به نام خدا

دانشگاه تهران

نام و نام خانوادگی	**پرهام بیچرانلو – آناهیتا هاشم زاده**
شماره دانشجویی	**۸۱۰۱۰۰۳۰۳ - ۸۱۰۱۰۰۵۰۲**
تاریخ ارسال گزارش	**۱۴۰۱,۰۸,۱۰**

دانشکده مهندسی برق و کامپیوتر

درس شبکه‌های عصبی و یادگیری عمیق

تمرین اول

فهرست

شکل‌ها

جدول‌ها

ا

پاسخ ۱. شبکه عصبی Mcculloch-Pitts

۱-۱. ضرب کننده باینری دو بیتی

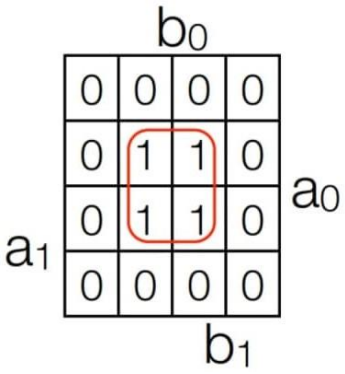
قسمت الف)

در این قسمت برای هر خروجی یک شبکه در نظر گرفته شده است، به این صورت که در ابتدا با استفاده از جدول درستی و همچنین جدول کارنو معادلات مربوط به هر خروجی را ساده شده و سپس با حداقل threshld و تعداد نورون، شبکه مربوط به آن طراحی شده است.

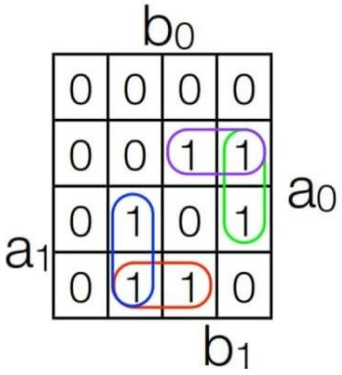
جدول ۱. جدول درستی ضرب کننده ۲ بیتی

2*2-bit multiplier truth table							
a1	a0	b1	b0	y3	y2	y1	y0
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

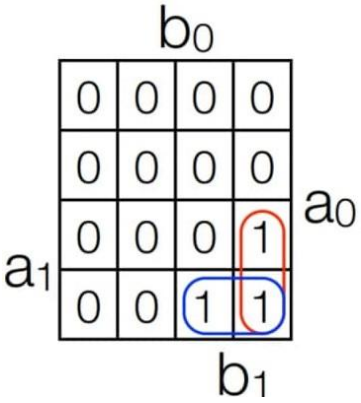
جدول ۲. جدول کارنو ضرب کننده ۲ بیتی



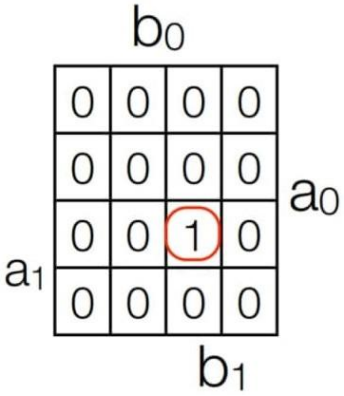
$y_0 = a_0 b_0$



$y_1 = a_1 a_0 b_0 + a_1 b_1 c_+ + a_1 a_0 b_1 + a_0 b_1 \bar{b}_0$

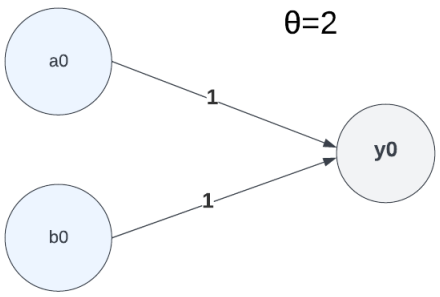


$y_2 = a_1 a_0 b_1 + a_1 b_1 \bar{b}_0$



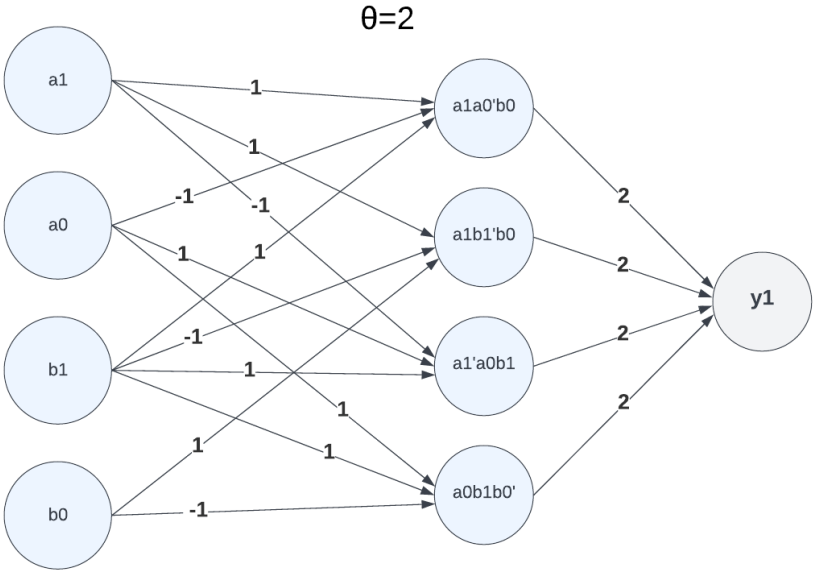
$y_3 = a_1 a_0 b_1 b_0$

- **شبکه خروجی y0:** با استفاده از جدول کارنو معادله این خروجی برابر با $y_0 = a_0 b_0$ می باشد، اگر که threshold را برابر ۲ و همینطور وزن ها را برابر ۱ در نظر بگیریم با استفاده از فرمول $y\_in == \sum_{i=1}^{n} w_i x_i$ مجموع ورودی وزن دار شده را بدست آورده و با threshold مقایسه میکنیم خواهیم دید که در این حالت معادله موردنظر بدست خواهد آمد.



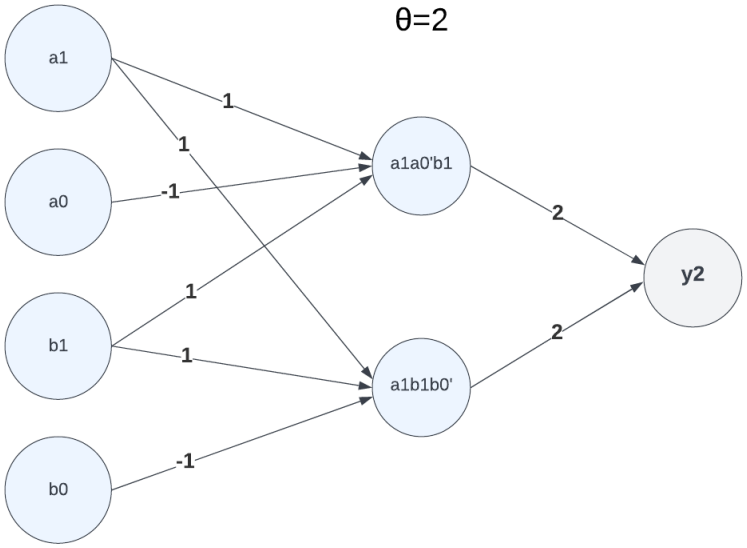
شکل ۱. شبکه مربوط به خروجی y0

- **شبکه خروجی y1:** همانند y0 معادله ساده شده این خروجی را نیز از طریق جدول کارنو بدست آورده که برابر با $y_1 = a_1\bar{a}_0 b_0 + a_1 b_1 c_+ + \bar{a}_1 a_0 b_1 + a_0 b_1 \bar{b}_0$ خواهد بود، سپس جاهایی که بین ورودی ها ضرب قرار دارد وزن های ورودی مثبت را ۱ و ورودی های منفی را برابر ۱- در نظر گرفته، و همچین وزن بین نورون هایی که جمع شده اند را برابر ۲ قرار داده ایم و با استفاده از فرمول y_in= $\sum_{i=1}^{n} w_i x_i$ مجموع ورودی وزن دار شده را برای هر لایه نورون بدست میاوریم، که کوچکترین threshold برای ساخت این معادله برابر ۲ می باشد. در نهایت با مقایسه y_in با threshold خواهیم دید که در این حالت معادله موردنظر بدست خواهد آمد.



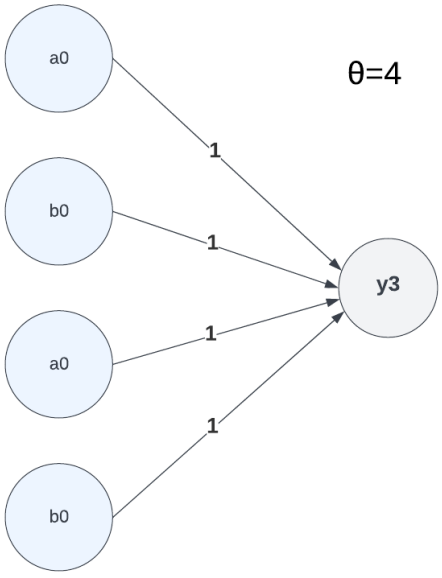
شکل ۲. شبکه مربوط به خروجی **y1**

- **شبکه خروجی y2:** همانند y0 و y1 معادله ساده شده این خروجی را از طریق جدول کارنو بدست آورده که برابر با $y_2 = a_1\bar{a}_0 b_1 + a_1 b_1 \bar{b}_0$ خواهد بود، سپس مشابه شبکه قبل جاهایی که بین ورودی ها ضرب قرار دارد وزن های ورودی مثبت را ۱ و ورودی های منفی را برابر ۱- در نظر گرفته، و همچین وزن بین نورون هایی که جمع شده اند را برابر ۲ قرار داده ایم و با استفاده از فرمول y_in= = $\sum_{i=1}^{n} w_i x_i$ مجموع ورودی وزن دار شده را برای هر لایه نورون بدست میاوریم، که کوچکترین threshold برای ساخت این معادله برابر ۲ می باشد. در نهایت با مقایسه y_in با threshold خواهیم دید که در این حالت معادله موردنظر بدست خواهد آمد.



شکل ۳. شبکه مربوط به خروجی **y2**

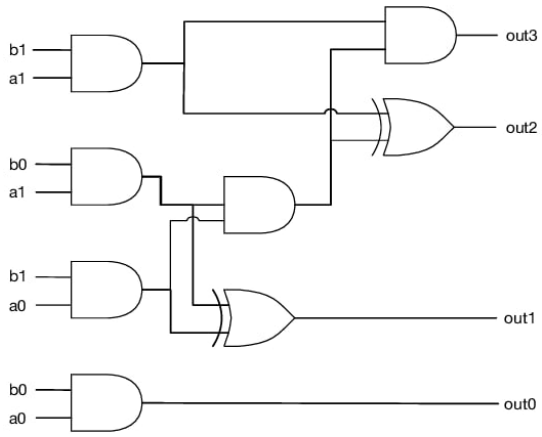
- **شبکه خروجی y3:** همانند y0 , y1 و y2 معادله ساده شده این خروجی را نیز از طریق جدول کارنو بدست آورده که برابر با $y_3 = a_1 a_0 b_1 b_0$ می باشد، چون در اینجا همه ورودی های معادله مثبت می باشد پس تمامی وزن ها را برابر ۱ در نظر گرفته و با استفاده از فرمول y_in= $\sum_{i=1}^{n} w_i x_i$ = مجموع ورودی وزن دار شده را برای نورون ها بدست می اوریم، که کوچکترین threshold برای ساخت این معادله برابر ۴ می باشد. در نهایت با مقایسه y_in با threshold خواهیم دید که در این حالت معادله موردنظر بدست خواهد آمد.



شکل ۴. شبکه مربوط به خروجی **y3**

قسمت ب)

در این قسمت با دو روش ضرب کننده دو بیتی باینری پیاده سازی شده است (حالت دوم مدنظر سوال بوده و راه حل اصلی است) که در روش اول با استفاده گیت های And و Or و And Not (Xor) نورون Mcculloch-Pitts گیت های مدار ضرب کننده دو بیتی (شکل۵) را عینا پیاده سازی شده که در این حالت مقدار threshold در تمامی شبکه ها برابر با ۲ در نظر گرفته شده است هر چند مقدار threshold در این حالت کمتر می باشد ولی تعداد نورون ها بیشتر می باشد، با توجه به صورت سوال که گفته شده نورون کمتر از اهمیت بیشتری برخوردار است پس در حالت دوم شبکه های طراحی شده در قسمت الف که با استفاده از معادلات ساده شده جدول کارنو ضرب کننده بدست آمده، پیاده سازی شده است و در نهایت ۱۶ حالت ضرب محاسبه شده است.



شکل ۵. مدار منطقی ضرب کننده دو بیتی

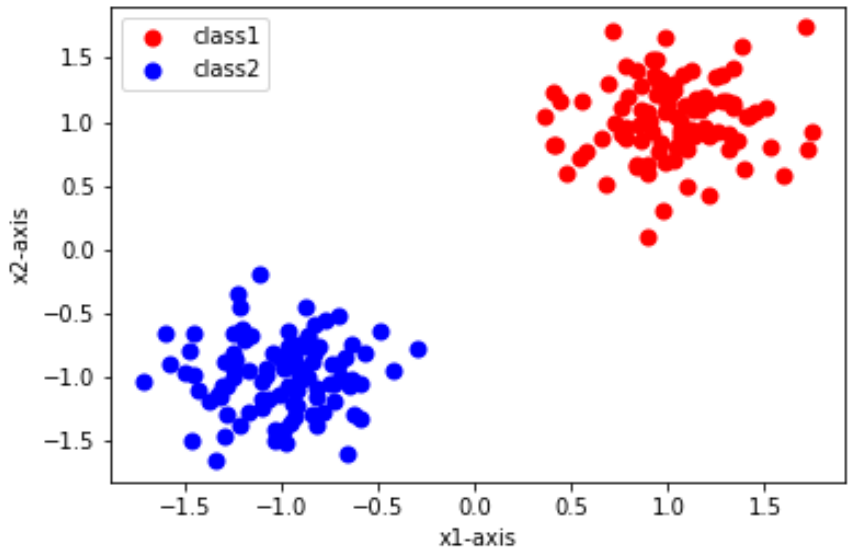
پاسخ ۲ – AdaLine and MadaLine

۲-۱. AdaLine

قسمت الف)

داده‌ها با استفاده از np.random.normal(mean,std,n) تولید شده است. برای اینکه داده‌های یکسان برای هربار تولید کنیم از random.seed استفاده کردیم.

سپس نمودار پراکندگی با استفاده از کتابخونه matplotlib رسم شده است.(شکل ۶)



شکل ۶. نمودار پراکندگی داده‌های تولید شده

قسمت ب)

برای این کار ابتدا داده‌های دو کلاس را با هم ترکیب کرده و بر می‌زنیم. و یک سری کارهای جزئی دیگر برای آماده کردن داده برای دادن به مدل را انجام دادیم. که این قدم در تابع preprocess_data انجام شده است.

سپس با توجه به الگوریتمی که در کتاب آمده است مدل Adaline را پیاده سازی کردیم. که در ادامه شرح مختصری از روش پیاده سازی می‌دهیم.

- مقدار دهی اولیه: برای این کار از تابع initialize(k) استفاده کردیم که مقدار رندومی را به وزن‌ها و بایاس می‌دهد. نکته مهم این است که وزن‌های تولید شده را در ضریبی به نام k ضرب می‌کنیم.که مقدار آن برابر ۰/۰۵ قرار دادیم تا وزن‌ها تا حد ممکن کوچک باشند.
- مرحله forward: وزن‌ها را در ورودی‌ها ضرب می‌کنیم. البته برای سرعت بالاتر و موازی سازی اجرا از روش vectorization استفاده کردیم. یعنی ورودی‌ها را به صورت برداری در وزن‌ها ضرب کردیم.

$$net = \sum_{i=1}^{n} w_i x_i + b \rightarrow net = np.dot(w, x) + b$$

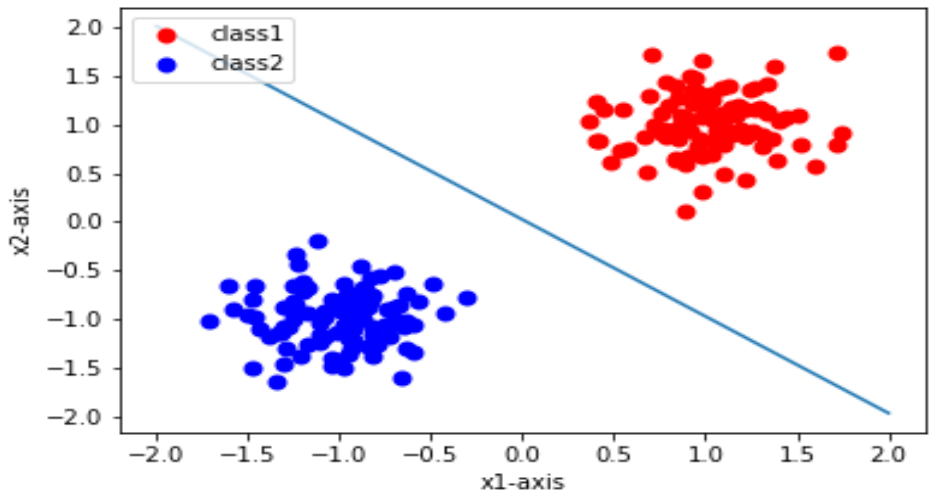
که ابعاد w: (1,2) است و ابعاد x: (2,1) است پس ضرب آن‌ها اسکالر می‌شود.

- استفاده از تابع فعال ساز: برای این کار هم از تابع فعال ساز اصلی این الگوریتم یعنی sign استفاده کریم و هم از نسخه پیشرفته‌تر یعنی tanh استفاده کردیم.
- بدست آوردن تابع هزینه: که در تابع calculate_cost_sign و calculate_cost_tanh با توجه به فرمول آن‌ها پیاده شده است.
- آپدیت کردن وزن‌ها: تابع update_delta و update_tanh برای این کار در نظر گرفته شده‌اند. که اینجا هم برای پرهیز از حلقه از روش ضرب برداری استفاده کردیم. برای همین مجبور به تغییر ابعاد ماتریس‌ها شدیم. نکته مهم مقدار آلفا یا نرخ یادگیری است که یک هایپر پارامتر است که مقدار آن تاثیر زیادی در تعداد ایپاک‌ها و نتیجه دارد.
- شرط توقف: اگر تابع هزینه از یک اپسیلونی کمتر شده از تابع خارج شده و آخرین وزن‌ها و بایاس را بر میگرداند. در غیر اینصورت تا ماکسیمم ایپاک داده شده ادامه می‌دهد.
- کل فرآیند آموزش در تابع Adaline با فراخوانی تابع‌های فوق انجام می‌شود.

در ادامه خط جدا کننده بدست آمده از مدل آورده شده است.

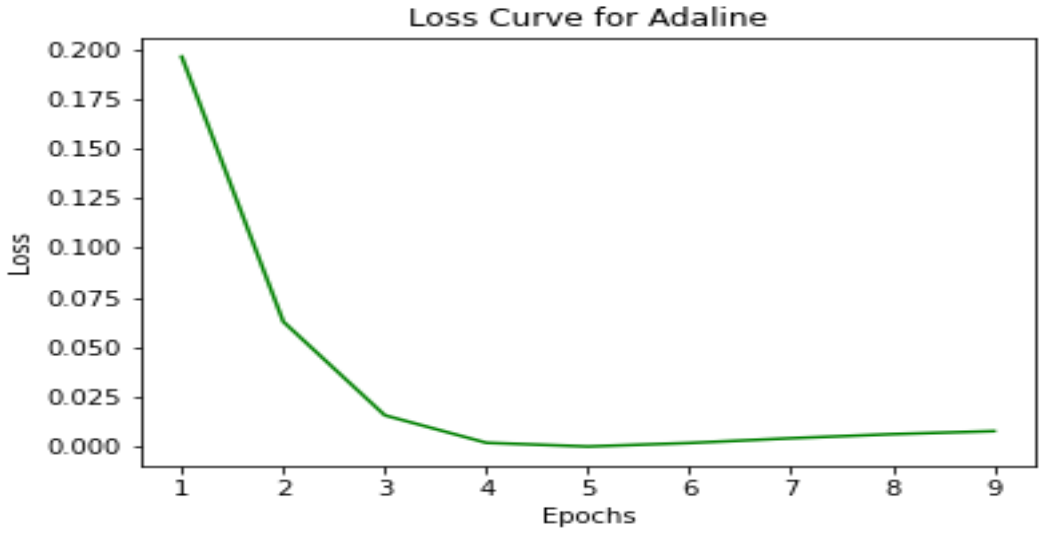
$$net = [-0.46854459 \quad -0.47096977\ ] \begin{bmatrix} x1 \\ x2 \end{bmatrix} + 0.0104582$$

که شکل خط جدا کننده فوق در شکل ۷ آمده است. خط جدا کننده به طور کامل و خوب هر دو داده را از هم جدا کرده است. دلیلشم این است که دو داده به طور خطی از هم قابل تفکیک هستند و تعداد هر دو کلاس یکسان است همچنین شکل توزیع آن‌ها مشابه است.



شگل ۷. نمودار پراکندگی همراه چدا کننده

نمودار هزینه در طول ایپاک در شکل ۸ آمده است. تابع هزنیه هم در طول زمان ابتدا کم شده است اما سپس به کندی افزایش پیدا کرده است.بهترین ایپاک به نظر ۵ است. با تعداد محدودی ایپاک خطا کم شد پس یعنی داده‌ها به راحتی قابل تفکیک هستند و نیاز به مدل پیچیده ندارند.

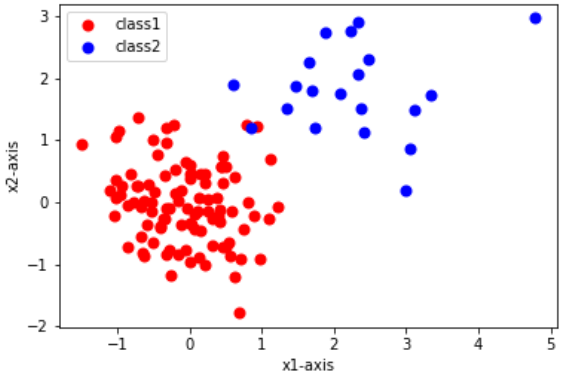


شکل ۸. نمودار تابع هزینه درگذر ایپاک

همین کار با تابع فعال ساز tanh هم انجام شده است که به دلیل مشابهت زیاد دیگر از آوردن نمودار آن صرف نظر می‌کنیم.

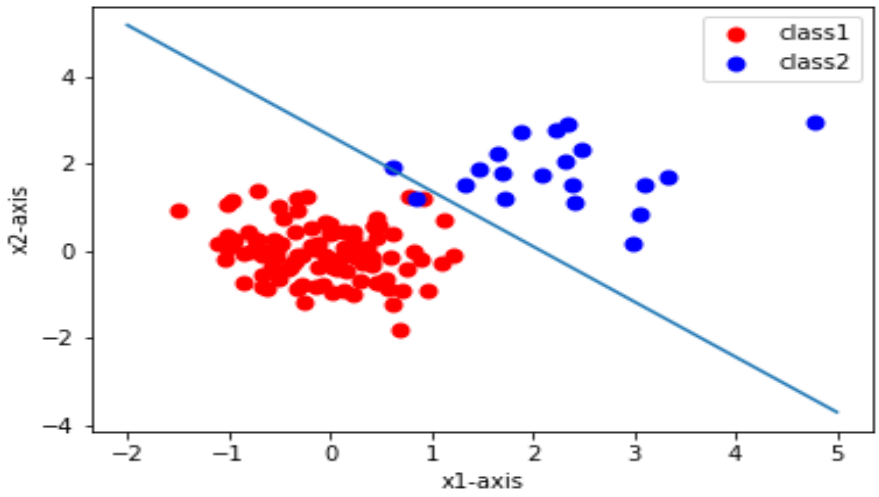
قسمت ج)

مانند قسمت الف داده‌های جدید تولید و نمایش داده می‌شوند.(شکل۹)



شکل ۹. نمودار پراکندگی داده‌های تولید شده – قسمت ج

برای این شکل هم مدل را با هر دو تابع فعال ساز فرخوانی می‌کنیم البته اینجا فقط نتایج برای sign آورده شده است. خط جدا کننده در شکل ۱۰ قابل مشاهده است.



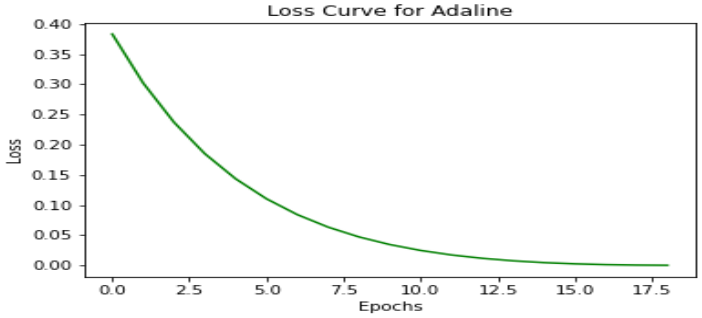
شگل ۱۰. نمودار پراکندگی همراه خط چدا کننده –قسمت ج

یکی از داده‌ها اشتباهی دسته بندی شده است. چون به صورت خطی دو دسته کاملا جدایی پذیر نیستند. برای همین با مدل خطی نمی‌توان آن‌ها را کاملا جدا کرد اما همین دسته بندی هم قابل قبول است جون داده آبی که درست تشخیص داده نشده است را هم اگر مدل حتی می‌توانست درست لیبل بزند مدل قدرت تعمیم خودش را از دست می‌داد.

نکته دیگر این است که با اینکه در کتاب گفته شده اگر تعداد و شکل توزیع داده‌ها مثل شکل بالا باشد تضمینی برای دسته بندی خوب وجود ندارد اما این مدل توانسته به خوبی چواب دهد.

برای همین مدل اما با تابع فعال ساز tanh هم تقریبا همین جواب را بدست می‌آوریم.

نمودار تغییرات تابع هزینه در طول ایپاک برای مدل Adaline برای دسته بندی داده‌ها در شکل ۱۱ آمده است.



شگل ۱۱. نمودار تابع هزینه درگذر ایپاک

نمودار شکل بالا نشان می‌دهد مقدار نرخ یادگیری مناسب بوده چون به صورت منطقی و با سرعت خوبی loss را کم کرده است.

مقایسه نتایج قسمت الف و ج:

- هر دو داده به خوبی جدا شده‌اند اما داده‌های قسمت اول کامل جدا اند درحالی که در قسمت ج یکی از داده‌ها اشتباهی دسته بندی شده است.
- مدل در قسمت الف به دلیل حاشیه زیادی که با هر دو کلاس دارد قدرت تعمیم بیشتری برای دسته بندی داده‌های دیده نشده از توزیع دارد.
- مدل در قسمت الف سریع‌تز loss را کاهش می‌دهد چون جدا کردن داده‌ها ساده‌تر است..
-

۱-۲. Madaline

قسمت الف)

MR1: این شبکه دو لایه دارد یک لایه هیدن و یک لایه خروجی. در این شبکه وزن‌های لایه خروجی ثابت است و عموما لایه خروجی نقش OR را دارد و وزن‌ها و بایاس این لایه به گونه‌ای تنظیم می‌شوند که OR خروجی نورون‌های لایه هیدن را بسازند. البته می‌توان به جای OR منطق AND هم استفاده کرد. نسخه‌های پیشرفته‌تر با تعداد هیدن لایه‌های بیشتر هم دارد که به آن نمی‌پردازیم.

۱. ابتدا وزن‌ها و بایاس رو مقداردهی اولیه می‌کنیم.
۲. یک حلقه می‌زنیم تا به شرط خاتمه برسیم:
۳. مقدار خروجی را برای هر نورون لایه مخفی بدست می‌اوریم.
۴. خروجی هر نورون را به تابع فعال ساز می‌دهیم که اگر خروجی مثبت باشد یک و اگر خروجی منفی باشد منفی یک را برمی‌گرداند.
۵. حالا مقدار خروجی برای لایه آخر را با استفاده از ضرب وزن‌ها در خروجی نورون‌های لایه مخفی بدست می‌آوریم و نتیجه را به تابع فعال ساز sign می‌دهیم.
۶. ارور را بدست می‌آوریم:

- اگر درست لیبل زده باشم نیازی به آپدیت وزن نیست.در غیر اینصورت:
- اگر لیبل درست ۱ باشد آنگاه یعنی ما -۱ لیبل زدیم پس تمام خروجی‌های نورون‌های لایه مخفی منفی هستند. پس اگر یکی را

مثبت کنیم خط بهتر می‌شود برای همین نزدیک‌ترین وزن به صفر را آپدیت می‌کنیم.

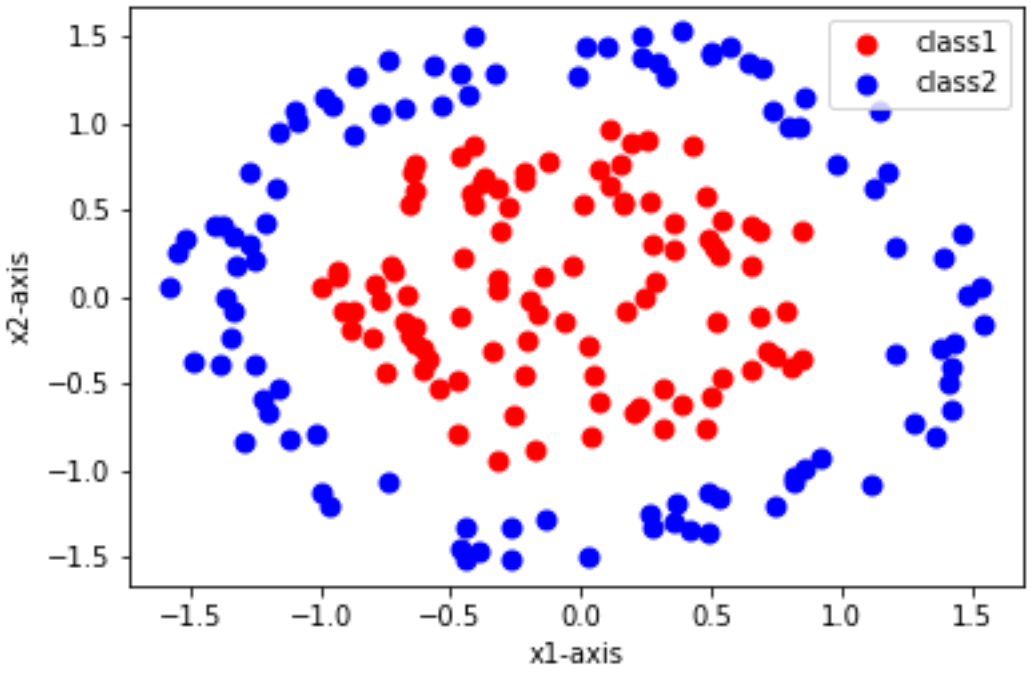
- اگر لیبل درست ۱- باشد یعنی لیبل ما ۱ است پس برای درست کردن باید تمام وزن‌های مثبت را کم کنیم تا منفی یا نزدیک منفی شوند.

۷. شرط خاتمه: اگر تغییر وزن متوقف شود یا به حداکثر تعداد ایپاک‌ها برسیم یا به ارور قابل قبولی برسیم.

قسمت ب)

آماده سازی دیتا: داده را ابتدا با کتابخانه pandas خوندیم و بعد shuffle کردیم. سپس برای اینکه الگوریتم MR1 برای دسته بندی لیبل biplolar نوشته شده ولی ورودی ما binary است، لیبل‌های ۰ را به ۱- تبدیل کردیم.

رسم کردن دیتا: شکل ۱۲ نمودار پراکندگی دو دسته داده را نشان می‌دهد.



شکل ۱۲. نمودار پراکندگی داده‌ها

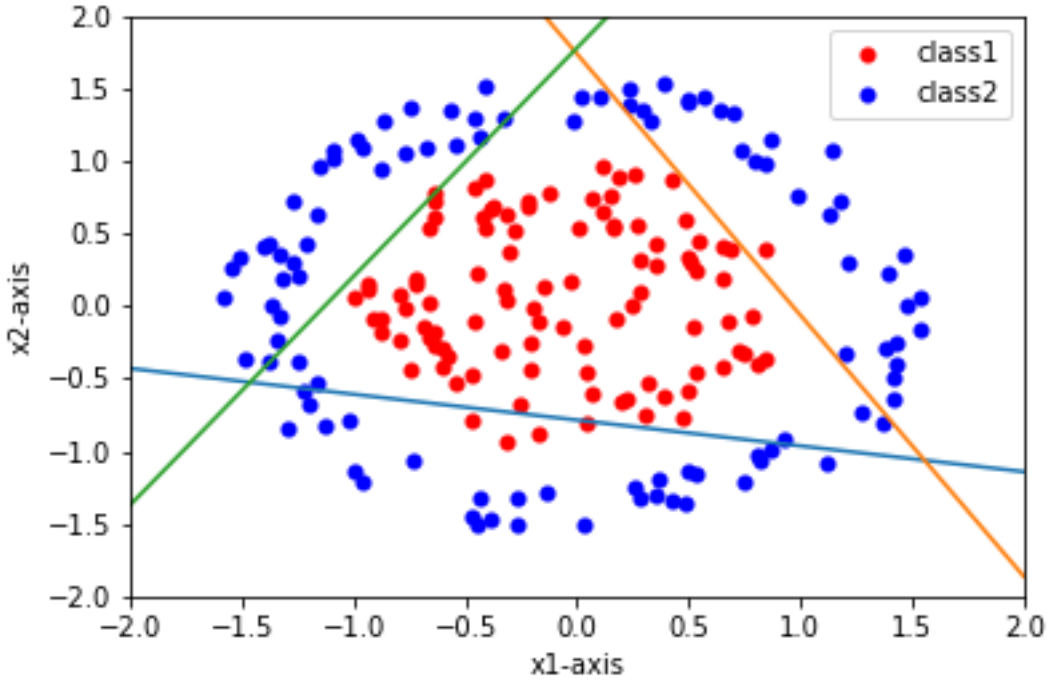
پیاده سازی: مطابق الگوریتم گفته شده در قسمت الف یعنی استفاده از MR1 .

- تقریبا در همه توابع از تکنیک vectorization استفاده شده است. برای همین برخی جاها از reshape استفاده کردیم تا قابل ضرب ماتریسی شوند.
- تابع هزینه میانگین ارور همه‌ی نمونه‌ها در هر ایپاک در نظر گرفته شده است. که خود ارور برای هر نمونه هم MSE در نظر گرفته شده است.
- وزن‌های اولیه کوچک در نظر گرفته شده است.
- برای وزن‌های ثابت لایه خروجی چون منطق OR داریم پس وزن همه نورون‌ها را یک در نظر گرفتیم و بایاس را برابر تعداد نورون‌ها منهای یک در نظر گرفتیم تا تنها وقتی مقدار خروجی منفی شود که همه نورون‌ها منفی یک باشند.

نتایج:

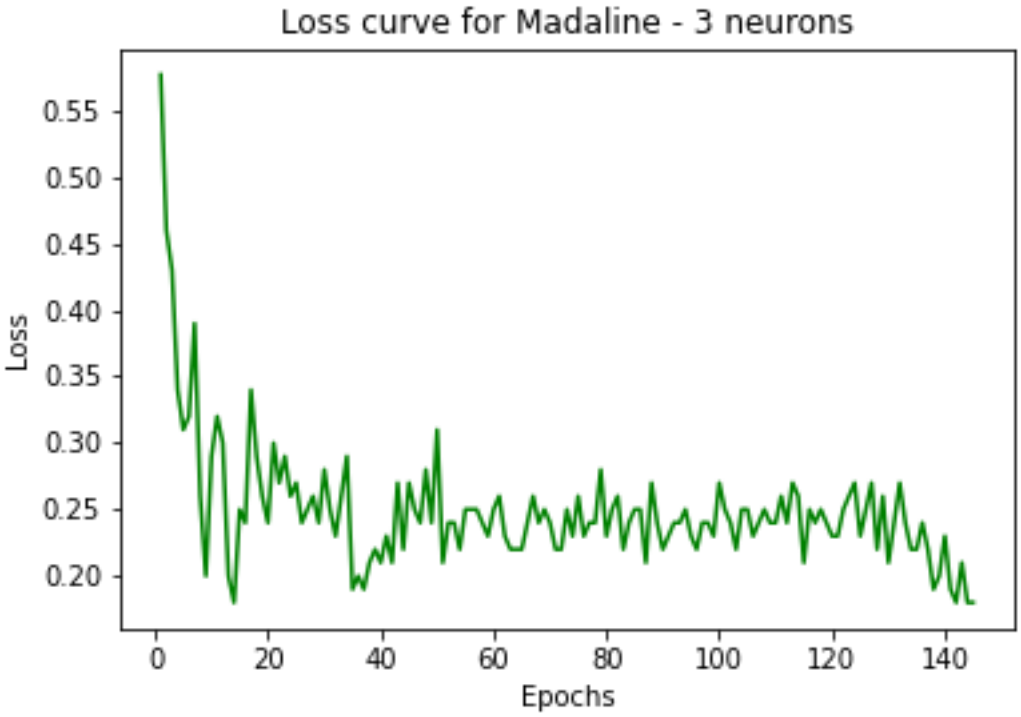
شبکه با ۳ نورون در لایه مخفی:

انتظار داریم که سه خط به ما بدهد که محیط داخل این مثلث مربوط به یک کلاس و خارج مثلث به کلاس دیگر باشد. گه در شکل۱۳ نتیجه طبقه بندی توسط مدل با ۳ نورون میانی نمایش داده شده است.



شکل ۱۳. طبقه بندی داده‌ها با ۳ نورون

نمودار loss آن هم به صورت شکل ۱۴ است. که همانطور که مشخص است با گذشت زمان کمتر می‌شود. این نوسانی بودنش هم نشان می‌دهد مسئله بهینه محلی زیاد دارد.



شکل ۱۴. نمودار میانگین **loss** با ۳ نورون

متریک‌ها:

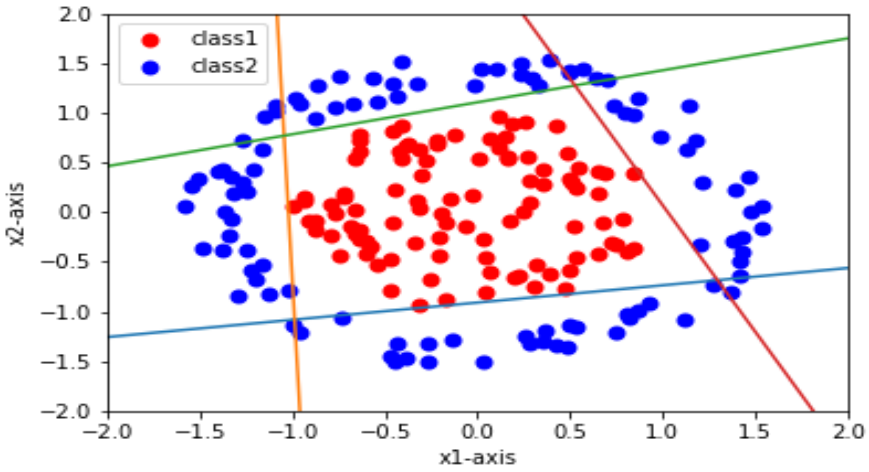
	precision	recall	f1-score	support
-1.0	0.92	0.95	0.94	100
1.0	0.95	0.92	0.93	100
accuracy			0.94	200
macro avg	0.94	0.94	0.93	200
weighted avg	0.94	0.94	0.93	200

تحلیل: دقت ۹۴ درصد شده است. و این بهترین درصدی است که با تغییر هایپرپارامترها مثل نرخ یادگیری و ضریب اولیه وزن بدست آوردم. و بعید است درصد بهتری بتوان بدست آورد. دلیل این موضوع که نمی‌توان دقت کامل داشت این است که فضای مرزی این دو کلاس در واقعیت منحنی شکل است پس هر چقدر تعداد اضلاع بیشتر باشد بهتر می‌توان طبقه بندی کرد. و سه ضلع برای این کار کم است.

تعداد ایپاک: ۱۴۵. البته بستگی به شرط توقف دارد. چون همانطور که گفته شد مسیر بهینه سازی دره و بهینه محلی زیاد دارد شرط توقف تاثیر مهمی در تعداد ایپاک این شبکه دارد که الگوریتم در کدام بهینه محلی بیاستد. اگر سریع توقف کنیم ممکن است جواب بهتری را که جلوتر است را از دست بدهیم اگر برای توقف سختگیری کنیم هم ممکن است از بهینه محلی قابل قبول بگذریم.

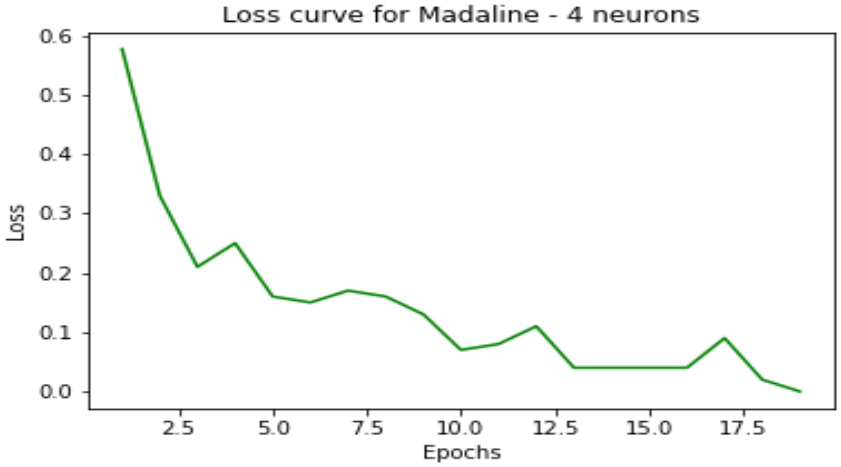
شبکه با ۴ نورون در لایه مخفی:

انتظار داریم که چهار خط به ما بدهد. که در شکل ۱۴ نتیجه طبقه بندی توسط مدل با ۴ نورون میانی نمایش داده شده است.



شکل ۱۵. طبقه بندی داده‌ها با ۴ نورون

نمودار تغییرات loss هم در شکل ۱۵ آورده شده است. این نمودار با شیب نسبتا ملایمی و با نوسان کم مقدار loss کمتر شده است. این یعنی بهینه محلی کمتری برای این مسئله وجود دارد.



شکل ۱۶. نمودار میانگین **loss** با ۴ نورون

متریک‌ها:

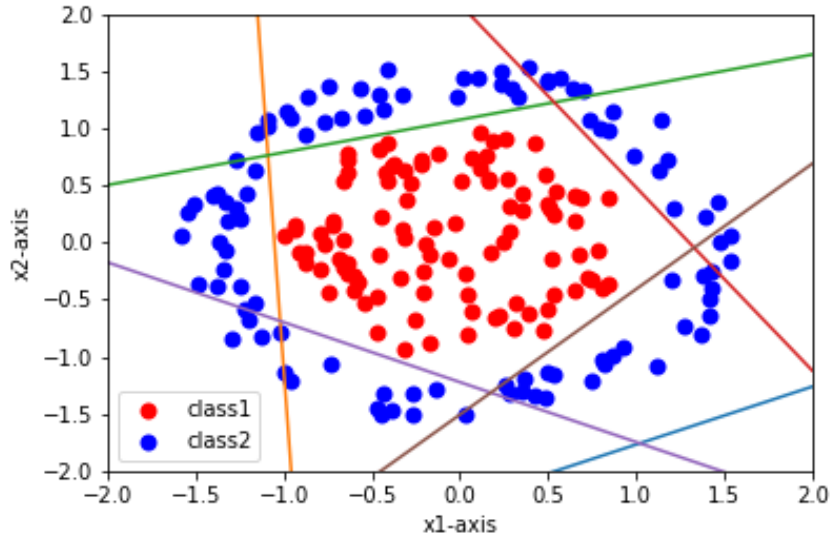
	precision	recall	f1-score	support
-1.0	1.00	1.00	1.00	100
1.0	1.00	1.00	1.00	100
accuracy			1.00	200
macro avg	1.00	1.00	1.00	200
weighted avg	1.00	1.00	1.00	200

تحلیل: دقت ۱۰۰ درصد است یعنی تمامی داده‌ها به درستی دسته بندی شده اند. پس مشخص می‌شود که چهار ضلع برای فیت کردن به این دیتا کافی است. البته باید در نظر داشت که این دقت روی خود داده ترین است. اگر داده تست بدهیم شاید دقت کم شود. از رو شکل به نظر قدرت تعمیم مناسبی هم دارد چون خیلی هم فیت داده‌ها نشده است.

تعداد ایپاک: ۱۹. که یعنی نرخ یادگیری مناسب است و مدل کار سختی برای فیت شدن ندارد. و فضای بهینه سازی تقریبا محدب است.

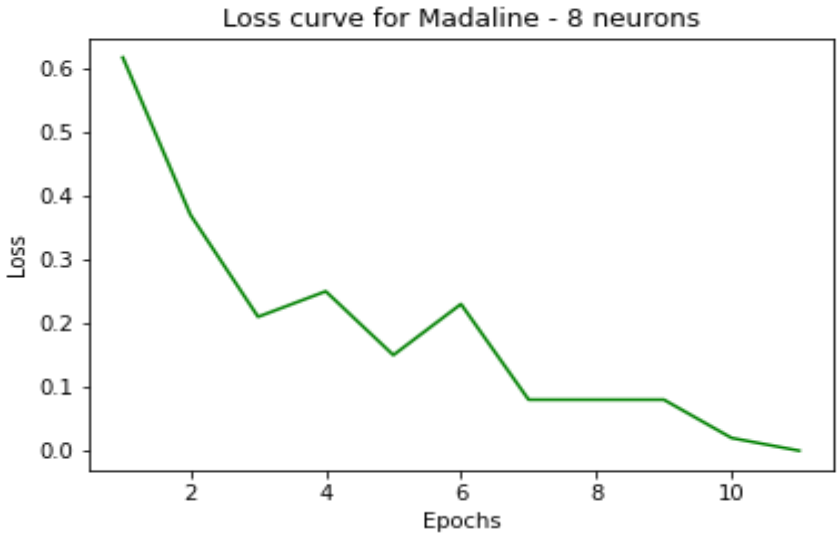
شبکه با ۸ نورون در لایه مخفی:

انتظار داریم هشت خط جدا کننده داشته باشیم. که در شکل ۱۶ آمده است.



شکل ۱۷. طبقه بندی داده‌ها با ۸ نورون

در شکل ۱۷ نمودار تغییرات loss برای این شبکه نمایش داده شده است. فضای بهینه سازی برای آن هم تقریبا محدب است..



شکل ۱۸. نمودار میانگین **loss** ۸ نورون

متریک‌ها:

	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

تحلیل: دقت آن ۱۰۰ درصد است. انعطاف بیشتری نسبت به مدل قبلی دارد. ولی از هر ۸ خط برای کلاس بندی استفاده نشده است و عملا فقط پنج خط در طبقه بندی اهمیت دارند. علتش میتونه این باشه که زود loss صفر میشه و فرصت آموزش برای خط‌های دیگر نیست و نیازی هم به آن‌ها نیست چون همون پنج خط برای دقت کامل کفایت می‌کند.

تعداد ایپاک: ۱۱. نرخ یادگیری مناسب است. مدل به خوبی از عهده مدل کردن داده بر می‌آید.

تحلیل کلی:

- اگر تعداد نورون‌ها را بیشتر کنیم مدل بهتر می‌تواند به دیتا فیت شود.
- وقتی تعداد نورون‌ها بیشتر می‌شود الگوریتم سریع‌تر loss را کم می‌کند. علتش هم این است که وقتی تعداد خط‌ها زیاد است وظیفه طبقه بندی داده بر روی تعداد بیشتری خط است و نیاز کمتری به تغییرات وزن خود دارند. در صورتی که اگر تعداد خط کم باشد برای فیت شدن تغییرات زیادی وزنش می‌کند تا بتواند نقاط بیشتری را تحت پوشش قرار دهد.

پاسخ ۳ – Restricted Boltzmann Machine

۳-۱. سیستم توصیه‌گر

ددلاینش نرسیده است. جواب این تمرن بعدا ارسال خواهد شد.

.

پاسخ ۴ – پرسپترون چند لایه (MLP)

۴-۱. Multi Layer Perceptron

A)

در ابتدا فایل csv خوانده شده و داده را لود میکنیم که شامل ۲۱۶۱۳ داده در ۲۱ ستون می باشد. پس از فراخوانی تابع info مشاهده میکنیم که تمامی فیچرها numerical (int , float) هستند و تنها داده categorical (object) در ستون تاریخ قرار دارد و همچنین جزییات دیگری شامل تعداد داده های غیر نال را میبینیم.

جدول ۳. نوع داده ستون های **dataframe**

column	Dtype
id	int64
date	object
price	float64
bedrooms	int64
Bathrooms	float64
sqft_living	int64
sqft_lot	int64
Floors	float64
Waterfront	int64
View	int64
Condition	int64
grade	int64
sqft_above	int64
sqft_basement	int64
yr_built	int64
yr_renovated	int64
zipcode	int64
lat	float64
long	float64

sqft_living15	int64
sqft_lot15	int64

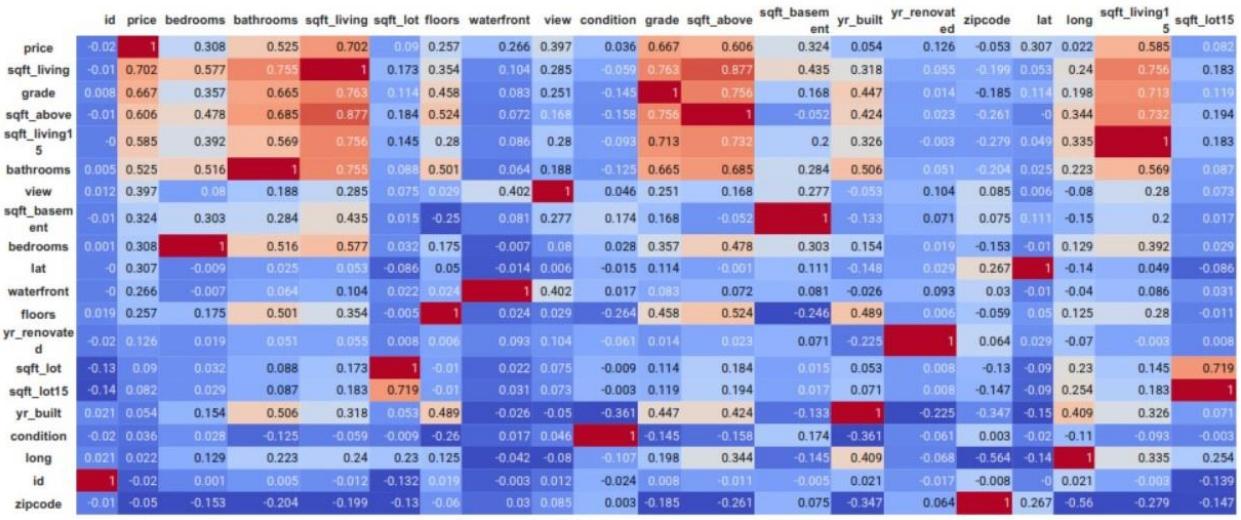
.

B)

در این بخش از ما خواسته شده که تعداد داده های Nan تمامی ستون ها گزارش شود، همانطور که پیشتر گفته شد با استفاده از دستور info در بخش قبل و همینطور دستور isnull میتوان تعداد داده های تهی را بدست آورد که مشاهده شد هیچ ستونی دارای داده Nan نمی باشد.

C)

پس از رسم ماتریس correlation و جدا کردن فیچر قیمت و سپس مرتب کردن فیچرها بر اساس بزرگترین correlation مشاهده می شود که فیچر sqft_living (square footage of the home) بیشترین correlation را با price دارد.



	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
price	-0.02	1	0.308	0.525	0.702	0.09	0.257	0.266	0.397	0.036	0.667	0.606	0.324	0.054	0.126	-0.053	0.307	0.022	0.585	0.082
sqft_living	-0.01	0.702	0.577	0.755	1	0.173	0.354	0.104	0.285	-0.059	0.763	0.877	0.435	0.318	0.055	-0.199	0.053	0.24	0.756	0.183
grade	0.008	0.667	0.357	0.665	0.763	0.114	0.458	0.083	0.251	-0.145	1	0.756	0.168	0.447	0.014	-0.185	0.114	0.198	0.713	0.119
sqft_above	-0.01	0.606	0.478	0.685	0.877	0.184	0.524	0.072	0.168	-0.158	0.756	1	-0.052	0.424	0.023	-0.261	-0	0.344	0.732	0.194
sqft_living15	-0	0.585	0.392	0.569	0.756	0.145	0.28	0.086	0.28	-0.093	0.713	0.732	0.2	0.326	-0.003	-0.279	0.049	0.335	1	0.183
bathrooms	0.005	0.525	0.516	1	0.755	0.088	0.501	0.064	0.188	-0.125	0.665	0.685	0.284	0.506	0.051	-0.204	0.025	0.223	0.569	0.087
view	0.012	0.397	0.08	0.188	0.285	0.075	0.029	0.402	1	0.046	0.251	0.168	0.277	-0.053	0.104	0.085	0.006	-0.08	0.28	0.073
sqft_basement	-0.01	0.324	0.303	0.284	0.435	0.015	-0.25	0.081	0.277	0.174	0.168	-0.052	1	-0.133	0.071	0.075	0.111	-0.15	0.2	0.017
bedrooms	0.001	0.308	1	0.516	0.577	0.032	0.175	-0.007	0.08	0.028	0.357	0.478	0.303	0.154	0.019	-0.153	-0.01	0.129	0.392	0.029
lat	-0	0.307	-0.009	0.025	0.053	-0.086	0.05	-0.014	0.006	-0.015	0.114	-0.001	0.111	-0.148	0.029	0.267	1	-0.14	0.049	-0.086
waterfront	-0	0.266	-0.007	0.064	0.104	0.022	0.024	1	0.402	0.017	0.083	0.072	0.081	-0.026	0.093	0.03	-0.01	-0.04	0.086	0.031
floors	0.019	0.257	0.175	0.501	0.354	-0.005	1	0.024	0.029	-0.264	0.458	0.524	-0.246	0.489	0.006	-0.059	0.05	0.125	0.28	-0.011
yr_renovated	-0.02	0.126	0.019	0.051	0.055	0.008	0.006	0.093	0.104	-0.061	0.014	0.023	0.071	-0.225	1	0.064	0.029	-0.07	-0.003	0.008
sqft_lot	-0.13	0.09	0.032	0.088	0.173	1	-0.01	0.022	0.075	-0.009	0.114	0.184	0.015	0.053	0.008	-0.13	-0.09	0.23	0.145	0.719
sqft_lot15	-0.14	0.082	0.029	0.087	0.183	0.719	-0.01	0.031	0.073	-0.003	0.119	0.194	0.017	0.071	0.008	-0.147	-0.09	0.254	0.183	1
yr_built	0.021	0.054	0.154	0.506	0.318	0.053	0.489	-0.026	-0.05	-0.361	0.447	0.424	-0.133	1	-0.225	-0.347	-0.15	0.409	0.326	0.071
condition	-0.02	0.036	0.028	-0.125	-0.059	-0.009	-0.26	0.017	0.046	1	-0.145	-0.158	0.174	-0.361	-0.061	0.003	-0.02	-0.11	-0.093	-0.003
long	0.021	0.022	0.129	0.223	0.24	0.23	0.125	-0.042	-0.08	-0.107	0.198	0.344	-0.145	0.409	-0.068	-0.564	-0.14	1	0.335	0.254
id	1	-0.02	0.001	0.005	-0.012	-0.132	0.019	-0.003	0.012	-0.024	0.008	-0.011	-0.005	0.021	-0.017	-0.008	-0	0.021	-0.003	-0.139
zipcode	-0.01	-0.05	-0.153	-0.204	-0.199	-0.13	-0.06	0.03	0.085	0.003	-0.185	-0.261	0.075	-0.347	0.064	1	0.267	-0.56	-0.279	-0.147

شکل ۱۹. ماتریس **correlation** جدول

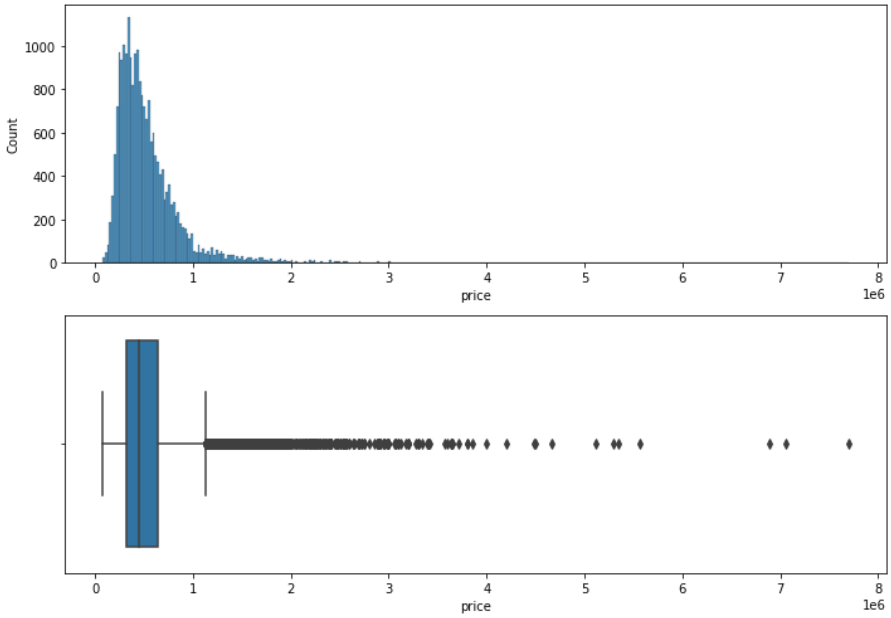
	price
price	1.000
sqft_living	0.702
grade	0.667
sqft_above	0.606
sqft_living15	0.585
bathrooms	0.525
view	0.397
sqft_basement	0.324
bedrooms	0.308
lat	0.307
waterfront	0.266
floors	0.257
yr_renovated	0.126
sqft_lot	0.090
sqft_lot15	0.082
yr_built	0.054
condition	0.036
long	0.022
id	-0.017
zipcode	-0.053

شکل ۲۰. میزان رابطه **price** با سایر فیچرها

از آنجایی که ستون ID دارای مقادیر یکتا می باشد و رابطه ای با price ندارد در ادامه این ستون را حذف می کنیم.

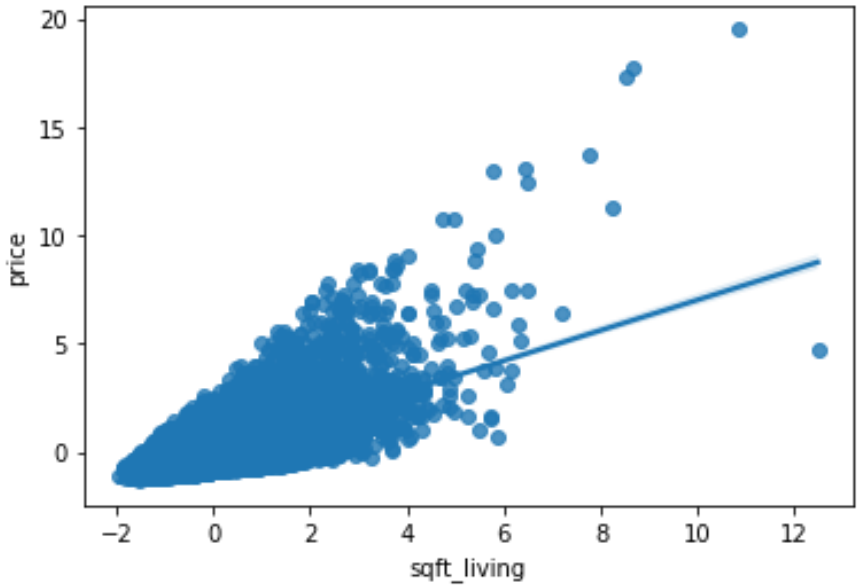
(D

با بررسی نمودار توزیع قیمت مشاهده می کنیم که بیشتر قیمت ها در بازه صفر تا یک میلیون قرار دارد و همچنین تعدادی outlier تا ۸ میلیون نیز دیده می شود.



شکل ۲۱. نمودار توزیع price

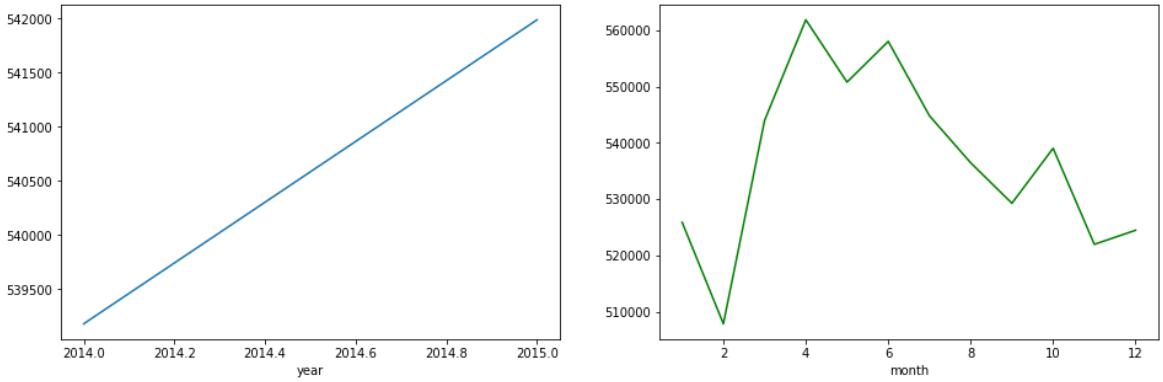
فیچر sqft_living با قیمت بالاترین correlation را داراست که با رسم نمودار scatter مشاهده می شود که این دو فیچر با هم رابطه مثبت دارند.



شکل ۲۲. نمودار رابطه price و sqft_living

(E

همانطور که خواسته شده ستون date به دو ستون ماه و سال تقسیم شد و ستون date نیز از جدول حذف شده است. در ادامه با رسم نمودار بررسی میکنیم که قیمت در ماه و سال چطور تغییر میکند.



شکل ۲۳. نمودار price در ماه و سال

قیمت مسکن هر ساله افزایش داشته و در تابستان قیمت مسکن نسبت به سایر ماه ها بالاتر بوده است.

(F

در ابتدا ۸۰ درصد داده را به train و ۲۰ درصد به test اختصاص داده شد، سپس داده آموزش به دو قسمت train و validation تقسیم شده است. تقسیم بندی داده ها به شرح زیر می باشد:

جدول ۴. split داده ها

X Train	(13832, 20)
Y Train	(13832, 1)
X Validation	(3458, 20)
Y Validation	(3458, 1)
X Test	(4323, 20)
Y Test	(4323, 1)

(G

در این قسمت با استفاده از دستور MinMaxScaler داده های آموزش و تست را اسکیل کرده و تمامی داده ها بین محدوده صفر و یک قرار میگیرند.

(H

در این قسمت یک MLP با دو لایه مخفی ساخته شد که ورودی شبکه ۲۰ (x_train_scaled.shape[1]) و خروجی ۱ (y_train.shape[1]) هست و لایه های مخفی دارای ۳۰ و ۱۵ نورون می باشند. از تابع فعال ساز Relu در لایه های شبکه استفاده شده است.

```
MLP(
  (input_fc): Linear(in_features=20, out_features=30, bias=True)
  (hidden_fc): Linear(in_features=30, out_features=15, bias=True)
  (output_fc): Linear(in_features=15, out_features=1, bias=True)
)
```

I)

چهار شبکه با دو تابع loss (L1,MSE) و دو تابع optimizer (SGD,Adam) متفاوت طراحی شده است در جزییات آن به شرح زیر می باشد:

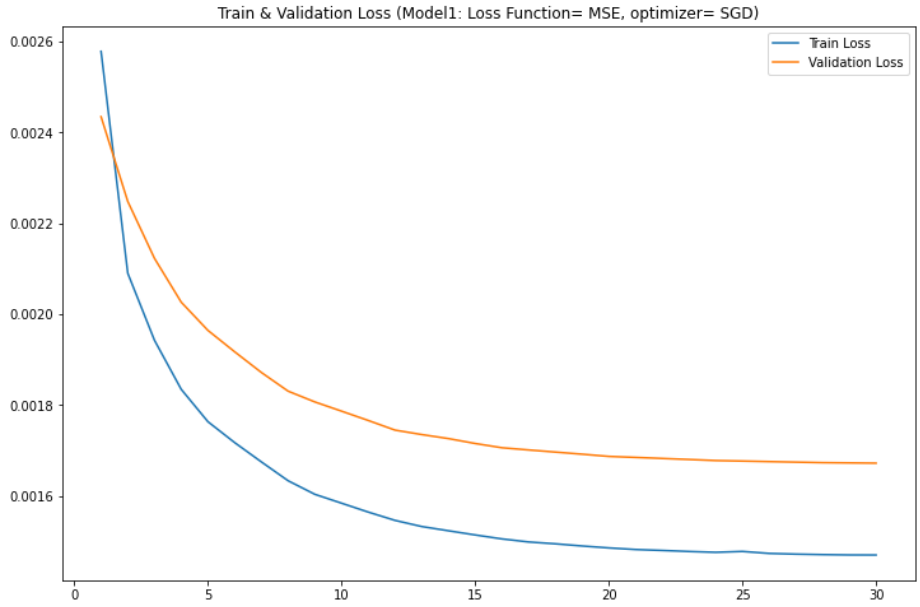
جدول ۵. Net hyper parameters

Network Used Hyper Parameters		
Batch Size		10
Epochs		30
Input		20
Hidden layer 1		30
Hidden layer 2		15
Output		1
Activation Functions		ReLU
➢ **Model1**	Loss Function	MSE
	Optimizer	SGD
	Learning rate	0.001
	momentum	0.9
	Weight decay	0.000004
	Scheluler	LR decreases by factor of 0.5 every 4 epochs
	Minimum Train Loss reached	0.0008
	Minimum Val Loss reached	0.0010
➢ **Model2**	Loss Function	MSE

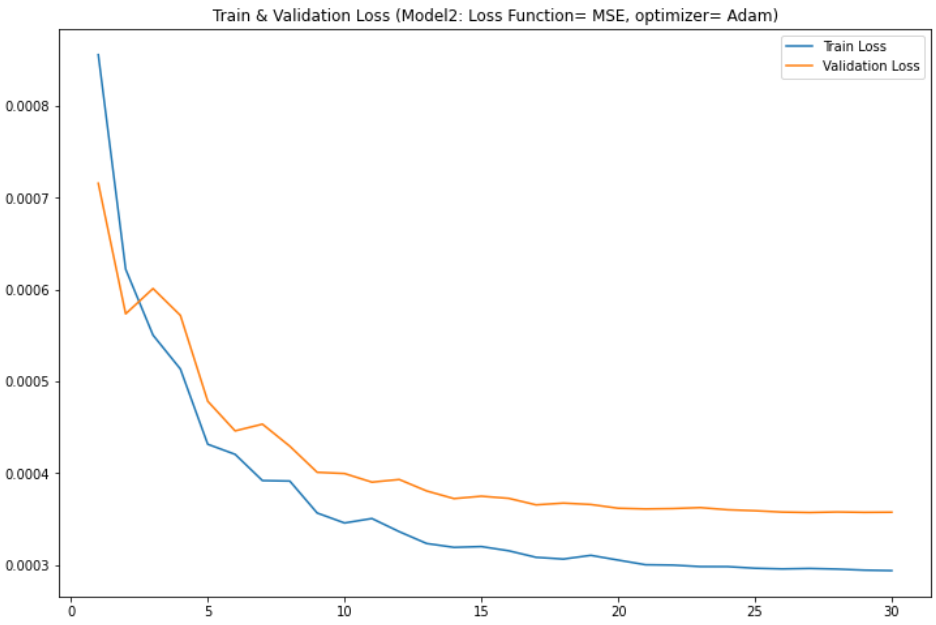
	Optimizer	Adam
	Scheluler	LR decreases by factor of 0.5 every 4 epochs
	Minimum Train Loss reached	0.0002
	Minimum Val Loss reached	0.0003
➢ **Model3**	Loss Function	L1
	Optimizer	SGD
	Learning rate	0.001
	momentum	0.9
	Weight decay	0.000004
	Scheluler	LR decreases by factor of 0.5 every 4 epochs
	Minimum Train Loss reached	0.0139
	Minimum Val Loss reached	0.0144
➢ **Model4**	Loss Function	L1
	Optimizer	Adam
	Scheluler	LR decreases by factor of 0.5 every 4 epochs
	Minimum Train Loss reached	0.0096
	Minimum Val Loss reached	0.0103

J)

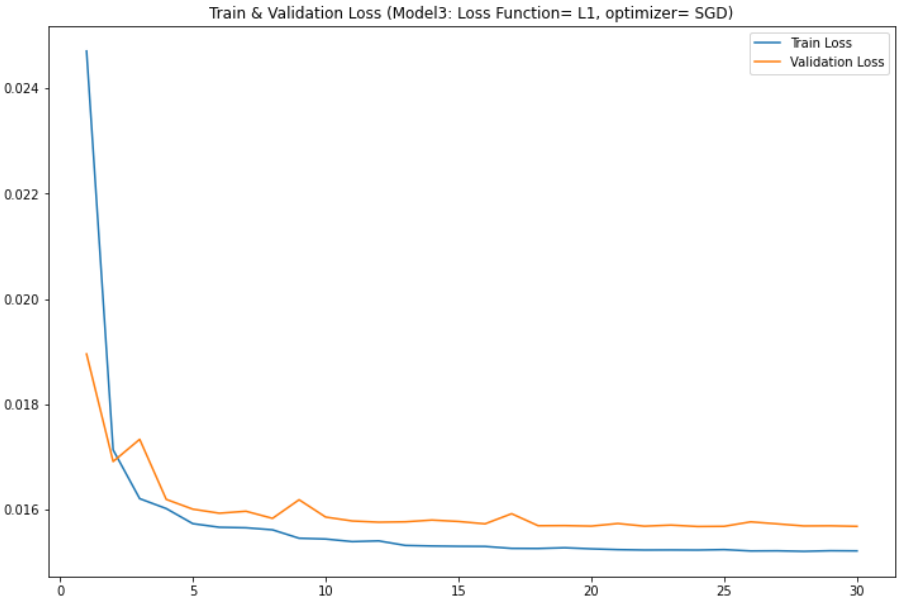
در این بخش هر چهار مدل را با تابع هزینه و optimizerهای متفاوت آموزش داده و نمودار loss و validation loss آنها بعد از ۳۰ ایپاک بررسی شده است.



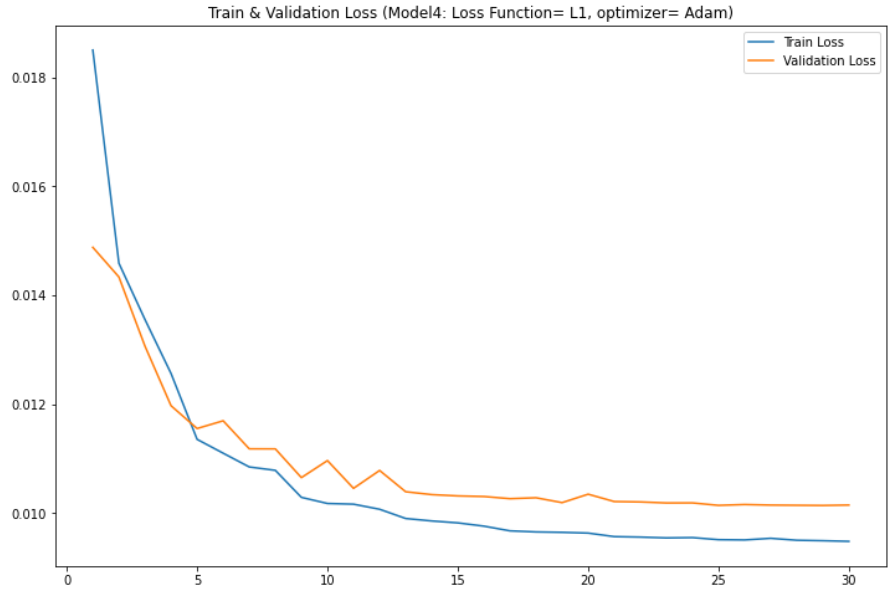
شکل ۲۴. Train & Validation Loss model1



شکل ۲۵. Train & Validation Loss model2



شکل ۲۶. Train & Validation Loss model3



شکل ۲۷. Train & Validation Loss model4

با توجه به شکل فوق دیده می شود بعد از گذشت ۲۵ الی ۳۰ ایپاک مقدار خطا هم برای داده های آموزش و هم ارزیابی بهبود آنچنانی نداشته است. با توجه به Scheluler بعد از گذشت ۴ ایپاک سرعت آموزش نصف می شود تقریبا در تمام مدل ها به جز مدل اول نوساناتی در مقدار خطا دیده می شود که بعد از گذشت ۱۵ الی ۲۰ ایپاک نوسان کم شده و سیر نزولی نمودار طی می شود.

همانطور که انتظار می رفت در مسئله رگرشن مدل با optimizer، Adam و تابع هزینه MSE عملکرد بهتری دارد، با توجه به مدل ۲ که دارای چنین پارامترهایی هست می بینیم که مقدار خطا از مینیمم مقدار شروع شده و به مینیمم مقدار به نسبت سایر خطاها رسیده است پس در ادامه از این مدل برای پیشبینی استفاده می شود.

(K

در این بخش پنج داده به طور تصادفی از دادگان تست انتخاب شده و با استفاده از مدل دوم قیمت را برای آنها پیش بینی کرده و سپس فاصله قیمت پیش بینی شده با قیمت واقعی اندازه گرفته شده است:

Data 1:

Predicted price= 369369.3, Price= 366999.9

difference between predicted and real price= 2369.3

Data 2:

Predicted price= 249242.5, Price= 275000

difference between predicted and real price= 25757.5

Data 3:

Predicted price= 416184.9, Price= 374999.9

difference between predicted and real price= 41185

Data 4:

Predicted price= 294895.8, Price= 210000

difference between predicted and real price= 84895.8

Data 5:

Predicted price= 230283.9, Price= 259999.9

difference between predicted and real price= 29716