A Monte Carlo & Neural Network Approach to Evaluate Portfolio Risk

Tongda (Carl) Che

https://github.com/carlche15

Some things are best left unsaid and unknown.

----*Neural Network*

Things take time, in both statistical and real world.

----*Cross Correlation*

After all the trials, you may not succeed, but you are getting closer.

---*Monte Carlo Simulation*

Overview of MCN VaR model

MCN VaR is a risk model that combines Neural Network, Cross Correlation, and Monte Carlo techniques. It first decomposes the portfolio into three parts: mortgages backed securities, other fixed income Securities (bonds and interest rate derivatives), stocks and stock options according to their different risk nature and then adopts different methods to model the risk factors inherited in each part, and finally use Monte Carlo method to simulate the loss distribution of the portfolio.

I did this because I have been trying to use some of the mentioned techniques in a trading project and a couple of days ago I had an interesting conversation about the evaluation of VaR. So it naturally occurs to me that it might be a valid trial to use such techniques in the VaR calculation. MCN merely represents Monte Carlo & Neural network, just for convenience.

Modeling MBSs' Risk

MBS's risk is mostly contributed by payment behaviors of mortgagors, which are motivated by many factors such as prevailing interest rate, the mortgage coupon, macro factors, natural events, the length since the loan is issued, and even personal reasons, and .etc. All these factors (with, of course so many undetected ones) work together in a subtle way and finally determine a mortgagor's prepayment behavior. Modeling a behavior-related stochastic process such as mortgage prepayment is a big challenge and there aren't many public accessible models out there. [1]Furthermore, a parametric approach to model things that we don't understand very well may introduce significant biases.

The MCN model uses a Neural Network and thus non-parametric method to predict the prepayment rate by some inputs and has achieved some degree of accuracy as shown later. Input factors include but not limited to prevailing long-term interest rate, mortgage coupon, the holding length of the loan. Furthermore, the input also includes interest rate a month ahead, two months ahead, and .etc. This is because it takes time for a mortgager to decide to refinance her loan and actually launch her plan, so it is reasonable for us to consider events happened before.

[1] There are some, however, such as PSA, Richard (JPM) and so on

Then how are the Neural Network's predicting results be incorporated in the VaR evaluation process of the portfolio? As mentioned, one of the inputs of the Neural Network is current interest, which has been thoroughly researched and there are many factor(s) models to describe it. There are continuous time short rate models such as Hull-White, Ho&Lee, CIR, Vasicek models. The MCN model first trains the Neural Network by historical data via a supervised learning process, and the random part of Monte Carlo Simulation of MBS comes from short rate dynamic, which is one of its input. [2]

Input Selection

* Mortgage rate: Conventional Fixed-Rate Mortgages; participating dealers: BC, CSFB, JPM, MS
* Current spot rate: Treasury long-term yield
* One month lagged spot rate
* Two moths lagged spot rate
* Three months lagged spot rate
* Time since the loan is issued

Output: Predicted CPR (Conditional Prepayment Rate)

Neural Network Structure:

After various trials, it seems a simple neural network of one hidden layer with $2 * N + 1$ where N is the number of inputs produces the optimal result. The out-of-sample predict result is presented in Figure 1.1

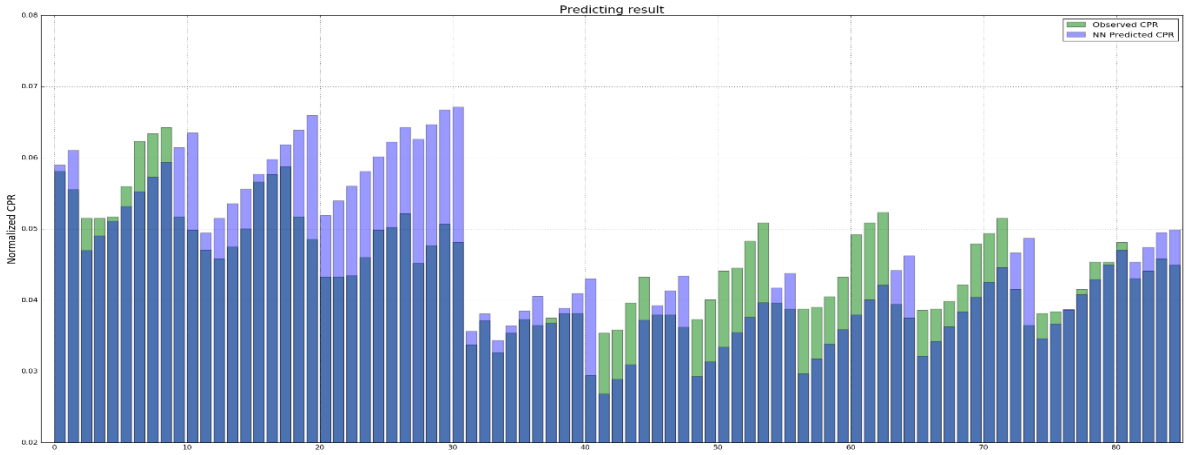

Figure 1.1 CPR predicting result by Neural Network with lagged inputs

[2] In the following Monte Carlo Process, what really being simulated are actually the inputs of this CPR predicting model---spot rate. With predicted CPR and interest rate, one could thereafter price the mortgage backed securities.

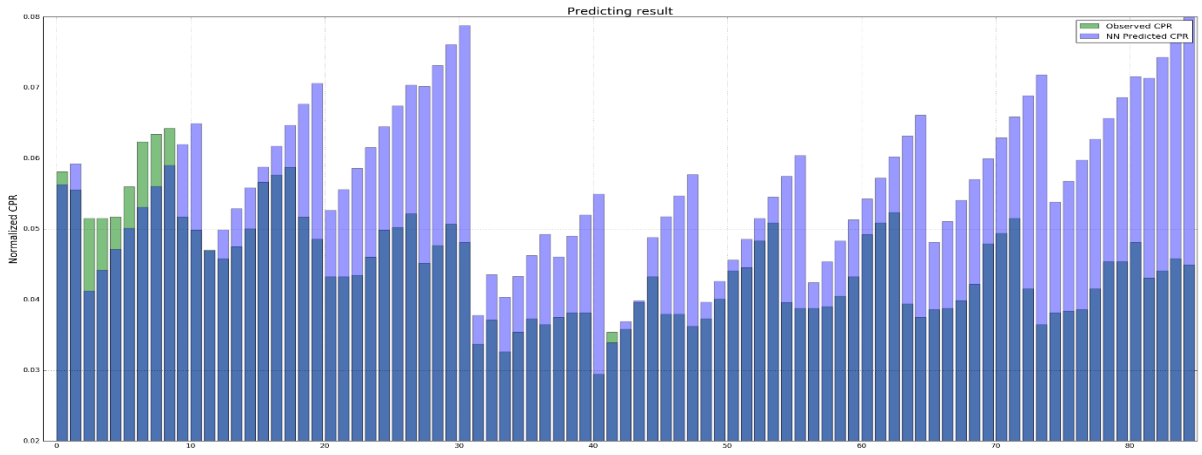

Figure 1.2

Figure 1.1 CPR predicting result by Neural Network without lagged input

As we see, Neural Network with lagged input outperforms with the one without lagged information in the out-of sample-test, which suggests the validity of cross-correlation between interest rate and CPR.

Modeling stocks and stock options' risk

In this research, I also assume there are various stocks and stock options in the portfolio, specifically, stocks, plain vanilla call and put options, Asian type call and put options. Further, since I am commutating VaR within a short time range, the discount factor is omitted.

In this demonstrative model, I will be using "brute force" Pure Monte Carlo Simulation to evaluate each asset in the portfolio. By "brute force" I mean the full paths of underlying assets will be simulated if necessary, and thus we could price of any financial instrument if we know its parametric model. This could, however, be extremely computationally expensive but since we are conducting experiments on a small portfolio, computational resource is considered to be sufficient. [3]

Assume the stocks follows Geometric Brownian Process, which satisfy the SDE:

$$\mathrm{d}s_t = s_t\left(\mu_t d_t + \sigma_t \,\mathrm{d}w_t^p\right)$$

[3] A good approach to measure the value at risk of a portfolio with derivatives is Delta-Gamma method, which captures portfolio's quadratic sensitivity with regard to underlying assets' price changes. $\Delta P = \frac{\partial p}{\partial t}\Delta t + \delta^T \Delta s + \frac{1}{2}\Delta s^T \Gamma \Delta s$. This method requires our prior knowledge of $\delta = \frac{\partial P}{\partial s}$ and $\Gamma = \frac{\partial^2 P}{\partial s \partial s}$, which are hard to compute but are able to get from other sources.

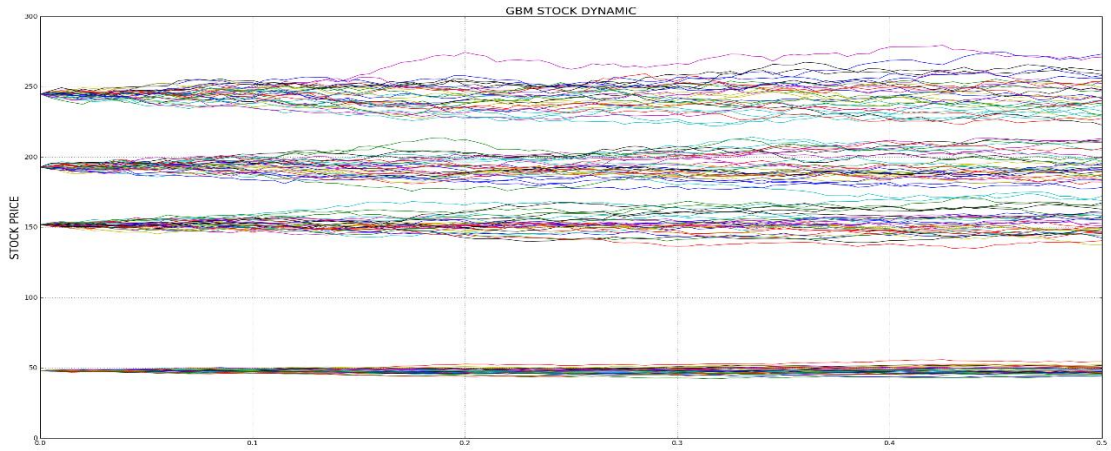

Figure 2.1 dynamics of four stocks with different spot price

For convenience, I also assume that μ, σ are constant. So that a close form solution of The above SDE is available:

$$s_t = s_0 \mathrm{e}^{\left(\mu t - \frac{6^2}{2}t + \partial w_t\right)}$$

This is a big convenience for plain vanilla option simulation in which cases we could skip the simulation of stock dynamics by using directly the equation below.

$$c_T = max\left[\left(s_0 \mathrm{e}^{\left(\mu T - \frac{6^2}{2}T + 6wT\right)} - k\right), 0\right]$$

But the portfolio also contains exotic option(s), for example, floating strike Asian options, the strike of which is the average price of its underlying asset as the equation below suggested. Derivatives such as this are path- dependent, so within each Monte Carlo Simulation, we will have to generate a full path of its underlying asset, which is very expensive.

$$k = \frac{1}{T}\int_0^T s_t \,\mathrm{d}t = \frac{1}{T}\int_0^T s_0 \mathrm{e}^{\left(\mu - \frac{\sigma^2}{2}\right)t + \sigma w_t} \,\mathrm{d}t$$

The underlying assets of derivatives in the portfolio are also correlated. I assume they follow a multivariate Gaussian distribution defined by their variance-covariance: $\Delta S \sim N(\mu, \Sigma)$. In practice, the procedure of to generate n correlated random variable is as follows:

(1) Calculate the variance-covariance matrix Σ.
(2) Generate n*1 i.i.d standard Gaussian random variables vector Z.
(3) Perform Cholesky decomposition on Σ : $\Sigma = CC^T$
(4) Z'=C*Z

Figure 2.2 shows the Monte Carlo simulation of loss distribution of stock investments, figure 2.3 present loss distribution of stock derivatives. In both cases, the impacts of interest rate shift are omitted. The experimental portfolio holds large short positions on stock options (which make it riskier), thus we could better measure how well the model capture the risk.

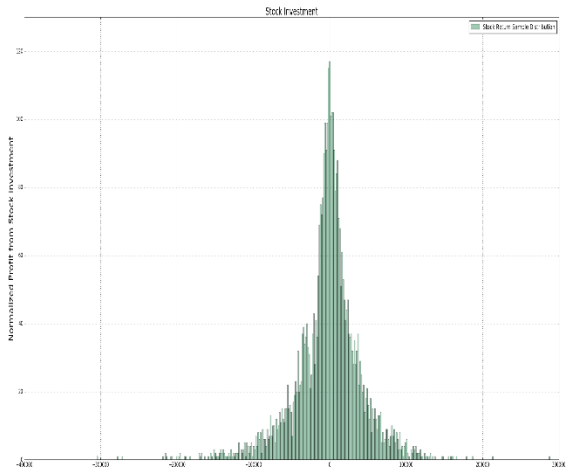

Figure 2.2 loss distribution of stock investments

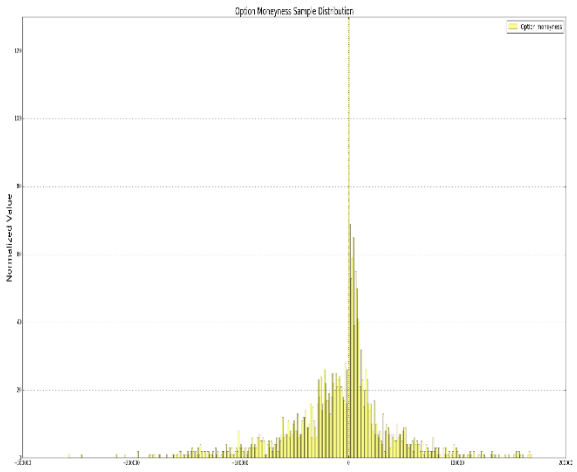

Figure 2.3 loss distribution of stock option investments

Fixed Income Instruments' Risk

In the following part, I arbitrarily assume that short rate evolves according to Vasicek model. One benefit of this assumption is that Plain Bond has an analytic solution under this short rate model.

$$\frac{\mathrm{d}B(t,T)}{B(t,T)} = B(\tau)\,\mathrm{d}w_t^* + r_t\,\mathrm{d}t \approx B(\tau)\,\mathrm{d}w_t^p + r_t\,\mathrm{d}t$$

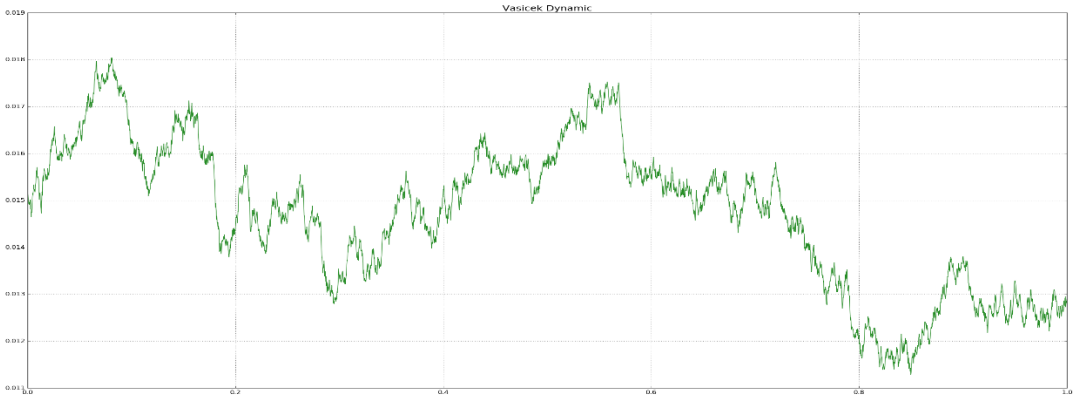

Figure 3.1 dynamics of short rate according to Vasicek model

On the other hand, clear traces (by simulation of course) of short rate allow us to price most kinds of fixed income instruments (fixed coupon bonds, bond with embedded options). One has to recall in this part I omit the credit risk reside in all instruments, and thus I exclude floater, swap, and swaptions from the portfolio.

Figure 3.2 shows the simulation results of bond investments.[4]

[4] The only risk factor of bond investments is interest rate risk. In practice, there are credit risk, liquidity risk, operational risk and so on, which could be captured by models calibrated by market information e.g. OAS.

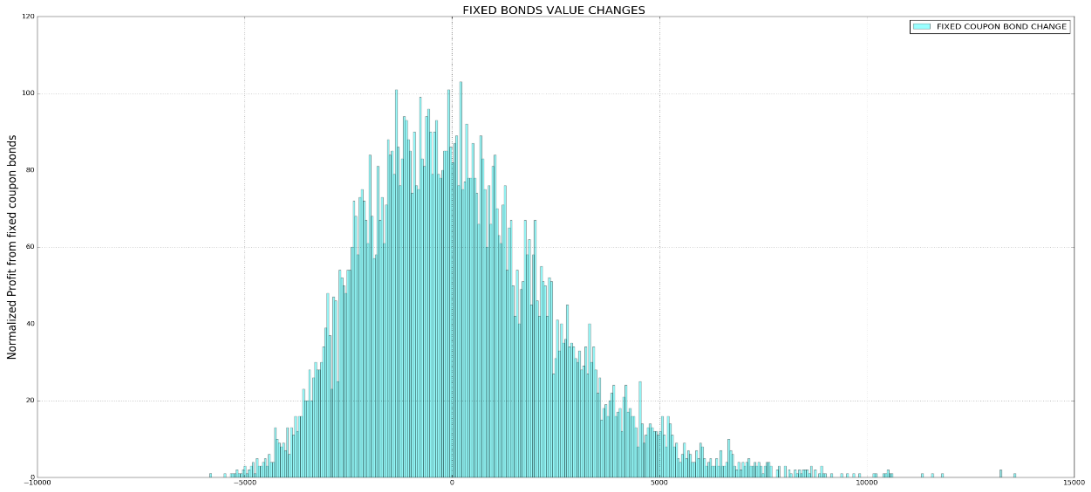

Figure 3.2 Loss distribution of bond investments

Now, the only thing left is to incorporate short rate in the Neural Network model I developed before. The main purpose of this, as mentioned before, is to establish a relationship between short rate process we try to simulate and the NN model that could predict the prepayment rate. The specific procedure is as follows:

(1) Construct term continuous structure model (such as Nelson-Siegel model) and calibrated it to the market.
(2) Simulating short rate, within each simulation, do (3) , (4) and (5)
(3) By the definition of spot rate (the average rate of short rate) $R(t,T) = \frac{1}{T-t}\int_t^T r_S \, \mathrm{d}S$ solve for spot rate R
(4) Use solved R along with other inputs calculate for CPR of MBSs.
(5) Calculate for MBSs' Prepayment Loss

The process of (3) and (4) could also be directly done by:

$$R(t,T) = R_\infty + \left(R_{\infty - r_t}\right)\frac{B(\tau)}{\tau} + \frac{\sigma^2 B^2(\tau)}{4k\tau}$$

Where B(τ) is zero bond price calculated via our short rate model, R_∞ is a term calculated from Vasicek model.

With predicted CPR with respect to each simulated interest rate path incorporated, the sample distribution of MBS prepayment and interest rate loss is presented in figure 3.3.

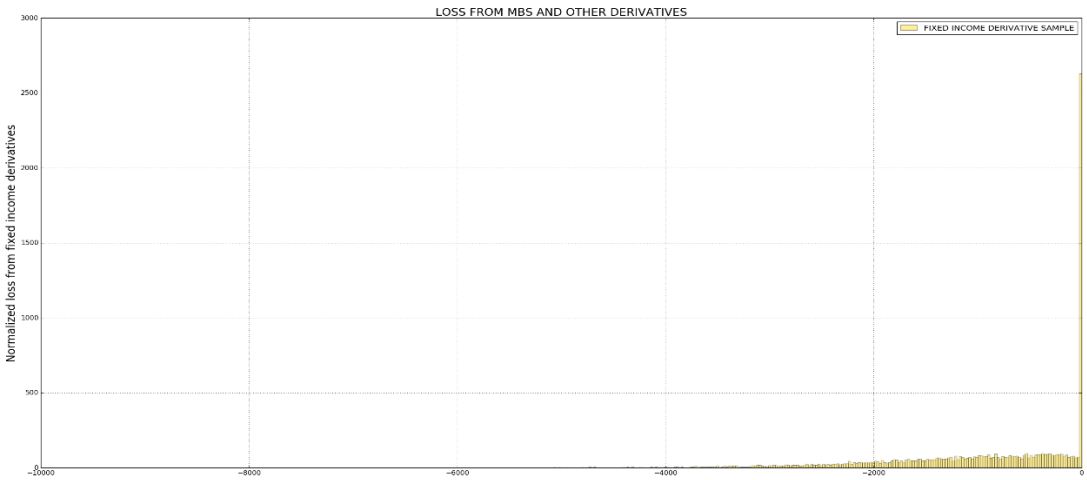

Figure 3.3 Prepayment loss and interest loss distribution of MBS

After hours of efforts, I have (naively) model the risks reside in stocks, stock derivatives, and fixed income instruments. The risk factors in these sections are related, which is modeled by assuming a multivariate Gaussian distribution of underlying Brownian motion. At the end of the day, I summed up the loss distribution of each sector and got the result of figure 3.4, the total loss distribution of the portfolio. By the final result, one can easily calculate VaR by easy math.

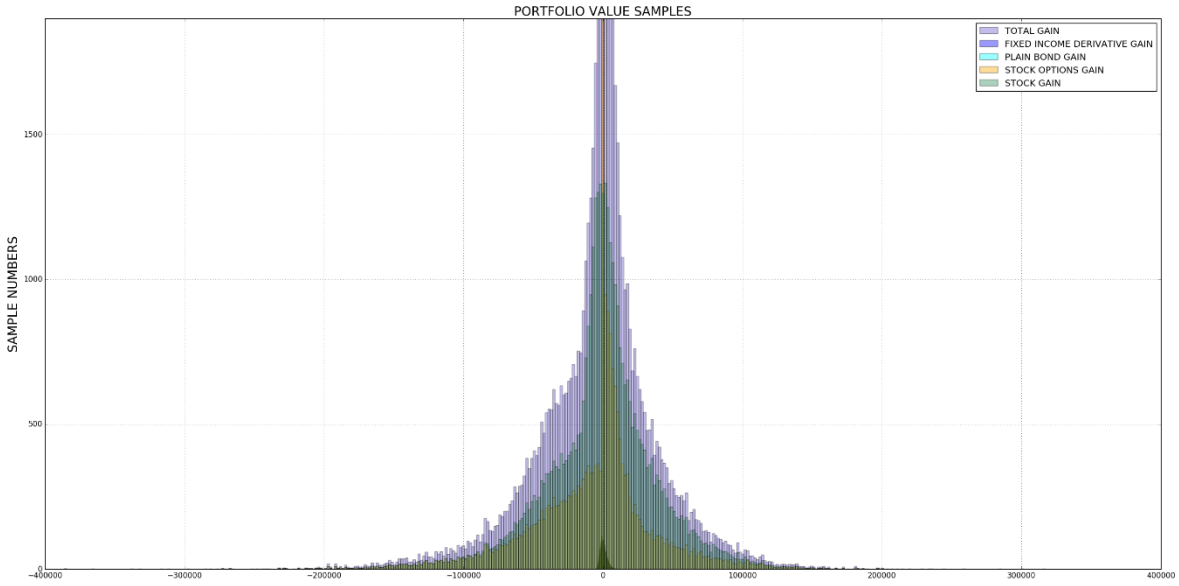

Figure 3.4 Loss distribution of portfolio

Summary

MCN VaR model is a combination of Neural Network, Cross Correlation, and Monte Carlo techniques. The validity of which still needs to be further examined.

23:00 pm at Mugar Library, Boston University

Appendix: Research Python Code

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from scipy import optimize
from  sklearn.preprocessing import normalize

class trainer(object):
    def __init__(self,NN):
        self.N=NN

    def cost_func(self,parameters,x,y):
        self.N.set_parameters(parameters)
        cost=self.N.total_err(x,y)
        gradient=self.N.weight_gradient(x,y)
        return cost,gradient

    def train(self,x,y):
        self.x=x
        self.y=y
        parameters0=self.N.get_parameters()
        res=optimize.minimize(self.cost_func,x0=parameters0,jac=True, \
                              args=(x,y),method="BFGS",options={"maxiter":20000,"disp":True})
        self.N.set_parameters(res.x)
        print res.x

class neural_network(object):
    def __init__(self, hidden_layer=3,output_layer=1,Lambda=0.0005):

        self.input_layer=6
        self.hidden_layer=self.input_layer*2+1
        self.output_layer=1
        self.Lambda=Lambda
        self.w1=np.random.randn(self.input_layer,self.hidden_layer)
        self.w2=np.random.randn(self.hidden_layer,self.output_layer)

    def forward_prop(self,x):
        self.z2=np.dot(x,self.w1)
        self.a2=self.sigmoid(self.z2)
        self.z3=np.dot(self.a2,self.w2)
        y_hat=self.sigmoid(self.z3)
        return y_hat



    def weight_gradient(self,x,y):
        # gradient with respect to w2
        yhat=self.forward_prop(x)
        sigma3=-1*(y-yhat)*self.sigmoid_prime(self.z3)
```

```
        w2gradient=np.dot(np.transpose(self.a2),sigma3)+self.Lambda*self.w2


        w1gradient=np.dot(sigma3,np.transpose(self.w2))
        w1gradient*=self.sigmoid_prime(self.z2)
        w1gradient=np.dot(np.transpose(x),w1gradient)+self.Lambda*self.w1

        return np.concatenate((w1gradient.ravel(),w2gradient.ravel()))


        #gradient with respect to w1

    def sigmoid_prime(self,x):
        return (2/(np.exp(x)+np.exp(-x)))**2

    def total_err(self,x,y):
        yhat=self.forward_prop(x)
        J= 0.5 * sum((y -yhat) ** 2)+(self.Lambda/2)+0.5*self.Lambda*(np.sum(self.w1**2)+np.sum(self.w2**2))
        return J

    def sigmoid(self,x):
        return np.tanh(x)

    def get_parameters(self):
        return np.concatenate((self.w1.ravel(),self.w2.ravel()))

    def set_parameters(self,parameters):
        w1_start=0
        w1_end=self.input_layer*self.hidden_layer
        w2_end=w1_end+self.hidden_layer*self.output_layer
        self.w1=np.reshape(parameters[w1_start:w1_end],(self.input_layer,self.hidden_layer))
        self.w2=np.reshape(parameters[w1_end:w2_end],(self.hidden_layer,self.output_layer))



data = pd.read_excel("SSS.xlsx", parse_dates=True)
data.loc[:,"MORTGAGE COUPON"]=normalize((data.loc[:,"MORTGAGE COUPON"].values))[0]
data.loc[:,"CPR"]=normalize((data.loc[:,"CPR"].values))[0]
data.loc[:,"CURRENT SPOT"]=normalize((data.loc[:,"CURRENT SPOT"].values))[0]
data.loc[:,"LAGGED SPOT1"]=normalize((data.loc[:,"LAGGED SPOT1"].values))[0]
data.loc[:,"LAGGED SPOT2"]=normalize((data.loc[:,"LAGGED SPOT2"].values))[0]
data.loc[:,"LAGGED SPOT3"]=normalize((data.loc[:,"LAGGED SPOT3"].values))[0]
data.loc[:,"HOLDING TIME"]=normalize((data.loc[:,"HOLDING TIME"].values))[0]

training_set=data[:int(len(data)*0.7)]
training_input=training_set[["MORTGAGE COUPON","CURRENT SPOT","LAGGED SPOT1",
                             "LAGGED SPOT2","LAGGED SPOT3","HOLDING TIME"]].as_matrix()
training_output=training_set[["CPR"]].as_matrix()

testing_set=data[int(len(data)*0.7):]
testing_input=testing_set[["MORTGAGE COUPON","CURRENT SPOT","LAGGED SPOT1",
                             "LAGGED SPOT2","LAGGED SPOT3","HOLDING TIME"]].as_matrix()
testing_output=testing_set[["CPR"]].as_matrix()


pass
NN=neural_network()
TR=trainer(NN)
TR.train(training_input,training_output)

"fitting result"
#
# predicting_result=[]
# for i in range(len(training_input)):
#     res_temp=NN.forward_prop(training_input[i])
#     predicting_result.append(res_temp)
#
# fig=plt.figure()
# ax=fig.add_subplot(111)
# ax.bar(np.arange(len(training_input)),training_output,color="green",alpha=0.5)
# ax.bar(np.arange(len(training_input)),predicting_result,color="blue",alpha=0.4)
# ax.legend(["True","Fitted"])
# plt.ylim([0.02,0.08])
# plt.xlim([-1,85])
#
# plt.grid()
# plt.show()
```

```
"testing result"


predicting_result=[]
for i in range(len(testing_input)):
    res_temp=NN.forward_prop(testing_input[i])
    predicting_result.append(res_temp)

fig=plt.figure()
ax=fig.add_subplot(111)
ax.bar(np.arange(len(testing_input)),testing_output,color="green",alpha=0.5)
ax.bar(np.arange(len(testing_input)),predicting_result,color="blue",alpha=0.4)
ax.legend(["True","Fitted"])
plt.ylim([0.02,0.08])
plt.xlim([-1,85])

plt.grid()
plt.show()


# data.loc[:,0].values=normalize(list(data.ix[:,0].values))[0]
# print data.ix[:,0]
```

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from scipy.stats import norm


zerobond=0

def diffusion(time,sigma,sample=1):

    step=len(time)
    delta_t=(float(time[-1]-time[0]))/step
    brownian_one=norm.rvs(size=step,loc=0,scale=delta_t*np.sqrt(sigma))
    return brownian_one


def vasicek_model(time,r0,k,miu,sigma):
    step=len(time)
    delta_t = (float(time[-1] - time[0])) /step
    b=diffusion(time,sigma)
    short_rate=[r0]
    for i in range(len(time)):
        dr=k*(miu-short_rate[-1])*delta_t+b[i]
        short_rate.append(short_rate[-1]+dr)
    return short_rate


def bond_dynamic():

    T=1.0
    b=[0.87]
    steps=50
    r0=0.013 #parameters of short rate model
    miu=0.015 #parameters of short rate model
    sigma = 0.15 #parameters of short rate model
    k=3.8 #parameters of short rate model
    time=np.linspace(0,T,steps)
    delta_t=T/steps
    rvs= norm.rvs(loc=0, scale=1, size=steps)
    short_rate=vasicek_model(time,r0,k,miu,sigma)
    for i in range(len(time)):
        db=(b[-1]**2)*sigma*np.sqrt(delta_t)*rvs[i]+b[-1]*short_rate[i]*delta_t
        b.append(b[-1]+db)
    return b[-1]


def bondNopt_var(simulation_num, principle,opt_position):
```

```
    b=0.87#starting price
    strike=0.8# callbale strike
    z=[]
    zopt=[]
    for i in range(simulation_num):
        z.append(principle*(bond_dynamic()-b))
        v=opt_position*max(0,(bond_dynamic()-strike))
        zopt.append(v)

    return z,zopt
```

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from scipy.stats import norm

def simu(data,C,s0,step,miu,delta_t,sigma,T,sim_num):
    bm = []
    for i in range(len(data)):
        temp = norm.rvs(loc=0, scale=1, size=step)
        bm.append(temp)
    bm=np.dot(C,bm)
    dynamics = []
    for x in range(len(data)):
        temp = [s0[x]]
        for j in range(step):
            ds = temp[-1] * (miu * delta_t + sigma*np.sqrt(delta_t) * bm[x][j])
            temp.append(ds + temp[-1])
        dynamics.append(temp)

    for i in range(len(dynamics)):
        plt.plot(np.linspace(0, T, step + 1), dynamics[i])



#s=s0*exp(ut+sigmawt)
def stockopt_var(rvs,C,simulation_num,s0,k,position):
    bm = np.dot(C, rvs)
    sigma=0.25*np.ones(len(s0))
    r=0.05
    T=1.0
    sum = 0
    z = []
    #generating Z

    for i in range(len(s0)):

        for j in range(len(bm[0])):
            st = s0[i] * np.exp(r * T - 0.5 * sigma[i] ** 2 * T + sigma[i] * np.sqrt(T) * bm[i][j])
            p = position[i]*max(0, st - k[i])
            z.append(p)
            sum += p

    return z
```

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from scipy.stats import norm
from  sklearn.preprocessing import normalize


def stock_var(rvs,C,simulation_num,s0,k,position):
    bm =np.dot(C, rvs)
    sigma=0.25*np.ones(len(s0))
    r=0.05
    T=1.0
    sum = 0
    z = []
```

```
    for i in range(len(s0)):

        for j in range(len(bm[0])):
            st = s0[i] * np.exp(r * T - 0.5 * sigma[i] ** 2 * T + sigma[i] * np.sqrt(T) * bm[i][j])
            p = position[i]*(st - s0[i])
            z.append(p)
            sum += p

    return z
```

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from scipy.stats import norm
from  sklearn.preprocessing import normalize
from stocks import*
from stock_options import *
from bonds_and_options import *


data = pd.read_excel("equity.xlsx", parse_dates=True)
# data.loc[:,"TSLA"]=normalize((data.loc[:,"TSLA"].values))[0]
# data.loc[:,"MAC"]=normalize((data.loc[:,"MAC"].values))[0]
# data.loc[:,"BIIB"]=normalize((data.loc[:,"BIIB"].values))[0]
# data.loc[:,"GS"]=normalize((data.loc[:,"GS"].values))[0]

data=data.as_matrix()
data=data.T

cov_matrix=np.corrcoef(data)
C=np.linalg.cholesky(cov_matrix)
Ct=C.transpose()


T=0.5
miu=0.03
sigma=0.08
step=100
delta_t=T/step
s0=data[:,0]
k=s0
simulation_num=10000
position=[-1000,-8000,2000,300]
stock_position=[900,700,-1500,-500]
zero_principle=20000.0
bondoptposition=-10000

rvs=[]
for i in range(len(s0)):
    temp = norm.rvs(loc=0, scale=1, size=simulation_num)
    rvs.append(temp)
bm = np.dot(C, rvs)


# for i in range(30):
#     simu(data,C,s0,step,miu,delta_t,sigma,T,simulation_num)
#
#
# plt.title("GBM STOCK DYNAMIC",fontsize=20)
# plt.ylabel("STOCK PRICE",fontsize=20)
# plt.grid()
# plt.show()




var_stock=stock_var(rvs,C,simulation_num,s0,k,stock_position)
var_stock_opt=stockopt_var(rvs,C,simulation_num,s0,k,stock_position)
var_bond,var_bond_opt=bondNopt_var(simulation_num,zero_principle,bondoptposition)
```

```
var_portfolio=var_stock+var_stock_opt+var_bond+var_bond_opt


plt.hist(var_portfolio,bins=400,alpha=0.40,color="slateblue")
plt.hist(var_bond_opt,bins=400,alpha=0.40,color="blue")
plt.hist(var_bond,bins=400,alpha=0.40,color="cyan")
plt.hist(var_stock_opt,bins=400,alpha=0.40,color="orange")
plt.hist(var_stock,bins=400,alpha=0.40,color="seagreen")
plt.legend(["TOTAL GAIN","FIXED INCOME DERIVATIVE GAIN","PLAIN BOND GAIN","STOCK OPTIONS GAIN","STOCK GAIN"])
plt.title("PORTFOLIO VALUE SAMPLES",fontsize=20)
plt.ylabel("SAMPLE NUMBERS",fontsize=20)
plt.ylim([0,1900])
plt.grid()
plt.show()
```