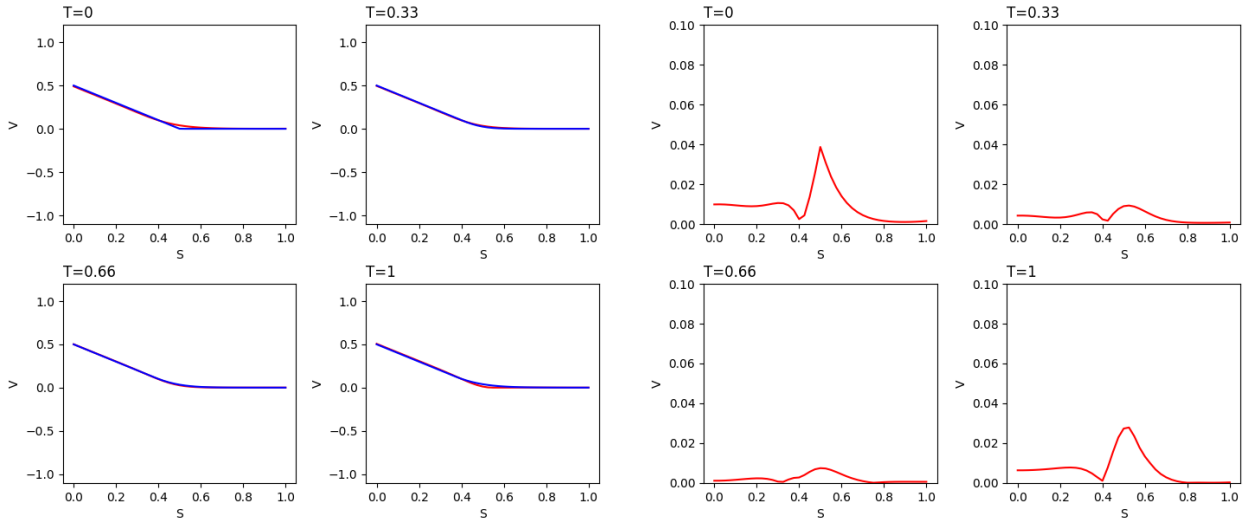
Fundamentally, we can observe that the free boundary loss term has alot more significance when it comes to pricing an American put options due to the Linear Complementarity problem.
Conversely, from the diagrams below we can see that by adding too much significance onto the free boundary loss term by increasing the weight leads to an increase in error.



(a) Results comparison L2·1000 (b) Absolute Error Graph L2·1000

Figure 31: Comparison and Error graphs

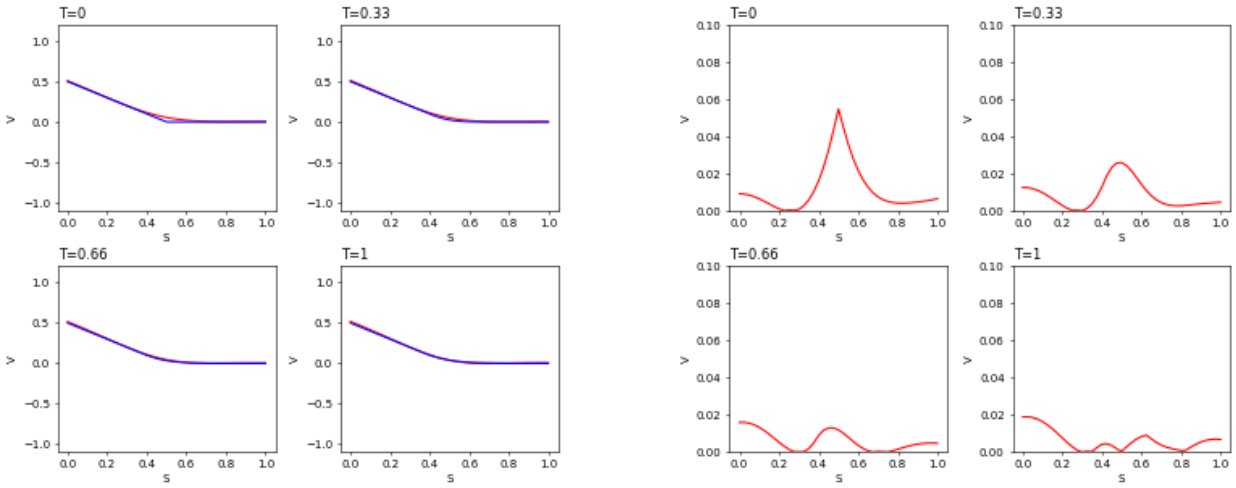
Thereby, we can conclude that just the right amount of weight needs to be added to the error term in order to yield results with minimal error. Additionally, re-training the Network with 15000 samples improves the results when the loss terms are equally distributed as can be shown below.



(a) Results comparison L2·1 (b) Absolute Error Graph L2·1

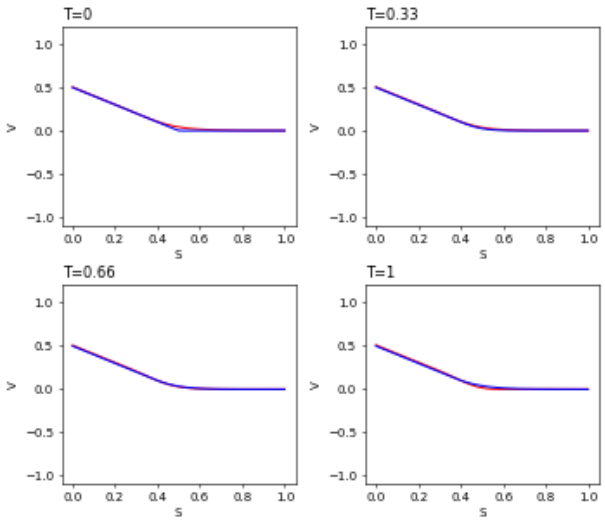
Figure 32: Comparison and Error graphs

Increasing the sample size and the weights significance results in a decrease in the Network's error but the same rule for correct weight adjustment still applies. We can see this from the figures below:

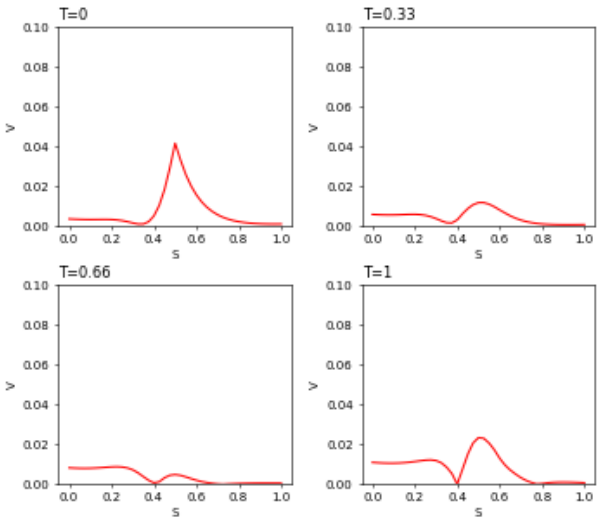


(a) Results comparison L2·100 (b) Absolute Error Graph L2·100

Figure 33: Comparison and Error graphs



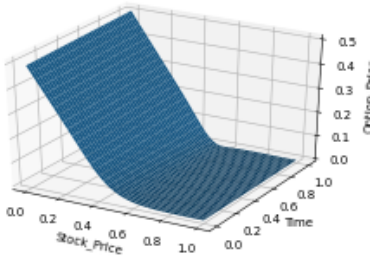
(a) Results comparison L2·1000



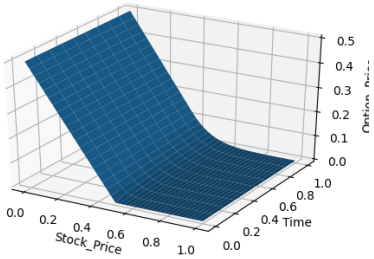
(b) Absolute Error Graph L2·1000

Figure 34: Comparison and Error graphs

We can conclude the Network performed best when the sample size was 15000 and the free boundary term was multiplied by 100.
Below are the 3D figures for American put options. The leftmost figure was created by evaluating the Neural Network at 10 different time steps while the rightmost figure was created by evaluating the Finite Difference method at 10 different time steps.



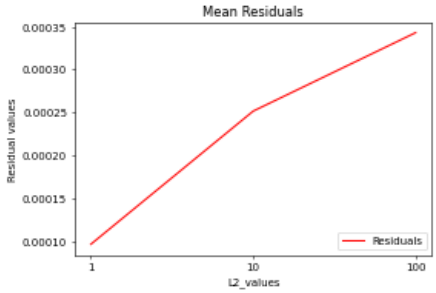
(a) 3D Neural Network Put Option



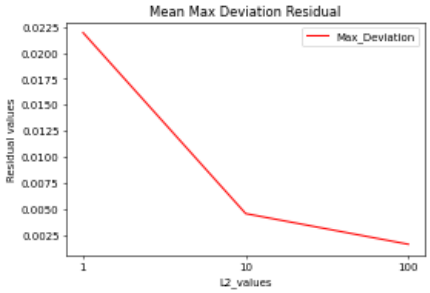
(b) 3D Finite Difference Put Option

Figure 35: 3D figures

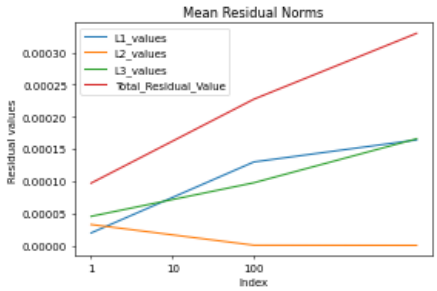
Here are some mean summary statistics for the loss terms. These results were summarised by running the Neural Network 5 times to avoid converging at a local minima.



(a) Total Mean Residual values



(b) Mean Max L2 Residual value

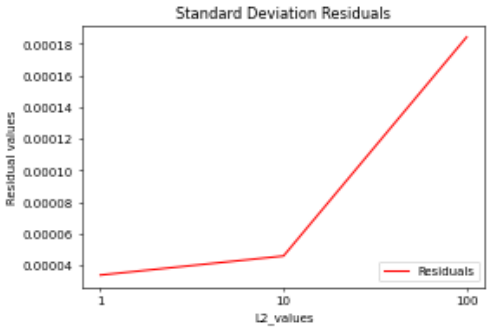


(c) Individual & Total Mean Residual values

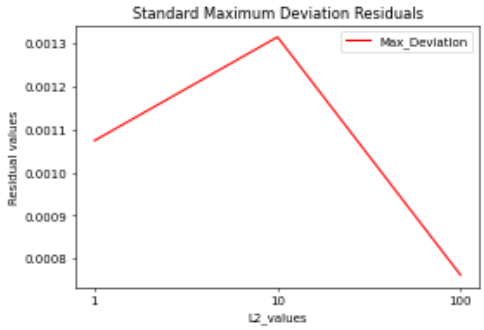
(d) Put Option Mean Residual Statistics

Similiar to the American call options, we can see the Total Mean Residual value increases as more weight is added onto a single loss term which tells us that it is converging slower to a local minima. The Mean max residual value continuously decreases as more weight is added to the free boundary loss term. This tell use the boundary condition is getting violated far less as the weight is increased. Also, we can see that as the free boundary term decreases, a resulting increase occurs within the differential operator and initial operator.
However, for a put option this isn't necessarily a bad thing as more weight on the free boundary loss term leads to less error.

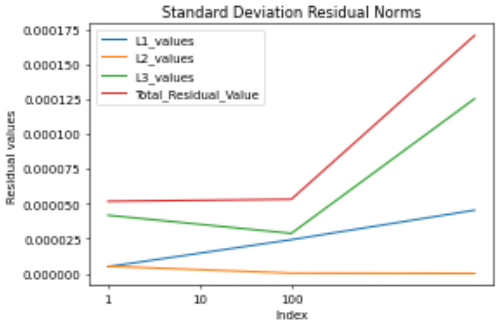
We have also analysed the standard deviations of the Neural Network's performance. The results are shown below:



(a) Total Standard deviation Residual values



(b) Standard Max deviation L2 Residual value



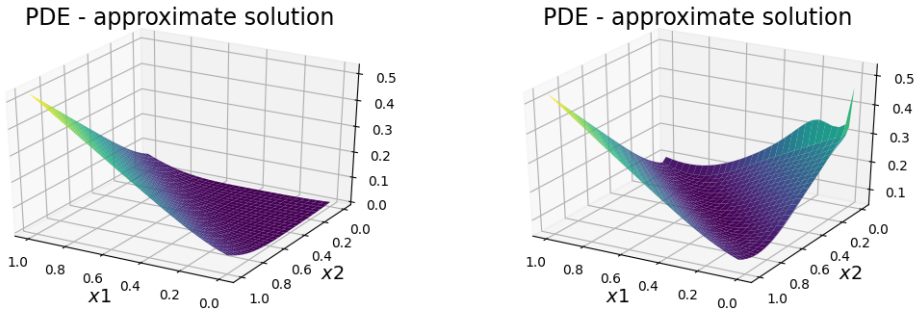
(c) Individual & Total Standard deviation Residual values

(d) Put Option Mean Residual Statistics

Simlar to American call options, Total Standard Deviation residual values increase by adding extra weight to a single loss term, this tell us we get more deviation from the Network. On the other hand, Standard Max Deviation decreases as more weight is added to the L2 term. This tell us as more weight is added to the L2 error term, less outliers will occur for that individual term for a put option.
Finally, as the deviation of the L2 error term drops the deviation of the other two error terms increase. For an American put option this isn't a necessarily bad since the free boundary term needs to have more emphasis.

8.5.3 Multi-Dimensional American option results

Below are 3D plots for American call and put options trained on the Neural Network for the 2 dimensional case where the option is valued on 2 underlying assets except 1.



(a) Multi-Dimensional American call option

(b) Multi-Dimension American put option

Figure 38: 3D graphs (L2·100)

Looking at a bird's eye overview we can see that as both stock prices increase, the value of the call option increases. In contrast, we can see the value of the put option increase as the stock price decreases.
To compare the results of the 2D numerical approximation we could increase the dimensions of the Finite Different Method to two dimensions.
However, if we further increase the dimensions of the Neural Network to 10, 20 or 200 dimension we would need to use the Least Squares Monte Carlo method for comparison.

9 Conclusion

In conclusion, the analysis suggests Deep Neural Networks are suitable numerical methods for pricing American options.
In the case of American calls, it has been made clear that we get our most optimized result by not adding any extra weight to the free boundary loss term. In addition, increasing the weight on the free boundary loss term decreases the Network's performance as the absolute error and summary statistics increase. This results in slower convergence and more deviation.
For American puts, the opposite effect occurred since increasing the weighted significance on the free boundary error term decreased the overall error of the Neural Network.
However, adding too much weight to the free boundary term increased the error once again. This suggests just the right amount of significance needs to be given in order to achieve best results.
Additionally, upon increasing the samples we realise a significant drop in the overall error for both option types. This implies if we place enough sampled points into the Network, our overall error will significantly decrease. This is reinforced by state of the art GPU's and CPU's as they are being developed thereby resulting in less Network error, but more importantly, less computation time.
The decrease in computation time is what separates Neural Networks from other Numerical Methods and therefore could have the potential for being the new methodology adopted by investment banks to price options.
Nevertheless, it should be noted that while Neural Networks have shown good performance for pricing options, small errors still exist in the Network and the free boundary loss term has to be finely tuned in each individual option case which results the user to have some prior knowledge about the problem.
Moreover, as we increase the dimensions of the American option we can see the Neural Network is able to give some approximate result.

10 Future Work

Looking ahead, the power of Neural Network's could be used to analyse the price of Multi-dimensional American options in different dimension cases i.e. 5, 20, 200, 500 dimensions. To compare our results we would use the Least Squares Monte Carlo Method as a ground truth.
To significantly reduce the computational time complexity of increased dimensions a CUDA graphics card can be used. We can also increase the amount of samples inputted into the Network for both the one dimension and Multi-dimension. This will reduce the error while not have a high time complexity payoff if we run the Neural Network with a GPU. The decrease in time complexity is what will give Neural Network's an edge over Monte Carlo Methods for pricing American options. In addition, we could further improve the hidden layer architecture of the Network to stabilize gradient flow thus resulting in lower error.

References

[1] *"The Pricing of Options and Corporate Liabilities"*

Authors = Fischer Black and Myron Scholes.

Publish Year = 1973
https://www.cs.princeton.edu/courses/archive/fall09/cos323/papers/black scholes73.pdf

[2] *"JPMorgan's Equity Derivatives Haul Soars to $1.5 Billion"*

Authors = Michelle F Davis, Donal Griffin, and Max Abelson.

Publish Date = 26/03/2020
https://www.bloomberg.com/news/articles/2020-03-26 /jpmorgan-s-equity-derivatives-haul-said-to-soar-to-1-5-billion

[3] *"Valuing American Options by Simulation: A Simple Least-Squares Approach"*

Authors = Francis A. Longstaf,Eduardo S. Schwartz .

Publish Year = 2001
https://people.math.ethz.ch/ hjfurrer/teaching/ LongstaffSchwartzAmericanOptionsLeastSquareMonteCarlo.pdf

[4] *"Numerical Methods in Finance with C++ "*

Authors = Maciej j. Capinski, Tomasz Zastawniak

Publish Year=2012
https://www-cambridge-org.ezproxy.library.qmul.ac.uk/ core/books/numerical-methods-in-finance-with-c/97CD376C1AB7325C640A7A9C87259E79

[5] *"DGM: A deep learning algorithm for solving partial differential equations"*

Authors = Justin Sirignano,Konstantinos Spiliopoulos

Publish Year=07/09/2018
https://arxiv.org/abs/1708.07469

[6] European Call Option Payoff Diagram
https://www.sec.gov/Archives/edgar/data /1053092/000089109213004721/e53788fwp.htm

[7] European Put Option Payoff Diagram
https://ebrary.net/12884/management/payoff option contracts

[8] Continuous Time Stochastic Processes

Author = Michael J. Phillips
https://qmplus.qmul.ac.uk/pluginfile.php /1793233/mod
resource/content/0/QMUL MTH771P Lecture Notes 07.pdf

[9] Brownian Motion Diagram
https://blogs.ethz.ch/kowalski/2009/02/08/a-smoother-brownian-motion/

[10] Stochastic Processes II
https://ocw.mit.edu/courses/mathematics/18-s096-t
opics-in-mathematics-with-applications-in-finance-fall-2013/video
-lectures/lecture-17-stochastic-processes-ii/

[11] Ito's Calculus
https://ocw.mit.edu/courses/mathematics/18-s096-
topics-in-mathematics-with-applications-in-finance-
fall-2013/video-lectures/lecture-18-ito-calculus/

[12] *"Four Derivations of the Black Scholes PDE"*

Author = Fabrice Douglas Rouah
https://www.frouah.com/finance20notes/Black20Scholes20PDE.pdf

[13] *"Optimal Stopping Times American Options"*
http://www.stat.uchicago.edu/ lalley/Courses/391/Lecture15.pdf

[14] *"American Options"*

Author = Michael J. Phillips
https://qmplus.qmul.ac.uk/pluginfile.php/1793230/ mod
resource/content/0/QMUL MTH771P Lecture Notes 06.pdf

[15] *"American Options"*
https://courses.maths.ox.ac.uk/node/view material/4110

[16] *"Designing Your Neural Networks"*

Author = Lavanya Shukla

Year Published = 23/09/2019
https://towardsdatascience.com/designing-your-neural-networks-a5e4617027ed

[17] *"A Quick Introduction to Neural Networks"*

Author = ujjwalkarn

Year Published = 09/08/2016
https://ujjwalkarn.me/2016/08/09/quick-intro-neural-networks/

[18] *"Stochastic Gradient Descent — Clearly Explained !!"*

Author = Aishwarya V Srinivasan

Year Published = 07/09/2019
https://towardsdatascience.com/-stochastic-gradient-descent-clearly-explained-53d239905d31

[19] *"Understanding Forward propagation:"*

Author = Ayoub Benaissa

Year Published = 30/01/2019
https://medium.com/@bucky01roberts/ understanding-forward-propagation-8e639d1474b7

[20] *"Machine Learning & Deep Learning Fundamentals"*
https://deeplizard.com/learn/playlist/PLZbbT5o s2xq7LwI2y8QtvuXZedL6tQU

[21] *"WHAT IS GRADIENT DESCENT IN MACHINE LEARNING?"*

Author = Saugat Bhattarai

Year Published = 22/06/2018
https://saugatbhattarai.com.np/what-is-gradient-descent-in-machine-learning

[22] *"Understanding LSTM Networks?"*

Author = Colah

Year Published = 27/08/2015
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

[23] *"Lecture 10 — Recurrent Neural Networks"*

Authors = Fei-Fei Li & Justin Johnson & Serena Yeung

Year Published = 04/05/2017
http://cs231n.stanford.edu/slides/2017/cs231n 2017 lecture10.pdf

[24] *"Long short-term memory"* https://en.wikipedia.org/wiki/Long short-term memory/media/File:The LSTM cell.png

[25] *"Highway Networks"*

Authors = Rupesh Kumar Srivastava, Klaus Greff, Jürgen Schmidhuber

Year Published = 03/05/2015
https://arxiv.org/pdf/1505.00387.pdf

[26] *"Review: Highway Networks — Gating Function To Highway (Image Classification)"*

Author = Sik-Ho Tsang

Year Published = 13/02/2019
https://towardsdatascience.com/review-highway-networks-gating-function-to-highway-image-classification-5a33833797b5

[27] *"Option Pricing Using The Binomial Model"*
https://www.goddardconsulting.ca/option-pricing-binomial-index.html

[28] *"On Pricing Options with Finite Difference Methods"*

Author = Quintus Zhang

Year Published = 24/05/2017
https://www.goddardconsulting.ca/option-pricing-binomial-index.html

[29] *"Option Pricing Using The Explicit Finite Difference Method"*
https://www.goddardconsulting.ca/option-pricing-finite-diff-explicit.html

[30] *"Solving Nonlinear and High-Dimensional Partial Differential Equations via Deep Learning"* Authors = Ali Al-Aradi, Adolfo Correia, Danilo Naiff, Gabriel Jardim, Yuri Saporito Year Published = 21/11/2018
https://arxiv.org/abs/1811.08782

[31] *"Deep Learning for Speech Recognition"*

Authors = Open Data Science

Year Published = 12/08/2019
https://medium.com/@ODSC/deep-learning-for-speech-recognition-cbbebab15f0d

[32] *"Deep Learning for NLP: An Overview of Recent Trends"*

Authors = Elvis

Year Published = 24/08/2018
https://medium.com/dair-ai/ deep-learning-for-nlp-an-overview-of-recent-trends-d0d8f40a776d

[33] *"Finite Difference Methods in Financial Engineering: A Partial Differential Equation Approach"*

Authors = Daniel L Duffy
https://www.wiley.com/en-us/Finite+Difference+Methods+in+Financial+Engineering%3A+A+Partial+Differential+Equation+Approach-p-9780470858820

[34] *"A Gentle Introduction to Object Recognition With Deep Learning"*

Author = Jason Brownlee

Year = 22/05/2019
https://machinelearningmastery.com/object-recognition-with-deep-learning/

[35] *"Stochastic Processes and Advanced Mathematical Finance"*

Author = Steven R. Dunbar

https://www.math.unl.edu/ sdunbar1/MathematicalFinance/Lessons/BrownianMotion/QuadraticVariation/quadraticvariation.pdf