به نام خدا

تمرین پنجم

(شبکه‌های بازگشتی)

هوش مصنوعی و یادگیری ماشین

نام دانشجو:

امیرعلی محمودزاده طوسی

۸۱۰۶۰۳۱۴۲

استاد درس:

دکتر شریعت پناهی

تیر ۱۴۰۴

https://github.com/amiralimt/ai_hw5

مقدمه

هدف این تمرین، آشنایی با شبکه‌های عصبی بازگشتی و پیچشی برای مدل‌سازی پدیده‌های وابسته به زمان بود. در این راستا، از این شبکه‌ها برای پیش‌بینی عمر مفید باقیمانده (Remaining Useful Life) موتور توربوفن هواپیما استفاده شد.

مجموعه داده:

برای این منظور، از زیرمجموعه FD001 مجموعه داده C-MAPSS ناسا استفاده گردید. این مجموعه داده شامل اطلاعات سری زمانی از ۲۱ سنسور مختلف و ۳ تنظیم عملیاتی برای ۱۰۰ موتور است که تا زمان خرابی کامل کار کرده‌اند.

رویکرد کلی:

فرآیند انجام تمرین شامل مراحل زیر بود:

۱. **آماده‌سازی داده‌ها:** پاک‌سازی، انتخاب ویژگی، نرمال‌سازی و تبدیل داده‌ها به فرمت مناسب (پنجره‌های لغزان).
۲. **پیاده‌سازی مدل‌ها:** ساخت و آموزش پنج معماری مختلف شامل CNN, LSTM, CNN+LSTM, LSTM+CNN و LSTM+Attention.
۳. **تنظیم ابرپارامترها:** استفاده از ابزار خودکار KerasTuner برای یافتن بهترین تنظیمات برای مدل CNN.
٤. **ارزیابی و مقایسه:** تحلیل عملکرد تمام مدل‌ها با استفاده از معیارهای کمی و کیفی و انتخاب بهترین مدل.

ساختار کلی داده‌ها:

هر مجموعه داده از چندین سری زمانی چندمتغیره تشکیل شده است.

هر سری زمانی مربوط به یک موتور مجزا از ناوگانی با نوع یکسان است.

داده‌ها به دو زیرمجموعه آموزشی (Training) و آزمون (Test) تقسیم شده‌اند.

هر موتور در ابتدای کار دارای مقداری فرسودگی اولیه و تنوع ساخت است که برای ما نامشخص است. این فرسودگی اولیه، عادی تلقی شده و به عنوان خطا در نظر گرفته نمی‌شود.

شرایط عملیاتی و خطا:

موتور در ابتدای هر سری زمانی به طور عادی کار می‌کند و در نقطه‌ای از زمان دچار یک خطا می‌شود.

در مجموعه داده آموزشی، این خطا تا زمان از کار افتادن کامل موتور ادامه پیدا می‌کند.

در مجموعه داده آزمون، سری زمانی مدتی قبل از خرابی کامل، متوقف می‌شود.

داده‌ها شامل سه تنظیم عملیاتی هستند که تأثیر قابل توجهی بر عملکرد موتور دارند. همچنین داده‌ها با نویز حسگرها همراه هستند.

هدف مسئله:

هدف اصلی، پیش‌بینی تعداد چرخه‌های عملیاتی باقیمانده تا خرابی یعنی RUL: Remaining Useful Life برای موتورهای مجموعه آزمون است.

یک فایل مجزا حاوی مقادیر واقعی RUL برای داده‌های آزمون نیز ارائه شده است.

مشخصات زیرمجموعه FD001

این مجموعه داده که ما در تمرین از آن استفاده می‌کنیم، شامل ۱۰۰ سری زمانی برای آموزش و ۱۰۰ سری زمانی برای آزمون است.

فقط یک نوع شرایط عملیاتی (در سطح دریا) و یک نوع حالت خطا (فرسودگی کمپرسور فشار بالا HPC - Degradation) را پوشش می‌دهد. این موضوع، FD001 را به ساده‌ترین زیرمجموعه تبدیل می‌کند.

بخش الف: پیاده‌سازی مدل‌ها

مرحله ۱: آماده‌سازی و پیش‌پردازش داده‌ها

بارگذاری و انتخاب دادگان (Feature Selection):

ابتدا دادگان زیر مجموعه FD001 را در گوگل درایو بارگذاری می کنیم و ارتباط گوگل درایو و کولب را با دستور drive mount برقرار می کنیم و مسیر داده ها را مشخص میکنیم.

ما از کتابخانه pandas برای خواندن و مدیریت داده‌ها استفاده خواهیم کرد. این کتابخانه ابزاری قدرتمند برای کار با داده‌های جدولی است.

داده‌های مجموعه C-MAPSS در فایل‌های متنی (.txt) با مقادیر جدا شده توسط فاصله ذخیره شده‌اند و ستون‌ها نام مشخصی ندارند. بنابراین، ما باید خودمان نام ستون‌ها را تعریف کنیم.

بر اساس مستندات مجموعه داده، ستون‌ها به این ترتیب هستند:

- ستون ۱: شماره موتور(Unit Number)
- ستون ۲: چرخه زمانی (Time in cycles)
- ستون‌های ۵-۳: تنظیمات عملیاتی (Operational settings)
- ستون‌های ۲۶-۶: ۲۱ داده حسگر (Sensor measurements)

برای اینکه مطمئن شویم داده‌ها به درستی بارگذاری شده‌اند، ۵ سطر اول هر DataFrame نمایش داده می شود.

داده‌های آموزشی:

	unit_nr	cycle	op_setting_1	op_setting_2	op_setting_3	sensor_1	sensor_2	sensor_3	sensor_4	sensor_5	...	sensor_12	sensor_13	sensor_14	sensor_15	sensor_16	sensor_17	sensor_18	sensor_19	sensor_20	sensor_21
0	1	1	-0.0007	-0.0004	100.0	518.67	641.82	1589.70	1400.60	14.62	...	521.66	2388.02	8138.62	8.4195	0.03	392	2388	100.0	39.06	23.4190
1	1	2	0.0019	-0.0003	100.0	518.67	642.15	1591.82	1403.14	14.62	...	522.28	2388.07	8131.49	8.4318	0.03	392	2388	100.0	39.00	23.4236
2	1	3	-0.0043	0.0003	100.0	518.67	642.35	1587.99	1404.20	14.62	...	522.42	2388.03	8133.23	8.4178	0.03	390	2388	100.0	38.95	23.3442
3	1	4	0.0007	0.0000	100.0	518.67	642.35	1582.79	1401.87	14.62	...	522.86	2388.08	8133.83	8.3682	0.03	392	2388	100.0	38.88	23.3739
4	1	5	-0.0019	-0.0002	100.0	518.67	642.37	1582.85	1406.22	14.62	...	522.19	2388.04	8133.80	8.4294	0.03	393	2388	100.0	38.90	23.4044

5 rows × 26 columns

داده‌های آزمون:

	unit_nr	cycle	op_setting_1	op_setting_2	op_setting_3	sensor_1	sensor_2	sensor_3	sensor_4	sensor_5	...	sensor_12	sensor_13	sensor_14	sensor_15	sensor_16	sensor_17	sensor_18	sensor_19	sensor_20	sensor_21
0	1	1	0.0023	0.0003	100.0	518.67	643.02	1585.29	1398.21	14.62	...	521.72	2388.03	8125.55	8.4052	0.03	392	2388	100.0	38.86	23.3735
1	1	2	-0.0027	-0.0003	100.0	518.67	641.71	1588.45	1395.42	14.62	...	522.16	2388.06	8139.62	8.3803	0.03	393	2388	100.0	39.02	23.3916
2	1	3	0.0003	0.0001	100.0	518.67	642.46	1586.94	1401.34	14.62	...	521.97	2388.03	8130.10	8.4441	0.03	393	2388	100.0	39.08	23.4166
3	1	4	0.0042	0.0000	100.0	518.67	642.44	1584.12	1406.42	14.62	...	521.38	2388.05	8132.90	8.3917	0.03	391	2388	100.0	39.00	23.3737
4	1	5	0.0014	0.0000	100.0	518.67	642.51	1587.19	1401.92	14.62	...	522.15	2388.03	8129.54	8.4031	0.03	390	2388	100.0	38.99	23.4130

5 rows × 26 columns

داده‌های RUL واقعی برای آزمون:

	RUL
0	112
1	98
2	69
3	82
4	91

حذف ویژگی‌هایی با مقدار ثابت یا تغییرات بسیار کم (Low Variance Features)

- **منطق کار:** ویژگی یا حسگری که در طول تمام پروازها مقدار ثابتی دارد (یا تغییرات آن نزدیک به صفر است)، هیچ اطلاعات مفیدی برای پیش‌بینی خرابی به ما نمی‌دهد. چون هیچ روندی (trend) را نشان نمی‌دهد، برای مدل یادگیرنده بی‌ارزش است.
- **روش پیاده‌سازی:** ساده‌ترین راه برای شناسایی این ویژگی‌ها، محاسبه **انحراف معیار (Standard Deviation)** برای هر ستون در داده‌های آموزشی است. اگر انحراف معیار یک ستون صفر باشد، یعنی تمام مقادیر آن ستون یکسان هستند.

انحراف معیار برای هر ویژگی	
unit_nr	**2.922763e+01**
cycle	6.888099e+01
op_setting_1	2.187313e-03
op_setting_2	2.930621e-04
op_setting_3	0.000000e+00
sensor_1	6.537152e-11
sensor_2	5.000533e-01
sensor_3	6.131150e+00
sensor_4	9.000605e+00
sensor_5	3.394700e-12
sensor_6	1.388985e-03
sensor_7	8.850923e-01
sensor_8	7.098548e-02
sensor_9	2.208288e+01
sensor_10	4.660829e-13
sensor_11	2.670874e-01
sensor_12	7.375534e-01
sensor_13	7.191892e-02
sensor_14	1.907618e+01
sensor_15	3.750504e-02
sensor_16	1.556432e-14
sensor_17	1.548763e+00
sensor_18	0.000000e+00
sensor_19	0.000000e+00
sensor_20	1.807464e-01
sensor_21	1.082509e-01

همانطور که در خروجی مشاهده می‌شود:

- **انحراف معیار دقیقاً صفر:** op_setting_3 , sensor_18 , sensor_19

این سه ویژگی در تمام داده‌های آموزشی کاملاً ثابت هستند.

- **انحراف معیار بسیار نزدیک به صفر (عملاً ثابت):**

Sensor_1 , 5 , 10 , 16

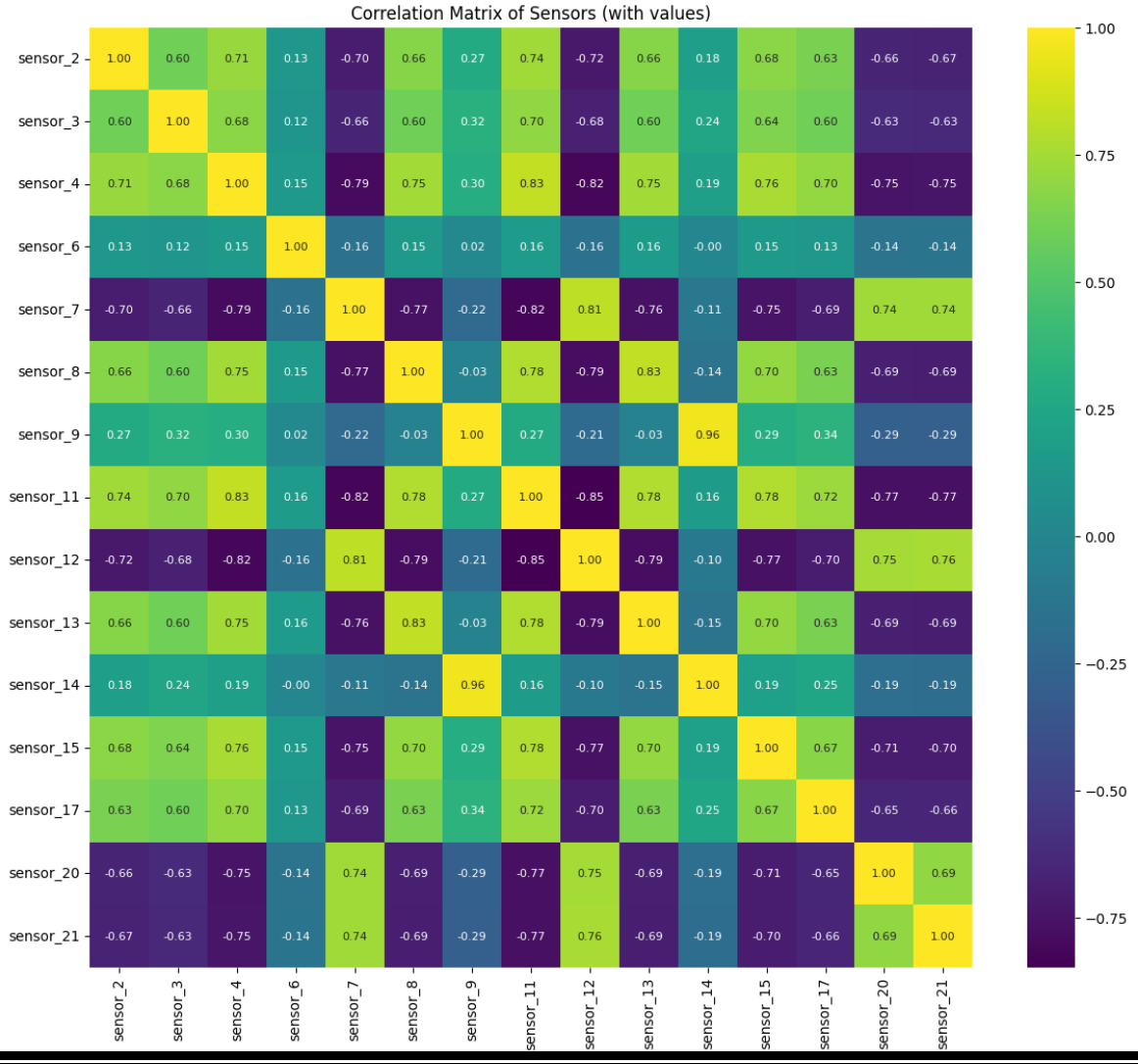
این ویژگی‌ها نیز تغییرات ناچیزی دارند و اطلاعات مفیدی به مدل اضافه نمی‌کنند.

بنابراین، طبق دستورالعمل تمرین، این ستون‌ها را از هر دو مجموعه داده آموزش و ارزیابی حذف می‌کنیم تا مدل ساده‌تر و کارآمدتر شود.

حذف ویژگی‌های تکراری یا بسیار مشابه (Duplicate or Highly Similar Features)

بهترین راه برای پیدا کردن ویژگی‌های بسیار مشابه، محاسبه ماتریس همبستگی است. این ماتریس نشان می‌دهد که هر دو ویژگی چقدر به هم مرتبط هستند. اگر دو حسگر همبستگی نزدیک به ۱ (رابطه مستقیم) یا منفی ۱ (رابطه معکوس) داشته باشند، یعنی اطلاعات تقریباً یکسانی را ارائه می‌دهند و می‌توان یکی از آن‌ها را حذف کرد.

برای اینکه بتوانیم ماتریس همبستگی را به صورت گرافیکی و خوانا ببینیم، از کتابخانه‌های matplotlib و seaborn استفاده می‌کنیم. یک هیت‌مپ (Heatmap) از ماتریس همبستگی تولید می‌کنیم که در آن، رنگ‌های روشن نشان‌دهنده همبستگی بالا هستند.



با بررسی دقیق ماتریس، یک مورد بسیار برجسته وجود دارد:

- همبستگی بین sensor_9 و sensor_14 برابر با 0.96 است.

این مقدار بسیار به ۱ نزدیک است و به این معناست که این دو حسگر اطلاعات تقریباً یکسانی را ثبت می‌کنند. وقتی یکی از آن‌ها افزایش می‌یابد، دیگری نیز تقریباً به همان نسبت افزایش می‌یابد. بنابراین، نگه داشتن هر دوی آن‌ها در مدل ضروری نیست و می‌توانیم یکی را حذف کنیم تا از پیچیدگی غیرضروری مدل جلوگیری کنیم.

همچنین جفت‌های دیگری با همبستگی بالا (بین ۰.۸ تا ۰.۹) وجود دارند، مانند sensor_11,12 با مقدار 0.85 اما به عنوان یک قاعده کلی، معمولاً با آستانه‌های سخت‌گیرانه‌تر مانند ۰.۹ یا ۰.۹۵ شروع می‌کنیم تا فقط واضح‌ترین موارد افزونگی را حذف کنیم.

ما یکی از دو سنسور sensor_9 یا sensor_14 را حذف می‌کنیم. انتخاب بین این دو تفاوت چندانی ایجاد نمی‌کند، پس به صورت قراردادی sensor_14 را حذف می‌کنیم.

تعداد ۱۸ ویژگی بعد از این مرحله باقیمانده است.

ارتباط ویژگی‌ها با متغیر هدف (Correlation with Target, Correlation Matrix)

در این مرحله، ما باید متغیر هدف، یعنی عمر مفید باقیمانده (RUL) را برای داده‌های آموزشی محاسبه کنیم و سپس همبستگی هر یک از ویژگی‌های باقی‌مانده را با این RUL بسنجیم. ویژگی‌هایی که ارتباط بسیار ضعیفی با RUL دارند، احتمالاً برای پیش‌بینی آن مفید نخواهند بود.

منطق محاسبه: برای هر موتور، RUL در هر چرخه برابر است با:

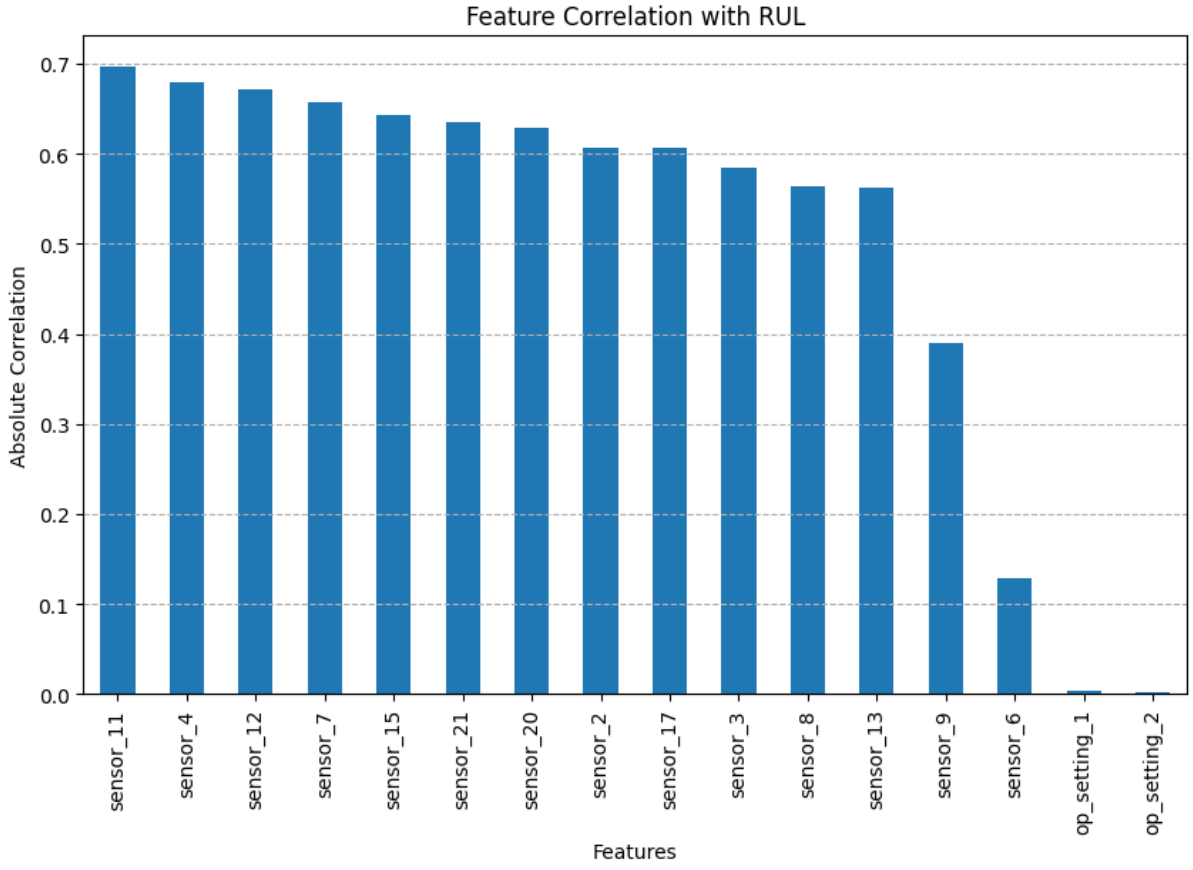
(آخرین چرخه عمر آن موتور) - (چرخه فعلی)

پس از اینکه ستون RUL را با موفقیت ایجاد کردیم، همبستگی تمام ویژگی‌های دیگر (به خصوص سنسورها) را با این ستون محاسبه خواهیم کرد تا ببینیم کدام‌ها بیشترین و کمترین تاثیر را دارند.

- **همبستگی نزدیک به ۱ یا ۱-**: نشان‌دهنده ارتباط قوی است (ویژگی برای پیش‌بینی بسیار مفید است).
- **همبستگی نزدیک به ۰**: نشان‌دهنده ارتباط ضعیف است (ویژگی احتمالاً برای پیش‌بینی مفید نیست).

کد همبستگی‌ها را محاسبه کرده، آن‌ها را بر اساس قدر مطلق (برای در نظر گرفتن روابط مثبت و منفی) مرتب می‌کند و به صورت یک لیست و یک نمودار میله‌ای نمایش می‌دهد تا تحلیل آن ساده باشد.

همبستگی هر ویژگی با RUL	
sensor_11	0.696228
sensor_4	0.678948
sensor_12	0.671983
sensor_7	0.657223
sensor_15	0.642667
sensor_21	0.635662
sensor_20	0.629428
sensor_2	0.606484
sensor_17	0.606154
sensor_3	0.584520
sensor_8	0.563968
sensor_13	0.562569
sensor_9	0.390102
sensor_6	0.128348
op_setting_1	0.003198
op_setting_2	0.001948



- **ویژگی‌های مهم**: حسگرهایی مانند sensor_11, sensor_4, sensor_12 و sensor_7 دارای همبستگی بالایی (بیشتر از ۰.۶) با RUL هستند. این‌ها ارزشمندترین ویژگی‌های ما برای ساخت یک مدل دقیق خواهند بود.

- **ویژگی‌های با اهمیت کم:** در سمت راست نمودار، سه ویژگی را می‌بینیم که همبستگی بسیار پایینی با RUL دارند:
  - sensor_6 همبستگی حدود ۰.۱۳
  - op_setting_1 همبستگی نزدیک به صفر
  - op_setting_2 همبستگی نزدیک به صفر

طبق دستورالعمل تمرین منطقی است که این سه ویژگی را از مجموعه داده خود حذف کنیم، زیرا اطلاعات مفیدی برای پیش‌بینی عمر باقی‌مانده ارائه نمی‌دهند.

تعداد ۱۳ ویژگی بعد از این مرحله باقیمانده است.

اهمیت ویژگی‌ها بر اساس مدل (Random Forest Feature Importance)

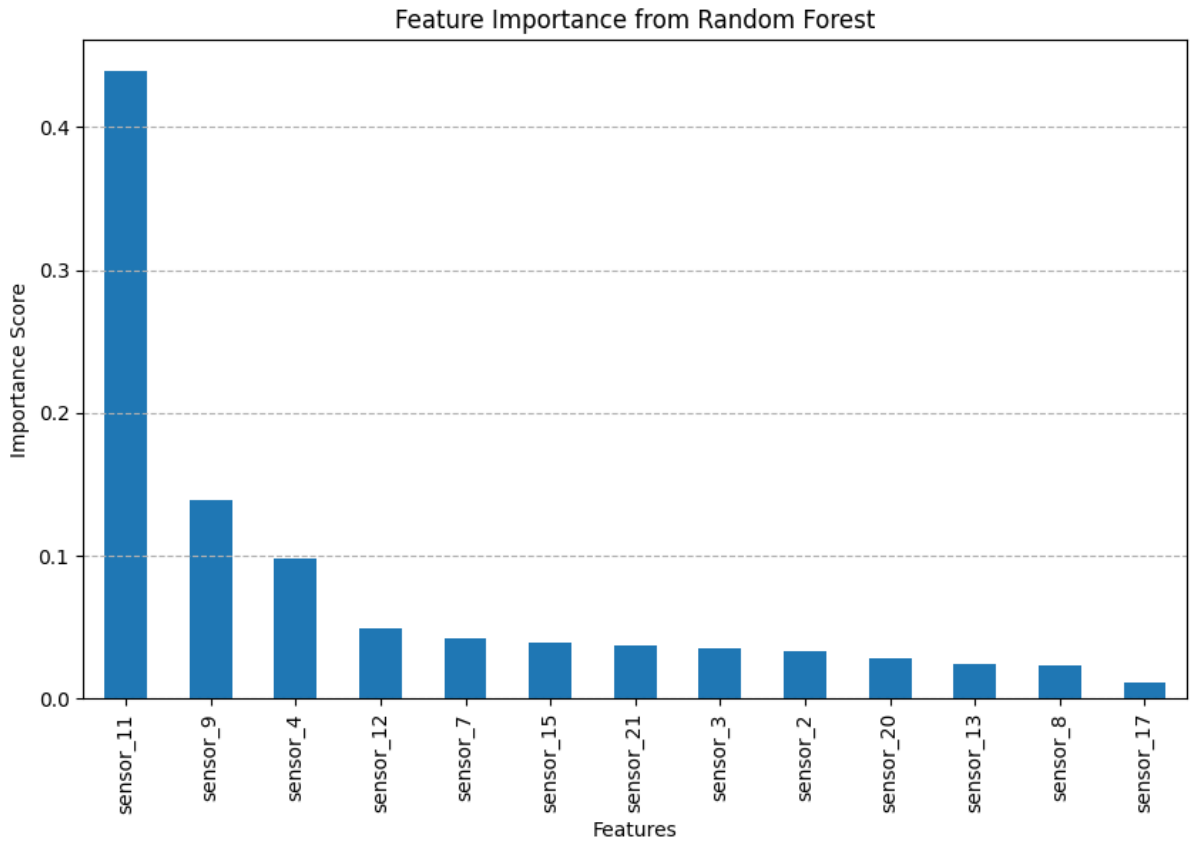
این روش، به جای استفاده از آمارهای ساده مانند همبستگی، از یک مدل یادگیری ماشین مانند Random Forest برای ارزیابی اهمیت هر ویژگی استفاده می‌کند. این روش معمولاً دیدگاه دقیق‌تری ارائه می‌دهد.

همبستگی فقط روابط خطی بین ویژگی و هدف را اندازه‌گیری می‌کند. اما مدل‌هایی مانند جنگل تصادفی (Random Forest) می‌توانند روابط پیچیده‌تر و غیرخطی را نیز تشخیص دهند. این مدل با ساختن تعداد زیادی درخت تصمیم و میانگین‌گیری از نتایج آن‌ها، به هر ویژگی یک امتیاز اهمیت اختصاص می‌دهد که نشان می‌دهد آن ویژگی چقدر در کاهش خطای کلی مدل مؤثر بوده است.

- ویژگی‌های نهایی (۱۳ سنسور) را به عنوان ورودی (X) و ستون RUL را به عنوان خروجی (y) تعریف می‌کنیم.
- یک مدل RandomForestRegressor را روی این داده‌ها آموزش می‌دهیم. (تعداد ۱۰۰ درخت تصمیم در جنگل و random state=42 برای اطمینان از تکرارپذیری نتایج)
- اهمیت ویژگی‌ها را از مدل آموزش‌دیده استخراج کرده و نمایش می‌دهیم.

اهمیت ویژگی‌ها بر اساس مدل	
sensor_11	0.439063
sensor_9	0.138502
sensor_4	0.098400
sensor_12	0.049073
sensor_7	0.042435
sensor_15	0.039369
sensor_21	0.037532

sensor_3	0.034732
sensor_2	0.033352
sensor_20	0.028298
sensor_13	0.024430
sensor_8	0.023334
sensor_17	0.011480



این نمودار را با نمودار قبلی (مبتنی بر همبستگی) مقایسه کنیم:

تایید ویژگی‌های کلیدی: هر دو روش، چه همبستگی خطی و چه مدل Random Forest ، به طور مشترک حسگرهایی مانند sensor_11، sensor_4، و sensor_12 را به عنوان مهم‌ترین ویژگی‌ها شناسایی کرده‌اند. این به ما اطمینان زیادی می‌دهد که این حسگرها واقعاً برای پیش‌بینی RUL حیاتی هستند.

کشف روابط غیرخطی: جالب‌ترین نکته در این نمودار، اهمیت بالای sensor_9 است. در تحلیل همبستگی، این سنسور اهمیت متوسط رو به پایینی داشت. اما در مدل Random Forest ، این سنسور دومین ویژگی مهم است. این اختلاف نشان می‌دهد که sensor_9 احتمالاً یک رابطه غیرخطی و پیچیده با RUL دارد. همبستگی ساده قادر به کشف این رابطه نبود، اما مدل قدرتمند Random Forest توانست اهمیت آن را تشخیص دهد. این دقیقاً ارزش انجام این مرحله را نشان می‌دهد.

این تحلیل به ما اطمینان داد که مجموعه ۱۳ ویژگی نهایی ما، یک مجموعه قوی و معتبر است. نیازی به حذف ویژگی دیگری نیست و می‌توانیم با همین ۱۳ ویژگی به مراحل بعد برویم.

فرآیندی که انجام شد انتخاب ویژگی (Feature Selection) نام داشت، یکی از مهم‌ترین مراحل در پروژه‌های یادگیری ماشین است. هدف ما داشتن بیشترین تعداد ویژگی نیست، بلکه داشتن بهترین و مفیدترین ویژگی‌هاست. با حذف ویژگی‌های بی‌فایده و تکراری:

- مدل سریع‌تر آموزش می‌بیند.
- احتمال بیش‌برازش (Overfitting) کاهش می‌یابد.
- مدل روی سیگنال‌های اطلاعاتی واقعی تمرکز می‌کند و می‌تواند به دقت بالاتری برسد.

ما با این کار، مجموعه داده را پاک‌سازی کرده و آن را برای ساخت یک مدل قوی‌تر و کارآمدتر آماده کرده‌ایم.

نرمال سازی (Normalization):

شبکه‌های عصبی به مقیاس داده‌ها حساس هستند. اگر یک ویژگی مقادیری بین ۰ تا ۱ و ویژگی دیگر مقادیری بین ۱۰۰۰ تا ۲۰۰۰ داشته باشد، مدل در یادگیری دچار مشکل می‌شود. نرمال‌سازی تمام ویژگی‌ها را در یک مقیاس یکسان قرار می‌دهد که باعث می‌شود:

- آموزش مدل سریع‌تر و پایدارتر شود.
- مدل به دقت بالاتری دست یابد.

تمرین دو روش را پیشنهاد داده است:

- **Min-Max Scaler** تمام داده‌ها را به بازه مشخصی، معمولاً [۰, ۱]، منتقل می‌کند.
- **Z-score (Standard Scaler)** داده‌ها را طوری تغییر می‌دهد که میانگین صفر و انحراف معیار یک داشته باشند.

ما از روش Min-Max استفاده خواهیم کرد که برای شبکه‌های عصبی بسیار رایج و موثر است.

Scaler را فقط روی داده‌های آموزشی برازش (fit) می‌کنیم و سپس از همین Scaler برازش‌شده برای transform (تبدیل) کردن هر دو مجموعه آموزشی و آزمون استفاده می‌کنیم. این کار نیز برای جلوگیری از نشت اطلاعات از مجموعه آزمون به فرآیند آموزش است.

ساخت پنجره‌های لغزان و برچسب‌گذاری (Sliding Window):

شبکه‌های عصبی مثل CNN و LSTM به ورودی‌هایی با اندازه ثابت و یکسان نیاز دارند. از آنجایی که تاریخچه عمر هر موتور طول متفاوتی دارد، ما نمی‌توانیم آن‌ها را مستقیماً به شبکه بدهیم. روش پنجره لغزان، سری‌های زمانی طولانی را به تعداد زیادی نمونه کوچک‌تر با طول یکسان تقسیم می‌کند. این کار به مدل اجازه می‌دهد الگوهای کوتاه‌مدت را تشخیص دهد و تعداد نمونه‌های آموزشی را نیز افزایش می‌دهد.

۱. اندازه پنجره (۳۰ چرخه) انتخاب می‌کنیم.

۲. این پنجره را روی داده‌های هر موتور حرکت می‌دهیم. اولین نمونه شامل چرخه‌های ۱ تا ۳۰، دومین نمونه شامل چرخه‌های ۲ تا ۳۱ و الی آخر خواهد بود.

۳. برای هر پنجره، عمر مفید باقیمانده (RUL) مربوط به آخرین چرخه آن پنجره را به عنوان برچسب (Label) در نظر می‌گیریم.

```
شکل داده‌های ورودی (X_train): (17631, 30, 13)
شکل برچسب‌ها (y_train): (17631,)
```

ابعاد داده‌ها نشان می‌دهد که فرآیند ساخت پنجره‌ها با موفقیت انجام شده. ما اکنون ۱۷,۶۳۱ نمونه آموزشی داریم که هر کدام یک توالی ۳۰ مرحله‌ای از ۱۳ ویژگی هستند.

همانطور که در تمرین اشاره شده، مقادیر بسیار بزرگ RUL می‌تواند آموزش مدل را دشوار کند. مدل ممکن است تلاش زیادی برای پیش‌بینی دقیق RUL در زمانی که موتور هنوز سالم است (مثلاً ۲۰۰ چرخه تا خرابی) انجام دهد، در حالی که پیش‌بینی دقیق‌تر در نزدیکی زمان خرابی اهمیت بیشتری دارد.

برای حل این مشکل، یک حد بالا (۱۳۰ چرخه) برای RUL تعیین می‌کنیم و تمام مقادیر بزرگ‌تر از آن را برابر با این حد قرار می‌دهیم.

تقسیم داده‌ها (Train/Test Split):

ما باید داده‌های test_df را نیز به صورت پنجره آماده کنیم. اما منطق آن کمی متفاوت است: برای هر موتور در مجموعه آزمون، ما فقط به آخرین پنجره موجود نیاز داریم، زیرا هدف پیش‌بینی RUL از آن نقطه به بعد است. سپس این پنجره را با مقدار واقعی RUL از فایل RUL_FD001.txt جفت می‌کنیم.

همچنین، داده‌های آموزشی را که به صورت پنجره هستند، با نسبت ۳۰/۷۰ به دو مجموعه آموزش (Train) و اعتبارسنجی (Validation) تقسیم می کنیم.

در این روش، پنجره‌هایی از یک موتور یکسان ممکن است هم در مجموعه آموزش و هم در مجموعه اعتبارسنجی قرار بگیرند. این یک روش استاندارد و رایج است، اما یک روش سخت‌گیرانه‌تر، استفاده از تقسیم گروهی (Group Split) است که در آن تمام پنجره‌های مربوط به یک موتور خاص، فقط در یکی از دو مجموعه آموزش یا اعتبارسنجی قرار می‌گیرند. با این حال، با توجه به اینکه ارزیابی نهایی ما روی مجموعه آزمون با موتورهای کاملاً جدید انجام می‌شود، روش به کار رفته برای نظارت بر فرآیند آموزش قابل قبول و معتبر است.

چرا نباید داده‌های آزمون و آموزش را ترکیب و سپس تقسیم کنیم؟

ترکیب کردن مجموعه داده آموزشی (train_df) با مجموعه داده آزمون نهایی (test_df) یکی از اشتباهات رایج و جدی در یادگیری ماشین است که به آن نشت داده (Data Leakage) می‌گویند.

- مجموعه آزمون (test_df) نقش امتحان نهایی را دارد: این داده‌ها باید تا آخرین لحظه دست‌نخورده باقی بمانند تا بتوانیم عملکرد واقعی مدل را روی داده‌هایی که هرگز ندیده است، بسنجیم.
- اگر این دو مجموعه را ترکیب کنیم، اطلاعاتی از داده‌های آزمون به فرآیند آموزش نشت می‌کند. در نتیجه، مدل در مرحله ارزیابی عملکردی غیرواقعی و بیش از حد خوش‌بینانه از خود نشان می‌دهد، زیرا در واقع بخشی از سوالات امتحان نهایی را قبلاً دیده است.

چرا تقسیم را به بعد از نرمال سازی و ساخت پنجره ها موکول کردیم؟

- **حفظ توالی داده‌ها:** ما با داده‌های سری زمانی سروکار داریم. اگر قبل از ساخت پنجره‌ها، داده‌ها را به صورت تصادفی تقسیم کنیم، توالی زمانی چرخه‌های موتورها به هم می‌ریزد. مثلاً ممکن است چرخه‌های ۱۰ تا ۲۰ یک موتور در مجموعه آموزش و چرخه‌های ۲۱ تا ۳۰ همان موتور در مجموعه اعتبارسنجی قرار بگیرد که منطقی نیست.

- **ایجاد تقسیم‌بندی واقع‌گرایانه:** روش بهتر این است که ابتدا تمام پنجره‌های ممکن را از کل داده‌های آموزشی (train_df) بسازیم. سپس، این پنجره‌ها را به دو دسته آموزش (training) و اعتبارسنجی (validation) تقسیم کنیم. این کار تضمین می‌کند که هر پنجره (که یک توالی کامل است) دست‌نخورده باقی می‌ماند.

اگر طبق ترتیب خطی تمرین عمل می‌کردیم:

۱ .چالش اصلی: از هم گسیختگی سری‌های زمانی(Temporal Coherence Leakage)

این بزرگترین و جدی‌ترین مشکل است. تابع train_test_split به صورت تصادفی سطرها را برای تقسیم انتخاب می‌کند. این برای داده‌های معمولی مشکلی ندارد، اما برای داده‌های سری زمانی یک فاجعه است. تصور کنید موتور شماره ۵ در مجموع ۲۰۰ چرخه عمر دارد. وقتی ما train_test_split را روی سطرهای این دیتافریم اجرا می‌کنیم، ممکن است چرخه‌های ۱ تا ۱۵۰ این موتور به صورت تصادفی در مجموعه آموزش (new_train_df) و چرخه‌های ۱۵۱ تا ۲۰۰ آن در مجموعه اعتبارسنجی (val_df) قرار بگیرد.

- **نتیجه:**
  - **داده‌های آموزشی ناقص:** وقتی می‌خواهیم روی new_train_df پنجره بسازیم، تاریخچه موتور شماره ۵ در چرخه ۱۵۰ به پایان می‌رسد. الگوریتم ما به اشتباه تصور می‌کند که عمر این موتور ۱۵۰ بوده و RUL را بر این اساس محاسبه می‌کند که کاملاً غلط است.
  - **داده‌های اعتبارسنجی بی‌معنی:** مجموعه val_df شامل چرخه‌های پایانی عمر موتور شماره ۵ است بدون آنکه تاریخچه اولیه آن را داشته باشد. این داده‌ها به تنهایی قابل استفاده برای ساخت پنجره نیستند.

این کار باعث نشت اطلاعات و از هم گسیختگی توالی زمانی می‌شود و کل فرآیند آموزش و اعتبارسنجی را بی‌اعتبار می‌کند.

۲ .چالش دوم: ناکارآمدی و پیچیدگی کد

حتی اگر مشکل اول را نادیده بگیریم، مجبور بودیم که تمام منطق پیچیده ساخت پنجره‌های لغزان را دو بار (یک بار برای مجموعه آموزش و یک بار برای مجموعه اعتبارسنجی) اجرا کنیم که بهینه نیست.

روشی که در پیش گرفتیم، این مشکلات را حل می‌کند:

۱. ابتدا روی کل داده‌های آموزشی (train_df) که تاریخچه کامل هر ۱۰۰ موتور را دارد، تمام پنجره‌های صحیح و کامل را استخراج کردیم. در این مرحله، هر پنجره یک نمونه داده مستقل و با برچسب RUL صحیح است.

۲. سپس، این مجموعه بزرگ از پنجره‌ها را (که دیگر توالی زمانی در بین خودشان معنایی ندارد) به صورت تصادفی به دو بخش آموزش و اعتبارسنجی تقسیم کردیم.

این روش تضمین می‌کند که هر پنجره‌ای که مدل می‌بیند، یک توالی زمانی معتبر و دست‌نخورده از تاریخچه یک موتور است و هیچ داده‌ای به اشتباه برچسب‌گذاری یا ناقص نمی‌شود. این رویکرد، استاندارد طلایی برای کار با داده‌های سری زمانی در چنین مسائلی است.

تفاوت کلیدی بین اعتبارسنجی و آزمون نهایی

ما در این پروژه با دو نوع تست سروکار داشتیم:

۱. **مجموعه اعتبارسنجی(Validation Set)** : این همان مجموعه ۳۰ درصدی بود که ما از داده‌های آموزشی خودمان جدا کردیم. هدف این مجموعه فقط نظارت بر فرآیند آموزش بود. یعنی با آن می‌فهمیدیم که آیا مدل در حال یادگیری است یا دچار بیش‌برازش شده.

۲. **مجموعه آزمون نهایی(Final Test Set)** : این همان مجموعه test_FD001.txt است که از ابتدا توسط ناسا جدا شده بود و شامل ۱۰۰ موتور کاملاً جدید و دیده نشده بود. تمام نتایج نهایی ما (جدول معیارها، نمودار پیش‌بینی در برابر واقعیت، و مقایسه نهایی مدل‌ها) بر اساس عملکرد مدل روی این مجموعه به دست آمده است.

شکل نهایی داده های آموزش: (13,30,12341) (12341,)
شکل نهایی داده های اعتبارسنجی: (13,30,5290) (5290,)
شکل نهایی داده های آزمون: (13,30,100) (100,)

مرحله ۲: پیاده‌سازی شبکهٔ عصبی پیچشی (CNN)

یک شبکه عصبی پیچشی یک‌بعدی (1D CNN) می سازیم. این نوع شبکه برای پیدا کردن الگوهای محلی در داده‌های سری زمانی (مانند یک الگوی خاص در ۱۰ چرخه متوالی) بسیار کارآمد است.

دقیقاً معماری و پارامترهای مشخص شده در فایل تمرین را پیاده‌سازی خواهیم کرد:

Conv1D با ۶۴ فیلتر، کرنل ۳، فعال‌سازیReLU ، پدینگ
MaxPooling1D با اندازه پنجره ۲
Conv1D با ۱۲۸ فیلتر، کرنل ۳، فعال‌سازیReLU ، پدینگ
GlobalAveragePooling1D
Dense با ۶۴ نورون، فعال‌سازیReLU
Dropout با نرخ ۰/۲
Dense با یک نورون برای خروجیRUL

آموزش:

تابع خطا: میانگین مربعات خطا(MSE)

بهینه‌ساز: Adam با نرخ یادگیری ۰/۰۰۱

اندازه دسته(batch size) : ۶۴

تعداد دوره(epochs) : ۵۰

معیار ارزیابی: خطای میانگین قدر مطلق(MAE)

با استفاده از کتابخانه Keras (بخشی ازTensorFlow) مدل را تعریف، کامپایل و سپس آموزش می‌دهیم.

مرحله ۳: پیاده‌سازی شبکهٔ LSTM

در این مرحله یک شبکه بازگشتی از نوع LSTM (حافظه طولانی کوتاه‌مدت) می‌سازیم. این شبکه‌ها به طور خاص برای یادگیری وابستگی‌ها در داده‌های ترتیبی و سری زمانی طراحی شده‌اند. دوباره دقیقاً از معماری مشخص شده در تمرین استفاده خواهیم کرد.

LSTM با ۱۰۰ واحد، بازگرداندن توالی
Dropout با نرخ ۰/۲
LSTM با ۵۰ واحد، فقط خروجی آخرین گام
Dense با ۶۴ نورون، فعال‌سازی ReLU
Dropout با نرخ ۰/۲
Dense با یک نورون برای مقدارRUL

کامپایل و آموزش:

تابع خطا: MSE
بهینه‌ساز: Adam با نرخ یادگیری۰/۰۰۱
اندازهٔ دسته و تعداد دوره مشابه مدل CNN برای فراهم شدن امکان مقایسه

مرحله ۴ : تنظیم ابرپارامترها (Hyperparameter Tuning)

این مرحله را در دو سطح اولیه و تکمیلی انجام خواهیم داد:

تنظیم خودکار ابرپارامترها با KerasTuner

روند کار:

۱. یک هایپرمدل (Hypermodel) می‌سازیم. این یک تابع است که مدل ما را تعریف می‌کند، اما به جای مقادیر ثابت برای پارامترها، از متغیرهایی استفاده می‌کند که KerasTuner می‌تواند آن‌ها را تغییر دهد (مثلاً hp.Int برای تعداد فیلترها یا hp.Choice برای نرخ یادگیری.)

۲. یک تیونر (Tuner) انتخاب می‌کنیم. ما از RandomSearch استفاده می‌کنیم که به صورت تصادفی ترکیبات مختلفی از پارامترها را امتحان می‌کند.

۳. جستجو را با یک تعداد محدود آزمایش (trials) اجرا می‌کنیم تا بهترین ترکیب را پیدا کند.

ابتدا باید KerasTuner را نصب کنیم:

```
pip install -q -U keras-tuner!
```

در مرحله اول:

- ما به KerasTuner اجازه دادیم تا تعداد فیلترها، نرخ Dropout و نرخ یادگیری را از بین گزینه‌های مشخصی انتخاب کند.
- max_trials=5 یعنی تیونر فقط ۵ ترکیب مختلف را امتحان می‌کند تا فرآیند خیلی طولانی نشود.
- هدف (objective) کمینه کردن val_mean_absolute_error است، یعنی پیدا کردن مدلی که کمترین خطا را روی داده‌های اعتبارسنجی دارد.

نتیجه این بخش:

```
val_mean_absolute_error: 17.45041275024414
Best val_mean_absolute_error So Far: 16.65687370300293
Total elapsed time: 00h 02m 22s
جستجو به پایان رسید.
بهترین تعداد فیلترها: ۹۶
بهترین نرخDropout: 0.30000000000000004
بهترین نرخ یادگیری: ۰,۰۰۱
```

این خروجی به ما می‌گوید که در آن ۵ آزمایش اولیه، بهترین عملکرد (کمترین خطای MAE روی داده‌های اعتبارسنجی) با تنظیمات زیر به دست آمده است:

- تعداد فیلترها: ۹۶
- نرخDropout : ۰.۳
- نرخ یادگیری: ۰.۰۰۱

در مرحله تکمیلی انتخاب نوع بهینه ساز را نیز اضافه میکنیم و تعداد ترکیب ها را بیشتر می کنیم:

val_mean_absolute_error: 12.47687816619873

Best val_mean_absolute_error So Far: 12.062032699584961

Total elapsed time: 00h 03m 22s

جستجوی کامل‌تر به پایان رسید.

بهترین نوع بهینه‌ساز:rmsprop

بهترین تعداد فیلترها: ۹۶

بهترین نرخDropout: 0.4

بهترین نرخ یادگیری: ۰,۰۱

- بهبود قابل توجه: توانستیم با تنظیم پارامترها، خطای اعتبارسنجی (val_mean_absolute_error) را از حدود ۱۶.۶ در جستجوی اولیه به ۱۲.۰۶ کاهش دهیم. این یک بهبود چشمگیر است و ارزش این مرحله را به خوبی نشان می‌دهد.
- بهینه‌ساز برتر: KerasTuner تشخیص داد که برای این معماری و داده، بهینه‌ساز RMSprop عملکرد بهتری نسبت به Adam داشته است.
- نرخ یادگیری بهینه: نرخ یادگیری بالاتر (0.01) به همراه RMSprop بهترین نتیجه را داده است که نشان می‌دهد مدل توانایی یادگیری سریع‌تر را داشته است.
- ساختار مدل: مدل همچنان به تعداد فیلترهای نسبتاً بالا (۹۶) و یک نرخ Dropout متوسط (۰.۴) نیاز داشته است.

چرا اندازه پنجره (Window Size) را اضافه نکردیم؟

این پارامتر را به دلیل فنی و عملی بسیار مهمی اضافه نمی‌کنیم:

- اندازه پنجره یک پارامتر آماده‌سازی داده است، نه یک ابرپارامتر مدل. تغییر اندازه پنجره (مثلاً از ۳۰ به ۴۰) به این معنی است که باید کل ساختار داده‌های ورودی (X_train_final, X_val, X_test) از اول ساخته شود.
- فرآیند بسیار زمان‌بر و غیرعملی خواهد بود. اگر می‌خواستیم اندازه پنجره را با KerasTuner تنظیم کنیم، این ابزار باید برای هر بار آزمایش یک اندازه پنجره جدید، کل مراحل زیر را از نو تکرار می‌کرد:

۱. به داده‌های خام اولیه برگردد.

۲. کل فرآیند ساخت پنجره‌های لغزان را با اندازه جدید اجرا کند.

۳. یک مجموعه داده کاملاً جدید بسازد.

٤. یک مدل را روی این داده جدید آموزش دهد.

این کار فرآیند تیونینگ را فوق‌العاده کند و سنگین می‌کند. به همین دلیل، اندازه پنجره معمولاً به عنوان یک انتخاب ساختاری در مرحله مهندسی ویژگی در نظر گرفته می‌شود و به صورت دستی و با فاصله زمانی زیاد آزمایش می‌شود، نه در یک حلقه تیونینگ خودکار.

۲ .چرا اندازه دسته (Batch Size) را اضافه نکردیم؟

تنظیم این پارامتر ممکن است، اما به دو دلیل آن را در کد خود قرار ندادیم:

- پیچیدگی بیشتر کد: batch_size یک پارامتر معماری مدل نیست که در تابع build_hypermodel تعریف شود، بلکه پارامتری است که به متد fit داده می‌شود. تنظیم آن با KerasTuner به روش‌های پیشرفته‌تری (مانند نوشتن یک حلقه آموزش سفارشی) نیاز دارد.

❖ **کدام پارامترها بیشترین تأثیر را بر عملکرد مدل داشتند؟**

بر اساس دو آزمایشی که با KerasTuner انجام دادیم، دو پارامتر زیر به وضوح بیشترین و چشمگیرترین تأثیر را بر عملکرد مدل داشتند:

۱. **نوع بهینه‌ساز(Optimizer Type)** : این تاثیرگذارترین پارامتر بود. صرفاً با افزودن RMSprop به فضای جستجو و انتخاب آن توسط تیونر، کمترین خطای اعتبارسنجی (val_MAE) از حدود ۱۶.۶ به ۱۲.۰۶ کاهش یافت. این نشان می‌دهد که معماری مدل ما با بهینه‌ساز RMSprop سازگاری بسیار بهتری نسبت به Adam داشت.

۲. **نرخ یادگیری(Learning Rate)** : این پارامتر ارتباط نزدیکی با نوع بهینه‌ساز داشت. در حالی که در جستجوی اول با Adam، نرخ یادگیری بهینه 0.001 بود، در جستجوی دوم، RMSprop توانست با نرخ یادگیری بالاتر یعنی 0.01 به عملکرد بسیار بهتری دست یابد. این نشان می‌دهد که انتخاب صحیح بهینه‌ساز می‌تواند به مدل اجازه دهد تا با گام‌های بزرگ‌تر و با اطمینان بیشتری یاد بگیرد.

پارامترهای دیگر مانند تعداد فیلترها و نرخ Dropout نیز تاثیرگذار بودند، اما تغییرات آنها چنین بهبود چشمگیری را ایجاد نکرد.

❖ جدول اجرای مدل با تنظیمات مختلف و نتایج

در اینجا خلاصه‌ای از نتایج به دست آمده در فرآیند تنظیم ابرپارامترها ارائه می‌شود:

آزمایش	پارامترهای تنظیم شده	بهترین ترکیب پیدا شده	کمترین خطای اعتبارسنجی(MAE)
جستجوی اولیه	نرخ یادگیری، تعداد فیلترها، نرخ Dropout	lr=0.001, filters=96, dropout=0.3	۱۶.۶۶~
جستجوی کامل‌تر	نوع بهینه‌ساز، نرخ یادگیری، تعداد فیلترها، نرخDropout	optimizer=rmsprop, lr=0.01, filters=96, dropout=0.4	۱۲.۰۶~

این جدول به وضوح نشان می‌دهد که جستجوی دوم، که در آن بهینه‌ساز نیز به عنوان یک ابرپارامتر در نظر گرفته شد، منجر به کشف مدلی با عملکرد به مراتب بهتر گردید.

❖ بحث درباره تعادل بین زمان آموزش، دقت و پیچیدگی مدل

در پروژه‌های یادگیری عمیق، همیشه یک بده‌بستان (Trade-off) بین این سه عامل وجود دارد:

- **پیچیدگی و دقت:** یک مدل پیچیده‌تر (مثلاً با لایه‌ها یا فیلترهای بیشتر) پتانسیل یادگیری الگوهای پیچیده‌تری را دارد و می‌تواند به دقت بالاتری دست یابد. در آزمایش ما نیز، مدل با ۹۶ فیلتر بهتر از مدل‌های ساده‌تر عمل کرد. اما پیچیدگی بیش از حد خطر بیش‌برازش (Overfitting) را افزایش می‌دهد، یعنی مدل به جای یادگیری، داده‌های آموزشی را حفظ می‌کند.
- **پیچیدگی و زمان آموزش:** رابطه این دو مستقیم است. هر چه مدل پیچیده‌تر باشد، پارامترهای بیشتری برای یادگیری دارد و در نتیجه هر دوره (epoch) از آموزش زمان بیشتری می‌برد.

- **دقت و زمان آموزش (در فرآیند Tuning):** این مهم‌ترین تعادلی بود که در این مرحله با آن روبرو شدیم. ما برای پیدا کردن مدلی با دقت بالاتر (MAE کمتر)، زمان محاسباتی بیشتری را صرف کردیم. آموزش یک مدل ساده چند دقیقه طول کشید، اما اجرای فرآیند KerasTuner برای ۸ آزمایش مختلف، زمان بیشتری برد. این نشان می‌دهد که اختصاص زمان بیشتر برای جستجوی سیستماتیک ابرپارامترها، می‌تواند منجر به افزایش قابل توجهی در دقت نهایی شود.

هدف، پیدا کردن یک نقطه بهینه است: مدلی که به اندازه کافی پیچیده باشد تا به دقت خوبی برسد، اما نه آنقدر پیچیده که آموزش آن به طور غیرعملی طولانی شود یا دچار بیش‌برازش گردد. فرآیند تنظیم ابرپارامترها دقیقاً برای یافتن همین نقطه بهینه انجام می‌شود.

مرحله ۵: پیاده‌سازی مدل ترکیبی CNN+LSTM

این مدل از قدرت هر دو معماری CNN و LSTM به صورت ترکیبی استفاده می‌کند:

- لایه CNN ابتدا الگوهای محلی و ویژگی‌های مهم را از توالی زمانی استخراج می‌کند.
- لایه LSTM خروجی لایه CNN را دریافت کرده و وابستگی‌های زمانی و الگوهای بلندمدت را در این ویژگی‌های استخراج‌شده یاد می‌گیرد.

ما دقیقاً معماری خواسته شده در تمرین را پیاده‌سازی می‌کنیم:

Conv1D با ۶۴ فیلتر، کرنل ۳، فعال‌سازیReLU ، پدینگ مساوی
MaxPooling1D با اندازهٔ ۲
LSTM با ۱۰۰ واحد، بدون بازگرداندن توالی
Dropout با نرخ ۰/۳
Dense با ۶۴ نورون، فعال‌سازی ReLU
Dense با یک نورون برای خروجی RUL

مرحله ۵.۲: پیاده‌سازی مدل ترکیبی LSTM + CNN

در این بخش، ما ترتیب لایه‌های مدل ترکیبی را معکوس می‌کنیم. یعنی ابتدا داده‌ها وارد لایه LSTM شده و سپس خروجی آن به لایه CNN داده می‌شود.

منطق معماری جدید

۱. لایه LSTM ابتدا کل توالی ورودی را پردازش می‌کند تا وابستگی‌های زمانی و حافظه بلندمدت را در آن لحاظ کند. برای اینکه خروجی این لایه قابل استفاده برای لایه CNN بعدی باشد، باید حتماً یک توالی کامل را برگرداند (یعنی return_sequences=True).

۲. لایه CNN سپس این توالی غنی‌شده توسط LSTM را دریافت کرده و الگوهای محلی مهم را از آن استخراج می‌کند.

مرحله ۶: پیاده‌سازی معماری LSTM+ATTENTION

یک مدل LSTM استاندارد، در حین پردازش یک توالی، ممکن است به تمام بخش‌های آن به یک اندازه توجه کند. اما در بسیاری از مسائل، برخی از گام‌های زمانی اهمیت بیشتری در پیش‌بینی نهایی دارند. برای مثال، ممکن است داده‌های مربوط به ۱۰ چرخه آخر عمر موتور، بسیار مهم‌تر از داده‌های ۱۰۰ چرخه قبل‌تر باشند.

سازوکار توجه (Attention Mechanism) به مدل اجازه می‌دهد تا یاد بگیرد که به کدام بخش از توالی ورودی توجه بیشتری کند. این سازوکار وزن‌هایی را به خروجی‌های هر گام زمانی LSTM اختصاص می‌دهد و به مدل کمک می‌کند روی اطلاعات کلیدی تمرکز کرده و اطلاعات کم‌اهمیت را نادیده بگیرد. این کار طبق توضیحات تمرین، باعث موارد زیر می‌شود:

- افزایش دقت پیش‌بینی
- تفسیرپذیری بیشتر مدل
- بهبود یادگیری وابستگی‌های بلندمدت

پیاده‌سازی لایه Attention در Keras به سادگی لایه‌های دیگر نیست و معمولاً به یکی از دو روش زیر انجام می‌شود:

۱. استفاده از لایه Attention یا AdditiveAttention که در خود Keras موجود است.

۲. نوشتن یک لایه سفارشی(Custom Layer)

ما از روش اول که ساده‌تر و استانداردتر است، استفاده می‌کنیم.

بخش ب: ارزیابی عملکرد مدل‌ها

در این بخش، ما عملکرد پنج مدلی که ساخته‌ایم را با استفاده از نمودارها و معیارهای عددی تحلیل و مقایسه خواهیم کرد:

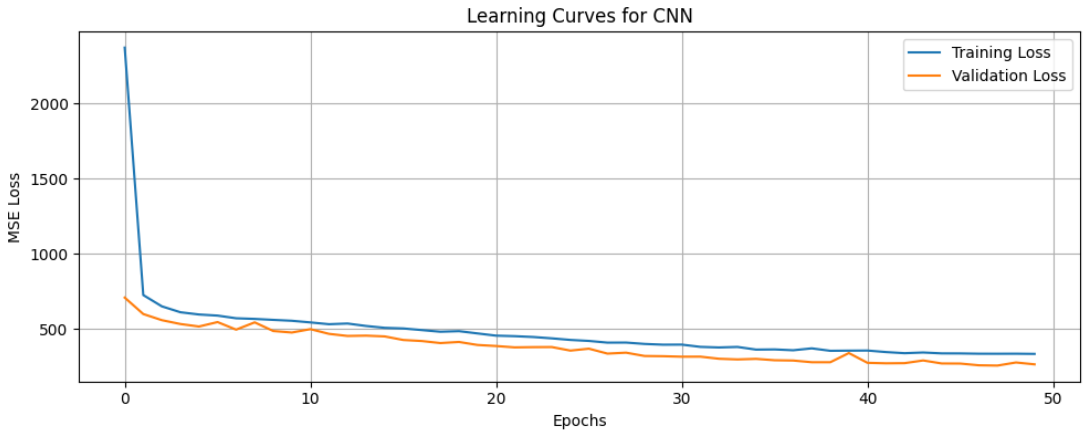
- ✓ CNN
- ✓ LSTM
- ✓ CNN + LSTM
- ✓ LSTM + CNN
- ✓ LSTM + Attention

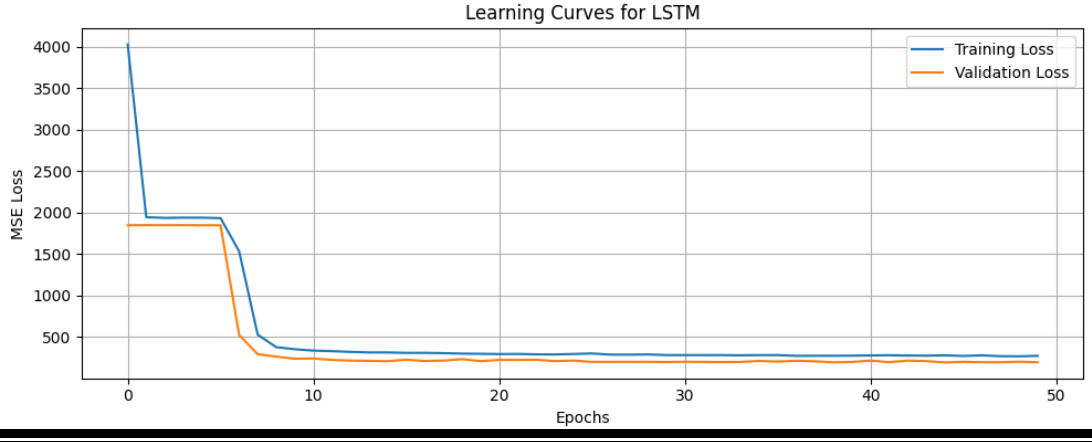
۱- نمودارها و مقایسهٔ بصری - منحنی‌های یادگیری

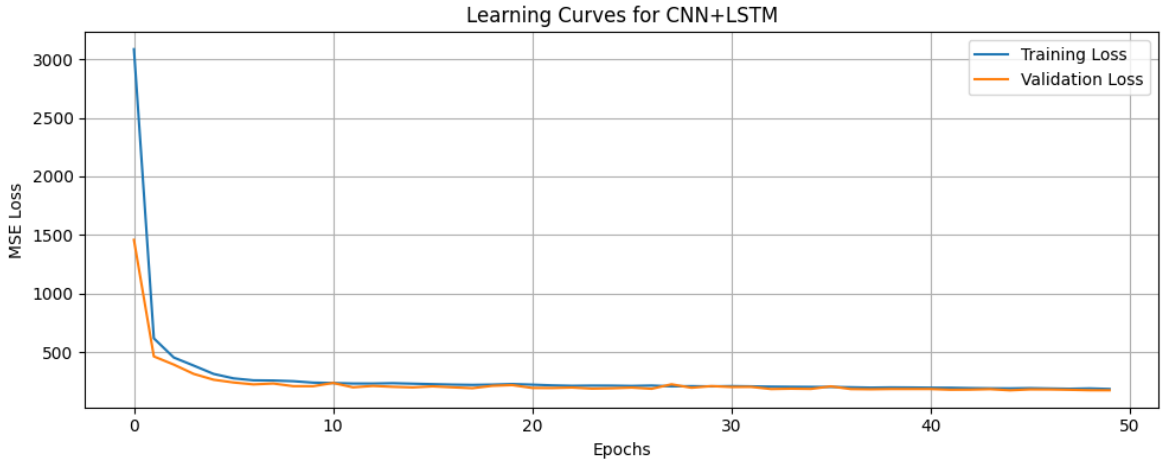
اولین قدم، رسم منحنی‌های خطای آموزش و اعتبارسنجی (loss curves) برای هر مدل است. این نمودارها به ما کمک می‌کنند تا بفهمیم هر مدل چگونه یاد گرفته است و آیا نشانه‌هایی از بیش‌برازش (Overfitting) یا کم‌برازش (Underfitting) در آن وجود دارد.

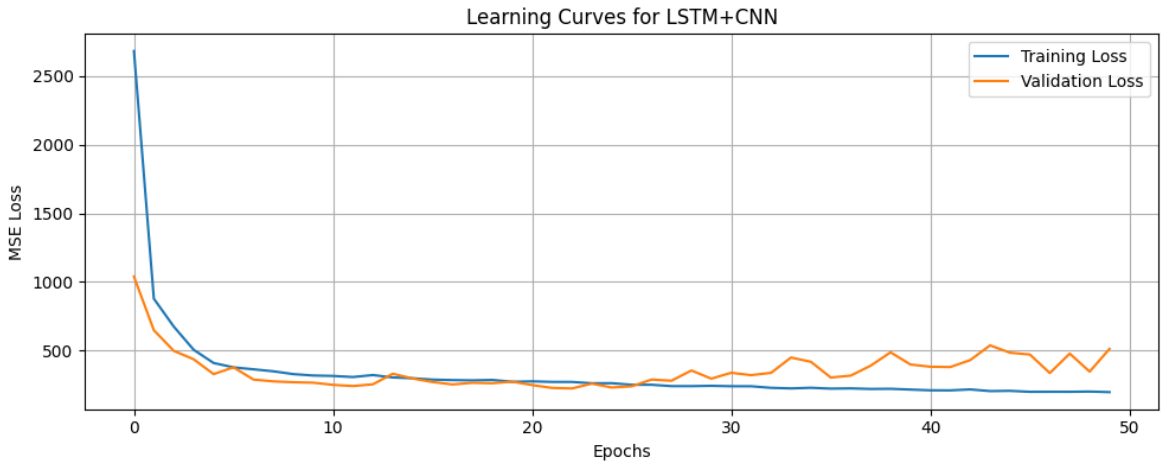
Overfitting: زمانی رخ می‌دهد که خطای آموزش بسیار کمتر از خطای اعتبارسنجی باشد (دو منحنی از هم فاصله زیادی می‌گیرند). این یعنی مدل داده‌های آموزشی را حفظ کرده اما قدرت تعمیم به داده‌های جدید را ندارد.

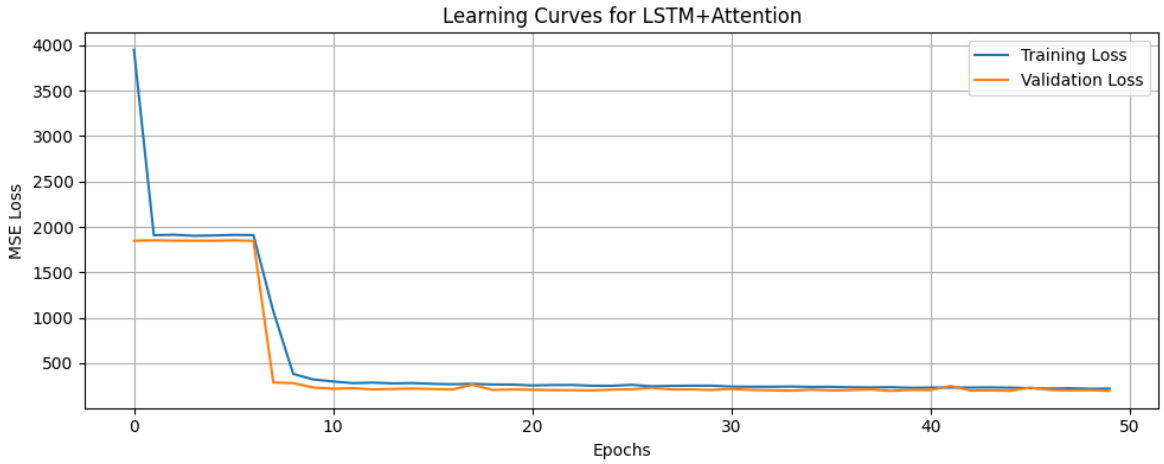
Underfitting: زمانی رخ می‌دهد که هر دو خطای آموزش و اعتبارسنجی بالا باقی بمانند. این یعنی مدل به اندازه کافی پیچیده نبوده تا الگوهای موجود در داده‌ها را یاد بگیرد.











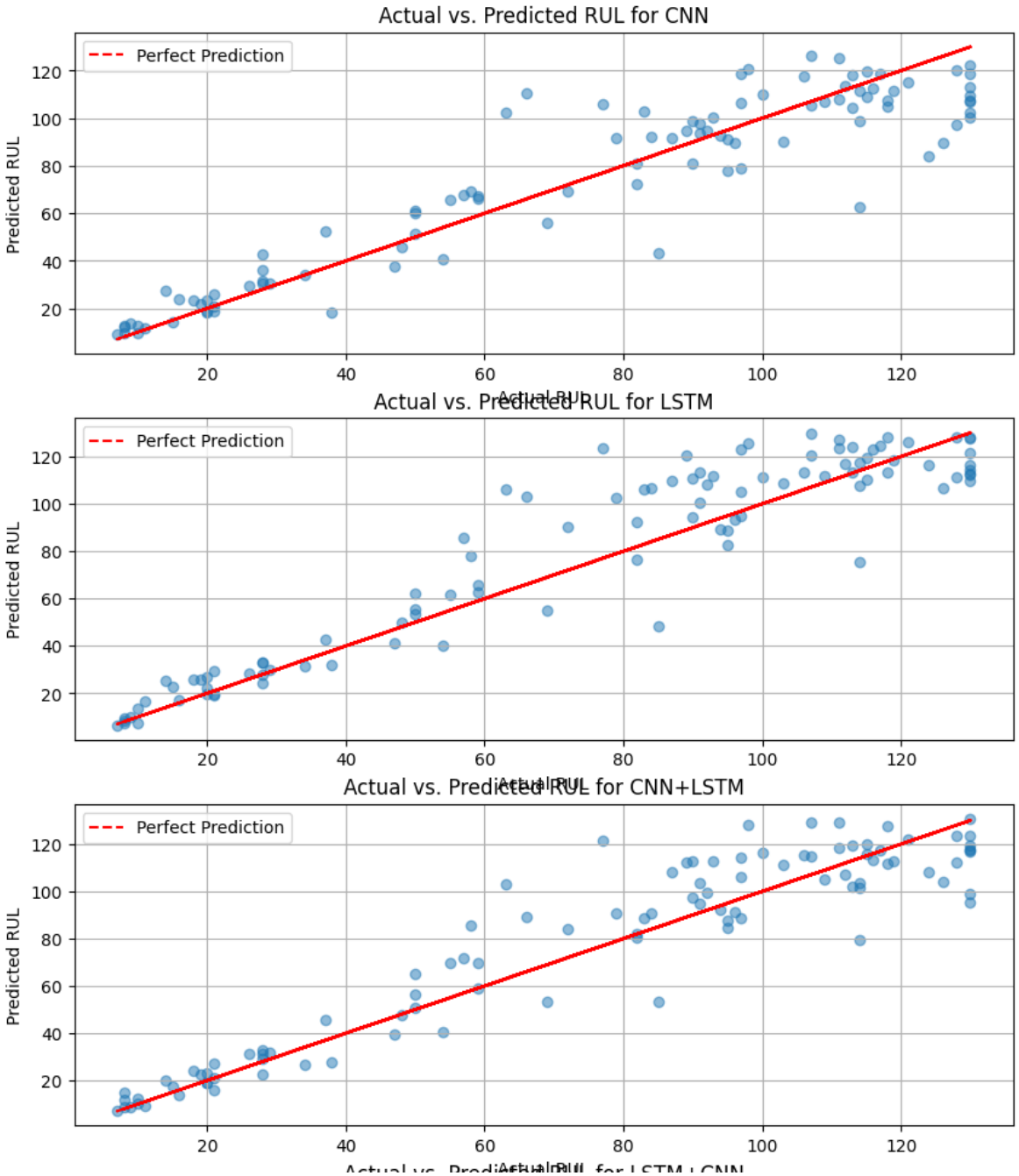
تحلیل منحنی‌های یادگیری

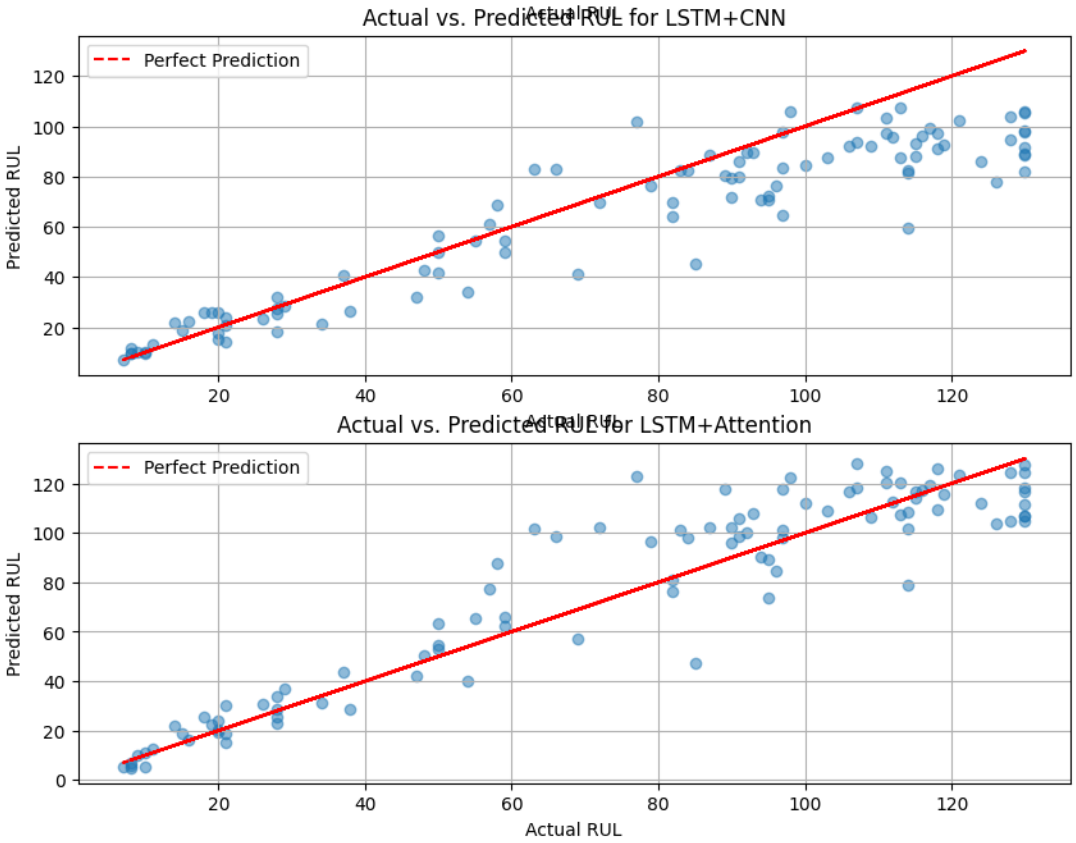
- همگرایی سریع: این ویژگی کاملاً مشهود است. تقریباً تمام مدل‌ها، به ویژه مدل‌های مبتنی بر LSTM، در چند دوره (epoch) ابتدایی با یک کاهش چشمگیر در خطا مواجه شده و به سرعت به یک خطای پایین و پایدار همگرا می‌شوند. این نشان می‌دهد که معماری‌ها به خوبی برای یادگیری الگوهای اصلی در داده‌ها مناسب هستند.
- عدم وجود بیش‌برازش (Overfitting) شدید: در اکثر نمودارها، منحنی خطای آموزش (آبی) و خطای اعتبارسنجی (نارنجی) روند بسیار نزدیکی را دنبال می‌کنند. این یک نشانه عالی است و به

این معناست که مدل‌ها توانسته‌اند یادگیری خود را به داده‌های جدید (مجموعه اعتبارسنجی) به خوبی تعمیم دهند. مدل LSTM+CNN کمی نوسان و فاصله در خطای اعتبارسنجی خود نشان می‌دهد که می‌تواند نشانه پایداری کمتر آن نسبت به سایر معماری‌ها باشد.

- تمام مدل‌ها به خوبی آموزش دیده‌اند. مدل‌های ترکیبی و مبتنی بر LSTM به دلیل درک بهتر وابستگی‌های زمانی، به نظر می‌رسد به سطح خطای پایین‌تری نسبت به مدل خالص CNN دست یافته‌اند.

مقدار واقعی در برابر مقدار پیش‌بینی‌شده RUL





عملکرد کلی: تمام مدل‌ها توانسته‌اند روند کلی داده‌ها را به خوبی یاد بگیرند. نقاط پیش‌بینی به صورت معناداری حول خط قطری (پیش‌بینی ایده‌آل) جمع شده‌اند که نشان‌دهنده کارایی کلی مدل‌هاست.

مقایسه مدل‌ها:

مدلCNN : این مدل بیشترین پراکندگی را در پیش‌بینی‌های خود نشان می‌دهد.

مدل‌های ترکیبی وLSTM : این مدل‌ها به وضوح پیش‌بینی‌های دقیق‌تری دارند و نقاط آن‌ها به خط ایده‌آل نزدیک‌تر است.

CNN+LSTM به عنوان مدل برتر: از نظر بصری به نظر می‌رسد که مدل CNN+LSTM بهترین عملکرد را دارد. نقاط آبی در نمودار این مدل، به طور چشمگیری متراکم و نزدیک به خط قرمز هستند که با نتایج جدول معیارها (که در آن CNN+LSTM کمترین خطا را داشت) کاملاً مطابقت دارد. مدل‌های LSTM+Attention و LSTM نیز عملکرد بسیار قوی و نزدیکی از خود نشان می‌دهند.

نتیجه‌گیری از نمودارها: مدل‌هایی که قادر به درک وابستگی‌های زمانی هستند (LSTM و ترکیبات آن) به طور واضحی بهتر از مدل CNN عمل کرده‌اند. در این اجرا، معماری ترکیبی CNN+LSTM با ترکیب قابلیت استخراج ویژگی CNN و حافظه زمانی LSTM، بهترین دقت بصری را به نمایش گذاشته است.

۲- معیارهای ارزیابی مدل

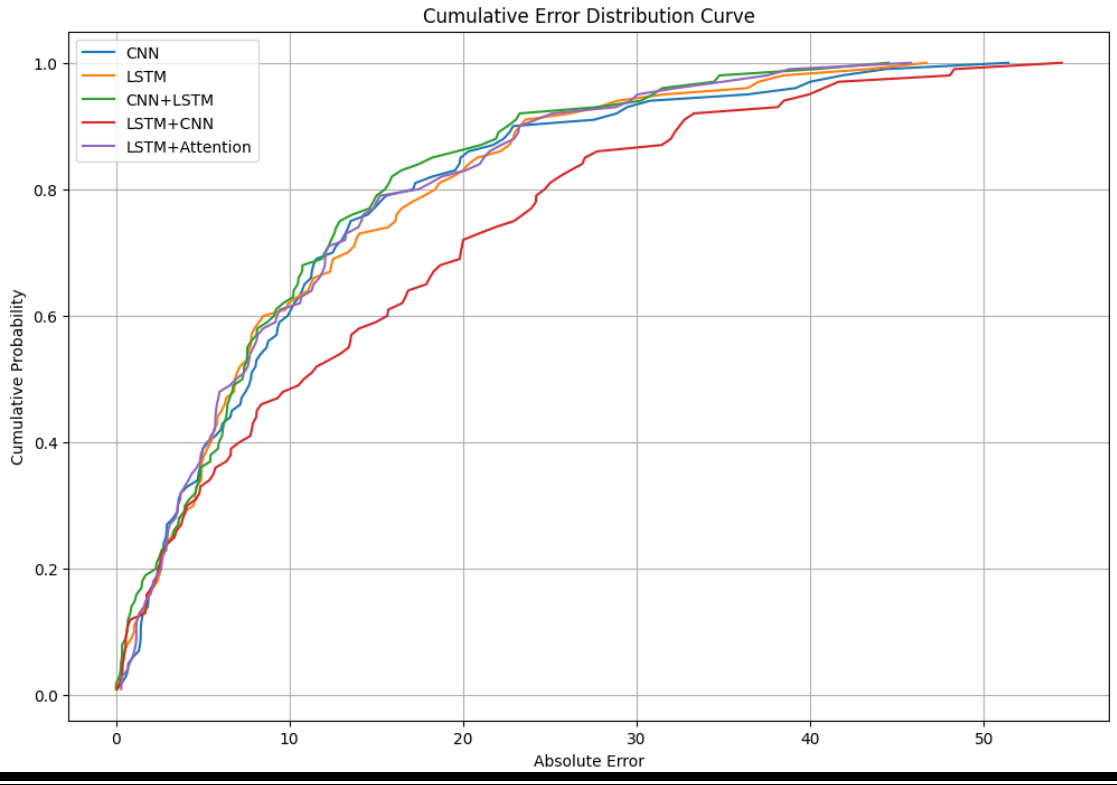
اگرچه بازرسی بصری مفید است، اما ما برای یک مقایسه دقیق و عینی به معیارهای عددی نیاز داریم. طبق درخواست تمرین، اکنون معیارهای زیر را برای هر مدل روی داده‌های آزمون محاسبه خواهیم کرد:

- RMSE (ریشه میانگین مربعات خطا): شبیه به MSE است اما واحد آن با متغیر هدف (RUL) یکسان است. این معیار خطاهای بزرگ را بیشتر جریمه می‌کند.
- MAE (میانگین قدر مطلق خطا): میانگین اختلاف مطلق بین مقادیر پیش‌بینی‌شده و واقعی است. تفسیر آن آسان است.
- R^2 (ضریب تعیین): نشان می‌دهد که چه نسبتی از واریانس در RUL واقعی توسط مدل قابل پیش‌بینی است. مقدار نزدیک‌تر به ۱ بهتر است.

مدل	زمان آموزش (ثانیه) Colab T4 GPU	RMSE	MAE	MAPE	R-squared (R^2)
CNN+LSTM	105s	13.733	9.968	0.156	0.886
LSTM+ATTENTION	122s	14.276	10.393	0.166	0.876
LSTM	131s	14.804	10.721	0.170	0.867
CNN	63s	15.274	10.881	0.177	0.859
LSTM+CNN	110s	19.344	14.414	0.191	0.773

رسم منحنی خطای تجمعی(Cumulative Error Curve)

این نمودار نشان می‌دهد که چه درصدی از پیش‌بینی‌ها، خطایی کمتر از یک مقدار مشخص دارند. هر چه نمودار یک مدل سریع‌تر به سمت ۱۰۰٪ (یا ۱.۰) برود، آن مدل بهتر است.



۳. تحلیل عملکرد (خلاصه)

۱.کدام مدل دقیق‌ترین پیش‌بینی را ارائه داد؟ به نظر شما دلیل عملکرد بهتر آن چه بود؟

با توجه به تمام شواهد موجود ، مدل ترکیبی CNN+LSTM دقیق‌ترین پیش‌بینی را ارائه داد.

دلایل این نتیجه‌گیری:

معیارهای عددی:

در جدول نتایج، این مدل در تمام معیارهای کلیدی بهترین عملکرد را دارد: کمترین RMSE (۱۳.۷۳)، کمترین MAE (۹.۹۶)، کمترین MAPE (۰.۱۵) و بیشترین R-squared (۰.۸۸۶).

نمودار خطای تجمعی:

در این نمودار، منحنی مدل CNN+LSTM (خط سبز) یکی از سریع‌ترین منحنی‌ها در رسیدن به احتمال تجمعی بالا است، که نشان می‌دهد درصد بالایی از خطاهای آن کوچک بوده‌اند.

نمودار پیش‌بینی در برابر واقعیت:

پراکندگی نقاط در نمودار این مدل به وضوح کمتر و تجمع آنها به دور خط ایده‌آل (خط قرمز) بسیار متراکم است که دقت بالای آن را به صورت بصری تایید می‌کند.

دلیل عملکرد بهترCNN+LSTM :

دلیل اصلی برتری این معماری، همکاری مؤثر بین دو نوع شبکه با قابلیت‌های متفاوت است:

استخراج ویژگی توسطCNN : لایه Conv1D در ابتدای مدل مانند یک استخراج‌کننده ویژگی هوشمند عمل می‌کند. این لایه الگوهای محلی و کوتاه‌مدت را در توالی‌های ۳۰ مرحله‌ای داده‌های سنسورها تشخیص می‌دهد و آن‌ها را به یک نمایش فشرده‌تر و پرمعناتر تبدیل می‌کند.

درک توالی توسطLSTM : لایه LSTM این ویژگی‌های از پیش پردازش‌شده را دریافت کرده و وظیفه اصلی خود، یعنی درک وابستگی‌های زمانی و الگوهای بلندمدت بین این ویژگی‌ها را انجام می‌دهد.

به عبارت دیگر، CNN ابتدا چه چیزی مهم است را در هر لحظه تشخیص می‌دهد و LSTM سپس روند تغییر این چیزهای مهم در طول زمان چگونه است را یاد می‌گیرد. این استراتژی دو مرحله‌ای در این اجرا، بهترین نتیجه را به همراه داشته است.

۲. آیا هیچ یک از مدل‌ها نشانه‌هایی از بیش‌برازش (Overfitting) داشت؟ چگونه متوجه شدید؟

خیر، بر اساس نمودارهای یادگیری، هیچ یک از مدل‌ها نشانه‌های بیش‌برازش شدید را نشان ندادند.

این موضوع را با بررسی نمودار خطای آموزش در برابر خطای اعتبارسنجی برای هر مدل متوجه شدیم. در تمام مدل‌های موفق، منحنی خطای آموزش و خطای اعتبارسنجی روند بسیار نزدیکی به هم داشتند و با هم کاهش می‌یافتند. اگر بیش‌برازش رخ می‌داد، ما شاهد یک واگرایی مشخص (فاصله گرفتن دو منحنی از هم) بودیم. استفاده از لایه‌های Dropout در تمام معماری‌ها نیز نقش موثری در جلوگیری از وقوع بیش‌برازش داشته است.

۳. درباره تعادل بین دقت (Accuracy) و زمان آموزش بحث کنید.

نتایج این تمرین به خوبی این تعادل را به تصویر می‌کشد.

مدل‌های ساده و سریع: مدل CNN به دلیل ماهیت پردازش موازی خود، معمولاً سریع‌ترین مدل برای آموزش است. اما در این مسئله که درک توالی زمانی اهمیت دارد، این مدل دقت کمتری نسبت به مدل‌های پیچیده‌تر داشت.

مدل‌های پیچیده و دقیق: مدل‌های مبتنی بر LSTM و مدل‌های ترکیبی به دلیل ماهیت بازگشتی و پردازش مرحله به مرحله، ذاتاً کندتر از CNN هستند. مدل CNN+LSTM که دقیق‌ترین مدل ما بود، به دلیل داشتن هر دو نوع لایه، یک مدل نسبتاً پیچیده محسوب می‌شود و زمان آموزش بیشتری نسبت به CNN نیاز دارد.

نتیجه‌گیری از تعادل:

ما برای دستیابی به دقت بالاتر، زمان آموزش بیشتری را هزینه کردیم. مدل CNN+LSTM با افزایش پیچیدگی معماری، توانست به درک عمیق‌تری از داده‌ها برسد و خطای پیش‌بینی را به شکل قابل توجهی کاهش دهد. در یک کاربرد واقعی، انتخاب بین این مدل‌ها به نیاز پروژه بستگی دارد:

اگر دقت حداکثری اولویت اول باشد (مانند مسائل حیاتی در صنعت هوانوردی)، هزینه کردن زمان بیشتر برای آموزش یک مدل پیچیده مانند CNN+LSTM کاملاً منطقی است.

اگر سرعت در اولویت باشد و بتوان از مقداری خطا چشم‌پوشی کرد، ممکن است یک مدل ساده‌تر انتخاب شود.

چرا نتایج در هر بار اجرا کمی متفاوت است؟

این تفاوت در نتایج کاملاً طبیعی است و به دلیل وجود عوامل تصادفی (Stochasticity) در فرآیند آموزش شبکه‌های عصبی رخ می‌دهد. مهم‌ترین این عوامل عبارتند از:

۱. **مقداردهی اولیه تصادفی وزن‌ها(Random Weight Initialization)** : آموزش شبکه عصبی مانند شروع یک سفر در یک نقشه پیچیده برای پیدا کردن پایین‌ترین دره (کمترین خطا) است. نقطه‌ای که سفر از آن شروع می‌شود (وزن‌های اولیه) به صورت تصادفی انتخاب می‌شود. یک نقطه شروع متفاوت می‌تواند منجر به پیدا کردن یک مسیر کمی متفاوت و در نتیجه یک دره (حداقل محلی) متفاوت شود.

۲. **تصادفی بودنDropout** : لایه‌های Dropout در هر مرحله از آموزش، به صورت تصادفی برخی از نورون‌ها را غیرفعال می‌کنند. الگوی این غیرفعال‌سازی در هر بار اجرای کامل کد، متفاوت است.

۳. **تصادفی بودن در بهینه‌سازها:** برخی از الگوریتم‌های بهینه‌سازی نیز ممکن است دارای اجزای تصادفی در نحوه انتخاب دسته‌های داده (mini-batch) در هر دوره باشند.

اینکه نتایج کمی تغییر می‌کنند، نشان‌دهنده ضعف مدل‌ها نیست، بلکه طبیعت فرآیند آموزش است. نکته کلیدی این است که گروه مدل‌های برتر (CNN+LSTM, LSTM+Attention, LSTM) در چند اجرا ثابت مانده‌اند. این به ما اطمینان می‌دهد که این معماری‌های ترکیبی و بازگشتی، به طور کلی برای این مسئله بسیار مناسب هستند. در اجرای اخیر، معماری CNN+LSTM توانسته مسیر بهینه‌تری را در فرآیند آموزش پیدا کند و به عنوان مدل برتر ظاهر شود.