به نام خدا

دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

درس شبکه‌های عصبی و یادگیری عمیق

مینی پروژه دوم

نام و نام خانوادگی	مریم ریاضی
شماره دانشجویی	810197518
تاریخ ارسال گزارش	۱۰ خرداد

فهرست گزارش سوالات

سوال ۱ – Stock Market Prediction

هدف از این بخش از پروژه آشنایی و پیاده سازی شبکه های LSTM,GRU,RNN میباشد.

ایتدا کتابخانه های لازم لود میشوند و دیتا (google,apple) را باید با هم merge کنیم تا آن ها را با هم به شبکه بدهیم تا خروجی دو بعدی دریافت کنیم که یک بعد پیشبینی apple و یک بعد پیشبینی google را به ما نشان دهد (پیشبینی برای ستون close انجام میشود).در نتیجه در ابتدا این دو دیتا را merge کردیم که با توجه به شکل دیتای close_y برای دیتا GOOG و دیتا closre_x برای دیتا APPL میباشد.

merged

	Date	High_x	Low_x	Open_x	Close_x	Volume_x	Adj Close_x	High_y	Low_y	Open_y	Close_y	Volume_y	Adj Close_y
0	2010-01-04	30.642857	30.340000	30.490000	30.572857	123432400.0	26.601469	313.579620	310.954468	312.304413	312.204773	3927000.0	312.204773
1	2010-01-05	30.798571	30.464285	30.657143	30.625713	150476200.0	26.647457	312.747742	309.609497	312.418976	310.829926	6031900.0	310.829926
2	2010-01-06	30.747143	30.107143	30.625713	30.138571	138040000.0	26.223597	311.761444	302.047852	311.761444	302.994293	7987100.0	302.994293
3	2010-01-07	30.285715	29.864286	30.250000	30.082857	119282800.0	26.175119	303.861053	295.218445	303.562164	295.940735	12876600.0	295.940735
4	2010-01-08	30.285715	29.865715	30.042856	30.282858	111902700.0	26.349140	300.498657	293.455048	294.894653	299.885956	9483900.0	299.885956
...	...	...	...	...	...	...	...	...	...	...	...	...	...
2259	2018-12-24	151.550003	146.589996	148.149994	146.830002	37169200.0	144.656540	1003.539978	970.109985	973.900024	976.219971	1590300.0	976.219971
2260	2018-12-26	157.229996	146.720001	148.300003	157.169998	58582500.0	154.843475	1040.000000	983.000000	989.010010	1039.459961	2373300.0	1039.459961
2261	2018-12-27	156.770004	150.070007	155.839996	156.149994	53117100.0	153.838562	1043.890015	997.000000	1017.150024	1043.880005	2109800.0	1043.880005
2262	2018-12-28	158.520004	154.550003	157.500000	156.229996	42291400.0	153.917389	1055.560059	1033.099976	1049.619995	1037.079956	1414800.0	1037.079956
2263	2018-12-31	159.360001	156.479996	158.529999	157.740005	35003500.0	155.405045	1052.699951	1023.590027	1050.959961	1035.609985	1493300.0	1035.609985

2264 rows × 13 columns

همانطور که میدانیم ابتدا لازم است تا بر روی دیتا ها preprocessing انجام شود برای این کار ابتدا بررسی میکنیم که آیا داده ای هست که nan باشد که با توجه به شکل ۱ دیدیم که داده ای به این صورت نیست سپس با استفاده از کد شکل ۲ داده ها را بین ۰ و ۱ scale کردیم.

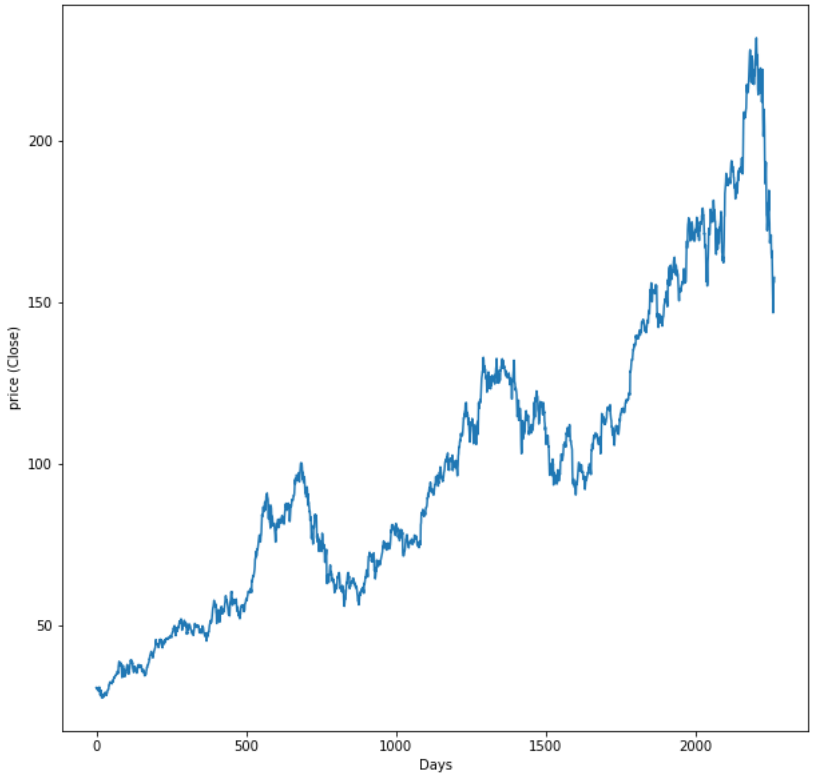
```
[ ] merged.isna().sum()

Date          0
High_x        0
Low_x         0
Open_x        0
Close_x       0
Volume_x      0
Adj Close_x   0
High_y        0
Low_y         0
Open_y        0
Close_y       0
Volume_y      0
Adj Close_y   0
dtype: int64
```

شکل ۱ : چک کردن داده ها

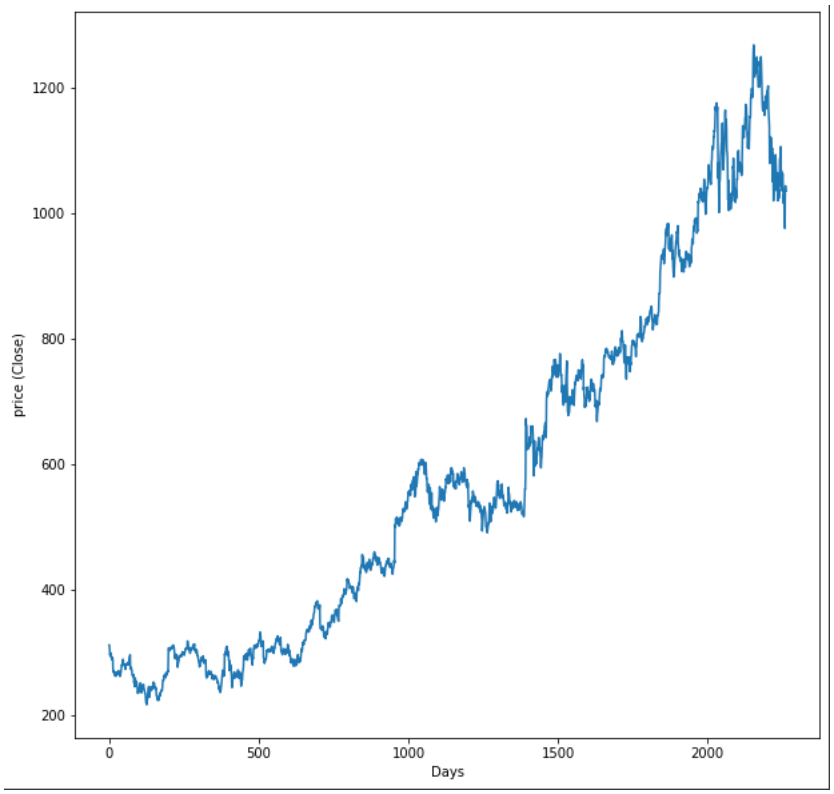
```
] from sklearn import preprocessing

  scaler = preprocessing.MinMaxScaler(copy=True, feature_range=(0, 1))
  #select only numeric columns
  X = scaler.fit_transform(merged.select_dtypes(np.number))
  print (X)
```

شکل ۲: کد استفاده شده برای scale کردن داده ها بین ۰ و ۱

مقادیر close به صورت جداگانه برای دو دیتا پلات شده اند



شکل ۳ : نمودار دیتای close برای دیتا apple



شکل 4 : نمودار دیتای close برای دیتا googles

سپس باید داده کل را با پنجره های زمانی ۳۰ روز ایجاد کنیم/ برای ایجاد این پنجره های زمانی از تابع شکل ۵ استفاده کردم :

```
[ ]  def data_for_training(data_s1, period1):
        a1 = np.arange(len(data_s1) - period1 + 1)
        data_list1 = []
        for i in range(len(a1)):
            sample_list = []
            for j in range(period1):
                value1 = data_s1[i + j]
                sample_list.append(value1)
            data_list1.append(sample_list)
        data_list1 = np.array(data_list1)
        return data_list1
```

شکل ۵ : تابع برای جدا کردن پنجره های زمانی

```
[ ]  period = 30 ##30 days
     data_list = data_for_training(X, period)
     print(data_list.shape)
     data_list[0].shape

     (2235, 30, 12)
     (30, 12)
```

شکل ۶: پیاده سازی تابع شکل ۵ و دیدن shape خروجی (جدا شدن پنجره های زمانی ۳۰ روزه)

در بخش split نیز ۱۰ درصد از داده برای تست جدا شده اند .

الف)

اول شبکه LSTM را بررسی میکنیم:

اول شبکه را پیاده سازی میکنیم و مدل را ترین میکنیم جزییات در شکل ۷ نشان داده شده است.

(در مدل طراحی شده زیر batch ها مختلف تابع های activation مختلف و... امتحان شدند تا به بهترین حالت مدل رسیدم که در شکل ۷ نشان داده شده است.)

```
[ ]  model = Sequential()
     model.add(LSTM(units=50, return_sequences=True, input_shape=(x_train.shape[1],12)))

[ ]  model.add(Dropout(0.2))

[ ]  model.add(LSTM(units = 50,return_sequences = True))
     model.add(Dropout(0.2))

[ ]  model.add(LSTM(units = 50,return_sequences = True))
     model.add(Dropout(0.2))

[ ]  model.add(LSTM(units = 50))
     model.add(Dropout(0.2))

[ ]  model.add(Dense(units = 2))

[ ]
     model.compile(optimizer = 'adam',loss = 'mean_squared_error',metrics = ["accuracy"])

     model.fit(x_train,y_train,epochs = 50, batch_size = 32)
```

شکل ۷ – مدل LTSM و آموزش آن

```
Epoch 40/50
56/56 [==============================] - 3s 59ms/step - loss: 0.0042 - accuracy: 0.7500
Epoch 41/50
56/56 [==============================] - 3s 59ms/step - loss: 0.0040 - accuracy: 0.7461
Epoch 42/50
56/56 [==============================] - 3s 59ms/step - loss: 0.0039 - accuracy: 0.7338
Epoch 43/50
56/56 [==============================] - 3s 58ms/step - loss: 0.0041 - accuracy: 0.7438
Epoch 44/50
56/56 [==============================] - 3s 61ms/step - loss: 0.0042 - accuracy: 0.7377
Epoch 45/50
56/56 [==============================] - 3s 60ms/step - loss: 0.0041 - accuracy: 0.7589
Epoch 46/50
56/56 [==============================] - 3s 60ms/step - loss: 0.0039 - accuracy: 0.7483
Epoch 47/50
56/56 [==============================] - 3s 60ms/step - loss: 0.0040 - accuracy: 0.7472
Epoch 48/50
56/56 [==============================] - 3s 59ms/step - loss: 0.0039 - accuracy: 0.7427
Epoch 49/50
56/56 [==============================] - 3s 59ms/step - loss: 0.0041 - accuracy: 0.7455
Epoch 50/50
56/56 [==============================] - 3s 58ms/step - loss: 0.0038 - accuracy: 0.7517
<keras.callbacks.History at 0x7fc4b1f069d0>
```

شکل ۸ – نتایج آموزش مدل بعد از ۵۰ ایپاک

```
]
  model.compile(optimizer = 'adam',loss = 'mean_squared_error',metrics = ["accuracy"])

  model.fit(x_train,y_train,epochs = 50, batch_size = 32)
```

شکل ۹ – compile , fit کردن مدل

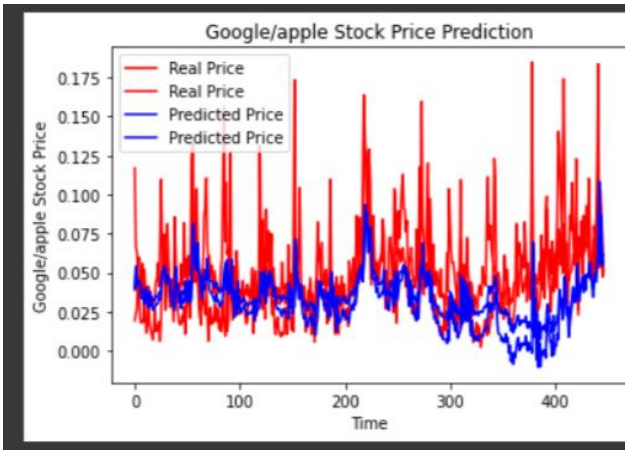
در این مدل از دو لایه LSTM استفاده شده که هر کدام ۵۰ یونیت دارند . در آخر یک لایه dense با دو یونیت قرار داده شده است برای گرفتن خروجی از دو شرکت و تابع loss را mean squared error در نظر گرفتیم.

زمان اجرا برای ۵۰ ایپاک :۱۰ میکرو ثانیه

کمترین خطا : ۰,۰۰۳

دقت مدل ۷۳٪

نمودار مقدار پیشبینی شده و مقدار واقعی برای ده درصد روزهای آخر:



شکل ۱۰ – مقدار پیشبینی شده و مقدار واقعی برای LSTM

همانطور که مشاهده می شود، خروجی نمودار از لحاظ trendشبیه می باشد و هم چنین با توجه به پلات به y=x نزدیک است که در نتیجه آن پیش بینی با دقت خوبی انجام شده است.

GRU :

```
from keras.layers import GRU
%time
model = Sequential()
model.add(GRU(50, batch_input_shape=(None, 29,12 ), recurrent_dropout=0))
model.add(Dense(2, activation='relu'))
model.compile(loss='mse', optimizer='adam', metrics=['mae'])
model.summary()
```

```
CPU times: user 5 µs, sys: 0 ns, total: 5 µs
Wall time: 9.06 µs
Model: "sequential_10"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 gru_8 (GRU)                 (None, 50)                9600

 dense_8 (Dense)             (None, 2)                 102

=================================================================
Total params: 9,702
Trainable params: 9,702
Non-trainable params: 0
```

شکل ۱۱ – مدل پیاده شده GRU

در این مدل از یک لایه GRU با ۵۰ یونیت و یک لایه dense با دو نورون خروجی در نظر گرفته شده است.

```
100] model.compile(optimizer = 'adam',loss = 'mean_squared_error',metrics = ["accuracy"])
```

تابع فعالساز relu و برای loss / mean squared error در نظر گرفته شده است.

```
Epoch 38/50
120/120 [==============================] - 2s 15ms/step - loss: 0.0055 - accuracy: 0.7958 - val_loss: 7.5884e-04 - val_accuracy: 0.5279
Epoch 39/50
120/120 [==============================] - 2s 15ms/step - loss: 0.0055 - accuracy: 0.7923 - val_loss: 0.0014 - val_accuracy: 0.4860
Epoch 40/50
120/120 [==============================] - 2s 15ms/step - loss: 0.0054 - accuracy: 0.7839 - val_loss: 7.6929e-04 - val_accuracy: 0.5670
Epoch 41/50
120/120 [==============================] - 2s 15ms/step - loss: 0.0055 - accuracy: 0.7874 - val_loss: 0.0011 - val_accuracy: 0.6508
Epoch 42/50
120/120 [==============================] - 2s 15ms/step - loss: 0.0055 - accuracy: 0.7825 - val_loss: 0.0010 - val_accuracy: 0.4358
Epoch 43/50
120/120 [==============================] - 2s 15ms/step - loss: 0.0055 - accuracy: 0.7867 - val_loss: 9.6443e-04 - val_accuracy: 0.7095
Epoch 44/50
120/120 [==============================] - 2s 15ms/step - loss: 0.0055 - accuracy: 0.7692 - val_loss: 7.7792e-04 - val_accuracy: 0.5000
Epoch 45/50
120/120 [==============================] - 2s 16ms/step - loss: 0.0055 - accuracy: 0.7804 - val_loss: 0.0020 - val_accuracy: 0.3603
Epoch 46/50
120/120 [==============================] - 2s 15ms/step - loss: 0.0054 - accuracy: 0.7965 - val_loss: 8.1060e-04 - val_accuracy: 0.7263
Epoch 47/50
120/120 [==============================] - 2s 15ms/step - loss: 0.0055 - accuracy: 0.7825 - val_loss: 0.0019 - val_accuracy: 0.3659
Epoch 48/50
120/120 [==============================] - 2s 15ms/step - loss: 0.0055 - accuracy: 0.7825 - val_loss: 0.0014 - val_accuracy: 0.3659
Epoch 49/50
120/120 [==============================] - 2s 15ms/step - loss: 0.0054 - accuracy: 0.7797 - val_loss: 0.0020 - val_accuracy: 0.4385
Epoch 50/50
120/120 [==============================] - 2s 15ms/step - loss: 0.0054 - accuracy: 0.7832 - val_loss: 7.3981e-04 - val_accuracy: 0.7123
```

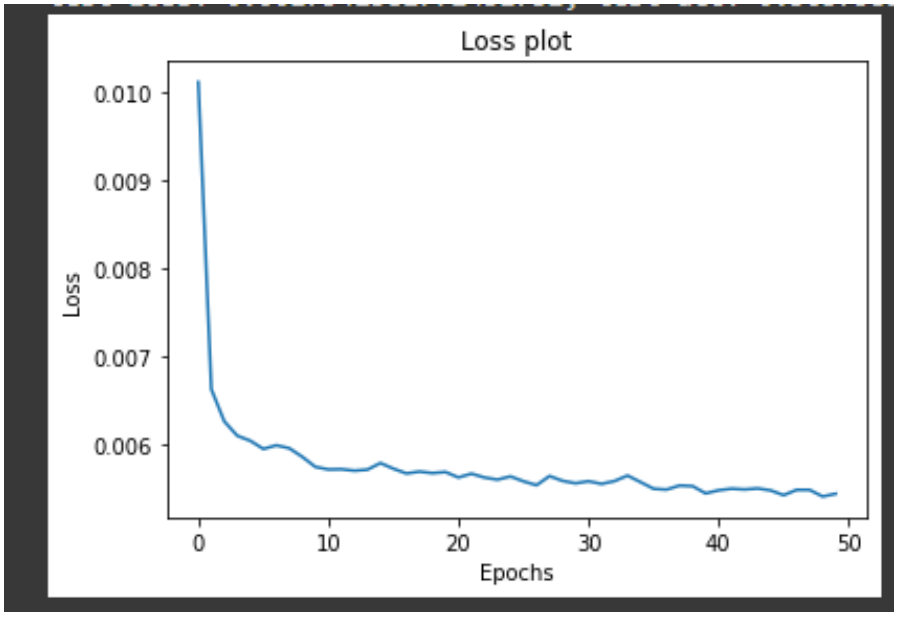
شکل ۱۲ – نتایج آموزش مدل بعد از ۵۰ ایپاک

زمان اجرا :

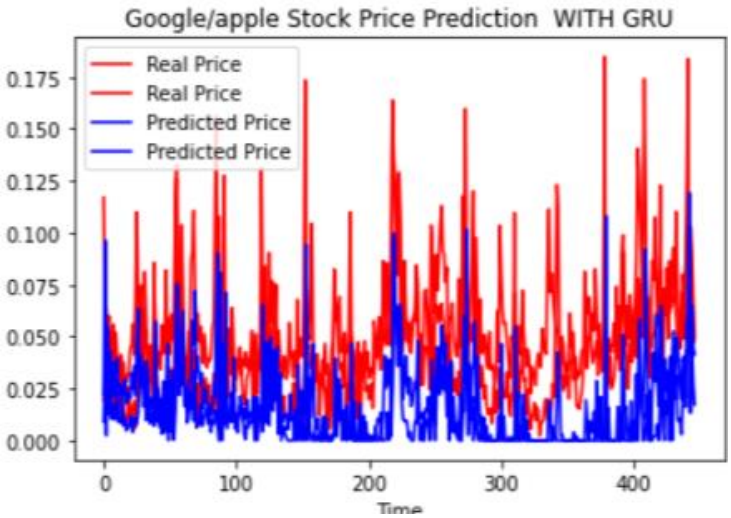
۹ میکرو ثانیه

کمترین loss:۰,۰۰۵

دقت مدل :۷۹٪



شکل ۱۳ – مقدار loss در GRU



شکل ۱۴ – مقدار پیشبینی شده و مقدار واقعی برای GRU

همانطور که مشخص است، پیشبینی GRUبهتر از LSTMعمل کرده است(دقت بیشتر و نمودار ها نشان دهنده این موضوع هستند)

برای RNN :

```
%time
model = Sequential()
model.add(SimpleRNN(units=100, input_shape=x_train.shape[1:], activation="relu", recurrent_dropout=0.0))
model.add(Dense(2, activation='relu'))
model.compile(loss='mse', optimizer='adam', metrics=['mae'])
model.summary()
```

```
CPU times: user 4 µs, sys: 0 ns, total: 4 µs
Wall time: 10.5 µs
Model: "sequential_11"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 simple_rnn (SimpleRNN)      (None, 100)               11300

 dense_9 (Dense)             (None, 2)                 202

=================================================================
Total params: 11,502
Trainable params: 11,502
Non-trainable params: 0
_________________________________________________________________
```

شکل ۱۵- مدل اجرایی RNN

در کد فوق با استفاده از تابع فعال ساز reluتعداد 100یونیت را درنظر گرفته و در انتها به منظور پیشبینی خروجی دو شرکت یک لایه denseقرار می دهیم.

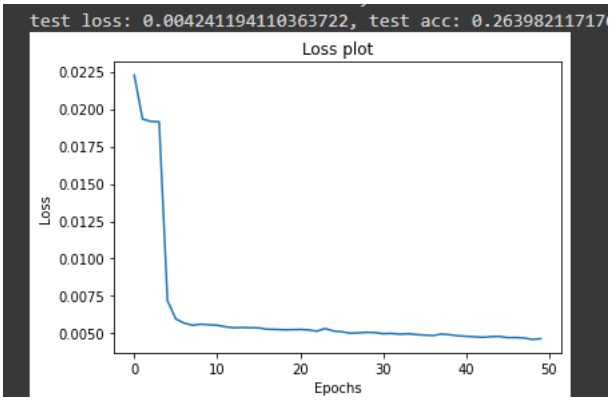
زمان اجرا :

۷,۱۵ میکر. ثانیه

کمترین loss :۰,۰۰۴۶

دقت:

۸۱٪



شکل ۱۶ -نمودار loss

همانطور که مشاهده می شود، مقدار Lossالگوریتم RNNاز هر دو الگوریتم دیگر کمی بدتر می باشد

مقایسه سه مدل :

جدول ۱ – مقایسه سه مدل

loss	دقت	زمان اجرا	
۰,۰۰۳	۷۳٪	۱۰ میکرو	LSTM
۰,۰۰۵	۷۹٪	۹,۵۴ میکرو	GRU
۰,۰۰۵۲	۷۸٪	۷,۱۵ میکرو	RNN

همانطور که مشخص است پیش بینی GRU از شبکه ها بهتر عمل کرده است و مقدار loss آن از rnn کمتر و از LSTM بیشتر است / اما زمان اجرا LSTM از هر سه شبکه بیشتر است.

اگر بخواهیم به طور کلی در نظر بگیریم به نظر GRU مدل مناسبی محسوب میشود.

در طراحی شبکه های عصبی با هدف تداعی کردن یک پترن، نیاز به حافظه است. حال پترن ورودی میتواند به صورت explicitیا implicitباشد. منظور از ورودی implicitآن است که ورودی ها توسط یک اردر زمانی یا مکانی به شبکه داده میشوند. درغیراینصورت ورودی ها explicitاست. طراحی یک شبکه عصبی برای ورودیهای explicitساده تر از طراحی شبکه برای داده های implicitاست و تنها به حافظه استاتیکی نیاز دارند. در مسائل واقعی و پرچالش تر ابعاد داده های ورودی ثابت نبوده و وجود نویز غیرقابل انکار است. دراینگونه مسائل که ورودی ها implicitهستند، از شبکه های recurrentاستفاده میشود. در شبکه های recurrentپس از طی کردن مسیر feedforwardدر راستای ساخت خروجی، آنرا به ورودی اعمال میکنند. در این حالت خروجی همواره از exogenous inputبهره میبرد تا تداعی را به درستی انجام دهد. قاعده یادگیری در شبکه های RNN، gradient updating ruleاست. در ادامه سه نمونه از شبکه های recurrentبررسی خواهد شد.

در سلول RNNورودی در لحظه کنونی با خروجیhidden state لحظه قبل ترکیب شده و پس از عبور از تابع فعالساز ،hidden state / tanhلحظه کنونی یا همان حافظه را میسازند. تابع tanhبرای کنترل فلو اطلاعاتی در شبکه استفاده میشود. در ادامه یک سلول RNNآمده است

گفته شد که در شبکه های RNNاز قاعده یادگیری گرادیان استفاده میکنند. بنابراین با افزایش طول ورودی های ،implicitمسیرهای برگشتی طولانیتر شده و محاسبات سخت خواهد شد. همچنین ازآنجایی که از توابع فعالساز با مشتق زیر یک استفاده میشود، در زنجیره های طولانی، مولفه خطای

برگشتی برای وزن هایی که در زمان های دور هستند، کوچک میشود و واکشی اطلاعات از لحظه کنونی برای به روزرسانی وزنهای متاثر از داده های قدیمی ضعیف خواهد بود.(مشکل vanishing gradient(برای حل این مشکل میتوان از توابع فعالسازی همچون ReLUبهره برد اما این تابع نیز برای مقادیر نامثبت، مشکلساز میشود و باید به دنبال ساز و کارهایی غیر از توابع فعالساز رفت. درواقع شبکههای RNNبه دلیل سادگی، برای کاربردهایی که نیاز به short depencyدارند، استفاده میشوند. علت بدتر عمل کردن این شبکه ها هم این است که دارای long term memoryنیستند

:LSTM

برای حل مشکل بازیابی اطلاعات برای داده های با فاصله طولانی و برقراری long dependency استفاده از ماژول LSTMپیشنهاد میشودLSTM .اجازه میدهد دادهها از زمانهای دور ذخیره گردند. به عنوان مثال برای پردازش یک پاراگراف در راستای تولید متن، RNNکلاسیک اطلاعات مهمی که در ابتدای متن هستند را درنظر نمیگیرد. حال سلولهایی همانند LSTMباعث میشوند مشکل short-term memoryحل شود؛ زیرا با وجود مکانیزمهای داخلی (گیتها) فلو اطلاعاتی را کنترل میکنند. در ادامه یک سلول LSTMآمده است

سلول LSTMهمانند ،RNNفلو اطلاعاتی را کنترل میکند و در مسیر forwardانتشار میدهند با این تفاوت که در سلول LSTMعملیات متفاوتی انجام میشود. این عملیات به LSTMاجازه میدهند تا اطلاعات را حفظ و یا پاک کنند. هسته مرکزی LSTMدرواقع cell stateو گیتهای آن است که منجر میشود اطلاعات مفید (مهم نیست برای چه مدت پیش هستند) در حافظه محفوظ بمانند. از طرفی گیتها ممکن است به ذنجیره، اطلاعاتی بیافزایند و یا از حذف کنند. درواقع این گیتها هستند که یاد میگیرند اطلاعاتی مفید بوده و یا باید فراموش شود. گیتها دارای تابع فعالساز سیگموید هستند. تفاوت SIGMOIDو tanhدر آن است که SIGMOIDخروجی را بین ۰و ۱میبرد. بنابراین اگر اطلاعاتی باید فراموش گردد در صفر ضرب شده و حذف میگردد. در LSTMسه گیت مختلف وجود دارد که فلو اطلاعاتی را کنترل میکنند.(گیت فراموشی، گیت ورودی و گیت خروجی)

گیت خروجی: این گیت تصمیم میگیرد که hidden stateبعدی چه باید باشد. درواقع hidden stateدارای اطلاعاتی از ورودیهای قبلی است و برای پیشبینی نیز استفاده میشود. عملکرد این گیت

بدین صورت است که hidden stateقبلی و ورودی کنونی به تابع sigmoidداده شده و cell stateجدید به تابع tanhاعمال میشود. حال خروجی tanhو sigmoidضرب میشوند تا اطلاعاتی که hidden stateباید داشته باشد مشخص گردد. درواقع خروجی سلول، hidden stateاست.

گیت فراموشی: اطلاعات ورودی کنونی و خروجی(hidden state) قبل ترکیب شده و به SIGMOIDاعمال میشوند و خروجی آن مقداری بین صفر تا یک دارد.

با استفاده از خروجی گیت فراموشی و گیت ورودی، اطلاعات لازم برای محاسبه cell stateساخته شده است. درواقع گیت فراموشی تصمیم میگیرد که اطلاعات ساخته شده در گیت ورودی مهم بوده و یا نه و cell stateجدید ساخته میشود

گیت ورودی: کاربرد این گیت در راستای updateکردن cell stateاست. اطلاعات inputکنونی و hidden stateقبلی ترکیب شده و به sigmoidاعمال میشوند. حال sigmoidتصمیم میگیرد که چه اطلاعاتی باید updateشوند. همچنین ترکیب inputکنونی و hidden stateقبلی وارد یک tanhشده و خروجی sigmoidو tanhیا یکدیگر ضرب میشوندsigmoid .تصمیم میگیرد که چه اطلاعاتی مهم بوده و باید حفظ شوند

سلول LSTMدر مقایسه با ،RNNاز درجه آزادی بیشتری برخوردار است و امکان ترکیب ورودی ها با داده های بیشتری وجود داشته که منجر به کنترل بهتر خروجی ها نیز میشود. بنابراین سلول LSTMکنترل بهتر پارامترها و درنتیجه نتایج بهتر را به ارمغان می آورد اما هزینه آن پیچیدگی و عملیات بیشتر است.

:GRU

سلول GRUنسل جدیدی از شبکههای عصبی recurrentاست و شباهت زیادی با LSTMدارد. در GRU cell state حذف شده و از hidden stateبرای انتقال اطلاعات استفاده میکند. همچنین گیتهای آن updateو resetهستند . گیت update:این گیت شبیه به گیت فراموشی و گیت ورودی LSTMعمل میکندوتصمیم میگیرد که چه اطلاعاتی حذف و یا اضافه شود.

درواقع سلول ،GRUمحاسبات کمتری داشته که منجر میشود در مقایسه با LSTMدارای سرعت بیشتری باشد. البته هر یک از GRUو LSTMبسته به کاربرد ممکن است بهتر از دیگری باشد.

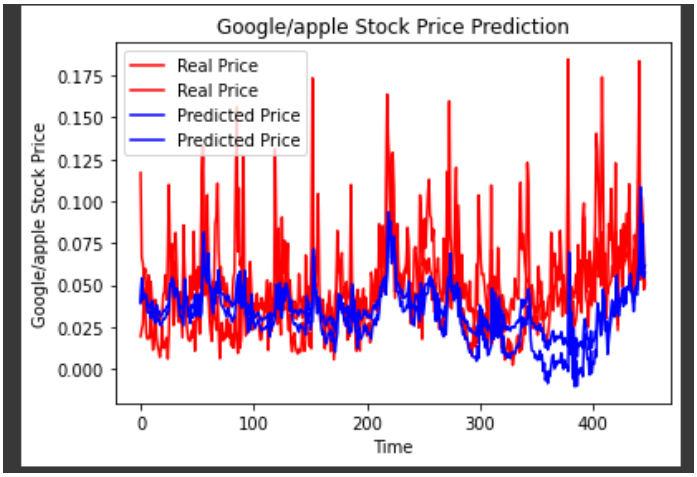
باتوجه به نتایج سه الگوریتم، و جدول ۱ در صفحه ۸ به نظر می رسد الگوریتم GRUبهترین نتیجه را ارائه کرده است.

ب) نحوه عملکرد MSE ,MPE برای هر سه شبکه :

برای LTSM :(حالت MSE)

```
[47] Epoch 42/50
     56/56 [==============================] - 3s 59ms/step - loss: 0.0050 - accuracy: 0.7103
     Epoch 43/50
     56/56 [==============================] - 3s 58ms/step - loss: 0.0050 - accuracy: 0.7254
     Epoch 44/50
     56/56 [==============================] - 3s 58ms/step - loss: 0.0050 - accuracy: 0.7220
     Epoch 45/50
     56/56 [==============================] - 3s 58ms/step - loss: 0.0050 - accuracy: 0.7282
     Epoch 46/50
     56/56 [==============================] - 3s 60ms/step - loss: 0.0050 - accuracy: 0.7265
     Epoch 47/50
     56/56 [==============================] - 3s 59ms/step - loss: 0.0050 - accuracy: 0.7248
     Epoch 48/50
     56/56 [==============================] - 3s 58ms/step - loss: 0.0051 - accuracy: 0.7142
     Epoch 49/50
     56/56 [==============================] - 3s 58ms/step - loss: 0.0051 - accuracy: 0.7232
     Epoch 50/50
     56/56 [==============================] - 3s 58ms/step - loss: 0.0051 - accuracy: 0.7299
     <keras.callbacks.History at 0x7fb80313d550>
```

شکل ۱۷ – خروجی مدل LTSM برای MSE



دقت :۷۳٪ / loss : ۰,۰۰۳ /

زمان اجرا : ۱۰ میکرو ثانیه

برای LTSM :(حالت MAPE)

زمان اجرا : ۸,۵۸ میکروثانیه / loss : 4500 / دقت :۴۵ درصد

```
56/56 [==============================] - 3s 60ms/step - loss: 9455.7461 - accuracy: 0.3798
Epoch 39/50
56/56 [==============================] - 3s 62ms/step - loss: 19885.7598 - accuracy: 0.6734
Epoch 40/50
56/56 [==============================] - 4s 80ms/step - loss: 12775.6709 - accuracy: 0.4698
Epoch 41/50
56/56 [==============================] - 4s 70ms/step - loss: 11304.0215 - accuracy: 0.6773
Epoch 42/50
56/56 [==============================] - 4s 71ms/step - loss: 39902.9180 - accuracy: 0.6762
Epoch 43/50
56/56 [==============================] - 3s 60ms/step - loss: 12955.2217 - accuracy: 0.6734
Epoch 44/50
56/56 [==============================] - 3s 59ms/step - loss: 11077.9619 - accuracy: 0.6035
Epoch 45/50
56/56 [==============================] - 4s 68ms/step - loss: 7283.3955 - accuracy: 0.5570
Epoch 46/50
56/56 [==============================] - 4s 79ms/step - loss: 30602.4414 - accuracy: 0.4463
Epoch 47/50
56/56 [==============================] - 3s 62ms/step - loss: 14887.1260 - accuracy: 0.6622
Epoch 48/50
56/56 [==============================] - 3s 61ms/step - loss: 4504.8799 - accuracy: 0.3742
Epoch 49/50
56/56 [==============================] - 3s 61ms/step - loss: 39302.5547 - accuracy: 0.4385
Epoch 50/50
56/56 [==============================] - 3s 61ms/step - loss: 11772.7461 - accuracy: 0.4469
<keras.callbacks.History at 0x7fb7fc65a650>
```

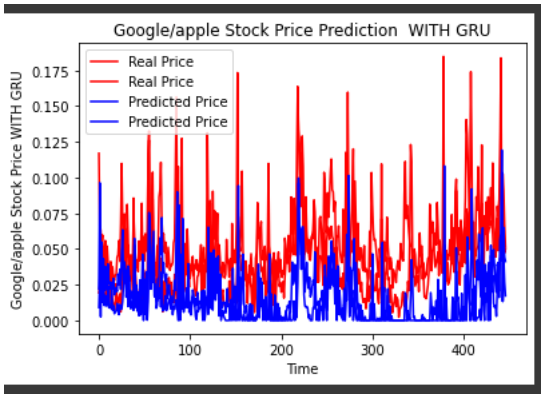
شکل ۱۸ – خروجی مدل برای LTSMبرای MAPE

برای GRU :(حالت MSE)

```
from keras.layers import GRU
%time
model = Sequential()
model.add(GRU(50, batch_input_shape=(None, 29,12 ), recurrent_dropout=0))
model.add(Dense(2, activation='relu'))
model.compile(loss='mse', optimizer='adam', metrics=['mae'])
model.summary()
```

```
120/120 [==============================] - 2s 15ms/step - loss: 0.0055 - accuracy: 0.7825 - val_loss: 0.0010 - val_accuracy: 0.4358
Epoch 43/50
120/120 [==============================] - 2s 15ms/step - loss: 0.0055 - accuracy: 0.7867 - val_loss: 9.6443e-04 - val_accuracy: 0.7095
Epoch 44/50
120/120 [==============================] - 2s 15ms/step - loss: 0.0055 - accuracy: 0.7692 - val_loss: 7.7792e-04 - val_accuracy: 0.5000
Epoch 45/50
120/120 [==============================] - 2s 16ms/step - loss: 0.0055 - accuracy: 0.7804 - val_loss: 0.0020 - val_accuracy: 0.3603
Epoch 46/50
120/120 [==============================] - 2s 15ms/step - loss: 0.0054 - accuracy: 0.7965 - val_loss: 8.1060e-04 - val_accuracy: 0.7263
Epoch 47/50
120/120 [==============================] - 2s 15ms/step - loss: 0.0055 - accuracy: 0.7825 - val_loss: 0.0019 - val_accuracy: 0.3659
Epoch 48/50
120/120 [==============================] - 2s 15ms/step - loss: 0.0055 - accuracy: 0.7825 - val_loss: 0.0014 - val_accuracy: 0.3659
Epoch 49/50
120/120 [==============================] - 2s 15ms/step - loss: 0.0054 - accuracy: 0.7797 - val_loss: 0.0020 - val_accuracy: 0.4385
Epoch 50/50
120/120 [==============================] - 2s 15ms/step - loss: 0.0054 - accuracy: 0.7832 - val_loss: 7.3981e-04 - val_accuracy: 0.7123
```

شکل ۱۹ – خروجی مدل GRUبرای MSE



زمان اجرا : ۹,۵۴ میکرو / دقت :۷۹% / loss : ۰/۰۰۵

برای GRU :(حالت MAPE)

زمان اجرا : ۹,۵۴ میکرو / دقت :۷۶% / loss : ۷۱

```
[180] Epoch 36/50
      120/120 [==============================] - 2s 15ms/step - loss: 71.8550 - accuracy: 0.7622 - val_loss: 69271.1250 - val_accuracy: 0.3492
      Epoch 37/50
      120/120 [==============================] - 2s 16ms/step - loss: 70.9838 - accuracy: 0.7615 - val_loss: 48902.1289 - val_accuracy: 0.3492
      Epoch 38/50
      120/120 [==============================] - 2s 15ms/step - loss: 69.0007 - accuracy: 0.7538 - val_loss: 28651.6484 - val_accuracy: 0.3492
      Epoch 39/50
      120/120 [==============================] - 2s 15ms/step - loss: 59.6607 - accuracy: 0.6762 - val_loss: 22641.5664 - val_accuracy: 0.3492
      Epoch 40/50
      120/120 [==============================] - 2s 15ms/step - loss: 55.9359 - accuracy: 0.6643 - val_loss: 53418.7305 - val_accuracy: 0.3492
      Epoch 41/50
      120/120 [==============================] - 2s 15ms/step - loss: 31940.2773 - accuracy: 0.7196 - val_loss: 34319.6211 - val_accuracy: 0.3492
      Epoch 42/50
      120/120 [==============================] - 2s 15ms/step - loss: 73.4969 - accuracy: 0.7622 - val_loss: 86146.5859 - val_accuracy: 0.3492
      Epoch 43/50
      120/120 [==============================] - 2s 15ms/step - loss: 73.4705 - accuracy: 0.7622 - val_loss: 40980.7461 - val_accuracy: 0.3492
      Epoch 44/50
      120/120 [==============================] - 2s 15ms/step - loss: 72.9556 - accuracy: 0.7622 - val_loss: 74107.2656 - val_accuracy: 0.3492
      Epoch 45/50
      120/120 [==============================] - 2s 15ms/step - loss: 72.5240 - accuracy: 0.7622 - val_loss: 38270.8008 - val_accuracy: 0.3492
      Epoch 46/50
      120/120 [==============================] - 2s 14ms/step - loss: 72.4069 - accuracy: 0.7622 - val_loss: 59701.9844 - val_accuracy: 0.3492
      Epoch 47/50
      120/120 [==============================] - 2s 14ms/step - loss: 72.7511 - accuracy: 0.7622 - val_loss: 46927.7930 - val_accuracy: 0.3492
      Epoch 48/50
      120/120 [==============================] - 3s 25ms/step - loss: 71.5703 - accuracy: 0.7622 - val_loss: 45236.8359 - val_accuracy: 0.3492
      Epoch 49/50
      120/120 [==============================] - 2s 19ms/step - loss: 71.9350 - accuracy: 0.7622 - val_loss: 30645.9785 - val_accuracy: 0.3492
      Epoch 50/50
      120/120 [==============================] - 2s 15ms/step - loss: 71.2178 - accuracy: 0.7622 - val_loss: 54151.1992 - val_accuracy: 0.3492
```

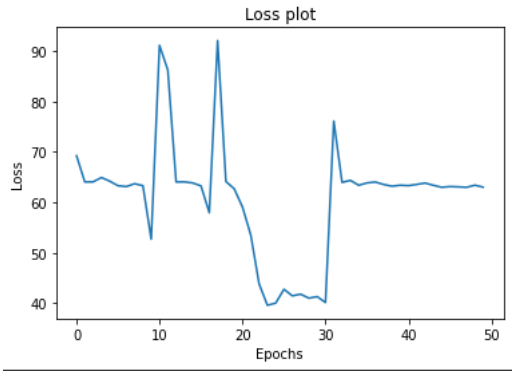
شکل ۲۰– خروجی مدل GRUبرای MAPE

برای RNN :(حالت MAPE)

زمان اجرا : ۱۰میکرو / دقت :۷۶% / loss : ۶۳

```
120/120 [==============================] - 1s 7ms/step - loss: 63.3036 - accuracy: 0.7622 - val_loss: 68514.4609 - val_accuracy: 0.3492
Epoch 42/50
120/120 [==============================] - 1s 7ms/step - loss: 63.5411 - accuracy: 0.7622 - val_loss: 128659.3828 - val_accuracy: 0.3492
Epoch 43/50
120/120 [==============================] - 1s 8ms/step - loss: 63.8135 - accuracy: 0.7622 - val_loss: 86877.6562 - val_accuracy: 0.3492
Epoch 44/50
120/120 [==============================] - 1s 8ms/step - loss: 63.3714 - accuracy: 0.7622 - val_loss: 76318.7344 - val_accuracy: 0.3492
Epoch 45/50
120/120 [==============================] - 1s 7ms/step - loss: 62.9763 - accuracy: 0.7622 - val_loss: 55951.8008 - val_accuracy: 0.3492
Epoch 46/50
120/120 [==============================] - 1s 7ms/step - loss: 63.1117 - accuracy: 0.7622 - val_loss: 61411.4297 - val_accuracy: 0.3492
Epoch 47/50
120/120 [==============================] - 1s 7ms/step - loss: 63.0593 - accuracy: 0.7622 - val_loss: 45816.1133 - val_accuracy: 0.3492
Epoch 48/50
120/120 [==============================] - 1s 7ms/step - loss: 62.9675 - accuracy: 0.7622 - val_loss: 102867.7344 - val_accuracy: 0.3492
Epoch 49/50
120/120 [==============================] - 1s 8ms/step - loss: 63.3928 - accuracy: 0.7622 - val_loss: 78598.6953 - val_accuracy: 0.3492
Epoch 50/50
120/120 [==============================] - 1s 7ms/step - loss: 62.9980 - accuracy: 0.7622 - val_loss: 63955.8555 - val_accuracy: 0.3492
```

شکل ۲۱– خروجی مدل RNNبرای MAPE

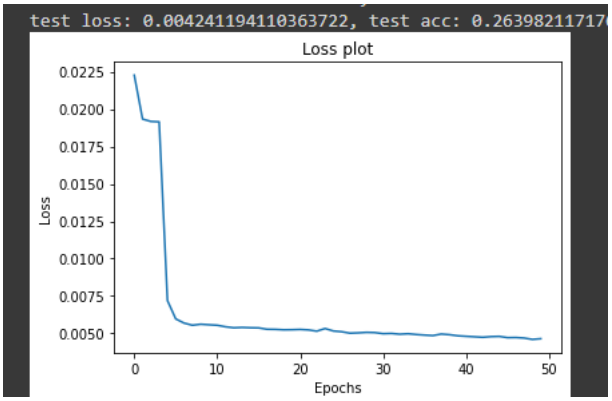


برای RNN :(حالت MSE)

۷,۱۵ میکروثانیه/ کمترین loss :۰,۰۰۴۶ / دقت:81%

```
%time
model = Sequential()
model.add(SimpleRNN(units=100, input_shape=x_train.shape[1:], activation="relu", recurrent_dropout=0.0))
model.add(Dense(2, activation='relu'))
model.compile(loss='mse', optimizer='adam', metrics=['mae'])
model.summary()

CPU times: user 4 µs, sys: 0 ns, total: 4 µs
Wall time: 10.5 µs
Model: "sequential_11"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 simple_rnn (SimpleRNN)      (None, 100)               11300

 dense_9 (Dense)             (None, 2)                 202

=================================================================
Total params: 11,502
Trainable params: 11,502
Non-trainable params: 0
_________________________________________________________________
```

شکل 22– مدل اجرایی RNN



شکل 23 -نمودار loss

جدول ۳- مقایسه MAPE, MSE برای سه شبکه

	زمان اجرا(میکرو)	دقت	loss
LSTM(MSE)	10	۷۳٪	۰,۰۰۳
LSTM(MAPE)	۸,۵۸	۴۵٪	۴۵۰۰
GRU(MSE)	۹,۵۴	۷۹٪	۰,۰۰۵
GRU(MAPE)	۹,۵	۷۶٪	۷۱
RNN(MSE)	۷,۱۵	۸۱٪	۰,۰۰۴۶
RNN(MAPE)	۱۰	۷۶٪	۶۳

همانطور که میبینیم مقدارloss برای حالت MAPE خیلی بیشتر از حالت MSE است و هم چنیندقت در شبکه های RNN, GRU زیاد تغییری نکرده است ولی در LSTM تغییر زیادی کرده است .

(زمان های اجرا نیز میتوان گفت زیاد تغییری نکرده اند)

در کل میتوان گفت عملکرد MSE مناسب تر از MAPE است.

علت این است که در MSE شبکه را برای خطا های بزرگ PENALIZE میکند در صورتی که MAPE این کاررا انجام نمیدهد و در این سوال از آن جا که قیمت ها نوسان زیادی دارند و دو شرکت باهم مقایسه میشوند استفاده از MSE بهتر است.

مجموع مربعات خطا، تابع پیشفرض مسائل Regressionاست. این خطا بهصورت میانگین مجذور اختلاف بین خروجیهای پیشبینی شده و خروجی Targetمحاسبه میگرددMSE .جدای از علامت

Predictو ،Targetهمواره دارای مقدار مثبت است و بهترین مقدار خطا برای آن صفر خواهد بود. استفاده از مجذور خطا نمایانگر آن است که اشتباهات بزرگتر منجر به خطاهای بیشتر میشود

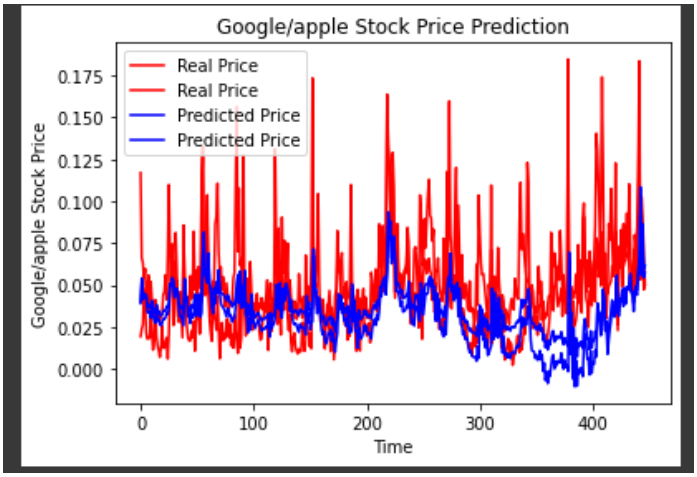
پ) نحوه عملکرد برای توابع بهینه ساز مختلف:

LTSM ;

استفاده از adam

```
[47] Epoch 42/50
     56/56 [==============================] - 3s 59ms/step - loss: 0.0050 - accuracy: 0.7103
     Epoch 43/50
     56/56 [==============================] - 3s 58ms/step - loss: 0.0050 - accuracy: 0.7254
     Epoch 44/50
     56/56 [==============================] - 3s 58ms/step - loss: 0.0050 - accuracy: 0.7220
     Epoch 45/50
     56/56 [==============================] - 3s 58ms/step - loss: 0.0050 - accuracy: 0.7282
     Epoch 46/50
     56/56 [==============================] - 3s 60ms/step - loss: 0.0050 - accuracy: 0.7265
     Epoch 47/50
     56/56 [==============================] - 3s 59ms/step - loss: 0.0050 - accuracy: 0.7248
     Epoch 48/50
     56/56 [==============================] - 3s 58ms/step - loss: 0.0051 - accuracy: 0.7142
     Epoch 49/50
     56/56 [==============================] - 3s 58ms/step - loss: 0.0051 - accuracy: 0.7232
     Epoch 50/50
     56/56 [==============================] - 3s 58ms/step - loss: 0.0051 - accuracy: 0.7299
     <keras.callbacks.History at 0x7fb80313d550>
```

شکل 24 – خروجی مدل LTSM برای adam



دقت :۷۳٪ / loss : ۰,۰۰۳ /

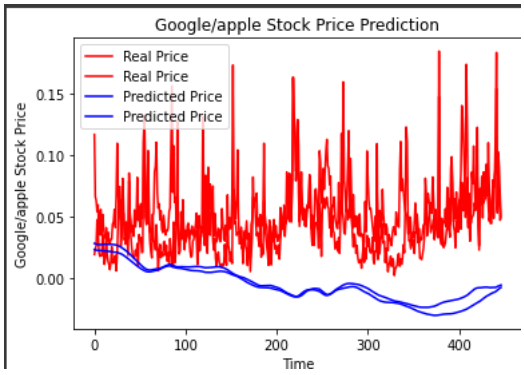
زمان اجرا : ۱۰ میکرو ثانیه

استفاده از adagrad :

```
model.add(Dropout(0.2))
model.compile(loss='mse', optimizer='ADAgrad', metrics=['mae'])
```

```
56/56 [==============================] - 2s 43ms/step - loss: 0.0077 - accuracy: 0.6784
Epoch 45/50
56/56 [==============================] - 2s 42ms/step - loss: 0.0077 - accuracy: 0.6751
Epoch 46/50
56/56 [==============================] - 2s 43ms/step - loss: 0.0080 - accuracy: 0.6723
Epoch 47/50
56/56 [==============================] - 2s 43ms/step - loss: 0.0078 - accuracy: 0.6874
Epoch 48/50
56/56 [==============================] - 2s 43ms/step - loss: 0.0078 - accuracy: 0.6779
Epoch 49/50
56/56 [==============================] - 2s 43ms/step - loss: 0.0078 - accuracy: 0.6885
Epoch 50/50
56/56 [==============================] - 2s 44ms/step - loss: 0.0077 - accuracy: 0.6745
<keras.callbacks.History at 0x7f1719a21650>
```

شکل 25– خروجی مدل LTSM برای adamgrad



دقت :67٪ / loss : / ۰,۰۰۸

زمان اجرا : 5 میکرو ثانیه

Lstm برای RMSpop :

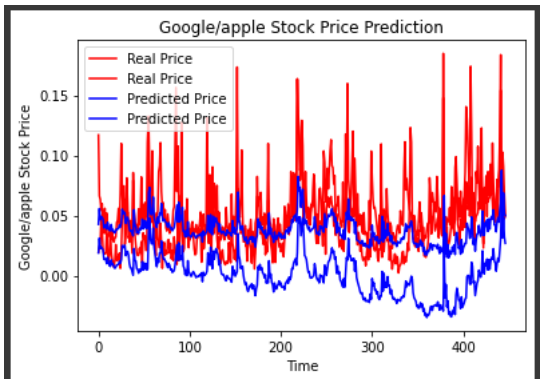
دقت :73% / loss : / ۰,۰۰۵۴

زمان اجرا : ۴,۷۷ میکرو ثانیه

```
model.add(Dense(units = 2))
model.compile(optimizer = 'RMSProp',loss = 'mse',metrics = ["accuracy"])
```

```
Epoch 45/50
56/56 [==============================] - 2s 44ms/step - loss: 0.0054 - accuracy: 0.7153
Epoch 46/50
56/56 [==============================] - 2s 45ms/step - loss: 0.0054 - accuracy: 0.7187
Epoch 47/50
56/56 [==============================] - 2s 43ms/step - loss: 0.0053 - accuracy: 0.7103
Epoch 48/50
56/56 [==============================] - 2s 44ms/step - loss: 0.0052 - accuracy: 0.7103
Epoch 49/50
56/56 [==============================] - 2s 44ms/step - loss: 0.0054 - accuracy: 0.7248
Epoch 50/50
56/56 [==============================] - 2s 44ms/step - loss: 0.0054 - accuracy: 0.7260
<keras.callbacks.History at 0x7f1716246150>
```

شکل ۲۶– خروجی مدل LTSM برای RMSpop

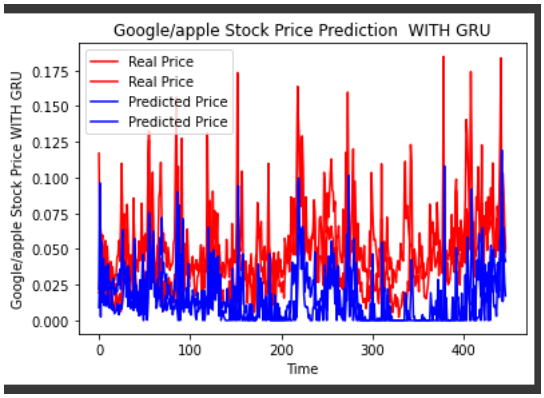


برای GRU حالت ADAM:

```
from keras.layers import GRU
%time
model = Sequential()
model.add(GRU(50, batch_input_shape=(None, 29,12 ), recurrent_dropout=0))
model.add(Dense(2, activation='relu'))
model.compile(loss='mse', optimizer='adam', metrics=['mae'])
model.summary()
```

```
120/120 [==============================] - 2s 15ms/step - loss: 0.0055 - accuracy: 0.7825 - val_loss: 0.0010 - val_accuracy: 0.4358
Epoch 43/50
120/120 [==============================] - 2s 15ms/step - loss: 0.0055 - accuracy: 0.7867 - val_loss: 9.6443e-04 - val_accuracy: 0.7095
Epoch 44/50
120/120 [==============================] - 2s 15ms/step - loss: 0.0055 - accuracy: 0.7692 - val_loss: 7.7792e-04 - val_accuracy: 0.5000
Epoch 45/50
120/120 [==============================] - 2s 16ms/step - loss: 0.0055 - accuracy: 0.7804 - val_loss: 0.0020 - val_accuracy: 0.3603
Epoch 46/50
120/120 [==============================] - 2s 15ms/step - loss: 0.0054 - accuracy: 0.7965 - val_loss: 8.1060e-04 - val_accuracy: 0.7263
Epoch 47/50
120/120 [==============================] - 2s 15ms/step - loss: 0.0055 - accuracy: 0.7825 - val_loss: 0.0019 - val_accuracy: 0.3659
Epoch 48/50
120/120 [==============================] - 2s 15ms/step - loss: 0.0055 - accuracy: 0.7825 - val_loss: 0.0014 - val_accuracy: 0.3659
Epoch 49/50
120/120 [==============================] - 2s 15ms/step - loss: 0.0054 - accuracy: 0.7797 - val_loss: 0.0020 - val_accuracy: 0.4385
Epoch 50/50
120/120 [==============================] - 2s 15ms/step - loss: 0.0054 - accuracy: 0.7832 - val_loss: 7.3981e-04 - val_accuracy: 0.7123
```

شکل ۲۷ – خروجی مدل GRUبرای ADAM



زمان اجرا : ۹,۵۴ میکرو / دقت :۷۹% / loss : ۰/۰۰۵

برای GRU حالت adagrad:

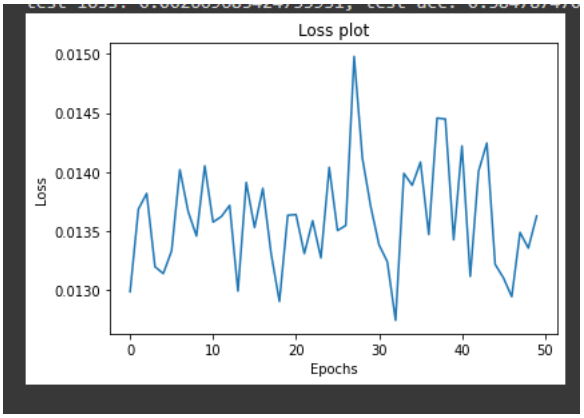
زمان اجرا : ۵ میکرو / دقت :۶۸% / loss : ۰/۰۱

```
model.add(Dropout(0.2))
model.compile(loss='mse', optimizer='ADAgrad', metrics=['mae'])
model.summary()
```

```
CPU times: user 2 µs, sys: 0 ns, total: 2 µs
Wall time: 4.29 µs
Model: "sequential_7"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 gru_4 (GRU)                 (None, 50)                9600

 dropout_20 (Dropout)        (None, 50)                0

 dense_7 (Dense)             (None, 2)                 102

 dropout_21 (Dropout)        (None, 2)                 0

=================================================================
Total params: 9,702
Trainable params: 9,702
Non-trainable params: 0
_________________________________________________________________
```

```
120/120 [==============================] - 1s 11ms/step - loss: 0.0132 - accuracy: 0.6916 - val_loss: 0.0011 - val_accuracy: 0.6732
Epoch 46/50
120/120 [==============================] - 1s 11ms/step - loss: 0.0131 - accuracy: 0.7042 - val_loss: 0.0011 - val_accuracy: 0.6788
Epoch 47/50
120/120 [==============================] - 1s 11ms/step - loss: 0.0129 - accuracy: 0.7063 - val_loss: 0.0011 - val_accuracy: 0.6704
Epoch 48/50
120/120 [==============================] - 1s 11ms/step - loss: 0.0135 - accuracy: 0.6930 - val_loss: 0.0011 - val_accuracy: 0.6592
Epoch 49/50
120/120 [==============================] - 1s 12ms/step - loss: 0.0134 - accuracy: 0.6818 - val_loss: 0.0011 - val_accuracy: 0.6648
Epoch 50/50
120/120 [==============================] - 1s 11ms/step - loss: 0.0136 - accuracy: 0.6986 - val_loss: 0.0011 - val_accuracy: 0.6760
```

شکل ۲۷ – خروجی مدل GRUبرای adagrad



شکل ۲۸ – نمودار loss برای GRU حالت adagard

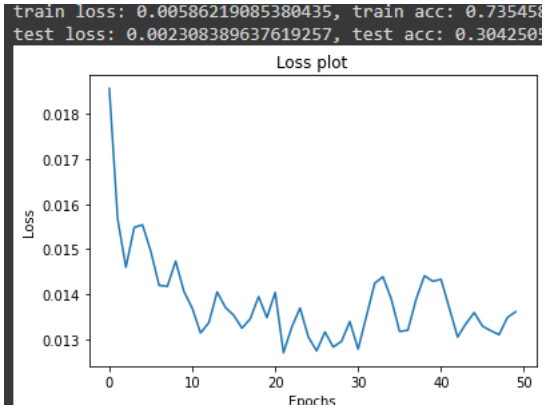
برای GRU حالت RMSPOP:

زمان اجرا : ۶,۴۴میکرو / دقت 69% / loss : ۰/۰۲

```
CPU times: user 3 µs, sys: 0 ns, total: 3 µs
Wall time: 5.96 µs
Model: "sequential_10"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 gru_7 (GRU)                 (None, 50)                9600

 dropout_26 (Dropout)        (None, 50)                0

 dense_10 (Dense)            (None, 2)                 102

 dropout_27 (Dropout)        (None, 2)                 0

=================================================================
Total params: 9,702
Trainable params: 9,702
Non-trainable params: 0
_________________________________________________________________
```

```
[57] 120/120 [==============================] - 1s 11ms/step - loss: 0.0144 - accuracy: 0.6664 - val_loss: 0.0019 - val_accuracy: 0.6285
     Epoch 40/50
     120/120 [==============================] - 1s 11ms/step - loss: 0.0143 - accuracy: 0.6727 - val_loss: 0.0018 - val_accuracy: 0.6425
     Epoch 41/50
     120/120 [==============================] - 1s 11ms/step - loss: 0.0143 - accuracy: 0.6755 - val_loss: 0.0011 - val_accuracy: 0.6955
     Epoch 42/50
     120/120 [==============================] - 1s 11ms/step - loss: 0.0137 - accuracy: 0.6923 - val_loss: 8.0761e-04 - val_accuracy: 0.4218
     Epoch 43/50
     120/120 [==============================] - 1s 11ms/step - loss: 0.0131 - accuracy: 0.7028 - val_loss: 7.8557e-04 - val_accuracy: 0.5726
     Epoch 44/50
     120/120 [==============================] - 1s 11ms/step - loss: 0.0133 - accuracy: 0.7035 - val_loss: 0.0013 - val_accuracy: 0.3855
     Epoch 45/50
     120/120 [==============================] - 1s 11ms/step - loss: 0.0136 - accuracy: 0.6888 - val_loss: 0.0019 - val_accuracy: 0.3631
     Epoch 46/50
     120/120 [==============================] - 1s 12ms/step - loss: 0.0133 - accuracy: 0.7028 - val_loss: 0.0014 - val_accuracy: 0.3855
     Epoch 47/50
     120/120 [==============================] - 1s 11ms/step - loss: 0.0132 - accuracy: 0.6944 - val_loss: 8.9394e-04 - val_accuracy: 0.4804
     Epoch 48/50
     120/120 [==============================] - 1s 11ms/step - loss: 0.0131 - accuracy: 0.6818 - val_loss: 9.4888e-04 - val_accuracy: 0.4050
     Epoch 49/50
     120/120 [==============================] - 1s 11ms/step - loss: 0.0135 - accuracy: 0.6930 - val_loss: 0.0015 - val_accuracy: 0.4693
     Epoch 50/50
     120/120 [==============================] - 1s 11ms/step - loss: 0.0136 - accuracy: 0.6923 - val_loss: 8.5969e-04 - val_accuracy: 0.4665
```

شکل ۲۹ – خروجی مدل GRUبرای rmspop



شکل ۳۰ – نمودار loss برای GRU حالت rmspop

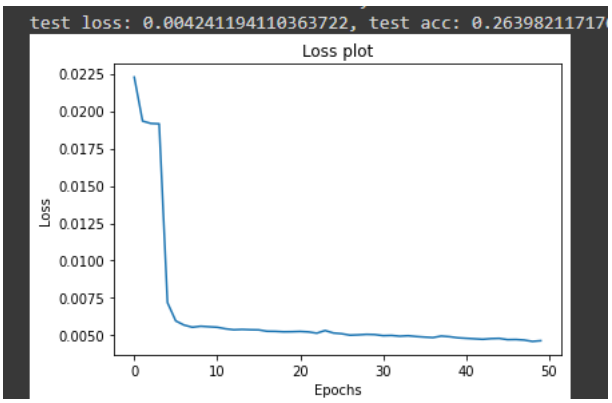
برای RNN حالت adaM:

۷,۱۵ میکروثانیه/ کمترین loss :۰,۰۰۴۶ / دقت:81%

```
%time
model = Sequential()
model.add(SimpleRNN(units=100, input_shape=x_train.shape[1:], activation="relu", recurrent_dropout=0.0))
model.add(Dense(2, activation='relu'))
model.compile(loss='mse', optimizer='adam', metrics=['mae'])
model.summary()
```

```
CPU times: user 4 µs, sys: 0 ns, total: 4 µs
Wall time: 10.5 µs
Model: "sequential_11"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 simple_rnn (SimpleRNN)      (None, 100)               11300

 dense_9 (Dense)             (None, 2)                 202

=================================================================
Total params: 11,502
Trainable params: 11,502
Non-trainable params: 0
_________________________________________________________________
```

شکل 31- مدل اجرایی RNN



شکل 32 -نمودار loss

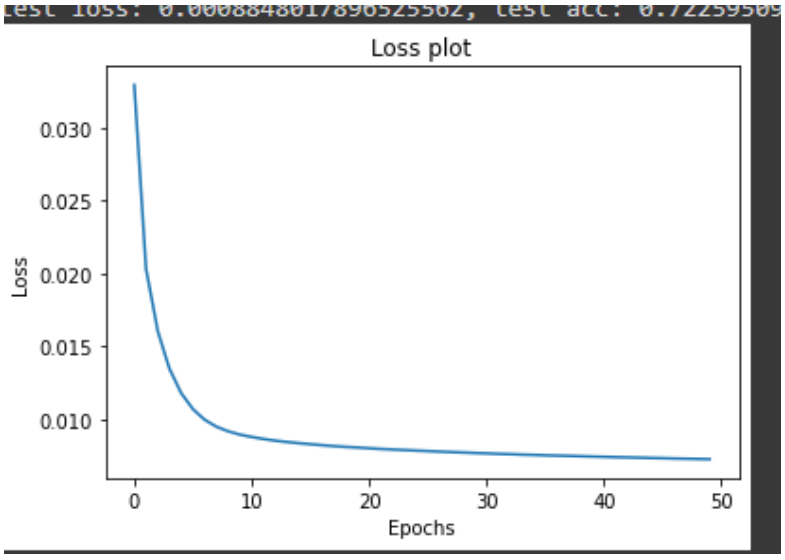
برای RNN حالت adagrad:

زمان اجرا : 6,2 میکرو / دقت :۷۳% / loss : ۰/۰۰۷

```
CPU times: user 2 µs, sys: 0 ns, total: 2 µs
Wall time: 6.2 µs
Model: "sequential_13"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 simple_rnn (SimpleRNN)      (None, 50)                3150

 dense_11 (Dense)            (None, 2)                 102

=================================================================
Total params: 3,252
Trainable params: 3,252
Non-trainable params: 0
_________________________________________________________________
```

شکل 33- مدل اجرایی RNN

```
Epoch 41/50
120/120 [==============================] - 1s 6ms/step - loss: 0.0074 - accuracy: 0.7140 - val_loss: 0.0012 - val_accuracy: 0.6145
Epoch 42/50
120/120 [==============================] - 1s 5ms/step - loss: 0.0074 - accuracy: 0.7168 - val_loss: 0.0013 - val_accuracy: 0.6285
Epoch 43/50
120/120 [==============================] - 1s 5ms/step - loss: 0.0074 - accuracy: 0.7154 - val_loss: 0.0013 - val_accuracy: 0.6341
Epoch 44/50
120/120 [==============================] - 1s 6ms/step - loss: 0.0074 - accuracy: 0.7175 - val_loss: 0.0013 - val_accuracy: 0.6285
Epoch 45/50
120/120 [==============================] - 1s 5ms/step - loss: 0.0073 - accuracy: 0.7154 - val_loss: 0.0013 - val_accuracy: 0.6369
Epoch 46/50
120/120 [==============================] - 1s 5ms/step - loss: 0.0073 - accuracy: 0.7210 - val_loss: 0.0013 - val_accuracy: 0.6425
Epoch 47/50
120/120 [==============================] - 1s 5ms/step - loss: 0.0073 - accuracy: 0.7217 - val_loss: 0.0012 - val_accuracy: 0.6341
Epoch 48/50
120/120 [==============================] - 1s 5ms/step - loss: 0.0073 - accuracy: 0.7196 - val_loss: 0.0012 - val_accuracy: 0.6397
Epoch 49/50
120/120 [==============================] - 1s 5ms/step - loss: 0.0073 - accuracy: 0.7210 - val_loss: 0.0013 - val_accuracy: 0.6425
Epoch 50/50
120/120 [==============================] - 1s 5ms/step - loss: 0.0073 - accuracy: 0.7245 - val_loss: 0.0012 - val_accuracy: 0.6425
```



شکل 34 -نمودار loss

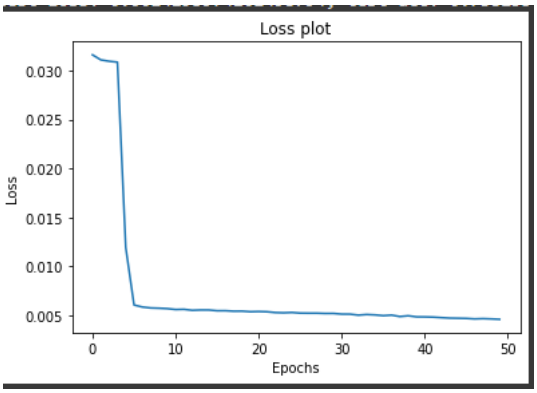
برای RNN حالت rmspop:

زمان اجرا : 6.44 میکرو / دقت :۸۰% / loss : ۰/۰۰۴

```
model.summary()

CPU times: user 3 µs, sys: 0 ns, total: 3 µs
Wall time: 6.91 µs
Model: "sequential_14"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 simple_rnn_1 (SimpleRNN)    (None, 50)                3150

 dense_12 (Dense)            (None, 2)                 102

=================================================================
Total params: 3,252
Trainable params: 3,252
Non-trainable params: 0
```

شکل 35- مدل اجرایی RNN

```
Epoch 40/50
120/120 [==============================] - 1s 6ms/step - loss: 0.0048 - accuracy: 0.7811 - val_loss: 0.0015 - val_accuracy: 0.3994
Epoch 41/50
120/120 [==============================] - 1s 6ms/step - loss: 0.0048 - accuracy: 0.7895 - val_loss: 0.0014 - val_accuracy: 0.4078
Epoch 42/50
120/120 [==============================] - 1s 6ms/step - loss: 0.0048 - accuracy: 0.7846 - val_loss: 0.0012 - val_accuracy: 0.4553
Epoch 43/50
120/120 [==============================] - 1s 6ms/step - loss: 0.0047 - accuracy: 0.7839 - val_loss: 0.0011 - val_accuracy: 0.6844
Epoch 44/50
120/120 [==============================] - 1s 6ms/step - loss: 0.0047 - accuracy: 0.7818 - val_loss: 0.0039 - val_accuracy: 0.6648
Epoch 45/50
120/120 [==============================] - 1s 5ms/step - loss: 0.0047 - accuracy: 0.7895 - val_loss: 0.0017 - val_accuracy: 0.6676
Epoch 46/50
120/120 [==============================] - 1s 5ms/step - loss: 0.0047 - accuracy: 0.7902 - val_loss: 0.0010 - val_accuracy: 0.7011
Epoch 47/50
120/120 [==============================] - 1s 5ms/step - loss: 0.0046 - accuracy: 0.7930 - val_loss: 0.0013 - val_accuracy: 0.7095
Epoch 48/50
120/120 [==============================] - 1s 5ms/step - loss: 0.0046 - accuracy: 0.7874 - val_loss: 8.7607e-04 - val_accuracy: 0.5363
Epoch 49/50
120/120 [==============================] - 1s 6ms/step - loss: 0.0046 - accuracy: 0.7916 - val_loss: 9.4937e-04 - val_accuracy: 0.7095
Epoch 50/50
120/120 [==============================] - 1s 5ms/step - loss: 0.0046 - accuracy: 0.7748 - val_loss: 0.0012 - val_accuracy: 0.7151
```



شکل 36 -نمودار loss

ADAM

در روش ADAM از نرخ یادگیری متفاوت برای آپدیت کردن هر پارامتر استفاده میشود. همچنین علاوه بر ،Learning Rateاز ترمMomentum متفاوت نیز استفاده میشود

$$\widehat{m_t} = \frac{m_t}{1-\beta_1^t}$$

$$\widehat{v_t} = \frac{v_t}{1-\beta_2^t}$$

بنابراین الگوریتم Adamاز رابطه زیر برای آپدیت کردن پارامترها استفاده میکند

$$\theta_{t+1} = \theta_t - \frac{\alpha}{\sqrt{\widehat{v_t}} + \varepsilon}.\widehat{m_t}$$

مقادیر رایج برای β۱،β2و εبه ترتیب ۰,۹، ۰,۹۹۹و ۱۰-۸است. درواقع روش Adamمنجر به همگرایی سریعتر نسبت به سایر الگوریتم های بهینه سازی میشود. همچنین با مشکلاتی همچون همگرایی نرخ یادگیری به صفر و کاهش سرعت همگرایی تابع خطا، مواجه نمیشود. به طورکلی الگوریتم Adamبهتر از الگوریتم های Adaptiveدیگر همانند RMSPropعمل میکند. حال اگر داده های ورودی به اصطلاح sparseباشند، روشه ایی مانند SGDو Momentumضعیف عمل میکنند و باید از روش های Adaptiveاستفاده کرد. برای دست یابی به همگرایی سریعتر در مدلهای عمیق و پیچیده، الگوریتم Adamبهتر از سایرین عمل میکند.

RMSprop :

بهینه ساز Root mean square propagationمنجر به کاهش نوسانات می شود. همچنین نیازی به تنظیم دستی نرخ یادگیری ندارد بلکه به صورت اتوماتیک آن را تنظیم میکند. در روش ،RMSProp به روزرسانی پارامترها به صورت زیر است

$$For\ each\ Parameter\ w^j$$
$$(j\ subscript\ dropped\ for\ clarity)$$
$$\nu_t = \rho\nu_{t-1} + (1-\rho) * g_t^2$$
$$\Delta\omega_t = -\frac{\eta}{\sqrt{\nu_t + \epsilon}} * g_t$$
$$\omega_{t+1} = \omega_t + \Delta\omega_t$$

درواقع برای هر پارامتر، میانگین نمایی مجذور گرادیان آن محاسبه میشود. استفاده از مجذور گرادیان منجر میشود، وزن پارامترهای پایانی بیشتر از قبلی ها بهروزرسانی شود. سپس در معادله دوم، میزان step توسط میانگین نمایی محاسبه میگردد. به عنوان مثال اگر میانگین w1بزرگتر از میانگین w2باشد، stepیادگیری برای w1کوچکتر از w2خواهد بود و منجر به یافتن مینمیم ها میشود. بنابراین هنگامی که تابع هزینه به نقاط مینیمم نزدیک میشود، RMSPropاز قدم های کوچکتر استفاده میکند.

ADAGRAD :

در این تابع هزینه هدف همان تغییر دادن سرعت update ها در راستای ویژگی های مختلف هست.

جدول ۳ - مقایسه optimizer های مختلف برای سه شبکه

	زمان اجرا	loss	دقت
LSTM(adam)	۱۰ میکرو	۰,۰۰۳	۷۳٪
LSTM(adamgrad)	۵ میکرو	۰,۰۰۸	۶۷٪
LSTM(RSMprop)	۴,۷۷ میکرو	۰,۰۰۵۴	۷۳٪
GRU(adam)	۹,۵۴ میکرو	۰,۰۰۵	۷۹٪
GRU(adamgrad)	۵ میکرو	۰,۰۱	۶۸٪
GRU(RSMprop)	۶,۴۴ میکرو	۰,۰۲	۶۹٪
RNN(adam)	۷,۱۵ میکرو	۰,۰۰۴۶	۸۱٪
RNN(adamgrad)	۶,۲ میکرو	۰,۰۰۷	۷۳٪
RNN(RSMprop)	۶,۴۴ میکرو	۰,۰۰۴۵	۷۹٪

با توجه به جدول سه میتوان گفت در کل از نظر خطا و دقت ADAM بهتر عمل کرده است / هم چنین قابل ذکر است که در حالت RSMprop زمان اجرا کاهش یافته است.

برای شبکه LSTM , adam از همه مناسب تر است

برای شبکه GRU نیز adam از همه مناسب تر است و برای شبکه RNN نیز هم RSMprop , adam مناسب هستند.

د) تاثیر dropoutبر سلولهای بازگشتی را روی شبکههای طراحی شده بررسی کنید.

برای شبکه ها با اضافه کردن سه dropout (0.2) تست شده اند.

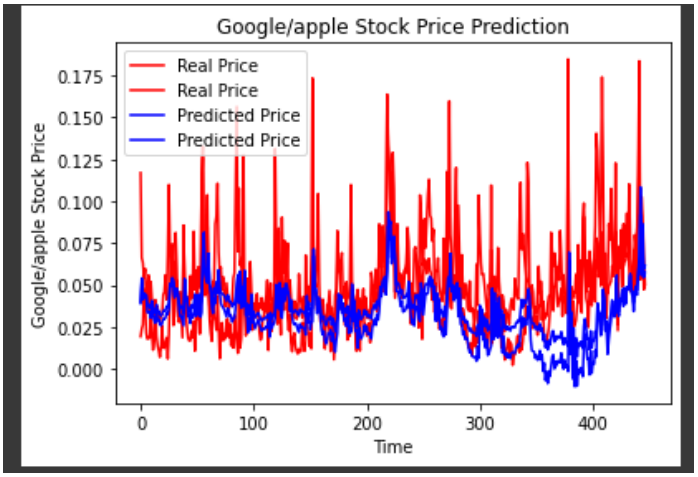
LSTM با drop out

```
[ ]  model = Sequential()
     model.add(LSTM(units=50, return_sequences=True, input_shape=(x_train.shape[1],12)))

[ ]  model.add(Dropout(0.2))

[ ]  model.add(LSTM(units = 50,return_sequences = True))
     model.add(Dropout(0.2))

[ ]  model.add(LSTM(units = 50,return_sequences = True))
     model.add(Dropout(0.2))

[ ]  model.add(LSTM(units = 50))
     model.add(Dropout(0.2))

[ ]  model.add(Dense(units = 2))

[ ]
     model.compile(optimizer = 'adam',loss = 'mean_squared_error',metrics = ["accuracy"])

     model.fit(x_train,y_train,epochs = 50, batch_size = 32)
```

شکل ۳۷- مدل LSTM باdropout

```
[47]  Epoch 42/50
      56/56 [==============================] - 3s 59ms/step - loss: 0.0050 - accuracy: 0.7103
      Epoch 43/50
      56/56 [==============================] - 3s 58ms/step - loss: 0.0050 - accuracy: 0.7254
      Epoch 44/50
      56/56 [==============================] - 3s 58ms/step - loss: 0.0050 - accuracy: 0.7220
      Epoch 45/50
      56/56 [==============================] - 3s 58ms/step - loss: 0.0050 - accuracy: 0.7282
      Epoch 46/50
      56/56 [==============================] - 3s 60ms/step - loss: 0.0050 - accuracy: 0.7265
      Epoch 47/50
      56/56 [==============================] - 3s 59ms/step - loss: 0.0050 - accuracy: 0.7248
      Epoch 48/50
      56/56 [==============================] - 3s 58ms/step - loss: 0.0051 - accuracy: 0.7142
      Epoch 49/50
      56/56 [==============================] - 3s 58ms/step - loss: 0.0051 - accuracy: 0.7232
      Epoch 50/50
      56/56 [==============================] - 3s 58ms/step - loss: 0.0051 - accuracy: 0.7299
      <keras.callbacks.History at 0x7fb80313d550>
```

شکل ۳۸ – خروجی مدل LTSM با dropout



دقت :۷۳٪ / loss : / ۰,۰۰۳

زمان اجرا : ۱۰ میکرو ثانیه

LSTM بدون dropout

دقت :۵۴٪ / loss : / ۵۰۰ ۳

زمان اجرا : ۸,۵۸ میکرو ثانیه

```
model = Sequential()
model.add(LSTM(units=50, return_sequences=True, input_shape=(x_train.shape[1],12)))

model.add(LSTM(units = 50,return_sequences = True))

model.add(LSTM(units = 50,return_sequences = True))

model.add(LSTM(units = 50))
model.add(Dense(units = 2))
model.compile(optimizer = 'adam',loss = 'mse',metrics = ["accuracy"])
```

شکل ۳۹ – مدل LSTM بدون drop out

```
1] 56/56 [==============================] - 3s 59ms/step - loss: 6206.3296 - accuracy: 0.6353
   Epoch 47/50
   56/56 [==============================] - 3s 59ms/step - loss: 9277.1699 - accuracy: 0.3400
   Epoch 48/50
   56/56 [==============================] - 3s 59ms/step - loss: 4361.2832 - accuracy: 0.4189
   Epoch 49/50
   56/56 [==============================] - 3s 61ms/step - loss: 11090.1475 - accuracy: 0.3708
   Epoch 50/50
   56/56 [==============================] - 3s 61ms/step - loss: 3539.0784 - accuracy: 0.6292
   <keras.callbacks.History at 0x7fca3587b650>
```

شکل ۴۰ – نتیج ترین کردن مدل

GRU بدون dropout :

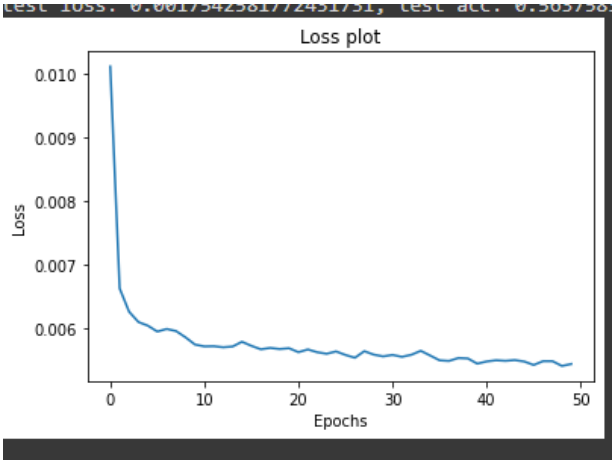
زمان اجرا : ۹,۵۴ میکرو / دقت : ٪۷۹ / loss : ۰,۰۰۵۴

```
from keras.layers import GRU
%time
model = Sequential()
model.add(GRU(50, batch_input_shape=(None, 29,12 ), recurrent_dropout=0))
model.add(Dense(2, activation='relu'))
model.compile(loss='mse', optimizer='adam', metrics=['mae'])
model.summary()
```

```
CPU times: user 5 µs, sys: 0 ns, total: 5 µs
```

```
120/120 [==============================] - 2s 15ms/step - loss: 0.0055 - accuracy: 0.7825 - val_loss: 0.0010 - val_accuracy: 0.4358
Epoch 43/50
120/120 [==============================] - 2s 15ms/step - loss: 0.0055 - accuracy: 0.7867 - val_loss: 9.6443e-04 - val_accuracy: 0.7095
Epoch 44/50
120/120 [==============================] - 2s 15ms/step - loss: 0.0055 - accuracy: 0.7692 - val_loss: 7.7792e-04 - val_accuracy: 0.5000
Epoch 45/50
120/120 [==============================] - 2s 16ms/step - loss: 0.0055 - accuracy: 0.7804 - val_loss: 0.0020 - val_accuracy: 0.3603
Epoch 46/50
120/120 [==============================] - 2s 15ms/step - loss: 0.0054 - accuracy: 0.7965 - val_loss: 8.1060e-04 - val_accuracy: 0.7263
Epoch 47/50
120/120 [==============================] - 2s 15ms/step - loss: 0.0055 - accuracy: 0.7825 - val_loss: 0.0019 - val_accuracy: 0.3659
Epoch 48/50
120/120 [==============================] - 2s 15ms/step - loss: 0.0055 - accuracy: 0.7825 - val_loss: 0.0014 - val_accuracy: 0.3659
Epoch 49/50
120/120 [==============================] - 2s 15ms/step - loss: 0.0054 - accuracy: 0.7797 - val_loss: 0.0020 - val_accuracy: 0.4385
Epoch 50/50
120/120 [==============================] - 2s 15ms/step - loss: 0.0054 - accuracy: 0.7832 - val_loss: 7.3981e-04 - val_accuracy: 0.7123
```

شکل ۴۱ – مدل GRU بدون drop out



شکل 42– نمودار خطا مدل GRU بدون drop out

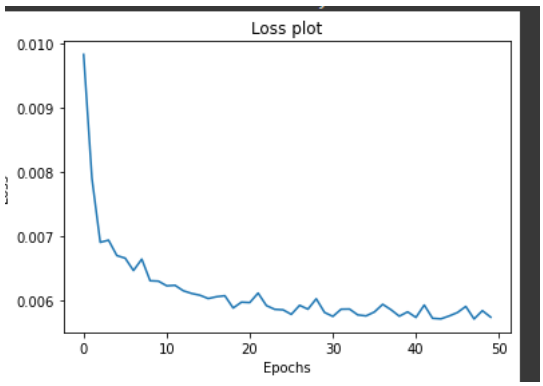
GRU با dropout:

زمان اجرا : 7.39میکرئ / دقت 83٪ /// loss :0.0052

```
[45] from keras.layers import GRU
     %time
     model = Sequential()
     model.add(GRU(50, batch_input_shape=(None, 29,12 ), recurrent_dropout=0.2))
     model.add(Dropout(0.3))
     model.add(Dense(2, activation='relu'))
     model.compile(loss='mse', optimizer='adam', metrics=["accuracy"])
     model.summary()

     CPU times: user 3 µs, sys: 0 ns, total: 3 µs
     Wall time: 6.2 µs
     Model: "sequential_6"
     _________________________________________________________________
      Layer (type)                 Output Shape              Param #
     =================================================================
      gru_2 (GRU)                  (None, 50)                9600

      dropout_4 (Dropout)          (None, 50)                0

      dense_5 (Dense)              (None, 2)                 102

     =================================================================
     Total params: 9,702
     Trainable params: 9,702
     Non-trainable params: 0
     _________________________________________________________________
```

شکل ۴۳ – شبکه GRUبا dropout

```
120/120 [==============================] - 3s 25ms/step - loss: 0.0059 - accuracy: 0.7860 - val_loss: 8.6501e-04 - val_accuracy: 0.5112
Epoch 38/50
120/120 [==============================] - 3s 25ms/step - loss: 0.0059 - accuracy: 0.7951 - val_loss: 9.3903e-04 - val_accuracy: 0.6872
Epoch 39/50
120/120 [==============================] - 3s 24ms/step - loss: 0.0058 - accuracy: 0.7860 - val_loss: 9.7734e-04 - val_accuracy: 0.3994
Epoch 40/50
120/120 [==============================] - 3s 25ms/step - loss: 0.0058 - accuracy: 0.8000 - val_loss: 0.0011 - val_accuracy: 0.3659
Epoch 41/50
120/120 [==============================] - 3s 24ms/step - loss: 0.0057 - accuracy: 0.7818 - val_loss: 0.0011 - val_accuracy: 0.7291
Epoch 42/50
120/120 [==============================] - 3s 25ms/step - loss: 0.0059 - accuracy: 0.7916 - val_loss: 7.3861e-04 - val_accuracy: 0.4972
Epoch 43/50
120/120 [==============================] - 3s 25ms/step - loss: 0.0057 - accuracy: 0.7853 - val_loss: 0.0010 - val_accuracy: 0.4050
Epoch 44/50
120/120 [==============================] - 3s 25ms/step - loss: 0.0057 - accuracy: 0.7860 - val_loss: 8.1015e-04 - val_accuracy: 0.4637
Epoch 45/50
120/120 [==============================] - 3s 25ms/step - loss: 0.0058 - accuracy: 0.7762 - val_loss: 9.3041e-04 - val_accuracy: 0.3994
Epoch 46/50
120/120 [==============================] - 3s 25ms/step - loss: 0.0058 - accuracy: 0.7734 - val_loss: 0.0011 - val_accuracy: 0.7374
Epoch 47/50
101/120 [========================>.....] - ETA: 0s - loss: 0.0058 - accuracy: 0.7756
```



شکل ۴۴ – نمودار خطا شبکه GRUبا dropout

RNN بدون dropout :

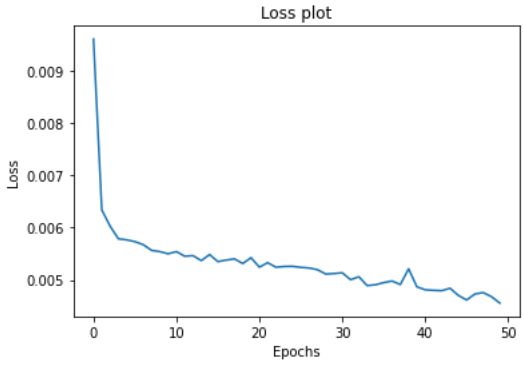
زمان اجرا : 8.58 میکرئ / دقت :76٪ /// loss : 0.004

```
%time
model = Sequential()
model.add(SimpleRNN(units=50, input_shape=x_train.shape[1:], activation="relu", recurrent_dropout=0.0))
model.add(Dense(2, activation='relu'))
model.compile(optimizer = 'adam',loss = 'mse',metrics = ["accuracy"])
model.summary()
```

```
CPU times: user 0 ns, sys: 5 µs, total: 5 µs
Wall time: 9.78 µs
Model: "sequential_8"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 simple_rnn_3 (SimpleRNN)    (None, 50)                3150

 dense_7 (Dense)             (None, 2)                 102

=================================================================
Total params: 3,252
Trainable params: 3,252
Non-trainable params: 0
_________________________________________________________________
```

شکل 45 – شبکه RNN بدون dropout

```
Epoch 42/50
120/120 [==============================] - 1s 8ms/step - loss: 0.0048 - accuracy: 0.7776 - val_loss: 9.7563e-04 - val_accuracy: 0.7179
Epoch 43/50
120/120 [==============================] - 1s 8ms/step - loss: 0.0048 - accuracy: 0.7804 - val_loss: 8.5957e-04 - val_accuracy: 0.4553
Epoch 44/50
120/120 [==============================] - 1s 7ms/step - loss: 0.0048 - accuracy: 0.7748 - val_loss: 8.9705e-04 - val_accuracy: 0.4609
Epoch 45/50
120/120 [==============================] - 1s 7ms/step - loss: 0.0047 - accuracy: 0.7762 - val_loss: 9.7893e-04 - val_accuracy: 0.4749
Epoch 46/50
120/120 [==============================] - 1s 8ms/step - loss: 0.0046 - accuracy: 0.7839 - val_loss: 0.0012 - val_accuracy: 0.5223
Epoch 47/50
120/120 [==============================] - 1s 7ms/step - loss: 0.0047 - accuracy: 0.7895 - val_loss: 0.0012 - val_accuracy: 0.7235
Epoch 48/50
120/120 [==============================] - 1s 7ms/step - loss: 0.0048 - accuracy: 0.7748 - val_loss: 0.0019 - val_accuracy: 0.3771
Epoch 49/50
120/120 [==============================] - 1s 8ms/step - loss: 0.0047 - accuracy: 0.7853 - val_loss: 8.1989e-04 - val_accuracy: 0.6313
Epoch 50/50
120/120 [==============================] - 1s 7ms/step - loss: 0.0046 - accuracy: 0.7825 - val_loss: 0.0011 - val_accuracy: 0.5670
```



شکل 46 – نمودار خطا شبکه RNN بذون dropout

RNN با drop out :

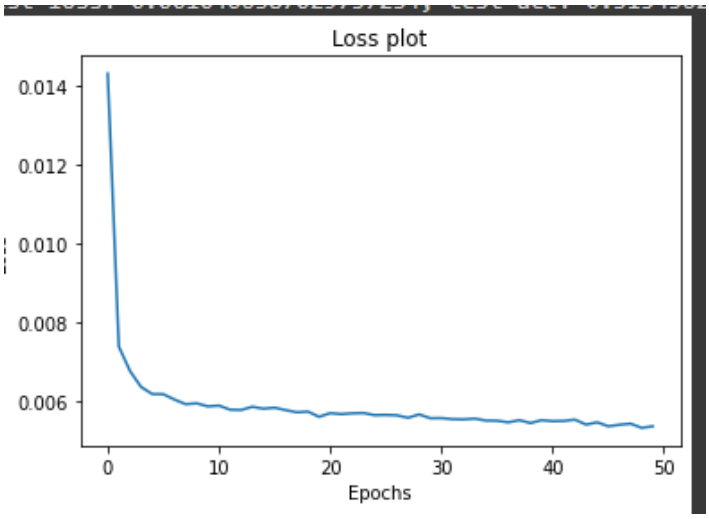
زمان اجرا : 7.87 میکرئ / دقت :79٪ /// loss :0.005

```
%time
model = Sequential()
model.add(SimpleRNN(units=50, input_shape=x_train.shape[1:], activation="relu", recurrent_dropout=0.2))
model.add(Dense(2, activation='relu'))
model.compile(optimizer = 'adam',loss = 'mse',metrics = ["accuracy"])
model.summary()
```

```
CPU times: user 4 µs, sys: 0 ns, total: 4 µs
Wall time: 8.58 µs
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 simple_rnn_1 (SimpleRNN)    (None, 50)                3150

 dense_4 (Dense)             (None, 2)                 102

=================================================================
Total params: 3,252
Trainable params: 3,252
Non-trainable params: 0
_________________________________________________________________
```

شکل 47 – شبکه RNNبا dropout

```
Epoch 43/50
120/120 [==============================] - 1s 9ms/step - loss: 0.0056 - accuracy: 0.8028 - val_loss: 0.0012 - val_accuracy: 0.4022
Epoch 44/50
120/120 [==============================] - 1s 8ms/step - loss: 0.0054 - accuracy: 0.7916 - val_loss: 0.0013 - val_accuracy: 0.4749
Epoch 45/50
120/120 [==============================] - 1s 9ms/step - loss: 0.0055 - accuracy: 0.7944 - val_loss: 8.3910e-04 - val_accuracy: 0.4050
Epoch 46/50
120/120 [==============================] - 1s 9ms/step - loss: 0.0054 - accuracy: 0.7916 - val_loss: 0.0011 - val_accuracy: 0.3883
Epoch 47/50
120/120 [==============================] - 1s 9ms/step - loss: 0.0054 - accuracy: 0.7930 - val_loss: 0.0011 - val_accuracy: 0.3827
Epoch 48/50
120/120 [==============================] - 1s 9ms/step - loss: 0.0054 - accuracy: 0.8119 - val_loss: 7.2457e-04 - val_accuracy: 0.5084
Epoch 49/50
120/120 [==============================] - 1s 9ms/step - loss: 0.0053 - accuracy: 0.7853 - val_loss: 9.4473e-04 - val_accuracy: 0.4078
Epoch 50/50
```



شکل 48 – نمودار خطا شبکه RNNبا dropout

جدول ۴ - مقایسه dropoutو بدون dropout

	زمان اجرا(میکروثانیه)	دقت	loss
LSTM(with dropout)	۱۰	۷۳٪	۰,۰۰۳
LSTM(WITHOUT dropout)	۸,۵۸	۵۴٪	۳۵۰۰
GRU(with dropout)	۷,۳۹	۸۳٪	۰,۰۰۵۱
GRU(without dropout)	۹,۵۴	۷۹٪	۰,۰۰۵۴
RNN(with dropout)	۷,۸۷	۷۹٪	۰,۰۰۵
RNN(without dropout)	۸,۵۸	۷۶٪	۰,۰۰۴

همانطور که در جدول ۴ میبینیم معمولا با اضافه کردن dropout دقت بالا میرود مقدار خطا کم میشود و زمان اجرا کاهش میابد .

در نتیجه مدل با dropout بهتر عمل میکند. ولی باید به نکات زیر با توجه به جدول دقت داشته باشیم زیرا که در شبکه ی LSTMتاثیر منفی و در بعضی پارامتر های GRUنیز تاثیر منفی کمی گذاشته علت این امر آن است که شبکه LSTMدر این سوال کم عمق هست و بنابراین تعداد نورون های کمی دارند و با افزودن dropoutآموزش شبکه به کندی صورت می گیرد .اگر دیتاست ما پیچیدگی بیشتری داشت، برای آموزش شبکه از شبکه ها پیچیده تر با عمق بیشتری استفاده می کردیم(نورون های بیشتری داشتیم) و افزودن dropoutبه فراگیرتر شدن مدل ما کمک میکند و همچنین میدانیم که شبکه های LSTMو GRUنسبت به حالت اولیه اشان نمی توانند بهبود زیادی داشته باشند و توان آن ها در همین حدود است.

سوال ۲ – عنوان سوال

قسمت الف:

در این سوال ابتدا فایل متنی داده شده را آپبلود کردم. برای پیش پردازش متن، ابتدا تمام کاراکترهای متن را به lowercase تبدیل کردم. این کار باعث می شود که داده اضافی نداشته باشیم. هدف از این سوال این است که ببینیم مدل طراحی شده تا چه میزان می تواند حرف بعدی را درست پیش بینی کند و کوچک یا بزرگ بودن حروف، تاثیری در این موضوع ندارد. در مرحله بعد تمام کراکترهایی که حروف و یا عدد نبودند، حذف کردم. سئس با استفاده از دستور tokenize متن حاصل را به لیستی از کلمات تبدیل کردم و مجددا با چسباندن این لیست به هم، متن جدیدی ساختم. این کار باعث می شود که کراکترهای اضافی باعث اشتباه کردن مدل نشوند. در ادامه نیز از تمام کاراکترهای به کار رفته در متن، یک dictionary ساختم و متن را به sequence هایی با طول ۳۰ حرف تقسیم کردم.

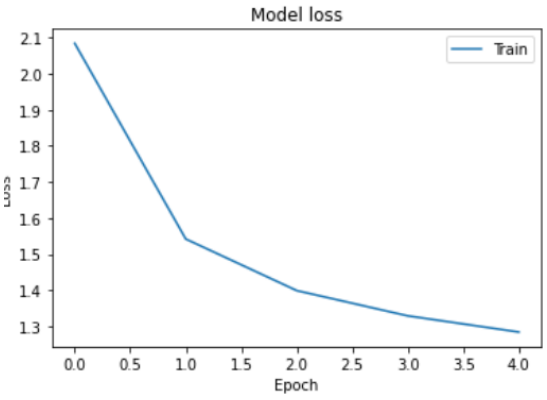
مدل طراحی شده به صورت زیر است:

```
model = Sequential()
model.add(layers.LSTM(512, input_shape=(X.shape[1], X.shape[2]), return_sequences=True))
model.add(layers.Dropout(0.2))
model.add(layers.LSTM(512))
model.add(layers.Dropout(0.2))
model.add(layers.Dense(256, activation='tanh'))
model.add(layers.Dense(Y.shape[1], activation='softmax'))
model.summary()
model.compile(optimizer='Adam', loss = 'kl_divergence', metrics=['accuracy'])
```

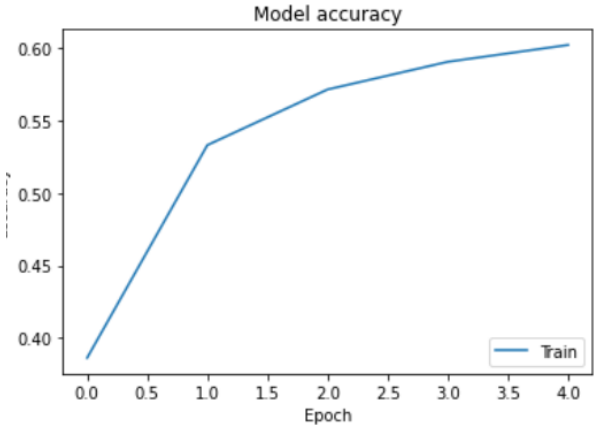
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 30, 512)           1052672

 dropout (Dropout)           (None, 30, 512)           0

 lstm_1 (LSTM)               (None, 512)               2099200

 dropout_1 (Dropout)         (None, 512)               0

 dense (Dense)               (None, 256)               131328

 dense_1 (Dense)             (None, 35)                8995

=================================================================
Total params: 3,292,195
Trainable params: 3,292,195
Non-trainable params: 0
```

به دلیل زمان زیاد ران شدن هر ایپاک، فقط ۵ ایپاک برای آموزش دادن شبکه در نظر گرفته شد. نمودارهای خطا و درستی شبکه به صورت زیر هستند.

مشخصات شبکه:



نمودار خطای شبکه



نمودار درستی شبکه

توضیح کد تولید متن: در این قسمت برای تولید متنی ۲۰۰ کلمه ای، یک تابع تعریف کردم. در این تابع ابتدا متنی به طول ۳۰ که همان طول در نظر گرفته شده برای sequence ها برای آموزش شبکه بود، به صورت رندوم از مجموعه x انتخاب می شود(.(seedبعد از تغییر اندازه این متن، آن را به شبکه می دهیم تا حرف بعدی را پیش بینی کند. پیش بینی در واقع بر این اساس است که حرفی را انتخاب می کنیم که بیشترین احتمال وقوع را داشته باشد. در انتهای حلقه ۲۰۰ تایی نیز یک حرف از ابتدای متن seed کم کرده و حرف پیش بینی شده را به انتهای آن اضافه می کنیم تا پیش بینی در مرحله بعد، با متن آپدیت شده باشد.

Optimizer	Adam
Loss function	kl_divergence
Batch size	128

نتیجه:

...Seed Text: found out about the tournamen

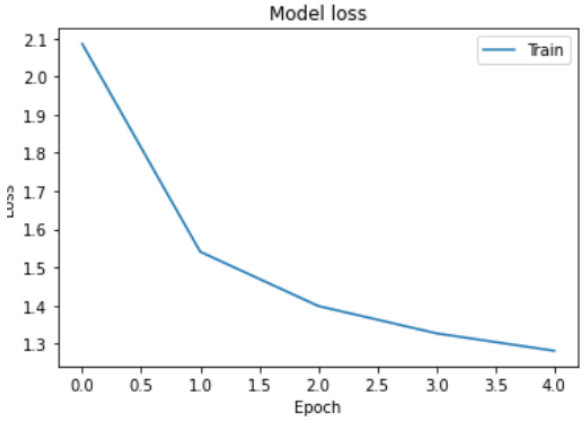
...Generated: found out about the tournament i was sure you d be a seat said harry she said so the second task he s a second the second task harry said harry s face and said so the second task he s a second the second task harry said harry s f

قسمت ب:

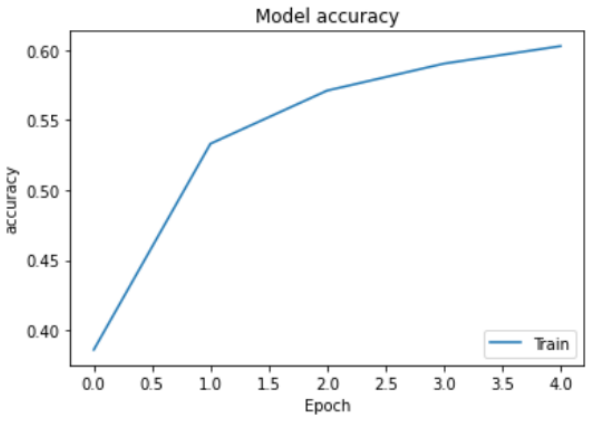
Optimizer	Adam
Loss function	categorical_crossentropy
Batch size	128

در این بخش دو تابع خطای دیگر را بررسی می کنیم. توابع categorical_crossentropy و categorical hinge.

حالت اول:



نمودار خطای شبکه



نمودار درستی شبکه

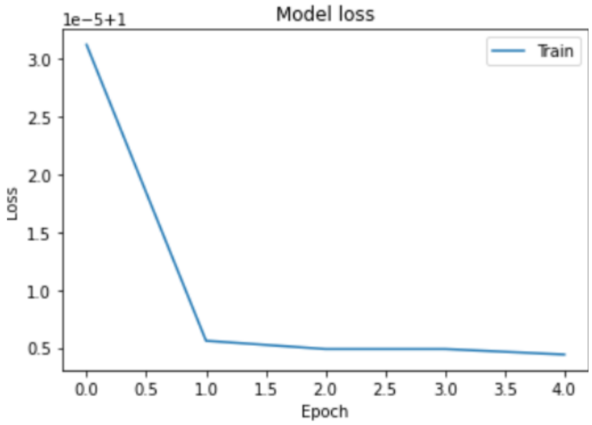
نتیجه شبکه:

...Seed Text: he reappeared looking entirely

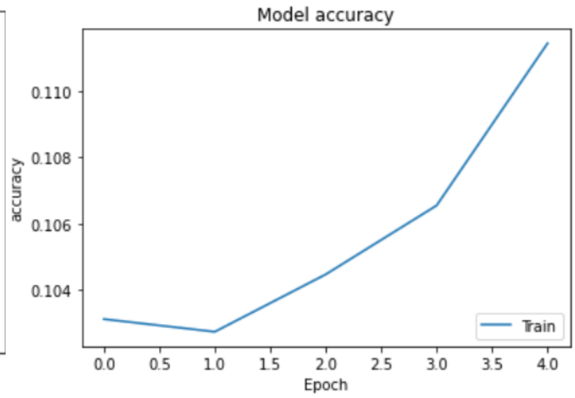
...Generated: he reappeared looking entirely and several parchment and seemed to be a look of the skytherins were still since they were still since they were still since they were still since they were still since they were still since they wer

حالت دوم:

Optimizer	Adam
Loss function	categorical_hinge
Batch size	128



نمودار خطای شبکه



نمودار درستی شبکه

تیجه شبکه:

...Seed Text: l posts a rainbow arced sudden

...Generated: l posts a rainbow arced sudden   u u u u u u u u u u u u u u u u u u u u u u u u u u u u u u u u u u u u u u u u u u u u u u u u u u u u u u u u u u u u u u u u u u u u u u u

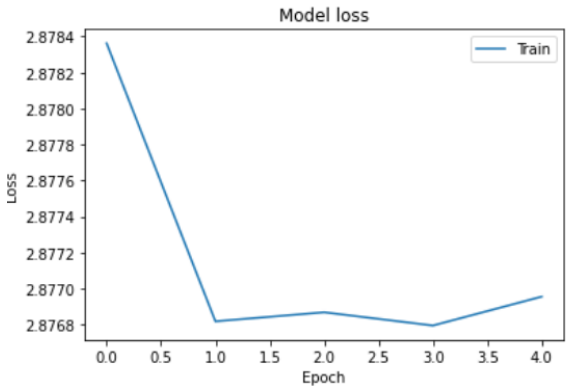
با توجه به نتایج هر سه تابع خطایی که استفاده کردیم، می بینیم که به وضوح تاب خطای categorical- عملکرد خوبی ندارد و درصد درستی آن تقریبا ۱۱٪ بود. در میان این سه، تابع hinge را به عنوان بهترین تابع خطا انتخاب می کنیم اگرچه این تابع و تابع categorical_crossentropy عملکردی بسیاری شبیه به هم داشتند و درصد درستی هردوی آن ها در ایپاک آخر kl_divergence .تقریبا ۶۰٪ بود

قسمت ج:

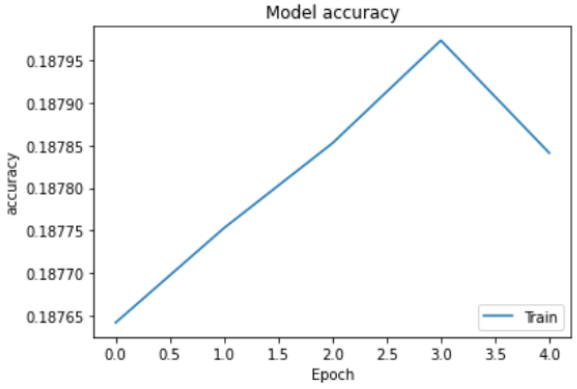
در این بخش ابتدا دو تابع بهینه ساز جدید را امتحان می کنیم. بهترین تابع بهینه ساز را نیز برای دو مقدار متفاوت batch size امتحان می کنیم تا به بهترین مدل برسیم.

مرحله ۱:

Optimizer	RMSprop
Loss function	categorical_hinge
Batch size	128



نمودار خطای شبکه



نمودار درستی شبکه

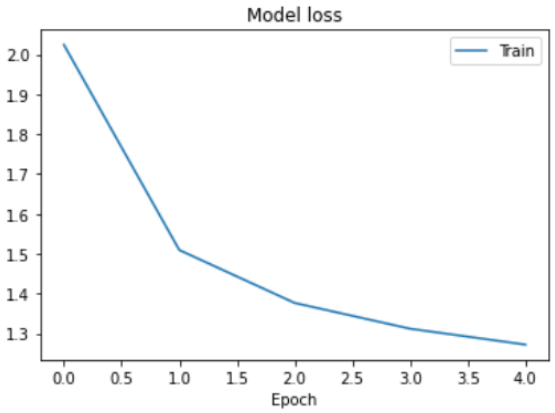
نتیجه شبکه:

...Seed Text: id the teachers they kept prac

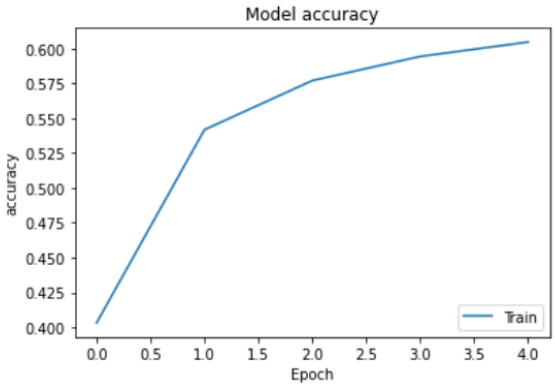
...Generated: id the teachers they kept prac

مرحله ۲:

Optimizer	Nadam
Loss function	categorical_hinge
Batch size	128



نمودار خطای شبکه



نمودار درستی شبکه

نتیجه شبکه:

...Seed Text: breath in a long low hiss and

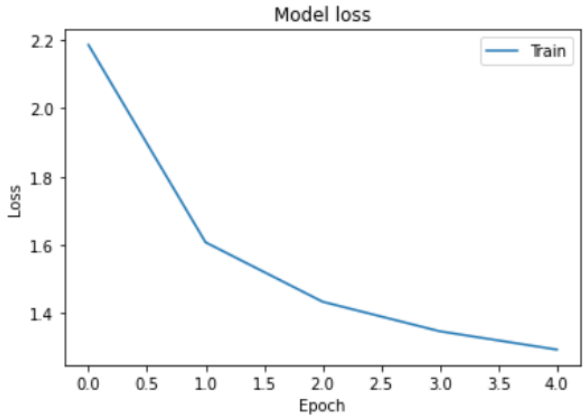
...Generated: breath in a long low hiss and the start of the corridor was staring at the corridor and the start of the corridor was staring at the corridor and the start of the corridor was staring at the corridor and the start of the corridor

در میان سه تابع بهینه ساز امتحان شده Adam, Nadam, RMSprop از نتایج نمودارهای خطا و درستی و متن نتیجه، می بینیم که تابع بهینه ساز Adam عملکرد بهتری دارد.

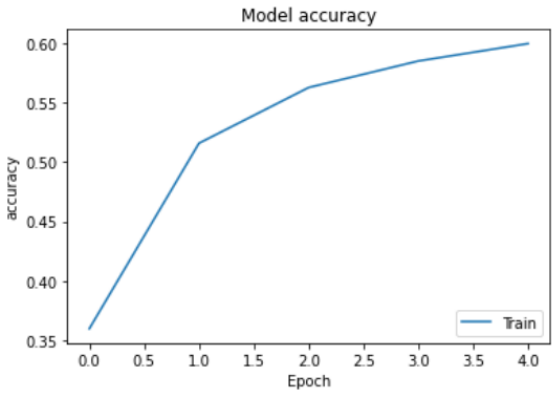
حالا دو مقدار متفاوت ۶۴ و ۲۵۶ را برای batch size انتخاب می کنیم.

مرحله ۳:

Optimizer	Nadam
Loss function	categorical_hinge
Batch size	256



نمودار خطای شبکه



نمودار درستی شبکه

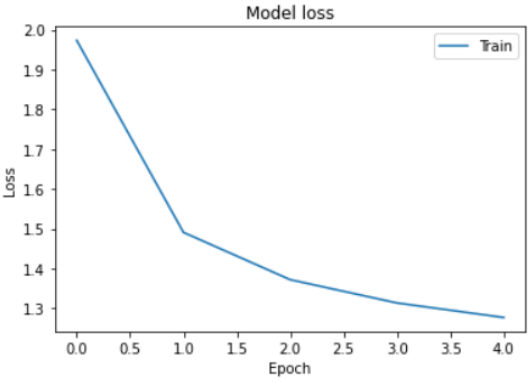
نتیجه متن:

...Seed Text: isibllity cloak he saw her zoo

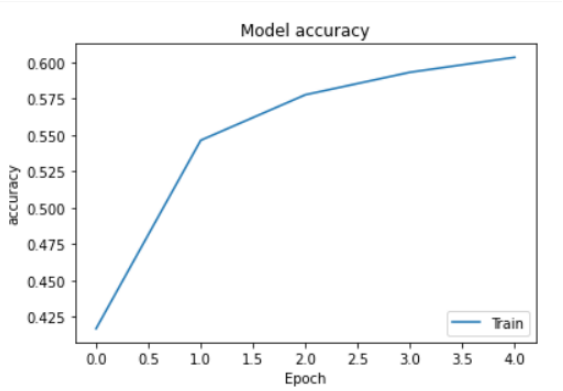
...Generated: isibllity cloak he saw her zooming around the cark arts the triwizard tournament the triwizard tournament the triwizard tournament the triwizard tournament the triwizard tournament the triwizard tournament the triwizard tournament

مرحله ۴:

Optimizer	Nadam
Loss function	categorical_hinge
Batch size	۶۴



نمودار خطای شبکه



نمودار درستی شبکه

نتیجه متن:

...Seed Text: y wouldn t want to work for an

...Generated: y wouldn t want to work for an extremely students before the standing students where he was standing the standing students and the standing students and the standing students and the standing students and the standing students and

همانطور که مشاهده می شود با تغییر تعداد ایپاک ها، نتیجه خطا و درستی تغییر چندانی نمی کند. اما به نظر می رسد که متن تولید شده نهایی عملکرد بهتری دارد.

نکته: در تمام متن های تولیده شده، می بینیم که از جایی به بعد کاراکترها و کلمات تکراری تولید می شوند. این موضوع به شبکه طراحی شده برمی گردد. برای حل این مشکل می توان تعداد dropout های مدل را کمتر کرد. با توجه هب زمان زیاد ران شدن هر شبکه، به مدل طراحی شده اکتفا کردم اما جای بهبود دارد.