به نام خدا

دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

درس شبکه‌های عصبی و یادگیری عمیق

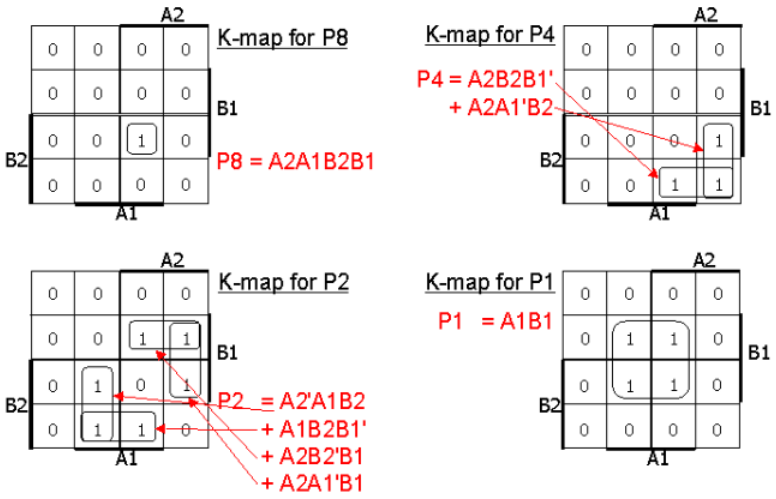
تمرین اول

نام و نام خانوادگی	**فاطمه نائینیان – محمد عبائیانی**
شماره دانشجویی	**810198432-810198479**
تاریخ ارسال گزارش	**1401.08.08**

فهرست

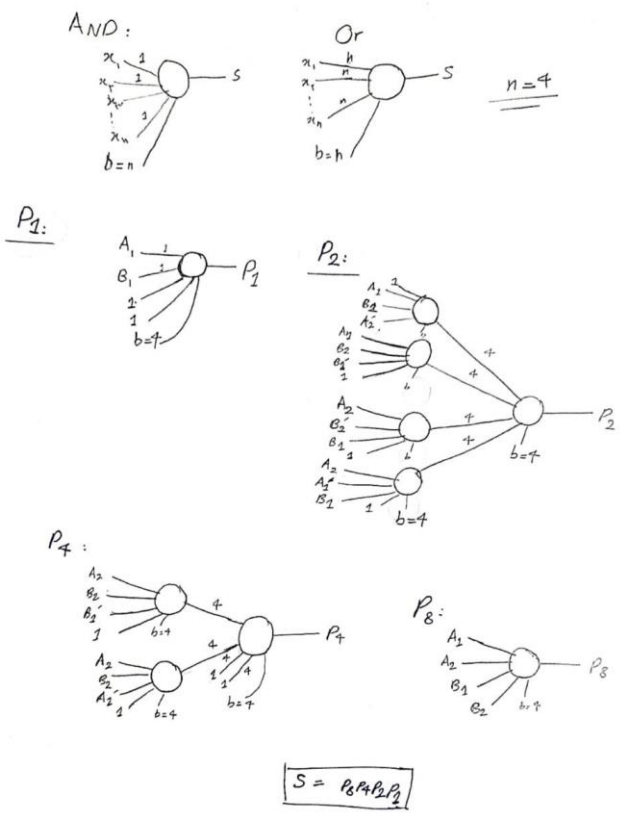
پاسخ 1. شبکه عصبی Mcculloch-Pitts

بخش الف) برای پیاده سازی ضرب کننده ی دو بیتی از روش k-map و نمایش دو لایه ی or-and استفاده کرده و سپس گیت های استفاده شده را با نورون ها پیاده سازی میکنیم



شکل 1**k-map of 2 bit multiplier**

شبکه ی عصبی معادل برای خروجی ها به شکل زیر می باشد:



شکل 2 شبکه عصبی متناظر **k-map**

برای ثابت ماندن threshold ها همه ی آنها را روی مقدار 4 تنظیم نموده (تمام گیت ها 4 ورودی دارند) و وزن ها را مطابق عملکرد گیت ها تنظیم کرده که برای and وزن همه ی ورودی ها برابر 1 و برای or وزن همه برابر 4 میباشد.

همچنین گیت not با وزن ورودی 1- و مقدار threshold برابر 0.5- قابل پیاده سازی است.

بخش ب)

پیاده سازی نورون اصلی و گیت ها:

```
def mult2_bit(A2,A1,B2,B1):

    A2n = NOT(A2)
    B2n = NOT(B2)
    A1n = NOT(A1)
    B1n = NOT(B1)

    ##p8
    p8 = AND(A1,A2,B1,B2)

    ##p4
    y2p1 = AND(A2,B2,B1n,1)
    y2p2 = AND(A2,B2,A1n,1)
    p4 = OR(y2p1,y2p2,0,0)

    ##p2
    y3p1 = AND(B1n,B2,A1,1)
    y3p2 = AND(A1,A2n,B2,1)
    y3p3 = AND(A1n,A2,B1,1)
    y3p4 = AND(B1,B2n,A2,1)
    p2 = OR(y3p1,y3p2,y3p3,y3p4)

    ##p1
    p1 = AND(A1,B1,1,1)

    return [p8,p4,p2,p1]

mult2_bit(0,1,0,1)
```

```
def Mac(x1,x2,x3,x4, w1, w2, w3, w4, b):
    S = x1*w1 + x2*w2 + x3*w3+ x4*w4
    return b <= S

def AND(x1,x2,x3,x4):
    return Mac(x1,x2,x3,x4, 1,1,1,1,4)

def OR(x1,x2,x3,x4):
    return Mac(x1,x2,x3,x4, 4,4,4,4,4)

def NOT(x):
    return -1*x >= -0.5
```

شکل 3 پیاده سازی نورون ها

خروجی ها:

```
 A1 A0 B1 B0 -> p8p4p2p1
(0, 0, 0, 0)   0 0 0 0
(0, 0, 0, 1)   0 0 0 0
(0, 0, 1, 0)   0 0 0 0
(0, 0, 1, 1)   0 0 0 0
(0, 1, 0, 0)   0 0 0 0
(0, 1, 0, 1)   0 0 0 1
(0, 1, 1, 0)   0 0 1 0
(0, 1, 1, 1)   0 0 1 1
(1, 0, 0, 0)   0 0 0 0
(1, 0, 0, 1)   0 0 1 0
(1, 0, 1, 0)   0 1 0 0
(1, 0, 1, 1)   0 1 1 0
(1, 1, 0, 0)   0 0 0 0
(1, 1, 0, 1)   0 0 1 1
(1, 1, 1, 0)   0 1 1 0
(1, 1, 1, 1)   1 0 0 1
```

شکل 4 نمایش تمام حالات خروجی

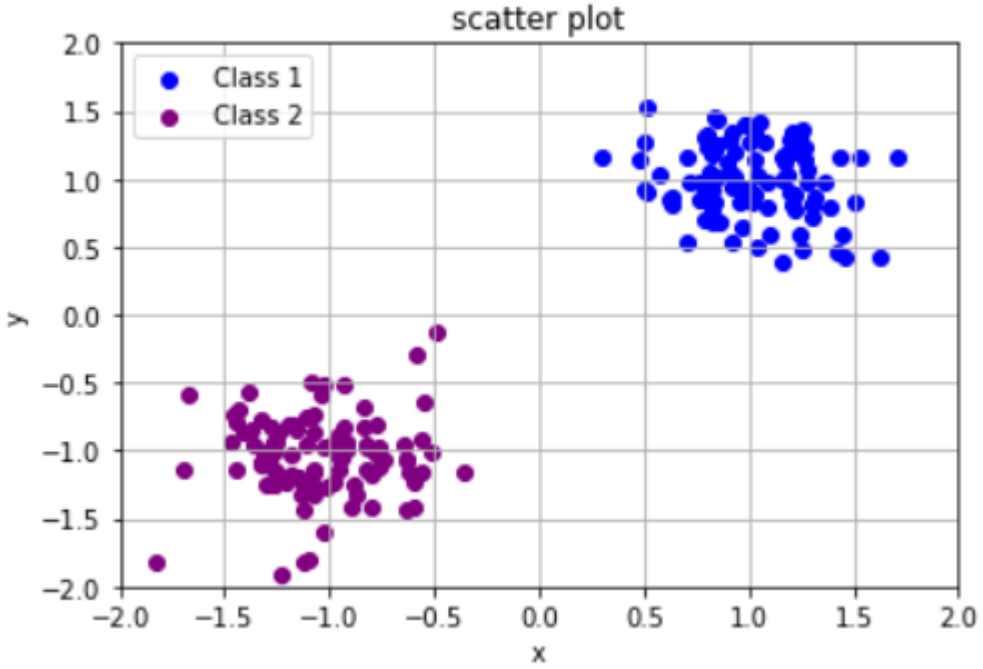
پاسخ ۲ - AdaLine and MadaLine

2-1. Adaline

الف)

دسته اول: شامل ۱۰۰ داده است، که متغیر x آن دارای میانگین ۱ و انحراف معیار۰.3 و متغیر yآن هم دارای میانگین ۱ و انحراف معیار ۰.3 است.
دسته دوم: شامل ۱۰۰ داده است، که متغیر x آن دارای میانگین ۱ - و انحراف معیار۰.3 و متغیر y آن هم دارای میانگین ۱ - و انحراف معیار ۰.3 است.
نمودار پراکندگی آنها به کل زیر می شود.

با کتابخانه numpy از تابع random.normal با کمک میانگین و انحراف معیار های داده شده، داده ها را تولید میکنیم. سپس با تابع scatter در matplotlib داده ها را رسم میکنیم.



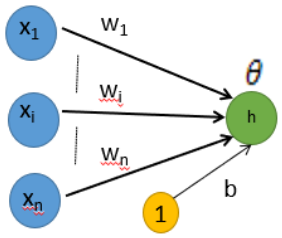
شکل 5 نمایش داده های تولید شده در حالت اول

مشاهده می شود که چون میانگین دو دسته داده از همدیگر دور است و انحراف معیار کوچکی دارند، داده ها فاصله خوبی از همدیگر دارند و بنظر می اید به خوبی قابل جداسازی باشند.

ب)

در روش Adaline یک لایه شبکه داریم که باید برای وزن های آن مقدار مناسب را اموزش دهیم. برای این کار ابتدا به صورت رندوم مقادیری را به وزن ها و بایاس میدهیم.

AdaLine is a type of one-layer network



(1)

$$\text{net} = \sum_{i=1}^{n} w_i x_i + b$$

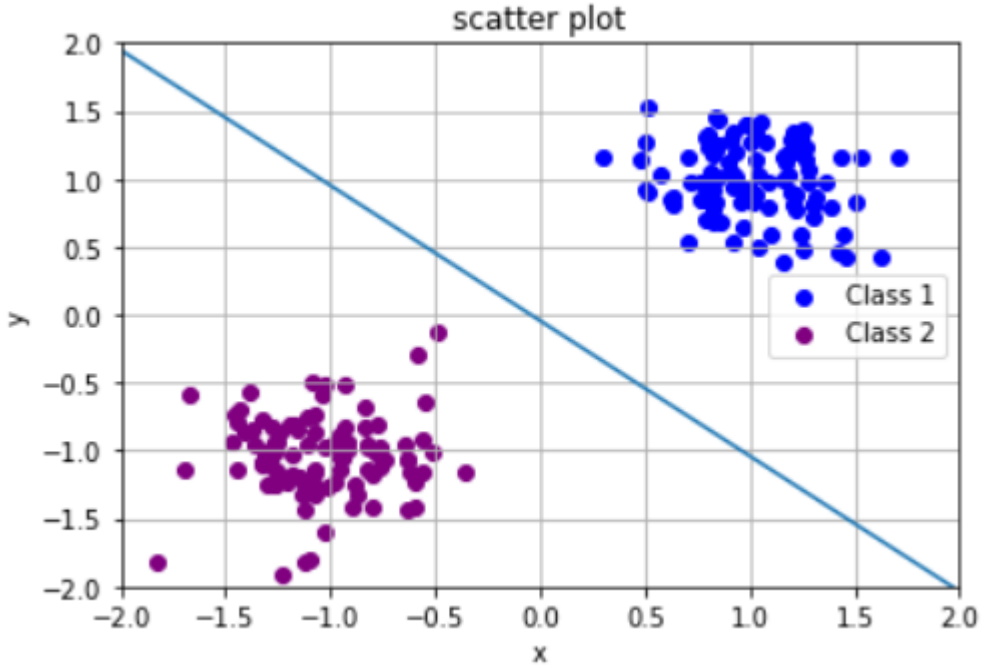
$$h = f(net) = \begin{cases} +1 & \text{net} \geq 0 \\ -1 & \text{net} < 0 \end{cases}$$

شکل 6 فرم کلی **adaline**

برای کلاس 1 تارگت 1- و برای کلاس 2 تارگت 1 انتخاب کرده و سپس یک دیتا فریم از x و y و تارگت درست میکنم.

سپس به تعداد epoch های مدنظر داده ها را به مدل میدهیم. دو روش point wise و batch برای اموزش داده ها وجود دارد. در این بخش داده ها را به صورت point wise به مدل دادیم. یعنی هر بار برای هر داده وزن ها و بایاس را اپدیت کردیم.

سپس در اخر با کمک وزن ها و بایاس خط جدا کننده را در کنار داده ها رسم میکنیم. میبینیم خط جدا کننده به خوبی دو کلاس را جدا کرده است.

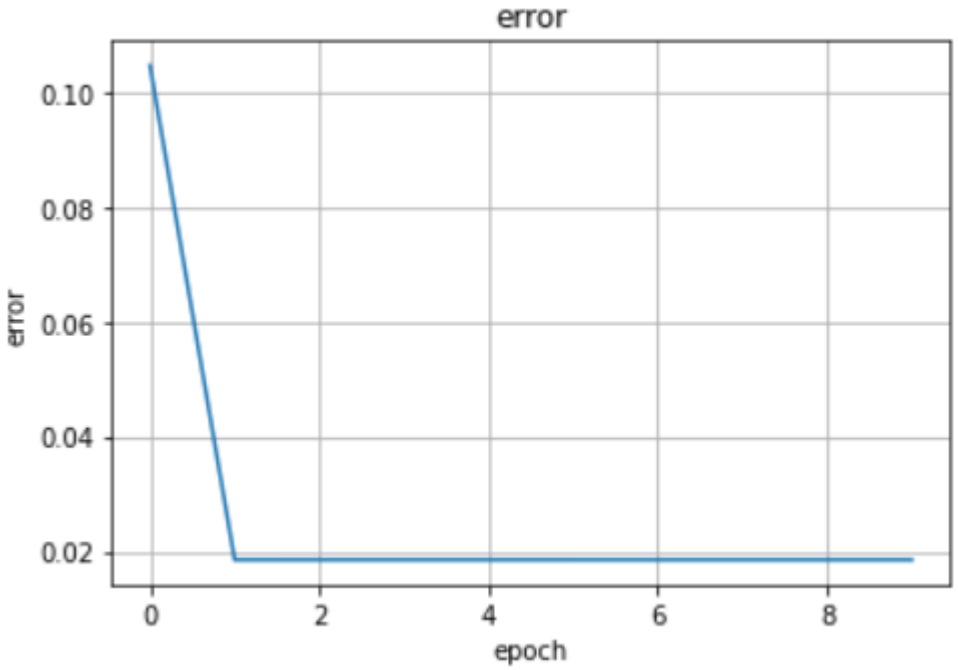


شکل 7 نمایش خط جدا کننده برای داده های حالت اول

دلیل اینکه داده ها به خوبی جدا شدند این است که میانگین و انحراف معیار کلاس ها سبب شده بود تا داده ها فاصله خوبی از یکدیگر داشته باشند. بنابراین این امکان را فراهم میکند تا بتوان با

یک خط کاملا دو کلاس را جدا کرد. از طرفی دو کلاس به نوعی باهم تقارن دارند یعنی توزیع و تعداد یکسان دارند. از طرفی خطا به سمت صفر میل میکند.

نمودار خطا را با فرمول $\frac{1}{2}(t-net)^2$ برای هر epoch به ازای همه داده ها نمایش میدهیم.



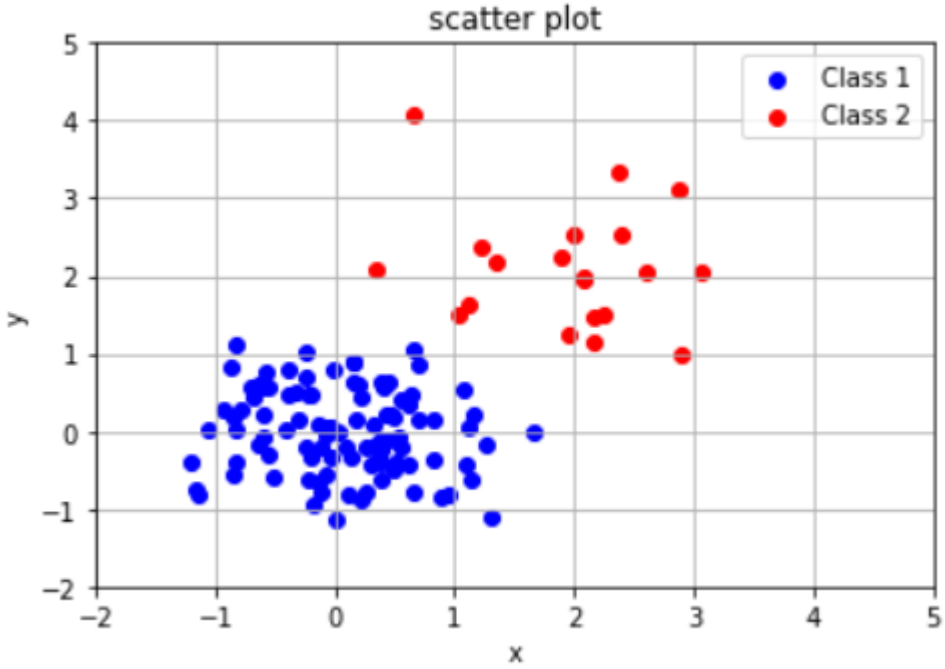
شکل 8 خطای به دست امده در هر **epoch** برای کل داده ها در حالت اول

ج)

حال برای دو دسته داده دیگر بند های قبل را تکرار میکنیم.

دسته اول: شامل ۱۰۰ داده است، که متغیر x آن دارای میانگین صفر و انحراف معیار ۰.6 و متغیر y آن هم دارای میانگین صفر و انحراف معیار ۰.6 است.
دسته دوم: شامل ۲۰ داده است، که متغیر x آن دارای میانگین 2 و انحراف معیار ۰.8 و متغیر yآن هم دارای میانگین 2 و انحراف معیار ۰.8 است.

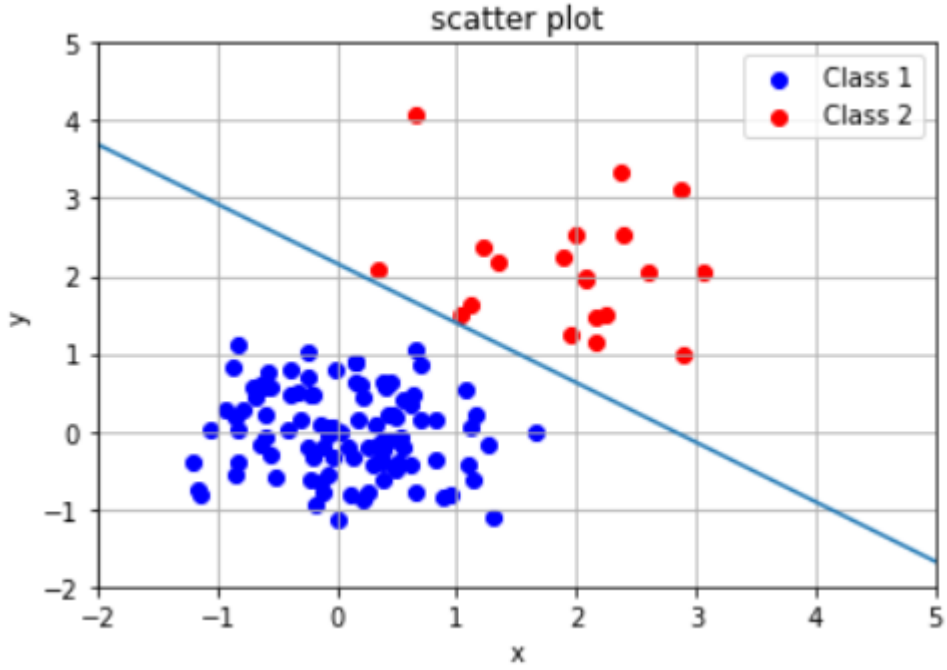
این بار میانگین و انحراف معیار سبب میشود تا داده ها با هم overlap داشته باشند و از طرفی تعداد داده های هر کلاس نیز برابر نیست. پس انتظار می رود که نتوانیم با یک خط دو کلاس را کاملا تفکیک کنیم.



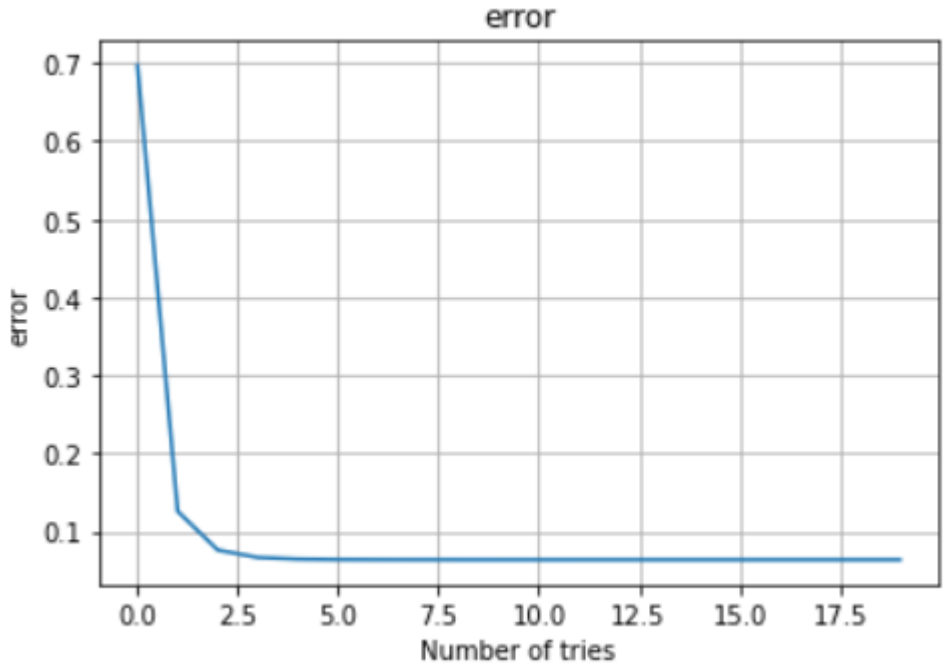
شکل 9 پراکندگی داده های حالت دوم

به روش قبل سعی میکنیم تا خط جدا کننده را پیدا کنیم.

پراکندگی داده ها و برابر نبودن تعداد انها و عدم تقارن باعث می شود تا الگوریتم Adaline به global optimal نرسد. بنابراین دقت پایین تری خواهیم داشت.



شکل 10 خط جدا کننده به دست امده برای داده های حالت دوم



شکل 11 نمودار خطای به دست امده برای داده های حالت دوم

همانطور که دیده می شود خطا در این حالت خیلی از حالت قبلی بیشتر است .

بنابراین نتیجه میگیریم اینکه تعداد داده های موجود از هر کلاس تقریبا برابر باشد تاثیر زیادی در پیدا کردن بهترین خط جدا کننده دارد.

2-2. Madaline

الف)

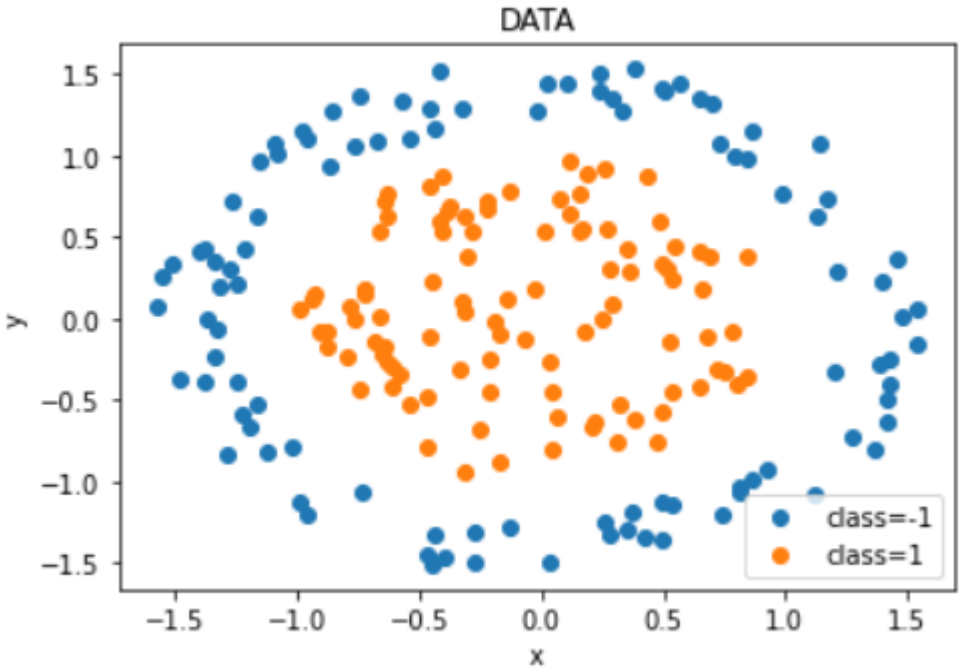
در این قسمت به بررسی دو الگوریتم MRI و MRII میپردازیم.
میدانیم شبکه Madaline از تعدادی لایه پنهان نورون Adaline و یک لایه خروجی نورون M&Pتشکیل شده است. در الگوریتم MRI تنها وزن نورونهای Adaline تنظیم میشود اما در MRII تمامی وزنها در حین train شدن به روزرسانی میشوند.
از آنجایی که الگوریتم MRII کاملتر بوده و در سالهای آتی به وجود آمده، در این بخش از تمرین از آن استفاده میکنیم.
توضیح کاملتر الگوریتم: MRII
- قدم اول: دادن مقدار اولیه به متغیرها
- قدم دوم: جمع وزندار ورودیها را برای همه نورونهای Adaline در تمامی لایه ها محاسبه میکنیم.
- قدم سوم: خروجی شبکه عصبی را به دست می آوریم.

- قدم چهارم: اگر خروجی شبکه عصبی با لیبل کلاس مد نظر (Target) یکسان است نیاز به به روزرسانی وزنها نداشته و به قدم بعد میرویم. در غیر این صورت، به ترتیب فاصله از صفر وزن تمامی نورون هایی که جمع وزندار ورودیهای آنها در محدوده مشخصی است را به روزرسانی میکنیم.
نحوه به روزرسانی: خروجی نورون را به حالت دیگر آن تغییر میدهیم و سپس خروجی شبکه را محاسبه میکنیم. اگر مقدار خطای ما کمتر شد، طبق قانون دلتا (با Target خروجی متفاوت نورون)
به تغییر وزنها میپردازیم.
- قدم پنجم: اگر شرط پایان train کردن صحیح است به ادامه کد میپردازیم. در غیر این صورت به
قدم دوم پرش میکنیم.

ب)

با کمک کتابخانه pandas و با استفاده از تابع read_csv دیتاست Madaline را در ژوپیتر لود میکنیم. سپس با تابع replace تارگت کلاس 0 را برابر 1 و تارگت کلاس 1 را برابر -1 قرار میدهیم.

حال با تابع scatter در matplotlib پراکندگی داده ها را نمایش میدهیم.

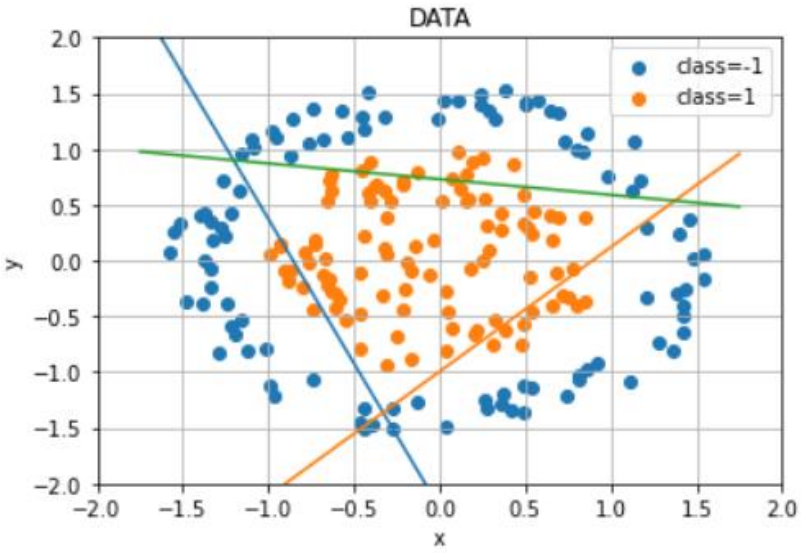


شکل 12 پراکندگی داده های دیتاست **madaline**

حال یک کلاس به اسم Mad که سه تابع دارد را مینویسیم. در تابع init مقادیر اولیه از جمله تعداد نورون ها و ایتریشن ها و لرنینگ ریت را میگیریم. سپس در تابع Fit به تعداد epoch ها

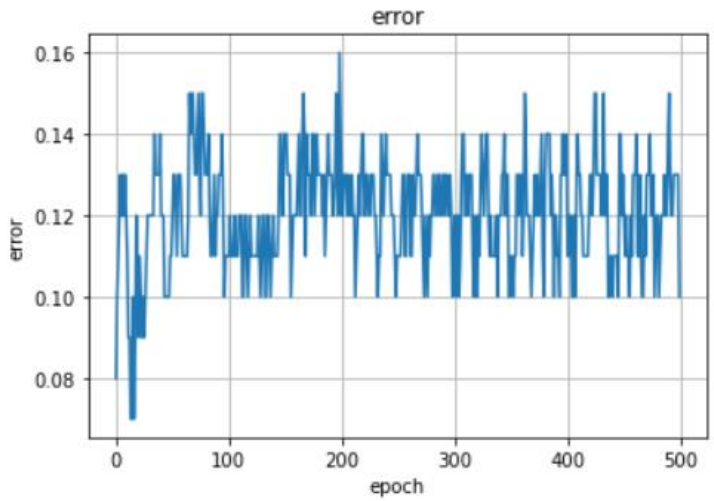
همه داده ها را به مدل می دهیم. به صورتی که به صورت point wise داده ها را میدهیم و طبق الگوریتم MRII وزن ها را اپدیت میکنیم. در تابع سوم show_error نمودار خطا و loss را نمایش میدهد. تابع accuracy نیز میزان دقت را نمایش میدهد. به صورتی که دقت را برابر تعداد تارگت های پیش بینی شده درست، تقسیم بر تعداد کل داده ها به دست می اورد. همچنین تابع sign به عنوان activation function عمل میکند. با تابع plot هم داده ها و خط های به دست امده را رسم میکنیم.

برای حالت سه نورون داریم:



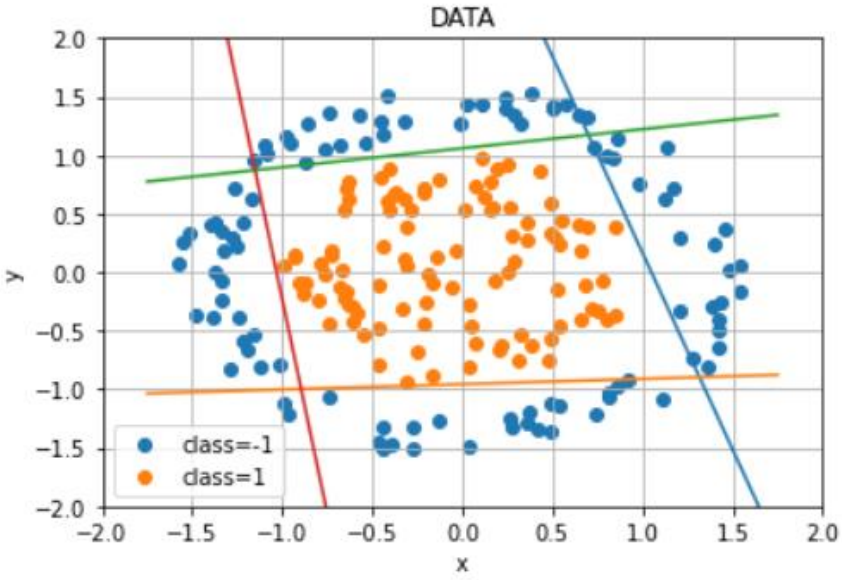
شکل 13 جداسازی داده ها با 3 نورون

```
accuracy=  86.5
```



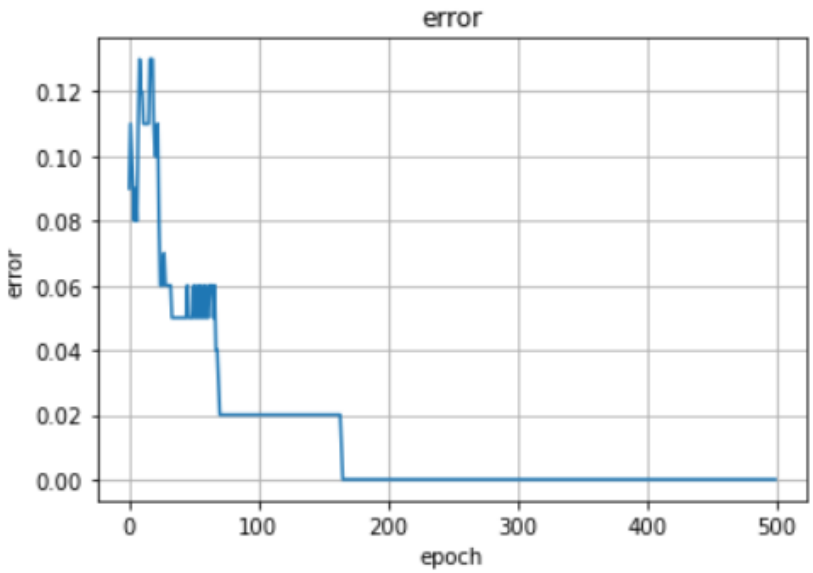
شکل 14 دقت و خطای به دست امده برای 3 نورون

برای حالت 4 نورون داریم:



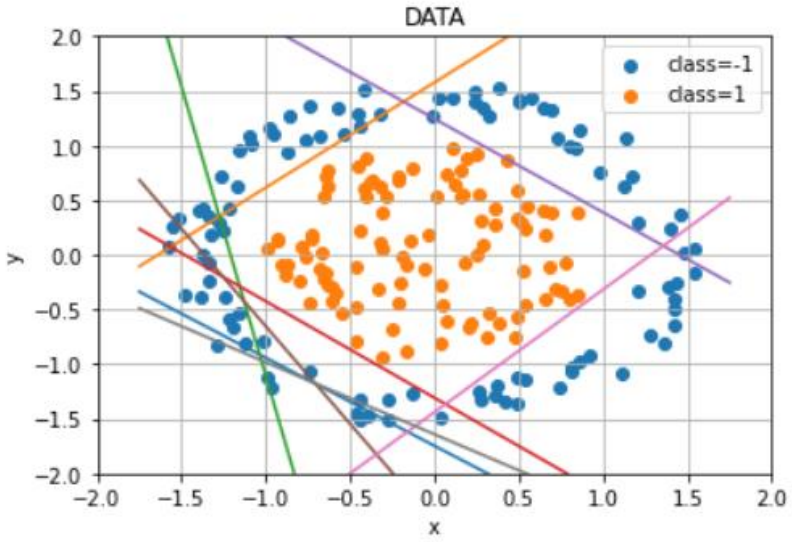
شکل 15 جداسازی داده ها با 4 نورون

```
accuracy=  100.0
```



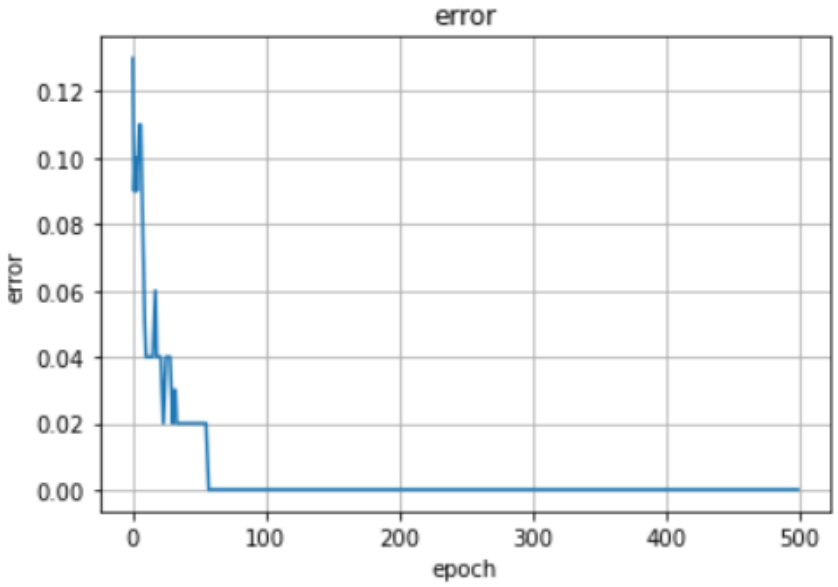
شکل 16 دقت و خطای به دست امده برای 4 نورون

برای حالت 8 نورون داریم:



شکل 17 جداسازی داده ها با 8 نورون

```
accuracy=  100.0
```



شکل 18 دقت و خطای به دست امده برای 8 نورون

ج) در هر سه حالت در 500 اپیاک نتایج را نمایش داده ایم.

انتظار ما از madaline این است که با n نورون n ضلعی بسازد اما میبینیم برای تعداد نورون های 3 و 4 سه و چهار ضلعی ساخته است اما برای نورون های بیشتر از 4، واقعا n ضلعی ساخته نمی شود. وجود نورون های بیشتر سبب یادگیری بهتر و افزایش دقت می شود اما مسئله میتواند به خوبی با 4 نورون نیز کار کند.

حال به جزئیات به صورت دقیق تری نگاه میکنیم.

در حالت سه نورون، دقت 86.5 درصد است و میبینیم خطا ثابت نمی شود و به صورت نوسانی همیشه خطا وجود دارد حتی اگر تعداد ایپاک را خیلی خیلی زیاد کنیم باز هم خطا کاهش نمی یابد.

در حالت چهار نورون، دقت 100 درصد است. از روی نمودار خطا میبینیم که تقریبا بعد از 170 ایپاک خطا به صفر متمایل شده است.

در حالت هشت نورون، دقت 100 درصد است. نمودار خطا نشان میدهد تقریبا بعد از 60 ایپاک خطا به صفر متمایل شده است.

حال از مقایسه این سه حالت میفهمیم که هرچه تعداد نورون های در لایه پنهانی بیشتر شود دقت افزایش می یابد و تعداد ایپاک کمتری نیاز است تا مدل اموزش ببیند.

پاسخ ۴ – MLP

(A

با کمک read_csv از کتابخانه pandas دیتاست را وارد برنامه میکنیم.

با تابع ()head میتوانیم 5 عضو اول این دیتاست را بخوانیم. سپس با تابع ()info مشخصات دیتاست را مبینیم. این مشخصات شامل تمامی ستون ها، تعداد داده های غیر null و جنس داده های هر ستون می شود. همچنین اطلاعاتی از حجم داده ها و تعداد سطر ها نیز دیده می شود.

مشخصات دیتاست به صورت زیر است:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21613 entries, 0 to 21612
Data columns (total 21 columns):
id               21613 non-null int64
date             21613 non-null object
price            21613 non-null float64
bedrooms         21613 non-null int64
bathrooms        21613 non-null float64
sqft_living      21613 non-null int64
sqft_lot         21613 non-null int64
floors           21613 non-null float64
waterfront       21613 non-null int64
view             21613 non-null int64
condition        21613 non-null int64
grade            21613 non-null int64
sqft_above       21613 non-null int64
sqft_basement    21613 non-null int64
yr_built         21613 non-null int64
yr_renovated     21613 non-null int64
zipcode          21613 non-null int64
lat              21613 non-null float64
long             21613 non-null float64
sqft_living15    21613 non-null int64
sqft_lot15       21613 non-null int64
dtypes: float64(5), int64(15), object(1)
memory usage: 3.5+ MB
```

شکل 19 تابع **info** در دیتاست **houses**

(B

با تابع ()isnull میتوانیم تعداد داده های NaN را در هر ستون مشاهده کنیم. همانطور که مشخص است تمامی داده ها دارای مقدار میباشند و دیتای NaN نداریم.

```
df.isna().sum()
```

```
id               0
date             0
price            0
bedrooms         0
bathrooms        0
sqft_living      0
sqft_lot         0
floors           0
waterfront       0
view             0
condition        0
grade            0
sqft_above       0
sqft_basement    0
yr_built         0
yr_renovated     0
zipcode          0
lat              0
long             0
sqft_living15    0
sqft_lot15       0
dtype: int64
```

شکل 20 تابع **isna** برای پیدا کردن تعداد داده های **NaN**

(C

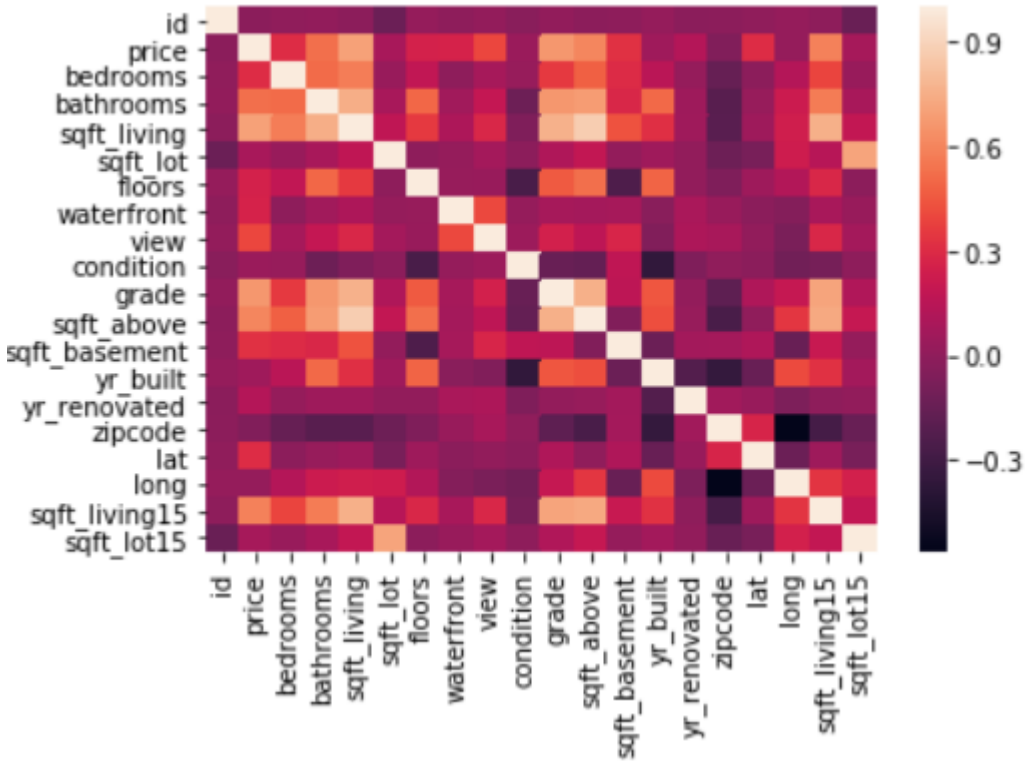
با تابع corr() میتوانیم هم بستگی ستون ها به یکدیگر را بررسی کنیم.

ماتریس مورد نظر به شکل زیر میباشد

	id	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition
id	1.000000	-0.016762	0.001286	0.005160	-0.012258	-0.132109	0.018525	-0.002721	0.011592	-0.023783
price	-0.016762	1.000000	0.308350	0.525138	0.702035	0.089661	0.256794	0.266369	0.397293	0.036362
bedrooms	0.001286	0.308350	1.000000	0.515884	0.576671	0.031703	0.175429	-0.006582	0.079532	0.028472
bathrooms	0.005160	0.525138	0.515884	1.000000	0.754665	0.087740	0.500653	0.063744	0.187737	-0.124982
sqft_living	-0.012258	0.702035	0.576671	0.754665	1.000000	0.172826	0.353949	0.103818	0.284611	-0.058753
sqft_lot	-0.132109	0.089661	0.031703	0.087740	0.172826	1.000000	-0.005201	0.021604	0.074710	-0.008958
floors	0.018525	0.256794	0.175429	0.500653	0.353949	-0.005201	1.000000	0.023698	0.029444	-0.263768
waterfront	-0.002721	0.266369	-0.006582	0.063744	0.103818	0.021604	0.023698	1.000000	0.401857	0.016653
view	0.011592	0.397293	0.079532	0.187737	0.284611	0.074710	0.029444	0.401857	1.000000	0.045990
condition	-0.023783	0.036362	0.028472	-0.124982	-0.058753	-0.008958	-0.263768	0.016653	0.045990	1.000000
grade	0.008130	0.667434	0.356967	0.664983	0.762704	0.113621	0.458183	0.082775	0.251321	-0.144674
sqft_above	-0.010842	0.605567	0.477600	0.685342	0.876597	0.183512	0.523885	0.072075	0.167649	-0.158214
sqft_basement	-0.005151	0.323816	0.303093	0.283770	0.435043	0.015286	-0.245705	0.080588	0.276947	0.174105
yr_built	0.021380	0.054012	0.154178	0.506019	0.318049	0.053080	0.489319	-0.026161	-0.053440	-0.361417
yr_renovated	-0.016907	0.126434	0.018841	0.050739	0.055363	0.007644	0.006338	0.092885	0.103917	-0.060618
zipcode	-0.008224	-0.053203	-0.152668	-0.203866	-0.199430	-0.129574	-0.059121	0.030285	0.084827	0.003026
lat	-0.001891	0.307003	-0.008931	0.024573	0.052529	-0.085683	0.049614	-0.014274	0.006157	-0.014941
long	0.020799	0.021626	0.129473	0.223042	0.240223	0.229521	0.125419	-0.041910	-0.078400	-0.106500
sqft_living15	-0.002901	0.585379	0.391638	0.568634	0.756420	0.144608	0.279885	0.086463	0.280439	-0.092824
sqft_lot15	-0.138798	0.082447	0.029244	0.087175	0.183286	0.718557	-0.011269	0.030703	0.072575	-0.003406

شکل 21 ماتریس **correlation**

همچنین میتوان این ماتریس را به شکل heatmap کشید که به صورت زیر می شود.



شکل 22**heatmap correlation**

طبق ماتریس بالا بیشترین کورلیشن با قیمت مربوط به پارامتر sqft_living میباشد که این مقدار حدود 0.7 میباشد.

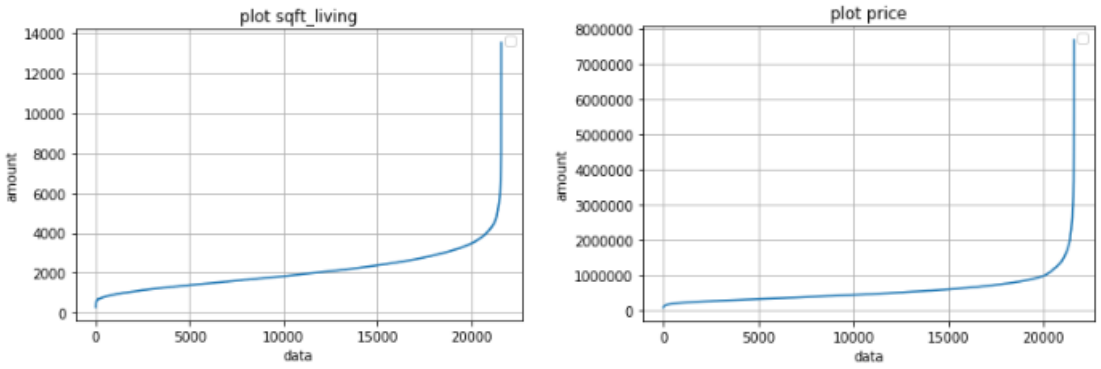
```
1 correlation['price']
```

```
id              -0.016762
price            1.000000
bedrooms         0.308350
bathrooms        0.525138
sqft_living      0.702035
sqft_lot         0.089661
floors           0.256794
waterfront       0.266369
view             0.397293
condition        0.036362
grade            0.667434
sqft_above       0.605567
sqft_basement    0.323816
yr_built         0.054012
yr_renovated     0.126434
zipcode         -0.053203
lat              0.307003
long             0.021626
sqft_living15    0.585379
sqft_lot15       0.082447
Name: price, dtype: float64
```

شکل 23 مقادیر **correlation** برای قیمت

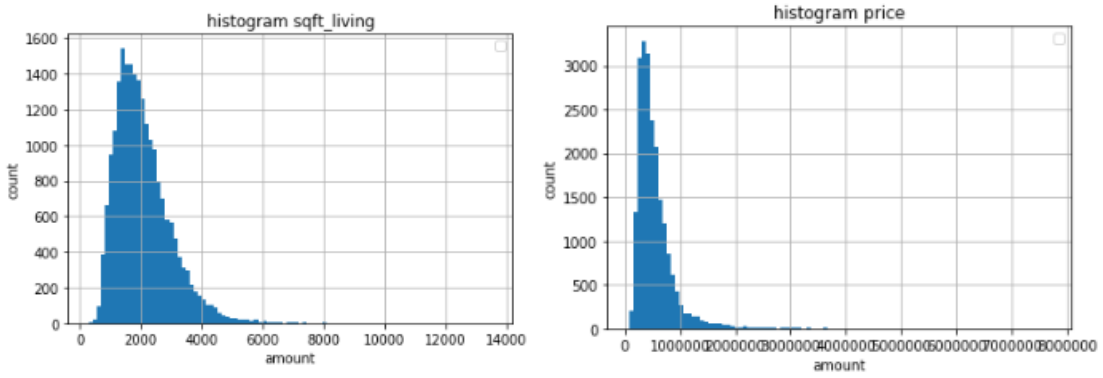
(D

نمودار های قیمت و sqft_living:



شکل 24 مقایسه نمودار اندازه داده ها به صورت **sort** شده

توزیع ها:



شکل 25 مقایسه هیستوگرام توزیع داده ها

همانطور که از نمودار ها مشخص است این دو متغیر دارای توزیع های نسبتا مشابهی هستند که این اتفاق کوریلیشن بالای آنها را توجیه میکند.

(E

داده های ستون date شامل ماه، سال، روز می شود که اگر ان را به شکل یک string در نظر بگیریم، میتوانیم به شکل زیر داده ها را جدا کنیم. سپس ستون date را حذف کرده و دو ستون year و month را اضافه میکنیم.

```
df['year'] = df['date'].str[0:4]
df['month'] = df['date'].str[4:6]
df.drop('date', axis=1, inplace=True)
```

شکل 26 نحوه تفکیک سال و ماه

ستون ها پس از اعمال تغییرات به شکل زیر است:

```
Index(['id', 'price', 'bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot',
       'floors', 'waterfront', 'view', 'condition', 'grade', 'sqft_above',
       'sqft_basement', 'yr_built', 'yr_renovated', 'zipcode', 'lat', 'long',
       'sqft_living15', 'sqft_lot15', 'year', 'month'],
      dtype='object')
```

شکل 27 اسامی ستون های باقی مانده بعد از تفکیک سال و ماه

(F

ابتدا از کتابخانه sklearn تابع train_test_split را import میکنیم. سپس مقدار test_size را 0.2 قرار میدهیم . y را برابر با ستون قیمت، و باقی ستون ها را برابر با فیچر ها در x قرار میدهیم و به تابع بالا ورودی میدهیم. خروجی تابع شامل x و y های اموزش و تست هستند.

(G

در این بخش داده های اسکیل شده بین -1 تا 1 قرار میگیرند و این به عملکرد بهتر الگوریتم کمک میکند. برای این کار از کتابخانه sklearn تابع preprocessing را ایمورت میکنیم. با تابع

MinMaxScaler ابتدا تابع را با x_train فیت میکنیم تا برحسب داده های ورودی اسکیل کند، سپس داده های تست و اموزش را با پارامتر های فیت شده، به محیط جدید میبریم و در محدوده -1 تا 1 قرار میگیرند.

(H

از کتابخانه keras توابع sequential, Dense, Activation, Adam را به برنامه ایمپورت میکنیم. حال با تابع add در کلاس sequential میتوانیم لایه اضافه کنیم. بنابراین یک لایه ابتدایی، دو لایه پنهان و یک لایه نهایی اضافه میکنیم. چون 20 عدد فیچر داریم بنابراین ورودی سه لایه اول را 20 عدد قرار میدهیم و چون خروجی لایه باید قیمت باشد پس یک خروجی داریم. بنابراین سه لایه اول هرکدام 20 خروجی و لایه اخر 1 خروجی دارد. در این مرحله نوع activation function انتخاب شده اهمیت دارد.

حال باید مدل را کامپایل کنیم که در آن نوع optimizer و loss اهمیت دارد.

سپس با داده های آموزش ، مدل را train میکنیم و در نهایت با داده های تست، مقادیری را پیش بینی میکنیم.

برای مثال میتوان به شکل زیر یک شبکه عصبی mlp را تولید کرد.

```
model_1 = Sequential()
model_1.add(Dense(20 , activation = "relu"))
model_1.add(Dense(20, activation = "relu"))
model_1.add(Dense(20, activation = "relu"))
model_1.add(Dense(1))
model_1.compile(optimizer='SGD',loss='mae')
```

شکل 28 یک نمونه از نحوه ایجاد شبکه **mlp**

(I

برای activation function گزینه های متعددی از جمله relu , sigmoid , softmax , softplus ,softsign ,tanh, selu , elu, exponential وجود دارد.

چندین optimizer موجود است که از جمله انها SGD , RMSprop , Adam , Adadelta , Adagrad , Adamax , Nadam , Ftrl میباشند.

همچنین در تابع losses در keras انواع مختلفی یافت می شود.

حال با توجه به اینکه گزینه های زیادی برای انتخاب وجود دارد باید متناسب با نوع فیچر اپشن های بهتر را انتخاب کرد.

از optimizer های Adam و Adagrad و همچنین از loss های mae و mean_squared_logaritmic_error و activation function از نوع relu استفاده میکنیم.

Mae یا همان mean absolute error ، ابتدا قدر مطلق اختلاف مقدار پیش بینی شده را از مقدار اصلی حساب میکند، سپس بین همه مقادیر به دست امده میانگین میگیرد و به عنوان خطای ان مجموعه داده خروجی میدهد.

$$\mathrm{MAE} = \frac{\sum_{i=1}^{n} |y_i - x_i|}{n}$$

MAE = mean absolute error
y_i = prediction
x_i = true value
n = total number of data points

شکل 29 فرمول **mae**

mean_squared_logaritmic_error ابتدا لگاریتم مقدار پیش بینی شده و مقدار اصلی به علاوه 1 را پیدا میکند و سپس به توان دو میرساند و از ان میانگین میگیرد و خروجی میدهد.

$$L(y, \hat{y}) = \frac{1}{N} \sum_{i=0}^{N} (\log(y_i + 1) - \log(\hat{y}_i + 1))^2$$

شکل 30 فرمول **msle**

الگوریتم Adam یک روش آماری کاهش گرادیان (stochastic gradient descent)میباشد که بر مبنای تخمین ممان(مشتق) مرتبه ی اول و مرتبه دوم کار میکند.

این روش از نظر محاسباتی بهینه است و نسبت به تغییر گرادیان به صورت مورب حساسیت ندارد که این باعث میشود که الگوریتمی مناسب برای داده هایی با تعداد پارامتر های زیاد باشد

الگوریتم Adagrad الگوریتمی میباشد که ضریب یادگیری را با توجه به نرخ به روز رسانی هر پارامتر تغییر میدهد، به این صورت که هرچقدر نرخ به روز رسانی یک پارامتر بیشتر باشد این الگوریم با ضریب کمتری آن را تغییر میدهد که این امر باعث fine tune شدن پارامتر هایی با تاثیر بیشتر میشود.

(J

حال دو مدل را با اپشن هایی که بالا توضیح دادیم تولید میکنیم و برای آنها loss , loss validation را حساب میکنیم.

مدل اول:

از optimizer = relu و loss=mae استفاده شده است و مدل در 250 ایپاک اموزش میبیند.

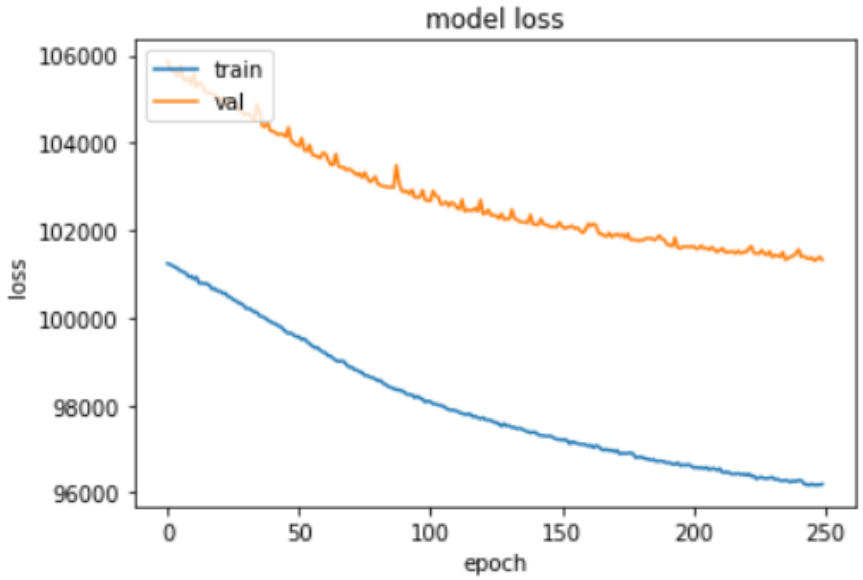
```
model_1 = Sequential()
model_1.add(Dense(20 , activation = "relu"))
model_1.add(Dense(20, activation = "relu"))
model_1.add(Dense(20, activation = "relu"))
model_1.add(Dense(1))
model_1.compile(optimizer='Adam',loss='mae')
```

```
model_1.fit(x = X_train,y = y_train, validation_data=(X_test,y_test), epochs = 250, batch_size = 32)
Epoch 242/250
541/541 [==============================] - 1s 1ms/step - loss: 96280.2969 - val_loss: 101395.0391
Epoch 243/250
541/541 [==============================] - 1s 2ms/step - loss: 96209.4453 - val_loss: 101402.7266
Epoch 244/250
541/541 [==============================] - 1s 1ms/step - loss: 96184.5703 - val_loss: 101371.5312
Epoch 245/250
541/541 [==============================] - 1s 2ms/step - loss: 96172.9297 - val_loss: 101346.2734
Epoch 246/250
541/541 [==============================] - 1s 1ms/step - loss: 96199.4922 - val_loss: 101377.5234
Epoch 247/250
541/541 [==============================] - 1s 2ms/step - loss: 96175.4531 - val_loss: 101306.2891
Epoch 248/250
541/541 [==============================] - 1s 2ms/step - loss: 96178.5781 - val_loss: 101340.3906
Epoch 249/250
541/541 [==============================] - 1s 2ms/step - loss: 96181.6016 - val_loss: 101391.2656
Epoch 250/250
541/541 [==============================] - 1s 1ms/step - loss: 96193.8281 - val_loss: 101327.5547

<keras.callbacks.History at 0x1b6d2512be0>
```

شکل 31 مدل اول

میبینیم که در نمودار loss مدل اول بعد از 250 ایپاک loss خیلی کاهش یافته اما همچنان این مقدار بسیار بسیار زیاد است و همین سبب پیش بینی های نادرست می شود.



شکل 32 نمودار **loss** مدل اول

مدل دوم:

از optimizer=Adagrad و loss=msle استفاده شده است. مدل در 500 ایپاک اموزش میبیند.

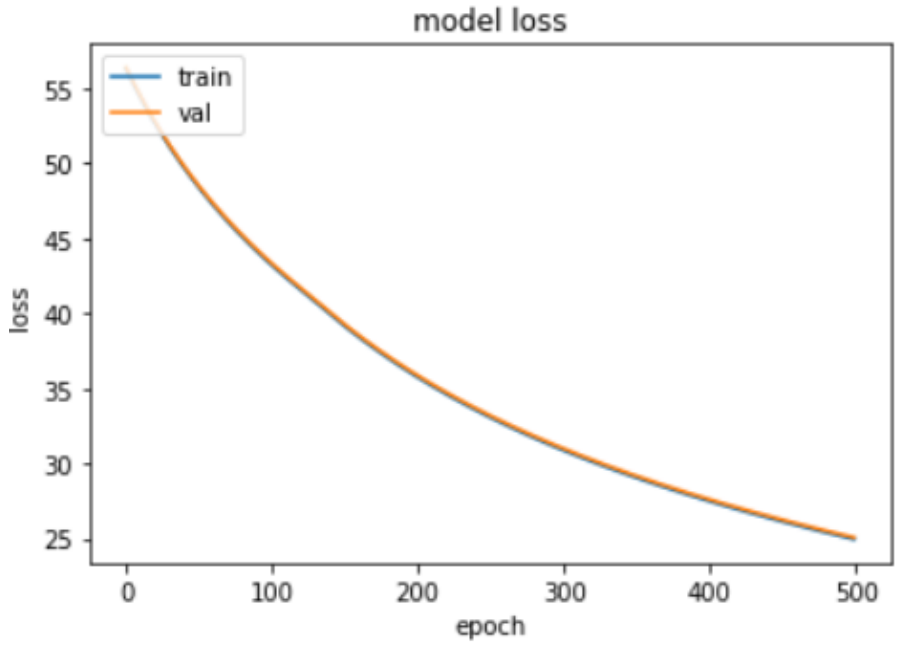
```
model_2 = Sequential()
model_2.add(Dense(20 , activation = "relu"))
model_2.add(Dense(20, activation = "relu"))
model_2.add(Dense(20, activation = "relu"))
model_2.add(Dense(1))
model_2.compile(optimizer='Adagrad',loss='mean_squared_logarithmic_error')
```

```
model_2.fit(x = X_train,y = y_train, validation_data=(X_test,y_test), epochs = 500, batch_size = 32)
Epoch 492/500
541/541 [==============================] - 1s 1ms/step - loss: 25.1402 - val_loss: 25.2609
Epoch 493/500
541/541 [==============================] - 1s 1ms/step - loss: 25.1172 - val_loss: 25.2379
Epoch 494/500
541/541 [==============================] - 1s 2ms/step - loss: 25.0943 - val_loss: 25.2149
Epoch 495/500
541/541 [==============================] - 1s 1ms/step - loss: 25.0714 - val_loss: 25.1920
Epoch 496/500
541/541 [==============================] - 1s 1ms/step - loss: 25.0486 - val_loss: 25.1691
Epoch 497/500
541/541 [==============================] - 1s 2ms/step - loss: 25.0258 - val_loss: 25.1462
Epoch 498/500
541/541 [==============================] - 1s 2ms/step - loss: 25.0030 - val_loss: 25.1235
Epoch 499/500
541/541 [==============================] - 1s 2ms/step - loss: 24.9803 - val_loss: 25.1008
Epoch 500/500
541/541 [==============================] - 1s 1ms/step - loss: 24.9577 - val_loss: 25.0781

<keras.callbacks.History at 0x1b6c31e87b8>
```

شکل 33 مدل دوم

در این حالت خطای لگاریتمی به شدت کاهش یافته اما به دلیل اینکه مقیاس ما لگاریتم است همچنان خطا مقدار زیادی دارد و احتمال میرود پیش بینی ها خطای زیادی داشته باشند.



شکل 34 نمودار **loss** مدل دوم

از مشاهده نمودار های loss نتیجه میگیریم با توجه به اینکه loss و validation loss هر دو در حال کاهش هستند، میتوانیم همچنان تعداد ایپاک ها را زیاد کنیم تا به دقت بیشتری برسیم و نمودار ها واگرا نمی شوند بنابر این تا اینجای کار overfit رخ نداده است.

(K

حال به صورت رندوم 5 داده تست را نشان میدهیم و قیمت پیش بینی شده و قیمت اصلی انها را نمایش میدهیم. در این پیش بینی از مدل اول استفاده شده است. میبینیم نتایج دارای خطا هستند اما محدوده قابل قبولی دارند.

```
y_predict = model_1.predict(X_test)
samples = np.random.randint(1,X_test.shape[0],5)
for i in samples:
    print("perdicted" ,[i]," : ",y_predict[i])
    print("real" ,[i]," : ",y_test[i],"\n -------")
```

```
136/136 [==============================] - 0s 907us/step
perdicted [1283]  :  [1050087.1]
real [1283]  :  2205000.0
 -------
perdicted [2584]  :  [512211.22]
real [2584]  :  399000.0
 -------
perdicted [2723]  :  [314124.9]
real [2723]  :  342000.0
 -------
perdicted [807]  :  [769725.44]
real [807]  :  965000.0
 -------
perdicted [3713]  :  [257704.95]
real [3713]  :  264950.0
 -------
```

شکل 35 مقادیر پیش بینی شده برای 5 خانه رندوم