به نام خدا

دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

درس شبکه‌های عصبی و یادگیری عمیق

تمرین اول

نام و نام خانوادگی	سارا رستمی – محمدامین شاهچراغی
شماره دانشجویی	۸۱۰۱۰۰۳۵۵ - ۸۱۰۱۹۹۱۹۶
تاریخ ارسال گزارش	۱۴۰۱/۰۸/۰۷

فهرست

شکل‌ها

جدول‌ها

پاسخ ۱. شبکه عصبی Mccullotch-Pitts

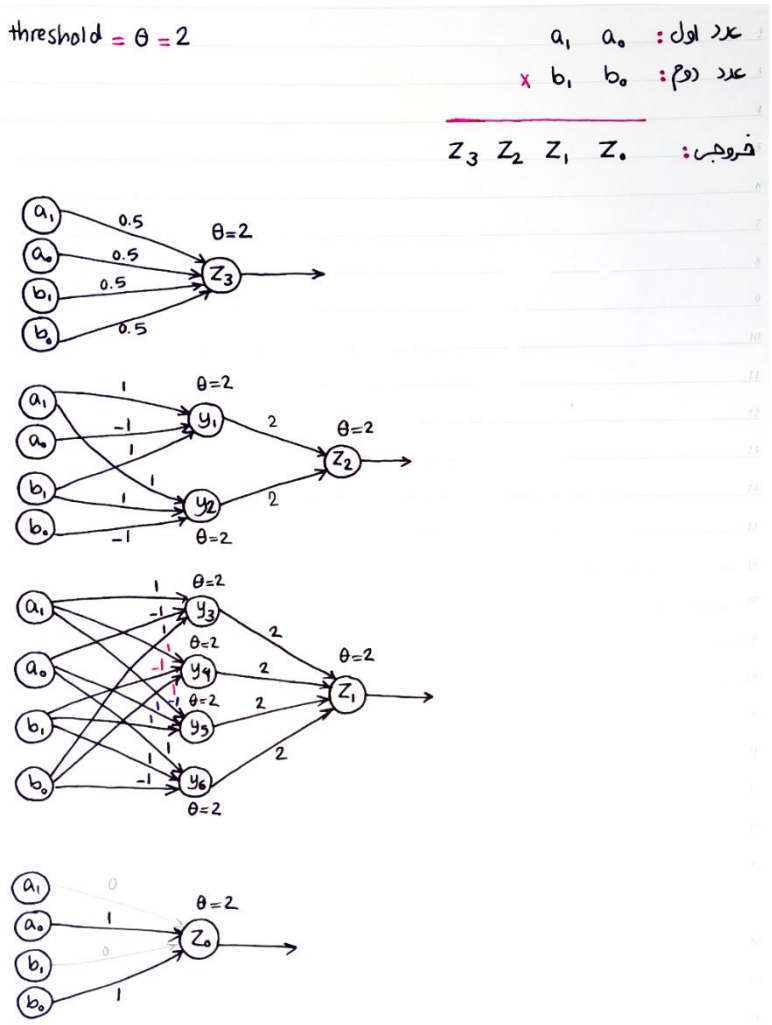
۱-۱. ضرب‌کننده‌ی باینری دوبیتی

الف) ابتدا جدول درستی یک ضرب‌کننده‌ی دوبیتی را رسم کرده و معادله‌ی خروجی‌ها را از طریق جدول کارنو بدست می‌آوریم. معادله‌ی هر یک از بیت‌های خروجی به صورت زیر می‌باشد:

$$Z_3 = a_1 a_0 b_1 b_0 \quad \text{و} \quad Z_2 = a_1 \bar{a}_0 b_1 + a_1 b_1 \bar{b}_0$$

$$Z_1 = a_1 \bar{a}_0 b_0 + a_1 \bar{b}_1 b_0 + \bar{a}_1 a_0 b_1 + a_0 b_1 \bar{b}_0 \quad \text{و} \quad Z_0 = a_0 b_0$$

با استفاده از معادلات بالا شبکه را رسم کرده و وزن یال‌ها را طبق شکل ۱ قرار می‌دهیم.



شکل ۱ – شبکه‌عصبی یک ضرب‌کننده‌ی باینری دوبیتی با استفاده از نورون **M&P**

در شکل ۱ ، وزن یال‌هایی که رسم نشده اند(یا کمرنگ رسم شده‌اند مثل دو یال متصل به نورون z_0) برابر

با صفر است. به طور مثال از ورودی b_0 یالی به نورون y_1 رسم نشده که منظور همان یالی با وزن صفر است.

ب) تابعی(به نام mp_neuron) برای پیاده‌سازی یک نورون M&P تعریف می‌کنیم که ورودی‌های شبکه و وزن‌های شبکه را به صورت numpy array و همچنین threshold نورون را به عنوان آرگومان‌های ورودی می گیرد و طبق تابع پله با میزان threshold خروجی ۰ و ۱ برمی‌گرداند. سپس یک تابع برای ضرب‌کننده‌ی دوبیتی پیاده‌سازی کردیم که ورودی شبکه (یعنی دو عدد دوبیتی) را به صورت یک آرایه با اندازه‌ی ۴ به عنوان آرگومان ورودی می گیرد. در داخل این تابع ۱۰ بار تابع mp_neuron با ورودی ها و وزن‌های مشخص شده در شکل ۱ فراخوانی می‌شود. هر فراخوانی این تابع خروجی یکی از نورون های y و z را محاسبه می‌کند. در نهایت تابع two_bit_multiplier خروجی های شبکه (یعنی Z_0 و Z_1 و Z_2 و Z_3) را برمی‌گرداند.

نتیجه‌ی فراخوانی تابع با تمام حالت جدول درستی برای دو عدد دوبیتی را در شکل ۲ مشاهده می‌کنید.

```
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]
```

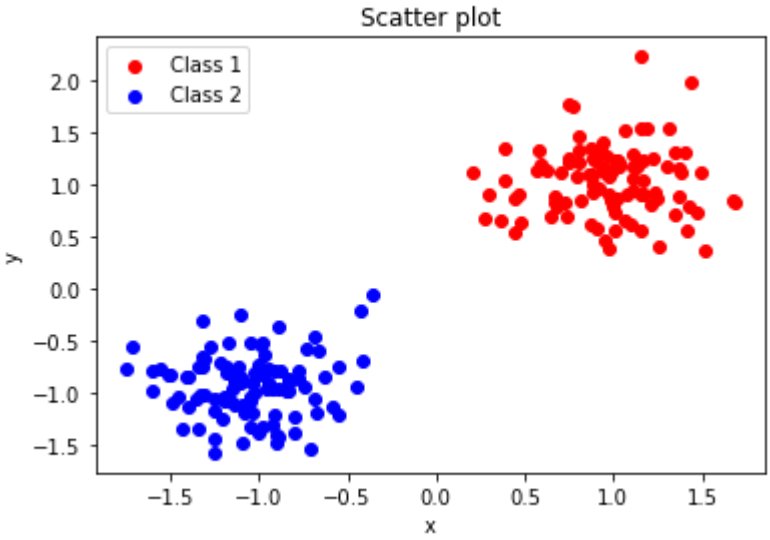
شکل ۲ - تمامی حالات ضرب‌کننده‌ی باینری دو بیتی **M&P**

پاسخ ۲ – AdaLine and MAdaLine

۲-۱. AdaLine

الف) ابتدا داده‌ها را به طور تصادفی از توزیع‌های نرمال با میانگین و انحراف معیار مذکور generate می‌کنیم و برچسب ۱ و ۱- را به ترتیب به داده‌های کلاس ۱ و ۲ assign می‌کنیم.

نمودار پراکندگی دو دسته داده را در شکل ۳ مشاهده می‌کنید.



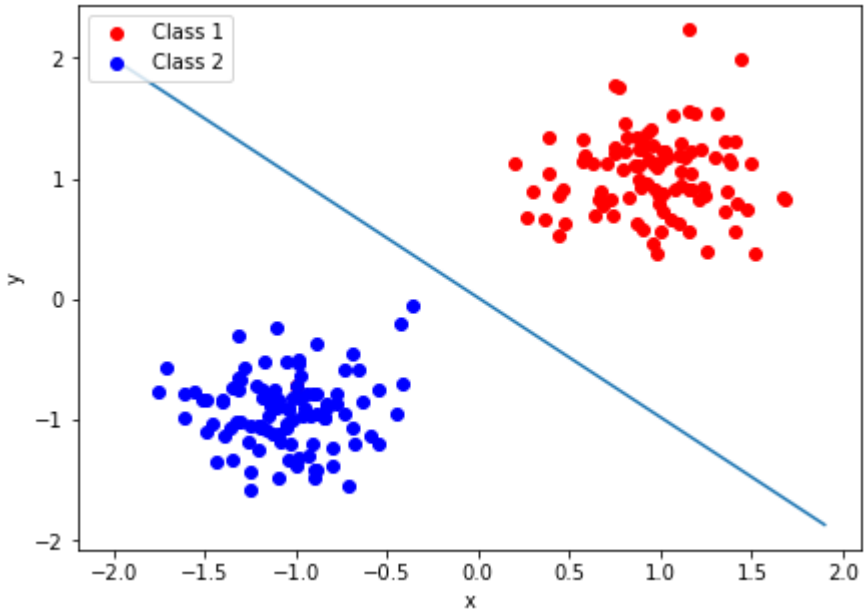
شکل ۳ - نمودار پراکندگی دو دسته داده بخش الف

ب) آموزش شبکه را تا زمانی ادامه می‌دهیم که یا خطای همه‌ی نمونه‌های train کمتر از یک مقدار مشخصی باشد(مقدار default ما در اینجا threshold = 0.5 می‌باشد) یا تعداد epochها به ۵۰ برسد. آرگومان‌های ورودی تابع Adaline، داده‌ی ورودی شبکه، آلفا (learning rate) و threshold برای شرط خاتمه‌ی شبکه می‌باشد. این تابع به عنوان خروجی میزان خطا در هر epoch، وزن‌های یالها و bias را در انتهای یادگیری را برمی‌گرداند.

به ازای مقادیر مختلف برای پارامترهای آلفا و threshold ، شبکه را با داده را train کردیم و در نهایت مقدار alpha = 0.001 و threshold = 0.33 را برای آموزش شبکه برگزیدیم که در نتیجه‌ی آن، شبکه‌ی ما طی ۱۱ تا epoch آموزش یافت. معادله‌ی خط جداساز بدست آمده توسط شبکه به صورت زیر می‌باشد:

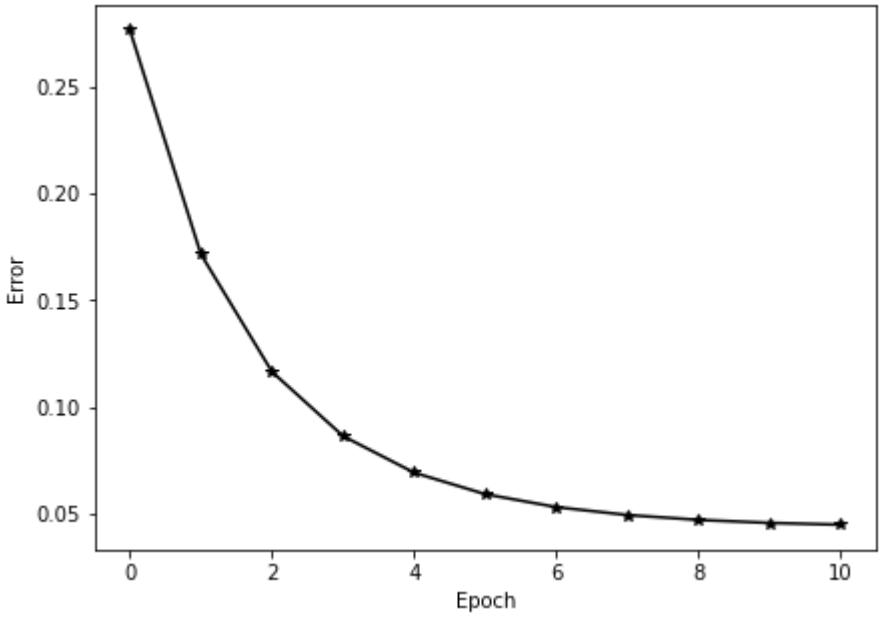
$$z = 0.45406417 * x + 0.49104063 * y - 0.00368949$$

این خط جداساز را در شکل ۴ مشاهده می‌کنید.



شکل ۴ - خط جداساز یادگیری شده توسط شبکه **AdaLine** برای داده‌های الف

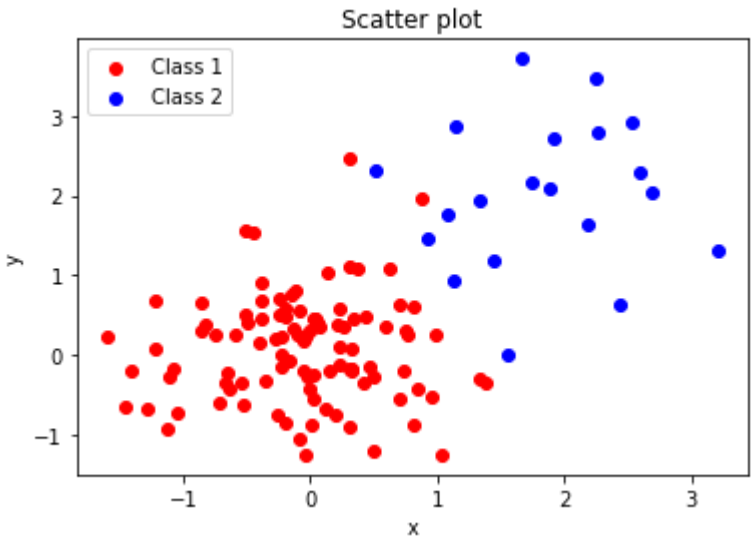
نمودار تغییرات خطا بر حسب epoch را در شکل ۵ مشاهده می‌کنید.



شکل ۵ - نمودار تغییرات خطای **mean square error** برحسب **epoch** برای داده‌های الف

همانطور که در نمودار شکل ۵ مشاهده می‌کنید، میزان خطا با گذشت epochها، از حدود ۰.۳ به حدود ۰.۰۴ (عددی نزدیک به صفر) کاهش پیدا می‌کند. پس این‌طور به نظر می‌رسد که شبکه به درستی آموزش یافته. دقت شود که این نمودار نشان‌دهنده‌ی خطای آخرین نمونه در هر epoch می‌باشد.

ج) نمودار پراکندگی دو دسته داده‌ی جدید را در شکل۶ مشاهده می‌کنید.



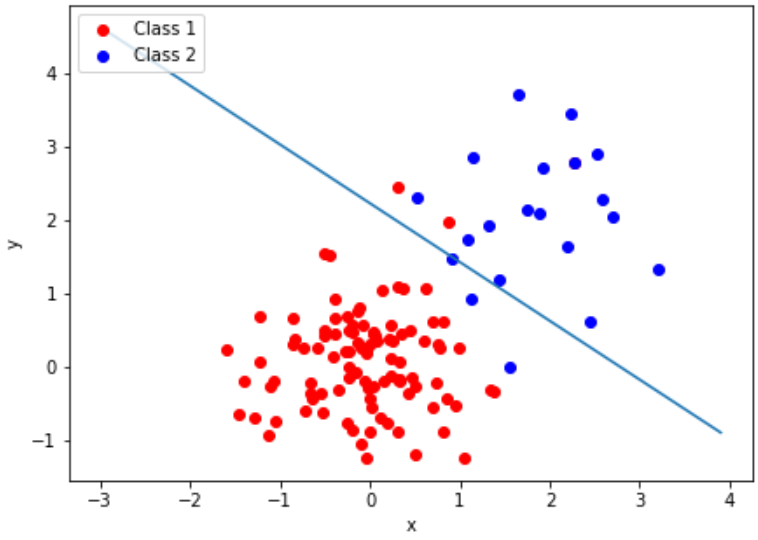
شکل ۶ - نمودار پراکنندگی دو دسته داده بخش ج

این بار الگوریتم AdaLine (حتی با تغییر مقادیر پارامتر های آلفا و threshold) با شرط خاتمه‌ی threshold همگرا نمی‌شود و ۵۰ تا epoch را می‌گذراند. همانطور که انتظار می‌رفت از آنجایی که تعداد داده‌های دو کلاس نامتوازن(unbalanced) می‌باشد، روش AdaLine نتوانست به درستی دو دسته را از هم جدا سازد و حاوی تعدادی خطاست. همچنین از آنجایی که داده‌ها پراکنندگی زیادی دارند و قابل جداسازی با یک خط نمی‌باشند، AdaLine در چنین شرایطی به خوبی قادر به جداسازی داده‌ها نمی‌باشد.

معادله‌ی خط جداساز یادگرفته شده توسط شبکه به صورت زیر می‌باشد:

$$z = -0.39622478 * x - 0.3176017 * y - 0.88442759$$

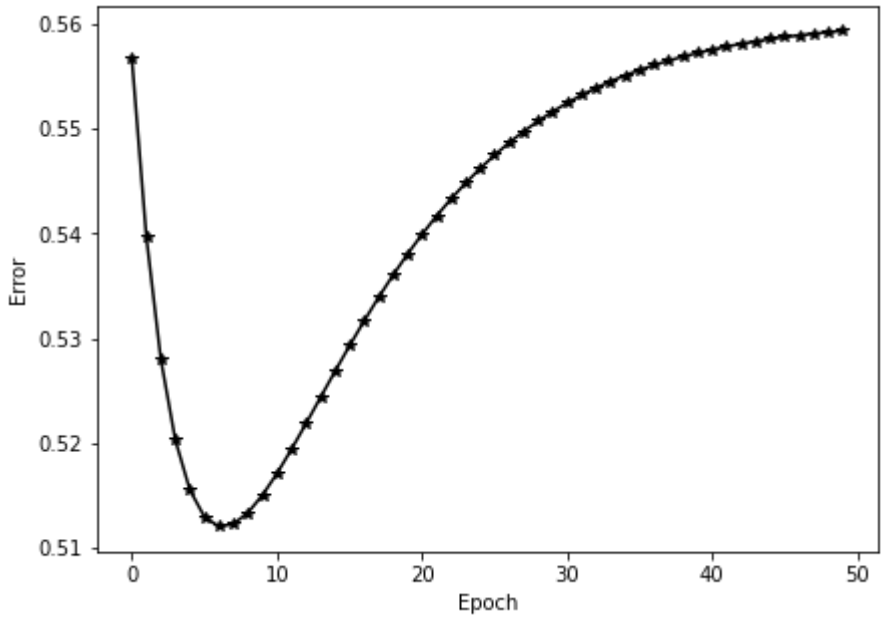
با مشاهده‌ی شکل ۷ می‌توان خط جداساز یادگرفته توسط شبکه را دید.



شکل ۷ - خط جداساز یادگیری شده توسط شبکه

نمودار تغییرات خطا بر حسب epoch را در شکل ۸ مشاهده می‌کنید. میزان خطا کلا بالاست و روند نزولی ندارد.

به دلایل ذکر شده در بالا، نمودار تغییرات خطا بر حسب epoch، مانند بخش الف نزولی نمی‌شود چرا که AdaLine نمی‌تواند خطی پیدا کند که قادر به جداسازی داده‌ها باشد.



شکل ۸ - نمودار تغییرات خطای **mean square error** برحسب **epoch** برای داده‌های ج

MAdaLine .۲-۲

الف) الگوریتم MRI (که در واقع همان ورژن اصلی MAdaLine می‌باشد) یکی از الگوریتم‌های Madaline است.

در این الگوریتم در لایه خروجی وزن ها ثابت هستند و تنها وزن ها و بایاس های لایه‌های میانی تنظیم میشوند. در این الگوریتم ابتدا مقدار اولیه‌ی کوچک برای وزن‌ها در نظر میگیریم. همچنین نرخ یادگیری مقدار کوچکی دارد. سپس تا زمانی که شرط توقف (ثابت شدن تقریبی وزن ها در دو تکرار متوالی یا طی تعداد مشخصی Epoch) برقرار نباشد مراحل تکرار میشود.

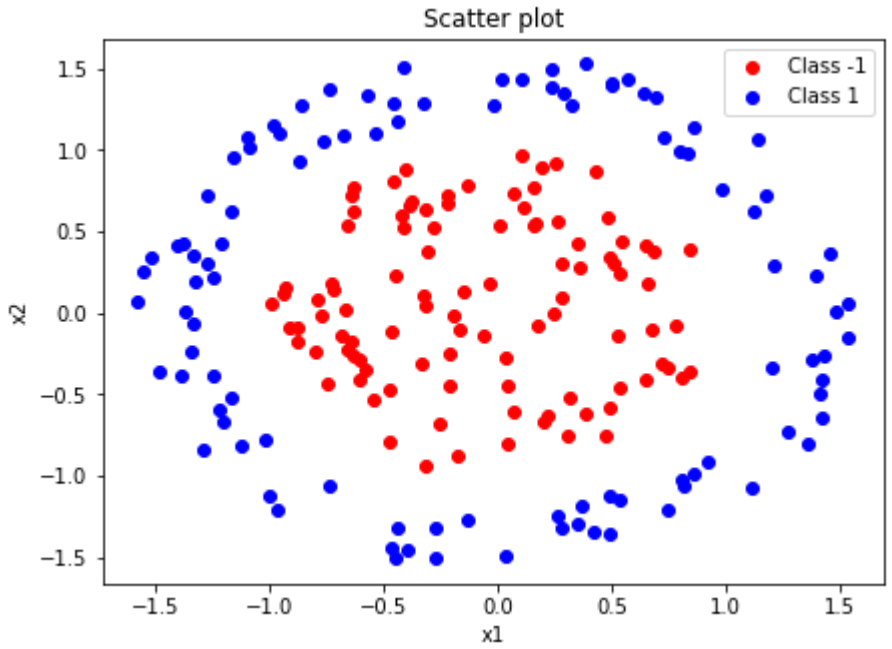
حال مقدار شبکه برای هر نورون Adaline را حساب میکنیم و خروجی را پس از عبور از تابع فعالساز به دست می‌آوریم. اگر خروجی محاسبه شده با تارگت متناظر برابر بود، وزن ها آپدیت نمی شوند در غیر این صورت اگر تارگت برابر یک باشد وزن های مربوط به نورونی که مقدار net ورودی از همه به صفر نزدیک تر است با استفاده از الگوریتم گرادیان کاهشی آپدیت می‌شود. اما اگر تارگت مربوطه برابر با ۱- باشد، وزن های تمام نورون هایی که مقدار net آنها مثبت است آپدیت می‌شود. وزن و بایاس لایه نهایی را هم بر اساس تعداد نورون لایه میانی طوری تنظیم میکنیم که گیت OR تشکیل دهد .

ب) ابتدا داده ها را میخوانیم و در دیتافریم ذخیره میکنیم سپس به آنهایی متعلق به گروه ۱ هستند برچسب -۱ و به آنهایی که متعلق به گروه ۲ هستند برچسب ۱ میدهیم .شکل زیر مجموعه داده را پس از این عملیات نشان میدهد .

	0	1	2
0	-0.642823	0.720606	-1.0
1	-0.218126	0.677263	-1.0
2	-0.582930	-0.347496	-1.0
3	0.285127	0.091750	-1.0
4	-0.335577	-0.313893	-1.0
...	...	...	...
195	0.744066	-1.206548	1.0
196	-0.457547	1.286227	1.0
197	-1.020000	-0.783926	1.0
198	1.363429	-0.800250	1.0
199	-1.246702	-0.388615	1.0

شکل ۹ - داده های ورودی با برچسب ۱ و -۱

نمودار پراکندگی داده‌ها را در شکل زیر مشاهده می‌کنید.



شکل ۱۰- نمودار پراکندگی داده‌های دیتاست **MadaLine**

برای آموزش شبکه و جداسازی کلاسی به نام MadalineNetwork را در پایتون تعریف کردیم که شامل متد هایی برای موارد زیر است :

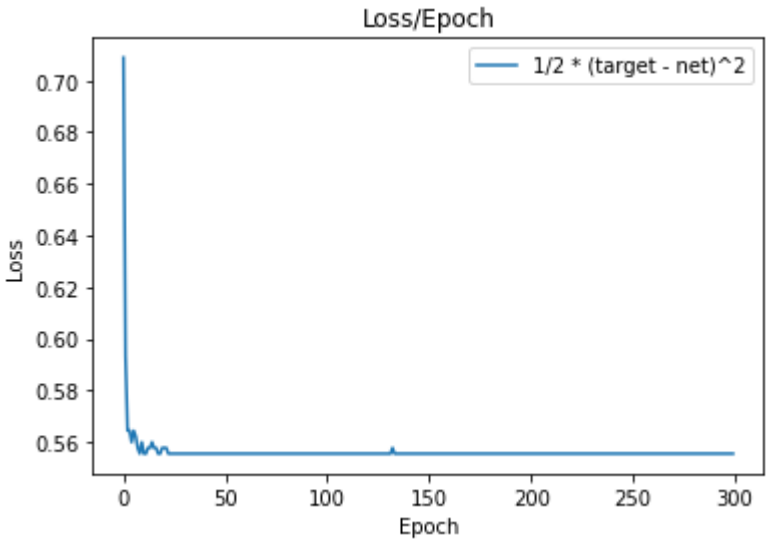
- تنظیم متغیر ها و وزن دهی اولیه (مقادیر کوچک و تصادفی)
- محاسبه مقادیر خروجی لایه های پنهان و خروجی نهایی
- تابع فعالساز
- تصحیح وزن ها و بایاس ها بر اساس الگوریتم mri
- محاسبه خطا بر اساس تابع ½ *$(t - net)$^ 2
- نمایش خطا ها بر اساس ایپاک ها
- آموزش مدل
- پیشبینی بر اساس مدل به دست آمده و محاسبه دقت مدل

.

حال مدل را با ترین میکنیم و با به ترتیب ۳و۴و۸ نورون جداسازی را انجام میدهیم . (ماکسیمم epoch را ۳۰۰ و نرخ یادگیری را ۰.۱ قرار داده ایم)

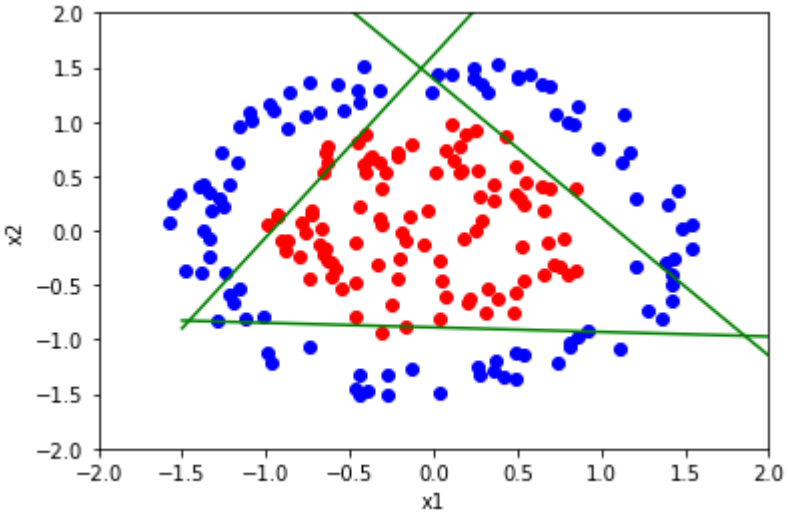
MadaLine با سه نورون:

۳۰۰ epoch انجام میشود .



شکل ۱۱- نمودار loss/epoch با ۳ نورون

حالا جداسازی میکنیم :



شکل ۱۲- تفکیک نقاط با ۳ نورون

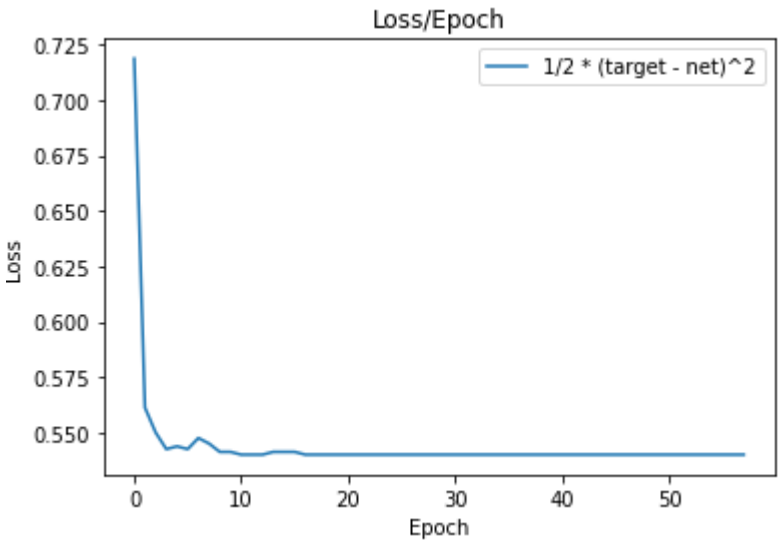
سپس دقت جدا سازی را نمایش میدهیم که accuracy اینجا ۸۸٪ شد . مدل با ۳ نورون به دقت کامل نمیرسد.

جدول ۲ – دقت پیشبینی با ۳ نورون

```
3 Hidden neurons
300  completed epochs
              precision    recall  f1-score   support

        -1.0       0.87      0.89      0.88       100
         1.0       0.89      0.87      0.88       100

    accuracy                           0.88       200
   macro avg       0.88      0.88      0.88       200
weighted avg       0.88      0.88      0.88       200

Accuracy of prediction is: 0.88
```

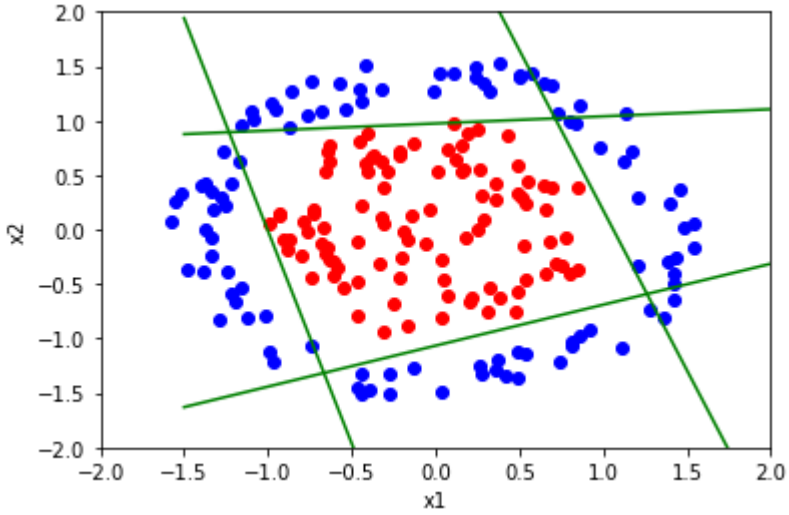
MadaLine با چهار نورون:

epoch ۵۷ انجام میشود و به دقت کامل میرسیم .



شکل ۱۳- نمودار loss/epoch با ۴ نورون

حالا جداسازی میکنیم :



شکل ۱۴- تفکیک نقاط با ۴ نورون

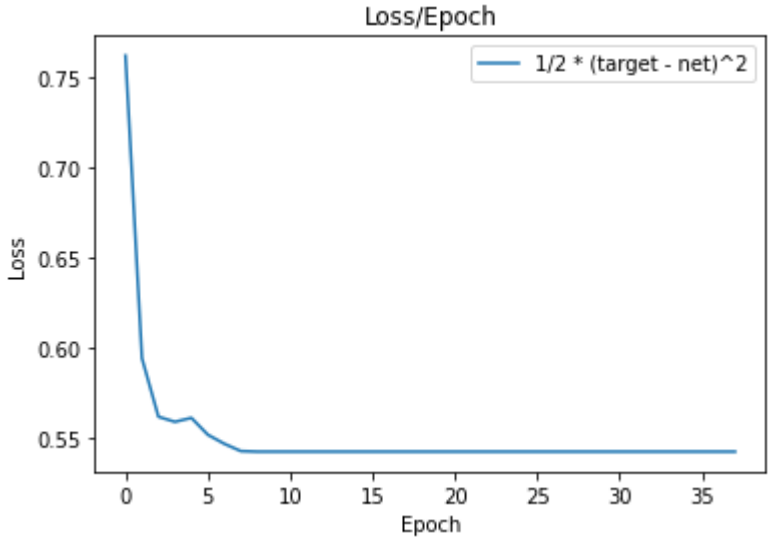
سپس دقت جدا سازی را نمایش میدهیم که accuracy اینجا ۱۰۰٪ شد و مدل کاملا نقاط را از هم تفکیک کرد .

جدول ۲ – دقت پیشبینی با ۴ نورون

```
4 Hidden neurons
Weights are not changing terminate training
57  completed epochs
              precision    recall  f1-score   support

        -1.0       1.00      1.00      1.00       100
         1.0       1.00      1.00      1.00       100

    accuracy                           1.00       200
   macro avg       1.00      1.00      1.00       200
weighted avg       1.00      1.00      1.00       200

Accuracy of prediction is: 1.0
```

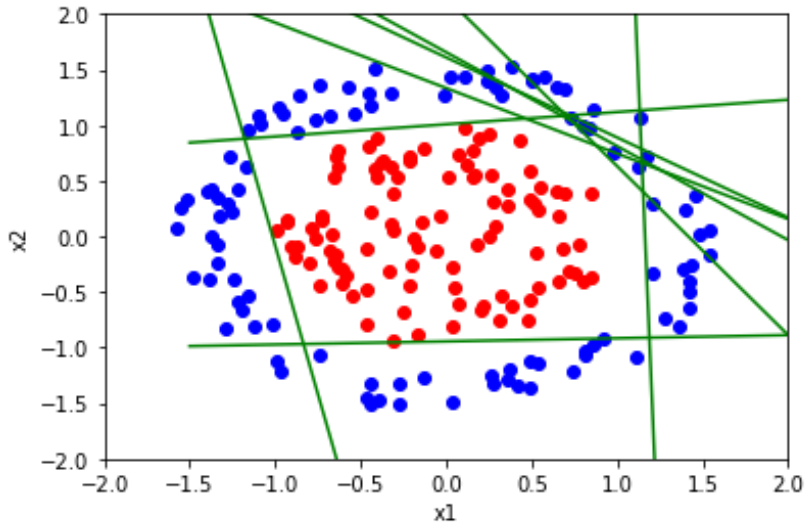
MadaLine با هشت نورون:

۳۷ epoch انجام میشود و به دقت کامل میرسیم .



شکل ۱۵- نمودار loss/epoch با ۸ نورون

حالا جداسازی میکنیم :



شکل ۱۶- تفکیک نقاط با ۸ نورون

سپس دقت جدا سازی را نمایش میدهیم که accuracy اینجا ۱۰۰٪ شد و مدل کاملا نقاط را از هم تفکیک کرد .

جدول ۳ – دقت پیشبینی با ۸ نورون

```
8 Hidden neurons
Weights are not changing terminate training
37  completed epochs
              precision    recall  f1-score   support

        -1.0       1.00      1.00      1.00       100
         1.0       1.00      1.00      1.00       100

    accuracy                           1.00       200
   macro avg       1.00      1.00      1.00       200
weighted avg       1.00      1.00      1.00       200

Accuracy of prediction is: 1.0
```

ج) حال به مقایسه در دقت و ایپاک در حالت های ۳،۴و۸ نورون میپردازیم .

جدول ۴ – مقایسه در ۳ حالت

تعداد نورون	۳	۴	۸
تعداد ایپاک	۳۰۰	۵۷	۳۷
دقت	٪۸۸	٪۱۰۰	٪۱۰۰

با ۳ نورون به تفکیک کامل نرسیدیم و حداقل ۴ نورون نیاز است تا دقت ٪۱۰۰ داشته باشیم. وقتی تعداد نورون ها را بالاتر میبریم دقت افزایش میابد ؛ در واقع تعداد نورون ها در اینجا همان خط های جدا کننده هستند بنابراین با نورون های بیشتر نواحی به شکل جامع تر از هم جدا میشوند پس اگر نقاطی در آینده اصافه شوند مدل آنها را هم بهتر تفکیک خواهد کرد .

همچنین هر قدر تعداد نورون ها بالاتر باشد سریعتر به همگرایی میرسیم و با تعداد ایپاک کمتری میتوانیم تفکیک را انجام دهیم . وقتی تعداد نورون ها ۸ تا میشود همچنان که در شکل ۱۶ مشخص است ۵ خط داده ها را کاملا از هم جدا میکنند و ۳ خط دیگر بلا استفاده میمانند در واقع مدل در هنگام ترین شدن دیگر وزن های آنها را بروزرسانی نکرده است چرا که ۵ ضلعی برای جداسازی کفایت میکند .

پاسخ ۳ – Restricted Boltzmann Machine

۳-۱. سیستم توصیه‌گر

A) داده های مربوط به فیلم ها و امتیازات را میخوانیم و در ۲ دیتا فریم ذخیره میکنیم ، سطر شماره صفر که نام ستون ها است که بعنوان داده خوانده شده را حذف میکنیم و برای ستون ها نامگذاری مناسب انجام میدهیم . سپس ۵ مورد اول و انتهایی فیلم ها و امتیازات را نمایش میدهیم .

جدول ۵ - ۵ مورد نخست فیلم‌ها

	MovieID	Title	Genres
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
1	2	Jumanji (1995)	Adventure|Children|Fantasy
2	3	Grumpier Old Men (1995)	Comedy|Romance
3	4	Waiting to Exhale (1995)	Comedy|Drama|Romance
4	5	Father of the Bride Part II (1995)	Comedy

جدول ۶- ۵ مورد آخر فیلم‌ها

	MovieID	Title	Genres
9737	193581	Black Butler: Book of the Atlantic (2017)	Action|Animation|Comedy|Fantasy
9738	193583	No Game No Life: Zero (2017)	Animation|Comedy|Fantasy
9739	193585	Flint (2017)	Drama
9740	193587	Bungo Stray Dogs: Dead Apple (2018)	Action|Animation
9741	193609	Andrew Dice Clay: Dice Rules (1991)	Comedy

جدول ۷ - ۵ مورد اول امتیازها

	UserID	MovieID	Rating	Timestamp
0	1	1	4.0	964982703
1	1	3	4.0	964981247
2	1	6	4.0	964982224
3	1	47	5.0	964983815
4	1	50	5.0	964982931

جدول ۸ - ۵ مورد آخر امتیازها

	UserID	MovieID	Rating	Timestamp
100831	610	166534	4.0	1493848402
100832	610	168248	5.0	1493850091
100833	610	168250	5.0	1494273047
100834	610	168252	5.0	1493846352
100835	610	170875	3.0	1493846415

ابعاد هر دو مجموعه داده را نمایش میدهیم :

ابعاد فیلم ها : (9742, 3)

ابعاد امتیازات : (100836, 4)

سپس ستون جدیدی ایجاد میکنیم و شماره هر سطر را در آن ذخیره میکنیم :

جدول ۹ - اضافه کردن ستون **List Index**

	MovieID	Title	Genres	List Index
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	0
1	2	Jumanji (1995)	Adventure|Children|Fantasy	1
2	3	Grumpier Old Men (1995)	Comedy|Romance	2
3	4	Waiting to Exhale (1995)	Comedy|Drama|Romance	3
4	5	Father of the Bride Part II (1995)	Comedy	4

B) بر اساس ستون شناسه فیلم دو مجموعه داده را با هم ادغام میکنیم .

جدول ۱۰ - ادغام دو مجموعه داده

	MovieID	Title	Genres	List Index	UserID	Rating	Timestamp
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	0	1	4.0	964982703
1	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	0	5	4.0	847434962
2	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	0	7	4.5	1106635946
3	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	0	15	2.5	1510577970
4	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	0	17	4.5	1305696483

C) ستون های Title ، Genres ، Timestamp در محاسبات مورد نظر برای این روش تاثیر گذار نیستند چرا که در این روش وقتی کاربران مختلف به یک فیلم امتیاز مشابهی داده باشند احتمال اینکه در آینده نیز به یک فیلم امتیاز یکسانی بدهند بالا است . ولی بین ۳ ستون ذکر شده در میان کاربران ارتباطی وجود ندارد و کاربر آنها را تعیین نمیکند و مقادیر ثابتی دارند که ویژگی های خود فیلم است برای اشاره به فیلم ها فقط شناسه آنها کافی است بنابراین آنها را حذف میکنیم تا راحتتر با دیتا فریم کار کنیم .

جدول ۱۱ - حذف ستون‌های اضافی

	MovieID	List Index	UserID	Rating
0	1	0	1	4.0
1	1	0	5	4.0
2	1	0	7	4.5
3	1	0	15	2.5
4	1	0	17	4.5

D)در دیتافریم به دست آمده داده ها را بر اساس شناسه هر کاربر گروه بندی میکنیم .

جدول ۱۲ - دسته‌بندی بر اساس **UserId**

UserID	MovieID	List Index	Rating
1	1	0	4.0
10	296	257	1.0
100	3	2	3.5
101	223	190	4.0
102	3	2	5.0

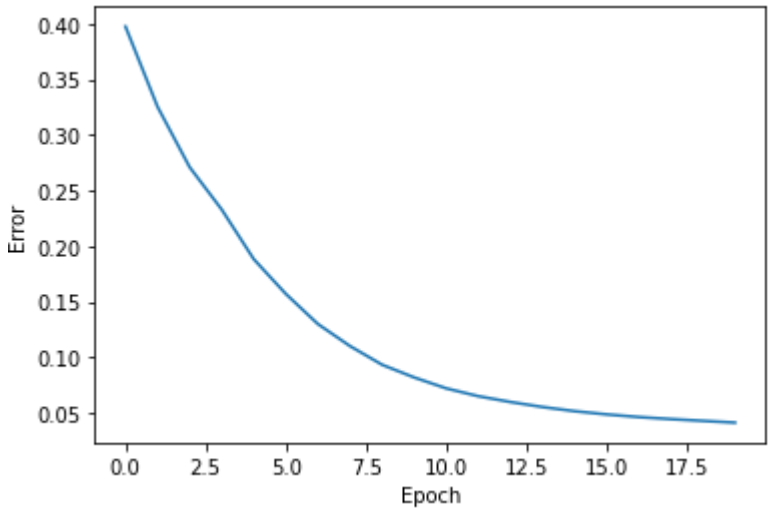
E) چون تمام امتیازات فیلم ها بین ۱ تا ۵ است با تقسیم همه به ۵ نرمالسازی انجام میشود و همه امتیازات بین ۰ تا ۱ قرار میگیرند . لیست train_X را میسازیم که هر سطر آن مربوط به یک کاربر است و طول آن سطر به اندازه ی تعداد فیلم ها است که در هر خانه امتیاز مربوط به یک فیلم ذخیره شده است .

F) برای ساخت مدل از کتابخانه tensorflow کمک میگیریم تعداد hidden layer ها را ۲۰ نورون و برای visible layer ها به تعداد فیلم ها در نظر میگیریم . bias و وزن هایی که دو لایه را به یکدیگر متصل میکنند را در placeholder ذخیره میکنیم . لایه ها را ایجاد میکنیم و توابع فعالساز آنها را relu و sigmoid قرار میدهیم ، learning rate را برابر ۰.۱ قرار میدهیم . گرادیان ها را میسازیم در اینجا باید Contrastive Divergence را حداکثر کنیم .

G) مدل را در ۲۰ epoch ترین میکنیم ، مقدار خطا در هر ایپاک را در شکل ۱۷ مشاهده می کنید. همانطور که نمودار خطا بر حسب ایپاک را در شکل زیر می‌بینید مقدار خطا در هر ایپاک کاهش پیدا می‌کند. پس شبکه به خوبی آموزش دیده شد.

```
0.3976148
0.32536286
0.2710497
0.23288001
0.18816066
0.1568857
0.12977919
0.10995163
0.0931578
0.081718236
0.07188674
0.064838976
0.05963376
0.055109594
0.05136182
0.048449032
0.04607109
0.04423375
0.04264587
0.041051622
```

شکل ۱۷ - مقدار خطا در هر ایپاک



شکل ۱۸ - نمودار خطا بر حسب **epoch** در سیستم توصیه‌گر

H) حالا مدل RBM را داریم برای اینکه فیلم های مورد علاقه یک کاربر خاص را پیدا کنیم اطلاعات مربوط به امتیاز فیلم های او که در train_X ذخیره کرده بودیم را به مدل وارد(feed) میکنیم و بر اساس آن ورودی را بازسازی میکنیم . کاربر شماره ۲۷ را به شکل تصادفی انتخاب میکنیم و train_X[27] را وارد میکنیم . مدل امتیاز تمام فیلم ها را برای او پیشبینی میکند . دیتا فریم را بر اساس امتیازات sort میکنیم و ۱۵ فیلمی که برای کاربر ۲۷ بیشترین امتیاز پیشنهاد داده شده توسط مدل را دارند نشان میدهیم.

این ۱۵ فیلم پیشنهادی را در جدول ۱۳ مشاهده می کنید.

جدول ۱۳ - ۱۵ فیلم با بیشترین امتیاز پیشنهادی مدل برای کاربر ۲۷

	MovieID	Title	Genres	List Index	Recommendation Score
257	296	Pulp Fiction (1994)	Comedy|Crime|Drama|Thriller	257	1.000000
277	318	Shawshank Redemption, The (1994)	Crime|Drama	277	1.000000
314	356	Forrest Gump (1994)	Comedy|Drama|Romance|War	314	1.000000
1939	2571	Matrix, The (1999)	Action|Sci-Fi|Thriller	1939	0.999683
510	593	Silence of the Lambs, The (1991)	Crime|Horror|Thriller	510	0.999587
224	260	Star Wars: Episode IV - A New Hope (1977)	Action|Adventure|Sci-Fi	224	0.658822
97	110	Braveheart (1995)	Action|Drama|War	97	0.531372
461	527	Schindler's List (1993)	Drama|War	461	0.517449
418	480	Jurassic Park (1993)	Action|Adventure|Sci-Fi|Thriller	418	0.503866
3638	4993	Lord of the Rings: The Fellowship of the Ring,...	Adventure|Fantasy	3638	0.481299
46	50	Usual Suspects, The (1995)	Crime|Mystery|Thriller	46	0.473460
398	457	Fugitive, The (1993)	Thriller	398	0.472437
507	589	Terminator 2: Judgment Day (1991)	Action|Sci-Fi	507	0.470731
911	1210	Star Wars: Episode VI - Return of the Jedi (1983)	Action|Adventure|Sci-Fi	911	0.459292
123	150	Apollo 13 (1995)	Adventure|Drama|IMAX	123	0.454230

پاسخ ۴ – MLP

۴-۱. Multi-Layer Perceptron

A) دیتاست را در دیتافریمی به نام houses لود می‌کنیم. نتیجه‌ی اجرای تابع info() بر روی دیتاست خوانده شده را در شکل ۱۹ مشاهده می‌کنید.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21613 entries, 0 to 21612
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   id             21613 non-null  int64
 1   date           21613 non-null  object
 2   price          21613 non-null  float64
 3   bedrooms       21613 non-null  int64
 4   bathrooms      21613 non-null  float64
 5   sqft_living    21613 non-null  int64
 6   sqft_lot       21613 non-null  int64
 7   floors         21613 non-null  float64
 8   waterfront     21613 non-null  int64
 9   view           21613 non-null  int64
 10  condition      21613 non-null  int64
 11  grade          21613 non-null  int64
 12  sqft_above     21613 non-null  int64
 13  sqft_basement  21613 non-null  int64
 14  yr_built       21613 non-null  int64
 15  yr_renovated   21613 non-null  int64
 16  zipcode        21613 non-null  int64
 17  lat            21613 non-null  float64
 18  long           21613 non-null  float64
 19  sqft_living15  21613 non-null  int64
 20  sqft_lot15     21613 non-null  int64
dtypes: float64(5), int64(15), object(1)
memory usage: 3.5+ MB
```

شکل ۱۹ - خروجی تابع **info()** روی دیتاست **houses**

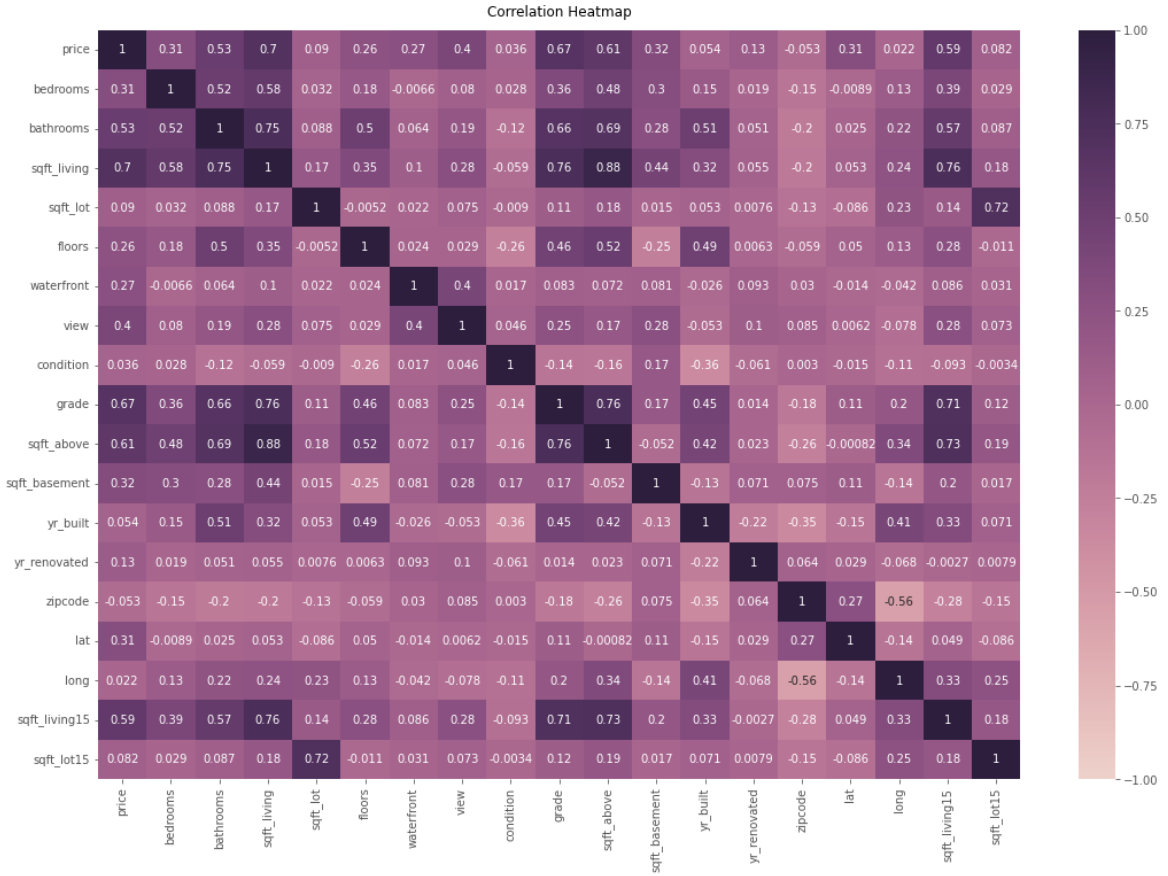
B) با تابع اجرای ()isna روی هر ستون می‌توان nan یا non-nan بودن داده‌های موجود در دیتافریم را فهمید. پس کافیست تابع ()sum را هم روی ()houses.isna اعمال کنیم. ()houses.isna.sum تعداد مقادیر nan هر ستون را به ما می‌دهد که آن را در شکل ۲۰ مشاهده می‌کنید.

```
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 ۲۰ - تعداد مقادیر **nan** در هر یک از ستون‌های دیتاست **houses**

همانطور که در شکل ۲۰ می‌بینید ما در این دیتاست مقدار nan نداریم.

C) ماتریس correlation ویژگی‌ها را در شکل ۲۱ مشاهده می‌کنید. همانطور که در ماتریس قابل مشاهده است، ویژگی sqft_living بیشترین همبستگی را با ویژگی price داشته که مقدار این همبستگی برابر با 0.702035 می‌باشد.

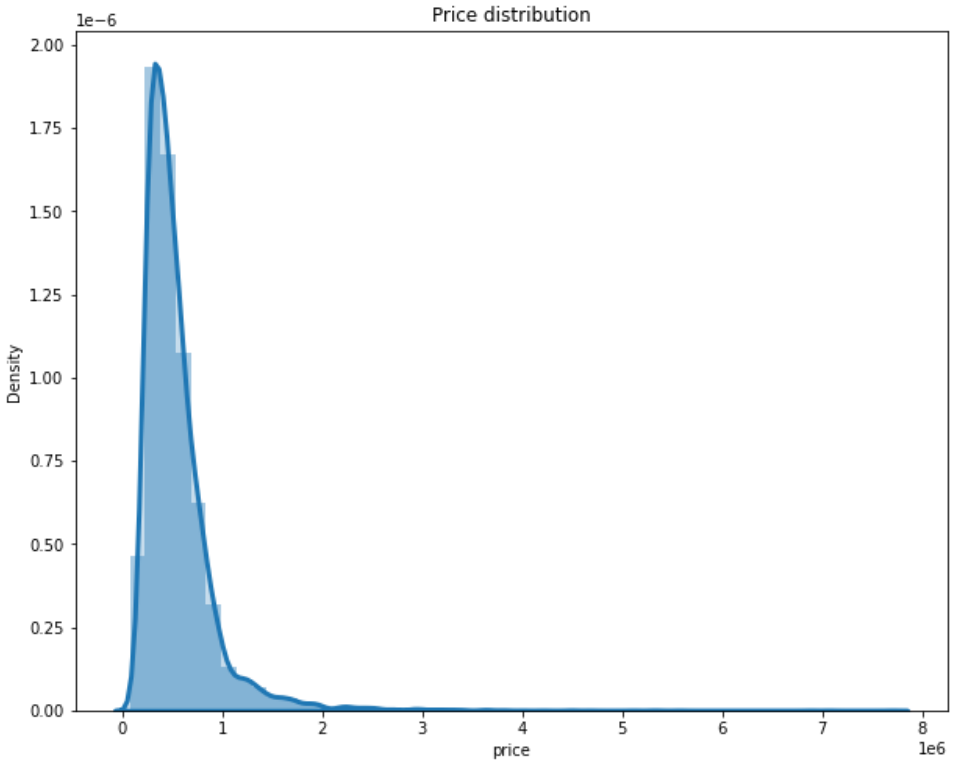


شکل ۲۱- ماتریس **correlation** ویژگی‌های دیتاست **houses**

```
highest correlation with price & the feature name:

correlation    0.702035
Name: sqft_living, dtype: float64
```

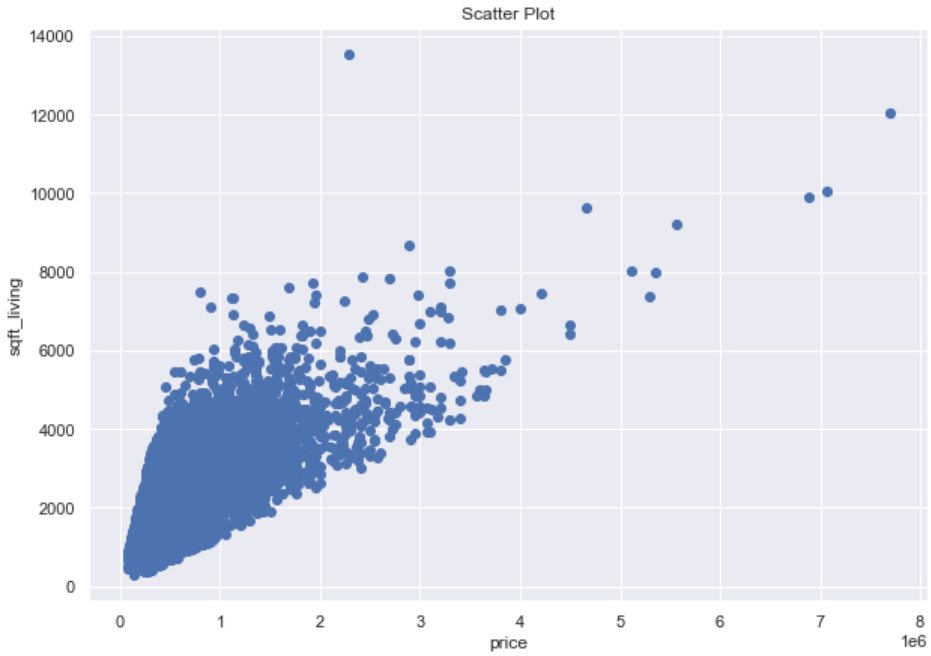
شکل ۲۲- ویژگی با بیشترین میزان **correlation** با ویژگی **price**

D) نمودار توزیع ویژگی price را در شکل ۲۳ مشاهده می‌کنید. این ویژگی یک توزیع با چولگی راست می‌باشد. یعنی فراوانی خانه‌ها با قیمت‌های کم نسبت به خانه های با قیمت بالا خیلی بیشتر است.



شکل ۲۳ - نمودار توزیع قیمت خانه‌ها

نمودار پراکندگی دو ویژگی price و sqft_living (که بیشترین میزان همبستگی را با price دارد) را در شکل ۲۴ مشاهده می‌کنید. مانند انتظار نمودار این دو ویژگی همبستگی بالای آنها را نشان می‌دهد.



شکل ۲۴ - نمودار پراکندگی دو ویژگی **price** و **sqft_living**

به طور کلی با افزایش price (قیمت)، مقدار sqft_living (مساحت اتاق پذیرایی) هم افزایش می‌یابد و بلعکس که چنین اتفاقی منطقی می‌باشد.

E) با parse کردن ستون date، مقدار سال و ماه را استخراج کرده‌date ، به int تبدیل کرده و این دو ستون(month و year) را به دیتافریم اضافه می‌کنیم. خود ستون date را هم drop می‌کنیم. نمایش دو ستون جدید را در شکل ۲۵ مشاهده می‌کنید.

	year	month
0	2014	10
1	2014	12
2	2015	2
3	2014	12
4	2015	2
...	...	...
21608	2014	5
21609	2015	2
21610	2014	6
21611	2015	1
21612	2014	10

21613 rows × 2 columns

شکل ۲۵- دو ستون **year** و **month**

F) همه‌ی ستون ها به جز id و price را به عنوان مجموعه‌ی x و ستون price را به عنوان مجموعه‌ی y (target) انتخاب می‌کنیم. سپس با استفاده از متد train_test_split() از کتابخانه‌ی sklearn date، داده‌ها را به شیوه‌ی گفته شده به مجموعه‌های train و test تقسیم می کنیم. برای reproducibility این تقسیم، random sate را set کردیم. این کد را در شکل ۲۶ مشاهده می کنید.

```
from sklearn.model_selection import train_test_split

X = houses.loc[:,'bedrooms':].to_numpy()
y = houses['price'].to_numpy()
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

شکل ۲۶- تقسیم دیتاست به داده‌های **train** و **test**

G) در کد شکل ۲۷، با استفاده از تابع fit_transform() دیتای train را center می‌کنیم. سپس با استفاده از همان میانگین و انحراف معیار که برای داده‌ی train استفاده شد داده‌ی test را transform می‌کنیم.

```python
from sklearn.preprocessing import MinMaxScaler

scaler1 = MinMaxScaler()
scaler2 = MinMaxScaler()

X_train_scaled = scaler1.fit_transform(X_train)
y_train_scaled = scaler2.fit_transform(y_train.reshape(-1, 1))
X_test_scaled = scaler1.transform(X_test)
y_test_scaled = scaler2.transform(y_test.reshape(-1, 1))
```

شکل ۲۷ - **MinMax Scalar** کردن داده‌ها

H) از کتابخانه‌ی tensorflow استفاده می‌کنیم. شبکه‌ای با سه لایه می‌سازیم. لایه‌ی پنهان اول ۱۵ نورون، لایه‌ی پنهان دوم ۷ نورون و لایه‌ی خروجی ۱ نورون دارد. از Relu function به عنوان تابع فعالساز نورون‌های هر سه لایه استفاده می‌کنیم. از Mean Absolute Error(MAE) به عنوان loss function استفاده می‌کنیم . optimizer را هم Stochastic Gradient Descent قرار می‌دهیم. تعداد epochها را هم ۵۰ قرار می‌دهیم. در نهایت با x و y آموزش(train) که به روش Minmax، scale شده‌اند، مدل را آموزش می‌دهیم. همانطور که در شکل ۲۸ می بینیم، مقدار mae در هر epoch کاهش پیدا می‌کند. کاهش چشمگیری در ایپاک چهارم می‌بینیم (مقدار خطای mae از 0.0569 به 0.0291 کاهش می‌یابد) و پس از آن مقدار کاهش کمتر می‌باشد تا اینکه در ایپاک ۵۰م مقدار خطا به 0.0172 می‌رسد. پس شبکه به درستی آموزش می‌بیند.

```
Epoch 1/50
541/541 [==============================] - 1s 2ms/step - loss: 0.0612 - mae: 0.0612
Epoch 2/50
541/541 [==============================] - 1s 2ms/step - loss: 0.0609 - mae: 0.0609
Epoch 3/50
541/541 [==============================] - 1s 2ms/step - loss: 0.0569 - mae: 0.0569
Epoch 4/50
541/541 [==============================] - 1s 2ms/step - loss: 0.0291 - mae: 0.0291
Epoch 5/50
541/541 [==============================] - 1s 2ms/step - loss: 0.0222 - mae: 0.0222
Epoch 6/50
541/541 [==============================] - 1s 2ms/step - loss: 0.0205 - mae: 0.0205
Epoch 7/50
541/541 [==============================] - 1s 2ms/step - loss: 0.0195 - mae: 0.0195
Epoch 8/50
541/541 [==============================] - 1s 2ms/step - loss: 0.0188 - mae: 0.0188
Epoch 9/50
541/541 [==============================] - 1s 2ms/step - loss: 0.0181 - mae: 0.0181
Epoch 10/50
541/541 [==============================] - 1s 2ms/step - loss: 0.0178 - mae: 0.0178
Epoch 11/50
541/541 [==============================] - 1s 2ms/step - loss: 0.0175 - mae: 0.0175
Epoch 12/50
541/541 [==============================] - 1s 2ms/step - loss: 0.0172 - mae: 0.0172
Epoch 13/50
...
Epoch 49/50
541/541 [==============================] - 1s 2ms/step - loss: 0.0146 - mae: 0.0146
Epoch 50/50
541/541 [==============================] - 1s 2ms/step - loss: 0.0147 - mae: 0.0147
```

شکل ۲۸- آموزش شبکه‌ی **MLP** سه لایه

I) در شکل ۲۹ نتیجه‌ی آموزش شبکه با Adadelta optimizer و در شکل ۳۰ نتیجه‌ی آموزش شبکه با RMSprop optimizer را مشاهده می‌کنید. در هردو این حالات loss function انتخابی mean absolute error می‌باشد. می‌بینید میزان خطا شبکه با RMSprop کمتر از شبکه با Adadelta می‌باشد. در هردو این optimizerها نیازی به دستی تغییر دادن نرخ یادگیری نیست چرا که نرخ یادگیری به طور adaptive تغییر می‌کند. RMSprop شباهت زیادی به Adadelta دارد. تنها تفاوت آنها در نحوه مدیریت گرادیان‌های گذشته می‌باشد.

```
Epoch 1/50
541/541 [==============================] - 2s 2ms/step - loss: 0.5131 - mae: 0.5131
Epoch 2/50
541/541 [==============================] - 1s 2ms/step - loss: 0.4909 - mae: 0.4909
Epoch 3/50
541/541 [==============================] - 1s 2ms/step - loss: 0.4644 - mae: 0.4644
Epoch 4/50
541/541 [==============================] - 1s 2ms/step - loss: 0.4350 - mae: 0.4350
Epoch 5/50
541/541 [==============================] - 1s 2ms/step - loss: 0.4034 - mae: 0.4034
Epoch 6/50
541/541 [==============================] - 1s 2ms/step - loss: 0.3701 - mae: 0.3701
Epoch 7/50
541/541 [==============================] - 1s 2ms/step - loss: 0.3356 - mae: 0.3356
Epoch 8/50
541/541 [==============================] - 1s 3ms/step - loss: 0.3005 - mae: 0.3005
Epoch 9/50
541/541 [==============================] - 1s 2ms/step - loss: 0.2655 - mae: 0.2655
Epoch 10/50
541/541 [==============================] - 1s 2ms/step - loss: 0.2311 - mae: 0.2311
Epoch 11/50
541/541 [==============================] - 1s 2ms/step - loss: 0.1985 - mae: 0.1985
Epoch 12/50
541/541 [==============================] - 1s 2ms/step - loss: 0.1686 - mae: 0.1686
Epoch 13/50
...
Epoch 49/50
541/541 [==============================] - 2s 3ms/step - loss: 0.0471 - mae: 0.0471
Epoch 50/50
541/541 [==============================] - 1s 2ms/step - loss: 0.0469 - mae: 0.0469
```

شکل ۲۹ - آموزش شبکه با **Adadelta optimizer**

```
Epoch 1/50
541/541 [==============================] - 4s 2ms/step - loss: 0.0270 - mae: 0.0270
Epoch 2/50
541/541 [==============================] - 1s 2ms/step - loss: 0.0179 - mae: 0.0179
Epoch 3/50
541/541 [==============================] - 1s 2ms/step - loss: 0.0159 - mae: 0.0159
Epoch 4/50
541/541 [==============================] - 1s 2ms/step - loss: 0.0151 - mae: 0.0151
Epoch 5/50
541/541 [==============================] - 1s 2ms/step - loss: 0.0146 - mae: 0.0146
Epoch 6/50
541/541 [==============================] - 1s 2ms/step - loss: 0.0142 - mae: 0.0142
Epoch 7/50
541/541 [==============================] - 1s 2ms/step - loss: 0.0138 - mae: 0.0138
Epoch 8/50
541/541 [==============================] - 1s 2ms/step - loss: 0.0136 - mae: 0.0136
Epoch 9/50
541/541 [==============================] - 1s 2ms/step - loss: 0.0133 - mae: 0.0133
Epoch 10/50
541/541 [==============================] - 1s 2ms/step - loss: 0.0131 - mae: 0.0131
Epoch 11/50
541/541 [==============================] - 1s 2ms/step - loss: 0.0128 - mae: 0.0128
Epoch 12/50
541/541 [==============================] - 1s 2ms/step - loss: 0.0127 - mae: 0.0127
Epoch 13/50
...
Epoch 49/50
541/541 [==============================] - 1s 2ms/step - loss: 0.0112 - mae: 0.0112
Epoch 50/50
541/541 [==============================] - 1s 2ms/step - loss: 0.0112 - mae: 0.0112
```

شکل ۳۰- آموزش شبکه با **RMSprop optimizer**

در شکل۳۱ نتیجه‌ی آموزش شبکه با Mean Absolute Error loss function و در شکل ۳۲ نتیجه‌ی آموزش شبکه با Binary Cross-Entropy loss function را مشاهده می‌کنید.

```
Epoch 1/50
541/541 [==============================] - 2s 2ms/step - loss: 0.0248 - mae: 0.0248
Epoch 2/50
541/541 [==============================] - 1s 2ms/step - loss: 0.0167 - mae: 0.0167
Epoch 3/50
541/541 [==============================] - 1s 2ms/step - loss: 0.0156 - mae: 0.0156
Epoch 4/50
541/541 [==============================] - 1s 2ms/step - loss: 0.0150 - mae: 0.0150
Epoch 5/50
541/541 [==============================] - 1s 2ms/step - loss: 0.0146 - mae: 0.0146
Epoch 6/50
541/541 [==============================] - 1s 2ms/step - loss: 0.0142 - mae: 0.0142
Epoch 7/50
541/541 [==============================] - 1s 2ms/step - loss: 0.0139 - mae: 0.0139
Epoch 8/50
541/541 [==============================] - 1s 2ms/step - loss: 0.0136 - mae: 0.0136
Epoch 9/50
541/541 [==============================] - 1s 2ms/step - loss: 0.0134 - mae: 0.0134
Epoch 10/50
541/541 [==============================] - 1s 2ms/step - loss: 0.0131 - mae: 0.0131
Epoch 11/50
541/541 [==============================] - 1s 2ms/step - loss: 0.0129 - mae: 0.0129
Epoch 12/50
541/541 [==============================] - 1s 2ms/step - loss: 0.0126 - mae: 0.0126
Epoch 13/50
...
Epoch 49/50
541/541 [==============================] - 1s 2ms/step - loss: 0.0110 - mae: 0.0110
Epoch 50/50
541/541 [==============================] - 1s 2ms/step - loss: 0.0110 - mae: 0.0110
```

شکل ۳۱ - آموزش شبکه با **Mean Absolute Error loss function**

```
Epoch 1/50
541/541 [==============================] - 2s 2ms/step - loss: 0.2269 - mae: 0.0246
Epoch 2/50
541/541 [==============================] - 1s 2ms/step - loss: 0.2201 - mae: 0.0173
Epoch 3/50
541/541 [==============================] - 1s 3ms/step - loss: 0.2195 - mae: 0.0161
Epoch 4/50
541/541 [==============================] - 1s 2ms/step - loss: 0.2192 - mae: 0.0156
Epoch 5/50
541/541 [==============================] - 1s 2ms/step - loss: 0.2190 - mae: 0.0153
Epoch 6/50
541/541 [==============================] - 1s 2ms/step - loss: 0.2189 - mae: 0.0150
Epoch 7/50
541/541 [==============================] - 1s 3ms/step - loss: 0.2188 - mae: 0.0148
Epoch 8/50
541/541 [==============================] - 1s 2ms/step - loss: 0.2187 - mae: 0.0146
Epoch 9/50
541/541 [==============================] - 1s 2ms/step - loss: 0.2187 - mae: 0.0145
Epoch 10/50
541/541 [==============================] - 1s 2ms/step - loss: 0.2186 - mae: 0.0144
Epoch 11/50
541/541 [==============================] - 1s 2ms/step - loss: 0.2186 - mae: 0.0144
Epoch 12/50
541/541 [==============================] - 1s 2ms/step - loss: 0.2186 - mae: 0.0143
Epoch 13/50
...
Epoch 49/50
541/541 [==============================] - 1s 2ms/step - loss: 0.2173 - mae: 0.0107
Epoch 50/50
541/541 [==============================] - 2s 3ms/step - loss: 0.2173 - mae: 0.0108
```

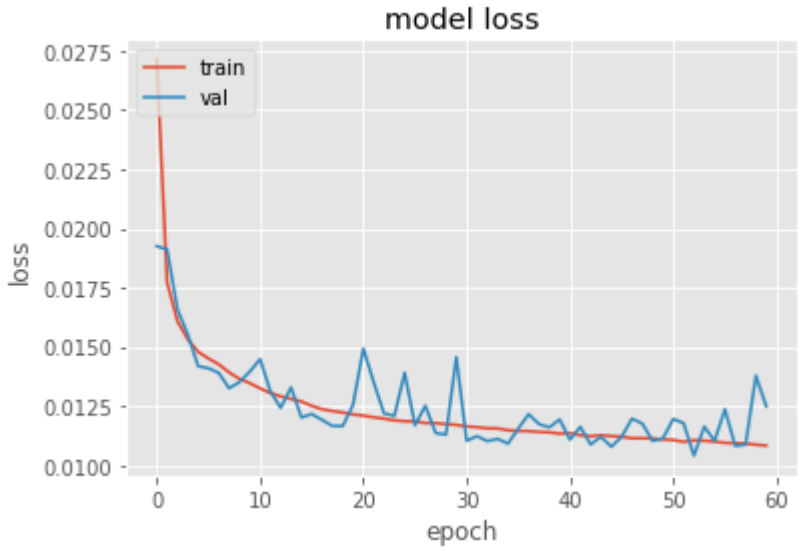
شکل ۳۲ - آموزش شبکه با **Binary Cross-Entropy loss function**

J) ۲۵ درصد از داده‌های train را به عنوان validation در نظر گرفتیم و شبکه را آموزش دادیم که نتیجه‌ی آن را در شکل ۳۳ مشاهده می‌کنید. از آنجایی که بهترین نتایج را با RMSprop optimizer و MAE loss function گرفته بودیم، شبکه را با آنها و با ۶۰ ایپاک train می‌کنیم.

```
Epoch 1/60
406/406 [==============================] - 2s 4ms/step - loss: 0.0271 - mae: 0.0271 - val_loss: 0.0193 - val_mae: 0.0193
Epoch 2/60
406/406 [==============================] - 1s 2ms/step - loss: 0.0177 - mae: 0.0177 - val_loss: 0.0191 - val_mae: 0.0191
Epoch 3/60
406/406 [==============================] - 1s 2ms/step - loss: 0.0161 - mae: 0.0161 - val_loss: 0.0166 - val_mae: 0.0166
Epoch 4/60
406/406 [==============================] - 1s 2ms/step - loss: 0.0153 - mae: 0.0153 - val_loss: 0.0155 - val_mae: 0.0155
Epoch 5/60
406/406 [==============================] - 1s 2ms/step - loss: 0.0148 - mae: 0.0148 - val_loss: 0.0142 - val_mae: 0.0142
Epoch 6/60
406/406 [==============================] - 1s 3ms/step - loss: 0.0145 - mae: 0.0145 - val_loss: 0.0141 - val_mae: 0.0141
Epoch 7/60
406/406 [==============================] - 1s 3ms/step - loss: 0.0143 - mae: 0.0143 - val_loss: 0.0139 - val_mae: 0.0139
Epoch 8/60
406/406 [==============================] - 1s 3ms/step - loss: 0.0139 - mae: 0.0139 - val_loss: 0.0133 - val_mae: 0.0133
Epoch 9/60
406/406 [==============================] - 1s 2ms/step - loss: 0.0136 - mae: 0.0136 - val_loss: 0.0135 - val_mae: 0.0135
Epoch 10/60
406/406 [==============================] - 1s 2ms/step - loss: 0.0135 - mae: 0.0135 - val_loss: 0.0140 - val_mae: 0.0140
Epoch 11/60
406/406 [==============================] - 1s 2ms/step - loss: 0.0132 - mae: 0.0132 - val_loss: 0.0145 - val_mae: 0.0145
Epoch 12/60
406/406 [==============================] - 1s 2ms/step - loss: 0.0130 - mae: 0.0130 - val_loss: 0.0131 - val_mae: 0.0131
Epoch 13/60
...
Epoch 59/60
406/406 [==============================] - 1s 2ms/step - loss: 0.0109 - mae: 0.0109 - val_loss: 0.0138 - val_mae: 0.0138
Epoch 60/60
406/406 [==============================] - 1s 2ms/step - loss: 0.0108 - mae: 0.0108 - val_loss: 0.0125 - val_mae: 0.0125
```

شکل۳۳ - آموزش شبکه **MLP** با دادهی**train** و **validation**

نمودار loss و validation loss را در شکل ۳۴ مشاهده می کنید. نوسان میزان loss دادههای validation با گذر ایپاکهای بیشتر، کمتر میشود. پس به نظر میرسد که مدل، نسبتاً قابل قبول آموزش یافته است.



شکل ۳۴ - نمودار **train loss** و **validation loss** بر حسب **epoch**

K) ۵ داده‌ی تصادفی انتخاب شده از مجموعه‌ی test عبارتند از:

[485000, 340000, 335606, 425000, 490000]

پیش‌بینی مدل به دادن این ۵ داده به صورت زیر است:

[453907.22, 405090.75, 400137.62, 528343.75, 1559611.4]

اختلاف قیمت پیش‌بینی شده توسط مدل با قیمت واقعی را در شکل ۳۵ مشاهده می‌کنید:

```
difrrence in the prediction

[31092.78125, -65090.75, -64531.625, -103343.75, -1069611.375]
```

شکل ۳۵ - اختلاف قیمت پیش‌بینی شده توسط شبکه با قیمت واقعی