به نام خدا

دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

شبکه های عصبی و یادگیری عمیق

مینی پروژه 2

نام و نام خانوادگی	مهسا مسعود – امید واهب
شماره دانشجویی	810196582 -810196635
تاریخ ارسال گزارش	1400/4/ 5

فهرست گزارش سوالات

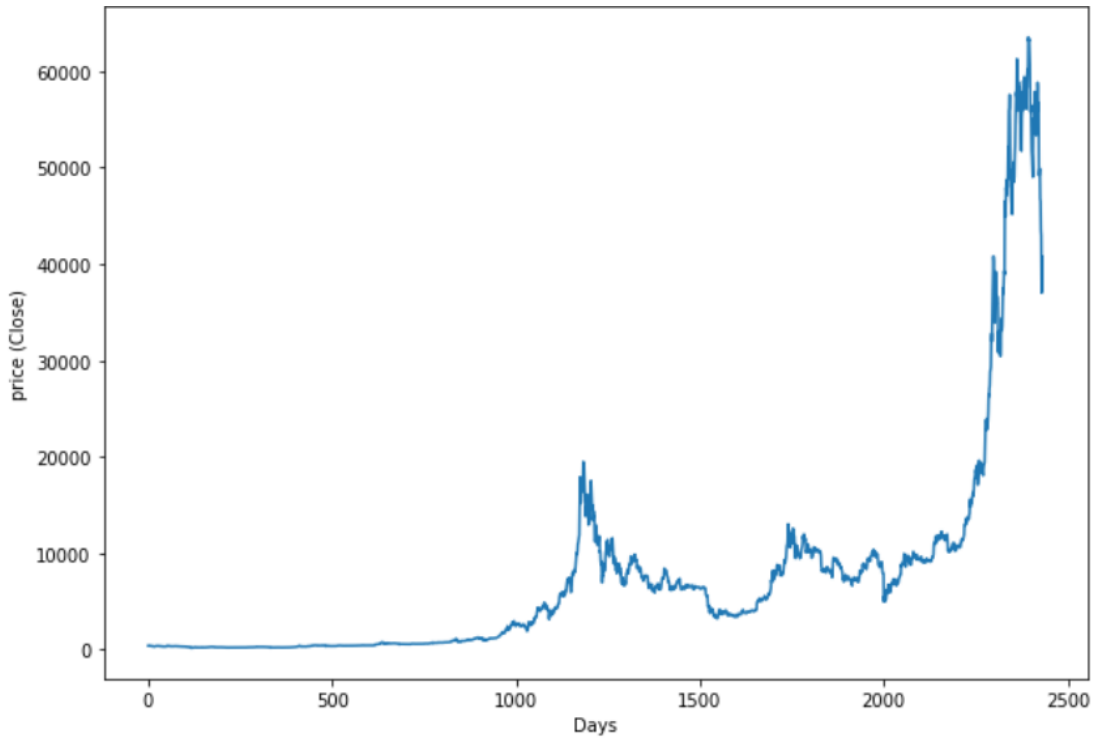
سوال 1 – آشنایی با کاربرد (شبکه های عصبی بازگشتی) در سری زمانی

در این قسمت به صورت چکیده هدف از این سری تمرین برای سوال اول آشنایی با سری های زمانی و کاربرد شبکه های LSTM و GRU و RNN در بازارهای بیت کوین بود.

ابتدا کتابخانه های لازم لود می شوند و تابع موردنظر برای لود داده ها نوشته می شود.

1)

مقادیر close به صورت زیر نمایش داده می شوند:



شکل 1-1 – مقدار close برای تمام روز ها

همانطور که مشاهده می شود، قیمت سهام با روز رسم شده است

```
''' data and target '''
data = df2[['High', 'Low', 'Open', 'Volume','Close']].to_numpy()
```

```
'''Scale data '''
from sklearn import preprocessing

# MinMax
scaler = preprocessing.MinMaxScaler(feature_range=(0, 1))
data = scaler.fit_transform(data)
```

در کد فوق، داده ها خوانده شده و نرمالایز می شوند. (از minmaxscaler برای نرمالیزیشن استفاده شده است.)

سپس داده های کل با یک window با پریود25 روز ایجاد می شود. برای ایجاد این پنجره 25 روزه نیز تابع data_for_trainig نوشته شده است. ایجاد می شوند و در بخش split داده های 10 درصد از روزهای انتهایی به عنوان داده تست جدا می گردند.

ابتدا از شبکه LSTM استفاده میکنیم.

سپس مدل ساخته و آموزش داده می شود. پارامتر ها نیز در تصویر و کد مشخص می باشند:

```
model = Sequential()
model.add(LSTM(50, batch_input_shape=(None, 24, 5), return_sequences=True, recurrent_dropout=0))
model.add(LSTM(50, return_sequences=False, recurrent_dropout=0))
model.add(Dense(2, activation='relu'))
model.compile(loss='mse', optimizer='adam', metrics=['mae'])
model.summary()
```

```
Model: "sequential_67"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_8 (LSTM)                (None, 24, 50)            11200
_________________________________________________________________
lstm_9 (LSTM)                (None, 50)                20200
_________________________________________________________________
dense_107 (Dense)            (None, 2)                 102
=================================================================
Total params: 31,502
Trainable params: 31,502
Non-trainable params: 0
_________________________________________________________________
```

در کد فوق از دو لایه LSTM استفاده شده است (می توانست حتی تک لایه هم باشد) که هر کدام دارای 50 یونیت می باشند. در لایه آخر نیز برای خروجی دو شرکت یک لایه dense با دو نورون قرار داده شده است. تابع loss نیز mse می باشد.

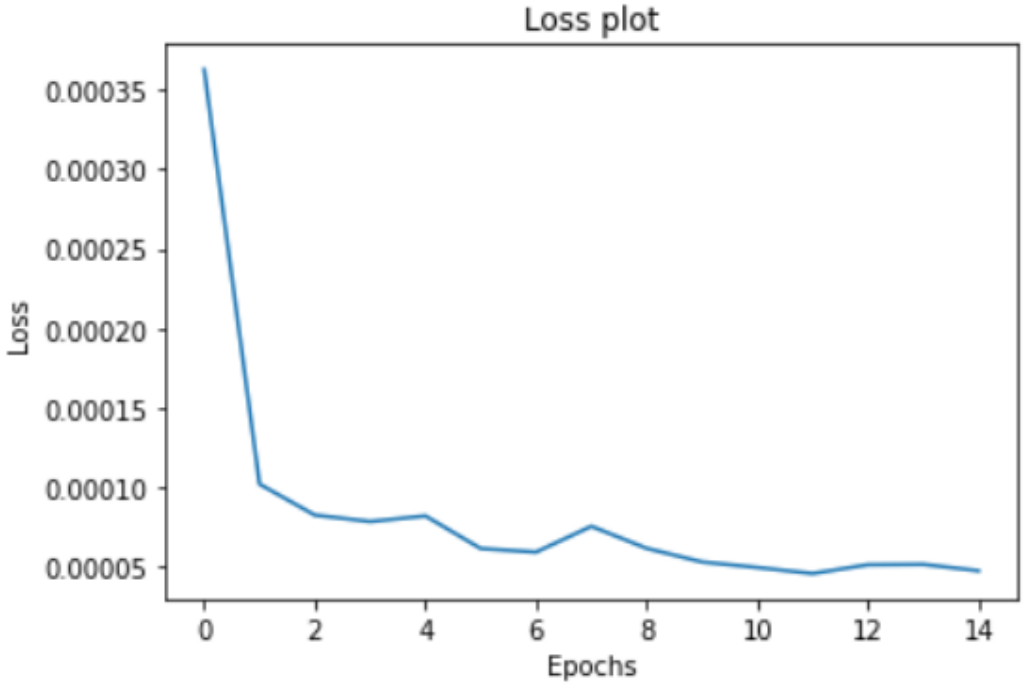
زمان اجرای عملیات:

11.4 میکرو ثانیه

کمترین loss در ایپاک اول : 0.0001

سپس نتایج Visualize می گردند:

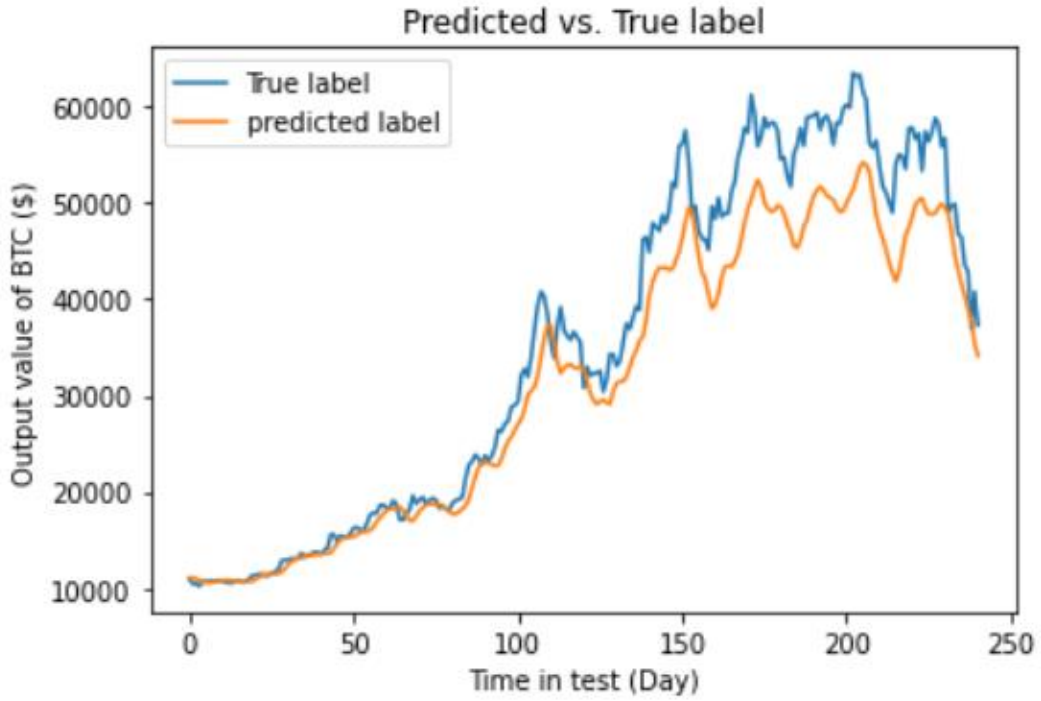
نمودار مقدار Loss:



شکل 1-2 – مقدار loss در LSTM

همانطور که مشاهده می شود، مقدار loss در همان epoch های اول به سرعت کاهش می یابد.

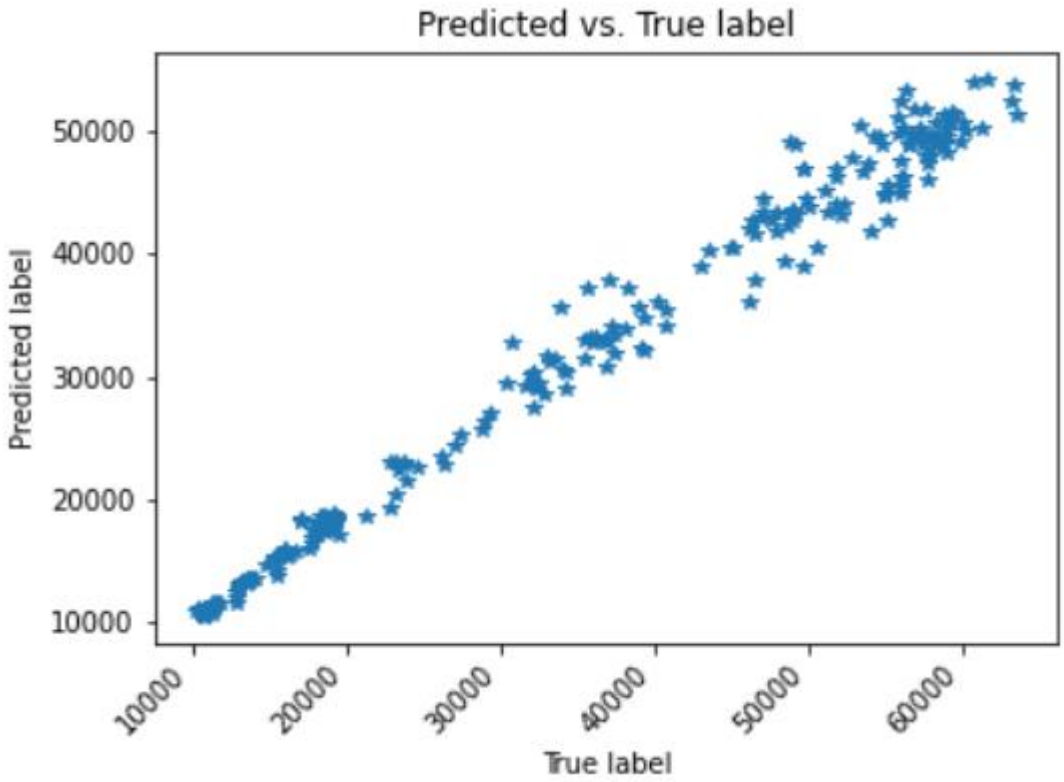
نمودار مقدار پیشبینی شده و مقدار واقعی برای ده درصد روزهای آخر:



شکل 1-3 – مقدار پیش بینی شده و مقدار واقعی در LSTM

همانطور که مشاهده می شود، خروجی نمودار از لحاظ trend شبیه می باشد.

در انتها نیز برای بررسی دقیقتر موضوع، نمودار داده های پیشبینی شده و مقدار واقعی را رسم کردیم که نتیجه به صورت زیر می باشد:



شکل 1-4 – مقدار پیش بینی شده و مقدار واقعی در LSTM به صورت scatter

محور x مقدار True label و محور y مقدار پیشبینی شده می باشد، همانطور که مشاهده می شود، نمودار نزدیک خط y=x می باشند و پیشبینی مناسبی صورت گرفته است.

. برای GRU داریم:

```
model = Sequential()
model.add(GRU(100, batch_input_shape=(None, 24,5 ), recurrent_dropout=0))
model.add(Dense(2, activation='relu'))
model.compile(loss='mse', optimizer='RMSprop', metrics=['mae'])
model.summary()
```

```
Model: "sequential_68"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
gru_10 (GRU)                 (None, 100)               32100
_________________________________________________________________
dense_108 (Dense)            (None, 2)                 202
=================================================================
Total params: 32,302
Trainable params: 32,302
Non-trainable params: 0
```

کد موردنظر برای این بخش به صورت فوق می باشد. با درنظر گرفتن 50 یونیت و تابع فعالساز relu و loss که mse است مدل را ساخته و کامپایل می کنیم.

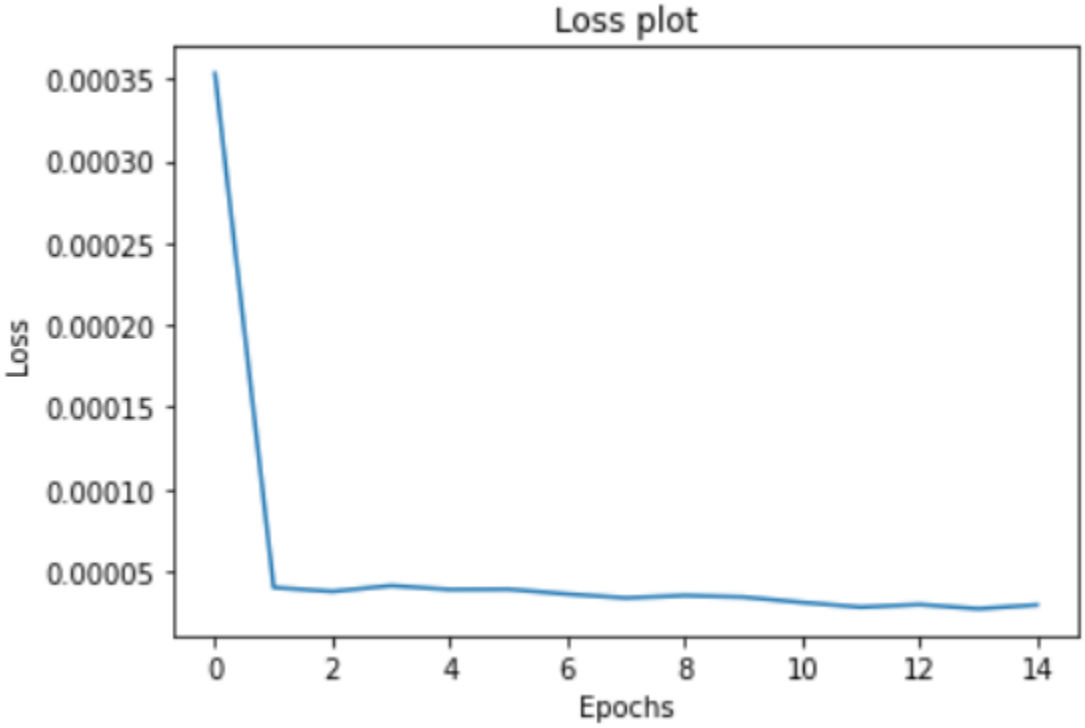
نتایج به دست آمده به صورت زیر می باشند:

زمان اجرای عملیات:

6.77 میکرو ثانیه

کمترین loss در ایپاک اول : 0.00005

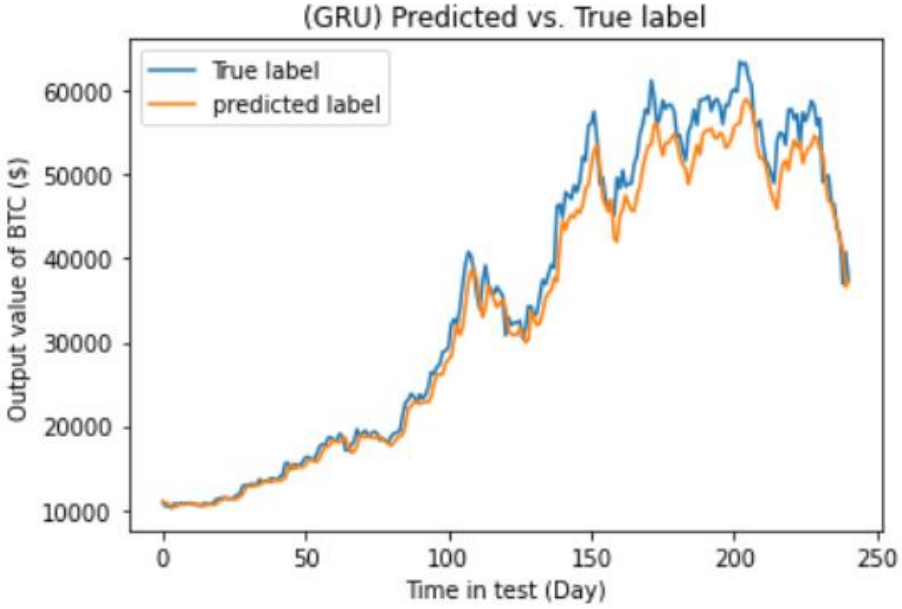
برای Loss داریم:



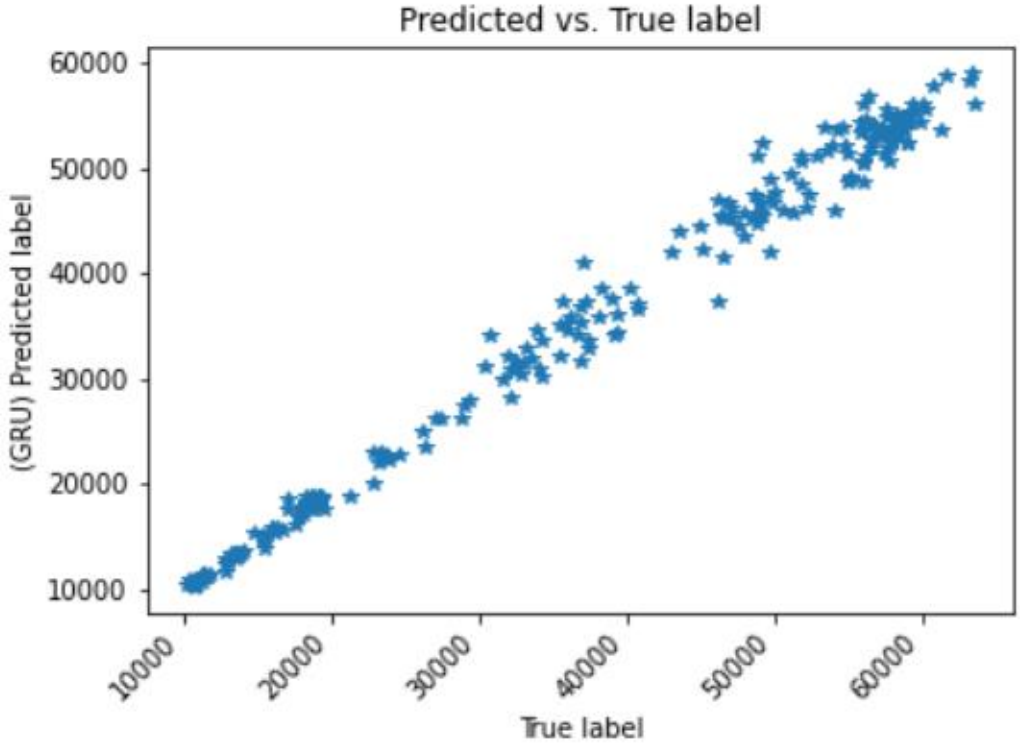
شکل 1-5 – مقدار loss در GRU

همانطور که مشخص است، Loss الگوریتم GRU کمتر از LSTM می باشد و بهینه تر است.

برای پیشبینی ها داریم:



شکل 1-6 – مقدار پیش بینی شده و حقیقی در GRU



شکل 1-7 – مقدار پیش بینی شده و حقیقی در GRU در scatter plot

همانطور که مشخص است، پیشبینی GRU بهتر از LSTM عمل کرده است.

حال کد فوق را برای RNN اجرا می کنیم.

برای RNN کد q1_rnn.py زده شده است.

برای مدل RNN داریم:

```
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()
```

```
Model: "sequential_69"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
simple_rnn_5 (SimpleRNN)     (None, 100)               10600
_________________________________________________________________
dense_109 (Dense)            (None, 2)                 202
=================================================================
Total params: 10,802
Trainable params: 10,802
Non-trainable params: 0
_________________________________________________________________
```

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

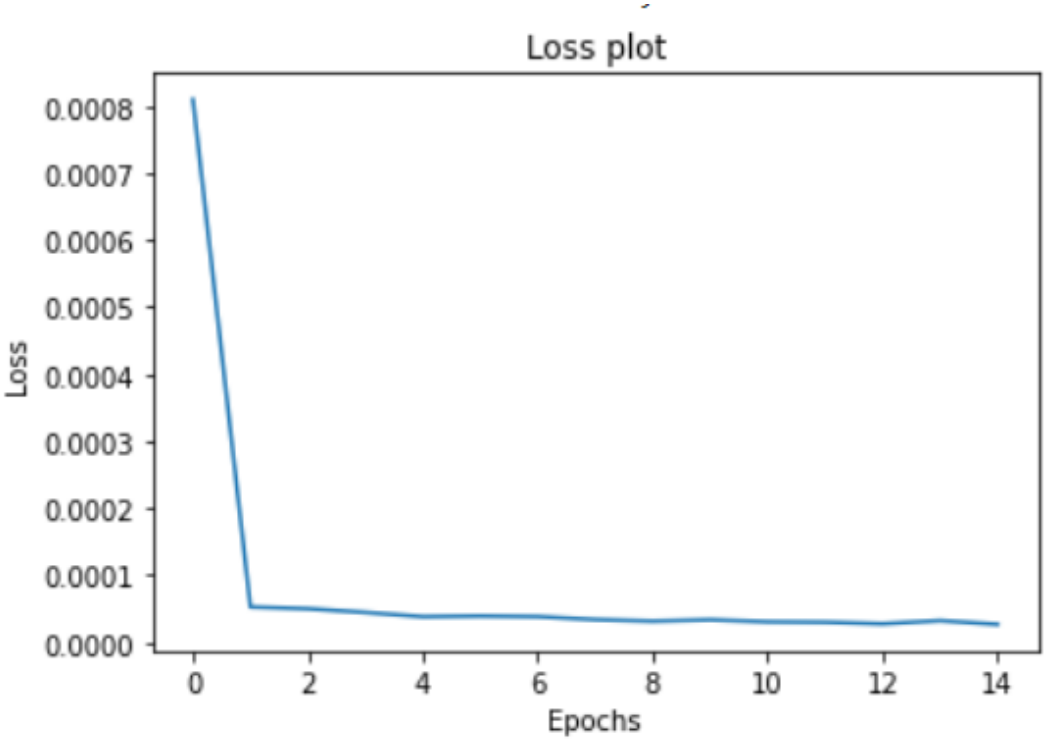
زمان اجرای عملیات:

5.96 میکرو ثانیه

کمترین loss در ایپاک اول : 0.0001

خروجی نیز به صورت زیر می باشد:

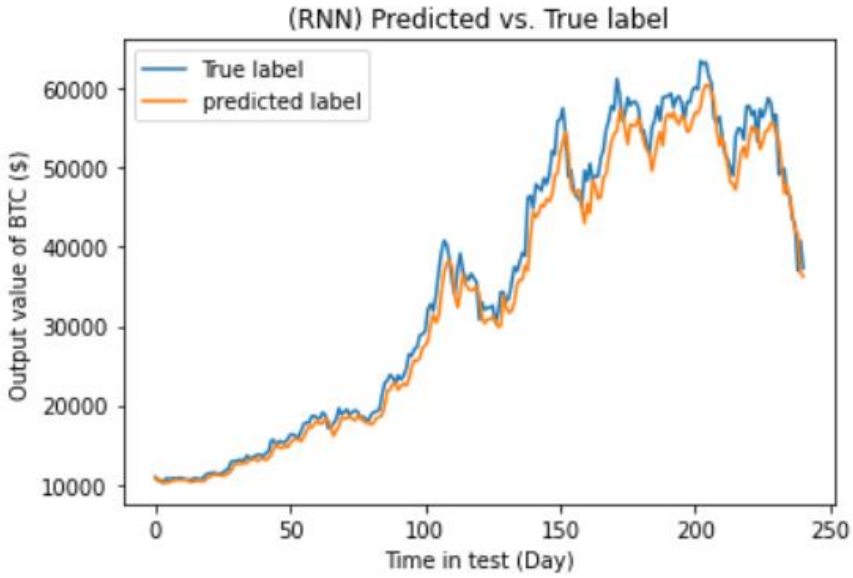
مقدار LOSS:



شکل 1-8 – مقدار loss در RNN

همانطور که مشاهده می شود، مقدار Loss الگوریتم RNN از هر دو الگوریتم دیگر کمی بدتر می باشد.

مقدار پیشبینی شده:



شکل 1-9 – مقدار پیش بینی شده و حقیقی در RNN

جدول 1-1- مقایسه عملکرد شبکه ها

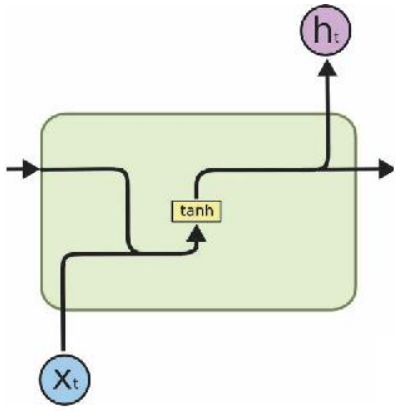
شبکه مورد نظر	Time (میکرو ثانیه)	Test loss	خطای mae بعد از 15 ایپاک
LSTM	11.4	0.0093	0.0702
GRU	6.77	0.0020	0.0318
RNN	5.96	0.0029	0.0323

. مقایسه سه ماژول RNN، LSTM و GRU

در طراحی شبکه‌های عصبی با هدف تداعی کردن یک پترن، نیاز به حافظه است. حال پترن ورودی می‌تواند به صورت explicit یا implicit باشد. منظور از ورودی implicit آن است که ورودی‌ها توسط یک اردر زمانی یا مکانی به شبکه داده می‌شوند. درغیراینصورت ورودی‌ها explicit است. طراحی یک شبکه عصبی برای ورودی‌های explicit ساده‌تر از طراحی شبکه برای داده‌های implicit است و تنها به حافظه استاتیکی نیاز دارند. در مسائل واقعی و پرچالش‌تر ابعاد داده‌های ورودی ثابت نبوده و وجود نویز غیرقابل انکار است. دراین‌گونه مسائل که ورودی‌ها implicit هستند، از شبکه‌های recurrent استفاده می‌شود. در شبکه‌های recurrent پس از طی کردن مسیر feedforward در راستای ساخت خروجی، آنرا به ورودی اعمال می‌کنند. در اینحالت خروجی همواره از exogenous input بهره می‌برد تا تداعی را به‌درستی انجام دهد. قاعده یادگیری در شبکه‌های RNN، gradient updating rule است. در ادامه سه نمونه از شبکه‌های recurrent بررسی خواهد شد.

- **سلول RNN**

در سلول RNN ورودی (input) در لحظه کنونی با خروجی (hidden state) لحظه قبل ترکیب شده و پس از عبور از تابع فعال‌ساز tanh، hidden state لحظه کنونی یا همان حافظه را می‌سازند. تابع tanh برای کنترل فلو اطلاعاتی در شبکه استفاده می‌شود. در ادامه یک سلول RNN آمده است.



شکل 10 – سلول RNN

پیشتر گفته شد که در شبکه‌های RNN از قاعده یادگیری گرادیان استفاده می‌کنند. بنابراین با افزایش طول ورودی‌های implicit، مسیرهای برگشتی طولانی‌تر شده و محاسبات سخت خواهد شد. همچنین ازآنجایی‌که از توابع فعال‌ساز با مشتق زیر یک استفاده می‌شود، در ذنجیره‌های طولانی، مولفه خطای برگشتی برای وزن‌هایی که در زمان‌های دور هستند، کوچک می‌شود و واکشی اطلاعات از لحظه کنونی برای به‌روزرسانی وزن‌های متاثر از داده‌های قدیمی ضعیف خواهد بود.(مشکل vanishing gradient) برای حل این مشکل می‌توان از توابع فعال‌سازی همچون ReLU بهره برد اما این تابع نیز برای مقادیر نامثبت، مشکل‌ساز می‌شود و باید به دنبال ساز و کارهایی غیر از توابع فعال‌ساز رفت. درواقع شبکه‌های RNN به دلیل سادگی، برای کاربردهایی که نیاز به short depency دارند، استفاده می‌شوند.

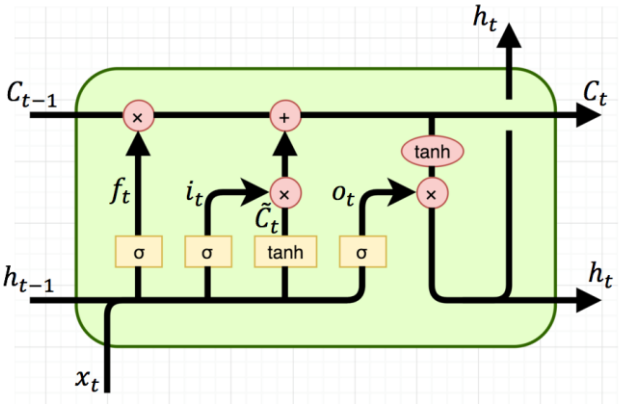
علت بدتر عمل کردن این شبکه ها هم این است که دارای long term memory نیستند.

- **سلول LSTM**

برای حل مشکل بازیابی اطلاعات برای داده‌های با فاصله طولانی و برقراری long dependency استفاده از ماژول LSTM پیشنهاد می‌شود. LSTM اجازه می‌دهد داده‌ها از زمان‌های دور ذخیره گردند. به عنوان مثال برای پردازش یک پاراگراف در راستای تولید متن، RNN کلاسیک اطلاعات مهمی که در ابتدای متن هستند را درنظر نمی‌گیرد. حال سلول‌هایی همانند LSTM باعث می‌شوند مشکل short-

term memory حل شود؛ زیرا با وجود مکانیزم‌های داخلی (گیت‌ها) فلو اطلاعاتی را کنترل می‌کنند.

در ادامه یک سلول LSTM آمده است.



شکل 11– سلول LSTM

سلول LSTMهمانند RNN، فلو اطلاعاتی را کنترل می‌کند و در مسیر forward انتشار می‌دهند با این تفاوت که در سلول LSTM عملیات متفاوتی انجام می‌شود. این عملیات به LSTM اجازه می‌دهند تا اطلاعات را حفظ و یا پاک کنند. هسته مرکزی LSTM درواقع cell state و 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 SIGMOID و tanh در آن است که SIGMOID خروجی را بین ۰ و ۱ می‌برد. بنابراین اگر اطلاعاتی باید فراموش گردد در صفر ضرب شده و حذف می‌گردد. در LSTM سه گیت مختلف وجود دارد که فلو اطلاعاتی را کنترل می‌کنند. گیت فراموشی، گیت ورودی و گیت خروجی.

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

گیت ورودی: کاربرد این گیت در راستای update کردن cell state است. اطلاعات input کنونی و hidden state قبلی ترکیب شده و به sigmoid اعمال می‌شوند. حال sigmoid تصمیم می‌گیرد

که چه اطلاعاتی باید update شوند. همچنین ترکیب input کنونی و hidden state قبلی وارد یک tanh شده و خروجی sigmoid و tanh یا یکدیگر ضرب می‌شوند. sigmoid تصمیم می‌گیرد که چه اطلاعاتی مهم بوده و باید حفظ شوند.

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

گیت خروجی: این گیت تصمیم می‌گیرد که hidden state بعدی چه باید باشد. درواقع hidden state دارای اطلاعاتی از ورودی‌های قبلی است و برای پیشبینی نیز استفاده می‌شود. عملکرد این گیت بدین صورت است که hidden state قبلی و ورودی کنونی به تابع sigmoid داده شده و cell state جدید به تابع tanh اعمال می‌شود. حال خروجی tanh و sigmoid ضرب می‌شوند تا اطلاعاتی که hidden state باید داشته باشد مشخص گردد. درواقع خروجی سلول، hidden state است.

به طور کلی گیت فراموشی تصمیم می‌گیرد که چه اطلاعاتی از حال‌های قبلی باید حفظ شود. گیت ورودی تصمیم می‌گیرد چه اطلاعاتی از ورودی جدید با حالت‌های قبل مرتبط بوده است. گیت خروجی نیز hidden state بعدی را مشخص می‌کند.

درواقع سلول LSTM در مقایسه با RNN، از درجه آزادی بیشترب برخوردار است و امکان ترکیب ورودی‌ها با داده‌های بیشتری وجود داشته که منجر به کنترل بهتر خروجی‌ها نیز می‌شود. بنابراین سلول LSTM کنترل بهتر پارامترها و درنتیجه نتایج بهتر را به ارمغان می‌آورد اما هزینه آن پیچیدگی و عملیات بیشتر است.

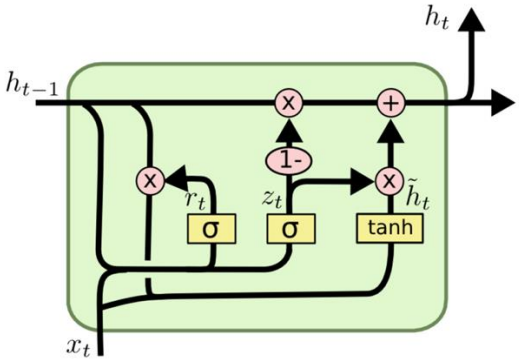
- **سلول GRU**

سلول GRU نسل جدیدی از شبکه‌های عصبی recurrent است و شباهت زیادی با LSTM دارد. در GRU، cell state حذف شده و از hidden state برای انتقال اطلاعات استفاده می‌کند. همچنین گیت‌های آن update و reset هستند (یک گیت کمتر از LSTM).

گیت update: این گیت شبیه به گیت فراموشی و گیت ورودی LSTM عمل می‌کند و تصمیم می‌گیرد که چه اطلاعاتی حذف و یا اضافه شود.

گیت reset: گیت ریست نیز تصمیم می‌گیرد که چه اطلاعات گذشته‌ای باید حذف شوند.

درواقع سلول GRU، محاسبات کمتری داشته که منجر می‌شود در مقایسه با LSTM دارای سرعت بیشتری باشد. البته هر یک از GRU و LSTM بسته به کاربدر ممکن است بهتر از دیگری باشد. در ادامه یک سلول GRU قابل مشاهده است.



شکل 12– سلول GRU

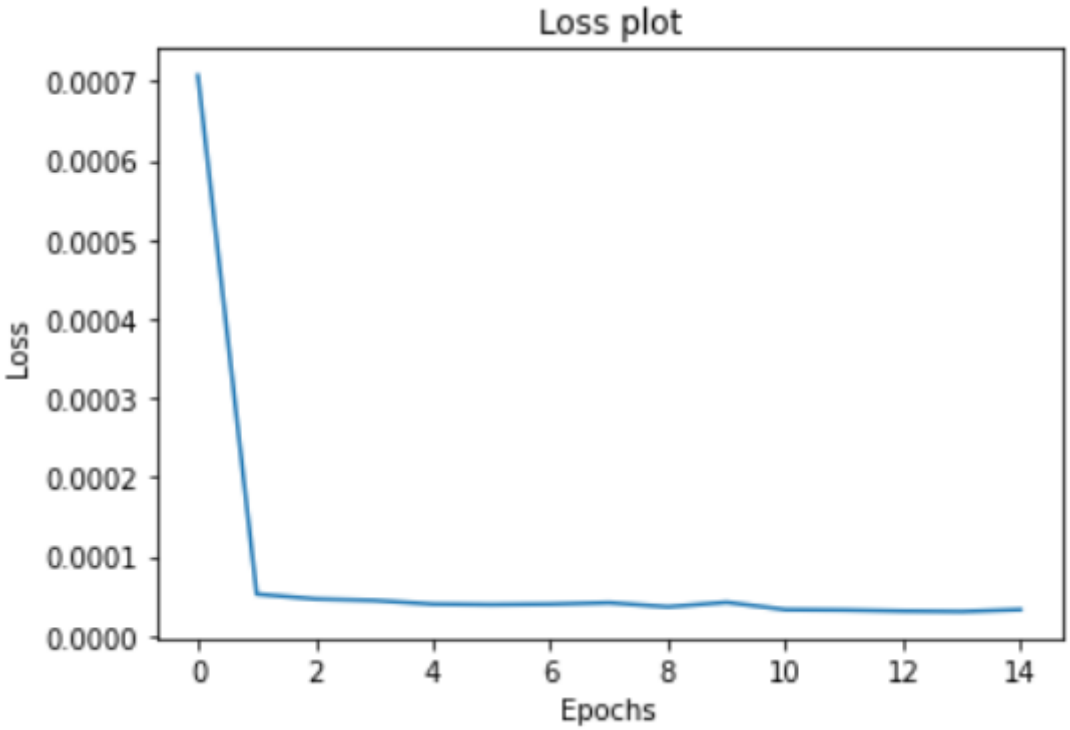
باتوجه به نتایج سه الگوریتم، به نظر می رسد الگوریتم GRU بهترین نتیجه را ارائه کرده است.

3)

در این قسمت به بررسی اثر optimizer ها و loss function های مختلف می پردازیم.

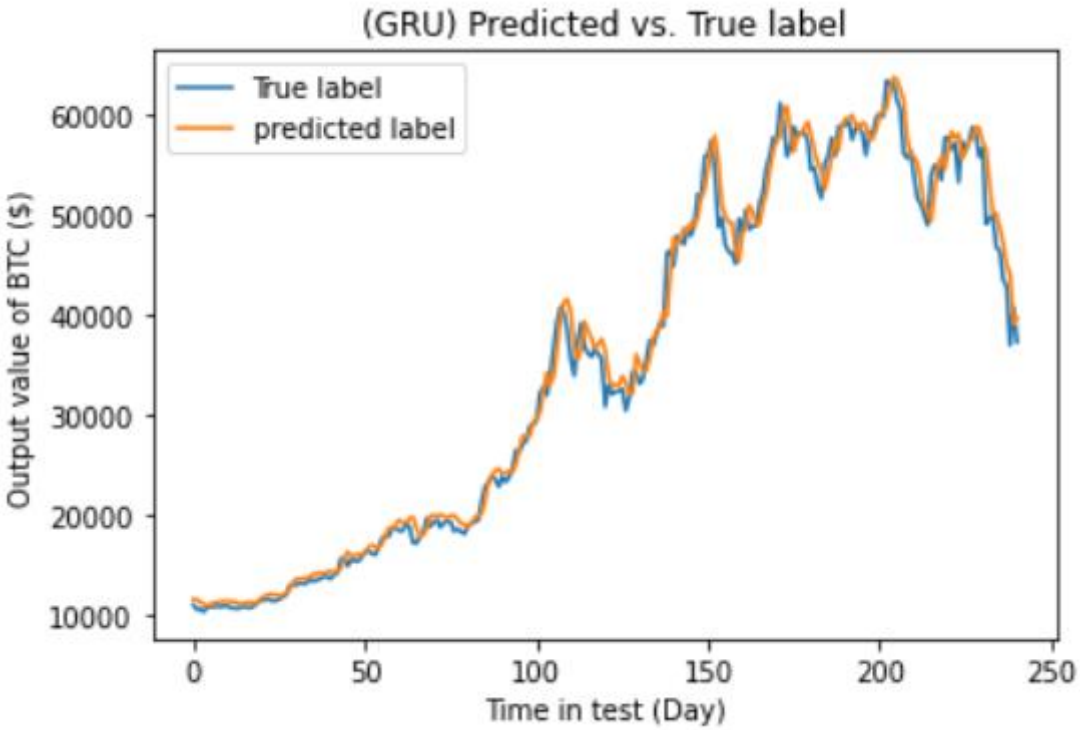
برای حالت MSE داریم:

برای Loss داریم:



شکل 1-13 – مقدار loss در GRU با تابع MSE

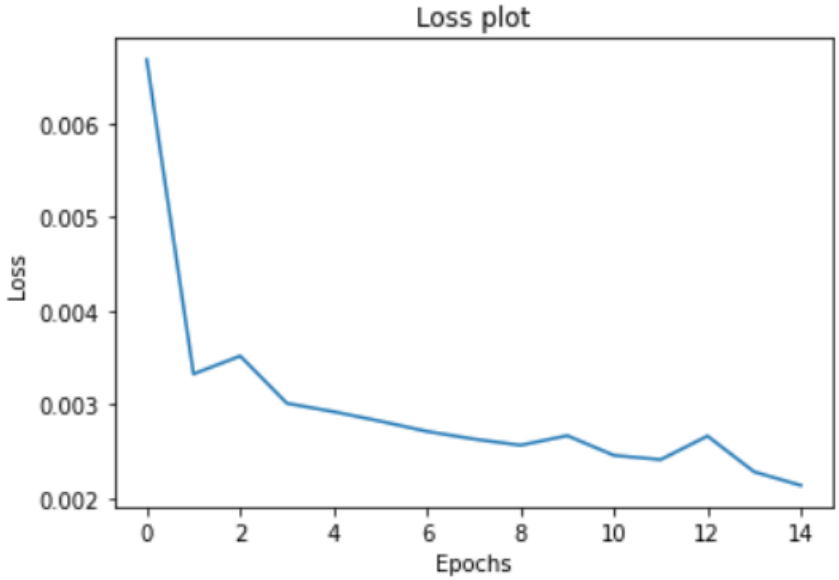
برای مقدار پیشبینی شده نیز داریم:



شکل 1-14 – مقدار پیش بینی شده و حقیقی در GRU با تابع MSE

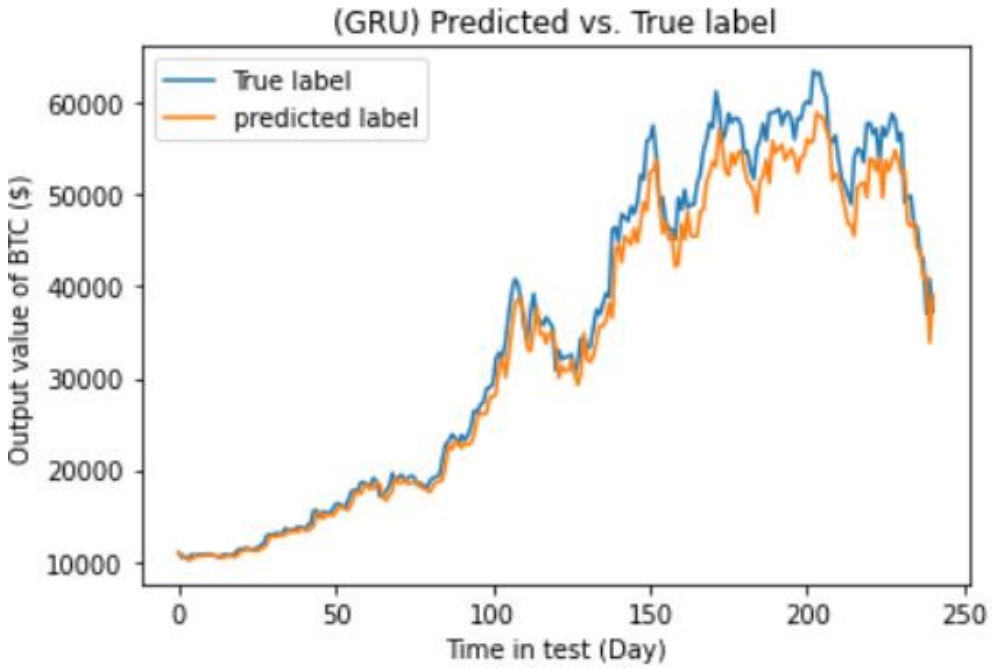
<u>برای حالت MAE</u>:

<u>برای Loss</u>:



شکل 1-15 – مقدار loss در GRU با تابع MAE

برای مقدار پیشبینی شده:



شکل 1-16 – مقدار حقیقی و پیش بینی شده در GRU با تابع MAE

مقایسه: همانطور که مشاهده می شود، اگر تابع loss را MAE قرار دهیم، عملکرد مدل ضعیف می شود و تابع MSE عملکرد بسیار بهتری دارد. علت این امر آن است تابع هزینه ی MSE شبکه را برای خطا های بزرگ هم penalize می کند در صورتی که شبکه ی MAE برای خطاهای بزرگ لزوما penalizeای در نظر نمی گیرد و در این سوال هم از آنجا که در range قیمت ها نوسان های زیادی مشاهده می شود، استفاده از MSE بهتر است.

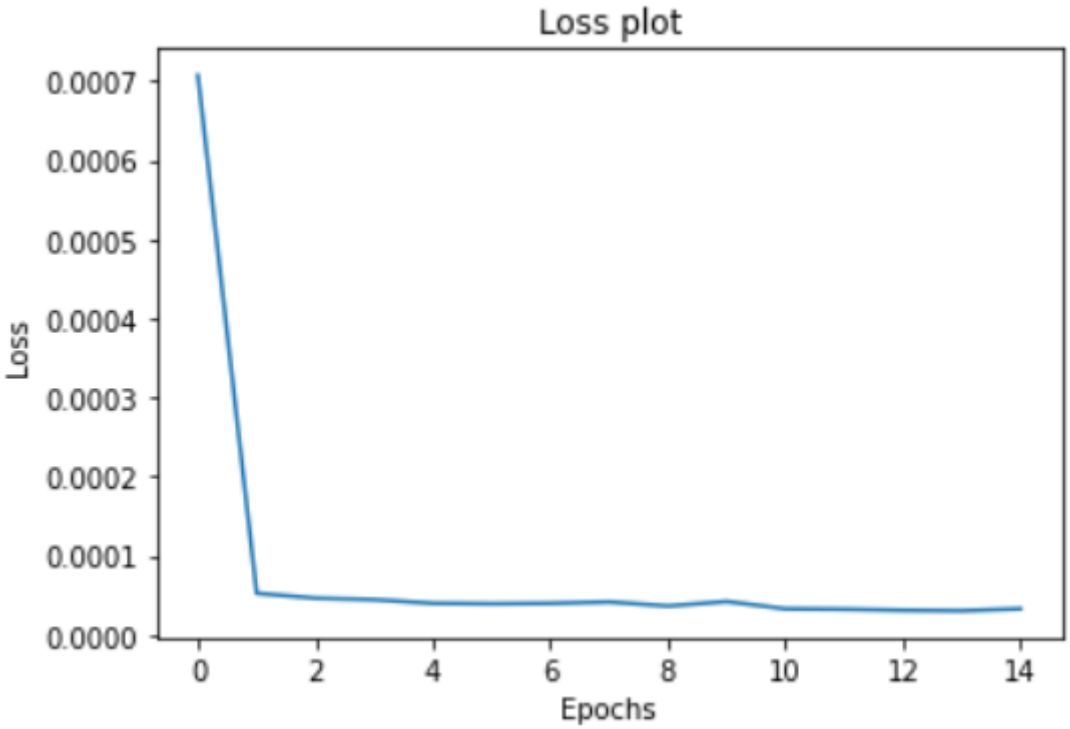
مجموع مربعات خطا، تابع پیش‌فرض مسائل Regression است. این خطا به‌صورت میانگین مجذور اختلاف بین خروجی‌های پیش‌بینی شده و خروجی Target محاسبه می‌گردد. MSE جدای از علامت Predict و Target، همواره دارای مقدار مثبت است و بهترین مقدار خطا برای آن صفر خواهد بود. استفاده از مجذور خطا نمایانگر آن است که اشتباهات بزرگ‌تر منجر به خطاهای بیش‌تر می‌شود.

$$MSE\ Loss: J(y) = \frac{1}{n}\sum_{i=1}^{n}(t_i - h_i)^2$$

برای اوپتیمایز های مختلف داریم:

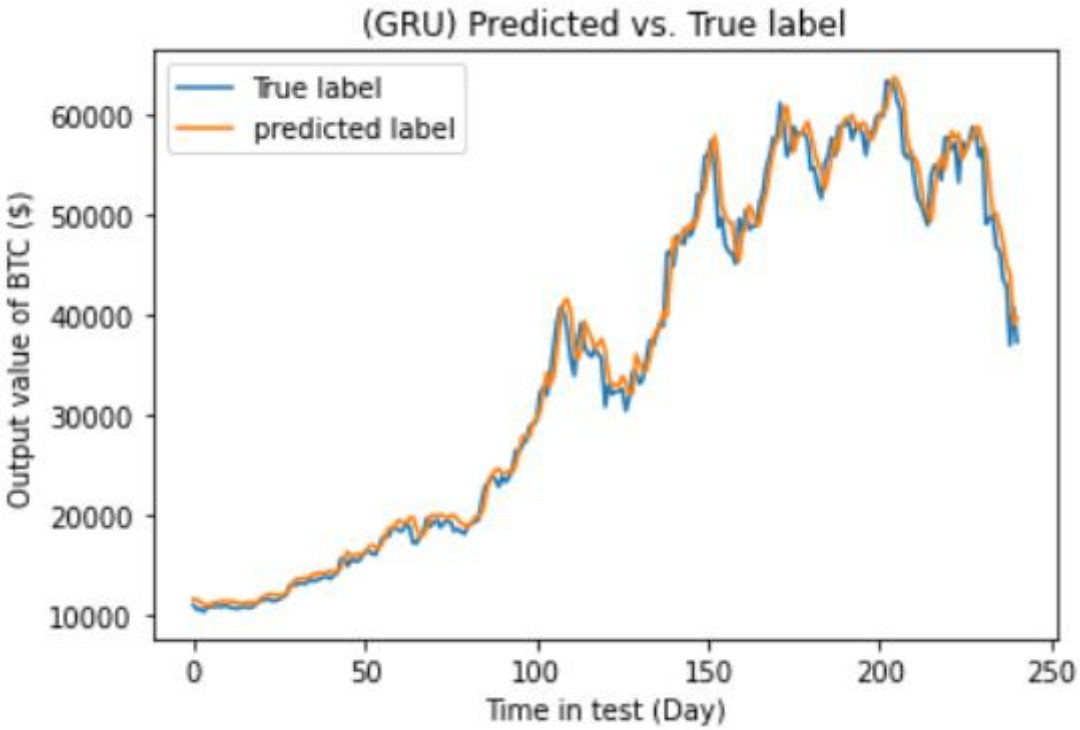
برای adam:

مقدار loss:



شکل 1-17 – مقدار loss در GRU با تابع اپتیمایزر adam

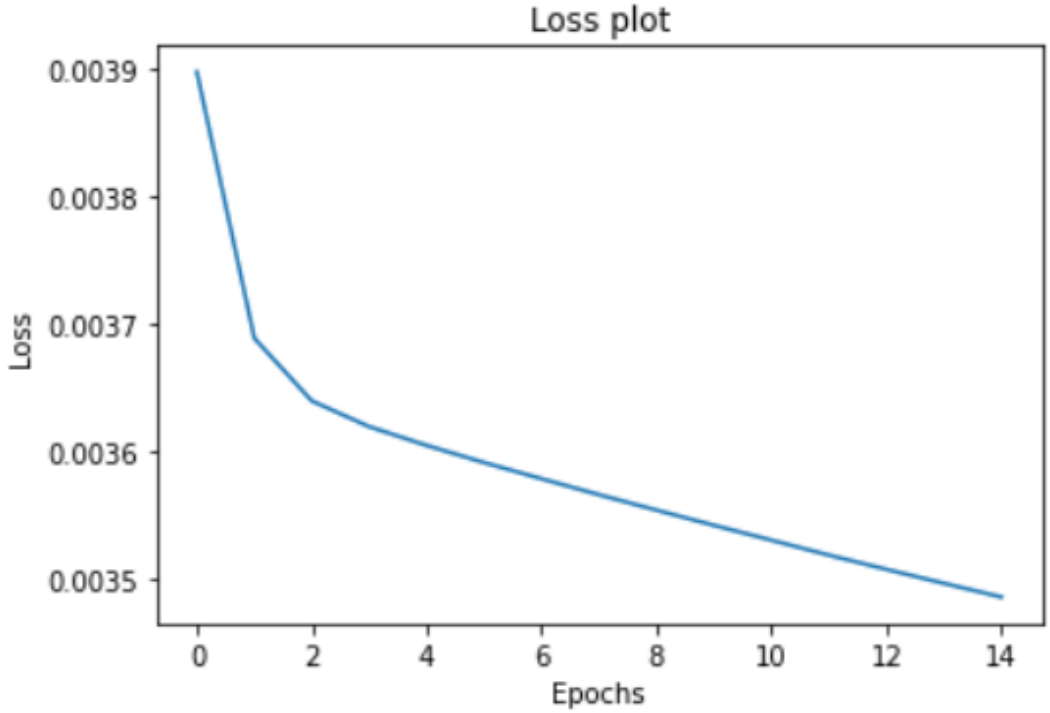
مقداری پیشبینی شده:



شکل 1-18 – مقدار پیش بینی شده و حقیقی در GRU با تابع اپتیمایزر adam

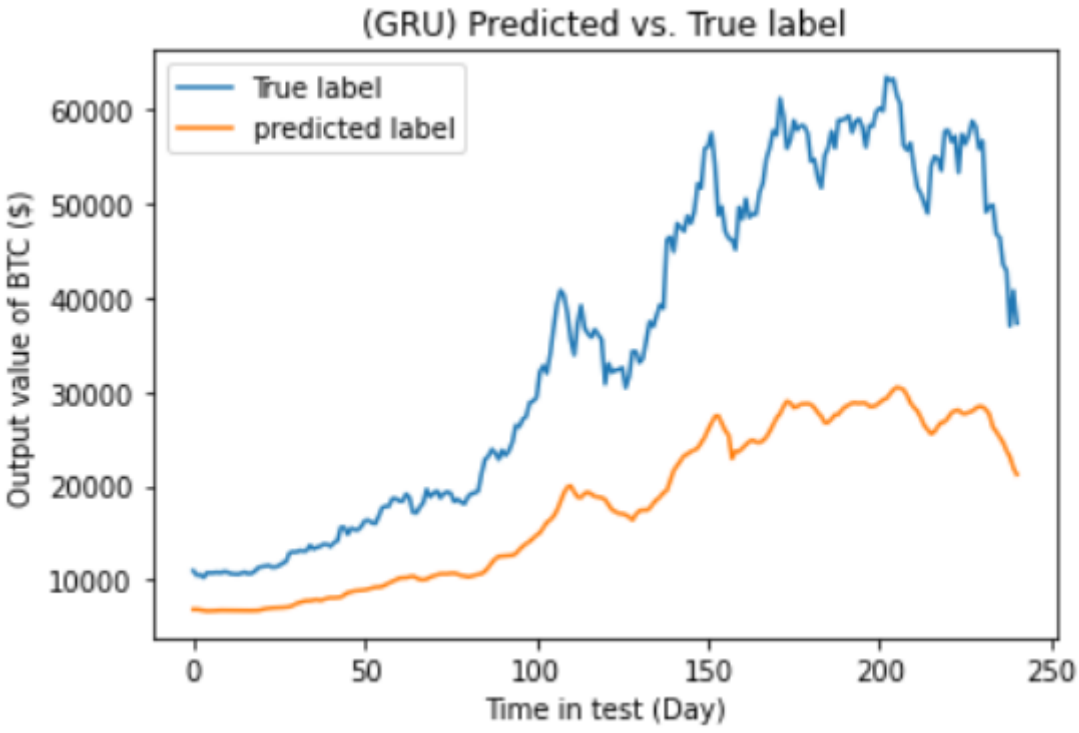
برای adagrad:

مقدار loss:



شکل 1-19 – مقدار loss در GRU با تابع اپتیمایزر adagrad

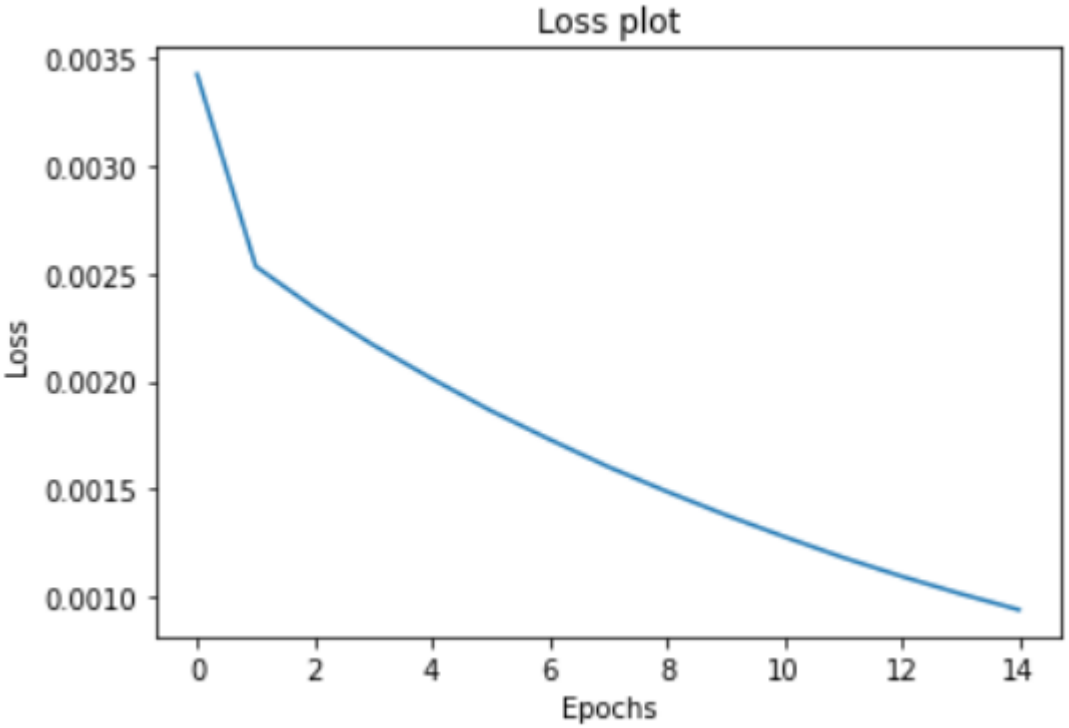
مقداری پیشبینی شده:



شکل 1-20 – مقدار پیش بینی شده و حقیقی در GRU با تابع اپتیمایزر adagrad

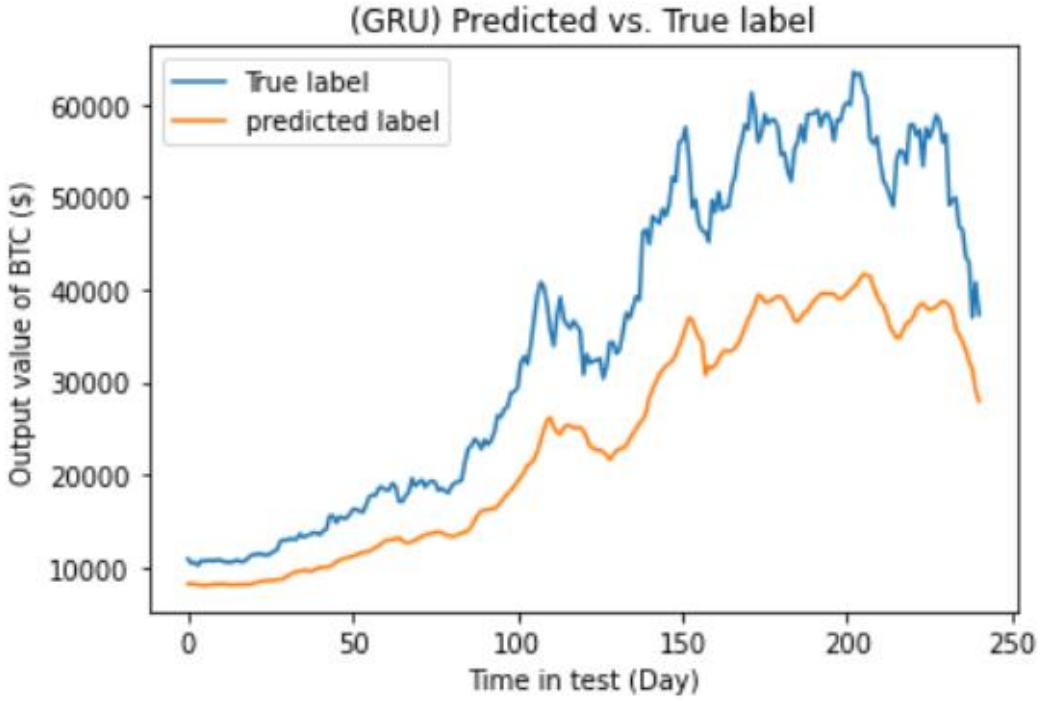
برای SGD:

مقدار loss :



شکل 1-21 – مقدار loss در GRU با تابع اپتیمایزر SGD

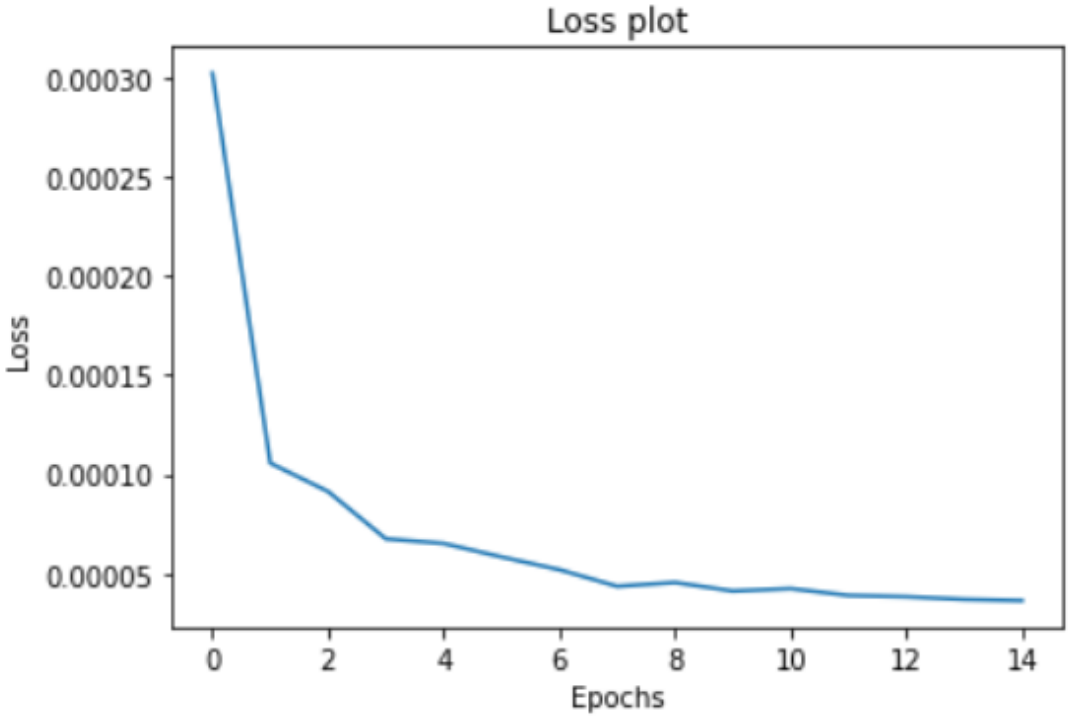
مقدار پیشبینی شده:



شکل 1-22 – مقدار پیش بینی شده و حقیقی در GRU با تابع اپتیمایزر SGD

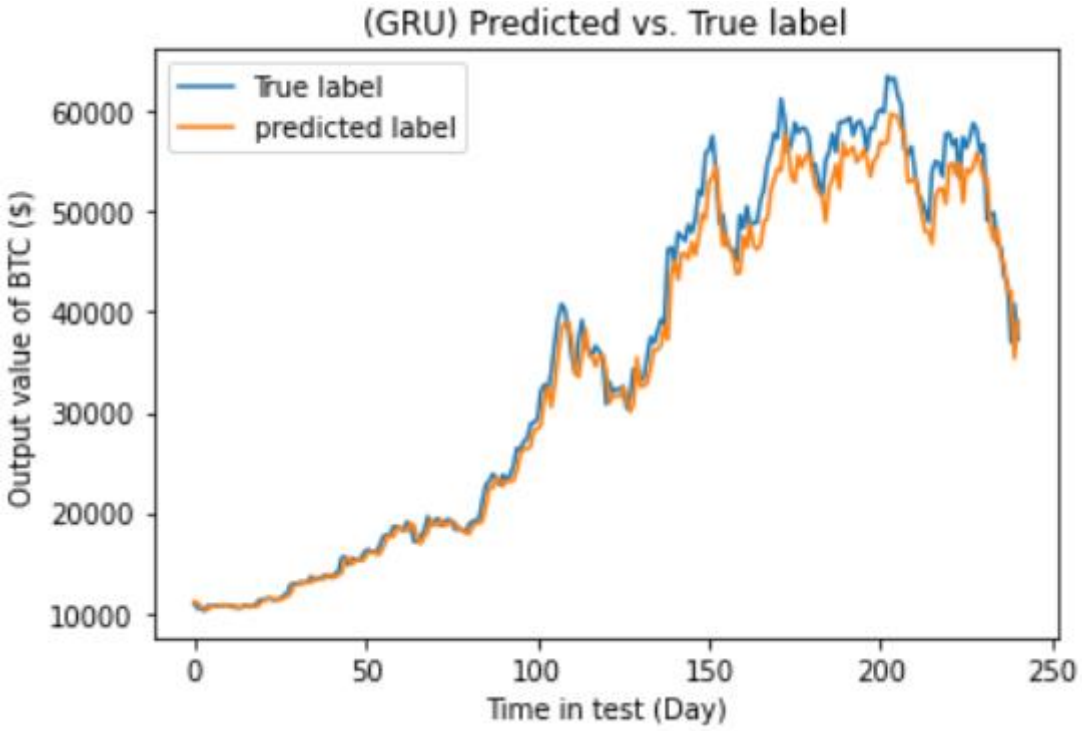
<u>برای RMSProp</u>:

مقدار loss:



شکل 1-23 – مقدار loss در GRU با تابع اپتیمایزر RMSProp

مقداری پیشبینی شده:



شکل 1-24 – مقدار پیش بینی شده و حقیقی در GRU با تابع اپتیمایزر RMSProp

- **Optimizers :**

RMSProp

بهینه‌ساز 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، به‌روزرسانی 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 قبلی‌ها به‌روزرسانی شود. سپس در معادله دوم، میزان step توسط میانگین نمایی محاسبه می‌گردد. به عنوان مثال اگر میانگین w1 بزرگتر از میانگین w2 باشد، step یادگیری برای w1 کوچکتر از w2 خواهد بود و منجر به یافتن مینیمم‌ها می‌شود. بنابراین هنکامی که تابع هزینه به نقاط مینیمم نزدیک می‌شود، RMSProp از قدم‌های کوچکتر استفاده می‌کند.

- **Adam**

در روش Adam(Adaptive Moment Estimation)، همانند RMSProp از نرخ یادگیری متفاوت برای آپدیت کردن هر پارامتر استفاده می‌شود. همچنین علاوه بر Learning Rate، از ترم Momentum 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 Adam از 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 β_1، β_2 و ε به ترتیب ۰,۹، ۰,۹۹۹ و 10^{-8} است. درواقع روش Adam منجر به همگرایی سریع‌تر نسبت به سایر الگوریتم‌های بهینه‌سازی می‌شود. همچنین با مشکلاتی همچون همگرایی نرخ یادگیری به صفر و کاهش سرعت همگرایی تابع خطا، مواجه نمی‌شود.

به‌طورکلی الگوریتم Adam بهتر از الگوریتم‌های Adaptive دیگر همانند RMSProp عمل می‌کند. حال اگر داده‌های ورودی به‌اصطلاح sparse باشند، روش‌هایی مانند SGD و Momentum ضعیف

عمل می‌کنند و باید از روش‌های Adaptive استفاده کرد. برای دستیابی به همگرایی سریع‌تر در مدل‌های عمیق و پیچیده، الگوریتم Adam بهتر از سایرین عمل می‌کند.

- **SGD**

برای دیتاست های بزرگ استفاده از Gradient Descent به خاطر حجم بالای محاسبات مناسب نبوده و در این مواقع از SGD استفاده می کردیم .در این روش در هر iteration یکی از داده ها انتخاب می شود و عملیات update کردن وزن ها روی آن صورت می گیرد .این روش به طور خاص برای دیتا هایی مناسب است که دچار redundancy هستند.

- **Adagrad**

در این تابع هزینه همانند قسمت قبلی هدف همان تغییر دادن سرعت update ها در راستای ویژگی های مختلف هستیم.

مقایسه: بهترین الگوریتم برای بهینه سازی باتوجه به مقادیر loss و دقت و سرعت بالاتر ،الگوریتم adam می باشد.

جدول 1-2- مقایسه عملکرد الگوریتم های بهینه سازی

الگوریتم	Time (micro seconds)	mae	loss
Adam	**6.91**	0.0270	0.0015
Ada grad	**5.96**	0.3773	0.1887
SGD	**7.63**	0.4216	0.2489
RMSProp	**9.78**	0.5539	0.3860

4)

حال به تاثیر dropout می پردازیم:

تاثیر dropout:

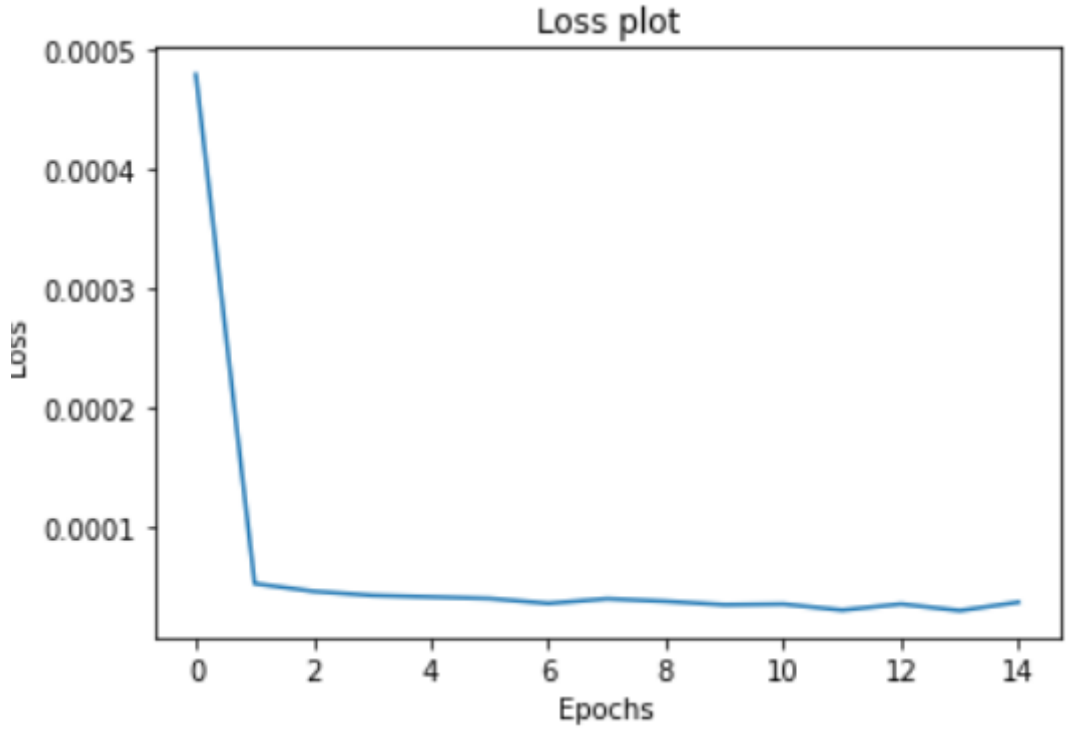
بخش drop out را اضافه می کنیم(0.2) و نتیجه را بررسی می کنیم.

جدول 1-3- مقایسه تاثیر dropout روی عملکر هر شبکه

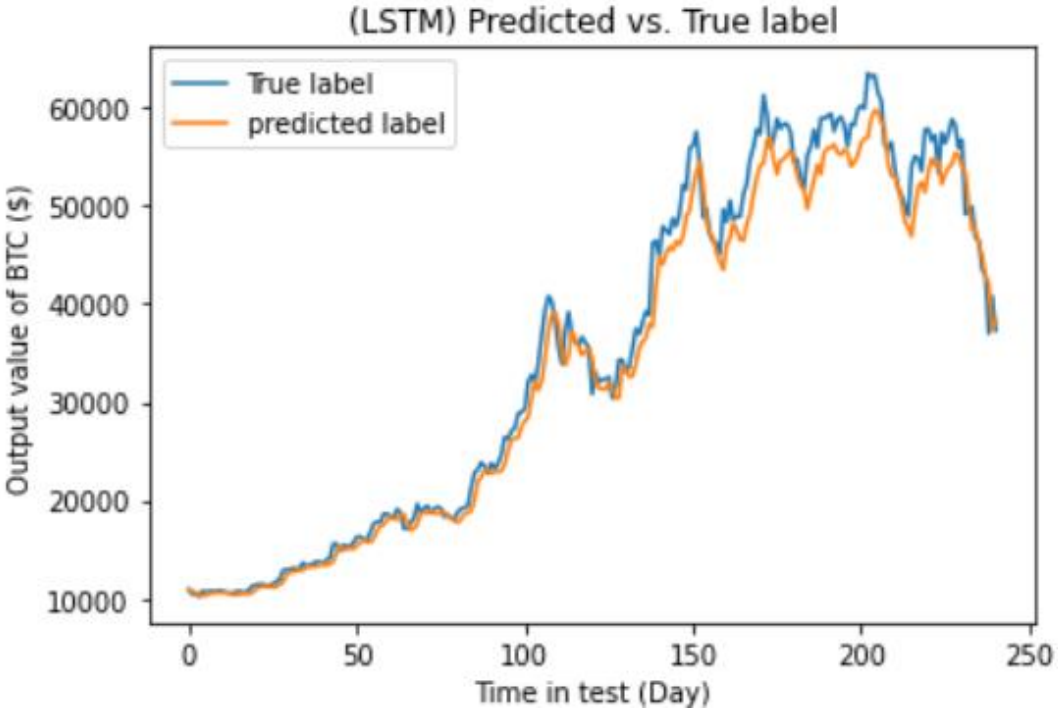
Network	(with dropout)			(no dropout)		
	time	loss	MAE	time	loss	MAE
LSTM	10.3	0.0152	0.0920	6.7	0.0093	0.0702
GRU	8.58	0.0018	0.0302	6.77	0.0020	0.0318
RNN	6.2	0.0017	0.0289	5.96	0.0029	0.0323

برای نمودارها نیز داریم:

برای GRU:

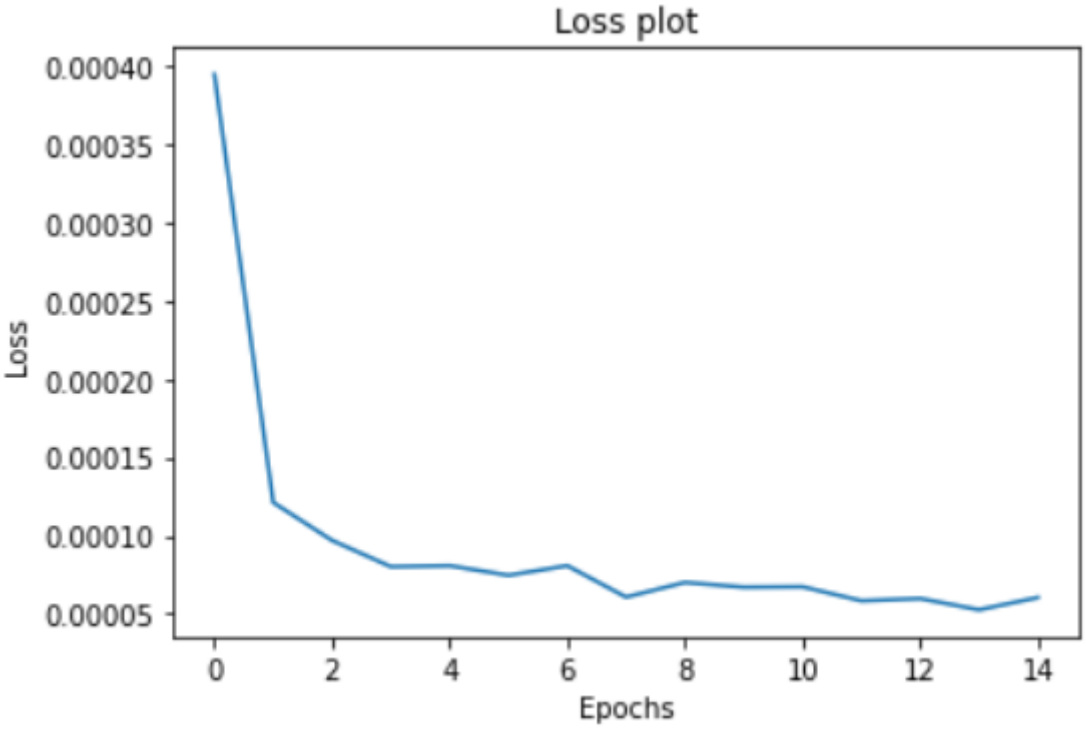


شکل 1-25 – مقدار loss در GRU با dropout = 0.2

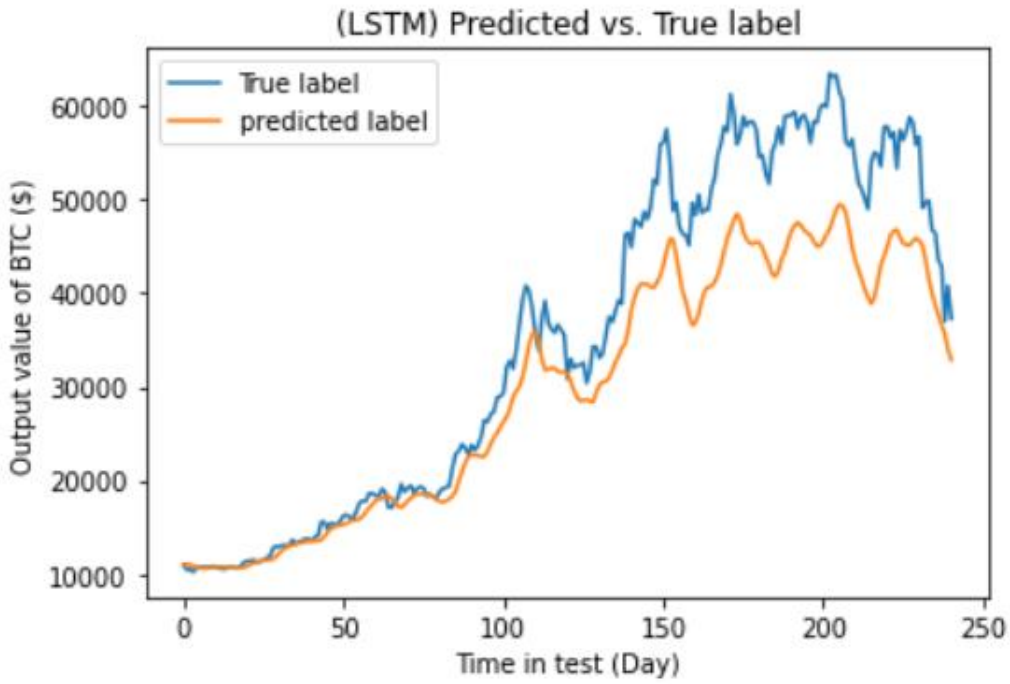


شکل 1-26 – مقدار پیش بینی شده و حقیقی در GRU با dropout = 0.2

برای LSTM داریم:

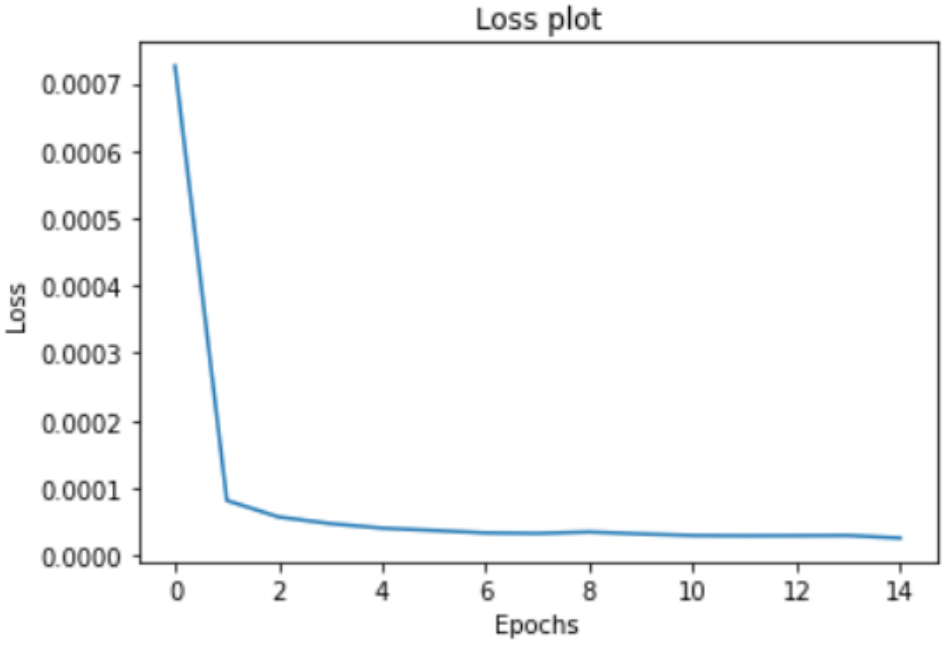


شکل 1-27 – مقدار loss در LSTM با dropout = 0.2

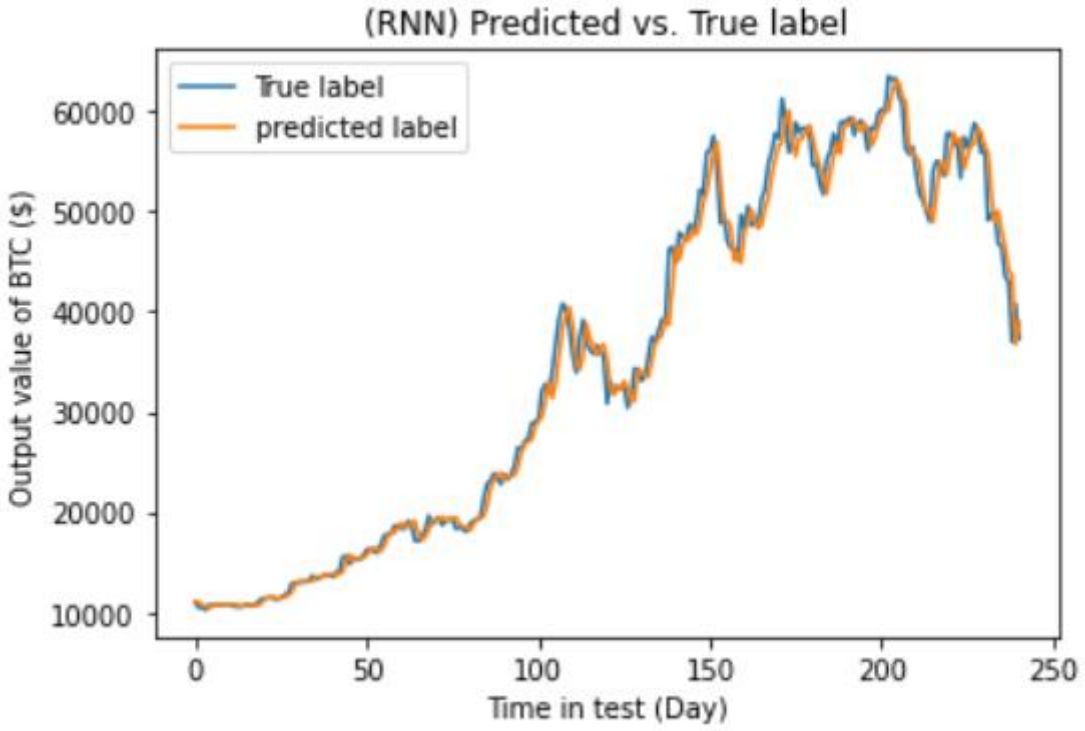


شکل 1-28 – مقدار پیش بینی شده و حقیقی در LSTM با dropout = 0.2

برای RNN نیز داریم:



شکل 1-29 – مقدار loss در RNN با dropout = 0.2



شکل 1-30 – مقدار پیش بینی شده و حقیقی در RNN با dropout = 0.2

تحلیل :

از مقایسه مقادیر جدول بالا می توانیم نتیجه گیری کنیم که افزودن dropout به شبکه، در بهبود آن تاثیر گذار است .ولی در شبکه ی LSTM تاثیر منفی و در بعضی پارامتر های GRU نیز تاثیر منفی کمی گذاشته علت این امر آن است که شبکه LSTM در این سوال کم عمق هست و بنابراین تعداد نورون های کمی دارند و با افزودن dropout آموزش 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 و در نتیجه تعداد نورون های بیشتری
داشتیم و افزودن dropout به فراگیرتر شدن مدل ما کمک می کرد .همچنین میدانیم که شبکه های LSTM و GRU نسبت به حالت اولیه اشان نمی توانند بهبود زیادی داشته باشند و توان آن ها در همین حدود است.

(5

در این قسمت به طراحی یک MLP با تابع optimizer آدام و تابع loss ، MSE میپردازیم :

```
CPU times: user 3 µs, sys: 0 ns, total: 3 µs
Wall time: 7.63 µs
Model: "sequential_34"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_38 (Dense)             (None, 256)               30976
_________________________________________________________________
activation_4 (Activation)    (None, 256)               0
_________________________________________________________________
dense_39 (Dense)             (None, 100)               25700
_________________________________________________________________
activation_5 (Activation)    (None, 100)               0
_________________________________________________________________
dense_40 (Dense)             (None, 100)               10100
_________________________________________________________________
activation_6 (Activation)    (None, 100)               0
_________________________________________________________________
dense_41 (Dense)             (None, 32)                3232
_________________________________________________________________
activation_7 (Activation)    (None, 32)                0
_________________________________________________________________
dense_42 (Dense)             (None, 1)                 33
=================================================================
Total params: 70,041
Trainable params: 70,041
Non-trainable params: 0
```

زمان : `Wall time: 8.34 µs`

`loss: 0.0013`

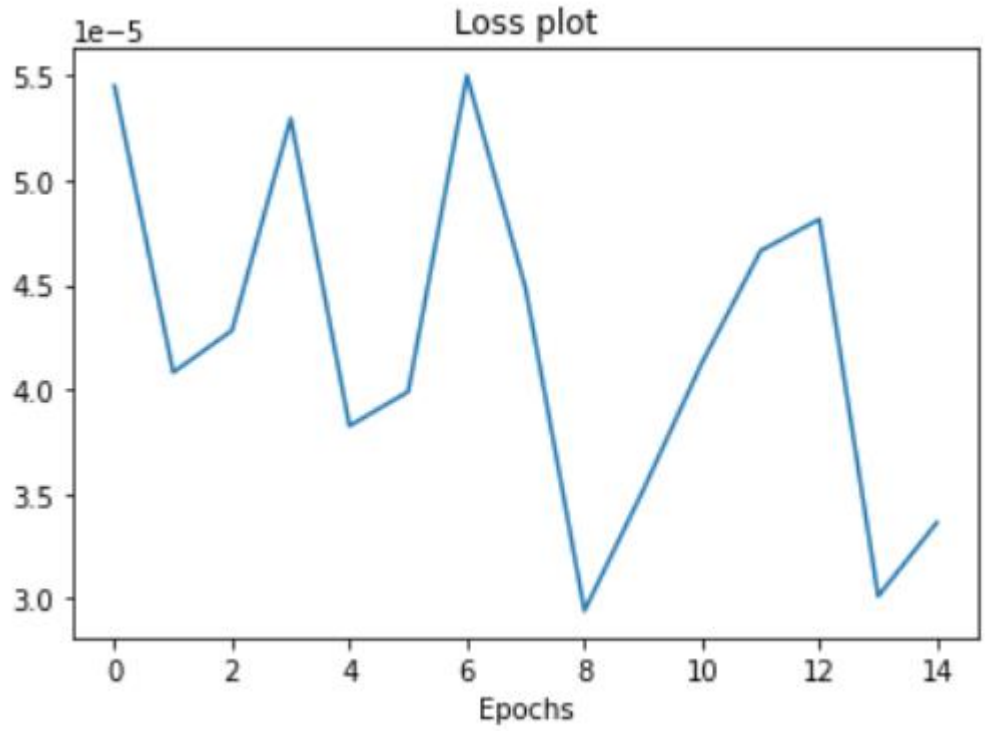
جدول 1-4- مقایسه عملکرد شبکه ها و مقایسه آن ها با شبکه MLP

شبکه مورد نظر	Time (میکرو ثانیه)	Test loss	خطای mae بعد از 15 ایپاک
LSTM	11.4	0.0093	0.0702
GRU	6.77	0.0020	0.0318
RNN	5.96	0.0029	0.0323
MLP	6.2	0.0031	0.082

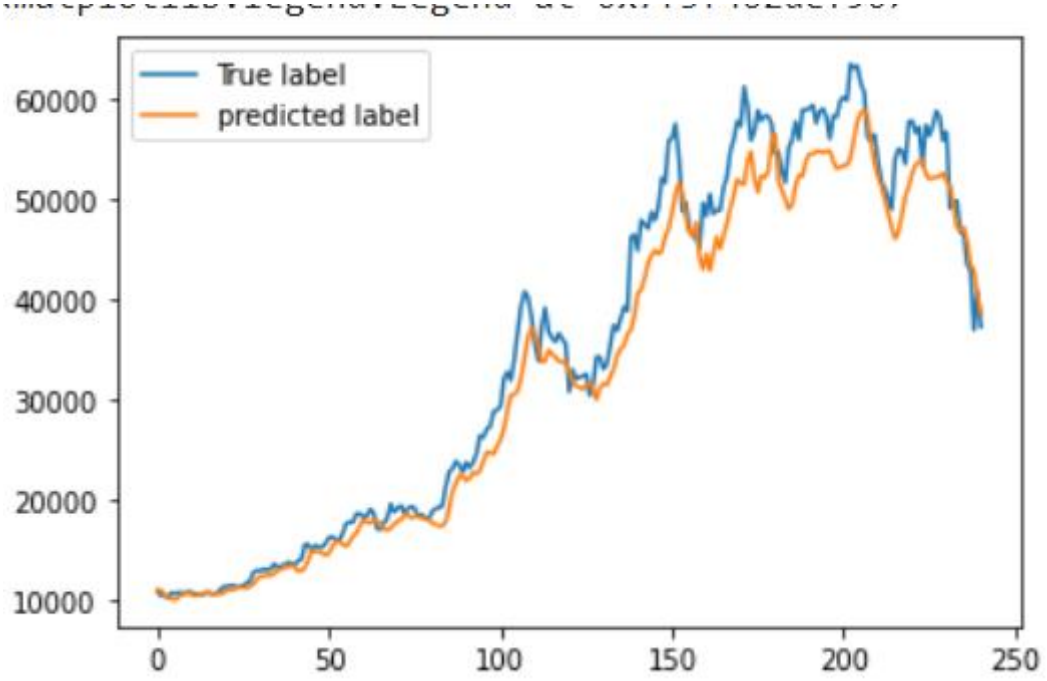
از مقایسه ی نتایج جدول بالا می توانیم نتیجه بگیریم که شبکه ی MLP به علت سادگی بیشتر

از بعضی شبکه ها سریعتر عمل می کند .دقت MLP از دقت شبکه های LSTM و GRU و RNN کمتر است .علت بهتر بودن عملکرد شبکه های GRU LSTMو RNN در این است که در آنها، سیر زمانی ورودی ها در نظر گرفته می شود .یکی ازعلت های به وجود آمدن شبکه های sequential هم همین است لذا در این جا که دیتا های ما به صورت سری زمانی هستند، این شبکه ها عملکرد بهتری دارند و هرچقدر طول بازه ی داده های ما در هر دوره بیشتر شود، بیشتر می توانیم عملکرد بهتر این شبکه ها را مشاهده کنیم .البته ذکر این نکته ضروری است که اگر دیتاهای ما به صورت داده های متنی بود و این توالی داده ها بیشتر اهمیت داشت، تمایز میان عملکرد
این شبکه ها را بیشتر می دیدیم.

.یکی از راهکار هایی که می توانیم برای بهبود عملکرد شبکه ی MLP ارائه دهیم، افزودن dropoutاست .با این کار این شبکه دارای generalization بیشتری می شود.



شکل 1-31 – مقدار loss در MLP بدون dropout



شکل 1-32 – مقدار پیش بینی شده و حقیقی در MLP بدون dropout

6)

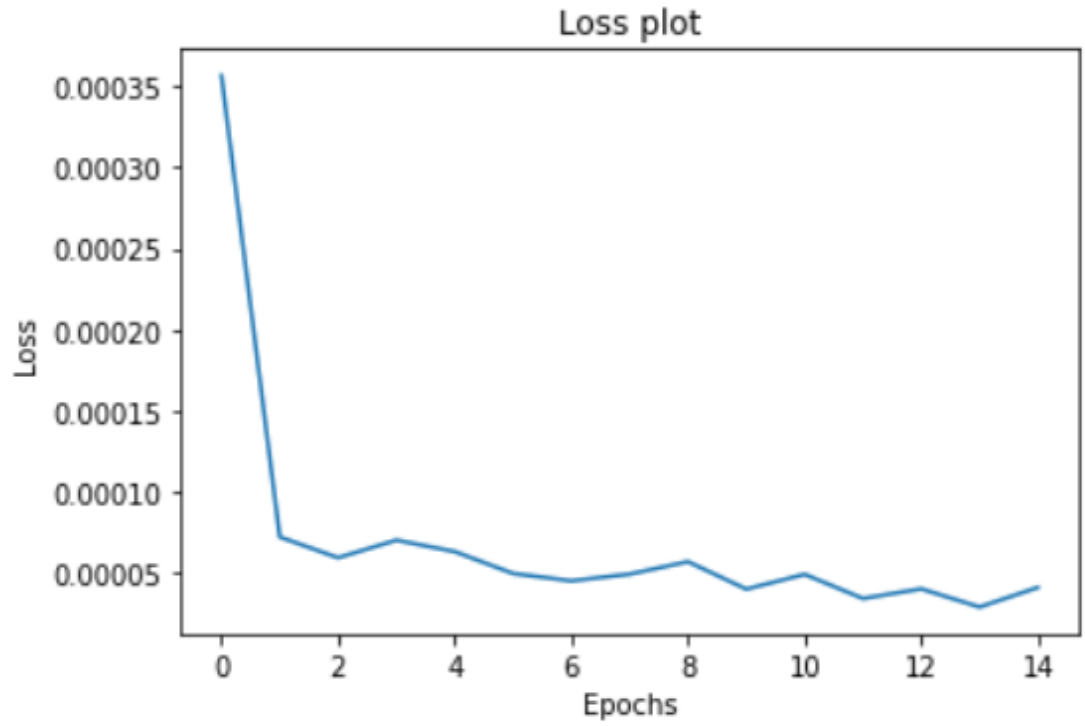
وجود تمامی ویژگی ها در آموزش شبکه الزامی نیست بلکه می توانیم یکی ویژگی هایی که باهم correlation بالایی دارند را حذف کنیم .برای یافتن این ویژگی های میتوان از scatter plot های متفاوت بین هر دو ویژگی و برای کاهش ابعاد متناظر میتوان از LDA و PCA استفاده کرد.

۷)

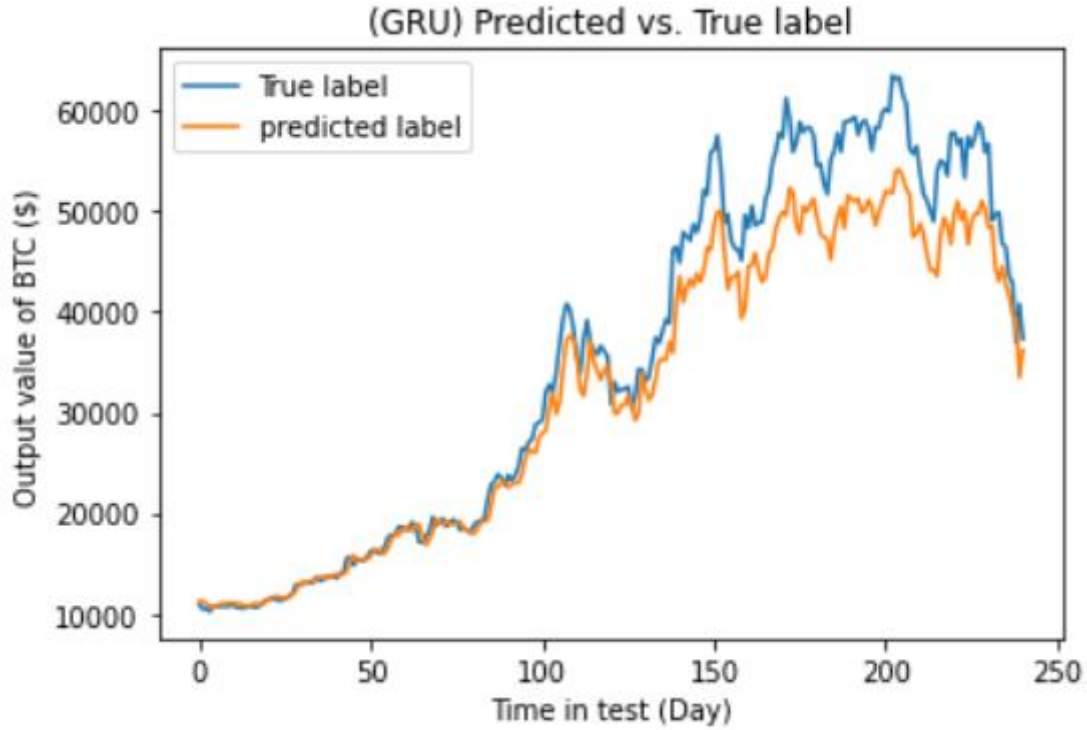
در این بخش به شبکه برنده یعنی GRU یک لایه دیگر با مشخصات زیر اضافه میکنیم:

```
model = Sequential()
model.add(GRU(50, input_shape=(x_train.shape[1],5 ), return_sequences=True))
model.add(GRU(100, return_sequences=True  , recurrent_dropout=0.0))
model.add(GRU(100, return_sequences=False  , recurrent_dropout=0.0))
model.add(Dense(2, activation='relu'))
model.compile(loss='mse', optimizer='adam', metrics=['mae'])
model.summary()
```

بدون dropout :

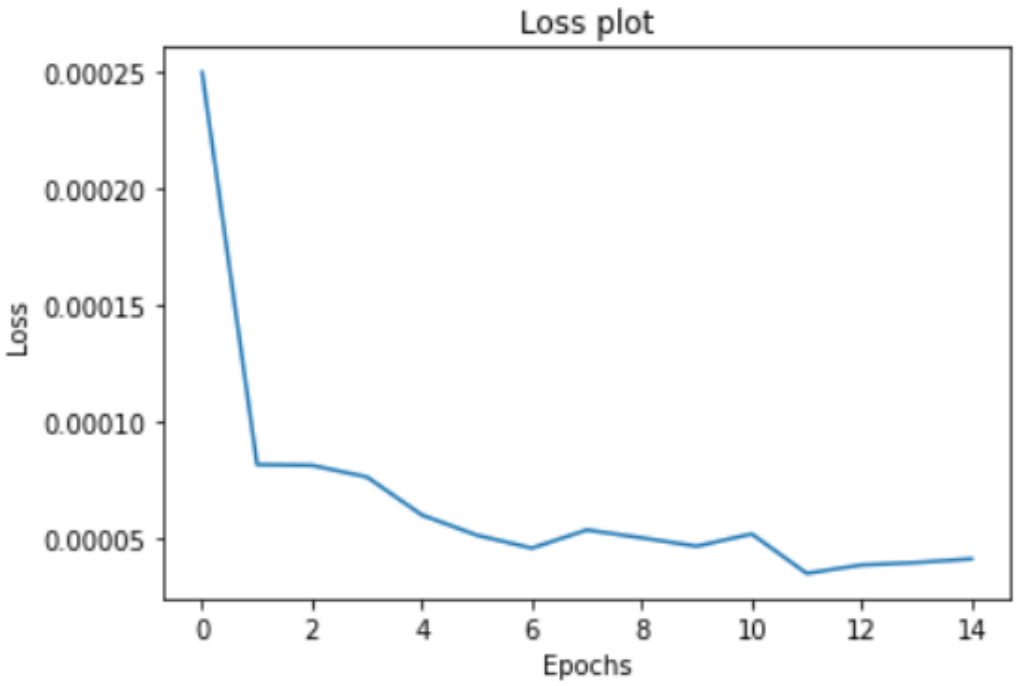


شکل 1-33 – مقدار loss در GRU با دو لایه و بدون dropout

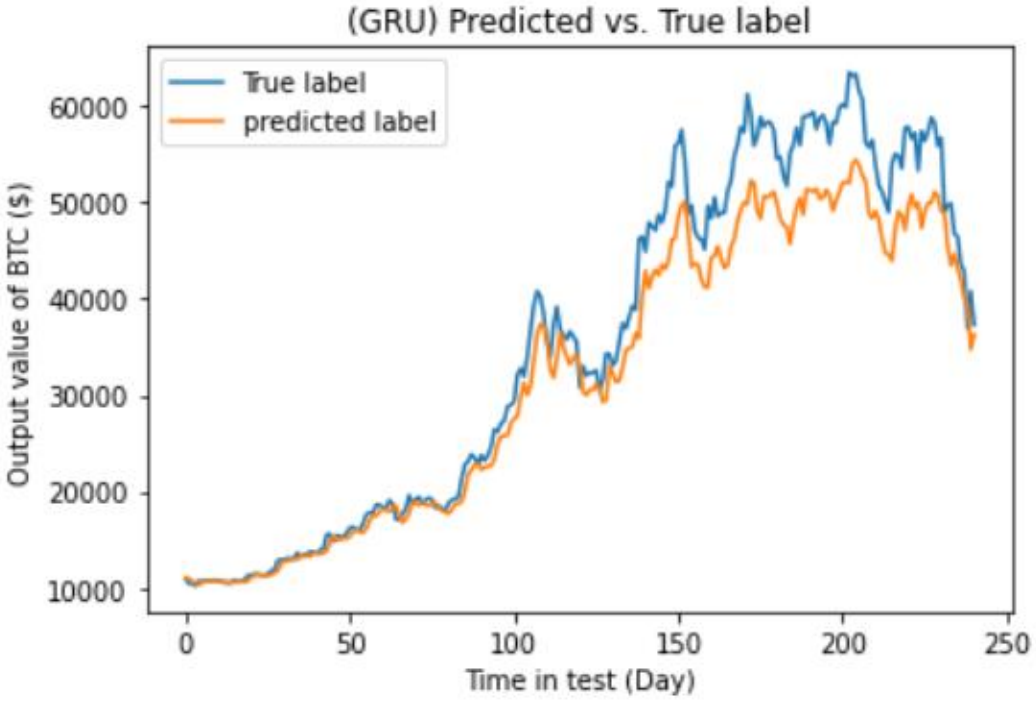


شکل 1-34 – مقدارپیش بینی شده و حقیقی در GRU با دو لایه و بدون dropout

با dropout :



شکل 1-35 – مقدار loss در GRU با دو لایه و با dropout = 0.2



شکل 1-36 – مقدارحقیقی و پیش بینی شده در GRU با دو لایه و با dropout = 0.2

جدول 1-5- مقایسه عملکرد شبکه GRU با تعداد لایه بیشتر و اثر dropout

تعداد لایه های بازگشتی	Time	Loss	MAE
1	6.77	0.0020	0.0318
3(بدون dropout)	7.39	0.0052	0.0516
3(با dropout)	11	0.0067	0.0602

همان طور که از جدول بالا مشخص است، شبکه ی GRU با سه لایه ی مخفی بازگشتی عملکرد ضعیف تری دارد .علت این امر هم این است که با افزودن لایه ی جدید به شبکه، می توانیم در عمق بیشتری عملیات یادگیری را انجام دهیم و روابط بین داده ها را با جزئیات بیشتری پیدا کنیم . ولی در این مسئله که طول sequence ما کم است افزودن این پیچیدگی ها اثر معکوس بر روی عملکرد شبکه می گذارد و سبب بهبود عملکرد آن نمی شود .همچنین در این دیتاست وابستگی داده ای بین دیتاهای موجود در یک sequence کم است لذا افزودن لایه ی جدید پیچیدگی مدل را بیشتر می کند و نتیجه افزایش پیچیدگی، generalization مدل را کم می کند.

(9

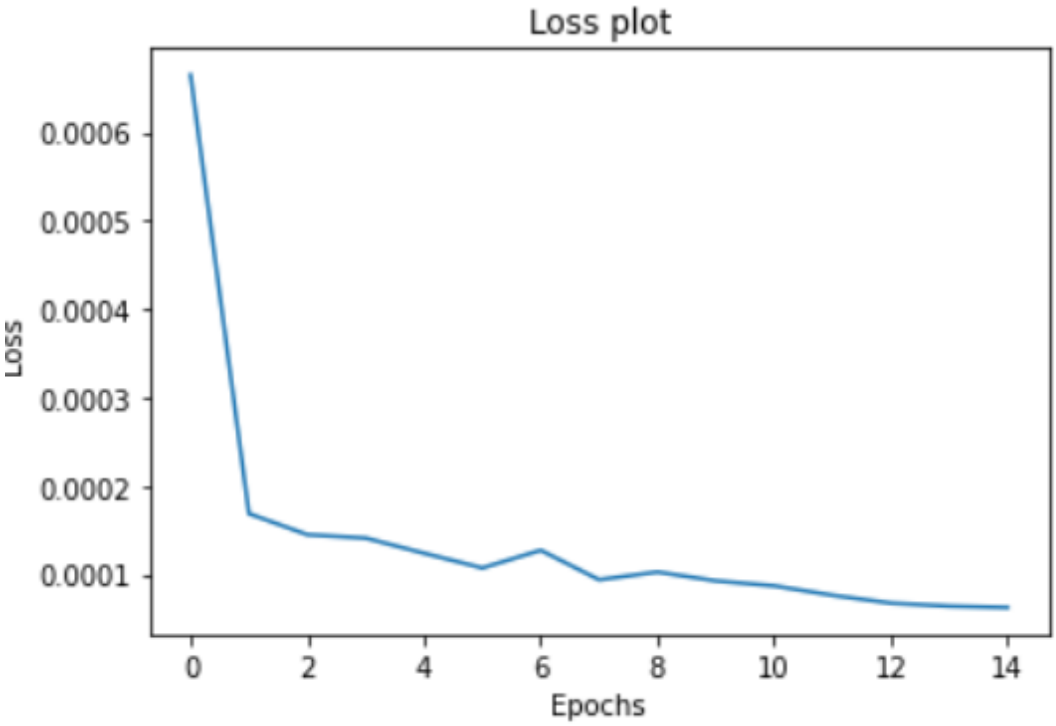
در این بخش به طراحی یک CNN-LSTM با مشخصات زیر میپردازیم:

```
model = Sequential()
model.add(Conv1D(input_shape=x_train.shape[1:] , filters=256, kernel_size=3, strides=1, padding='valid', activation='relu'))
model.add(MaxPooling1D(pool_size=2))
model.add(LSTM(50, batch_input_shape=(None, 24, 5), return_sequences=True, recurrent_dropout=0.2))
model.add(LSTM(50, return_sequences=False, recurrent_dropout=0.2))
model.add(Dense(2, activation='relu'))
model.compile(loss='mse', optimizer='adam', metrics=['mae'])
model.build((None,25,4))
model.summary()
```

```
Model: "sequential_41"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d (Conv1D)              (None, 22, 256)           4096
_________________________________________________________________
max_pooling1d (MaxPooling1D) (None, 11, 256)           0
_________________________________________________________________
lstm_8 (LSTM)                (None, 11, 50)            61400
_________________________________________________________________
lstm_9 (LSTM)                (None, 50)                20200
_________________________________________________________________
dense_54 (Dense)             (None, 2)                 102
=================================================================
Total params: 85,798
Trainable params: 85,798
Non-trainable params: 0
_________________________________________________________________
```

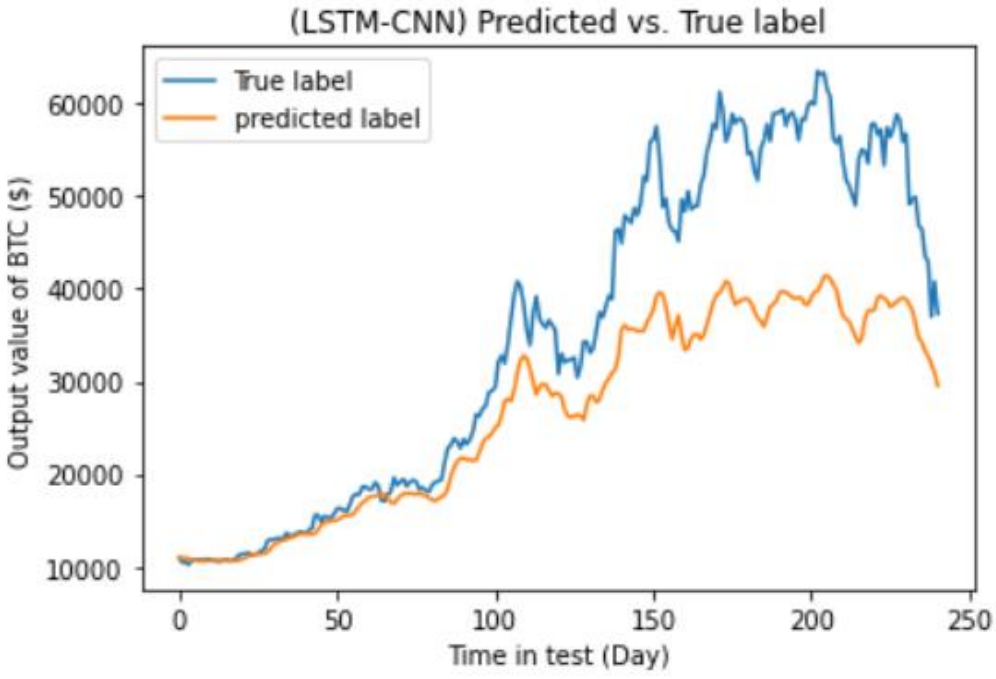
این شبکه دارای یک لایه Conv1D و یک لایه maxpooling است. .کاربرد این شبکه در این جا برای افزایش یادگیری است.افزودن یک لایه ی CNN در ابتدای شبکه ی LSTM ، به شبکه عمق بیشتری می دهد که شبکه قابلیت دریافت اطلاعات بیشتری از داده های ورودی پیدا می کند .لایه ی CNNارتباط بین 3 روز متوالی را بدست می آورد و داخل یک آرایه ذخیره می کند سپس این آرایه به شبکه ی LSTM داده می شود و اطلاعات موجود افزایش می یابد.

مقدار تابع loss :



شکل 1-37 – مقدار loss در CNN-LSTM

مقادیر حقیقی و پیش بینی شده :



شکل 1-38 – مقادیر پیش بینی شده و حقیقی در CNN-LSTM

جدول 1-6- مقایسه عملکرد همه شبکه ها و CNN-LSTM

شبکه مورد نظر	Time (میکرو ثانیه)	Test loss	خطای MAE بعد از 15 ایپاک
LSTM	11.4	0.0093	0.0702
GRU	6.77	0.0020	0.0318
RNN	5.96	0.0029	0.0323
MLP	6.2	0.0031	0.082
CNN-LSTM	6.91	0.0346	0.1405

مشاهده می شود که عملکرد شبکه افت کرده است زیرا شبکه ی CNN-LSTM پیچیدگی بیشتری نسبت به دیگر شبکه ها دارد و این پیچیدگی ، تعداد داده های بالاتری را برای training می طلبد درحالی که دیتا ست ما دست نخورده باقی مانده است.

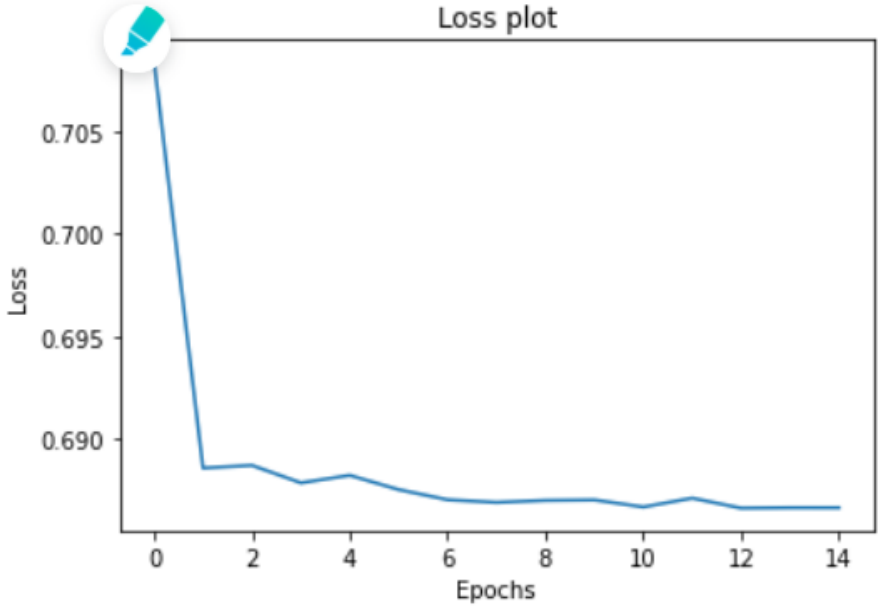
در راستای بهبود عملکرد شبکه می توانیم به آن دیتای بیشتری بدهیم(مثلا با روش هایی مثل data augmentation) یا در این مورد طول بازه های sequence را زیاد کنیم.

(۱۰

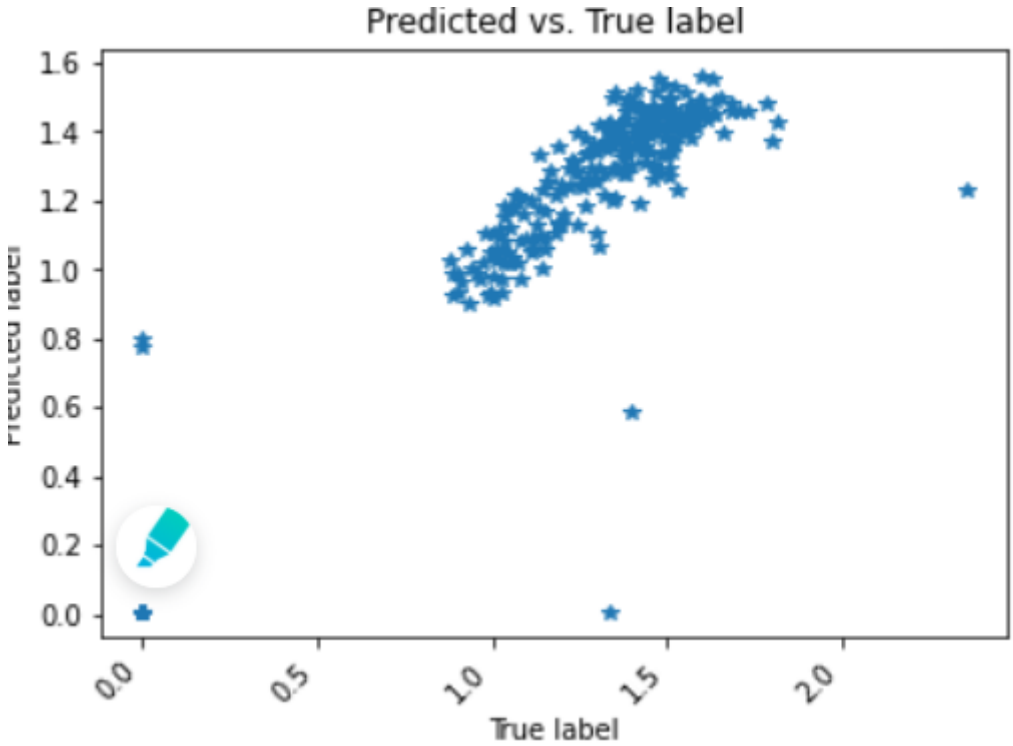
نکته: از MinMaxScaler در تمام مراحل قبل استفاده کرده بودیم برای همین دیگر اینجا تکرار نکردیم.

ابتدا از power transformer استفاده میکنیم:

این تبدیل داده ها را از حالت شبه گوسی به گوسی تبدیل می کند و یکنواختی بیشتری را رقم میزند.

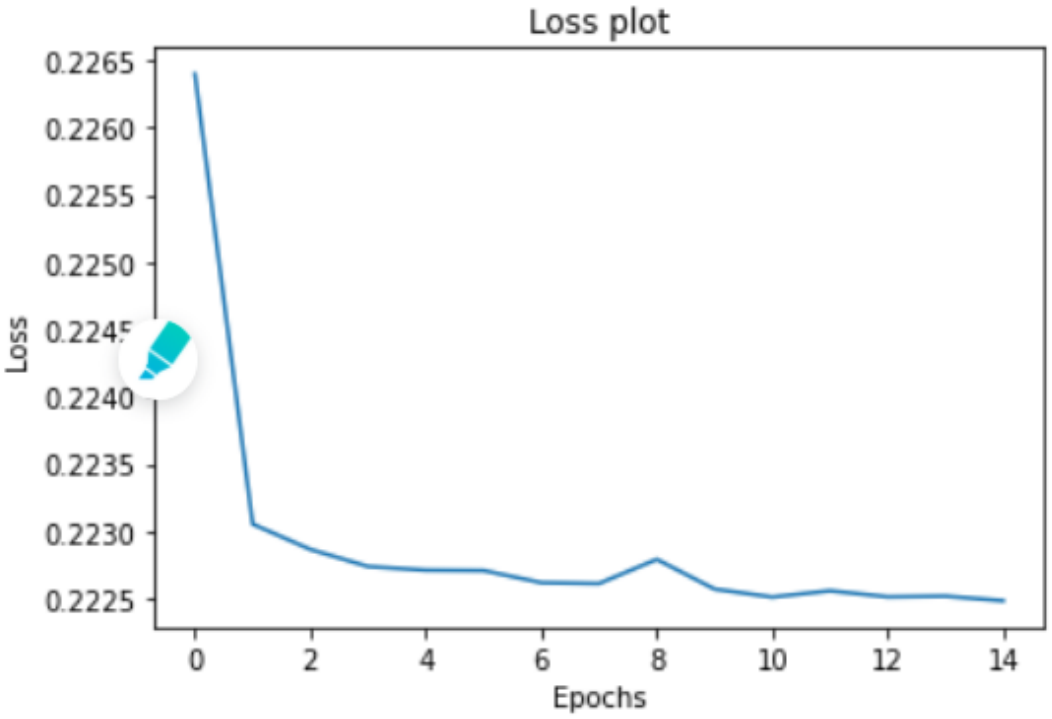


شکل 1-39 – مقدار loss در GRU با Power Transformer

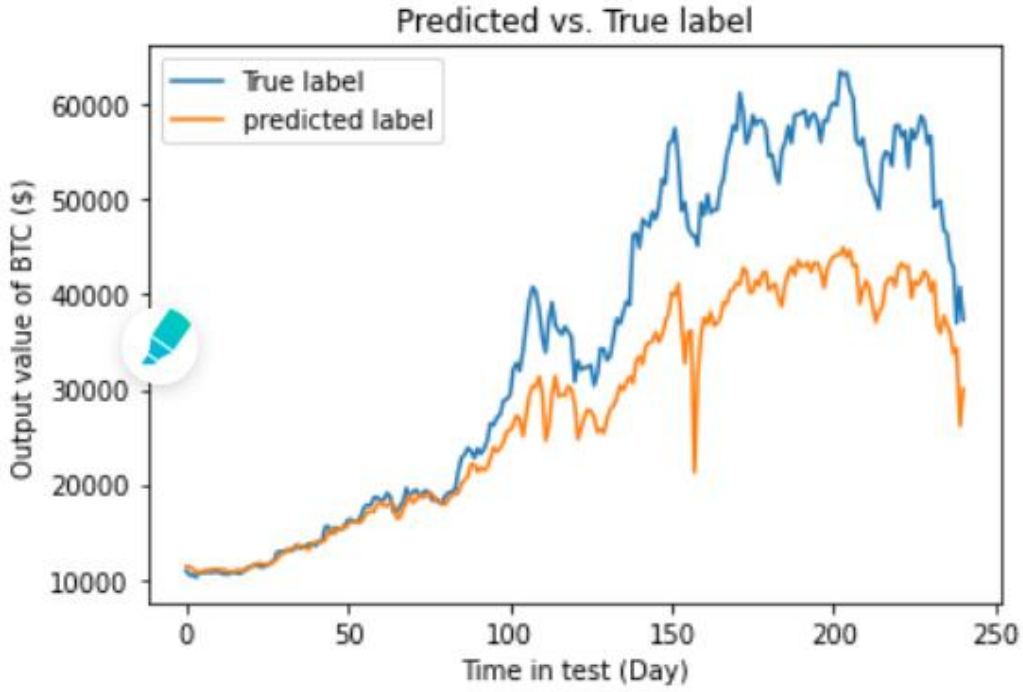


شکل 1-40 – مقدار حقیقی و پیش بینی شده در GRU با Power Transformer

تبدیل standard scaler :



شکل 1-41 – مقدار حقیقی و پیش بینی شده در GRU با Standard Scaler



شکل 1-42 – مقدار حقیقی و پیش بینی شده در GRU با Standard Scaler

جدول 1-7- مقایسه عملکرد تبدیل های مختلف روی شبکه GRU (برنده)

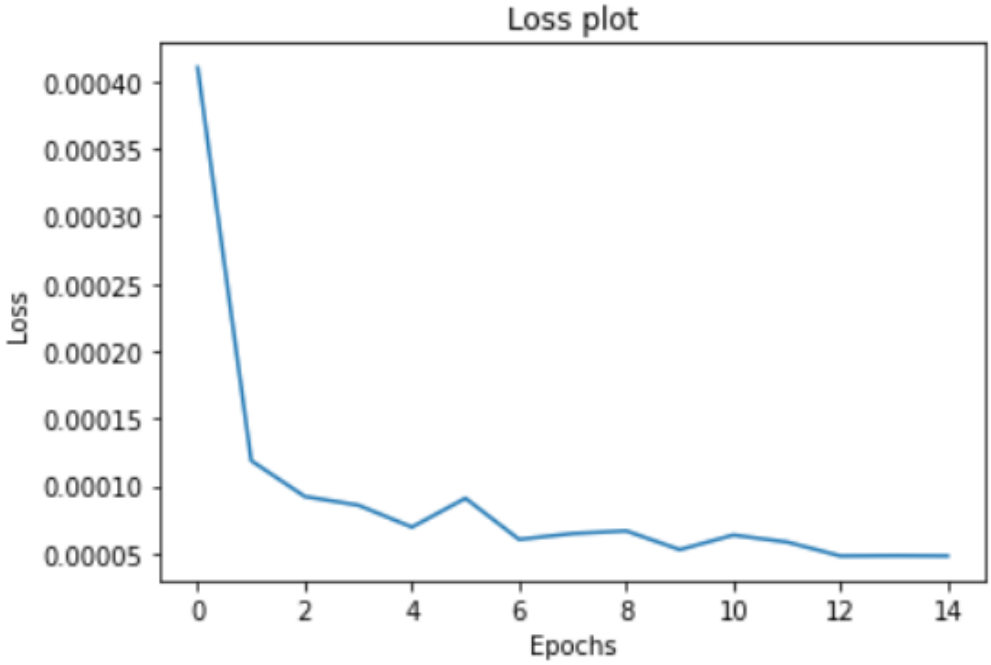
Preprocessing type	Time (میکرو ثانیه)	Test loss	خطای MAE بعد از 15 ایپاک
Power transformer	5.72	0.0374	0.0647
MinMax Scaling	6.77	0.0020	0.0318
Standard scaler	9.3	0.7670	0.6590

مشاهده می شود که MinMaxScaling که از ابتدا به کار برده بودیم از بقیه موارد بهتر کار میکند و این به این دلیل است که مقادیر ما را که بسیار بزرگ هستند، بین -۱ تا ۱ اسکیل کرده و از آن جایی که توزیع ما گوسی نبوده این مورد از standard scaler بسیار بهتر عمل میکند (تابع standard scaler فرض می کند که دیتا ی ما به طور نرمال فیچر ها را توزیع کرده است و فقط میانگین را به ۰ و واریانس را به ۱ می برد که چون فرض در ابتدا غلط بود نتیجه مطلوب حاصل نشد.) ،در power transformation نیز سعی این است که دیتا شبیه حالت گوسی شود اما اگر فیچر منفی داشته باشیم ، این مورد نیز به خوبی عمل نمیکند. همچنین MinMaxScaler برای مواردی که standard deviation مثل الان کوچک است ، مناسب است.

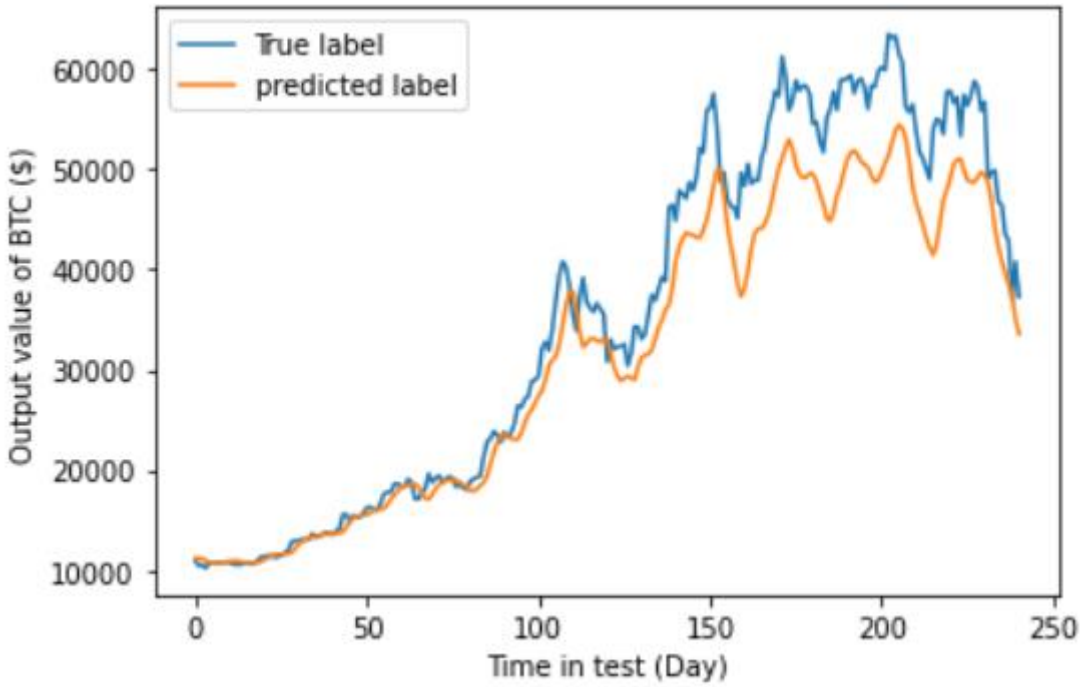
بنابراین بهترین نتیجه از MinMaxScaling حاصل می شود.

(۱۱

در این قسمت به این گونه عمل می کنیم که از آنجاییکه x_test مان چندین sequence دارد، ابتدا sequence اول آن را به مدل train شده میدهیم و خروجی را به عنوان آخرین sequence از ورودی بعدی که همان x_test است اما مقدار sequence آخرش با این مقدار خروجی شبکه که دفعه اول بدست آوردیم جایگزین می شود. در ادامه نیز همین روند را برای همه روز ها ادامه می دهیم.



شکل 1-43 – مقدار loss بدون دیدن لیبل حقیقی در GRU



شکل 1-44 – مقادیر پیش بینی شده و حقیقی بدون دیدن لیبل حقیقی در GRU

زمان اجرا :Wall time: 11.2 µs

مقدار loss : 0.0080

مقدار MAE : 0.0657

همان طور که می بینیم عملکرد هر ۳ این مقادیر به نسبت حالت برنده(GRU با یک لایه و بدون dropout و اپتیمایزر adam و تابع خطای MSE) بدتر شده اند و علت هم آن است که ما داریم از مقادیر پیش بینی شده قبلی استفاده می کنیم.

سوال 3 – آشنایی با کاربرد (شبکه های عصبی بازگشتی) در متن

در این سوال داده های فایل sentiment_final که محتوای تعدادی توییت هستند را بررسی می کنیم و پس از پیش پردازش های لازم، مدلی روی این داده های متنی آموزش می دهیم که مثبت یا منفی بودن محتوای توییت را حدس بزند. کد این سوال در فایل Project2_Q3.ipynb موجود است.

1- اکثر دیتاست های واقعی بالانس نیستند و تعداد یکی از کلاس ها بیشتر است که می تواند مشکل زا باشد. بزرگترین مشکلی که ایجاد می شود و از آن با پارادوکس دقت یاد می شود، این است که ممکن مدل ما به 90 درصد دقت دست یابد ولی این دقت صرفا حاصل از distribution داده ها باشد یعنی مدل ما درست آموزش ندیده است و فقط یک کلاس که تعداد بالایی دارد را خروجی می دهد و چون آن کلاس 90 درصد داده ها را تشکیل می دهد، دقت بالایی می گیریم. این اتفاق در داده هایی که بالانس نیستند زیاد اتفاق می افتد زیرا وقتی تعداد داده های یک کلاس بالا باشد روی وزن ها و پارامتر های مدل نیز بیشتر اثر می گذارد و مدل آموزش داده شده در اکثر مواقع لیبل پرتکرار را خروجی خواهد داد. از مسائلی که به خوبی این مشکل دیده می شود مسئله تشخیص سرطان و بیماری های دیگر است که معمولا کمتر از یک درصد از مردم دارن و وقتی یک دیتاست از مردم جامعه جمع می شود تا مدلی برای تشخیص آن آموزش داده شود اگر مدل ما روی تمامی افراد لیبل غیرسرطانی را بزند به دقت 99 درصد که خیلی بالا است می رسد و لی در عمل هیچ اطلاعاتی را به یاد نسپرده است. برای حل ایم مشکل روش ها و متدهای مختلفی پیش روی ما قرار دارد که به اختصار آنها را بحث می کنیم. اولین کار، حل مشکل از مبدا است یعنی داده های بیشتری از کلاس minority جمع کنیم و کل مشکل را حل کنیم ولی خوب معمولا این کار عملی نیست. راه دیگر زمانی که هیچ یک از متد ها جواب ندهد استفاده از متریک دیگری برای سنجش و آموزش مدل است یعنی به جای دقت که از داده ی imbalanced اثر می پذیرد، از معیارهای دیگری مثل f1 score، recall و precision استفاده کنیم. راه دیگر تغییر الگوریتم و یا اثر دادن پنالتی برای خروجی دادن لیبل اکثریت توسط مدل است. راه آخر که از بقیه موارد نیز بیشتر استفاده می شود resample کردن داده است که یا با undersample کلاس پرتعدادتر انجام می شود و یا با oversample کردن کلاس کم تعدادتر به عبارت دیگر یا تعدادی از داده های کلاس با مقادیر زیاد را حذف کرده و رندوم فقط بخشی از داده های آن را نگه می داریم که این کار وقتی تعداد داده های ما زیاد نیست اصلا توصیه نمی شود زیرا اطلاعاتی که با آنها می توانستیم مدل را بهبود دهیم را دور میریزیم و یا اینکه داده های کلاس با تعداد کمتر را چند بار کپی کرده و یا به کمک روش هایی آنها را augment کرده و یا داده های

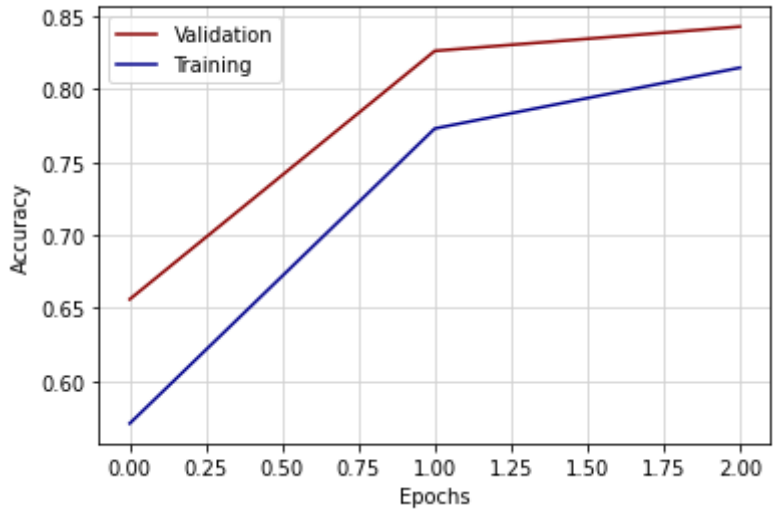
جدیدی از روی آنها می سازیم تا تعداد دو کلاس بالانس شود. این کار برای داده های عددی مناسب تر است ولی در این مساله چون جنس داده های ما از جنس متن هستند نمی توانیم داده هایی جدید generate کنیم که معنادار باشند پس چهار بار مقادیر کلاس positive را کپی می کنیم تا distribution این دو کلاس بالانس شود.

2- حال به سراغ پیش پردازش داده ها می رویم. ابتدا همه ی حروف را Lowercase می کنیم زیرا بزرگ و کوچکی کلمات اطلاعات اضافه تری نمی دهند و فقط پیچیدگی بیش از حد به مدل اضافه می کنند. سپس علایم punctuation را حذف می کنیم زیرا اطلاعاتی ندارند و قابل یادگیری نیستند. در مرحله بعد پیش پردازشی مناسب این تایپ داده یعنی توییت می کنیم و emoji ها را با کلمات معادلشان جایگزین می کنیم زیرا این نوع داده مدل را به مشکل می اندازد. در مرحله بعد URL ها و تگ های html را نیز حذف می کنیم. سپس جملات را به کلمات می شکنیم و stop words زبان انگلیسی را از آن حذف می کنیم یعنی کلماتی که در جملات خیلی تکرار می شوند و معنا و مفهومی برای فرایند یادگیری ما ندارند و باقی گذاشتن آنها موجب اختلال در عملکرد مدل است. در انتها نیز به کمک spell check اشتباهات تایپی را نیز برطرف می کنیم. از پیش پردازش های دیگر که انجام شد حذف اعداد بود همچنین lemmatize کردن را نیز قصد داشتیم بکنیم ولی چون متن ها توییت بودند و ممکن بود این پیش پردازش موثر نباشد و بدون آن نیز به دقت نسبتا خوبی رسیدیم از آن صرف نظر شد.

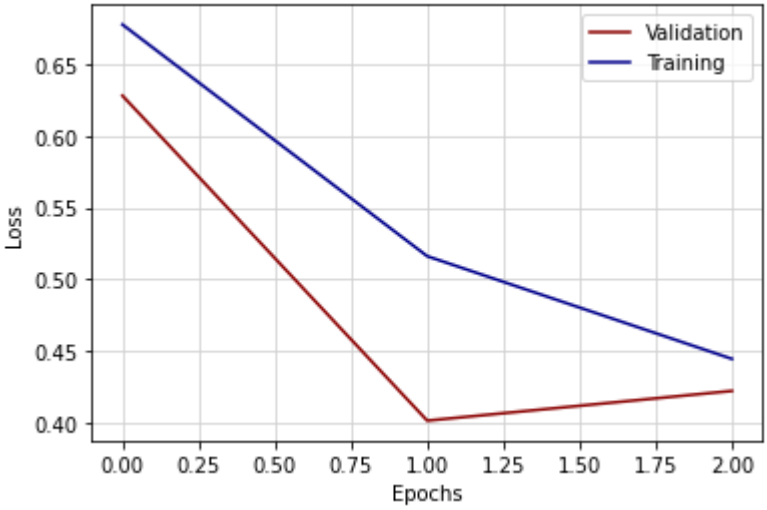
3- حال word embedding را با ساز 10000 کلمه پرتکرار انجام می دهیم.

4- سپس شبکه deep را پیاده سازی می کنیم که شامل یک لایه embedding و یک لایه lstm با 100 سلول و یک لایه Dense برای تولید خروجی مدنظر است. از optimizer معمول یعنی adam استفاده شد و چون شبکه recurrent است از activation function معمول این نوع شبکه یعنی sigmoid استفاده شد. معیار خا نیز binary_crossentropy است و در چند ایپاک آموزش انجام می شود که خاصیت شبکه های lstm است.

5- نمودار خطا و دقت مدل به شرح زیر است:



شکل3 - 1 : نمودار دقت مدل در ایپاک ها



شکل3 - 2 : نمودار خطا مدل در ایپاک ها

می توان دید که در چند ایپاک مدل آموزش دیده است و سریع Overfit می شود که در شبکه های LSTM زیاد دیده می شود و نیاز به early stopping دارند.

6- حال توسط مدل آموزش دیده predict می کنیم و نتایج به شرح زیر خواهد بود:

```
confusion matrix=
 [[1408    327]
 [456    1197]]


F1 Score: 75.35%
Precision: 78.54%
Recall: 72.41%
```

سوال 4 – آشنایی با مقالات مربوط

2 مقاله A COMPARISON OF TRANSFORMER AND LSTM و A New Concept using LSTM Neural Networks for Dynamic System Identification به پیوست در قالب ویدیو و پاورپوینت ارایه شده اند.