



به نام خدا  
دانشگاه تهران  
دانشکده مهندسی  
برق و کامپیوتر



درس شبکه‌های عصبی و یادگیری عمیق  
تمرین چهارم

e

نام و نام خانوادگی	نام نام خانوادگی - نام خانوادگی
شماره دانشجویی	810199357 - 810199395
تاریخ ارسال گزارش	1402.09.29

## فهرست

- پاسخ 1. پیش بینی سری زمانی.....2
- پاسخ ۱.۱. دائلود داده ها.....3
- پاسخ ۲.۱. کاوش در داده های سری زمانی و آشنایی با تعوری ها و کتابخانه های معروف.....4
- پاسخ ۳-۱. Time Series Split.....5
- ۴-۱. آماده سازی ورودی و خروجی مدل.....5
- ۵-۱. مدل های شبکه عصبی حافظه دار.....6
- 6-۱. Forecast Naïve.....13
- پرسش ۲. پیش بینی افکار خودکشی در رسانه های اجتماعی.....14
- ۲.۱. پیش پردازش داده.....14
- ۲.۲. ساخت ماتریس جاسازی.....14
- ۲.۳. آموزش مدل های یادگیری عمیق.....15
- ۲.۴. مقایسه نتایج.....17

شکل‌ها

[preprocessing.2.1](#)

[CNN + 2-layer LSTM.2.2](#)

[train & val loss of 3 models.2.3](#)

[train/val acc of 3 models.2.4](#)

[شمای دیتای دانلود شده](#)

[نمودار Close Return برای سهم آمازون](#)

[نمودار Candlestick سهم آمازون](#)

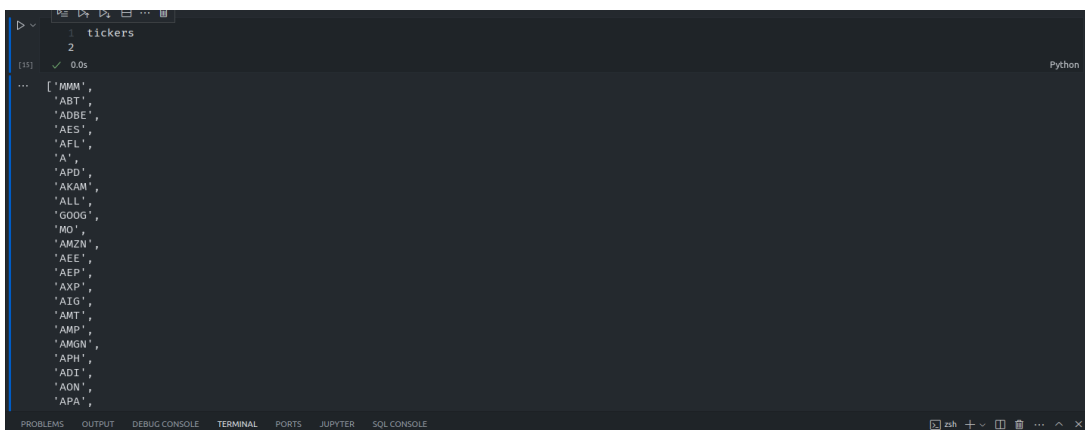
[نمودار بالا پیش بینی های مدل های GRU \(سبز رنگ\) ...](#)

[LSTM Training and Loss at the end](#)

## پاسخ 1 . پیش بینی سری زمانی

## پاسخ ۱.۱ . دانلود داده ها

در این بخش با استفاده از کتابخانه yahoo finance داده های ۵۰۰ شرکت برتر آمریکا را دانلود کرده ایم



```
tickers
2
[15] ✓ 0.0s
... ['MMM',
      'ABT',
      'ADBE',
      'AES',
      'AFL',
      'A',
      'APD',
      'AKAM',
      'ALL',
      'GOOG',
      'MO',
      'AMZN',
      'AEE',
      'AEP',
      'AXP',
      'AIG',
      'AMT',
      'AMP',
      'AMGN',
      'APH',
      'ADI',
      'AON',
      'APA',
      ...]
```

لیست بعضی ticker ها

1 symbol_date															
Date	GWW						LOW						ADBE		
	Open	High	Low	Close	Adj Close	Volume	Open	High	Low	Close	...	Low	Close	Adj Close	Volume
2010-01-04	97.239998	97.970001	96.199997	97.239998	76.182076	535200	23.510000	23.590000	23.139999	23.160000	...	36.650002	37.090000	37.090000	4710200
2010-01-05	96.959999	97.489998	96.139999	97.279999	76.213440	594500	23.120001	23.129999	22.740000	22.920000	...	36.869999	37.700001	37.700001	7108800
2010-01-06	97.269997	98.250000	97.269997	97.650002	76.503326	486800	22.840000	23.170000	22.750000	22.980000	...	37.200001	37.619999	37.619999	5336400
2010-01-07	97.269997	98.699997	96.709999	98.570000	77.224083	650100	23.240000	23.809999	23.070000	23.639999	...	36.810001	36.889999	36.889999	5576700
2010-01-08	98.000000	99.849998	97.769997	99.730003	78.132874	705600	23.510000	23.639999	23.250000	23.590000	...	36.340000	36.689999	36.689999	5429200
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2023-12-19	836.380005	841.000000	830.109985	830.190002	830.190002	209100	224.630005	226.330002	224.509995	225.419998	...	598.080017	604.640015	604.640015	4150700
2023-12-20	818.429993	832.900024	814.229980	816.010010	816.010010	430800	222.479996	224.729996	221.240005	222.009995	...	595.070007	596.059998	596.059998	2587300
2023-12-21	823.390015	825.659973	819.119995	824.039978	824.039978	138400	223.789993	224.440002	222.229996	223.550003	...	594.820007	600.140015	600.140015	3192800
2023-12-22	825.320007	832.840027	821.960022	827.849976	827.849976	164500	224.039993	225.080002	221.990005	223.000000	...	596.000000	598.750000	598.750000	1659800
2023-12-22	829.559998	831.994995	826.219971	829.474976	829.474976	58713	222.899994	223.695007	222.630005	222.919998	...	596.580017	598.239990	598.239990	1087182

شمای دیتای دانلود شده

```

1 after_2010 = table[
2     [
3         datetime.strptime(dt, "%Y-%m-%d") < datetime(2010, 1, 1)
4         for dt in table["Date added"]
5     ]
6 ]

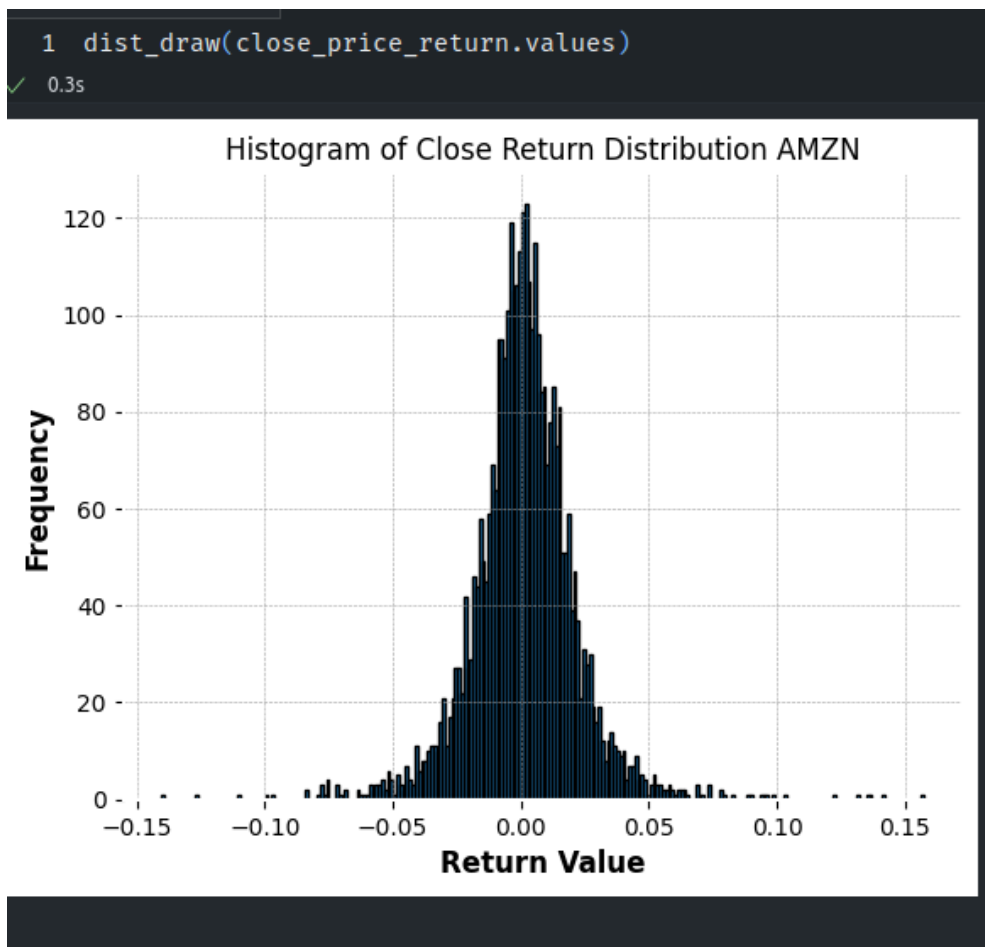
```

	Symbol	Security	GICS Sector	GICS Sub-Industry	Headquarters Location	Date added	CIK	Founded
0	MMM	3M	Industrials	Industrial Conglomerates	Saint Paul, Minnesota	1957-03-04	66740	1902
2	ABT	Abbott	Health Care	Health Care Equipment	North Chicago, Illinois	1957-03-04	1800	1888
5	ADBE	Adobe Inc.	Information Technology	Application Software	San Jose, California	1997-05-05	796343	1982
7	AES	AES Corporation	Utilities	Independent Power Producers & Energy Traders	Arlington, Virginia	1998-10-02	874761	1981
8	AFL	Aflac	Financials	Life & Health Insurance	Columbus, Georgia	1999-05-28	4977	1955
...	...	...	...	...	...	...	...	...
495	WYNN	Wynn Resorts	Consumer Discretionary	Casinos & Gaming	Paradise, Nevada	2008-11-14	1174922	2002
496	XEL	Xcel Energy	Utilities	Multi-Utilities	Minneapolis, Minnesota	1957-03-04	72903	1909
498	YUM	Yum! Brands	Consumer Discretionary	Restaurants	Louisville, Kentucky	1997-10-06	1041061	1997
500	ZBH	Zimmer Biomet	Health Care	Health Care Equipment	Warsaw, Indiana	2001-08-07	1136869	1927
501	ZION	Zions Bancorporation	Financials	Regional Banks	Salt Lake City, Utah	2001-06-22	109380	1873

های بعد از ۲۰۱۰ ticker

## پاسخ ۲.۱. کاوش در داده های سری زمانی و آشنایی با تعوری ها و کتابخانه های معروف

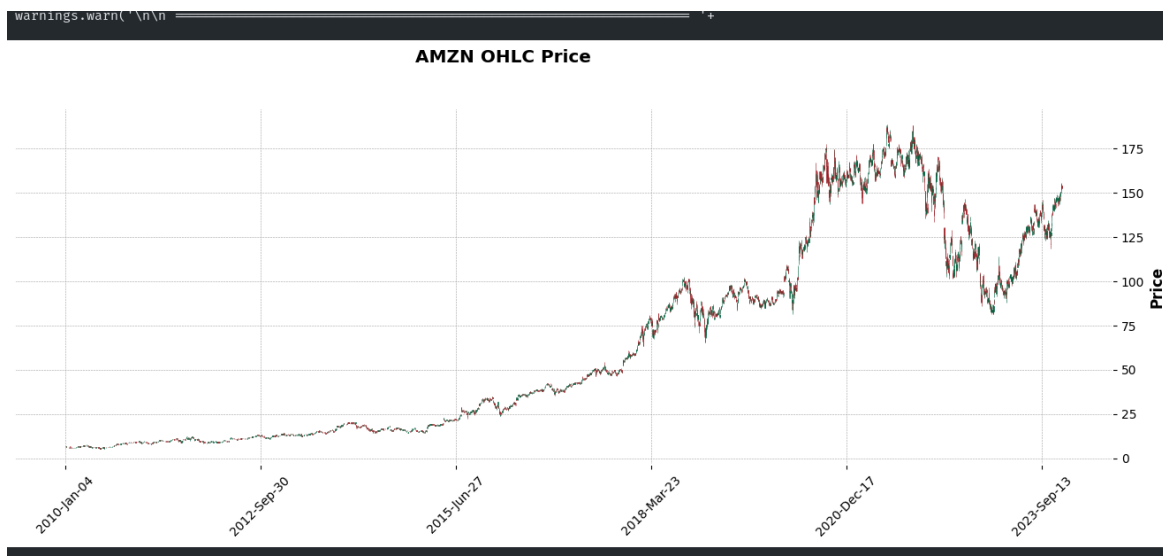
در صورت وجود داده های null در ابتدای سری زمانی می توانیم از روش های متفاوت مانند back fill استفاده کنیم که در واقع اولین دیتای داخل سری زمانی که دارای دیتا است را به عقب propagate خواهیم کرد. به این روش Back Filling گفته می شود که برای هندل کردن داده هایی است که در ابتدا یا قبل از دیتاپوینت های در منطقه چگالی بالا دیتا است استفاده می شود. در یک روش دیگر می توانیم به کل nan ها را حذف کنیم. و یا بازه زمانی ای را در نظر بگیریم که برای آن داده داریم. در صورت وجود داده ها در وسط سری زمانی می توانیم از روش Forward Filling استفاده کنیم در این روش اولین داده غیر nan روز های قبل استفاده خواهد شد. و آن را به جلو Propagate می کنیم. در روش های دیگر می توانیم Frequency داده ها را تغییر دهیم. مثلاً دیتا در اسکیل ثانیه ای پس و پیش موجود نیست ولی در اسکیل ساعتی موجود است و در نتیجه می توانیم از اسکیل ساعتی استفاده کنیم که برای آن فرقی ندارد که در یک سری ثانیه ها دیتا موجود نیست در این روش می توانیم از اولین دیتای یک time bucket و یا آخر و یا اول آن استفاده کنیم.



نمودار Close Return برای سهم آمازون

با توجه به نظریه بازار کارا و همچنین نظریه Random Walk قیمت یک سهم در یک بازار کارا از پترن خاصی پیروی نکرده و یک Random Walk را تشکیل می دهد . دلیل این موضوع می تواند اتفاقات میکرو و ماکروی بسیار زیادی باشد که روی عرضه و تقاضا و در نتیجه قیمت معامله شده یک سهم تاثیر می گذارد و تغییرات آن را رندم می کند همچنین چون این اتفاقات بسیاری از مواقع از هم مستقل هستند با توجه به Central Limit Theorem تجمیع توزیع های متفاوت مستقل از یکدیگر می تواند یک توزیع Gaussian یا نرمال را تولید کند همانند بسیاری از پدیده های طبیعی.

همچنین یک تعبیر از Return این است که اطلاعات اضافی امروز درباره دیروز را شامل می شود. اگر Random Walk Theory درست باشد این Return ها باید حاصل از یک Random Walk باشد که در این صورت با نموداری که می بینیم تطابق خواهد داشت چون Return حاصل جمع قیمت دیروز با یک اختلاف است که این اختلاف حاوی اطلاعات جدید امروز است چون بهترین Indicator قیمت امروز قیمت دیروز است پس قطعات اطلاعات جدید بسیار رندم و مستقل از هم و با توزیع های متفاوت و همچنین از نوعی نویز غیر قابل پیش بینی هستند در نتیجه با احتمال بالایی توزیع آن ها فرم زنگوله ای خواهد داشت.



نمودار Candlestick سهم آمازون

### پاسخ ۱-۳ . Time Series Split

وقتی که با داده های سری زمانی کار میکنیم نیاز داریم که دیتا را طوری به مدل بدهیم که مدل داده های آینده را نبینند و روی آن Train نشود یعنی به شکلی که Validation آن درباره آینده باشد و به شکلی که Train آن روی گذشته همان آینده باشد. با استفاده از TimeSeriesSplit می توانیم به طور Sequential دیتا تولید کنیم و به شکلی دادگان تولید کنیم که Cheat نکرده و Look Ahead Bias نداشته باشیم. این فرآیندی است که در واقع قبل از Train کردن مدل Implement کرده ایم و به چند دلیل در ابتدای کار انجام ندادیم تا بتوانیم بعضی کار ها را با دادگان انجام دهیم قبل اینکه به فرمی دربیايد که نتوانیم از آن به طور یکپارچه استفاده کنیم.

### ۱-۴ آماده سازي ورودی و خروجی مدل

در این قسمت ورودی های LSTM را تولید میکنیم و به شکل یک لیست از دو Tensor که یکی شامل  $y$  که قیمت روز های بعد از یک tensor به سایز window size که شامل روز های قبل از روزی است که می خواهیم پیش بینی کنیم خروجی می دهیم همچنین در این قسمت با استفاده از MinMaxScaler داده ها را به بازه ۰ تا ۱ هل

داده ایم .

```
> 1 def generate_tickers(h_input, window_size=90):
2     closes = h_input
3     y_tickers_sp500 = []
4     x_tickers_sp500 = []
5     X_time_series = []
6     Y_time_series = []
7     for i in range(window_size, len(h_input)):
8         X_time_series.append(closes.iloc[i - window_size : i])
9         Y_time_series.append(closes[["Close"]].iloc[i])
10
11     X_time_series = np.array(X_time_series)
12     Y_time_series = np.array(Y_time_series)
13     x_tickers_sp500.append(X_time_series[(31 - window_size) :])
14     y_tickers_sp500.append(Y_time_series[(31 - window_size) :])
15     x_tickers_sp_500 = torch.tensor(x_tickers_sp500[0])
16     y_tickers_sp_500 = torch.tensor(y_tickers_sp500[0])
17     return (x_tickers_sp_500, y_tickers_sp_500)
18
19
20 # TODO Train without scaling and report performance
[29] ✓ 0.0s
```

```
> 1 sample_tickers= generate_tickers(wanted_symbols_ohlc_normalized['GOOGL'])
2 sample_tickers
[31] ✓ 0.8s
```

```
... (tensor([[[[0.1141, 0.1141, 0.1150, 0.1153, 0.1153, 0.0552],
[0.1324, 0.1350, 0.1352, 0.1376, 0.1376, 0.1099],
[0.0168, 0.0168, 0.0174, 0.0171, 0.0171, 0.1532],
...,
[0.9512, 0.9603, 0.9657, 0.9622, 0.9622, 0.0259],
[0.9268, 0.9299, 0.9384, 0.9379, 0.9379, 0.0200],
[0.0027, 0.0027, 0.0029, 0.0037, 0.0037, 0.1669]]]]))
```

## 1-5 مدل های شبکه عصبی حافظه دار

هر سه شبکه برای حل کردن مشکل Vanishing Gradient در شبکه های عصبی Recurrent به کار می روند و کار آن ها پردازش داده هایی است که به شکل دنباله هستند . در داخل هر واحد این شبکه های عصبی gate هایی به کار رفته که هرکدام وظیفه ای دارند .

**LSTM** : واحد ها به شکل sequential به هم دیگر وصل شده اند و خروجی هر یونیت به یونیت بعدی می رود

دو نوع State داریم که یکی از ان ها Hidden State و دیگری State نامیده می شود

Forget Gate : این gate می تواند بخشی از اطلاعات را از state واحد فعلی پاک کند و از فرمول زیر استفاده می کند

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

Input Gate : این Gate تصمیم میگیرد که چه اطلاعاتی را در داخل استیت واحد بروز کند و از فرمول زیر استفاده می کند

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

Output Gate : این Gate تصمیم میگیرد که خروجی و ورودی Hidden State بعدی چه باشد و از فرمول زیر استفاده می کند

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$

سپس یک منطق برای اپدیت کردن استیت واحد داریم که از فرمول های زیر استفاده می کند در کلیه فرمول ها W به معنای وزن است و x به معنی ورودی و h به معنای state خواهد بود و سپس از فرمول های زیر برای اپدیت کردن هر استیت استفاده می کنیم

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

$$h_t = o_t * \tanh(C_t)$$

**Bi-LSTM** همانگونه که از اسم ان پیداست از دو LSTM استفاده می کند که یکی از آن ها داده را از اول به آخر پردازش کرده و سپس دومی داده را از آخر به اول پردازش می کند و تمام gate های این دو LSTM با LSTM قبلی مشترک است برای تعیین کردن خروجی از هر دو LSTM استیت پنهان اخر را برداشته و باهم concat کرده و به عنوان خروجی استفاده میکنیم. پس این شبکه مقدار خوبی شباهت به LSTM دارد.



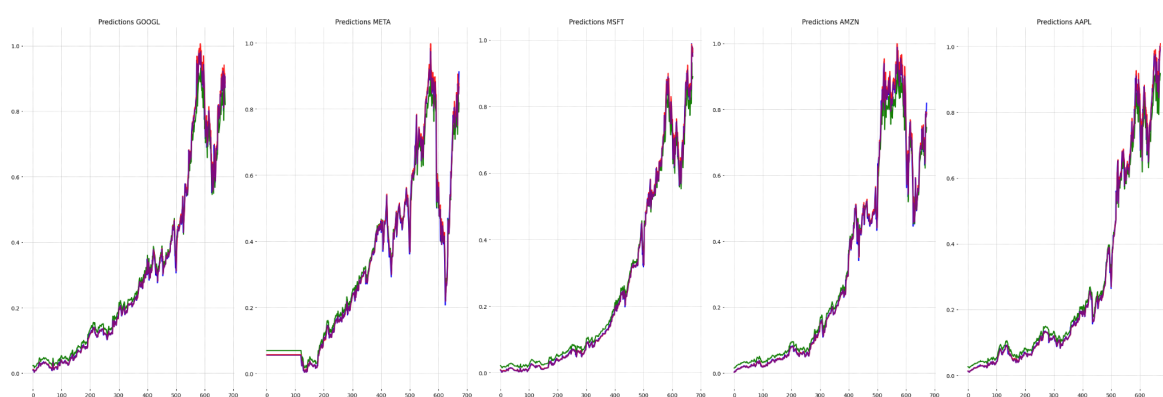
## GRU

این RNN ساختار LSTM را ساده تر کرده و دو Gate را باهم ترکیب کرده است همچنین Cell State و Hidden State دیگر دو مفهوم جداگانه نیستند و از Update Gate برای درجه مقداری که برای اپدیت کردن Activation و همچنین مقداری که از state قبلی نگه میدارید استفاده می کند

از Reset Gate برای تصمیم گیری درباره فراموش کردن اطلاعات قدیمی استفاده می کند

## نحوه آموزش

آموزش این شبکه ها مانند شبکه های عصبی دیگر نیز از Backpropagation استفاده می کند با این تفاوت که Back Propagation این نوع شبکه ها از نوع زمانی یا Time Back Propagation است که ابتدا Network را در زمان باز کرده و سپس Back Propagation را در ساختار باز شده شبکه که مرتبط به زمان است انجام می دهد. این شبکه ها ورودی خود را به شکل داده هایی در قالب دنباله یا Sequence می بینند و در نتیجه ترتیب برای آنها مهم است



نمودار بالا پیش بینی های مدل های GRU (سبز رنگ)

Bi LSTM ( قرمز رنگ )

LSTM (سبز رنگ)

## درباره ارور ها

- **MAPE** یک درصد است که حاصل جمع قدر مطلق اختلاف واقعیت و پیش بینی تقسیم بر پیش بینی تقسیم بر تعداد و ضرب در ۱۰۰ درصد است برای اینکه درصد بشود . نکته خوب درباره MAPE این است که از Scale Independent است در نتیجه با زیاد شدن قیمت سهم برای مثال می توان هنوز معیار خوبی برای اختلاف قیمت های پیش بینی شده و قیمت واقعی داشت که از Scale اعداد Independent هستند. همچنین مشخصه های خوب MAE را دارد که در پایین توضیح می دهیم.
- **MAE** برعکس MAPE یک کسر تبدیل شده نیست که میانگین قدر مطلق پیش بینی و واقعیت را نشان می دهد و چون از جنس همان اعدادی است که می گیرد و آن ها را وارد تابع های درجه چند نمی کند خطایی است که حس کلی ای نسبت به تفاوت یا نویز پیش بینی به صورت دقیق دارد و از یک واحد با واقعیت و پیش بینی است . همچنین این نوع ارور همه ارور ها را به شکل یکسان می گیرد چون قدر مطلق و اختلاف ساده است و باعث نمی شود که یک ارور خاص punish شود برای همین نسبت به MSE robust تر است . این متریک وقتی استفاده می شود که یک تعبیر مستقیم و غیر پیچیده از ارور نیاز داریم و همچنین وقتی استفاده می شود که توزیع ارور انتظار نمی رود که نرمال باشد.
- **MSE** این نوع ارور اختلاف را به توان دو رسانده تا علامت آن را از بین ببرد و در نتیجه ارور ها را بزرگ تر می کند چون به توان ۲ می رساند استفاده کردن از این متریک کمک کننده است چون ارور ها را به شکل سنگین ترین نسبت به قبل

پانیش می کند ولی در عوض قابلیت تفسیر خطا را پایین تر می آورد .

#### نتایج Train و Validation

```
1 Epoch 50/50,  
2 Train Loss : 0.00015876523684710264 Epoch 50/50,  
3 Train MAE Value: 0.009602880105376244 Epoch 50/50,  
4 Train MSE Value: 0.00015876523684710264 Epoch 50/50,  
5 Train MAPE Value: 1.2495348453521729 Epoch 50/50,  
6 Validation Loss: 0.00011803489178419113 Epoch 50/50,  
7 Validation MAE Value: 0.009006072767078876 Epoch 50/50,  
8 Validation MSE Value: 0.00011803489178419113 Epoch 50/50,  
9 Validation MAPE Value: 0.9912139773368835  
10
```

#### MLP Validation and Train loss at the the end of training

```
1  
2 Train Loss : 2.2790100047132e-06 Epoch 50/50,  
3 Train MAE Value: 0.0005451450124382973 Epoch 50/50,  
4 Train MSE Value: 2.2790100047132e-06 Epoch 50/50,  
5 Train MAPE Value: inf Epoch 50/50,  
6 Validation Loss: 4.986322528566234e-05 Epoch 50/50,  
7 Validation MAE Value: 0.005741837900131941 Epoch 50/50,  
8 Validation MSE Value: 4.986322528566234e-05 Epoch 50/50,  
9 Validation MAPE Value: 1.1759973764419556  
10
```

#### Bi LSTM Training and Loss at the end

```
1 Epoch 50/50,  
2 Train Loss : 0.00015857162361498922 Epoch 50/50,  
3 Train MAE Value: 0.009561282582581043 Epoch 50/50,  
4 Train MSE Value: 0.00015857162361498922 Epoch 50/50,  
5 Train MAPE Value: 1.2435681819915771 Epoch 50/50,  
6 Validation Loss: 0.00011234272824367508 Epoch 50/50,  
7 Validation MAE Value: 0.00886174663901329 Epoch 50/50,  
8 Validation MSE Value: 0.00011234272824367508 Epoch 50/50,  
9 Validation MAPE Value: 0.9726211428642273
```

#### LSTM Training and Loss at the end

```

1 Train Loss : 9.632241244617035e-07 Epoch 33/50,
2 Train MAE Value: 0.0007290191133506596 Epoch 33/50,
3 Train MSE Value: 9.632241244617035e-07 Epoch 33/50,
4 Train MAPE Value: 5.498422145843506 Epoch 33/50,
5 Validation Loss: 8.707779102223867e-07 Epoch 33/50,
6 Validation MAE Value: 0.0007237766403704882 Epoch 33/50,
7 Validation MSE Value: 8.707779102223867e-07 Epoch 33/50,
8 Validation MAPE Value: 2.4908320903778076
9

```

#### GRU training and loss at the end

```

1 Train Loss : 0.00016180536476895213 Epoch 50/50,
2 Train MAE Value: 0.009689419530332088 Epoch 50/50,
3 Train MSE Value: 0.00016180536476895213 Epoch 50/50,
4 Train MAPE Value: 1.2593879699707031 Epoch 50/50,
5 Validation Loss: 0.00010870080586755648 Epoch 50/50,
6 Validation MAE Value: 0.008700436912477016 Epoch 50/50,
7 Validation MSE Value: 0.00010870080586755648 Epoch 50/50,
8 Validation MAPE Value: 0.9522894620895386
9

```

#### CNN training and loss at the end

```

1 Epoch 50/50
2 Train Loss : 2.0452348508115392e-06 Epoch 50/50
3 Train MAE Value: 0.0005352533771656454 Epoch 50/50
4 Train MSE Value: 2.0452348508115392e-06 Epoch 50/50
5 Train MAPE Value: inf Epoch 50/50
6 Validation Loss: 0.00030142173636704683 Epoch 50/50
7 Validation MAE Value: 0.013794311322271824 Epoch 50/50
8 Validation MSE Value: 0.00030142173636704683 Epoch 50/50
9 Validation MAPE Value: 3.346870183944702
10

```

#### Conv LSTM Train & Validation Loss

## نتیجه استفاده نکردن از Scaling برای داده ها

ابتدا نتایج بعضی مدل هایی که ترین شده اند را روی داده های Scale شده میبینیم

LSTM = سبز

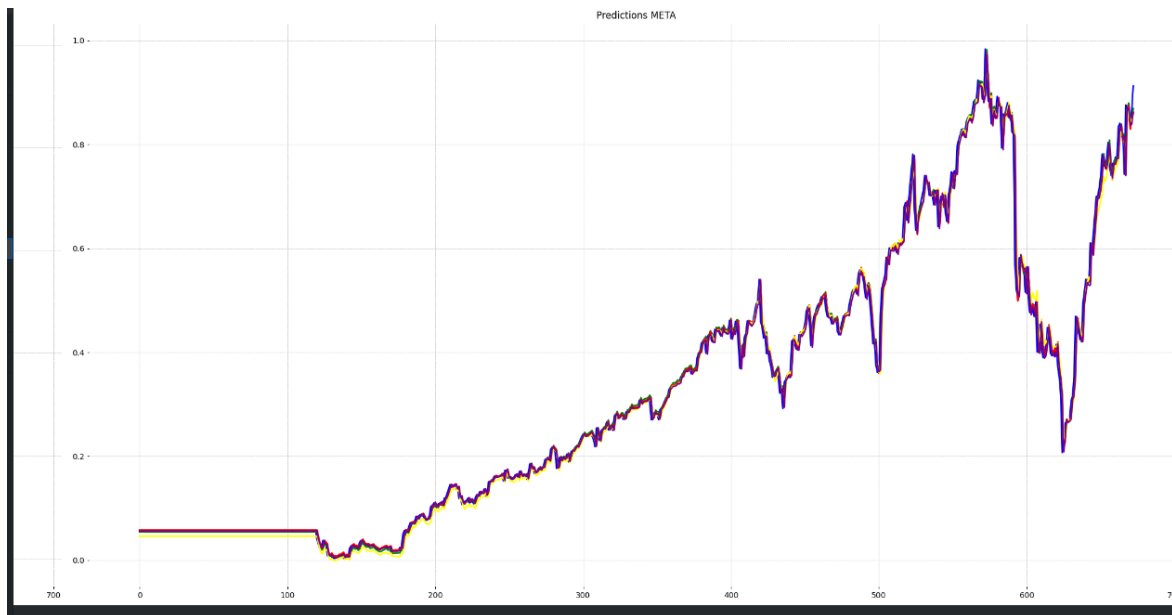
آبی = سهم اصلی

GRU = بنفش

قرمز = Bi LSTM

MLP = صورتی

Conv LSTM = سیاه



## عملکرد روی داده Scale شده

```
Model : <class '__main__.GRUNet'>
Test MAE Value: 0.009411647412797375 Test MSE Value: 0.000356031023520545
Test MAPE Value: 4.0194237133991875

Model : <class '__main__.LSTMBidirectionalSupportingNetwork'>
Test MAE Value: 0.009612590035469928 Test MSE Value: 0.0003568254817452168
Test MAPE Value: 4.7304205297486694

Model : <class '__main__.LSTMNet'>
Test MAE Value: 0.009830762473864668 Test MSE Value:
0.00035153377299276215 Test MAPE Value: 5.838442574682189

Model : <class '__main__.SimpleMLP'>
Test MAE Value: 0.013940928515819823 Test MSE Value:
0.00041492124455413005 Test MAPE Value: 13.456184867178166

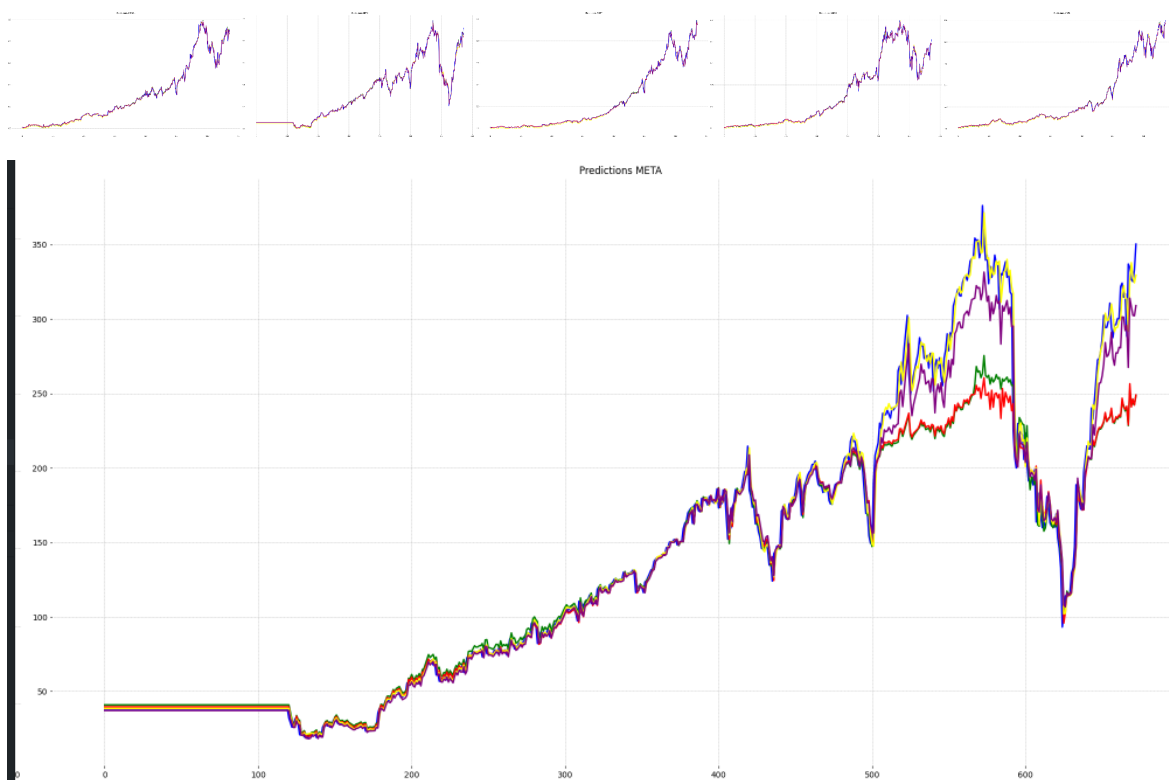
Model : <class 'Naive Forecast'>
```

```

Test MAE Value: 0.00921957838134566 Test MSE Value: 0.00034736469240984324
Test MAPE Value: 3.9747886424122574
Model : <class '__main__.CNN_LSTM'>
Test MAE Value: 0.22508162779325097 Test MSE Value: 0.12921181294836223
Test MAPE Value: 69.26961666547906

```

همانطور که در شکل زیر مشخص است ( قیمت های پیش بینی شده برای فیس بوک نشان داده شده ند) نمودار با رنگ آبی قیمت اصلی سهام است نمودار قرمز رنگ و سبز رنگ و همچنین نمودار بنفش Recurrent هستند که به مراتب بدتر از حالتی که Scale کرده بودیم عمل کرده اند (همچنین Loss و Error حاصل از تست را می توانیم مشاهده کنیم) دلیل اینکه Scaling تاثیر بدی روی LSTM و هم خانواده های آن دارد به این دلیل است که حساسیت آن به دلیل مستقیما و یک لایه خطی بودن روی داده های ورودی بسیار بالاست و همچنین چون تعداد Recurrent Unit ها کم است باعث می شود که Feed Back بزرگی از داده های غیر خطی وقتی ورودی می دهیم در بقیه State ها ایجاد شود. همچنین LSTM ها دارای مشکلات Vanishing و Exploding Gradients هستند و داده های Unscaled می توانند این موارد را تشدید کنند همین الگو را در نتایجی که از Conv LSTM گرفته ایم می توانیم مشاهده کنیم همچنین Sigmoid انتظار دارد که مقدار هایی که در بازه ۰ تا ۱ هستند بگیرد و کل داده های باید بین ۰ و ۱ باشند تا LSTM ها رفتار عادی خود را انجام دهند و در نتیجه داده اسکیل نشده بسیار مضر خواهد بود.



عکس ها با کیفیت بهتر در داخل فایل آپلودی وجود دارند و قابل مشاهده هستند  
همانطوری که در پایین میبینیم عملکرد ریکارنت یونیت ها بسیار بدتر از MLP و Naive Forecast است (نسبت به حالتی که Scale شده بود دیتا)

```

Model : <class '__main__.GRUNet'>

```

```

Test MAE Value: 2.8468125769854655 Test MSE Value: 31.124696579607516 Test
MAPE Value: 3.909458693504471

Model : <class '__main__.LSTMBidirectionalSupportingNetwork'>

Test MAE Value: 58.42331218356364 Test MSE Value: 6408.640598998081 Test
MAPE Value: 100.40749787656038

Model : <class '__main__.LSTMNet'>

Test MAE Value: 58.31268070051779 Test MSE Value: 6394.1261800137445 Test
MAPE Value: 100.02635410694963

Model : <class '__main__.SimpleMLP'>

Test MAE Value: 1.8305975119036693 Test MSE Value: 12.783340291214007 Test
MAPE Value: 3.0081611903920566

Model : <class '__main__.Simple1DCNN'>

Test MAE Value: 1.813053447105555 Test MSE Value: 12.717143913269588 Test
MAPE Value: 3.053542101845375

Model : <class 'function'>

Test MAE Value: 1.761871320886031 Test MSE Value: 12.685627211113918 Test
MAPE Value: 2.8174181811962113

Model : <class '__main__.CNN_LSTM'>

Test MAE Value: 58.279686490452974 Test MSE Value: 6395.195318740368 Test
MAPE Value: 99.79732124337897

```

### تحلیل نتایج مدل ها

با توجه به نتایج به دست آمده ( در قسمت بالا ارور های هر بخش قراره داده شده ) بهترین مدل نسبت به Naive Forecast GRU است و BiLSTM مانند مقاله بهتر از LSTM عادی عمل کرده است . بقیه شبکه ها مانند CNN و MLP نسبت به هر ۳ LSTM based یا RNN ها بدتر عمل کرده اند که انتظار می رفت همینطور باشد چون عملا LSTM و BiLSTM از این شبکه ها در انتهای خود برای انجام Regression استفاده می کنند و ذاتا نوعی Preprocessing برای این شبکه ها حساب می شوند . دلیل اینکه BiLSTM بهتر نسبت به LSTM عمل می کند

۱. به دلیل این است که تعداد Hidden unit های بیشتری دارد ( در ۲ ضرب می شود ) و مدل پیچیده ای است  
 ۲. در داخل خودش نوعی Augmentation در حال انجام دادن است و داده های Ticker مربوطه را یکبار از ته به اول و یکبار از اول به ته می خواند که می تواند باعث شود اطلاعات بیشتری را از روند سهم دریافت کند ( با اینکه ذاتا داده های آن یکسان هستند )

۳. همچنین چون BiLSTM دو قسمت است عملا واریانس آن باید بهتر باشد چون نوعی Wisdom of the Crowd را در داخل خود Implement کرده و از جواب دو LSTM استفاده می کند .

همچنان به دلیل تعداد داده کم و مقدار کم Train شدن نتورک و همچنین درست انجام ندادن Hyper Parameter Fine Tuning و رسیدن به اعداد معقول برای تعداد Hidden State نتوانسته ایم بهتر از Naive Forecast عمل کنیم

**دلیل بهتر بودن GRU نسبت به Bi LSTM**

این نوع شبکه Efficient تر Peformant تر است و همانطور که در Notebook مشاهده می شود در زمان کمتری Train می شود و همچنین ساده تر نیز است . به دلیل همین سادگی است که ممکن است باعث شده باشد که Overfitting کمتری نسبت به Bi LSTM انجام شده باشد . به دلیل مکانیزم های ساده تر Gating این نوع RNN می تواند سریع تر Converge کند . در کل دو مدل مقدار زیادی از یکدیگر فاصله ندارند و در متریک MSE بسیار شبیه یکدیگر و در متریک های دیگر متفاوت هستند

## 6-۱. Forecast Naïve

با توجه به این در حال بررسی یکی از بزرگ ترین بازار های مالی دنیا هستیم که شاید نزدیک ترین نمونه به بازار هایی باشد که است و انتظار می رود که بهتر از efficient درباره آن صحبت می کند و در نتیجه به مقدار خوبی کارا و Random Walk نظریه Naïve Forecast پیش بینی کردن سخت باشد که ثابت کردیم هست. بهتر از مدل های ما عمل می کند . Scaled و چه غیر Scaled با اختلاف چه Naïve Forecast در دو خروجی بالا می بینیم که ها می LSTM بهتر برای Window size احتمالا با جستجو و پیدا کردن پارامتر های بهتر برای شبکه های عصبی و پیدا کردن (توانیم به نتایج بهتری دست پیدا کنیم (همانند مقاله

```
Model : <class '__main__.GRUNet'>
```

```
Test MAE Value: 0.00989416360621541 Test MSE Value: 0.0003747091786817085
Test MAPE Value: 4.16689051298975
```

```
Model : <class '__main__.LSTMBidirectionalSupportingNetwork'>
```

```
Test MAE Value: 0.009612590035469928 Test MSE Value: 0.0003568254817452168
Test MAPE Value: 4.7304205297486694
```

```
Model : <class '__main__.LSTMNet'>
```

```
Test MAE Value: 0.009830762473864668 Test MSE Value:
0.00035153377299276215 Test MAPE Value: 5.838442574682189
```

```
Model : <class '__main__.SimpleMLP'>
```

```
Test MAE Value: 0.013940928515819823 Test MSE Value:
0.00041492124455413005 Test MAPE Value: 13.456184867178166
```

```
Model : <class 'function'>
```

```
Test MAE Value: 0.00921957838134566 Test MSE Value: 0.00034736469240984324
Test MAPE Value: 3.9747886424122574
```

```
Model : <class '__main__.Simple1DCNN'>
```

```
Test MAE Value: 0.010657007854961582 Test MSE Value: 0.0003712811376826124
Test MAPE Value: 5.888829931710761
```

```
Model : <class '__main__.CNN_LSTM'>
```

```
Test MAE Value: 0.22508162779325097 Test MSE Value: 0.12921181294836223
Test MAPE Value: 69.26961666547906
```

## پرسش ۲. پیش بینی افکار خودکشی در رسانه های اجتماعی

### ۲.۱. پیش پردازش داده

```
1 print(df.loc[0,"tweet"], '\n\n')
2 print(df.loc[0,"clean_text"])
3 print(df.loc[0,"clean_text2"])
[11] ✓ 0.0s
.. my life is meaningless i just want to end my life so badly my life is completely empty and i dont want to have
life meaningless want end life badli life complet empti dont want creat mean creat mean pain long hold back ur
life meaningless want end life badly life completely empty dont want create meaning creating meaning pain long
```

#### 2.1.preprocessing

در اینجا ابتدا کارکتر های خاص را حذف نموده و سپس از tokenizer برای توکن کردن جمله ها استفاده کردیم. و سپس word stop های را حذف کردیم کلماتی مانند the, ... و در آخر کار از lemmatizer برای پیدا کردن ریشه کلمات استفاده کردیم.

### ۲.۲. ساخت ماتریس جاسازی

همان طور که در مقاله به آن اشاره شده بود از مدل از پیش آموزش دیده google استفاده کردیم. ابتدا مدل local خودمان را روی داده ها ترین کردیم. سپس وزن هایی که بین vocabulary این دو مدل یکی بود را با استفاده از مدل google آپدیت کردیم. و در آخر تعدادی از کلماتی که در مدل گوگل نبودند به علت typo ، آنها را به صورت hardcoded آپدیت کردیم و پس از، مدل رو دوباره به روی کلماتی که در گوگل نبودند آموزش دادیم تا embed vector آن ها را نیز یاد بگیرد. روش های دیگر و ساده تری نیز برای بخش وجود دارد. مانند حذف کامل توکن هایی که در گوگل نیست (OOV).

علت استفاده از این ماتریس این هست. که از طریق این ماتریس می توانیم به صورت خیلی فشرده تر هر توکن یکتا را بیان کنیم. در واقع چیزی که به مدل feed میشود، وکتوری از اندیس ها میباشد. یعنی هر کلمه با یک عدد که اندیس آن در ماتریس embedding هست جایگزین میشود. این روش بهتری برای نمایش جمله ها و فید کردن آن ها به مدل میباشد. در مدل داخل این اندیس ها از یک لایه embedding رد میشوند. و هرکدام به یک vector تبدیل می شوند. که ابعاد این وکتور نشان دهنده feature dimension در نظر گرفته شده برای هر توکن میباشد. هر چقدر این بعد بیشتر باشد، ما در واقع اطلاعات بیشتری برای هر توکن ذخیره میکنیم. با استفاده از embedding ماتریس، ما درواقع کلمات را به یک فضای پیوسته map میکنیم. به جای اینکه از روش های one-hot encode استفاده کنیم.

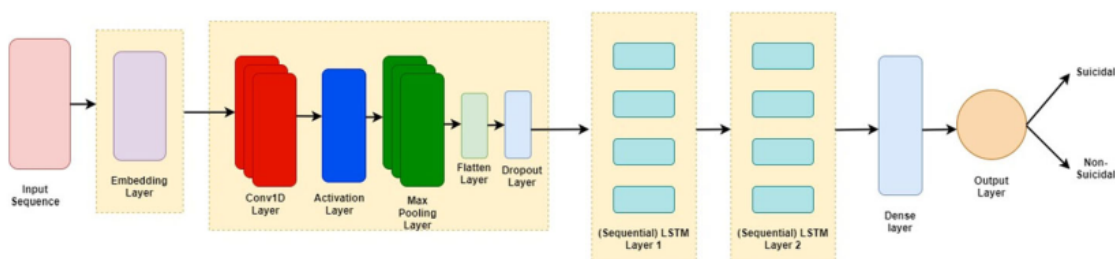
این ماتریس در واقع مانند یک دیکشنری عمل میکند و هر کلمه یا اندیس آن را به صورت دقیق تر به یک وکتوری n بعدی میبرد. که در واقع یک فضای پیوسته n بعدی میباشد. پر آن فضا اعداد مرتبط بهم در کنار همدیگر قرار



میگیرند. در صورتی که این مدل ها به خوبی آموزش دیده باشند، امکان انجام اعمال ریاضی مانند زیر هم ممکن است:

king - man + woman = queen

### ۲.۳. آموزش مدل های یادگیری عمیق



### 2.2.CNN + 2-layer LSTM

با توجه به ساختار ذکر شده در مقاله مدل را پیاده سازی کردیم.

LSTM:

```
... 100%|██████████| 100/100 [00:14<00:00, 7.10it/s]
train_loss: 0.3808263210952282
Epoch: 01, Train Loss: 0.381, Train Acc: 81.62%, Val. Loss: 0.267672, Val. Acc: 89.10%
100%|██████████| 100/100 [00:13<00:00, 7.21it/s]
train_loss: 0.2171166642010212
Epoch: 02, Train Loss: 0.217, Train Acc: 91.34%, Val. Loss: 0.245338, Val. Acc: 88.96%
100%|██████████| 100/100 [00:13<00:00, 7.32it/s]
train_loss: 0.20094430677592753
Epoch: 03, Train Loss: 0.201, Train Acc: 91.90%, Val. Loss: 0.210629, Val. Acc: 91.27%
100%|██████████| 100/100 [00:13<00:00, 7.30it/s]
train_loss: 0.1749265770614147
Epoch: 04, Train Loss: 0.175, Train Acc: 92.90%, Val. Loss: 0.216291, Val. Acc: 91.34%
100%|██████████| 100/100 [00:13<00:00, 7.27it/s]
train_loss: 0.16630947411060334
Epoch: 05, Train Loss: 0.166, Train Acc: 93.40%, Val. Loss: 0.204669, Val. Acc: 92.04%
100%|██████████| 100/100 [00:13<00:00, 7.23it/s]
train_loss: 0.14208526324480772
Epoch: 06, Train Loss: 0.142, Train Acc: 94.14%, Val. Loss: 0.203467, Val. Acc: 92.46%
100%|██████████| 100/100 [00:14<00:00, 7.05it/s]
train_loss: 0.12439254809170962
Epoch: 07, Train Loss: 0.124, Train Acc: 95.08%, Val. Loss: 0.192820, Val. Acc: 92.67%
100%|██████████| 100/100 [00:13<00:00, 7.21it/s]
train_loss: 0.11639294855296611
Epoch: 08, Train Loss: 0.116, Train Acc: 95.42%, Val. Loss: 0.226516, Val. Acc: 91.65%
100%|██████████| 100/100 [00:13<00:00, 7.18it/s]
train_loss: 0.10329423336312175
Epoch: 09, Train Loss: 0.103, Train Acc: 96.15%, Val. Loss: 0.201532, Val. Acc: 92.90%
100%|██████████| 100/100 [00:13<00:00, 7.37it/s]
train_loss: 0.09006756560876966
Epoch: 10, Train Loss: 0.090, Train Acc: 96.61%, Val. Loss: 0.241807, Val. Acc: 92.46%
Total training time: 144.05111622810364s
```

LSTM2:

```
100%|██████████| 100/100 [00:16<00:00, 5.95it/s]
Epoch: 01, Train Loss: 0.348, Train Acc: 84.75%, Val. Loss: 0.255895, Val. Acc: 88.85%
100%|██████████| 100/100 [00:16<00:00, 5.92it/s]
Epoch: 02, Train Loss: 0.229, Train Acc: 90.61%, Val. Loss: 0.223276, Val. Acc: 91.06%
100%|██████████| 100/100 [00:16<00:00, 5.91it/s]
Epoch: 03, Train Loss: 0.201, Train Acc: 92.35%, Val. Loss: 0.218803, Val. Acc: 91.38%
100%|██████████| 100/100 [00:16<00:00, 6.08it/s]
Epoch: 04, Train Loss: 0.178, Train Acc: 92.87%, Val. Loss: 0.199719, Val. Acc: 91.77%
100%|██████████| 100/100 [00:16<00:00, 5.98it/s]
Epoch: 05, Train Loss: 0.176, Train Acc: 93.06%, Val. Loss: 0.210152, Val. Acc: 91.92%
100%|██████████| 100/100 [00:16<00:00, 6.03it/s]
Epoch: 06, Train Loss: 0.144, Train Acc: 94.39%, Val. Loss: 0.207227, Val. Acc: 90.88%
100%|██████████| 100/100 [00:16<00:00, 5.99it/s]
Epoch: 07, Train Loss: 0.142, Train Acc: 94.40%, Val. Loss: 0.197659, Val. Acc: 92.65%
100%|██████████| 100/100 [00:16<00:00, 5.94it/s]
Epoch: 08, Train Loss: 0.122, Train Acc: 95.29%, Val. Loss: 0.223333, Val. Acc: 91.38%
100%|██████████| 100/100 [00:16<00:00, 6.15it/s]
Epoch: 09, Train Loss: 0.109, Train Acc: 95.76%, Val. Loss: 0.202109, Val. Acc: 92.46%
100%|██████████| 100/100 [00:17<00:00, 5.85it/s]
Epoch: 10, Train Loss: 0.095, Train Acc: 96.37%, Val. Loss: 0.218448, Val. Acc: 91.64%
```

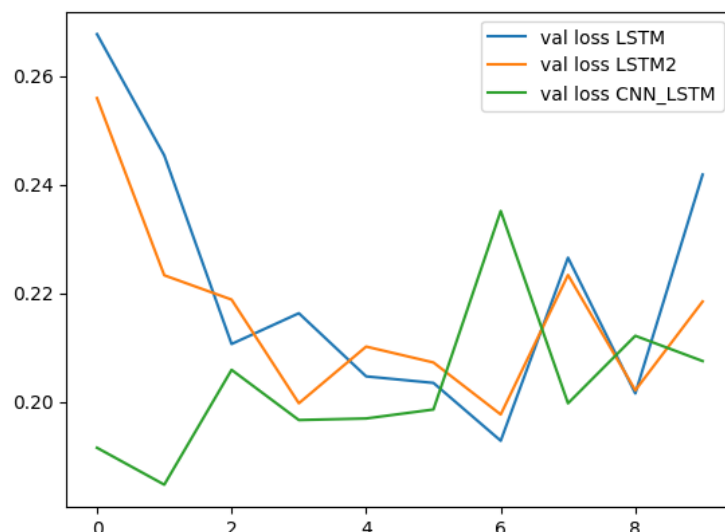
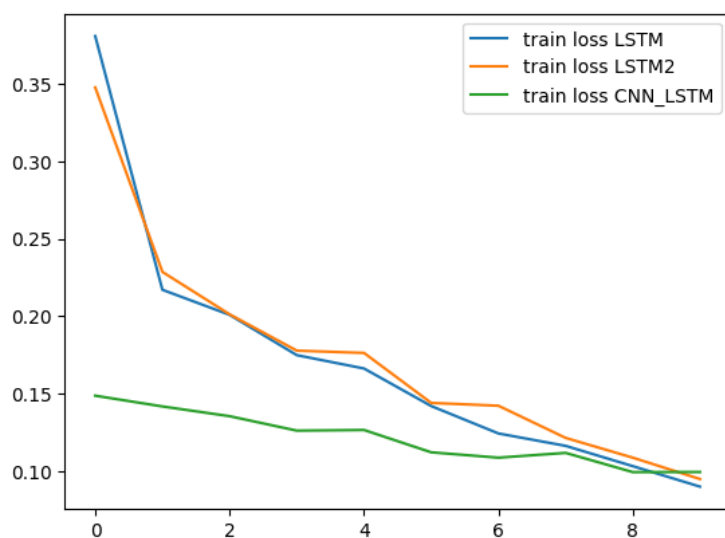
CNN-2-LAYER-LSTM:

```

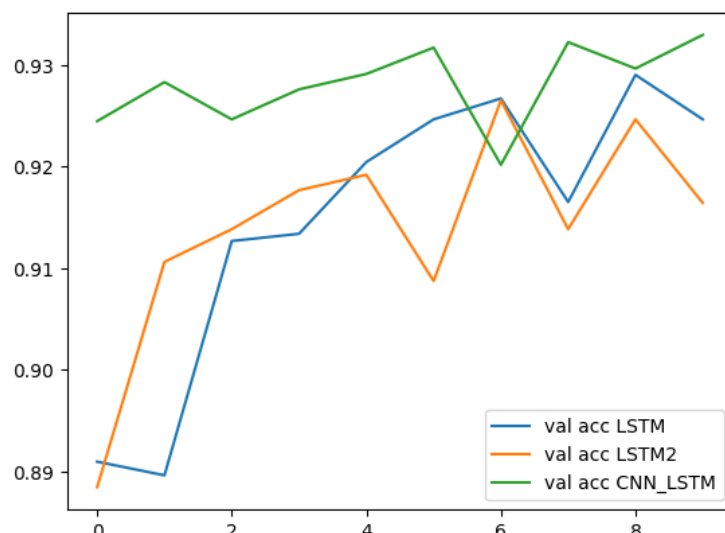
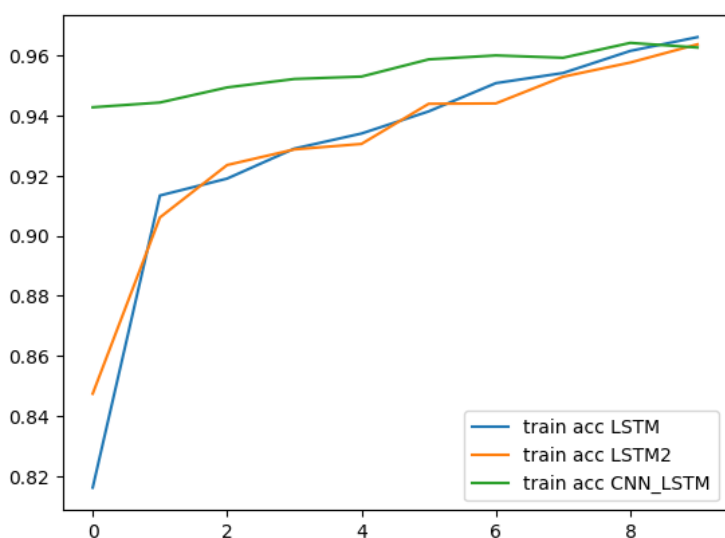
100%|██████████| 100/100 [00:13<00:00, 7.55it/s]
Epoch: 01, Train Loss: 0.373, Train Acc: 84.56%, Val. Loss: 0.258683, Val. Acc: 89.93%
100%|██████████| 100/100 [00:13<00:00, 7.35it/s]
Epoch: 02, Train Loss: 0.245, Train Acc: 90.59%, Val. Loss: 0.252355, Val. Acc: 89.06%
100%|██████████| 100/100 [00:13<00:00, 7.61it/s]
Epoch: 03, Train Loss: 0.218, Train Acc: 91.62%, Val. Loss: 0.222153, Val. Acc: 91.31%
100%|██████████| 100/100 [00:13<00:00, 7.38it/s]
Epoch: 04, Train Loss: 0.204, Train Acc: 92.11%, Val. Loss: 0.210352, Val. Acc: 91.85%
100%|██████████| 100/100 [00:13<00:00, 7.15it/s]
Epoch: 05, Train Loss: 0.198, Train Acc: 91.92%, Val. Loss: 0.204525, Val. Acc: 92.18%
100%|██████████| 100/100 [00:13<00:00, 7.47it/s]
Epoch: 06, Train Loss: 0.182, Train Acc: 93.14%, Val. Loss: 0.207471, Val. Acc: 91.63%
100%|██████████| 100/100 [00:13<00:00, 7.35it/s]
Epoch: 07, Train Loss: 0.176, Train Acc: 92.84%, Val. Loss: 0.216691, Val. Acc: 91.52%
100%|██████████| 100/100 [00:13<00:00, 7.45it/s]
Epoch: 08, Train Loss: 0.163, Train Acc: 93.66%, Val. Loss: 0.209167, Val. Acc: 92.60%
100%|██████████| 100/100 [00:13<00:00, 7.19it/s]
Epoch: 09, Train Loss: 0.158, Train Acc: 94.04%, Val. Loss: 0.217443, Val. Acc: 90.87%
100%|██████████| 100/100 [00:13<00:00, 7.46it/s]
Epoch: 10, Train Loss: 0.149, Train Acc: 94.11%, Val. Loss: 0.197322, Val. Acc: 92.89%
100%|██████████| 100/100 [00:13<00:00, 7.57it/s]
Epoch: 11, Train Loss: 0.149, Train Acc: 94.28%, Val. Loss: 0.191528, Val. Acc: 92.45%
100%|██████████| 100/100 [00:13<00:00, 7.21it/s]
Epoch: 12, Train Loss: 0.142, Train Acc: 94.43%, Val. Loss: 0.184735, Val. Acc: 92.83%
100%|██████████| 100/100 [00:13<00:00, 7.34it/s]
Epoch: 13, Train Loss: 0.136, Train Acc: 94.94%, Val. Loss: 0.205873, Val. Acc: 92.46%
100%|██████████| 100/100 [00:13<00:00, 7.22it/s]
Epoch: 14, Train Loss: 0.126, Train Acc: 95.22%, Val. Loss: 0.196630, Val. Acc: 92.76%
100%|██████████| 100/100 [00:14<00:00, 7.13it/s]
Epoch: 15, Train Loss: 0.127, Train Acc: 95.30%, Val. Loss: 0.196918, Val. Acc: 92.91%
100%|██████████| 100/100 [00:13<00:00, 7.35it/s]
Epoch: 16, Train Loss: 0.112, Train Acc: 95.87%, Val. Loss: 0.198572, Val. Acc: 93.17%
100%|██████████| 100/100 [00:13<00:00, 7.43it/s]
Epoch: 17, Train Loss: 0.109, Train Acc: 96.00%, Val. Loss: 0.235111, Val. Acc: 92.02%
100%|██████████| 100/100 [00:13<00:00, 7.27it/s]
Epoch: 18, Train Loss: 0.112, Train Acc: 95.92%, Val. Loss: 0.199714, Val. Acc: 93.22%
100%|██████████| 100/100 [00:13<00:00, 7.21it/s]
Epoch: 19, Train Loss: 0.099, Train Acc: 96.42%, Val. Loss: 0.212140, Val. Acc: 92.96%
100%|██████████| 100/100 [00:13<00:00, 7.47it/s]
Epoch: 20, Train Loss: 0.100, Train Acc: 96.26%, Val. Loss: 0.207492, Val. Acc: 93.29%

```

## ۲.۴. مقایسه نتایج



2.3.train & val loss of 3 models



2.4.train/val acc of 3 models

همان طور که مشاهده میکنید، مدل اخر `cnn_lstm` به علت استفاده از `cnn` که میتواند `spatial relation` داده ها را تحلیل کند، که در اینجا همان ارتباط یک کلمه با کلمات مجاورش در یک جمله میباشد، اندکی بهتر از دو مدل دیگر عمل کرده است. از طرفی مدل های دیگر دچار `overfit` میشوند. اما این مدل به علت استفاده از لایه `dropout`، قابلیت `generalization` بهتری دارد و در نتیجه روی داده `val` بهتر عمل میکند.

البته ممکن است تفاوت این مدل ها، در ایپاک های بالاتر بیشتر شود.