

گزارش تمرین چهارم یادگیری عمیق

۹۸۲۴۳۰۹۲

بخش اول؛ سوالات:

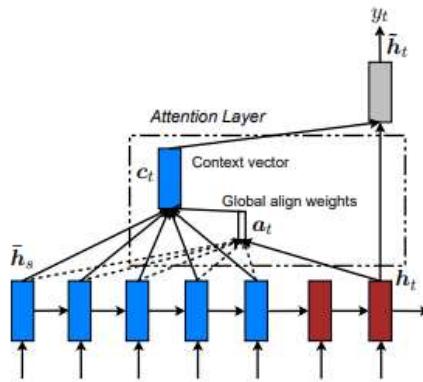
۱. تفاوت دو حالت hard attention و soft attention را توضیح دهید.

در مکانیسم توجه نرم برای محاسبه بردار زمینه، از میانگین وزن دار حالت های پنهان انکدرها استفاده می شود. اما در توجه سخت آن ازتابع attention score استفاده می کنیم تا یک حالت پنهان را انتخاب کنیم و از آن برای محاسبه context vector استفاده کنیم.

در توجه نرم اگر از back propagation برای محاسبه گرادیان استفاده کنیم، روند آسان تر و خطی ای خواهیم داشت.

۲. تفاوت local attention و global attention را به طور کامل و با رسم شکل توضیح دهید.

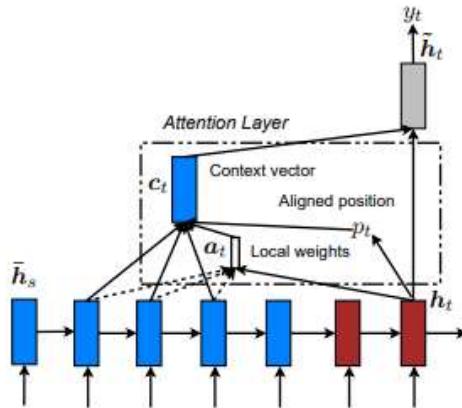
در global attention ورودی از هر استیت منبع (انکدر) و استیت های دیکدر قبل از استیت فعلی برای محاسبه خروجی در نظر گرفته می شود. در زیر نمودار مدل توجه جهانی است.



همانطور که در شکل مشاهده می شود، a_t وزن های تراز یا وزن توجه با استفاده از هر مرحله انکدر و h_t دیکدر مرحله قبلی محاسبه می شود. سپس با استفاده از a_t بردار زمینه با در نظر گرفتن حاصل ضرب وزن های global attention و هر گام decoder محاسبه می شود. سپس به سلول RNN داده می شود تا خروجی decoder را پیدا کند.

مدل توجه محلی

این مدل با مدل global attention متفاوت است به گونه ای که در مدل توجه محلی تنها چند موقعیت از انکدر برای محاسبه وزن های تراز at استفاده می شود. در زیر نمودار مدل توجه محلی آمده است.



با توجه تصویر می توان اشاره کرد، ابتدا موقعیت تک تراز (pt) پیدا می شود سپس پنجره ای از کلمات از لایه منبع (انکدر) به همراه (h_t) برای محاسبه وزن تراز و بردار زمینه استفاده می شود.

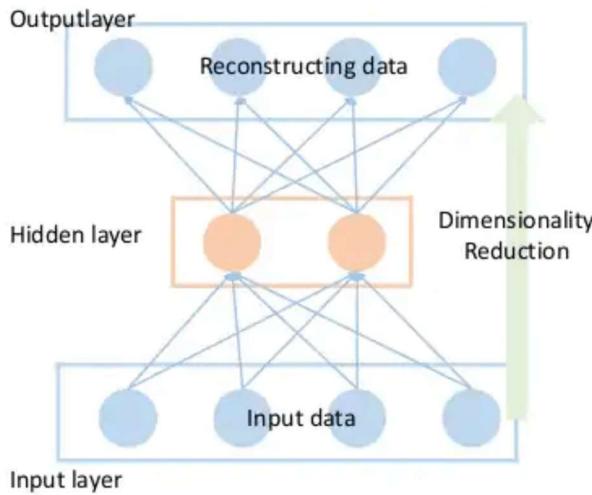
۳. توضیح دهید چرا autoencoder یک روش خودنظرارتی است؟

در روش خودنظرارتی می دانیم که به دلایلی همچون کمبود داده های آموزشی یا برچسب نداشتیم داده ها، شبکه سعی می کند از قسمتی از داده های ورودی برای به عنوان داده تست برای تsek های دیگر استفاده شود برای مثال حالتی که قسمتی از تصاویر ورودی را ماسکه می کنند تا شبکه بتواند آن قسمت را پیش بینی کند تا یک بازنمایی از تصاویر را بتواند یادبگیرد در حالی که تصاویر برای یک تsek اصلی دیگر می باشد به عبارت دیگر شبکه سعی می کند به نوعی داده های ورودی را در خروجی نیز تولید کند. از طرفی دیگر در اتونکدر ها نیز هدف آن است که شبکه داده ورودی را در خروجی مجدد تولید کند در عین حالی که در لایه های میانی یم بازنمایی از داده ها را یادگرفته است بنابراین می توان تیجه گرفت که اتونکدر ها یک روش خودنظرارتی هستند.

۴. آیا autoencoder یک روش کاهش بعد است؟ چرا؟

atonkdr ها شاخه ای از شبکه های عصبی هستند که اساساً اطلاعات متغیرهای ورودی را در یک فضای ابعادی کاهش یافته فشرده می کنند و سپس مجموعه داده های ورودی را دوباره ایجاد می کنند تا دوباره آن را آموزش دهند. مؤلفه کلیدی در اینجا لایه پنهان گلوگاه است. در این لایه اطلاعات داده های ورودی فشرده می شود و بنابراین با استخراج این لایه از مدل، اکنون هر گره می تواند به عنوان یک

متغیر به همان روشی که هر جزء اصلی انتخاب شده به عنوان یک متغیر در مدل های بعدی استفاده می شود، در نظر گرفته شود.



۵. معناری قسمت **encoder** و **decoder** در یک **autoencoder** از چه جهات شباهت دارد؟

یکسانی چه پارامترها یا ویژگی هایی در این دو قسمت الزامی است؟

در اتوانکدر ها انکدر برای تولید یک بازنمایی از ویژگی های کاهش یافته ورودی توسط یک لایه پنهان استفاده می شود. دیکدر برای بازسازی ورودی اولیه از خروجی انکدر با به حداقل رساندن عملکرد تلفات استفاده می شود. بنابراین لازم است که اندازه خروجی لایه انکدر برابر با اندازه ورودی دیکدر باشد و از آنجایی که در اتوانکد ها باید ورودی و خروجی شبکه یکسان باشند باید اندازه ورودی انکدر با خروجی دیکدر نیز برابر باشد.

۶. مدل های **autoencoder** در چه تسک هایی کاربرد دارند؟ مختصراً توضیح دهید. (حداقل دو

(مورد)

از کاربرد های **autoencoder** ها می توان به کاهش ابعاد، ازبین بردن نویز تصویر و بازیابی اطلاعات اشاره کرد.

کاهش ابعاد

کاهش ابعاد یکی از اولین کاربردها در یادگیری عمیق بود.

در مقاله ای یک **autoencoder** چند لایه با پشته ای از RBM ها از قبل آموزش داده شد و سپس از وزن آن ها برای مقداردهی اولیه یک اتوانکدر عمیق با لایه های پنهان استفاده شد که در این اتوانکدر به تدریج لایه های پنهانش کوچکتر شده تا اینکه به گلوگاه ۳۰ نورون رسید. این ۳۰ بعد به دست آمد

خطای بازنمایی کمتری به نسبت ۳۰ مؤلفه اولیه (PCA) به همراه داشت و یک بازنمایی را آموخت که تفسیر کیفی آن آسان‌تر بود و به وضوح خوش‌های داده را از هم جدا می‌کرد.

بازنمایی ابعاد می‌تواند عملکرد تسک‌هایی مانند طبقه‌بندی را بهبود بخشد. در واقع، مشخصه کاهش ابعاد، قرار دادن مثال‌های مرتبط معنایی در نزدیکی یکدیگر است.

حذف نویز تصویر

اتوانکردهای می‌توانند برای انجام حذف نویز تصویر کارآمد و بسیار دقیق باشند. برخلاف روش‌های سنتی حذف نویز، اتوانکردهای نویز را جستجو نمی‌کنند، آنها تصویر را از داده‌های نویزدار که قبلاً یعنی بازنمایی از آن را یادگرفته‌اند استخراج می‌کنند. سپس نمایش از حالت فشرده خارج می‌شود تا تصویری بدون نویز ایجاد شود. بنابراین اتوانکردهای می‌توانند نویز را از تصاویر پیچیده‌ای که نمی‌توان از طریق روش‌های سنتی حذف نویز کرد، حذف کنند.

بخش دوم؛ پیاده‌سازی:

توضیحات کد:

```
▶ import pandas as pd  
!git clone https://github.com/SBU-CE/Deep-Learning  
stk_data = pd.read_csv('./Deep-Learning/spring-2022/assignments/project-4/GOOG.csv', index_col='Date')  
stk_data.head()
```

	Open	High	Low	Close	Adj Close	Volume
Date						
2013-12-20	542.117981	548.528992	541.968567	548.255005	548.255005	6547651
2013-12-23	551.851501	555.816650	550.496582	555.467957	555.467957	3456106

در این قسمت فایل دیتا سمت خوانده شده و در متغیر از نوع فریم `stk_data` قرار گفته است. این دیتا سمت دارای ۶ ستون است که فقط ستون `date` و `close` مورد استفاده قرار می‌گیرد.

```

import matplotlib.dates as mdp
import matplotlib.pyplot as plt
import datetime as dt
import numpy as np
plt.figure(figsize=(15,10))
plt.gca().xaxis.set_major_formatter(mdp.DateFormatter('%Y-%m-%d'))
plt.gca().xaxis.set_major_locator(mdp.DayLocator(interval=180))
x_dates= [dt.datetime.strptime(d,'%Y-%m-%d').date() for d in stk_data.index.values]
|
plt.plot(x_dates, stk_data['Close'], label='Close')
plt.xlabel(' Time Scale')
plt.ylabel('USD ($)')
plt.legend()
plt.gcf().autofmt_xdate()
plt.show()

```

در این قسمت استفاده از توابع کتابخانه ای `pyplot` نمودار داده های `close` بر حسب تاریخ که به صورت سری زمانی است رسم شده است.

```

[ ] from sklearn.preprocessing import MinMaxScaler, StandardScaler
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_percentage_error
from sklearn.model_selection import train_test_split
from sklearn.model_selection import TimeSeriesSplit

import torch
import torch.nn as nn

#Data Preprocessing
#Target and Features

data_close=stk_data.filter(['Close'])
sub_data=stk_data.iloc[:,0:4]
#feature Scaling
s_data=data_close.values
date_index=stk_data.index
sca=MinMaxScaler(feature_range=(0,1))
normal_data=sca.fit_transform(s_data)

```

در این قسمت ابتدا توابع مورد نیاز کتابخانه `sklearn` بارگذاری شده اند.

در قسمت دوم نیز ستون داهای `close` جدا شده است. برای اینکه داهای اثر یکسانی در یادگیری داشته باشند بین صفر و یک نرمال شده اند و در متغیر `normal_data` قرار گرفته است که این متغیر در قسمت های بعدی برای یادگیری و تست استفاده خواهد شد.

```

def data_split(data, step_size):
    x,y,z=[],[],[]
    for i in range(step_size,len(data)):
        x.append(data[i-step_size:i,-1])
        y.append(data[i-1,-1])
    return np.array(x), np.array(y)

window_size=30 #The number of days to
train_rate=0.8

x1, y1=data_split(normal_data, step_size=window_size)

split_index=int(np.ceil(len(x1)*(train_rate)))
x_train,x_test=x1[:split_index],x1[split_index:]
y_train,y_test=y1[:split_index],y1[split_index:]

print(x1.shape,x_train.shape,x_test.shape,y_train.shape, y_test.shape)

x_test=np.reshape(x_test,(x_test.shape[0],x_test.shape[1],1))
x_train=np.reshape(x_train,(x_train.shape[0],x_train.shape[1],1))
y_train=np.reshape(y_train,(y_train.shape[0],1))
y_test=np.reshape(y_test,(y_test.shape[0],1))

print(x_train.shape)
print(y_test.shape)

```

در این قسمت پیش پردازش ابتدایی داده ها انجام شده است. ابتدا تابعی نوشته شده است که داده ها را به تعداد روز مشخص شده در `step_size` که در واقع یک پنجره است دسته بندی می کند. یعنی هر روز از داده ها رابه عنوان یک سطر از داده ها در نظر گرفته و روز اخر را هم به عنوان برچسب یا خروجی در نظر می گیرد. در هنگام فراخوانی با توجه به خواسته مساله `step_size` را ۳۰ روز در نظر می گیریم و بنابراین داده سی روز اول را یک سطر ر نظر می گیریم و داه روز سی ام را نتیجه پیش بینی این ۳۰ روز در نظر می گیریم و به همین ترتیب تا آخر داده ها ادامه می دهیم.

در اخر هم با نسبت ۲۰ به ۸۰ داده ها را به یادگیری و تست تقسیم می کنیم که در متغیرهای ورودی `x` و `x_train` و `x_test` و متغیرهای خروجی یا برچسب `y_train` و `y_test` قرار می دهیم. با توجه به این که ورودی های شبکه های LSTM و GRU باید سه بعدی باشند انها را ریشیپ می کنم.

```

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import LSTM
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import *
from tensorflow.keras.callbacks import EarlyStopping

```

برای پیاده سازی شبکه از توابع کتابخانه ای keras استفاده می کنیم و بنابراین انها را در این قسمت بارگزاری می کنیم.

```

#LSTM Model One
lstm1=Sequential()
lstm1.add(LSTM(60,input_shape=(x_train.shape[1],x_train.shape[2]),activation='tanh',return_sequences=False))
lstm1.add(Dropout(0.2))
lstm1.add(Dense(1))
lstm1.compile(loss='mse',optimizer='adam')
lstm1.summary()

```

در ای قسمت مدل اول شبکه LSTM را ایجاده کرده ایم. این مدل شامل یک لایه LSTM با 60 واحد و یک لایه درآپات برای جلوگیری از بیش برازش و یک لایه دنس خروجی است. تابع خطا MSE، بهینه ساز آن و تابع فعال ساز نیز tanh در نظر گرفته شده است.

ساختار مدل و پارامترهای آن به صورت زیر است:

```

Model: "sequential"
=====
Layer (type)          Output Shape         Param #
=====
lstm (LSTM)           (None, 60)          14880
dropout (Dropout)     (None, 60)          0
dense (Dense)         (None, 1)           61
=====
Total params: 14,941
Trainable params: 14,941
Non-trainable params: 0

```

```

history=lstm1.fit(x_train,y_train,epochs=100,batch_size=10, verbose=1)

y_test_pred=lstm1.predict(x_test)
y_train_pred=lstm1.predict(x_train)
y_test_nn=sca.inverse_transform(y_test)
rmse=mean_squared_error(y_test,y_test_pred,squared=False)

y_test_pred_nn=sca.inverse_transform(y_test_pred)
y_train_pred_nn=sca.inverse_transform(y_train_pred)

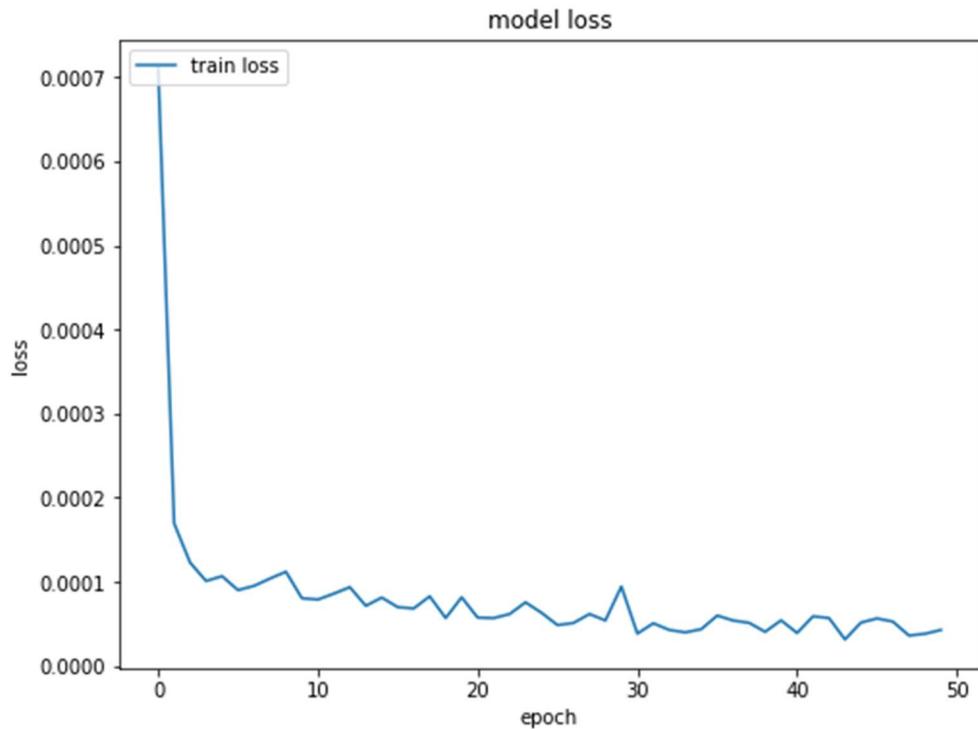
rmse1=mean_squared_error(y_test_nn,y_test_pred_nn,squared=False)
mape=mean_absolute_percentage_error(y_test,y_test_pred)

print('Normalized Rmse=',rmse, 'RMSE=',rmse1,'MAPE=',mape)

plt.figure(figsize=(8, 6))
plt.plot(history.history['loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train'], loc='upper left')
plt.show()

```

در این قسمت ابتدا با استفاده از تابع `fit` شبکه را با ۱۰۰ ایپاک آموزش داده ایم و با استفاده از شبکه آموزش داده شده با استفاده از داده های `x_test` داده های پیش بینی شده توسط شبکه در `y_test_pred` قرار گرفته است. در این قسمت RMSE که میانگین مربع خطای برای داده های پیش بینی شده (`y_test_pred`) و داده های اصلی یعنی (`y_test`) محاسبه شده است. البته چون داده های نرمال هستند این RMSE هم برای داده های نرمال است و بین صفر و یک هست. داده ها را به صورت اصلی و غیر نرمال شده هم استفاده کردیم و رسم کردیم که برای داده های اصلی است حساب کرده ایم. همچنین MAPE را هم برای داده های `loss` که درصد خطای مطلق میانگین است را هم برای داده های تست محاسبه کرده ایم. در پایان نیز نمودار خطای `loss` در ایپاک ها را هم رسم کرده ایم که روند آموزش شبکه و کاهش `loss` در ایپاک ها نشان می دهد.

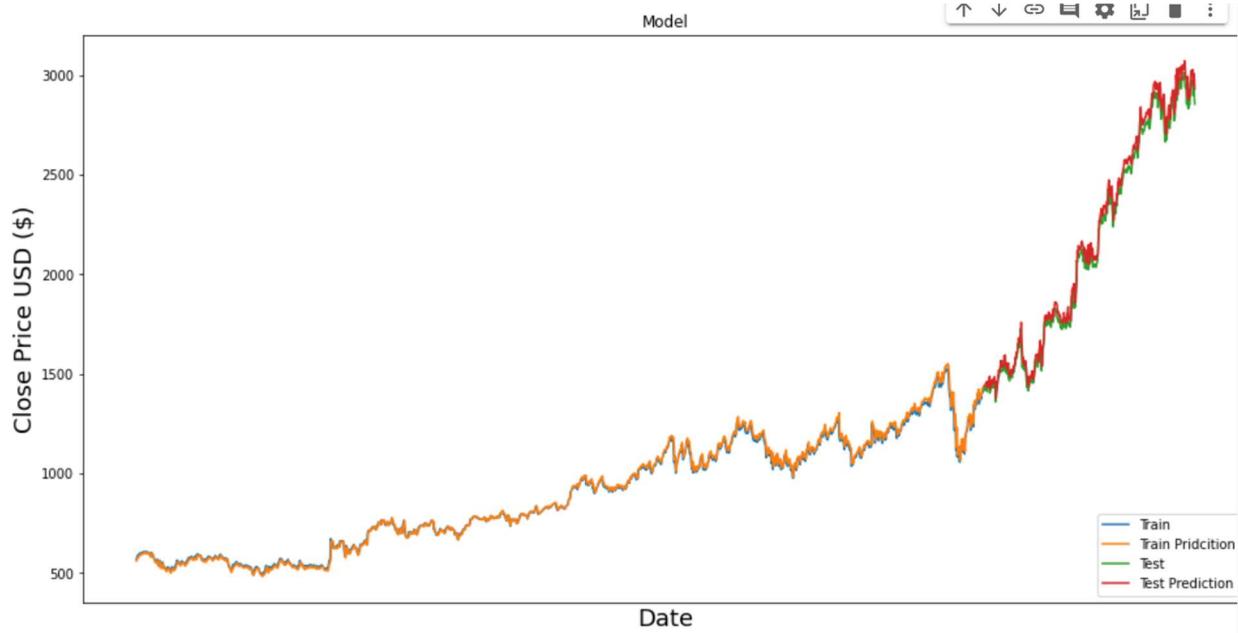


```
train=data_close[window_size:split_index+window_size]
valid=data_close[split_index+window_size:]

train['Prediction'] =y_train_pred_nn
valid['Prediction'] =y_test_pred_nn

#Visualize the data
#print(valid)
plt.figure(figsize=(16,8))
plt.title('Model')
plt.xlabel('Date', fontsize=18)
plt.ylabel('Close Price USD ($)', fontsize=18)
plt.plot(train[['Close', 'Prediction']])
plt.plot(valid[['Close', 'Prediction']])
plt.gca().xaxis.set_major_locator(md.DateLocator(interval=120))
plt.legend(['Train', 'Train Prediction', 'Test', 'Test Prediction'], loc='lower right')
plt.show()
#valid.head()
```

در نهایت در این قسمت نمودار خروجی اصلی و خروجی پیش بینی شده (Close) بر حسب زمان برای داده های تست و یادگیری را رسم می کنیم که نمونه ای از به صورت زیر است:



```
#LSTM Model two
lstm2=Sequential()
lstm2.add(LSTM(60,input_shape=(x_train.shape[1],x_train.shape[2]),activation='tanh',return_sequences=True))
lstm2.add(Dropout(0.2))
lstm2.add(LSTM(units = 60, activation='tanh', return_sequences = False))
lstm2.add(Dropout(0.2))
lstm2.add(Dense(1))
lstm2.compile(loss='mse',optimizer='adam')
lstm2.summary()

history=lstm2.fit(x_train,y_train,epochs=100,batch_size=10, verbose=1)

y_test_pred=lstm2.predict(x_test)
y_train_pred=lstm2.predict(x_train)

rmse=mean_squared_error(y_test,y_test_pred,squared=False)
y_test_pred_nn=sca.inverse_transform(y_test_pred)

y_test_nn=sca.inverse_transform(y_test)
y_train_pred_nn=sca.inverse_transform(y_train_pred)

rmse1=mean_squared_error(y_test_nn,y_test_pred_nn,squared=False)
mape=mean_absolute_percentage_error(y_test,y_test_pred)

print('Normalized Rmse=',rmse, 'RMSE=',rmse1,'MAPE=',mape)
```

ذریان قسمت مثل جالت قبلی مدل دوم LSTM را پیاوه سازی کرده و آموزش دادیم. این مدل دارای دو لایه پشتہ ای lstm با ۶۰ واحد پنهان می باشد.

```
#LSTM Model Three
lstm3=Sequential()
lstm3.add(LSTM(50, input_shape=(x_train.shape[1],x_train.shape[2]),activation='tanh',return_sequences=True))
lstm3.add(Dropout(0.2))
lstm3.add(LSTM(units = 50, activation='tanh', return_sequences = True))
lstm3.add(Dropout(0.2))
lstm3.add(LSTM(units = 50, activation='tanh', return_sequences = False))
lstm3.add(Dropout(0.2))
lstm3.add(Dense(1))
lstm3.compile(loss='mse',optimizer='adam')
lstm3.summary()

history=lstm3.fit(x_train,y_train,epochs=100,batch_size=10, verbose=1)

y_test_pred=lstm3.predict(x_test)
y_train_pred=lstm3.predict(x_train)

rmse=mean_squared_error(y_test,y_test_pred,squared=False)

y_test_pred_nn=sca.inverse_transform(y_test_pred)
y_test_nn=sca.inverse_transform(y_test)

y_train_pred_nn=sca.inverse_transform(y_train_pred)
rmse1=mean_squared_error(y_test_nn,y_test_pred_nn,squared=False)
mape=mean_absolute_percentage_error(y_test,y_test_pred)
|
print('Normalized Rmse=',rmse, 'RMSE=',rmse1,'MAPE=',mape)
```

در این فسنت هم مثل دو مدل قبلی مدل سوم lstm را پیاده سازی کردیم که دارای سه لایه پشتہ ای است که دارای ۵۰۸۵۱ پارامتر قابل یادگیری است.

```

#model GRU1
GRU1 = Sequential()
GRU1.add(GRU(50, input_shape=(30, 1)))
GRU1.add(Dense(units = 512, activation = 'tanh'))
GRU1.add(Dropout(0.2))
GRU1.add(Dense(units = 1, activation = 'linear'))
GRU1.summary()

GRU1.compile(loss='mse', optimizer='adam')

history=GRU1.fit(x_train,y_train,epochs=100,batch_size=10, verbose=1)

y_test_pred=GRU1.predict(x_test)
y_train_pred=GRU1.predict(x_train)

rmse=mean_squared_error(y_test,y_test_pred,squared=False)

y_test_pred_nn=sca.inverse_transform(y_test_pred)
y_test_nn=sca.inverse_transform(y_test)

y_train_pred_nn=sca.inverse_transform(y_train_pred)
rmse1=mean_squared_error(y_test_nn,y_test_pred_nn,squared=False)
mape=mean_absolute_percentage_error(y_test,y_test_pred)

print('Normalized Rmse=',rmse, 'RMSE=',rmse1,'MAPE=',mape)

```

در این قسمت یک شبکه GRU ایجاد کردیم. این شبکه دارای یک لایه GRU با ۵۰ واحد و یک لایه دنس با ۱۲ نرون و یک لایه دنس با یک نرون در خروجی است. برای جلوگیری از بیش برازش از دراپ اب با نسبت ۰،۲ استفاده شده است. در اخر هم شبکه را آموزش داده و میانگین مربع خطای را محاسبه کرده نمودار loss بر حسب ایپاک ها را برای نمایش روند یادگیری رسم کرده ایم. نمودار مقادیر پیش بینی شده و مقادیر اصلی نیز مثل شبکه های دیگر در ادامه رسم شده است.

```
#model GRU2
GRU2 = Sequential()
GRU2.add(GRU(50, input_shape=(30, 1), return_sequences=True))
GRU2.add(Dropout(0.2))
GRU2.add(GRU(512, input_shape=(30, 1)),)
GRU2.add(Dense(units = 512, activation = 'tanh'))
GRU2.add(Dropout(0.2))
GRU2.add(Dense(units = 1, activation = 'linear'))
GRU2.summary()

GRU2.compile(loss='mse', optimizer='adam')

history=GRU2.fit(x_train,y_train,epochs=100,batch_size=10, verbose=1)

y_test_pred=GRU2.predict(x_test)
y_train_pred=GRU2.predict(x_train)

rmse=mean_squared_error(y_test,y_test_pred,squared=False)

y_test_pred_nn=sca.inverse_transform(y_test_pred)
y_test_nn=sca.inverse_transform(y_test)

y_train_pred_nn=sca.inverse_transform(y_train_pred)
rmse1=mean_squared_error(y_test_nn,y_test_pred_nn,squared=False)
```

در این قسمت نیز شبکه دوم GRU با دو لایه پشته ای GRU ایجاد و آموزش داده شده است و نمودارها و محاسبات هم مثل شبکه قبلی انجام شده است.

```

#model GRU3
GRU3 = Sequential()
GRU3.add(GRU(50, input_shape=(30, 1), return_sequences=True))
GRU3.add(Dropout(0.2))
GRU3.add(GRU(50, input_shape=(30, 1), return_sequences=True))
GRU3.add(Dropout(0.2))
GRU3.add(GRU(512, input_shape=(30, 1)))
GRU3.add(Dense(units = 512, activation = 'tanh'))
GRU3.add(Dropout(0.2))
GRU3.add(Dense(units = 1, activation = 'linear'))
GRU3.summary()

GRU3.compile(loss='mse', optimizer='adam')

history=GRU3.fit(x_train,y_train,epochs=100,batch_size=10, verbose=1)

y_test_pred=GRU3.predict(x_test)
y_train_pred=GRU3.predict(x_train)

rmse=mean_squared_error(y_test,y_test_pred,squared=False)

y_test_pred_nn=sca.inverse_transform(y_test_pred)
y_test_nn=sca.inverse_transform(y_test)

y_train_pred_nn=sca.inverse_transform(y_train_pred)
rmse1=mean_squared_error(y_test_nn,y_test_pred_nn,squared=False)

```

در نهایت سومین شبکه GRU با استفاده از سه لایه پشته ای ایجاد و آموزش دیده شده است و محاسبات و نمودارهای لازم انجام شده است. ساختار شبکه به صورت زیر است:

```

for m in range(20,200,10):
    av_rmse=0
    av_rmse1=0
    av_mape=0

#LSTM Model One
    for i in range(5):
        lstm1=Sequential()
        lstm1.add(LSTM(m,input_shape=(x_train.shape[1],x_train.shape[2]),activation='tanh',return_sequences=False))
        lstm1.add(Dropout(0.2))
        lstm1.add(Dense(1))
        lstm1.compile(loss='mse',optimizer='adam')
        #lstm1.summary()
    #plot_model(lstm1, to_file='model_plot.png', show_shapes=True, show_layer_names=True)

    print(m, ' ',i)
    history=lstm1.fit(x_train,y_train,epochs=50,batch_size=10, verbose=0)
    y_test_pred=lstm1.predict(x_test)
    y_train_pred=lstm1.predict(x_train)

    rmse=mean_squared_error(y_test,y_test_pred,squared=False)
    av_rmse=av_rmse+rmse
    y_test_pred_nn=sca.inverse_transform(y_test_pred)
    y_train_pred_nn=sca.inverse_transform(y_train_pred)
    y_test_nn=sca.inverse_transform(y_test)
    rmse1=mean_squared_error(y_test_nn,y_test_pred_nn,squared=False)
    mape=mean_absolute_percentage_error(y_test,y_test_pred)
    av_rmse1=av_rmse1+rmse1
    av_mape=av_mape+mape
    lstm1.reset_states()
    plt.figure(figsize=(8, 6))
    plt.plot(history.history['loss'])
    #plt.plot(history.history['val_loss'])
    plt.title('model loss')
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.legend(['train loss'], loc='upper left')
    plt.show()
    print('Normalized Rmse=',rmse, 'RMSE=',rmse1,'MAPE=',mape)

print('Mean Norm RMSE=',av_rmse/5,'Mean RMSE=',av_rmse1/5,'Mean MAPE=',av_mape/5)

```

برای تنظیم پارامترها از ساختارهایی مانند این کد استفاده شده است. مثلا برای تنظیم تعداد واحدهای LSTM ها از یک حلقه استفاده شده است و در این حلقه تعداد واحدها از ۲۰ تا ۲۰۰ با گامهای ۱۰ تایی تغییر می کند و در هر گام هم ۵ اجرا گرفته و میانگین خطای را ثبت می کنیم. در نهایت در تعداد واحدی که میانگین خطای کم شده است به عنوان مدل انتخاب می شود.

توضیحات بیشتر در فایل جداگانه تنظیم پارامترها توضیح داده شده است.

تنظیم مقدار های پرپارامتر ها:

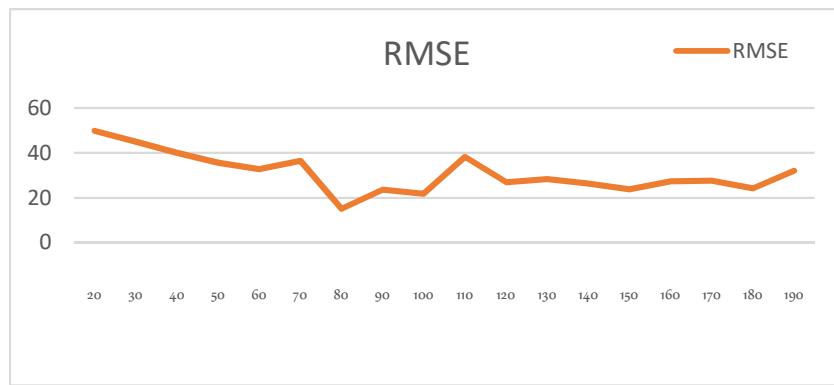
برای تنظیم پارامترها از روش یکی متغیر بقیه ثابت استفاده شده است. سه متغیر خیلی مهم در این مساله عبارتند از:

- تعداد واحدهای هر LSTM یا GRU
- نسبت Drop Out
- اندازه Batch Size

برای پیدا کردن مقدار بهینه اندازه واحد ها در هر لایه GRU یا LSTM بقیه پارامترها را ثابت گرفته و مقدار این متغیر را از ۲۰ تا ۱۸۰ تغییر دادیم. این کار را در دو یکی از شبکه های LSTM و یک شبکه GRU انجام دادیم (به علت محدودیتهای GPU) و نتیجه بهینه را که ۸۰ واحد بود به همه شبکه ها تعمیم دادیم. نتیج بدست آمده در جدولها و نمودارهای زیر علت انتخاب عدد ۸۰ را مشخص می کنند.

جدول ۱- مقدار خطای نسبت به واحدهای LSTM در شبکه اول

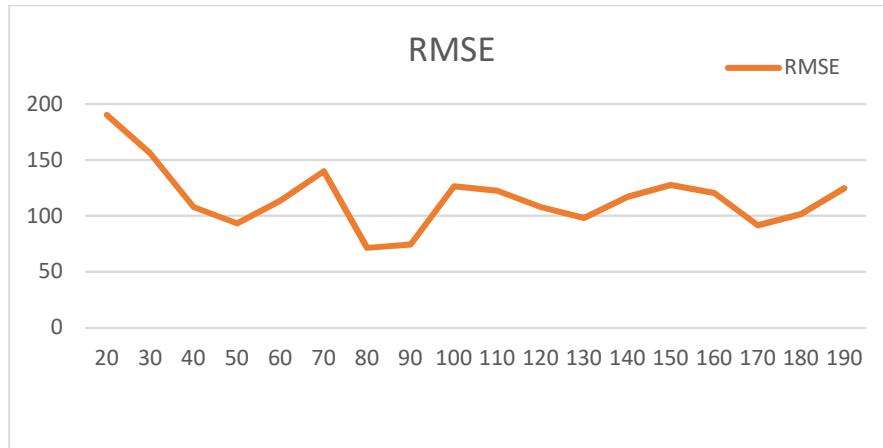
تعداد واحدهای هر (LSTM1) LSTM	میانگین RMSE در ۵ اجرا
20	49.84
30	45.00
40	40.122
50	35.64
60	32.79
70	36.55
80	15.14
90	23.65
100	21.87
110	38.22
120	27.00
130	28.42
140	26.37
150	23.88
160	27.42
170	27.63
180	24.23
190	32.02



شکل ۱- مقدار خطا نسبت به تعداد واحدهای LSTM مقدار بهینه در ۸۰ است

جدول ۲- مقدار خطا نسبت به واحدهای GRU (GRU2) در شبکه سوم

تعداد واحدهای هر GRU (GRU2)	میانگین RMSE در ۱۵ اجرا
20	190.24
30	155.57
40	107.96
50	93.19
60	113.65
70	139.77
80	71.363
90	74.37
100	126.56
110	122.39
120	107.97
130	98.23
140	116.92
150	127.53
160	120.31
170	91.642
180	101.44
190	124.83

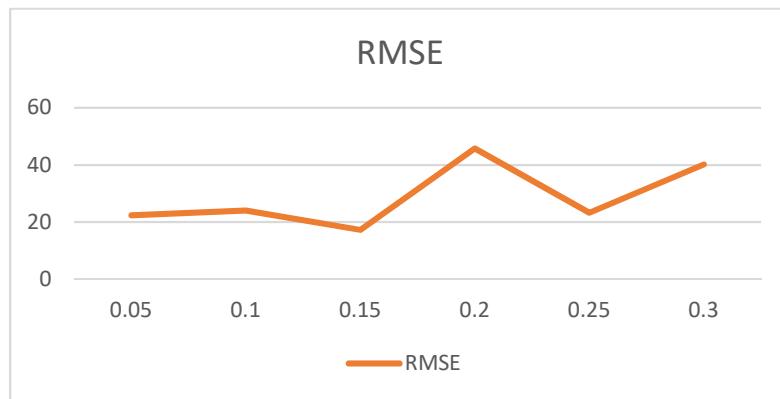


شکل ۲- مقدار خطأ نسبت به تعداد واحدها در GRU- مقدار بهینه در ۸۰ است

Drop Out نسبت

این متغیر نیز در شبکه ها متغیر مهمی است و برای جلوگیری از بیش برازش است. نتایج بدست آمده از تنظیم این متغیر در جدول زیر برای یکی از شبکه ها دیده می شود. مقدار بهینه ۱۵،۰ است که در همه شبکه ها از این نسبت استفاده شده است.

جدول ۳- تغییرات خطأ نسبت به تغییرات نسبت Drop Out	
dropout	درصد میانگین RMSE در ۱۵ اجرا
0.05	22.39
0.1	24.01
0.15	17.24
0.2	45.77
0.25	23.22
0.3	40.20

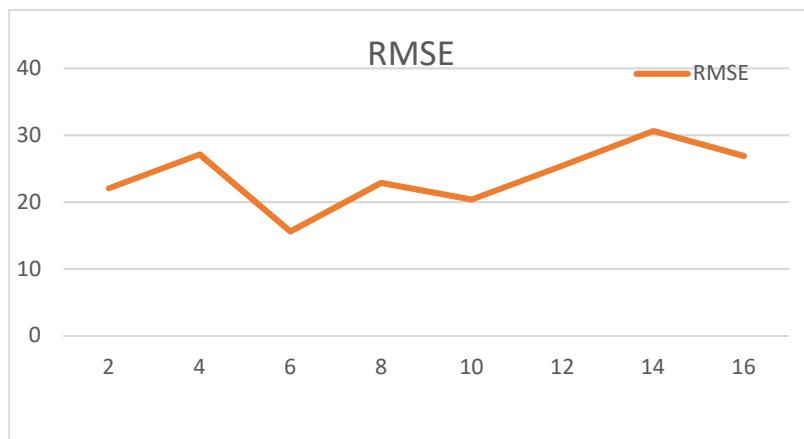


شکل ۳- تغییرات خطأ نسبت به تغییرات نسبت Drop Out- در نمودار مقدار بهینه ۱۵،۰ است

تنظیم Batch Size

این پارامتر هم در هنگام یادگیری از پارامترهای مهم است. مقدار این پارامتر از ۲ تا ۱۸ تغییر داده شده است و بهینه مقدار batch size در عدد ۸ طبق جدول و نمودار زیر اتفاق افتاده است.

جدول ۴- تغییرات مقدار خطأ با تغییر Batch Size	
Batch Size	میانگین RMSE در ۵ اجرا
2	28.26
4	22.07
6	27.14
8	15.61
10	22.88
12	20.38
14	25.44
16	30.63
18	26.87



شکل ۴- تغییرات خطأ نسبت به تغییرات Batch Size- مقدار بهینه عدد ۸ است.

در نتیجه بر اساس بررسی انجام شده برای همه شبکه ها مقدار پارامتر ها به صورت زیر انتخاب و استفاده شده است.

Lstm_Gru_Units=80

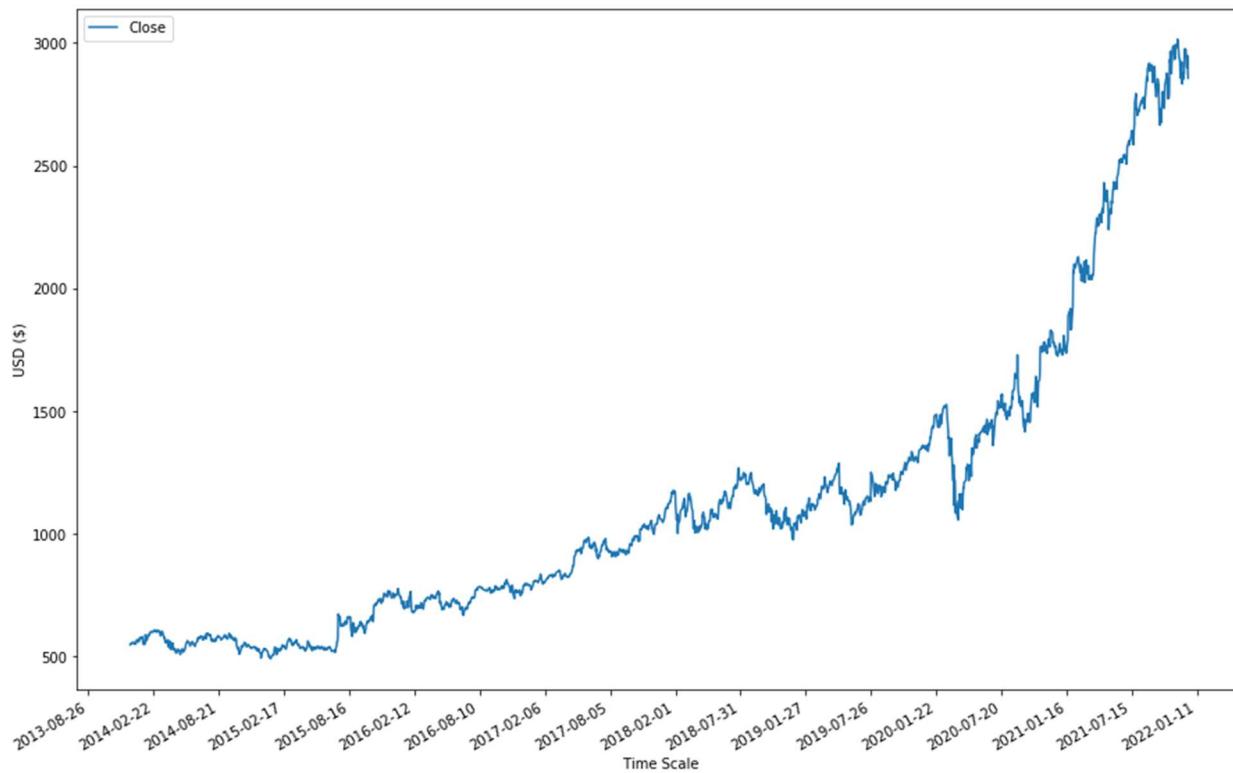
Batch Size=8

DropOut=0.15

نتایج شبکه ها و تحلیل و بررسی آنها:

داده ها

داده های استفاده شده در این پروژه قیمت در بورس در هنگام بسته شدن است. این داده ها از سال ۲۰۱۲ تا ۲۰۲۲ بصورت زمانی جمع اوری شده است. نمودار زمانی آن به صورت شکل زیر است:

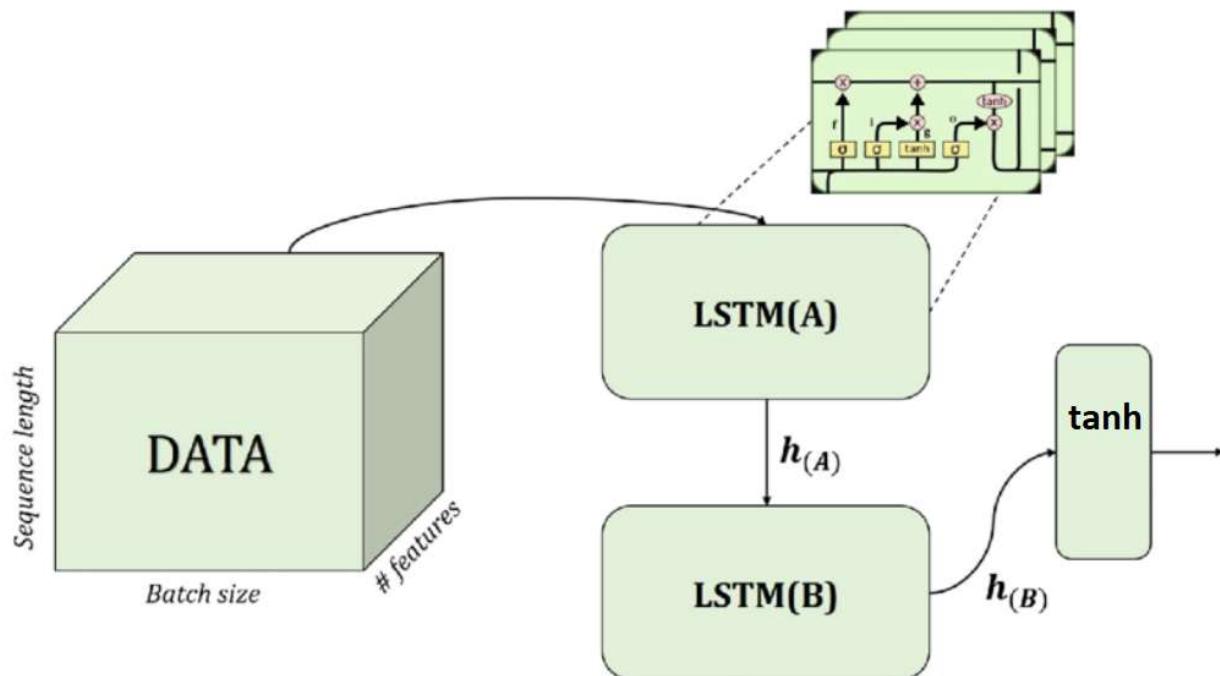


هدف از انجام این پروژه این است که با استفاده از داده های قبلی قیمت در روزهای بعدی پیش بینی شود. در این پروژه باید با استفاده از داده های $30 \times n$ روز گذشته قیمت روز بعدی پیش بینی شود. بنابراین اولین کار این است که داده های بازاری شود. یعنی اطلاعات هر سی روز به عنوان ورودی و اطلاعات روز بعدی به عنوان خروجی در نظر گرفته شود. در نهایت ما یک ماتریس $1 \times 30 \times n$ خواهیم داشت که n تعداد داده های ۳ی روزه است و چون فقط قیمت زمان بسته شدن را داریم بعد سوم یک است. همچنین یک ماتریس $1 \times n$ داریم که قیمت در روز سی ام است که خروجی یا برچسب ها هستند. چون داده ها نیز به صورت سری زمانی است باید یک قسمت از داده ها را برای آموزش انتخاب کنیم که ۸۰ درصد داده ها را برای آموزش انتخاب می کنیم و ۲۰ درصد باقی مانده را برای تست کردن سیستم کنار می گذاریم.

شبکه

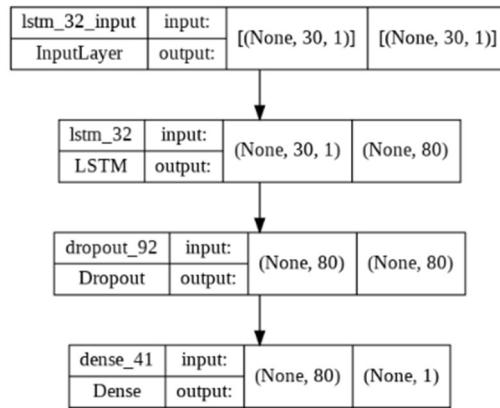
برای انجام پیش بینی از روی داده های سری زمانی باید از شبکه هایی استفاده کرد که دارای حافظه باشند زیرا به ذخیره حالت های قبلی داده ها نیاز داریم. یکی از شبکه هایی که دارای حافظه هستند شبکه های LSTM می باشند. ما از ساختار شکل زیر برای پیاه سازی پیش بینی داده های بورس استفاده کردیم. این ساختار به صورت پشته ای از شبکه های LSTM است. ما سه ساختار شبکه ایجاد کردیم که در LSTM1 یک لایه شبکه lstm وجود دارد در LSTM2 دو لایه شبکه lstm بصورت پشته ای استفاده کردیم و در مدل LSTM3 از سه لایه شبکه lstm استفاده کردیم.

همچنین ورودی شبکه های LSTM باید به شکل سه بعدی باشد که با استفاده از یک پنجره که در این مساله اندازه پنجره ۳۰ روز بود و چون فقط هدف پیش بینی قیمت در زمان بسته شدن (close) بود داده ها به صورت ماتریس $n \times 30 \times 1$ تبدیل شدند که n هم تعداد داده ها یا نمونه ها می باشد.



الف) نتایج حاصل برای شبکه LSTM

۱- شبکه اول پیاده سازی شده دارای یک لایه LSTM است ساختار شبکه به شکل زیر است:

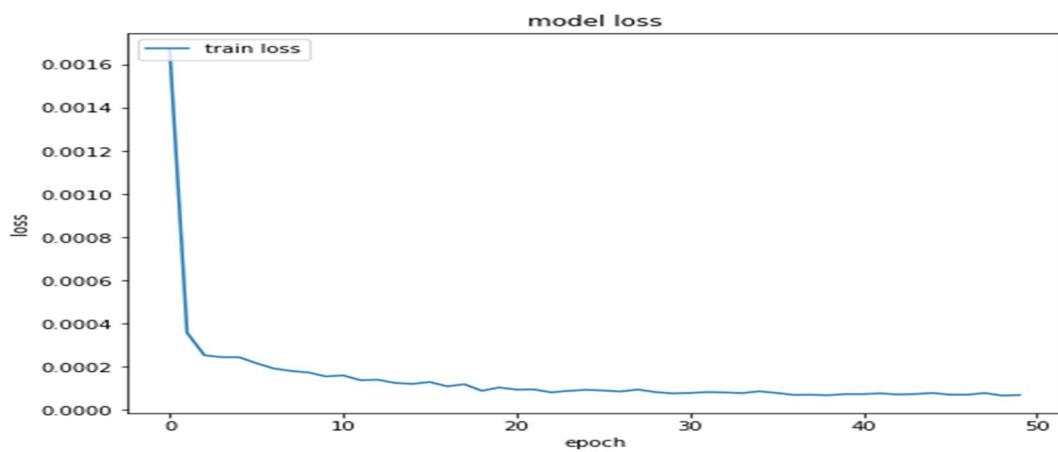


شکل ۱- ساختار شبکه اول LSTM

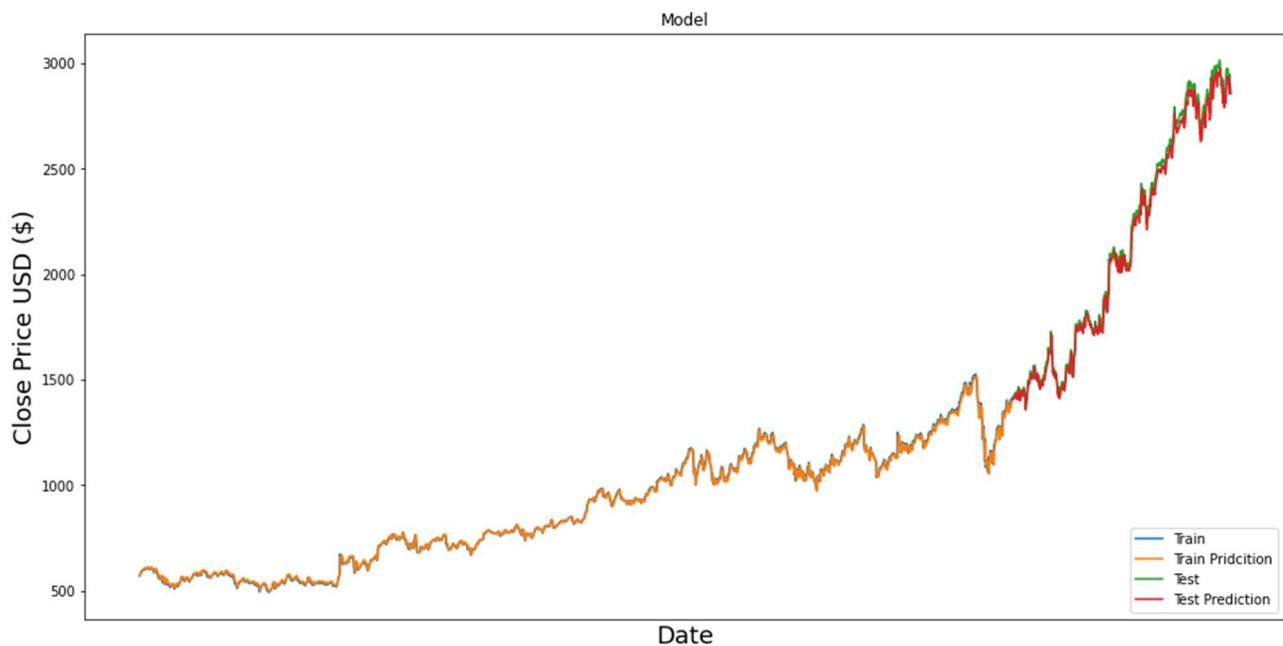
برای ارزیابی شبکه از دو معیار میانگین مربع خطأ (RMSE) و همچنین درصد میانگین مطلق خطأ (MAPE) استفاده کردیم. البته با توجه به این که داده ها به صورت نرمال شده وجود دارند در حال نرمال نیز RMSE را حساب کرده ایم. از طرف دیگر با توجه به این که مقدار دهی اولیه شبکه ها تصادفی است ممکن است ذر هر اجرا یک جواب متفاوت داشته باشیم بنابراین ما ۱۰ بار شبکه را آموزش دادیم و میانگین نتایج ۱۰ بار اجرا و بهترین را در نظر گرفته ایم. جدول ۱ نتایج ۱۰ بار اجرا و میانگین معیارها در ۱۰ بار اجرا و بهترین نتیجه را نشان می دهد.

جدول ۱- نتایج حاصل از اجرای ۱۰ بار شبکه اول LSTM

# اجرا	Normalize RMSE	RMSE	MAPE
1	0.005	12.80	0.005
2	0.004	11.66	0.006
3	0.003	8.74	0.003
4	0.002	7.048	0.003
5	0.017	43.48	0.022
6	0.002	6.04	0.003
7	0.013	34.29	0.016
8	0.016	41.14	0.022
9	0.003	7.79	0.003
10	0.010	25.58	0.013
MEAN	0.007	19.86	0.010
Best	0.003	6.04	0.003

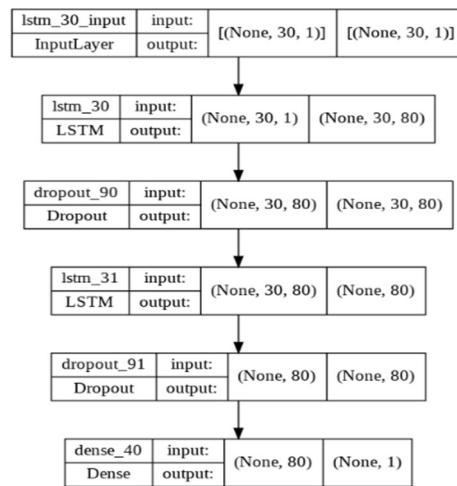


شکل ۲- روند همگرایی مدل اول LSTM



شکل ۳- نمودار مقدار تخمین نسبت به مقدار واقعی در داده های آموزشی و تست در مدل اول LSTM

-۲- شبکه دوم پیاده سازی شده دارای دو لایه LSTM است ساختار شبکه به شکل زیر است:

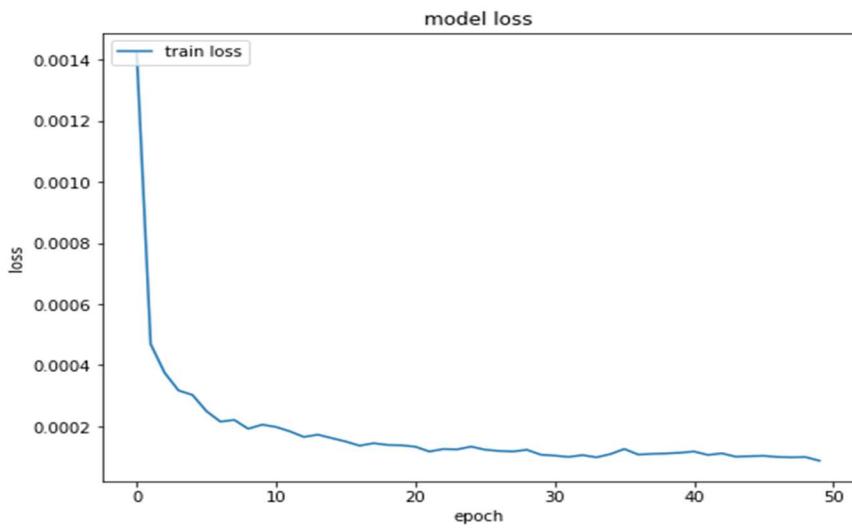


شکل ۴- ساختار شبکه دوم

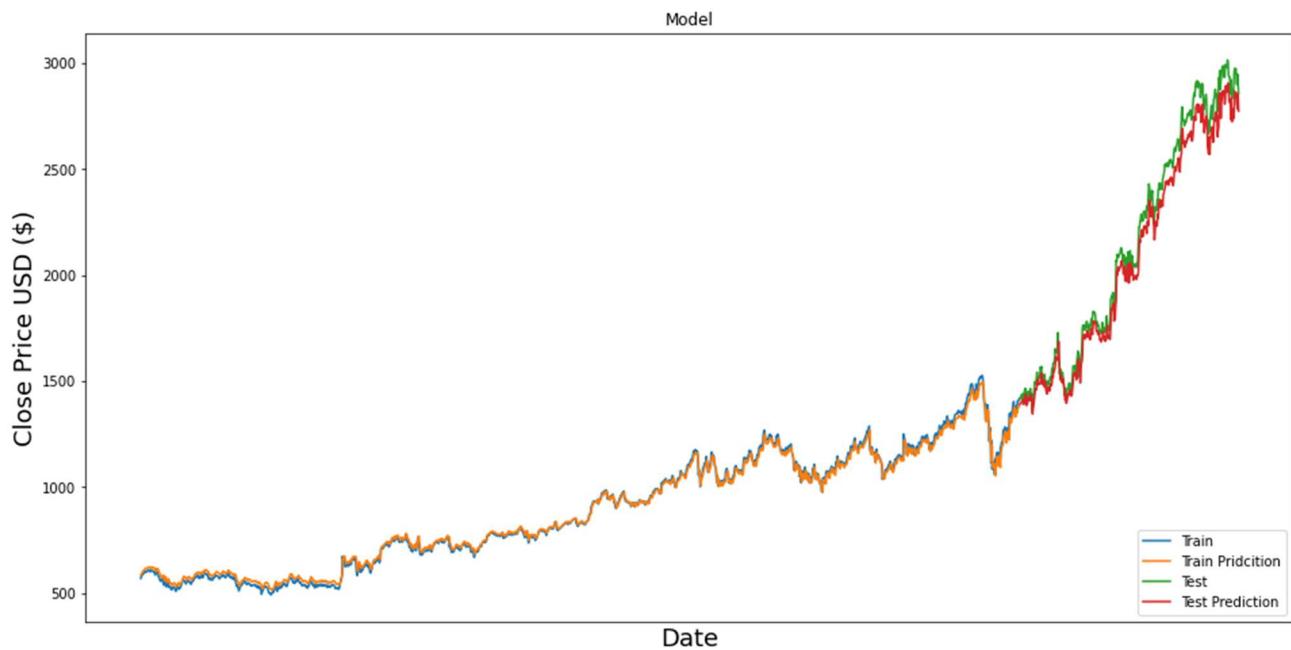
نتایج حاصل از اجرای ده بار این شبکه در جدول ۲ نشان داده شده است.

جدول ۲- نتایج حاصل از اجرای ۱۰ بار شبکه دوم

# اجرا	Normalize RMSE	RMSE	MAPE
1	0.004	11.69	0.003
2	0.020	52.55	0.026
3	0.007	17.92	0.009
4	0.006	15.52	0.005
5	0.005	14.69	0.005
6	0.009	24.52	0.012
7	0.004	10.56	0.004
8	0.022	56.55	0.025
9	0.008	21.92	0.011
10	0.028	71.07	0.036
MEAN	0.011	29.70	0.014
BEST	0.004	10.56	0.004

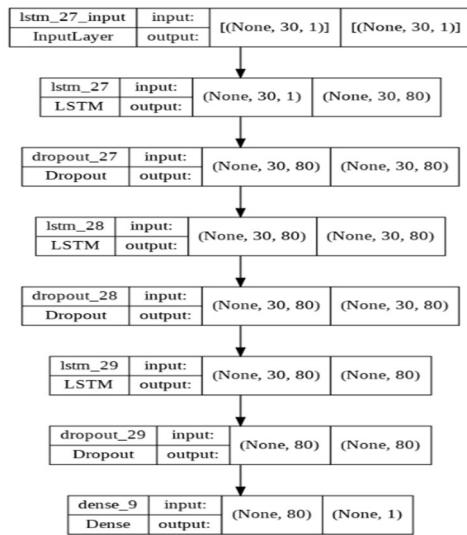


شکل ۵- روند همگرایی مدل دوم LSTM



شکل ۶- نمودار مقدار تخمین نسبت به مقدار واقعی در داده های آموزشی و تست در مدل اول LSTM

-۳ شبکه سوم پیاده سازی شده دارای سه لایه LSTM است ساختار شبکه به شکل زیر است:

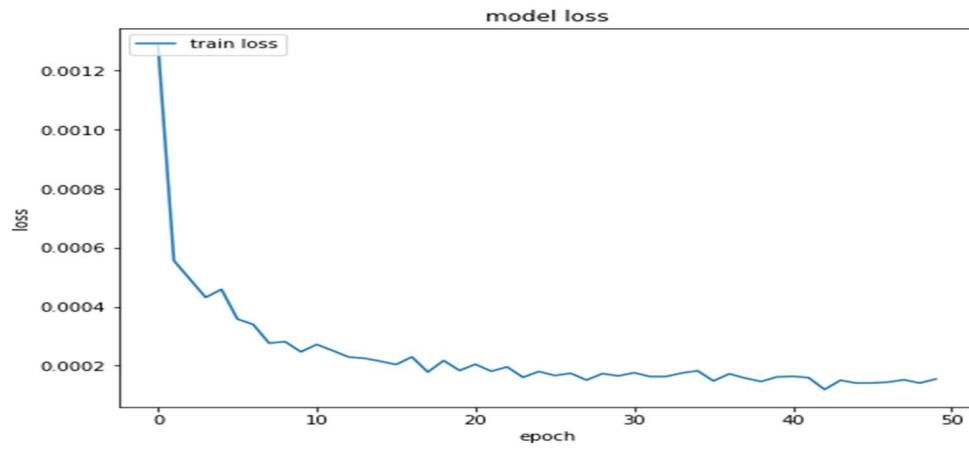


شکل ۷- ساختار شبکه سوم

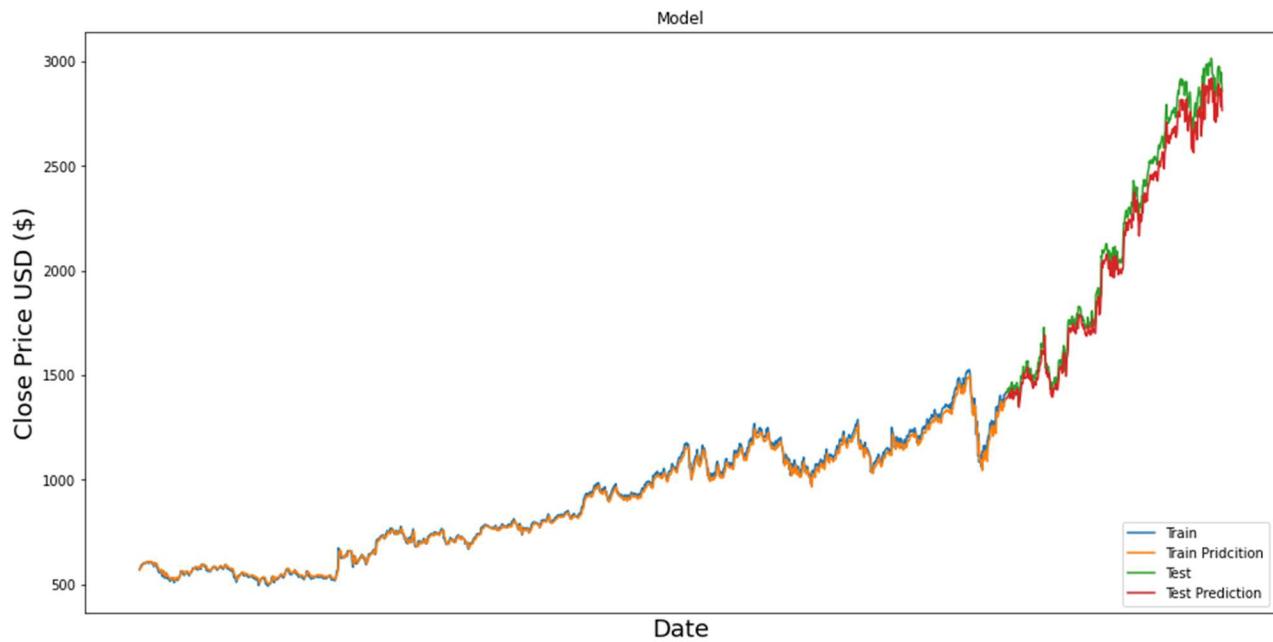
نتایج حاصل از اجرای ۱۰ با این شبکه در جدول ۳ نشان داده شده است.

جدول ۳- نتایج حاصل از اجرای ۱۰ بار شبکه سوم LSTM

# اجرا	Normalize RMSE	RMSE	MAPE
1	0.018	46.05	0.021
2	0.020	51.83	0.030
3	0.017	44.09	0.018
4	0.028	70.94	0.039
5	0.057	145.54	0.084
6	0.010	26.21	0.009
7	0.046	116.444	0.062
8	0.018	45.97	0.015
9	0.011	29.35	0.018
10	0.025	65.53	0.033
MEAN	0.025	64.20	0.033
BEST	0.010	26.21	0.009



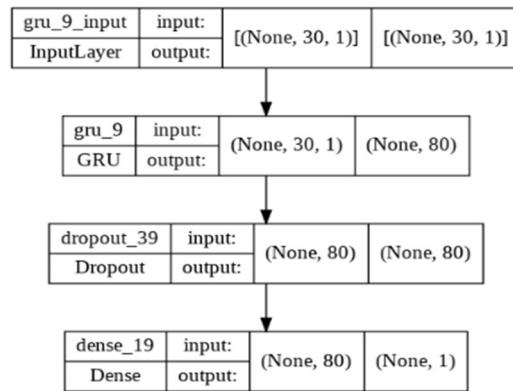
شکل ۸- روند همگرایی مدل سوم LSTM



شکل ۹- نمودار مقدار تخمین نسبت به مقدار واقعی در داده های آموزشی و تست در مدل سوم LSTM

ب) مدل GRU

۱- شبکه اول پیاده سازی شده دارای یک لایه GRU است ساختار شبکه به شکل زیر است:

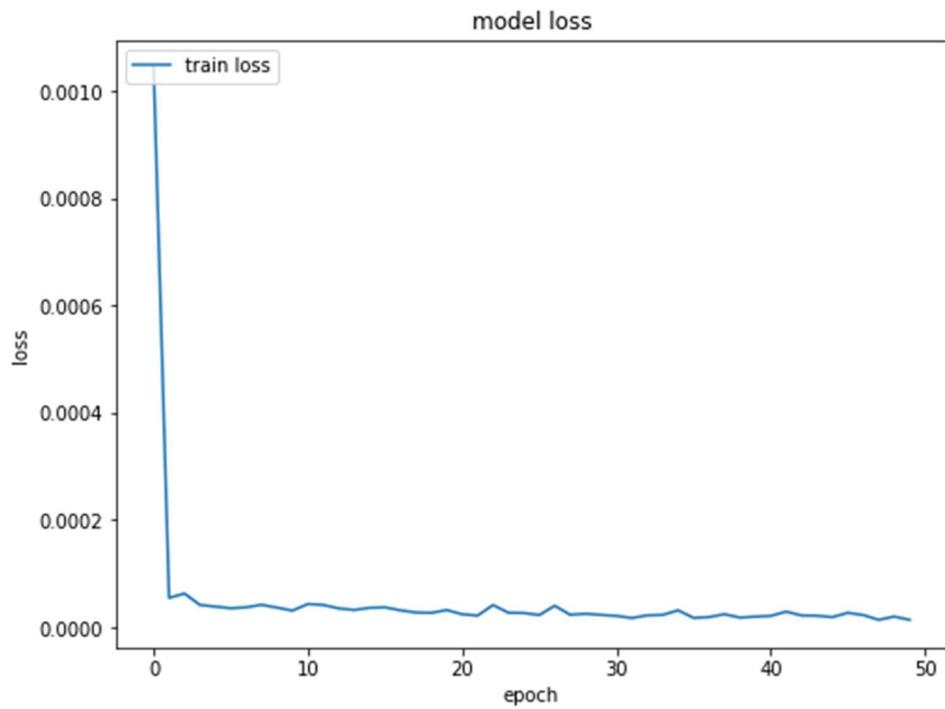


شکل ۱۰- ساختار شبکه اول GRU

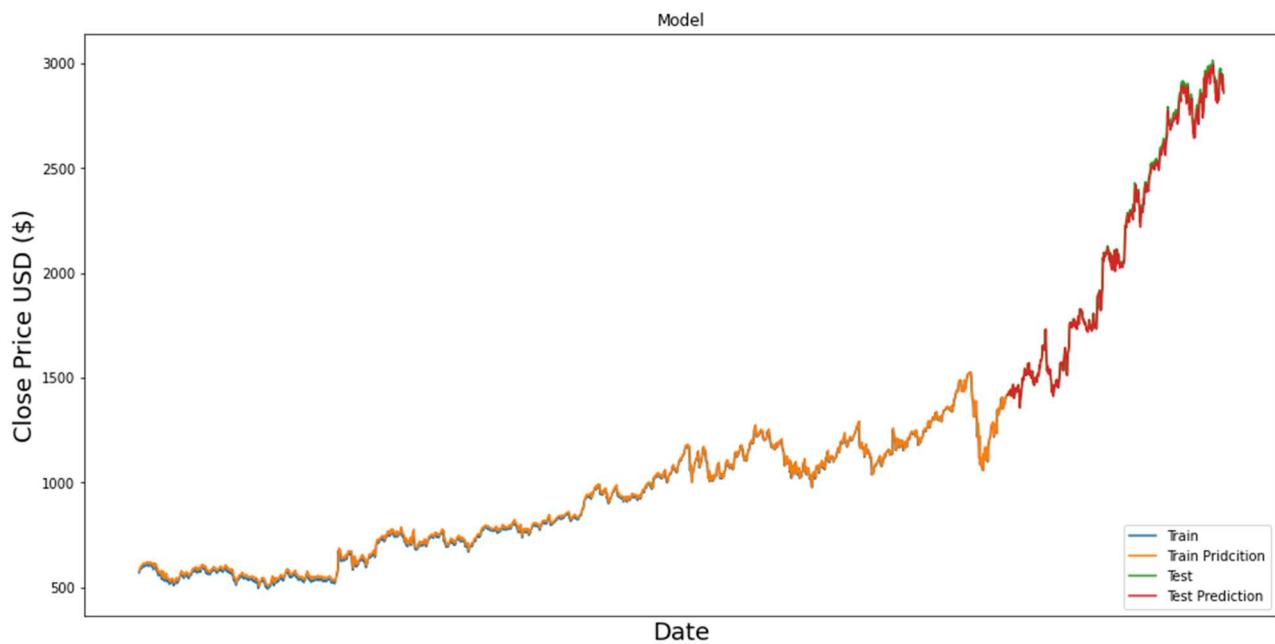
نتایج حاصل از ۱۰ با اجرای این شبکه در جدول ۴ نشان داده شده است

جدول ۴- نتایج حاصل از اجرای ۱۰ بار شبکه اول GRU

# اجرا	Normalize RMSE	RMSE	MAPE
1	0.019	49.24	0.026
2	0.003	9.14	0.004
3	0.006	16.52	0.009
4	0.009	22.75	0.011
5	0.010	25.42	0.012
6	0.011	27.82	0.015
7	0.013	32.80	0.015
8	0.013	35.30	0.018
9	0.007	18.29	0.007
10	0.005	13.70	0.005
MEAN	0.009	25.10	0.012
BEST	0.003	9.14	0.004

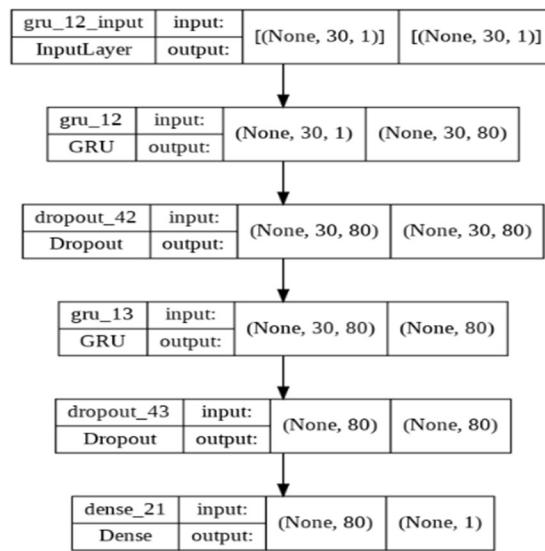


شکل ۱۱ - روند همگرایی مدل اول GRU



شکل ۱۲ - نمودار مقدار تخمین نسبت به مقدار واقعی در داده های آموزشی و تست در مدل اول GRU

۲- شبکه دوم پیاده سازی شده دارای دو لایه GRU است ساختار شبکه به شکل زیر است:

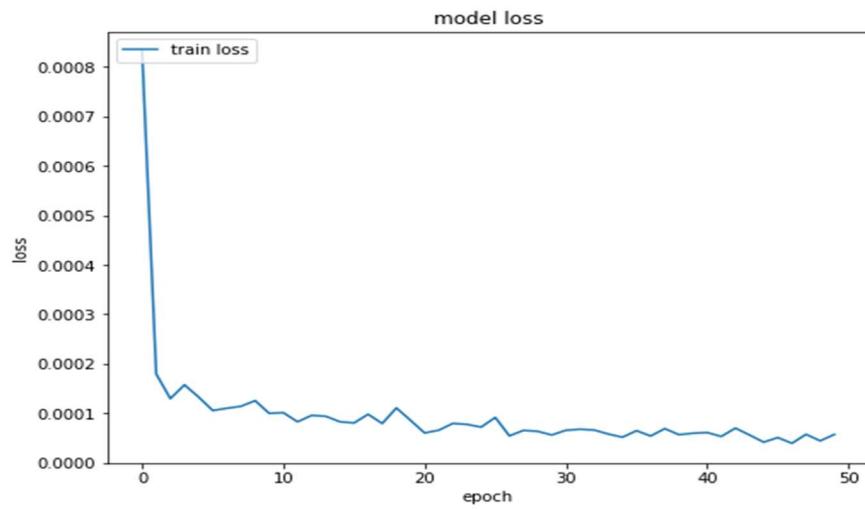


شکل ۱۳- ساختار شبکه دوم GRU

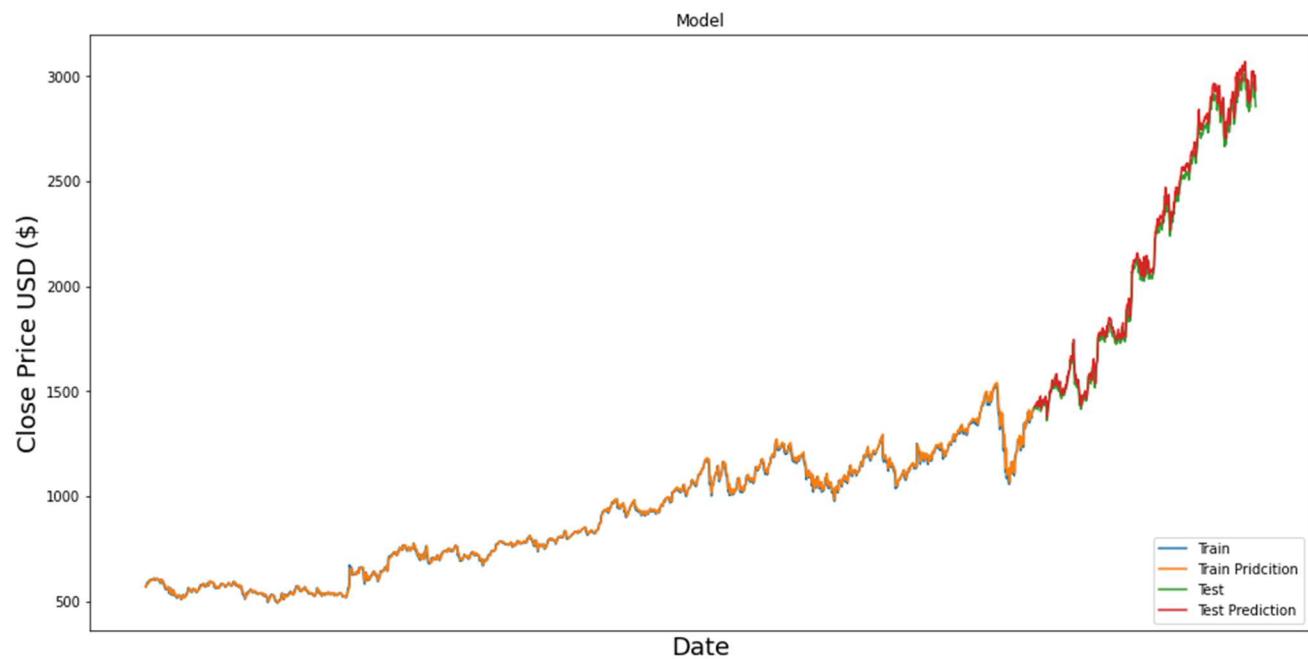
نتایج حاصل از ۱۰ بار اجرای شبکه در جدول ۵ نشان داده شده است.

جدول ۵- نتایج حاصل از اجرای ۱۰ بار شبکه دوم GRU

# اجرا	Normalize RMSE	RMSE	MAPE
1	0.031	79.79	0.044
2	0.024	60.64	0.030
3	0.006	16.98	16.98
4	0.017	43.24	0.023
5	0.004	12.36	0.004
6	0.017	44.69	0.018
7	0.023	58.92	0.030
8	0.002	6.09	0.002
9	0.011	29.88	0.016
10	0.012	32.21	0.016
MEAN	0.015	38.48	0.019
BEST	0.002	6.09	0.002

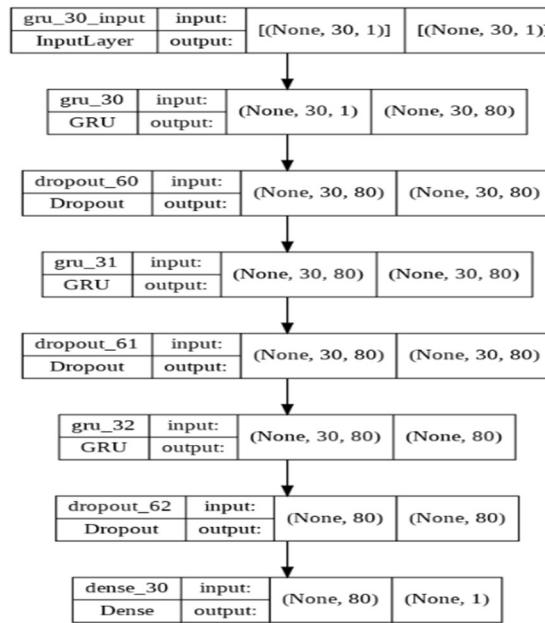


شکل ۱۴ - روند همگرایی مدل دوم GRU



شکل ۱۵ - نمودار مقدار تخمین نسبت به مقدار واقعی در داده های آموزشی و تست در مدل دوم GRU

۳- شبکه سوم پیاده سازی شده دارای سه لایه GRU است ساختار شبکه به شکل زیر است:

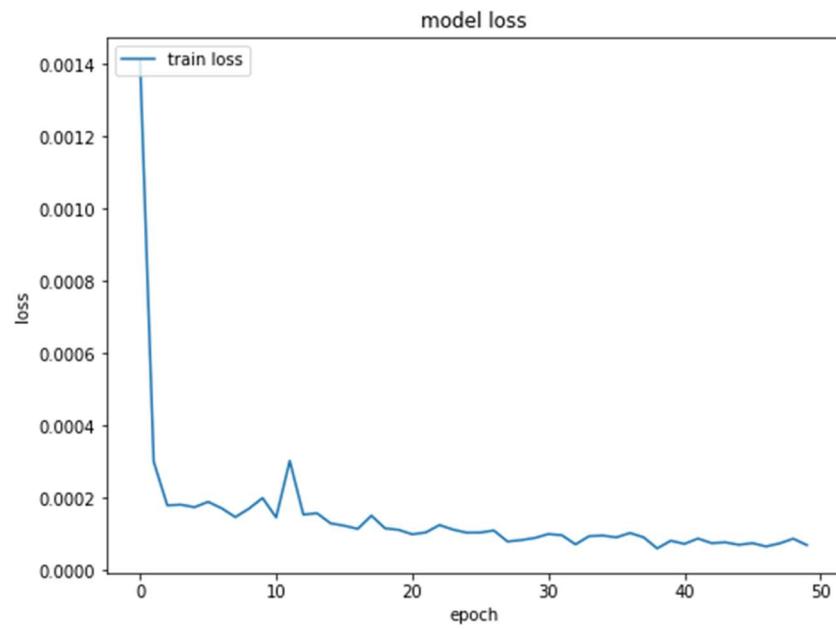


شکل ۱۶- ساختار شبکه سوم GRU

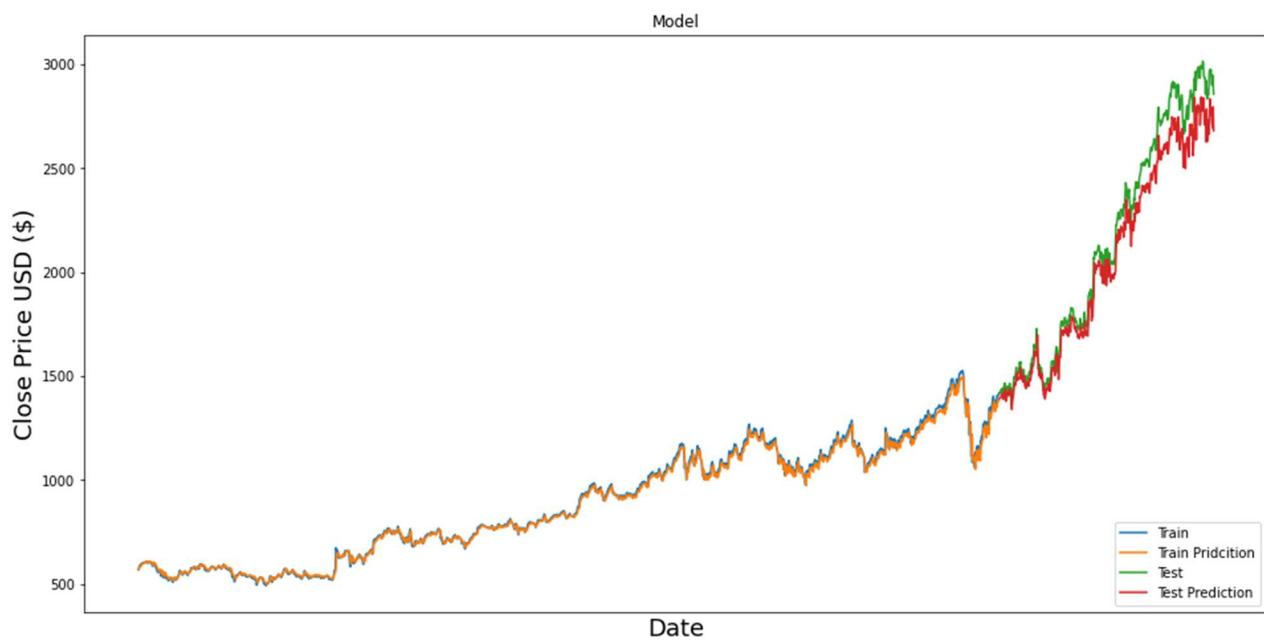
نتایج حاصل از ۱۰ بار اجرای این شبکه در جدول ۶ نشان داده شده است.

جدول ۶- نتایج حاصل از اجرای ۱۰ بار شبکه سوم GRU

# اجرا	Normalize RMSE	RMSE	MAPE
1	0.019	49.31	0.032
2	0.034	87.62	0.035
3	0.040	101.50	0.039
4	0.013	34.88	0.011
5	0.012	31.25	0.015
6	0.053	135.52	0.071
7	0.025	65.33	0.033
8	0.012	30.53	0.018
9	0.014	36.45	0.011
10	0.041	105.61	0.047
MEAN	0.026	67.80	0.031
BEST	0.012	31.25	0.015



شکل ۱۷ - روند همگرایی مدل سوم GRU



شکل ۱۸ - نمودار مقدار تخمین نسبت به مقدار واقعی در داده های آموزشی و تست در مدل سوم GRU

مقایسه مدل‌ها:

در جدول ۷ نتایج حاصل از ۳ مدل LSTM و سه مدل GRU نشان داده شده است. بر اساس این جدول مدل اول LSTM در بین مدل‌های LSTM بهترین نتیجه را هم از نظر بهترین شبکه وهم از نظر میانگین مقدارها دارد. در میان مدل‌های GRU نیز مدل اول GRU بهترین نتیجه را دارد. در مقایسه بین مدل‌های و LSTM مدل LSTM نتیجه بهتری دارد و سپس مدل اول GRU.

نتیجه‌ای که از این جدول حاصل می‌شود این است که در این مساله مدل‌های ساده تر جواب بهتری دارند و این می‌توان به این دلیل باشد که اولاً حج داده‌های ما کم است و ثانیاً تعداد ویژگی‌ها هم یکی است و بسیار داده‌ها ساده هستند. در صورت استفاده از ویژگی‌های بیشتر شاید مدل‌های پیچیده تر جواب بهتری داشته باشند

مدل	MEAN RMSE	BEST RMSE
LSTM1	19.86	6.04
LSTM2	64.20	26.21
LSTM3	64.20	26.21
GRU1	<u>25.10</u>	<u>9.14</u>
GRU2	38.48	6.09
GRU3	67.80	31.25