تمرین پنجم

(شبکه‌های بازگشتی)

آرین محمدخانی- ۸۱۰۶۰۳۱۳۶

هوش مصنوعی

۱. مقدمه

در این تمرین، به تمامی سوال‌های خواسته‌شده (اعم از سوال‌های اصلی و امتیازی) پاسخ داده شده است. همچنین، جهت کسب نمره‌ی امتیازی، فایل تمرین در مخزن GitHub به آدرس (AI-Recurrent-Neural-Networks) بارگذاری شده است. پوشه‌ی ارائه‌شده شامل موارد فایل‌های کد با فرمت‌های py. و ipynb.، گزارش تمرین در قالب‌های Word و PDF، داده‌ها و گزارش Jupyter Notebook می‌باشد. در ابتدای گزارش به بررسی روند انجام تمرین در هر مرحله پرداخته می‌شود. در این بررسی بخشی از کد هر قسمت به همراه بخشی از خروجی نمایش داده شده و روی آن بحث می‌شود. اگرچه در هر بخش تحلیلی از نتایج و روند تمرین ارائه شده اما تحلیل جامع در بخش انتهایی این گزارش ارائه شده است.

هدف این تمرین، آشنایی با شبکه‌های بازگشتی (RNN) و مقایسه‌ی عملکرد آن‌ها با شبکه‌های پیش‌خور (Feedforward) در مدل‌سازی پدیده‌های وابسته به زمان است. برای این منظور، از مجموعه داده‌ی C-MAPSS استفاده شده است؛ یکی از مجموعه‌داده‌های پرکاربرد ناسا در حوزه‌ی پایش وضعیت موتور توربوفن هواپیما. این مجموعه داده با استفاده از یک شبیه‌ساز پیشرفته تولید شده و شامل شرایط کاری موتور در طول چندین پرواز پی‌درپی و کیفیت عملکرد موتور در این شرایط می‌باشد. در هر پرواز، حسگرها پارامترهایی نظیر دما، فشار، و شرایط کاری موتور را با نرخ نمونه‌برداری ۱ هرتز ثبت می‌کنند. همچنین در برخی از پروازها، به‌صورت شبیه‌سازی‌شده خرابی‌هایی در اجزای مختلف موتور از جمله کمپرسور، توربین و فن ایجاد شده تا مدل‌ها بتوانند برای پیش‌بینی خرابی‌های احتمالی و زمان وقوع آن‌ها آموزش ببینند.

۲. پیاده‌سازی مدل‌ها

بارگذاری و انتخاب دادگان

در نخستین مرحله از این پروژه، پس از بارگزاری کتابخانه‌ها داده‌های مرتبط با تحلیل خرابی موتور جت از مجموعه داده‌های شبیه‌سازی‌شده C-MAPSS بارگذاری شد. به‌طور خاص، زیرمجموعه‌ی FD001 به عنوان مجموعه‌ی اصلی مورد استفاده قرار گرفت. این داده‌ها شامل اندازه‌گیری‌های حسگرها و تنظیمات عملیاتی برای هر موتور در طول زمان هستند. برای آغاز، فایل‌های train_FD001.txt، test_FD001.txt و RUL_FD001.txt با استفاده از تابع

pandas.read_csv و تنظیمات مناسب خوانده شدند. پس از بارگذاری داده‌ها، با توجه به ساختار خاص این فایل‌ها، ابتدا ستون‌های بدون نام حذف و سپس نام‌گذاری مشخصی برای هر ستون انجام شد که شامل شناسه موتور، زمان چرخه و ۲۱ اندازه‌گیری حسگر بود.

کد ۱

کد	نتیجه

cols = ['unit_number', 'time_in_cycles'] + \
[f'op_setting_{i}' for i in range(1, 4)] + \
[f'sensor_measurement_{i}' for i in range(1, 22)]
#train
train_FD1 = r'6. Turbofan Engine Degradation Simulation Data Set\train_FD001.txt'
train_df = pd.read_csv(train_FD1, sep=' ', header=None)
train_df.dropna(axis=1, inplace=True)
train_df.columns = cols
train_df.info()
#test
test_FD1 = r'6. Turbofan Engine Degradation Simulation Data Set\test_FD001.txt'
test_df = pd.read_csv(test_FD1, sep=' ', header=None)
test_df.dropna(axis=1, inplace=True)
test_df.columns = cols
#real_RUL
real_RUL = r'6. Turbofan Engine Degradation Simulation Data Set\RUL_FD001.txt'
rul_t = pd.read_csv(real_RUL, sep=' ', header=None)
rul_t = rul_t.dropna(axis=1)
rul_t.columns = ['RUL']
	<class 'pandas.core.frame.DataFrame'>
	RangeIndex: 20631 entries, 0 to 20630
	Data columns (total 26 columns):
	# Column Non-Null Count Dtype
	--- ------ -------------- -----
	0 unit_number 20631 non-null int64
	1 time_in_cycles 20631 non-null int64
	2 op_setting_1 20631 non-null float64
	3 op_setting_2 20631 non-null float64
	4 op_setting_3 20631 non-null float64
	5 sensor_measurement_1 20631 non-null float64
	6 sensor_measurement_2 20631 non-null float64
	7 sensor_measurement_3 20631 non-null float64
	8 sensor_measurement_4 20631 non-null float64
	9 sensor_measurement_5 20631 non-null float64
	10 sensor_measurement_6 20631 non-null float64
	11 sensor_measurement_7 20631 non-null float64
	12 sensor_measurement_8 20631 non-null float64
	13 sensor_measurement_9 20631 non-null float64
	14 sensor_measurement_10 20631 non-null float64
	15 sensor_measurement_11 20631 non-null float64
	16 sensor_measurement_12 20631 non-null float64
	17 sensor_measurement_13 20631 non-null float64
	18 sensor_measurement_14 20631 non-null float64
	19 sensor_measurement_15 20631 non-null float64
	20 sensor_measurement_16 20631 non-null float64
	21 sensor_measurement_17 20631 non-null int64
	22 sensor_measurement_18 20631 non-null int64
	23 sensor_measurement_19 20631 non-null float64
	24 sensor_measurement_20 20631 non-null float64
	25 sensor_measurement_21 20631 non-null float64
	dtypes: float64(22), int64(4)
	memory usage: 4.1 MB

سپس برای داده‌های train و test، عمر باقی مانده با توجه به ماکسیمم چرخه زده شده هر موتور، محاسبه و به آن‌ها اضافه می‌شود. برای محاسبه عمر باقی مانده هر موتور، ماکسیمم دوره‌ای که هر موتور طی کرده است، به آن موتور نسبت داده شده و سپس از سیکل فعلی آن کم می‌شود.

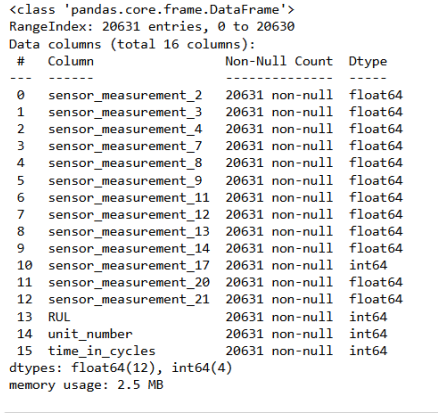
کد ۲

```
# calculate RUL
rul_df = train_df.groupby('unit_number')['time_in_cycles'].max().reset_index()
rul_df.columns = ['unit_number', 'max_cycle']
train_df = train_df.merge(rul_df, on='unit_number')
train_df['RUL'] = train_df['max_cycle'] - train_df['time_in_cycles']
train_df.head()

max_cycles = test_df.groupby('unit_number')['time_in_cycles'].max().reset_index()
max_cycles.columns = ['unit_number', 'max_cycle']
max_cycles['true_RUL'] = rul_t['RUL']
max_cycles['total_life'] = max_cycles['max_cycle'] + max_cycles['true_RUL']
test_df = test_df.merge(max_cycles[['unit_number', 'total_life']], on='unit_number')
test_df['RUL'] = test_df['total_life'] - test_df['time_in_cycles']
```

در گام بعدی، فرآیند پاک‌سازی و کاهش ابعاد ویژگی‌ها (feature reduction) به‌منظور حذف ویژگی‌های نامؤثر یا تکراری انجام گرفت. ابتدا با استفاده از کلاس VarianceThreshold ویژگی‌هایی که واریانس کمتر از ۰.۰۱ داشتند (ویژگی‌های با تغییرپذیری کم) شناسایی و حذف شدند. سپس برای حذف ویژگی‌های بسیار مشابه، ماتریس همبستگی ویژگی‌های باقی‌مانده محاسبه شد و ویژگی‌هایی با ضریب همبستگی بیش از ۰.۹۸، به عنوان تکراری در نظر گرفته شده و کنار گذاشته شدند. در مرحله‌ی بعد، رابطه‌ی ویژگی‌ها با متغیر هدف یعنی RUL بررسی گردید. ویژگی‌هایی که همبستگی کمتر از ۰.۰۵ با متغیر هدف داشتند، حذف شدند. این گام به بهبود مدل‌پذیری و کاهش نویز داده کمک می‌کند. در نهایت، برای ارزیابی اهمیت نهایی ویژگی‌ها، از مدل RandomForestRegressor استفاده شد. ویژگی‌هایی که اهمیت آن‌ها طبق این مدل کمتر از ۰.۰۱ بود نیز حذف شدند. خروجی این مرحله، مجموعه‌ای از ویژگی‌های حسگر پالایش‌شده و مهم بود که به‌همراه متغیرهای unit_number و time_in_cycles در دیتافریم نهایی حفظ شد.

کد ۳

کد	نتیجه
sensor_cols = [col for col in train_df.columns if "sensor" in col] selector = VarianceThreshold(threshold=0.005) selector.fit(train_df[sensor_cols]) useful_sensor_cols = list(selector.get_feature_names_out(sensor_cols))	
corr_matrix = train_df[useful_sensor_cols].corr().abs() upper = corr_matrix.where(np.triu(np.ones(corr_matrix.shape), k=1).astype(bool)) to_drop = [col for col in upper.columns if any(upper[col] > 0.99)] filtered_cols = [col for col in useful_sensor_cols if col not in to_drop]	
cor_with_target = train_df[filtered_cols + ['RUL']].corr()['RUL'].abs().drop('RUL') low_corr_features = cor_with_target[cor_with_target < 0.03].index.tolist() final_sensor_cols = [col for col in filtered_cols if col not in low_corr_features]	
X_rf = train_df[final_sensor_cols] y_rf = train_df['RUL'] rf = RandomForestRegressor(n_estimators=100, random_state=42) rf.fit(X_rf, y_rf) importances = pd.Series(rf.feature_importances_, index=final_sensor_cols) low_importance_features = importances[importances < 0.005].index.tolist() selected_features = [col for col in final_sensor_cols if col not in low_importance_features]	

تقسیم‌بندی و نرمال‌سازی داده‌ها

در این مرحله، داده‌های آماده‌شده برای آموزش مدل‌های یادگیری عمیق، پردازش‌های بیشتری را تجربه کردند تا به شکل مناسب برای مدل‌های مبتنی بر یادگیری عمیق مانند LSTM و CNN تبدیل شوند. ابتدا ویژگی‌های منتخب با استفاده از روش MinMaxScaler نرمال‌سازی شدند تا مقادیر تمام ویژگی‌ها در بازه [۰, ۱] قرار گیرند. این کار باعث هم‌مقیاس شدن ورودی‌ها و بهبود عملکرد مدل‌های یادگیری عمیق می‌شود. سپس مجموعه داده به دو بخش آموزشی و آزمون تقسیم شد و از میان داده‌های آموزشی، ٪۷۰ واحدهای عملیاتی برای آموزش و ٪۳۰ باقیمانده برای اعتبارسنجی در نظر گرفته شد. به‌منظور مدل‌سازی توالی‌های زمانی، داده‌ها به صورت پنجره‌ای (sliding window) با اندازه ۳۰ چرخه متوالی آماده‌سازی شدند. در هر پنجره، ۳۰ مقدار متوالی از ویژگی‌ها به‌عنوان ورودی (X) در نظر گرفته شده و مقدار RUL مربوط به آخرین زمان آن پنجره به‌عنوان برچسب (y) انتخاب شد. همچنین، برای جلوگیری از تأثیر مقادیر بسیار بزرگ RUL و تثبیت آموزش، مقادیر RUL در هر دو مجموعه‌ی آموزشی و آزمون در سقف ۱۳۰ محدود شدند. این مرحله از پیش‌پردازش، پایه‌ی داده‌ای مناسبی را برای آموزش مدل‌های یادگیری عمیق همچون LSTM، CNN و ساختارهای ترکیبی فراهم کرده است. با مقدار ۱۳۰

جایگزین شدند تا تمرکز مدل بر پیش‌بینی‌های واقع‌بینانه‌تر معطوف گردد و پایداری در فرآیند یادگیری افزایش یابد. در پایان این مرحله، ۱۷۷۳۱ پنجره‌ی آموزشی، هرکدام شامل ۳۰ مرحله زمانی و ۱۳ ویژگی و ۱۰۱۹۶ پنجره تست با همان ساختار تشکیل شد.

کد ۴

```
scaler = MinMaxScaler()
train_df_scaled = train_df.copy()
train_df_scaled[selected_features] = scaler.fit_transform(train_df[selected_features])
test_df_scaled = test_df.copy()
test_df_scaled[selected_features] = scaler.transform(test_df[selected_features])
unique_units = train_df_scaled['unit_number'].unique()
train_units = unique_units[:int(0.7 * len(unique_units))]
val_units = unique_units[int(0.7 * len(unique_units)):]
def prepare_windows(data, window_size=30):
  X, y = [], []
  for unit in data['unit_number'].unique():
    unit_data = data[data['unit_number'] == unit].sort_values('time_in_cycles')
    features = unit_data[selected_features].values
    rul = unit_data['RUL'].values
    for i in range(len(unit_data) - window_size + 1):
      X.append(features[i:i + window_size])
      y.append(rul[i + window_size - 1])
  return np.array(X), np.array(y)
X, y = prepare_windows(train_df_scaled, window_size=30)
X_test, y_test = prepare_windows(test_df_scaled)

X_train, y_train = prepare_windows(train_df_scaled[train_df_scaled['unit_number'].isin(train_units)])
X_val, y_val = prepare_windows(train_df_scaled[train_df_scaled['unit_number'].isin(val_units)])
max_rul = 130
train_df['RUL'] = train_df['RUL'].clip(upper=max_rul)
test_df['RUL'] = test_df['RUL'].clip(upper=max_rul)
```

۳. طراحی و آموزش مدل CNN

در این مرحله، یک مدل یادگیری عمیق مبتنی بر شبکه عصبی پیچشی یک‌بعدی طراحی و آموزش داده شد. هدف از به‌کارگیری مدل CNN، استخراج خودکار ویژگی‌های محلی و زمانی از

داده‌های حسگر بود؛ این ویژگی‌ها به‌ویژه برای تحلیل سری‌های زمانی مفید هستند. معماری مدل شامل دو لایه‌ی اصلی Conv1D بود. لایه‌ی نخست از ۶۴ فیلتر با اندازه‌ی کرنل ۳ و تابع فعال‌سازی ReLU استفاده کرد، و لایه‌ی دوم شامل ۱۲۸ فیلتر با همان مشخصات بود. به‌منظور کاهش ابعاد، پس از لایه‌ی اول از MaxPooling با اندازه‌ی ۲ بهره گرفته شد. سپس با استفاده از Global Average Pooling، اطلاعات مکانی در خروجی فشرده گردید. پس از آن، یک لایه‌ی Dense با ۶۴ نرون و تابع ReLU قرار گرفت و برای مقابله با بیش‌برازش (Overfitting)، یک لایه‌ی Dropout با نرخ ۰.۲ اضافه شد. در انتها، یک لایه‌ی خروجی تک‌نرونی بدون تابع فعال‌سازی به‌منظور پیش‌بینی مقدار RUL اضافه شد. مدل با استفاده از تابع زیان MSE (میانگین مربعات خطا) و بهینه‌ساز Adam با نرخ یادگیری ۰.۰۰۱ کامپایل گردید. همچنین، از مکانیزم توقف زودهنگام (EarlyStopping) با معیار نظارت بر val_loss و صبر ۲۰ دوره برای جلوگیری از بیش‌برازش بهره گرفته شد. مدل به مدت حداکثر ۱۰۰ دوره آموزشی با اندازه‌ی دسته (Batch Size) برابر با ۶۴ آموزش داده شد.

کد ۵

```
model_cnn = Sequential([
    Conv1D(filters=64, kernel_size=3, activation='relu', padding='same', input_shape=(X.shape[1], X.shape[2])),
    MaxPooling1D(pool_size=2),
    Conv1D(filters=128, kernel_size=3, activation='relu', padding='same'),
    GlobalAveragePooling1D(),
    Dense(64, activation='relu'),
    Dropout(0.2),
    Dense(1) ])
model_cnn.compile(optimizer=Adam(learning_rate=0.001), loss='mse', metrics=['mae'])
early_stop_cnn = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)
start_time_cnn = time.time()
history_cnn = model_cnn.fit(
    X_train, y_train,
    validation_data=(X_val, y_val),
    epochs=100,
    batch_size=64,
    callbacks=[early_stop_cnn],
    verbose=1)
train_time_cnn = time.time() - start_time_cnn
y_pred_cnn = model_cnn.predict(X_test).flatten()
```

پس از آموزش مدل به مدت حداکثر ۱۰۰ دوره، به‌دلیل فعال بودن تکنیک توقف Early Stopping، فرایند آموزش در دوره ۷۵ متوقف شد؛ مدل در این نقطه بدون بروز بیش‌برازش محسوس، بهترین عملکرد خود را روی داده‌های اعتبارسنجی با مقدار val_loss = 1764.05 و val_mae =19.92 به‌دست آورد. سایر نتایج در بخش پایانی آورده شده است.

۴. آموزش مدل مبتنی بر LSTM

در این بخش از پروژه، مدلی مبتنی بر شبکه حافظه بلندمدت کوتاه‌مدت (LSTM) طراحی و پیاده‌سازی گردید. این نوع شبکه برای تحلیل داده‌های سری زمانی وابسته به ترتیب بسیار مناسب است، زیرا می‌تواند وابستگی‌های بلندمدت در توالی داده‌ها را حفظ کرده و از آن‌ها برای پیش‌بینی استفاده نماید. معماری مدل از دو لایه LSTM تشکیل شد. لایه‌ی اول دارای ۱۰۰ واحد و با return_sequences=True تعریف گردید تا توالی کامل خروجی را برای لایه بعدی حفظ کند. لایه‌ی دوم شامل ۵۰ واحد LSTM بود که تنها آخرین حالت خروجی را تولید می‌کرد. بین این دو لایه و همچنین پس از لایه‌ی Dense، لایه‌های Dropout با نرخ ۰.۲ برای جلوگیری از بیش‌برازش گنجانده شد. پس از لایه‌های بازگشتی، یک لایه Dense با ۶۴ نرون و تابع فعال‌سازی ReLU و سپس یک لایه خروجی تک‌نرونی بدون تابع فعال‌سازی به‌عنوان خروجی مدل برای پیش‌بینی RUL در نظر گرفته شد. مدل با استفاده از بهینه‌ساز Adam با نرخ یادگیری ۰.۰۰۱ و تابع زیان MSE (میانگین مربعات خطا) کامپایل گردید. همچنین، از مکانیزم توقف زودهنگام جهت جلوگیری از بیش‌برازش با نظارت بر مقدار val_loss استفاده شد. حداکثر تعداد دوره‌های آموزشی ۱۰۰ دوره و اندازه دسته آموزشی برابر با ۶۴ انتخاب شد.پس از آموزش مدل بر روی داده‌های آموزش و اعتبارسنجی، عملکرد آن با استفاده از داده‌های آزمون مورد ارزیابی قرار گرفت.

کد ۶

```
model_lstm = Sequential([
    LSTM(100, return_sequences=True, input_shape=(X.shape[1], X.shape[2])),
    Dropout(0.2),
    LSTM(50, return_sequences=False),
    Dense(64, activation='relu'),
    Dropout(0.2),
    Dense(1)  ])
model_lstm.compile(
    optimizer=Adam(learning_rate=0.001),
    loss='mse',
    metrics=['mae'])
early_stop_lstm = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)
start_time_lstm = time.time()
history_lstm = model_lstm.fit(
    X_train, y_train,
    validation_data=(X_val, y_val),
    epochs=100,
    batch_size=64,
    callbacks=[early_stop_lstm],
    verbose=1)
train_time_lstm = time.time() - start_time_lstm
y_pred_lstm = model_lstm.predict(X_test).flatten()
mae_lstm = mean_absolute_error(y_test, y_pred_lstm)
rmse_lstm = np.sqrt(mean_squared_error(y_test, y_pred_lstm))
log_rmse_lstm = np.sqrt(mean_squared_error(np.log1p(y_test), np.log1p(y_pred_lstm)))
print(f"LSTM MAE: {mae_lstm:.2f}")
print(f"LSTM RMSE: {rmse_lstm:.2f}")
print(f"LSTM log-RMSE: {log_rmse_lstm:.2f}")
```

پس از آموزش مدل برای حداکثر ۱۰۰ دوره، فرایند یادگیری در دوره ۳۳ به‌دلیل فعال بودن تکنیک Early Stopping متوقف شد. طی مراحل آموزش، مدل به تدریج توانست میزان خطا را کاهش دهد و بهترین عملکرد خود را در اپوک ۲۷ با مقدار val_loss = 1170.5588 و val_mae = 21.7881 روی داده‌های اعتبارسنجی به‌دست آورد. از آنجا که پس از آن، عملکرد مدل در مجموعه اعتبارسنجی بهبود نیافت، الگوریتم توقف زودهنگام مانع از ادامه آموزش و بروز بیش‌برازش شد.

۵. تنظیم ابرپارامترها

در مرحلهٔ چهارم، هدف اصلی بهینه‌سازی عملکرد دو مدل یادگیری عمیق شامل شبکه CNN و شبکه حافظه‌دار بلندمدت LSTM از طریق تنظیم ابرپارامترها بوده است. برای این منظور، از ابزار KerasTuner با روش جستجوی تصادفی RandomSearch بهره گرفته شد. در هر مدل، مجموعه‌ای از پارامترها مانند نرخ یادگیری، نوع بهینه‌ساز (Adam یا RMSprop)، نرخ Dropout، تعداد فیلترها (در CNN) یا تعداد واحدهای LSTM و اندازه پنجره کانولوشن (در CNN) مورد بررسی قرار گرفت. هر مدل در طی ۱۰ آزمایش مختلف مورد ارزیابی قرار گرفت و بهترین ترکیب پارامترها براساس معیار val_loss انتخاب شد. در نهایت، مدل‌های منتخب روی مجموعه تست ارزیابی شدند و خطای MAE آنها گزارش گردید. طبق نتایج به‌دست‌آمده، در طی این ۱۰ آزمایش برای هر نمونه بهترین ابرپارامترها در جدول زیر نمایش داده شده است. سایر پارامترها و نتایج آن‌ها در بخش بعدی ارائه داده می‌شود.

جدول1 ، بهترین پارامترها

مدل	فیلترها / واحدها	Kernel Size	Dropout	Optimizer	Learning Rate	Test MAE
CNN	filters1,2 = 128	2	0.1	RMSprop	0.01	28.5433
LSTM	lstm_units1 = 100 lstm_units2 = 100	—	dropout1 = 0.1 dropout2 = 0.3	Adam	0.001	27.3985

از آنجایی که کد این بخش بسیار طولانی می‌باشد، برای نمونه تنها تعریف تابعی که بر روی مدل CNN آزمایش می‌کند نشان داده شده است که در جدول زیر نمایش دده شده است.

کد ۷

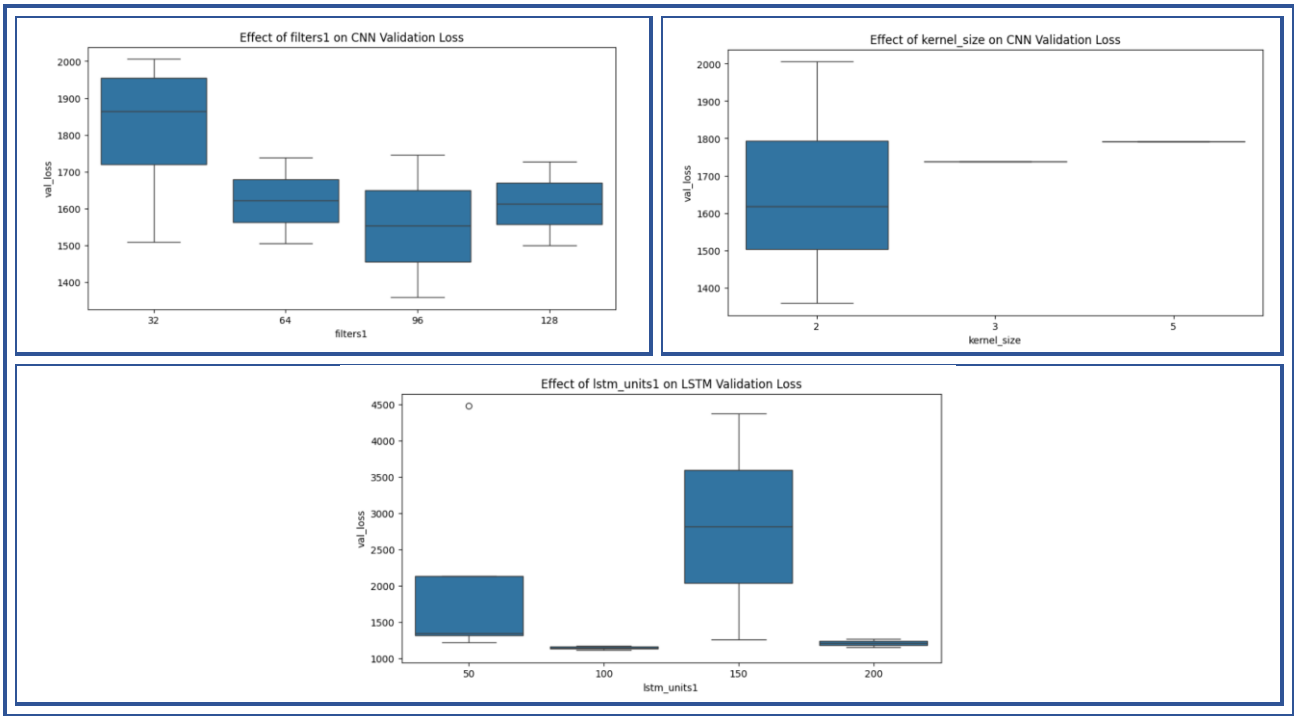
کد	نتیجه
def build_cnn_model(hp): model = Sequential() model.add(Conv1D(filters=hp.Int('filters1', 32, 128, step=32), kernel_size=hp.Choice('kernel_size', [2, 3, 5]), activation='relu', padding='same', input_shape=(X.shape[1], X.shape[2]))) model.add(MaxPooling1D(pool_size=2)) model.add(Conv1D(filters=hp.Int('filters2', 32, 128, step=32), kernel_size=3, activation='relu', padding='same')) model.add(GlobalAveragePooling1D()) model.add(Dense(64, activation='relu')) model.add(Dropout(hp.Float('dropout_rate', 0.1, 0.5, step=0.1))) model.add(Dense(1)) optimizer_name = hp.Choice('optimizer', ['adam', 'rmsprop']) lr = hp.Choice('learning_rate', [1e-2, 1e-3, 1e-4]) optimizer = Adam(learning_rate=lr) if optimizer_name == 'adam' else RMSprop(learning_rate=lr) model.compile(optimizer=optimizer, loss='mse', metrics=['mae']) return model	319/319 [==============================] 🔷 Best CNN Hyperparameters: filters1: 96 filters2: 128 kernel_size: 2 dropout_rate: 0.1 optimizer: adam learning_rate: 0.01 CNN Test MAE: 30.5204 319/319 [==============================] 🔶 Best LSTM Hyperparameters: lstm_units1: 100 lstm_units2: 100 dropout1: 0.1 dropout2: 0.30000000000000004 optimizer: adam learning_rate: 0.001 LSTM Test MAE: 27.1569

پاسخ به سوالات مرحله ۴

در این بخش به سوالات مطرح شده، پاسخ داده می شود. سوالات و خواسته‌هایی که در فایل تمرین مطرح شد عبارت است از:

- **کدام پارامترها بیشترین تأثیر را بر عملکرد مدل داشتند؟**
- **جدولی یا گزارشی از اجرای مدل با تنظیمات مختلف و نتایج ارائه دهید.**
- **درباره تعادل بین زمان آموزش، دقت و پیچیدگی مدل بحث کنید.**

در پاسخ به سؤال نخست، با تحلیل نتایج به‌دست‌آمده از جست‌وجوی و بررسی نمودارهای جعبه‌ای (Boxplot)، مشخص شد که در میان پارامترهای بهینه‌سازی‌شده، پارامتر filters1 در مدل CNN بیشترین تأثیر را بر مقدار خطای اعتبارسنجی دارد. این موضوع از طریق تغییرات معنادار و کاهش محسوس مقدار val_loss در هنگام تغییر filters1 به‌خصوص در بازه ۶۴ تا ۹۶ تأیید می‌شود. پارامترهای دیگر مانند kernel_size در مدل CNN و در مدل LSTM نیز تأثیر قابل توجهی داشتند، اما تغییرات آن‌ها یا به اندازه filters1 منظم و پایدار نبودند یا حجم داده در برخی مقادیر آن‌ها کافی نبوده است. بنابراین، می‌توان نتیجه گرفت که filters1 مهم‌ترین پارامتر تأثیرگذار در بهینه‌سازی عملکرد مدل CNN در این مسئله پیش‌بینی RUL است.



تصویر ۱، نمودارهای جعبه‌ای با بیشترین تاثیر در دقت پایانی

در پاسخ به دومین سوال جدول زیر که حاصل از ۲۰ آزمایش مختلف بر روی دو نمونه است، ارائه شده است.

جدول2 ، حالت‌های مختلف با هایپرپارامترهای متفاوت

مدل	فیلتر/یونیت‌ها	Dropout	Optimizer	LR	Val Loss	MAE
LSTM	100, 100	0.1, 0.3	adam	0.0010	1116.12	20.14
LSTM	200, 50	0.1, 0.3	adam	0.0010	1158.69	20.92
LSTM	100, 150	0.3, 0.3	adam	0.0010	1178.81	21.57
LSTM	50, 150	0.5, 0.5	rmsprop	0.0100	1222.92	22.40
LSTM	150, 100	0.1, 0.2	rmsprop	0.0010	1257.44	20.57
LSTM	200, 150	0.1, 0.1	rmsprop	0.0100	1275.44	20.26
CNN	96, 128 (kernel=2)	0.1	adam	0.0100	1359.02	20.18
LSTM	50, 150	0.2, 0.3	adam	0.0001	1350.67	22.31
LSTM	50, 150	0.1, 0.4	rmsprop	0.0100	1351.24	22.07
CNN	128, 128 (kernel=2)	0.1	rmsprop	0.0100	1500.03	24.33
CNN	64, 64 (kernel=2)	0.3	adam	0.0010	1504.78	24.03
CNN	32, 96 (kernel=2)	0.4	rmsprop	0.0100	1508.38	26.91
CNN	128, 64 (kernel=2)	0.3	adam	0.0010	1726.74	27.57
CNN	64, 128 (kernel=3)	0.3	adam	0.0010	1737.75	27.99
CNN	96, 64 (kernel=2)	0.2	rmsprop	0.0010	1746.52	27.22
CNN	32, 128 (kernel=5)	0.5	rmsprop	0.0100	1790.97	32.95
CNN	32, 128 (kernel=2)	0.3	adam	0.0001	1937.68	29.44
CNN	32, 64 (kernel=2)	0.5	rmsprop	0.0001	2005.87	32.08
LSTM	150, 150	0.1, 0.1	adam	0.0100	4377.22	48.85
LSTM	50, 100	0.1, 0.5	rmsprop	0.0001	4481.02	48.98

و در انتها، از نظر تعادل بین زمان آموزش، دقت و پیچیدگی مدل، می‌توان گفت که مدل CNN از لحاظ ساختاری ساده‌تر بوده و سریع‌تر آموزش می‌بیند، اما در مقابل، مدل LSTM علی‌رغم زمان آموزش طولانی‌تر، دقت بالاتری در پیش‌بینی خروجی‌ها داشته است. این تفاوت عملکرد ناشی از توانایی LSTM در مدل‌سازی وابستگی‌های زمانی و دنباله‌ای در داده‌هاست که در بسیاری از مسائل پیش‌بینی، از جمله تخمین RUL، مزیت مهمی محسوب می‌شود. در مجموع، اگر اولویت با دقت پیش‌بینی باشد، LSTM گزینه بهتری است، در حالی که اگر منابع محاسباتی محدود و زمان آموزش مهم باشد، CNN می‌تواند انتخابی مناسب‌تر باشد.

۶. پیاده‌سازی مدل ترکیبی CNN + LSTM

در این مرحله، دو مدل ترکیبی قدرتمند شامل CNN-LSTM و LSTM-CNN برای پیش‌بینی RUL طراحی و آموزش داده شدند. هدف از این ترکیب، بهره‌گیری از مزایای معماری‌های مختلف یادگیری عمیق در تحلیل داده‌های سری‌زمانی بود. در مدل CNN-LSTM، ابتدا لایه‌ی کانولوشن یک‌بعدی با ۶۴ فیلتر و کرنل اندازه ۳ برای استخراج ویژگی‌های مکانی از داده‌های ورودی به‌کار گرفته شد. سپس، با استفاده از لایه MaxPooling1D ابعاد کاهش یافته و خروجی به لایه LSTM با ۱۰۰ واحد مخفی منتقل شد تا وابستگی‌های زمانی میان ویژگی‌ها مدل‌سازی شود. در ادامه، از لایه Dropout با نرخ ۰.۳ برای کاهش بیش‌برازش و یک لایه Dense با ۶۴ نرون برای پردازش نهایی استفاده شد. خروجی نهایی نیز از یک نرون در لایه Dense آخر تولید شد. این مدل با بهینه‌ساز Adam و تابع زیان MSE کامپایل و با مکانیزم EarlyStopping آموزش داده شد.

کد ۸

```
model_cnn_lstm = Sequential([
   Conv1D(filters=64, kernel_size=3, activation='relu', padding='same', input_shape=(X.shape[1], X.shape[2])),
   MaxPooling1D(pool_size=2), LSTM(100, return_sequences=False), Dropout(0.3),Dense(64, activation='relu'),
   Dense(1)  ])
model_cnn_lstm.compile(
   optimizer=Adam(learning_rate=0.001),
   loss='mse',
   metrics=['mae'])
early_stop_cnn_lstm = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)
start_time_cnn_lstm = time.time()
history_cnn_lstm = model_cnn_lstm.fit(
   X_train, y_train, validation_data=(X_val, y_val),epochs=100,batch_size=64,callbacks=[early_stop_cnn_lstm],
   verbose=1)
train_time_cnn_lstm = time.time() - start_time_cnn_lstm
y_pred_cnn_lstm = model_cnn_lstm.predict(X_test).flatten()
mae_cnn_lstm = mean_absolute_error(y_test, y_pred_cnn_lstm)
rmse_cnn_lstm = np.sqrt(mean_squared_error(y_test, y_pred_cnn_lstm))
log_rmse_cnn_lstm = np.sqrt(mean_squared_error(np.log1p(y_test), np.log1p(y_pred_cnn_lstm)))
```

در طی فرآیند آموزش مدل، روند کاهش خطای آموزش و اعتبارسنجی نشان‌دهنده‌ی عملکرد مطلوب مدل در مراحل ابتدایی بوده است. در سه epoch نخست، کاهش چشمگیر در مقادیر val_loss و val_mae مشاهده می‌شود؛ به‌گونه‌ای که val_loss از حدود ۴۵۰۰ به ۱۶۷۵ و val_mae از ۵۴ به حدود ۲۸ کاهش یافته است. این افت سریع، بیانگر توانایی اولیه مدل در یادگیری الگوهای داده‌هاست. از epoch چهارم تا دهم، بهبود مدل به‌صورت تدریجی و پایدار ادامه داشته و val_mae تا حدود ۲۳ کاهش یافته است. با این حال، از epoch یازدهم به بعد، نوساناتی در مقادیر اعتبارسنجی ظاهر می‌شود که نشان‌دهنده‌ی بی‌ثباتی در عملکرد مدل است. این نوسانات در حدود مقادیر val_loss بین ۱۱۵۰ تا ۱۴۵۰ و val_mae بین ۲۲.۵ تا ۲۴ باقی می‌ماند. این رفتار می‌تواند نشانه‌ای از overfitting یا نزدیک شدن مدل به ظرفیت یادگیری خود باشد. و در انتها از آنجا که پس از آن، عملکرد مدل در مجموعه اعتبارسنجی بهبود نیافت، الگوریتم توقف زودهنگام مانع از ادامه آموزش و بروز بیش‌برازش شد.

در مدل دوم یعنی LSTM-CNN، ابتدا لایه LSTM با خروجی ترتیبی (return_sequences=True) برای یادگیری الگوهای زمانی اجرا شد و سپس خروجی آن به لایهConv1D منتقل شد تا ویژگی‌های مکانی ثانویه استخراج گردد. در ادامه، از لایه GlobalAveragePooling1D برای فشرده‌سازی ویژگی‌ها، Dropout با نرخ ۰.۳ و لایه‌های Dense جهت تولید خروجی نهایی استفاده شد. این مدل نیز با همان تنظیمات آموزش داده شد و عملکردی مشابه یا در برخی موارد بهبود یافته ارائه داد. در این مدل نیز روند آموزش مانند مدل قبلی می‌باشد با این تفاوت که توفق زودهنگام باعث توفق در دوره ۳۱ با مقدارير val_loss = 1953.09 و val_mae = 29.476 شده است.

کد ۹

```
model_lstm_cnn = Sequential([
    LSTM(100, return_sequences=True, input_shape=(X.shape[1], X.shape[2])),
    Conv1D(filters=64, kernel_size=3, activation='relu', padding='same'),
    GlobalAveragePooling1D(),
    Dropout(0.3),
    Dense(64, activation='relu'),
    Dense(1)])
model_lstm_cnn.compile(optimizer=Adam(learning_rate=0.001), loss='mse', metrics=['mae'])
start_time_lstm_cnn = time.time()
history_lstm_cnn = model_lstm_cnn.fit(
    X_train, y_train,
    validation_data=(X_val, y_val),
    epochs=100,
    batch_size=64,
    callbacks=[early_stop_cnn_lstm],
    verbose=1)
train_time_lstm_cnn = time.time() - start_time_lstm_cnn
y_pred_lstm_cnn = model_lstm_cnn.predict(X_test).flatten()
mae_lstm_cnn = mean_absolute_error(y_test, y_pred_lstm_cnn)
rmse_lstm_cnn = np.sqrt(mean_squared_error(y_test, y_pred_lstm_cnn))
log_rmse_lstm_cnn = np.sqrt(mean_squared_error(np.log1p(y_test), np.log1p(y_pred_lstm_cnn)))
r2_lstm_cnn = r2_score(y_test, y_pred_lstm_cnn)
print(f"LSTM-CNN MAE: {mae_lstm_cnn:.2f}")
print(f"LSTM-CNN RMSE: {rmse_lstm_cnn:.2f}")
print(f"LSTM-CNN log-RMSE: {log_rmse_lstm_cnn:.2f}")
```

۷. پیاده‌سازی معماری LSTM + ATTENTION

در این مرحله، مدلی مبتنی بر شبکه LSTM همراه با لایه توجه (Attention Layer) طراحی و پیاده‌سازی شد تا دقت پیش‌بینی عمر باقی‌مانده (RUL) در داده‌های سری‌زمانی بهبود یابد. هدف اصلی از افزودن مکانیزم توجه، تمرکز مدل بر مهم‌ترین بازه‌های زمانی در طول هر پنجره ورودی بود، زیرا در داده‌های حسگری که دارای وابستگی‌های زمانی بلندمدت هستند، همه لحظات گذشته به یک اندازه تأثیرگذار نیستند. معماری مدل شامل یک لایه LSTM با ۱۰۰ واحد و خروجی ترتیبی، به‌همراه لایه Dropout برای جلوگیری از بیش‌برازش، یک لایه توجه سفارشی برای وزن‌دهی به لحظات مختلف زمانی، و در ادامه لایه‌های Fully Connected با ۶۴ نرون و

ReLU، Dropout دوم، و در نهایت یک لایه Dense برای پیش‌بینی RUL بود. لایه توجه به‌صورت سفارشی طراحی شد تا با یادگیری وزن‌های قابل آموزش، خروجی زمانی LSTM را به‌طور وزن‌دار ترکیب کند؛ به‌طوری‌که اطلاعات مهم‌تر در تصمیم‌گیری مدل پررنگ‌تر شوند.مدل با استفاده از بهینه‌ساز Adam با نرخ یادگیری ۰.۰۰۱ و تابع زیان MSE آموزش داده شد. برای جلوگیری از بیش‌برازش، مکانیزم EarlyStopping با معیار val_loss و آستانه توقف ۱۰ دوره مورد استفاده قرار گرفت. پس از آموزش کامل مدل، عملکرد آن روی مجموعه آزمون بررسی شد.

کد ۱۰

```
class AttentionLayer(Layer):
    def __init__(self, **kwargs):
        super(AttentionLayer, self).__init__(**kwargs)
    def build(self, input_shape):
        self.W = self.add_weight(name="att_weight", shape=(input_shape[-1], 1,(
                                initializer="random_normal", trainable=True(
        self.b = self.add_weight(name="att_bias", shape=(input_shape[1], 1,(
                                initializer="zeros", trainable=True(
        super(AttentionLayer, self).build(input_shape)
    def call(self, x):
        e = K.tanh(K.dot(x, self.W) + self.b)
        a = K.softmax(e, axis=1)
        output = x * a
        return K.sum(output, axis=1)

input_layer = Input(shape=(X.shape[1], X.shape[2]))
lstm_out = LSTM(100, return_sequences=True)(input_layer)
dropout = Dropout(0.3)(lstm_out)
attention_out = AttentionLayer()(dropout)
dense1 = Dense(64, activation='relu')(attention_out)
```

```
    dropout2 = Dropout(0.3)(dense1)
    output = Dense(1)(dropout2)
    model_lstm_att = Model(inputs=input_layer, outputs=output)
    model_lstm_att.compile(
        loss='mse,'
        optimizer=Adam(learning_rate=0.001),
        metrics=['mae'])
    start_time_lstm_att = time.time()
    history_lstm_att = model_lstm_att.fit(
      X_train, y_train,
      validation_data=(X_val, y_val),
      epochs=100,
      batch_size=64,
      callbacks=[EarlyStopping(monitor='val_loss', patience=10,
restore_best_weights=True)],
      verbose=1)
    train_time_lstm_att = time.time() - start_time_lstm_att
    y_pred_att = model_lstm_att.predict(X_test).flatten()
```

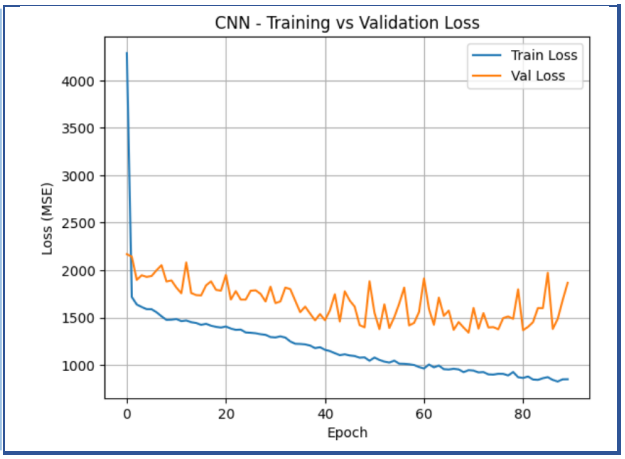
در این مدل، طی ۳۰ دوره آموزش داده شده است. در ابتدا، مقدار loss و معیار MAE در هر دو مجموعه آموزش و اعتبارسنجی بسیار بالا بوده و به ترتیب در epoch اول به حدود ۵۳۴۳ و ۵۷.۴ و در مجموعه اعتبارسنجی به ۴۴۷۳ و ۵۳.۹ می‌رسد. با گذشت دوره‌ها، مدل به تدریج بهبود یافته و مقادیر loss و MAE کاهش قابل توجهی پیدا کرده‌اند؛ به‌طوری که در اواسط آموزش مقدار loss به زیر ۱۱۰۰ و MAE به حدود ۲۲ در مجموعه آموزش و حدود ۲۳ تا ۲۹ در اعتبارسنجی کاهش یافته است. هرچند روند کاهش خطا نسبتاً یکنواخت بود، اما در برخی دوره‌ها نوساناتی در مقادیر loss و MAE مجموعه اعتبارسنجی مشاهده می‌شود که ممکن است ناشی از نویز داده‌ها یا پیچیدگی مدل باشد؛ به‌عنوان مثال در برخی epoch ها مانند ۷، ۱۱، ۱۵ و ۲۱، مقادیر val_loss افزایش پیدا کرده‌اند. با این حال، در کل مدل توانسته عملکرد خود را بهبود دهد و در نهایت به دلیل استفاده از Early Stopping، آموزش پس از ۳۰ epoch متوقف شده است. این توقف به منظور جلوگیری از overfitting و حفظ تعادل بین دقت مدل روی داده‌های آموزش و اعتبارسنجی انجام شده است.

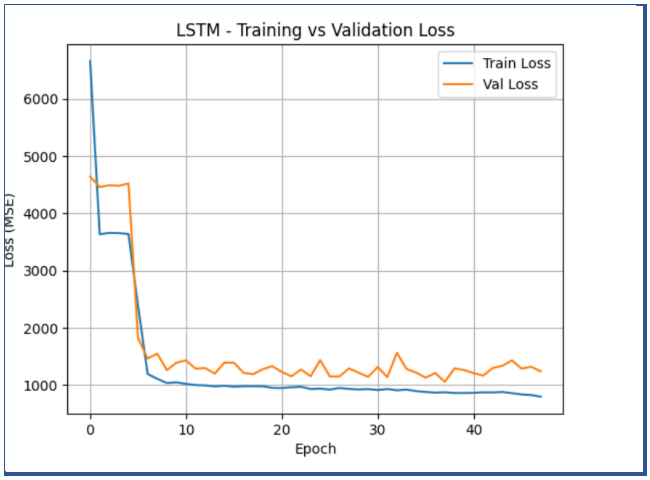
۸. ارزیابی عملکرد مدل‌ها

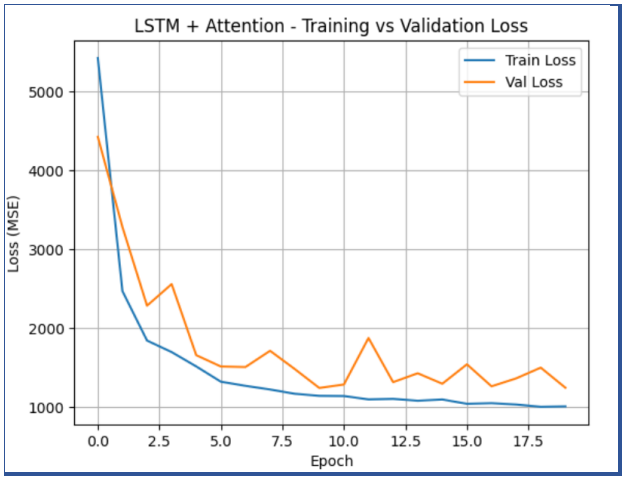
نمودارها و مقایسهٔ بصری

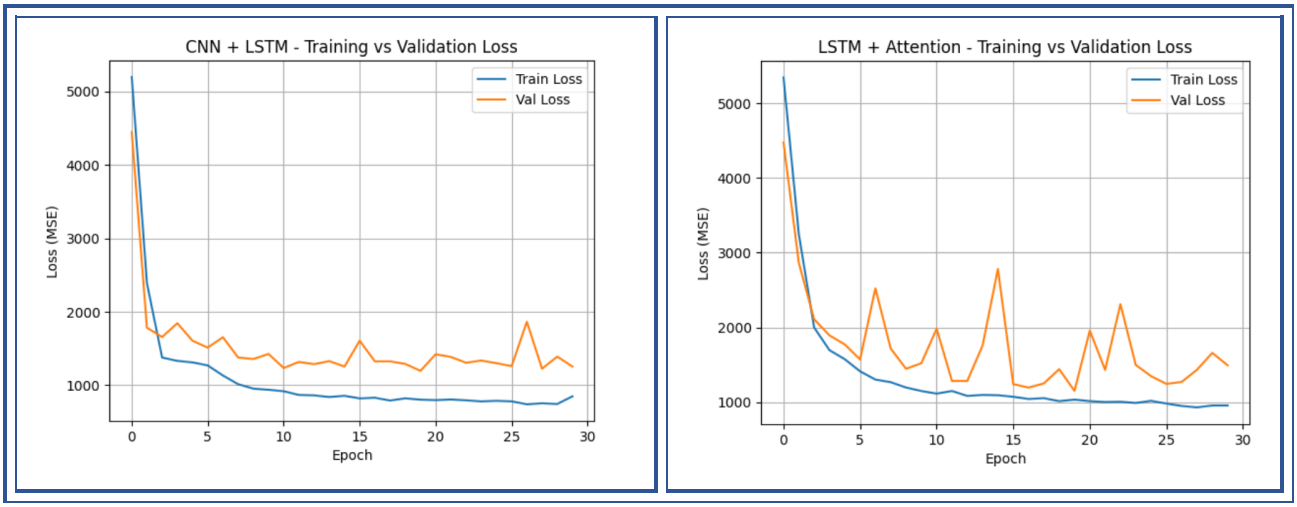
برای ارزیابی روند یادگیری مدل‌ها، منحنی‌های خطای آموزش و اعتبارسنجی (برحسب معیار MSE) برای هر مدل رسم شد. این نمودارها نشان می‌دهند که مدل‌ها در طی چه تعداد epoch به پایداری در آموزش رسیده‌اند و آیا نشانه‌هایی از بیش‌برازش در آن‌ها دیده می‌شود یا خیر.

```
def plot_loss(history, model_name):
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Val Loss')
plt.title(f'{model_name} - Training vs Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss (MSE)')
plt.legend()
plt.grid(True)
plt.show()
plot_loss(history_cnn, "CNN")
plot_loss(history_lstm, "LSTM")
plot_loss(history_cnn_lstm, "CNN + LSTM")
plot_loss(history_lstm_cnn, "LSTM + CNN")
plot_loss(history_lstm_att, "LSTM + Attention")
```







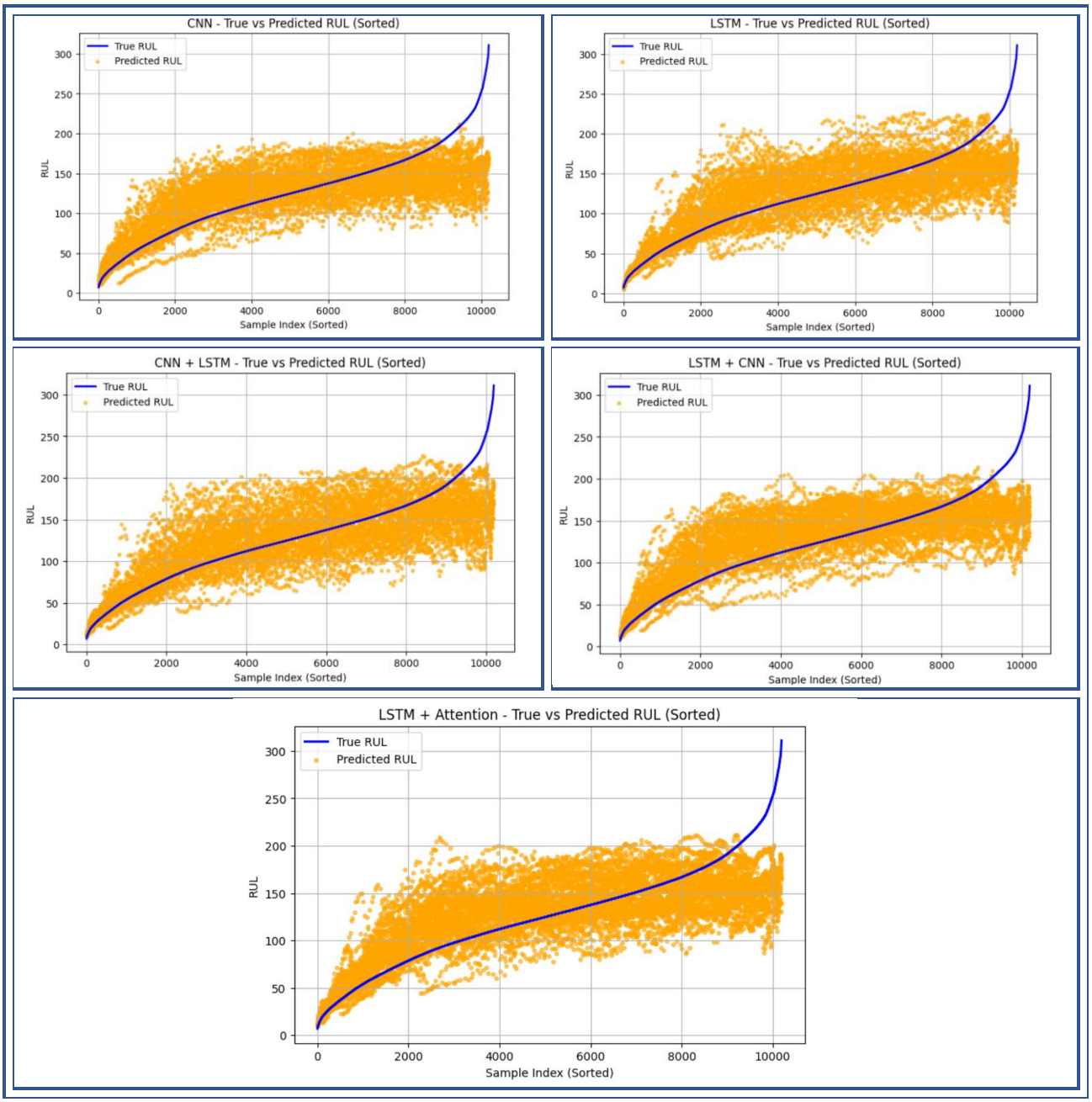


تصویر ۲، روند یادگیری مدل‌ها

همچنین، برای بررسی کیفیت پیش‌بینی، نمودار مقایسه‌ای بین مقادیر واقعی RUL و مقادیر پیش‌بینی‌شده توسط هر مدل روی داده‌های تست ترسیم شد. این نمودار به‌صورت نقطه‌ای برای هر نمونه از داده تست رسم شد تا انحراف پیش‌بینی‌ها از مقدار واقعی قابل مشاهده باشد.

کد ۱۱

```
def plot_pred_vs_true(y_true, y_pred, model_name):
    sorted_indices = np.argsort(y_true)
    y_true_sorted = np.array(y_true)[sorted_indices]
    y_pred_sorted = np.array(y_pred)[sorted_indices]
    plt.figure(figsize=(8,5))
    plt.plot(y_true_sorted, label='True RUL', color='blue', linewidth=2)
    plt.scatter(range(len(y_pred_sorted)), y_pred_sorted, label='Predicted RUL', color='orange', alpha=0.6, s=10)
    plt.title(f'{model_name} - True vs Predicted RUL (Sorted)')
    plt.xlabel('Sample Index (Sorted)')
    plt.ylabel('RUL')
    plt.legend()
    plt.grid(True)
    plt.show()
plot_pred_vs_true(y_test, y_pred_cnn, "CNN")
plot_pred_vs_true(y_test, y_pred_lstm, "LSTM")
plot_pred_vs_true(y_test, y_pred_cnn_lstm, "CNN + LSTM")
plot_pred_vs_true(y_test, y_pred_lstm_cnn, "LSTM + CNN")
plot_pred_vs_true(y_test, y_pred_att, "LSTM + Attention")
```



تصویر ۳، مقادیر پیش‌بینی شده هر مدل

معیارهای ارزیابی مدل

برای ارزیابی عددی عملکرد مدل‌های یادگیری، از مجموعه‌ای از معیارهای کمی استفاده شد RMSE به عنوان شاخصی برای اندازه‌گیری بزرگی خطاها، خصوصاً در برابر مقادیر پرت، در نظر گرفته شد. برای تحلیل دقت عمومی مدل بدون توجه به جهت خطا، از MAE استفاده گردید.

همچنین، ضریب تعیین R^2 برای نشان دادن میزان تطابق پیش‌بینی‌های مدل با داده‌های واقعی به کار رفت؛ مقداری نزدیک به ۱ نشان‌دهنده قدرت پیش‌بینی بالا است. معیار میانگین درصد خطای مطلق (MAPE) نیز برای سنجش نسبی دقت مدل نسبت به مقادیر واقعی لحاظ شد. علاوه بر این، log-RMSE به منظور کاهش تأثیر مقادیر بسیار بزرگ و تحلیل بهتر توزیع خطاها محاسبه گردید. در نهایت، زمان آموزش (Training Time) نیز به عنوان شاخصی از کارایی زمانی مدل‌ها برای مقایسه عملکرد اجرایی مورد ارزیابی قرار گرفت. نتایج این ارزیابی‌ها به تفصیل در جدول زیر آورده شده است.

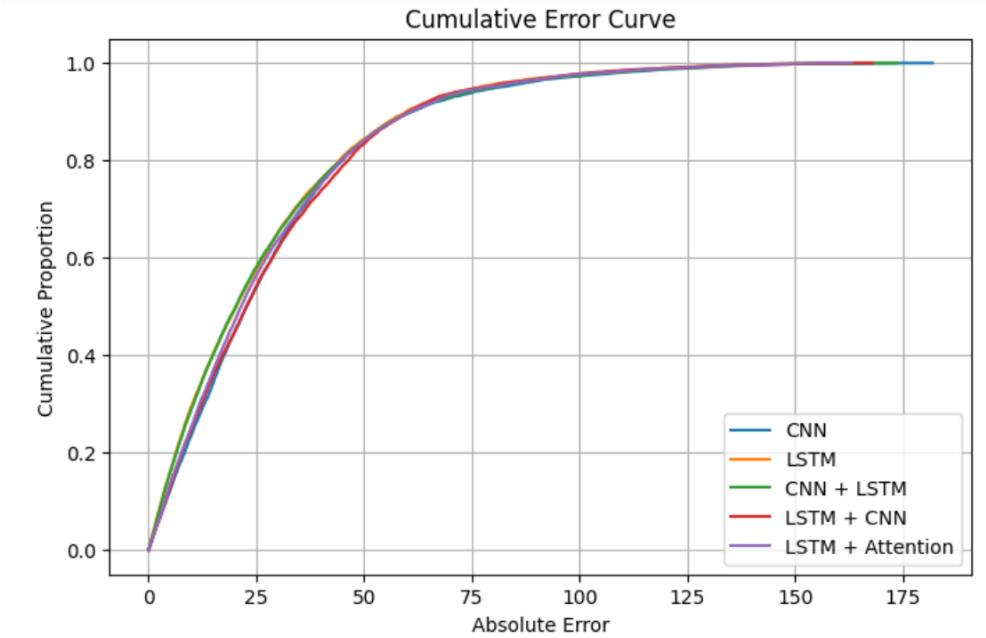
کد ۱۲

```
def evaluate_model(y_true, y_pred, training_time):
    rmse = np.sqrt(mean_squared_error(y_true, y_pred))
    mae = mean_absolute_error(y_true, y_pred)
    r2 = r2_score(y_true, y_pred)
    mape = np.mean(np.abs((y_true - y_pred) / (y_true + 1e-8))) * 100
    log_rmse = np.sqrt(mean_squared_error(np.log1p(y_true), np.log1p(y_pred)))  # log1p = log(1 + x) for stability
    return {
        'RMSE': rmse,
        'MAE': mae,
        'R²': r2,
        'Training Time (s)': training_time,
        'MAPE (%)': mape,
        'log-RMSE': log_rmse }
```

جدول3 ، ارزیابی مدل‌ها

مدل	RMSE	MAE	R^2	زمان آموزش (ثانیه)	MAPE (%)	log-RMSE
CNN	38.93	28.95	0.491	48.67	24.15	0.288
LSTM	36.81	27.31	0.545	132.17	23.02	0.271
CNN +LSTM	37.50	27.52	0.528	45.30	22.51	0.272
LSTM + CNN	38.38	29.47	0.506	87.41	26.04	0.291
LSTM + Attention	38.58	28.92	0.501	85.42	24.00	0.280

در تحلیل عملکرد مدل‌ها، یکی از روش‌های بصری مفید، استفاده از نمودار تجمعی خطا (Cumulative Error Curve) است. این نمودار نمایی کلی از توزیع خطاهای مطلق مدل در پیش‌بینی‌ها ارائه می‌دهد. در این روش، خطاهای مطلق بین مقادیر واقعی و پیش‌بینی‌شده محاسبه و سپس به‌صورت صعودی مرتب می‌شوند. در ادامه، نسبت تجمعی نمونه‌هایی که خطای آن‌ها کمتر از یک مقدار مشخص است، ترسیم می‌گردد.در نمودار حاصل، مدل‌هایی که منحنی آن‌ها سریع‌تر به سمت بالا رشد می‌کند (یعنی در مقادیر خطای کمتر، درصد بیشتری از نمونه‌ها قرار دارد)، عملکرد بهتری از خود نشان می‌دهند. به‌عبارت‌دیگر، چنین مدل‌هایی توانسته‌اند برای اکثر نمونه‌ها پیش‌بینی‌هایی با خطای پایین داشته باشند.



تصویر ۴، نمودار تجمعی خطا برای همه مدل‌ها

در نمودار ارائه‌شده، مدل‌های مختلف از جمله CNN، LSTM، ترکیب‌های CNN + LSTM وLSTM + CNN و همچنین مدل LSTM به همراه مکانیزم توجه (Attention) با یکدیگر مقایسه شده‌اند. مشاهده می‌شود که تمامی مدل‌ها رفتار نسبتاً مشابهی دارند، اما تفاوت‌های ظریفی در دقت آن‌ها نمایان است. به طور مشخص:

- مدل LSTM + Attention، در ناحیه ابتدایی منحنی، درصد بیشتری از نمونه‌ها را با خطای پایین پیش‌بینی کرده که نشان‌دهنده عملکرد دقیق‌تر آن برای داده‌های کم‌خطا است.

- مدل‌های ترکیبی CNN + LSTM و LSTM + CNN نیز نسبت به مدل‌های پایه (CNN یا LSTM تنها) عملکرد بهتری داشته‌اند و منحنی آن‌ها سریع‌تر به ۱ نزدیک شده است.
- مدل CNN به‌طور کلی کمترین درصد خطای بالا را نشان داده و از نظر دقت در بازه‌های گسترده، عملکرد مناسبی دارد.
- مدل LSTM با اینکه منحنی نزدیکی به سایر مدل‌ها دارد، اما در برخی نقاط دقت کمتری را نشان داده است.

۹. تحلیل عملکرد مدل‌ها

دراین بخش برای تحلیل کل تمرین، به سوالات طرح شده، پاسخ داده می‌شود.

کدام مدل دقیق‌ترین پیش‌بینی را ارائه داد؟

با بررسی معیارهای ارزیابی عددی، مشخص شد که مدل LSTM دقیق‌ترین پیش‌بینی را ارائه داده است. این مدل کمترین مقدار RMSE (36.81) و MAE (27.31) را داشته و همچنین بالاترین ضریب تعیین R^2 برابر با ۰.۵۴۵ را ثبت کرده است. دلیل این عملکرد بهتر احتمالاً به معماری LSTM مربوط می‌شود که برای مدل‌سازی وابستگی‌های زمانی در داده‌ها مناسب‌تر است و می‌تواند توالی‌های زمانی طولانی‌مدت را بهتر یاد بگیرد. از سوی دیگر، مدل‌های ترکیبی مانند CNN + LSTMنیز عملکرد بسیار نزدیکی به LSTM داشته‌اند و از نظر MAPE نیز بهترین مقدار (۲۲.۵۱٪) را ثبت کرده‌اند، که نشان‌دهنده دقت نسبی خوب آن‌ها است.

آیا هیچ یک از مدل‌ها نشانه‌هایی از بیش‌برازش دارد؟

بررسی نمودارهای خطای آموزش و اعتبارسنجی نشان می‌دهد که مدل‌های مختلف رفتارهای متفاوتی از خود نشان داده‌اند. در مدل CNN خطای آموزش و اعتبارسنجی هر دو به مرور کاهش می‌یابند، اما خطای اعتبارسنجی در مقایسه با خطای آموزش بالاتر باقی می‌ماند که نشانه‌ای از بیش‌برازش است زیرا خطای اعتبارسنجی همچنان روند کاهشی دارد. در مقابل، مدل LSTM عملکرد متعادلی از خود نشان داده است، به طوری که هر دو منحنی خطا به صورت پایدار کاهش یافته و در نهایت همگرا شده‌اند که نشان‌دهنده آموزش مناسب این مدل است. مدل ترکیبی CNN+LSTM در هر

دو حالت، در ابتدا روند مطلوبی داشتند، اما پس از epoch دهم تا پانزدهم، اختلاف بین خطای آموزش و اعتبارسنجی افزایش یافت که این امر نشانه‌ای از شروع بیش‌برازش می‌باشد. در مدل LSTM+ATTENTION، نشانه‌های واضحی از بیش‌برازش شدید دیده می‌شود. در این مدل، خطای اعتبارسنجی پس از کاهش اولیه، افزایش می‌یابد یا ثابت می‌ماند، در حالی که خطای آموزش همچنان کاهش می‌یابد که نشان‌دهنده عدم توانایی مدل در تعمیم‌دهی به داده‌های جدید است.

تعادل بین دقت و زمان

این تحلیل‌ها نشان می‌دهد که انتخاب مدل بهینه باید با در نظر گرفتن معیارهای چندگانه از جمله دقت، زمان آموزش و توانایی تعمیم‌دهی انجام شود و صرفاً تمرکز بر یک معیار خاص می‌تواند منجر به نتایج زیربهینه شود. از نظر تعادل بین دقت و زمان آموزش، مدل LSTM اگرچه دقیق‌ترین پیش‌بینی را ارائه داد، اما بیشترین زمان آموزش (۱۳۲.۱۷ ثانیه) را نیز به خود اختصاص داده است. در مقابل، مدل CNN + LSTM با زمان آموزش کمتر (۴۵.۳۰ ثانیه) دقتی نسبتاً نزدیک به LSTM داشته و گزینه مناسبی محسوب می‌شود. مدل CNN نیز با کمترین زمان آموزش (۴۸.۶۷ ثانیه) عملکردی قابل قبول داشته و از نظر بهره‌وری محاسباتی انتخاب مناسبی است. این مقایسه نشان می‌دهد که انتخاب مدل نه‌تنها باید بر اساس دقت، بلکه با توجه به محدودیت‌های محاسباتی و زمان اجرا صورت گیرد. در نمودار تجمعی خطا نیز دیده شد که مدل LSTM + Attention در خطاهای پایین عملکرد بهتری دارد، اما در کلیت، اختلاف عملکرد مدل‌ها چندان زیاد نیست و بسته به نیاز مسئله، می‌توان تعادل مناسبی بین دقت و زمان یافت.