به نام خدا

دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

درس شبکه‌های عصبی و یادگیری عمیق

تمرین چهارم

نام و نام خانوادگی	محمد ناصری – مریم عباس‌زاده
شماره دانشجویی	810100406 - 810100486
تاریخ ارسال گزارش	1401.10.02

فهرست

شکل‌ها

جدول‌ها

پاسخ 1. تخمین آلودگی هوا

۱-۱. سوالات تشریحی

Linear interpolation method.1-1-1 :

یکی از روش‌های ساده برای جایگزین کردن missing values است. از این تکنیک ساده، برای تخمین زدن مقادیر nan ای که بین مقادیر موجود قرار دارند استفاده می‌شود. این مفهوک بر این استوار است که نرخ تغییر بین مقادیر شناخته شده ثابت است و می‌توان با استفاده از یک فرمول شیب ساده از این مقادیر محاسبه کرد. سپس با استفاده از یکی از نقاط و نرخ تغییر می‌توان مقدار مجهول بین دو نقطه شناخته شده را محاسبه کرد. این روش برای داده های سری زمانی نسبت به روش های دیگر مناسب تر است. فرمول Linear interpolation به صورت زیر محاسبه می‌شود:

$$\text{Linear Interpolation}(y) = y_1 + (x - x_1)\frac{y_2 - y_1}{x_2 - x_1}$$

Pearson correlation .1-1-2

ضریب همبستگی پیرسون (r) رایج ترین روش برای اندازه گیری همبستگی خطی است .عددی بین 1- و 1 است که قدرت و جهت رابطه بین دو متغیر را اندازه گیری می کند. این ضریب دو بخش دارد: مقدار عددی و علامت. مقدار عددی نشان می‌دهد چقدر رابطه خطی بین دو متغیر قدرتمند است. علامت نشان می‌دهد جهت این رابطه مثبت است یا منفی. اگر ضریب همبستگی مثبت باشد، به این مفهوم است که افزایش در مقادیر یک متغیر با افزایش در مقادیر متغیر دیگر همراه است. همین‌طور کاهش در مقادیر یک متغیر با کاهش در مقادیر متغیر دیگر همراه است. در این حالت اگر نمودار پراکندگی دو متغیر رسم شود، می‌توان خطی با شیب مثبت را از بین نقاط برازش داد. به همین ترتیب اگر ضریب همبستگی منفی باشد، می‌توان خطی با شیب منفی را از بین نقاط برازش داد. هرچه مقدار مطلق ضریب همبستگی (صرف‌نظر از علامت) به ۱ نزدیک باشد، نشان می‌دهد شدت رابطه خطی بین دو متغیر قوی‌تر است. در مقابل ضریب همبستگی نزدیک صفر نشان می‌دهد که رابطه خطی بسیار ضعیفی بین متغیرهای X و Y برقرار است. ضریب همبستگی پیرسون به صورت زیر محاسبه می‌شود:

$$r = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}}$$

R^2 : Coefficient Of Determination .1-1-3

ضریب تعیین یا ضریب تشخیص Coefficient Of Determination قدرت توضیح دهندگی مدل را نشان می‌دهد. ضریب تعیین نشان می‌دهد که چند درصد از تغییرات متغیر وابسته توسط متغیرهای مستقل توضیح داده می‌شود. تغییرات کل متغیر وابسته برابر است با تغییرات توضیح داده شده توسط رگرسیون بعلاوه تغییرات توضیح داده نشده. این شاخص یکی از شاخص‌های برازش مدل است که قدرت پیش‌بینی متغیر وابسته (ملاک) براساس متغیرهای مستقل (پیش‌بین) را نشان می‌دهد. مقدار این شاخص بین صفر تا یک می‌باشد و اگر از ۶/۰ بیشتر باشد نشان می‌دهد متغیرهای مستقل تا حد زیادی توانسته‌اند تغییرات متغیر وابسته را تبیین کنند.

ضریب تشخیص در معادلات رگرسیونی با علامت R^2 نشان داده می‌شود و بیانگر میزان احتمال هم‌بستگی میان دو دسته داده در آینده می‌باشد. این ضریب در واقع نتایج تقریبی پارامتر موردنظر در آینده را بر اساس مدل ریاضی تعریف شده که منطبق بر داده‌های موجود است، بیان می‌دارد .در واقع معیاری است از این که خط رگرسیون، چقدر خوب خوانده‌ها را معرفی می‌کند. اگر خط رگرسیون از تمام نقاط بگذرد توانائی معرفی همه متغیرها را دارد و هرچه از نقاط دورتر باشد نشان دهنده توانائی کمتر است.

$$R^2 = \frac{\sum_{i=1}^{n}(y_i - \hat{y})^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

1-2. دیتاست

دیتاست مورد استفاده در این قسمت را به کمک دستور زیر در Colab فراخوانی کرده و فایل های دادگان را از فایل زیپ اکسترکت کرده‌ایم .

```
! kaggle datasets download sid321axn/beijing-multisite-airquality-
data-set
! unzip beijing-multisite-airquality-data-set
```

به کمک کتابخانه pandas و دستور pd.read_csv() دیتاست‌های مربوط به هریک از سایت‌های چینی را فراخوانی شده‌اند:

```
df_Aotizhongxin = pd.read_csv(
'/content/PRSA_Data_Aotizhongxin_201303
0120170228.csv',
 parse_dates={'Date': [1,2,3,4]},
 date_parser=timeparser,
 index_col=['Date'],
 na_values=['NaN','?'])
```

1-3. پیش پردازش

همانطور که در صورت سوال خواسته شده تا پیش پردازش‌های ذکر شده را برای تمامی ستون‌های دیتاست سایت Aotizhongxin و همچنین فقط برای ستونهای PM2 5.از باقی سایت‌ها انجام شود، در ابتدا این فیچرها در دیتافریمی با نام unify_df تجمیع کرده‌ایم.

Missing value 1-3-1

رایج ترین متدها برای جایگذاری داده های از دست رفته، استفاده از میانگین و میانه است، اما در خصوص داده‌های سری زمانی استفاده از متد linear interpolation نتیجه‌ی بهتری را برمیگرداند.

تعداد missing value های هر یک از فیچرها به صورت زیر می‌باشد.

```
unify_df.isnull().sum()
```

No	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
Changping pm2.5	774
Dingling pm2.5	779
Dongsi pm2.5	750
Guanyuan pm2.5	616
Gucheng pm2.5	646
Huairou pm2.5	953
Nongzhanguan pm2.5	628
Shunyi pm2.5	913
Tiantan pm2.5	677
Wanliu pm2.5	382
Wanshouxigong pm2.5	696

شکل 1. داده‌های مفقود شده

با استفاده از دستور زیر، داده‌های مفقود شده را به کمک داده‌های قبل و بعدشان جایگزین می‌کنیم.

```
unify_df = unify_df.interpolate()
```

Encoding Categorical Variable 1-3-2

تنها ستون غیر عددی، ستون wd می‌باشد که برای تبدیل آن به داده‌های عددی از یک دیکشنری به صورت زیر کمک گرفته‌ایم.

```
wd_dic  = {
    'N' : 0,
    'NNE' : 22.5,
    'NE' : 45,
    'ENE' : 67.5,
    'E' : 90,
    'ESE' : 112.5,
    'SE' : 135,
    'SSE' : 157.5,
    'S' : 180,
    'SSW' : 202.5,
    'SW' : 225,
    'WSW' : 247.5,
    'W' : 270,
    'WNW' : 292.5,
    'NW' : 315,
    'NNW' : 337.5,
    'N' : 360
}
unify_df = unify_df.replace({'wd': wd_dic})
```

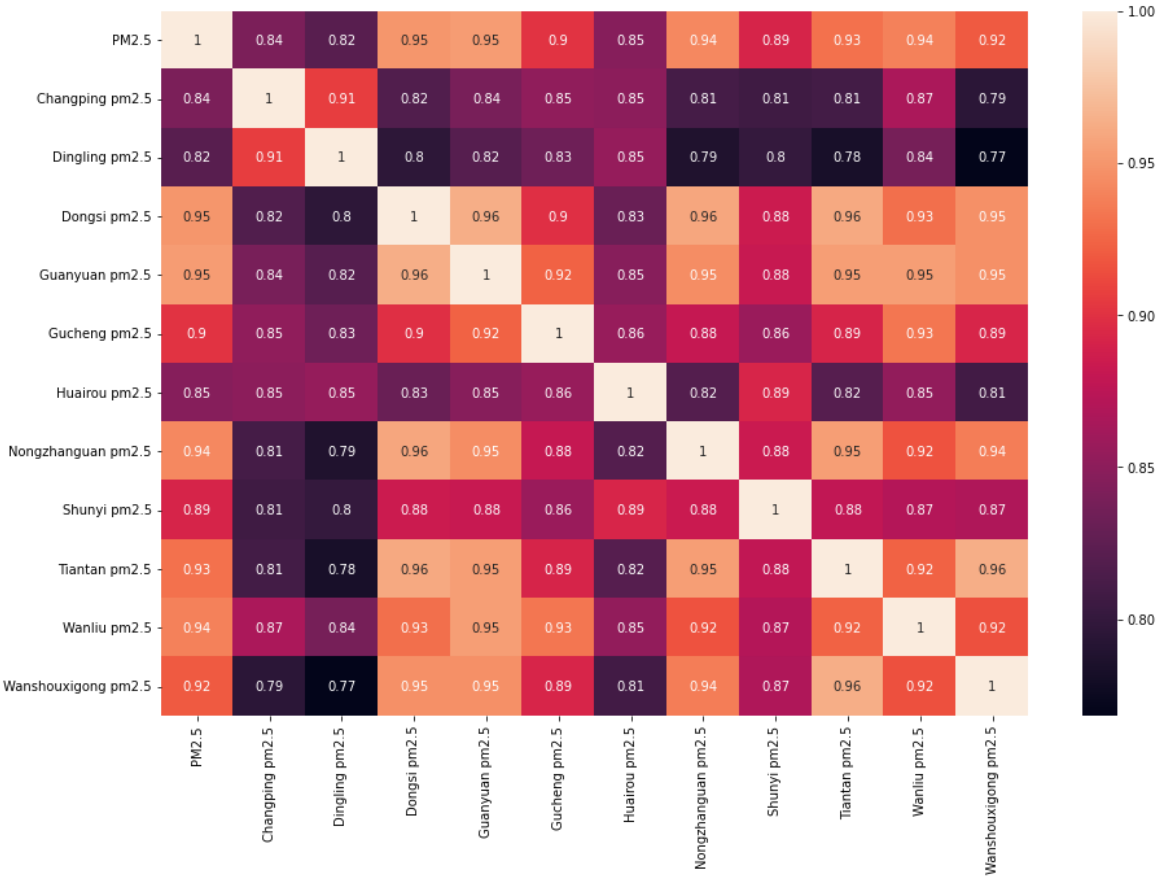
Nomarlization 1-3-3

برای نرمالایز کردن داده‌ها، MinMaxScaler را از کتابخانه‌ی sklearn فراخوانی کرده و بر روی داده‌های موجود پیاده کرده‌ایم.

```
scaler = MinMaxScaler()
scaled_df = scaler.fit_transform(unify_df)
```

Pearson Correlation 1-3-4

فیچرهایی که خواسته شده تا برای آن‌ها مقدار همبستگی پیرسون را بیابیم را در دیتافریمی با نام Air_Data قرار داده و سپس به کمک کتابخانه‌ی seaborn هیتمپ همبستگی را نشان داده‌ایم.



شکل 2. Pearson Correlation Heatmap

Feature selection 1-3-5

فایل اکسل با 20 ویژگی (داده های PM2.5 تمامی استگاه ها به همراه PM10, CO, TEMP, PRES, DEWP, RAIN,wd, WSPM مربوط به ایستگاه Aotizhongx) با نام Feature_selection.csv پیوست شده است.

Supervised dataset 1-3-6

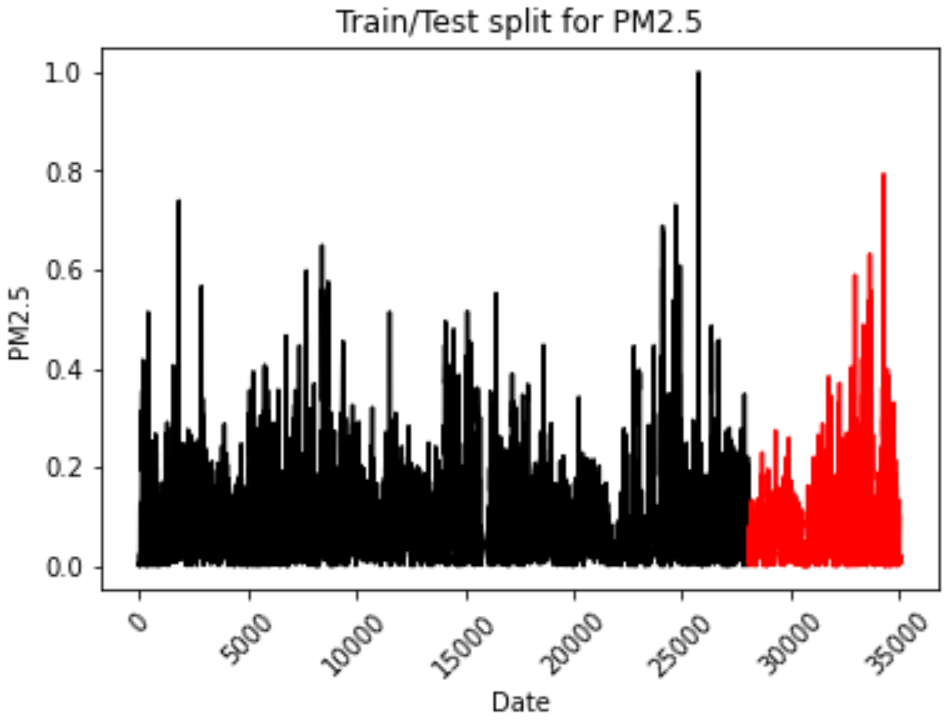
برای آماده‌سازی داده‌ها برای train و تبدیل آن‌ها به lag های 1 و 7 روزه، آن‌ها را به کمک تابع زیر به فرم Supervised در آورده‌ایم.

```
def series_to_supervised(data, n_in=1, n_out=1, dropnan=True):
  n_vars = 1 if type(data) is list else data.shape[1]
  df = pd.DataFrame(data)
  cols, names = list(), list()
  # input sequence (t-n, ... t-1)
  for i in range(n_in, 0, -1):
    cols.append(df.shift(i))
    names += [('var%d(t-%d)' % (j+1, i)) for j in range(n_vars)]
  # forecast sequence (t, t+1, ... t+n)
  for i in range(0, n_out):
    cols.append(df.shift(-i))
    if i == 0:
      names += [('var%d(t)' % (j+1)) for j in range(n_vars)]
    else:
      names += [('var%d(t+%d)' % (j+1, i)) for j in range(n_vars)]
  # put it all together
  agg = pd.concat(cols, axis=1)
  agg.columns = names
  # drop rows with NaN values
  if dropnan:
    agg.dropna(inplace=True)
  return agg
reframed_1 = series_to_supervised(selected, 1*24, 1)
reframed_7 = series_to_supervised(selected, 7*24, 1)
```

شکل 3. Series to Supervised

سپس داده‌هایی که به فرم Supervised درآمده‌اند را به داده‌های ترین و تست تقسیم کرده‌ایم.

```
def test_train_split(reframed):
  values = reframed.values
  n_train_hours = 28052
  train = values[:n_train_hours, :]
  test = values[n_train_hours:, :]
  train_X, train_y = train[:, :-1], train[:, -1]
  test_X, test_y = test[:, :-1], test[:, -1]
  train_X = train_X.reshape((train_X.shape[0], 1, train_X.shape[1]))
  test_X = test_X.reshape((test_X.shape[0], 1, test_X.shape[1]))
  train_X = np.asarray(train_X)
  test_y = np.asarray(test_y)
  return train_X, train_y, test_X, test_y
```



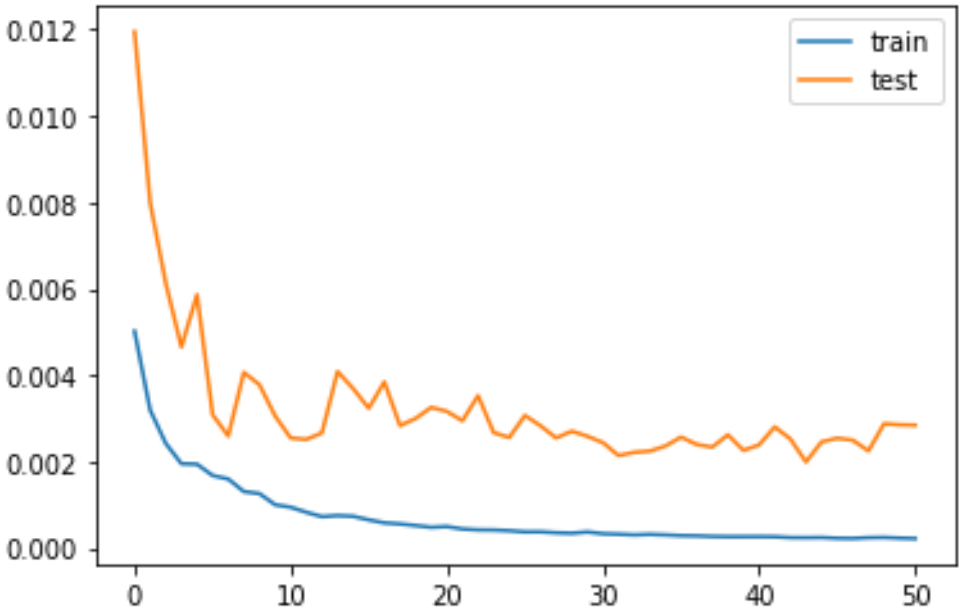
شکل4. Train/Test Split

1-4 آموزش شبکه

در مدل CNN-LSTM طراحی شده، شبکه حاوی 1D CNN است که شامل سه لایه کانولوشن با 64، 64 و 32 آشکارساز با کرنل سایز 3 است که به طور متوالی در شبکه قرار گرفته‌اند. بین سه لایه کانولوشن، لایه BatchNormalization استفاده می شود. یک لایه MaxPooling1D با اندازه pool:3 بعد از هر لایه کانوولوشن قرار گرفته است. آخرین لایه‌ها مربوط به LSTM است که شامل دو لایه با 100 و 50 واحد در هر لایه می‌باشند، سپس یک لایه Dense با یک خروجی ساختار شبکه را تشکیل می‌دهد.

```
def CNN_LSTM(train_X):

  tf.keras.backend.clear_session()
  model = tf.keras.models.Sequential([
      tf.keras.layers.Conv1D(filters=64,
                             kernel_size=3,
                             strides=1,
                             padding="causal",
                             activation="relu",
```

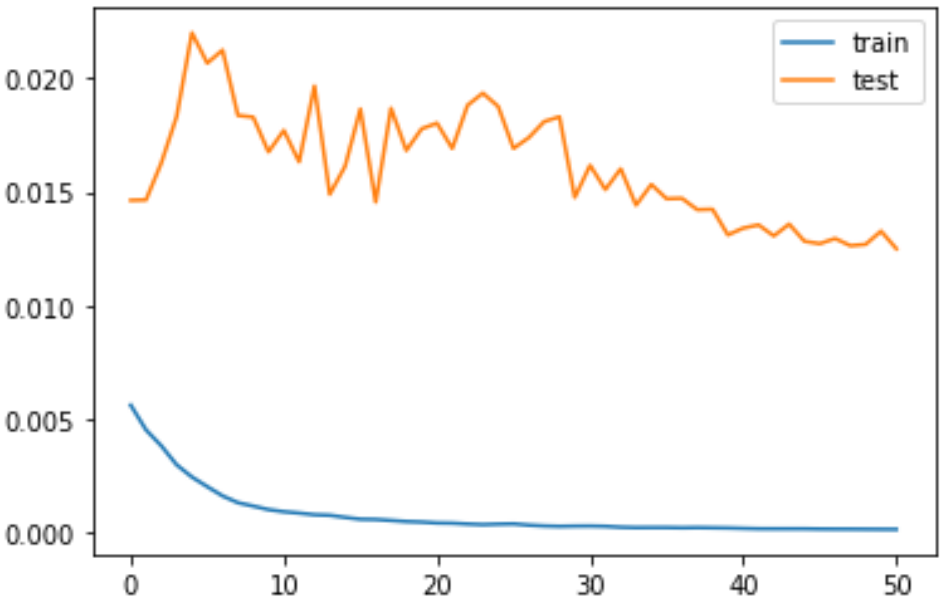
```
                            input_shape=(train_X.shape[1],
                            train_X.shape[2])),
    tf.keras.layers.BatchNormalization(axis=-1,
                                       momentum=0.99,
                                       epsilon=0.001,),
    tf.keras.layers.MaxPooling1D(pool_size=3, strides=1,
                                 padding="same"),

    tf.keras.layers.Conv1D(filters=64, kernel_size=3, strides=1,
                           padding="causal", activation="relu"),
    tf.keras.layers.BatchNormalization(axis=-1,
                                       momentum=0.99,
                                       epsilon=0.001,),
    tf.keras.layers.MaxPooling1D(pool_size=3, strides=1,
                                 padding="same"),

    tf.keras.layers.Conv1D(filters=32, kernel_size=3, strides=1,
                           padding="causal", activation="relu"),
    tf.keras.layers.BatchNormalization(axis=-1,
                                       momentum=0.99,
                                       epsilon=0.001),
    tf.keras.layers.MaxPooling1D(pool_size=3, strides=1,
                                 padding="same"),

    tf.keras.layers.LSTM(100, return_sequences=True),
    tf.keras.layers.LSTM(50, return_sequences=True),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(1)
])

opt = tf.keras.optimizers.Adam(learning_rate=0.001, decay=0.0001)

model.compile(loss=tf.keras.losses.MeanSquaredError(),
              optimizer=opt,
              metrics=['accuracy',
                       tf.keras.metrics.MeanAbsoluteError(),
                       tfa.metrics.RSquare(),
                       tf.keras.metrics.RootMeanSquaredError()])
model.summary()

return model
```



شکل5. Training/Test Loss

LOSS	ACCURACY	MAE	R2	RMSE
0.011954	0.01095	0.09665	0.371591	0.109338

جدول1. نتایج مدل برای Lag 1 روز



شکل6. Training/Test Loss

LOSS	ACCURACY	MAE	R2	RMSE
0.0146322	0.014524	0.0792244	0.1612676	0.1209639

جدول2. نتایج مدل برای Lag 7 روز

با مقایسه CNN-LSTM در تاخیرهای 1 روزه با تاخیرهای 7 روزه، RMSEافزایش می یابد، MAE و R2 کاهش می یابد.

پاسخ ۲ – تشخیص اخبار جعلی

۲-۱. توضیحات مدل ها

RNN ها دارای حلقه های بازخورد در لایه بازگشتی هستند. این به آنها اجازه می دهد تا اطلاعات را در "حافظه" در طول زمان حفظ کنند. اما، آموزش RNN های استاندارد برای حل مشکلاتی که نیاز به یادگیری وابستگی های زمانی طولانی مدت دارند، می تواند دشوار باشد.

ها شبکه عصبی قدرتمند و قوی هستند و می‌توان آنها را امیدوارکننده‌ترین الگوریتم‌های مورد استفاده دانست؛ زیرا تنها الگوریتمی است که حافظه داخلی دارد. مانند بسیاری از الگوریتم‌های یادگیری عمیق دیگر، شبکه‌های عصبی بازگشتی نسبتاً قدیمی هستند. افزایش قدرت محاسباتی همراه با حجم انبوه داده‌ای که اکنون باید با آن کار کنیم و ابداع شبکه Long Short-term memory (LSTM)باعث شده‌اند کهRNN ها به شبکه‌هایی پیشگام تبدیل شوند.

RNNها به دلیل وجود حافظه داخلی می‌توانند مفاهیم مهمی در مورد ورودی دریافتی خود به خاطر بسپارند که به آنها این امکان را می‌دهد در پیش بینی اتفاقات بعدی بسیار دقیق باشند. به همین دلیل است که این الگوریتم‌ها برای داده‌های توالی مانند سری زمانی، داده گفتار، متن، داده‌های مالی، صوت، ویدیو، آب و هوا و بسیاری موارد دیگر، الگوریتم ارجح هستند. شبکه‌های عصبی بازگشتی در مقایسه با سایر الگوریتم‌ها می‌توانند درک عمیق‌تری از توالی ایجاد کنند.

شبکه‌های LSTM نسخه‌ی توسعه یافته‌ای از شبکه‌های عصبی بازگشتی هستند که حافظه را توسعه می‌دهند. بنابراین برای یادگیری از تجربیات مهمی که فاصله زمانی زیادی بین آنها وجود دارد، بسیار مناسب هستند.

شبکه‌های LSTM نسخه‌ای از شبکه‌های RNN هستند که حافظه را بسط می‌دهند .LSTM به عنوان بلوک ساختاری لایه‌های RNN استفاده می‌شود. شبکه‌های LSTM به داده‌ها، «وزن‌هایی» را اختصاص می‌دهد و به RNN این امکان را می‌دهد تا اطلاعات جدید را وارد کند، اطلاعات را فراموش کند و یا به آنها اهمیت کافی دهد تا روی خروجی اثر بگذارد. واحدهای یک LSTM به عنوان واحدهای ساختاری لایه‌های یک RNN استفاده می‌شوند که اغلب شبکه LSTM نامیده می‌شوند.

LSTMها شبکه‌های RNN را قادر می‌سازند تا ورودی‌ها را در مدت زمان طولانی به خاطر بسپرند. این بدین خاطر است کهLSTM ها حاوی اطلاعاتی در حافظه هستند، دقیقاً مانند حافظه یک کامپیوتر . LSTMمی‌تواند اطلاعات را از حافظه خود بخواند، روی آن بنویسد و یا اطلاعات را پاک کند. این حافظه را می‌توان به عنوان یک سلول گیت دار در نظر گرفت و در اینجا گیت به آن معنی است که سلول بر اساس اهمیتی که به اطلاعات می‌دهد، تصمیم می‌گیرد که اطلاعات را ذخیره یا حذف کند (یعنی گیت‌ها را باز یا بسته کند). تخصیص اهمیت از طریق وزن‌ها صورت می‌گیرد، که توسط الگوریتم آموخته می‌شوند. این بدان معنی است که شبکه در طول زمان یاد می‌گیرد که چه اطلاعاتی مهم هستند و کدام اطلاعات مهم نیستند.

هنگام انجام مدل‌سازی متن معمولی، بیشتر وظایف پیش‌پردازش و کار مدل‌سازی بر ایجاد داده‌ها به صورت متوالی متمرکز است. نمونه‌هایی از این کارها می‌تواند برچسب‌گذاری POS، حذف کلمات توقف، ترتیب‌بندی متن باشد. اینها روش هایی هستند که سعی می کنند داده ها را با تلاش کمتر طبق الگوی شناخته شده توسط یک مدل درک کنند. می تواند نتایج را بدهد.

شبکه های LSTM می تواند ویژگی خاص خود را داشته باشد. LSTM دارای ویژگی است که از طریق آن می تواند توالی داده ها را به خاطر بسپارد. این یک ویژگی دیگر دارد که روی حذف اطلاعات استفاده نشده کار می کند و همانطور که می دانیم داده های متنی همیشه حاوی اطلاعات استفاده نشده زیادی است که می تواند توسط LSTM حذف شود تا زمان محاسبه و هزینه کاهش یابد.

بنابراین اساساً ویژگی حذف اطلاعات استفاده نشده و به خاطر سپردن توالی اطلاعات، LSTM را به ابزاری قدرتمند برای انجام طبقه بندی متن یا سایر وظایف مبتنی بر متن تبدیل می کند.

مدل پیشنهادی از توانایی CNN برای استخراج ویژگی‌های محلی و LSTM برای یادگیری وابستگی‌های بلندمدت استفاده می‌کند .ابتدا، یک لایه CNN از Conv1D برای پردازش بردارهای ورودی و استخراج ویژگی های محلی که در سطح متن قرار دارند استفاده می شود .خروجی لایه CNN ورودی لایه RNN واحدها/سلول‌های LSTM است که در ادامه می‌آید .لایه RNN از ویژگی های محلی استخراج شده توسط CNN استفاده می کند و وابستگی های طولانی مدت ویژگی های محلی مقالات خبری را که آنها را به عنوان جعلی یا واقعی طبقه بندی می کند، می آموزد.
ترکیب CNN-RNN در چندین کار طبقه بندی و رگرسیون موفقیت آمیز به اثبات رسیده است، زیرا آنها توانایی گرفتن هر دو ویژگی محلی و ترتیبی داده های ورودی را دارند .به عنوان مثال، آنها برای تشخیص احساسات و تشخیص زبان اشاره از جریان های ویدئویی استفاده شده اند، و از توانایی آنها برای یادگیری ویژگی های صحنه با استفاده از CNN و ویژگی های متوالی با استفاده از RNN استفاده می شود .در مورد وظایفNLP ، RNN می‌تواند ویژگی‌های زمانی و زمینه‌ای را از متن یاد بگیرد و

وابستگی‌های طولانی‌مدت بین موجودیت‌های متن و ویژگی‌های مهم را که با استفاده از توانایی CNN در مدیریت روابط فضایی شناسایی می‌شوند، ثبت کند.
مدل‌های یادگیری عمیق با وجود مزایایی که دارند، محدودیت‌های عملی خاصی دارند، مانند مشکل در یافتن فراپارامترهای بهینه برای هر مسئله و مجموعه داده، نیاز به مجموعه داده‌های آموزشی بزرگ، و عدم تفسیرپذیری، که تأثیر مستقیمی بر عملکرد آن‌ها در مدل‌های جدید دارد .و وظایف ناشناخته و آنها را وادار می کند که مانند اوراکل های جعبه سیاه رفتار کنند .پیشرفت‌های اخیر در روش‌های الهام‌گرفته از زیستی، بهینه‌سازی پارامترهای یادگیری عمیق را امکان‌پذیر می‌کند و اساس الگوریتم‌های بهینه‌سازی نسل بعدی برای یادگیری ماشین را تشکیل می‌دهد .مدل ترکیبی پیشنهادی می‌تواند به طور قابل توجهی از بهینه‌سازی فراپارامترهای آن بهره‌مند شود و بخشی از کار بعدی در این زمینه برای بررسی تکنیک‌های مختلف الهام‌گرفته از زیستی و یافتن مناسب‌ترین آنها برای کار فعلی است.

2-2 ورودی مدل

Word embeddingها بردار های عددی هستند که نمایانگر کلمات یک لغت نامه اند و کاربردهای گسترده ای هم در حوزه پردازش زبان طبیعی دارند. Word embedding به شما اجازه میدهد تا بطور غیرصریح اطلاعاتی را از دنیای بیرونی به مدل های زبانی خود اضافه کنید

مفهوم اصلی word embedding این است که تمامی لغات استفاده شده در یک زبان را میتوان توسط مجموعه ای از اعداد اعشاری (در قالب یک بردار) بیان کرد Word embedding .ها بردارهای-n بعدی ای هستند که تلاش میکنند معنای لغات و محتوای آنها را با مقادیر عددی خود ثبت و ضبط کنند. هر مجموعه ای از اعداد یک "بردار کلمه" معتبر بحساب می آید که الزاما برای ما سودمند نیست، آن مجموعه ای از بردار کلمات برای کاربردهای مورد نظر ما سودمندند که معنای کلمات ، ارتباط بین آنها و محتوای کلمات مختلف را همانطور که بصورت طبیعی [توسط ما] مورد استفاده قرار گرفته اند، بدست آورده باشند.

پس word embedding های سودمند چند مشخصه کلیدی دارند:

- هر کلمه دارای یک word embedding (یا بردار) یکتاست که تنها شامل لیستی ساده از اعداد به ازای هر کلمه است.
- word embeddingها چند بعدی اند و عموما یک مدل خوب، embeddingهایی با طولی بین ۵۰ الی ۵۰۰ بعد دارد.
- به ازای هر کلمه، embeddingمعنای آن کلمه را بدست می آورد.
- کلمات مشابه در نهایت به مقادیر embedding مشابه میرسند.

word embeddingها در گستره زیادی از عملیات های مرتبط با پردازش زبان طبیعی مورد استفاده قرار میگیرند

- علاوه بر تکنیک های مدلسازی نظیر شبکه های عصبی مصنوعی، word embeddingsبه طرز شگفت آوری دقت دسته بندی متن را در بسیاری ازدامنه های نظری، سرویس مشتری، تشخیص هرزنامه، دسته بندی اسناد و ...را بهبود داده است.
- word embeddingبرای بهبود کیفیت ترجمه زبانهای مختلف از طریق aligning single-language word embedding با استفاده از یک ماتریس تبدیل مورد استفاده قرار گرفته است.
- بردار کلمات همچنین جهت بهبود دقت در کاربردهای مبتنی بر جستجوی اسناد و بازیابی اطلاعات، که در آن تنظیمات جستجو نیازمند جستجوی کلید واژه های دقیق نیست و نسبت به املا نیز حساسیت ندارد استفاده شده اند.

لازم به ذکر است کاربردهای مبتنی بر بردار کلمات و بطور خاص Word2vec محدود به تجزیه جملات نبوده و چیزی فراتر از آن است. از آن میتوان بر روی گستره زیادی از داده ها همانند، ژن، سورس کد، پسندها در یک شبکه اجتماعی، playlistها، گراف های شبکه های اجتماعی و سری های سمبولیک و... که میتوانند حاوی الگوهایی باشد، استفاده کرد.

همانطور که می دانیم مدل های یادگیری ماشینی نمی توانند متن را پردازش کنند، بنابراین باید راهی برای تبدیل این داده های متنی به داده های عددی پیدا کنیم .تکنیک‌هایی مانند Bag of Words و TF-IDFمی‌توانند به استفاده از این وظیفه کمک کنند .جدای از این، می‌توانیم از دو تکنیک دیگر مانند رمزگذاری one-hot یا از اعداد منحصربه‌فرد برای نمایش کلمات در واژگان استفاده کنیم .روش دوم کارآمدتر از رمزگذاری one-hot است زیرا به جای یک بردار پراکنده، اکنون یک بردار متراکم داریم .بنابراین این رویکرد حتی زمانی کار می‌کند که دایره لغات ما زیاد باشد.
روش‌های word embeddings عبارت اند از:

word2Vec:

در Word2Vec به هر کلمه یک بردار اختصاص داده می شود .ما با یک بردار تصادفی یا یک بردار داغ شروع می کنیم .بردار تک داغ: نمایشی که در آن تنها یک بیت در یک بردار 1 است. اگر 500 کلمه در پیکره وجود داشته باشد، طول بردار 500 خواهد بود. پس از اختصاص بردارها به هر کلمه، یک اندازه پنجره می گیریم و در کل پیکره تکرار می کنیم .

Continuous Bag-of-Words model (CBOW):

CBOWاحتمال وقوع یک کلمه را با توجه به کلمات اطراف آن پیش بینی می کند .ما می توانیم یک کلمه یا یک گروه از کلمات را در نظر بگیریم .اما برای سادگی، از یک کلمه متنی استفاده می کنیم و سعی می کنیم یک کلمه هدف واحد را پیش بینی کنیم.

Skip-gram model:

معماری مدل Skip-gram معمولاً سعی می‌کند به عکس آنچه مدل CBOW انجام می‌دهد دست یابد . سعی می کند کلمات متن منبع (کلمات اطراف) را با یک کلمه هدف (کلمه مرکزی) پیش بینی کند . عملکرد مدل skip-gram کاملاً شبیه به CBOW است، اما فقط تفاوت در معماری شبکه عصبی آن و نحوه تولید ماتریس وزن وجود دارد.

GloVe:

(بردارهای جهانی برای نمایش کلمات) یک روش جایگزین برای ایجاد جاسازی کلمات است .این مبتنی بر تکنیک های فاکتورسازی ماتریس در ماتریس کلمه-زمینه است .ماتریس بزرگی از اطلاعات همزمان ساخته می‌شود و شما هر «کلمه» (ردیف‌ها) را می‌شمارید، و چقدر این کلمه را در برخی «زمینه‌ها» (ستون‌ها) در یک مجموعه بزرگ می‌بینیم .معمولا، ما مجموعه خود را به روش زیر اسکن می کنیم: برای هر عبارت، ما به دنبال اصطلاحات زمینه در محدوده ای می گردیم که با اندازه پنجره قبل از عبارت و اندازه پنجره بعد از عبارت تعریف شده است .همچنین برای کلمات دورتر وزن کمتری قائلیم.

در این سوال ما از روش GloVe به صورت زیر استفاده کرده‌ایم.

```
#Using Pre-trained word embeddings: GLOVE Method
GLOVE_DIR = "/content"
embeddings_index = {}
f = open(os.path.join(GLOVE_DIR, 'glove.6B.100d.txt'), encoding="utf8")
for line in f:
    values = line.split()
    word = values[0]
    vecs = np.asarray(values[1:], dtype='float32')
    embeddings_index[word] = vecs
f.close()
embedding_matrix = np.zeros((len(word_index) + 1, EMBEDDING_DIM))
for word, i in word_index.items():
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        embedding_matrix[i] = embedding_vector

embedding_layer = Embedding(len(word_index) + 1, EMBEDDING_DIM, weights=[embedding_matrix], input_length=MAX_SEQUENCE_LENGTH)
```

شکل 7. Word Embedding

2-3 پیاده سازی

2-3-1 پیش پردازش

برای پیش پردازش داده‌ها ابتدا Punctuation ,IP و URL های موجود در دیتا را حذف کرده‌ایم. Stopwords ها را به کمک کتابخانه‌ی `nltk.corpus` حذف کرده‌ایم و برای Stemming از `porterstemming` استفاده کرده‌ایم.

```
def pre_process_news(news):

    lemm = WordNetLemmatizer()
    ps = PorterStemmer()
    stop_words = set(stopwords.words('english'))
    news = news.lower()
    news = re.sub("[^0-9a-z]", ' ' , news) # removes punctuations
    news = re.sub(r"http\S+", "", news)
    tokenized = news.split(" ")
    news = [ ps.stem(word) for word in tokenized if word not in stop_words] # apply stemming and drop stopwords
    news = " ".join(news)  # join the tokens into a string
    news = lemm.lemmatize(news)
    return news
News['processed_text'] = News['text'].apply(lambda news : pre_process_news(news))
data = News["processed_text"]
```

شکل 8. Data Preprocessing

به کمک tokenizer متن اخبار را به توکن‌ها تبدیل کرده‌ایم.

```
tokenizer = Tokenizer(filters='!"#$%&()*+,-./:;<=>?@[\\]^_`{|}~\t\n', lower = True, split = ' ')
tokenizer.fit_on_texts(texts = data)
sequences = tokenizer.texts_to_sequences(texts = data)
word_index = tokenizer.word_index
```

شکل 9. Tokenizer

در نهایت داده‌ها برای آموزش مدل به قسمت‌های train, test, validation تقسیم کرده‌ایم.

```
from sklearn.model_selection import train_test_split
indices = np.arange(data.shape[0])
np.random.shuffle(indices)
data = data[indices]
labels = labels[indices]
x_train, x_test, y_train, y_test = train_test_split( data, labels, test_size=0.20, random_state=42)
x_test, x_val, y_test, y_val = train_test_split( x_test, y_test, test_size=0.50, random_state=42)
```

شکل 10. Train/Test Split

```
x_test, x_valid, y_test, y_valid = train_test_split(x_test, y_test,
                                                    test_size=0.20,
                                                    random_state=42)
```

یک لایه‌ی embedding به کمک متد GLOVE ایجاد کرده‌ایم که پیش‌تر نشان‌داده شده است و آن را ورودی مدل شبکه قرار داده‌ایم.

2-3-2 آموزش مدل ها

مدل یادگیری عمیق ترکیبی پیشنهادی با استفاده از مدل متوالی کتابخانه Python یادگیری عمیق Keras مدل Sequential از چندین لایه نورون تشکیل شده است:

• اولین لایه شبکه عصبی embedding layer است: این لایه ورودی است که از طریق آن با ارائه ماتریس تعبیه آماده شده از کلمات تعبیه شده از قبل آموزش داده شده استفاده می شود. مدل با تغذیه در داده های آموزشی، آموزش داده می شود.

• لایه بعدی لایه CNN یک بعدی (Conv1D) برای استخراج ویژگی های محلی با استفاده از 128 فیلتر با اندازه 5 است. پیش فرض تابع فعال سازی واحد خطی اصلاح شده (ReLU) استفاده می شود.

• پس از آن، بردارهای ویژگی بزرگ تولید شده توسط CNN ادغام می شوند با وارد کردن آنها به یک لایه MaxPooling1D با اندازه پنجره 2، به منظور کاهش نمونه برداری از بردارهای ویژگی بدون تاثیر در کارایی شبکه، استفاده می‌شود.

• خروجی لایه MaxPooling1D به لایه‌ی RNN (LSTM) وارد می‌شود. این ورودی برای آموزش LSTM استفاده می‌شود، که ویژگی‌های وابسته بلندمدت نقشه‌های ویژگی ورودی را خروجی می‌دهد، در حالی که یک حافظه بعد خروجی روی 32 تنظیم شده است.

• در نهایت، بردارهای ویژگی آموزش دیده با استفاده از یک لایه متراکم طبقه بندی می شوند

که بعد فضای خروجی را به 1 کاهش می دهد که با برچسب طبقه بندی (یعنی جعلی یا جعلی نیست) مطابقت دارد. در این لایه از تابع فعالسازی sigmoid استفاده شده است.

این مدل با استفاده از optimizer، adaptive moment estimation (Adam) برای تعریف نرخ یادگیری در هر تکرار آموزش داده شده است. از binary crossentropy در این مدل ها به عنوان loss function استفاده شده است. تعداد epochs = 10 و batch_size = 64 در نظر گرفته شده است.

```
# CNN-RNN Model
modell = Sequential()
modell.add(embedding_layer)
modell.add(Conv1D(filters=128, kernel_size=5, padding='valid', activation='relu'))
modell.add(MaxPooling1D(pool_size=2))
modell.add(LSTM(100, dropout=0.2))
modell.add(BatchNormalization())
modell.add(Dense(32, activation='relu'))
modell.add(Dense(1, activation='sigmoid'))
modell.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy',f1_m, precision_m, recall_m])
print(modell.summary())
history=modell.fit(x_train, y_train, epochs=EPOCHS, batch_size=64, validation_data=(x_valid, y_valid))
```

شکل 11. Hybrid(CNN-RNN) Model

برای ارزیابی مدل هایبرید فوق داده‌هارا بار دیگر توسط مدل RNN زیر آمورش داده‌ایم تا نتایج حاصل از هر مدل را با هم مقایسه کنیم تا متوجه شویم برای دیتاست موجود کدام مدل نتیجه‌ی بهتری را برمیگرداند.

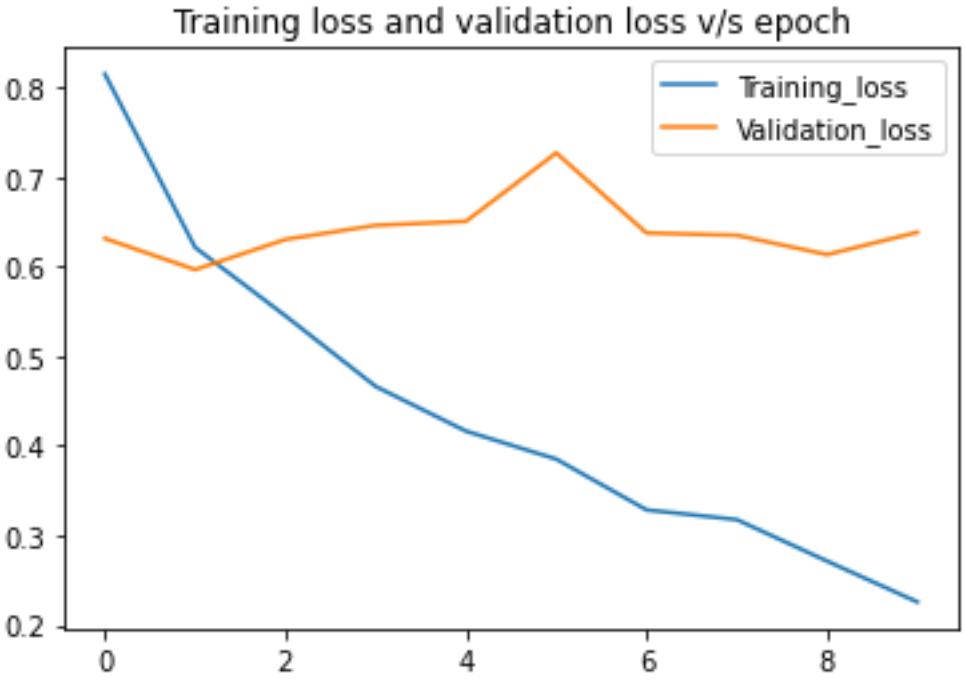
```
# RNN_MODEL
EPOCHS = 10
model = Sequential()
model.add(embedding_layer)
model.add(LSTM(100, dropout=0.2))
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy',f1_m, precision_m, recall_m])
print(model.summary())
history_rnn=model.fit(x_train, y_train, epochs=EPOCHS, batch_size=64, validation_data=(x_valid,y_valid))
```

شکل 12. RNN Model

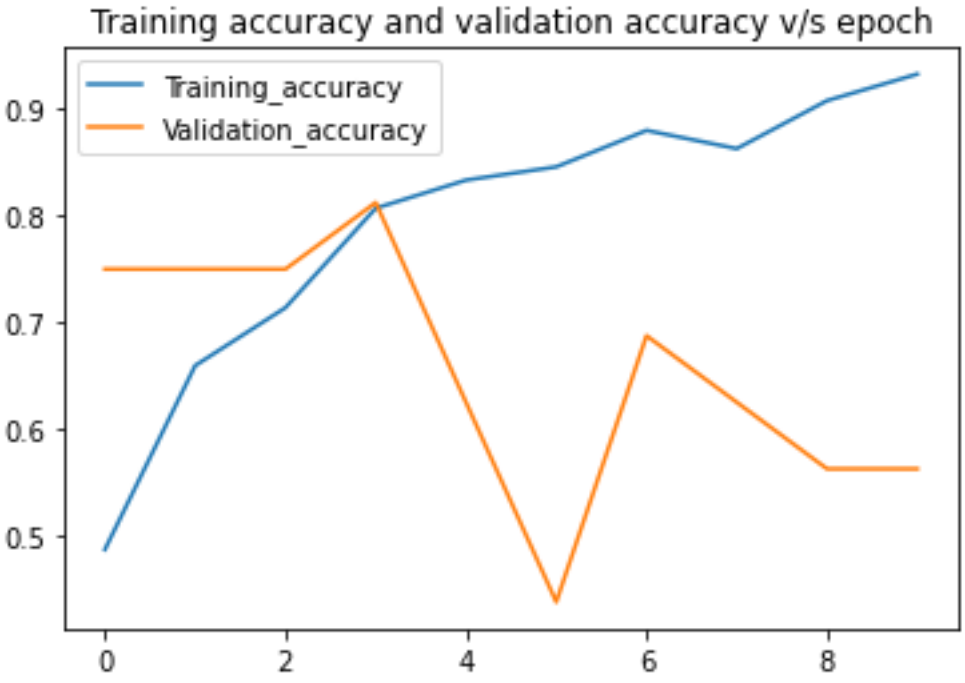
3-3-2 تحلیل نتایج

Loss	Accuracy	F1-Score	Precision	Recall
0.76064	0.46875	0.58632	0.47435	0.79605

جدول 3. نتایج حاصل از مدل CNN-RNN



شكل 13. Training/validation Loss Hybrid



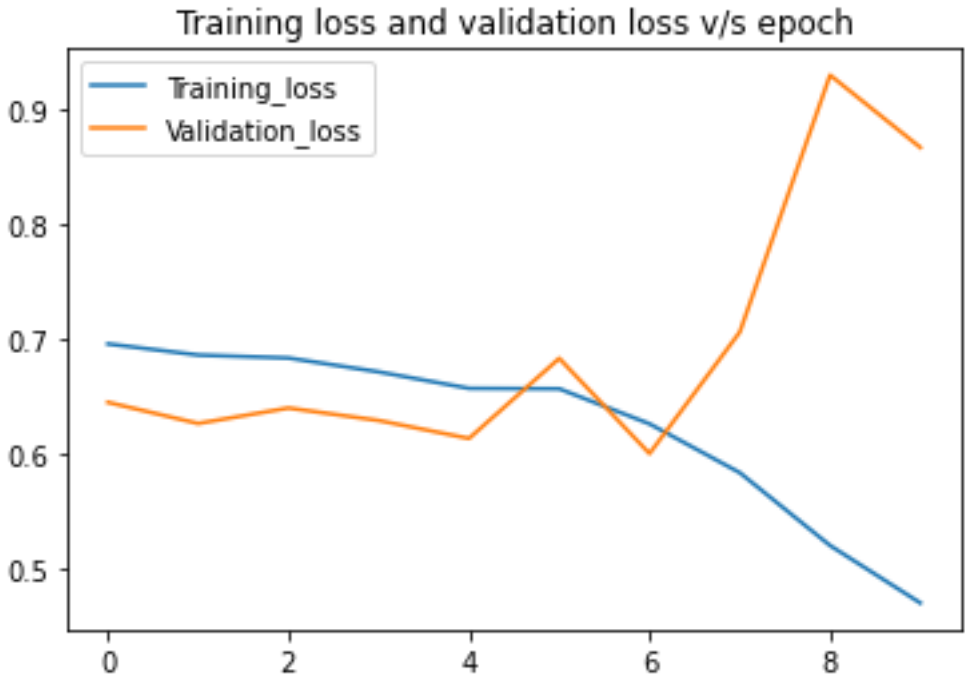
شكل 14. Training/Validation accuracy Hybrid

نتایج نشان می دهد که در حالی که دقت ترین و loss بهینه است پس از epoch 6، صحت اعتبارسنجی در همه موارد تقریباً یکسان است. دوره‌ها و پایین‌تر از آن چیزی است که هنگام آموزش (و اعتبارسنجی) به دست می‌آید.

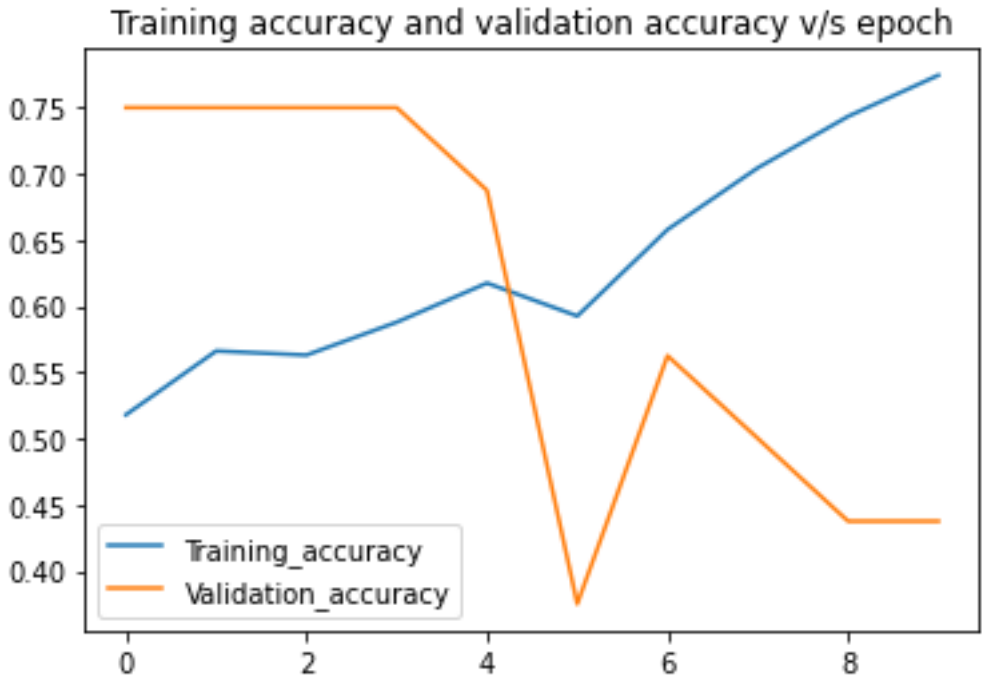
نمودار loss function، overfitting مدل را نشان می دهد، زیرا loss با روند افزایشی نوسان زیادی دارد .

Loss	Accuracy	F1-Score	Precision	Recall
0.93572	**0.46875**	0.455455	0.44444	0.471491

جدول4. نتایج حاصل از مدل RNN



شکل15. Training/validation accuracy RNN



شکل16. Training/validation Loss RNN

نمودار های فوق برای مدل RNN نشان می‌دهد که مقدار Ir (Iteration rate) برای این مدل بزرگ انتخاب شده است.

نتایج نشان می‌دهد که روش ترکیبی CNN-RNN پیشنهادی به طور قابل‌توجهی بهتر از روش RNN از نظر دقت و recall است.

.