Problem Chosen	2022 MCM/ICM Summary Sheet	Team Control Number
C		2219948

ANN Neural Network Maximizing Decision Based on LSTM Value Prediction

Summary

We focus on maximizing the profit **per day**, which also means that the overall profit is maximized in the long run. Therefore, we use two models to maximize the profit per day:

Model one: Use **LSTM** to predict the price of the next day using all the previous data.The Long Short Term network uses all the data before today as a training set. After training, the Long Short Term network can predict the price of the next day in order to provide information for maximizing profits today.

Model two: ANN with constrained neuron .The input to the artificial neural network (4.1) is [1000, 0, 0], representing the value of [cash, gold, bitcoin]. ANN goes through several hidden layers, the number of hidden layers is determined by the time period, and its output is the final value. Among them, neuron means the ratio of [cash, gold, bitcoin] exchanged with each other every day. Neurons connected to the same parent node and are limited to 1. By training the neural network, ANN is able to give the best strategy for each day and the final payoff. At the same time, we prove that Model 2 can provide the optimal trade strategy, which maximizes the returns.

First, we performed a sensitivity analysis on Model One to assess its predictive power. After filling in the data of non-trading days of gold, we used model 1 to predict the prices of gold and bitcoin from the 31st day to the last day, and obtained the actual growth rate and predicted growth rate of each day.

Second, we use the actual growth rate and Model 2 to calculate the best decisions and final returns for each day of the past five years. Our model shows that $ 1,000 invested 5 years ago will be $**670,205,1622,912** five years later, assuming the daily growth rates of gold and Bitcoin are known.

After that, we use the growth rate predicted by Model 1 for each day and Model 2 to calculate the maximum gain that can be obtained in 5 years. Our model shows that an investment of $1,000 5 years ago will turn into $**201755.0312** in 5 years, without knowing the future price trend.

Finally, we perform a sensitivity analysis on the model by varying the operation interval and the number of forecast sets, evaluating the impact of transaction price on the strategy. At the end of the article, we explain and communicate with traders about our strategies, models, and results.

Keywords: LSTM; ANN with constrained neuron;time series forecast;

Contents

1 Introduction

1.1 Outlilne

Our goal is to develop and implement models to better maximize revenue, which can make optimal decisions based on previous data, and evaluate models and make targeted recommendations.

- builds a model that predicts the price of gold and bitcoin for the next day using data from the previous day.
- builds a model that can make the best decision for the day based on the predicted data for the next day, and calculate the final benefit of the decision.
- First, use the first model to predict the price of the next day using the data before each day, take the predicted value of model 1 as a known value, use model 2 to make decisions, and finally calculate the final result based on the real data.
- evaluates and analyzes the sensitivity of the model, and proves that the decision given by model 2 can maximize the benefits, and explains the model and puts forward suggestions for use.

1.2 Main Assumptions

Note that gold has trading days and non-trading days while Bitcoin does not, in order to better fit the data to the model.

- We assume that when bitcoin and gold are less than the minimum purchase price, very little cash can still buy gold and bitcoin.
- We assume that the gold price on the non-trading day is the same as the previous last trading day and can be bought and sold.
- We assume that the future price of Bitcoin is mainly related to the price of the last 30 days

2 Notations

Table 1: the definition of symbol

Symbol	Definition
α	gold growth rate
β	Bitcoin's growth rate
A, B	matrix

3 Model I: Prediction model based on LSTM

This papers uses LSTM networks to predict the price of gold and bitcoin after each day, respectively, to determine the best decision preparation for the subsequent ANN model.

3.1 Model testing and dataset determination

3.1.1 Model testing

(1)Dataset

We choose the BCHAIN-MKPRU.csv test model, and we take the last 90 samples of its dataset as the training set, and the rest as the training set.1 is the price trend of Bitcoin and the division of the training set and the test set, the yellow one is the training set, and the green one is the test set.

Figure 1: Bitcoin price trend and data division

(2)Parameter selection and data processing

We assume that the future price of Bitcoin is mainly related to the price of the last 30 days, so we set

$$N - gram = 30$$

N-gram is a parameter in the LSTM that determines the dimension of the input.

We set the value of N-gram to 30, which means that every 30 consecutive data in the original data is used as a training set, and the data of the next day is used as a label.We set the value of N-gram to 30, which means that every 30 consecutive data in the original data is used as a training set, and the data of the next day is used as a label

(3)Train and Test

We train each sample 100 times and test it on the test set to compare with the real data and get the following results:

The left image shows the trained data and the data predicted by the model, while the right image shows the comparison between the predicted data and the real data

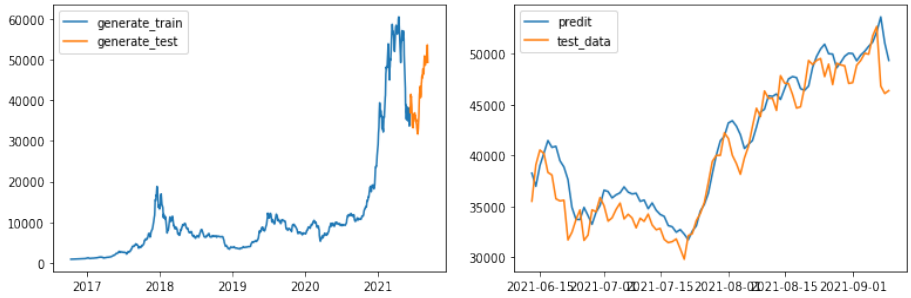


Figure 2: Actual vs Predicted

3.2 Numerical prediction

Since the data of gold is partially missing and does not include the data of non-trading days, we fill in the missing data with the data of the previous last trading day.

We use the model to predict the price of gold and bitcoin after each day, using the previous data, and compare it with the real data.

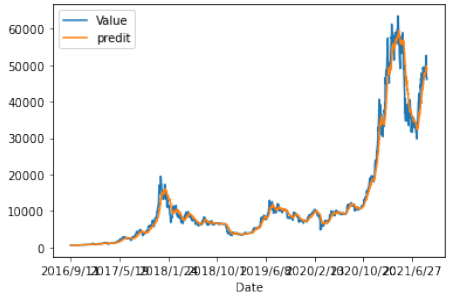


Figure 3: Actual vs Predicted

4 Model II: ANN with constrained neuron

4.1 Building an artificial neural network

Artificial neural network is a parallel distributed system. It adopts a completely different mechanism from traditional artificial intelligence and information processing technology. It overcomes the shortcomings of traditional artificial intelligence based on logical symbols in processing intuition and unstructured information. Features of self-organization and real-time learning.

In Model 1, we have used LSTM to get the predicted price for each day, and we find the actual and predicted growth rate of gold and Bitcoin each day in order to better build the neural network.

We assume that the value of cash, gold, and bitcoin held on day i is

$$[m_1, g_1, b_1], i = 1, 2, 3, \ldots, 1826$$

We assume that n is a natural number ,then

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix}_{n\times n} \quad B = \begin{bmatrix} b_{11} & b_{12} & \dots & b_{1n} \\ b_{21} & b_{22} & \dots & b_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ b_{n1} & b_{n2} & \dots & b_{nn} \end{bmatrix}_{n\times n}$$

Then we define the symbol $*$,@ satisfies the following operation

$$A * B = \begin{bmatrix} a_{11}b_{11} & a_{12}b_{12} & \dots & a_{1n}b_{1n} \\ a_{21}b_{11} & a_{22}b_{12} & \dots & a_{2n}b_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{n1}b_{n1} & a_{n2}b_{n2} & \dots & a_{nn}b_{nn} \end{bmatrix}$$

$$A@B = \begin{bmatrix} \sum_{i=1}^{n} a_{1i}b_{i1} & \sum_{i=1}^{n} a_{1i}b_{i2} & \dots & \sum_{i=1}^{n} a_{1i}b_{in} \\ \sum_{i=1}^{n} a_{2i}b_{i1} & \sum_{i=1}^{n} a_{2i}b_{i2} & \dots & \sum_{i=1}^{n} a_{2i}b_{in} \\ \vdots & \vdots & \vdots & \vdots \\ \sum_{i=1}^{n} a_{ni}b_{i1} & \sum_{i=1}^{n} a_{ni}b_{i2} & \dots & \sum_{i=1}^{n} a_{ni}b_{in} \end{bmatrix}$$

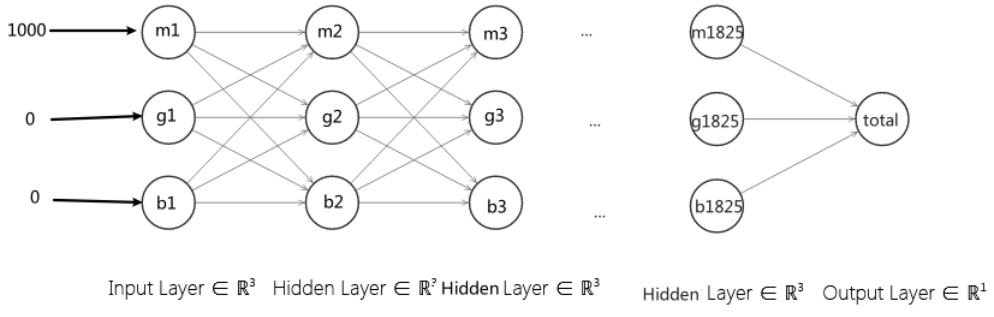
So the change in cash, gold, bitcoin from day 1 to day 2 can be expressed by the following equation:

$$[m_i, g_i, b_i]@(\begin{bmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{bmatrix} * \begin{bmatrix} 1 & 0.99 & 0.98 \\ 0.99 & 1 & 0.9702 \\ 0.98 & 0.9702 & 1 \end{bmatrix}) \begin{bmatrix} 1 & 0 & 0 \\ 0 & \alpha & 0 \\ 0 & 0 & \beta \end{bmatrix} = [m_{i+1}, g_{i+1}, b_{i+1}]$$

Where C_{ij} means the ratio of cash, gold, bitcoin exchanged from the first day to the second day.

α means gold's Growth Rate from Day 1 to Day 2; β means bitcoin's Growth Rate from Day 1 to Day 2

After this, we only need to know the C matrix to get the state of the next day, and the C matrix just represents the decision of each day. Therefore, we build a neural network based on this.



We can get such a neural network. However, we only notice that the neurons of the neural network are constrained, the sum of the neurons from each node must be 1, and the value of the neuron is in (0, 1) between.

4.2 Training of Artificial Neural Networks

The traditional neural network uses the shaving descent algorithm to find the minimum:

4.2.1 How to update the weights of traditional neural networks

(1)Define the loss function In a linear regression problem, we define that

$$y = w * x + b$$

y means dependent variable ,x means the independent variable,w, bmeans the independent variable,then we define loss function:

$$loss = \sum_{i}^{N} (wx_i + b - y_i)^2$$

Then we use Gradient descent method to solve the most suitable w, b to minimize $loss$

(2)Gradient descent method We use

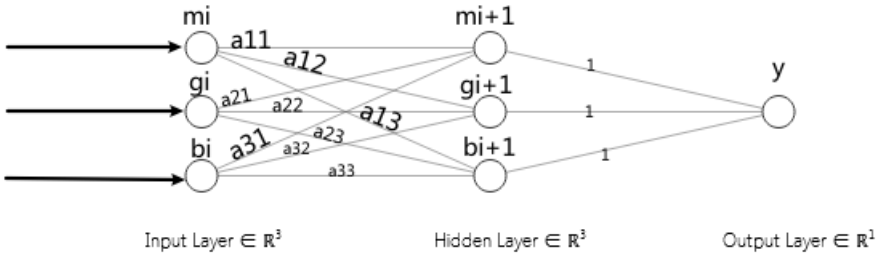
$$w_{i+1} = w_i - \alpha * \frac{dloss}{dwi} \qquad (1)$$

$$b_{i+1} = b_i - \alpha * \frac{dloss}{dbi} \qquad (2)$$

where α is a Constants which means Gradient descent speed.By continuously iterating (1) and (2),until it tends to be stable at last, so that we figure the most suitable w and b,which makes the value of loss smallest.

4.2.2 How to update the weights of our neural network

We consider the network layer from day i to day i+1:



We can get the system of equations:

$$\begin{cases} y = (a_{11} + a_{12} * 0.99 * \alpha + a_{13} * 0.98 * \beta) * m_i + (a_{21} * 0.98 + a_{22} * \alpha + a_{23} * 0.9702 * \beta) * g_i \\ \qquad\qquad + (a_{31} * 0.98 + a_{32} * 0.9702 * \alpha + a_{33} * \beta) * b_i, \\ m_i, g_i, b_i > 0, \alpha, \beta > 0 \\ a_{ij} > 0, 0 < i, j < 1 \\ a11 + a12 + a13 = 1 \\ a21 + a22 + a23 = 1 \\ a31 + a32 + a33 = 1 \end{cases}$$

because:

$$m_i, g_i, b_i > 0$$

Therefore if and only if

$$m_{i+1}, g_{i+1}, b_{i+1}$$

is maximum,y is maximum.because

$$m_i > 0, \alpha, \beta > 0, a_{11} + a_{12} + a_{13} = 1$$

and

$$\begin{cases} \dfrac{\partial m_{i+1}}{\partial a_{11}} = m_i \\ \dfrac{\partial m_{i+1}}{\partial a_{12}} = 0.99 * \alpha * m_i \\ \dfrac{\partial m_{i+1}}{\partial a_{13}} = 0.98 * \beta * m_i \end{cases}$$

makes $max(\frac{\partial m_{i+1}}{\partial a_{11}}, \frac{\partial m_{i+1}}{\partial a_{12}}, \frac{\partial m_{i+1}}{\partial a_{13}})$ of Coefficient is 1.In this way, we have completed the update of the parameters.

4.3 Use ANN figure total money

4.3.1 Use ANN figure total money assume we know tomorrow data

In order to test the effect of model II, we use the actual growth rate to calculate the benefits within 5 years. The result is as follows:

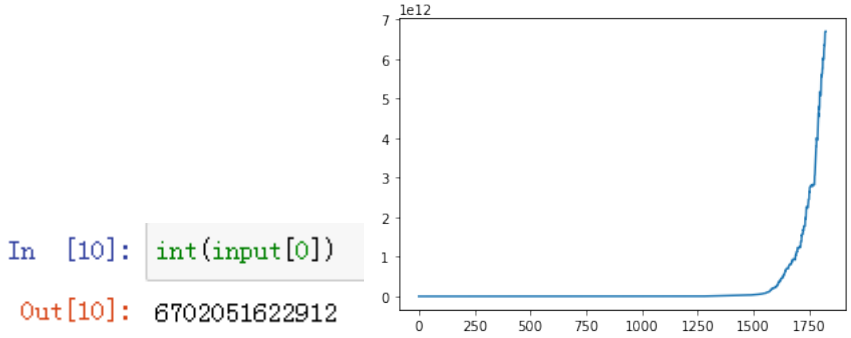


Figure 4: Final results and gain graph

With the actual data,our model shows that there will be $6,702,051,622,912 in the five years.

4.3.2 Use ANN figure total money with the predited data

Then we use data of predited by model one to figure total gain in five years.The result is as follows:

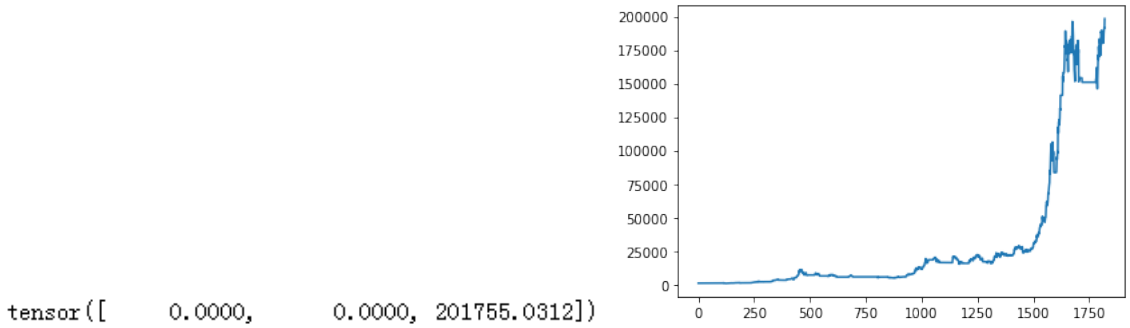


Figure 5: Final results and gain graph

5 sensitivity analysis

In order to explore the sensitivity of the model, we set different parameters.

The first column of the table and the second column show the impact of different initial values of cash, gold, and bitcoin on models. The data predicted by the LSTM model is predicted with the real data, and the third column and fourth column show the prediction data forward movement of the model. Table Fifth and Sixth Columns Display Columns The Impact of Decision Interval on Models.

Influence of initial value		Prediction curve offset		interval days	
[m,g,b]	out	Offset	out	1	201755.0312
[1000,0,0]	201755.0312	zero day	16707.7793	2	2857.5825
[0,1000,0]	181458.7812	one day	52095.0742	3	6190.3413
[0,0,1000]	196965.4375	two days	135380.25	4	1912.0824
[1000,1000,1000]	576700	three days	201755.0312	5	709.5153
[3000,0,0]	605265.625	four days	285937.7812	10	3602.0391
[0,3000,0]	553928.375	five days	199559.3281	15	682.197

Figure 6: The Impact of Different Parameter Values on Models

Through the above contrast, it is not difficult to obtain, the initial value of the data is not much affected by the model; the model is sensitive to the predicted value; the predicted value can be improved; the model is used for the maximum share of daily benefits. When the decision time becomes long, the model sensitivity is greatly reduced.

6 Strengths and weaknesses

6.1 Strengths

- The model can maximize daily benefits
- The model is less risky
- Model adaptability is strong

6.2 weaknesses

- Model two strong relying model first prediction results

References

[1] Wu Bo. Stock Price Forecast Based on LSTM Model [D]. Dalian University of Technology, 2021.DOI: 10.26991 / D.cnki.gdllu.2021.003347.

Appendices

Appendix A First appendix

Dear,investor:

Stock is unpredictable, if you know the daily price trend, you can earn $1,000$ 600,000,000,000, but this means that you have the interests every day. When the principal is large enough, wealth will accumulate quickly. Therefore, although our model is too far from the target results, our model only needs to predict tomorrow's rise and fall, of course, sometimes it only needs to predict it to one, LSTM's prediction probability is 0.6, only It takes 0.6, our model has won! Of course, you may notice that our model puts all the money in a basket, but don't have to worry, but the price will be keen to perceive and timely stop. You can modify the model to accommodate more, you can purchase a variety of items at the same time, you can also achieve better predictive effects and decision-making recommendations by adjusting the parameters or for longer taps.

Sincerely yours,

Your friends

Appendix B Second appendix

Here are simulation programmes we used in our model as follow.

Input python source:

```
import torch
from torchvision import datasets
import torchvision
from torchvision import transforms
from torch.utils.data import DataLoader
from torch.nn import functional as F
from torch import optim,nn
from matplotlib import pyplot as plt
import numpy as np
from sympy import Matrix
from sympy.matrices import dense
```

```
from math import ceil,floor,atan
import pandas as pd

Bit = pd.read_csv("BCHAIN-MKPRU.csv", index_col=0)
gold = pd.read_csv("LBMA-GOLD.csv", index_col=0)

decide_days=5

Gold=pd.DataFrame(np.random.randn(1827//decide_days,4))
bit=pd.DataFrame(np.random.randn(1827//decide_days,4))
for i in range(1827//decide_days):
    bit.iloc[i,:]=Bit.iloc[i*decide_days,:]
    Gold.iloc[i,:]=gold.iloc[i*decide_days,:]
for j in range(1827//decide_days-1):
    bit.iloc[j+1,3]=(bit.iloc[j+1,3]-bit.iloc[j,3])/bit.iloc[j,3]+1
    Gold.iloc[j+1,3]=(Gold.iloc[j+1,3]-Gold.iloc[j,3])/Gold.iloc[j,3]+1
num = 1827//decide_days

days=1826
increase_rate=torch.randn(days,1,3)
square_rate=torch.zeros(days,3,3)
brokerage=torch.tensor([[1,0.99,0.98],[0.99,1,0.99*0.98],[0.98,0.98*0.99,1]])
input=torch.tensor([1000.0,0,0])
weight=torch.zeros(days,3,3)

increase_rate[:,:,0]=1
for i in range(days-1):
    increase_rate[i,:,1]=gold.iloc[i+1,1]
    increase_rate[i,:,2]=Bit.iloc[i+1,1]

for i in range(days):
    square_rate[i,0,0]=1
    square_rate[i,1,1]=increase_rate[i,:,1]
    square_rate[i,2,2]=increase_rate[i,:,2]

def maxmoney(input,current_days,weight):
    current_brokerage=torch.zeros(3,3)
    for i in range(3):
        target=brokerage[i]*increase_rate[current_days]
        index=int(target.argmax())
        weight[current_days,i,index]=1
        current_brokerage[i,i]=brokerage[i,index]
    out=input@weight[current_days]@current_brokerage@square_rate[current_days]
    return out,weight[current_days]

input=torch.tensor([1000.0,0,0])
s=pd.Series(np.random.randn(1824))
for g in range(days-1):
    print(g,input)
    input,weight[g]=maxmoney(input,g,weight)
    s[g]=int(max(input))

s.plot()
```

```
import pandas as pd
import matplotlib.pyplot as plt
import datetime
import torch
import torch.nn as nn
import numpy as np
from torch.utils.data import Dataset, DataLoader

n=30
LR=0.0001

def get_data_n_days(series, n):
    df = pd.DataFrame()
    for i in range(n):
        df['c%d' % i] = series.tolist()[i:-(n - i)]
    df['last'] = series.tolist()[n:]
    return df

def readData(column, n,train_end):
    df = pd.read_csv("BCHAIN-MKPRU.csv", index_col=0)
    df_column = df[column].copy()
    df_column_train, df_column_test = df_column[:train_end], df_column[train_end - n:]
    target_df = get_data_n_days(df_column_train, n)
    return target_df, df_column, df.index.tolist()

class RNN(nn.Module):
    def __init__(self, input_size):
        super(RNN, self).__init__()
        self.rnn = nn.LSTM(
            input_size=input_size,
            hidden_size=64,
            num_layers=1,
            batch_first=True
        )
        self.out = nn.Sequential(
            nn.Linear(64, 1)
        )

    def forward(self, x):
        r_out, (h_n, h_c) = self.rnn(x, None)
        out = self.out(r_out)
        return out

class TrainSet(Dataset):
    def __init__(self, data):

        self.data, self.label = data[:, :-1].float(), data[:, -1].float()

    def __getitem__(self, index):
        return self.data[index], self.label[index]

    def __len__(self):
        return len(self.data)
```

```
def get_train_data(train_end):
    df, df_all, df_index = readData('Value',30,train_end)
    df_all = np.array(df_all.tolist())
    df_numpy = np.array(df)
    df_numpy_mean = np.mean(df_numpy)
    df_numpy_std = np.std(df_numpy)
    df_numpy = (df_numpy - df_numpy_mean) / df_numpy_std
    df_tensor = torch.Tensor(df_numpy)
    trainset = TrainSet(df_tensor)
    trainloader = DataLoader(trainset, batch_size=10, shuffle=True)
    return trainloader

def get_data_n_days_predit(series, n):
    df = pd.DataFrame()
    for i in range(n):
        df['c%d' % i] = series.tolist()[i:-(n - i)]
    df['last'] = series.tolist()[n:]
    return df

def predit_one(rnn,df,train_end):
    #rnn=RNN(n)

    #currnt_data=df.index[-1]
    #target_data=pd.date_range(df.index[-1],periods=2)[1]
    #df.loc[target_data]=[value]
    return value

df = pd.read_csv("BCHAIN-MKPRU.csv", index_col=0)

current=0

for i in range(current,1826-30):

    trainloader=get_train_data(-(i+1))

    rnn = RNN(n)
    optimizer = torch.optim.Adam(rnn.parameters(), lr=LR)
    loss_func = nn.MSELoss()

    for step in range(10+i//100):
        for tx, ty in trainloader:
            output = rnn(torch.unsqueeze(tx, dim=0))
            loss = loss_func(torch.squeeze(output), ty)
            optimizer.zero_grad()  # clear gradients for this training step
            loss.backward()  # back propagation, compute gradients
            optimizer.step()
        print(i,1826-i,step, loss)
    current=i

    df1=df.iloc[-i-31:-i-1,0]
    x=get_data_n_days_predit(df1, 29)

    x_numpy = np.array(x)
    x_numpy_mean = np.mean(x_numpy)
```

```
    x_numpy_std = np.std(x_numpy)
    x_numpy = (x_numpy - x_numpy_mean) / x_numpy_std
    x_tensor = torch.Tensor(x_numpy)

    target_tensor=torch.zeros(1,x_tensor.size()[0],x_tensor.size()[1])
    target_tensor[0]=x_tensor

    value=float(rnn(target_tensor))
    value = value * x_numpy_std+x_numpy_mean

    df.iloc[-(i+1),2]=value
    df.to_csv("BCHAIN-MKPRU.csv")

predit_increase_rate=torch.randn(days,1,3)
predit_square_rate=torch.zeros(days,3,3)
brokerage=torch.tensor([[1,0.99,0.98],[0.99,1,0.99*0.98],[0.98,0.98*0.99,1]])
input=torch.tensor([1000.0,0,0])
weight=torch.zeros(days,3,3)

predit_increase_rate[:,:,0]=1
for i in range(days-4):
    predit_increase_rate[i,:,1]=gold.iloc[i+4,3]
    predit_increase_rate[i,:,2]=Bit.iloc[i+4,3]

for i in range(days):
    predit_square_rate[i,0,0]=1
    predit_square_rate[i,1,1]=predit_increase_rate[i,:,1]
    predit_square_rate[i,2,2]=predit_increase_rate[i,:,2]

def maxmoney(input,current_days,weight):
    current_brokerage=torch.zeros(3,3)
    for i in range(3):
        target=brokerage[i]*predit_increase_rate[current_days]
        index=int(target.argmax())
        weight[current_days,i,index]=1
        current_brokerage[i,i]=brokerage[i,index]
    out=input@weight[current_days]@current_brokerage@square_rate[current_days]
    return out,weight[current_days]

input=torch.tensor([0.0,3000,0])
y=pd.Series(np.random.randn(1826))
for g in range(days-2):
    print(g,input)
    input,weight[g]=maxmoney(input,g,weight)
    y[g]=int(max(input))

y[:-5].plot()

Bit = pd.read_csv("BCHAIN-MKPRU.csv", index_col=0)
gold = pd.read_csv("LBMA-GOLD.csv", index_col=0)

decide_days=5

Gold=pd.DataFrame(np.random.randn(1827//decide_days,4))
```

```
bit=pd.DataFrame(np.random.randn(1827//decide_days,4))
for i in range(1827//decide_days):
    bit.iloc[i,:]=Bit.iloc[i*decide_days,:]
    Gold.iloc[i,:]=gold.iloc[i*decide_days,:]
for j in range(1827//decide_days-1):
    bit.iloc[j+1,3]=(bit.iloc[j+1,2]-bit.iloc[j,2])/bit.iloc[j,2]+1
    bit.iloc[j+1,1]=(bit.iloc[j+1,0]-bit.iloc[j,0])/bit.iloc[j,0]+1
    Gold.iloc[j+1,3]=(Gold.iloc[j+1,2]-Gold.iloc[j,2])/Gold.iloc[j,2]+1
    Gold.iloc[j+1,1]=(Gold.iloc[j+1,0]-Gold.iloc[j,0])/Gold.iloc[j,0]+1
num = 1827//decide_days

days=num
increase_rate=torch.randn(days,1,3)
square_rate=torch.zeros(days,3,3)
brokerage=torch.tensor([[1,0.99,0.98],[0.99,1,0.99*0.98],[0.98,0.98*0.99,1]])
input=torch.tensor([1000.0,0,0])
weight=torch.zeros(days,3,3)

increase_rate[:,:,0]=1
for i in range(days-1):
    increase_rate[i,:,1]=Gold.iloc[i+1,1]
    increase_rate[i,:,2]=bit.iloc[i+1,1]

for i in range(days):
    square_rate[i,0,0]=1
    square_rate[i,1,1]=increase_rate[i,:,1]
    square_rate[i,2,2]=increase_rate[i,:,2]

predit_increase_rate=torch.randn(days,1,3)
predit_square_rate=torch.zeros(days,3,3)
brokerage=torch.tensor([[1,0.99,0.98],[0.99,1,0.99*0.98],[0.98,0.98*0.99,1]])
input=torch.tensor([1000.0,0,0])
weight=torch.zeros(days,3,3)

predit_increase_rate[:,:,0]=1
for i in range(days-4):
    predit_increase_rate[i,:,1]=gold.iloc[i+4,3]
    predit_increase_rate[i,:,2]=Bit.iloc[i+4,3]

for i in range(days):
    predit_square_rate[i,0,0]=1
    predit_square_rate[i,1,1]=predit_increase_rate[i,:,1]
    predit_square_rate[i,2,2]=predit_increase_rate[i,:,2]

def maxmoney(input,current_days,weight):
    current_brokerage=torch.zeros(3,3)
    for i in range(3):
        target=brokerage[i]*predit_increase_rate[current_days]
        index=int(target.argmax())
        weight[current_days,i,index]=1
        current_brokerage[i,i]=brokerage[i,index]
    out=input@weight[current_days]@current_brokerage@square_rate[current_days]
    return out,weight[current_days]
```

```
input=torch.tensor([1000.0,0,0])
y=pd.Series(np.random.randn(1826))
for g in range(days-2):
    print(g,input)
    input,weight[g]=maxmoney(input,g,weight)
    y[g]=int(max(input))
```

Input python source:

```
import pandas as pd
import matplotlib.pyplot as plt
import datetime
import torch
import torch.nn as nn
import numpy as np
from torch.utils.data import Dataset, DataLoader

def get_data_n_days(series, n):
    df = pd.DataFrame()
    for i in range(n):
        df['c%d' % i] = series.tolist()[i:-(n - i)]
    df['y'] = series.tolist()[n:]
    return df

def readData(column, n,train_end):
    df = pd.read_csv("BCHAIN-MKPRU.csv", index_col=0)
    df_column = df[column].copy()
    df_column_train, df_column_test = df_column[:train_end], df_column[train_end - n:]
    target_df = get_data_n_days(df_column_train, n)
    return target_df, df_column, df.index.tolist()

class RNN(nn.Module):
    def __init__(self, input_size):
        super(RNN, self).__init__()
        self.rnn = nn.LSTM(
            input_size=input_size,
            hidden_size=64,
            num_layers=1,
            batch_first=True
        )
        self.out = nn.Sequential(
            nn.Linear(64, 1)
        )

    def forward(self, x):
        r_out, (h_n, h_c) = self.rnn(x, None)
        out = self.out(r_out)
        return out


class TrainSet(Dataset):
    def __init__(self, data):

        self.data, self.label = data[:, :-1].float(), data[:, -1].float()
```

```
    def __getitem__(self, index):
        return self.data[index], self.label[index]

    def __len__(self):
        return len(self.data)

n = 30
LR = 0.0001
EPOCH = 100
train_end = -90



df, df_all, df_index = readData('Value', 30,-90)

df_all = np.array(df_all.tolist())
plt.plot(df_index, df_all, label='real-data')
plt.plot(df_index[:-90], df_all[:-90], label='train_data')
plt.plot(df_index[-90:], df_all[-90:], label='test_data')

df_numpy = np.array(df)

df_numpy_mean = np.mean(df_numpy)
df_numpy_std = np.std(df_numpy)

df_numpy = (df_numpy - df_numpy_mean) / df_numpy_std

df_tensor = torch.Tensor(df_numpy)

trainset = TrainSet(df_tensor)
trainloader = DataLoader(trainset, batch_size=10, shuffle=True)

rnn = RNN(n)
optimizer = torch.optim.Adam(rnn.parameters(), lr=LR)  # optimize all cnn parameters
loss_func = nn.MSELoss()

for step in range(EPOCH):
    for tx, ty in trainloader:
        output = rnn(torch.unsqueeze(tx, dim=0))
        loss = loss_func(torch.squeeze(output), ty)
        optimizer.zero_grad()  # clear gradients for this training step
        loss.backward()  # back propagation, compute gradients
        optimizer.step()
    print(step, loss)



generate_data_train = []
generate_data_test = []

test_index = len(df_all) + train_end
```

```
df_all_normal = (df_all - df_numpy_mean) / df_numpy_std
df_all_normal_tensor = torch.Tensor(df_all_normal)
for i in range(n, len(df_all)):
    x = df_all_normal_tensor[i - n:i]
    x = torch.unsqueeze(torch.unsqueeze(x, dim=0), dim=0)
    y = rnn(x)
    if i < test_index:
        generate_data_train.append(torch.squeeze(y).detach().numpy() * df_numpy_std + df_nu
    else:
        generate_data_test.append(torch.squeeze(y).detach().numpy() * df_numpy_std + df_nu
plt.plot(df_index[n:train_end], generate_data_train, label='generate_train')
plt.plot(df_index[train_end:], generate_data_test, label='generate_test')
plt.legend()
plt.show()

plt.plot(df_index[train_end:], generate_data_test, label='predit')
plt.plot(df_index[-90:], df_all[-90:], label='test_data')
plt.legend()
plt.show()


def get_data_n_days(series, n):
    df = pd.DataFrame()
    for i in range(n):
        df['c%d' % i] = series.tolist()[i:-(n - i)]
    df['last'] = series.tolist()[n:]
    return df

def readData(column, n,train_end):
    df = pd.read_csv("BCHAIN-MKPRU.csv", index_col=0)
    df_column = df[column].copy()
    target_df = get_data_n_days(df_column, n)
    return target_df, df_column, df.index.tolist()



df, df_all, df_index = readData('Value', 30,0)

df_all = np.array(df_all.tolist())
plt.plot(df_index, df_all, label='real-data')
plt.plot(df_index[:0], df_all[:0], label='train_data')
plt.plot(df_index[0:], df_all[0:], label='test_data')

df_numpy = np.array(df)

df_numpy_mean = np.mean(df_numpy)
df_numpy_std = np.std(df_numpy)

df_numpy = (df_numpy - df_numpy_mean) / df_numpy_std

df_tensor = torch.Tensor(df_numpy)

trainset = TrainSet(df_tensor)
trainloader = DataLoader(trainset, batch_size=10, shuffle=True)
```

```
rnn = RNN(n)
optimizer = torch.optim.Adam(rnn.parameters(), lr=LR)  # optimize all cnn parameters
loss_func = nn.MSELoss()

for step in range(EPOCH):
    for tx, ty in trainloader:
        output = rnn(torch.unsqueeze(tx, dim=0))
        loss = loss_func(torch.squeeze(output), ty)
        optimizer.zero_grad()  # clear gradients for this training step
        loss.backward()  # back propagation, compute gradients
        optimizer.step()
    print(step, loss)


df = pd.read_csv("BCHAIN-MKPRU.csv", index_col=0)
df.index=pd.date_range('9/11/2016','9/10/2021')


def get_data_n_days_predit(series, n):
    df = pd.DataFrame()
    for i in range(n):
        df['c%d' % i] = series.tolist()[i:-(n - i)]
    df['last'] = series.tolist()[n:]
    return df

def predit_one(df):
    df1=df.iloc[-30:,]
    x=get_data_n_days_predit(df1['Value'], 29)

    x_numpy = np.array(x)
    x_numpy_mean = np.mean(x_numpy)
    x_numpy_std = np.std(x_numpy)
    x_numpy = (x_numpy - x_numpy_mean) / x_numpy_std  #

    x_tensor = torch.Tensor(x_numpy)


    target_tensor=torch.zeros(1,x_tensor.size()[0],x_tensor.size()[1])
    target_tensor[0]=x_tensor

    value=float(rnn(target_tensor))
    value = value * x_numpy_std+x_numpy_mean

    currnt_data=df.index[-1]
    target_data=pd.date_range(df.index[-1],periods=2)[1]
    df.loc[target_data]=[value]

    return df

def predit_many(times):
    for i in range(times):
        predit_one(df)
```

```
predit_many(15)

plt.plot(df.index[-400:-9], df.iloc[-400:-9], label='real_data')
plt.plot(df.index[-10:], df.iloc[-10:], label='predit_data')
```