به نام خدا

درس یادگیری عمیق

تمرین سری پنجم

مدرس درس:
سرکار خانم دکتر داوودآبادی

تهیه شده توسط:
الناز رضایی ۹۸۴۱۱۳۸۷

تاریخ ارسال: ۱۴۰۱/۱/۱٪

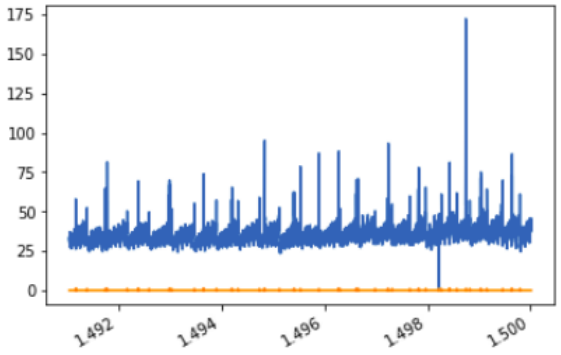
در این تمرین قرار است با استفاده از مدل‌های RNN به تشخیص ناهنجاری‌های موجود در مجموعه داده سری زمانی پرداخته شود. در این مسئله تعدادی داده سری زمانی از دامنه‌های مختلف در اختیار شما قرار داده شده است و وظیفه شما طراحی یک مدل برای تشخیص خودکار ناهنجاری در دامنه‌های مختلف است. منظور از دامنه این است که ممکن است Time step ،کران بالا و کران پایین هر سری از داده‌ها متفاوت باشد. برای مثال ممکن است یک مجموعه داده، داده‌هایی بین ۱- تا ۱ و مجموعه‌ای دیگر داده‌هایی بین ۱۰۰۰- تا ۱۰۰۰ داشته باشد.

این تمرین یک مسئله کلاس بندی باینری است، کلاس ۱ نشان‌دهنده این است که داده مورد نظر خراب و ناهنجار است و ۰ نشان‌دهنده این است که داده مورد نظر مشکلی ندارد.

توجه: در این مسئله متریک مورد نظر F1-score است و توجه شما باید روی این باشد که این معیار را افزایش دهید.

سوال ۱:

ابتدا داده‌ها را Load کنید و آن‌ها را در یک نمودار رسم کنید. (۵ نمره)



عکس1 یک نمونه از نمودار مورد نظر برای یک مجموعه داده سری زمانی

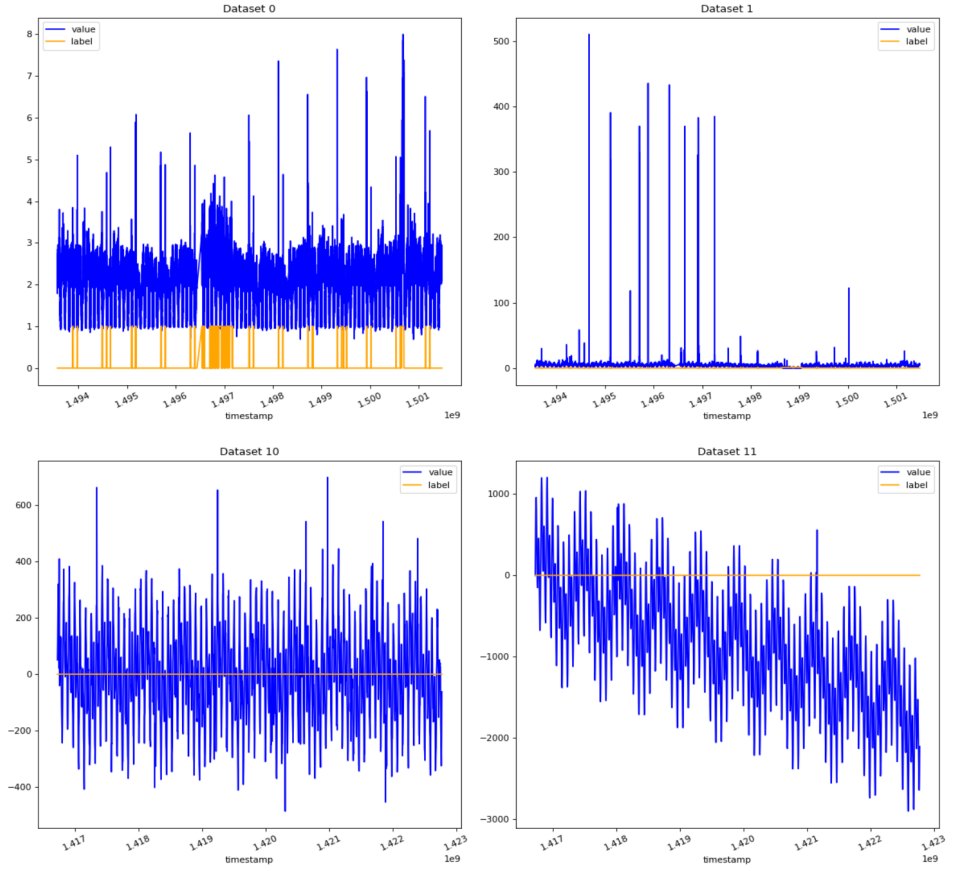
نمودار آبی نشان‌دهنده تغییرات سری زمانی و خط نارنجی نشان‌دهنده برچسب هر Time step است.

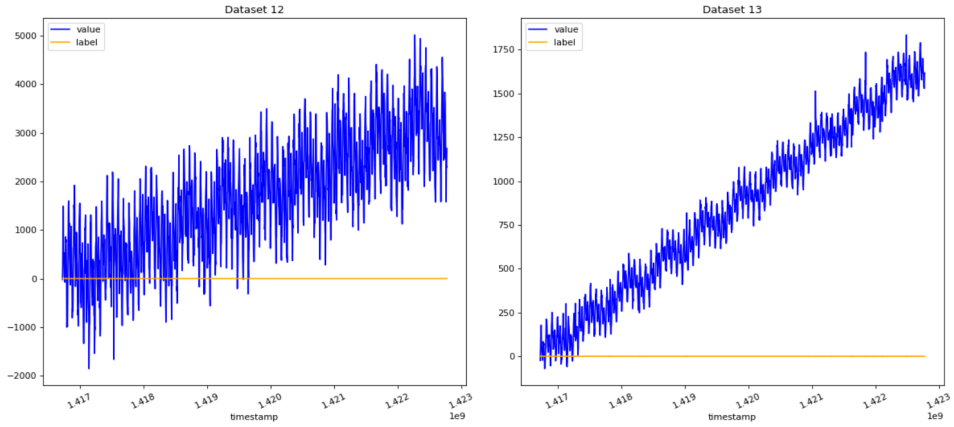
پاسخ ۱:

با استفاده از کدهای زیر، ۶ فایل csv را انتخاب کرده و هر کدام را در یک نمودار رسم می‌کنیم. رنگ آبی مقدار، و رنگ نارنجی labelهای موجود در هر فایل را نمایش می‌دهد.

```python
# show chart of 5 dataset(csv file) randomly

csv_path = ["0.csv", "1.csv", "10.csv", "11.csv", "12.csv", "13.csv"]
dataset = []
for i in range(6):
  dataset.append(pd.read_csv(csv_path[i]))

titles = [
  "Dataset 0",
  "Dataset 1",
  "Dataset 10",
  "Dataset 11",
  "Dataset 12",
  "Dataset 13"
]

feature_keys = [
  "value",
  "label",
]

colors = [
  "blue",
  "orange",
]

date_time_key = "timestamp"

def show_raw_visualization(data):
  i = 0
  j = 0
  k = 0
  fig, axes = plt.subplots(
    nrows=3, ncols=2, figsize=(15, 20), dpi=80, facecolor="w", edgecolor="k"
  )
  for d in data:
    time_data = d[date_time_key]
    value = feature_keys[0]
    label = feature_keys[1]
    c_value = colors[0]
    c_label = colors[1]
    t_value = d[value]
    t_label = d[label]
    t_value.index = time_data
    t_value.head()
    t_label.index = time_data
    t_label.head()
    ax = t_value.plot(
      ax=axes[i, j],
      color=c_value,
      title="{}".format(titles[k]),
      rot=25,
    )
    ax = t_label.plot(
      ax=axes[i, j],
      color=c_label,
      title="{}".format(titles[k]),
      rot=25,
    )

    if (j == 0):
      j = 1

    elif (j == 1):
      j = 0
      i += 1

    k += 1
  ax.legend(feature_keys)
  plt.tight_layout()

show_raw_visualization(dataset)
```

نتایج به دست آمده برای این بخش، در تصاویر زیر آورده شده‌اند.





سوال ۲:

نمودارها را تحلیل کنید و چالش‌های آموزش این داده‌ها در یک مدل RNN را نام ببرید. (ذکر حداقل ۲ مورد) (۱۰ نمره)

پاسخ ۲:

این نمودارها همگی دارای مقدار value متغیر، مثلا نمودار اول بین ۰ و ۸، نمودار دوم بین ۰ و ۵۰۰، و مقدار label باینری ۰ و ۱ هستند. از مشکلات این نمودارها، می‌توان به این بازه بسیار متغیر value در هر نمودار و balance نبودن data(برابر نبودن تعداد ۰ و ۱ در هر نمودار) اشاره کرد.

سوال ۳:

سه مدل RNN (GRU، LSTM، Simple)طراحی کنید و داده‌ها را بدون هیچ پیش‌پردازش ی به این مدل‌ها بدهید. آن‌ها را آموزش داده و سپس نتایج را با هم مقایسه کنید. (۱۵ نمره)

پاسخ ۳:

- Simple RNN: در ابتدا مدل Simple RNN خود را مطابق شکل زیر پیاده‌سازی می‌کنیم.

```python
# Simple
def Simple_RNN(hidden_units, dense_units, last_units, input_shape, activation):
  model = Sequential()
  model.add(SimpleRNN(hidden_units, input_shape=input_shape))
  model.add(Dense(units=dense_units, activation=activation[0]))
  model.add(Dense(units=last_units, activation=activation[1]))
  model.compile(loss='binary_crossentropy', optimizer='adam', metrics=(f1_score_m,))
  return model

simple_model = Simple_RNN(128, 64, 1, (1, 1), ['relu', 'sigmoid'])
history = simple_model.fit(x_train, y_train, epochs=EPOCHS, batch_size=BATCH_SIZE, validation_data=(x_test, y_test))
```

```
Epoch 1/5
8869/8869 [==============================] - 42s 5ms/step - loss: 0.0798 - f1_score_m: 0.0454 - val_loss: 0.0778 - val_f1_score_m: 0.1007
Epoch 2/5
8869/8869 [==============================] - 41s 5ms/step - loss: 0.0773 - f1_score_m: 0.1203 - val_loss: 0.0783 - val_f1_score_m: 0.1457
Epoch 3/5
8869/8869 [==============================] - 41s 5ms/step - loss: 0.0757 - f1_score_m: 0.1322 - val_loss: 0.0756 - val_f1_score_m: 0.0950
Epoch 4/5
8869/8869 [==============================] - 41s 5ms/step - loss: 0.0751 - f1_score_m: 0.1365 - val_loss: 0.0754 - val_f1_score_m: 0.1134
Epoch 5/5
8869/8869 [==============================] - 49s 5ms/step - loss: 0.0749 - f1_score_m: 0.1368 - val_loss: 0.0753 - val_f1_score_m: 0.1142
```

- LSTM: حال مدل خود را با استفاده از GRU پیاده‌سازی می‌کنیم. این مدل، دارای ۳ گیت است و ۴ برابر Simple RNN پارامتر دارد. کندتر است و از محوشدگی جلوگیری می‌کند. بنابراین توقع داریم این مدل، عملکرد بهتری نسبت به حالت قبل داشته باشد.

```python
# LSTM
def LSTM_RNN(hidden_units, dense_units, last_units, input_shape, activation):
  model = Sequential()
  model.add(LSTM(hidden_units, input_shape=input_shape))
  model.add(Dense(units=dense_units, activation=activation[0]))
  model.add(Dense(units=last_units, activation=activation[1]))
  model.compile(loss='binary_crossentropy', optimizer='adam', metrics=(f1_score_m,))
  return model

lstm_model = LSTM_RNN(128, 64, 1, (1, 1), ['relu', 'sigmoid'])
history = lstm_model.fit(x_train, y_train, epochs=EPOCHS, batch_size=BATCH_SIZE, validation_data=(x_test, y_test))
```

```
Epoch 1/5
8869/8869 [==============================] - 55s 6ms/step - loss: 0.0746 - f1_score_m: 0.0064 - val_loss: 0.0697 - val_f1_score_m: 0.0553
Epoch 2/5
8869/8869 [==============================] - 44s 5ms/step - loss: 0.0692 - f1_score_m: 0.0584 - val_loss: 0.0690 - val_f1_score_m: 0.0597
Epoch 3/5
8869/8869 [==============================] - 48s 5ms/step - loss: 0.0680 - f1_score_m: 0.0876 - val_loss: 0.0680 - val_f1_score_m: 0.1140
Epoch 4/5
8869/8869 [==============================] - 47s 5ms/step - loss: 0.0673 - f1_score_m: 0.1197 - val_loss: 0.0688 - val_f1_score_m: 0.1176
Epoch 5/5
8869/8869 [==============================] - 44s 5ms/step - loss: 0.0670 - f1_score_m: 0.1351 - val_loss: 0.0691 - val_f1_score_m: 0.1575
```

همانطور که مشاهده می‌شود، loss روی داده‌های validation در این حالت کاهش یافته و دقت افزایش می‌یابد که مطابق پیش‌بینی ما است.

- GRU: در این بخش، مدل خود را با GRU پیاده‌سازی می‌کنیم. GRU دارای دو گیت اضافه‌تر با نام‌های Reset و Update نسبت به Simple RNN می‌باشد. پارامترهای آن ۳ برابر Simple RNN است و مسائل را بهتر حل می‌کند. GRU نسبت به LSTM جدیدتر و سریع‌تر است؛ چرا که تعداد پارامترهای کمتری نسبت به LSTM دارد.

```python
# GRU
def GRU_RNN(hidden_units, dense_units, last_units, input_shape, activation):
  model = Sequential()
  model.add(LSTM(hidden_units, input_shape=input_shape))
  model.add(Dense(units=dense_units, activation=activation[0]))
  model.add(Dense(units=last_units, activation=activation[1]))
  model.compile(loss='binary_crossentropy', optimizer='adam', metrics=(f1_score_m,))
  return model

gru_model = GRU_RNN(128, 64, 1, (1, 1), ['relu', 'sigmoid'])
history = gru_model.fit(x_train, y_train, epochs=EPOCHS, batch_size=BATCH_SIZE, validation_data=(x_test, y_test))
```

```
Epoch 1/5
8869/8869 [==============================] - 54s 6ms/step - loss: 0.0762 - f1_score_m: 7.9552e-04 - val_loss: 0.0703 - val_f1_score_m: 0.0000e+00
Epoch 2/5
8869/8869 [==============================] - 48s 5ms/step - loss: 0.0693 - f1_score_m: 0.0226 - val_loss: 0.0681 - val_f1_score_m: 0.0809
Epoch 3/5
8869/8869 [==============================] - 49s 6ms/step - loss: 0.0681 - f1_score_m: 0.0782 - val_loss: 0.0674 - val_f1_score_m: 0.1211
Epoch 4/5
8869/8869 [==============================] - 48s 5ms/step - loss: 0.0676 - f1_score_m: 0.1064 - val_loss: 0.0673 - val_f1_score_m: 0.1287
Epoch 5/5
8869/8869 [==============================] - 50s 6ms/step - loss: 0.0673 - f1_score_m: 0.1190 - val_loss: 0.0685 - val_f1_score_m: 0.1319
```

همانطور که مشخص است، در این حالت f1_score نسبت به Simple RNN بیشتر و نسبت به LSTM کمتر است.

سوال ۴:

چند روش برای پیش‌پردازش داده‌ها ارائه دهید که باعث شود یادگیری آن‌ها توسط مدل‌ها ساده‌تر شود. (حداقل ۲) (۱۰ نمره)

پاسخ ۴:

- روش اول) StandardScaler(): با حذف میانگین و مقیاس بندی به واریانس واحد، ویژگی‌ها را استاندارد می‌کنیم. نمره استاندارد یک نمونه x به صورت زیر محاسبه می‌شود:

$$z = \frac{x - u}{s}$$

u در اینجا میانگین و s واریانس می‌باشد. z نیز داده نرمال شده است.

```python
# Preprocces method 1
obj = StandardScaler()
obj.fit(x_train.reshape((-1, 1)))
x_train_standard = obj.transform(x_train.reshape((-1,1))).reshape((-1,1,1))
x_test_standard = obj.transform(x_test.reshape((-1,1))).reshape((-1,1,1))
```

- روش دوم) ()Minmaxscalar: با مقیاس بندی هر feature به یک محدوده مشخص، ویژگی‌ها را تغییر می‌دهد. این estimator هر feature را به صورت جداگانه مقیاس و ترجمه می‌کند به طوری که در محدوده داده شده در مجموعه آموزشی قرار گیرد، به عنوان مثال. بین صفر و یک.

```
# Preprocces method 2
obj = MinMaxScaler()
obj.fit(x_train.reshape((-1, 1)))
x_train_standard = obj.transform(x_train_standard.reshape((-1,1))).reshape((-1,1,1))
x_test_standard = obj.transform(x_test_standard.reshape((-1,1))).reshape((-1,1,1))
```

حال داده‌های preprocess شده را به مدل‌های خود می‌دهیم. توقع داریم با این کار مدل‌ها بهتر train شوند ولی در این مثال، به علت تغییرات شدید در مقادیر، مدل نه تنها بهبود پیدا نمی‌کند، بلکه عملکرد ضعیف‌تری از خود نشان می‌دهد. مقدار f1__score آن برای هر مدل کاهش یافته و loss نسبت به حالت قبل به ازای هر مدل افزایش می‌یابد.

- Simple RNN:

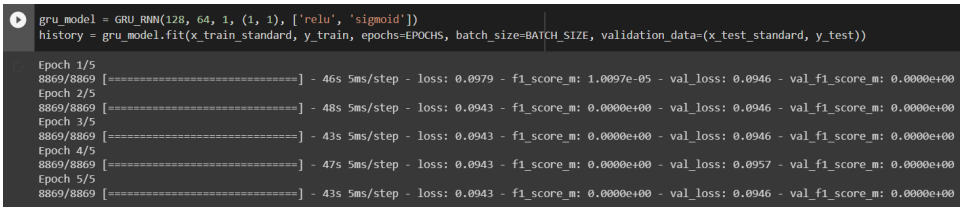
```
# Train 3 models again
# Simple RNN
simple_model = Simple_RNN(128, 64, 1, (1, 1), ['relu', 'sigmoid'])
history = simple_model.fit(x_train_standard, y_train, epochs=EPOCHS, batch_size=BATCH_SIZE, validation_data=(x_test_standard, y_test))
```

```
Epoch 1/5
8869/8869 [==============================] - 46s 5ms/step - loss: 0.0964 - f1_score_m: 8.4776e-06 - val_loss: 0.0946 - val_f1_score_m: 0.0000e+00
Epoch 2/5
8869/8869 [==============================] - 46s 5ms/step - loss: 0.0944 - f1_score_m: 0.0000e+00 - val_loss: 0.0948 - val_f1_score_m: 0.0000e+00
Epoch 3/5
8869/8869 [==============================] - 40s 4ms/step - loss: 0.0944 - f1_score_m: 0.0000e+00 - val_loss: 0.0948 - val_f1_score_m: 0.0000e+00
Epoch 4/5
8869/8869 [==============================] - 46s 5ms/step - loss: 0.0943 - f1_score_m: 0.0000e+00 - val_loss: 0.0947 - val_f1_score_m: 0.0000e+00
Epoch 5/5
8869/8869 [==============================] - 45s 5ms/step - loss: 0.0943 - f1_score_m: 0.0000e+00 - val_loss: 0.0948 - val_f1_score_m: 0.0000e+00
```

- LSTM:

```
[55] lstm_model = LSTM_RNN(128, 64, 1, (1, 1), ['relu', 'sigmoid'])
     history = lstm_model.fit(x_train_standard, y_train, epochs=EPOCHS, batch_size=BATCH_SIZE, validation_data=(x_test_standard, y_test))

     Epoch 1/5
     8869/8869 [==============================] - 45s 5ms/step - loss: 0.0976 - f1_score_m: 8.4776e-06 - val_loss: 0.0948 - val_f1_score_m: 0.0000e+00
     Epoch 2/5
     8869/8869 [==============================] - 44s 5ms/step - loss: 0.0943 - f1_score_m: 0.0000e+00 - val_loss: 0.0946 - val_f1_score_m: 0.0000e+00
     Epoch 3/5
     8869/8869 [==============================] - 46s 5ms/step - loss: 0.0943 - f1_score_m: 0.0000e+00 - val_loss: 0.0946 - val_f1_score_m: 0.0000e+00
     Epoch 4/5
     8869/8869 [==============================] - 44s 5ms/step - loss: 0.0943 - f1_score_m: 0.0000e+00 - val_loss: 0.0950 - val_f1_score_m: 0.0000e+00
     Epoch 5/5
     8869/8869 [==============================] - 47s 5ms/step - loss: 0.0943 - f1_score_m: 0.0000e+00 - val_loss: 0.0946 - val_f1_score_m: 0.0000e+00
```

- GRU:



سوال ۵:

توضیح دهید چرا معیارهای دیگر نظیر Loss و Accuracy برای ارزیابی مدل‌ها مناسب نیستند. (۱۰ نمره)

پاسخ ۵:

در ابتدا معیارهای Precision، Recall و f1_score را به همراه کابرد هر یک توضیح می‌دهیم.

Percision: درصد نمونه‌هایی که توسط مدل به عنوان کلاس مثبت تشخیص داده شده‌اند و درست بودند. در واقع، Precision به ما می‌گوید مدل در پیش‌بینی‌هایی که به عنوان کلاس مثبت داشته چقدر دقیق بوده است. این معیار برای زمانی مناسب است که هزینه False Positive زیاد است. مثال نباید ایمیلی که اسپم نیست را اسپم پیش‌بینی کنیم. مقدار Precision، از رابطه زیر به دست می‌آید:

$$Precision = \frac{TP}{TP + FP}$$

Recall: درصد نمونه‌هایی که مثبت بوده‌اند و به درستی توسط مدل تشخیص داده شده‌اند. این معیار برای زمانی مناسب است که که هزینه False Negative زیاد است. مثال اگر یک فرد بیمار را سالم تشخیص بدهیم بسیار بد است. مقدار Recall نیز توسط رابطه زیر تعریف می‌شود.

$$Recall = \frac{TP}{TP + FN}$$

F1-score: این معیار از ترکیب Precision و Recall به دست می‌آید و PR را با یک عدد خلاصه می‌کند. این معیار زمانی مناسب است که می‌خواهیم یک Balance میان Precision و Recall

برقرار کنیم. همچنین این معیار برای زمانی که که توزیع داده‌ها در کلاس‌ها نابرابر است مناسب است. فرمول این معیار، به شرح زیر است.

$$F_1 - score = \frac{2PR}{P+R}$$

در این مسئله چون داده‌ها balance نیست و توزیع داده‌ها در کلاس‌ها نابرابر است، و همچنین هزینه False Positive یا False Negative زیاد است، معیار accuracy مناسب نیست. معیار مناسب در این مسئله، f1_score است که ترکیبی از هر دو معیار Precision و Recall است.

سوال ۶:

در این مسئله بررسی می‌کنیم که آیا می‌توان با استفاده از pre-training نتیجه بهتری نسبت به مدل‌هایی که در مراحل قبل پیاده‌سازی شده است گرفت یا خیر. به آموزش یک مدل بر روی یک تسک یا مجموعه داده و استفاده از پارامترها و وزن‌های آن در مدل دیگری برای یک تسک یا مجموعه داده‌ای متفاوت، pre-training می‌گویند.

- ابتدا یک تسک Self-supervised مرتبط با مسئله اصلی تعریف کنید. (راهنمایی: تسک تعریف شده می‌تواند پیش‌بینی سری زمانی در Time step های بعدی باشد. یا می‌توان تعدادی از داده‌ها را حذف کرد و تسک شما این باشد که مدل داده‌های حذف شده را تخمین زده و بازیابی کند.)
- یک مدل برای تسک تعریف شده طراحی و پیاده‌سازی کرده و آموزش دهید.
- لایه‌های انتهایی مدل را طوری تغییر داده تا مدل متناسب با تسک اصلی شود.
- لایه‌های ابتدایی را فریز کرده و مدل را روی تسک اصلی Fine-tune کنید.

آیا نتیجه بهتری حاصل شد؟ (۳۰ نمره)

پاسخ ۶:

- ابتدا، مطابق پیشنهاد داده شده در صورت سوال، پیش‌بینی سری زمانی در تایم استپ‌های بعدی را برای داده‌های آموزشی و تست به دست می‌آوریم.

```
x_train_pred, y_train_pred, x_test_pred, y_test_pred = [], [], [], []
x_train_flatten = x_train.flatten()
x_test_flatten = x_test.flatten()
for i in range(0, len(x_train_flatten) - 1):
  x_train_pred.append(x_train_flatten[i])
  y_train_pred.append(x_train_flatten[i + 1])
x_train_pred, y_train_pred = np.array(x_train_pred).reshape((-1, 1)), np.array(y_train_pred)
for i in range(0, len(x_test_flatten) - 1):
  x_test_pred.append(x_test_flatten[i])
  y_test_pred.append(x_test_flatten[i + 1])
x_test_pred, y_test_pred = np.array(x_test_pred).reshape((-1, 1)), np.array(y_test_pred)
```

- همانطور که مشخص است، با استفاده از این روش، مقدار f1__score به مقدار قابل توجهی افزایش یافت.

```
# compile and train the model
model = keras.Sequential()
model.add(LSTM(128, input_shape=(1, 1), return_sequences=True))
model.add(LSTM(64, return_sequences=True))
model.add(Dense(1))
model.compile(loss='mse', optimizer='adam', metrics=(f1_score_m,))
history = simple_model.fit(x_train_pred, y_train_pred, epochs=EPOCHS, batch_size=BATCH_SIZE, validation_data=(x_test_pred, y_test_pred))
```

```
Epoch 1/5
8869/8869 [==============================] - 41s 5ms/step - loss: -1006606656.0000 - f1_score_m: 0.9363 - val_loss: -1172369536.0000 - val_f1_score_m: 0.9361
Epoch 2/5
8869/8869 [==============================] - 40s 5ms/step - loss: -1303572992.0000 - f1_score_m: 0.9363 - val_loss: -1494534272.0000 - val_f1_score_m: 0.9361
Epoch 3/5
8869/8869 [==============================] - 46s 5ms/step - loss: -1634746112.0000 - f1_score_m: 0.9363 - val_loss: -1850554624.0000 - val_f1_score_m: 0.9361
Epoch 4/5
8869/8869 [==============================] - 43s 5ms/step - loss: -2004150528.0000 - f1_score_m: 0.9363 - val_loss: -2245633280.0000 - val_f1_score_m: 0.9361
Epoch 5/5
8869/8869 [==============================] - 45s 5ms/step - loss: -2407468288.0000 - f1_score_m: 0.9363 - val_loss: -2679506944.0000 - val_f1_score_m: 0.9361
```

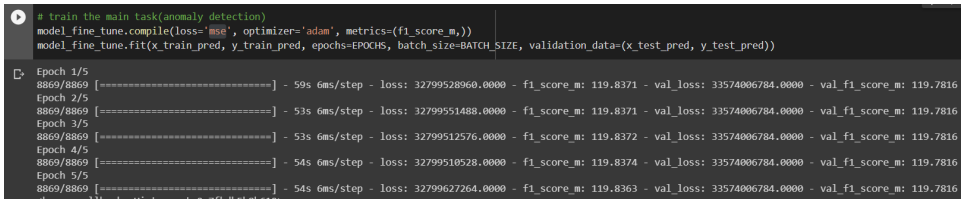
- ابتدا لایه انتهایی این مدل را حذف می‌کنیم.

```
[19] # delete last layer of model
     model_fine_tune = keras.Model(inputs=model.input, outputs=model.layers[-2].output)
```

- حال دو لایه Dense با ۱۰ نورون و ۱ نورون اضافه کرده و عملیات fine-tunning را انجام می‌دهیم.

```
[20] # freeze all remaining layers except the last one
     for layer in model_fine_tune.layers[:-1]:
       layer.trainable = False

     # add 2 dence layer to the model
     layer1 = layers.Dense(10)(model_fine_tune.output)
     layer2 = layers.Dense(1, activation='sigmoid')(layer1)
     model_fine_tune = keras.Model(inputs=model_fine_tune.input, outputs=layer2)
```

همانطور که در شکل زیر نیز مشخص است، مقدار f1_score به طور شگفت‌انگیزی افزایش یافت که این نشان می‌دهد عملکرد مدل بهتر شده است.



سوال ۷:

داده‌های ارائه شده در این تمرین بسیار نامتعادلاند. (تعداد داده‌ها با برچسب ۰ بسیار بیشتر از داده‌های با برچسب ۱ است) یک راهکار برای حل این مشکل ارائه کنید و با اعمال آن عملکرد مدل را بهبود دهید. (۲۰ نمره)

پاسخ ۷:

برای حل این مشکل، می‌توانیم از روش upsampling و downsampling استفاده کنیم.
downsampling روشی است که در آن نمونه‌های منفی را به صورت تصادفی برای متعادل کردن مجموعه داده‌ها انتخاب می‌کنیم. upsampling نیز نمونه‌گیری تصادفی از نمونه‌های مثبت با جایگزینی برای ایجاد مجموعه داده‌های متعادل می‌باشد. بنابراین طبق تعاریف گفته شده در بالا، x_train، y_train، x_test، y_test جدید، به شکل زیر در می‌آیند.

```
[28] mjr = data[data["label"] == 0]
     mnr = data[data["label"] == 1]

     downsampled = resample(mjr, replace=True, random_state=42, n_samples=len(mnr))
     upsampled = pd.concat([downsampled, mnr])

     x_upsample = upsampled["value"]
     x_upsample = x_upsample.values.reshape(-1, 1, 1)
     y_upsample = upsampled.iloc[:, -1]
     x_train_up, x_test_up, y_train_up, y_test_up = train_test_split(x_upsample, y_upsample, random_state=0)
```

پس از آموزش این مدل با LSTM، مشاهده می‌شود که دقت بالا رفته و مشکل loss که در بخش‌های قبل دیده می‌شد، بسیار کاهش یافته است و عملکرد مدل بسیار بهتر شده است.

```
lstm_model = LSTM_RNN(128, 64, 1, (1, 1), ['relu', 'sigmoid'])
history = lstm_model.fit(x_train_up, y_train_up, epochs=EPOCHS, batch_size=BATCH_SIZE, validation_data=(x_test_up, y_test_up))

Epoch 1/5
338/338 [==============================] - 4s 7ms/step - loss: 0.5389 - f1_score_m: 0.7462 - val_loss: 0.4816 - val_f1_score_m: 0.7964
Epoch 2/5
338/338 [==============================] - 2s 5ms/step - loss: 0.4631 - f1_score_m: 0.8098 - val_loss: 0.4520 - val_f1_score_m: 0.8172
Epoch 3/5
338/338 [==============================] - 2s 5ms/step - loss: 0.4454 - f1_score_m: 0.8178 - val_loss: 0.4444 - val_f1_score_m: 0.8173
Epoch 4/5
338/338 [==============================] - 2s 5ms/step - loss: 0.4418 - f1_score_m: 0.8183 - val_loss: 0.4446 - val_f1_score_m: 0.8180
Epoch 5/5
338/338 [==============================] - 2s 5ms/step - loss: 0.4377 - f1_score_m: 0.8208 - val_loss: 0.4396 - val_f1_score_m: 0.8221
```

سوال ۸:

با استفاده از یک روش آماری سعی کنید این مسئله را حل کنید. آیا با استفاده از روش‌های آماری نتایج بهتری حاصل می‌شود؟ (۲۰ نمره)

پاسخ ۸:

Z-score میزان فاصله یک نقطه داده از میانگین را به عنوان مضربی از انحراف معیار اندازه گیری می‌کند. مقادیر مطلق بزرگ امتیاز Z نشان دهنده یک ناهنجاری است.

$$z - score = \frac{x - \mu}{\sigma}$$

```
data = pd.read_csv("0.csv")
data['z-score'] = stats.zscore(data['value'])
display(data)
```

	timestamp	value	label	z-score
0	1493568000	1.901639	0	-0.074118
1	1493568060	1.786885	0	-0.249654
2	1493568120	2.000000	0	0.076341
3	1493568180	1.885246	0	-0.099195
4	1493568240	1.819672	0	-0.199501
...	...	...	...	...
128557	1501475400	2.684211	0	1.122956
128558	1501475460	2.526316	0	0.881429
128559	1501475520	2.614035	0	1.015611
128560	1501475580	2.736842	0	1.203465
128561	1501475640	2.491228	0	0.827757