**Math6373 Final**

Ying-Yu Huang (yingyu010365@gmail.com)

**1. data information**

**1.1 data source**

The close price of Netflix stock from 01/02/2014 to 12/31/2017. Data source is Yahoo Finance. Totally 1007 non-NA cases are used. we want to predict the future stock price S(t+1) given the last 20 observed stock prices S(t), S(t-1), S(t-2), ..., S(t-19).

**1.2 data normalization**

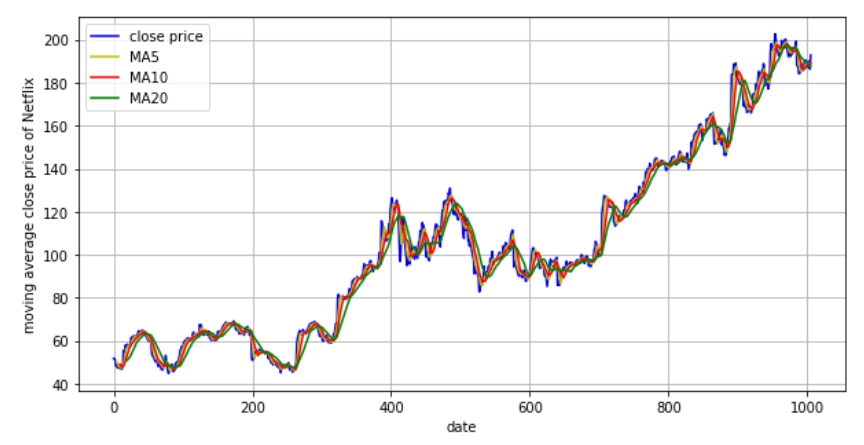
* Compute the moving mean of each stock with the formula below.

MA5(t) = [S(t-4) + S(t-3) + S(t-2) +S(t-1) + S(t) ] /5

MA10(t) = [S(t-9) + S(t-8) + ... + S(t) ] /10

MA20(t) = [S(t-19) + S(t-18) + ... + S(t) ] /20

* Then get a new data set with 987 cases, each case has 18 features.
* Plot the 4 curve of S, MA5, MA10, MA20



**1.3 training set and test set**

* Choose the close price of Netflix as the response vector Y values from 01/08/2014 to 12/31/2017, the dimension is (987, 1)
* Combine X and Y as DATA
* Select 900 rows of DATA randomly as the training set, the other 87 rows as test set and validation set, and form the Xtrain, Xtest, Ytrain and Ytest for the MLP predictor.

**1.4 The goal of this work**

* Develop a MLP to decrease the dimension of the input data and at the same time retain at least 95% of the variance of it, which can predict the close price of Netflix by the dimension-reduced data.

**2. AutoEncoder by MLP**

**2.1 the size of hidden layer**

By PCA, the smallest r<=18, which makes the ratio of explanation variance of the first r principal components to be 95%, is 1. So, the initial size of hidden layer is 1.

**2.2 create the architecture of MLP**

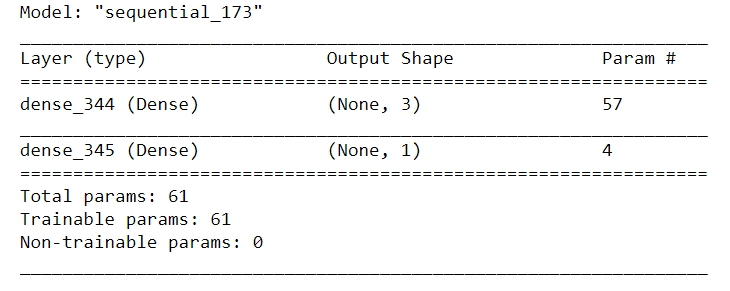
Layer 1: input layer, 20 neurons

Layer 2: hidden layer, 3 neurons, response function is RELU

Layer 3: output layer, 1 neurons, response function is RELU

Number of unknown parameters is 61

Number w of weights and thresholds in this MLP = 900/61 ≈ 15



**2.3 the option of MLP**

* **the learning algorithm**
  + The learning algorithm is Gradient Descent
  + Initial learning rate is 0.05, decay rate is =1e-7
* **the loss functions**
  + Mean square error
* **the initialization of W**
  + initial weight is random value of 'RandomNormal'
  + initial bias is 10 for layer 2 and 5 for layer 3
* **Batch learning**
  + Batch size = 16
  + Epoch = 2000

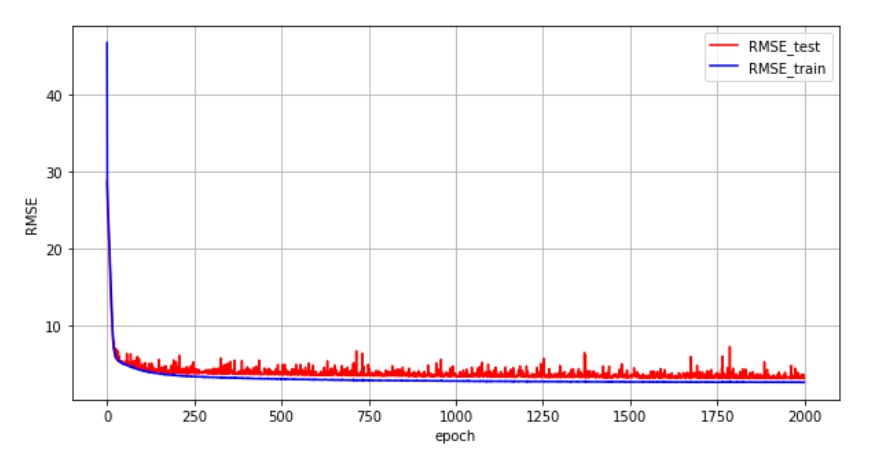
**2.4 Best model and performance of MLP**

**2.4.1 Model parameters tuning**

* Tuning learning rate in [0.1, 0.0001, 0.000001] 🡪 0.000001 is the best
* Tuning batch size in [16, 32, 64] 🡪 16 is the best
* Tuning h in [ 2, 3, 4] 🡪 3 is the best

**2.4.2 Plot the evolution of RMSE versus the number of batches**

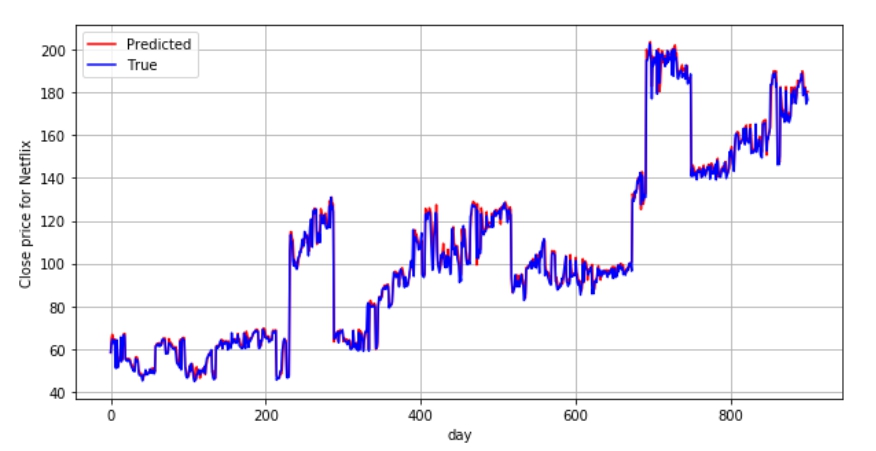
* The performance of best model is **RMSE-train = 8.048, RMSE-test = 10.238**
* The root of mean square error of training set and test set has a sharp dropping down at the first 10 epochs, then keep decreasing slowly.
* The root of mean square error of test set oscillates seriously at the beginning, then the amplitude shrinks along with the training procedure. The reason may be that the batch size is small and some of the fitting on batches have large bias.
* The unstableness of the performance of test set shows that this model may has poor robust and the accuracy of the prediction may have a large variability.



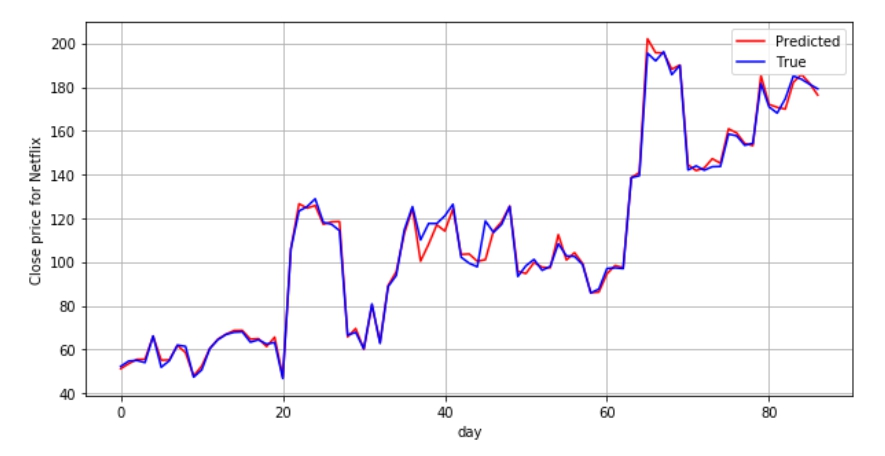
**2.4.3 plot the true price and the predicted price of Netflix vs the date**

* Predict the close price of Netflix on training set and test set by the best MLP model
* From the plot curves below, the predicted close price is close to the true one, but not good enough. Maybe the prediction MLP need to be tuned farther.

True price of Netflix and the predicted price vs the date on training set

****

True price of Netflix and the predicted price vs the date on test set

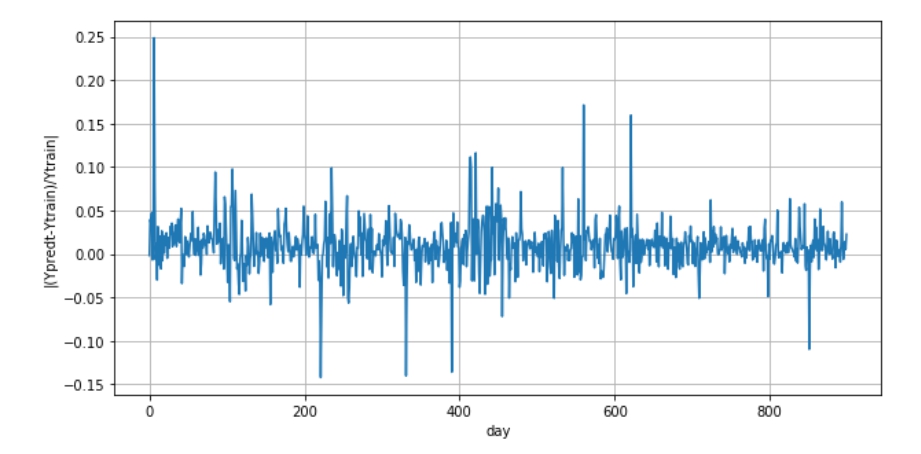
****

**2.4.4 Compute the Mean Relative Errors of Prediction MREP**

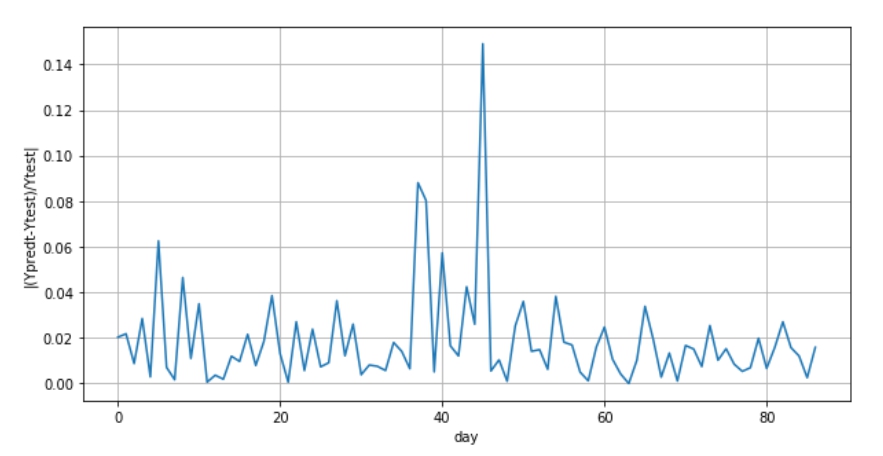
* The Mean Relative Errors of training set is 0.0194, of test set is 0.0190

The average errors between the true and predict values are approximately 0.019.

The difference between predy and true y on training set



The difference between predy and true y on test set

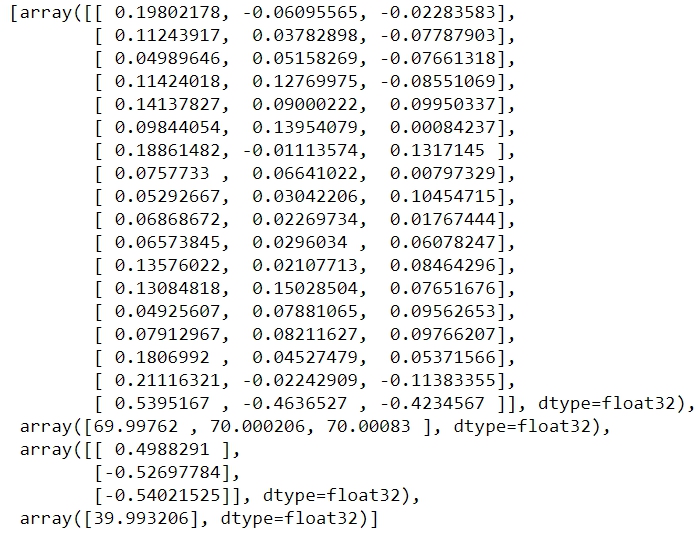


**2.5 extract the state of hidden layer on training set and test set**

* Display the mean activity of the neurons in hidden layer

329.25607, 112.40544, 72.77997

* Display all the weights W1 ... Wk linking the neurons NOD1 ... NODk to the output node.

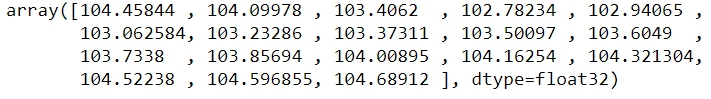


* Compute and display IMPj = Wj Yj = average impact of NODj on the prediction Zt

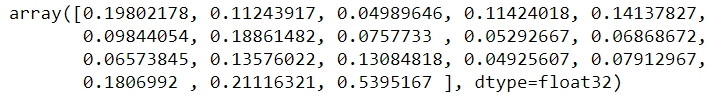
164.24251, -59.235176, -39.31685

The hidden neuron NOD\* with maximal impact on Zt is the first neuron

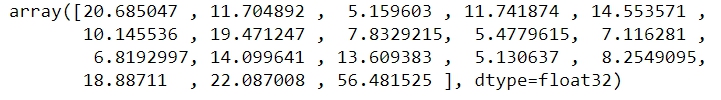
* Compute and display the mean activities X1 ... X18 of the 18 input neurons.



* Display all the weights U1 ... U18 linking the input nodes INP1 ... INP18 to the neuron NOD\*



* Compute Fs= Us Xs which is the average impact of input feature "s" on the key hidden neuron NOD\*



The 5 input features with the largest impact on NOD\* are 56.481525, 22.087008, 20.685047, 19.471247, 18.88711. These input features may be used to roughly predict the stock, especially the last one (56.481525) because it’s way bigger than the other four features.

So, we can say that the previous day of the stock has the most impact on it.

Code:

# HW data prepare

import pandas as pd

import numpy as np

data = (pd.read\_csv('NFLX.csv')[['Date','Close']])

data.isna().sum()

data.head()

data.shape

#For 20 ≤t≤N-1 , compute the following three moving averages of the time series S

#MA5(t) = [S(t-4) + S(t-3) + S(t-2) +S(t-1) + S(t) ] /5

#MA10(t) = [S(t-9) + S(t-8) + ... + S(t) ] /10

#MA20(t) = [S(t-19) + S(t-18) + ... + S(t) ] /20

data\_close=np.array(data['Close'])

MA5=[]

for i in range(1002):

MA5.append(sum(data\_close[i:i+5])/5)

MA10=[]

for i in range(997):

MA10.append(sum(data\_close[i:i+10])/10)

MA20=[]

for i in range(987):

MA20.append(sum(data\_close[i:i+20])/20)

#Plot the 4 curves S(t) , MA5(t), MA10(t), MA20(t), on the same graph

import matplotlib.pyplot as plt

MA5\_plot = np.append(np.repeat(np.nan, 4),MA5)

MA10\_plot = np.append(np.repeat(np.nan, 9),MA10)

MA20\_plot = np.append(np.repeat(np.nan, 19),MA20)

data\_t = data\_close[0:1006]

fig=plt.figure(figsize=(10,5))

ax=fig.add\_subplot(1,1,1)

ax.grid()

ax.plot(range(1006), data\_t, c='b')

ax.plot(range(1006),MA5\_plot,c='y')

ax.plot(range(1006),MA10\_plot,c='r')

ax.plot(range(1006),MA20\_plot,c='g')

ax.set\_xlabel('date')

ax.set\_ylabel('moving average close price of Netflix')

plt.legend(['close price','MA5','MA10','MA20'],loc='best')

plt.show

#3 Training and Test sets for an MLP predictor :

#On each day t ≥ 20 , the recent past of the series S will be defined as the 1x18 line vector

#Vt = [ MA5(t), MA10(t), MA20(t), S(t), S(t-1), S(t-2)..., S(t-13), S(t-14)]

V = np.empty(shape=[987,18])

for t in range(987):

V[t,0:3]= np.array([MA5[t+15],MA10[t+10],MA20[t]])

V[t,3:18]= np.array(data\_close[t+5:t+20])

V.shape

TARGt=data\_close[20:1007]

date = data.iloc[20:1007,0]

data\_all = np.column\_stack((V, TARGt, date)) # combine the date and true Y

#Split the set PredCases, with 90% cases in the training set PredTRAIN, and 10 % cases in the test set PredTEST

from sklearn.model\_selection import train\_test\_split

second\_size = 87/987

train, test = train\_test\_split(data\_all, test\_size=second\_size,random\_state=123)

train=pd.DataFrame(train)

test=pd.DataFrame(test)

train=train.sort\_values(by=[19]).iloc[:,0:19]

test=test.sort\_values(by=[19]).iloc[:,0:19]

train.to\_csv('train.csv')

test.to\_csv('test.csv')

Xtrain = np.matrix(train.iloc[:,0:18]).T

Xtrain.shape

G = np.corrcoef(Xtrain.astype(float))

#4 MLP predictor

#Implement PCA on the set of all input vectors Vt , with t= 20 ,21, ..., N .

#Determine the number k of principal components which preserves 95% of the variance (see HW3) and fix dim(K)= k.

from numpy import linalg as la

eigenv=la.eig(G)

eigenValues=eigenv[0]

eigenValues=np.sort(eigenValues)[::-1]

#Compute the smallest number "h95" of eigenvalues preserving 95% of the total sum of eigenvalues

eigen\_sum=0

Rj =[]

for i in range(18):

eigen\_sum = eigen\_sum + eigenValues[i]

percentage = eigen\_sum / 18

Rj.append(percentage)

smallest=0

for i in range(18):

if Rj[i] >= 0.99:

smallest = i

break

smallest

import tensorflow as tf

import numpy as np

import pandas as pd

# load data

train = pd.read\_csv("train.csv", dtype='float32').iloc[:,1:20]

test = pd.read\_csv("test.csv", dtype='float32').iloc[:,1:20]

train.shape, test.shape

Xtrain = np.array(train.iloc[:,0:18])

Ytrain = np.array(train.iloc[:,18])

Xtest = np.array(test.iloc[:,0:18])

Ytest = np.array(test.iloc[:,18])

Ytrain = np.reshape(Ytrain,(900,1))

Ytest = np.reshape(Ytest,(87,1))

import tensorflow.keras as keras

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

from tensorflow.keras.initializers import Constant

# constructing the autoencoder

# determine h through PCA on your own data

# try to find suitable initializers for your own data

h = 3

model = Sequential()

model.add(Dense(h, activation='relu', input\_dim=18, kernel\_initializer=keras.initializers.RandomNormal(mean=0.0, stddev=0.05, seed=1),bias\_initializer=Constant(value=70)))

model.add(Dense(1, activation='relu', kernel\_initializer=keras.initializers.RandomNormal(mean=0.0, stddev=0.05, seed=1),bias\_initializer=Constant(value=40)))

model.summary()

from tensorflow.keras import optimizers, losses

model.compile(optimizer=optimizers.SGD(learning\_rate=0.000001, decay=1e-7), loss='mean\_squared\_error')

from tensorflow.keras import callbacks

# the following callback to record losses after each batch

class MyHistory(callbacks.Callback):

def on\_train\_begin(self, logs={}):

self.MSEtrain = []

self.MSEtest = []

def on\_batch\_end(self, batch, logs={}):

self.MSEtrain.append(self.model.evaluate(Xtrain,Ytrain,verbose = 0))

self.MSEtest.append(self.model.evaluate(Xtest,Ytest,verbose = 0))

MyMonitor = MyHistory()

es = callbacks.EarlyStopping(monitor='val\_loss', mode='min', verbose=1, patience=100, restore\_best\_weights=True)

# For saving the best model choosen by keras.

filepath = 'BestModel.h5'

checkpointer = callbacks.ModelCheckpoint(filepath=filepath, monitor='val\_loss', save\_best\_only=True, mode='auto', save\_freq='epoch')

Monitor = model.fit(Xtrain, Ytrain, epochs=2000, batch\_size=16, callbacks = [MyMonitor, es,checkpointer], validation\_data = (Xtest, Ytest), verbose = 2)

from tensorflow.keras.models import load\_model

# Restore the best model and calculate confusion matrices.

bestmodel = load\_model('BestModel.h5')

Xtrani\_MSE = bestmodel.evaluate(Xtrain,Ytrain)

Xtest\_MSE = bestmodel.evaluate(Xtest,Ytest)

print(Xtrani\_MSE, Xtest\_MSE)

# plot loss and val\_loss to choose the best epoch

loss\_tr = Monitor.history['loss']

loss\_te = Monitor.history['val\_loss']

RMSE\_tr = np.sqrt(loss\_tr)

RMSE\_te = np.sqrt(loss\_te)

import matplotlib.pyplot as plt

fig=plt.figure(figsize=(10,5))

ax=fig.add\_subplot(1,1,1)

ax.grid()

ax.plot(range(len(loss\_tr)),RMSE\_te,c='r')

ax.plot(range(len(loss\_tr)),RMSE\_tr,c='b')

ax.set\_xlabel('epoch')

ax.set\_ylabel('RMSE')

#ax.set\_ylim(0,8)

plt.legend(['RMSE\_test','RMSE\_train'],loc='best')

plt.show

# plot the predicted Y and true Y on training set

Ypred = bestmodel.predict(Xtrain)

import matplotlib.pyplot as plt

fig=plt.figure(figsize=(10,5))

ax=fig.add\_subplot(1,1,1)

ax.grid()

ax.plot(range(len(Ypred)),Ypred,c='r')

ax.plot(range(len(Ypred)),Ytrain,c='b')

ax.set\_xlabel('day')

ax.set\_ylabel('Close price for Netflix')

#ax.set\_yticks([0.4,0.5,0.6,0.7,0.8,0.9,1.0,1.1])

plt.legend(['Predicted','True'],loc='best')

plt.show

# plot the predicted Y and true Y on test set

Ypredt = bestmodel.predict(Xtest)

fig=plt.figure(figsize=(10,5))

ax=fig.add\_subplot(1,1,1)

ax.grid()

ax.plot(range(len(Ypredt)),Ypredt,c='r')

ax.plot(range(len(Ypredt)),Ytest,c='b')

ax.set\_xlabel('day')

ax.set\_ylabel('Close price for Netflix')

#ax.set\_yticks([0.4,0.5,0.6,0.7,0.8,0.9,1.0,1.1])

plt.legend(['Predicted','True'],loc='best')

plt.show

np.average(np.absolute((Ypred-Ytrain))/Ypred)

np.average(np.absolute((Ypredt-Ytest))/Ypredt)

# plot the diffrence between predy and true y

diff = (np.array(Ypredt)-np.array(Ytest))/np.array(Ytest)

fig=plt.figure(figsize=(10,5))

ax=fig.add\_subplot(1,1,1)

ax.grid()

ax.plot(range(len(Ypredt)),np.abs(diff))

ax.set\_xlabel('day')

ax.set\_ylabel('|(Ypredt-Ytest)/Ytest|')

#ax.set\_yticks([0.4,0.5,0.6,0.7,0.8,0.9,1.0,1.1])

#plt.legend(['Predicted','True'],loc='best')

plt.show

diff1 = (np.array(Ypred)-np.array(Ytrain))/np.array(Ytrain)

fig=plt.figure(figsize=(10,5))

ax=fig.add\_subplot(1,1,1)

ax.grid()

ax.plot(range(len(Ypred)),diff1)

ax.set\_xlabel('day')

ax.set\_ylabel('|(Ypredt-Ytrain)/Ytrain|')

#ax.set\_yticks([0.4,0.5,0.6,0.7,0.8,0.9,1.0,1.1])

#plt.legend(['Predicted','True'],loc='best')

plt.show

# save the prediction result to excel

np.savetxt('Ypredtr.csv',Ypred)

np.savetxt('Ypredt.csv',Ypredt)

np.savetxt('Ytr.csv',Ytrain)

np.savetxt('Yt.csv',Ytest)

# display the mean activity of the neurons in hidden layer

Htrain = bestmodel.layers[0](Xtrain).numpy()

Htest = bestmodel.layers[0](Xtest).numpy()

np.savetxt('Htrain.csv',Htrain)

np.savetxt('Htest.csv',Htest)

MHtrain = np.mean(Htrain,axis=0)

MHtrain

W = bestmodel.get\_weights()

W

# compute and display IMPj = Wj Yj = average impact of NODj on the prediction Zt

IMP = MHtrain \* W\_layer3

IMP

# Identify the hidden neuron NOD\* with maximal impact on Zt is the first neuron

# Compute and display the mean activities X1 ... X18 of the 18 input neurons.

MXtrain = np.mean(Xtrain,axis=0)

MXtrain

# Display all the weights U1 ... U18 linking the input nodes INP1 ... INP18 to the neuron NOD\*

W\_layer2 = W[0]

W\_layer2

W\_layer2\_NOD = W\_layer2[:,0]

W\_layer2\_NOD

# compute Fs= Us Xs which is the average impact of input feature "s" on the key hidden neuron NOD\*

Fs = MXtrain \* W\_layer2\_NOD

Fs

# Identify the 5 input features with the largest impact on NOD\*.