بسم الله الرحمن الرحیم

محمد عرفان زارع زردینی

تمرین سری دوم هوش مصنوعی

1)

ابتدا کتابخانه های مورد نیاز جهت محاسبات ریاضی و کشیدن نمودار اضافه می‌کنیم که شامل کتابخانه‌های numpy، matplotlib می شود.

برای پیاده سازی شبکه RBF و بدست آوردن توزیع مرکز ها ،از کتابخانه sklearn استفاده شده است .

پرسپترون چند لایه پیاده سازی می شود پس باید کراس را نیز اضافه کرد.

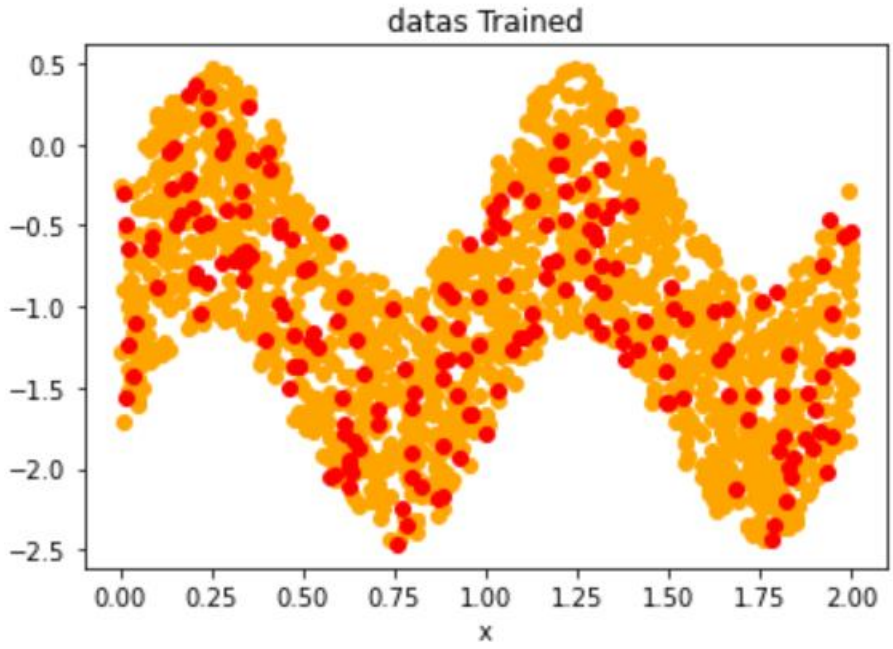
تابعی برای تولید داده‌ها به‌صورت تصادفی نوشتیم . تابع به تعداد مشخص شده در ورودی طبق فرمول صورت سوال نقاط را تولید می کند.ما 1500 داده جهت تمرین و 200 داده جهت تست می سازیم.

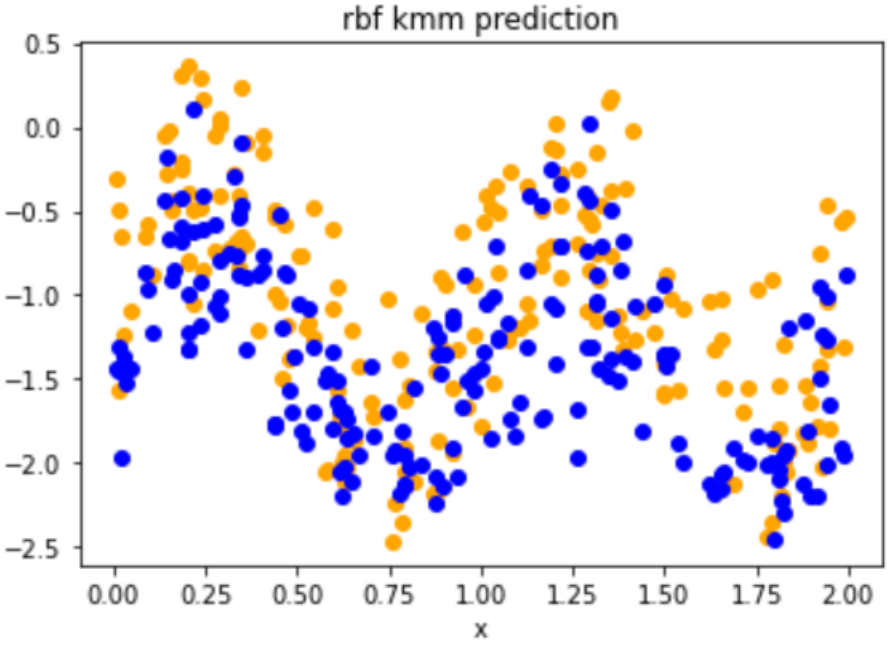
برای مشاهده نقاط ، در نمودار رسم می‌کنیم. که داده‌های آموزش و داده‌های تست با رنگ های متفاوت مشخص شده اند .

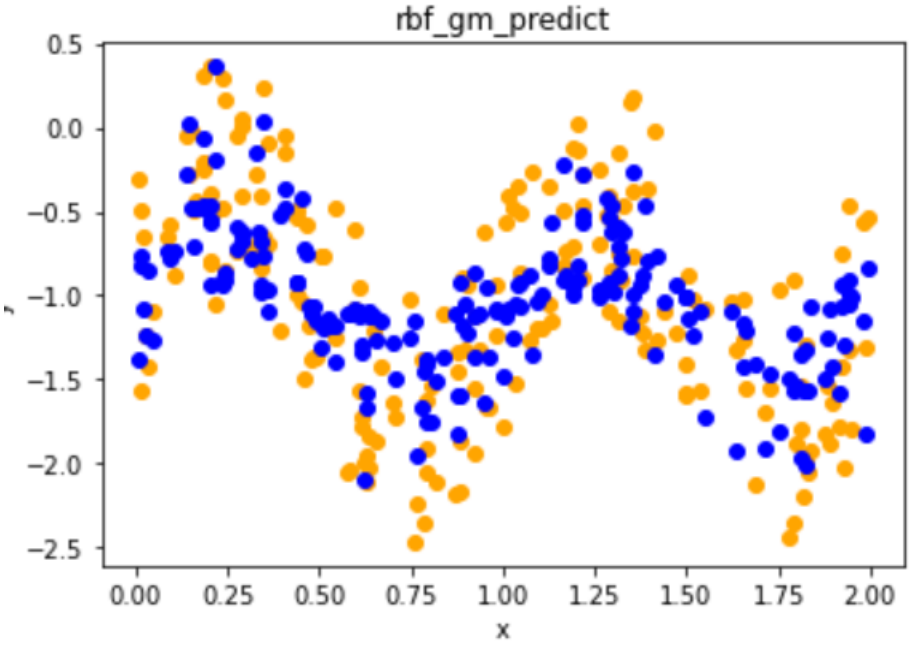
حال شبکه rbf را پیاده می کنیم. تعداد ورودی‌ها و تعداد خوشه‌های شبکه را تعریف می کنیم. در ادامه شعاع همسایگی را برابر هفت‌صدم قرار می‌دهیم و epoch را برابر 300 می گذاریم.

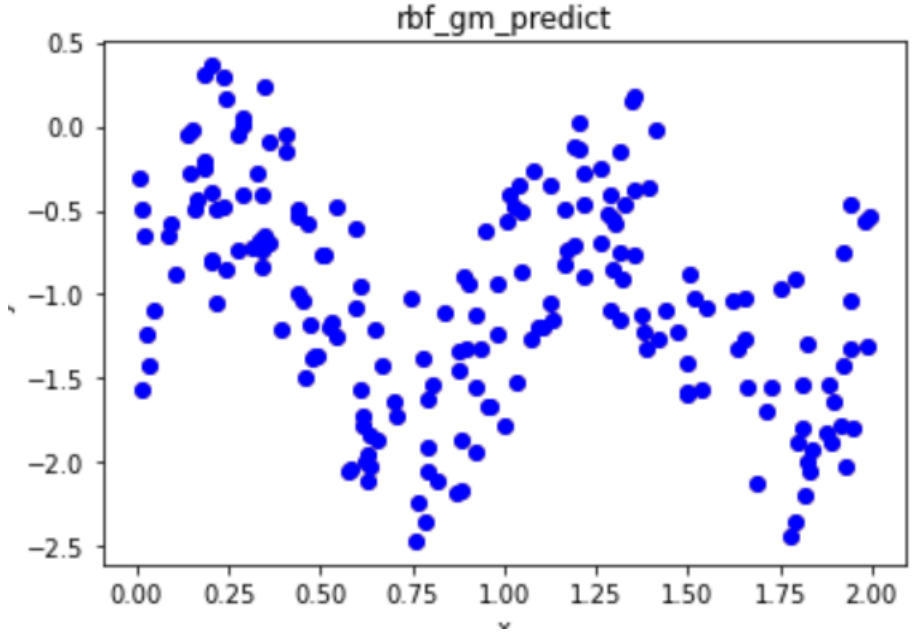
در بخش mlpما از یک مدل با معماری mse loss استفاده کرده تا روی داده های تولیدی آموزش ببیند.

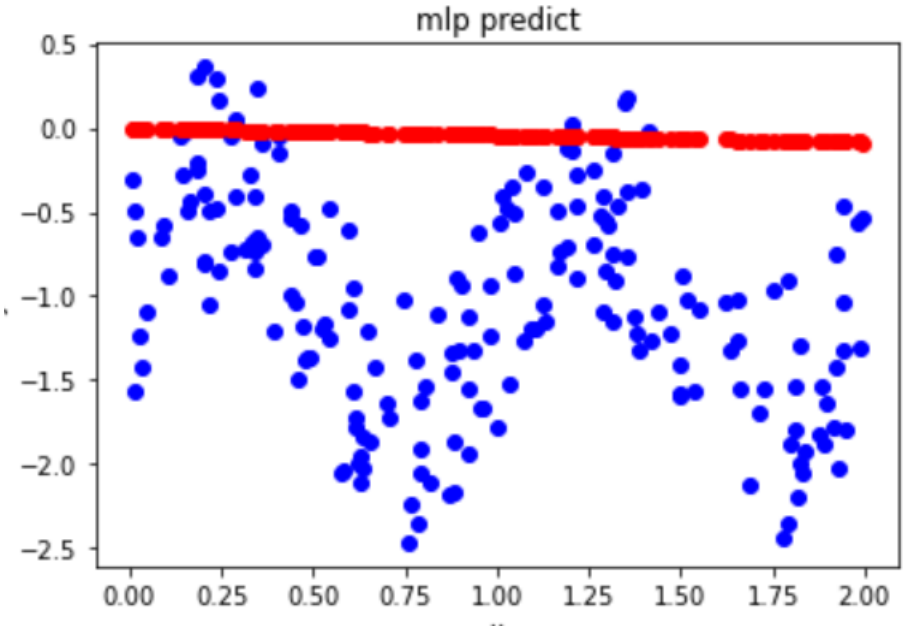
در rbf ما برای پیاده سازی شبکه از numpy و sklearn استفاده کردیم. برای پیدا کردن مرکز ها نیز از الگوریتم KMeans یا GMM استفاده شدهشکل ها به همراه عنوانشان در زیر موجود اند.






rbf_gm_predict
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−2.5
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
x


rbf_gm_predict
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−2.5
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00



مقایسه و تحلیل:

داده تولیدی مان به سبب μ نویز دارد. با توجه به نمودار ها، MLPنویز را یاد نمی گیرد و تنها تابع سینوسی را آموخته است .شبکه RBF اما بسبب استفاده از توابع Radial می تواند نویز را یاد بگیرد. همچنین بسبب وجود نویز، طول می کشد تا MLP تابع را یاد بگیرد اما این مشکل در RBF وجود ندارد و آموزش سریعتر می باشد.

2)

Year: Month: Date: Sa Su Mo Tu We Th Fr Subject:

سؤال ۱)

در ابتدا بیست ولی توان ذخیره کرد تا شکل ذخیره سازی reverse در شبکه HoPfield وجود دارد و ترتیب ها به

صورت دیگالت ذخیره می شود که ببینیم ولی اگر ۱ یک هست می توان از state های [1,1,1,1] و

[1,1,-1,-1] هر یک حالت را ذخیره شود حال به سراغ متن ماتریس وزن می دهیم.

۳ پترن هست که ما ماتریس وزن هر یک را جداگانه محاسبه می کنیم و سپس ماتریس ها را جمع می زنیم

$$w_{ij}^{P_i} = x_i^{P_i} x_j^{P_i}$$

$P_1 = (1,1,1,1)$

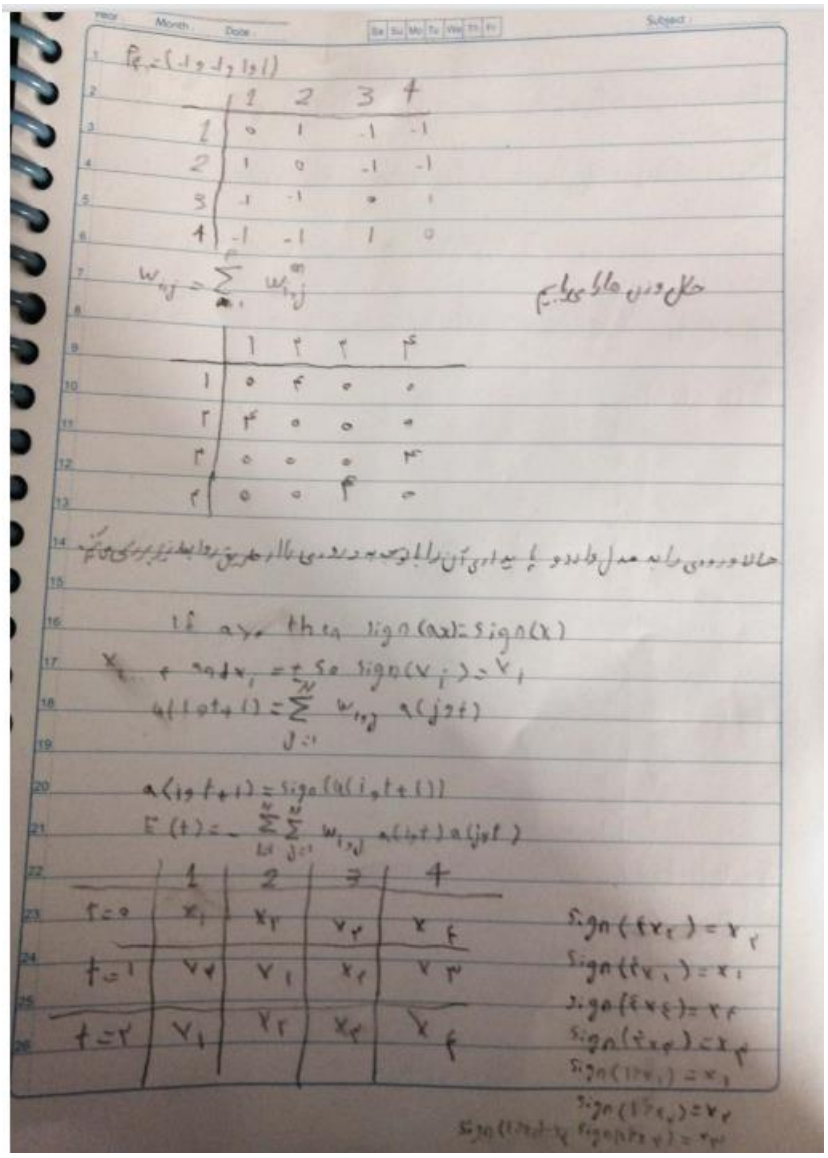
	1	2	3	4
1	0	1	1	1
2	1	0	1	1
3	1	1	0	1
4	1	1	1	0

$P_2 = (-1,-1,-1,-1)$

	1	2	3	4
1	0	1	1	1
2	1	0	1	1
3	1	1	0	1
4	1	1	1	0

$P_3 = (1,1,-1,1)$

	1	2	3	4
1	0	1	-1	-1
2	1	0	-1	-1
3	-1	-1	0	1
4	-1	-1	1	0



3)

باید از som یک بعدی استفاده کنیم و برای دقیق شدن مدل ها، از kohonen استفاده می نماییم. برای kohonen، بخش initرا حالت عادی انجام می دهیم . ورودی آن شهرها می باشد. شبکه ی kohonen همه شهر هارو بهم وصل می کند .این روش با cluster قسمت بندی می نماید و خروجی این روش، شبکه یک بعدی هستش که به شکل خودکار، جواب مسئله را خروجی می دهد.

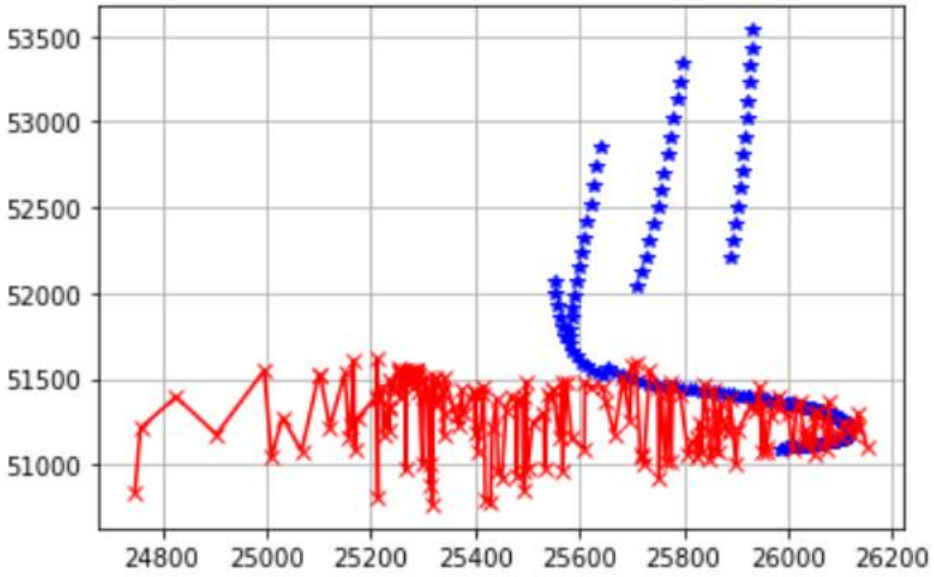
4)

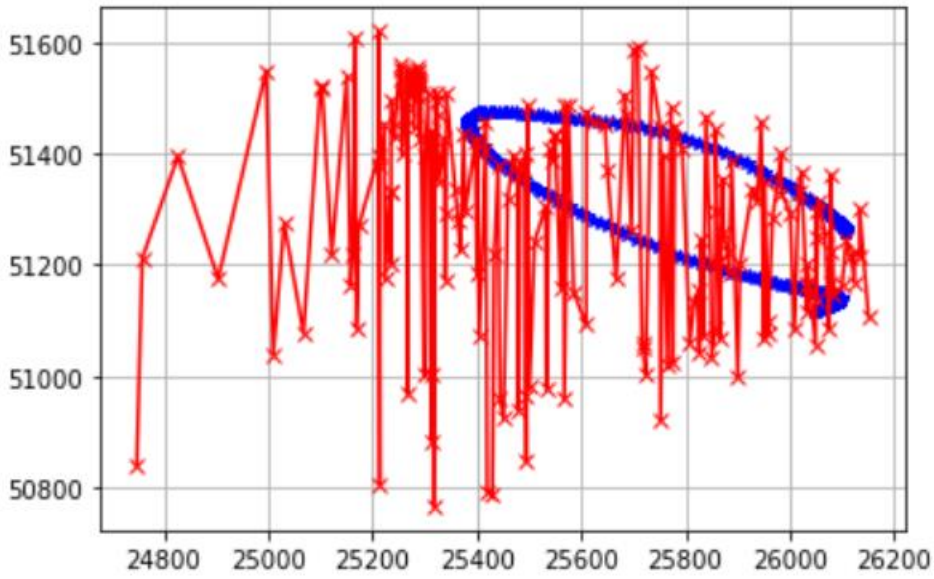
در حل سوال از library های numpy برای محاسبات ماتریس و آرایه ها به صورت هوشمند استفاده می شود. برای رسم نمودار ها از matplotlib و csv برای خواندن دیتا های ورودی از فایل استفاده می شود. تابع ans_plot برای رسم پاسخ ها و سیر مسیر جواب می باشد. تابع derive_city برای خواندن داده ها و ریختنشون در یک آرایه استفاده می شود. در تابع algorithm_kohonen هم الگوریتم مدنظرمان پیاده سازی شده.تابع __init__ برای هر object که می سازد یک سری فیلد هارا تعریف میکند که فیلد ها : n – 1 که متغییر کمکی برای ساخت سیگما در تابع همسایگی میباشد2– radius شعاع همسایگی است a هم نورون های map شده در مدل kohonen میباشد.

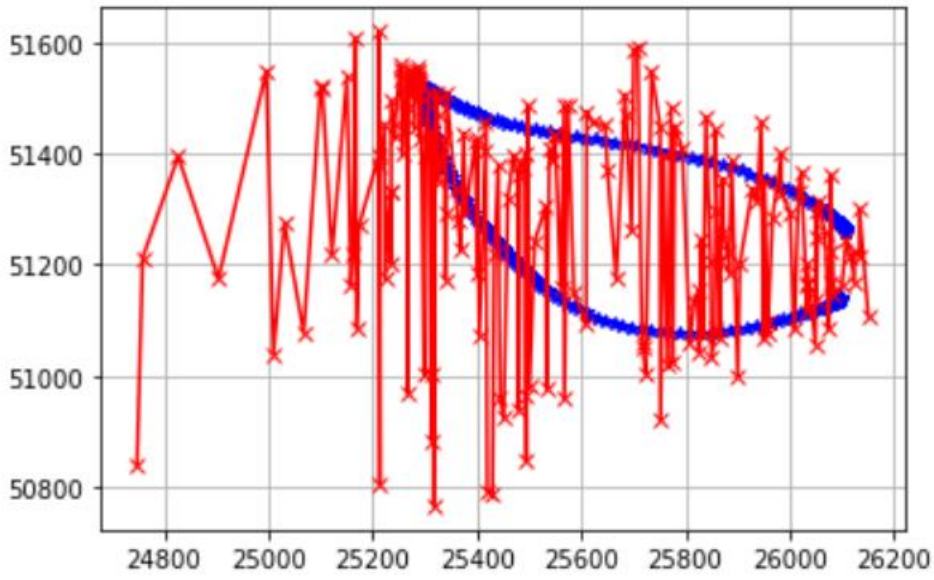
حال به داخل تابع optimize میرویم که پیاده سازی الگوریتم خود kohonen میباشد.

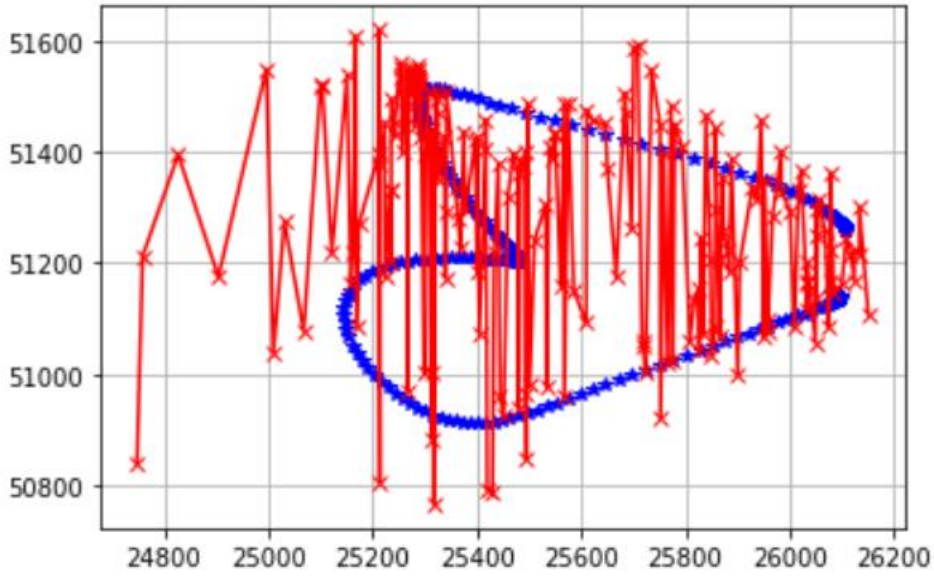
به تعداد epoch های که میدهیم iteration انجام میدهیم. در اینجا میایم به نسبت iteration ها سیگما را مقدار دهی میکنیم . در ادامه حلقه بعدی میاید وزن ها را اپدیت میکند یعنی وزن برنده را پیدا میکند و انتخاب میکند.

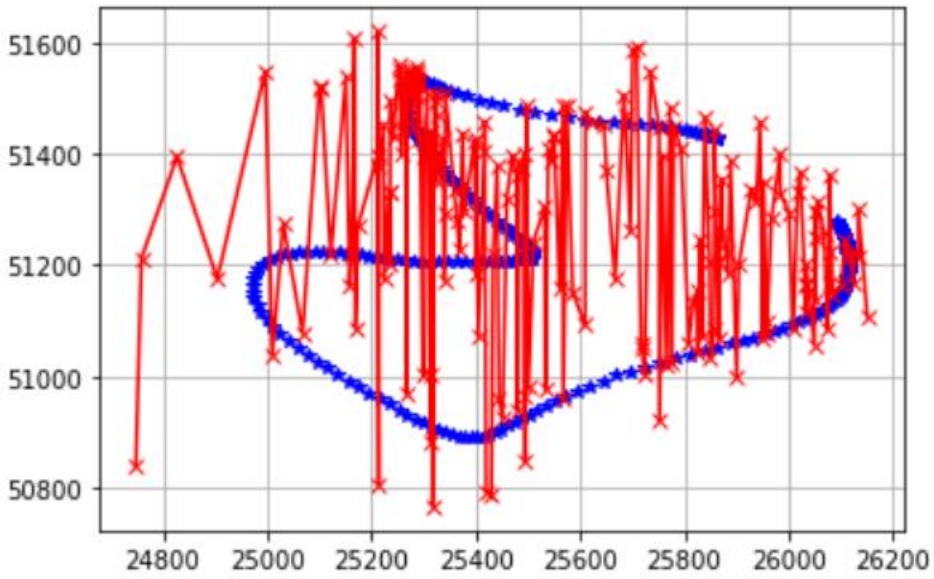
در اخر به هر 40 تا iteration که میرسیم نمودار تابع را رسم میکنیم تا بتوانیم درک بهتری از این کار های بالا داشته باشیم.

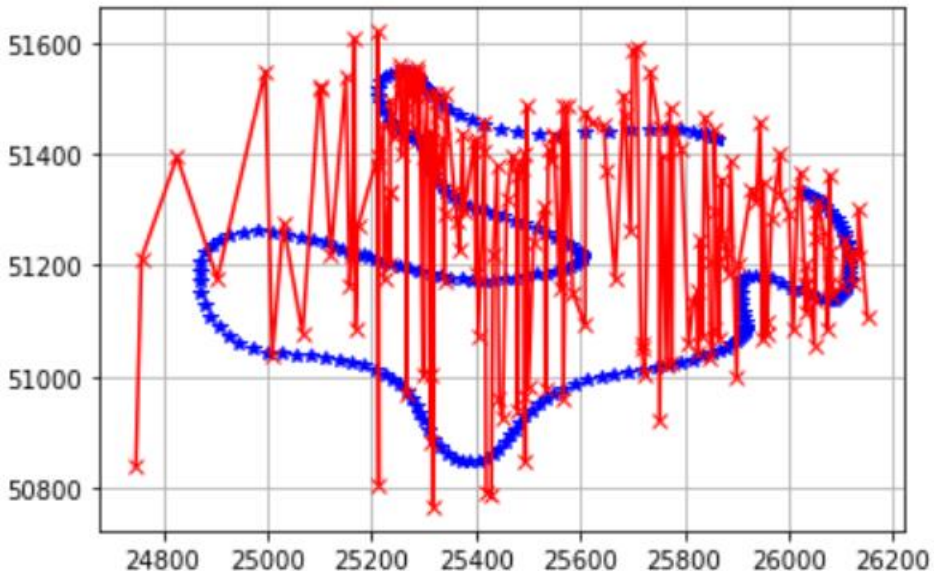

53500
53000
52500
52000
51500
51000
24800
25000
25200
25400
25600
25800
26000
26200

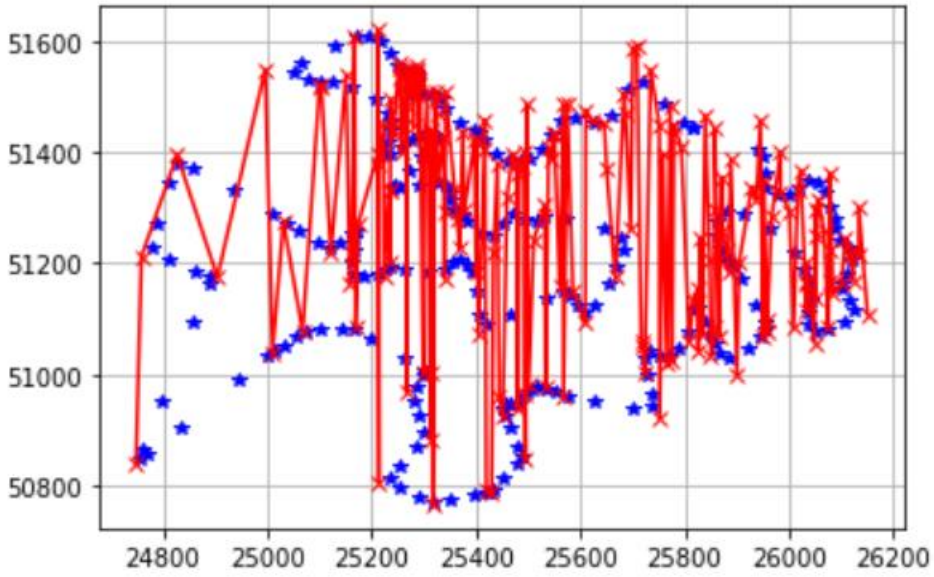

51600
51400
51200
51000
50800
24800
25000
25200
25400
25600
25800
26000
26200

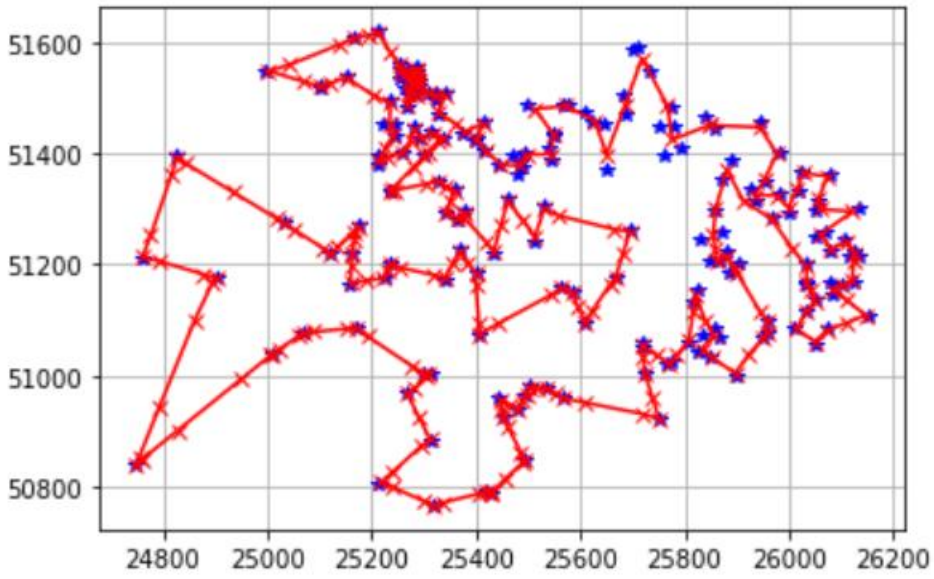

51600
51400
51200
51000
50800
24800
25000
25200
25400
25600
25800
26000
26200


51600
51400
51200
51000
50800
24800
25000
25200
25400
25600
25800
26000
26200


51600
51400
51200
51000
50800
24800
25000
25200
25400
25600
25800
26000
26200


51600
51400
51200
51000
50800
24800
25000
25200
25400
25600
25800
26000
26200


51600
51400
51200
51000
50800
24800
25000
25200
25400
25600
25800
26000
26200


51600
51400
51200
51000
50800
24800
25000
25200
25400
25600
25800
26000
26200