به نام خدا

دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

درس شبکه‌های عصبی و یادگیری عمیق

تمرین چهارم

نام و نام خانوادگی	حمید نعمتی – مهرداد نوربخش
شماره دانشجویی	810100492 - 810100495
تاریخ ارسال گزارش	۱۴۰۱،۰۷،۰۱

فهرست

شکل‌ها

پاسخ 1. تخمین آلودگی هوا

لینک کولب اول:

https://colab.research.google.com/drive/1xp7ByLuRcYnf8gd0R1DFykz3M3Dic_XV?usp=sharing

لینک کولب دوم:

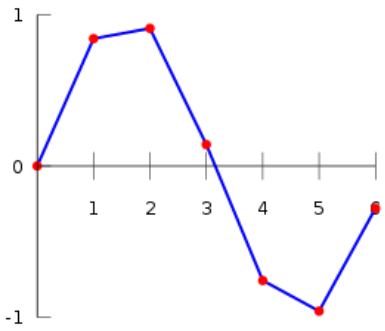
https://drive.google.com/file/d/1pm84WBI2TZa1M2fXAJdOsA6serbGxl--/view?usp=sharing

محتوای هر دو یکسان است. فقط مدل‌های ۵ و ۶ در کولب اول دیگر GPU نداشتند.

۱-۱. سوالات تشریحی

- **Linear interpolation method**: برای پر کردن missing value ها در یک سری زمانی از چند روش استفاده میشود. یکی از این روش‌های interpolation یا درون یابی است که اولین داده‌ی قبل از missing value ها و اولین داده بعد از missing value ها را با یک خط بهم وصل میکنیم. این روش از برای مسئله آلودگی هوا روش مناسبی است زیرا تغییر هوا فرایندی ناگهانی نیست و تدریجی اتفاق میافتد. عیب این روش استفاده از داده‌ی اینده است. و اندکی دقت مدل را به صورت غلط بالا میبرد زیرا که مدل از روی شیب خط تمام دادگان miss شده را میتواند درست پیشبینی کند.

شکل 1: شیوه انجام **interpolation** در سری زمانی



شکل 2: یک مثال دیگر از **interpolation** برای تبدیل داده گسسته به پیوسته

در سری زمانی‌های مثل بورس که نباید look ahead bias داشته باشیم و به هیچ وجه نباید از دانستن داده‌های اینده کمک بگیریم از interpolation استفاده نمیکنیم. به جای این روش از روش‌های forward filling و Backward filling استفاده میشود.

شکل 3: روش **forward filling** برای حل مشکل داده‌های گمشده در سری زمانی های مالی

شکل 4: روش **Backward filling** برای حل مشکل داده‌های گمشده در سری زمانی های مالی

- **Pearson correlation**: رایج ترین روش اندازه‌گیری correlation خطی دو ویژگی است (**bivariate correlation**) و مقداری بین -1 و 1 برمیگرداند. اگر عدد 0 باشد یعنی هیچ correlationی وجود ندارد. اگه مثبت باشد یعنی دو ویژگی در جهت یکسان رشد میکنند و افزایش یکی با افزایش دیگری همراه است و مقدار منفی به معنی در خلاف جهت حرکت کردن دو متغیر است. فرمول محاسبه ان مانند شکل زیر است. cov همان Covariance است ($E[(X-\mu_X)(Y-\mu_Y)]$) و σ انحراف معیار متغیر.

$$\rho_{XY} = \frac{cov(X,Y)}{\sigma_X \sigma_Y}$$

$$\rho_{X,Y} = \frac{\mathbb{E}[XY] - \mathbb{E}[X]\mathbb{E}[Y]}{\sqrt{\mathbb{E}[X^2] - (\mathbb{E}[X])^2}\,\sqrt{\mathbb{E}[Y^2] - (\mathbb{E}[Y])^2}}.$$

- **R^2 or coefficient of determination**: معیاری است که میزان پیشبینی پذیری متغییر وابسته از متغیر مستقل را نشان میدهد. هرچه خط رگرسیون بهتری به مدل فیت شود مقدار آن به یک نزدیکتر است. این معیار مقداری بین 0 و ۱ دارد (البته در زمان‌های خاصی میتواند منفی هم باشد) و از این نظر قابل تفصیر تر از MAE و RMSE است. فرمول آن به شکل زیر است:

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}} \quad , \quad SS_{res} = \sum_i e_i^2 \quad , \quad SS_{tot} \sum_i (y_i - \bar{y})^2$$

۲-۱. دیتاست

ابتدا داده‌ها را در drive قرار داده و سپس اسم تمام فایل‌ها را با glob میخوانیم سپس با استفاده از pandas فایل‌های csv را تبدیل به data-frame میکنیم و لیستی ازین data-frame ها میسازیم.

```
[2]  from google.colab import drive
     drive.mount('/content/MyDrive/')

     Mounted at /content/MyDrive/

[3]  !ls "/content/MyDrive/MyDrive/Projects_Data/PRSA_Data"

     PRSA_Data_Aotizhongxin_20130301-20170228.csv
     PRSA_Data_Changping_20130301-20170228.csv
     PRSA_Data_Dingling_20130301-20170228.csv
     PRSA_Data_Dongsi_20130301-20170228.csv
     PRSA_Data_Guanyuan_20130301-20170228.csv
     PRSA_Data_Gucheng_20130301-20170228.csv
     PRSA_Data_Huairou_20130301-20170228.csv
     PRSA_Data_Nongzhanguan_20130301-20170228.csv
     PRSA_Data_Shunyi_20130301-20170228.csv
     PRSA_Data_Tiantan_20130301-20170228.csv
     PRSA_Data_Wanliu_20130301-20170228.csv
     PRSA_Data_Wanshouxigong_20130301-20170228.csv

[4]  path_list = glob.glob('/content/MyDrive/MyDrive/Projects_Data/PRSA_Data/*')

     df_list = []
     for i, path in enumerate(path_list):
         df_list.append( pd.read_csv(path) )
```

شکل 5: دیتاست

۳-۱. پیش پردازش

پیش پردازش‌ها را طبق آنچه در مقاله گفته شده است انجام میدهیم. مواردی هم در مقاله گفته نشده است را سعی کرده‌ایم منطقی ترین راه برای پیش پردازش داده‌ها را انتخاب کنیم و انجام دهیم.

۱-۳-۱. Missing Value

ابتدا نگاهی به ستون‌هایی که داده گمشده دارند میاندازیم:

Missing Values

```
[8] df_list[0].isna().sum()

No            0
year          0
month         0
day           0
hour          0
PM2.5       925
PM10        718
SO2         935
NO2        1023
CO         1776
O3         1719
TEMP         20
PRES         20
DEWP         20
RAIN         20
wd           81
WSPM         14
station       0
dtype: int64

[9] # number of missing Values in whole Aotizhongxin dataset
    df_list[0].isna().sum().sum()

7271
```

شکل 6: لیست داده‌های گم شده در هر ستون

سپس interpolation را برای همه ستون‌های PM2.5 در همه دیتاست‌ها انجام میدهیم.

```
[10] # Linear interpolation Methods: linear, index, values
     # we use linear method because we don't have missing index

[11] for i, df in enumerate(df_list):
         df['PM2.5'].interpolate(limit_direction='both', inplace=True)

     df_list[0].isna().sum()

No            0
year          0
month         0
day           0
hour          0
PM2.5         0
PM10        718
SO2         935
NO2        1023
CO         1776
O3         1719
TEMP         20
PRES         20
DEWP         20
RAIN         20
wd           81
WSPM         14
station       0
dtype: int64
```

شکل 7: لیست داده‌های گمشده برای هر ستون پس از پر کردن **PM2.5**

حال سایر ستون‌های دیتاست اصلی که قرار است در آینده مدل را روی آن اموزش دهیم را با interpolation اصلاح میکنیم تا داده گمشده نداشته باشند. وجود داده گمشده باعث Nan شدن تمام معیارها در شبکه عصبی و در نتیجه اموزش ندیدن آن است.

```
[13] # now rest of the columns in Aotizhongxin

[14] from pandas.api.types import is_numeric_dtype

     for i, col in enumerate( df_list[0].columns ):
         if is_numeric_dtype( df_list[0][col] ):
             df_list[0][col].interpolate(limit_direction='both', inplace=True)

     df_list[0].isna().sum()

No          0
year        0
month       0
day         0
hour        0
PM2.5       0
PM10        0
SO2         0
NO2         0
CO          0
O3          0
TEMP        0
PRES        0
DEWP        0
RAIN        0
wd         81
WSPM        0
station     0
dtype: int64
```

شکل 8: لیست داده‌های گمشده پس از پر کردن تمام ستون‌های عددی با **interpolation**

هنوز یک ستون مانده است که داده گمشده دارد. این ستون دارای داده عددی نیست و interpolation برای آن معنی ندارد. مقادیر گمشده این ستون را در بخش‌های بعدی پر میکنیم. مشکلی که وجود دارد مقاله هیچ اشاره ای به چگونه پر شدن این ستون نمیکند. روش interpolation برای این ستون چندان منطقی به نظر نمیرسد زیرا که عددهایی که در این ستون‌های قرار میگیرند با 16 جهت جغرافیایی ما در تضاد هستند. اگر نگاهی عمیق تر به ستون بیاندازیم میبینیم یکی از گروه‌ها با اختلاف غالب تر است پس با همان داده most frequent داده‌های گمشده را پر میکنیم. شکل زیر مربوط به پر کردن داده گمشده پس از تبدیل داده گروه بندی شده به داده عددی است:

```
df_list[0]['wd'].value_counts()
```

```
45.0     5140
67.5     3950
225.0    3359
90.0     2608
22.5     2445
247.5    2212
202.5    2098
0.0      2066
315.0    1860
112.5    1717
337.5    1589
135.0    1341
180.0    1304
270.0    1171
292.5    1101
157.5    1022
Name: wd, dtype: int64
```

```
[22] # interpolation is not a good option to fill the missing values here. we fill them with most frequent direction
     df_list[0]['wd'].fillna(df_list[0]['wd'].value_counts().index[0], inplace=True)
```

```
[23] df_list[0]['wd'].isna().sum()
```

```
0
```

شکل 9: تعداد تکرار هر کدام از جهت‌های جغرافیایی باد در ستون **wd**

Encoding Categorical Variable .۱-۳-۲

ستون wd را طبق گفته مقاله به داده‌ی عددی تبدیل میکنیم که نشان دهنده درجه است.

```
[17] directions_list = ['N', 'NNE', 'NE', 'ENE', 'E', 'ESE', 'SE', 'SSE', 'S', 'SSW', 'SW', 'WSW', 'W', 'WNW', 'NW', 'NNW']
```

```
[18] for i, val in enumerate(df_list[0]['wd']):
         if val not in directions_list: continue
         df_list[0]['wd'][i] = 22.5 * directions_list.index( df_list[0]['wd'][i] )
```

```
<ipython-input-18-e6bb46125a0a>:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_list[0]['wd'][i] = 22.5 * directions_list.index( df_list[0]['wd'][i] )
```

```
[19] df_list[0]['wd'] # This panda.series still has missing values
```

```
0        337.5
1          0.0
2        337.5
3        315.0
4          0.0
         ...
35059    315.0
35060    292.5
35061    315.0
35062    337.5
35063     22.5
Name: wd, Length: 35064, dtype: object
```

شکل 10: تبدیل داده‌ی **Categorical** ستون **wd** به داده‌های عددی

Normalization .۱-۳-۳

نرمال سازی داده‌های مسئله‌ی ضروری است زیرا که اگر اعداد یکی از ستون‌ها بسیار بزرگ و ستون دیگر بسیار کوچک باشد مدل تمام تلاشش را روی ستون با داده بزرگ میگزارد زیرا که باعث کاهش loss بیشتری میشود. پس کار درست این است که همه ستون‌هایی که در آموزش نهایی شبکه دخیل هستند را نرمال سازی کنیم.

Normalization

In order to improve the prediction accuracy, we normalize the values of $PM_{2.5}$ concentration using the Min-Max normalization, the method is given in the equation 13:

```
[24] from sklearn.preprocessing import MinMaxScaler

[25] for i, df in enumerate(df_list):
         if 'No' in df.columns:
             df.drop('No', axis=1, inplace=True)
         mms = MinMaxScaler()
         # The Article is normalizing Train and test data together
         df['PM2.5'] = mms.fit_transform(np.array( df['PM2.5'] ).reshape(-1, 1))

[26] for i, col in enumerate(df_list[0].columns):
         if col in ['station', 'year', 'month', 'day', 'hour', 'PM2.5']:
             continue
         else:
             mms = MinMaxScaler()
             df_list[0][col] = mms.fit_transform( np.array( df_list[0][col] ).reshape(-1, 1) )
```

شکل 11: نرمال کردن ستون **PM2.5** در تمام جدول‌ها و نرمال کردن داده‌های ایستگاهی که برای آموزش مدل استفاده میشود.

Pearson Correlation .۱-۳-۴

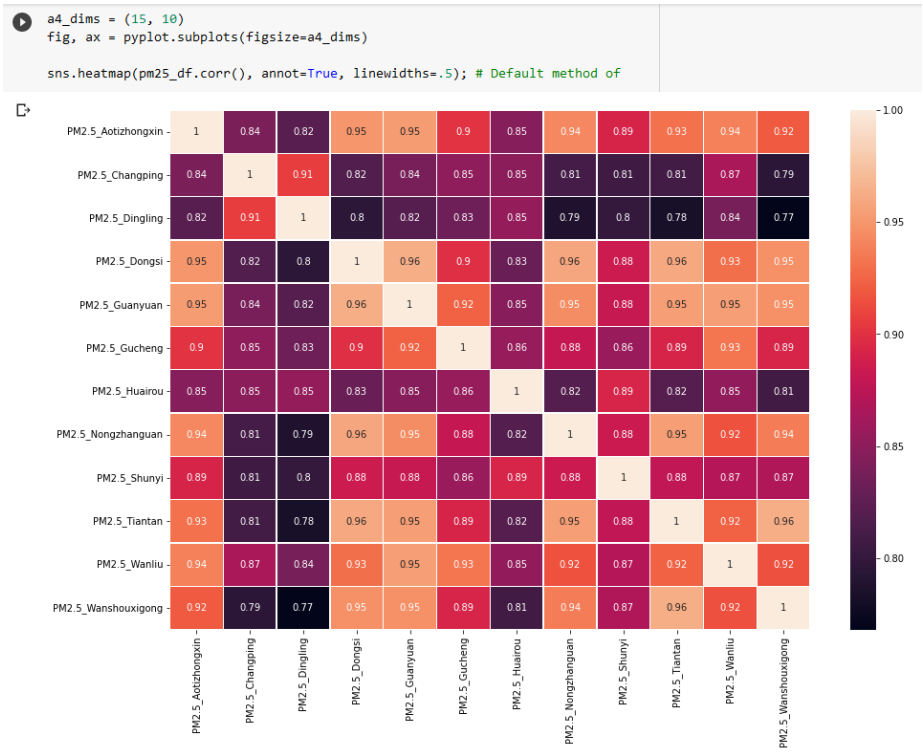
ابتدا دیتاستی میسازیم از ستون‌هایی که قرار است correlation آنها را حساب کنیم سپس با استفاده از خود pandas این correlationها را حساب میکنیم.

```
pm25_list = []
for i, df in enumerate(df_list):
    pm25_list.append( df_list[i]['PM2.5'] )

pm25_df = pd.DataFrame( np.array(pm25_list).T )
pm25_df.columns = ['PM2.5_Aotizhongxin', 'PM2.5_Changping', 'PM2.5_Dingling', 'PM2.5_Dongsi', 'PM2.5_Guanyuan',
                   'PM2.5_Gucheng', 'PM2.5_Huairou', 'PM2.5_Nongzhanguan', 'PM2.5_Shunyi', 'PM2.5_Tiantan', 'PM2.5_Wanliu',
                   'PM2.5_Wanshouxigong']
pm25_df
```

	PM2.5_Aotizhongxin	PM2.5_Changping	PM2.5_Dingling	PM2.5_Dongsi	PM2.5_Guanyuan	PM2.5_Gucheng	PM2.5_Huairou	PM2.5_Nongzhanguan	PM2.5_Shunyi	PM2.5_Tiantan	PM2.5_Wanliu	PM2.5_Wanshouxigong
0	0.001117	0.001136	0.001139	0.008174	0.002950	0.005208	0.006579	0.003563	0.001065	0.003667	0.006283	0.006024
1	0.005587	0.001136	0.004556	0.001362	0.002950	0.005208	0.002632	0.007126	0.010650	0.003667	0.007330	0.008032
2	0.004469	0.001136	0.002278	0.005450	0.001475	0.003906	0.002632	0.001188	0.012780	0.003667	0.001047	0.005020
3	0.003352	0.001136	0.003417	0.000000	0.001475	0.005208	0.001316	0.003563	0.010650	0.003667	0.009424	0.005020
4	0.000000	0.001136	0.002278	0.000000	0.001475	0.003906	0.001316	0.003563	0.010650	0.002445	0.001047	0.005020
...	...	...	...	...	...	...	...	...	...	...	...	...
35059	0.010056	0.029545	0.009112	0.017711	0.016224	0.015625	0.018421	0.014252	0.026624	0.020782	0.009424	0.008032
35060	0.011173	0.011364	0.011390	0.020436	0.026549	0.032552	0.025000	0.019002	0.047923	0.009780	0.013613	0.010040
35061	0.014525	0.005682	0.006834	0.027248	0.020649	0.026042	0.019737	0.015439	0.017039	0.018337	0.011518	0.011044
35062	0.020112	0.010227	0.007973	0.027248	0.013274	0.009115	0.011842	0.010689	0.017039	0.014670	0.010471	0.009036
35063	0.017877	0.020455	0.011390	0.036785	0.019174	0.013021	0.011842	0.009501	0.013845	0.014670	0.005236	0.010040

35064 rows × 12 columns

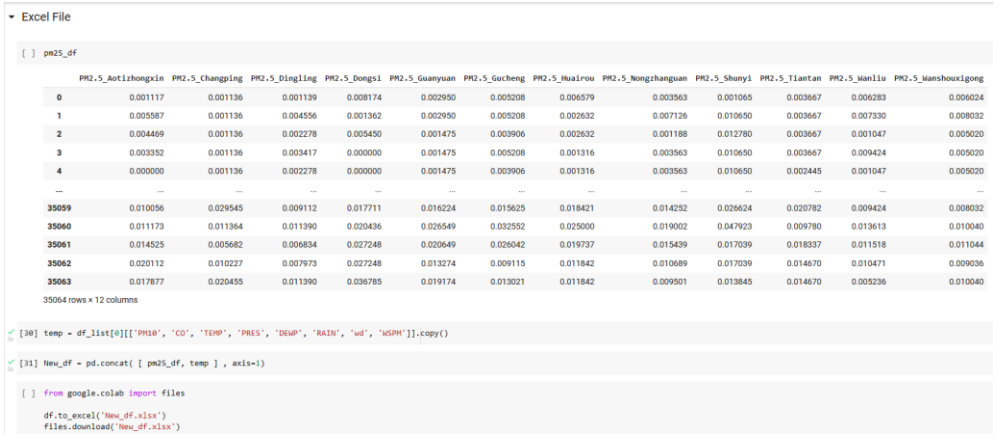
شکل 12: یک **dataframe** حاوی تمام **PM2.5** های موجود در تمام ایستگاه‌ها



شکل 13: به دست اوردن **pearson correlation** با استفاده از **pandas**

۱-۳-۵. Feature selection

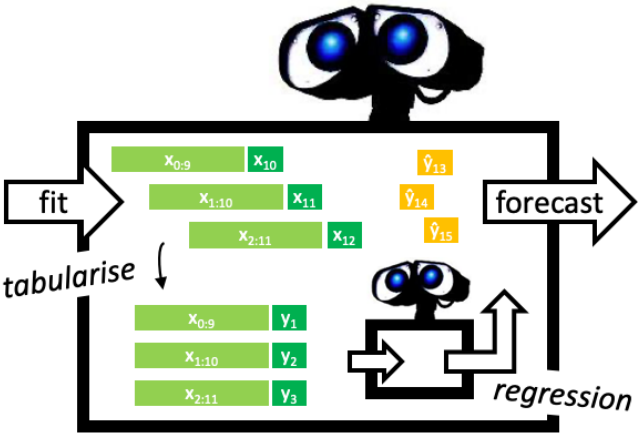
در شکل زیر new_df را میسازیم که همان جدول درخواست شده در صورت سوال است.



شکل 14: انجام **feature selection** و اسخراج آن از محیط کولب به صورت یک فایل اکسل

۱-۳-۶. Supervised Dataset

در شکل زیر که از document کتابخانه Sktime آورده شده است (ما از خود کتابخانه استفاده نکردیم و تمام کارها را دستی انجام دادیم.) به خوبی شیوه تبدیل یک مسئله سری زمانی به یک مسئله classification یا regression دیده میشود.



شکل 15: تبدیل داده‌ی سری زمانی به یک مسئله **supervised**

ابتدا X و Y را برای حالت lag 1 میسازیم. lag 1 یعنی فقط به داده‌های ساعت گذشته توجه میکنیم برای پیشبینی PM2.5 ساعت بعد. سپس train-test spilt را انجام میدهیم. این کار در سری زمانی با بقیه مسائل یادگیری ماشین متفاوت است زیرا که اگر کل داده‌ها را shuffle کنیم و سپس تقسیم کنیم دچار پدیده ای به اسم look ahead bias میشویم که دقت واقعی مدل را نمایش نمیدهد.

```
[32] X = New_df.copy()

     split = int(len(New_df)*0.8) + 1

     X_train_1_lag = X[:split]
     X_test_1_lag  = X[split:-1]

     print(len(X_train_1_lag))
     print(len(X_test_1_lag ))

     28052
     7011

[33] Y = New_df['PM2.5_Aotizhongxin']

     Y_train_1_lag = Y[1:split+1]
     Y_test_1_lag  = Y[split+1:]

     print(len(Y_train_1_lag))
     print(len(Y_test_1_lag ))

     28052
     7011
```

شکل 16: انجام **train-test-split** به صورت دستی

حال داده‌های 7 روز گذشته را کنار هم میگزاریم و حالت lag 7 را بسازیم. در این حالت چند ستون اولیه دارای مقادیر null میشوند و باید drop شوند.

```
num_lags = 7
concat_list = []

for i in range(num_lags):
    concat_list.append( New_df.shift(i) )

X_7_lag = pd.concat( concat_list, axis=1)
X_7_lag
```

	PM2.5_Aotizhongxin	PM2.5_Changping	PM2.5_Dingling	PM2.5_Dongsi	PM2.5_Guanyuan	PM2.5_Gucheng	PM2.5_Huairou	PM2.5_Nongzhanguan	PM2.5_Shunyi	PM2.5_Tiantan	...
0	0.001117	0.001136	0.001139	0.008174	0.002950	0.005208	0.006579	0.003563	0.001065	0.003667	...
1	0.005587	0.001136	0.004556	0.001362	0.002950	0.005208	0.002632	0.007126	0.010650	0.003667	...
2	0.004469	0.001136	0.002278	0.005450	0.001475	0.003906	0.002632	0.001188	0.012780	0.003667	...
3	0.003352	0.001136	0.003417	0.000000	0.001475	0.005208	0.001316	0.003563	0.010650	0.003667	...
4	0.000000	0.001136	0.002278	0.000000	0.001475	0.003906	0.001316	0.003563	0.010650	0.002445	...
...	...	...	...	...	...	...	...	...	...	...	...
35059	0.010056	0.029545	0.009112	0.017711	0.016224	0.015625	0.018421	0.014252	0.026624	0.020782	...
35060	0.011173	0.011364	0.011390	0.020436	0.026549	0.032552	0.025000	0.019002	0.047923	0.009780	...
35061	0.014525	0.005682	0.006834	0.027248	0.020649	0.026042	0.019737	0.015439	0.017039	0.018337	...
35062	0.020112	0.010227	0.007973	0.027248	0.013274	0.009115	0.011842	0.010689	0.017039	0.014670	...
35063	0.017877	0.020455	0.011390	0.036785	0.019174	0.013021	0.011842	0.009501	0.013845	0.014670	...

35064 rows × 140 columns

شکل 17: جدول حاوی داده‌های **7** ساعت گذشته که شامل 140 ستون است

```
[35] X_train_7_lag = X_7_lag[num_lags:split]
     X_test_7_lag  = X_7_lag[split:-1]

     Y_train_7_lag = Y[num_lags+1:split+1]
     Y_test_7_lag  = Y[split+1:]

     print(len(X_train_7_lag))
     print(len(X_test_7_lag ))
     print(len(Y_train_7_lag))
     print(len(Y_test_7_lag ))

     28045
     7011
     28045
     7011

[36] X_train_1_lag.shape

     (28052, 20)
```

شکل 18: انجام **train-test-split** برای حالت **7 lag**

شیوه دیگری هم برای انجا این کار به صورت ساده تر بود که یکی از آنها را در زیر نمایش میدهیم:

Method 2 of train test split

```
[ ] train_dataset = tf.keras.utils.timeseries_dataset_from_array(
        X[:split],
        Y[1:split+1],
        sequence_length=1,
        sampling_rate=1,
        batch_size=32,
    )

    test_dataset  = tf.keras.utils.timeseries_dataset_from_array(
        X[split:-1],
        Y[split+1:],
        sequence_length=1,
        sampling_rate=1,
        batch_size=32,
    )
```

شکل 19: انجام **train-test-split** با استفاده از کتابخانه‌ها

۴-۱. آموزش شبکه

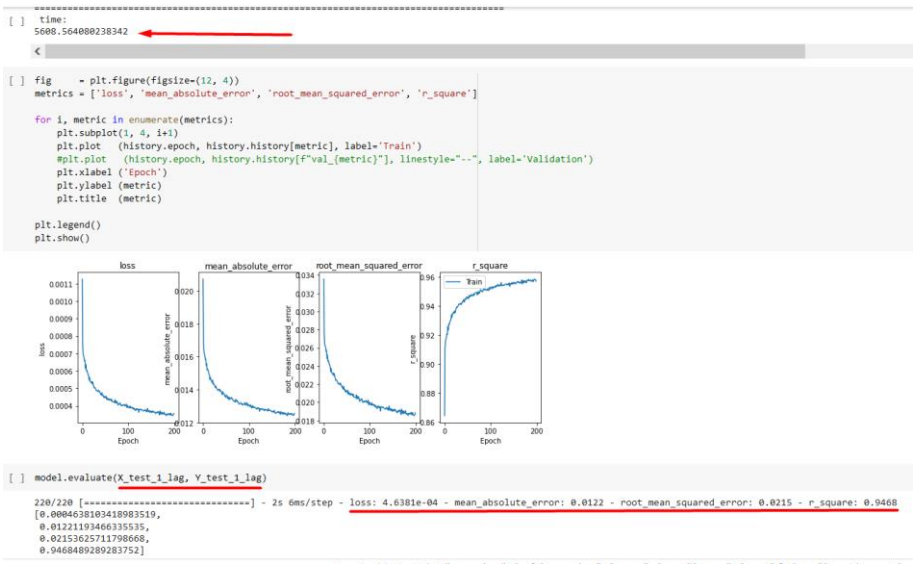
تعدادی از جزئیات شبکه در مقاله اعلام نشده است. ما برای اطمینان هر حالتی را که شک داشتیم که احتمالا منظور مقاله بوده زده‌ایم و در نهایت شش مدل ایجاد شد. مدل یک تا چهار برای زمانی است که از lag 1 استفاده میکنیم و مدل پنج و شش برای حالتی است که از lag 7 استفاده میکنیم. مدل دو مدل برتر برای حالت lag 1 است. همچنین یافتیم که استفاده نکردن از داده validation در اینجا بهتر است و مقاله هم گویا استفاده نکرده است و اسمی از آن نبرده است. همچنین اگر از recurrent drop out به جای drop out معمولی استفاده کنیم نتیجه بهتری میگیریم. مدل‌های 5 و 6 به دلیل بزرگتر بودن اندازه ورودی خیلی کندتر آموزش میبینند. در شکل‌های زیر مدل‌ها را میبینیم:

Model 1: Sequental model, Without validation data, using 1 lag, exteranl drop out

```
[ ] model = Sequential()

    model.add(Conv1D(64, kernel_size=3, activation='relu', padding='causal', input_shape=(20, 1) )) # (inputs.shape[1], inputs.shape[2])
    model.add(BatchNormalization())
    model.add(Conv1D(64, kernel_size=3, activation='relu', padding='causal'))
    model.add(BatchNormalization())
    model.add(Conv1D(32, kernel_size=3, activation='relu', padding='causal'))
    model.add(MaxPooling1D(pool_size=3))
    model.add(LSTM(100 , activation='relu', return_sequences=True))
    model.add(Dropout(0.2))
    model.add(LSTM(50  , activation='relu',))
    model.add(Dropout(0.3))
    model.add(Dense(1 , activation='relu'))

    model.compile(optimizer = tf.keras.optimizers.Adam(learning_rate=0.001, decay=0.0001 ) , loss = 'mse' ,
                  metrics =[
                            tf.keras.metrics.MeanAbsoluteError(),
                            tf.keras.metrics.RootMeanSquaredError(),
                            RSquare()
                           ])

    print( model.summary() )
    print('\n=================================================================================')

    # checkpoint     = tf.keras.callbacks.ModelCheckpoint("model.hdf5", monitor="val_loss", verbose=1, mode='min', save_best_only=True)
    early_stopping = tf.keras.callbacks.EarlyStopping  (patience=50, min_delta=1e-3, verbose=1, mode='min', restore_best_weights=True)

    start   = time()
    history = model.fit(X_train_1_lag, Y_train_1_lag, epochs=200, batch_size=32, verbose=1, callbacks=[early_stopping])

    print('\n=================================================================================\n time: ')
    print( time() - start )
```

شکل 20: مدل ۱

در مقاله اشاره‌ای به استفاده از داده validation نشد. ما مدل‌هایی با و بدون validation را اموزش دادیم. مدل یک مدل بودن validation است. همچنین در کنار LSTM به واژه dropout اشاره شده است و این ابهام دارد که dropout داخل LSTM است یا لایه dropout. همچنین یکی از مدل‌ها را به صورت Functional به جای Sequential تعریف کردیم ولی چون نتایج یکسان میداد اقدام به اجرای آن نکردیم ولی متوجه شدیم با این روش میشود شبکه‌هایی با ساختارهای بسیار پیچیده‌تر را ساخت. مدل یک حالت لایه drop out را در نظر گرفته است.



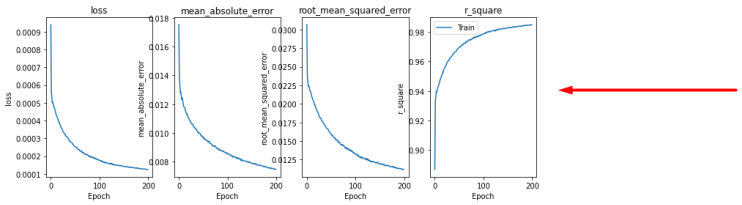
شکل 21: ارزیابی مدل ۱

میبینیم که خطاهای بسیار خوبی (خطای کمتر بهتر است) روی داده تست دارد مدل شماره یک و با نوسان ریزی به سمت کاهش خطا میرود. در مدل دوم که مشابه مدل اول است و فقط لایه dropout به صورت داخلی برای LSTM تعریف میشود و در حقیقت recurrent drop out است، میبینیم که مدل به صورت هموار تری خطا را کاهش میدهد. در مدل یک MAE برابر 0.0122 است و در مدل دوم برابر 0.0091 که بسیار بهتر از مدل یک است. پس recurrent drop out بهتر از لایه Dropout است در اینجا ولی مشکلی که ایجاد میکند افزایش زمان اموزش است.

شکل 22: مدل ۲

```
time:
9502.105081558228
```

```
fig     = plt.figure(figsize=(12, 4))
metrics = ['loss', 'mean_absolute_error', 'root_mean_squared_error', 'r_square']

for i, metric in enumerate(metrics):
    plt.subplot(1, 4, i+1)
    plt.plot   (history_2.epoch, history_2.history[metric], label='Train')
    #plt.plot   (history.epoch, history.history[f"val_{metric}"], linestyle="--", label='Validation')
    plt.xlabel ('Epoch')
    plt.ylabel (metric)
    plt.title  (metric)

plt.legend()
plt.show()
```




```
model_2.evaluate(X_test_1_lag, Y_test_1_lag)

119/220 [===============>.............] - ETA: 0s - loss: 2.1380e-04 - mean_absolute_error: 0.0091 - root_mean_squared_error: 0.0146 - r_square: 0.9265
```

شکل 23: ارزیابی مدل ۲

Model 3: Functional model, Without validation data, using 1 lag, internal drop out

```
# Functional way of making the same network
inputs = keras.Input(shape=(20, 1))
hidden = keras.layers.Conv1D(64, kernel_size=3, activation='relu', padding='causal', input_shape=(20, 1) ) (inputs)
hidden = keras.layers.BatchNormalization()(hidden)
hidden = keras.layers.Conv1D(64, kernel_size=3, activation='relu', padding='causal')(hidden)
hidden = keras.layers.BatchNormalization()(hidden)
hidden = keras.layers.Conv1D(32, kernel_size=3, activation='relu', padding='causal')(hidden)
hidden = keras.layers.MaxPooling1D(pool_size=3)(hidden)
hidden = keras.layers.LSTM(100 , activation='relu', return_sequences=True, recurrent_drop = 0.2)(hidden)
hidden = keras.layers.LSTM(50  , activation='relu', recurrent_drop = 0.3)(hidden)
hidden = keras.layers.Dense(1 , activation='relu')(hidden)
output = keras.layers.Dense(10, activation='softmax')(hidden)

model_3 = keras.Model(inputs=inputs, outputs=output)

model_3.compile(optimizer = tf.keras.optimizers.Adam(learning_rate=0.001, decay=0.0001 ) , loss = 'mse' ,
                metrics =[
                          tf.keras.metrics.MeanAbsoluteError(),
                          tf.keras.metrics.RootMeanSquaredError(),
                          RSquare()
                         ])

print( model_3.summary() )
print('\n=================================================================================')

# checkpoint     = tf.keras.callbacks.ModelCheckpoint("model.hdf5", monitor="val_loss", verbose=1, mode='min', save_best_only=True)
early_stopping = tf.keras.callbacks.EarlyStopping  (patience=50, min_delta=1e-3, verbose=1, mode='min', restore_best_weights=True)

start   = time()
history = model_3.fit(X_train_1_lag, Y_train_1_lag, epochs=200, batch_size=32, verbose=1, callbacks=[early_stopping])
print('\n=================================================================================\n time: ')
print( time() - start )
```

شکل 24: مدل ۳

مدل سوم را به صورت functional تعریف کردیم تا با این شیوه ساختن شبکه عصبی هم آشنا شویم. ولی نیازی به اجرای آن نیست زیرا که نتیجه مشابهی میدهد.

در مدل چهارم 20 درصد داده آموزشی را به عنوان داده Validation در نظر میگیریم و عملکرد مدل نه تنها افت میکند بلکه سریع به early-stopping میخوریم و مدل درست اموزش نمیبیند.

Model 4: Sequental model, With validation data, using 1 lag, exteranl drop out

```
[45] split_2 = int( len(X_train_1_lag) * 0.8 )

     X_val_1_lag   = X_train_1_lag[split_2:]
     X_train_1_lag = X_train_1_lag[:split_2]

     Y_val_1_lag   = Y_train_1_lag[split_2:]
     Y_train_1_lag = Y_train_1_lag[:split_2]

     print(len(X_train_1_lag))
     print(len(X_val_1_lag  ))

     print(len(Y_train_1_lag))
     print(len(Y_val_1_lag  ))

     20196
     5050
     20196
     5050
```

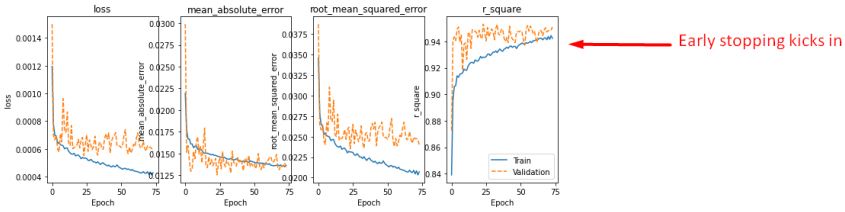
شکل 25: اضافه کردن **validation**

```
[47] model_4 = Sequential()

     model_4.add(Conv1D(64, kernel_size=3, activation='relu', padding='causal', input_shape=(20, 1) )) # (inputs.shape[1], inputs.shape[2])
     model_4.add(BatchNormalization())
     model_4.add(Conv1D(64, kernel_size=3, activation='relu', padding='causal'))
     model_4.add(BatchNormalization())
     model_4.add(Conv1D(32, kernel_size=3, activation='relu', padding='causal'))
     model_4.add(MaxPooling1D(pool_size=3))
     model_4.add(LSTM(100 , activation='relu', return_sequences=True))
     model_4.add(Dropout(0.2))
     model_4.add(LSTM(50  , activation='relu',))
     model_4.add(Dropout(0.3))
     model_4.add(Dense(1 , activation='relu'))

     model_4.compile(optimizer = tf.keras.optimizers.Adam(learning_rate=0.001, decay=0.0001 ) , loss = 'mse' ,
                     metrics =[
                               tf.keras.metrics.MeanAbsoluteError(),
                               tf.keras.metrics.RootMeanSquaredError(),
                               RSquare()
                              ])

     print( model_4.summary() )
     print('\n=================================================================================')

     # checkpoint     = tf.keras.callbacks.ModelCheckpoint("model.hdf5", monitor="val_loss", verbose=1, mode='min', save_best_only=True)
     early_stopping = tf.keras.callbacks.EarlyStopping  (patience=50, verbose=1, mode='min', restore_best_weights=True)

     start   = time()
     history_4 = model_4.fit(X_train_1_lag, Y_train_1_lag, epochs=200, batch_size=32, validation_data=(X_val_1_lag, Y_val_1_lag), verbose=1, callbacks=[early_stopping])

     print('\n=================================================================================\n time: ')
     print( time() - start )
```

شکل 26: مدل ۴

```
[47]  time:
      1177.6890740394592
```

```
[48] fig     = plt.figure(figsize=(12, 4))
     metrics = ['loss', 'mean_absolute_error', 'root_mean_squared_error', 'r_square']

     for i, metric in enumerate(metrics):
         plt.subplot(1, 4, i+1)
         plt.plot   (history_4.epoch, history_4.history[metric], label='Train')
         plt.plot   (history_4.epoch, history_4.history[f"val_{metric}"], linestyle="--", label='Validation')
         plt.xlabel ('Epoch')
         plt.ylabel (metric)
         plt.title  (metric)

     plt.legend()
     plt.show()
```




```
[49] model_4.evaluate(X_test_1_lag, Y_test_1_lag)

     220/220 [==============================] - 2s 9ms/step - loss: 4.0483e-04 - mean_absolute_error: 0.0118 - root_mean_squared_error: 0.0201 - r_square: 0.9536
     [0.0004048290720675140 6,
      0.011771468445658684,
      0.020120363682508 47,
      0.953607976436615]
```

شکل 27: ارزیابی مدل ۴

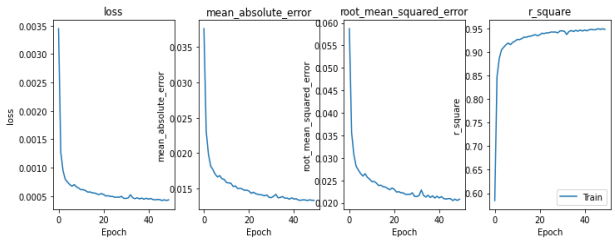
مدل ۵ و ۶ برای حالتی است که از داده 7 روز گذشته استفاده میکنیم و چون ۲۰ ستون داریم برای هر روز در این مدل ها ورودی 20*7=140 است که بزرگتر است از قبل و کندتر آموزش میبینند شبکه و چون ابعاد بالا رفته همچنین مشکل curse of dimensionality را داریم و دقت مدل بهبود پیدا نمیکند.

Model 5: Sequental model, Without validation data, using 7 lag, exteranl drop out

```
model_5 = Sequential()

model_5.add(Conv1D(64, kernel_size=3, activation='relu', padding='causal', input_shape=(140, 1) ))
model_5.add(BatchNormalization())
model_5.add(Conv1D(64, kernel_size=3, activation='relu', padding='causal'))
model_5.add(BatchNormalization())
model_5.add(Conv1D(32, kernel_size=3, activation='relu', padding='causal'))
model_5.add(MaxPooling1D(pool_size=3))
model_5.add(LSTM(100 , activation='relu', return_sequences=True))
model_5.add(Dropout(0.2))
model_5.add(LSTM(50  , activation='relu',))
model_5.add(Dropout(0.3))
model_5.add(Dense(1 , activation='relu'))

model_5.compile(optimizer = tf.keras.optimizers.Adam(learning_rate=0.001, decay=0.0001 ) , loss = 'mse' ,
              metrics =[
                        tf.keras.metrics.MeanAbsoluteError(),
                        tf.keras.metrics.RootMeanSquaredError(),
                        RSquare()
                       ])

print( model_5.summary() )
print('\n=================================================================================')

# checkpoint     = tf.keras.callbacks.ModelCheckpoint("model.hdf5", monitor="val_loss", verbose=1, mode='min', save_best_only=True)
early_stopping = tf.keras.callbacks.EarlyStopping  (patience=50, min_delta=1e-3, verbose=1, mode='min', restore_best_weights=True)

start   = time()
history_5 = model_5.fit(X_train_7_lag, Y_train_7_lag, epochs=200, batch_size=32, verbose=1, callbacks=[early_stopping])

print('\n=================================================================================\n time: ')
print( time() - start )
```

شکل 28: مدل ۵

```
[41] fig     = plt.figure(figsize=(12, 4))
     metrics = ['loss', 'mean_absolute_error', 'root_mean_squared_error', 'r_square']

     for i, metric in enumerate(metrics):
         plt.subplot(1, 4, i+1)
         plt.plot   (history_5.epoch, history_5.history[metric], label='Train')
         #plt.plot   (history_5.epoch, history_5.history[f"val_{metric}"], linestyle="--", label='Validation')
         plt.xlabel ('Epoch')
         plt.ylabel (metric)
         plt.title  (metric)

     plt.legend()
     plt.show()
```




```
[42] model_5.evaluate(X_test_7_lag, Y_test_7_lag)

     220/220 [==============================] - 9s 36ms/step - loss: 4.3188e-04 - mean_absolute_error: 0.0140 - root_mean_squared_error: 0.0208 - r_square: 0.9505
     [0.0004318786377552897,
      0.013958108611404896,
      0.020781690254807472,
      0.9505082368850708]
```

شکل 29: ارزیابی مدل ۵

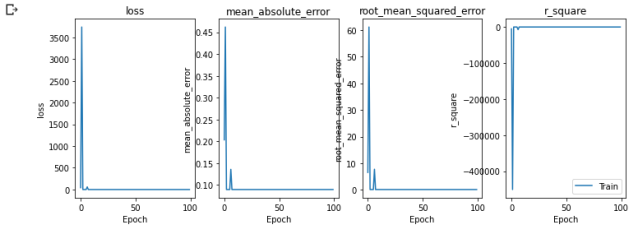
Model 6: Sequental model, Without validation data, using 7 lag, internal drop out

```
[ ]  model_6 = Sequential()

     model_6.add(Conv1D(64, kernel_size=3, activation='relu', padding='causal', input_shape=(140, 1) ))
     model_6.add(BatchNormalization())
     model_6.add(Conv1D(64, kernel_size=3, activation='relu', padding='causal'))
     model_6.add(BatchNormalization())
     model_6.add(Conv1D(32, kernel_size=3, activation='relu', padding='causal'))
     model_6.add(MaxPooling1D(pool_size=3))
     model_6.add(LSTM(100 , activation='relu', return_sequences=True,  recurrent_dropout = 0.2))
     model_6.add(LSTM(50  , activation='relu',  recurrent_dropout = 0.3))
     model_6.add(Dense(1 , activation='relu'))

     model_6.compile(optimizer = tf.keras.optimizers.Adam(learning_rate=0.001, decay=0.0001 ) , loss = 'mse' ,
                     metrics =[
                               tf.keras.metrics.MeanAbsoluteError(),
                               tf.keras.metrics.RootMeanSquaredError(),
                               RSquare()
                              ])

     print( model_6.summary() )
     print('\n=================================================================================')

     # checkpoint     = tf.keras.callbacks.ModelCheckpoint("model.hdf5", monitor="val_loss", verbose=1, mode='min', save_best_only=True)
     early_stopping = tf.keras.callbacks.EarlyStopping  (patience=50, min_delta=1e-3, verbose=1, mode='min', restore_best_weights=True)

     start   = time()
     history_6 = model_6.fit(X_train_7_lag, Y_train_7_lag, epochs=200, batch_size=32, verbose=1, callbacks=[early_stopping])

     print('\n=================================================================================\n time: ')
     print( time() - start )
```

شکل 30: مدل ۶

```
[35]  time:
      16175.417157649994
```

```
fig     = plt.figure(figsize=(12, 4))
metrics = ['loss', 'mean_absolute_error', 'root_mean_squared_error', 'r_square']

for i, metric in enumerate(metrics):
    plt.subplot(1, 4, i+1)
    plt.plot   (history_6.epoch, history_6.history[metric], label='Train')
    #plt.plot   (history_6.epoch, history_6.history[f"val_{metric}"], linestyle="--", label='Validation')
    plt.xlabel ('Epoch')
    plt.ylabel (metric)
    plt.title  (metric)

plt.legend()
plt.show()
```




```
[37] model_6.evaluate(X_test_7_lag, Y_test_7_lag)

220/220 [==============================] - 9s 35ms/step - loss: 0.0161 - mean_absolute_error: 0.0856 - root_mean_squared_error: 0.1267 - r_square: -0.8404
[0.0160596817731857 3,
 0.0856353566050529 5,
```

شکل 31: ارزیابی مدل ۶

تمام بخش‌های سوال انجام داده شد و در اینجا سوال ۱ به پایان میرسد.

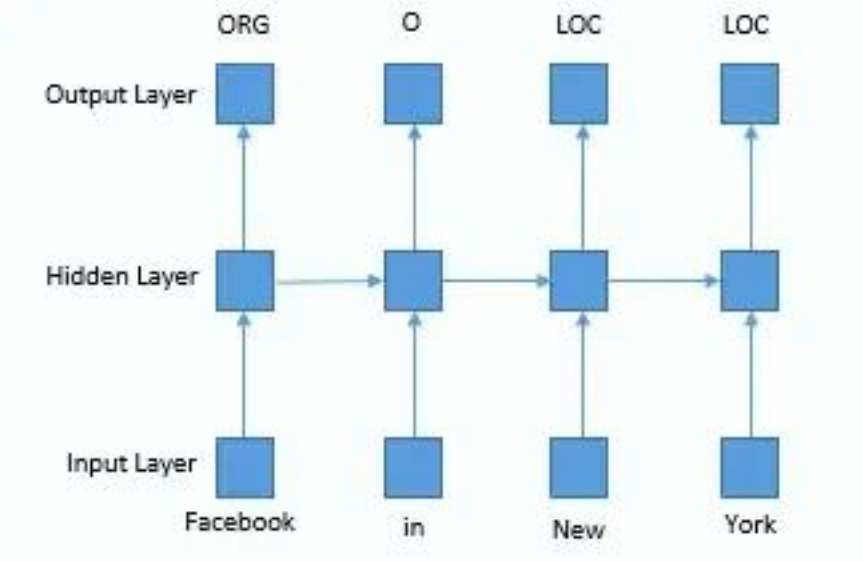
پاسخ ۲ – تشخیص اخبار جعلی

لینک کولب:

NN_HW4_Q2.ipynb - Colaboratory (google.com)

۲-۱. توضیحات مدل‌ها

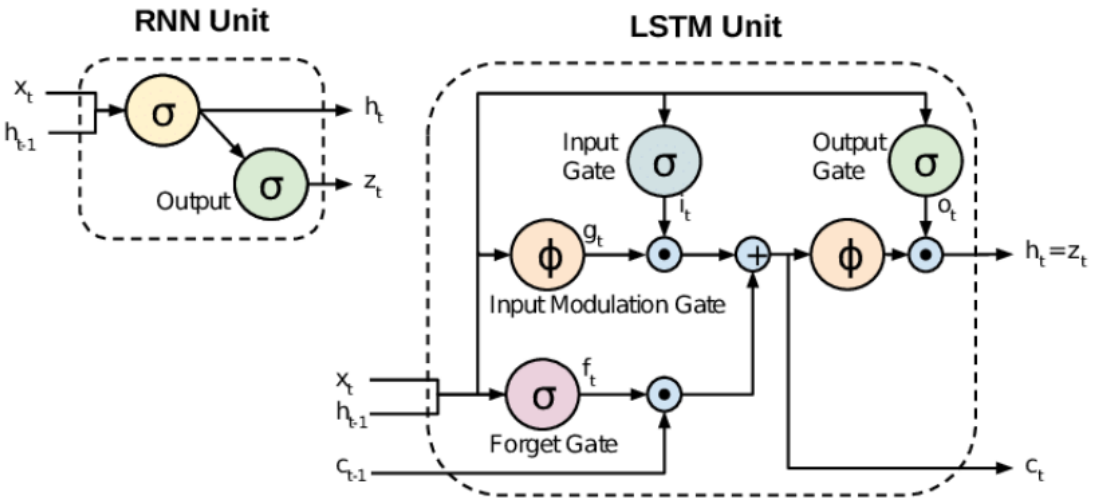
شبکه‌ی RNN همانطور که از نام آن پیدا است یکسری پردازش بازگشتی بر روی داده‌های ورودی، جهت یادگیری انجام می‌دهد. این پردازش بازگشتی باعث می‌شود تا RNN هنگام پردازش یک دنباله، حافظه‌ای از آن چه قبل از دنباله‌ی فعلی آمده است داشته باشد. این کار با به خاطر سپردن خروجی هر مرحله انجام می‌شود. در هر مرحله RNN اطلاعات مربوط به ویژگی‌های فعلی را ذخیره می‌کند و در مراحل بعدی از آن استفاده می‌کند. این کار باعث می‌شود تا RNN بتواند خروجی فعلی را با توجه به ویژگی‌های long distance relation ها را تشخیص دهد. مشکل RNN این است که اطلاعات را فقط به خاطر میسپارد و فراموش نمیکند و ساختار داخل آن بسیار ساده است به طوری که در هر unit حتما چیزی را به خاطر میسپارد این مشکل‌ها در شبکه‌های پیشرفته تر LSTM و GRU حل شده اند و هم روی اینکه چه چیزی را ذخیره کنیم کنترل داریم و هم اینکه چه چیزی را فراموش کنیم. مشکل دیگر RNN این است که هرچه Realtion ها long distance تر مشوند ضعیف‌تر عمل میکند که در LSTM و GRU با داشتن یک cell state این مشکل تا حدی حل شده است.



شکل 32: ساختار **RNN**

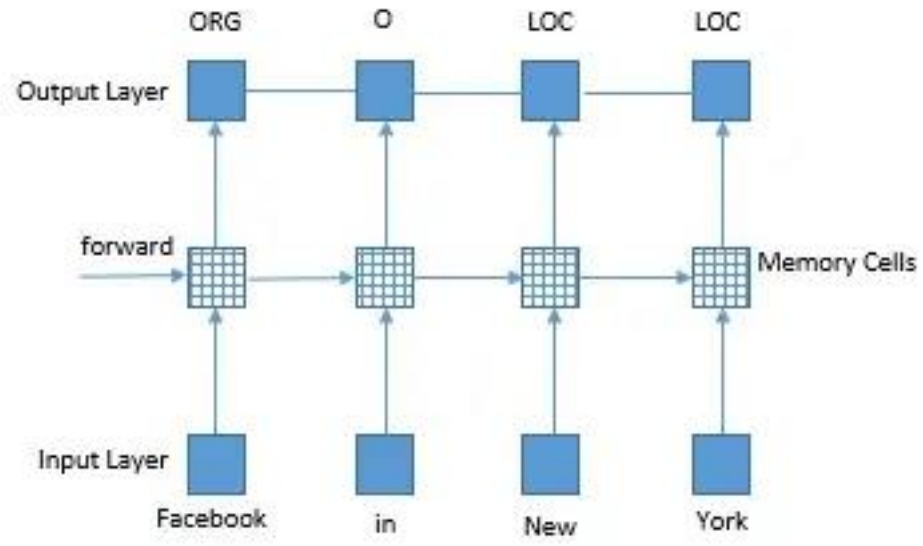
در پردازش زبان طبیعی چون هر کلمه به کلمه‌های قبلی مربوط است، مثل ضمیر و صفت و زمان فعل استفاده از شبکه RNN مفید است تا این روابط کشف شوند. در مسائل پردازش متن معمولا RNN برای

مسائلی خوب است که در آن به دنبال پیش‌بینی در سطح word-level هستیم برای مثال تشخیص نام، تشخیص بخشی از جمله و ... . اگرچه RNNها می‌توانند وابستگی‌های میان ویژگی‌های داده را یاد بگیرند اما این یادگیری فقط بر روی اطلاعات اخیر ورودی انجام می‌شود و در درازمدت ممکن است موثر نباشد. همچنین RNNها مشکل vanishing gradient نیز دارند. برای یادگیری وابستگی‌های درازمدت در ورودی، از LSTM استفاده می‌شود. ساختار LSTM شبیه به RNN است با این تفاوت که LSTM می‌تواند اطلاعات long-term را ذخیره کند.



شکل 33: تفاوت واحد **RNN** و **LSTM**

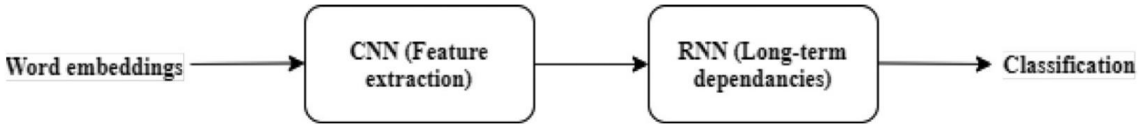
همانطور که مشاهده می‌شود در هر LSTM Unit، سه گیت input، output و forget وجود دارد که این گیت‌ها مشخص می‌کنند که اطلاعات چه موقع و به چه مقداری و برای چه مدتی در حافظه نگه‌داری شوند. گیت input مشخص می‌کند که کدام یک از ورودی‌ها برای نگه‌داری و تغییر حافظه باید استفاده شوند. گیت forget مشخص می‌کند که چه جزئیاتی می‌توانند نادیده گرفته شوند و نیازی به ذخیره‌ی آن‌ها نیست و گیت output نیز خروجی را بر اساس ورودی و اطلاعات موجود در حافظه، تولید می‌کند.



شکل 34: ساختار **LSTM**

بنابراین با قرار دادن واحدهای LSTM، می‌توان وابستگی‌های طولانی مدت را در ورودی پیدا کرد. در مسائل پردازش متن نیز زمانی که نیاز داشته باشیم تا مدل context متن ورودی را درک کند، مانند مسئله‌ی تشخیص نوع خبر، استفاده از LSTM مناسب است.

می‌دانیم که CNNها در استخراج ویژگی‌ها توانمند هستند و در متن هم یک کلمه بیشترین وابستگی را به کلمات نزدیک به خود دارد. در مدل Hybrid معرفی شده، از CNN برای استخراج ویژگی‌های محلی و از LSTM برای یادگیری وابستگی‌های درازمدت استفاده شده است. ابتدا به کمک CNN، ویژگی‌های محلی که در سطح text هستند از ورودی استخراج می‌شوند و سپس خروجی CNN، وارد RNN می‌شود. واحدهای حافظه در RNN، LSTM هستند. LSTM از ویژگی‌های استخراج شده توسط CNN استفاده می‌کند و وابستگی‌های درازمدت در ویژگی‌های محلی را یاد می‌گیرد و از آن برای تشخیص جعلی بودن یا نبودن خبر استفاده می‌کند.



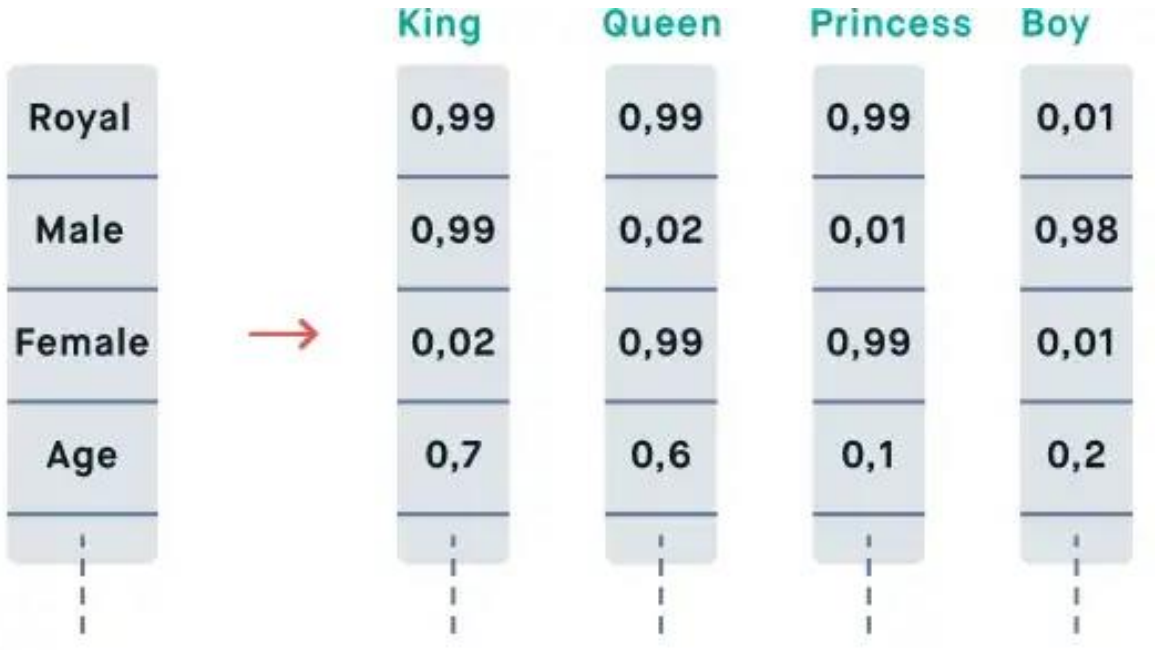
شکل 35: مدل **Hybrid** معرفی شده در مقاله

در مدل‌های RNN، تنها ویژگی‌های sequential ورودی استخراج می‌شوند و از آن‌ها استفاده می‌شود اما زمانی که از CNN نیز کمک گرفته می‌شود، علاوه بر استخراج این ویژگی‌ها، ویژگی‌های محلی نیز استخراج می‌شوند که باعث می‌شود در بعضی از مسائل بسیار مفید باشد برای مثال تشخیص sign

language از روی ویدئو. به کمک CNN ویژگی‌هایی که در تصویر وجود دارد استخراج می‌شود و به کمک RNN ویژگی‌های sequential.

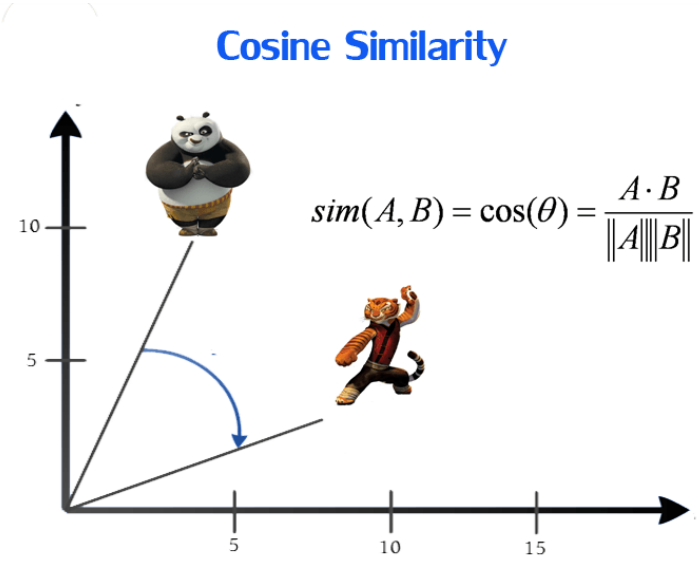
۲-۲. ورودی مدل

زمانی که با مسائل پردازش متن و داده‌های متنی سروکار داریم، برای آن که بتوانیم ورودی را به مدل‌ها بدهیم باید راهی پیدا کنیم تا داده‌های متنی را به بردارهای عددی تبدیل کنیم تا برای مدل قابل فهم باشد. کلمات موجود در یک متن را می‌توان به صورت بردارهایی نمایش داد که به آن‌ها word embeddings گفته می‌شود و هر کلمه یک بردار مخصوص به خود را دارد. هدف استفاده از word embeddings این است که مفهوم یک کلمه و همچنین ارتباط semantic و syntactic میان کلمات یک متن درک شود (در ابتدا روش‌های one hot مانند TFIDF را در NLP داشتیم سپس روش‌های مانند Glove و Word2vec روی کار آمدند) برای این کار لازم است تا کلماتی که از نظر معنایی با هم مرتبط هستند، بردار متناظر آن‌ها تا حد امکان به یکدیگر نزدیک باشد و کلماتی که از نظر معنایی متفاوتند، بردار متفاوتی نیز داشته باشند. این کار به کمک embedding matrix انجام می‌شود که یک ماتریس از اعداد بین 0 تا 1 است. زمانی که شباهت میان دو کلمه زیاد باشد درایه‌ی متناظر با آن دو کلمه در ماتریس نیز به 1 نزدیک‌تر است.



شکل 36: **Embedding Matrix**

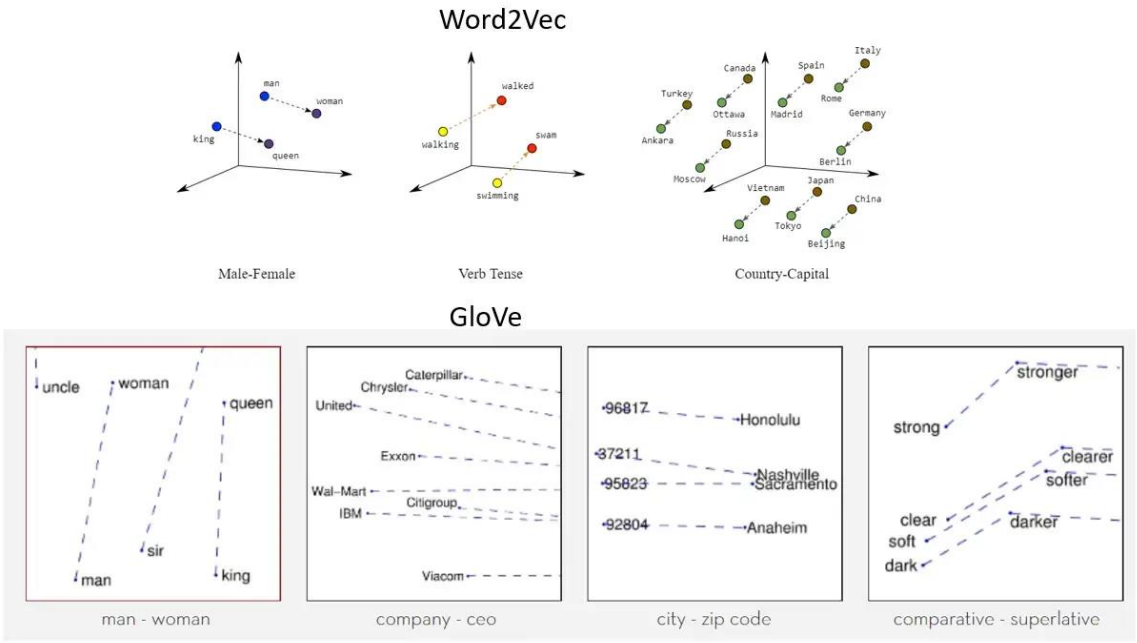
برای مثال دو کلمه‌ی King و Royal از نظر معنایی و context مشابهت دارند در نتیجه درایه‌ی متناظر آن‌ها در ماتریس نیز مقداری نزدیک به 1 دارد. میزان شباهت میان کلمات بر اساس یک معیار به نام Cosine Similarity محاسبه می‌شود.



شکل 37: **Cosine Similarity**

برای تهیه‌ی word embeddings روش‌های مختلفی وجود دارد که این روش‌ها نیازمند داشتن یک corpus متنی بزرگ هستند که روی آن‌ها پردازش انجام شده و اطلاعات مربوط به کلمات corpus استخراج شده و در نهایت بردارهای متناظر کلمات تولید می‌شود. از آن جایی که این کار نیازمند آموزش مدل و انجام محاسبات است، معمولا از word embeddings های آماده استفاده می‌شود که Word2Vec یکی از معروف‌ترین آن‌ها است که از شبکه‌ی عصبی جهت ساختن embedding استفاده می‌کند.

روشی که ما استفاده می‌کنیم Glove نام دارد. Glove در واقع نسخه‌ی بهبود یافته‌ی Word2Vec است که از یک الگوریتم unsupervised استفاده می‌کند. در Glove، بردارها بر اساس فرکانس تکرار کلمات در کنار یکدیگر در متن، ساخته می‌شوند. در این روش ابتدا یک ماتریس X ساخته می‌شود که ردیف‌های آن کلمات و ستون‌های آن context ها هستند. مقدار X_{ij} برای تعداد دفعاتی است که کلمه‌ی i در context j ظاهر شده است. با سعی در کاهش خطای reconstruction، این ماتریس در نهایت با فاکتورگیری به یک ماتریس در ابعاد پایین‌تر تبدیل می‌شود که در آن هر ردیف نشان دهنده‌ی بردار مرتبط با یک کلمه است. در Glove بر خلاف Word2Vec که از یک پنجره از همسایگی کلمات برای تعریف context استفاده می‌کند، از روش‌های آماری بر روی کل corpus برای تعریف context استفاده می‌شود.



شکل 38: مقایسه‌ی **Glove** و **Word2Vec**

۳-۲. پیاده‌سازی

۱-۳-۲. پیش پردازش‌ها

در ابتدا که داده را میخوانیم به صورت زیر است و قابل استفاده برای مدل شبکه عصبی نیست.

	unit_id	article_title	article_content	source	date	location	labels
0	1914947530	Syria attack symptoms consistent with nerve ag...	Wed 05 Apr 2017 Syria attack symptoms consiste...	nna	4/5/2017	idlib	0
1	1914947532	Homs governor says U.S. attack caused deaths b...	Fri 07 Apr 2017 at 0914 Homs governor says U.S...	nna	4/7/2017	homs	0
2	1914947533	Death toll from Aleppo bomb attack at least 112	Sun 16 Apr 2017 Death toll from Aleppo bomb at...	nna	4/16/2017	aleppo	0
3	1914947534	Aleppo bomb blast kills six Syrian state TV	Wed 19 Apr 2017 Aleppo bomb blast kills six Sy...	nna	4/19/2017	aleppo	0
4	1914947535	29 Syria Rebels Dead in Fighting for Key Alepp...	Sun 10 Jul 2016 29 Syria Rebels Dead in Fighti...	nna	7/10/2016	aleppo	0
...	...	...	...	...	...	...	...
799	1965511221	Turkish Bombardment Kills 20 Civilians in Syria	28-08-2016 Turkish Bombardment Kills 20 Civili...	manar	8/28/2016	aleppo	1
800	1965511222	Martyrs as Terrorists Shell Aleppos Salah Eddin	17-08-2016 Martyrs as Terrorists Shell Aleppos...	manar	8/1/2016	aleppo	1
801	1965511224	Chemical Attack Kills Five Syrians in Aleppo SANA	03-08-2016 Chemical Attack Kills Five Syrians ...	manar	8/3/2016	aleppo	0
802	1965511226	5 Killed as Russian Military Chopper Shot down...	01-08-2016 5 Killed as Russian Military Choppe...	manar	8/1/2016	idlib	1
803	1965511231	Syrian Army Kills 48 ISIL Terrorists in Deir E...	April 6 2017 Syrian Army Kills 48 ISIL Terrori...	manar	4/4/2017	deir ezzor	1

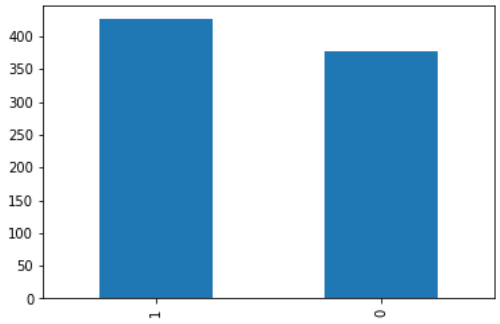
804 rows × 7 columns

شکل 39: داده خام

ولی خوشبختانه داده‌های بالانس هستند و در این زمینه کار اضافه ای لازم نیست.

```
[ ]  df['labels'].value_counts().plot.bar()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f9ba48631f0>
```



شکل 40: کلاس‌ها **balance** هستند

مقاله میگوید که داده را ابتدا با Regular expression ها متن را تمیز تر کنیم و سپس به داده‌های اموزش و تست تقسیم کنیم. ما اینکار را به شکل زیر انجام دادیم و تلاش کردیم که این مرحله را به دقت انجام دهیم تا کیفیت مدل در آینده بهتر شود.

```
ps = PorterStemmer()

for i in range(len(X)):
    text = X['article_title'][i].lower() + " " + X['article_content'][i].lower() + " " + X['source'][i] + " " + X['location'][i]

    text = re.sub('\d+',         ' INT '        , text)
    text = re.sub('!',           ' EXCLAMATION ', text)
    text = re.sub('\?',          ' QUESTION '   , text)
    text = re.sub('\\$|\\£|\\€', ' MONEY '      , text)
    text = re.sub('\+',          ' PLUS '       , text)
    text = re.sub('=',           ' EQUALL '     , text)
    text = re.sub('&',           ' AMPERSAND '  , text)
    text = re.sub('#',           ' TAG '        , text)
    text = re.sub('\.',          ''             , text)
    text = re.sub('%',           'PERCENT'      , text)

    text = re.sub('http://\S+|https://\S+', 'LINK'  , text)
    text = re.sub('http[s]?://\S+'        , 'LINK'  , text)
    text = re.sub(r"http\S+"              , "LINK"  , text)

    text = re.sub(r'(\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3})', 'IP', text)

    text = text.translate(str.maketrans('', '', string.punctuation))

    tokens = word_tokenize(text)

    filtered_text = [w for w in tokens if not w.lower() in stopwords.words('english')]

    filtered_text_stemmed = [ps.stem(word) for word in filtered_text]

    filtered_text_stemmed_joined = ' '.join(filtered_text_stemmed)

    X['article_content'][i] = filtered_text_stemmed_joined

X.drop(['article_title', 'source', 'location', 'unit_id', 'date'], axis=1, inplace=True)
X
```

شکل 41: **Stem** کردن داده‌ها و جاگذاری کاراکتر یا عبارت‌های نامتعارف با یک متن ثابت و تبدیل کردن کل متن به **lower case**

همانطور که دیده میشود از اعداد تا علامت‌ها و لینک‌ها و آی‌پی ها را پوشش داده ایم و سپس tokenization را انجام دادیم تا جمله به عبارات معنادار کوچتر تقسیم شود سپس با stemming قسمت

اضافه واژه را حذف میکنیم تا به ریشه کلمه برسیم. سعی شد از تمام ستون‌های که حاوی اطلاعات معناداری هستند استفاده شود. همچنین کلمات پرتکرار و بی معنا یا همان stop word هم حذف شده اند.

سپس با رعایت اندازه کلاس‌ها مدل را به داده اموزش و تست تقسیم میکنیم.

```
[ ] # Train-Test-Split
    X_train, X_test, y_train, y_test = train_test_split(X, Y, stratify=Y, test_size=0.2, random_state=4)
```

شکل 42: **train-test-split**

در قسمت بعدی تبدیل کردن این متن تمیز شده به vector هایی است که بتوان به مدل داد ولی در این قسمت مقاله ابهام حیاتی وجود دارد که به آن اشاره میکنیم. در متن زیر شیوه کار توضیح داده شده است که پس از pad کردن ما باید وکتورهای با سایز 300 داشته باشیم ولی وقتی به شکلی که مقاله برای ساختار مدل آورده است نگاه میکنیم میبینیم که در واقع این اندازه 100 است و اندازه embedding برابر 300 است و شک داریم که این دو مقدار جابه‌جا اعلام شده است که در زیر گفته خود را اثبات میکنیم. ترجیح ما اقدام بر اساس شکبه معرفی شده بود تا متن مقاله زیرا که نتایج نزدیکتری به مقاله را حاصل کرد.

4.2.2. Mapping text to vectors using word embeddings

The label matrices for training and test sets are encoded and the text is vectorized by tokenization using the Tokenizer function of the Keras library. The task is repeated separately for training and test texts in the two datasets. The tokenizer is fitted on the pre-processed training corpus which is converted to sequences of integers. The length of sequences is set to 300 and a post-padding is applied in order to use it for model training. This is done because the length of each sequence varies. By fixing the length of each text sequence to 300, it is necessary to append zeros (zero values) in each sequence that is shorter than the fixed length.

In order for the CNN to perform local feature extraction, pre-trained word embeddings are used. An embedding matrix is prepared using the GloVe pre-trained word embeddings.[12] GloVe was trained with a dataset of six billion words or tokens using a vocabulary of 400 thousand words, and provides embeddings in a range of dimensions. Word embeddings of 100 dimensions are used in this research. The embeddings' matrix is prepared using only the words that occur in the tokenized training corpus.

شکل 43: آنچه مقاله راجع به **embedding** گفته است.

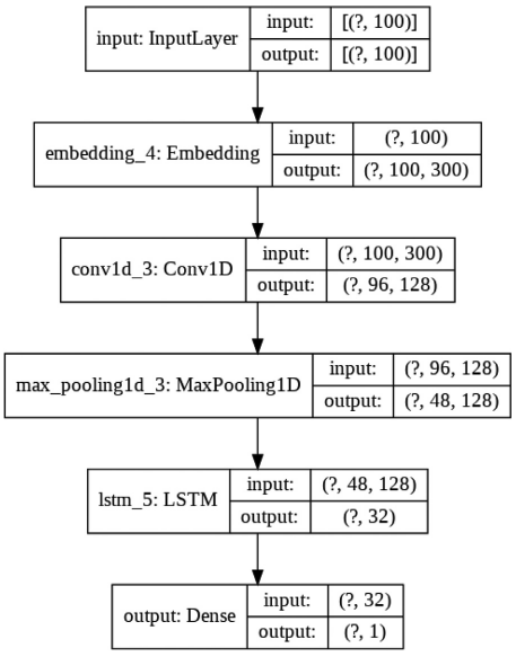


Fig. 4. FA-KES Model Summary.

شکل 44: شمای شبکه

همانطور که در تصویر بالا میبینید ورودی یعنی همان متن pad شده داری اندازه ۱۰۰ است و پس از embedding شدن سایز آن 300 شده است. پس بر این فرض ادامه پیش پردازش را جلو میبریم.

در ادامه متن را توکن بندی کرده و با انجام padding سایز همه ورودی‌ها را به اندازه ثابت ۱۰۰ میرسانیم.

```
[ ] # Tokenizer For Training Set
    # init the tokenizer with a out_of_vocabulary token
    tokenizer = Tokenizer(num_words=10000, oov_token="<OOV>")

    tokenizer.fit_on_texts( X_train['article_content']  )

    # vectorization
    sequences_train = tokenizer.texts_to_sequences( X_train['article_content'] )
    # Padding
    sequences_padded_train = pad_sequences(sequences_train, maxlen=100, padding='post')

    word_index_train = tokenizer.word_index
```

```
[ ] # First article verctorized
    print( sequences_train[0] )

    [117, 51, 40, 100, 291, 94, 339, 233, 2, 2, 117, 34, 382, 1442, 647, 51, 40, 3177, 1738, 64, 30,
```

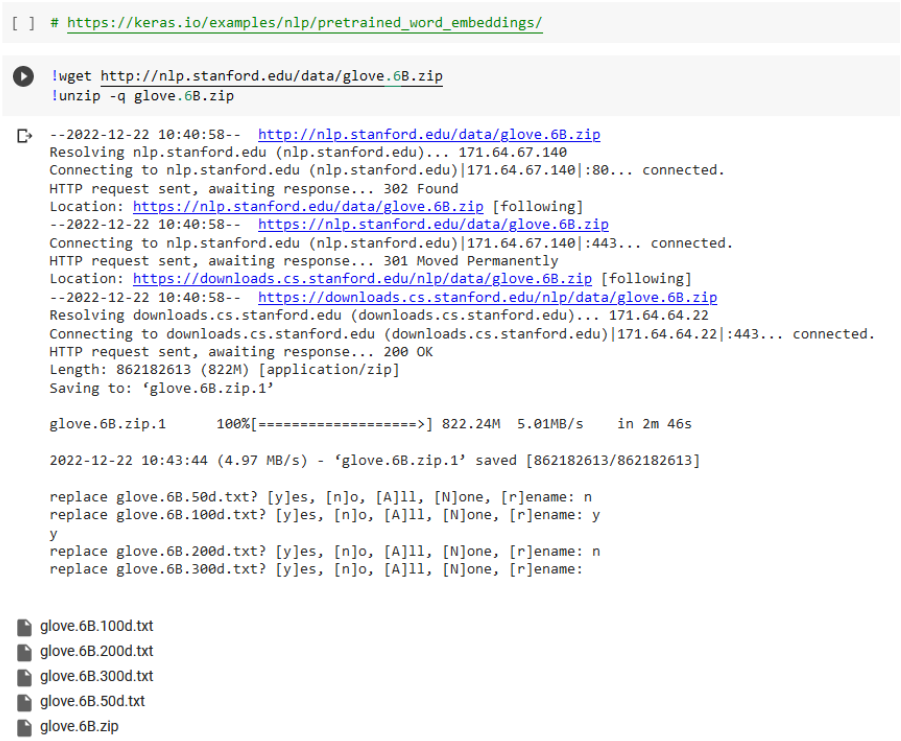
```
[ ] # Intgers that represent words. Sorted by frequncy.
    print( word_index_train )

    {'<OOV>': 1, 'int': 2, 'kill': 3, 'syrian': 4, 'said': 5, 'syria': 6, 'aleppo': 7, 'attack': 8,
```

```
[ ] sequences_padded_train.shape

    (643, 300)
```

شکل 45: **tokenization and padding**

چون که tokenizer به متن fit میشود پس لازم است جدا برای داده اموزش و تست این کار را انجام دهیم.

طبق وبسایت خود keras پیش میرویم و embedding ها را دانلود میکنیم و سپس ماتریس embedding را میسازیم.



شکل 46: دانلود **Embedding** ها

```
[ ] path_to_glove_file = os.path.join(
        os.path.expanduser("glove.6B.300d.txt")
    )

    embeddings_index = {}
    with open(path_to_glove_file) as f:
        for line in f:
            word, coefs = line.split(maxsplit=1)
            coefs = np.fromstring(coefs, "f", sep=" ")
            embeddings_index[word] = coefs

    print("Found %s word vectors." % len(embeddings_index))
```

```
Found 400000 word vectors.
```

```
[ ] num_tokens = len(word_index_train) + 2
    embedding_dim = 300
    hits = 0
    misses = 0

    # Prepare embedding matrix
    embedding_matrix = np.zeros((num_tokens, embedding_dim))
    for word, i in word_index_train.items():
        embedding_vector = embeddings_index.get(word)
        if embedding_vector is not None:
            # Words not found in embedding index will be all-zeros.
            # This includes the representation for "padding" and "OOV"
            embedding_matrix[i] = embedding_vector
            hits += 1
        else:
            misses += 1
    print("Converted %d words (%d misses)" % (hits, misses))
```

```
Converted 3874 words (3146 misses)
```

شکل 47: ساختن ماتریس **Embedding**

در اینجا پیش پردازش‌ها تمام میشود و به سراغ آموزش مدل میرویم.

۲-۳-۲. آموزش مدل‌ها

ابتدا نگاهی به جزئیات گفته شده در مقاله میکنیم.

4.2.3. Model implementation in Keras

The proposed hybrid deep learning model is implemented using the Sequential model of the Keras deep learning Python library. The Sequential model comprises several layers of neurons:

- The first layer of the neural network is the Keras embedding layer. This is the input layer through which the pre-trained word embeddings are utilized by providing the prepared embedding matrix and the model is trained by feeding in the training data.
- The next layer is the one-dimensional CNN layer (Conv1D) for extraction of local features by using 128 filters of size 5. The default Rectified Linear Unit (ReLU) activation function is used.
- After that, the large feature vectors generated by CNN are pooled by feeding them in to a MaxPooling1D layer with a window size of 2, in order to down-sample the feature vectors, reduce the amount of parameters, and consequently the computations without affecting the network's efficiency.

- The pooled feature maps are fed into the RNN (LSTM) layer that follows. This input is used to train the LSTM, which outputs the long-term dependent features of the input feature maps, while retaining a memory. The dimension of the output is set to 32. The default linear activation function (i.e. $f(x) = x$) of Keras is used in this layer.
- Finally, the trained feature vectors are classified using a Dense layer that shrinks the output space dimension to 1, which corresponds to the classification label (i.e. fake or not fake). This layer applies the Sigmoid activation function.

The model is trained using the adaptive moment estimation (Adam) optimizer to define the learning rate in each iteration, the binary cross-entropy as the loss function, and the accuracy for the evaluation of results. The training is performed for 10 epochs using a batch size of 64.

Model summaries of FA-KES and ISOT datasets are shown in Figs. 4 and 5 respectively.

شکل 48: متن مقاله راجع به جزئیات شبکه

توابعی برای محاسبه معیارهای گفته شده در مقاله مینویسم:

```python
from keras import backend as K

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_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()))
```

شکل 49: توابع مربوط به معیارهای استفاده شده در ارزیابی مدل‌ها

سپس لایه embedding را مینویسم که بین هر دو شبکه مشترک است:

```
[ ]  from tensorflow.keras.layers import Embedding
     from tensorflow.keras import layers

     # Turns positive integers (indexes) into dense vectors of fixed size.
     embedding_layer = Embedding(
         input_dim = len(word_index_train) + 2,  # Size of the vocabulary
         output_dim = 300, # Dimension of the dense embedding
         embeddings_initializer = keras.initializers.Constant(embedding_matrix),
         trainable = False,
         input_length = 100 # Length of input sequences
     )
```

شکل 50: لایه **embedding**

حال شبکه Hybrid و سپس شبکه LSTM را میسازیم.

Hybrid Model (CNN-LSTM)

```
[ ]  model = Sequential()
     model.add(embedding_layer)
     model.add(Conv1D(128, 5, activation='relu'))
     model.add(MaxPooling1D(2))
     model.add(LSTM(32, activation="linear", dropout=0.1, recurrent_dropout=0.0))
     model.add(Dense(1, activation='sigmoid'))

     model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy', f1_m, precision_m, recall_m])

     print( model.summary() )
     print('\n=================================================================================')

     start   = time()
     history = model.fit(sequences_padded_train, y_train, epochs=10, batch_size=64, verbose=1)
     print('\n=================================================================================\n time: ')

     print( time() - start )
```

```
Model: "sequential_44"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_10 (Embedding)    (None, 100, 300)          2106600

 conv1d_41 (Conv1D)          (None, 96, 128)           192128

 max_pooling1d_52 (MaxPoolin  (None, 48, 128)          0
 g1D)

 lstm_52 (LSTM)              (None, 32)                20608

 dense_52 (Dense)            (None, 1)                 33

=================================================================
Total params: 2,319,369
Trainable params: 212,769
Non-trainable params: 2,106,600
_________________________________________________________________
None
```

شکل 51: ساختار شبکه ترکیبی

اگر خروجی هر لایه که در Summery شبکه آمده است را نگاه کنیم دقیقا با آنچه مقاله گفته منطبق است.

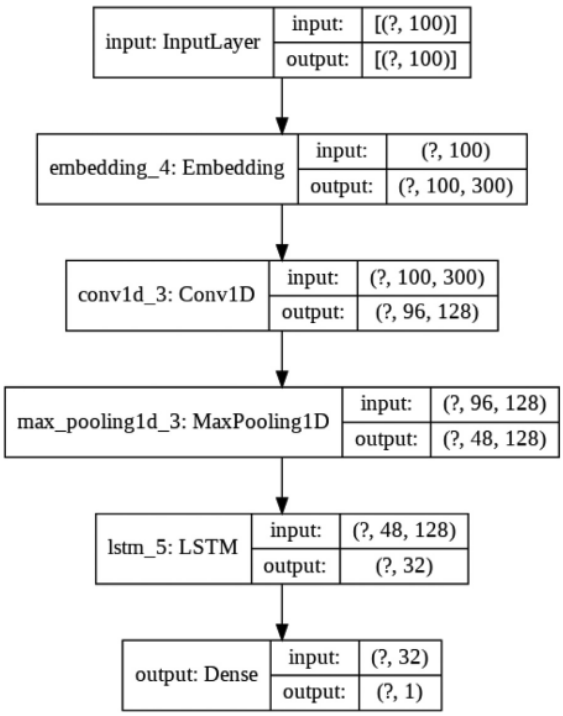


Fig. 4. FA-KES Model Summary.

شکل 52: شمای شبکه‌ی ترکیبی در مقاله

▾ **RNN Model (LSTM)**

```
[ ] model_2 = Sequential()
    model_2.add(embedding_layer)
    model_2.add(MaxPooling1D(3))
    model_2.add(LSTM(32, activation="linear"))
    model_2.add(Dense(1, activation='sigmoid'))

    model_2.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy', f1_m, precision_m, recall_m])

    print( model_2.summary() )
    print('\n=================================================================================')

    start   = time()
    history_2 = model_2.fit(sequences_padded_train, y_train, epochs=10, batch_size=64, verbose=1)
    print('\n=================================================================================\n time: ')

    print( time() - start )
```

```
Model: "sequential_32"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_10 (Embedding)    (None, 100, 300)          2106600

 max_pooling1d_40 (MaxPoolin  (None, 33, 300)          0
 g1D)

 lstm_40 (LSTM)              (None, 32)                42624

 dense_40 (Dense)            (None, 1)                 33

=================================================================
Total params: 2,149,257
Trainable params: 42,657
Non-trainable params: 2,106,600
```

شکل 53: ساختار شبکه **LSTM**

شبکه‌های بالا را آموزش میدهیم و نتایج آن را در بخش بعدی بحث میکنیم.

۴-۲. تحلیل نتایج

ابتدا به نتایجی که مقاله به دست آورده است نگاهی میاندازیم.

Table 5
Results of all models on the FA-KES dataset.

Dataset	Accuracy	Precision	Recall	F_1 score
LR	0.49 ± 0.008	0.50	0.49	0.49
RF	0.53 ± 0.009	0.56	0.53	0.54
MNB	0.38 ± 0.008	0.39	0.38	0.32
SGD	0.47 ± 0.009	0.49	0.47	0.48
KNNs	0.57 ± 0.008	0.58	0.57	0.57
DT	0.55 ± 0.006	0.56	0.55	0.55
AB	0.47 ± 0.005	0.49	0.47	0.47
(Elhadad et al., 2019)	0.58	0.63	0.58	0.50
CNN only	0.50 ± 0.006	0.55	0.50	0.48
RNN only	0.50 ± 0.007	0.51	0.50	0.50
Hybrid CNN-RNN	0.60 ± 0.007	0.59	0.60	0.59

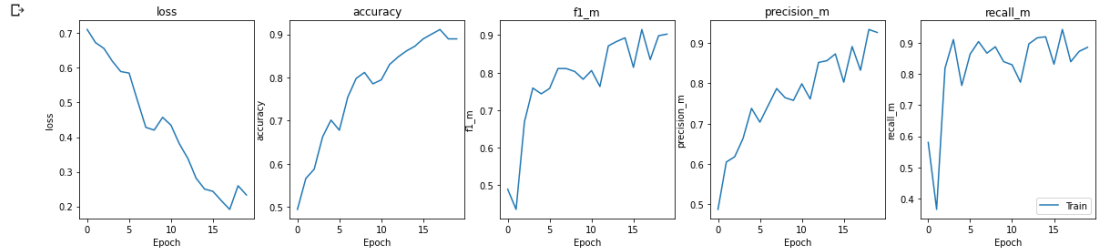
شکل 54: نتایج مقاله

نتایج شبکه ترکیبی:

```
================================================================================
Epoch 1/10
11/11 [==============================] - 3s 89ms/step - loss: 0.7209 - accuracy: 0.5008 - f1_m: 0.5004 - precision_m: 0.5209 - recall_m: 0.5792
Epoch 2/10
11/11 [==============================] - 1s 91ms/step - loss: 0.6372 - accuracy: 0.7030 - f1_m: 0.6616 - precision_m: 0.6395 - recall_m: 0.6961
Epoch 3/10
11/11 [==============================] - 2s 150ms/step - loss: 0.6102 - accuracy: 0.6874 - f1_m: 0.6793 - precision_m: 0.7179 - recall_m: 0.6717
Epoch 4/10
11/11 [==============================] - 2s 161ms/step - loss: 0.5584 - accuracy: 0.7465 - f1_m: 0.7735 - precision_m: 0.7282 - recall_m: 0.8368
Epoch 5/10
11/11 [==============================] - 2s 156ms/step - loss: 0.4828 - accuracy: 0.7932 - f1_m: 0.7423 - precision_m: 0.7041 - recall_m: 0.7936
Epoch 6/10
11/11 [==============================] - 2s 157ms/step - loss: 0.4584 - accuracy: 0.8118 - f1_m: 0.8160 - precision_m: 0.8127 - recall_m: 0.8519
Epoch 7/10
11/11 [==============================] - 2s 145ms/step - loss: 0.4450 - accuracy: 0.7885 - f1_m: 0.7380 - precision_m: 0.7181 - recall_m: 0.7875
Epoch 8/10
11/11 [==============================] - 2s 156ms/step - loss: 0.3950 - accuracy: 0.8243 - f1_m: 0.8328 - precision_m: 0.7655 - recall_m: 0.9308
Epoch 9/10
11/11 [==============================] - 2s 165ms/step - loss: 0.3822 - accuracy: 0.8289 - f1_m: 0.7679 - precision_m: 0.7343 - recall_m: 0.8105
Epoch 10/10
11/11 [==============================] - 1s 88ms/step - loss: 0.3607 - accuracy: 0.8491 - f1_m: 0.8696 - precision_m: 0.8524 - recall_m: 0.8921

================================================================================
 time:
17.525014638900757

[ ] model.evaluate(sequences_padded_test, y_test)

6/6 [==============================] - 0s 17ms/step - loss: 0.9608 - accuracy: 0.5714 - f1_m: 0.7376 - precision_m: 0.6304 - recall_m: 0.9226
[0.9608155488967896,
 0.5714285969734192,
 0.7375602722167969,
 0.6303901076316833,
 0.9226035475730896]
```

شکل 55: دقت روی داده آموزش و تست مدل ترکیبی

```
fig     = plt.figure(figsize=(20, 4))
metrics = ['loss', 'accuracy', 'f1_m', 'precision_m', 'recall_m']

for i, metric in enumerate(metrics):
    plt.subplot(1, 5, i+1)
    plt.plot   (history.epoch, history.history[metric], label='Train')
    plt.xlabel ('Epoch')
    plt.ylabel (metric)
    plt.title  (metric)

plt.legend()
plt.show()
```

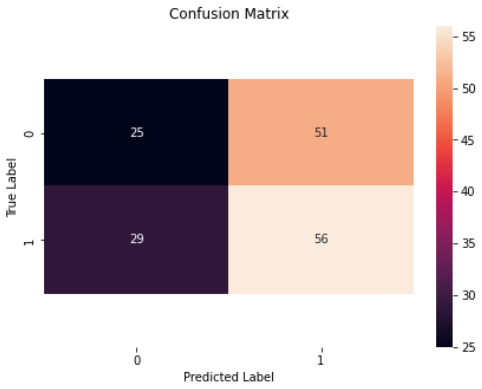


شکل 56: نمودار معیارهای خواسته شده

```
def plot_cm(y_true, preds):
    cm = confusion_matrix(y_true, preds)
    plt.figure(figsize=(7, 5))
    ax = sns.heatmap(cm, annot=True, fmt="d")
    bottom, top = ax.get_ylim()
    ax.set_ylim(bottom + 0.5, top - 0.5)
    plt.title('Confusion Matrix')
    plt.ylabel('True Label')
    plt.xlabel('Predicted Label')
    plt.show()
```

```
test_preds = np.round(model.predict(sequences_padded_test))

plot_cm(y_test, test_preds)
```

```
6/6 [==============================] - 0s 17ms/step
```



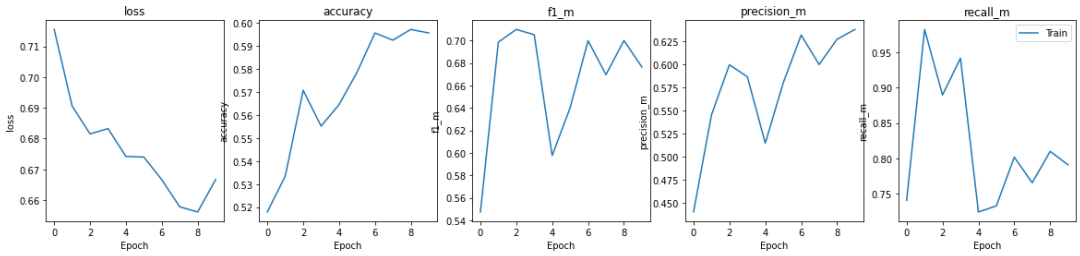
شکل 57: ماتریس **confusion**

نتایج شبکه LSTM:

```
[ ] ================================================================================
    Epoch 1/10
    11/11 [==============================] - 2s 22ms/step - loss: 0.7155 - accuracy: 0.5179 - f1_m: 0.5474 - precision_m: 0.4400 - recall_m: 0.7407
    Epoch 2/10
    11/11 [==============================] - 0s 22ms/step - loss: 0.6906 - accuracy: 0.5334 - f1_m: 0.6986 - precision_m: 0.5451 - recall_m: 0.9825
    Epoch 3/10
    11/11 [==============================] - 0s 22ms/step - loss: 0.6815 - accuracy: 0.5708 - f1_m: 0.7099 - precision_m: 0.5994 - recall_m: 0.8897
    Epoch 4/10
    11/11 [==============================] - 0s 23ms/step - loss: 0.6832 - accuracy: 0.5552 - f1_m: 0.7052 - precision_m: 0.5864 - recall_m: 0.9418
    Epoch 5/10
    11/11 [==============================] - 0s 22ms/step - loss: 0.6742 - accuracy: 0.5645 - f1_m: 0.5976 - precision_m: 0.5145 - recall_m: 0.7242
    Epoch 6/10
    11/11 [==============================] - 0s 22ms/step - loss: 0.6740 - accuracy: 0.5785 - f1_m: 0.6402 - precision_m: 0.5798 - recall_m: 0.7327
    Epoch 7/10
    11/11 [==============================] - 0s 24ms/step - loss: 0.6666 - accuracy: 0.5956 - f1_m: 0.6998 - precision_m: 0.6315 - recall_m: 0.8017
    Epoch 8/10
    11/11 [==============================] - 0s 23ms/step - loss: 0.6578 - accuracy: 0.5925 - f1_m: 0.6696 - precision_m: 0.5996 - recall_m: 0.7655
    Epoch 9/10
    11/11 [==============================] - 0s 23ms/step - loss: 0.6561 - accuracy: 0.5972 - f1_m: 0.7000 - precision_m: 0.6270 - recall_m: 0.8099
    Epoch 10/10
    11/11 [==============================] - 0s 22ms/step - loss: 0.6666 - accuracy: 0.5956 - f1_m: 0.6764 - precision_m: 0.6376 - recall_m: 0.7908

    ================================================================================
     time:
    3.995719432830810 5

[ ] model_2.evaluate(sequences_padded_test, y_test)

    6/6 [==============================] - 0s 6ms/step - loss: 0.6876 - accuracy: 0.5466 - f1_m: 0.5178 - precision_m: 0.4667 - recall_m: 0.5910
    [0.687576949596405,
     0.546583831102722,
     0.5177984237670898,
     0.46666666865348816,
     0.59102863073349]
```

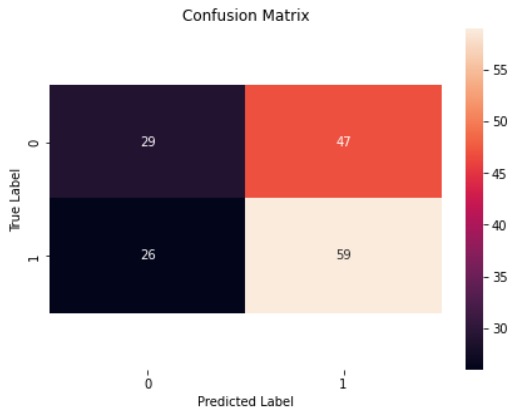
شکل 58: نتایج شبکه **LSTM** روی داده‌های آموزش و تست

```
[ ] fig     = plt.figure(figsize=(20, 4))
    metrics = ['loss', 'accuracy', 'f1_m', 'precision_m', 'recall_m']

    for i, metric in enumerate(metrics):
        plt.subplot(1, 5, i+1)
        plt.plot   (history_2.epoch, history_2.history[metric], label='Train')
        plt.xlabel ('Epoch')
        plt.ylabel (metric)
        plt.title  (metric)

    plt.legend()
    plt.show()
```



شکل 59: نمودار معیارهای خواسته شده برای شبکه **LSTM**

```
test_preds_2 = np.round(model_2.predict(sequences_padded_test))

plot_cm(y_test, test_preds_2)

6/6 [==============================] - 0s 6ms/step
```



شکل 60: ماتریس **confusion**

حال که تمام نتایج را نشان دادیم مقدار راجع به آنها بحث مکنیم.

این دو شبکه بسیار کم آموزش میبینند و به همین دلیل به مقدار اولیه بسیار حساس هستند و داده آموزش و تست هم کم است و نتایج چندان قابل اعتماد نیست. چون داده کم است نمیتوان زیاد مدل را آموزش داد زیرا که بیش برازش میشود.

مدل ترکیبی به دلیل داشتن شبکه کانولوشنی در خود توانایی استخراج ویژگی‌های محلی بیشتری را داشته و به چند کلمه بعدی هم توجه میکند در صورتی که LSTM تنها به گذشته توجه میکند و این میتواند عامل برتری مدل ترکیبی باشد.

همچنین مدل ترکیبی در نمودارها با روند ثابتی رشد میکند ولی برای مدل LSTM بسیار پر نوسان هستند نمودارها. این اتفاق نشان میدهد که LSTM برای استخراج ویژگی به خوبی CNN نمیتواند عمل کند.

اگر به ماتریس کانفیوژن نگاه کنیم میبینیم که مدل‌ها اکثر داده‌های تست را کلاس یک (که کلاس غالب تر است) تشخیص داده‌اند.

برای بهبود دقت میتوان از شبکه‌های پیچیده‌تر یا داده‌های بیشتر استفاده کرد. همچنین استفاده از embedding ها و توکنایزرهایی که روی تسک‌های مشابه اموزش دیده باشند هم مفید است.

مشکل دیگر مدل این است که فقط تعداد محدودی از واژه‌ها را میبیند که برای تصمیم گیری کافی نیست.

مشکل دیگر هر دول مدل استفاده فقط از واژه‌هایی که در گذشته آمده اند است. اگر مانند ترانسفورمرها از واژهای آینده هم استفاده میکردند دقت بالاتر داشتند. مدل Bi-LSTM هم احتمالا دقت بهتری داشته باشد تا روش LSTM خالی.

در هر دو به مقدار کمی آموزش دیده‌اند و بهتر بود تعداد اپیاک‌ها بیشتر باشد و شبکه اندکی عمیقتر و باریکتر تا بتوان ویژگی‌های پیچیده تری را استخراج کرد.

احتمالا استفاده از دو لایه کانولوشنی هم میتوانستم مفید باشد و ویژگی‌های بیشتر استخراج شوند.

اگر از مدل‌هایی که روی تسک‌های مشابه آموزش دیده‌اند به صورت zero shot استفاده میکردیم به نتایج قابل اعتمادتری میرسیدیم و احتمالا دقت هم بهتر میبود.

چیزی که واضح است دقت اموزش و تست در هر دو مدل با هم فاصله زیادی دارند که حاکی از بیش برازش شدن مدل است.

در اینجا تمام بخش‌های سوال دوم تکمیل میشود و سوال دوم هم به پایان میرسد.