**محمد امین قسوری 97521432 CI HW 2**

فهرست

[توضیحات سوال اول: 2](#_Toc70713626)

[مورد الف: 2](#_Toc70713627)

[نتیجه مورد الف: 5](#_Toc70713628)

[مورد ب 6](#_Toc70713629)

[نتیجه مورد ب 6](#_Toc70713630)

[مورد ج 7](#_Toc70713631)

[نتیجه مورد ج 7](#_Toc70713632)

[توضیحات سوال دو 8](#_Toc70713633)

[قسمت الف 8](#_Toc70713634)

[نتیجه مورد الف 8](#_Toc70713635)

[قسمت ب 9](#_Toc70713636)

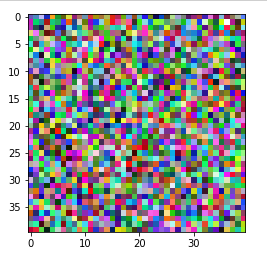
[نتیجه مورد ب 10](#_Toc70713637)

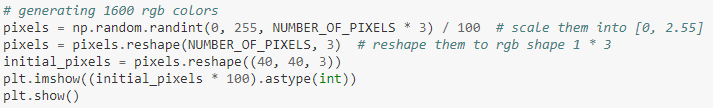
[قسمت ج 10](#_Toc70713638)

# توضیحات سوال اول:

سوال اول شامل سه قسمت بود که مورد سوم امتیازی بود. در مورد اول هدف صرفا طراحی یک مدل SOM بود که نه عملیات کاهش پارامتر learning rate دارد و نه عملیات کاهش شعاع همسایگی. در همان کلاسی که در قسمت اول ساخته شده است همه موارد هندل شده است و صرفا Decay مربوط به این موارد را عدد 1 قرار داده شده است. (که به معنی آن است که تغییری در موارد بالا با اجرای برنامه ایجاد نخواهد شد.)

برای ایجاد دیتاست رندوم از همان numpy استفاده شده است به همان تعدادی که در مسئله گفته شده است داده ساخته میشود و آبجکت های لازیم برای train شدن ساخته میشود. در هر بخش یکبار با روش Euclidean و یکبار با روش Cosine عملیات انجام شده است.





## مورد الف:

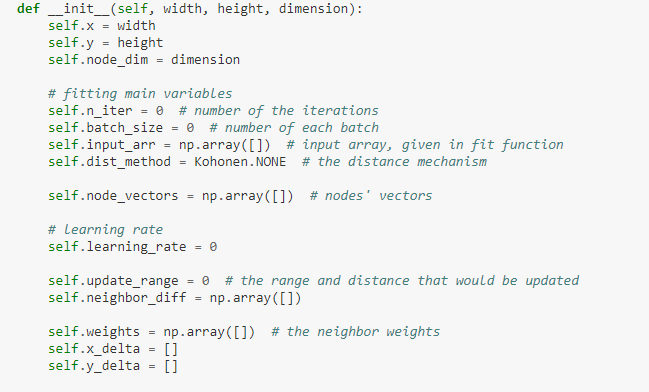
    LEARNING\_RATE\_DECAY = 1

    UPDATE\_RANGE\_FACTOR = 0.25

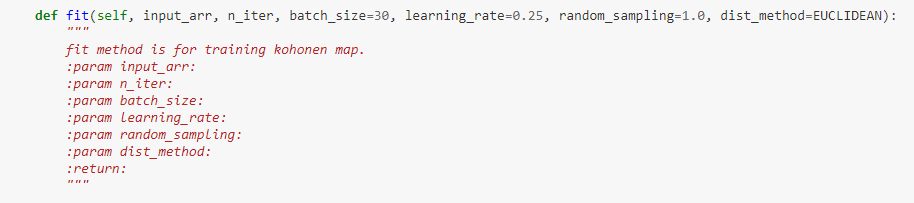
    UPDATE\_RANGE\_DECAY = 1

در این مرحله دو Decay ما 1 هستند که یعنی تاثیری روی پارامترهای مورد نظر ندارند. UPDATE\_RANGE\_FACTOR در حقیقت بیانگر آن است که در شروع لرنینگ شعاع همسایگی را چه نسبتی از طول نقشه در نظر بگیرد به عنوان مثال اگر نقشه ما 10 در 10 باشد با توجه به عدد 0.25 شعاع همسایگی 2 در نظر گرفته میشود. اگر میتوانستیم در ابتدا شعاع را بزرگتر بگیریم و رفته رفته آن را کم کنیم خیلی نتیجه بهتری میتوانستیم بگیریم.

کلیت کلاس Kohonen به این صورت است که در constructor طول و عرض نقشه و ابعاد هر خانه داده میشود که در این جا این ورودی ها 40، 40 و 3 هستند.

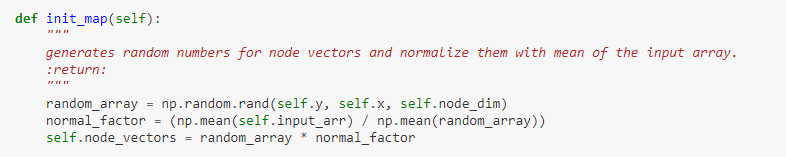


ورودی تابع اصلی ما که مدل را train میکند یا همان تابع fit بصورت زیر است.

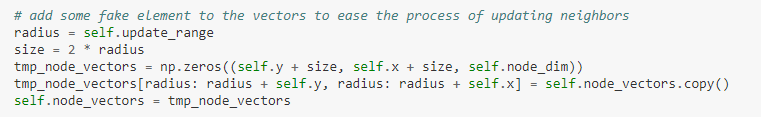


مواردی تکراری مانند دیتاست ورودی، تعداد تکرار، سایز batch و همین طور learning rate هستند و فکر کنم لازم به توضیح نیست. یک random\_sampling داریم که از کل دیتاست ما نسبتی را انتخاب میکند و در ادامه آن‎ها را بچ بچ میکند تا با سرعت بیشتری عملیات یادگیری تکمیل شود. همچنین dist\_method بیانگر نوع سنجش فاصله هست که روش Euclidean و Cosine در این کلاس زده شده و در ابتدا تابع فیت یکی باید انتخاب بشود.

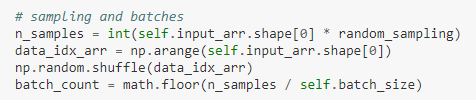
در ادامه نقشه خود را initialize میکنیم و اعدادی که جنریت میکنیم به هر حال رندوم هستند و مشکلی نباید بوجود بیاورند ولی با این حال ما میانگین مجموعه ورودی را هم در نظر میگیریم و یک ضریب کلی را در آن لحاظ میکنیم.



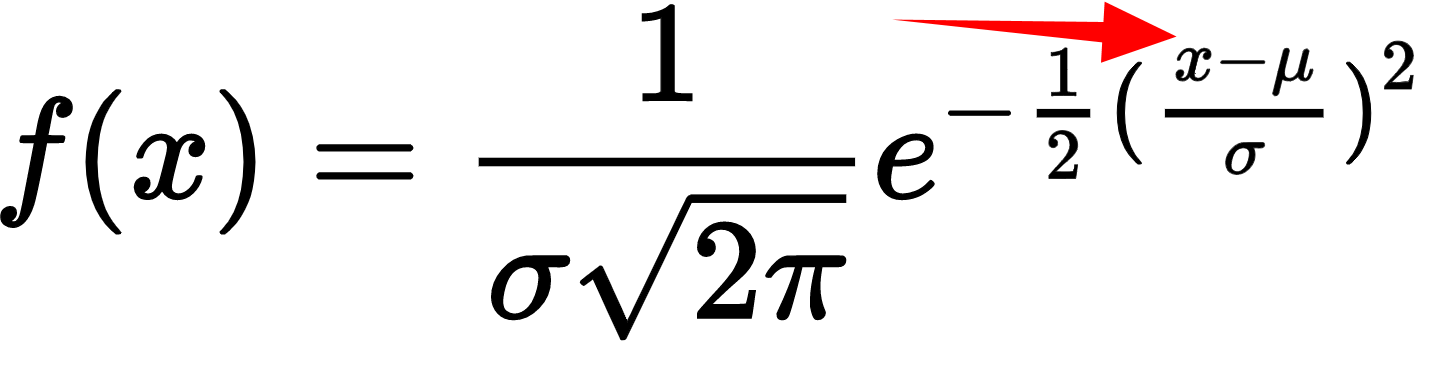
نکته دیگری که قابل ذکر است این است که موقع آپدیت کردن هر node رو نقشه باید اطرافیان آن را آپدیت کنیم و خوب میتوانیم برای راحتی کمی این نقشه را در هنگام train و fit شدن بزرگتر فرض کرد تا درگیر جزئیات همسایگی node ها قرار نگیریم. ( مثلا ممکن است یک node در گوشه باشد و سمت چپ و راست نداشته باشد و ... )



در ادامه با توجه به random\_samplingیک میزان از دیتاست را بر میداریم و تعداد بچ های مورد نیاز آن را حساب میکنیم.

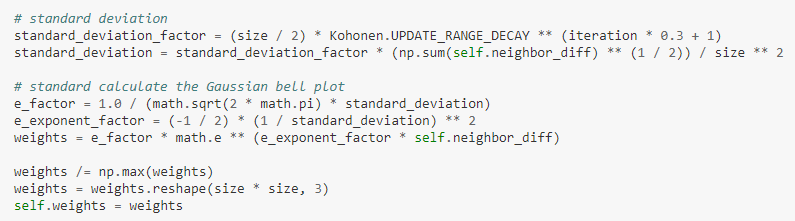


در ادامه نیاز است تابع calculate\_neighbor\_diff را یکبار صدا داریم. همانطور که میدانید برای حساب کردن معادله تابع Normal یا Gaussian bell نیاز است معادله زیر را بدست آوریم.

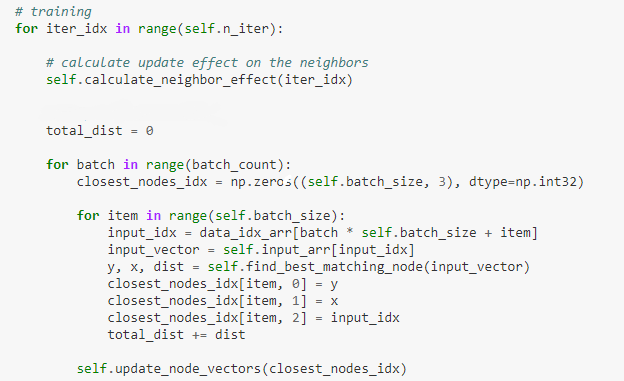


در این جا محاسبه عبارت رو به رو کمی از طرفی دردسر دارد ولی از طرفی هم برای همه یکی است و تنها یکبار لازم است مجذور اختلاف برای محاسبه واریانس یا sigma یا strandard deriviation حساب شود. در این تابع فقط یکبار اینکار با for انجام خواهد شد.

در ادامه وارد iteration اصلی میشویم که در هر حلقه تعدادی بچ با توجه به سایز batch هامون اجرا خواهند شد. در ابتدای هر iteration یک بار باید متد calculate\_neighbor\_effect را که ورودیش iteration است، صدا بزنیم. در حقیقت این متد همان معادله Gaussian bell را برای ما حساب میکند ولی خوب اگر ضریب Decay ما برای شعاع همسایگی 1 نبود علاوه بر محاسبه کردن معادله با توجه به iteration تاثیری نیز روی مقدار شعاع همسایگی میداشت. (در قسمت ج بیشتر بحث میشود).

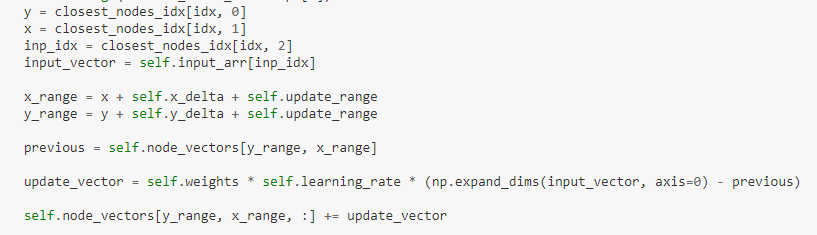


در ادامه برای تک تک موارد نزدیک ترین نود به آن مقدار را پیدا میکنیم و در داخل closest\_nodes\_idx قرار میدهیم تا با استفاده از تابع آپدیت و پاس دادن این index ها قادر به عملیات آپدیت باشیم.

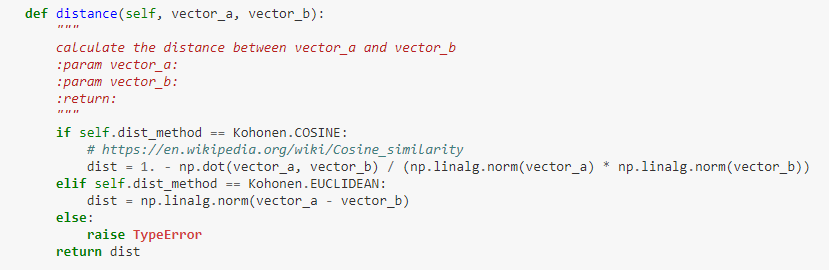


برای پیدا کردن نزدیک ترین نود از متد find\_best\_matching\_node استفاده میکنیم. که در آن با توجه به مکانیزمی که برای سنجش فاصله داریم بهترین مورد پیدا میشود و علاوه بر مکان آن فاصله را به ما بر میگرداند. همه distance ها را هم در total ذخیره میکنیم تا بتوانیم گزارشی بصورت لاگ بر اساس آن مکانیزم ارائه بدهیم.

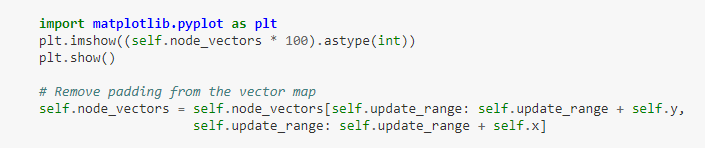
در متد آپدیت با توجه به نزدیک ترین node ای که از قبل بدست آوردیم باید نود ها را آپدیت کنیم حال در اینجا تاثیر روی هر نود متفاوت خواهد بود و با توجه به وزنی که در معادله نرمال بدست آوردیم و learning\_rate میزانی از اختلاف آن‎ها برطرف میشود و نود را آپدیت میکنیم.



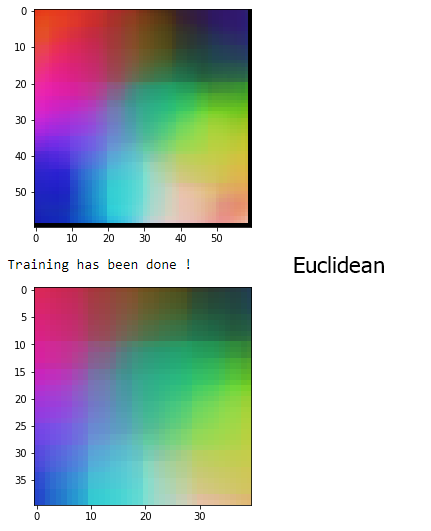
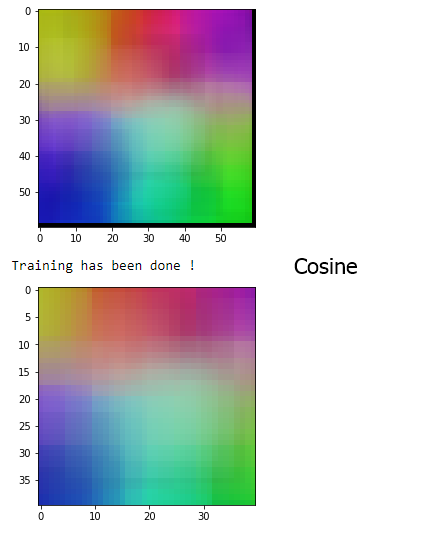
متد find distance بصورت زیر هست که صرفا معادله را پیاده سازی کردم.



در انتها آن یکسری خانه های فیکی که برای آسانی خودمان اضافه کردیم را پاک میکنیم ولی من بصورت دلخواه قبل این حذف تصویری نیز از وضعیت آن نشان داده ام.



### نتیجه مورد الف:



دو مورد نمایش داده شده است. بالایی ها هنوز خانه های fake حذف نشده اند.

## مورد ب

    LEARNING\_RATE\_DECAY = 0.76

    UPDATE\_RANGE\_FACTOR = 0.25

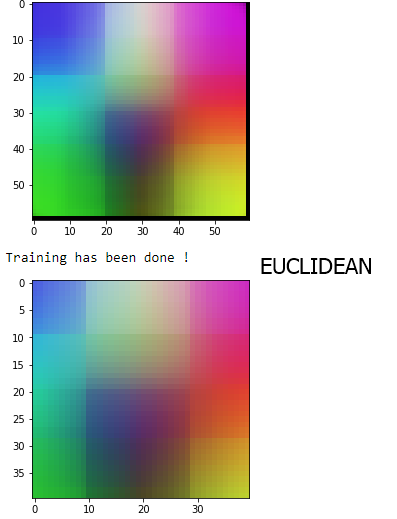
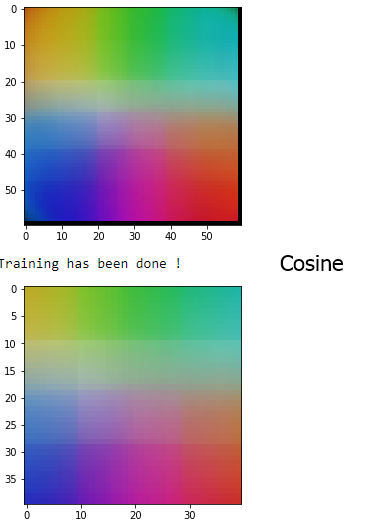
    UPDATE\_RANGE\_DECAY = 1

این دفعه Decay را فقط برای learning\_rate لحاظ کردیم. برای همین در آخر هر ایتریشن learning\_rate بصورت زیر تغییر میکند. که توضیحی بیشتر از این ندارد.



ثابت بودن Decay برای learning\_rate ایراداتی دارد. اگر ما تغییری در سرعت یادگیری ندهیم در آخرین iteration دقیقا با همان سرعت ابتدای برنامه مدل یادگیری را انجام میدهد و این باعث میشود که خوب این باعث میشود به یک پایان نرسیم و ممکن است دوباره یادگیری ما خراب شود.

### نتیجه مورد ب

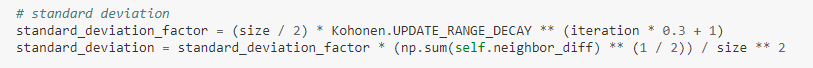


با استفاده از Decay که برای learning\_rate گذاشتیم خیالمان بابت مشکلی که بود کمی آسوده تر میشود ولی در حالت سمت چپ همانطور که مشاهده میشود احساس میشود که بخاطر ثابت بودن شعاع مشکلاتی پیش آمده است.

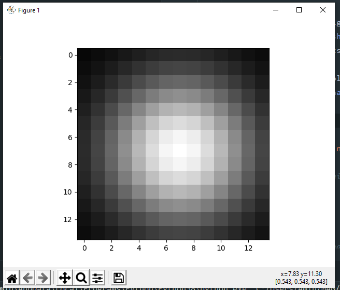
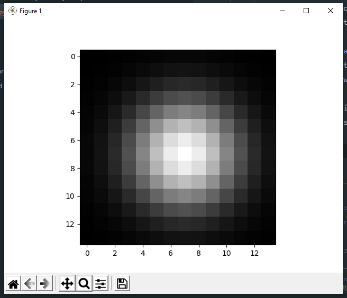
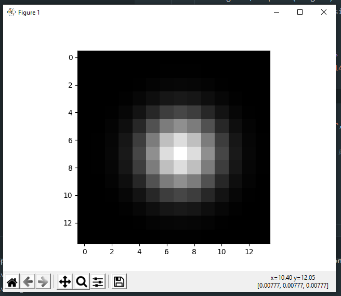
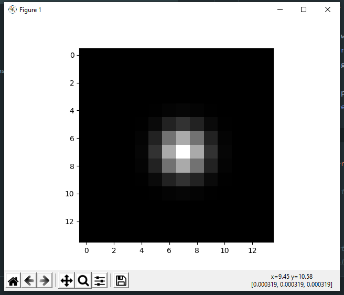
## مورد ج

ثابت بودن Decay برای شعاع همسایگی که در این جا update\_range نام گذاری شده است ایراد دارد که همان حالت پیچ خوردگی و پروانه ای هست. یعنی اگر این حالت بد پیش بیاید با توجه به ثابت بودن شعاع همسایگی هیچگونه نمیشه از آن جلوگیری کرد. ولی اگر در ابتدا شعاع را خیلی بزرگ در نظر بگیریم و رفته رفته آن را کم کنیم باعث میشود که اگر همچین مشکلی پیش آمده باشد برطرف شود.

حال برای این که این Decay را لحاظ کنیم باید آن را بصورت زیر لحاظ کنیم.

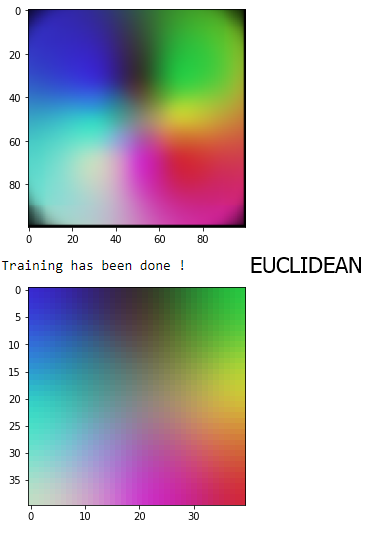
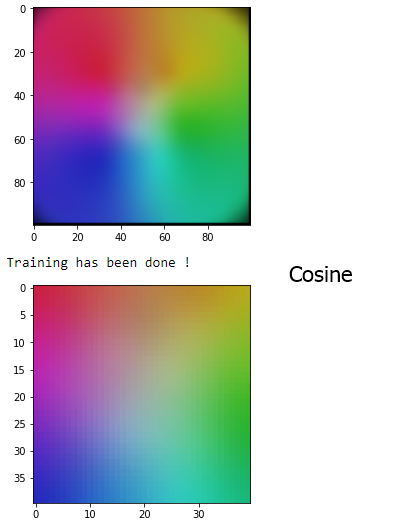


یک ضریب میسازیم که تابعی از iteration و مقدار Decay است و آن را در standard deviation یا همان سیگما ضرب میکنیم. نتیجه را مشاهده میکنیم تا ببینیم وزن‎ها چه شکلی میشوند.



در شکل های بالا میزان تاثیر گذاری روی همسایه‎ها یا همان شعاع همسایگی قابل مشاهده است که در طی iteration های یک تا سی پی در پی کوچکتر میشود و با این وجو دیگر حالت پروانه بوجود نخواهد آمد.

### نتیجه مورد ج

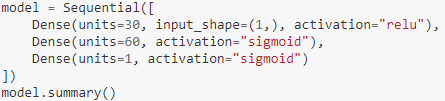


نتیجه نهایی برای مدل Kohonen 😊

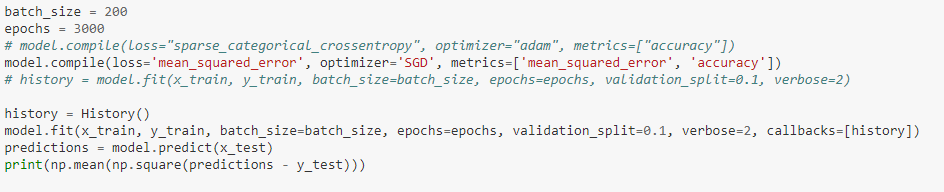
# توضیحات سوال دو

در قسنا اول با استفاده از MLP تابع سینوس را یادگیری میکنیم و در قسمت دوم همینکار را با استفاده از RBF انجام خواهیم داد و در انتها آن‎ها را با هم مقایسه خواهیم کرد.

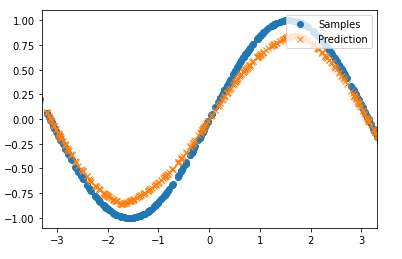
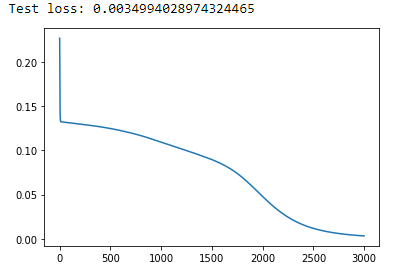
## قسمت الف



برای مدل MLP سه لایه در نظر میگیرم که اولی 30 نرون داره دومی 60 نرون و لایه اخر هم یک نرون کافی است و تابع activation را برای این دو لایه آخر sigmoid در نظر گرفته شده است. در انتها هم بروی دیتاست تست predict صورت میگیرد.



### نتیجه مورد الف



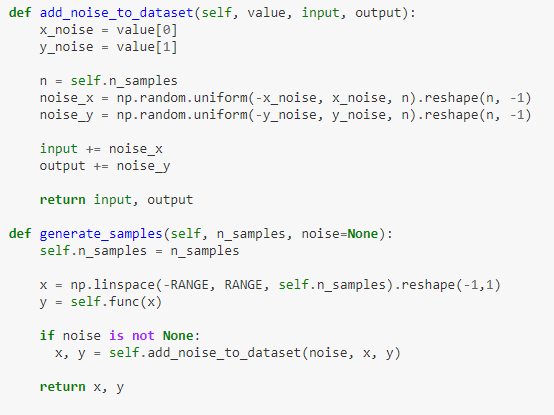
همانطور که مشاهده میکنید روی داده تستی هم بخوبی predict جواب داده و نمودار sample های ما و prediction بصورت بالا شده است.

## قسمت ب

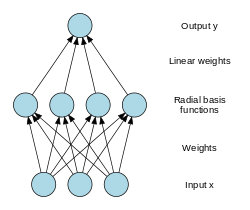
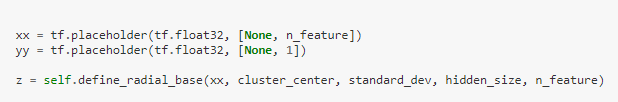
هنگام ساختن آبجکت باید تابعی که میخواید ترین بشه رو بدیم به constructor. در ادامه خود کلاس دیتاست سمپل همراه با noise تولید میکنه و تابع fit را در ادامه ما صدا میزنیم.



این دو تابع دیتاست تولید میکنند و در صورتی که noise داشته باشیم noise را ایجاد و به مقادیر اضافه میکند.



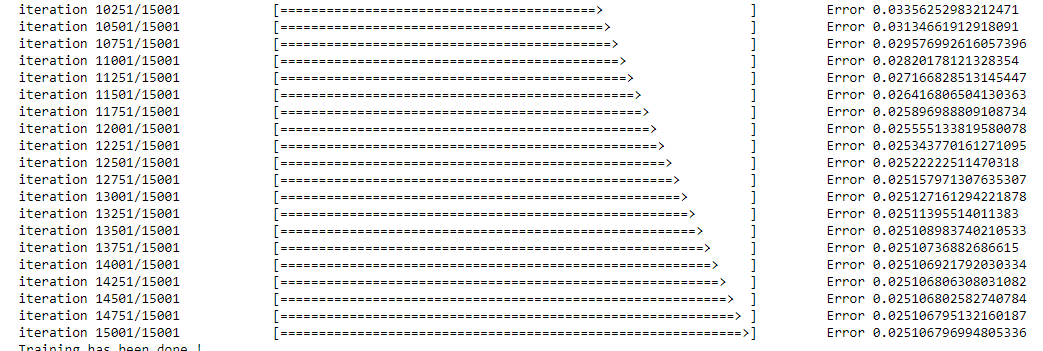
بشکل زیر مدل مربوط را ایجاد میکنیم.

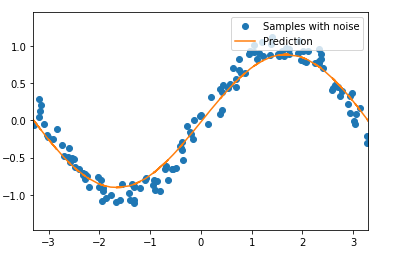
 

در تابع define\_radial\_base از همان معادله Gaussian bell استفاده شده است.

### نتیجه مورد ب

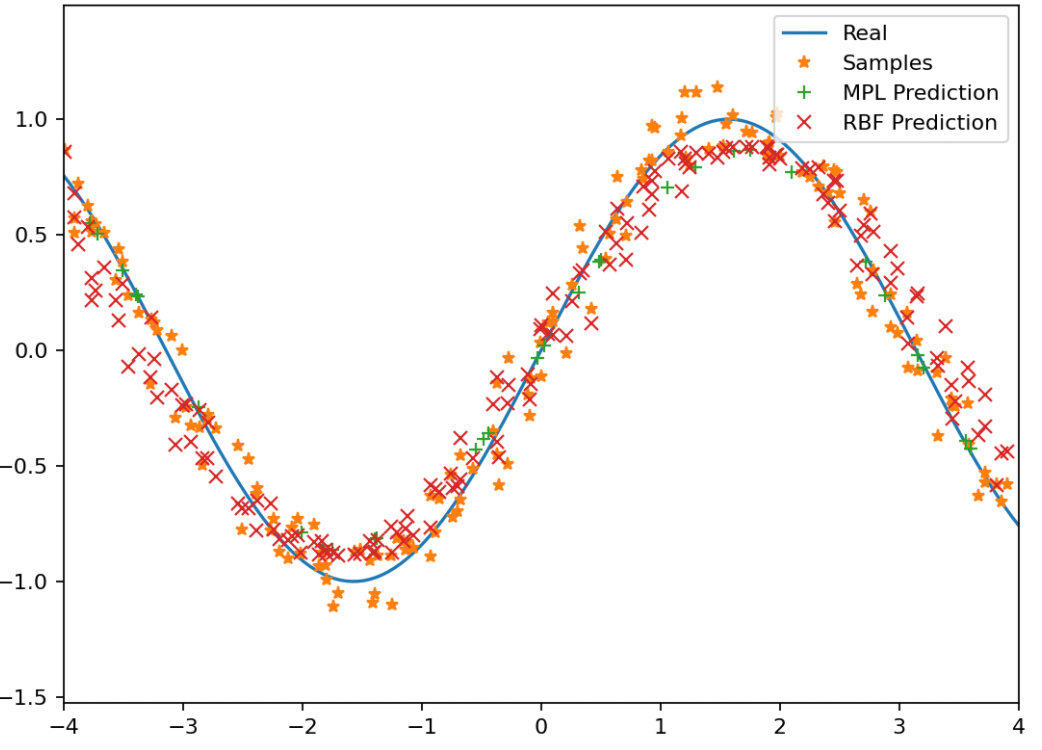
عملیات learn کردن بخوبی اجرا میشود و دیتاست noise دار با error دو صدم یادگیری را انجام میدهد.





## قسمت ج

در این قسمت نمونه های سمپل را که کمی noise دارند به همراه مقادیر بدون noise یا real نمایش میدهیم و در کنار آن پیشبینی‎های حاصل از یادگیری با مدل MLP و RBF نیز گذاشته شده است.



هر دو مدل به نظر خوب می‎آیند و توانسته اند این تابع را یاد بگیرند. نکته ای که هست مدل MLP زمان نسبتا بیشتری برای یادگیری نیاز داشت و به نظر میاید کمی دقت RBF نیز بیشتر باشد. ولی در مجموع همانطور که میبینید هر دو خوب هستند.