МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ “ЛЬВІВСЬКА ПОЛІТЕХНІКА”

ІНСТИТУТ КОМП’ЮТЕРНИХ НАУК ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

кафедра систем штучного інтелекту



**ЗВІТ**

про виконання лабораторної роботи №1

з курсу «Проектування систем глибинного навчання»

на тему «ПрогнозуваннянаосновіRNN LSTM GRU»

Виконав:

*ст. групи КНСШ-12*

*Карпінський Р.М*

Перевірив:

*Пелешко Д.Д*

ЛЬВІВ – 2021

**Мета:** Виконати задані завдання за темою ПрогнозуваннянаосновіRNN LSTM GRU.

**Завдання**

* Вивчити структуру LSTM та GRU та принцип побудови мережі за допомогоюKeras.
* Розібратися з Case1 – прогнозування сигналів. Згенерувати свій сигналнабазі sinта cos, та їх комбінації і спробувати спрогнозувати, оцінити точність.
* Розібрати з задачею прогнозування часових рядів фінансової природизадопомогою LSTM та GRU.
* Досягти кращої точності ніж наведено в прикладі, за рахунок переналаштуваннямережі.

**Виконання роботи:**

import numpy as np

import pandas as pd

import math

import sklearn

import sklearn.preprocessing

import datetime

import os

import matplotlib.pyplot as plt

%tensorflow\_version 1.x

import tensorflow as tf

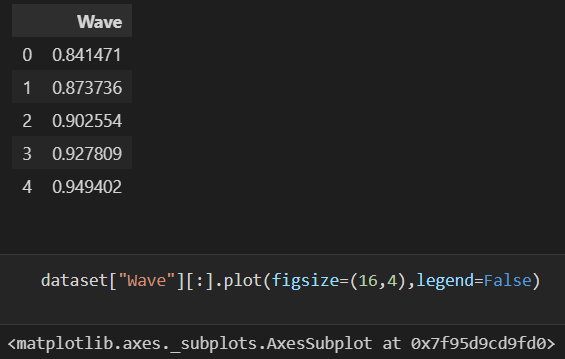
from keras.layers.recurrent import LSTM

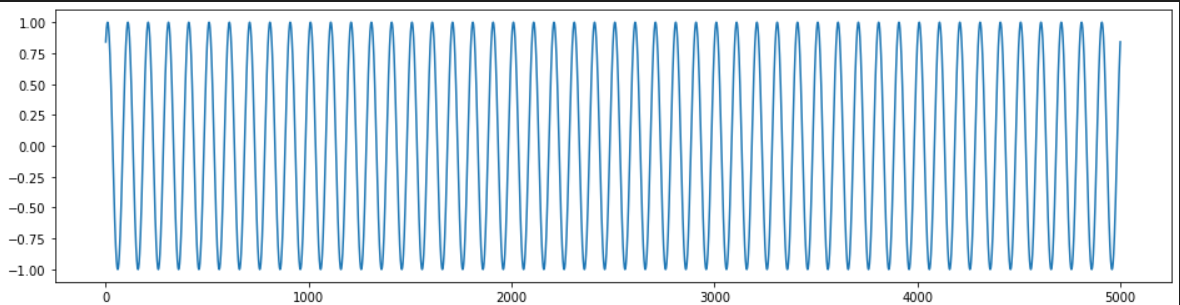
from keras.models import Sequential

from keras.layers import Dense, Dropout

dataset = pd.read\_csv('Sin Wave Data Generator.csv')

dataset.head(5)





def normalise\_windows(window\_data):

    # A support function to normalize a dataset

    normalised\_data = []

    for window in window\_data:

        normalised\_window = [((float(p) / float(window[0])) - 1) for p in window]

        normalised\_data.append(normalised\_window)

    return normalised\_data

def load\_data(dataset, column, seq\_len, normalise\_window):

    # A support function to help prepare datasets for an RNN/LSTM/GRU

    data = dataset.loc[:,column]

    sequence\_length = seq\_len + 1

    result = []

    for index in range(len(data) - sequence\_length):

        result.append(data[index: index + sequence\_length])

    if normalise\_window:

        result = normalise\_windows(result)

    result = np.array(result)

    #Last 10% is used for validation test, first 90% for training

    row = round(0.9 \* result.shape[0])

    train = result[:int(row), :]

    np.random.shuffle(train)

    x\_train = train[:, :-1]

    y\_train = train[:, -1]

    x\_test = result[int(row):, :-1]

    y\_test = result[int(row):, -1]

    x\_train = np.reshape(x\_train, (x\_train.shape[0], x\_train.shape[1], 1))

    x\_test = np.reshape(x\_test, (x\_test.shape[0], x\_test.shape[1], 1))

    return [x\_train, y\_train, x\_test, y\_test]

Enrol\_window = 100

feature\_train, label\_train, feature\_test, label\_test = load\_data(dataset, 'Wave', Enrol\_window, False)

print ('Datasets generated')

model = Sequential()

model.add(LSTM(50, return\_sequences=True, input\_shape=(feature\_train.shape[1],1)))

model.add(Dropout(0.2))

model.add(LSTM(100, return\_sequences=False))

model.add(Dropout(0.2))

model.add(Dense(1, activation = "linear"))

model.compile(loss='mse', optimizer='adam')

print ('model compiled')

print (model.summary())

Total params: 70,901 Trainable params: 70,901 Non-trainable params: 0

model = Sequential()

model.add(LSTM(40, return\_sequences=True, input\_shape=(feature\_train.shape[1],1)))

model.add(Dropout(0.2))

model.add(LSTM(70, return\_sequences=False))

model.add(Dropout(0.2))

model.add(Dense(1, activation = "linear"))

model.compile(loss='mse', optimizer='adam')

print ('model compiled')

print (model.summary())

history\_adv =  model.fit(feature\_train, label\_train, batch\_size=256, epochs=10, validation\_data = (feature\_test, label\_test))

val\_loss = history.history['val\_loss']

val\_loss\_adv = history\_adv.history['val\_loss']

val\_loss = val\_loss[1:]

val\_loss\_adv = val\_loss\_adv[1:]

epochs\_range = range(9)

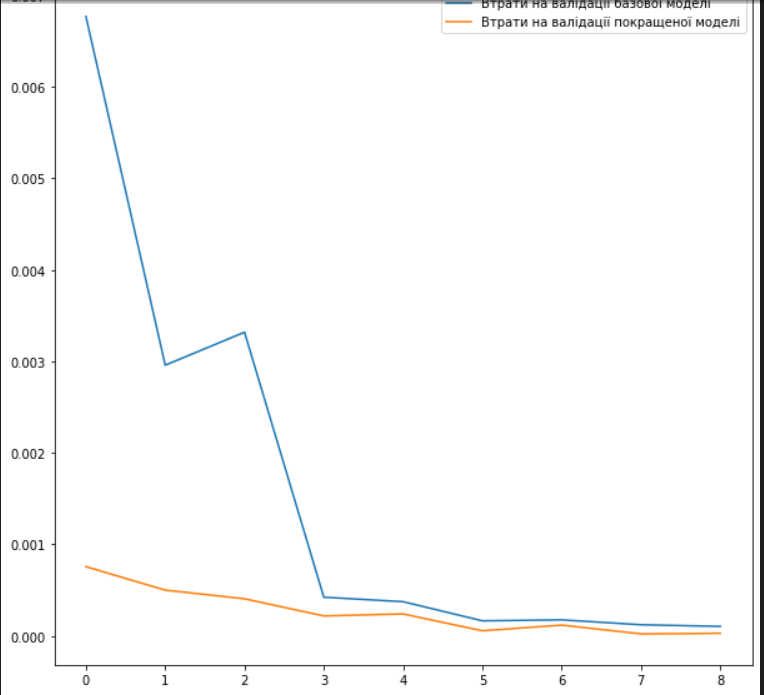
plt.figure(figsize=(10,10))

plt.plot(epochs\_range, val\_loss, label='Втрати на валідації базової моделі')

plt.plot(epochs\_range, val\_loss\_adv, label='Втрати на валідації покращеної моделі')

plt.legend(loc='upper right')

plt.title(f'Графіки втрат покращеної моделі')



from numpy import newaxis

def predict\_sequence\_full(model, data, window\_size):

    #Shift the window by 1 new prediction each time, re-run predictions on new window

    curr\_frame = data[0]

    predicted = []

    for i in range(len(data)):

        predicted.append(model.predict(curr\_frame[newaxis,:,:])[0,0])

        curr\_frame = curr\_frame[1:]

        curr\_frame = np.insert(curr\_frame, [window\_size-1], predicted[-1], axis=0)

    return predicted

def plot\_results(predicted\_data, true\_data):

    fig = plt.figure(facecolor='white')

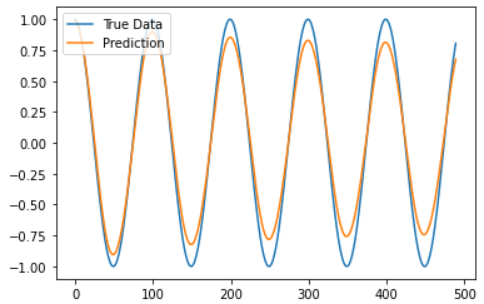
    ax = fig.add\_subplot(111)

    ax.plot(true\_data, label='True Data')

    plt.plot(predicted\_data, label='Prediction')

    plt.legend(loc='upper left')

    plt.show()



**\*\*Case 2. NY Stock Price Prediction RNN LSTM GRU\*\***

import numpy as np

import pandas as pd

from sklearn.preprocessing import MinMaxScaler

import matplotlib.pyplot as plt

# tensorflow 1

import tensorflow.compat.v1 as tf1

tf1.disable\_v2\_behavior()

# tensorflow 2

from tensorflow import keras

from tensorflow.keras import layers

# split data in 80%/10%/10% train/validation/test sets

valid\_set\_size\_percentage = 10

test\_set\_size\_percentage = 10

# split data in 80%/10%/10% train/validation/test sets

valid\_set\_size\_percentage = 10

test\_set\_size\_percentage = 10

#display parent directory and working directory

print(os.path.dirname(os.getcwd())+':', os.listdir(os.path.dirname(os.getcwd())))

print(os.getcwd()+':', os.listdir(os.getcwd()))

# import all stock prices

df = pd.read\_csv("prices-split-adjusted.csv", index\_col = 0)

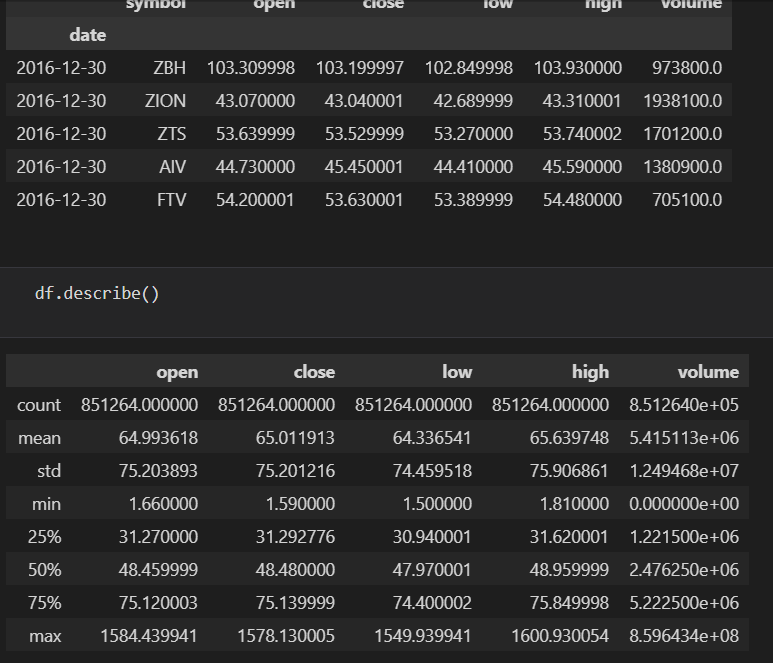
df.info()

df.head()

# number of different stocks

print('\nnumber of different stocks: ', len(list(set(df.symbol))))

print(list(set(df.symbol))[:10])



plt.figure(figsize=(15, 5));

plt.subplot(1,2,1);

plt.plot(df[df.symbol == 'EQIX'].open.values, color='red', label='open')

plt.plot(df[df.symbol == 'EQIX'].close.values, color='green', label='close')

plt.plot(df[df.symbol == 'EQIX'].low.values, color='blue', label='low')

plt.plot(df[df.symbol == 'EQIX'].high.values, color='black', label='high')

plt.title('stock price')

plt.xlabel('time [days]')

plt.ylabel('price')

plt.legend(loc='best')

#plt.show()

plt.subplot(1,2,2);

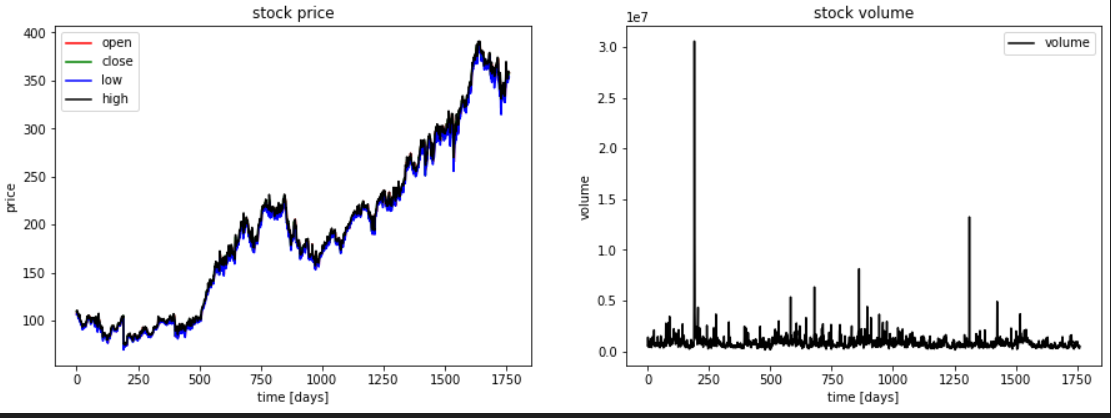
plt.plot(df[df.symbol == 'EQIX'].volume.values, color='black', label='volume')

plt.title('stock volume')

plt.xlabel('time [days]')

plt.ylabel('volume')

plt.legend(loc='best');



# function for min-max normalization of stock

def normalize\_data(df):

    min\_max\_scaler = sklearn.preprocessing.MinMaxScaler()

    df['open'] = min\_max\_scaler.fit\_transform(df.open.values.reshape(-1,1))

    df['high'] = min\_max\_scaler.fit\_transform(df.high.values.reshape(-1,1))

    df['low'] = min\_max\_scaler.fit\_transform(df.low.values.reshape(-1,1))

    df['close'] = min\_max\_scaler.fit\_transform(df['close'].values.reshape(-1,1))

    return df

# function to create train, validation, test data given stock data and sequence length

def load\_data(stock, seq\_len):

    data\_raw = stock.to\_numpy() # convert to numpy array

    data = []

    # create all possible sequences of length seq\_len

    for index in range(len(data\_raw) - seq\_len):

        data.append(data\_raw[index: index + seq\_len])

    data = np.array(data);

    valid\_set\_size = int(np.round(valid\_set\_size\_percentage/100\*data.shape[0]));

    test\_set\_size = int(np.round(test\_set\_size\_percentage/100\*data.shape[0]));

    train\_set\_size = data.shape[0] - (valid\_set\_size + test\_set\_size);

    x\_train = data[:train\_set\_size,:-1,:]

    y\_train = data[:train\_set\_size,-1,:]

    x\_valid = data[train\_set\_size:train\_set\_size+valid\_set\_size,:-1,:]

    y\_valid = data[train\_set\_size:train\_set\_size+valid\_set\_size,-1,:]

    x\_test = data[train\_set\_size+valid\_set\_size:,:-1,:]

    y\_test = data[train\_set\_size+valid\_set\_size:,-1,:]

    return [x\_train, y\_train, x\_valid, y\_valid, x\_test, y\_test]

# choose one stock

df\_stock = df[df.symbol == 'EQIX'].copy()

df\_stock.drop(['symbol'],1,inplace=True)

df\_stock.drop(['volume'],1,inplace=True)

cols = list(df\_stock.columns.values)

print('df\_stock.columns.values = ', cols)

# normalize stock

df\_stock\_norm = df\_stock.copy()

df\_stock\_norm = normalize\_data(df\_stock\_norm)

# create train, test data

seq\_len = 20 # choose sequence length

x\_train, y\_train, x\_valid, y\_valid, x\_test, y\_test = load\_data(df\_stock\_norm, seq\_len)

print('x\_train.shape = ',x\_train.shape)

print('y\_train.shape = ', y\_train.shape)

print('x\_valid.shape = ',x\_valid.shape)

print('y\_valid.shape = ', y\_valid.shape)

print('x\_test.shape = ', x\_test.shape)

print('y\_test.shape = ',y\_test.shape)

plt.figure(figsize=(15, 5))

plt.plot(df\_stock.open.values, color='red', label='open')

plt.plot(df\_stock.close.values, color='green', label='low')

plt.plot(df\_stock.low.values, color='blue', label='low')

plt.plot(df\_stock.high.values, color='black', label='high')

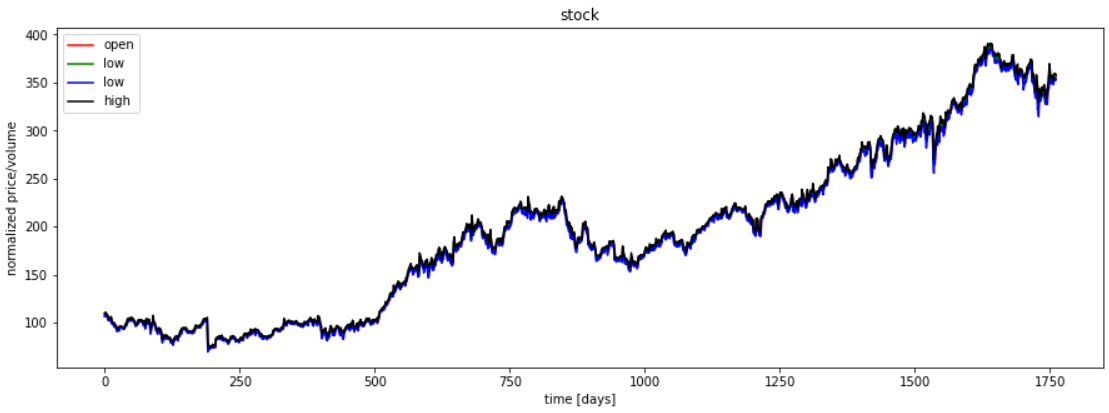
plt.title('stock')

plt.xlabel('time [days]')

plt.ylabel('normalized price/volume')

plt.legend(loc='best')

plt.show()



## Basic Cell RNN in tensorflow

index\_in\_epoch = 0;

perm\_array  = np.arange(x\_train.shape[0])

np.random.shuffle(perm\_array)

# function to get the next batch

def get\_next\_batch(batch\_size):

    global index\_in\_epoch, x\_train, perm\_array

    start = index\_in\_epoch

    index\_in\_epoch += batch\_size

    if index\_in\_epoch > x\_train.shape[0]:

        np.random.shuffle(perm\_array) # shuffle permutation array

        start = 0 # start next epoch

        index\_in\_epoch = batch\_size

    end = index\_in\_epoch

    return x\_train[perm\_array[start:end]], y\_train[perm\_array[start:end]]

# parameters

n\_steps = seq\_len-1

n\_inputs = 4

n\_neurons = 200

n\_outputs = 4

n\_layers = 2

learning\_rate = 0.001

batch\_size = 50

n\_epochs = 100

train\_set\_size = x\_train.shape[0]

test\_set\_size = x\_test.shape[0]

tf.reset\_default\_graph()

X = tf.placeholder(tf.float32, [None, n\_steps, n\_inputs])

y = tf.placeholder(tf.float32, [None, n\_outputs])

# use Basic RNN Cell

layers = [tf.contrib.rnn.BasicRNNCell(num\_units=n\_neurons, activation=tf.nn.elu)

          for layer in range(n\_layers)]

multi\_layer\_cell = tf.contrib.rnn.MultiRNNCell(layers)

rnn\_outputs, states = tf.nn.dynamic\_rnn(multi\_layer\_cell, X, dtype=tf.float32)

stacked\_rnn\_outputs = tf.reshape(rnn\_outputs, [-1, n\_neurons])

stacked\_outputs = tf.layers.dense(stacked\_rnn\_outputs, n\_outputs)

outputs = tf.reshape(stacked\_outputs, [-1, n\_steps, n\_outputs])

outputs = outputs[:,n\_steps-1,:] # keep only last output of sequence

loss = tf.reduce\_mean(tf.square(outputs - y)) # loss function = mean squared error

optimizer = tf.train.AdamOptimizer(learning\_rate=learning\_rate)

training\_op = optimizer.minimize(loss)

list\_mse\_valid = []

# run graph

with tf.Session() as sess:

    sess.run(tf.global\_variables\_initializer())

    for iteration in range(int(n\_epochs\*train\_set\_size/batch\_size)):

        x\_batch, y\_batch = get\_next\_batch(batch\_size) # fetch the next training batch

        sess.run(training\_op, feed\_dict={X: x\_batch, y: y\_batch})

        if iteration % int(5\*train\_set\_size/batch\_size) == 0:

            mse\_train = loss.eval(feed\_dict={X: x\_train, y: y\_train})

            mse\_valid = loss.eval(feed\_dict={X: x\_valid, y: y\_valid})

            list\_mse\_valid.append(mse\_valid)

            print('%.2f epochs: MSE train/valid = %.6f/%.6f'%(

                iteration\*batch\_size/train\_set\_size, mse\_train, mse\_valid))

    y\_train\_pred = sess.run(outputs, feed\_dict={X: x\_train})

    y\_valid\_pred = sess.run(outputs, feed\_dict={X: x\_valid})

    y\_test\_pred = sess.run(outputs, feed\_dict={X: x\_test})

ft = 0 # 0 = open, 1 = close, 2 = highest, 3 = lowest

## show predictions

plt.figure(figsize=(15, 5));

plt.subplot(1,2,1);

plt.plot(np.arange(y\_train.shape[0]), y\_train[:,ft], color='blue', label='train target')

plt.plot(np.arange(y\_train.shape[0], y\_train.shape[0]+y\_valid.shape[0]), y\_valid[:,ft],

         color='gray', label='valid target')

plt.plot(np.arange(y\_train.shape[0]+y\_valid.shape[0],

                   y\_train.shape[0]+y\_test.shape[0]+y\_test.shape[0]),

         y\_test[:,ft], color='black', label='test target')

plt.plot(np.arange(y\_train\_pred.shape[0]),y\_train\_pred[:,ft], color='red',

         label='train prediction')

plt.plot(np.arange(y\_train\_pred.shape[0], y\_train\_pred.shape[0]+y\_valid\_pred.shape[0]),

         y\_valid\_pred[:,ft], color='orange', label='valid prediction')

plt.plot(np.arange(y\_train\_pred.shape[0]+y\_valid\_pred.shape[0],

                   y\_train\_pred.shape[0]+y\_valid\_pred.shape[0]+y\_test\_pred.shape[0]),

         y\_test\_pred[:,ft], color='green', label='test prediction')

plt.title('past and future stock prices')

plt.xlabel('time [days]')

plt.ylabel('normalized price')

plt.legend(loc='best');

plt.subplot(1,2,2);

plt.plot(np.arange(y\_train.shape[0], y\_train.shape[0]+y\_test.shape[0]),

         y\_test[:,ft], color='black', label='test target')

plt.plot(np.arange(y\_train\_pred.shape[0], y\_train\_pred.shape[0]+y\_test\_pred.shape[0]),

         y\_test\_pred[:,ft], color='green', label='test prediction')

plt.title('future stock prices')

plt.xlabel('time [days]')

plt.ylabel('normalized price')

plt.legend(loc='best');

corr\_price\_development\_train = np.sum(np.equal(np.sign(y\_train[:,1]-y\_train[:,0]),

            np.sign(y\_train\_pred[:,1]-y\_train\_pred[:,0])).astype(int)) / y\_train.shape[0]

corr\_price\_development\_valid = np.sum(np.equal(np.sign(y\_valid[:,1]-y\_valid[:,0]),

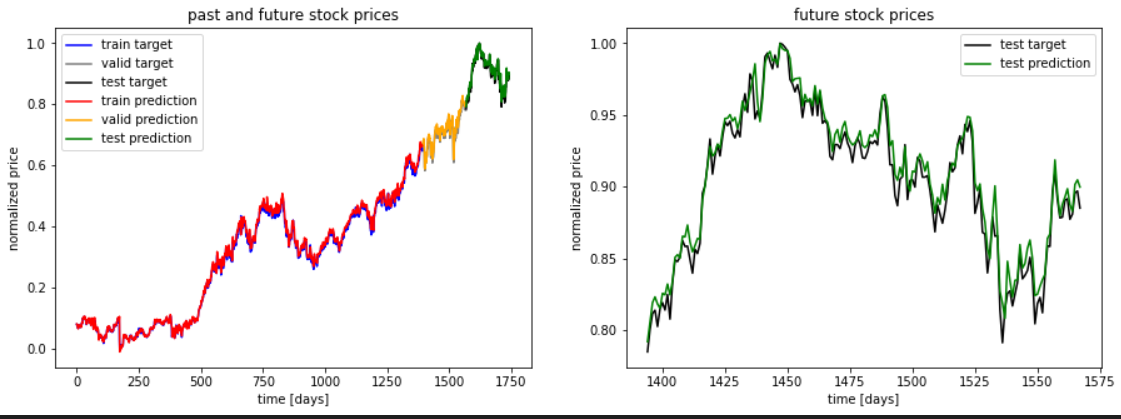
            np.sign(y\_valid\_pred[:,1]-y\_valid\_pred[:,0])).astype(int)) / y\_valid.shape[0]

corr\_price\_development\_test = np.sum(np.equal(np.sign(y\_test[:,1]-y\_test[:,0]),

            np.sign(y\_test\_pred[:,1]-y\_test\_pred[:,0])).astype(int)) / y\_test.shape[0]

print('correct sign prediction for close - open price for train/valid/test: %.2f/%.2f/%.2f'%(

    corr\_price\_development\_train, corr\_price\_development\_valid, corr\_price\_development\_test))



## Basic Cell RNN in tensorflow

index\_in\_epoch = 0;

perm\_array  = np.arange(x\_train.shape[0])

np.random.shuffle(perm\_array)

# function to get the next batch

def get\_next\_batch(batch\_size):

    global index\_in\_epoch, x\_train, perm\_array

    start = index\_in\_epoch

    index\_in\_epoch += batch\_size

    if index\_in\_epoch > x\_train.shape[0]:

        np.random.shuffle(perm\_array) # shuffle permutation array

        start = 0 # start next epoch

        index\_in\_epoch = batch\_size

    end = index\_in\_epoch

    return x\_train[perm\_array[start:end]], y\_train[perm\_array[start:end]]

# parameters

n\_steps = seq\_len-1

n\_inputs = 4

n\_neurons = 180

n\_outputs = 4

n\_layers = 2

dropout\_rate = 0

learning\_rate = 0.001

batch\_size = 50

n\_epochs = 100

train\_set\_size = x\_train.shape[0]

test\_set\_size = x\_test.shape[0]

tf.reset\_default\_graph()

X = tf.placeholder(tf.float32, [None, n\_steps, n\_inputs])

y = tf.placeholder(tf.float32, [None, n\_outputs])

# use Basic RNN Cell

layers = []

for i in range(n\_layers):

    layers.append(tf.contrib.rnn.DropoutWrapper(tf.contrib.rnn.BasicRNNCell(num\_units=n\_neurons, activation=tf.nn.elu),

                                              output\_keep\_prob=1-dropout\_rate))

multi\_layer\_cell = tf.contrib.rnn.MultiRNNCell(layers)

rnn\_outputs, states = tf.nn.dynamic\_rnn(multi\_layer\_cell, X, dtype=tf.float32)

stacked\_rnn\_outputs = tf.reshape(rnn\_outputs, [-1, n\_neurons])

stacked\_outputs = tf.layers.dense(stacked\_rnn\_outputs, n\_outputs)

outputs = tf.reshape(stacked\_outputs, [-1, n\_steps, n\_outputs])

outputs = outputs[:,n\_steps-1,:] # keep only last output of sequence

loss = tf.reduce\_mean(tf.square(outputs - y)) # loss function = mean squared error

optimizer = tf.train.AdamOptimizer(learning\_rate=learning\_rate)

training\_op = optimizer.minimize(loss)

list\_mse\_valid\_adv = []

# run graph

with tf.Session() as sess:

    sess.run(tf.global\_variables\_initializer())

    for iteration in range(int(n\_epochs\*train\_set\_size/batch\_size)):

        x\_batch, y\_batch = get\_next\_batch(batch\_size) # fetch the next training batch

        sess.run(training\_op, feed\_dict={X: x\_batch, y: y\_batch})

        if iteration % int(5\*train\_set\_size/batch\_size) == 0:

            mse\_train = loss.eval(feed\_dict={X: x\_train, y: y\_train})

            mse\_valid = loss.eval(feed\_dict={X: x\_valid, y: y\_valid})

            list\_mse\_valid\_adv.append(mse\_valid)

            print('%.2f epochs: MSE train/valid = %.6f/%.6f'%(

                iteration\*batch\_size/train\_set\_size, mse\_train, mse\_valid))

    y\_train\_pred = sess.run(outputs, feed\_dict={X: x\_train})

    y\_valid\_pred = sess.run(outputs, feed\_dict={X: x\_valid})

    y\_test\_pred = sess.run(outputs, feed\_dict={X: x\_test})

ft = 0 # 0 = open, 1 = close, 2 = highest, 3 = lowest

## show predictions

plt.figure(figsize=(15, 5));

plt.subplot(1,2,1);

plt.plot(np.arange(y\_train.shape[0]), y\_train[:,ft], color='blue', label='train target')

plt.plot(np.arange(y\_train.shape[0], y\_train.shape[0]+y\_valid.shape[0]), y\_valid[:,ft],

         color='gray', label='valid target')

plt.plot(np.arange(y\_train.shape[0]+y\_valid.shape[0],

                   y\_train.shape[0]+y\_test.shape[0]+y\_test.shape[0]),

         y\_test[:,ft], color='black', label='test target')

plt.plot(np.arange(y\_train\_pred.shape[0]),y\_train\_pred[:,ft], color='red',

         label='train prediction')

plt.plot(np.arange(y\_train\_pred.shape[0], y\_train\_pred.shape[0]+y\_valid\_pred.shape[0]),

         y\_valid\_pred[:,ft], color='orange', label='valid prediction')

plt.plot(np.arange(y\_train\_pred.shape[0]+y\_valid\_pred.shape[0],

                   y\_train\_pred.shape[0]+y\_valid\_pred.shape[0]+y\_test\_pred.shape[0]),

         y\_test\_pred[:,ft], color='green', label='test prediction')

plt.title('past and future stock prices')

plt.xlabel('time [days]')

plt.ylabel('normalized price')

plt.legend(loc='best');

plt.subplot(1,2,2);

plt.plot(np.arange(y\_train.shape[0], y\_train.shape[0]+y\_test.shape[0]),

         y\_test[:,ft], color='black', label='test target')

plt.plot(np.arange(y\_train\_pred.shape[0], y\_train\_pred.shape[0]+y\_test\_pred.shape[0]),

         y\_test\_pred[:,ft], color='green', label='test prediction')

plt.title('future stock prices')

plt.xlabel('time [days]')

plt.ylabel('normalized price')

plt.legend(loc='best');

corr\_price\_development\_train = np.sum(np.equal(np.sign(y\_train[:,1]-y\_train[:,0]),

            np.sign(y\_train\_pred[:,1]-y\_train\_pred[:,0])).astype(int)) / y\_train.shape[0]

corr\_price\_development\_valid = np.sum(np.equal(np.sign(y\_valid[:,1]-y\_valid[:,0]),

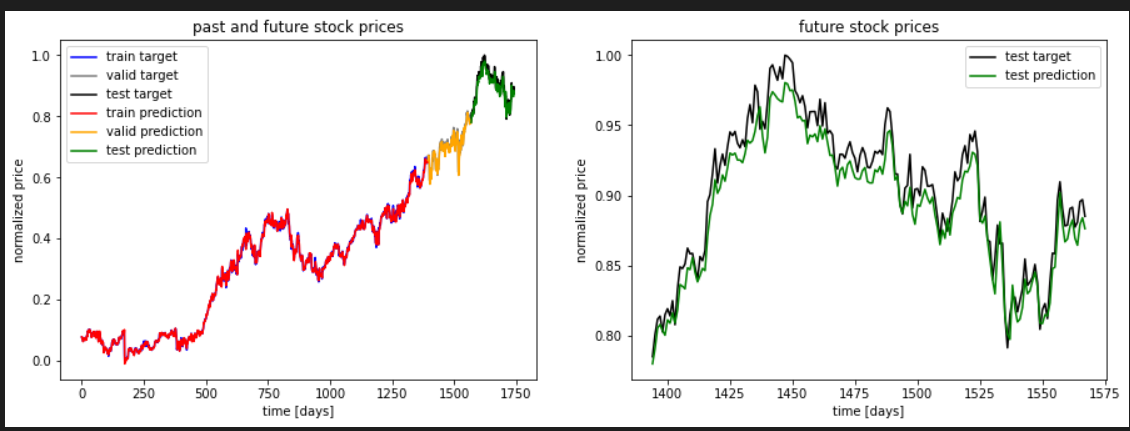
            np.sign(y\_valid\_pred[:,1]-y\_valid\_pred[:,0])).astype(int)) / y\_valid.shape[0]

corr\_price\_development\_test = np.sum(np.equal(np.sign(y\_test[:,1]-y\_test[:,0]),

            np.sign(y\_test\_pred[:,1]-y\_test\_pred[:,0])).astype(int)) / y\_test.shape[0]

print('correct sign prediction for close - open price for train/valid/test: %.2f/%.2f/%.2f'%(

    corr\_price\_development\_train, corr\_price\_development\_valid, corr\_price\_development\_test))



val\_loss = list\_mse\_valid

val\_loss\_adv = list\_mse\_valid\_adv

val\_loss = val\_loss[1:]

val\_loss\_adv = val\_loss\_adv[1:]

epochs\_range = range(20)

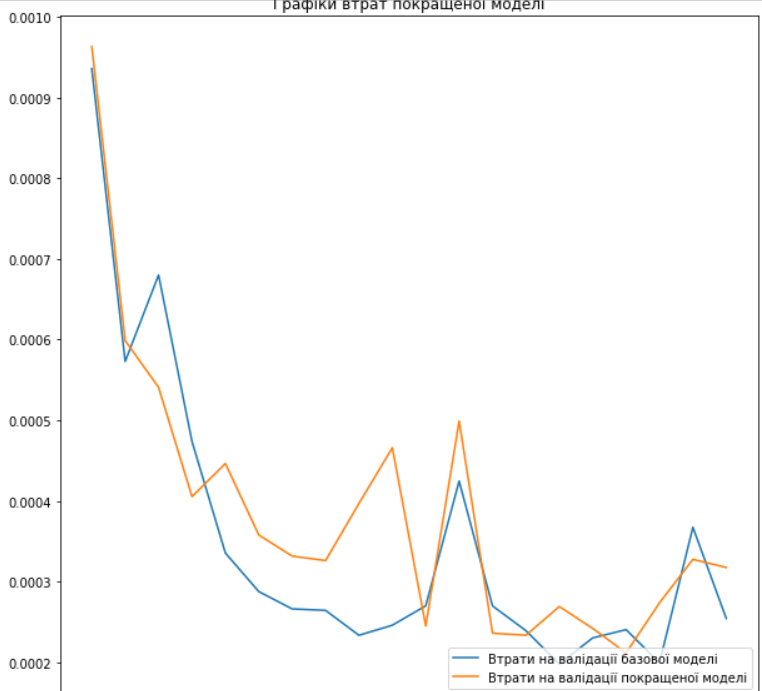
plt.figure(figsize=(10,10))

plt.plot(epochs\_range, val\_loss, label='Втрати на валідації базової моделі')

plt.plot(epochs\_range, val\_loss\_adv, label='Втрати на валідації покращеної моделі')

plt.legend(loc='lower right')

plt.title(f'Графіки втрат покращеної моделі')



**Висновок:** На даній лабораторній роботі, виконав поставлені завдання а саме Вивчити структуру LSTM та GRU та принцип побудови мережі, розібрався з Case1 – прогнозування сигналів, розібрав з задачею прогнозування часових рядів, досягнув кращої точності ніж наведено в прикладі.