بسمه تعالی

گزارش پروژه دوم هوش مصنوعی

استاد: دکتر عبدی

محمد حسین جلالی 97101456

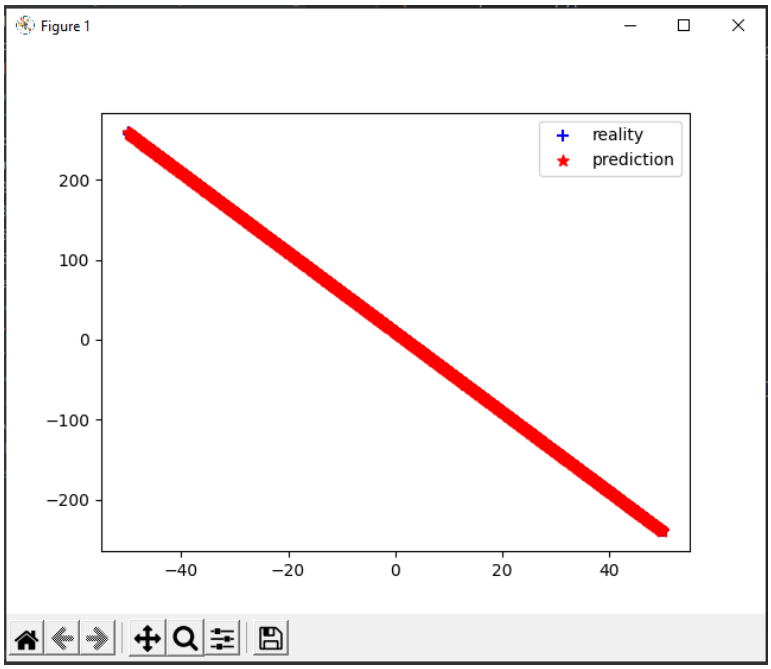
برای پیاده سازی پروژه از کتابخانه sklearn استفاده شده است.

بخش اول:

تابع generate data داده ها ترین و تست را اماده می کند. بعد این داده ها را به فرم مناسب در اورده و به MLPRegressor می دهیم. سپس نمودار پیش بینی شده و تابع واقعی را در کنار یک دیگر رسم میکنیم.

برای حل مسیله یک سری پارامتر وجود دارد که در ابتدا اورده شده است تا به راحتی با تغییر ان ها تاثیر ان ها را بررسی کرد.

ابتدا برای تابع خطی کد را تست میکنیم. بازه داده اموزشی (10 ,10-) و بازه داده آزمایشی هم (50 ,50-) است. تعداد داده تست هم 5000 درنظر گرفته و داده اموزشی هم 100 تا در نظر گرفتیم. به طور کلی در این بخش از solver نوع lbfgs استفاده کردیم که در دیتاست های کوچک خوب عمل میکند. برای تابع خطی activation نوع identity استفاده کردیم. در ضمیمه توضیحات این انواع اورده شده است. و نتیجه مناسبی دریافت کردیم:

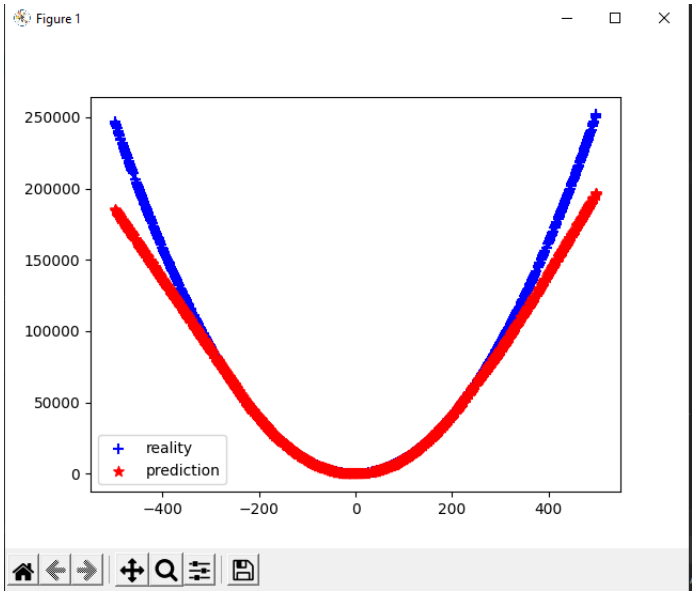


حال سعی می کنیم تابع درجه بالاتر را حدس بزنیم. در کل مشکلی در تابع های به این صورت وجود دارد این هست که ممکن است بازه اموزشی تمام رفتار تابع را نمی تواند پوشش بدهد. چون بازه ای که انتخاب می کنیم ممکن است باعث شود شبکه به درستی تابع را تشخیص ندهد. اگر هم بازه اموزشی را اندازه اندازه بازه تست کنیم عملا بیش برازش رخ داده و در بازه های بزگتر باز دچار خطا می شویم. سعی میکنیم با تغییر پارامتر های مختلف بهترین پیش بینی را انجام دهیم. حال با با مثال هایی درجه های بالاتر را بررسی میکنیم:

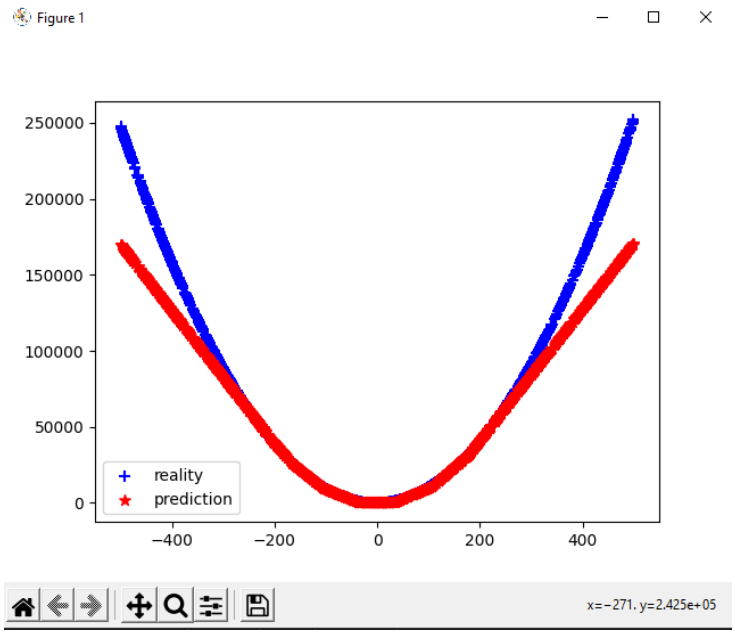
در ابتدا سعی میکنیم تابع درجه 2 را حدس بزنیم:

```
func = "x**2 + 5*x - 3"
```

برای این تابع از relu استفاده می کنیم. برای دقیق تر شدن تابع از 5 لایه 100 تایی استفاده میکنیم. بازه ترین هم نصف بازه تست در نظر گرفتیم:



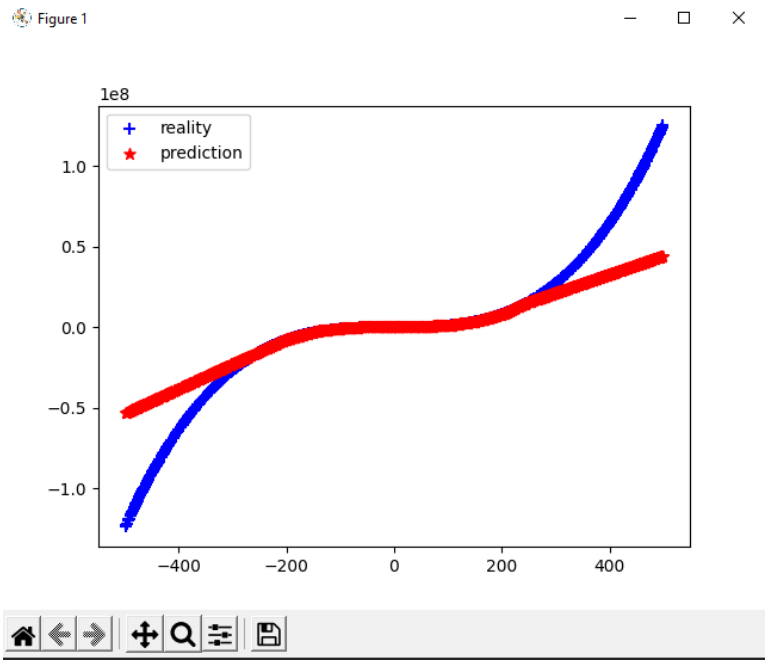
اگر از logistic و با identity استفاده کنیم تابعی که حدس میزند یک خط می شود که مفید نیست! اگر تعداد نورون و لایه ها را زیاد کنیم فرق خاصی نمی کند. مقدار ان را به دو لایه 10 تایی تغییر دادیم و نتیجه حاصل فرق خیلی زیاد با حالت قبل نکرد.



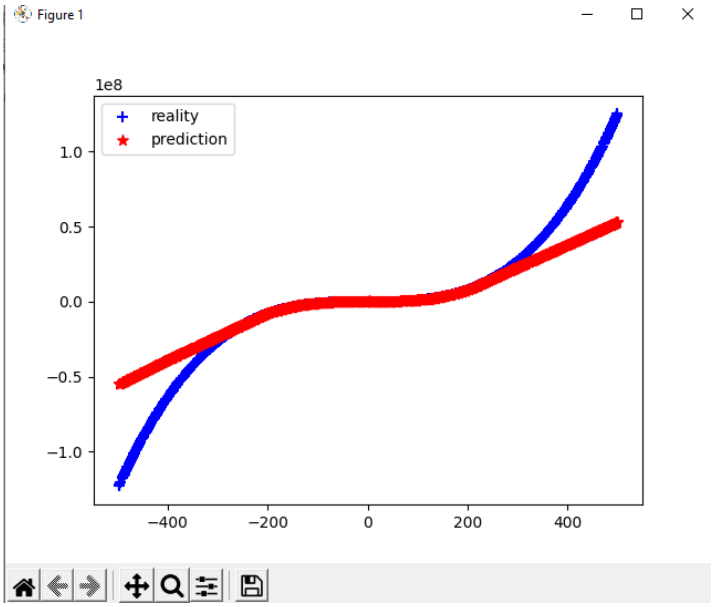
در بازه اموزشی تابع به صورت دقیق پیش بینی شده است و در خارج بازه اموزشی دچار خطا شده ایم. در دو حالت لایه های زیاد تابع دقیق تر پیش بینی شده ولی با توجه به عملکرد خوب لایه های کم این حالت ترجیح دارد.

تابع درجه 3: `func = "x**3 + 2*x**2 + 5*x - 3"`

اول با لایه های کم یعنی دو لایه 10 تایی تست می کنیم:

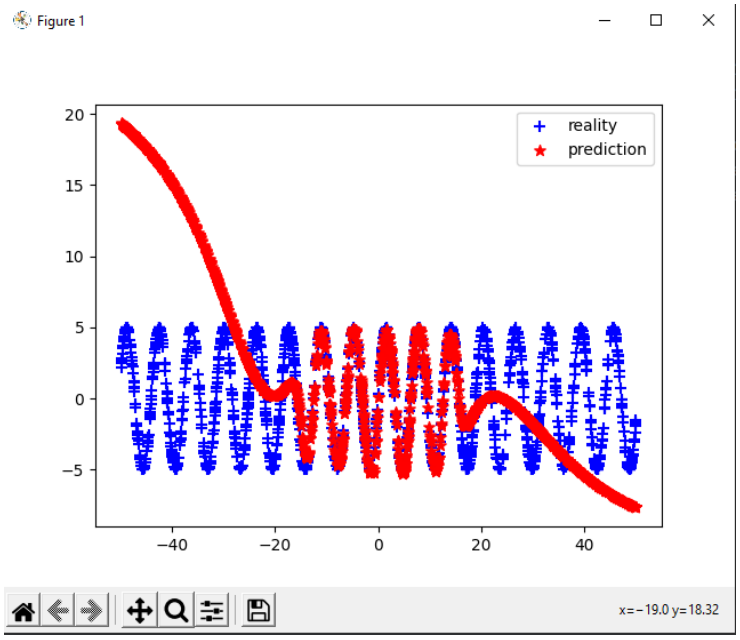


در حالت 5 لایه 100 تایی نتیجه کمی بهتر می شود و بیش از این مقدار تغییر خاصی ایجاد نمی کند.

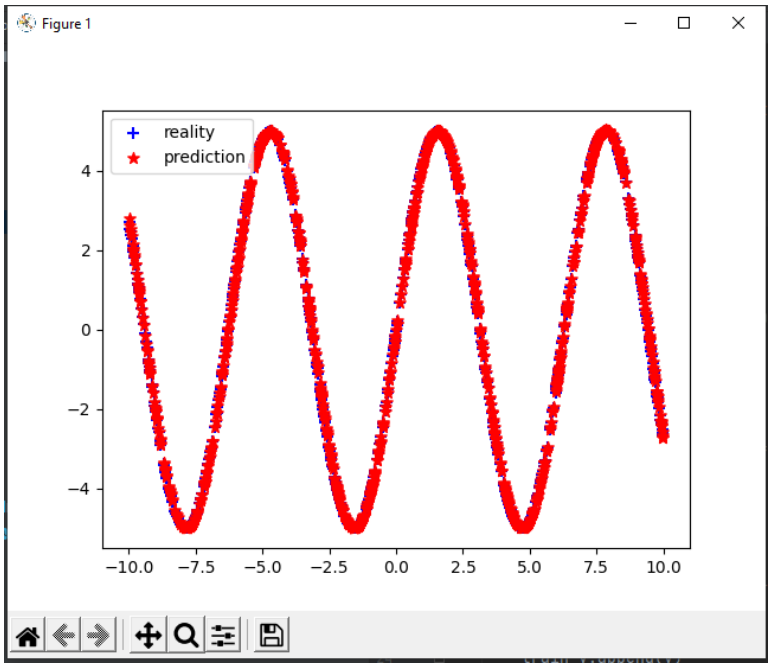


در کل برای تابع های درجه بالاتر عملکرد تخمین در بازه اموزشی خوب است و در خارج این بازه دچار خطا است.

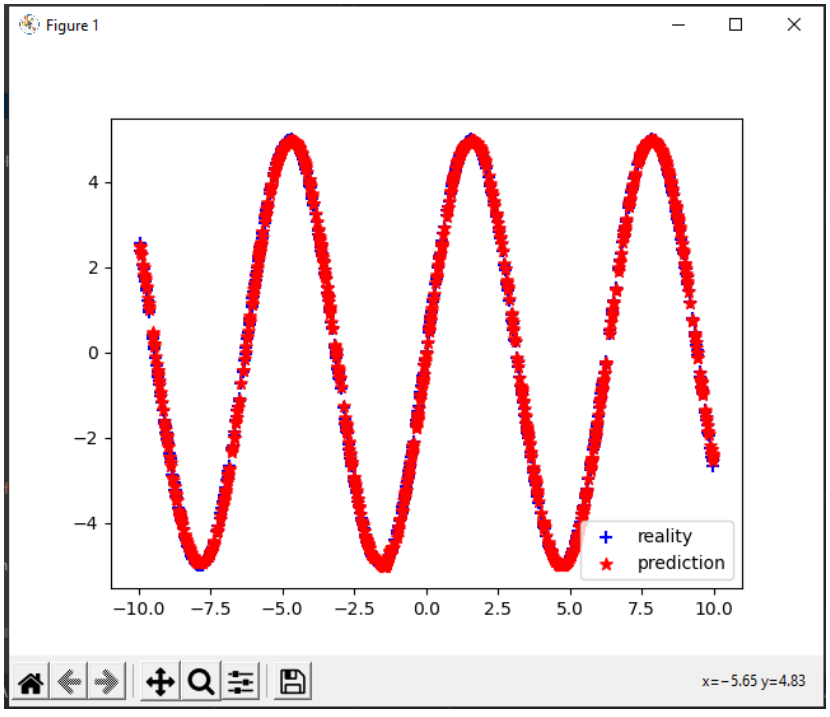
تابع سینوسی: ابتدا یک تابع ساده سینوسی را تست می کنیم. با 5 لایه 100 تایی نتیجه حاصل در بازه ترین قابل قبول است ولی در خارج از ان بازه نتیجه دارای خطای زیادی است.

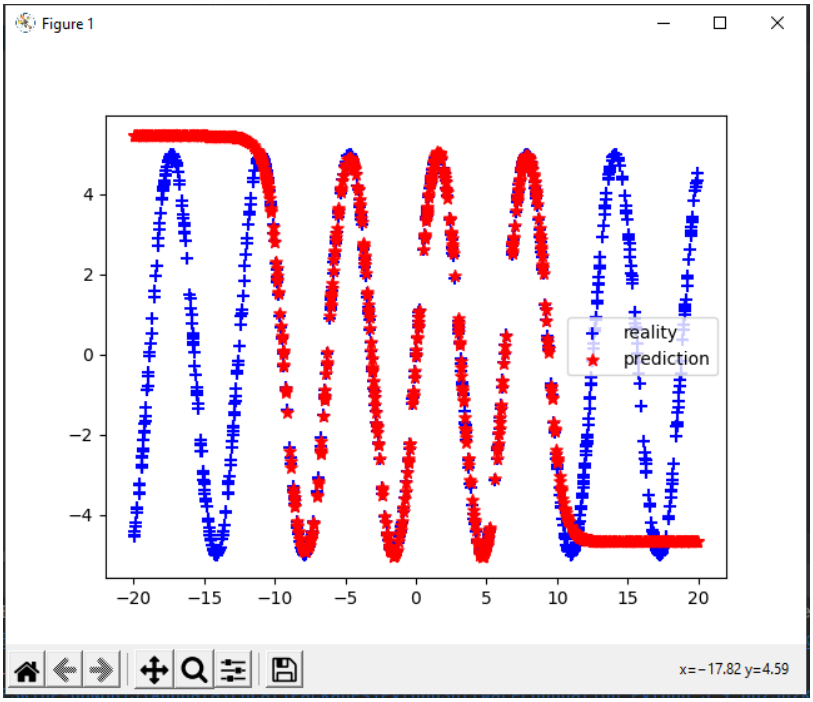


حتی اگر بازه اموزشی و ازمایشی یکسان باشد و بازه کوچکی هم انتخاب شود تخمین کامل درست عمل میکند:

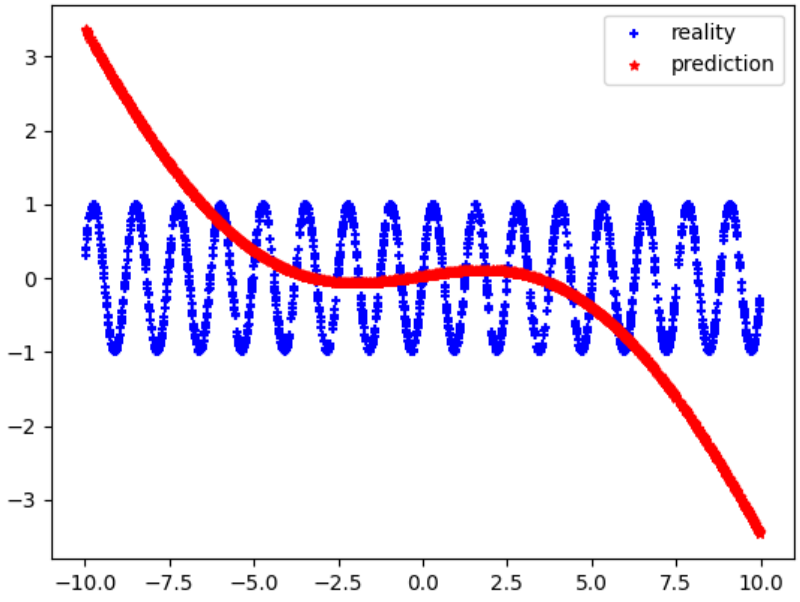


اگر در مصرف نورون زیاده روی نکنیم و دو لایه 10 تایی استفاده کنیم نتیجه در هر دو حالت مساوی بودن و نبودن بازه ها تغییر خاصی نمیکند:

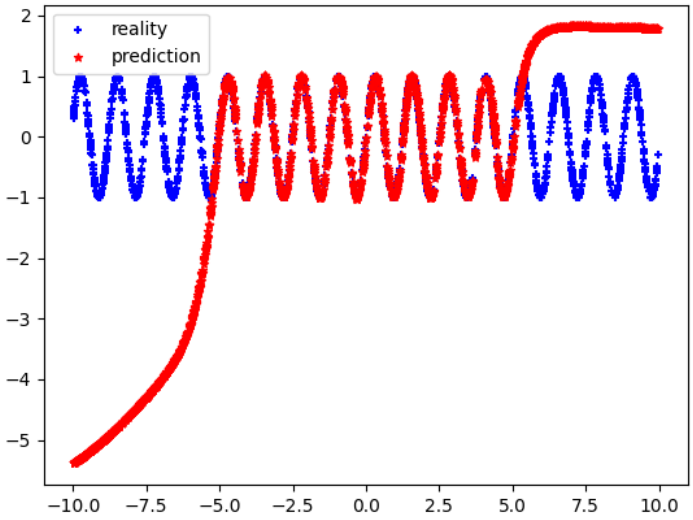




شبکه ما در اینجا ظاهرا دچار overfit شده است. نکته جالب اینجاست که یک شبکه 1000 لایه برای لایه پنهان همچین خروجی به ما می دهد:

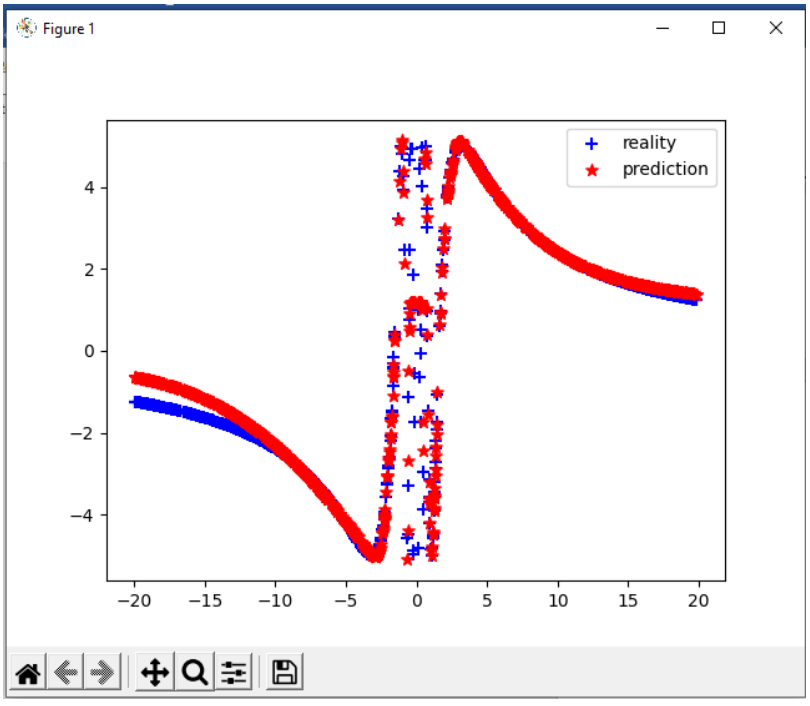


و از درک چرایی این گونه رفتار ها عاجزم! حال اگر به این لایه 1000 تایی یک لایه 10 تایی اضافه کنیم:

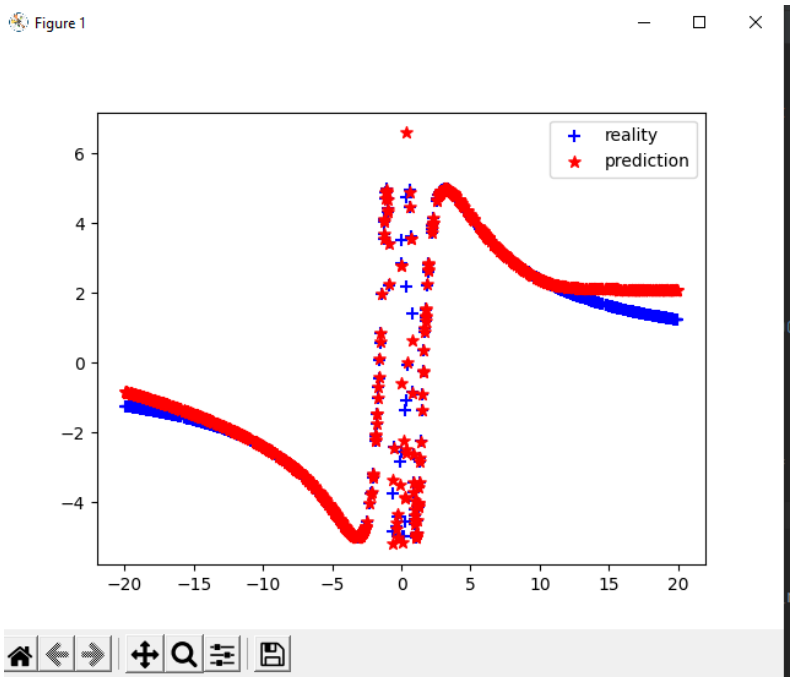


تابع عجیب غریب سینوسی: func = "5 * sin(5/x)"

استفاده از دو لایه 10 تایی:



5 لایه 100 تایی در لایه Hidden:



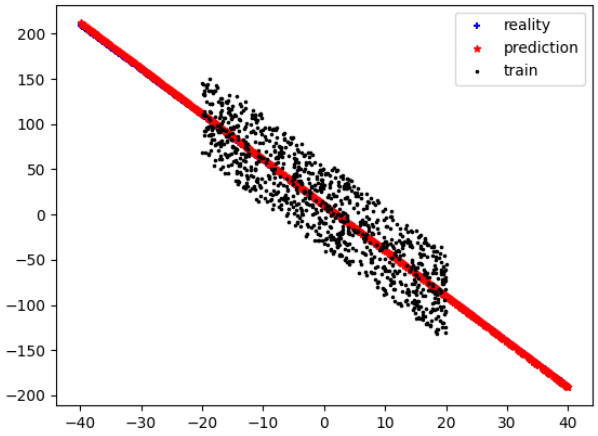
بخش دوم:

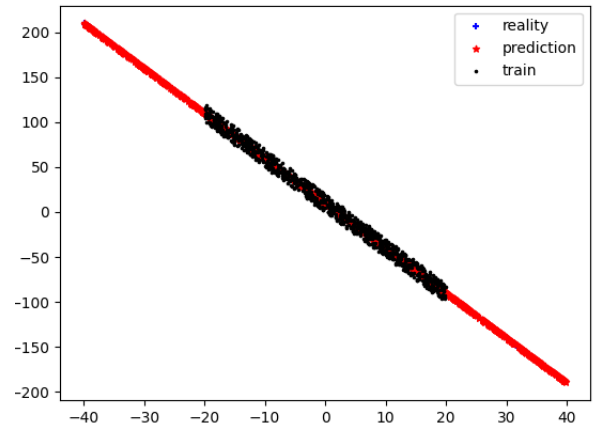
در این بخش به 3 تابع خطی درجه 2 و سینوسی عادی دو نوع نویز وارد می کنیم. در نویز نوع اول مقدار نویز عددی تصادفی در بازه ثابت است. نویز دوم عددی تصادفی بازه اش درصدی از خروجی تابع در همان نقطه است.

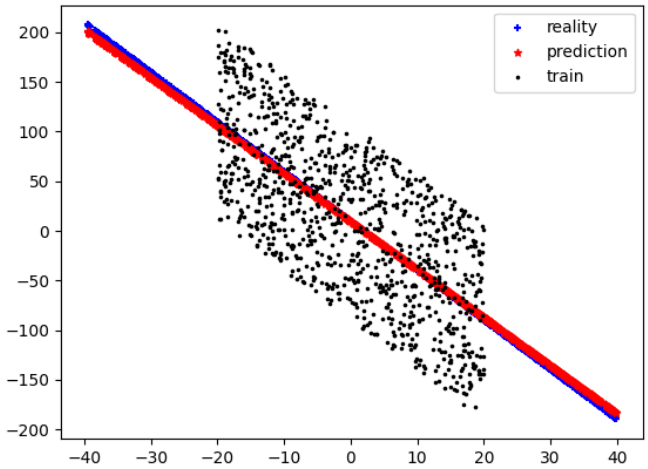
تابع خطی:

در عکس های سری اول نویز با بازه ثابت وارد شده و در سری عکس های دوم نویز با بازه متغیر وابسته به مقدار خروجی تابع در ان نقطه داده است. در نویز های متغیر بازه نویز در سه حالت 0.1,0.5,0.9 مقدار خروجی تابع تست شده است. برای تابع خطی نویز اثر زیادی روی تخمین نذاشته و عملکر مناسبی را شاهد هستیم.

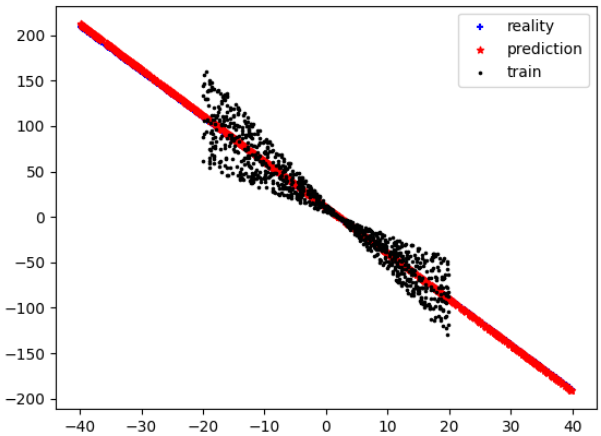
نویز ثابت:

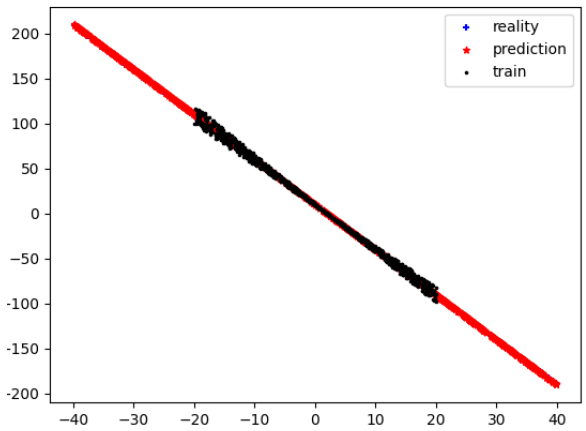

reality
prediction
train

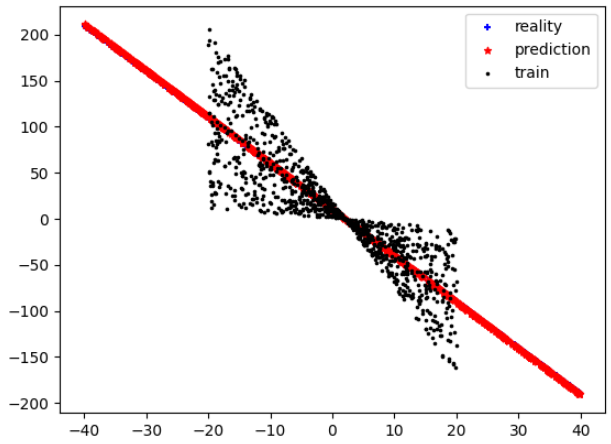

reality
prediction
train


reality
prediction
train

نویز متغیر:

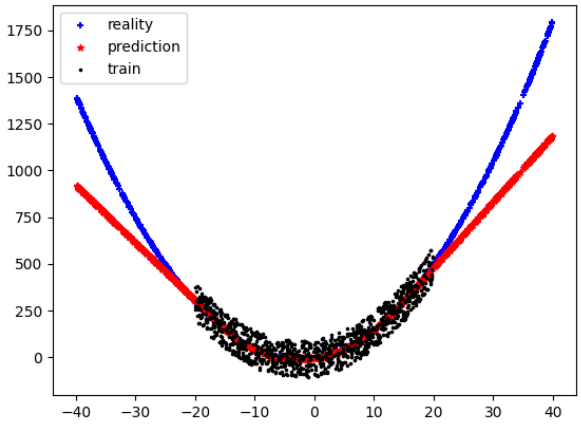

reality
prediction
train

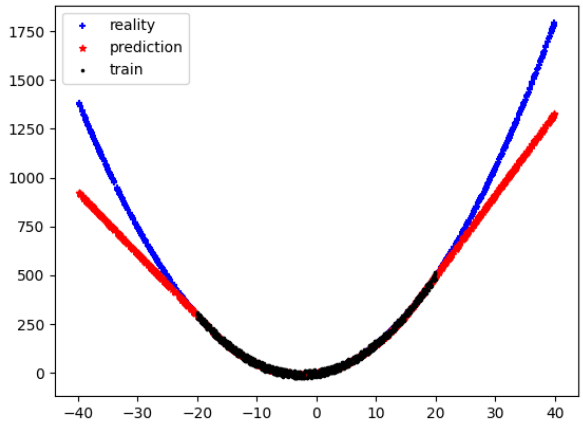

reality
prediction
train

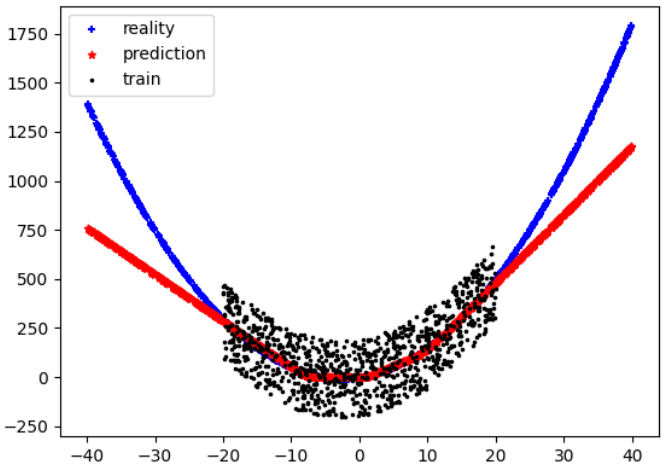


تابع سهمی: در این جا هم مانند تابع خطی عمل می کنیم و 6 سری نویز اعمال می کنیم

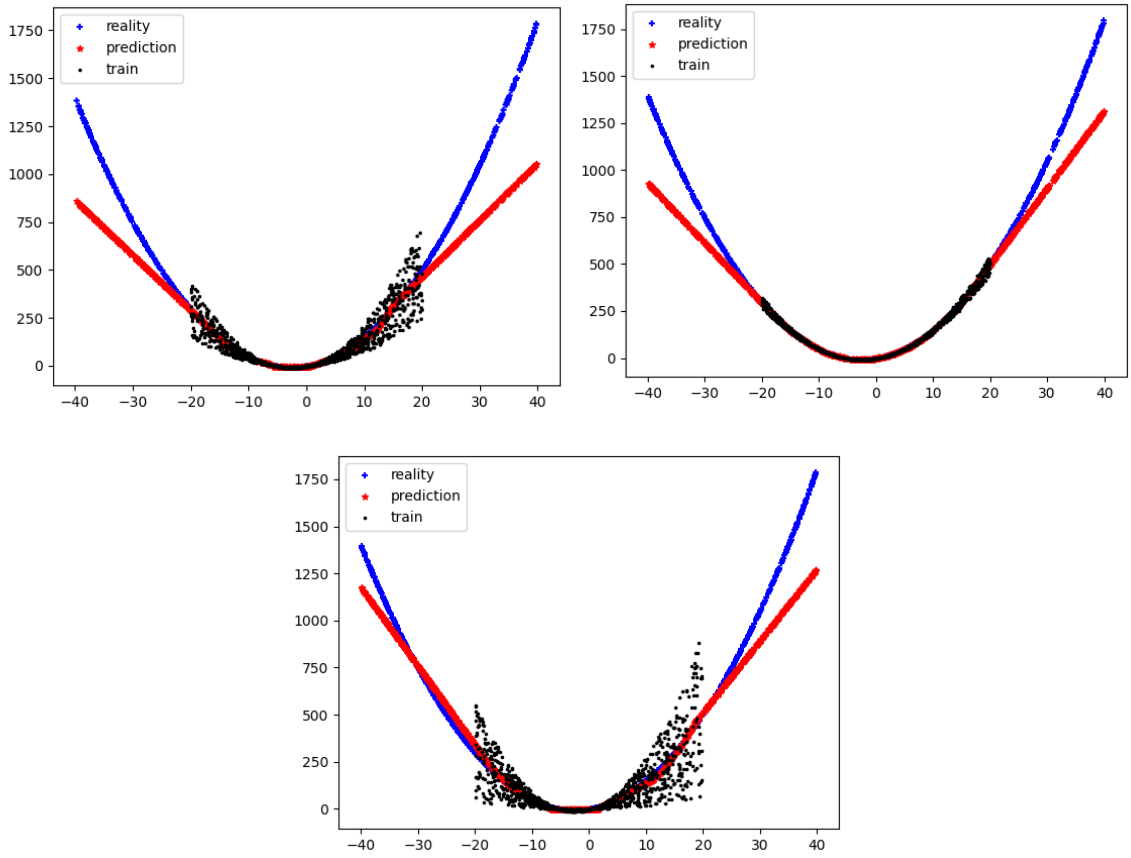
نویز ثابت:



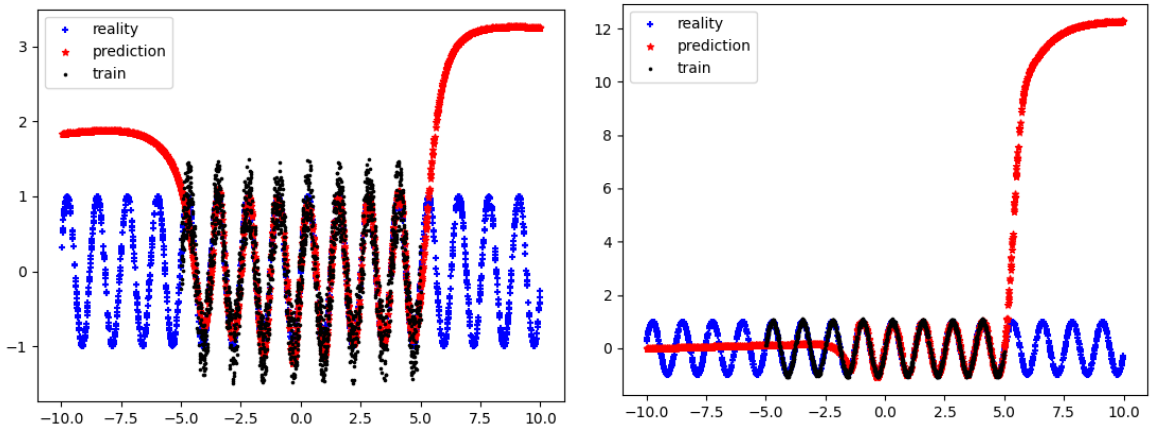


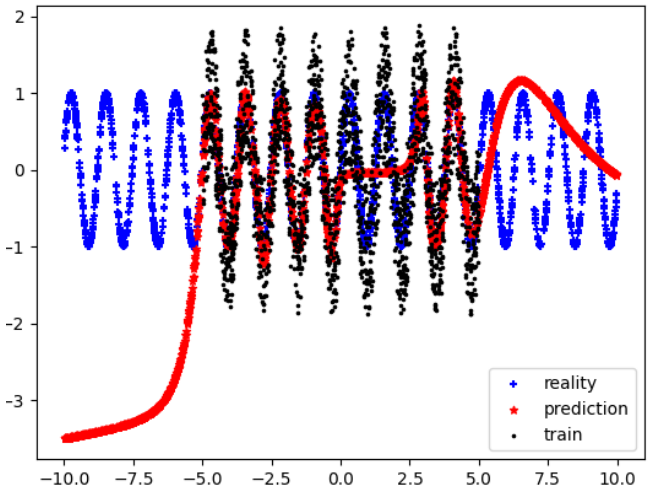


نویز متغیر:

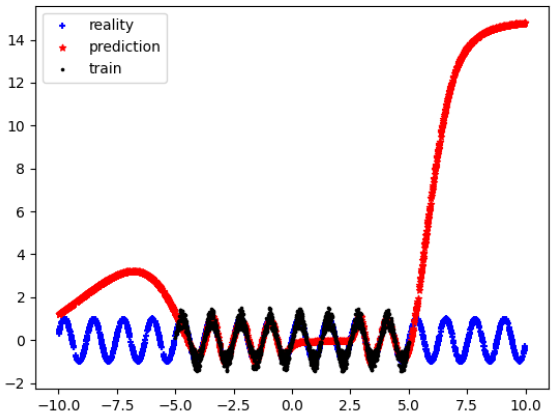


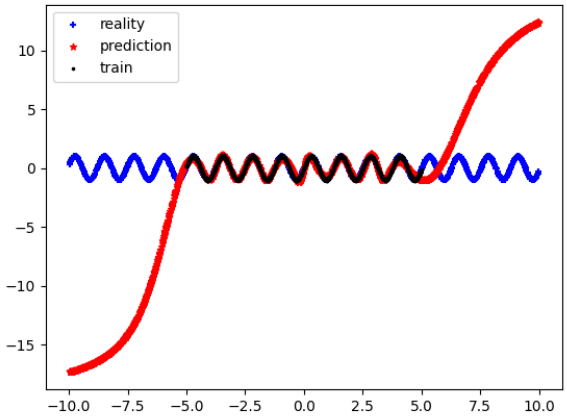
برای این تابع برای اکثر نویز ها عملکرد مشابه بدون نویز است و شبکه نویز را به خوبی مدیریت کرده است. ولی در حالت نویز زیاد متغیر شبکه کمی دچار خطای بیشتر شده است.

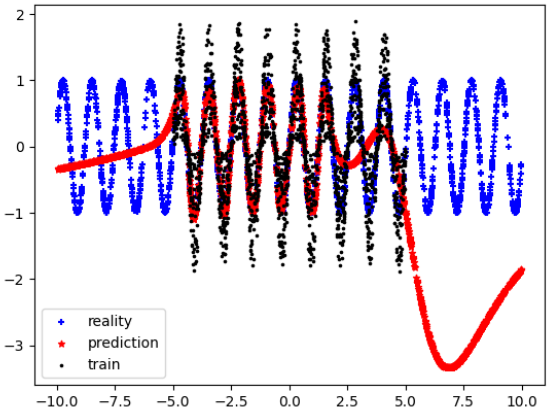




نویز متغیر:







با وجود تاثیر نویز روی عملکرد تخمین ولی مدیریت نویز قابل قبولی داریم.

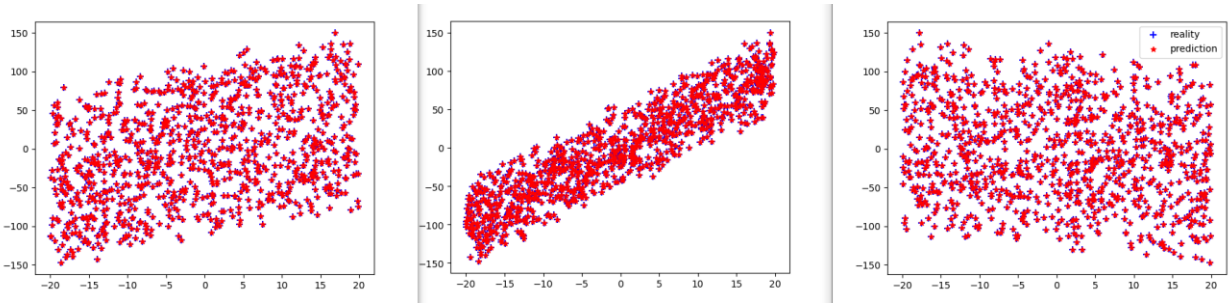
از ازمایشاتی که تا به حال انجام دادیم چند نتیجه مهم حاصل شد:

اول این که در کل شبکه عصبی برای تخمین تابع بهترین روش قطعا نیست و روش های دیگر به نظر کم هزینه ترند و عملکرد بهتری دارند. خصوصا در توابع حالت سینوسی تخمین تابع فقط در بازه ترین مناسب است و در بخش های دیگر خطای قابل توجه داریم. با این وجود شبکه عصبی توان بسیار خوبی در مدیریت و حذف نویز دارد به صورتی در همه حالت ها نویز تاثیر بسیار زیادی روی عملکرد تخمین نداشت.

بخش سوم:

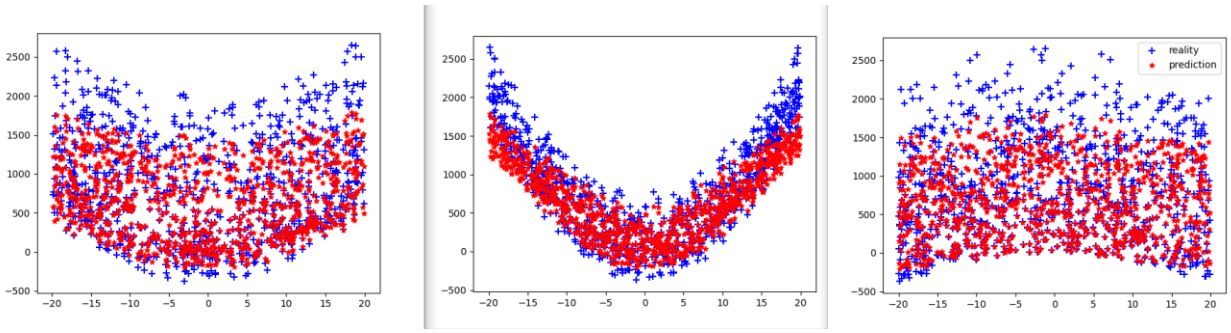
برای این قسمت همان تابع های بخش اول را 3 متغیره می کنیم به طوری که ورودی ان ها x y z است و یک خروجی دارد. حال برای 3 حالت خطی سهمی و سینوسی نمودار خروجی برحسب هر یک از ورودی ها را رسم میکنیم. برای این قسمت چون شاید نمودار واضح نباشد خطایی تعریف کردیم که برای هر نقطه که پیش بینی شده است مقدار خطا تقسیم بر مقدار واقعی را محاسبه کرده و در نهایت همه را جمع میکنیم:

تابع خطی: func = "2*x + 5*y - z + 1"



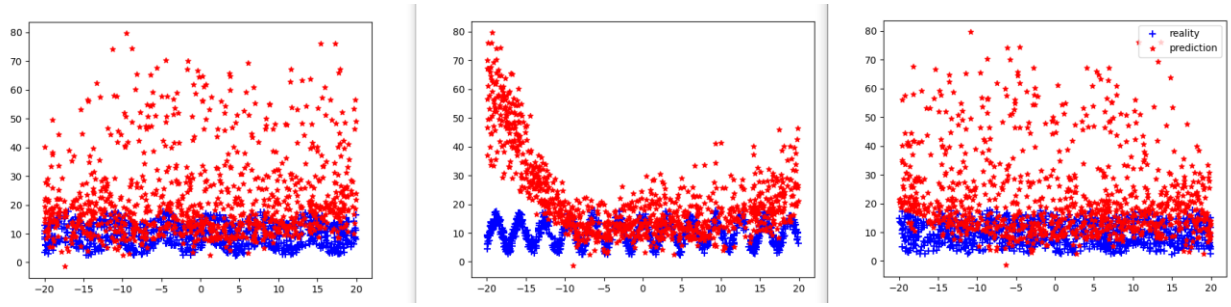
خطا: 0.642

تابع سهمی: func = "2*x**2 + 5*y**2 - z**2 - 3"



خطا: 31.26

تابع سینوسی: func = "2*sin(x) + 5*cos(2*y) - sin(z) + 10"

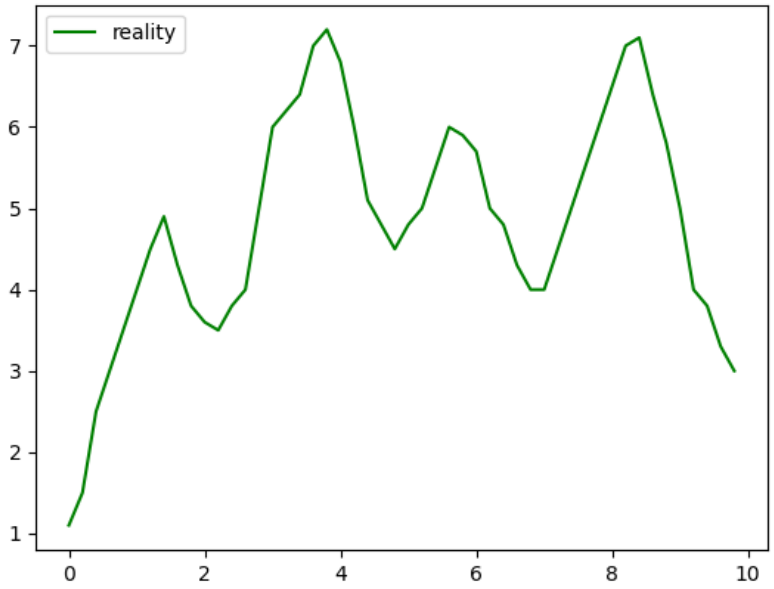


خطا بین 36 تا 48 با ران کردن های مختلف متغیر بود.

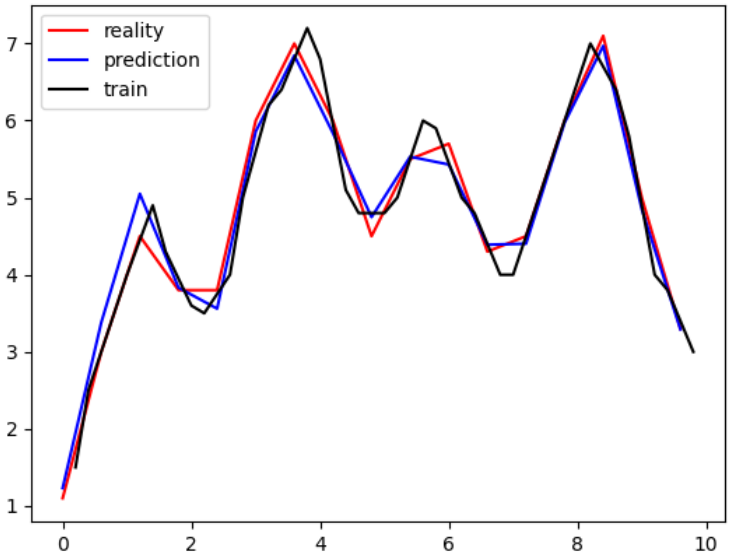
همان طور که در بخش اول خارج از بازه ترین برای سهمی و سینوسی خطا داشتیم اینجا هم دچار خطا می شویم و توابع سهمی و سینوسی به نسبت خطای زیادتری دارند.

بخش چهارم:

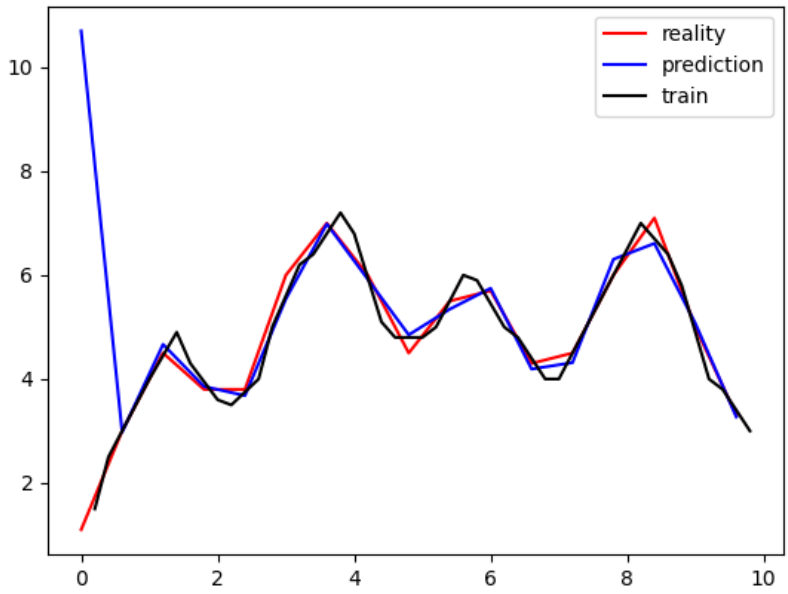
ابتدا نمودار عصبانیت حاصل از این دار فانی دنیا را رسم میکنیم که شبیه رشته کوه های زاگرس شده است:



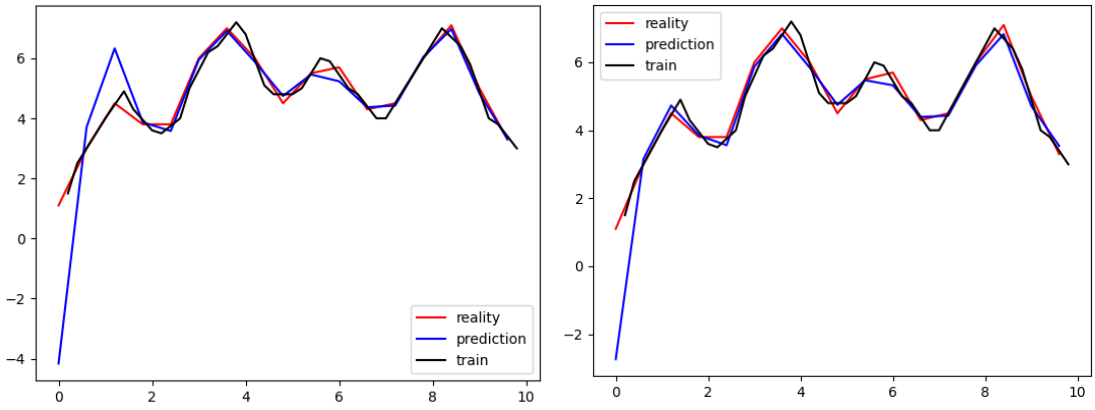
یک سوم دیتا را به تست و بقیه را برای اموزش اختصاص دادیم. با تابع فعال ساز tanh و لایه پنهان شامل 3 لایه 10 تایی است این نتیجه به دست امد:



حالت دیگری که لایه های پنهان به ترتیب 20 و 10 و 5 لایه داشته باشند حالت جالبی را رغم زد:



در کل عملکرد به نسبت خوبی را مشاهده کردیم البته نباید فراموش کرد که بازه ترین و تست یکسان بود. اگه لایه های پیچیده تری قرار میدادیم نتیجه خیلی بهتری نمیگرفتیم و طبعا حالت لایه ها ساده را ترجیح میدهیم. به ترتیب 5 لایه 100 تایی و (500,250,100):



مانند بخش 3 خطای محاسبه شده هم برای بهترین حالت در حدود 1.23 بود. که نتیجه می دهد تخمین خوبی انجام داده ایم.

بخش پنجم:

ابتدا باید عکس ها را اماده پردازش کنیم. در ابتدا هر عکس را لود کرده و ان را تبدیل به ارایه یک بعدی 256 تایی می کنیم. سپس داده های تست و ترین را اماده کرده به MLPClassifier میدهیم. و در نهایت دقت را محاسبه می کنیم.

نکته: برای ران کردن کد باید عکس ها در دو پوشه train و test در پوشه ای که خود کد قرار دارد وجود داشته باشد.

در ابتدا از فعال ساز identity و سالور از نوع adam به خاطر حجم زیاد دیتا و همین طور دو لایه 10 تایی استفاده می کنیم:

```
89.53662182361734
```

حال از 5 لایه 100 تایی استفاده می کنیم:

```
89.38714499252616
```

دو حالت بالا را با فعال ساز logistic تست می کنیم:

با 5 لایه 100 تایی:

```
90.03487792725461
```

2 لایه 10 تایی:

```
88.29098156452416
```

بهترین حالتی که توانستم برسم یک لایه 1500 تایی و فعال ساز relu بود:

```
94.17040358744394
```

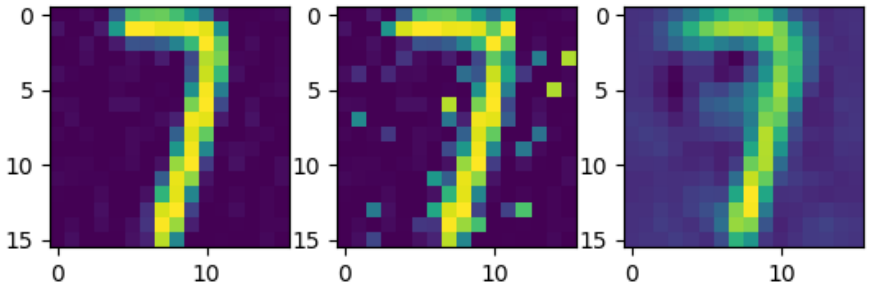
نکته: برای ران کردن کد باید عکس ها در دو پوشه train و test در پوشه خود کد وجود داشته باشند.

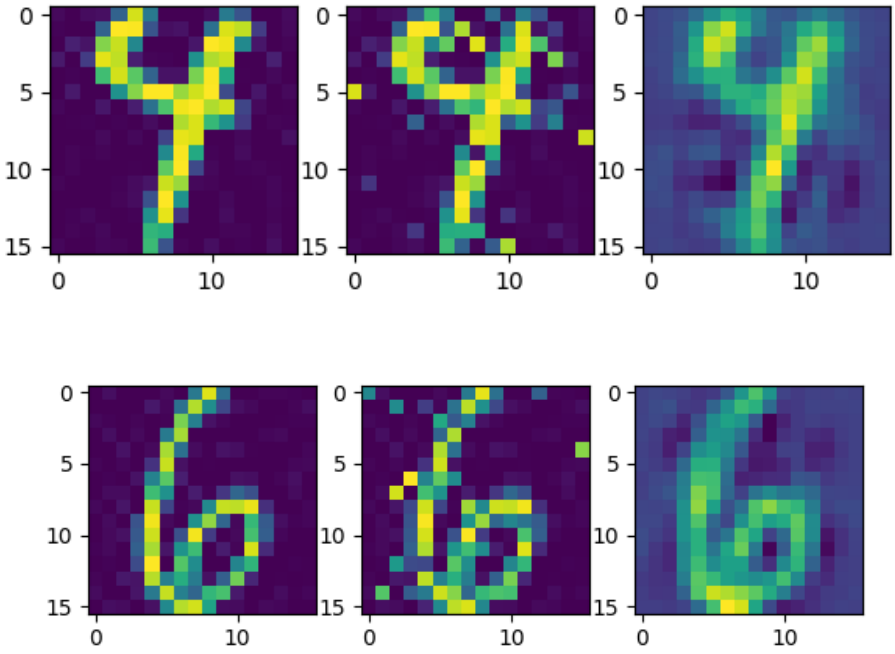
بخش ششم:

در این قسمت ابتدا مانند قبل عکس ها را تبدیل به دیتای مناسب برای ترین کردن می کنیم. برای وارد کردن نویز به عکس ها از استفاده می کنیم. برای وارد کردن نویز به عکس ها از مدل متفاوتی با قبل استفاده می کنیم. به این صورت که به ازای هر خانه ارایه ای که از عکس درست کردیم عددی رندوم بین صفر و یک تولید کرده و اگر این عدد رندوم از بازه نویز بیشتر بود ان وقت به جای ان خانه ارایه عددی رندوم ولی در بازه 0 تا 256 قرار می دهیم. به این شکل عکس ها به نحوی نویز دریافت می کنند که قابل تشخیص به راحتی نباشند. سه سطح مختلف را بررسی میکنیم:

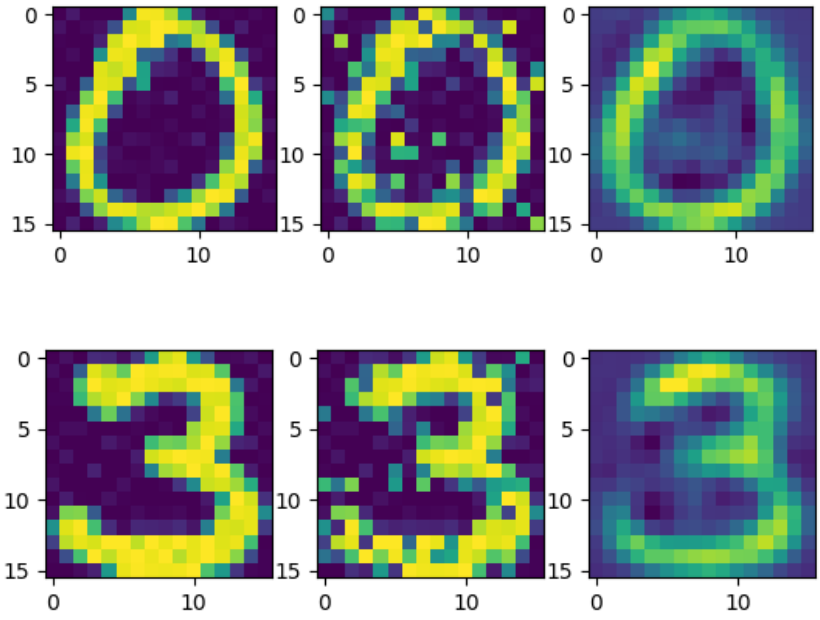
نکته: در عکس ها ردیف وسط عکس با نویز است. ردیف راست خروجی ما و ردیف چپ ورودی یعنی عکس بدون نویز است. برای ران کردن باید عکس ها داخل یک پوشه به نام img باشد که در همان پوشه کد قرار دارد.

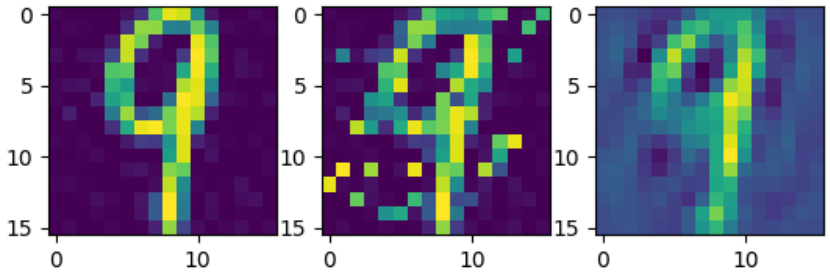
نویز کم: با 3 لایه 100 تایی و فعال ساز relu نتیجه خوبی حاصل شد:



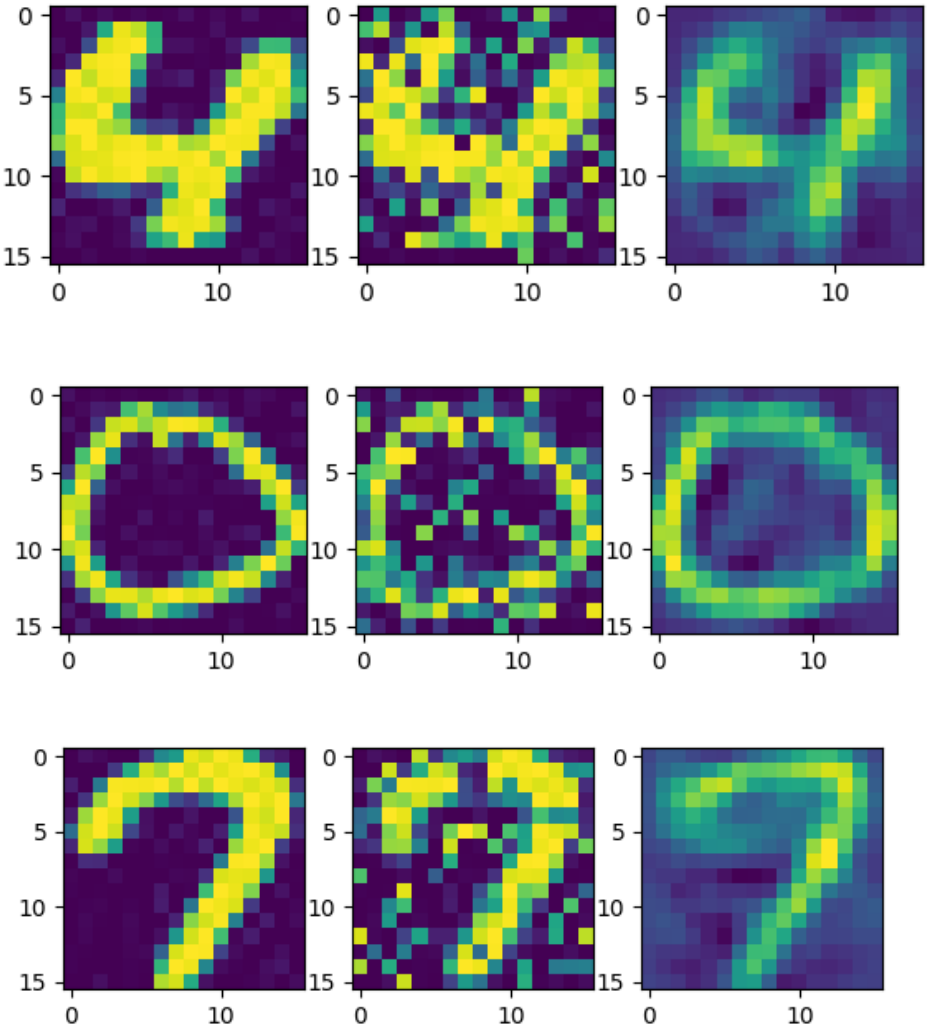


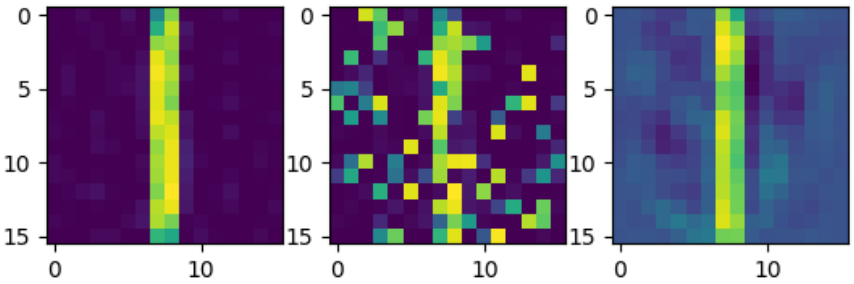
نویز متوسط: مثل قبل از 3 لایه 100 تایی با فعال ساز relu استفاده می کنیم:





نویز شدید: بهترین حالت ممکن به دست امده برای این حالت 3 لایه 100 تایی فعال ساز relu بود که نتیجه خوبی به ما داده است.





برای 3 حالت بالایی امتیاز تخمین را هم با تابع score خود کتابخانه sklearn حساب کردیم که به ترتیب 0.92 برای نویز کم 0.76 برای نویز متوسط و 0.61 برای نویز شدید شد که نتایج قابل قبولی است.