DNN in pricing derivatives with stochastic volatility

Tingting Zhou, Luhua Yang and Peter Decrem

[1,2]Financial Engineering, UC Berkeley, Berkeley, CA, USA.
[3]Rates Trading Department, Citi, New York City, NY, USA.

Contributing authors: tingting_zhou@berkeley.edu;
joseph_yang@berkeley.edu; pdecrem@gmail.com;

Abstract

Deep Neural Network in option pricing is a hot topic nowadays. Compare to the traditional Monte Carlo and Finite Difference method, Neural Networks has more advantages in dealing with high dimensional problems, and can quickly generate option price with a short period of time and a smaller cost once it is well-trained. In this paper, we first examined the implementation of Artificial Neural Network in pricing Vanilla European options. After obtaining a stable performance for option pricing, we then constructed another layer of neural network to generate smooth first and second order option Greeks. In the second stage of the experiment, we incorporated SABR volatility model to the network to learn the difference between closed form pricing method and Finite difference method to reduce the numerical implementation time and improve pricing accuracy.

Keywords: Deep Neural Network, Option Pricing, SABR Volatility Surface, Rates Product

1 Introduction

Artificial neural networks(ANNs) has been widely used to detect patterns in large dataset. As being a universal function approximators, ANN has also been used in claim pricing with the aim to accelerating the corresponding numerical methods. Most of the work focus on pricing and reducing the MSE of the test dataset. Take vanilla European option for instance, the analytical solution is famous Black-Scholes formula, a 5 layer neural network is able to

deliver a pretty close result to BS solution while with unbalanced MPE along moneyness. Therefore, in our first stage, we tried to construct a option pricing engine with smooth Greeks,we firstly constructed a 5 layer neural network to price European options, compared to previous work, we trained the model with mesh grid generated data to increase the amount of corner data points and tuned the hyper parameters to generate a more balanced performance along moneyness dimension. Once the pricing model is set up, we extended another ANN model as a weighted smoother, option price and other parameters will be fed into the second ANN model and deliver smooth first order and higher order Greeks. In the training process for the second ANN model, we froze all parameters in the first ANN model and tuned the second ANN model to smooth Greeks until a satisfying result was achieved.

In our second stage, we introduced stochastic volatility in the underlying process, which has a more complicated approximate analytical pricing solution and exact numerical solving methods. In this part, we introduced SABR stochastic volatility model(Hagan et al.,2002)[1] and its classic Lognormal approximate formula,Floc'h method which is regarded as lognormal expansion to the classic one, finite difference numerical method with boundary constraints and also arbitrage free finite difference method without boundary limitation. We implemented the above four methods and compare them. Finally, we convert price into implied volatility using Black-Scholes lognormal formula, and built ANN model to learn the difference of implied volatility between analytical pricing formula and numerical exact result. Any claim can be easily priced with the closed form pricing plus the difference estimated by well trained ANN model. The pricing efficiency will be highly improved taking advantage of the negligible time spent on analytical calculation and the prediction time cost by ANN.

2 Option pricing and asset model

In this section, two asset models are briefly presented, the geometric Brownian motion(GBM) asset model, which give rise to the Black-Scholes option pricing PDE and analytical pricing solution. Then the SABR model will be discussed, leading to the approximate two analytical pricing measures and PDE we need to solve with Finite difference method. We will use European option contracts as the examples here.

2.1 The Black-Scholes PDE and analytical solution

Our first model is the classic BS model in which the underlying asset is assumed to follow a Geometric Brownian Motion:

$$dS_t = \mu S_t dt + \sigma S_t dW_t$$

where the S_t is the price at time t of an non-dividend, W_t is a Wiener process,μ is the drift and the σ is annual volatility.A vanilla European option claim on the

underlying asset with a maturity T can be valued via the Black-Scholes PDE, which can be derived from replicating option payoff with underlying asset and risk free bond, or via a martingale approach. The option value $V(S,t)$ can be solved from the following PDE:

$$\frac{\partial V}{\partial t} + rS\frac{\partial V}{\partial S}\frac{1}{2}\sigma^2 S^2\frac{\partial^2 V}{\partial^2 t} - rV = 0$$

where r is the risk free rate, for a call option, when $t = T$ the PDE has a boundary condition that

$$V(S_T, T) = (S_T - K)^+$$

Here, K is the strike price. Correspondingly, the boundary condition for put option will be:

$$V(S_T, T) = (K - S_T)^+$$

Analytical solutions to the above PDE with boundary conditions exist and have been well proved in all related textbooks, we listed the results below:

$$V_{call}(t,S) = SN(d_1) - Ke^{-rt}N(d_2) \tag{1}$$

$$V_{put}(t,S) = Ke^{-rt}N(-d_2) - SN(-d_1) \tag{2}$$

$$d_1 = \frac{log(S/K) + (r - 0.5\sigma^2)(T-t)}{\sigma\sqrt{T-t}} \tag{3}$$

$$d_2 = d_1 - \sigma\sqrt{T-t} \tag{4}$$

N(.) denotes the CDF of normal distribution.

2.2 Implied volatility

In the real market, options are usually quoted with implied volatility, from above Black-Scholes formula, the implied volatility σ^* can be determined by solving the analytical formula if we have price.We can write the implied volatility as:

$$\sigma^* = BS^{-1}(V, S, K, T-t, r)$$

In our following steps, we all compute the the option prices with a underlying assets following GBM. Therefore, the implied volatility will be calculated using the above formula. Specific initial guess and optimization for root searching method will also be discussed.

2.3 The SABR model

In the classic SABR model, the asset forward follows the following SDE:

$$dF = VF^{\beta}dW_1 \tag{5}$$

$$dV = \nu V dW_2 \tag{6}$$

Where $V_0 = \alpha, F_0 = f$, ν denotes the volatility of volatility, and the W_1 and W_2 are two correlated Brownian Motions with correlation ρ. In a shifted SABR model, there is another parameter b which addresses the shifted level on F to enable the interest rate model to have negative values:

$$dF = V(F+b)^{\beta} dW_1 \tag{7}$$
$$dV = \nu V dW_2 \tag{8}$$

in our following discussion, we only consider the classic SABR model, indicating the $b = 0$.

2.3.1 Hagan Implied volatility

In Hagan et al.(2002), the lognormal volatility for the shifted model($f \neq K$,and $\beta \in [0,1]$) is:

$$\sigma_B(K) = \frac{1}{x(\zeta(K))} log(\frac{f+b}{K+b})[1 + (g(K) + \frac{1}{4}\rho\nu\alpha\beta(f+b)^{\frac{(\beta-1)}{2}}(K+b)^{\frac{(\beta-1)}{2}} \tag{9}$$

$$+ \frac{1}{24}(2-3\rho^2)\nu^2)(T-t)] \tag{10}$$

$$g(K) = \frac{1}{24}(\beta^2-1)(f+b)^{\beta-1}(K+b)^{\beta-1}\alpha^2 \tag{11}$$

$$\zeta(K) = \frac{\nu}{\alpha(1-\beta)}((f+b)^{1-\beta} - (K+b)^{1-\beta}) \tag{12}$$

$$x(\zeta) = \frac{1}{\nu} log(\frac{\sqrt{1-2\rho\zeta+\zeta^2} - \rho + \zeta}{1-\rho}) \tag{13}$$

When $\beta = 1, g(K) = 0$.
When $\beta = 0, g(K) = \frac{\alpha^2}{24(f+b)(K+b)}$.
When $f = K$,

$$\sigma_B(K) = \alpha(f+b)^{\beta-1}[1 + (g(K) + \frac{1}{4}\rho\nu\alpha\beta(f+b)^{(\beta-1)} + \frac{1}{24}(2-3\rho^2)\nu^2)(T-t)]$$

It's worthy to notice that the lognormal formula here is an approximation of the normal formula, so it won't be accurate or realistic at all in cases where the implied volatility is very high. That's also the conclusion we found when we compare Hagan lognormal volatility with other numerical methods.

2.3.2 Le Floc'h and Kennedy Implied volatility

The details of this method are explained in (Le Floc'h and Kennedy,2014)[2], the arbitrage free SABR method from Hagan et al.(2014) [3]can be seen as expansion of local volatility where the vanilla European call options price follow the Dupire PDE.Then the lognormal implied volatility expansions can

be obtained from the local volatility using the technique of (Andersen and Brotherton-Ratcliffe,2005).The implied volatility in le Floc'h and Kennedy method is:

$$\sigma_B(K) = \frac{1}{x(\zeta(K))} log(\frac{f+b}{K+b})[1 + (g(K) + \frac{1}{4}\rho\nu\alpha\Gamma(K))(T-t)] + O(T^2) \tag{14}$$

$$g(K) = -\frac{1}{x^2(\zeta(K))} log(\frac{log(\frac{f+b}{K+b})}{x(\zeta(K))} \sqrt{\frac{(f+b)(K+b)}{D(f)D(K)}}) \tag{15}$$

$$\Gamma(K) = \frac{(K+b)^\beta - (f+b)^\beta}{K-f} \tag{16}$$

$$D(K) = \sqrt{\alpha^2 + 2\alpha\rho\nu y(K) + \nu^2 y(K)^2} K^\beta \tag{17}$$

$$y(K) = \frac{(K+b)^{1-\beta} - (f+b)^{1-\beta}}{1-\beta} \tag{18}$$

For at the money (f = K) option, Floc'h's formula will generate the same result as Hagan's.

2.3.3 Finite Difference Method with boundary

For the classic SABR model, the corresponding PDE can be written as:

$$\frac{\partial u}{\partial t} - \frac{\nu^2}{2}\frac{\partial u}{\partial x} + \frac{1}{2}e^{2x}F_t^{2\beta}\frac{\partial^2 u}{\partial x^2} + \frac{\nu^2}{2}\frac{\partial^2 u}{\partial x^2} + \rho\nu e^{x_t}F_t^\beta\frac{\partial^2 u}{\partial F\partial x} - ru = 0$$

Where the $x = ln\alpha_t$. In order to numerically solve the PDE using Finite difference method, some boundary conditions need to be assumed. The details can can be found in (D.R.Brecher and A.E.Lindsay, 2010)[4]. Besides, in order to solve the solution for chi-squared distribution, a numerical root-finding algorithm is also implemented in the calculation. We won't talk too much on this topic and there exists a pricing engine in Quantlib libraries.

2.3.4 Finite Difference Method without boundary

As mentioned in the above sections, the classic SABR model is not arbitrage free as the probability density can become negative for low strikes and long maturities. Given the low interest rate environment,many authors have proposed meaningful improvements, including finite difference method leading to an arbitrage free 'SABR-like' model in (Andreasen and Huge,2011)[5] and another arbitrage-free version raised by Doust(2012)[6]. All of them will require absorption probability involves costly numerical computations. Hagan et al.(2014) proposed a new arbitrage-free SABR solution, based on a finite difference discretisation of the probability density, which provides a solution close to the classic SABR analytical formula and allows pricing with low rates.However, an oscillation issues exist as they used a Crank-Nicolson

time-stepping scheme. Antonow et al.(2015) proposed an alternate SABR without boundary limitation. In Le Floc'h and Kennedy(2015)[7], the authors improved Hagan's arbitrage-free SABR version and also combined the flexibility of boundary-free model.That's also the main model we implemented in this project. For arbitrage-free part, the transformation on the probability density can be found in Hagan et al.(2014) and another version is Andreasen and Huge(2015), we would like to talk more about the boundary free improvement. As we talked in the SABR model introduction section, to include the negative rates, in the classic model, we can either assign $\beta = 0$ or add a shift parameter b in the process. Here, we followed Antonov et al.(2015)[8]. and make the transition as:

$$dF = VL(F)dW_1 \tag{19}$$

$$L(F) =\mid F \mid^{\beta} \tag{20}$$

The difference between the classic SABR PDE and the free boundary SABR PDE formulation lies in the formulas of L(F) and in the boundary condition. All details about the difference and comparisons among different time stepping schemes are well explained in Le Floc'h and Kenndy(2015), the implementation of Matlab code is also available in our attachments.

3 Methodology

Following the steps mentioned in introduction part. The methodology of implementing ANN model will also be illustrated in this section. In the first stage, We present a neural network to approximate a Black-Scholes European option pricing model and an extended neural network model to get smooth Greeks. In the second stage,we build another neural network model to estimate the difference of implied volatility between analytical solution and numerical solution. The logic of model construction are the same, we will demonstrate the methodology with two main parts, including data generator and model predictor.

Model framework
-Generate sample parameters set within selected ranges -Calculate the corresponding option value or implied volatility with inputs and outputs -Split the whole data set into training and test part -Train the ANN on the training data set -Tune hyper parameters -Evaluate the model performance using test data

Table 1 Algorithm and Process

3.1 Artificial neural network

ANNs now is widely used in academic and industry world, it generally constitutes three levels of components: neurons, layers and the architecture. The architecture is determined by a combination of different layers which are made up of numerous artificial neurons. A neuron, which involves weights and biases, is the fundamental unit of ANNs. By connecting the neurons of adjacent layers, output signals of a previous layer enter a next layer as input signal. By stacking layers on top of each other, signals travel from the input layer through the hidden layers to the output layer potentially through cyclic or recurrent connections, and the ANN builds a mapping among input-output pairs. In our project, we build Feedforward neural network models, underlying algorithm of ANNs won't be discussed here, to simplify, we briefly introduce the model architecture,activation functions, optimization and some parameter settings.

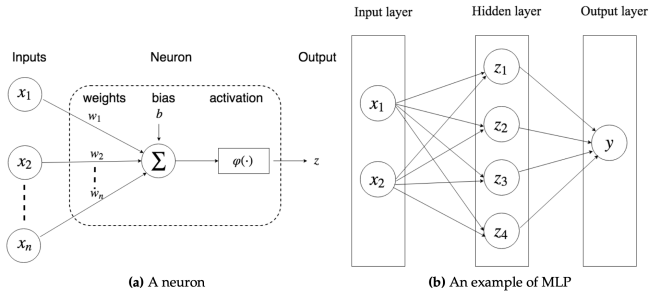

Fig. 1 Illustration of an Multi-layer perceptron (MLP)

MLPs are defined by the following parameters,

$$\theta = (W_1, b_1, W_2, b_2, ..., W_l, b_l)$$

where W_i denotes a weight matrix and b_i represents the added bias vector of ith layer. A function can then be expressed as follows,

$$y(x) = F(x \mid \theta)$$

Let $z_j^{(l-1)}$ represent the output value of the i-th neuron in the $(l-1)$ layer, and the $\phi()$ denote activation function, when $l = 0$, $z^{(0)}$ is the input layer, similarly, when $l = L$,$z^{(L)}$ is the output layer. the corresponding transfer function can be written as:

$$z_j^{(l)} = \phi^{(l)}(\sum_i w_{ij}^{(l)} z_i^{(l-1)} + b_j^{(l)})$$

For the activation function $\phi()$, we listed the following expressions for functions used in this project.

- ReLU, $\phi(x) = max(x, 0)$
- CELU, $\phi(x) = max(0, x) + min(0, \alpha \times (e^{x/\alpha} - 1))$
- Sigmoid,$\phi(x) = \frac{1}{1+e^{-x}}$
- Tanh, $\frac{e^x - e^{-x}}{e^x + e^{-x}}$

According to the Universal Approximation Theorem, a single-hidden-layer ANN with a sufficient number of neurons can approximate any continuous function. The distance between two functions is measured by the norm of a function. The distance function is equivalent to the loss function, and can be expressed as:

$$L(\theta) := D(f(x), F(x \mid \theta))$$

the $f(x)$ is the objective function, $F(x \mid \theta)$ is the neural network approximated function. The whole training process is to learn optimal weights and biases to make the loss function as small as possible. the problem can be formulated as an optimization problem given the known input and outputs and a target loss function.
A number of back-propagation gradient descent methods have been successfully applied to solve the optimized problem. for instance, Adam which we use in this project. Learning rate plays an important role during the training process, we fixed the learning rate as 0.001, which is pretty small, leading the training process learn slowly but avoid suffering oscillation.

3.2 Hyper parameters Tuning

Training deep neural networks involves numerous choices for the commonly called"hyper-parameters". These include the number of layers, neurons, and the specific activation function.Determining the depth (the number of hidden layers) and the width (the number of neurons) of the ANN is a challenging problem. Thanks to our instructor's valuable suggestions, we noticed that a five hidden layers ANN can price the vanilla European options well. Therefore, we start from 5 layers model and search for better performance among Neurons range, epochs number, activation functions and batch size.
As to the second stage, we followed the original parameter setting and model architecture in the blog. No more tuning happen there.

4 Numerical Results for option pricing using ANN

4.1 ANN model for option pricing

4.1.1 Model Structure

```
model=torch.nn.Sequential(
        torch.nn.Linear(5,128),
        torch.nn.ReLU(),
        torch.nn.Linear(128,1024),
        torch.nn.ReLU(),
        torch.nn.Linear(1024,128),
```

```
        torch.nn.ReLU(),
        torch.nn.Linear(128,16),
        torch.nn.CELU(),
        torch.nn.Linear(16,1))

optimizer=optim.Adam(model.parameters(), lr=0.001)
n_epochs = 80
batch_size = 64

train_dataset = torch.utils.data.TensorDataset(x_train, y_train.view(-1, 1))
train_loader = torch.utils.data.DataLoader(dataset=train_dataset,
                                           batch_size=batch_size,
                                           shuffle=True)
for epoch in range(n_epochs):
    for i, (features, price) in enumerate(train_loader):
        optimizer.zero_grad()
        output = model(features.float())
        loss = torch.nn.functional.mse_loss(output,price)
        loss.backward()
        optimizer.step()
```

4.1.2 Train data

We have two ways of generating training and testing data. The first way is random sampling. We set a reasonable range for our option pricing parameters. We then use numpy method to randomly sample a value between those range and calculate their option price using standard Black Scholes model. We use this procedure to generate 1000000 rows of data as the training data. in the table below:

Parameter	Min	Max
Stock Price	1	1
Strike	0.5	1.5
Volatility	0.1	0.4
Maturity	0.25	2
Rf	0.0025	0.025

Table 2 Parameter chosen for Random Sampling

The second method we used is called "Mesh Sampling". This time we generate the training data based on a fixed line space of parameter ranges. For example, for parameter K, we have range from 0.5 to 1.5, we create different entry for each 0.01 increments, Therefore we can obtain 100 different values. We do this for other parameters as well until we obtain our desired parameter density. Here is the ranges and increment setting:

Parameter	Min	Max	Increment	Num of entries
Stock Price	1	1	0	1
Strike	0.5	1.5	0.01	100
Volatility	0.1	0.4	0.02	15
Maturity	0.25	2	0.05	35
Rf	0.0025	0.025	0.0025	9

Table 3 Parameter chosen for Mesh Sampling

In total we have

$$100 \times 15 \times 35 \times 9 = 472500 \; samples$$

This is about half of the samples we have for previous method. And it yields even better performance in our model. Since it covers the different edge cases more evenly. We are gonna elaborate more about this in the later part.

4.1.3 Test data

For testing data, we are only using random sampling. There are two main reasons for this. The first reason is that we are using mesh sampling to train our model, and the parameters combinations are already viewed by the model. If we input the similar parameter combination to the model, the model is gonna output the same value it has learned before, which is totally useless. One might ask that why don't we change the increment and generate another set of testing data with different parameters with mesh sampling? we have thought about this and actually tried it. One problem with this approach is that it introduced a constant offset to the parameter we used to train, and therefore imposed a consistent bias to the prediction result. The second reason is that random sampling helps us testing the different corner cases of the model, and this is actually how the model is gonna be used in the real world application. Therefore we picked the random sampling method to generate out testing data. Here we use 100000 samples to test.

4.1.4 Results

Here is the pricing result plot. The model is able to accurately replicate the Black Scholes formula hence the option prices. The mean squared error is very low, down to 1.5*e-7.

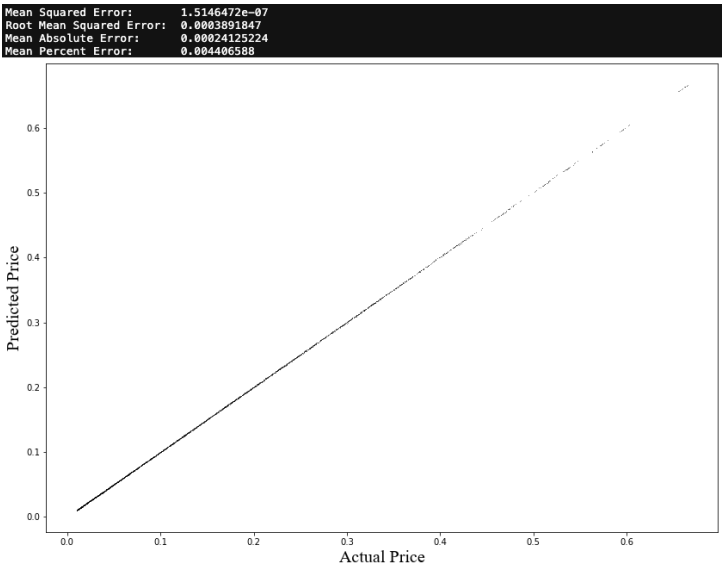

Fig. 2 Pricing result for ANN Model, mesh sampling for all parameters

Next we take a look at the absolute percent error with respect to moneyness. As we can see, when it is deep out of the money, the percent error tends to be high. This is due to the fact that the option price is very low at this region, nearly close to 0. Therefore even a small absolute error will create a large percent error.

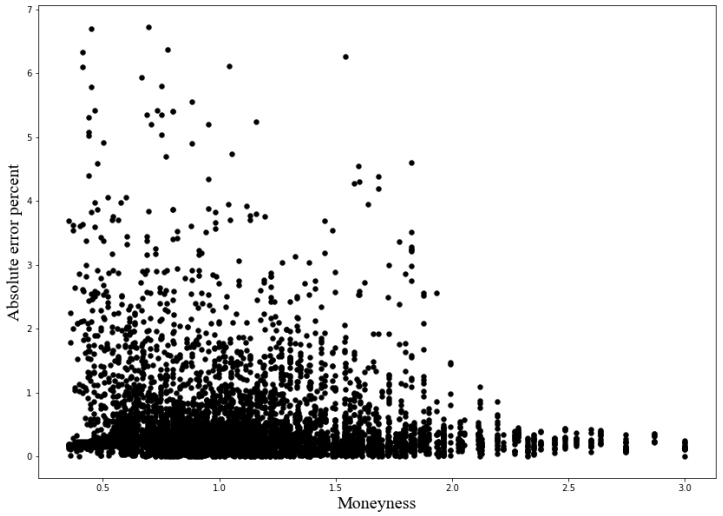

Fig. 3 Pricing result regarding different moneyness, mesh sampling for all parameters

4.2 ANN model for smoothing Greeks

After we have build an ANN model for option pricing with a consistent accuracy. We begin to realize one issue, which is that our model doesn't generate stable and smooth greeks from our option prices. Therefore we decide to add another model, the "fine-tune" model at the output side of our first model. This model helps smoothing the greeks and generate manful and reliable hedges.

4.2.1 Model Structure

```
fc = torch.nn.Sequential(
    torch.nn.Linear(1, 128),
    torch.nn.ReLU(),
    torch.nn.Linear(128,16),
    torch.nn.CELU(),
    torch.nn.Linear(16,2))  # same architecture, two outputs

model_transfer2.add_module('fc',fc)
```

4.2.2 Training data and testing data

In terms of training data, the parameters varies depending on the greek we are calculating. When we are doing the smoothing for Delta, we are using the outputted option price and the stock price. When we are doing smoothing for Vega, we are using the option price and volatility. This applies to other greeks as well, as long as we are using the option price and the varying parameter as the input. We use the 1000000 sample result generated by the previous model and proceed with the training and testing. Here we use a 8-2 ratio split for train and test.

4.2.3 Out of sample results

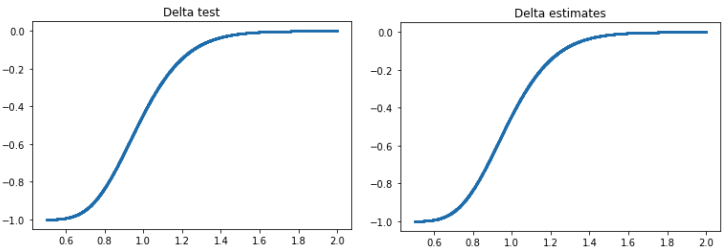

Fig. 4 (a) Theoretical delta respect to K (b) Smoothing NN delta respect to K

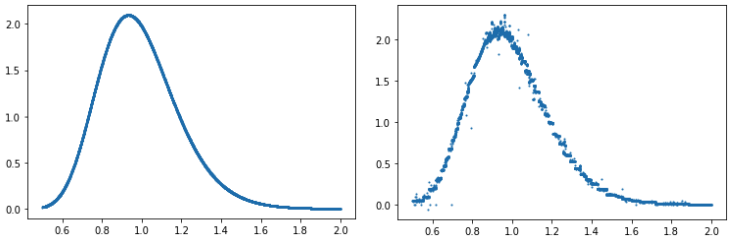

Fig. 5 (a) Theoretical gamma respect to K (b) Smoothing NN gamma respect to K

As we can see from the diagrams. The predicted delta is very smooth, nearly identical to the theoretical delta. However, the gamma is not smooth, we will need another layer to do the smoothing for second order greeks.

5 Numerical Results for Implied volatility Calculation using SABR model

5.1 Analytical results

In the second section,we already introduced two analytical solutions in a stochastic volatility world, to simplify, we call them as Hagan implied volatility and Floc'h implied volatility, and some differences have also been addressed in that section. In this part, we're going to present the difference more virtually, Firstly, we coded our own Hagan analytical solution and compared the results with QuantLib built in 'sabrVolatility' function, for a specific paramter set $[F_0 = f = 0.04, \alpha = 0.2, \beta = 1, \nu = 0.3, \rho = -0.5, T - t = 5]$, we can tell that the results are exactly matched(Figure 6). Therefore, in the following steps, we used the QuantLib built in functions as proxy of our own defined function to speed up the computation.

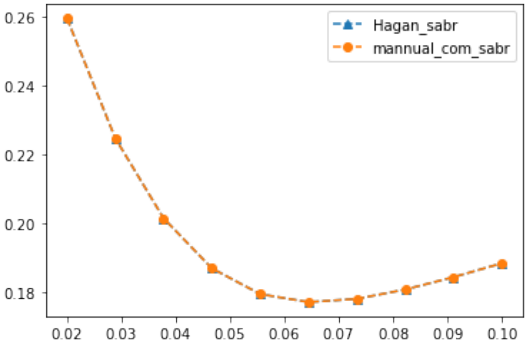

Fig. 6 comparison between self defined Hagan function and Quantlib function results

Now, by using QuantLib, we compared the implied volatility using Hagan and Floc'h(Figure 7). For the above chosen paramemters set, the inconsistency between two methods happened when strike is above 1, indicating the call option is a OTM option.

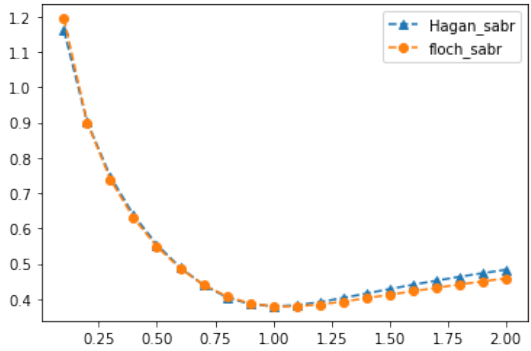

Fig. 7 Comparison of implied volatility between Floc'h and Hagan

However, this is not always the truth, we tried to modified maturity of the options with other parameters fixed, the results were completely different.In figure 8, the correlation between two Brownian Motions had been changed to -0.25, and then figure 9 changed the maturity from 1 year to 10 year. The difference of analytical solutions is such huge that the original blog chose to train a ANN model to learn the exact approximation of FDM from Floc's implied vol, as they believe Floc'h method has more flexibility in dealing with corner data point, or we can say extreme implied volatility. In our project, we trained two ANN models to estimate the difference between FDM and analytical methods, respectively, and compared the performance of two models.

5.2 Implied vol with boundary limits

When calculating the option value using finite difference method, we need another numerical comparable results to check the accuracy. We got 360,000 data points via Monte Carlo implementation, and the summary of the option values from Monte Carlo is listed below in table 4. As both finite difference method and Monte Carlo can generate 'exact' values, these two results should be close with each other. However, as we have illustrated in the model part. For Finite difference method with boundary limits, because of absorption, finite difference method will generate NAN with specific parameter set. To make the results reliable and identify the corner condition, we manually filtered some parameter sets in Case 1 and Case 2, then showed the absolute difference and percentage difference in figure 10,11,12 and 13.

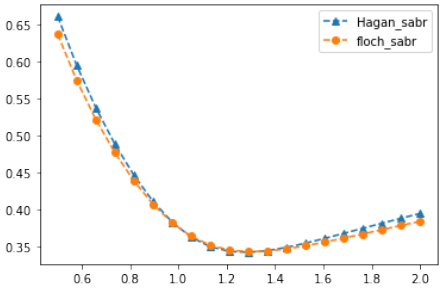

Fig. 8 Comparison of implied volatility between Floc'h and Hagan with negative correlation

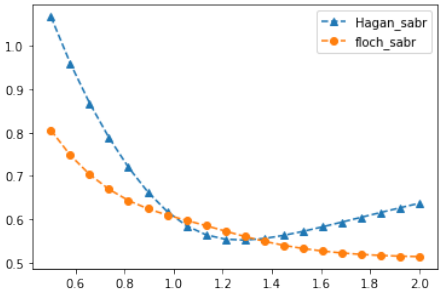

Fig. 9 Comparison of implied volatility between Floc'h and Hagan with negative correlation and longer maturity

	alpha	nu	beta	rho	strike	tenor	value	value_std
count	360000.00	360000.00	360000.00	360000.00	360000.00	360000.00	360000.00	360000.0
mean	0.12	0.22	0.42	-0.36	0.40	4.06	0.61	0.0
std	0.04	0.25	0.42	0.40	0.35	3.87	0.33	0.0
min	0.10	0.00	0.00	-0.80	-0.20	0.25	0.02	0.0
25%	0.10	0.00	0.00	-0.80	0.10	0.81	0.32	0.0
50%	0.10	0.00	0.25	0.00	0.40	3.00	0.61	0.0
75%	0.10	0.50	1.00	0.00	0.70	6.25	0.90	0.0
max	0.20	0.50	1.00	0.00	1.00	10.00	1.20	0.0

Table 4 Summary of parameters and option values from Monte Carlo method

In Case 1, the data was filtered with $\alpha = 0.2, \beta = 0.9999, \rho = 1.e-06, \nu = 1.e-06, 1.2 \geq K \geq 0.8, T \neq 0.25$, there are 11349 data points left which is shown in table 5. From figure 10 and 11, we find that the MC method tends to generate smaller values compared to FDM although both the absolute error and percentage error are both trivial, the difference is biased. In case 2, we augmented our sample by allowing $\alpha = 0.1$ or $\alpha = 0.2$, the conclusion is similar to Case 1, except that we discovered some NANs, for example, in table 6, the count of fdm-price is 18915 instead of 22698. It's difficult to identify the corner parameters set. The instability of QuantLib FDM pricing engine encourages us to develop a boundary free model in next part.

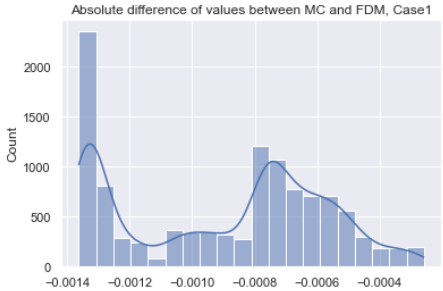

Fig. 10 Absolute difference of values between MC and FDM, Case 1

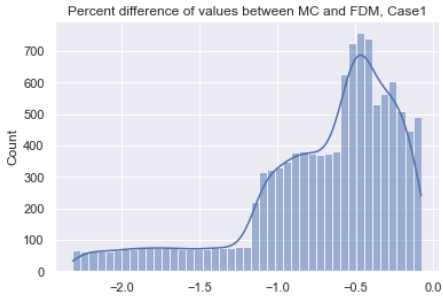

Fig. 11 Percentage difference of values between MC and FDM, Case 1

	alpha	beta	nu	rho	strike	tenor	mc_price	fdm_price
count	11349.0	11349.0000	11349.0	11349.0	11349.0000	11349.0000	11349.0000	11349.0000
mean	0.2	0.9999	0.0	0.0	1.0000	5.3333	0.1759	0.1768
std	0.0	0.0000	0.0	0.0	0.1155	3.6819	0.0824	0.0824
min	0.2	0.9999	0.0	0.0	0.8001	1.0000	0.0214	0.0219
25%	0.2	0.9999	0.0	0.0	0.9000	1.0000	0.1178	0.1190
50%	0.2	0.9999	0.0	0.0	1.0000	5.0000	0.1865	0.1876
75%	0.2	0.9999	0.0	0.0	1.1000	10.0000	0.2380	0.2390
max	0.2	0.9999	0.0	0.0	1.1999	10.0000	0.3377	0.3380

Table 5 Summary of parameters, price and IVs,Case1

	alpha	beta	nu	rho	strike	tenor	mc_price	fdm_price
count	22698.00	22698.0000	22698.0	22698.0	22698.0000	22698.0000	22698.0000	18915.0000
mean	0.15	0.9999	0.0	0.0	1.0000	5.3333	0.1375	0.1532
std	0.05	0.0000	0.0	0.0	0.1155	3.6819	0.0828	0.0784
min	0.10	0.9999	0.0	0.0	0.8001	1.0000	0.0015	0.0219
25%	0.10	0.9999	0.0	0.0	0.9000	1.0000	0.0678	0.0866
50%	0.15	0.9999	0.0	0.0	1.0000	5.0000	0.1327	0.1494
75%	0.20	0.9999	0.0	0.0	1.1000	10.0000	0.1989	0.2101
max	0.20	0.9999	0.0	0.0	1.1999	10.0000	0.3377	0.3380

Table 6 Summary of parameters, price and IVs,Case2

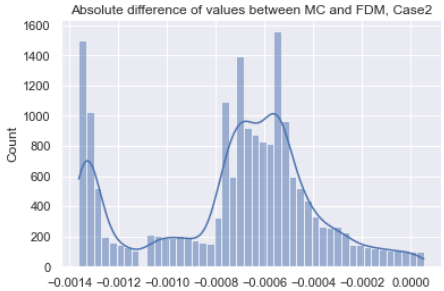

Fig. 12 Absolute difference of values between MC and FDM, Case 2

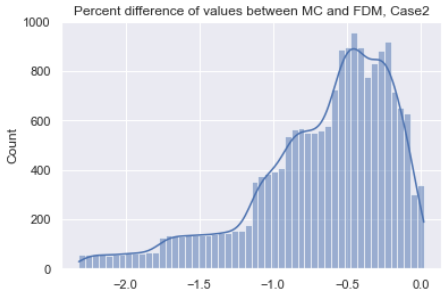

Fig. 13 Percentage difference of values between MC and FDM, Case 2

5.3 Implied vol without boundary limits

We have discussed about the instability of the finite difference method pricing engine, in order to generate more stable results using FDM, we implemented the Le Floc'h and Kennedy(2015) in MatLab. Besides Case 1 and Case 2. We generated fdm-price for all 299,988 data points in last section(filtered out negative strikes), the summary is shown in table 7, then computed the difference and presented the results with a 3D plots in figure 14. The error will increase as the maturities gets longer, and usually the larger errors are raised with deeper in the money options as the options values are relatively larger.
As we have shown that FDM results are biased but still close to the values got from MC method. In this section, we will generate a mesh grid parameters to cover the whole parameters universe instead of dataset from Monte Carlo method and test the stability of the new model. The data was generated from the following parameter range:

$$\alpha \in [0.1, 0.9], \beta \in [0.1, 1], \rho \in [-0.8, 0.2], \nu \in [0.1, 0.9], T \in [0.25, 10], K \in [0.5, 2]$$

The amount of simulated data points is $17 \times 10 \times 7 \times 7 \times 11 \times 7 = 641,410$, and the summary of the parameters and implied volatility is shown in table 6. There are originally 641,410 simulated parameter set, but we found only 627,776 available finite difference prices. NANs are still reported in boundary free finite difference method. However, this is because singular matrix error has been raised when we derived backward from terminal time scheme. We

believe this is a disadvantage of FDM itself and no good solution to solve the problem.

From table 8, we can roughly compare implied volatility of FDM and two analytical methods, the means of 641410 data are close for two analytical solutions, which is a bit smaller than FDM's mean, then the minimum volatility of two analytical methods are both negative, which do not make sense. And the maximum volatility of FDM can be as large as 10.43, which is absolutely different from maximums of two closed form solutions.

	alpha	nu	beta	rho	strike	tenor	value	fdm_value	mc_vol	fdm_vol
count	299988.0000	299988.0000	299988.0000	299988.0000	299988.0000	299988.0000	299988.0000	299988.0000	299988.0000	299988.0000
mean	0.1222	0.2222	0.4167	-0.3556	0.5000	4.0625	0.5178	0.5209	0.1964	0.2180
std	0.0416	0.2485	0.4249	0.3975	0.2887	3.8745	0.2732	0.2749	0.1124	0.3518
min	0.1000	0.0000	0.0000	-0.8000	0.0001	0.2500	0.0198	0.0189	0.0200	0.0200
25%	0.1000	0.0000	0.0000	-0.8000	0.2500	0.8125	0.2803	0.2826	0.1113	0.1038
50%	0.1000	0.0000	0.2500	0.0000	0.5000	3.0000	0.5110	0.5139	0.1983	0.1880
75%	0.1000	0.5000	1.0000	0.0000	0.7500	6.2500	0.7542	0.7582	0.2163	0.2004
max	0.2000	0.5000	1.0000	0.0000	1.0000	10.0000	1.0009	1.0734	0.9115	5.3100

Table 7 Summary of parameters, prices and IVs of MC and FDM

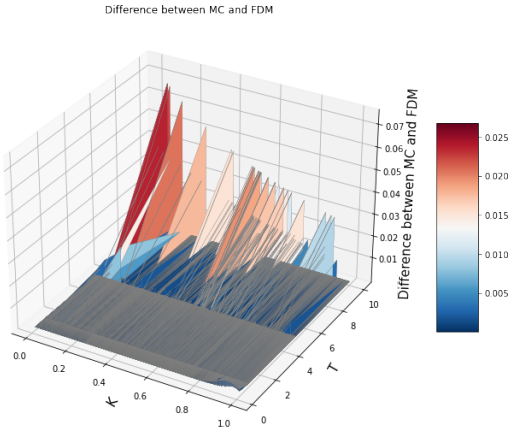

Fig. 14 3D plot of difference of values between MC and FDM

	alpha	nu	beta	rho	strike	tenor	hagan_vol	Floc'h_vol	fdm_price	fdm_vol
count	641410.00	641410.00	641410.00	641410.00	641410.00	641410.00	641410.00	641410.00	627776.00	641410.00
mean	0.50	0.56	0.44	-0.30	1.24	3.82	0.53	0.53	0.31	0.61
std	0.24	0.26	0.28	0.32	0.45	3.42	0.29	0.28	0.28	0.83
min	0.10	0.10	0.10	-0.80	0.50	0.25	-0.68	-0.83	0.00	0.00
25%	0.30	0.30	0.20	-0.60	0.90	0.50	0.31	0.31	0.06	0.29
50%	0.50	0.60	0.40	-0.30	1.20	3.00	0.50	0.50	0.25	0.49
75%	0.70	0.80	0.60	0.00	1.60	7.00	0.71	0.71	0.50	0.70
max	0.90	0.90	1.00	0.20	2.00	10.00	2.76	2.70	2.11	10.43

Table 8 Summary of parameters and IVs of FDM without boundary limitiation

6 Exact SABR Interpolation using Neural Networks

While using Monte Carlo and finite difference method can produce high accuracy option pricing result. These two methods are very time consuming and heavily rely on resources. As demonstrated in previous sessions, the Floc'h/Hagan volatility and the Black Scholes implied vol from MC method are very close. Therefore we consider using a Neural Network to predict the residual between our MC/FDM value and the Hagan/Floc'h value. We treat the MC/FDM vol as the ground truth y and use Hagan/Floc'h as our X feature input. According to the paper, we constructed a 5-layer Neural Network and achieved good result.

6.1 Model Structure

```
model=torch.nn.Sequential(
        torch.nn.Linear(7,20),
        torch.nn.Linear(20,100),
        torch.nn.Linear(100,400),
        torch.nn.Sigmoid(),
        torch.nn.Linear(400,10),
        torch.nn.Tanh(),
        torch.nn.Linear(10,1),
        torch.nn.Tanh())

optimizer=optim.Adam(model.parameters(), lr=0.001)
n_epochs = 80
batch_size = 64

train_dataset = torch.utils.data.TensorDataset(x_train, y_train.view(-1, 1))
train_loader = torch.utils.data.DataLoader(dataset=train_dataset,
                            batch_size=batch_size,
                            shuffle=True)
```

According to the paper, we are mainly using linear activation function to replicate the volatility difference. One thing to note is that the last output function from the paper is "Sigmoid". We discover issues with this approach, since sigmoid can only output positive result, which contradicts with our empirical observation: the difference between Hagan and FDM vol are somewhere positive and somewhere negative. As a result, we switched to "Tanh" activation to make a reasonable prediction.

6.2 Training Dataset and statistic summaries

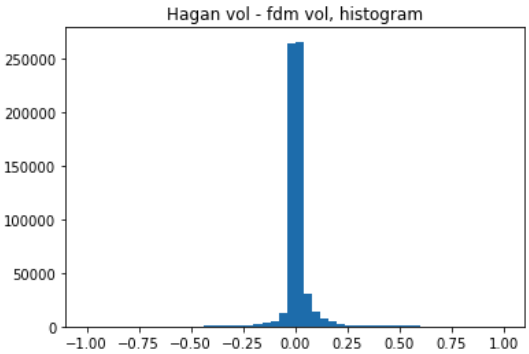

Fig. 15 Difference between Hagan vol and finite difference vol

As we can see from figure 15, the Hagan and finite difference vol are close for most of the time. There are some cases when one is greater or the other way round, depending on the parameter setting. The goal of our NN model is to predict this difference.

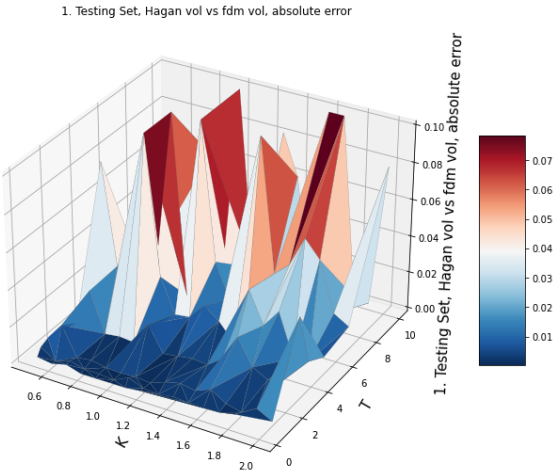

Fig. 16 Absolute Error distribution of Hagan vol and FDM vol

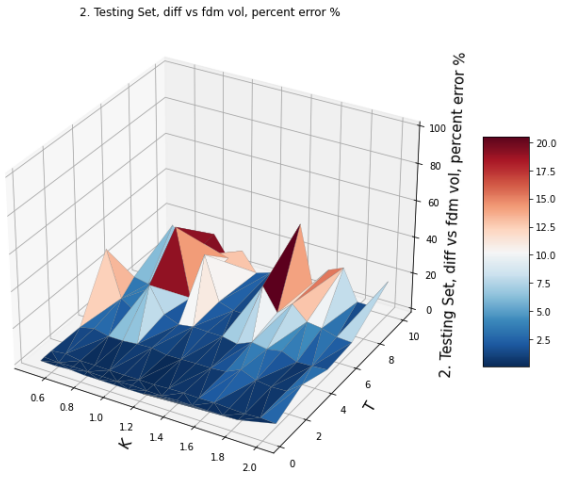

Fig. 17 Absolute Percent Error distribution of Hagan vol and FDM vol

As we can see from both the absolute error and absolute percent error distribution in figure 16 and figure 17, Hagan vol is different from FDM vol when the tenor becomes large. Therefore we put more samples in those region to improve the robustness of our NN model. While in other regions, the role of the NN model is to identify it and output 0, since the two vols are very close to each other.

6.3 Results and comments

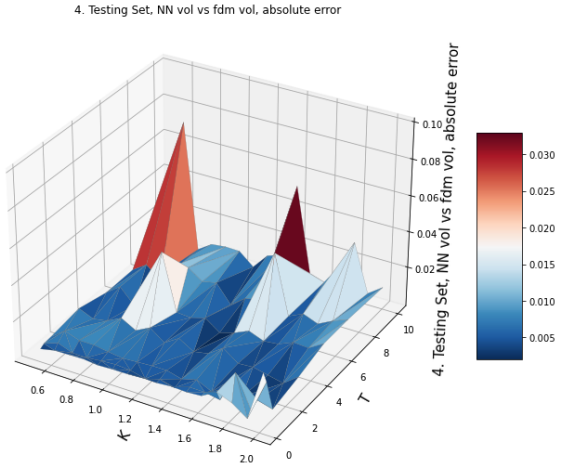

Fig. 18 Absolute Error distribution of NN vol and FDM vol

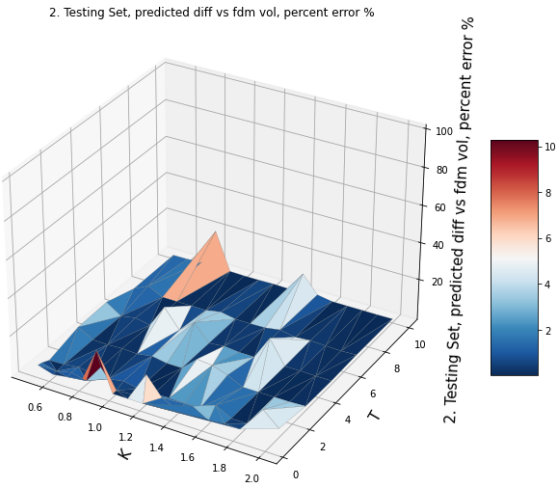

Fig. 19 Absolute Percent Error distribution of NN vol and FDM vol

Metric	Mean	25%	50%	75%
Absolute Error, NN	0.009896	0.004392	0.005353	0.009250
Absolute Error, Hagan	0.018420	0.000811	0.005565	0.023424
Absolute Percent Error, NN	4.35%	0.77%	1.31%	2.69%
Absolute Percent Error, Hagan	5.22%	0.13%	0.84%	4.08%

Table 9 Results for NN prediction and Floc'h vol

As we can see from table 9, our Neural Network model produces better and more stable result compared to hagan vol. In both the mean absolute error as well as the mean absolute percent error. in terms of the distribution, the Neural Network model has a more stable distribution and the Hagan model has more variance.

Let's take a look at figure 18 and 19. The Neural Network model is able to capture the sensitive regions when tenor is high. It is able to reduce the error level down to some extend, compared with figure 15 and 16. As for the stable regions, when the Hagan vol is close to FDM vol, the NN is doing a decent job in maintaining the overall error level down. But there is still some errors because the NN model has some randomness in that region. A better approach would be separating the two regions and use two different models to do the prediction. This way will yield better results.

6.4 Fine Tuning

The last step we have done is fine tuning. During the training and testing process, we observe several issues with our model. The first issue is that Hagan vol and FDM vol is very different in some regions, due to the incompatibility of our Quantlib library. We therefore cropped those corrupted data and

improved the performance of the model. The second issue is that there are certain regions when the difference between Hagan and FDM is reasonably high, due to absorbing states or a long tenor. We generated more data points in thorse regions to reinforce the learning of the model, and therefore achieving a better result. The following diagrams illustrate the difference in performance before and after those enhancements.

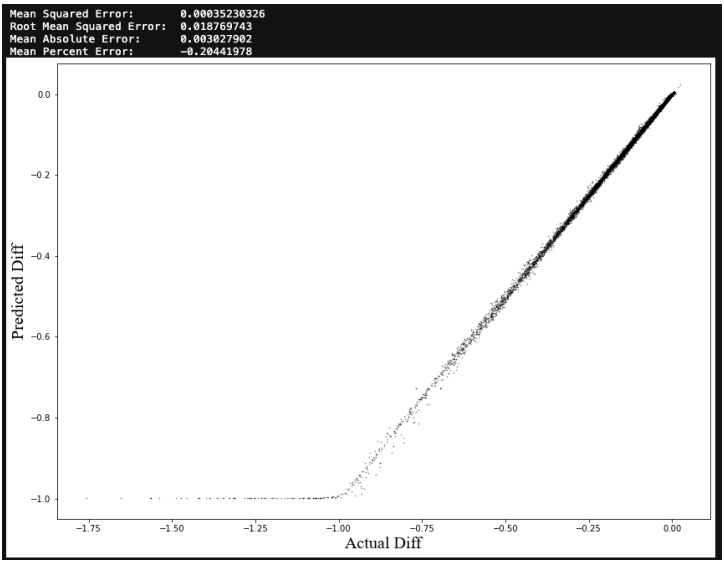

Fig. 20 Prediction error before fine tuning

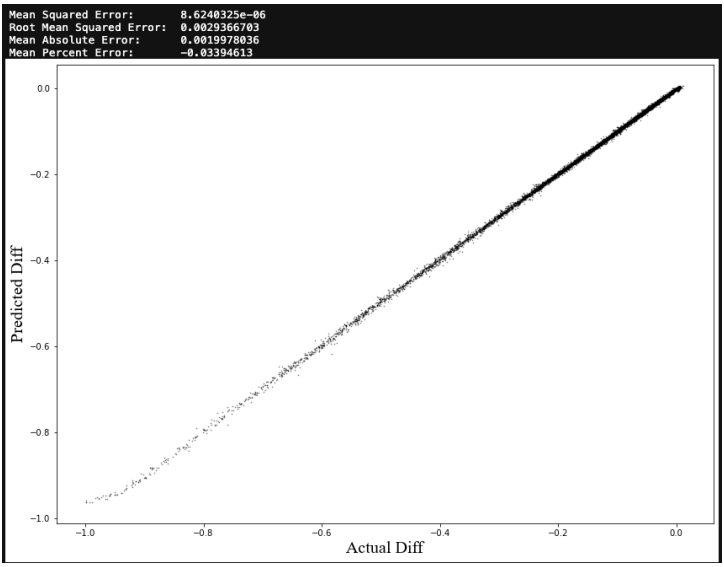

Fig. 21 Prediction error after fine tuning

6.5 Estimate difference between Hagan and FDM with narrow range training points

As we explained in our first stage, if we trained the model with Mesh sample, then predicted the outputs with 'static' parameters instead of the random test, usually we can see a better performance.

In this section,we will filter training data with specific criteria and try to estimate the difference of implied volatility between Hagan and FDM. The criteria we used to build the filter is shown in table 10, they are chose because of real world experience, usually β is around 0.2 to 0.3, and correlation tends to be negative. Summary of the filtered data is presented in table 11.

From the summary table,we can see that only 34307 data points are left, and for Hagan volatility, Floc'h volatility and FDM volatility, the statistics are much closer compared to the whole dataset in table 8. Now we can compute the difference by subtracting FDM volatility from Hagan one. Then the inputs and outputs for ANNs is prepared. We split the data set by 3:1 and use the same model as in part 6.1. The MSE of the test data is $8.297109e-05$, and figure 22 shows the predicted values with actual ones. The performance is worse compared to the whole data set. On one hand we can see the actual difference is in a narrower range [-0.3,0.2] compared to the unfiltered difference range. On the other hand, most of the predictions distribute around 0. The result indicates that the model will output smaller numbers to reduce the loss function as most target values are very small, so the model neglect corner data in our training data set. We illustrate this findings in figure 23 and figure 24, the actual percentage difference can be as large as 14% while the maximum

prediction percentage error is around 3.5%, indicating the larger difference has been underestimated.

Filter	Min	Max
fdm-price	0.1	inf
fdm-vol	0.1	inf
α	0.1	0.5
β	0.1	0.5
ρ	- inf	0.0

Table 10 Filter criteria

	alpha	nu	beta	rho	strike	tenor	hagan_vol	Floc'h_vol	fdm_price	fdm_vol
count	34307.0000	34307.0000	34307.0000	34307.0000	34307.0000	34307.0000	34307.0000	34307.0000	34307.0000	34307.0000
mean	0.3126	0.5673	0.3002	-0.4350	0.9953	5.1385	0.3790	0.3736	0.3307	0.3701
std	0.0804	0.2603	0.0817	0.2274	0.3850	3.4466	0.1218	0.1116	0.1671	0.1244
min	0.2000	0.1000	0.2000	-0.8000	0.5000	0.2500	0.1192	0.1543	0.1000	0.1503
25%	0.2000	0.3000	0.2000	-0.6000	0.7000	1.0000	0.2904	0.2914	0.1901	0.2861
50%	0.3000	0.6000	0.3000	-0.4000	0.9000	5.0000	0.3654	0.3624	0.3036	0.3563
75%	0.4000	0.8000	0.4000	-0.2000	1.2000	7.0000	0.4482	0.4399	0.4512	0.4345
max	0.4000	0.9000	0.4000	-0.1000	2.0000	10.0000	1.0881	0.9366	1.0343	5.3100

Table 11 Summary of filtered data

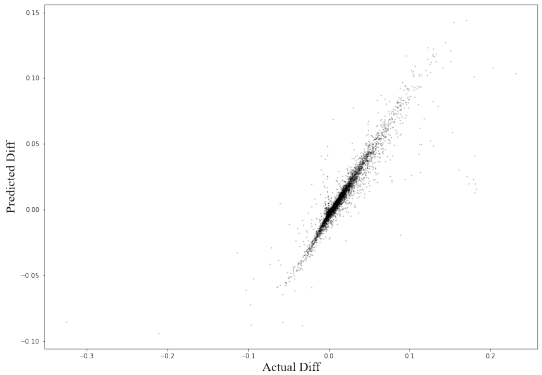

Fig. 22 Prediction of difference and actual target values

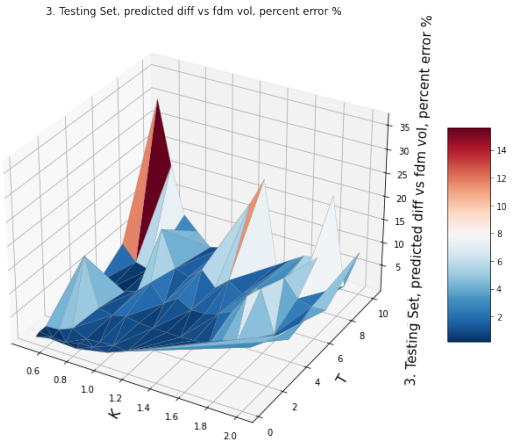

Fig. 23 Actual percentage difference of test data

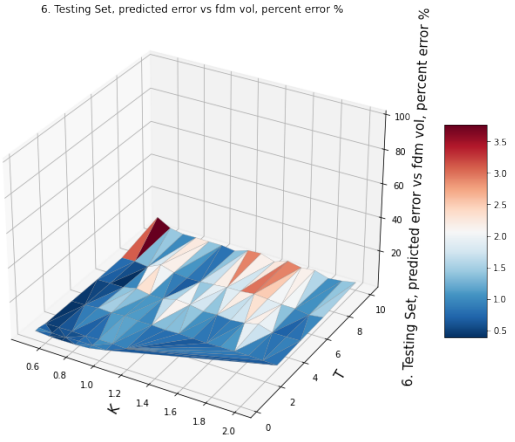

Fig. 24 Prediction of percentage difference of test data

Now, if we subtracted the estimated difference from Hagan implied volatility, we will get the approximate FDM implied volatility. We selected three combinations of parameters, the first parameter set is

$$[\alpha = 0.3, \beta = 0.3, \nu = 0.9, \rho = -0.8, T = 1]$$

and the implied volatility with strike range from [0.5,2] is shown in figure 25. The approximate volatility is above actual FDM volatility. The difference increases with the strike and then converges again. In the second parameter

set, we changed the maturity to 10, indicating the new parameters are:

$$[\alpha = 0.3, \beta = 0.3, \nu = 0.9, \rho = -0.8, T = 3]$$

we can tell from figure 26, approximates are almost the same with actual FDM. Now we increase the maturity to $T = 10$ in the third case, we find similar conclusion in figure 27. Therefore, for 'static' parameter, we see the performance of the model is pretty good. If we trained the model to estimate the difference between Hagan and FDM, the longer the tenor is, the closer between actual value and approximates will be.

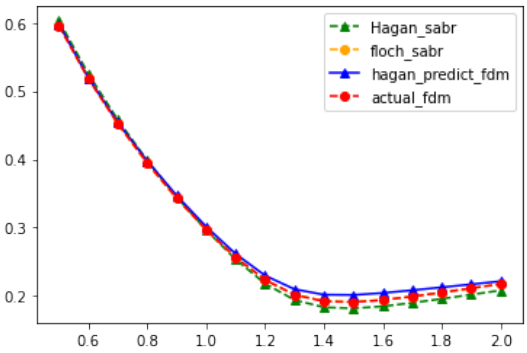

Fig. 25 Comparison between Hagan,Floc'h,FDM and ANN estimates

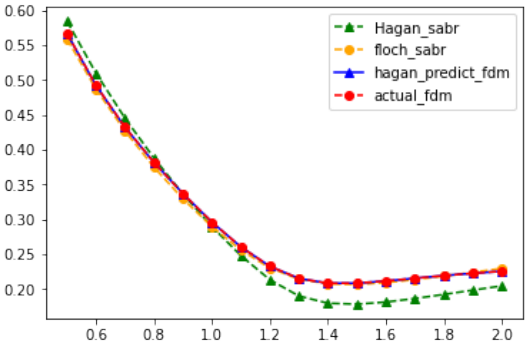

Fig. 26 Comparison between Hagan,Floc'h,FDM and ANN estimates

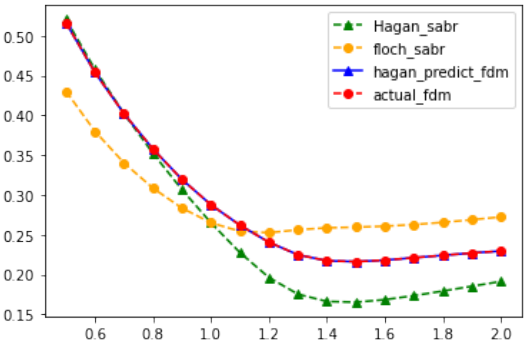

Fig. 27 Comparison between Hagan,Floc'h,FDM and ANN estimates

6.6 Estimate difference between Floc'h and FDM with narrow range training points

Same as we did in part 6.5. in this section,we will calculate difference between Floc'h implied volatility and FDM volatility, which is used as our outputs. Repeat the same training procedure. The approximates for three parameters combinations with various tenors are shown in figure 28,figure 29 and figure 30. One interesting conclusion we find is, in the first case, $T = 1$, the approximates are almost similar to the actual value, as the tenor increases, the inconsistency becomes larger, indicating a opposite conclusion with section 6.5.

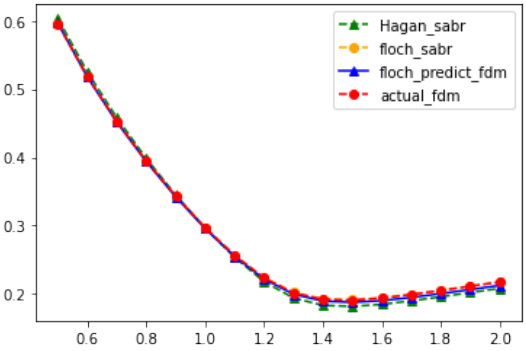

Fig. 28 Comparison between Hagan,Floc'h,FDM and ANN estimates

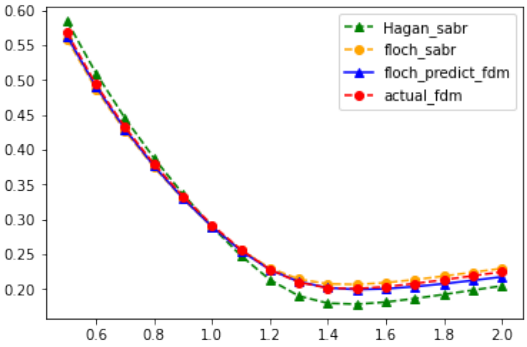

Fig. 29 Comparison between Hagan,Floc'h,FDM and ANN estimates

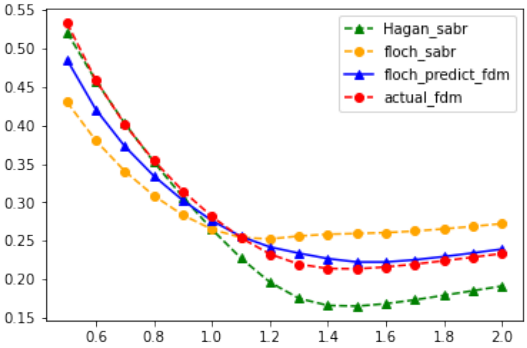

Fig. 30 Comparison between Hagan,Floc'h,FDM and ANN estimates

7 Conclusions and Limitations

7.1 Conclusions

In our two stage project, we firstly build a ANN model for vanilla European option pricing and add another ANN model to get smooth higher order Greeks, We have showed that using Mesh sample data to train the model outperformed random sample, and in the test case, our model delivered great results. Compared to others' work, our model focus more on the prediction percentage errors with strike instead of simple MSE. As we found most part work addressed small MSE while the percentage error for deep OTM option is relatively huge.Besides, we creatively build an extended ANN model to get smooth Greeks. Our result perfectly match the Delta and Gamma function. Compared with bagging method, our model significantly reduces the training time and computation cost.

In the second stage, we try to use ANN to estimate the difference between analytical implied volatility and numerical ones. At first,we presented various analytical methods and QuantLib built in funcions, then we compare the results with Monte carlo method and identify some instability. To reduce the impact of absorption when rates hit zero boundary or negative values, we constructed boundary free SABR model and calculated the implied volatility after we get the option price. Lastly, we build a ANN model to learn the difference and try to predict the real FDM implied volatility using the analytical solution and estimated difference from our model.We showed that both Monte Carlo numerical results and FDM results are reliable, they're very close to each other. And FDM can save lots of computation time compared with Monte Carlo method, but it has embedded limitation as singular matrix error may raise when we do the backward calculation. Using ANN model to learn the difference between analytical solution and numerical solution has great potential in reducing computation time, but the performance may be not applicable in the whole parameters universe, train the model and test the model with narrow parameters range will help.

7.2 limitations

In the implementation of this project, we found some limitations and difficulties existing in some process. including:

- QuantLib built in pricing engine for FDM with boundary is not stable nor reliable.
- When we transfer price into implied volatility, global optimization function was used, and sometimes returned initial guess. A better initial guess searching algorithm should be adopted.
- We followed the model architecture suggested in the author's blog, no grid searching and hyper parameters tuning in the second stage.
- The corner data points generated by some parameter set has not been further discussed and deeply looked into. The training process will be more efficient and accurate with parameters splitting.
- FDM method has multiple time step scheme choices, but singular matrix problem will always be a challenge in the backward calculation.

Appendix A

References

[1] Hagan, K.D.L.A. P.S., Woodward.D.: Managing smile risk. Numer. Math. **72**(2), 173–196 (2002)

[2] Le Floc'h, F., Kennedy, G.J.: Finite difference techniques for arbitrage free sabr. SSRN Electronic Journal (2014). https://doi.org/10.2139/ssrn.2402001

[3] Hagan, K.D.L.A. P.S., Woodward.D.: Arbitrage-free sabr. Wilmott **2014**(69), 60–75 (2004)

[4] Brecher, D., Lindsay, A.: Simulation of the cev process and the local martingale property. Mathematics and Computers in Simulation **82**(5), 868–878 (2010)

[5] Andreasen, J., Huge, B.N.: Zabr – expansions for the masses. SSRN Electronic Journal (2011)

[6] Doust, P.: No-arbitrage sabr. The Journal of Computational Finance **15**(3), 3–31 (2012)

[7] Le Floc'h, F., Kennedy, G.J.: Finite difference techniques for arbitrage free sabr. SSRN Electronic Journal (2014)

[8] Antonov, K.M. A., Spector, M.: The free boundary sabr: Natural extension to negative rates. SSRN Electronic Journal (2015)