ДОДАТОК А

Графічний матеріал кваліфікаційної роботи

Харківський національний університет радіоелектроніки

Кафедра ЕОМ

Метод прогнозування курсу валют за допомогою нейронної мережі

Кваліфікаційна робота
Другий(магістрський) рівень

Автор:
Корягіна П.О.
студ. гр. СПм-20-1

Керівник:
Проф. Кучук Н.Г.

Мета і задачі роботи

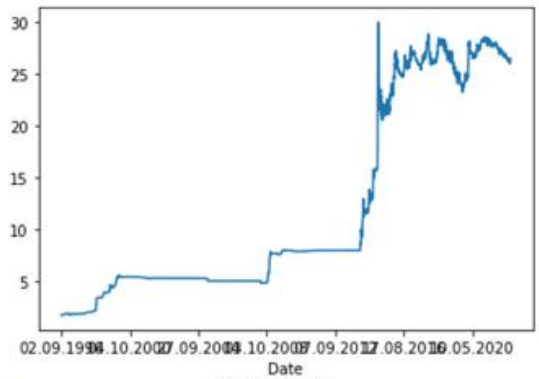
- Мета: описати метод прогнозування курсу валют за допомогою нейронної мережі
- Задачі:
  - Аналіз основних інструментів
  - Опис сучасного базового підходу прогнозування курсів валют
  - Створення нейронних мереж
  - Створення ансамблю нейронних мереж
  - Аналіз точності

Курс валют

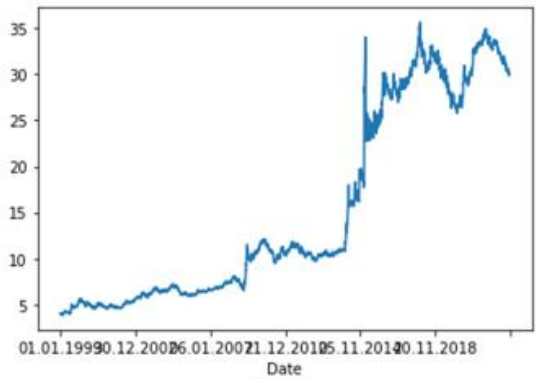
- Функції :
  - Подолання економічної обмеженості
  - Порівняння вартісних показників
  - Реалізація інтернаціональної вартості товару
  - Міжнародна торгівля
- Фактори:
  - темпи зростання продуктивності праці, темпи зростання ВНП країни, місце і роль країни у світовій торгівлі;
  - темп інфляції;
  - різниця відсоткових ставок у різних країнах;
  - використання валюти у міжнародних відносинах;
  - міра довіри до валюти на національному і світовому ринках;
  - стан платіжного балансу;

3

Курс валют.



Долар США



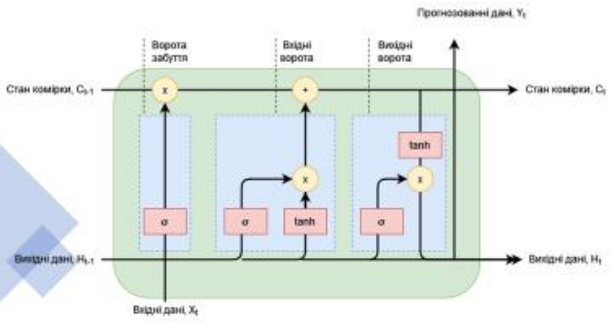
Євро

4

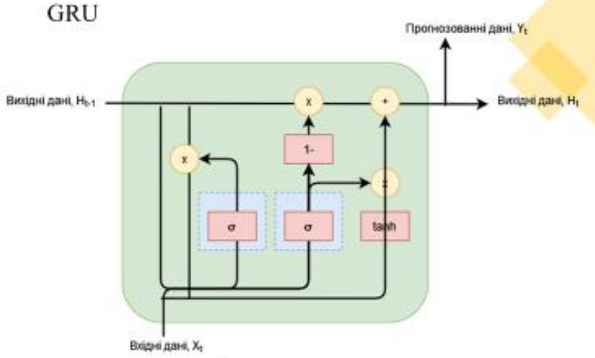
Рекурентні нейронні мережі

LSTM

+ Виправляє проблему зникаючого градієнта
+ Знаходить складні кореляції у часових рядах
- Ресурсоемні



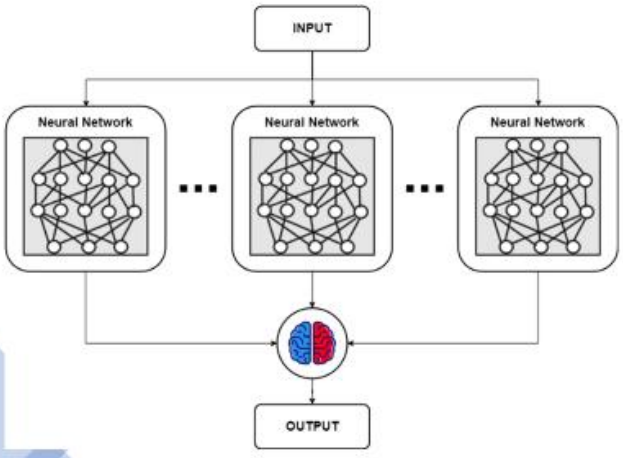
GRU



+ Виправляє проблему зникаючого градієнта
+ Враховує сезонні кореляції
+ Здібні опрацьовувати часові ряди з великим рівнем шуму
- Ресурсоемні

5

Ансамблі нейронних мереж



- Мета використання – підвищення ефективності прогнозування
- Використовує попередньо навчені нейронні
- Допомагає уникнути проблем пов'язаних з перенавчанням

6

Приклад програмної реалізації нейронних мереж та приклад формування ансамблю

Модель нейронної мережі:

```
model = tf.keras.models.Sequential()
model.add(tf.keras.layers.LSTM(8,

input_shape=x_train_uni.shape[-2:]))
model.add(tf.keras.layers.LeakyReLU())
model.add(tf.keras.layers.Dense(1))

model.compile(optimizer='Nadam',
              loss='mae')
```

Приклад формування ансамблю:

```
models = [model_0,model_1,model_2]
model_input =
tf.keras.Input(shape=x_train_uni.shape[-2:])
model_outputs = [model(model_input) for model
in models]
ensemble_output =
tf.keras.layers.Average()(model_outputs)
ensemble_model =
tf.keras.Model(inputs=model_input,
                       outputs=ensemble_output)
```

7

Оцінка точності нейронних мереж

ШНМ	Валюта	Точність прогнозування, %										Сер. точність, %	Коливання точності, %
		1	2	3	4	5	6	7	8	9	10		
LSTM	Долар США	82,9	84,4	80,3	82,2	82,5	82,2	84,7	84,9	84,0	82,9	83,1	4,6
	Долар США	82,5	85,5	85,8	87,3	83,2	84,7	83,3	82,5	86,9	83,0	84,47	4,8
	Долар США	83,2	84,1	84,4	85,1	82,9	85,2	83,3	82,7	86,8	85,8	84,35	4,1
	Євро	64,3	64,1	64,7	62,1	63,1	66,2	63,3	65,5	66,2	63,1	64,26	4,1
	Євро	52,1	51,6	49,8	50,7	50,2	49,5	48,8	47,9	52,8	50,9	50,43	4,9
	Євро	70,2	66,9	68,1	64,3	65,9	64,7	69,0	64,0	65,0	67,8	66,59	6,2
GRU	Долар США	62,4	62,3	59,3	61,8	62,4	60,2	60,8	60,5	59,0	62,7	61,14	3,7
	Долар США	75,6	80,7	78,0	74,8	77,0	81,4	75,3	75,3	74,1	79,6	77,18	7,3
	Долар США	80,0	81,1	81,0	80,0	81,3	80,3	82,1	80,0	81,1	79,6	80,65	2,5
	Євро	64,7	64,5	66,0	64,5	64,5	66,7	63,4	67,6	68,6	65,0	65,55	5,2
	Євро	61,6	64,5	65,7	65,0	67,1	68,4	62,6	65,5	64,1	65,2	64,97	6,8
	Євро	49,7	48,4	49,8	50,3	47,6	51,7	50,7	50,7	50,5	50,0	49,94	4,1

Оцінка точності ансамблів нейронних мереж

Вид ансамблю	Валюта	Точність прогнозування, %										Середня точність, %	Коливання
		1	2	3	4	5	6	7	8	9	10		
Конкатинація	Долар США	81,8	83,6	83,1	83,5	83,5	83,8	83,6	83,3	83,5	83,7	83,3	2
Середній		84,3	85	85,3	85,1	85	84,6	84,7	84,7	85,1	85	84,9	1
Ваговий		85.7	84.9	86.3	86.0	84.9	85.4	87.3	85.7	85.2	87.9	85,9	3
Конкатинація	Євро	64	65,4	65,2	65,5	65,9	66,1	66,3	65,9	66	65,7	65,6	2,3
Середній		68,8	68,9	67,5	67,8	67,4	67,6	67,7	67,7	67,9	67,9	67,9	1,5
Вес		67,9	67,6	69,7	66,0	66,6	69,8	67,1	64,0	65,7	64,0	66,8	5,8

Демонстрація

Висновки

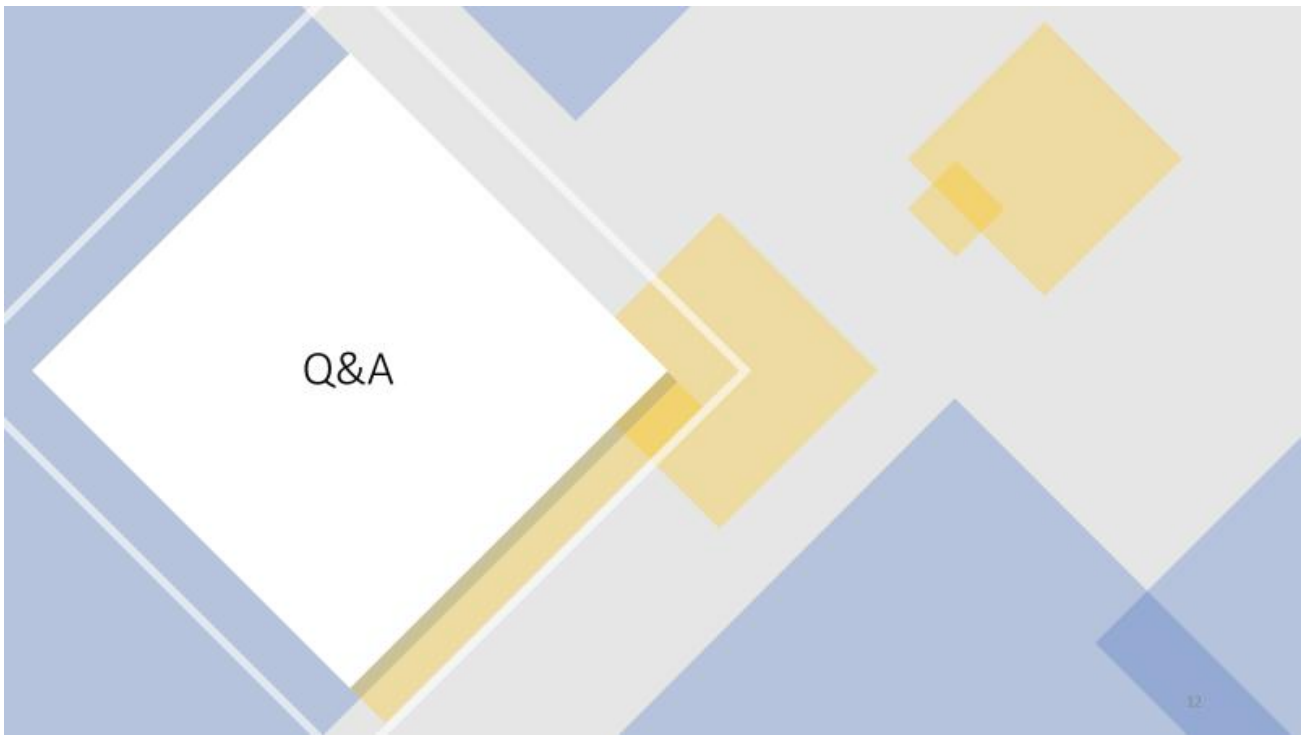
- У ході роботи було:
  - Сформовано та протестованою нейронні мережі з LSTM та GRU архітектурою
  - Створено та оцінено точність ансамблів рекурентних нейронних мереж
  - Визначено, можливість використання методу для курсів з різною кількістю коливань та шумів
- Перспективи розвитку:
  - Збільшення параметрів часового ряду
  - Збільшення періоду прогнозування

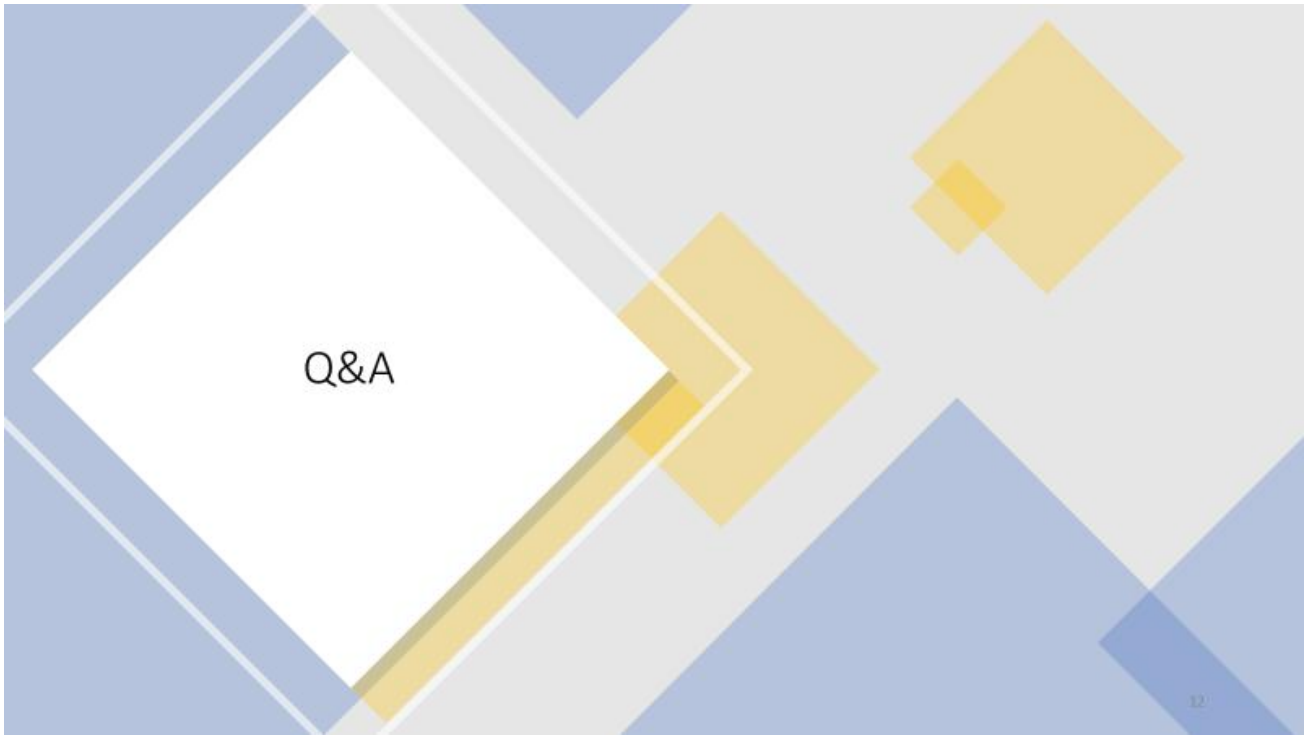
11

Висновки

- У ході роботи було:
  - Сформовано та протестованою нейронні мережі з LSTM та GRU архітектурою
  - Створено та оцінено точність ансамблів рекурентних нейронних мереж
  - Визначено, можливість використання методу для курсів з різною кількістю коливань та шумів
- Перспективи розвитку:
  - Збільшення параметрів часового ряду
  - Збільшення періоду прогнозування

11


Q&A


Q&A

ДОДАТОК Б

ЛІСТИНГ ПРОГРАММИ

Б.1 Лістинг програми прогнозування

```
#%%
from numpy.lib.function_base import average
import tensorflow as tf
from collections import Counter

import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np
import os
import pandas as pd

import DataPreprosesing as dp
import Visualize as visualize
import GetDataFromNationalBank as gd

stopSign = True

while stopSign:
    currency = input("Please enter currency: ")
    date = input("Please enter date to predict in format
D.MM.YYYY: ")

    uni_data, trainSplit = dp.readDataFromFile(currency)
    min = uni_data[:trainSplit].min()
    max = uni_data[:trainSplit].max()

    periodStartTime, periodEndTime =
gd.previousMonthBeforData(date)

    dataStart = gd.selectAndWriteToFileDataForTrain(currency,
periodStartTime, periodEndTime)

    data = dp.predictNormalizeData(dataStart, min, max)

    dataX, dataY = dp.univariate_data(data, 0, 21,
                                      20,
                                      0,
                                      0)

    #Load the model
    model_0 =
tf.keras.models.load_model(f'lstm_simple_{currency}_0.h5',
compile=False)
```

```
    model_0._name = ’model0’

    model_1 =
tf.keras.models.load_model(f’lstm_simple_{currency}_1.h5’,
compile=False)
    model_1._name = ’model1’

    model_2 =
tf.keras.models.load_model(f’lstm_simple_{currency}_2.h5’,
compile=False)
    model_2._name = ’model2’

    model_3 = tf.keras.models.load_model(f’gru_{currency}_0.h5’,
compile=False)
    model_3._name = ’model3’

    model_4 = tf.keras.models.load_model(f’gru_{currency}_1.h5’,
compile=False)
    model_4._name = ’model4’

    model_5 = tf.keras.models.load_model(f’gru_{currency}_2.h5’,
compile=False)
    model_5._name = ’model5’

    models = []
    if currency == "dollar":
        models = [model_0,model_1,model_2,model_4,model_5]
    elif currency == "euro":
        models = [model_0,model_2,model_3,model_4]

    model_input = tf.keras.Input(shape=dataX.shape[-2:])
    model_outputs = [model(model_input) for model in models]
    ensemble_output = tf.keras.layers.Average()(model_outputs)
    ensemble_model = tf.keras.Model(inputs=model_input,
outputs=ensemble_output)

    predictionData = ensemble_model.predict(dataX)[0][0]
    predictionVal = float(round(predictionData*(max -
min)+min,4))

    print("-----------------------------------------------------")
    print("Previous values: ")
    print(dataStart[:-1])
    print("-----------------------------------------------------")
    print(f"Prediction for {date}")
    print(predictionVal)
    print("-----------------------------------------------------")
    stopSign = input("Write ’false’ to stop: ")
# %%
```

Б.2 Лістинг функцій збору даних

```
import requests
from datetime import datetime, timedelta
import numpy as np
import xml.etree.ElementTree as ET

## День появления нац валюты - 02.09.1996
## Последний день когда доллар был до 20 гривен - 05.02.2015
## Делаем прогнозы для доллара, канадского доллара и фунта.
## Учим две сети на двух наборах данных - каждый набор половина
стандарта
## Берем два периода данных ?

## dollar = 169
## canadian dollar = 29
## GBP = 163
## euro = 196
HOST = 'old.bank.gov.ua'
FILE_NAME = 'control/uk/curmetal/currency/search'
FORM_TYPE = 'searchPeriodForm'
TIME_STEP = 'daily'
OUTER = 'xml'
EXECUTE = "Виконати"

def previousMonthBeforData(date):
    targetDate = datetime.strptime(date, "%d.%m.%Y").date()
    periodStartTime = (targetDate -
timedelta(30)).strftime("%d.%m.%Y")
    periodEndTime = (targetDate -
timedelta(1)).strftime("%d.%m.%Y")
    return str(periodStartTime), str(periodEndTime)

def writeDataToFile(fileName,root):
    filePATH = f"C:/Users/Baku-
chan/Desktop/Diploma_Full/data/{fileName}"
    file_writer = open(filePATH, "a")
    file_writer.write("Date,Rate\n")

    for child in root:
        writeLine = ''
        unitNumber = 1
        for childOne in child:
            if(childOne.tag == "date"):
                writeLine += childOne.text + ","
            if(childOne.tag == "number_of_units"):
                unitNumber = int(childOne.text)
            if(childOne.tag == "exchange_rate"):
                rate =  round(float(childOne.text) /
unitNumber,ndigits=3)
                writeLine += str(rate) + "\n"
        file_writer.write(writeLine)
```

```
    file_writer.close()
    return filePATH

def createInputArray(root):
    data = []
    for child in root:
        unitNumber = 1
        for childOne in child:
            if(childOne.tag == "number_of_units"):
                unitNumber = int(childOne.text)
            if(childOne.tag == "exchange_rate"):
                rate =  round(float(childOne.text) /
unitNumber,ndigits=3)
                data.append(rate)
    data.append(0)
    return np.array(data[-21:])

def selectAndWriteToFileDataForTrain(currency, periodStartTime,
periodEndTime, writeToFile = None):
    currency = str(currency)
    if(currency == 'dollar'):
        currency = '169'
        fileName = 'dollar.csv'
    elif(currency == 'canadian_dollar'):
        currency = '29'
        fileName = 'canadian_dollar.csv'
    elif(currency == 'GBP'):
        currency='163'
        fileName = 'GBP.csv'
    elif(currency == 'euro'):
        currency='196'
        fileName = 'euro.csv'

    url =
(f'https://old.bank.gov.ua/control/uk/curmetal/currency/search?'
+
                                'formType=searchPeriodForm' +
                                f'&time_step=daily' +
                                f'&currency={currency}' +

f'&periodStartTime={periodStartTime}' +

f'&periodEndTime={periodEndTime}' +
                                f'&outer=xml' +
                                f'&execute=Виконати')
    response = requests.get(url)
    root = ET.fromstring(response.content)

    if writeToFile:
       return writeDataToFile(fileName,root)
    else:
       return createInputArray(root)
```

Б.3 Лістинг функцій обробки даних

```
#%%
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
import pandas as pd
import tensorflow as tf

## Data prosesing constant
## Step is value of prediction step.
## If we need to predict next one val after 3 days step is 3
STEP = 1

BATCH_SIZE = 256
BUFFER_SIZE = 10000

def readDataFromFile(currency):
    df = pd.read_csv(f"C:/Users/Baku-
chan/Desktop/Diploma/data/{currency}.csv")
    data = df['Rate']
    data.index = df['Date']
    trainSplit = int(0.8*len(data))
    data = data.values
    return data, trainSplit

def standardizeData(data, trainSplit):
    uni_train_mean = data[:trainSplit].mean()
    uni_train_std = data[:trainSplit].std()
    uni_data = (data-uni_train_mean)/uni_train_std
    return uni_data, uni_train_mean, uni_train_std

def normalizeData(data, trainSplit):

    uni_train_min = data[:trainSplit].min()
    uni_train_max = data[:trainSplit].max()

    uni_data = (data-uni_train_min)/(uni_train_max -
uni_train_min)

    return uni_data, uni_train_min, uni_train_max

def predictNormalizeData(data, min, max):
    uni_data = (data-min)/(max - min)
    return uni_data

def prepareTrainAndTestSets(data, trainSplit, testData,
univariate_past_history, univariate_future_target, number=0):

    x_train_uni, y_train_uni = univariate_data(data, 0,
trainSplit,
```

```
univariate_past_history,
univariate_future_target,
number)

    x_val_uni, y_val_uni = univariate_data(data, trainSplit,
testData,
univariate_past_history,
univariate_future_target,
number)

    x_test_uni, y_test_uni = univariate_data(data, testData,
None,
univariate_past_history,
univariate_future_target,
number)
    return x_train_uni, y_train_uni, x_val_uni, y_val_uni,
x_test_uni, y_test_uni

def shuffleBatchCache(xSet,ySet):
  univariate = tf.data.Dataset.from_tensor_slices((xSet, ySet))
  univariate =
univariate.cache().shuffle(BUFFER_SIZE).batch(BATCH_SIZE).repeat
()
  return univariate

##Form arrays of values to train and test
def univariate_data(dataset, start_index, end_index,
history_size, target_size, number):
  data = []
  labels = []

  start_index = start_index + history_size +number
  if end_index is None:
    end_index = len(dataset) - target_size

  for i in range(start_index, end_index):
    indices = range(i-history_size, i)
    # Reshape data from (history_size,) to (history_size, 1)
    data.append(np.reshape(dataset[indices], (history_size, 1)))
    labels.append(dataset[i+target_size])
  return np.array(data), np.array(labels)

  return np.array(data), np.array(labels)

def returnDataToRealValues(predicted,
realValue,train_std,train_mean):
    predictionVal =
float(round((predicted*train_std)+train_mean,4))
    realVal =
float(round(((round(realValue,7))*train_std)+train_mean,4))
    infelicity = float(predictionVal - realVal)
    return infelicity
# %%
```

Б.4 Лістинг функцій навчання нейронної мережі LSTM

```
#%%
import tensorflow as tf

import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np
import os
import pandas as pd

from collections import Counter

import DataPreprosesing as dp
import Visualize as visualize

currency = "dollar"
number = 1
univariate_past_history = 20
univariate_future_target = 0
BATCH_SIZE = 256
BUFFER_SIZE = 10000

EVALUATION_INTERVAL = 500
EPOCHS = 60
mpl.rcParams['figure.figsize'] = (8, 6)
mpl.rcParams['axes.grid'] = False

uni_data, trainSplit = dp.readDataFromFile(currency)
testData = int(trainSplit+ ((len(uni_data) - trainSplit)//2))

uni_data, uni_train_min, uni_train_max =
dp.normalizeData(uni_data, trainSplit)

x_train_uni, y_train_uni, x_val_uni, y_val_uni, x_test_uni,
y_test_uni = dp.prepareTrainAndTestSets(uni_data, trainSplit,
testData, univariate_past_history, univariate_future_target,
number)

testSize = len(x_test_uni)
train_univariate, val_univariate =
dp.shuffleBatchCache(x_train_uni,y_train_uni,x_val_uni,y_val_uni
)

simple_lstm_model = tf.keras.models.Sequential([
    tf.keras.layers.LSTM(8, input_shape=x_train_uni.shape[-2:]),
    tf.keras.layers.LeakyReLU(),
    tf.keras.layers.Dense(1)
])

simple_lstm_model.compile(optimizer='Nadam', loss='mae')
```

```
history = simple_lstm_model.fit(train_univariate, epochs=EPOCHS,
                      steps_per_epoch=EVALUATION_INTERVAL,
                      validation_data=val_univariate,
validation_steps=50)

visualize.plot_train_history(history, 'Multi-Step Training and
validation loss')

simple_lstm_model.save(f'C:/Users/Baku-
chan/Desktop/Diploma/lstm_simple_{currency}_{number}.h5')

for x, y in val_univariate.take(2):
  plot = visualize.show_plot([x[0].numpy(), y[0].numpy(),
                      simple_lstm_model.predict(x)[0]], 0, 'Simple
LSTM model')
  plot.show()

  # %%
```

Б.5 Лістинг функцій навчання нейронної мережі GRU

```
#%%
import tensorflow as tf

import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np
import os
import pandas as pd

import DataPreprosesing as dp
import Visualize as visualize

from collections import Counter

## EVALUATION_INTERVAL=300 EPOCHS = 1000

currency = "euro"
number = 2
EVALUATION_INTERVAL = 500
EPOCHS = 60
univariate_past_history = 20
univariate_future_target = 1
STEP = 1
BATCH_SIZE = 256
BUFFER_SIZE = 10000

mpl.rcParams['figure.figsize'] = (8, 6)
mpl.rcParams['axes.grid'] = False

uni_data, trainSplit = dp.readDataFromFile(currency)
testData = int(trainSplit+ ((len(uni_data) - trainSplit)//2))

uni_data, uni_train_min, uni_train_max =
dp.normalizeData(uni_data, trainSplit)

x_train_uni, y_train_uni, x_val_uni, y_val_uni, x_test_uni,
y_test_uni = dp.prepareTrainAndTestSets(uni_data, trainSplit,
testData, univariate_past_history, univariate_future_target,
number)

testSize = len(x_test_uni)
train_univariate, val_univariate =
dp.shuffleBatchCache(x_train_uni,y_train_uni,x_val_uni,y_val_uni
)

simple_gru_model = tf.keras.models.Sequential([
    tf.keras.layers.GRU(8, input_shape=x_train_uni.shape[-2:]),
    # tf.keras.layers.Activation('linear'),
    tf.keras.layers.LeakyReLU(),
    tf.keras.layers.Dense(1)
```

```
])

simple_gru_model.compile(optimizer='Nadam', loss='mae')
##was RMSprop
##was Adagrad - i like this one
## was Adadelta - no
## Adamax 80%
## Nadam 80%
## SGD no
##tf.keras.optimizers.SGD(lr=0.01, decay=1e-7, momentum=0.9,
nesterov=False)

step_history = simple_gru_model.fit(train_univariate,
epochs=EPOCHS,

steps_per_epoch=EVALUATION_INTERVAL,

validation_data=val_univariate,
                                                validation_steps=50)

visualize.plot_train_history(step_history, 'Multi-Step Training
and validation loss')

simple_gru_model.save(f'C:/Users/Baku-
chan/Desktop/Diploma/gru_{currency}_{number}.h5')

for x, y in val_univariate.take(2):
  plot = visualize.show_plot([x[0].numpy(), y[0].numpy(),
                      simple_gru_model.predict(x)[0]], 0, 'Simple
LSTM model')
  plot.show()
```

Б.6 Лістинг функції оцінки точності прогнозування

```
resulted = []

for i in range(0,10):
    predicted = []
    prediction = []
    real = []
    diferense = []
    test_univariate =
tf.data.Dataset.from_tensor_slices((x_test_uni, y_test_uni))
    test_univariate =
test_univariate.cache().shuffle(BUFFER_SIZE).batch(BATCH_SIZE).r
epeat()

    for x, y in test_univariate.take(len(x_test_uni)):
        predictionData = ensemble_model.predict(x)[0][0]
        readlData = round(tf.keras.backend.get_value(y[0]),7)
        predictionVal =
float(round(predictionData*(uni_train_max -
uni_train_min)+uni_train_min,4))
        realVal = float(round(readlData*(uni_train_max -
uni_train_min)+uni_train_min,4))
        infelicity = round(float(predictionVal - realVal),3)
        diferense.append(infelicity)
        if(infelicity > -0.100):
            if(infelicity < 0.100):
                predicted.append(True)
            else:
                predicted.append(False)
        else:
            predicted.append(False)
        prediction.append(predictionVal)
        real.append(realVal)

    c = Counter(predicted)
    mostlyGood = c[True]
    calculatedAccurency = (mostlyGood/len(predicted))*100
    print(round(calculatedAccurency,1))
    resulted.append(round(calculatedAccurency,1))
    i+=1
print(resulted)
print(min(resulted))
print(max(resulted))
print(average(resulted))
```