



[Category]

مدلسازی توان مصرفی موتور BLDC

بهار 1404

فهرست مطالب

- 1 فصل 1: مقدمه و کلیات پروژه
- 2 فصل 2: گام اول
- 13 فصل 3: گام دوم
- 26 فصل 4: گام سوم

فهرست شکل‌ها

فصل 1: مقدمه و کلیات پروژه

در این پروژه، قصد داریم به طراحی مدل هوش مصنوعی بپردازیم، که با دریافت چند پارامتر موتور BLDC، قادر به پیش‌بینی و تخمین زدن یک یا چند پارامتر دیگر آن، با هدف بررسی و محاسبه توان مصرفی آن باشد.

بصورت کلی در لاگ های ثبت شده از موتور BLDC، پارامترهای زیر قابل مشاهده هستند:

- فرمان موتور (Motor-CMD)
- عمق (Depth)
- دور موتور (Motor-FB)
- جریان موتور (Power-Prop I)
- ولتاژ پک باتری موتور (Power-Prop V)

در هر گام از پروژه شمای متفاوت، همراه با ورودی و خروجی های متفاوت را در مدلسازی هوش مصنوعی خود دنبال خواهیم کرد که به تبع، با توجه به شرایط کلی آن گام، از مدل متفاوت، ساختار های متفاوت، پارامترهای متفاوت و.. استفاده خواهیم کرد.

زبان برنامه نویسی مورد استفاده در این پروژه، زبان پرکاربرد حوزه علوم داده و هوش مصنوعی، یعنی پایتون، و کتابخانه های مرتبط با هوش مصنوعی، شبکه عصبی، تغییرات داده ها و مصورسازی داده ها آن هست که همگی در بستر پلتفرم Jupyter Notebook پیاده سازی می‌شوند. در انتهای هر گام نیز تمامی فایل های اصلی مانند مدل تمرین داده شده، نمودار ها، و همچنین فایل های جانبی همچون مقیاس ساز داده ها ذخیره شده و در فایل ارسالی ضمیمه خواهد شد.

در هر گام از پروژه، بصورت کلی مراحل مشابهی طی خواهند شد:

- مرحله اول: مرتب و آماده سازی داده ها
- مرحله دوم: انتخاب و تمرین دادن مدل هوش مصنوعی
- مرحله سوم: ارزیابی دقیق تخمین زدن
- مرحله چهارم: مصورسازی عملکرد مدل در نمودار

لازم به ذکر است در هر گام، با توجه به شرایط موجود، موارد ذکر شده میتوانند شامل تغییراتی بشوند.

فصل 2: گام اول

در این گام اول از پروژه هدف ما طراحی مدل هوش مصنوعی است که بتواند با دریافت ورودی های دور موتور و عمق موتور، جریان موتور BLDC را در خروجی پیش‌بینی کند. در شکل زیر شما کلی مدل هدف را مشاهده می‌کنیم:



ورودی ها:

- دور موتور (Motor-FB)
- عمق (Depth)

خروجی:

- جریان موتور (Power-Prop I)

نوع مدلسازی هوش مصنوعی:

- رگرسیون (در بررسی ها متوجه مناسب نبودن آن می‌شویم)
- شبکه عصبی

مشخصات شبکه عصبی:

- ساختار : 2 – 128 – 64 – 32 – 1
- نوع : Sequential (ترتیبی)
- بهینه ساز : Adam
- تابع فعالساز : ReLU
- تابع زیان : میانگین مربعات خطأ (Mean Squared Error)

در ادامه به توضیح بخش به بخش پیاده سازی شده کدهای پروژه در نوت بوک، همراه نمایش نمودارها می‌پردازیم. لازم به ذکر است در روند گزارش، توضیحات مختصراً درباره توابع و پارامترهای استفاده شده و نحوه عملکرد آنها مانند تابع فعالساز، تابع زیان، معیار R² score و خواهیم پرداخت.

تمیز کردن و مرتب سازی داده‌ها :

در این مرحله پروژه جداگانه نوت بوک را به منظور تمیز و مرتب سازی داده‌های خام دریافت شده ایجاد میکنیم. سپس با وارد کردن دیتاست های CSV، و انجام اقدامات لازم در چند مرحله داده‌ها را تمیز، و ادغام کرده و آماده استفاده برای تمرین دادن مدل میکنیم.

اضافه کردن کتابخانه برای تمیز کردن داده‌ها

```
[100]: import pandas as pd
```

وارد کردن فایل‌های دیتاست

```
[102]: file1 = pd.read_csv('D:/AIjourney/DataSets/Roshd Center/Log1.csv')
file2 = pd.read_csv('D:/AIjourney/DataSets/Roshd Center/Log2.csv')
file3 = pd.read_csv('D:/AIjourney/DataSets/Roshd Center/Log3.csv')
```

در این مرحله از پروژه، تنها از کتابخانه‌ی پرکاربرد برای تغییرات مجموعه داده‌ها، یعنی پانداس (pandas) استفاده می‌کنیم. این کتابخانه بصورت کلی با هدف تغییر و تنظیم داده‌ها و همچنین تسهیل تغییر فرمت آنها استفاده گسترده‌ای دارد. در این بخش از کد هر سه فایل ذخیره شده دیتاست را به پروژه وارد میکنیم.

تعریفتابع برای تکرار داده جریان و ولتاژ موتور

```
[104]: def fill_power_prop_values(df):
    current_power_v = None
    current_power_i = None

    for index, row in df.iterrows():
        if not pd.isnull(row['Power-Prop V']) and not pd.isnull(row['Power-Prop I']):
            current_power_v = row['Power-Prop V']
            current_power_i = row['Power-Prop I']
        else:
            df.at[index, 'Power-Prop V'] = current_power_v
            df.at[index, 'Power-Prop I'] = current_power_i

    return df

[117]: file1 = fill_power_prop_values(file1)
file2 = fill_power_prop_values(file2)
file3 = fill_power_prop_values(file3)

[121]: # تست کردن
#file1.iloc[4000:4020]
```

هدف ما از این بخش اصلاح ناهماهنگی‌ها و برطرف کردن خلاه‌های (missing info) موجود در دیتاست میباشد. در قسمت اول تابعی را به منظور تکرار داده‌های جریان موتور (power-prop) و جریان موتور (power-prop V) در سطرهایی ماقبل که مقادیر آنها ثبت نشده است تعریف میکنیم و سپس هر سه فایل را در قالب دیتافریم کتابخانه پانداس و با صدا زدن تابع، آپدیت یا بروزرسانی میکنیم.

تعريف تابع برای حذف کردن داده های صفر یا عدم وجود برای عمق و فیدبک موتور

```
[107]: def clean_motor_and_depth(df):
    df = df.dropna(subset=['Motor-FB', 'Depth1'])
    df = df[(df['Motor-FB'] != 0) & (df['Depth1'] != 0)]
    return df

[123]: file1 = clean_motor_and_depth(file1)
file2 = clean_motor_and_depth(file2)
file3 = clean_motor_and_depth(file3)

[127]: # تست کردن
#file1.iloc[4000:4020]
```

در قسمت دوم تابعی دیگر به منظور حذف مقادیر صفر، یا مقادیر غیر عددی تهی (NaN) برای پارامتر دور موتور و عمق موتور استفاده میکنیم و مانند قسمت قبلی هر سه دیتافریم را با آن دست خوش تغییرات میکنیم. در این تابع خطوطی که مقادیر صفر یا تهی داشته باشند از دیتاست حذف میشوند.

تعريف تابع برای حذف کردن داده های صفر یا عدم وجود برای پاور پراپ ها

```
[130]: def clean_power_prop(df):
    df = df[(df['Power-Prop I'] != 0) & (df['Power-Prop V'] != 0)]
    return df

[132]: file1 = clean_power_prop(file1)
file2 = clean_power_prop(file2)
file3 = clean_power_prop(file3)

[142]: # تست کردن
#file1.iloc[5000:5020]
# به علت حذف سطرهای رکوردهای ناقص، تعداد کل سطرهای هر فایل کاسنه میشود
```

در قسمت سوم نیز تابعی مشابه قسمت دوم، و به منظور حذف مقادیر تهی یا صفر برای پارامتر جریان و ولتاژ موتور اجرا میکنیم و هر سه دیتافریم از داده ها را با آن بروزرسانی میکنیم.

ادغام کردن دیتاست های تمیز شده در یک فایل اکسل برای راحتی مدلسازی در قسمت دوم پروژه

```
[145]: file_cleaned = pd.concat([file1, file2, file3])

[153]: # تست کردن
#file_cleaned.shape
#file_cleaned.head(20)

[155]: import os

output_directory = 'D:/AIjourney/DataSets/Roshd Center'
file_name = 'cleaned_dataset.csv'

file_cleaned.to_csv(os.path.join(output_directory, file_name), index=False)

[159]: print("فایل دیتاست تمیز و ادغام شده در")
print(output_directory)
print("نخسته شد")
print("نخسته شد")
فایل دیتاست تمیز و ادغام شده در
D:/AIjourney/DataSets/Roshd Center
نخسته شد
```

در قسمت آخر از فاز اول پروژه نیز با استفاده از تابع `concat` از کتابخانه `pandas`، هر سه دیتافریم تمیز و مرتب سازی شده را ادغام کرده، و در قالب یک دیتافریم قرار میدهیم. سپس با وارد کردن کتابخانه سیستم عامل (OS) و با کمک از توابع این کتابخانه به همراه تابع `to_csv` از کتابخانه

پانداس، دیتافریم را در قالب یک فایل اکسل CSV در مسیر موردنظر ذخیره سازی میکنیم تا در فاز بعدی با استفاده از این فایل مدل خود را ساخته و تمرین بدهیم.

در تصویر زیر نمونه داده های دیتابست، قبل و بعد از تمیز و مرتب سازی را مشاهده میکنیم:

قبل از مرتب سازی

377	31:44.6	0	0	35		
378	31:44.7	0	0	36		
379	31:44.8	0	0	36		
380	31:44.9	0	0	36		
381	31:45.0	0	0	35		
382	31:45.1	0	0	34		
383	31:45.2	0	0	33		
384	31:45.3	0	0	33		
385	31:45.4	0	0	33		
386	31:45.5	744	0	33		
387	31:45.6	744	0	34	77.5	0.1
388	31:45.7	744	0	35		
389	31:45.8	744	0	35		
390	31:45.9	744	0	36		
391	31:46.0	744	0	38		
392	31:46.1	744	0	39		
393	31:46.2	744	0	39		
394	31:46.3	744	0	39		
395	31:46.4	744	57	38		
396	31:46.5	744	87	37		
397	31:46.6	744	128	36		
398	31:46.7	744	143	36	77.5	0.8
399	31:46.8	744	155			
400	31:46.9	744	158	35		
401	31:47.0	744	162	35		
402	31:47.1	744	175	36		

بعد از مرتب سازی

72	29:42.4	744	602	34	77	10.1
73	29:42.5	744	595	36	77.1	8.3
74	29:42.6	744	590	36	77.1	8.3
75	29:42.8	744	579	35	77.1	8.3
76	29:42.9	744	572	36	77.1	8.3
77	29:43.1	744	564	35	77.1	8.3
78	29:43.2	744	552	35	77.1	8.3
79	29:43.3	744	541	34	77.1	8.3
80	29:43.4	744	533	35	77.1	8.3
81	29:43.5	744	558	36	77	10.7
82	29:43.6	744	636	38	77	10.7
83	29:43.7	744	664	38	77	10.7
84	29:43.9	744	666	38	77	10.7
85	29:44.0	744	652	38	77	10.7
86	29:44.1	744	645	38	77	10.7
87	29:44.2	744	652	34	77	10.7
88	29:44.3	744	624	32	77	10.7
89	29:44.4	744	624	32	77	10.7
90	29:44.5	744	648	33	77.1	9.2
91	29:44.6	744	665	36	77.1	9.2

ساخت و تمرین مدل هوش مصنوعی :

در این مرحله ما از دو مدل هوش مصنوعی رگرسیون و شبکه های عصبی برای مدلسازی استفاده میکنیم و در انتهای نیز نتایج آنها را با یکدیگر مقایسه میکنیم:

- مدلسازی با روش رگرسیون خطی چندگانه
- مدلسازی با شبکه های عصبی

تصویر کلی در مسائلی از حوزه یادگیری ماشین، که هدف ما پیش‌بینی مقادیر خروجی (پیوسته) است، از دسته روش های رگرسیون، شامل رگرسیون خطی، رگرسیون غیرخطی، رگرسیون چندگانه و... استفاده میکنیم، که این روش های هم در قالب مدل های سنتی یادگیری ماشین (مدل رگرسیون خطی

چندگانه) و هم در قالب شبکه عصبی قابل پیاده سازی است که در ادامه با بررسی، روش مدلسازی مناسب را انتخاب میکنیم.

مدلسازی با رگرسیون خطی چندگانه :

اضافه کردن کتابخانه های ضروری

```
[5]: import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt
```

در ابتدا کتابخانه های موردنیاز را به پروژه وارد(import) میکنیم.

نشان دادن شما کلی داده ها در نمودار

```
[7]: df = pd.read_csv('D:/AIjourney/DataSets/Roshd Center/cleaned_dataset.csv')
#df.head(10)

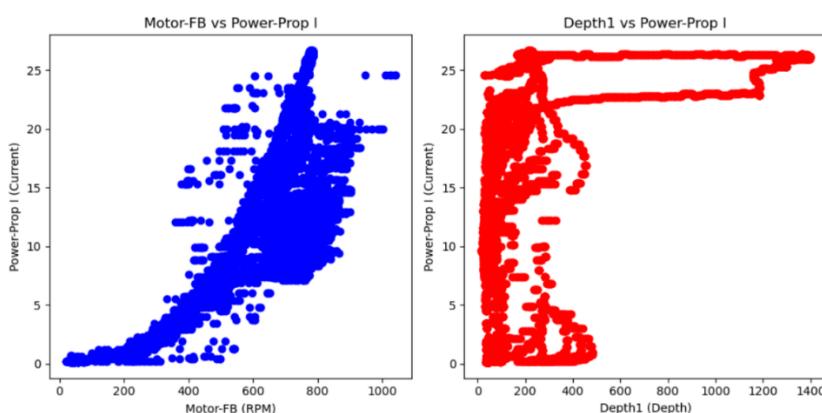
[8]: plt.figure(figsize=(10,5))

# داده ها با اینکس دور مهور
plt.subplot(1, 2, 1)
plt.scatter(df['Motor-FB'], df['Power-Prop I'], c='blue')
plt.xlabel('Motor-FB (RPM)')
plt.ylabel('Power-Prop I (Current)')
plt.title('Motor-FB vs Power-Prop I')

# داده ها با اینکس عمق مهور
plt.subplot(1, 2, 2)
plt.scatter(df['Depth1'], df['Power-Prop I'], c='red')
plt.xlabel('Depth1 (Depth)')
plt.ylabel('Power-Prop I (Current)')
plt.title('Depth1 vs Power-Prop I')

plt.tight_layout()
plt.show()
```

مانند فاز اول دیتاست تمیز شده را در قالب دیتا فریم کتابخانه پانداس در یک متغیر ذخیره سازی میکنیم. سپس با استفاده از کتابخانه Matplotlib، که کاربرد آن مصور سازی داده های مختلف، با تولید نمودار ها است، نمودار پارامترهای دور موتور- جریان و عمق موتور- جریان) را به نمایش می گذاریم تا بتوانیم با بررسی بهتر و دقیق تر، مدل مناسب جهت تمرین دادن را انتخاب کنیم.



همانطور که در تصویر ملموس است، در نمودار دور موتور- جریان(سمت چپ) رابطه ای نسبتاً خطی و مانند تابع exponential (افزایش لگاریتمی) قابل مشاهده است. اما در نمودار عمق موتور- جریان(سمت راست)، چنین چیزی نبوده و رابطه بین این دو متغیر غیرخطی و پیچیده است!

لذا با درنظر گرفتن هدف پروژه که مدلسازی با استفاده از داده عمق و دور موتور، و پیش‌بینی داده جریان موتور می‌باشد، میتوان انتظار داشت با روش شبکه‌های عصبی (به علت درک بهتر روابط غیرخطی و پیچیده) به مدلی با دقت تخمین بالاتر برسیم، اما در این پروژه ما هر دو روش رگرسیون و شبکه‌های عصبی را برای مدلسازی استفاده کرده و در انتهای مقایسه ای بر دقت آن دو خواهیم داشت.

مدلسازی با رگرسیون

قرار دادن داده‌های ایکس و واي

```
[12]: X = df[['Motor-FB', 'Depth1']]
y = df['Power-Prop I']
```

بخش بندی داده‌های تمرین و تست

```
[14]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

#X_train.shape
#X_test.shape
```

ابتدا داده‌های ایکس(ورودی) و داده‌های خروجی(ایگرگ) را با استفاده از دو دیتافریم جدا‌سازی می‌کنیم. سپس با استفاده از تابع train_test_split از کتابخانه Sklearn(Scikit-Learn) داده‌ها را به دو بخش داده تمرین و داده تست، با نرخ 80 به 20 تقسیم بندی می‌کنیم.

کتابخانه Sklearn کاربردهای بسیاری از جمله توابع کمک کننده در مدیریت داده‌ها، روش‌های مدلسازی یادگیری ماشین، مقیاس سازها، معیارهای ارزیابی دقت مدل و... را دارد.

ایجاد شی مدل رگرسیون و تمرین داده با داده‌های جدا شده تمرین

```
[16]: reg_model = LinearRegression()
reg_model.fit(X_train, y_train)
```

```
[16]: ▾ LinearRegression ⓘ ⓘ
LinearRegression()
```

پیش‌بینی جریان با استفاده از مدل تمرین داده شده و داده‌های تست

```
[18]: reg_predictions = reg_model.predict(X_test)
```

در مرحله بعدی، با استفاده از همین کتابخانه، شی(object) مدل رگرسیون خطی را ذخیره کرده و آن را با داده‌های تمرین(data) خود تمرین میدهیم. سپس با استفاده از مدل تمرین داده شده و ورودی داده‌های تست، به همان تعداد داده‌های جریان موتور را پیش‌بینی می‌کنیم و در متغیر ذخیره سازی می‌کنیم.

ارزیابی نتایج پیش‌بینی شده توسط مدل و داده‌های ایگرگ تست برای بررسی دقیق مدل

```
[20]: from sklearn.metrics import r2_score
print('دقیق مدل تمرین داده شده با روش رگرسیون :')
print(r2_score(y_train, reg_predictions))
print('نژدیک تر به عدد 1 نشان بیند دقت مدل است')
print('دقیق مدل تمرین داده شده با روش رگرسیون :')
print(0.5991130665533928)
print('نژدیک تر به عدد 1 نشان بیند دقت مدل است')
```

در مرحله آخر از رگرسیون، با وارد کردن R2 score از کتابخانه Sklearn به ارزیابی مدل می‌پردازیم، که این امر را با مقایسه داده‌های پیش‌بینی شده توسط مدل ما، برای جریان، با داده‌های اختصاص داده شده به تست یا همان `y_test` برای جریان، صورت می‌پذیرد.

همانطور که قابل مشاهده است، مقدار R2 score عدد 0.59 را بصورت تقریبی نشان میدهد که نشان از عملکرد ضعیف مدل تمرین داده شده است و به بیانی دیگر، مدل تمرین داده شده نتوانست به خوبی رابطه بین پارامترها را شبیه سازی کرده و با ورودی‌های جدید، مقدار دقیقی از جریان موتور را نشان بدهد.

توضیحات تکمیلی، نحوه عملکرد نرخ R2 score:

ضریب تعیین یا همان R2 score یکی از معیارهای ارزیابی مدل‌های رگرسیونی است که نشان می‌دهد مدل چه مقدار از واریانس داده‌های وابسته را توضیح می‌دهد. اینکار را با محاسبه اختلاف مقدار واقعی و مقدار پیش‌بینی شده و به توان دو رساندن آن، به ازای تمامی سطرها و تقسیم کردن آن بر مجموع مربعات کل به ازای تمامی سطرها انجام میدهد. تصویر زیر فرمول این نرخ را نشان میدهد.

$$R^2 = 1 - \frac{\sum_{i=1}^n (Y_i - \hat{Y}_i)^2}{\sum_{i=1}^n (Y_i - \bar{Y}_i)^2}$$

تصویر عادی این نرخ مقداری بین 0 و 1 خواهد داشت که هرچقدر به یک نزدیکتر باشد، به معنای بالاتر بودن دقت پیش‌بینی و توضیح دادن اکثر تغییرات موجود در داده‌ها است و هرچقدر نزدیک به صفر و یا حتی در شرایطی منفی باشد، به معنای پایین بودن نسبی دقت پیش‌بینی و قادر نبودن مدل در ارائه اطلاعات تغییرات موجود در داده‌ها، نسبت به میانگین است.

مدلسازی با شبکه های عصبی :

اضافه کردن کتابخانه های موردنیاز مدلسازی با شبکه های عصبی

```
[39]: import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Input
```

در ادامه پروژه به مدلسازی با شبکه های عصبی میپردازیم. ابتدا کتابخانه قدرتمند مختص هوش مصنوعی به نام تنسورفلو (TensorFlow) همراه با توابع موردنیاز آن را وارد پروژه نوت بوک میکنیم. این کتابخانه یکی از پرکاربردترین کتابخانه های زبان برنامه نویسی پایتون است که با هدف ساختن شبکه های عصبی، و تمرین دادن آنها طراحی شده است و امکانات بسیار کاملی از جمله تعیین ساختار شبکه، نوع نورون ها، توابع کمکی و تکمیلی، تنظیم کننده نرخ یادگیری و.. را دارد.

ساخت اولیه شبکه عصبی

```
[41]: model = Sequential()
model.add(Input(shape=(1,)))
model.add(Dense(128, activation='relu'))
model.add(Dense(64, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(1))

[43]: model.compile(optimizer='adam', loss='mean_squared_error')
```

در ادامه تنظیمات اولیه از جمله تعیین تابع فعالسازی، و راه اندازی شبکه عصبی را انجام داده و آن را با تنظیمات adam برای optimizer و mean squared error برای loss کامپایل میکنیم.

توضیحات توابع و پارامترهای استفاده شده در ساخت شبکه عصبی:

- **تابع فعالسازی ReLU :** یکی از پرکاربردترین تابع فعالسازی در شبکه های عصبی بوده که این تابع مقدارهای منفی را صفر می کند و مقدارهای مثبت را بدون تغییر عبور می دهد. کمک به حل مشکل محoscدگی گرادیان در شبکه های عصبی عمیق و اجرای سریع و محاسبات ساده از ویژگی های آن میباشد. در اصل تابع فعالساز عامل یادگیری روابط غیر خطی شبکه است!
- **بهینه ساز Adam :** یکی از محبوب ترین الگوریتم های بهینه سازی با کمک گرادیان نزولی برای شبکه های عصبی است و این الگوریتم ترکیبی از گرادیان نزولی با مومنتوم و RMSprop است و سرعت یادگیری را به طور تطبیقی برای هر پارامتر تنظیم می کند.
- **تابع زیان MSE :** یکی از رایج ترین توابع خطای در یادگیری ماشین و شبکه های عصبی است. این تابع مقدار میانگین مربعات اختلاف بین تخمین مدل از داده و مقدار واقعی داده را محاسبه می کند.

تمرين دادن شبکه عصبی با داده های تمرين و 100 دور تكرار

```
[45]: model.fit(X_train, y_train, epochs=100, batch_size=32, validation_split=0.2)
```

```
Epoch 1/100
335/335 2s 3ms/step - loss: 16.9727 - val_loss: 11.3738
Epoch 2/100
335/335 1s 2ms/step - loss: 10.3669 - val_loss: 9.7578
Epoch 3/100
335/335 1s 2ms/step - loss: 10.9726 - val_loss: 11.4984
Epoch 4/100
335/335 1s 2ms/step - loss: 10.1167 - val_loss: 12.1601
Epoch 5/100
335/335 1s 2ms/step - loss: 9.7872 - val_loss: 8.9475
Epoch 6/100
335/335 1s 2ms/step - loss: 8.6108 - val_loss: 8.4231
Epoch 7/100
335/335 1s 2ms/step - loss: 8.5009 - val_loss: 8.6715
Epoch 8/100
335/335 1s 2ms/step - loss: 8.7372 - val_loss: 7.8237
Epoch 9/100
335/335 1s 2ms/step - loss: 8.2267 - val_loss: 9.3780
```

در مرحله بعدی همانند مدل رگرسیون، شبکه عصبی ایجاد شده را با استفاده از داده های train، و تعداد تكرار 100 (epoch) تمرين ميدهيم. همانطور که قابل مشاهده است، شبکه عصبی به هر دور تكرار و انجام اعمال feedforward و backpropagation (مراحل يادگیری شبکه عصبی) به وزن ها و بایاس های بهتری ميرسد که اين پديده از روند نزولي مقدار loss يا همان خطا قابل حدس است.

پيشбинی جريان موتور با استفاده از داده های ورودی تست

```
[47]: predictions = model.predict(X_test)
105/105 0s 1ms/step
```

ارزیابی عملکرد مدل با مقایسه داده های بدست آمده و جريان های داده تست

```
[49]: r2_nn = r2_score(y_test, predictions)
print("دقت مدل با استفاده از شبکه های عصبی :")
print(r2_nn)
```

```
دقت مدل با استفاده از شبکه های عصبی :
0.9187541755472824
```

سپس به پيشбинی مقادير جريان، با استفاده از داده های ورودی تست ميپردازيم و با مقایسه آنها با مقادير خروجی تست، مدل را ارزیابی ميکنیم.

همانطور که قابل مشاهده است مقدار R2 score برای مدل تمرين داده شده شبکه عصبی، افزایش چشمگيری داشته و به مقدار قابل قبول 0.91 رسیده است که نشان از دقت نسبتا خوب اين مدل در پيشбинی جريان موتور با داده های ورودی جدید در قالب دور و عمق موتور ميباشد.

نمونه پیش‌بینی با وارد کردن دور موتور و عمق موتور ورودی

```
[65]: test_sample = np.array([[280,69]])
test_prediction = model.predict(test_sample)
print('جريان پیشنهاد شده با دور 244 و عمق 57')
print(test_prediction)
# مقدار اصلی ثبت شده جریان با این داده ها در دیتاست جریان 1.8 آمیز است
# 1/1 _____ 0s 76ms/step
# جریان پیشنهاد شده با دور 244 و عمق 57
[[2.0044398]]
```

این امر را با دادن ورودی جدید بعنوان نمونه و پیش‌بینی آن توسط مدل مشاهده می‌کنیم.

ارزیابی مدل با دیتاست ثانویه :

خواندن تست دیتا ها

```
[57]: testData1 = pd.read_csv('D:/AIjourney/DataSets/Roshd Center/cleanedTest1.csv')
testData2 = pd.read_csv('D:/AIjourney/DataSets/Roshd Center/cleanedTest2.csv')
testData3 = pd.read_csv('D:/AIjourney/DataSets/Roshd Center/cleanedTest3.csv')

لیبل زدن داده های ثانویه بصورت جداگانه

[64]: xTest1 = testData1[['Motor-FB', 'Depth1']]
yTest1 = testData1['Power-Prop 1']
xTest2 = testData2[['Motor-FB', 'Depth1']]
yTest2 = testData2['Power-Prop 1']
xTest3 = testData3[['Motor-FB', 'Depth1']]
yTest3 = testData3['Power-Prop 1']

پیش‌بینی جریان بر اساس داده های ثانویه توسط مدل شبکه عصبی

[68]: testPred1 = modelNN.predict(xTest1)
testPred2 = modelNN.predict(xTest2)
testPred3 = modelNN.predict(xTest3)

49/49 _____ 0s 5ms/step
54/54 _____ 0s 2ms/step
```

در این بخش، ابتدا فایل دیتاست های ثانویه که قبل تر آنها را همانند دیتاست اصلی مرتب سازی کرده بودیم (سه فایل جداگانه) را وارد می‌کنیم. در ادامه لیبل زدن (تقسیم بندی ورودی و خروجی ها با پانداس) و پیش‌بینی جریان (دادن داده های جدید به مدل تمرین داده شده قبلی)، با دریافت ورودی های دیتاست های جدید، همانند قسمت های قبلی انجام میدهیم.

محاسبه دقت جریان محسوسه شده بر اساس داده های ثانویه

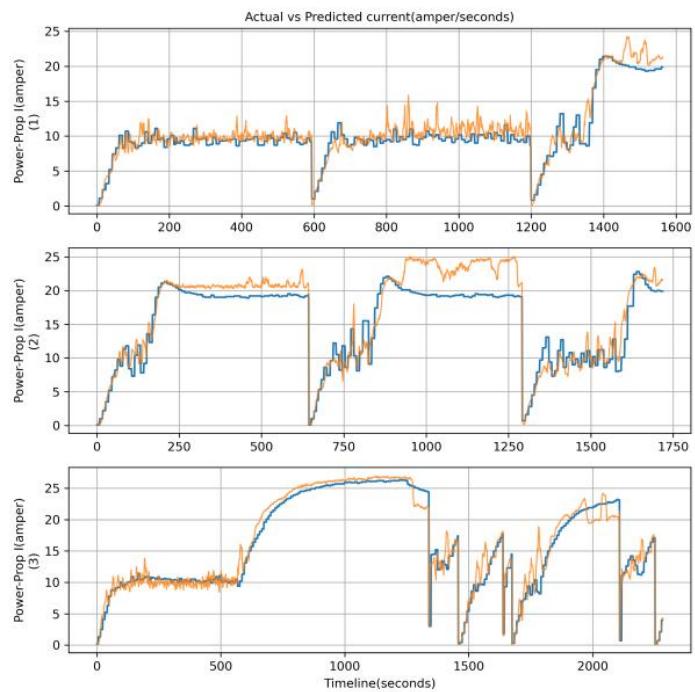
```
[73]: testR2_1 = r2_score(yTest1, testPred1)
testR2_2 = r2_score(yTest2, testPred2)
testR2_3 = r2_score(yTest3, testPred3)
print("نکت پیشنهادی جریان بر اساس داده های ثانویه")
print(testR2_1)
print("نکت پیشنهادی جریان بر اساس داده های ثانویه")
print(testR2_2)
print("نکت پیشنهادی جریان بر اساس داده های ثانویه")
print(testR2_3)

نکت پیشنهادی جریان بر اساس داده های ثانویه:
0.8785224780194866
نکت پیشنهادی جریان بر اساس داده های ثانویه:
0.7968454923460312
نکت پیشنهادی جریان بر اساس داده های ثانویه:
0.9456421621533911
```

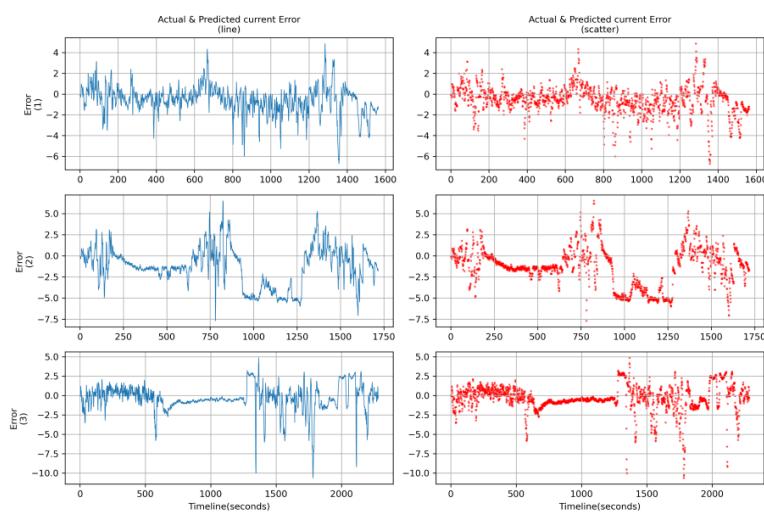
سپس ارزیابی جریان پیش‌بینی شده و واقعی دیتاست ثانویه را، با R2 score انجام میدهیم. درصد این نرخ در درست پیش‌بینی کرده جریان واقعی دیتاست های یک تا سه، به ترتیب 0.79، 0.87 و 0.94 است که میانگین 87.3 درصدی دقت پیش‌بینی جریان در دیتاست های ثانویه را نشان میدهد.

تصویرسازی عملکرد مدل:

در آخرین مرحله از گام اول پروژه به تصویرسازی داده های بدست آمده از مقایسه جریان واقعی و تخمین زده شده در دیتاست دوم، و همچنین مقدار خطای آن، با کمک از کتابخانه Matplotlib میپردازیم.



نمودار مقایسه جریان واقعی و تخمینی در دیتاست های ثانویه



مقدار خطای جریان در دیتاست های ثانویه

- مقادیر صفر: پیشبینی صحیح مقدار جریان
- مقادیر منفی یا مثبت: پیشبینی کمتر و یا پیشبینی بیشتر از مقدار واقعی جریان

فصل 3: گام دوم

در این گام اول از پروژه هدف ما طراحی مدل هوش مصنوعی است که بتواند با دریافت ورودی های دور موتور و عمق موتور، این بار علاوه بر جریان موتور BLDC، ولتاژ پک باتری موتور را نیز در خروجی پیشبینی کند. در شکل زیر شمای کلی مدل هدف را مشاهده میکنیم:



ورودی ها:

- دور موتور (Motor-FB)
- عمق (Depth)

خروجی ها:

- جریان موتور (Power-Prop I)
- ولتاژ پک باتری موتور (Power-Prop V)

نوع مدلسازی هوش مصنوعی:

- شبکه عصبی

مشخصات شبکه عصبی:

- ساختار : 2 – 128 – 64 – 32 – 2
- نوع : Sequential (ترتیبی)
- بهینه ساز : Adam
- تابع فعالساز : ReLU
- تابع زیان : میانگین مربعات خطأ (Mean Squared Error)

لازم به ذکر است علت استفاده از شبکه عصبی در این گام، اثبات مناسب نبودن روش رگرسیون چندگانه برای مدلسازی، با توجه به روابط بین پارامترهای دیتاست و غیرخطی بودن آنها است. روند کلی این گام به این صورت خواهد بود که ابتدا با روش استاندارد شروع به مدلسازی شبکه عصبی خواهیم کرد.

اگر نتایج دقت پیش‌بینی مدل مطلوب نبود، در ادامه و در هر مرحله از تکنیک‌های تکمیلی و کمکی مانند نرم‌السازی داده‌ها، اضافه کردن نویز یا میانگین متحرک، زیر شبکه سازی برای هر خروجی و استفاده خواهیم کرد تا در آخر به دقت مطلوب در پیش‌بینی جریان و ولتاژ موتور BLDC برسیم.

مراحل اصلی گام دوم:

- مدلسازی شبکه عصبی جدید (دو ورودی - دو خروجی)
- ارزیابی عملکرد در دیتاست اصلی
- استفاده از تکنیک نرم‌السازی داده‌ها برای بهبود دقت
- مقایسه و ارزیابی نتایج
- ارزیابی با دیتاست‌های تست ثانویه و مصورسازی

مدلسازی شبکه عصبی:

در این مرحله پروژه همانند گام اول، کتابخانه‌های ضروری را وارد پروژه کرده و فایل دیتاست اصلی که پیش تر مرتب سازی شده بود را در پروژه نوت بوک قرار می‌دهیم.

```
وارد کردن کتابخانه‌های ضروری
[2]: import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split

[12]: df = pd.read_csv('D:/AIjourney/DataSets/Roshd Center/cleaned_dataset.csv')

#df.head(10)
#df[8250:8260]
df.shape

[12]: (16744, 5)
```

در ادامه نیز لیبل زدن و تقسیم بندی داده‌های ورودی و خروجی، و سپس جداسازی داده‌ها با نرخ 80 به 20 برای مجموعه تمرین و تست داده‌ها را انجام میدهیم.

```
[14]: X = df[['Motor-FB', 'Depth1']]
Y = df[['Power-Prop I', 'Power-Prop V']]

#Y[9000:9010]
#X[10000:1010]

[22]: X_train, X_test, Y_train, Y_test = train_test_split(X_normalised, Y_normalised, test_size=0.2, random_state=42)

X_train.shape
#Y_test.shape

[22]: (13395, 2)
```

بعد از آماده سازی داده‌ها نوبت به وارد کردن کتابخانه‌های ضروری شبکه عصبی و همچنین ایجاد ساختار و شیء (object) شبکه عصبی میرسد. تنها تفاوت ساختاری ایجاد شده با شبکه عصبی گام اول پروژه در اضافه کردن دو نورون مصنوعی در لایه خروجی شبکه می‌باشد. بعد از ایجاد شبکه عصبی، آنرا با مجموعه داده train، تمرین میدهیم.

```
[24]: import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Input

[26]: from keras.models import Sequential
from keras.layers import Input, Dense

modelNN = Sequential()
modelNN.add(Input(shape=(2,))) # Input layer with 2 features: Depth and Motor-FB
modelNN.add(Dense(128, activation='relu')) # Hidden Layer 1
modelNN.add(Dense(64, activation='relu')) # Hidden Layer 2
modelNN.add(Dense(32, activation='relu')) # Hidden Layer 3
modelNN.add(Dense(2)) # Output layer with 2 nodes for Power-Prop I and Power-Prop V

modelNN.compile(optimizer='adam', loss='mean_squared_error')

[28]: modelNN.fit(X_train, Y_train, epochs=100, batch_size=32, validation_split=0.2)

Epoch 1/100
335/335 - 5s 5ms/step - loss: 0.0831 - val_loss: 0.0115
Epoch 2/100
335/335 - 1s 4ms/step - loss: 0.0109 - val_loss: 0.0104
Epoch 3/100
335/335 - 1s 4ms/step - loss: 0.0104 - val_loss: 0.0104
Epoch 4/100
335/335 - 1s 4ms/step - loss: 0.0095 - val_loss: 0.0103
```

ارزیابی عملکرد:

در قسمت بعدی و بعد از اتمام پروسه تمرین شبکه عصبی، با دادن داده های ورودی مجموعه تست به مدل، ولتاژ و جریان را پیش‌بینی می‌کنیم. در ادامه با وارد کردن معیارهای ارزیابی R2 score و MAE (محاسبه میانگین قدرمطلق خطاهای)، مقادیر پیش‌بینی شده توسط مدل و مقادیر واقعی جریان و ولتاژ واقع در مجموعه تست را مقایسه می‌کنیم.

```
[15]: prediction = modelNN.predict(X_test)
#predict_normalised.shape
105/105 - 0s 3ms/step

Evaluation

[17]: from sklearn.metrics import r2_score
from sklearn.metrics import mean_absolute_error

[27]: Y_test = np.array(Y_test) # در سورات نیاز بارای taghit datatype
type(Y_test)

[27]: numpy.ndarray

[29]: I_r2 = r2_score(Y_test[:, 0], prediction[:, 0])
V_r2 = r2_score(Y_test[:, 1], prediction[:, 1])

I_mae = mean_absolute_error(Y_test[:, 0], prediction[:, 0])
V_mae = mean_absolute_error(Y_test[:, 1], prediction[:, 1])
```

همانطور که قابل مشاهده است، دقت پیش‌بینی مدل برای پارامتر جریان قابل قبول و معادل 0.92 است. اما مدل در پیش‌بینی ولتاژ ناکام مانده و مقدار منفی 0.81 بیانگر این موضوع می‌باشد. همچنین میانگین خطای در جریان قابل قبول تراز ولتاژ است (با در نظر گرفتن دامنه تغییرات هر پارامتر).

لازم به ذکر است دامنه تغییرات جریان حدود 28 واحد (متغیر از 0 تا 28) و دامنه تغییرات ولتاژ تنها حدود 4 الی 5 واحد (متغیر بین 74 تا 78) می‌باشد.

```
[33]: print('Evaluation results:\n')
print("R² for Power-Prop I:", I_r2)
print("R² for Power-Prop V:", V_r2)
print("MAE for Power-Prop I:", I_mae)
print("MAE for Power-Prop V:", V_mae)
Evaluation results:
R² for Power-Prop I: 0.9268492821664127
R² for Power-Prop V: -0.8162018713489287
MAE for Power-Prop I: 1.391743095424942
MAE for Power-Prop V: 0.6276520413261774
```

برای درک بهتر نحوه پیش‌بینی مدل شبکه عصبی، نمونه داده‌ای شامل دور موتور و عمق را بعنوان ورودی به آن می‌دهیم تا نتیجه آن را با معادل واقعی آن در دیتابست لگ شده اصلی از موتور بررسی کنیم.

```
[35]: sampleData = [[740,688]]      # real Voltage = 76.1 / real current = 22.8
sampleData = np.array(sampleData)
samplePrediction = modelNN.predict(sampleData)
1/1
          0s 179ms/step

[37]: print('Sample data as input:',sampleData)
print('Prediction:',samplePrediction)
Sample data as input: [[740 688]]
Prediction: [[24.002934 74.83231 ]]

[39]: sampleCurrent = samplePrediction[0,0]
sampleVoltage = samplePrediction[0,1]

('مقدار پیش‌بینی شده با ورودی های'
print('740 = ')
print('688 = ')
('ضد موتور = ')
print('برابر است با : ')
print('جريان = ')
print(' ولتاژ = ')
print(' ولتاژ با ورودی های')

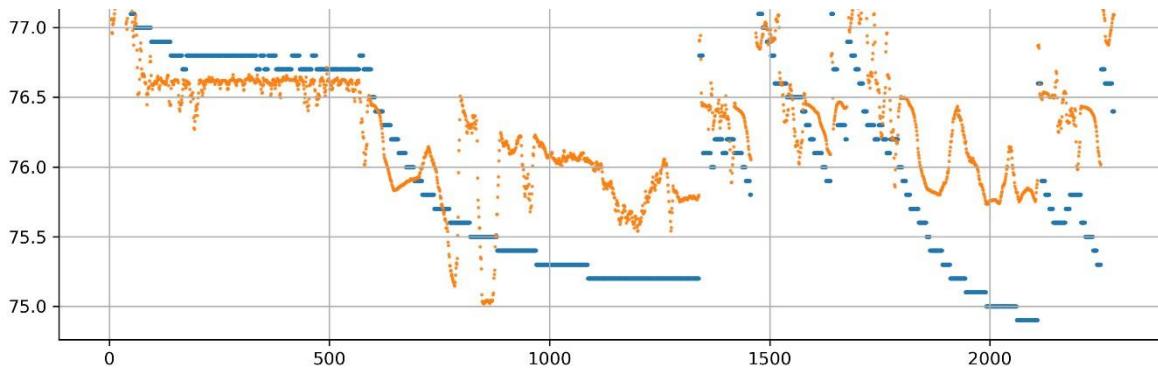
('مقدار پیش‌بینی شده با ورودی های'
740 = دور موتور
688 = ضد موتور
('برابر است با : ')
جريان = 24.002934
ولتاژ = 74.83231
ولتاژ با ورودی های
```

داده ورودی در اینجا عدد 740 برای دور موتور و 688 برای عمق موتور است که در فایل دیتابست اصلی، موتور ولتاژ 76.1 و جریان 22.8 آمپری را برای آن ثبت کرده است. مقادیر پیش‌بینی شده توسط مدل اما معادل 74.8 ولت و 24 آمپر به ازای این ورودی میباشند.

از انجام این تست، متوجه میشویم که اختلاف ولتاژ و جریان پیش‌بینی شده با مقدار اصلی آن تقریباً برابر 1.2 است، اما از آنجا که دامنه تغییرات ولتاژ تنها 5 واحد است، این اختلاف بیشتر از جریان که دامنه تغییرات آن بسیار بیشتر از ولتاژ است به چشم میخورد.

دلایل پیش‌بینی ضعیف ولتاژ:

- کم بودن دامنه تغییرات (بین 74 تا 78)
- روند پله‌ای نمودار ولتاژ (چند ثانیه ثابت و سپس تغییر در مقدار)



پله‌ای بودن لگ ثبت شده از ولتاژ، میتواند یکی از دلایل سخت بودن پیش‌بینی آن توسط مدل، در صورت استفاده ساده و تنها از شبکه عصبی و به کار نگرفتن تکنیک‌های کمکی باشد.

استفاده از تکنیک نرمالسازی:

همانطور که در اولین تلاش برای مدلسازی مشاهده شد، دقت مدل در پیش‌بینی ولتاژ مطلوب نبود. در ادامه با اضافه کردن تکنیک تکمیلی نرمالسازی داده‌ها در مرحله پیش‌پردازش داده‌ها، برای بار دوم و با هدف بهبود دقت پیش‌بینی ولتاژ اقدام به مدلسازی می‌کنیم.

بصورت کلی نرمالسازی داده‌ها با کمک مقیاس‌ساز (Scaler) در مواردی که پارامترها دامنه تغییرات متفاوت داشته و یا مقادیر عددی آنها نسبت به یکدیگر اختلاف زیادی دارند استفاده می‌شود. به این صورت که تمامی مقادیر عددی داده‌ها در یک نمودار نرمال، مقیاس بندی شده، و به مقیاس مناسب نسبت به یکدیگر می‌رسند. این کار معمولاً باعث بهبود دقت مدل و کاهش زمان یادگیری الگوریتم، به علت یکنواخت شدن مقیاس و کاهش دامنه مقادیر می‌شود.

برای انجام این امر از مقیاس‌ساز موجود در کتابخانه `sklearn` به اسم `MinMaxScaler` استفاده می‌کنیم. در ابتدا تابع `モوردنظر از کتابخانه را وارد پروژه کرده و دو شیء از آن می‌سازیم. یکی از شیء‌ها برای نرمالسازی داده‌های ورودی (دور و عمق)، و دیگری برای نرمالسازی داده‌های خروجی (جریان و ولتاژ) استفاده خواهد شد.`

```
[16]: from sklearn.preprocessing import MinMaxScaler
input_scaler = MinMaxScaler()
output_scaler = MinMaxScaler()
```

در مرحله بعدی داده‌های جدا شده ورودی و خروجی را با مقیاس کننده مرتبط خود می‌کنیم. با این امر نه تنها داده‌ها نرمالسازی شده و به مقیاسی معادل خود در بین ۰ تا ۱ میرسند، بلکه پارامترهای آماری متناسب به نرمالسازی داده‌ها مانند میانگین، مینیمم، ماکسیمم و... در شیء ساخته شده از مقیاس‌ساز `MinMaxScaler` مورد نظر ذخیره شده، تا بعداً برای انجام نرمالسازی یا عمل معکوس آن یعنی `De-normalization` (لغو مقیاس بندی) با داده‌های جدید آماده باشند.

```
[18]: X_normalised = input_scaler.fit_transform(X)
Y_normalised = output_scaler.fit_transform(Y)
```

داده های نرمالسازی شده بصورت زیر تصویر میباشند.

```
[20]: #X_normalised[1000:1010]
Y_normalised[1000:1010]
```

```
[20]: array([[0.62781955, 0.36842105],
 [0.64661654, 0.36842105],
 [0.64661654, 0.36842105],
 [0.64661654, 0.36842105],
 [0.64661654, 0.36842105],
 [0.64661654, 0.36842105],
 [0.64661654, 0.36842105],
```

در ادامه نیز به تمرین دادن شبکه عصبی میپردازیم. ساختار شبکه عصبی نسبت به دفعه قبلی تغییری نکرده، و فقط در این بخش داده های نرمالسازی شده را برای مدلسازی استفاده میکنیم.

بعد از اتمام مدلسازی، مانند دفعه قبل اقدام به پیشبینی مقدار خروجی با مجموعه داده تست میکنیم، و سپس نتایج واقعی و پیشبینی مدل را مقایسه و ارزیابی میکنیم.

```
[36]: I_r2 = r2_score(Y_test[:, 0], predict_normalised[:, 0])
V_r2 = r2_score(Y_test[:, 1], predict_normalised[:, 1])

I_mae = mean_absolute_error(Y_test[:, 0], predict_normalised[:, 0])
V_mae = mean_absolute_error(Y_test[:, 1], predict_normalised[:, 1])
```

```
[38]: print('Evaluation results:\n')

print("R^2 for Power-Prop I:", I_r2)
print("R^2 for Power-Prop V:", V_r2)

print("MAE for Power-Prop I:", I_mae)
print("MAE for Power-Prop V:", V_mae)

Evaluation results:

R^2 for Power-Prop I: 0.939339244222172
R^2 for Power-Prop V: 0.651533416142557
MAE for Power-Prop I: 0.044251525978481816
MAE for Power-Prop V: 0.06296488607752787
```

همانطور که قابل مشاهده است نرخ R² برای این مدل بهبود یافته. دقت مدل در پیشبینی جریان بر روی دیتاست اصلی تقریبا 0.94 و در پیشبینی ولتاژ 0.65 شده است. همچنین بهبود در میانگین خطاهای (معیار MAE) نیز کاملا ملموس میباشد.

نکته ای درباره دادن مقادیر ورودی به مدل:

همانطور که مشاهده شد، در روند تمرین دادن شبکه عصبی، از داده های مقیاس بندی شده توسط مقیاس ساز MinMaxScaler استفاده شد، که در نتیجه داده های وارد شده با مدل، همگی به مقیاسی بین 0 تا 1 تغییر یافته اند. حال از آنجا که مدل شبکه عصبی تمرین داده شده، با داده های مقیاس بندی شده تمرین داده شده است، لذا در هنگام دادن ورودی های جدید به مدل (به منظور تمرین بیشتر و یا بررسی عملکرد)، باید قبل از آن، داده های نمونه را با استفاده از مقیاس سازهای تنظیم شده (که

بصورت فایل pkl ذخیره شده اند) به مقیاس مناسب خود(بین صفر و یک) تغییر بدهیم، و سپس عنوان ورودی به مدل بدهیم. این امر با کمک تابع transform در کنار مقیاس ساز ورودی تنظیم شده input_scaler به وقوع می پیوندد.

همچنین داده های پیش‌بینی شده (خروجی) مدل شبکه عصبی نیز همچنان مقیاس بندی شده و متفاوت با مقادیر اصلی ولتاژ و جریان دیتابست هستند. برای لغو مقیاس آنها نیز باید از تابع inverse_transform در کنار مقیاس ساز خروجی (output_scaler) استفاده شود، تا داده های پیش‌بینی شده، به مقیاس استاندارد خود بازگردند!

در مرحله بعد نمونه داده ورودی کاملا مشابه با مرحله قبل به مدل میدهیم تا پیش‌بینی آن را با مدلسازی اولیه مقایسه کنیم.

```
[40]: sampleData = [[740,688]]          # real Voltage = 76.1 / real Current = 22.8
sampleData_normalised = input_scaler.transform(sampleData)
samplePredict_normalised = modelNN.predict(sampleData_normalised)
samplePredict = output_scaler.inverse_transform(samplePredict_normalised)

1/1 ━━━━━━━━ 0s 74ms/step
D:\AIjourney\programs\AnacondaFiles\Lib\site-packages\sklearn\base.py:493: UserWarning: X does not have valid feature names, but MinMaxScaler was fitted
with feature names
  warnings.warn(
[42]: print('sampleData_normalised:',sampleData_normalised)
print('samplePredict_normalised:',samplePredict_normalised)
print('samplePredict:',samplePredict)

sampleData_normalised: [[0.70490196 0.48736462]]
samplePredict_normalised: [[0.84012717 0.32487753]]
samplePredict: [[22.447382 76.13454 ]]
```

برای اینکار ابتدا ورودی نمونه، یعنی دور موتور 740 و عمق 688 را نرمالسازی یا همان مقیاس بندی می‌کنیم تا برای مدل قابل استفاده باشد. سپس بعد از پیش‌بینی، حاصل را با استفاده از مقیاس کننده خروجی، دی نرمالایز یا همان لغو مقیاس بندی می کنیم تا به مقادیر ولتاژ و جریان صحیح برسند.(output_scaler.inverse_transform).

همانطور که مشخص است پیش‌بینی ولتاژ 76.18 ولت، در ازای مقدار واقعی دیتابست که 76.1 ولت است می‌باشد. برای جریان نیز مقدار پیش‌بینی 22.44 در ازای مقدار واقعی 22.8 است که هر دو نشان از بهبود یافتن دقت مدل، مخصوصا در پیش‌بینی ولتاژ می‌باشند.

```
[71]: import joblib
[73]: modelNN.save('D:/AIjourney/NN_Models/Phase2/BLDC_NN_Model1.h5')
joblib.save('D:/AI/NN_Models/inputScaler1.pkl', input_scaler)
joblib.save('D:/AI/NN_Models/outputScaler1.pkl', output_scaler)
print('Model saved !!')

WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
Model saved !!
```

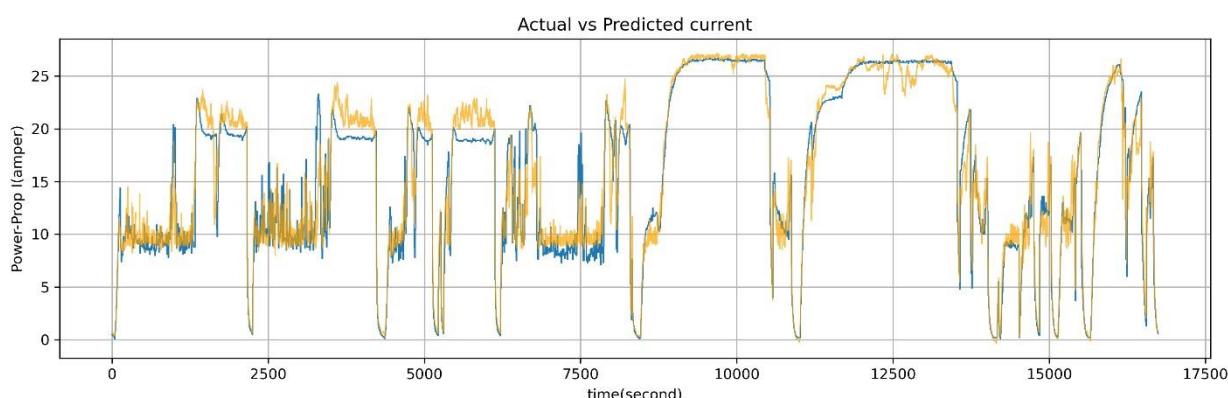
قبل از ادامه مراحل، مدل شبکه عصبی و مقیاس سازها را بصورت محلی بر روی دستگاه ذخیره میکنیم.

این امر به ما این امکان را میدهد تا در پروژه ها یا سیستم های دیگر به راحتی فایل های مورد نظر را انتقال داده، و از آنها در پیشビینی مقادیر جدید استفاده کنیم.

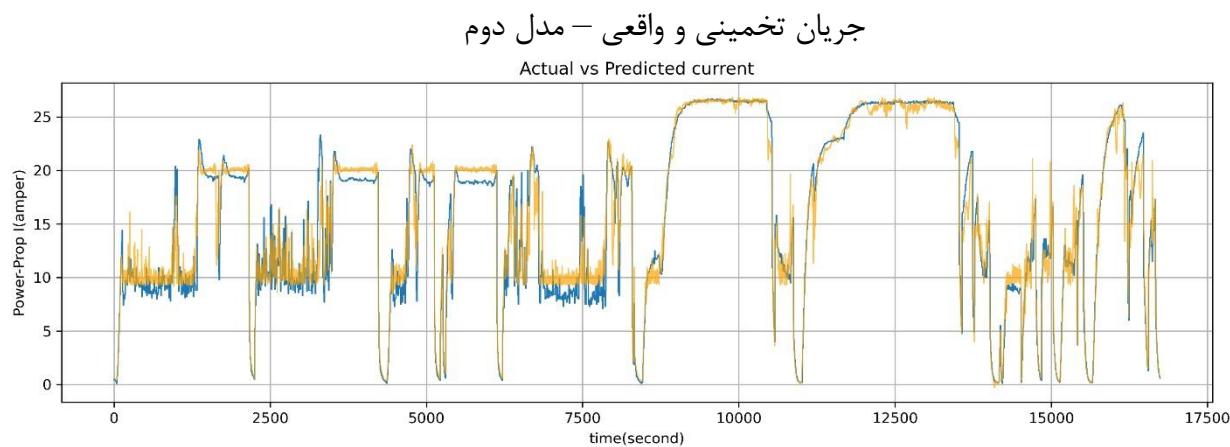
مقایسه و ارزیابی نتایج:

در این قسمت با استفاده از کتابخانه matplotlib اقدام به مصورسازی داده های بدست آمده بر روی نمودار ها میکنیم. این امر منجر به درک بهتر از نحوه عملکرد مدل تمرین داده شده در دفعه اول و دفعه دوم، در پیشビینی مقادیر جریان و ولتاژ میشود.

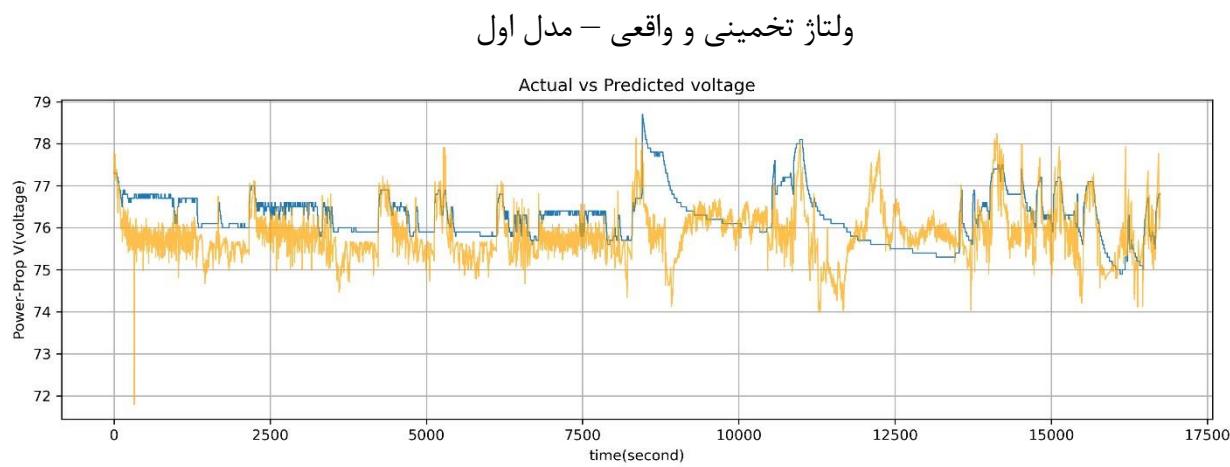
جریان تخمینی و واقعی – مدل اول



در این نمودار روند آبی رنگ جریان واقعی و روند نارنجی رنگ پیشビینی مدل در بار نخست مدلسازی است.



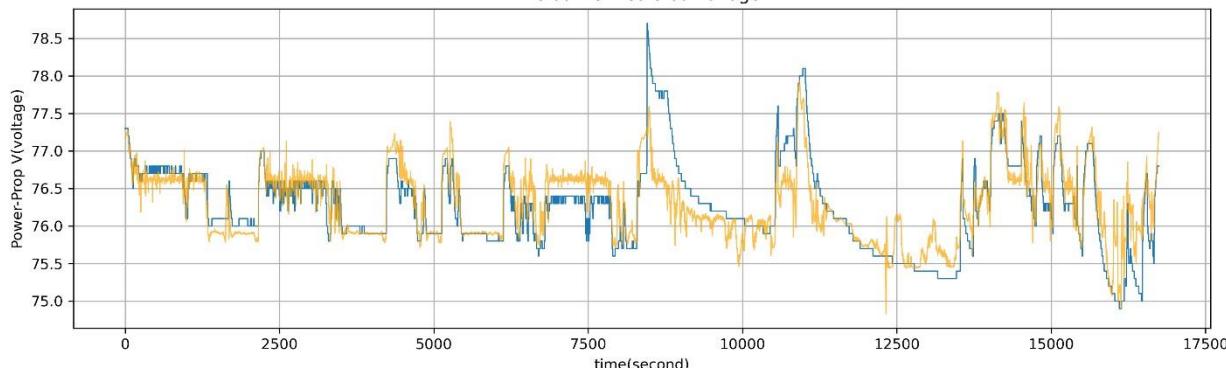
همانطور که قابل مشاهده است، در مدل دوم و با کمک گرفتن از تکنیک نرمالسازی، دقیق مدل در پیش‌بینی جریان بهبود یافته است.



کاملا مشهود است که مدل اول، در پیش‌بینی ولتاژ ناکام مانده و دقیق نامطلوبی در حدس زدن ولتاژ موتور، با ورودی دور و عمق موتور دارد.

ولتاژ تخمینی و واقعی – مدل دوم

Actual vs Predicted voltage



بهبود مدل دوم با کمک از نرمالسازی داده ها، در پیش‌بینی ولتاژ ملموس است!

ارزیابی مدل با دیتاست های ثانویه:

در این قسمت با استفاده از دیتاست های تست، که در گام نخست پروژه مرتب سازی شده اند، اقدام به تست و ارزیابی مدل تمرین داده شده دوم با نرمالسازی داده ها میکنیم(بدیهی است برای ارزیابی باید داده ها را با کمک مقیاس سازهای ذخیره شده، مقیاس بندی کنیم).

ابتدا به ایجاد پروژه جدید، وارد کردن کتابخانه های ضروری و مدل شبکه عصبی میپردازیم. در ادامه برای ایجاد بهینه ساز و تابع زیان شبکه عصبی، آنرا کامپایل میکنیم.

```
[1]: import tensorflow as tf
from tensorflow.keras.models import load_model
```

```
[3]: import joblib
```

```
[5]: loadedModel = load_model('D:/AIjourney/NN_Models/Phase2/BLDC_NN_Model1.h5')
loadedModel.compile(optimizer='adam', loss='mean_squared_error')
loadedModel.summary()
```

WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until you train or evaluate the model.

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 128)	384
dense_1 (Dense)	(None, 64)	8,256
dense_2 (Dense)	(None, 32)	2,080
dense_3 (Dense)	(None, 2)	66

Total params: 10,786 (42.13 KB)

Trainable params: 10,786 (42.13 KB)

Non-trainable params: 0 (0.00 B)

در ادامه به وارد کردن مقیاس سازهای ورودی و خروجی، که در پروژه قبل آنها را ایجاد، تنظیم و ذخیره کرده بودیم می کنیم.

```
[19]: inScaler = joblib.load('D:/AIjourney/NN_Models/Phase2/scalers/inputScaler1.pkl')
outScaler = joblib.load('D:/AIjourney/NN_Models/Phase2/scalers/outputScaler1.pkl')
```

در مرحله بعدی دیتاست های ثانویه را وارد پروژه میکنیم. در ادامه برای هر یک از آنها لیل زدن داده های ورودی و خروجی را اجرا، و هر کدام را با استفاده از مقیاس کننده های لود شده، در متغیری جداگانه نرمالسازی میکنیم(توجه: داده های ورودی با مقیاس ساز ورودی، و داده های خروجی با مقیاس ساز خروجی مقیاس بندی می شوند). در آخر نیز با استفاده از ورودی های نرمالسازی شده و ارائه آنها به مدل شبکه عصبی، اقدام به پیش‌بینی خروجی ها می کنیم.

```
[60]: dfTest1 = pd.read_csv('D:/AIjourney/DataSets/Roshd Center/CleanedTest1.csv')
dfTest2 = pd.read_csv('D:/AIjourney/DataSets/Roshd Center/CleanedTest2.csv')
dfTest3 = pd.read_csv('D:/AIjourney/DataSets/Roshd Center/CleanedTest3.csv')

[118]: test1_X = dfTest1[['Motor-FB','Depth1']]
test1_Y = dfTest1[['Power-Prop I','Power-Prop V']]

test2_X = dfTest2[['Motor-FB','Depth1']]
test2_Y = dfTest2[['Power-Prop I','Power-Prop V']]

test3_X = dfTest3[['Motor-FB','Depth1']]
test3_Y = dfTest3[['Power-Prop I','Power-Prop V']]

[150]: test1_X_norm = inScaler.transform(test1_X)
test2_X_norm = inScaler.transform(test2_X)
test3_X_norm = inScaler.transform(test3_X)

test1_Y_norm = outScaler.transform(test1_Y)
test2_Y_norm = outScaler.transform(test2_Y)
test3_Y_norm = outScaler.transform(test3_Y)

[170]: predTest1 = loadedModel.predict(test1_X_norm)
predTest2 = loadedModel.predict(test2_X_norm)
predTest3 = loadedModel.predict(test3_X_norm)

49/49 ━━━━━━ 0s 3ms/step
54/54 ━━━━ 0s 2ms/step
72/72 ━━━━ 0s 2ms/step
```

سپس نوبت به ارزیابی نرخ R2 برای مقایسه داده های خروجی پیش‌بینی شده در مقابل مقادیر واقعی آنها میرسد.

```
[176]: print('Current accuracy test1:',r2_score(test1_Y_norm[:,0], predTest1[:,0]))
print('Voltage accuracy test1:',r2_score(test1_Y_norm[:,1], predTest1[:,1]))

print('Current accuracy test2:',r2_score(test2_Y_norm[:,0], predTest2[:,0]))
print('Voltage accuracy test2:',r2_score(test2_Y_norm[:,1], predTest2[:,1]))

print('Current accuracy test3:',r2_score(test3_Y_norm[:,0], predTest3[:,0]))
print('Voltage accuracy test3:',r2_score(test3_Y_norm[:,1], predTest3[:,1]))
```

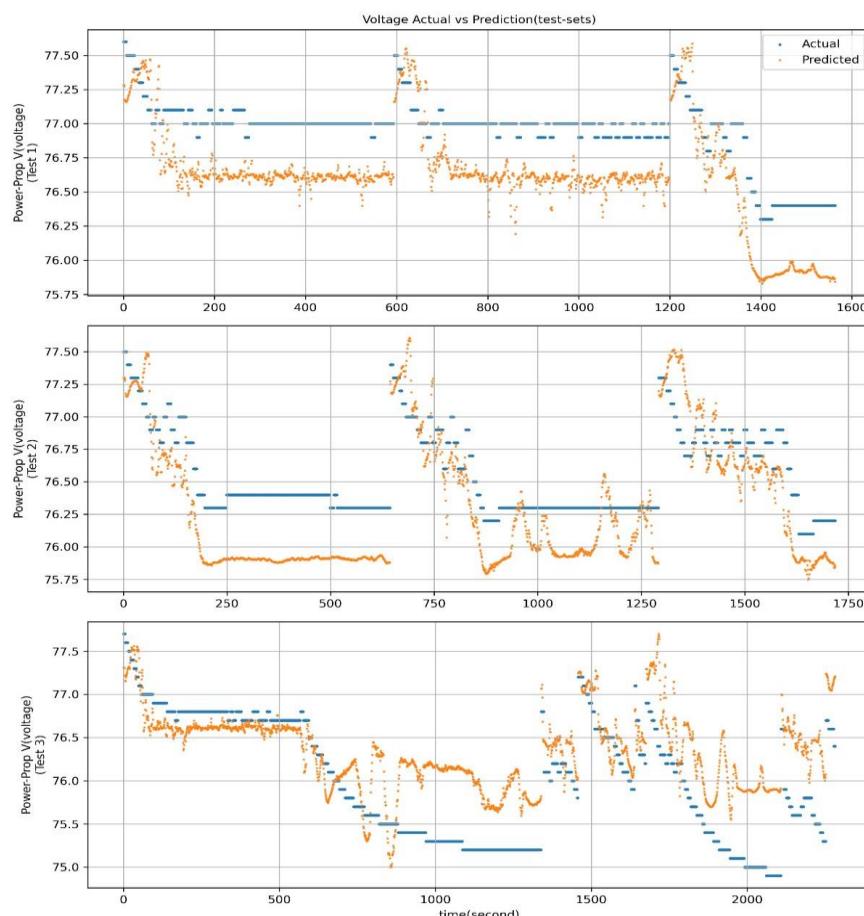
Current accuracy test1: 0.8932316540786757
 Voltage accuracy test1: -1.501104721011843
 Current accuracy test2: 0.925078597477937
 Voltage accuracy test2: -0.12562870620171607
 Current accuracy test3: 0.9523739492740982
 Voltage accuracy test3: 0.3641120762900182

در نتایج بدست آمده، دقت پیش‌بینی جریان برای دیتاست های تست اول تا سوم، به ترتیب برابر 0.89، 0.95 و 0.92 است که نشان از دقت بالایی دارد. اما برای ولتاژ نرخ های بدست آمده به ترتیب برابر منفی 1.5، منفی 0.36 و مثبت 0.12 است که نسبت به جریان عملکرد ضعیف تری می‌باشد.

کاهش دقت پیش‌بینی ولتاژ میتواند به دلیل اورفیت یا بیش برازش (عادت به پیش‌بینی دقیق روی دیتاست تمرین و ضعف در پیش‌بینی دیتاست های جدید) باشد. هر چند، مدل شبکه عصبی با کمک از تکنیک نرمال‌سازی، پیش‌بینی ولتاژ دقیق تری نسبت به مدل اول دارد، و حتی با در نظر گرفتن نرخ R2 نامطلوب که با توجه به کم بودن دامنه تغییرات پارامتر، به شدت حساس عمل می‌کند، تمامی پیش‌بینی های خود را در دامنه تغییرات ولتاژ انجام داده است.

برای درک بهتر این موضوع و همچنین نحوه عملکرد کلی در پیش‌بینی دیتاست های ثانویه، در ادامه به مصورسازی داده ها بر روی نمودار ها می‌کنیم.

ولتاژ تخمینی و واقعی – دیتاست های ثانویه



از نمودار حاصل، میتوان برداشت کرد که هر چند اختلاف ولتاژ واقعی و پیشビینی قابل انکار نیست، اما اکثر اختلافات ولتاژ تخمینی و ولتاژ واقعی، بسیار کم هستند(کمتر از 1 ولت است حداقل 3.1 ولت)، اما از آنجا که بازه تغییرات ولتاژ تنها 4 الی 5 ولت است، این اختلاف تاثیر بسزایی در تعیین نرخ R2 میگذارد.

همچنین قابل ذکر است که تمامی مقادیر ولتاژ پیشビینی شده توسط مدل شبکه عصبی، در دامنه تغییرات مقادیر واقعی ولتاژ بوده، و از آن خارج نشده اند.

جمع بندی:

در گام دوم پروژه که تا به اینجا انجام شده است، از دو مدلسازی استفاده کردیم. مدلسازی اول که بدون کمک از تکنیک های تکمیلی بود، پیشビینی قابل قبول در جریان، و نامطلوب در ولتاژ را داشت. در مدلسازی دوم اما با کمک از تکنیک نرمالسازی، دقیق ولتاژ بشدت بهبود یافته و مقادیر دقیق تری به ولتاژ واقعی را شاهد بودیم لازم به ذکر است در مدلسازی دوم، دقیق پیشビینی جریان نیز بهبود جزئی را شاهد بود.

و اما در تست و بررسی مدل با دیتاست های ثانویه، دقیق پیشビینی جریان با استفاده از نرخ R2 کاهش کمی داشت. مورد قابل تأمل کاهش چشمگیر نرخ R2 برای پیشビینی و تخمین ولتاژ توسط شبکه عصبی در دیتاست های ثانویه است که میتواند به دلیل اورفیت یا غیره باشد، که کوچک بودن دامنه تغییرات ولتاژ، وزن بشدت بیشتری در خطاهای ثبت شده توسط آن در نرخ R2 میدهد.

همچنین در صورت قابل قبول نبودن دقیق پیشビینی ولتاژ، امکان امتحان کردن راه ها و تکنیک های کمکی دیگر به منظور افزایش آن وجود دارد.

فصل 4: گام سوم

در گام سوم از پروژه، هدف ما طراحی مدل هوش مصنوعی است که بتواند با دریافت سه ورودی دور موتور، عمق و ولتاژ پک باتری، بتواند جریان موتور BLDC را در خروجی پیش بینی کند. در شکل زیر شمای کلی مدل هدف را مشاهده میکنیم:



ورودی ها:

- ولتاژ پک باتری موتور (Power-Prop V)
- عمق (Depth)
- دور موتور (Motor-FB)

خروجی ها:

- جریان موتور (Power-Prop I)

نوع مدلسازی هوش مصنوعی:

- شبکه عصبی

مشخصات شبکه عصبی:

- ساختار : 3 – 128 – 64 – 32 – 1
- نوع : Sequential (ترتیبی)
- بهینه ساز : AdamW
- تابع فعالساز : Leaky ReLU
- تابع زیان : میانگین مربعات خطأ (Mean Squared Error)

در این گام از پروژه و در تلاش اول، یک روش ساده را به منظور مدلسازی شبکه عصبی (سه ورودی – یک خروجی) امتحان میکنیم که به علت مطلوب نبودن نتیجه آن، در تلاش دوم شبکه عصبی را دست خوش تغییرات متعددی از جمله تغییر در پارامترها، تابع فعالساز، تابع زیان، تکنیک های کمکی و.. میکنیم. لازم به ذکر است تمامی موارد استفاده شده در روند مدلسازی، در ادامه گزارش توضیح داده خواهند شد.

تلash اول(مدلسازی به روش ساده)

در ابتدای تلش اول قصد داریم رابطه بین پارامترهای ولتاژ و جریان را با یک نمونه مدلسازی، بهتر درک کنیم و در ادامه تلش اول، یک شبکه عصبی ساده را با هدف پیش‌بینی جریان (دریافت سه ورودی) طراحی کنیم.

همانند گام‌های قبلی، ابتدا دیتاست مرتب سازی شده را وارد پروژه کرده، و با لیبل زدن دیتا‌های، داده‌ای ورودی (ولتاژ) و خروجی (جریان) را جداسازی می‌کنیم.

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split

df = pd.read_csv('D:/AIjourney/DataSets/Roshd Center/cleaned_dataset.csv')
#df.shape
```

سپس با استفاده از مقیاس‌ساز MinMaxScaler داده‌ها را مقیاس‌بندی و همچنین شیء مقیاس‌ساز را طبق داده‌ای تنظیم می‌کنیم. در ادامه نیز داده‌ها را به دو بخش تمرین و تست بخش‌بندی می‌کنیم.

```
[ ] inputScaler = MinMaxScaler()
outputScaler = MinMaxScaler()

[ ] X = df[['Power-Prop V']]
Y = df[['Power-Prop I']]

[ ] X_norm = inputScaler.fit_transform(X)
Y_norm = outputScaler.fit_transform(Y)

[ ] X_train, X_test, Y_train, Y_test = train_test_split(X_norm, Y_norm, test_size=0.2, random_state=42)
```

سپس نوبت به طراحی، ساخت و تمرین دادن شبکه عصبی با هدف مقایسه جریان و ولتاژ میرسد تا بتوانیم درک بهتری از رابطه بین این دو پارامتر داشته باشیم.

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Input

modelNN = Sequential([
    modelNN.add(Input(shape=(1,)))
    modelNN.add(Dense(128, activation='relu'))
    modelNN.add(Dense(64, activation='relu'))
    modelNN.add(Dense(32, activation='relu'))
    modelNN.add(Dense(1))
])

modelNN.compile(optimizer='adam', loss='mean_squared_error')
```

از ادامه مراحل را به علت تکراری بودن، و همچنین تستی بودن این شبکه گذر می‌کنیم تا به بررسی دقیق مدل در پیش‌بینی جریان، تنها با ورودی ولتاژ پک باشیم.

دقیق مدل در دیتاست اصلی

```
prediction1 = modelNN.predict(X_test)
105/105 ━━━━━━━━ is 5ms/step

# not necessary
test_loss = modelNN.evaluate(X_test, Y_test)
print(f'Test Loss: {test_loss}')

105/105 ━━━━━━━━ is 5ms/step - loss: 0.0301
Test Loss: 0.0300933372676372528

from sklearn.metrics import r2_score

evalCurrent = r2_score(Y_test, prediction1)
print('Current prediction accuracy:', evalCurrent)

Current prediction accuracy: 0.6537688347966657
```

دقت مدل در دیتاست ثانویه

```

predTest1 = modelNN.predict(test1_X_norm)
predTest2 = modelNN.predict(test2_X_norm)
predTest3 = modelNN.predict(test3_X_norm)

49/49 ————— 0s 4ms/step
54/54 ————— 0s 4ms/step
72/72 ————— 0s 3ms/step

predTest1_denoorm = outputScaler.inverse_transform(predTest1)
predTest2_denoorm = outputScaler.inverse_transform(predTest2)
predTest3_denoorm = outputScaler.inverse_transform(predTest3)

print("Current accuracy test1:",r2_score(test1_Y['Power-Prop 1'], predTest1_denoorm))
print("Current accuracy test2:",r2_score(test2_Y['Power-Prop 1'], predTest2_denoorm))
print("Current accuracy test3:",r2_score(test3_Y['Power-Prop 1'], predTest3_denoorm))

Current accuracy test1: 0.18942279298868249
Current accuracy test2: 0.29380178884045087
Current accuracy test3: 0.6912898657959892

```

همانطور که قابل مشاهده است با ارزیابی دقت این مدل در پیش‌بینی جریان، به دقت 0.65 در دیتاست اصلی و دقتهای 0.10، 0.29 و 0.69 در دیتاست‌های ثانویه میرسیم. از ارزیابی فوق میتوان به این نتیجه گیری رسانید که ارتباط متوسطی بین ولتاژ پک با تری ثبت شده و جریان موتور وجود دارد که بصورت قابل توجهی ضعیف تر از ارتباط بین پارامترهای عمق و دور موتور با جریان موتور است. همانطور که در گام‌های قبلی نیز بررسی شد، علت عدمه این امر، در اولویت اول رفتار پله ای داده‌های ولتاژ و در اولویت دوم، دامنه تغییرات کم آن است (البته با مقیاس بندی کردن داده‌ها، مورد دوم پوشش داده میشود).

در ادامه تلاش اول از گام سوم پروژه، به تمرین دادن مدل ساده شبکه عصبی با سه ورودی ولتاژ پک با تری، عمق و دور موتور و یک خروجی جریان میپردازیم.

انجام پیش‌پردازش داده‌ها

همانند گام‌های پیشین، ابتدا بخش بنده داده‌های ورودی و خروجی از دیتافریم اصلی را خواهیم داشت. سپس در ادامه مقیاس بندی داده‌های ورودی و خروجی و تقسیم بندی آنها به داده‌های تمرین و تست را انجام خواهیم داد. لازم به ذکر است از آنجا که مقیاس ساز قبلی (inputScaler) با تک ورودی ولتاژ تنظیم شده بود، لذا باید مقیاس ساز جدید (inputScalerNew) ساخته و آنرا مناسب با سه داده ورودی این گام تنظیم کنیم و سپس مقیاس بندی را انجام بدھیم.

```

[ ] X_new = df[['Power-Prop V','Depth1','Motor-FB']]
Y_new = df[['Power-Prop 1']]

[ ] #X_new.head()
#Y_new.head()

[ ] inputScalerNew = MinMaxScaler()

[ ] X_new_norm = inputScalerNew.fit_transform(X_new)
Y_new_norm = outputScaler.transform(Y_new)

[ ] #print(X_new_norm[1000:1010])
#print(Y_new_norm[1000:1010])

[ ] X_new_train, X_new_test, Y_new_train, Y_new_test = train_test_split(X_new_norm, Y_new_norm, test_size=0.2, random_state=42)

```

مدلسازی شبکه عصبی

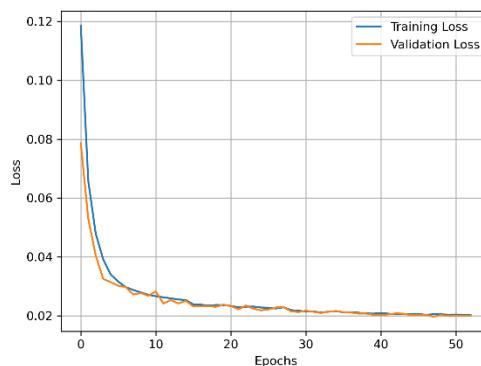
در ادامه تلاش اول، شبکه عصبی ساده را دقیقاً مطابق ساختار و پارامترهایی که در گام قبلی ساخته بودیم، ایجاد کرده و تمرین میدهیم. تنها تفاوت این شبکه عصبی با شبکه عصبی گام قبلی، در تعداد نورون های ورودی شبکه است که در این قسمت، 3 نورون برای دریافت ورودی ها تعیین میکنیم. پارامترهای تمرین دادن شبکه همچون تعداد دور های آن (epoch) و غیره هم به همان صورت خواهد بود.

```
[ ] modelNN_new = Sequential()
modelNN_new.add(Input(shape=(3,)))
modelNN_new.add(Dense(128, activation='relu'))
modelNN_new.add(Dense(64, activation='relu'))
modelNN_new.add(Dense(32, activation='relu'))
modelNN_new.add(Dense(1))

modelNN_new.compile(optimizer='adam', loss='mean_squared_error')

[ ] history_new = modelNN_new.fit(X_new_train, Y_new_train, epochs=100, batch_size=32, validation_split=0.2)
```

نمودار تغییرات خطای اعتبار سنجی در فرایند تمرین داده شدن



ارزیابی مدل با دیتاست اصلی و ثانویه

در ادامه مدل تمرین داده شده را با داده های تست جدا شده (از دیتاست اصلی) و همچنین دیتاست های مرتب سازی شده ثانویه ارزیابی (بر اساس ضریب تعیین یا همان R2 score) ارزیابی میکنیم.

ارزیابی مدل با دیتاست اصلی

```
[ ] prediction1_new = modelNN_new.predict(X_new_test)

# not necessary
test_loss_new = modelNN_new.evaluate(X_new_test, Y_new_test)
print(f'Test Loss: {test_loss_new}')

→ 105/105 ━━━━━━━━ 0s 2ms/step
→ 105/105 ━━━━━━ 0s 3ms/step - loss: 0.0025
Test Loss: 0.002314073033630848

[ ] evalCurrent_new = r2_score(Y_new_test, prediction1_new)
print('Current prediction accuracy(3 inputs):', evalCurrent_new)

→ Current prediction accuracy(3 inputs): 0.9740990210700539
```

ارزیابی مدل با دیتاست ثانویه

```
[ ] predTest1_deNorm_new = outputScaler.inverse_transform(predTest1_new)
predTest2_deNorm_new = outputScaler.inverse_transform(predTest2_new)
predTest3_deNorm_new = outputScaler.inverse_transform(predTest3_new)

[ ] print('Current accuracy test1:',r2_score(test1_Y_new['Power-Prop I'], predTest1_deNorm_new))
print('\n')
print('Current accuracy test2:',r2_score(test2_Y_new['Power-Prop I'], predTest2_deNorm_new))
print('\n')
print('Current accuracy test3:',r2_score(test3_Y_new['Power-Prop I'], predTest3_deNorm_new))

→ Current accuracy test1: 0.8684639715179797
Current accuracy test2: 0.7097694500751692
Current accuracy test3: 0.9351682826683599
```

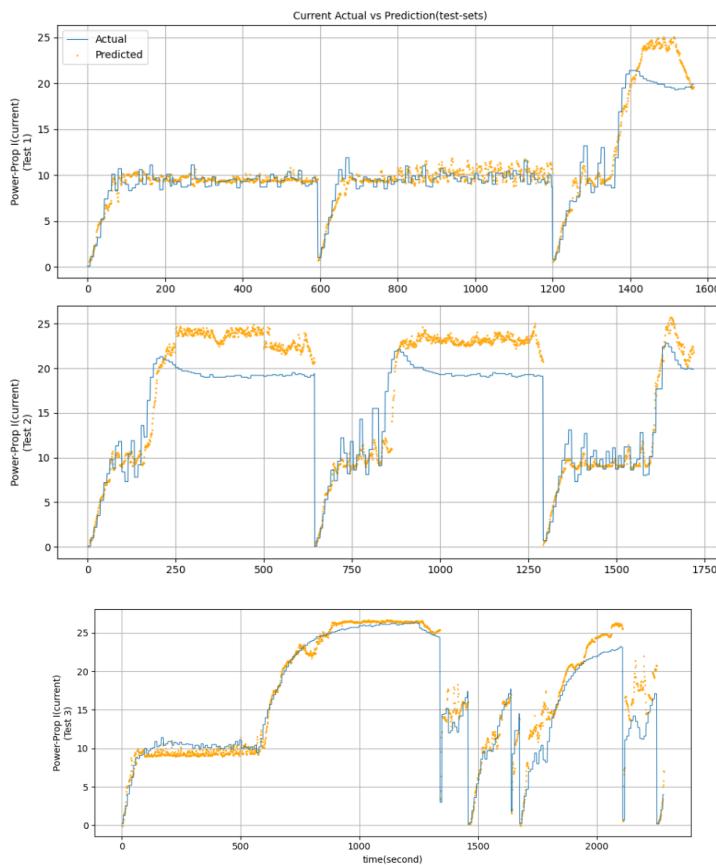
لازم به ذکر است برای انجام عمل ارزیابی با دیتاست ثانویه، مراحل گام های قبل (وارد کردن دیتاست، بخش بندی داده های ورودی و خروجی و مقیاس بندی) را قبل از وارد کردن داده ها به مدل و پیشビینی جریان خروجی، انجام میدهیم.

همانطور که مشهود است، نتیجه ارزیابی با دیتاست اصلی 0.97 و برای دیتاست های ثانویه به ترتیب 0.70، 0.86 و 0.93 با میانگین 0.83 است. از نتایج بدست آمده میتوان نتیجه گرفت که شبکه عصبی ساده، دقت نسبتاً خوبی در تخمین زدن جریان موتور دارد، که البته فضا برای بهتر کردن آن با استفاده از تکنیک های تکمیلی وجود دارد. نکته حائز اهمیت، اختلاف زیاد دقت مدل، بین دیتاست اصلی و دیتاست های ثانویه است که احتمالاً به علت وقوع پدیده overfitting میباشد.

- Overfit : در هنگامی رخ میدهد که مدل هوش مصنوعی بیش از حد (تعداد دور تکرار، epoch) بر روی دیتاست تمرین داده شود. حاصل این پدیده افزایش دقت قابل توجه مدل در پیشビینی مقادیر دیتاست استفاده شده برای تمرین، و ضعف در پیشビینی مقادیر خارج از دیتاست اصلی میباشد. به بیانی دیگر مدل ارتباط بین دیتاست اصلی را حفظ کرده و قادر به پیشビینی مقادیر خارج از آن، با دقت بالا نخواهد بود.

تصویرسازی

در مرحله آخر از تلاش اول، به تصویرسازی نمودار مقایسه جریان تخمینی و واقعی روی دیتاست های ثانویه توسط شبکه عصبی ساده میپردازیم. همانند گام های قبلی این عمل با کمک از کتابخانه Matplotlib انجام میشود.



در نمودار تولید شده، تقاط نارنجی رنگ جریان تخمین زده شده و خط آبی جریان واقعی است. اختلاف بین مقادیر واقعی و تخمینی نسبتاً کم هستند، اما در بعضی نقاط غیرقابل انکار می باشند.

نتیجه گیری تلاش اول

همانطور که مشاهده کردیم، دقت مدلسازی شبکه عصبی ساده برای سه ورودی ولتاژ، عمق، دور موتور و یک خروجی جریان، کاهش قابل توجهی نسبت به مدلسازی با همین ساختار شبکه عصبی، اما با دو ورودی عمق و دور موتور داشته است.

دلیل این امر نیز ارتباط ضعیف تر و کم معناتر ولتاژ پک باتری با جریان، در مقایسه با عمق و دور موتور با جریان است که قبل تر در ابتدای تلاش اول از همین گام پروژه نیز آن را بررسی کرده بودیم.

تلash دوم(مدلسازی همراه تکنیک های کمکی)

از آنجا که در این گام از پژوهش، هدف افزایش هرچه بیشتر دقت تخمین زدن جریان است، در این تلش با کمک از روش ها، تکنیک ها و پارامترهای تکمیلی سعی بر تمرین دادن شبکه عصبی میکنیم که قادر به پیش‌بینی جریان، با دقت بالاتری نسبت به مدل قبلی باشد. در ادامه شرح کامل این مدلسازی همراه توضیح تکنیک ها و پارامترهای کمکی استفاده شده را خواهیم داشت.

پیش پردازش داده ها

همانند گام های قبلی، ابتدا در پژوهه نوت بوک جدید، به وارد کردن کتابخانه های ضروری پایتون میپردازیم. سپس دیتاست مرتب سازی شده اصلی را وارد کرده و با لیبل زدن، بخش ورودی (ولتاژ پک باتری، عمق و دور موتور) و بخش خروجی (جریان موتور) را جداسازی و در دیتافریم های پانداس قرار میدهیم.

```

مدل سه ورودی (تلash دوم)
وارد کردن کتابخانه های ضروری
[4]: import pandas as pd
import numpy as np
from sklearn.preprocessing import RobustScaler
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt

وارد کردن دیتاست بعنوان دنیاگیری پانداس
[8]: df = pd.read_csv("D:/Ijjourney/DataSets/Roshd Center/cleaned_dataset.csv")
#df.shape

جداسازی داده های ورودی و خروجی
[10]: X = df[['Power-Prop V', 'Depth1', 'Motor-FB']] # سه ورودی
Y = df[['Power-Prop I']] # یک خروجی

```

در ادامه، به مقیاس بندی داده های ورودی و خروجی بصورت جداگانه با مقیاس سازها (scalers) میپردازیم. اولین تغییر نسبت به مدلسازی تلش اول در مقیاس بندی به چشم میخورد. در این تلش بجای استفاده از مقیاس ساز MinMaxScaler از مقیاس ساز RobustScaler استفاده میکنیم که هر دو از کتابخانه sklearn فراخوانی میشوند.

تفاوت این دو مقیاس ساز:

- MinMaxScaler : داده ها را به محدوده ای مشخص (معمولأً بین صفر و یک) مقیاس بندی می کند. این روش از حداقل و حداکثر مقدار داده برای محاسبه استفاده می کند. به این دلیل اگر داده های پرت وجود داشته باشند، روی مقیاس بندی تأثیر زیادی می گذارند.

RobustScaler : از داده‌های میانه و فاصله بین چارک‌ها استفاده می‌کند. فاصله بین چارک‌ها به معنای اختلاف بین چارک اول (بیست‌وپنج درصد ابتدایی داده) و چارک سوم (هفتادوپنج درصد ابتدایی داده) است. این روش به دلیل استفاده از این شاخص‌های مقاوم، حساسیت کمتری به داده‌های پرت دارد و تأثیر آن‌ها را کاهش می‌دهد. بنابراین برای داده‌هایی که دارای مقدارهای پرت یا نویز زیاد هستند، مناسب‌تر است.

در دادمه نیز مقیاس بندی و تقسیم بندی داده‌ها به بخش تمرین و تست را، درست مشابه گام و مراحل قبلی داریم.

مقیاس بندی داده‌ها و تقسیم بندی

ایجاد شی مقیاس ساز

```
[13]: inputScaler = Robustscaler()
outputScaler = Robustscaler()
```

تنظیم مقیاس ساز‌ها و مقیاس بندی داده (نمودار)

```
[15]: X_norm = inputScaler.fit_transform(X)
Y_norm = outputScaler.fit_transform(Y)
```

جداسازی و تقسیم بندی داده‌های تمرین و تست

```
[21]: X_train, X_test, Y_train, Y_test = train_test_split(X_norm, Y_norm, test_size=0.2, random_state=42)
```

مدلسازی شبکه عصبی (+ تکنیک‌های کمکی)

در قسمت بعدی کد، کتابخانه و مولفه‌های مورد استفاده (تکنیک‌های تکمیلی) در شبکه عصبی را وارد پروژه می‌کنیم تا در ادامه، با کمک آنها به ساخت و تمرین دادن مدل شبکه عصبی دقیق‌تر از تلاش قبلی بررسیم.

ایجاد شبکه عصبی و تمرین دادن مدل

وارد کردن کتابخانه ها

```
[24]: import tensorflow as tf
from tensorflow.keras import layers, models, regularizers
from tensorflow.keras.optimizers import AdamW
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
```

در بخش بعدی از کد نوت بوک، پیاده سازی کامل ساختار شبکه عصبی و استفاده از تکنیک‌های کمکی که در ابتدای این گام پروژه به آن اشاره شده بود را داریم. در این شبکه عصبی، از ۵ لایه (یک ورودی – سه لایه پنهان – یک خروجی) و دقیقاً از همان تعداد نورن‌های تلاش قبلی استفاده شده است. در میان لایه‌ها و در ادامه، موقع ساخت شی از شبکه عصبی و کامپایل کردن آن، از پارامتر و تکنیک‌ها استفاده شده است که در ادامه به توضیح هر کدام از آن‌ها و علت انتخاب آنها می‌پردازیم.

```

ساختن شبکه عصبی با تابع
[34]: def create_model():
    inputs = tf.keras.Input(shape=X_train.shape[1,])

    layer_1 = layers.Dense(128, activation="leaky_relu", kernel_regularizer=regularizers.l2(0.001))(inputs)
    layer_1 = layers.Dropout(0.3)(layer_1) # Dropout for regularization

    layer_2 = layers.Dense(64, activation="leaky_relu", kernel_regularizer=regularizers.l2(0.001))(layer_1)
    layer_2 = layers.Dropout(0.3)(layer_2)

    layer_3 = layers.Dense(32, activation="leaky_relu")(layer_2)

    output = layers.Dense(1, name='current_output')(layer_3)

    model = models.Model(inputs=inputs, outputs=output)
    return model

model_NN = create_model()

model_NN.compile(optimizer=AdamW(learning_rate=0.001),
                  loss="mse",
                  metrics=["mse"])

```

- **تابع فعالسازی activation="leaky_relu"** : در تلاش قبلی از تابع ReLU استفاده کرده بودیم. درReLU اگر مقدار ورودی بزرگتر یا مساوی صفر باشد، همان مقدار ورودی را برمی‌گرداند؛ اما اگر کمتر از صفر باشد، مقدار خروجی صفر می‌شود. اما در تابع Leaky-ReLU اگر مقدار ورودی بزرگ‌تر یا مساوی صفر باشد، خروجی همان مقدار ورودی خواهد بود. اما اگر مقدار ورودی کمتر از صفر باشد، خروجی برابر است با ضرب آن مقدار در عددی کوچک، مانند 0.01 . این ویژگی کمک می‌کند تا حتی در مقادیر منفی، خروجی کاملاً صفر نشود و مقدار کوچکی ایجاد گردد. این روش باعث کاهش مشکل غیرفعال شدن برخی نورون‌ها در شبکه عصبی می‌شود.
- **تکنیک تنظیم کننده kernel_regularizer=regularizers.l2(0.001)** : گزینه kernel_regularizer تنظیم کننده‌ای برای وزن‌های مدل تعریف می‌کند که از پیچیدگی بیش از حد مدل جلوگیری می‌کند. این تنظیم کننده از روش تنظیم L2 یا همان لایه دو استفاده می‌کند که مقدار جریمه‌ای به تابع زیان مدل اضافه می‌کند. مقدار این جریمه برابر است با مجموع توان دوم وزن‌های شبکه ضربدر عدد 0.001 . این کار وزن‌ها را کوچک‌تر نگه داشته و از بیش‌برازش مدل جلوگیری می‌کند.
- **تکنیک دراپ اوت layers.Dropout(0.3)(layer_1)** : در این قطعه کد، از لایه‌ای به نام ریزش واحدها استفاده شده است. این لایه به مدل کمک می‌کند تا از بیش‌برازش جلوگیری کند. مقدار ۰.۳ نشان‌دهنده این است که سی درصد از نورون‌های لایه مربوطه (که در اینجا لایه پنهان اول است) به صورت تصادفی در طول آموزش مدل غیرفعال می‌شوند. به همین علت شبکه عصبی از انتکای بیش از حد بر تعداً خاصی از نورون‌ها جلوگیری می‌کند. این عمل باعث جلوگیری از آموزش بیش از حد (بیش‌برازش - overfitting) می‌شود که در تلاش اول از این گام با آن مواجه شده بودیم. در این شبکه عصبی، از دراپ اوت هم در لایه اول و هم در لایه دوم پنهان استفاده کرده ایم.
- **بهینه ساز optimizer=AdamW(learning_rate=0.001)**

در تلاش اول از بهینه ساز Adam استفاده کردیم. آدام ترکیبی از دو تکنیک میانگین‌گیری نمایی گرادیان‌ها (Momentum) و محاسبه نرخ یادگیری مقیاس شده (RMSProp) است. این روش سرعت و دقیقیت بهینه‌سازی را افزایش می‌دهد.

آدام دابلیو الگوریتم نسخه بهبودیافته آدام است که به صورت ویژه برای روش تنظیم لایه دو (L2) اصلاح شده است. در آدام دابلیو، جریمه لایه دو (مقدار خطا تخمین) مستقیماً به وزن‌ها اعمال می‌شود، نه به گرادیان‌ها. این امر باعث می‌شود که مدل تعیین‌دهی بهتری داشته باشد.

و اما در ادامه کد از دو تکنیک کمکی باقی مانده استفاده می‌کنیم که آن دو نیز با هدف یادگیری هرچه بیشتر شبکه عصبی اعمال می‌شوند. سپس فرایند تمرین دادن مدل شبکه عصبی با دادن داده‌های تمرین به آن را خواهیم داشت.

```
اضافه کردن توقف زودهنگام و تنظیم کننده نرخ یادگیری
[20]: early_stop = EarlyStopping(monitors='val_loss', patience=5, restore_best_weights=True)
lr_scheduler = ReduceLROnPlateau(monitors='val_loss', factor=0.5, patience=3, min_lr=1e-5)

تمرین دادن مدل
[22]: history = model_NN.fit(X_train, Y_train, epochs=100, batch_size=32, validation_split=0.2,
                           callbacks=[early_stop, lr_scheduler])
```

• تکنیک توقف زودهنگام () : EarlyStopping()

این تکنیک آموزش مدل را زمانی متوقف می‌کند که عملکرد مدل روی داده‌های اعتبارسنجی (مانند کاهش خطای داده‌های اعتبارسنجی – val_loss) دیگر بهبود نیابد. هدف از این تکنیک نیز جلوگیری از بیش برآش مدل است.

- monitor='val_loss' : ناظارت بر میزان خطای اعتبارسنجی را بر عهده می‌گیرد
- patience=5 : به معنای صبر کردن برای 5 دور تکرار یا همان epoch متوالی است که اگر در آنها، خطای شبکه بهبودی نیابد، فرآیند تمرین دادن را متوقف می‌سازد

• تنظیم کننده نرخ یادگیری () : ReduceLROnPlateau()

این تکنیک، در موقعی که در چند دور تکرار، خطای اعتبارسنجی مدل در روند تمرین داده شدن بهبودی نیابد، نرخ یادگیری را کاهش میدهد تا باعث یادگیری بهتر، و بهبود تعیین‌دهی مدل بشود.

- monitor='val_loss' : ناظارت بر میزان خطای اعتبارسنجی را بر عهده می‌گیرد.
- factor=0.5 : کاهش نرخ یادگیری به نصف مقدار فعلی در هر بار اعمال شدن.
- patience=3 : تعداد دور تکرار های متوالی که قبل از اعمال شدن صبر می‌کند.
- min_lr=1e-5 : حداقل کف نرخ یادگیری که می‌تواند به آن کاهش نیابد. بعد از رسیدن به آن دیگر نرخ یادگیری کاهش نمی‌ابد. در اینجا برابر 0.00001 است.

مشاهده دقیق تر:

کاهش نرخ یادگیری در دور 23

تمرین دادن مدل

```
[40]: history = model_NN.fit(X_train, Y_train, epochs=100, batch_size=32, validation_split=0.2,
                           callbacks=[early_stop, lr_scheduler])
```

335/335 2s 7ms/step - loss: 0.0191 - mse: 0.0153 - val_loss: 0.0159 - val_mse: 0.0121 - learning_rate: 0.0010
Epoch 18/100
335/335 2s 7ms/step - loss: 0.0186 - mse: 0.0149 - val_loss: 0.0178 - val_mse: 0.0141 - learning_rate: 0.0010
Epoch 19/100
335/335 2s 7ms/step - loss: 0.0190 - mse: 0.0154 - val_loss: 0.0157 - val_mse: 0.0121 - learning_rate: 0.0010
Epoch 20/100
335/335 2s 7ms/step - loss: 0.0183 - mse: 0.0148 - val_loss: 0.0170 - val_mse: 0.0135 - learning_rate: 0.0010
Epoch 21/100
335/335 2s 7ms/step - loss: 0.0189 - mse: 0.0154 - val_loss: 0.0160 - val_mse: 0.0124 - learning_rate: 0.0010
Epoch 22/100
335/335 2s 7ms/step - loss: 0.0177 - mse: 0.0142 - val_loss: 0.0159 - val_mse: 0.0124 - learning_rate: 0.0010
Epoch 23/100
335/335 2s 7ms/step - loss: 0.0173 - mse: 0.0139 - val_loss: 0.0151 - val_mse: 0.0119 - learning_rate: 5.0000e-04
Epoch 24/100
335/335 2s 7ms/step - loss: 0.0174 - mse: 0.0142 - val_loss: 0.0148 - val_mse: 0.0118 - learning_rate: 5.0000e-04
Epoch 25/100
335/335 2s 7ms/step - loss: 0.0166 - mse: 0.0136 - val_loss: 0.0146 - val_mse: 0.0115 - learning_rate: 5.0000e-04
Epoch 26/100

پایان دادن زودهنگام به روند مدلسازی در دور 92 توسط

(قرار بر 100 دور بود) EarlyStopping

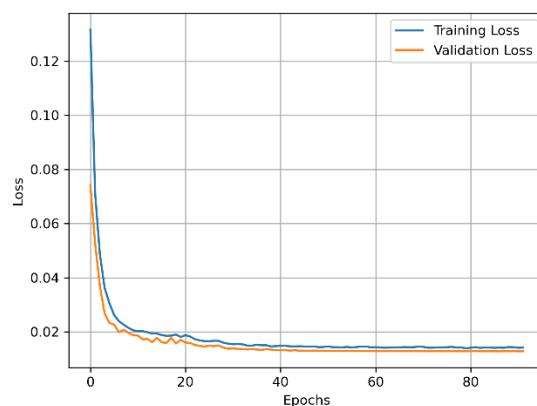
تمرین دادن مدل

```
[40]: history = model_NN.fit(X_train, Y_train, epochs=100, batch_size=32, validation_split=0.2,
                           callbacks=[early_stop, lr_scheduler])
```

Epoch 84/100
335/335 2s 6ms/step - loss: 0.0118 - mse: 0.0118 - val_loss: 0.0128 - val_mse: 0.0105 - learning_rate: 1.0000e-05
Epoch 85/100
335/335 2s 7ms/step - loss: 0.0138 - mse: 0.0115 - val_loss: 0.0128 - val_mse: 0.0106 - learning_rate: 1.0000e-05
Epoch 86/100
335/335 2s 7ms/step - loss: 0.0135 - mse: 0.0113 - val_loss: 0.0128 - val_mse: 0.0105 - learning_rate: 1.0000e-05
Epoch 87/100
335/335 2s 6ms/step - loss: 0.0140 - mse: 0.0118 - val_loss: 0.0128 - val_mse: 0.0105 - learning_rate: 1.0000e-05
Epoch 88/100
335/335 2s 7ms/step - loss: 0.0138 - mse: 0.0116 - val_loss: 0.0128 - val_mse: 0.0106 - learning_rate: 1.0000e-05
Epoch 89/100
335/335 2s 7ms/step - loss: 0.0145 - mse: 0.0123 - val_loss: 0.0128 - val_mse: 0.0106 - learning_rate: 1.0000e-05
Epoch 90/100
335/335 2s 7ms/step - loss: 0.0144 - mse: 0.0121 - val_loss: 0.0128 - val_mse: 0.0106 - learning_rate: 1.0000e-05
Epoch 91/100
335/335 2s 7ms/step - loss: 0.0141 - mse: 0.0118 - val_loss: 0.0128 - val_mse: 0.0105 - learning_rate: 1.0000e-05
Epoch 92/100
335/335 2s 7ms/step - loss: 0.0140 - mse: 0.0117 - val_loss: 0.0128 - val_mse: 0.0106 - learning_rate: 1.0000e-05

نمودار تغییرات خطای اعتبار سنجی در فرایند تمرین دادن

(رونمایی ملایم تر نسبت به تلاش اول – روش بدون تکنیک)



ارزیابی با دیتاست اصلی

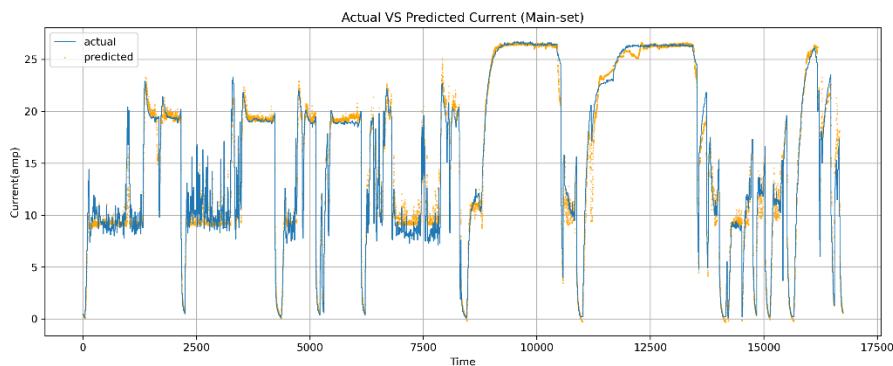
در بخش بعدی از تلاش دوم و بعد از اتمام فرایند مدلسازی، به بررسی دقیق مدل با داده های جدا شده برای تست (از دیتاست اصلی) میپردازیم. کد این قسمت نیز کاملا مشابه کد گام های قبلی است.

بعد از ارزیابی با نرخ R2 score، متوجه میشویم که مدل تمرین داده شده به دقیقیت بالا 0.96 در بخش تست، و 0.97 در پیش‌بینی کل دیتاست اصلی رسیده است که مقدار قابل قبولی است.

```
ازریابی عملکرد و دقیقیت مدل در پیش‌بینی
[48]: from sklearn.metrics import r2_score
ازریابی با داده تست
[54]: prediction1 = model_NN.predict(X_test)
evalCurrent = r2_score(Y_test, prediction1)
print('Current prediction accuracy:', evalCurrent)
105/105 ━━━━━━━━━━ 1s 9ms/step
Current prediction accuracy: 0.9680652423614775
ازریابی با کل دیتاست
[58]: predTotal = model_NN.predict(X_norm)
evalCurrentTotal = r2_score(Y_norm, predTotal)
print('Current prediction accuracy(whole dataset):', evalCurrentTotal)
524/524 ━━━━━━━━━━ 1s 3ms/step
Current prediction accuracy(whole dataset): 0.9718353908460392
```

برای درک بهتر دقیقیت مدل در دیتاست اصلی، به ضریب تعیین (R2 score) اتکا نکرده و با مصورسازی نمودار جریان تخمین زده در مقابل جریان واقعی، این کار را انجام میدهیم.

```
نمودار جریان واقعی و پیش‌بینی شده در دیتاست اصلی
[127]: detormpred = outputScaler.inverse_transform(predTotal)
timeline = np.arange(0, 16/44, 1)
plt.figure(figsize=(12,5))
plt.plot(timeline, df['Power-Prop I'], label="actual", linewidth=0.8)
plt.scatter(timeline, detormpred, alpha=0.7, s=2, label="predicted", color="orange")
plt.title("Actual VS Predicted Current (Main-set)")
plt.xlabel("Current (amp)")
plt.ylabel("Time")
plt.grid(True)
plt.legend()
plt.tight_layout()
plt.savefig("D:/AIjourney/newPics/Phase3/attempt1/Plots/currentMainset.png", format="png", dpi=300)
plt.show()
```



از نمودارها میتوان دریافت که مدل توانایی بالایی در پیش‌بینی جریان، با دیتاست اصلی دارد و روابط بین پارامترها را به خوبی یادگرفته است.

با بررسی نمودار، اطمینان بیشتری از بهبود دقیق مدل نسبت به تلاش اول بدست می‌آوریم. در بخش هایی از نمودار تلاش اول که اختلاف جریان واقعی و جریان تخمینی زیاد بوده است، در تلاش دوم کاهش یافته یا برطرف شده اند، و این امر کاملاً مشهود است.

ذخیره فایل ها

با اتمام تلاش دوم از گام سوم پروژه، به ذخیره سازی فایل های پروژه از جمله شبکه عصبی، و مقیاس سازها میپردازیم. برای ذخیره کردن شبکه عصبی از توابع موجود در کتابخانه tensorflow استفاده میکنیم و برای ذخیره کردن مقیاس سازها، باید کتابخانه joblib را به پروژه وارد کنیم.

```
ذخیره سازی مدل و مقیاس سازها
[103]: import joblib
ذخیره کردن مقیاس سازها
[107]: joblib.dump(inputscaler, "D:/AIjourney/HN_Models/Phase3/attempts/scalers/inputscaler.pkl")
joblib.dump(outputscaler, "D:/AIjourney/HN_Models/Phase3/attempts/scalers/outputScaler.pkl")
[107]: ['D:/AIjourney/HN_Models/Phase3/attempts/scalers/outputScaler.pkl']

ذخیره کردن شبکه عصبی
h5
ذخیره سازی با فرمت
[109]: model_NN.save("D:/AIjourney/HN_Models/Phase3/attempts/BLDC_Neural_Network_Model.h5")
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
ذخیره سازی با فرمت
[117]: model_NN.save("D:/AIjourney/HN_Models/Phase3/attempts/BLDC_Neural_Network_Model.keras")
```

جزئیات ذخیره سازی

- مدل شبکه عصبی : به دو فرمت keras. که بیشتر مناسب نسخه های جدید تر کتابخانه tensorflow.h5 هست، و همچنین فرمت مرسوم keras. که بصورت گسترده مورد استفاده قرار میگیرد و سازگاری بیشتری با اکثر فریمورک ها دارد. لازم به ذکر است هر دو روش تمامی ساختار شبکه عصبی، پیکربندی روند تمرین دادن، وضعیت بهینه ساز و... را ذخیره میکنند.
- مقیاس سازها: در ادامه هر دو مقیاس ساز ورودی و خروجی تنظیم شده و سازگار با داده های موتور را، با فرمت مرسوم pkl. ذخیره سازی میکنیم.

تلash سوم(تغییر داده های ولتاژ)

همانطور که مشاهده شد، در تلash دوم از گام سوم به دقت 90 درصد، در تخمین زدن جریان موتور توسط مدل شبکه عصبی رسیدیم که این امر با بکارگیری تکنیک ها و روش های تکمیلی محقق شد. در ادامه و در تلash سوم، سعی میکنیم تا با اعمال روش های جدید، دقت مدل در پیش بینی کردن جریان را به بالای 90 درصد برسانیم.

تکنیک استفاده شده در این تلash، تغییر دادن داده های پارامتر ولتاژ با هدف برطرف کردن رفتار پله ای است که مانع افزایش تعمیم پذیری و دقت مدل میباشد. از آنجا که رزولوشن نمونه برداری داده های ولتاژ در داده های لاغ، کمتر از پارامترهای دیگر همچون جریان، دور موتور و عمق است(عامل رفتار پله ای)، در این تلash یا استفاده از دو تکنیک میانگین متحرک و اضافه کردن نویز کنترل شده به داده های ولتاژ، روند پله ای این پارامتر را رفع کرده و به مدل شبکه عصبی، قابلیت بیشتری در درک بهتر روابط بین پارامترها بددهیم.

روند کلی این تلash مانند تلash های قبلی از این گام است و تنها موارد اضافه شده و مورد بررسی مطابق موارد ذیل است:

- اعمال میانگین متحرک بر پارامتر ولتاژ
- اعمال نویز کنترل شده بر ولتاژ

که بعد از بررسی و شرح کامل نحوه اعمال این دو تکنیک و همچنین بررسی تاثیر آنها در داده ها، به مصورسازی و ارزیابی دقت خواهیم پرداخت.

اعمال تغییرات در داده های ولتاژ

هر دو تغییر اعمال شده در این تلash، در بخش زیر که به کد اضافه شده، اعمال گردیده است. در این قطعه کد پارامتر جدیدی به دیتابست (که در قالب دیتابریم وارد پروژه شده است) اضافه میکنیم که دقیقاً نسخه تغییر یافته همان پارامتر ولتاژ قبلی (Power-Prop V_smoothed)، با نام جدید Power-Prop V است. در ادامه هر دو تکنیک اعمال شده را با جزئیات بیشتر شرح داده، و تغییرات آن نسبت به پارامتر ولتاژ عادی را مشاهده خواهیم کرد.

ایجاد تغییرات در داده های ولتاژ

```
: df['Power-Prop V_smoothed'] = df['Power-Prop V'].rolling(window=300, center=True).mean()
df['Power-Prop V_smoothed'].fillna(df['Power-Prop V'], inplace=True)
df['Power-Prop V_smoothed'] = df['Power-Prop V_smoothed'] + np.random.normal(0, 0.0001, size=len(df))
```

• میانگین متحرک (Rolling(window=300, center=True)) :

این تکنیک برای کاهش نوسانات ناگهانی داده‌ها استفاده می‌شود و باعث می‌شود داده‌های خام (در اینجا رفتار پله‌ای) به شکل صاف‌تر و قابل تحلیل‌تر در بیایند. اینتابع یک بازه (پنجره) به اندازه 300 داده (در اطراف داده‌ی انتخاب شده فعلی) را انتخاب و به ترتیب به جلو حرکت می‌کند. در هر بار، میانگین 300 داده اطراف داده‌ی انتخاب شده را محاسبه، و جایگزین مقدار فعلی داده می‌کند. بدیهی است هرچقدر عدد بازه یا پنجره بزرگ‌تر باشد، روند هموارتر و نرم‌تر می‌شود، اما افزایش بیش از حد آن منجر به از دست دادن جزئیات پارامتر، برای تمرین دادن مدل می‌شود در این قسمت با آزمون و خطاهای انجام شده، درنهایت به مقدار مناسب در محدوده 300 الی 400 برای بازه (پنجره) رسیدیم.

▪ window=300 : تعداد نقاط داده‌ای که در هر بازه برای محاسبه میانگین در نظر گرفته

می‌شود (در اینجا ۳۰۰ مقدار)

▪ center=True : میانگین‌گیری به گونه‌ای انجام می‌شود که مقدار محاسبه‌شده در مرکز بازه قرار بگیرد، نه در انتهای آن (داده‌ی انتخاب شده در مرکز بازه باشد).

• پرکردن خلاه‌های داده تغییر یافته (fillna()) :

این مرحله برای جلوگیری از ایجاد مقادیر خالی در داده‌های پردازش شده انجام می‌شود. به اینصورت که جاهای خالی ایجاد شده توسط تکنیک قبلی (300 داده اول و 300 داده انتهایی داده‌های ولتاژ) را با آخرین مقادیر معتبر قبلی (ولتاژهای تغییر نیافته) پر می‌کند.

▪ fillna(df['Power-Prop V'], inplace=True) : مقادیر نامعتبر تولید شده توسط

میانگین‌گیری متحرک با مقادیر اولیه ستون جایگزین می‌شود

▪ inplace=True : تغییرات مستقیماً روی همان DataFrame اعمال می‌شوند و نیازی به ذخیره نسخه جدید نیست.

• افزودن نویز تصادفی (np.random.normal(0, 0.0001, size=len(df))) :

برای جلوگیری از ایجاد یک الگوی بیش از حد صاف و غیرواقعي، نویز تصادفی افزوده می‌شود. در این تکنیک از توزیع نرمال (گوسی) با میانگین صفر و انحراف معیار 0.0001 برای ایجاد آرایه‌ای از اعداد تصادفی، هم اندازه و هم بعد با داده‌های ولتاژ استفاده می‌شود، و در ادامه این مقادیر بصورت ماتریسی با داده‌های ولتاژ تغییر یافته جمع زده می‌شوند.

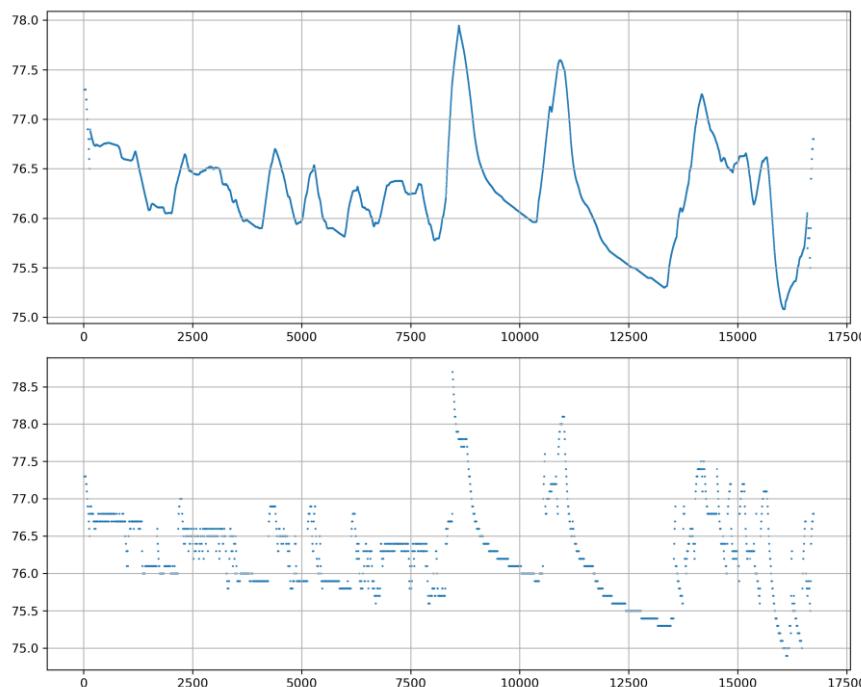
▪ (0, 0.0001) : نویز تصادفی با میانگین صفر و انحراف معیار 0.0001 به داده‌های هموارشده اضافه می‌شود

▪ size=len(df) : نویز به اندازه تعداد سطرهای DataFrame تولید شده و به مقدار هموارشده اضافه می‌شود.

داده های ولتاژ قبل و بعد از تغییرات

در ادامه با کمک از مصورسازی داده ها با کتابخانه matplotlib و همچنین نمایش نمونه داده های دیتافریم، دید بهتری نسبت به تغییرات اعمال شده بر روی داده های ولتاژ پیدا خواهیم کرد.

ولتاژ تغییر یافته(نمودار بالا)



همانطور که قابل مشاهده است روند پله ای و تغییرات ناگهانی به علت رزو لوشن پایین نمونه برداری، تا حد بسیار زیادی رفع شده است که این امر کمک بهتری به مدل شبکه عصبی، در درک روابط پارامترها خواهد کرد. در ادامه تغییرات ایجاد شده در داده ها را این بار با دیتافریم بررسی می کنیم.

تغییرات ایجاد شده در دیتاست								
[125]:	df.iloc[8210:8220]							
[125]:	Date	Time	DataName	Motor-FB	Depth1	Power-Prop V	Power-Prop I	Power-Prop V_smoothed
8210		05:20:7.7		752	197	75.7	18.9	75.954787
8211		05:20:8.8		748	198	75.7	18.9	75.958097
8212		05:21:0.0		747	196	75.7	18.9	75.961260
8213		05:21:1.1		747	195	75.7	18.9	75.964699
8214		05:21:2.2		743	192	75.7	18.9	75.968035
8215		05:21:3.3		746	192	75.8	18.5	75.971303
8216		05:21:4.4		740	189	75.8	18.5	75.974764
8217		05:21:5.5		742	185	75.8	18.5	75.977823
8218		05:21:6.6		738	181	75.8	18.5	75.981382
8219		05:21:7.7		743	178	75.8	18.5	75.984426

تغییرات مستمر داده های ولتاژ بعد از تغییرات اعمال شده، نسبت به تغییرات پله ای این پارامتر قبل از تغییرات ملموس است. (ستون Power-Prop V و Power-Prop V_smoothed)

مدلسازی شبکه عصبی (استفاده از داده ولتاژ تغییر یافته)

در ادامه تلاش سوم، به مدلسازی شبکه عصبی جدید میپردازیم که تنها تفاوت آن با تلاش قبلی، استفاده از داده ولتاژ تغییر یافته، در کنار عمق و دور موتور عنوان ورودی میباشد.

جداسازی داده های ورودی و خروجی

```
[256]: X = df[['Power-Prop V_smoothed', 'Depth1', 'Motor-FB']] # سه ورودی(ولتاژ تغییر یافته)
Y = df[['Power-Prop I']] # یک خروجی
```

ارزیابی دقیقت مدل

بعد از اتمام فرایند تمرین دادن مدل شبکه عصبی جدید، به ارزیابی دقیقت آن در دیتاست اصلی، و سپس دیتاست های ثانویه میپردازیم.

ارزیابی با داده تست

```
[280]: prediction1 = model_NN.predict(X_test)
evalCurrent = r2_score(Y_test, prediction1)
print('Current prediction accuracy:', evalCurrent)
105/105 ━━━━━━━━ 1s 5ms/step
Current prediction accuracy: 0.9401710406791182
ارزیابی با کل دیتاست
[282]: predTotal = model_NN.predict(X_norm)
evalCurrentTotal = r2_score(Y_norm, predTotal)
print('Current prediction accuracy(whole dataset):', evalCurrentTotal)
524/524 ━━━━━━━━ 1s 3ms/step
Current prediction accuracy(whole dataset): 0.9480315376164286
```

دقیقت مدل برای تخمین زدن جریان موتور در دیتاست اصلی بصورت زیر است:

- دقیقت در بخش تست : 94
- دقیقت در کل دیتاست : 94.8

محاسبه و ارزیابی دقیقت پیش‌بینی بر روی هر سه دیتاست ثانویه

```
[297]: print('Current accuracy test1:', r2_score(test1_Y['Power-Prop I'], predTest1_deNorm))
print('\n')
print('Current accuracy test2:', r2_score(test2_Y['Power-Prop I'], predTest2_deNorm))
print('\n')
print('Current accuracy test3:', r2_score(test3_Y['Power-Prop I'], predTest3_deNorm))
Current accuracy test1: 0.9311883351997174
Current accuracy test2: 0.9272663695111679
Current accuracy test3: 0.959105746913222
```

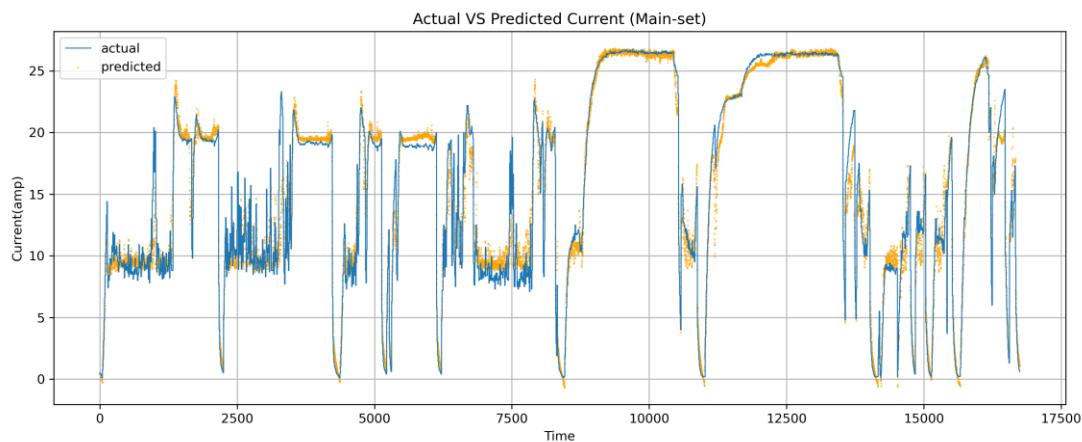
دقیقت مدل برای پیش‌بینی جریان خروجی، در دیتاست های ثانوی نیز مطابق زیر است:

- دیتاست اول : 93.1
- دیتاست دوم : 92.7
- دیتاست سوم : 95.9

نتایج بالا میانگین خوب 93.9 در ارزیابی با نرخ R2 score را نشان میدهد که افزایش بیش از 3 درصدی نسبت به مدل تلاش دوم (90.6) را نشان میدهد.

تصویرسازی نتایج پیش‌بینی

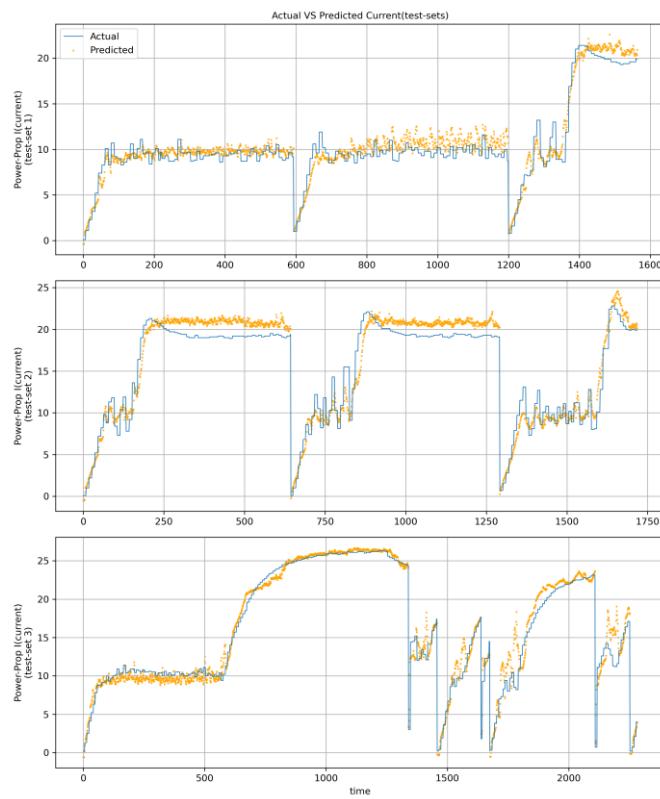
در ادامه به تصویرسازی و رسم نمودار برای مقایسه بهتر جریان پیش‌بینی شده توسط مدل جدید، و مدل قبلی مبادرایم تا از موثر بودن تغییرات اعمال شده، اطمینان بیشتری حاصل کنیم.



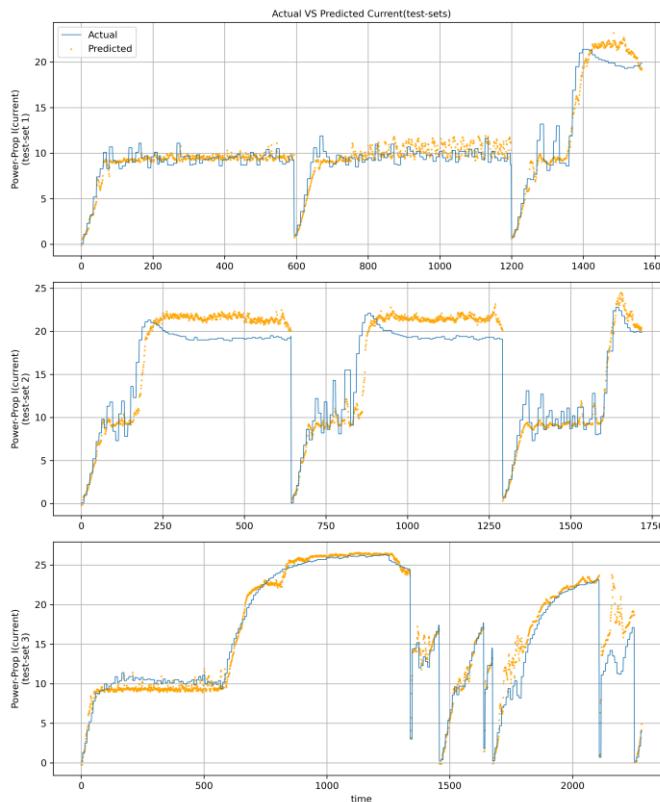
در دیتاست اصلی، تفاوت مقادیر تخمین زده شده و واقعی جریان، تنها افزایش کمی نسبت به مدل تلاش دوم داشته است و بصورت کلی همچنان توانایی بالای مدل در تخمین زدن جریان، با دیتاست اصلی را نشان میدهد.

در ادامه به تصویرسازی نتایج تخمین داده های جریان، با استفاده از دیتاست های ثانویه پرداخته و مدل تلاش دوم و سوم را با یکدیگر مقایسه میکنیم.

نمودار مدل تلاش سوم(جدید)



نمودار مدل تلاش دوم(قبلی)



با بررسی نمودارها، بهبود در تعمیم پذیری و دقت برای مدل تلاش سوم کاملا مشهود است و با اطمینان قوی تری، میتوانیم تاثیرگذار بودن تغییرات اعمال شده بر داده‌های ولتاژ را نتیجه گیری کنیم.

نگاهی به عملکرد مدل‌ها (در تخمین زدن جریان)

	نوع مدل	مقیاس ساز	تکنیک های تکمیلی	دیتاست اصلی	دیتاست 1 تست	دیتاست 2 تست	دیتاست 3 تست	دیتاست تست (میانگین)
گام اول (رگرسیون)	رگرسیون	X	X	0.59	X	X	X	X
گام اول (شبکه عصبی عصبی)	شبکه عصبی	X	X	0.91	0.87	0.79	0.94	0.866
گام دوم	شبکه عصبی	✓	X	0.94	0.89	0.92	0.95	0.92
گام سوم (تلاش اول)	شبکه عصبی	✓	X	0.97	0.86	0.70	0.93	0.83
گام سوم (تلاش دوم)	شبکه عصبی	✓	✓	0.96	0.92	0.86	0.94	0.906
گام سوم (تلاش سوم)	شبکه عصبی	✓	✓	0.94	0.93	0.92	0.96	0.939