دانشکده مهندسی کامپیوتر

شایان موسوی نیا

محمدحسین کریمیان

پروژه نهایی درس یادگیری عمیق

چت بات FAQ فارسی

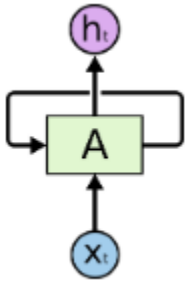
1. خلاصه پروژه

ما در این پروژه یک سیستم FAQ (پرسش پاسخ هوشمند) را پیاده سازی کردیم که در آن مانند سایر سیستم های FAQ، باید بتوان به پرسش های گوناگون کاربران جواب داد. فهرست سؤالات متداول (FAQ) اغلب در مقالات، وب‌سایت‌ها، فهرست‌های ایمیل و انجمن‌های آنلاین استفاده می‌شود که در آن سؤالات رایج معمولاً تکرار می‌شوند، برای مثال از طریق پست‌ها یا سؤالات کاربران جدید مرتبط با شکاف‌های دانش رایج.
هدف سؤالات متداول به طور کلی ارائه اطلاعات در مورد سؤالات یا نگرانی های متداول است. با این حال، قالب ابزار مفیدی برای سازماندهی اطلاعات است، و متنی که از سؤالات و پاسخ های آنها تشکیل شده است، صرف نظر از اینکه سؤالات واقعاً مکرر پرسیده می شوند یا خیر، ممکن است سؤال متداول نامیده شود.
داده های ما در این پروژه داده های شرکت همراه اول است. در این پروژه باید با تعلیم یک سیستم، آن سیستم قادر باشد تا پاسخ مناسبی به مشتریان دهند. یک سری سوال ورودی و خروجی وجود دارد که جواب هر سوال مشخص است. اگر سوال جدید پرسیده شد باید تشخیص داده شود که به کدام سوال نزدیک تر است و جواب همان سوال به عنوان جواب سوال جدید نیز در نظر گرفته شود. برای این عملیات، ما با پیاده سازی یک LSTM، داده ها را بررسی میکنیم و برای هر سوال احتمالی، جوابی مناسب پیدا میکنیم.

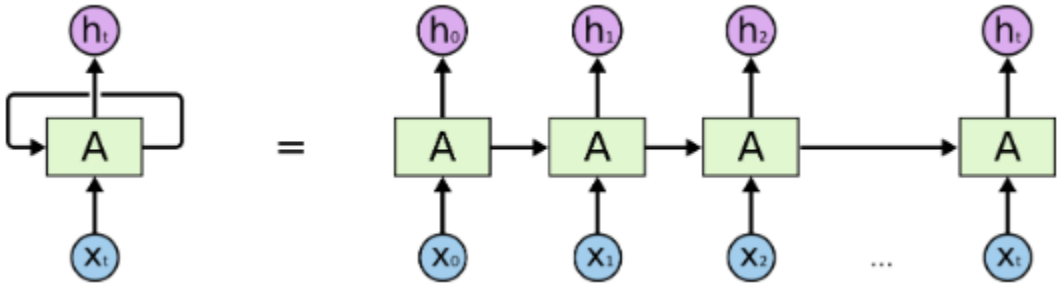
2. RNN و LSTM

در ابتدا باید با مفهوم RNN و LSTM آشنا شویم.
انسان ها هر ثانیه فکر خود را از صفر شروع نمی کنند. با خواندن این مقاله، هر کلمه را بر اساس درک خود از کلمات قبلی درک می کنید. همه چیز را دور نمی اندازید و دوباره از صفر شروع به فکر کردن می کنید. افکار شما ماندگار هستند
شبکه‌های عصبی سنتی نمی‌توانند این کار را انجام دهند، و به نظر یک نقص بزرگ است. به عنوان مثال، تصور کنید که می خواهید طبقه بندی کنید که چه نوع رویدادی در هر نقطه از یک فیلم اتفاق می افتد. مشخص نیست که چگونه یک شبکه عصبی سنتی می تواند از استدلال خود در مورد رویدادهای قبلی در فیلم برای اطلاع رسانی به رویدادهای بعدی استفاده کند.

شبکه های عصبی مکرر این مشکل را حل می کنند. آن‌ها شبکه‌هایی هستند که حلقه‌هایی در آن‌ها وجود دارد و به اطلاعات اجازه می‌دهند همچنان باقی بمانند.

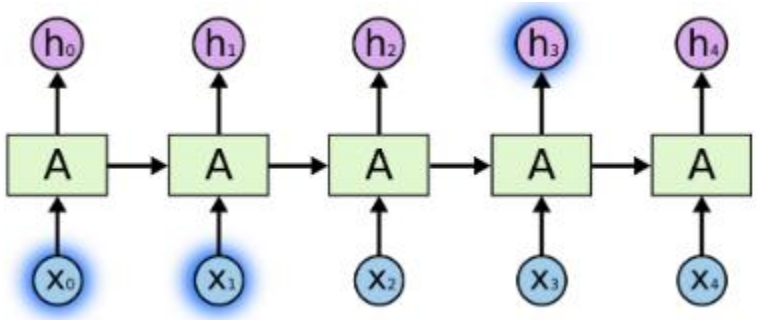


در نمودار بالا، تکه ای از شبکه عصبی، به برخی از ورودی xt نگاه می کند و مقدار ht را خروجی می دهد. یک حلقه اجازه می دهد تا اطلاعات از یک مرحله از شبکه به مرحله بعدی منتقل شود.
این حلقه‌ها باعث می‌شوند شبکه‌های عصبی مکرر نوعی مرموز به نظر برسند. با این حال، اگر کمی بیشتر فکر کنید، معلوم می‌شود که آنها تفاوت چندانی با یک شبکه عصبی معمولی ندارند. یک شبکه عصبی مکرر را می توان به عنوان چندین نسخه از یک شبکه در نظر گرفت که هر یک پیامی را به جانشینی ارسال می کند. در نظر بگیرید که اگر حلقه را باز کنیم چه اتفاقی می افتد:

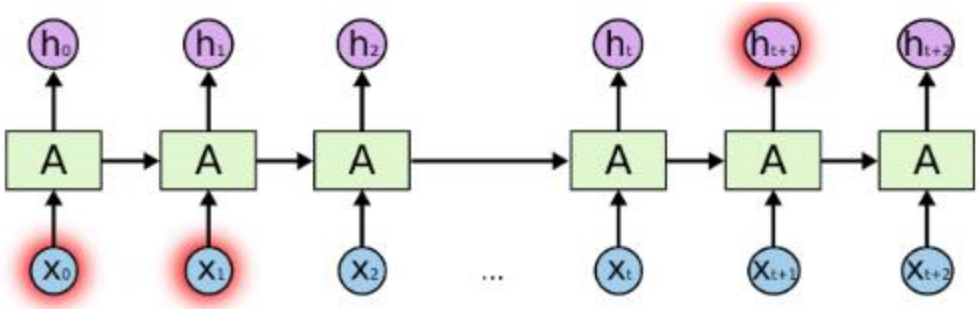


این ماهیت زنجیره‌وار نشان می‌دهد که شبکه‌های عصبی مکرر ارتباط نزدیکی با دنباله‌ها و فهرست‌ها دارند. آنها معماری طبیعی شبکه عصبی برای استفاده برای چنین داده هایی هستند.
و مطمئناً مورد استفاده قرار می گیرند! در چند سال گذشته، موفقیت‌های باورنکردنی در استفاده از RNN برای مشکلات مختلف حاصل شده است: تشخیص گفتار، مدل‌سازی زبان، ترجمه، شرح تصاویر... این فهرست ادامه دارد. من بحث در مورد شاهکارهای شگفت انگیزی که می توان با RNN ها به دست آورد را به پست وبلاگ عالی آندری کارپاتی، اثربخشی نامعقول شبکه های عصبی مکرر، واگذار می کنم. اما آنها واقعاً شگفت انگیز هستند.

ضروری برای این موفقیت ها استفاده از "LSTMs" است، یک نوع بسیار خاص از شبکه عصبی بازگشتی که برای بسیاری از وظایف، بسیار بهتر از نسخه استاندارد کار می کند. تقریباً تمام نتایج هیجان انگیز مبتنی بر شبکه های عصبی مکرر با آنها به دست می آید. این LSTM ها هستند که در این پروژه استفاده خواهد کرد.
یکی از جذابیت های RNN ها این ایده است که آنها ممکن است بتوانند اطلاعات قبلی را به وظیفه فعلی متصل کنند، مانند استفاده از فریم های ویدیویی قبلی ممکن است درک فریم فعلی را نشان دهد. اگر RNN ها بتوانند این کار را انجام دهند، بسیار مفید خواهند بود. اما آیا آنها می توانند؟ بستگی دارد.
گاهی اوقات، ما فقط باید به اطلاعات اخیر نگاه کنیم تا کار فعلی را انجام دهیم. به عنوان مثال، یک مدل زبان را در نظر بگیرید که سعی می کند کلمه بعدی را بر اساس کلمات قبلی پیش بینی کند. اگر می‌خواهیم آخرین کلمه را در «ابرها در آسمان هستند» پیش‌بینی کنیم، به زمینه بیشتری نیاز نداریم - کاملاً واضح است که کلمه بعدی آسمان است. در چنین مواردی، جایی که شکاف بین اطلاعات مربوط به مکانی که به آن نیاز است کم است، RNN ها می توانند یاد بگیرند که از اطلاعات گذشته استفاده کنند.

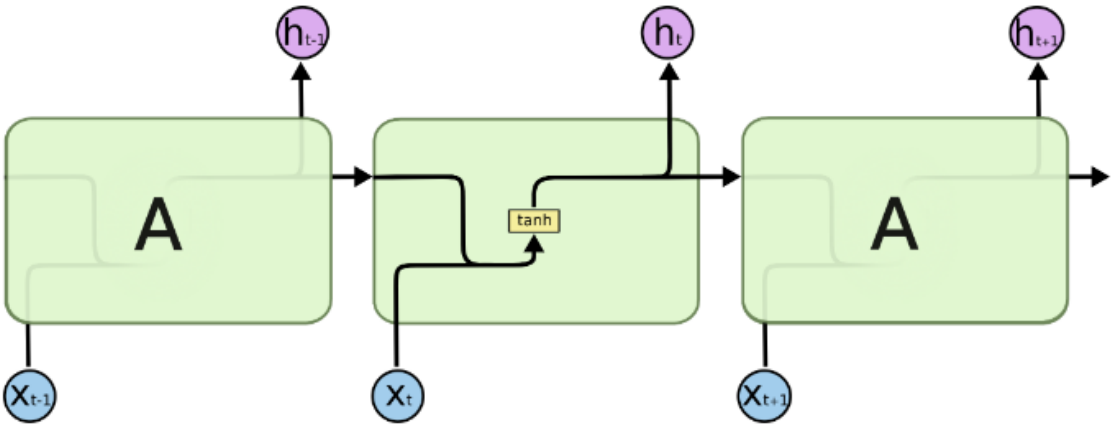


اما مواردی نیز وجود دارد که به زمینه بیشتری نیاز داریم. سعی کنید آخرین کلمه را در متن پیش‌بینی کنید: «من در فرانسه بزرگ شدم... من به زبان فرانسوی روان صحبت می‌کنم». اطلاعات اخیر حاکی از آن است که کلمه بعدی احتمالاً نام یک زبان است، اما اگر بخواهیم کدام زبان را محدود کنیم، به بافت فرانسه، از عقب تر، نیاز داریم. کاملاً امکان پذیر است که شکاف بین اطلاعات مربوطه و نقطه ای که در آن لازم است بسیار زیاد شود.
متأسفانه، با افزایش این شکاف، RNN ها قادر به یادگیری اتصال اطلاعات نیستند.

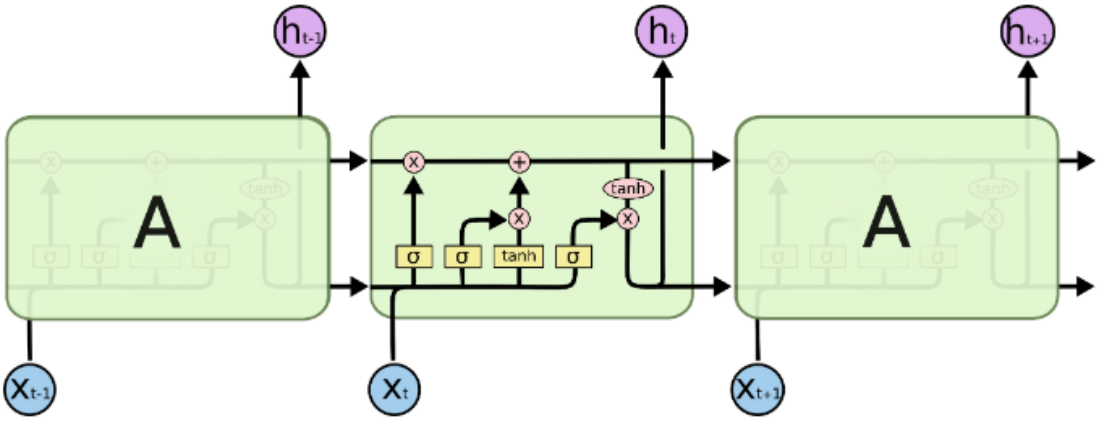


در تئوری، RNN ها کاملاً قادر به مدیریت چنین "وابستگی های طولانی مدت" هستند. یک انسان می تواند با دقت پارامترهایی را برای آنها انتخاب کنید تا مشکلات اسباب بازی این شکل را حل کند. متأسفانه، در عمل، به نظر می رسد RNN ها قادر به یادگیری آنها نیستند. این مشکل توسط Hochreiter (1991) [آلمانی] و Bengio و همکاران به طور عمیق مورد بررسی قرار گرفت. (1994)، که دلایل بسیار اساسی برای دشوار بودن آن پیدا کرد.خوشبختانه، LSTM ها این مشکل را ندارند!

شبکه‌های حافظه کوتاه‌مدت بلندمدت (که معمولاً به آنها «LSTM» می‌گویند) نوع خاصی از RNN هستند که قادر به یادگیری وابستگی‌های بلندمدت هستند. آنها توسط Hochreiter & Schmidhuber (1997) معرفی شدند، و توسط بسیاری از مردم در کارهای بعدی اصلاح و محبوب شدند.
LSTM ها به صراحت برای جلوگیری از مشکل وابستگی طولانی مدت طراحی شده اند. به خاطر سپردن اطلاعات برای مدت طولانی عملاً رفتار پیش فرض آنهاست، نه چیزی که برای یادگیری آن تلاش می کنند!همه شبکه های عصبی بازگشتی به شکل زنجیره ای از ماژول های تکرار شونده شبکه عصبی هستند. در RNN های استاندارد، این ماژول تکرار شونده ساختار بسیار ساده ای مانند یک لایه tanh دارد.



LSTMها همچنین دارای این ساختار زنجیره مانند هستند، اما ماژول تکرارشونده ساختار متفاوتی دارد. به جای داشتن یک لایه شبکه عصبی واحد، چهار لایه وجود دارد که به روشی بسیار خاص با هم تعامل دارند.



کلید LSTM ها وضعیت سلولی است، خط افقی که از بالای نمودار عبور می کند.
حالت سلولی به نوعی شبیه تسمه نقاله است. این به طور مستقیم در کل زنجیره اجرا می شود، تنها با برخی فعل و انفعالات خطی جزئی. بسیار آسان است که اطلاعات بدون تغییر در امتداد آن جریان یابد.

LSTM توانایی حذف یا اضافه کردن اطلاعات به وضعیت سلولی را دارد که به دقت توسط ساختارهایی به نام گیت تنظیم می شود.
گیت ها راهی هستند که به صورت اختیاری اطلاعات را از خود عبور می دهند. آنها از یک لایه شبکه عصبی سیگموئید و یک عملیات ضرب نقطه ای تشکیل شده اند.

لایه سیگموئید اعدادی بین صفر و یک را خروجی می‌دهد و توضیح می‌دهد که چه مقدار از هر جزء باید عبور کند. مقدار صفر به معنای "نگذارید هیچ چیز عبور کند"، در حالی که مقدار یک به معنای "اجازه دهید همه چیز از بین برود!"
یک LSTM دارای سه مورد از این دروازه ها برای محافظت و کنترل وضعیت سلول است.

3. توضیحات کد

در ابتدا نیاز هست که داده ها را از فایل اکسل خارج کنیم که به کمک کتابخانه panda به راحتی این کار قابل انجام است.

```
csvFile = 'HW/Extension1.xlsx'
df = pd.read_excel(csvFile, engine='openpyxl',)
df.head()
```

	question	answer
0	چگونه می توان از موفقیت آمیز بودن خرید اینترنت...	از انجام خرید سیم کارت از طریق پس...
1	چگونه باید از موفقیت آمیز بودن خرید اینترنتی ه...	از انجام خرید سیم کارت از طریق پس...
2	از چه راهی باید وضعیت خرید اینترنتی را مشاهده ...	از انجام خرید سیم کارت از طریق پس...
3	روشی که می‌توان مطمئن شد خرید اینترنتی به درست...	از انجام خرید سیم کارت از طریق پس...
4	برای این‌که مطمئن بشیم که خرید اینترنتی به درس...	از انجام خرید سیم کارت از طریق پس...

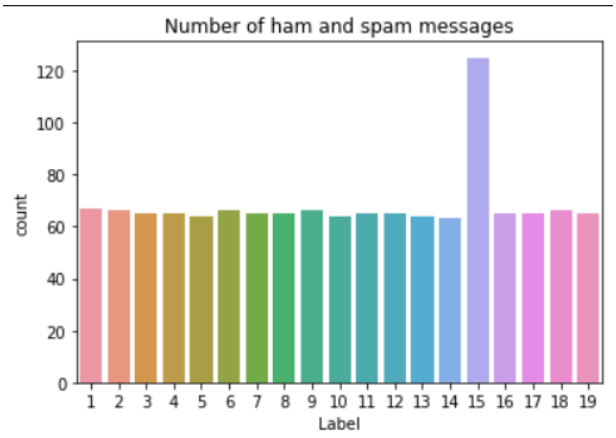
بعد از آن نیاز است که برای راحتی اموزش همه answer ها مربوطه را به یک عدد اختصاص دهیم که در بخش زیر این کار انجام شده است و در اخر سر یک دیکشنری ار label داریم که کلید آن عدد یا همان برچسب جدید و مقدار آن answer اولیه ما است.

```
labels = {}
counter = 0
for count in range(len(df.answer)):
    if df.answer[count] == df.answer[count]:
        if df.answer[count] not in labels.values():
            labels[counter] = df.answer[count]
            counter +=1
        df.answer[count] = counter
```

بعضی از خانه های فایل اکسل دارای مقدار نیستند و برای اینکه در روند اموزش مشکلی پیش نیاد، این مقادیر NaN را از داده ها حذف میکنیم.

```
df = df.dropna()
```

نمودار زیر نشانگر تعداد label ها موجود در مجموعه داده ما است.



در بخش بعدی، نیاز است که داده ها را به 2 بخش train و test بخش بندی کنیم و سپس label های این 2 بخش را از مقدار int به hot map تبدیل میکنیم که این کار با کمک to categorical انجام میشود.

```
X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size=0.2)
Y_ = to_categorical(Y, num_classes=len(labels))
Y__ = to_categorical(Y_test, num_classes=len(labels))
```

برای اینکه داده هایی اموزش را به مدل بدهیم باید از فرم string بودن خارج کنیم و برای اینکار نیاز به یک Tokenizer داریم که این روند را انجام دهد. در ادامه با fit کردن تمام داده ها برای روی این Tokenizer و درست کردن sequences matrix های مناسب، مجموعه داده را برای اموزش درست کردیم.

```
max_words = 500
max_len = 50
tok = Tokenizer(num_words=max_words)
tok.fit_on_texts(X)
sequences = tok.texts_to_sequences(X)
sequences_matrix_train = sequence.pad_sequences(sequences,maxlen=max_len, dtype='float')
```

سپس باید با تعریف توابع metric مناسب، recall، precision و f1-score مناسب بنویسیم.
در تشخیص الگو، بازیابی اطلاعات، تشخیص و طبقه‌بندی شی (یادگیری ماشین)، دقت و یادآوری معیارهای عملکردی هستند که برای داده‌های بازیابی شده از یک مجموعه، پیکره یا فضای نمونه اعمال می‌شوند.
دقت (که ارزش پیش‌بینی مثبت نیز نامیده می‌شود) کسری از نمونه‌های مربوطه در بین نمونه‌های بازیابی شده است، در حالی که یادآوری (همچنین به عنوان حساسیت شناخته می‌شود) کسری از نمونه‌های مربوطه است که بازیابی شده‌اند. بنابراین، هم دقت و هم یادآوری بر اساس ارتباط است.

امتیاز F1 یک معیار ارزیابی یادگیری ماشینی است که دقت یک مدل را اندازه گیری می کند. این نمرات دقت و یادآوری یک مدل را ترکیب می کند. متریک دقت محاسبه می کند که یک مدل چند بار پیش بینی درستی را در کل مجموعه داده انجام داده است.

```
[ ]  def recall_m(y_true, y_pred):
        true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
        possible_positives = K.sum(K.round(K.clip(y_true, 0, 1)))
        recall = true_positives / (possible_positives +
        K.epsilon())
        return recall

    def precision_m(y_true, y_pred):
        true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
        predicted_positives = K.sum(K.round(K.clip(y_pred, 0, 1)))
        precision = true_positives / (predicted_positives + K.epsilon())
        return precision

    def f1_score_m(y_true, y_pred):
        precision = precision_m(y_true, y_pred)
        recall = recall_m(y_true, y_pred)
        return 2*((precision*recall)/(precision+recall+K.epsilon()))
```

سپس یک مدل RNN سه تایی(Simple. LSTM و GRU) را پیاده سازی میکنیم. برای این کار از Keras استفاده میکنیم و اکتیویشن فانکشن لایه میانی را برابر ReLu میگذاریم و لایه خروجی هم sigmoid.
همچنین برای کامپایل کردن از adam به عنوان آپتیمایزر استفاده می کنیم و تابع loss را برابر MSE میگیریم.

```
[ ]  # Simple
    from tensorflow.keras import regularizers

    model_RNN = tf.keras.Sequential([
                               tf.keras.layers.SimpleRNN(units=sequence,  input_shape=(1,sequence)),
                               tf.keras.layers.Dense(sequence, activation='relu'),
                               tf.keras.layers.Dense(1, activation='sigmoid')
                              ])

    model_RNN.compile(optimizer='adam', loss='mean_squared_error', metrics=['accuracy',recall_m,precision_m,f1_score_m])
    model_RNN.summary()

    history_RNN = model_RNN.fit(A, B, epochs=100, verbose=1,validation_split=0.2,batch_size=64,
                                callbacks=keras.callbacks.EarlyStopping(monitor='val_f1_score_m',
                                                                        patience=5,
                                                                        restore_best_weights=True))
```

در عکس زیر، خروجی قطعه کد بالا را مشاهده می کنید.

```
Model: "sequential_19"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 simple_rnn_9 (SimpleRNN)    (None, 200)               80200

 dense_37 (Dense)            (None, 200)               40200

 dense_38 (Dense)            (None, 1)                 201

=================================================================
Total params: 120,601
Trainable params: 120,601
Non-trainable params: 0
_________________________________________________________________
Epoch 1/100
18916/18916 [==============================] - 111s 6ms/step - loss: 0.0843 - accuracy: 0.8989 - recall_m: 0.1662 - precision_m: 0.5186 - f1_score_m: 0.2367 - val_loss: 0.0980 - val_accuracy: 0.8847 - val_r
Epoch 2/100
18916/18916 [==============================] - 121s 6ms/step - loss: 0.0828 - accuracy: 0.9002 - recall_m: 0.1856 - precision_m: 0.5476 - f1_score_m: 0.2609 - val_loss: 0.0975 - val_accuracy: 0.8850 - val_r
Epoch 3/100
18916/18916 [==============================] - 109s 6ms/step - loss: 0.0820 - accuracy: 0.9009 - recall_m: 0.1987 - precision_m: 0.5610 - f1_score_m: 0.2764 - val_loss: 0.0981 - val_accuracy: 0.8828 - val_r
Epoch 4/100
18916/18916 [==============================] - 107s 6ms/step - loss: 0.0809 - accuracy: 0.9022 - recall_m: 0.2138 - precision_m: 0.5771 - f1_score_m: 0.2936 - val_loss: 0.0974 - val_accuracy: 0.8853 - val_r
Epoch 5/100
18916/18916 [==============================] - 109s 6ms/step - loss: 0.0802 - accuracy: 0.9029 - recall_m: 0.2220 - precision_m: 0.5894 - f1_score_m: 0.3038 - val_loss: 0.0979 - val_accuracy: 0.8843 - val_r
Epoch 6/100
18916/18916 [==============================] - 119s 6ms/step - loss: 0.0793 - accuracy: 0.9034 - recall_m: 0.2309 - precision_m: 0.5960 - f1_score_m: 0.3133 - val_loss: 0.0961 - val_accuracy: 0.8844 - val_r
Epoch 7/100
18916/18916 [==============================] - 109s 6ms/step - loss: 0.0789 - accuracy: 0.9039 - recall_m: 0.2344 - precision_m: 0.6008 - f1_score_m: 0.3179 - val_loss: 0.0974 - val_accuracy: 0.8847 - val_r
Epoch 8/100
18916/18916 [==============================] - 115s 6ms/step - loss: 0.0784 - accuracy: 0.9042 - recall_m: 0.2410 - precision_m: 0.6041 - f1_score_m: 0.3243 - val_loss: 0.0969 - val_accuracy: 0.8836 - val_r
```

در ادامه، مدل LSTM را با activation function هایی مانند قبل، درست می کنیم و برای history آن، از F1-score استفاده می کنیم.

```python
# LSTM
model_LSTM = tf.keras.Sequential([
                              tf.keras.layers.LSTM(units=sequence,  input_shape=(1,sequence)),
                              tf.keras.layers.Dropout(.1),
                              tf.keras.layers.Dense(sequence, activation='relu'),
                              tf.keras.layers.Dense(1, activation='sigmoid')
                             ])

model_LSTM.compile(optimizer='adam', loss='mean_squared_error', metrics=['accuracy',recall_m,precision_m,f1_score_m])
model_LSTM.summary()

history_LSTM = model_LSTM.fit(A, B, epochs=100, verbose=1,validation_split=0.2,batch_size=64,callbacks=keras.callbacks.EarlyStopping(monitor='val_f1_score_m',
                                          patience=5,
                                          restore_best_weights=True))
```

خروجی در شکل زیر نشان داده شده است.

```
Model: "sequential_21"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_11 (LSTM)              (None, 200)               320800

 dropout_3 (Dropout)         (None, 200)               0

 dense_41 (Dense)            (None, 200)               40200

 dense_42 (Dense)            (None, 1)                 201

=================================================================
Total params: 361,201
Trainable params: 361,201
Non-trainable params: 0
_________________________________________________________________
Epoch 1/100
18916/18916 [==============================] - 189s 10ms/step - loss: 0.0833 - accuracy: 0.8999 - recall_m: 0.1819 - precision_m: 0.5424 - f1_score_m: 0.2565 - val_loss: 0.0979 -
Epoch 2/100
18916/18916 [==============================] - 199s 11ms/step - loss: 0.0817 - accuracy: 0.9018 - recall_m: 0.2042 - precision_m: 0.5734 - f1_score_m: 0.2840 - val_loss: 0.0986 -
Epoch 3/100
18916/18916 [==============================] - 214s 11ms/step - loss: 0.0805 - accuracy: 0.9027 - recall_m: 0.2188 - precision_m: 0.5904 - f1_score_m: 0.3006 - val_loss: 0.0995 -
Epoch 4/100
18916/18916 [==============================] - 223s 12ms/step - loss: 0.0799 - accuracy: 0.9033 - recall_m: 0.2264 - precision_m: 0.5939 - f1_score_m: 0.3091 - val_loss: 0.0987 -
Epoch 5/100
18916/18916 [==============================] - 216s 11ms/step - loss: 0.0796 - accuracy: 0.9037 - recall_m: 0.2299 - precision_m: 0.6014 - f1_score_m: 0.3134 - val_loss: 0.0994 -
Epoch 6/100
18916/18916 [==============================] - 222s 12ms/step - loss: 0.0785 - accuracy: 0.9046 - recall_m: 0.2410 - precision_m: 0.6091 - f1_score_m: 0.3258 - val_loss: 0.0988 -
```

همچنین GRU مانند LSTM و RNN با قطعه کد زیر زده می شود.

```
# GRU
model_GRU = tf.keras.Sequential([
                            tf.keras.layers.GRU(units=sequence,  input_shape=(1,sequence)),
                            tf.keras.layers.Dropout(.1),
                            tf.keras.layers.Dense(sequence, activation='relu'),
                            tf.keras.layers.Dense(1, activation='sigmoid')
                           ])

model_GRU.compile(optimizer='adam', loss='mean_squared_error', metrics=['accuracy',recall_m,precision_m,f1_score_m])
model_GRU.summary()

history_GRU = model_GRU.fit(A, B, epochs=100, verbose=1,validation_split=0.2,batch_size=64,callbacks=keras.callbacks.EarlyStopping(monitor='val_f1_score_m',
                                        patience=5,
                                        restore_best_weights=True))
```

خروجی را در شکل زیر مشاهده می کنید.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 gru (GRU)                   (None, 200)               241200

 dropout (Dropout)           (None, 200)               0

 dense (Dense)               (None, 200)               40200

 dense_1 (Dense)             (None, 1)                 201

=================================================================
Total params: 281,601
Trainable params: 281,601
Non-trainable params: 0
_________________________________________________________________
Epoch 1/100
18916/18916 [==============================] - 400s 21ms/step - loss: 0.0890 - accuracy: 0.8927 - recall_m: 0.1979 - precision_m: 0.5649 - f1_score_m: 0.2765 - val_loss: 0.0737 -
Epoch 2/100
18916/18916 [==============================] - 353s 19ms/step - loss: 0.0866 - accuracy: 0.8953 - recall_m: 0.2322 - precision_m: 0.5996 - f1_score_m: 0.3164 - val_loss: 0.0735 -
Epoch 3/100
18916/18916 [==============================] - 349s 18ms/step - loss: 0.0855 - accuracy: 0.8965 - recall_m: 0.2425 - precision_m: 0.6117 - f1_score_m: 0.3282 - val_loss: 0.0735 -
Epoch 4/100
18916/18916 [==============================] - 246s 13ms/step - loss: 0.0845 - accuracy: 0.8976 - recall_m: 0.2551 - precision_m: 0.6239 - f1_score_m: 0.3423 - val_loss: 0.0737 -
Epoch 5/100
18916/18916 [==============================] - 319s 17ms/step - loss: 0.0839 - accuracy: 0.8981 - recall_m: 0.2607 - precision_m: 0.6285 - f1_score_m: 0.3483 - val_loss: 0.0737 -
Epoch 6/100
18916/18916 [==============================] - 330s 17ms/step - loss: 0.0832 - accuracy: 0.8989 - recall_m: 0.2678 - precision_m: 0.6357 - f1_score_m: 0.3570 - val_loss: 0.0733 -
Epoch 7/100
18916/18916 [==============================] - 367s 19ms/step - loss: 0.0830 - accuracy: 0.8992 - recall_m: 0.2686 - precision_m: 0.6355 - f1_score_m: 0.3580 - val_loss: 0.0731 -
Epoch 8/100
18916/18916 [==============================] - 228s 12ms/step - loss: 0.0824 - accuracy: 0.8997 - recall_m: 0.2768 - precision_m: 0.6419 - f1_score_m: 0.3664 - val_loss: 0.0728 -
Epoch 9/100
18916/18916 [==============================] - 300s 16ms/step - loss: 0.0820 - accuracy: 0.9002 - recall_m: 0.2809 - precision_m: 0.6465 - f1_score_m: 0.3707 - val_loss: 0.0740 -
Epoch 10/100
```

سپس برای این که نتیجه بهتری بگیریم، داده ها را با استفاده از تابع preprocess که در بالا توضیح دادیم، process میکنیم.

```
def preprocessing_X_new(df):
  arr_2 = np.array([df["value"]])
  arr_2 = preprocessing.normalize(arr_2, norm='l2')

  x, y = [], []
  for i in range(len(arr_2[0])-sequence):
    x.append(arr_2[0][i:i+sequence])

  return np.array(x)

A_new = preprocessing_X_new(X)
```

بعد از این کار، مدل سه گانه خود را دوباره آموزش می دهیم.
ابتدا برای RNN:

```
# Train 3 models again
# Simple
from tensorflow.keras import regularizers

model_RNN = tf.keras.Sequential([
                            tf.keras.layers.SimpleRNN(units=sequence,  input_shape=(1,sequence)),
                            tf.keras.layers.Dense(sequence, activation='relu'),
                            tf.keras.layers.Dense(1, activation='sigmoid')
                           ])

model_RNN.compile(optimizer='adam', loss='mean_squared_error', metrics=['accuracy',recall_m,precision_m,f1_score_m])
model_RNN.summary()

history_RNN_new = model_RNN.fit(A, B, epochs=100, verbose=1,validation_split=0.2,batch_size=64,
                           callbacks=keras.callbacks.EarlyStopping(monitor='val_f1_score_m',
                                       patience=5,
                                       restore_best_weights=True))
```

خروجی:

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 simple_rnn (SimpleRNN)      (None, 200)               80200

 dense (Dense)               (None, 200)               40200

 dense_1 (Dense)             (None, 1)                 201

=================================================================
Total params: 120,601
Trainable params: 120,601
Non-trainable params: 0
_________________________________________________________________
Epoch 1/100
18916/18916 [==============================] - 249s 13ms/step - loss: 0.0897 - accuracy: 0.8917 - recall_m: 0.1845 - precision_m: 0.5482 - f1_score_m: 0.2601 - val_loss: 0.0744
Epoch 2/100
18916/18916 [==============================] - 220s 12ms/step - loss: 0.0881 - accuracy: 0.8934 - recall_m: 0.2065 - precision_m: 0.5770 - f1_score_m: 0.2871 - val_loss: 0.0746
Epoch 3/100
18916/18916 [==============================] - 156s 8ms/step - loss: 0.0868 - accuracy: 0.8946 - recall_m: 0.2211 - precision_m: 0.5933 - f1_score_m: 0.3038 - val_loss: 0.0735 -
Epoch 4/100
18916/18916 [==============================] - 158s 8ms/step - loss: 0.0859 - accuracy: 0.8956 - recall_m: 0.2312 - precision_m: 0.6043 - f1_score_m: 0.3159 - val_loss: 0.0741 -
Epoch 5/100
18916/18916 [==============================] - 141s 7ms/step - loss: 0.0848 - accuracy: 0.8967 - recall_m: 0.2447 - precision_m: 0.6134 - f1_score_m: 0.3304 - val_loss: 0.0736 -
Epoch 6/100
18916/18916 [==============================] - 129s 7ms/step - loss: 0.0836 - accuracy: 0.8976 - recall_m: 0.2578 - precision_m: 0.6233 - f1_score_m: 0.3445 - val_loss: 0.0742 -
```

بعد نوبت مدل LSTM است:

```
# LSTM
model_LSTM = tf.keras.Sequential([
                            tf.keras.layers.LSTM(units=sequence,  input_shape=(1,sequence)),
                            tf.keras.layers.Dense(sequence, activation='relu'),
                            tf.keras.layers.Dense(1, activation='sigmoid')
                           ])

model_LSTM.compile(optimizer='adam', loss='mean_squared_error', metrics=['accuracy',recall_m,precision_m,f1_score_m])
model_LSTM.summary()

history_LSTM_new = model_LSTM.fit(A, B, epochs=100, verbose=1,validation_split=0.2,batch_size=64,callbacks=keras.callbacks.EarlyStopping(monitor='val_f1_score_m',
                                        patience=5,
                                        restore_best_weights=True))
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 200)               320800

 dense_2 (Dense)             (None, 200)               40200

 dense_3 (Dense)             (None, 1)                 201

=================================================================
Total params: 361,201
Trainable params: 361,201
Non-trainable params: 0
_________________________________________________________________
Epoch 1/100
18916/18916 [==============================] - 319s 17ms/step - loss: 0.0882 - accuracy: 0.8933 - recall_m: 0.2088 - precision_m: 0.5766 - f1_score_m: 0.2889 -
Epoch 2/100
18916/18916 [==============================] - 282s 15ms/step - loss: 0.0859 - accuracy: 0.8957 - recall_m: 0.2386 - precision_m: 0.6041 - f1_score_m: 0.3235 -
Epoch 3/100
18916/18916 [==============================] - 294s 16ms/step - loss: 0.0845 - accuracy: 0.8975 - recall_m: 0.2571 - precision_m: 0.6184 - f1_score_m: 0.3435 -
Epoch 4/100
18916/18916 [==============================] - 285s 15ms/step - loss: 0.0833 - accuracy: 0.8985 - recall_m: 0.2690 - precision_m: 0.6289 - f1_score_m: 0.3570 -
Epoch 5/100
18916/18916 [==============================] - 261s 14ms/step - loss: 0.0828 - accuracy: 0.8995 - recall_m: 0.2737 - precision_m: 0.6406 - f1_score_m: 0.3629 -
Epoch 6/100
18916/18916 [==============================] - 272s 14ms/step - loss: 0.0818 - accuracy: 0.9001 - recall_m: 0.2808 - precision_m: 0.6433 - f1_score_m: 0.3703 -
```

سپس GRU:

```
# GRU
model_GRU = tf.keras.Sequential([
                            tf.keras.layers.GRU(units=sequence,  input_shape=(1,sequence)),
                            tf.keras.layers.Dense(sequence, activation='relu'),
                            tf.keras.layers.Dense(1, activation='sigmoid')
                           ])

model_GRU.compile(optimizer='adam', loss='mean_squared_error', metrics=['accuracy',recall_m,precision_m,f1_score_m])
model_GRU.summary()

history_GRU_new = model_GRU.fit(A, B, epochs=100, verbose=1,validation_split=0.2,batch_size=64,callbacks=keras.callbacks.EarlyStopping(monitor='val_f1_score_m',
                                        patience=5,
                                        restore_best_weights=True))
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 gru_2 (GRU)                 (None, 200)               241200

 dense_4 (Dense)             (None, 200)               40200

 dense_5 (Dense)             (None, 1)                 201

=================================================================
Total params: 281,601
Trainable params: 281,601
Non-trainable params: 0
_________________________________________________________________
Epoch 1/100
18916/18916 [==============================] - 449s 24ms/step - loss: 0.0885 - accuracy: 0.8929 - recall_m: 0.2013 - precision_m: 0.5692 - f1_score_m: 0.2808 - val_loss: 0.0750 -
Epoch 2/100
18916/18916 [==============================] - 323s 17ms/step - loss: 0.0861 - accuracy: 0.8955 - recall_m: 0.2401 - precision_m: 0.6013 - f1_score_m: 0.3245 - val_loss: 0.0733 -
Epoch 3/100
18916/18916 [==============================] - 352s 19ms/step - loss: 0.0845 - accuracy: 0.8974 - recall_m: 0.2647 - precision_m: 0.6179 - f1_score_m: 0.3504 - val_loss: 0.0745 -
Epoch 4/100
18916/18916 [==============================] - 309s 16ms/step - loss: 0.0834 - accuracy: 0.8984 - recall_m: 0.2696 - precision_m: 0.6266 - f1_score_m: 0.3567 - val_loss: 0.0734 -
Epoch 5/100
18916/18916 [==============================] - 361s 19ms/step - loss: 0.0829 - accuracy: 0.8989 - recall_m: 0.2786 - precision_m: 0.6311 - f1_score_m: 0.3660 - val_loss: 0.0742 -
Epoch 6/100
18916/18916 [==============================] - 314s 17ms/step - loss: 0.0819 - accuracy: 0.8999 - recall_m: 0.2871 - precision_m: 0.6399 - f1_score_m: 0.3754 - val_loss: 0.0736 -
```

در ادامه، با تعریف یک preprocess جدید و مدل LSTM، یک مدل self supervised را آموزش می دهیم.

```
sequence = 100

def preprocessing_X_new(df):
  arr_2 = np.array([df["value"]])

  x, y = [], []
  for i in range(len(arr_2[0])-sequence-1):
    x.append([arr_2[0][i:i+sequence]])

  # print(x)
  hold = np.array(x)

  return hold

def preprocessing_Y_new(df):
  arr_2 = np.array([df["value"]])

  x, y = [], []
  for i in range(len(arr_2[0])-sequence-1):
    x.append([arr_2[0][i+sequence-1]])

  # print(x)
  hold = np.array(x)

  return hold
```

```
A_new = preprocessing_X_new(X)
B_new = preprocessing_Y_new(X)
```

```
# compile and train the model


model_LSTM = tf.keras.Sequential([
                            tf.keras.layers.LSTM(units=sequence,  input_shape=(1,sequence)),
                            tf.keras.layers.Dense(sequence, activation='relu'),
                            tf.keras.layers.Dense(1, activation='sigmoid')
                           ])

model_LSTM.compile(optimizer='adam', loss='mean_squared_error', metrics=['accuracy',recall_m,precision_m,f1_score_m])
model_LSTM.summary()

history_LSTM_ss = model_LSTM.fit(A_new, B_new, epochs=100, verbose=1,validation_split=0.2,batch_size=64,callbacks=keras.callbacks.EarlyStopping(monitor='val_f1_score_m',
                                        patience=5,
                                        restore_best_weights=True))
```

خروجی:

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 100)               80400

 dense_6 (Dense)             (None, 100)               10100

 dense_7 (Dense)             (None, 1)                 101

=================================================================
Total params: 90,601
Trainable params: 90,601
Non-trainable params: 0
_________________________________________________________________
Epoch 1/100
18918/18918 [==============================] - 119s 6ms/step - loss: 41237676032.0000 - accuracy: 0.1050 - recall_m: 0.9972 - precision_m: 0.9957 -
Epoch 2/100
18918/18918 [==============================] - 115s 6ms/step - loss: 41237786624.0000 - accuracy: 0.1054 - recall_m: 0.9973 - precision_m: 0.9964 -
Epoch 3/100
18918/18918 [==============================] - 117s 6ms/step - loss: 41237647360.0000 - accuracy: 0.1054 - recall_m: 0.9976 - precision_m: 0.9964 -
Epoch 4/100
18918/18918 [==============================] - 119s 6ms/step - loss: 41237770240.0000 - accuracy: 0.1054 - recall_m: 0.9969 - precision_m: 0.9960 -
Epoch 5/100
18918/18918 [==============================] - 124s 7ms/step - loss: 41237663744.0000 - accuracy: 0.1054 - recall_m: 0.9970 - precision_m: 0.9963 -
Epoch 6/100
18918/18918 [==============================] - 120s 6ms/step - loss: 41237725184.0000 - accuracy: 0.1054 - recall_m: 0.9971 - precision_m: 0.9963 -
Epoch 7/100
18918/18918 [==============================] - 125s 7ms/step - loss: 41237594112.0000 - accuracy: 0.1054 - recall_m: 0.9970 - precision_m: 0.9963 -
Epoch 8/100
18918/18918 [==============================] - 118s 6ms/step - loss: 41237737472.0000 - accuracy: 0.1054 - recall_m: 0.9967 - precision_m: 0.9964 -
Epoch 9/100
18918/18918 [==============================] - 121s 6ms/step - loss: 41237590016.0000 - accuracy: 0.1054 - recall_m: 0.9969 - precision_m: 0.9962 -
Epoch 10/100
```

با حذف لایه آخر LSTM، دوباره summary آن را مشاهده میکنیم:

```
# delete last layer of model
model_LSTM_new = tf.keras.Sequential()

for layer in model_LSTM.layers[:-1]:
  model_LSTM_new.add(layer)

model_LSTM_new.summary()
```

```
Model: "sequential_11"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 100)               80400

 dense_6 (Dense)             (None, 100)               10100

=================================================================
Total params: 90,500
Trainable params: 0
Non-trainable params: 90,500
```

بعد از آن، با پیش پردازش دوباره، مدل جدید بدون لایه آخر را کامپایل می کنیم و کد ها و نتایج آن را در زیر می بینیم.

```python
for layer in model_LSTM_new.layers[:-1]:
    layer.trainable = False
```

```python
# add 2 dence layer to the model
model_LSTM_new.add(tf.keras.layers.Dense(10, activation='relu'))
model_LSTM_new.add(tf.keras.layers.Dense(1, activation='sigmoid'))
```

```python
def preprocessing_Y(df):
  arr_1 = np.array([df["label"]])

  x =[]
  for i in range(len(arr_1[0])-sequence):
    if np.all(arr_1[0][i:i+sequence]==0):
      x.append(0)
    else:
      x.append(1)

  # print(x)
  hold = np.array(x).astype(int)
  return hold
```

```python
def preprocessing_X_new(df):
  arr_2 = np.array([df["value"]])
  arr_2 = preprocessing.normalize(arr_2, norm='l2')

  x, y = [], []
  for i in range(len(arr_2[0])-sequence):
    x.append(arr_2[0][i:i+sequence])
```

```python
  x, y = [], []
  for i in range(len(arr_2[0])-sequence):
    x.append(arr_2[0][i:i+sequence])

  return np.array(x)
```

```python
A = preprocessing_X(X)
B = preprocessing_Y(y)
```

```python
# train the main task(anomaly detection)
model_LSTM_new.compile(optimizer='adam', loss='mean_squared_error', metrics=['accuracy',recall_m,precision_m,f1_score_m])
model_LSTM_new.summary()

his_model_LSTM_new = model_LSTM_new.fit(A, B, epochs=100, verbose=1,validation_split=0.2,batch_size=64,callbacks=keras.callbacks.EarlyStopping(monitor='val_f1_score_m',
                                        patience=5,
                                        restore_best_weights=True))
```

```
Model: "sequential_11"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 100)               80400

 dense_6 (Dense)             (None, 100)               10100

 dense_10 (Dense)            (None, 10)                1010

 dense_11 (Dense)            (None, 1)                 11

=================================================================
Total params: 91,521
Trainable params: 1,021
Non-trainable params: 90,500
_________________________________________________________________
Epoch 1/100
18918/18918 [==============================] - 74s 4ms/step - loss: 0.0633 - accuracy: 0.9216 - recall_m: 0.0907 - precision_m: 0.2584 - f1_score_m: 0.1222 - val_loss: 0.0476 -
Epoch 2/100
18918/18918 [==============================] - 73s 4ms/step - loss: 0.0587 - accuracy: 0.9254 - recall_m: 0.1104 - precision_m: 0.3210 - f1_score_m: 0.1527 - val_loss: 0.0478 -
Epoch 3/100
18918/18918 [==============================] - 69s 4ms/step - loss: 0.0580 - accuracy: 0.9261 - recall_m: 0.1201 - precision_m: 0.3486 - f1_score_m: 0.1662 - val_loss: 0.0479 -
Epoch 4/100
18918/18918 [==============================] - 83s 4ms/step - loss: 0.0574 - accuracy: 0.9268 - recall_m: 0.1177 - precision_m: 0.3550 - f1_score_m: 0.1656 - val_loss: 0.0478 -
Epoch 5/100
18918/18918 [==============================] - 96s 5ms/step - loss: 0.0571 - accuracy: 0.9272 - recall_m: 0.1237 - precision_m: 0.3667 - f1_score_m: 0.1726 - val_loss: 0.0480 -
Epoch 6/100
18918/18918 [==============================] - 88s 5ms/step - loss: 0.0568 - accuracy: 0.9274 - recall_m: 0.1290 - precision_m: 0.3795 - f1_score_m: 0.1802 - val_loss: 0.0487 -
Epoch 7/100
18918/18918 [==============================] - 84s 4ms/step - loss: 0.0565 - accuracy: 0.9277 - recall_m: 0.1359 - precision_m: 0.3894 - f1_score_m: 0.1878 - val_loss: 0.0476 -
Epoch 8/100
18918/18918 [==============================] - 89s 5ms/step - loss: 0.0563 - accuracy: 0.9280 - recall_m: 0.1417 - precision_m: 0.4003 - f1_score_m: 0.1953 - val_loss: 0.0491 -
Epoch 9/100
```

سپس باید مشکل داده های unbalanced را کنترل کنیم. هندلر مناسب برای این قسمت از وزن های کلاس استفاده می کند، اما فقط برچسب را در کمتر از 2 بعد می پذیرد. y_train ما 20 بعد دارد. با تولید نمونه وزن ها و وزن دهی به کلاس ها، یک مدل LSTM جدید ساخته و کامپایل می کنیم:

```
sequence = 200
```

```
def preprocessing_X(df):
  arr_2 = np.array([df["value"]])

  x, y = [], []
  for i in range(len(arr_2[0])-sequence-1):
    x.append([arr_2[0][i:i+sequence]])

  # print(x)
  hold = np.array(x)

  return hold
```

```
def preprocessing_Y(df):
  arr_1 = np.array([df["label"]])

  x =[]
  for i in range(len(arr_1[0])-sequence):
    if np.all(arr_1[0][i:i+sequence]==0):
      x.append(0)
    else:
      x.append(1)
```

```
A = preprocessing_X(X)
B = preprocessing_Y(y)
```

```
def generate_sample_weights(training_data, class_weight_dictionary):
    sample_weights = [class_weight_dictionary[np.where(one_hot_row==1)[0][0]] for one_hot_row in training_data]
    return np.asarray(sample_weights)
```

```
unique, counts = np.unique(B, return_counts=True)
num = dict(zip(unique, counts))
```

```
class_weight = {0: 1,
                1: (num[0]/num[1])}
# LSTM
model_LSTM = tf.keras.Sequential([
                            tf.keras.layers.LSTM(units=sequence,  input_shape=(1,sequence)),
                            tf.keras.layers.Dense(sequence, activation='relu'),
                            tf.keras.layers.Dense(1, activation='sigmoid')
                           ])

model_LSTM.compile(optimizer='adam', loss='mean_squared_error', metrics=['accuracy',recall_m,precision_m,f1_score_m])
model_LSTM.summary()

history_LSTM_new = model_LSTM.fit(A, B, epochs=100, verbose=1,validation_split=0.2,batch_size=64,callbacks=keras.callbacks.EarlyStopping(monitor='val_f1_score_m',
                                        patience=5,
                                        restore_best_weights=True),
                                        class_weight =class_weight)
```

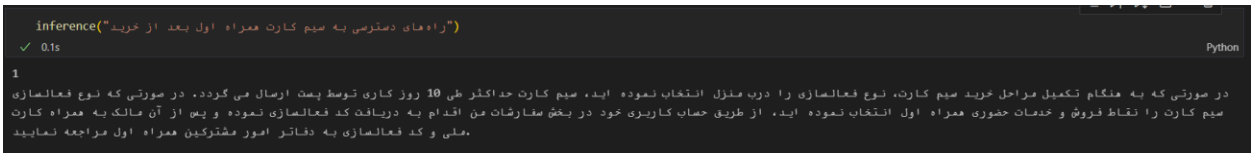
پس از این کار، یک تمتد برای پیدا کردن ناهنجاری درست می کنیم.

```
mean = np.mean(np.array([csvmain["value"]]))
std = np.std(np.array([csvmain["value"]]))
print('mean of the dataset is', mean)
print('std. deviation is', std)
```

```
mean of the dataset is 6566.659450792839
std. deviation is 181523.86640913255
```

```
threshold = 1
outlier = []
for i in np.array([csvmain["value"]])[0]:
    z = (i-mean)/std
    if z > threshold:
        outlier.append(i)
print('outlier in dataset is', outlier)
```

در انتها برای بررسی عملکرد، از تابع inference استفاده کردیم که وظیفه آن این است که یک جمله را به عنوان ورودی دریافت میکند و با پیشبینی بر روی مدل، تشخیص دهد که جمله ورودی مربوط به کدام کلاس است. در آخر سر کلاس مربوطه را به دیکشنری مربوط به label ها میدهیم که جمله مربوط به آن را چاپ کند.



برای ساخت این تابع ابتدا باید جمله ورودی را به tokenizer ای که داریم بدهیم و سپس آن را به یک matrix تبدیل میکنیم و به تابع پیش بینی مدل خود میدهیم. این تابع لیستی تولید میکند که احتمال هر کلاس را بازگو میکند و باید با کمک argmax، پاسخ خود را extract کنیم و جمله مربوط به آن را چاپ کنیم.

```
def inference(question):
  sequences = tok.texts_to_sequences(X)
  sequences_matrix_test = sequence.pad_sequences(sequences,maxlen=max_len, dtype='float')

  hold = model.predict(sequences_matrix_test)
  A_ = np.argmax(hold,axis=1)

  labels_pred = {}
  counter = 0
  for count in range(A_.shape[0]):
      if A_[count] == A_[count]:
          if A_[count] not in labels_pred.values():
              labels_pred[counter] = A_[count]
              counter +=1
          A_[count] = counter

  indx = int(list(labels_pred.keys())[list(labels_pred.values()).index(max(labels_pred.values()))])
  print(indx)
  print(labels[indx])
```