به نام خدا

نام: فاطمه زهرا بخشنده **استاد: دکتر مرضیه داود آبادی**

شماره دانشجویی: 98522157 **درس مبانی یادگیری عمیق**

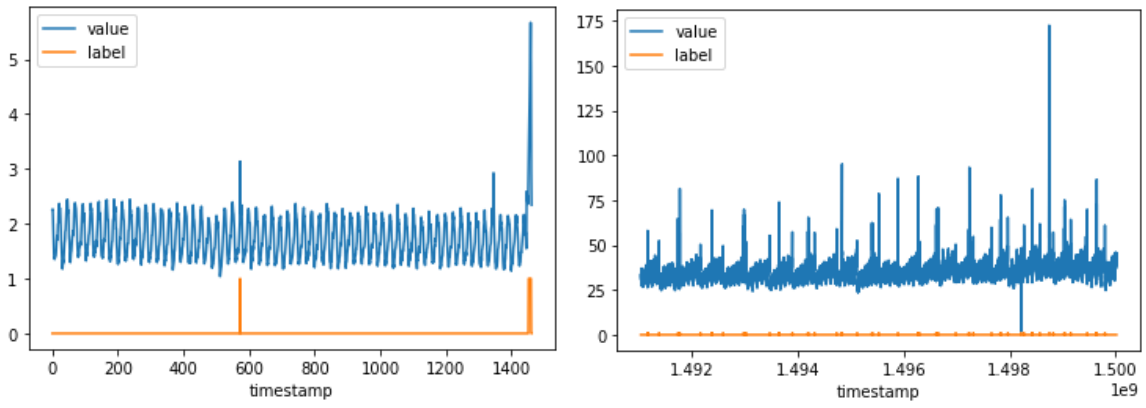
گزارش تمرین 5:

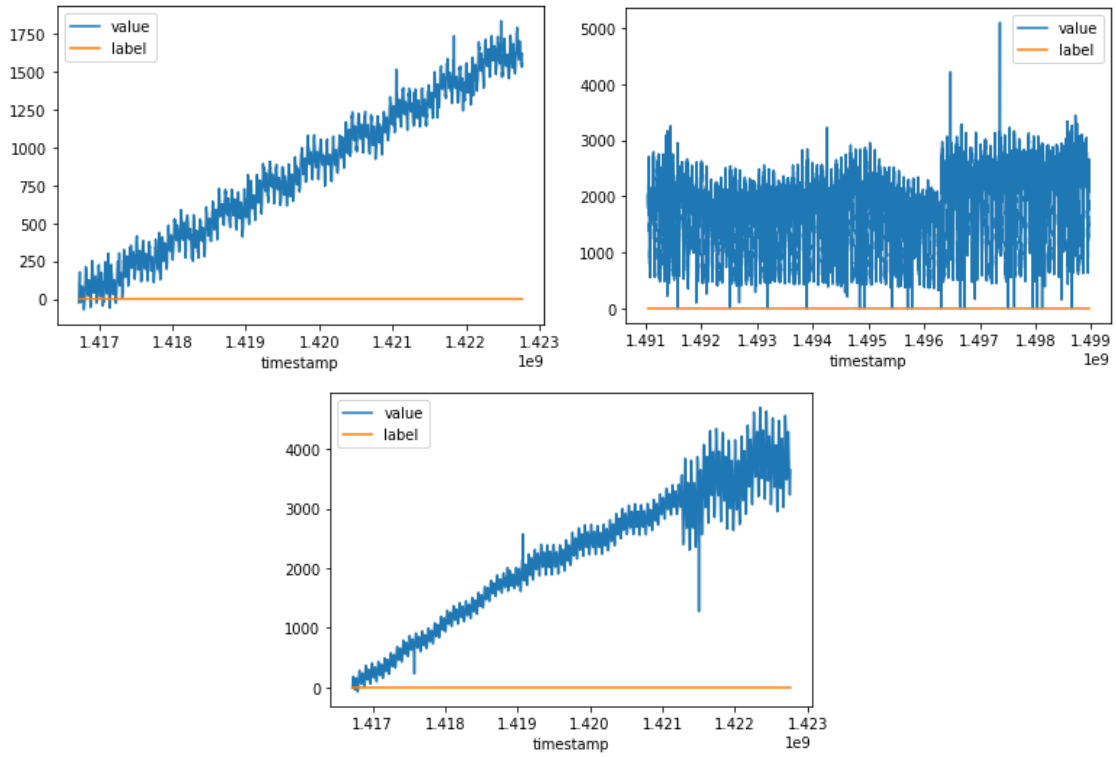
1- داده ها را load و unzip می کنیم. داده نمونه:

```
                value  label
timestamp
1493568000  1.901639      0
1493568060  1.786885      0
1493568120  2.000000      0
1493568180  1.885246      0
1493568240  1.819672      0
...              ...    ...
1501475400  2.684211      0
1501475460  2.526316      0
1501475520  2.614035      0
1501475580  2.736842      0
1501475640  2.491228      0

[128562 rows x 2 columns]
```

نمودار خواسته شده را برای 5 داده رندوم رسم می کنیم:





2- از آنجایی که با یک تسک مربوط به time series سر و کار داریم، مقدار خروجی به داده های قبلی نیز وابسته است و نیاز از مدل بازگشتی RNN استفاده کنیم.

همانطور که از نمودار ها نیز پیداست، داده ها بسیاز unbalance هستند و تعداد 0 ها در خروجی بسیار بیشتر از تعداد 1 هاست، که آموزش و ارزیابی را برای مدل سخت می کند، و مدل را به سمتی پیش می برد که خروجی بیشتر داده ها را 0 تشخیص دهد، و دیگر معیار هایی مثل accuracy برای ارزیابی مدل خوب نیستند.

تعداد داده ها زیاد است، و برخی از آن ها بسیار طولانی هستند. دامنه داده ها با هم متفاوت است. همچنین طول داده ها نیز با هم متفاوت است. به همین دلیل لازم است پیش از ورود داده ها به مدل، preprocess های مختلفی روی آن ها انجام داده و در آخر داده ها را در ادامه هم بچسبانیم و به مدل بدهیم.

چون باید ورودی را پنجره بندی کنیم و داده ها نیز طولانی هستند، زمان train روی این داده ها خیلی زیاد شد.

3- ابتدا داده ها را با انتخاب یک اندازه TIME_STEP، پنجره بندی می کنیم تا در پنجره های زمانی مختلف anomaly را بررسی کند. سپس همه داده ها را به هم می چسبانیم. داده ها را به train و test تقسیم می کنیم. تابع های build_model و compile_fit را می نویسیم تا برای مدل های مختلف از آن ها استفاده کنیم. از Early Stopping هم استفاده می کنیم تا زمانی که در val_f1_score_m بهبودی حاصل نشد train متوقف شود.

مدل simple RNN:

```
Model: "model_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_4 (InputLayer)        [(None, 32, 1)]           0

 simple_rnn_3 (SimpleRNN)    (None, 32)                1088

 dense_6 (Dense)             (None, 16)                528

 dense_7 (Dense)             (None, 1)                 17

=================================================================
Total params: 1,633
Trainable params: 1,633
Non-trainable params: 0
_________________________________________________________________
```

```
f1_score_m: 0.1989 - recall_m: 0.1818 - precision_m: 0.2486 - val_loss: 0.0679 - val_f1_score_m: 0.0556
```

مدل LSTM:

```
Model: "model_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_5 (InputLayer)        [(None, 32, 1)]           0

 lstm (LSTM)                 (None, 32)                4352

 dense_8 (Dense)             (None, 16)                528

 dense_9 (Dense)             (None, 1)                 17

=================================================================
Total params: 4,897
Trainable params: 4,897
Non-trainable params: 0
_________________________________________________________________
```

```
f1_score_m: 0.4548 - recall_m: 0.4379 - precision_m: 0.5061 - val_loss: 0.0339 - val_f1_score_m: 0.4655
```

مدل GRU:

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 32, 1)]           0

 gru (GRU)                   (None, 32)                3360

 dense (Dense)               (None, 16)                528

 dense_1 (Dense)             (None, 1)                 17

=================================================================
Total params: 3,905
Trainable params: 3,905
Non-trainable params: 0
```

```
f1_score_m: 0.4877 - recall_m: 0.4785 - precision_m: 0.5285 - val_loss: 0.0331 - val_f1_score_m: 0.4897
```

نتیجه مقایسه val_f1_score به صورت زیر است:

GRU > LSTM > Simple

4- از دو روش MinMaxScaler و Norm L2 برای normalization داده ها استفاده می کنیم.
MinMaxScalar ویژگی ها را با مقیاس بندی هر ویژگی به یک محدوده مشخص تبدیل می کند. این محدوده را می توان با تعیین پارامتر feature_range (به طور پیش فرض در (0,1)) تنظیم کرد. این scaler برای مواردی که توزیع گاوسی نیست یا انحراف معیار بسیار کم است بهتر عمل می کند.

$$x\_scaled = (x-min(x)) / (max(x)-min(x))$$

همچنین L2 Norm از square root همه مقادیر مجذور استفاده می کند.

$$x\_normalized = x / sqrt(sum((i^{**}2) \text{ for } i \text{ in } X))$$

پس از انجام preprocess ها، مدل ها را با دیتای جدید train می کنیم. نتیجه به صورت زیر است.

مدل simple RNN:

```
f1_score_m: 0.1779 - recall_m: 0.1607 - precision_m: 0.2264 - val_loss: 0.0732 - val_f1_score_m: 0.1377
```

مدل LSTM:

```
f1_score_m: 0.4636 - recall_m: 0.4482 - precision_m: 0.5123 - val_loss: 0.0357 - val_f1_score_m: 0.4510
```

مدل GRU:

```
- f1_score_m: 0.5064 - recall_m: 0.4958 - precision_m: 0.5465 - val_loss: 0.0309 - val_f1_score_m: 0.4959
```

در این حالت نتایج بهتری حاصل شد. f1_score برای مدل های GRU و LSTM افزایش یافت، و val_f1_score برای مدل های Simple و GRU افزایش یافت.

5- همانطور که گفتیم، در مسئله anomaly detection داده ها بسیار unbalance هستند و تعداد 0 ها در خروجی بسیار بیشتر از تعداد 1 هاست، که آموزش و ارزیابی را برای مدل سخت می کند، و مدل را به سمتی پیش می برد که خروجی بیشتر داده ها را 0 تشخیص دهد، در این حالت accuracy به مقدار خیلی خوبی می رسد و loss نیز minimum می شود، در صورتی که مدل در این حالت نمی‌تواند anomaly را تشخیص دهد و همه خروجی ها را 0 می دهد. پس معیار هایی مثل accuracy و loss برای ارزیابی این مدل خوب نیستند.
همچنین loss برای بهینه سازی مدل است و برای مقایسه مدل ها خوب نیست. پس از معیار f1 score استفاده می کنیم که تمام مقادیر را در نظر گرفته و برای این مسئله مناسب تر است.

6- ابتدا یک تسک supervised-Self مرتبط با مسئله اصلی تعریف می کنیم که سری زمانی را در time step های بعدی پیش بینی می کند. دیتاست مناسب را با توجه به تسکمان از روی دیتاست اصلی میسازیم. و مدل خود را تعریف و روی دیتاست جدید آموزش می دهیم.

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 32, 1)]           0

 lstm (LSTM)                 (None, 32, 32)            4352

 dropout (Dropout)           (None, 32, 32)            0

 lstm_1 (LSTM)               (None, 16)                3136

 dense (Dense)               (None, 1)                 17

=================================================================
Total params: 7,505
Trainable params: 7,505
Non-trainable params: 0
```

```
f1_score_m: 0.9873 - recall_m: 0.9810 - precision_m: 0.9948 - val_loss: 34289678336.0000 - val_f1_score_m: 1.0287
```

مدل را که با عنوان self_supervised.h5 ذخیره کردیم لود کرده و لایه آخر آن را حذف می کنیم. لایه های میانی آن را freeze می کنیم و 2 لایه dense به آن اضافه می کنیم.

```
Model: "model_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 32, 1)]           0

 lstm (LSTM)                 (None, 32, 32)            4352

 dropout (Dropout)           (None, 32, 32)            0

 lstm_1 (LSTM)               (None, 16)                3136

 dense_3 (Dense)             (None, 8)                 136

 dense_4 (Dense)             (None, 1)                 9

=================================================================
Total params: 7,633
Trainable params: 3,281
Non-trainable params: 4,352
```

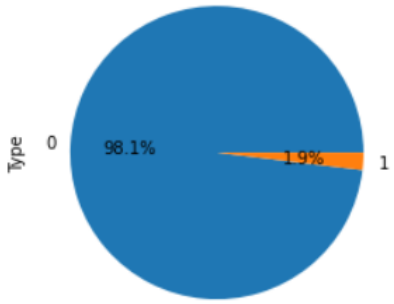
مدل را برای تسک اصلی خودمان fine tune می کنیم.

```
f1_score_m: 0.1798 - recall_m: 0.1601 - precision_m: 0.2324 - val_loss: 0.0562 - val_f1_score_m: 0.2404 -
```

در این حالت مدل زودتر شروع به یادگیری کرد و از ابتدا از جای بهتری شروع کرد. نتیجه آن از مدل simple RNN بهتر شد اما از مدل های LSTM و GRU بهتر نشد.

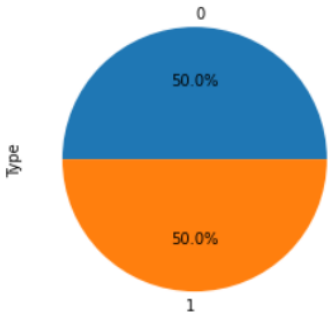
7- ابتدا تعداد داده های با برچسب 0 و تعداد داده های با برچسب 1 را بررسی می کنیم. با توجه به نمودار زیر در می یابیم نسبت داده های با برچسب 0 بسیار بیشتر است و دیتا بسیار نامتعادل است.

```
0    1484686
1      28795
Name: label, dtype: int64
<matplotlib.axes._subplots.AxesSubplot at 0x7ff0948bb1c0>
```



برای حل این مشکل می توان از Upsampling یا Downsampling استفاده کرد. Downsampling برای حذف رکوردها از کلاس های اکثریت به منظور ایجاد یک مجموعه داده متعادل تر انجام می شود. ساده‌ترین راه برای Downsampling کلاس اکثریت، حذف تصادفی رکوردها از آن دسته است.
این متد را روی دیتاست اعمال کرده و داده ها به صورت زیر کاملا متعادل می شوند.

```
0    2332395
1    2332395
Name: label, dtype: int64
<matplotlib.axes._subplots.AxesSubplot at 0x7ff09431eeb0>
```



چون مدل GRU بهترین عملکرد را تا به الان داشته است از این مدل برای train داده متعادل شده استفاده می‌کنیم.

```
f1_score_m: 0.9992 - recall_m: 0.9994 - precision_m: 0.9991 - val_loss: 0.0050 - val_f1_score_m: 0.9980
```

می بینیم که پیشرفت بزرگی حاصل شد و نتیجه تغییر بسیار خوبی کرد. به val_f1_score برابر 0.998 رسیدیم!

8- امتیازی: از روش آماری Zscore برای حل این مسئله استفاده می کنیم. Z-score یک معیار آماری مفید برای تشخیص ناهنجاری است. در یک توزیع آماری، Z-score به ما می گوید که دیتای داده شده چقدر از بقیه داده ها فاصله دارد.

$$Z\text{-}score = \frac{x - mean}{Standard\ Deviation}$$

برای محاسبه این معیار، از متد resample در کتابخانه scipy.stats استفاده می کنیم. سپس مانند کاری که در استفاده از مدل های RNN انجام می دادیم، با حرکت روی داده ها، بررسی می کنیم که داده مورد نظر در یک پنجره پیش از خودش چقدر به میانگین نزدیک است. اگر مقدار Zscore آن از یک یک threshold بیشتر بود anomaly را تشخیص داده و label این داده را 1 در نظر می گیریم. در غیر این صورت label این داده 0 می شود.
برای ارزیابی خروجی ای که با این روش بدست آوردیم، آن را با خروجی اصلی داده ها مقایسه می کنیم و accuracy و f1_score را برای آن محاسبه می کنیم.

```
accuracy: 50.54 %
```

```
f1_score: 0.549748242
```

نتیجه تقریبا نزدیک نتیجه بدست آمده از مدل GRU است ولی از آن بالاتر است. با عوض کردن threshold می توان به نتیجه بالاتری هم رسید. مثلا با تغییر threshold به 0.4، anomaly بیشتری تشخیص داده می‌شود به همین دلیل هم f1_score افزایش می یابد و به صورت زیر می شود:

```
f1_score: 0.742108
```

اما در این حالت دیتا های عادی بیشتری نیز به عنوان anomaly شناسایی شده اند.
در کل با توجه به تحقیقی که انجام دادم، بهتر است برای تسک anomaly detection در سری زمانی، از الگوریتم های دیگر Machine Learning نظیر الگوریتم های زیر استفاده کنیم:

- Support vector machine learning
- k-nearest neighbors (KNN)
- Bayesian networks
- Decision trees

منابع: لینک و لینک و لینک و لینک و لینک و لینک و لینک