به نام خدا

دانشگاه تهران

دانشکده مهندسی برق

و کامپیوتر

درس شبکه‌های عصبی و یادگیری عمیق

تمرین اول

نام و نام خانوادگی	مریم دادخواه – جواد سراج
شماره دانشجویی	810101151 - 810101186
تاریخ ارسال گزارش	1401.12.25

فهرست

شکل‌ها

جدول‌ها

پاسخ ۱ . شبکه ی عصبی Mcculloch-Pitts

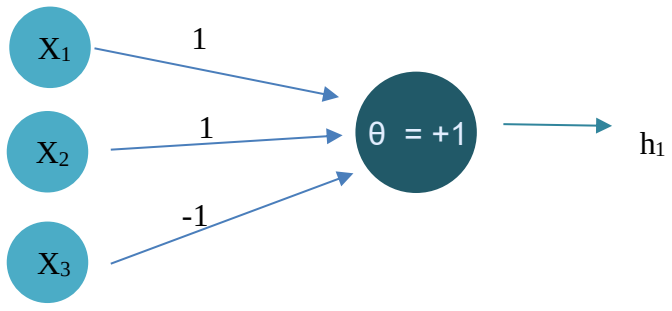
۱-۱. ماشین متناهی قطعی (DFA)

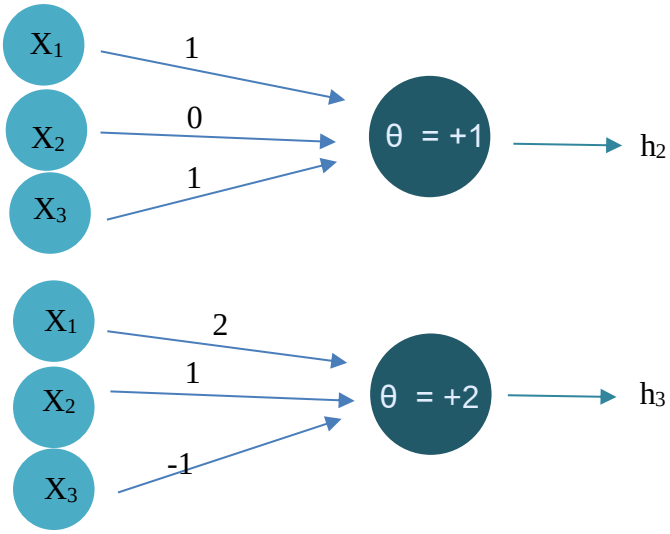
الف)

X_1	X_2	X_3	h_1	h_2	h_3
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	1	0	0
0	1	1	0	1	0
1	0	0	1	1	1
1	0	1	0	1	0
1	1	0	1	1	1
1	1	1	1	1	1

- $X_1 + X_2$: current state
- X_3 : input
- $h_1 + h_2$: next state
- h_3 : accept / reject

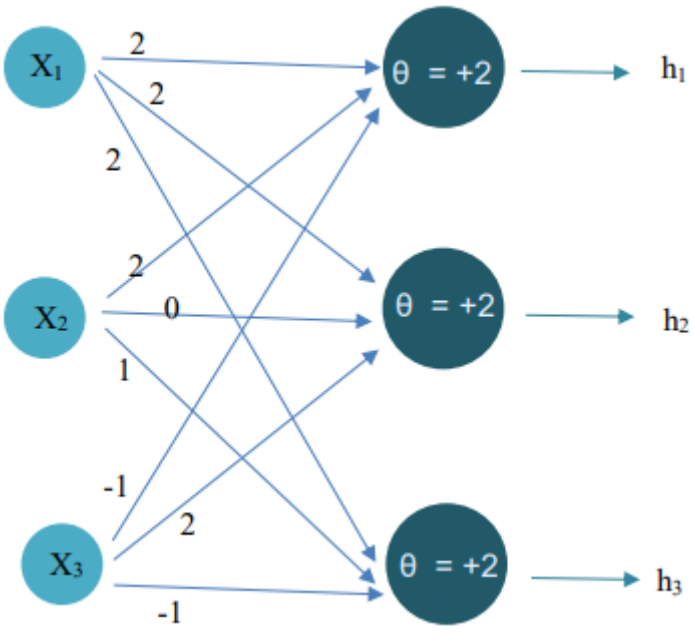
ب)





سه شکل بالا نشان دهنده سه شبکه **Mcculloch-Pitts** برای سه خروجی تعریف شده است. وزن های هر سه شبکه و نیز threshold ، با توجه به جدول قسمت الف محاسبه شده است. در شبکه های بالا X_i ها ورودی های شبکه هستند و اعداد روی هر یال ، وزن های شبکه به ازای ورودی متناظر است.

ج)



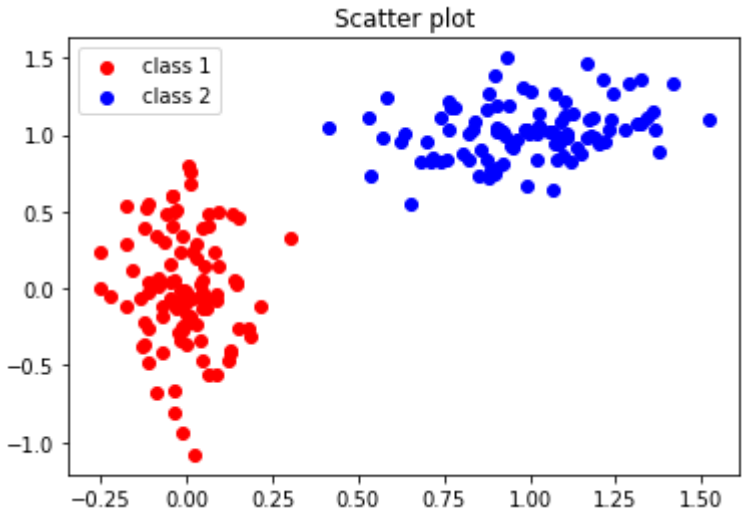
د)

```
DFA with current state as 11 with input as 1 goes to state  11  with output  1
DFA with current state as 11 with input as 0 goes to state  11  with output  1
DFA with current state as 10 with input as 1 goes to state  01  with output  0
DFA with current state as 10 with input as 0 goes to state  11  with output  1
DFA with current state as 01 with input as 1 goes to state  01  with output  0
DFA with current state as 01 with input as 0 goes to state  10  with output  0
DFA with current state as 00 with input as 1 goes to state  01  with output  0
DFA with current state as 00 with input as 0 goes to state  00  with output  0
```

پاسخ ۲. شبکه های AdaLine و MadaLine

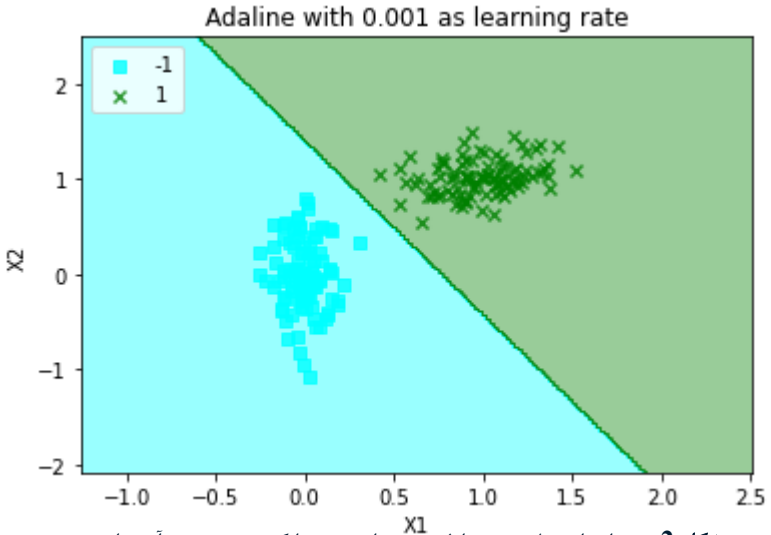
۱-۱. AdaLine

الف)

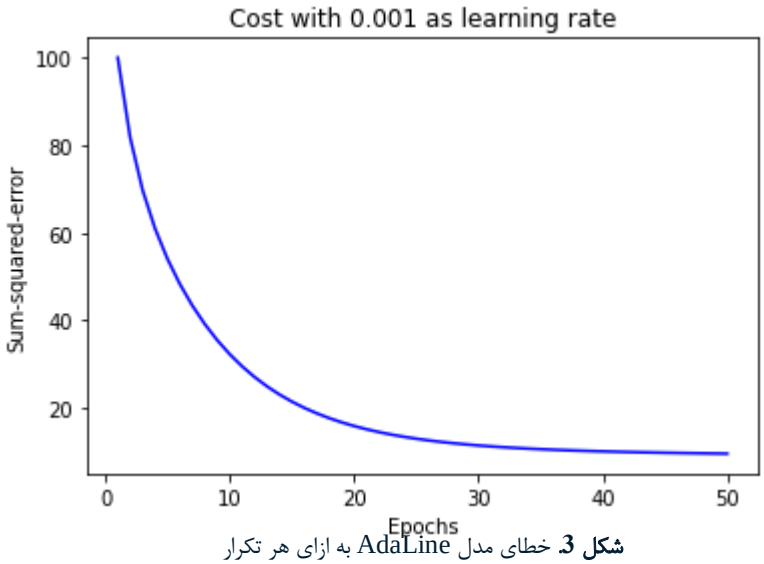


شکل 1. نمودار پراکندگی دو دسته داده اول

ب)



شکل 2. نمودار داده های دسته اول به همراه مرز جدا کننده به دست آمده از AdaLine



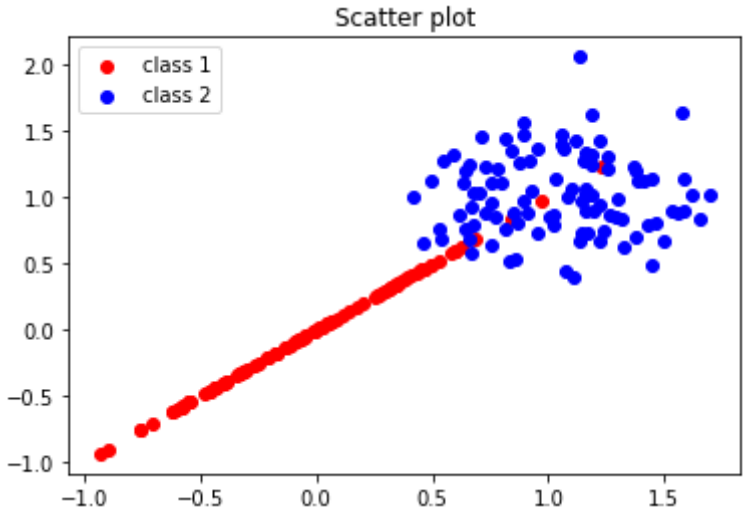
شکل 3. خطای مدل AdaLine به ازای هر تکرار

0	-0.890936
1	1.161293
2	0.635008

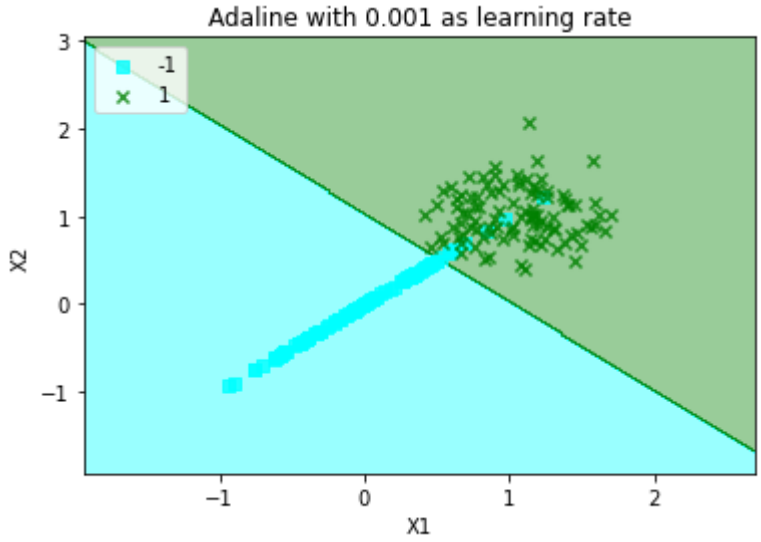
شکل 4. وزن های مدل AdaLine

همانطور که در شکل 2 مشاهده می شود، خط به دست آمده با مدل AdaLine کاملا داده های دو کلاس را از هم جدا می کند. دلیل این امر این است که دو کلاس از یکدیگر جدایی پذیر خطی هستند و نیز فاصله بین دو کلاس نیز به اندازه کافی زیاد است. همچنین با توجه به شکل 3، خطای مدل در حدود تکرار های 40 به بعد به صفر میل می کند.

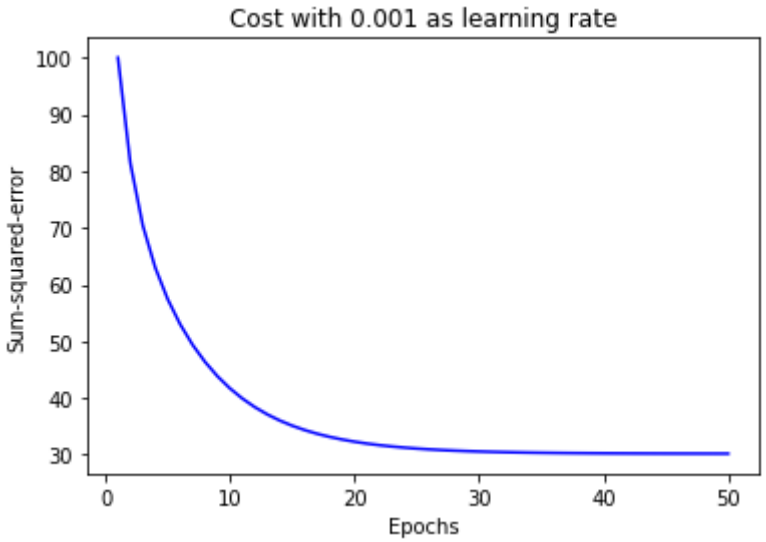
ج)



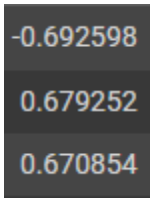
شکل 5. نمودار پراکندگی دو دسته داده دوم



شکل 6. نمودار داده های دسته دوم به همراه مرز جدا کننده از AdaLine



شکل 7. خطای مدل AdaLine به ازای هم تکرار



شکل 8. وزن های مدل AdaLine

د)

با توجه به نمودار پراکندگی داده های قسمت ب ، داده ها به صورت خطی جداپذیر نیستند و بنابراین انتظار می رود که AdaLine نتواند دوکلاس را به خوبی از یکدیگر تفکیک کند. همانطور که در نمودار شکل 6 نیز می شود، مرز جدا کننده نتوانسته داده های دو کلاس را کاملا از یکدیگر تفکیک کند و نیز با توجه به نمودار هزینه به ازای تکرار های مختلف که در شکل 7 مشاهده می شود ، در نهایت خطای الگوریتم در حدود 30 باقی می ماند و از حدود تکرار 20 به بعد خطای مدل کمتر نمی شود.

MadaLine .۲-۱

الف)

تفاوت دو الگوریتم MRI و MRII در این است که در اولی فقط وزن های مربوط به لایه های مخفی AdaLine تنظیم می شوند ولی در دومی همه ی وزن های شبکه تنظیم می شوند.

توضیح الگوریتم MRI :

1) ابتدا مقادیر کوچکی را به صورت رندوم به به وزن های لایه مخفی AdaLine اختصاص می دهیم. همچنین باید مقداری را نیز برای learning rate تعیین کنیم.
2) تا زمانی که شرط توقف برقرار نشده است، مراحل 3 تا 9 را تکرار کن.
3) به ازای هر داده ورودی به فرم S:t مراحل 4 تا 8 را تکرار کن. (توجه شود که t برچسب داده است که در این الگوریتم bipolar در نظر گرفته می شود)
4) به ازای هر داده ی ورودی S_i داریم : $X_i = S_i$
5) برای هر نورون لایه مخفی AdaLine ، net input را طبق فرمول زیر محاسبه می کنیم:

$Z_{in\_1} = b_1 + X_1W_{11} + X_2W_{21}$
$Z_{in\_2} = b_2 + X_1W_{12} + X_2W_{22}$

6) خروجی هر لایه مخفی AdaLine را طبق فرمول زیر محاسبه می کنیم:

$Z_1 = f(Z_{in\_1})$
$Z_2 = f(Z_{in\_2})$

که تابع F در فرمول بالا بصورت زیر تعریف می شود:

$$f(x) = \begin{cases} 1 & \text{if } x \geq 0; \\ -1 & \text{if } x < 0. \end{cases}$$

7) خروجی شبکه را طبق فرمول زیر تعیین می کنیم:

$$y\_in = b_3 + z_1v_1 + z_2v_2;$$
$$y = f(y\_in).$$

8) تشخیص خطا و آپدیت کردن وزن های لایه مخفی به این صورت است که اگر y = t بود (خطا رخ نداده بود) هیچ آپدیتی صورت نمی گیرد. در غیر این صورت آپدیت وزن های لایه مخفی طبق رابطه زیر صورت می گیرد:

اگر t = 1 ، تنها وزن نود هایی آپدیت می شود که net input آنها به 0 نزدیک تر است. آپدیت وزن ها نیز طبق رابطه زیر محاسبه می شود:

$$b_J(\text{new}) = b_J(\text{old}) + \alpha(1 - z\_in_J),$$
$$w_{iJ}(\text{new}) = w_{iJ}(\text{old}) + \alpha(1 - z\_in_J)x_i;$$

اگر t = -1 ، وزن همه نود هایی که net input مثبت داشته اند طبق رابطه زیر آپدیت می شود:

$$b_k(\text{new}) = b_k(\text{old}) + \alpha(-1 - z\_in_k),$$
$$w_{ik}(\text{new}) = w_{ik}(\text{old}) + \alpha(-1 - z\_in_k)x_i.$$

9) شرط توقف : اگر وزن ها دیگر آپدیت نشد یا به نتایج قابل قبولی رسید و یا اگر به ماکزیمم تعداد تکرار رسد متوقف شوید و در غیر این صورت ادامه دهید.

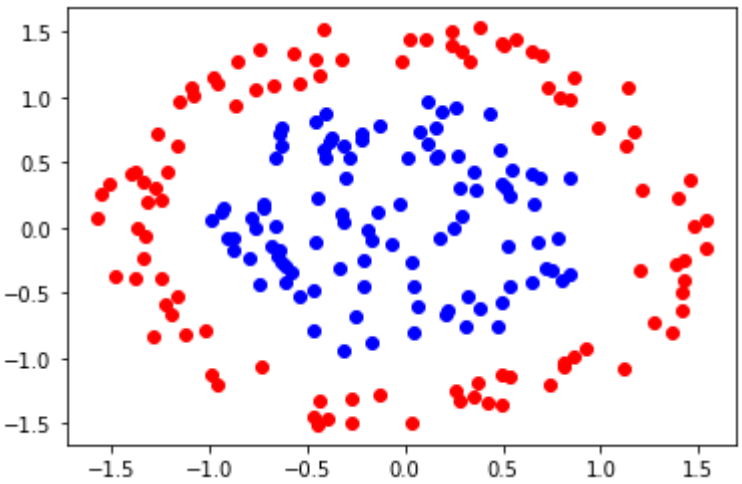
در مرحله 7 ، ما تنها زمانی وزن ها را آپدیت می کنیم که خطا رخ داده باشد و منطق آپدیت کردن وزن ها نیز به این صورت است که در نهایت خروجی شبکه به برچسب اصلی داده نزدیک شود.

زمانی که t = 1 است و خطا رخ می دهد ، یعنی خروجی همه نود های لایه مخفی 1- بوده (چرا که از منطق OR در لایه خروجی استفاده می شود) و حداقل یکی از نود ها باید خروجی 1 داشته باشد. لذا باید وزن نود هایی آپدیت شود که net input آنها به 0 نزدیک تر است.

همچنین اگر t= -1 و خطا رخ داده ، یعنی حداقل یکی از نود ها خروجی 1 داشته در صورتی که همه نود ها باید خروجی 1- داشته باشند. لذا وزن همه نود هایی که خروجی net input آنها مثبت است را آپدیت می کنیم.

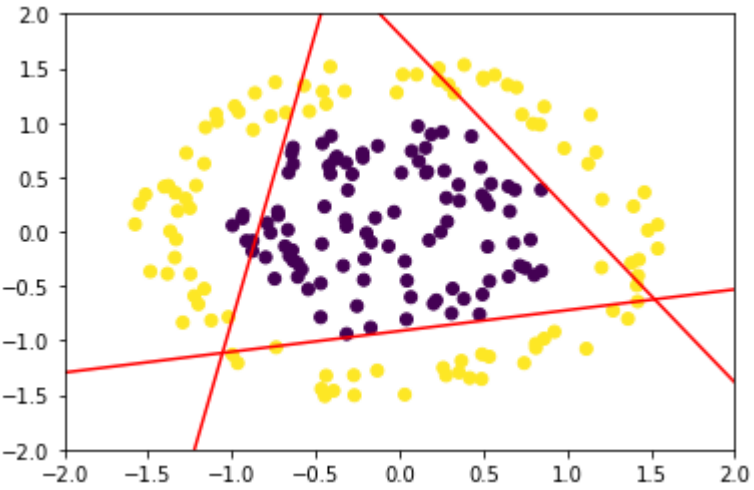
همچنین در الگوریتم فوق در لاید آخر از منطق OR استفاده کردیم ، ولی با توجه به نوع مسئله می توان از سایر تابع های منطقی نظیر AND و یا حتی تابع majority نیز استفاده کرد.

ب)



شکل 9. نمودار پراکندگی داده های MadaLine

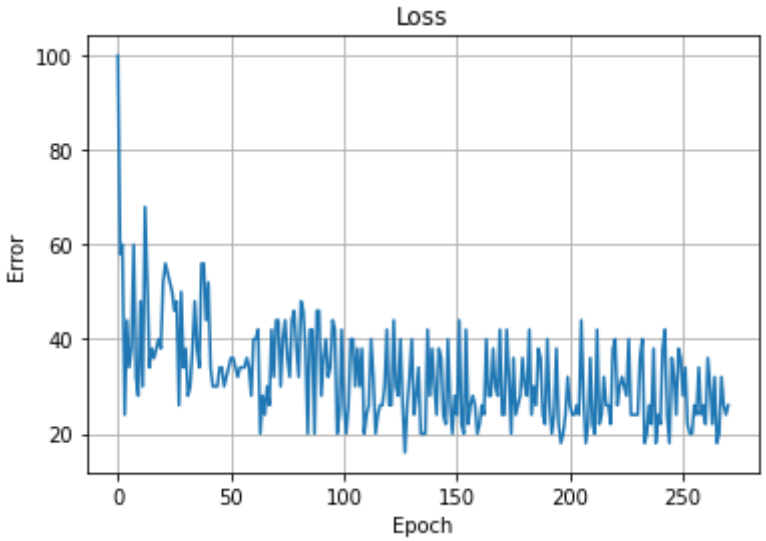
ج)



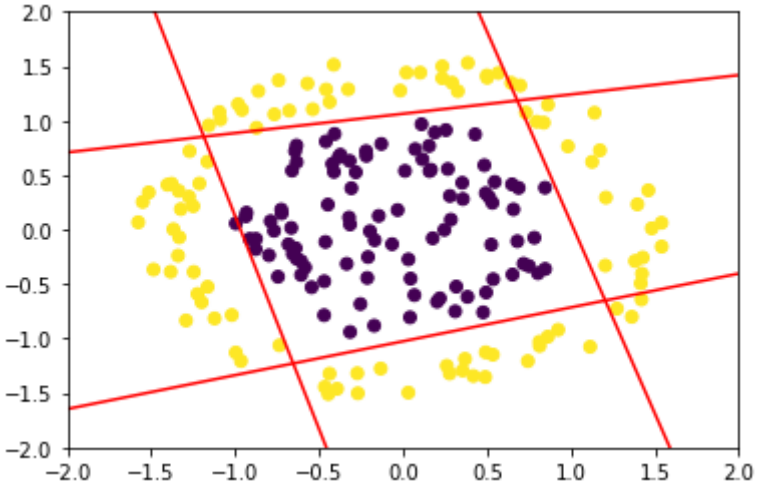
شکل 10. مرز جداکننده با MadaLine با 3 نورون

```
Total iterations with 3 lines: 271
Accuracy of prediction is: 0.875
```

شکل 11. تعداد ایپاک و دقت مدل MadaLine با 3 نورون



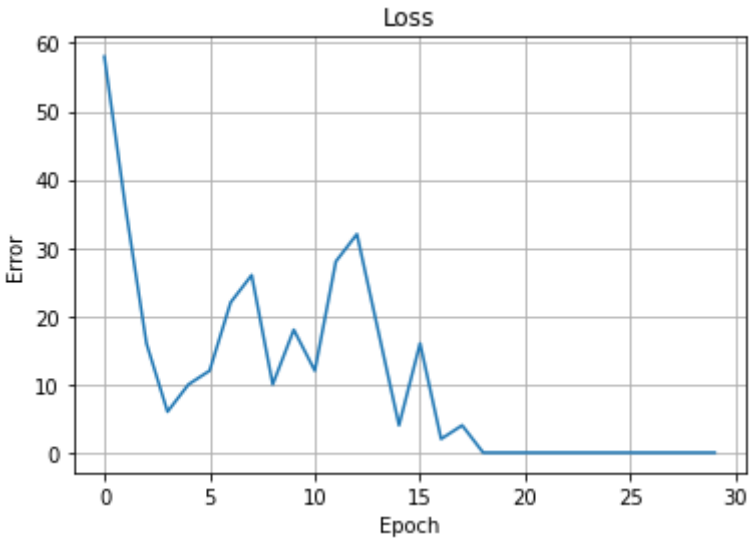
شکل 12. خطای مدل MadaLine با 3 نورون به ازای هر تکرار



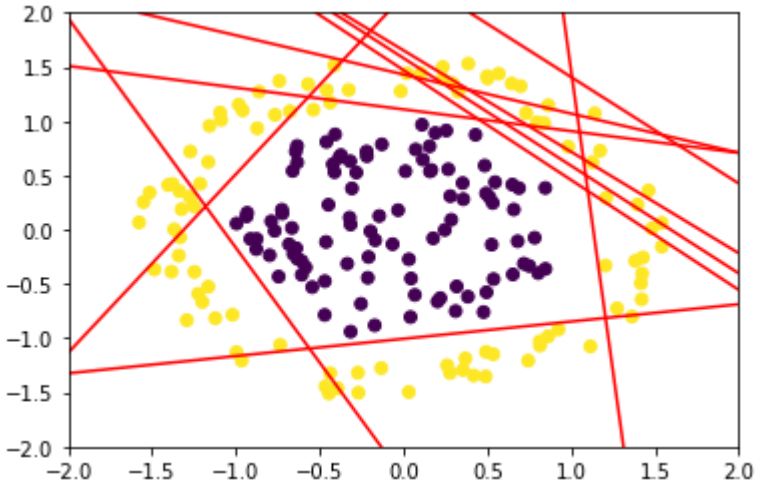
شکل 13. مرز جداکننده MadaLine با 4 نورون

```
Total iterations with 4 lines: 30
Accuracy of prediction is: 0.975
```

شکل 14. تعداد ایپاک و دقت مدل MadaLine با 4 نورون



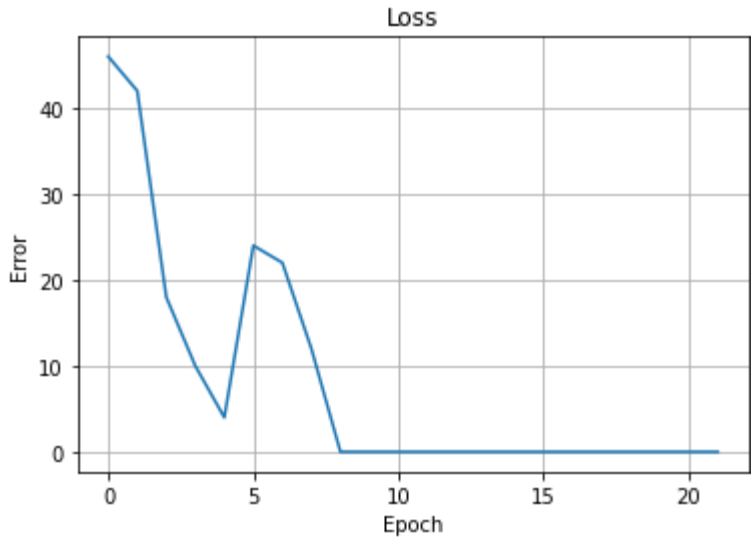
شکل 15. خطای مدل MadaLine با 4 نورون به ازای هر تکرار



شکل 16. مرز جداکننده MadaLine با 10 نورون

```
Total iterations with 10 lines: 22
Accuracy of prediction is: 0.95
```

شکل 17. تعداد ایپاک و دقت مدل MadaLine با 10 نورون



شکل 18. خطای مدل MadaLine با 10 نورون به ازای هر تکرار

د)

جدول 1. مقایسه دقت و تعداد ایپاک شبکه MadaLine با تعداد نورون متفاوت

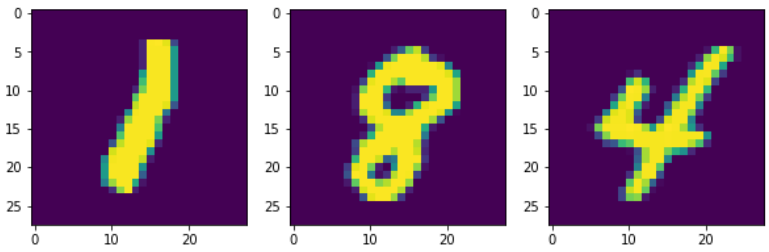
	دقت	تعداد ایپاک
شبکه اول (n=3)	0.87	271
شبکه دوم (n=4)	0.97	30
شبکه سوم (n=10)	0.95	22

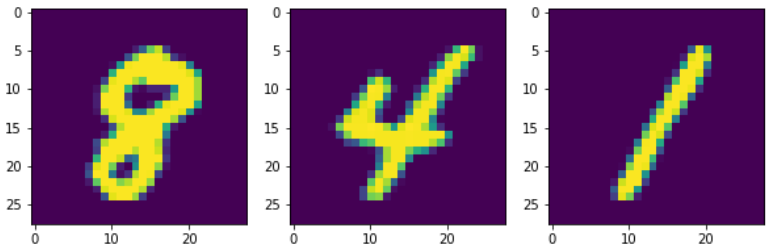
با مقایسه نتایج بالا (جدول 1) می توان دید که با افزایش تعداد نورون ها در شبکه MadaLine ، تعداد تکرار های لازم برای رسیدن به وزن های مطلوب کاهش می یابد. دلیل آن این است که افزایش تعداد نورون ها ، تعداد خطوط مرز بندی را افزایش می دهد و این امر آزادی عمل برای مرز بندی پیچیده تر دو کلاس را بیشتر می کند . همچنین زمانی که تعداد نورون ها 10 است ، تنها 5 خط در جداسازی دوکلاس نقش دارند و وزن های سایر خطوط به علت اینکه خطای رو داده های train صفر می شود، دیگر آپدیت نمی شوند و سایر خطوط redundant هستند. همچنین در جدول 1، دقت سه مدل بر روی داده های test نشان داده شده است. با توجه به شکل مرز بین دو کلاس، سه خط برای جداسازی آنها کافی نیست و به همین علت دقت روی داده های test به ازای 3 خط به نسبت دو مدل دیگر کمتر است و جداسازی کیفیت کمتری دارد. شرط توقف الگوریتم این است که اگر در 10 تکرار متوالی وزن ها کمتر از 0.0001 تغییر کند ، الگوریتم متوقف شود.

پاسخ ۳ – Auto-Encoders for classification

۳-۱. آشنایی و کار با دیتاست (پیش پردازش)

شکل 19. نمودار تعداد داده ها به ازای هر گروه برای داده های آموزش





شکل 20. نمونه های رندوم از دیتاست MNIST

```
# normalize
X_train = X_train.astype('float32') / 255
X_test = X_test.astype('float32') / 255

#Converting the labels into a one-hot vector
Y_train = to_categorical(Y_train)
Y_test = to_categorical(Y_test)
```

تصویر بالا نمونه کد استفاده شده برای normalize کردن داده های ورودی شبکه را نشان می دهد که داده ها را بین 0 و 1 ، scale می کند. همچنین برای برچسب ها نیز از روش one-hot استفاده کردیم.

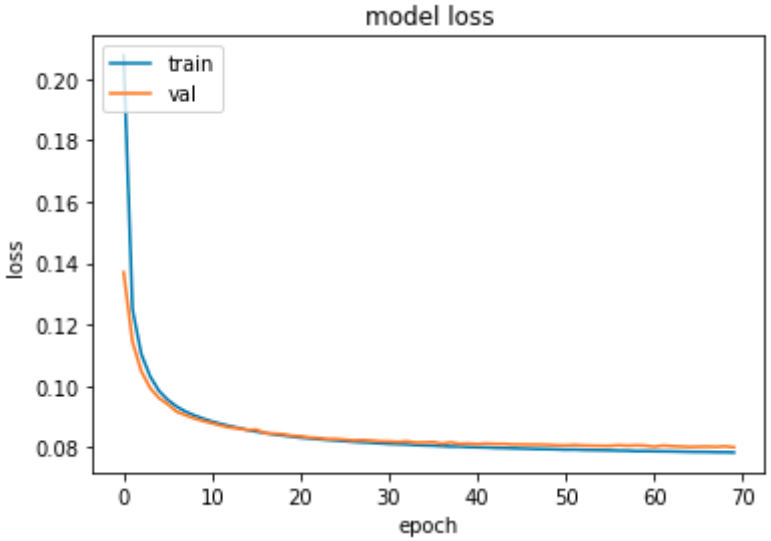
۳-۲.شبکه Auto-Encoder

```
encoding_dim = 30
input_img = keras.Input(shape=(784,))

encoded = layers.Dense(500,activation='relu')(input_img)
encoded = layers.Dense(100,activation='relu')(encoded)
encoded = layers.Dense(encoding_dim,activation='relu')(encoded)

decoded = layers.Dense(100,activation='relu')(encoded)
decoded = layers.Dense(500,activation='relu')(decoded)
decoded = layers.Dense(784, activation='sigmoid')(decoded)

autoencoder = keras.Model(input_img, decoded)
```

شکل 21. نمونه کد طراحی شبکه Auto-Encoder

همانطور که در شکل 21 نمایش داده شده است، شبکه Auto-Encoder طراحی شده دارای یک لایه input با 784 واحد است (تصاویر ورودی با ابعاد 28*28 به وکتور با ابعاد 1*784 تبدیل شده اند). بعد از لایه ورودی سه لایه مخفی به ترتیب با 500 ، 100 و 30 واحد قرار می گیرد که قسمت Encoder شبکه را تشکیل می دهد. برای هر سه لایه مخفی Encoder از تابع فعالساز relu استفاده شده است.

در ادامه دو لایه مخفی دیگر به ترتیب با 100 و 500 نورون و با تابع فعالساز relu قرار گرفته است که به همراه لایه خروجی با 784 نورون و با تابع فعالساز sigmoid ، همگی قسمت Decoder شبکه را تشکیل می دهد.



شکل 22. نمودار loss و validation loss برای Auto-Encoder

نمودار شکل 22، میزان loss را برای داده های test و train نشان می دهد. این نمودار در طی 70 epoch و با batch_size = 256 ایجاد شده است. همانطور که در این نمودار دیده می شود ، loss در داده testو train با شیب نسبتا زیادی تا 20 epoch کاهش می یابدو سپس با شیب ملایم تری کم می شود و تقریبا در حوالی70 epoch ثابت می شود.

۳-۳.طبقه بندی

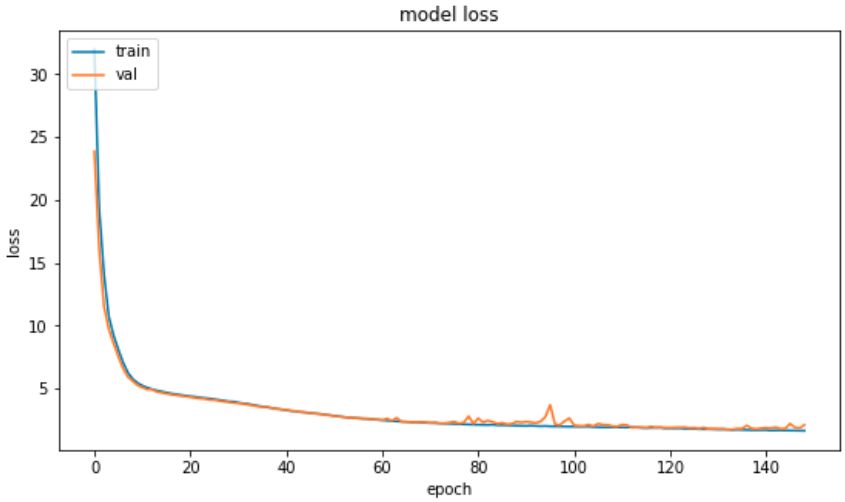
در این قسمت همانطور که خواسته شده ، قسمت Encoder از شبکه Auto-Encoder طراحی شده ، جدا و از خروجی آن (داده های کاهش بعد یافته) برای آموزش طبقه بند استفاده کردیم. برای آن که دید بهتری از دقت شبکه به دست آوریم ، ابتدا داده های x_test را به Encoder داده و کاهش بعد انجام دادیم و سپس داده های کاهش بعد یافته را به Decoder داده و تصاویر را reconstruct کردیم. نتیجه حاصل را می توان در شکل 23 مشاهده کرد (تصاویر ردیف اول ، تصاویر اصلی هستند.)

شکل 23. مقایسه تصاویر اصلی دیتاست با تصاویر reconstruct شده توسط شبکه

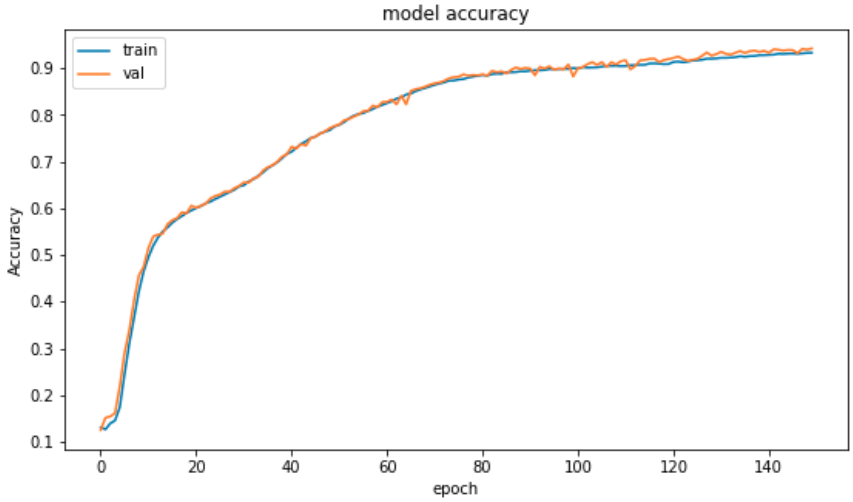
```
classifier = Sequential()
classifier.add(layers.Dense(20, input_shape=(encoding_dim,), activation='relu'))
classifier.add(layers.Dense(15, activation='relu'))
classifier.add(layers.Dense(10, activation='sigmoid'))
```

شکل 24. کد طراحی طبقه بند با دو لایه مخفی

در ادامه یک طبقه بند ساده با دولایه مخفی مشابه آنچه در شکل 24 نمایش داده شده است ، طراحی کرده و داده های کاهش بعد داده شده توسط Encoder را به عنوان داده های ورودی این طبقه بند استفاده کردیم.

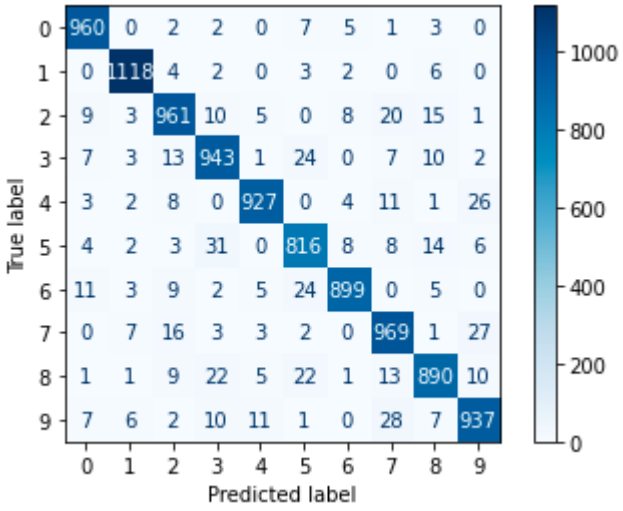


شکل 25. نمودار loss و validation loss برای طبقه بند



شکل 26. نمودار Accuracy و Validation Accuracy برای طبقه بند

طبقه بند 150 epoch تکرار شد و در نهایت نتیجه بر روی داده های test و train در دو شکل 25,26 نمایش داده شده است.برای این طبقه بند ، بر روی داده های test به دقت 94% و برای داده های train به دقت 93% رسیدیم . اختلاف کم بین دقت داده های test , train نشان از آن دارد که مدل ما بر روی داده های train، over fit نشده است. تابع loss نیز برای داده های test و train به ترتیب مقادیر 2.1 و1.63 دارد.



شکل 27. نمودار Confusion matrix برای طبقه بند

شکل 27 ، نمودار confusion matrix را برای طبقه بند طراحی شده نشان می دهد. همانطور که مشخص است مقادیر روی قطر اصلی به نسبت سایر بخش ها بیشتر است ، که این نشان از عملکرد خوب مدل در طبقه بندی است. (مدل اکثر کلاس ها را درست پیش بینی کرده است) از بین کلاس های این دیتاست (اعداد 0 تا 9)، مدل بهترین عملکرد را در خصوص تشخیص عدد 1 داشته است (سطر دوم ستون دوم) و به نسبت سایر کلاس ها ، در تشخیص عدد 5 ضعیف تر عمل کرده است. نکته جالبی که در این نمودار در برخی از ستون ها مشهود است این است که در برخی موارد به علت شباهت ظاهری دو عدد به هم مدل در تشخیص کلاس درست دچار مشکل شده است. مثلا اعداد (7,9) و (5,8,3) با هم و یا عدد 6 با5 و 2 با 7 و8 ، اشتباه گرفته شده اند. این مورد را می توان در خصوص عدم شباهت اعداد نیز بررسی کرد . مثلا عدد 6 و 7 هیچگاه با هم اشتباه گرفته نشده اند.

پاسخ ۴ – Multi-Layer Perceptron

۴-۱. آشنایی و کار با دیتاست (پیش پردازش)

I .فراخوانی فانکشن ()info

ابتدا با استفاده از کتابخانه pandas فایل csv را میخوانیم. پس از فراخوانی فانکشن ()info در pandas نتیجه به شکل زیر خواهد بود.

```
[6]   1 df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205 entries, 0 to 204
Data columns (total 26 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   car_ID            205 non-null    int64
 1   symboling         205 non-null    int64
 2   CarName           205 non-null    object
 3   fueltype          205 non-null    object
 4   aspiration        205 non-null    object
 5   doornumber        205 non-null    object
 6   carbody           205 non-null    object
 7   drivewheel        205 non-null    object
 8   enginelocation    205 non-null    object
 9   wheelbase         205 non-null    float64
 10  carlength         205 non-null    float64
 11  carwidth          205 non-null    float64
 12  carheight         205 non-null    float64
 13  curbweight        205 non-null    int64
 14  enginetype        205 non-null    object
 15  cylindernumber    205 non-null    object
 16  enginesize        205 non-null    int64
 17  fuelsystem        205 non-null    object
 18  boreratio         205 non-null    float64
 19  stroke            205 non-null    float64
 20  compressionratio  205 non-null    float64
 21  horsepower        205 non-null    int64
 22  peakrpm           205 non-null    int64
 23  citympg           205 non-null    int64
 24  highwaympg        205 non-null    int64
 25  price             205 non-null    float64
dtypes: float64(8), int64(8), object(10)
memory usage: 41.8+ KB
```

شکل 28. خروجی ()df.info از کتابخانه pandas

II .پیدا کردن Nanها در هر ستون

همان طور که از اطلاعات خروجی فانکشن()info در تصویر بالا مشخص است، تعداد عناصری که nan هستند برای هر ستون مشخص شده است. هیچ کدام از ستون‌ها در این دیتاست مقدار Nan نداشتند.

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

```
car_ID              0
symboling           0
CarName             0
fueltype            0
aspiration          0
doornumber          0
carbody             0
drivewheel          0
enginelocation      0
wheelbase           0
carlength           0
carwidth            0
carheight           0
curbweight          0
enginetype          0
cylindernumber      0
enginesize          0
fuelsystem          0
boreratio           0
stroke              0
compressionratio    0
horsepower          0
peakrpm             0
citympg             0
highwaympg          0
price               0
dtype: int64
```

شکل 29. تعداد nan ها در هر ستون

III. ساخت ستون carcompany

از ستون CarName بخش اول هر string را جدا می‌کنیم. بخش اول هر درایه نام شرکت است. سپس ستون Comapanyname را تشکیل میدهیم. پس از مشخص شدن نام شرکت، با کمک فانکشن drop در pandas ستون‌های Car_ID و CarName و symbolling را از دیتاست حذف میکنیم. همچنین نام شرکت‌هایی که اشتباه تایپ شده را ادیت میکنم. در نهایت جدول companyName به شکل زیر خلاصه می‌شود.

```
1 df["companyName"].value_counts()
```

```
toyota         32
nissan         18
mazda          17
mitsubishi     13
honda          13
subaru         12
peugeot        11
volvo          11
volkswagen     10
dodge           9
buick           8
bmw             8
audi            7
plymouth        7
saab            6
porsche         5
isuzu           4
jaguar          3
chevrolet       3
alfa-romero     3
renault         2
vw              2
mercury         1
Name: companyName, dtype: int64
```

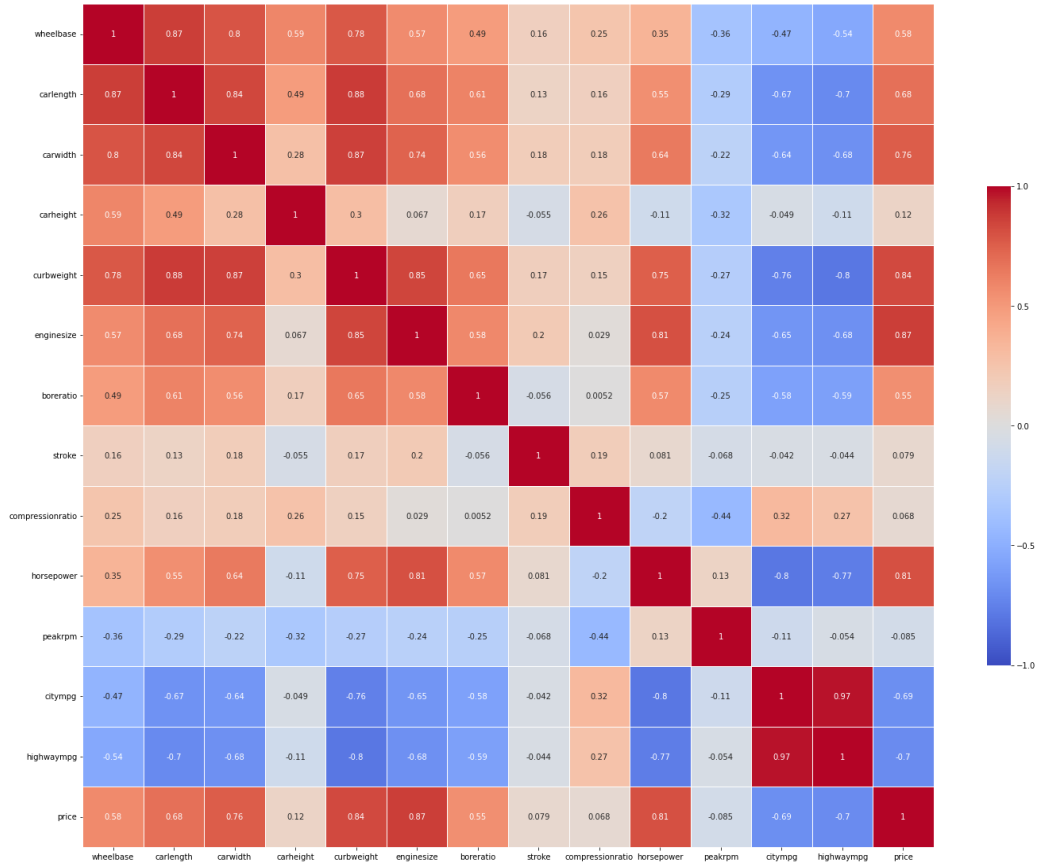
شکل 30. ستون companyName پس از صلاح نام برخی شرکت‌ها

IV . تبدیل متغیر های categorical به عددی

با استفاده از فانکشن get_dummies() از pandas ستون‌هایی که دارای مقادیر categorical هستند را به onehot تبدیل میکنیم و در ادامه به جای کار با مقادیر categorical با مقادیر عددی کار خواهیم کرد. پس از تبدیل مقادیر categorical به مقادیر عددی، تعداد ستون‌های (ویژگی‌ها) دیتاست به 75 ویژگی افزایش می‌یابد. به عنوان مثال ستون نوع سوخت به دو ستون گازی یا دیزلی تبدیل میشود که اگر خودرو گازی باشد ستون گاز 1 می‌شود اگر دیزلی باشد ستون دیزل 1 می‌شود.

V. رسم ماتریس Correlation

ابتدا correlation بین ستونهای دیتاست را با فانکشن corr() حساب می‌کنیم و سپس ماتریس correlation را برای دیتاست رسم میکنیم. به دلیل اینکه امکان نمایش ماتریس برای 76 ویژگی در اینجا نیست، شکل نمایش داده شده در زیر، به ازای ویژگی عددی قبل از اعمال onehot encoding، ماتریس correlation رسم شده است.

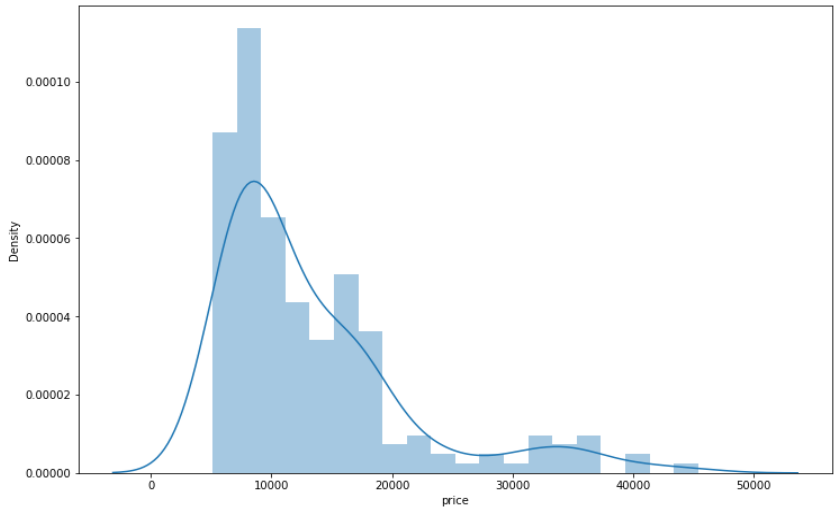


شکل 31. ماتریس correlation مجموعه داده ها

همانطور که در ماتریس correlation مشخص است ویژگی قیمت با ویژگی enginesize بیشترین همبستگی را دارد. مقدار همبستگی بین price و enginesize مقدار 0.87 است.

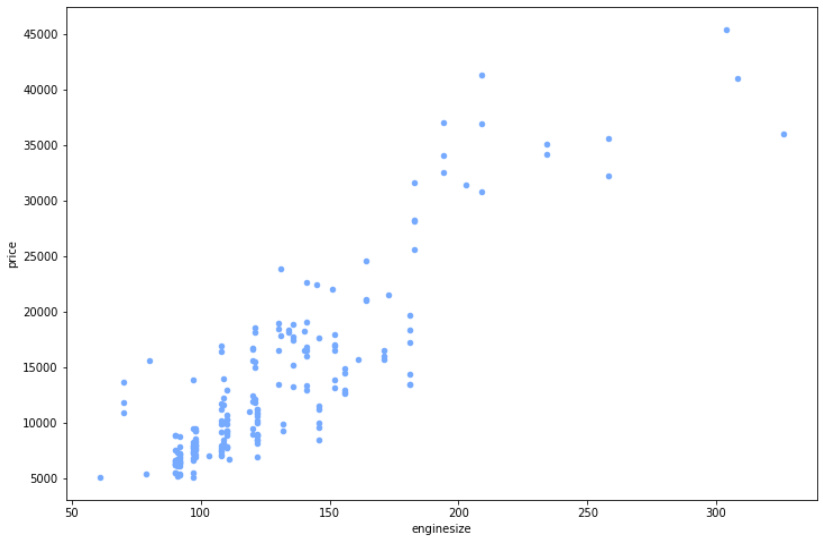
.VI

رسم توزیع قیمت و نمودار قیمت بر حسب enginesize در زیر آورده شده است.



شکل 32. نمودار توزیع قیمت

نمودار قیمت بر حسب enginesize که بیشترین وابستگی را با قیمت را دارد در زیر آورده شده است.



شکل 33. نمودار قیمت برحسب اندازه موتور

VII.

داده‌ها را به دو دسته 85٪ و 15٪ تقسیم می‌کنیم. برای این کار از کتابخانه sklearn و فانکشن train_test_split() استفاده می‌کنیم.

```
from sklearn.model_selection import train_test_split
```

```
X = df.loc[:,~df.columns.isin(['price'])]
y = df[["price"]]
Xtrain, Xtest, ytrain,  ytest =  train_test_split(X,y, test_size= 0.15, random_state=2023)
Xtrain.shape, Xtest.shape, ytrain.shape,  ytest.shape
ytrain, ytest = np.array(ytrain),  np.array(ytest)
```

شکل 34. تقسیم داده ها به train وtest

VIII.

به دلیل اینکه ستون‌های ویژگی‌ها ممکن است در واحدهای[1] مختلف و در رنج های مختلفی قرار بگیرند scaling نقش اساسی در پیش‌پردازش دارد. برای scaling باید به این نکته توجه شود داده‌های تست در scaling قرار نگیرند زیرا داده‌های تست را به عنوان داده‌هایی در نظر می‌گیریم که توسط مدل دیده نشده است. بنابراین در زمانی که داده‌ها scale می‌شوند داده‌های تست فقط با پارامترهایی که در دیتای آموزش

[1] Units

محاسبه شده‌اند scale می‌شوند. برای scale دیتاست از فانکشن MinMaxscale در sklearn استفاده کردیم.

```
from sklearn.preprocessing import MinMaxScaler
```

```
xscaler = MinMaxScaler()
Xtrain = xscaler.fit_transform(Xtrain)
Xtest = xscaler.transform(Xtest)
```

شکل 35. اسکیل کردن داده های train وtest

۲-۴. Multi-Layer Perceptron

ما سه مدل شبکه عصبی ساختیم. اولین مدل را با یک لایه مخفی، دومین مدل را با 2 لایه و سومین مدل با سه لایه مخفی ساختیم.

سه مدل ابتدایی با تعداد لایه های مخفی 1 و 2 و3 با تابع هزینه MSELoss را و بهینه ساز Adam آموزش دادیم. برای مقایسه سه شبکه از معیار $R^2\ Score$ استفاده کردیم. این معیار در ادامه گزارش توضیح داده شده است.

I. معرفی توابع هزینه و بهینه‌سازها

تابع هزینه MSE

Mean Square Error مقدار اختلاف مقدار پیشبینی شده توسط مدل و مقدار لیبل محاسبه می‌کند و به توان 2 میرساند و بر روی کل دیتاست مقادیر را میانگین می‌گیرد. به دلیل اینکه این تابع هزینه مقدار خطا را به توان 2 میرساند، زمانی که outliers زیاد باشد استفاده از این تابع هزینه مناسب نیست. فرمول محاسبه این هزینه در زیر آورده شده است.

$$MSE = \frac{1}{n}\sum(y_i - y_i')^2$$

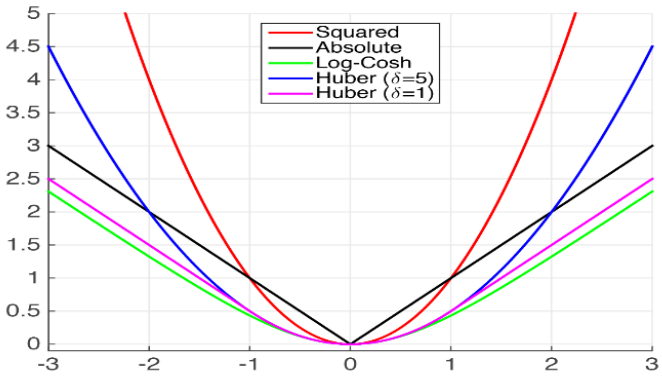
تابع هزینه Huber

این تابع هزینه، مزایای توابع L1 smooth و MSE را همزمان داراست. زمانی که هزینه نزدیک صفر است رفتار تابع هزینه مانند MSE است و زمانی که خطا زیاد می‌شود این تابع مشابه MAE[1] رفتار می‌کند.زمانی که اندازه خطا بالا میرود به دلیل ترم توان 2 در MSE، MSE به outliers حساس است اما این تابع هزینه حساسیت کمتری به outlier خواهد داشت. پارامتر delta در این تابع هزینه، مشخص می‌کند چقدر به outlierها حساس باشد.

[1] Mean Absolute Error

$$l_n = \begin{cases} 0.5(x_n - y_n)^2, & \text{if } |x_n - y_n| < delta \\ delta * (|x_n - y_n| - 0.5 * delta), & \text{otherwise} \end{cases}$$

در تصویر زیر توابع هزینه مقایسه شده اند:



شکل 36. مقایسه تابع هزینه MSE و MAE و Huber

روش‌های بهینه سازی RMSprop و Adam

گرادیان نزولی یکی از روش‌های بهینه‌سازی در شبکه‌های عصبی محسوب می‌شود. روش گرادیان نزولی یل توجه به گرادیان و مقدار نرخ یادگیری می‌توان تابع هزینه convex را به مینمم گلوبال رساند. اگر مقدار نرخ یادگیری زیاد ممکن است در نزدیک مینیمم گلوبال قرار بگیریم و به مینمم نرسیم و مقدار کم نرخ یادگیری سرعت همگرایی را کاهش می‌دهد.

بهینه‌سازهای گرادیان نزولی با تکانه[1] سریع‌تر به مینیمم همگرا می‌شوند. روش‌های مبتنی بر تکانه به دلیل اینکه به جهت گرادیان توجه می‌کنند و هر جا لازم باشد در جهت خاصی گام بلندتری نسبت به جهت‌های دیگر برمیدارند. روش RMSProp روشی مبتی بر تکانه است. الگوریتم آپدیت وزن توسط این روش در زیر آورده شده است. β در فرمول زیر تکانه را مشخص میکند.

[1] Momentum

$$v_{dw} = \beta \cdot v_{dw} + (1 - \beta) \cdot dw^2$$

$$v_{db} = \beta \cdot v_{dw} + (1 - \beta) \cdot db^2$$

$$W = W - \alpha \cdot \frac{dw}{\sqrt{v_{dw}} + \epsilon}$$

$$b = b - \alpha \cdot \frac{db}{\sqrt{v_{db}} + \epsilon}$$

بهینه ساز Adam مانند بهینه‌ساز RMSProp از تکانه جهت افزایش سرعت حرکت در جهت رسیدن به مینیمم استفاده میکند اما تفاوت آن با RMSprop در این است که RMSprop از تکانه مرتبه دوم برای آپدیت استفاده می‌کند اما Adam از هر دو تکانه مرتبه اول و دوم برای آپدیت وزن‌ها و کاهش تابع هزینه استفاده می‌کند. فرمول آپدیت وزن‌ها در زیر آورده شده است:

$$For\ each\ Parameter\ w^j$$
$$(j\ subscript\ dropped\ for\ clarity)$$

$$\nu_t = \beta_1 * \nu_{t-1} - (1 - \beta_1) * g_t$$
$$s_t = \beta_2 * s_{t-1} - (1 - \beta_2) * g_t^2$$

$$\Delta\omega_t = -\eta \frac{\nu_t}{\sqrt{s_t + \epsilon}} * g_t$$

$$\omega_{t+1} = \omega_t + \Delta\omega_t$$

$\eta : Initial\ Learning\ rate$
$g_t : Gradient\ at\ time\ t\ along\ \omega^j$
$\nu_t : Exponential\ Average\ of\ gradients\ along\ \omega_j$
$s_t : Exponential\ Average\ of\ squares\ of\ gradients\ along\ \omega_j$
$\beta_1, \beta_2 : Hyperparameters$

II. معرفی معیار R^2 Score

معیار R^2 نسبت واریانس توضیح داده شده توسط مدل به کل واریانس را محاسبه می‌کند. این آماره نشان دهنده درصد واریانس متغیرهای وابسته است که متغیرهای مستقل به طور جمعی توضیح می دهند. یا به عبارت دیگر مقدار واریانس توضیح داده شده توسط مدل رگرسیون نسبت به کل واریانس است. رابطه محاسبه این آماره به شکل زیر است.

Sum Squared Regression Error

$$R^2 = 1 - \frac{SS_{Regression}}{SS_{Total}}$$

Sum Squared Total Error

Sum Over All The Data Points — Square The Result

$$SS_{Total} = \sum (y_i - \bar{y})^2$$

Sum Squared Total Error — Each Data Point — Mean Value

Sum Over All The Data Points — Square The Result

$$SS_{Regression} = \sum (y_i - y_{Regression})^2$$

Sum Squared Regression Error — Each Data Point — Regression Value

III. نتیجه شبکه های عصبی با تعداد لایه های مختلف

شبکه 1 لایه مخفی، تابع هزینه MSE و بهینه‌ساز Adam به شکل زیر طراحی شده است:

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1                [-1, 100]           7,500
            Linear-2                  [-1, 1]             101
================================================================
Total params: 7,601
Trainable params: 7,601
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.00
Params size (MB): 0.03
Estimated Total Size (MB): 0.03
```

شکل 37. تعداد پارامترهای مدل با یک لایه مخفی

نتایج این شبکه MLP به شکل زیر است. مقدار R^2 در شکل قابل مشاهده است:

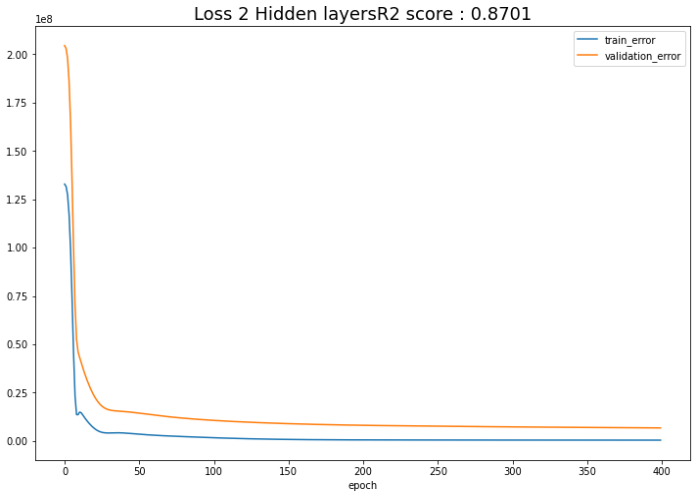
شکل 38. تابع هزینه مدل با 1 لایه مخفی

شبکه 2 لایه مخفی، تابع هزینه MSE و بهینه‌ساز Adam به شکل زیر طراحی شده است:

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1                [-1, 512]          38,400
            Linear-2                [-1, 100]          51,300
            Linear-3                  [-1, 1]             101
================================================================
Total params: 89,801
Trainable params: 89,801
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.00
Params size (MB): 0.34
Estimated Total Size (MB): 0.35
----------------------------------------------------------------
```

شکل 39. تعداد پارامترهای مدل با 2 لایه مخفی

نتایج این شبکه MLP به شکل زیر است. مقدار R^2 در شکل قابل مشاهده است:



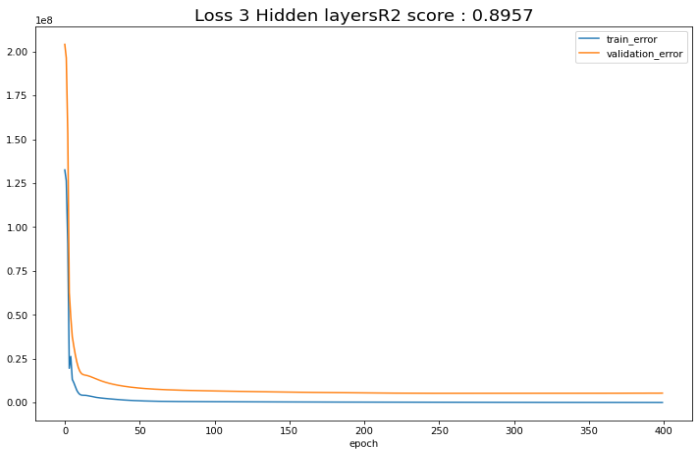
شکل 40. تابع هزینه مدل با 2 لایه مخفی

شبکه 3 لایه مخفی، تابع هزینه MSE و بهینه‌ساز Adam به شکل زیر طراحی شده است:

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1                  [-1, 512]          38,400
            Linear-2                  [-1, 256]         131,328
            Linear-3                   [-1, 64]          16,448
            Linear-4                    [-1, 1]              65
================================================================
Total params: 186,241
Trainable params: 186,241
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.01
Params size (MB): 0.71
Estimated Total Size (MB): 0.72
----------------------------------------------------------------
```

شکل 41. تعداد پارامترهای مدل با 3 لایه مخفی

نتایج این شبکه MLP به شکل زیر است. مقدار R^2 در شکل قابل مشاهده است:



شکل 42. تابع هزینه مدل با 3 لایه مخفی

IV. تحلیل نتایج و مقایسه 3 شبکه عصبی با تعداد لایه های مخفی مختلف

همان طور که در نتایج بالا دیده شد مقدار R^2 که بیانگر واریانس توضیح داده شده مدل به کل واریانس است، با افزایش تعداد لایه‌ها افزایش یافت و هر چه مقدار R^2 بیشتر باشد مدل رگرسیون، تخمین و پیشبینی بهتری دارد. مقادیر R^2 برای هر یک از شبکۀها در جدول زیر آورده شده است. تمام شبکه‌ها با بهینه‌ساز Adam و تابع هزینۀ MSE آموزش داده شده اند. با توجه به جدول زیر مدل با 3 لایه مخفی بهترین پیش‌بینی قیمت را داراست.

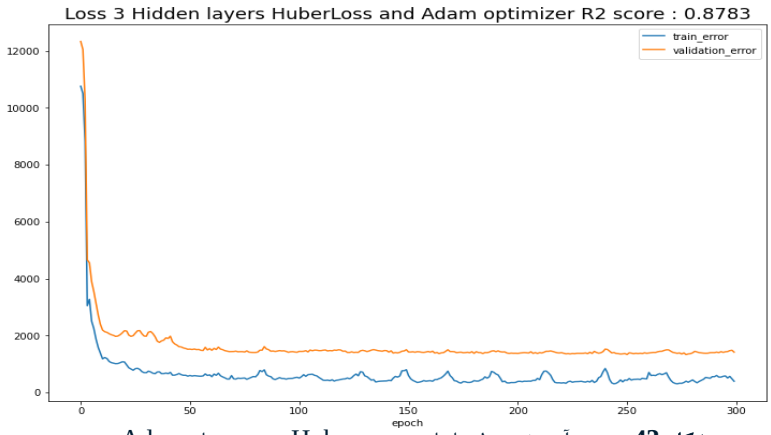
جدول 2. تعداد پارامترها و مقایسه R2 برای شبکه‌ها با تعداد لایه‌های مختلف

	تعداد پارامترهای آموزشی	R^2
شبکه با 1 لایه مخفی	*7601*	*0.7073*
شبکه با 2 لایه مخفی	*89601*	*0.8701*
شبکه با 3 لایه مخفی	*186241*	*0.8957*

V. بررسی تغییرات تابع هزینه و بهینه سازها بر روی مدل 3 لایه

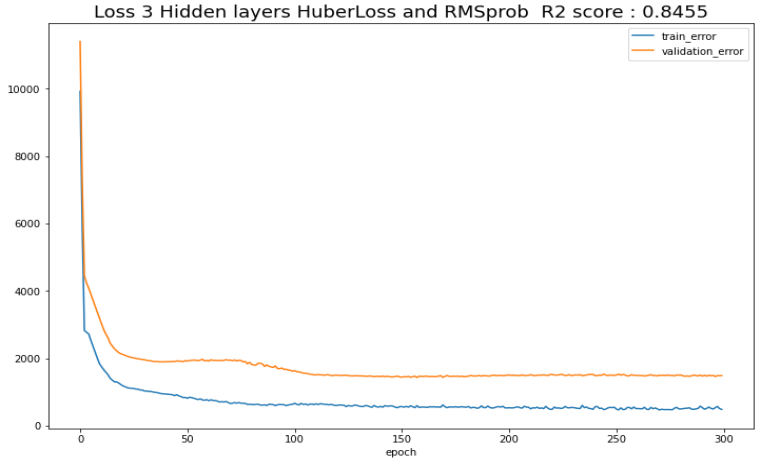
در این به بخش به ارائه نتایج شبکۀ عصبی با 3 لایه مخفی و مقایسه توابع هزینه و بهینه‌ساز های مختلف می‌پردازیم. توابع هزینه MSE و Huber و بهینه‌سازهای Adam و RMSprop که در بخش‌های قبل به تشریح آنها پرداختیم به کار گرفته‌شده اند تا بهترین مدل برای تخمین قیمت را بسازیم.

آموزش مدل با تابع هزینه Huber و بهینه ساز Adam



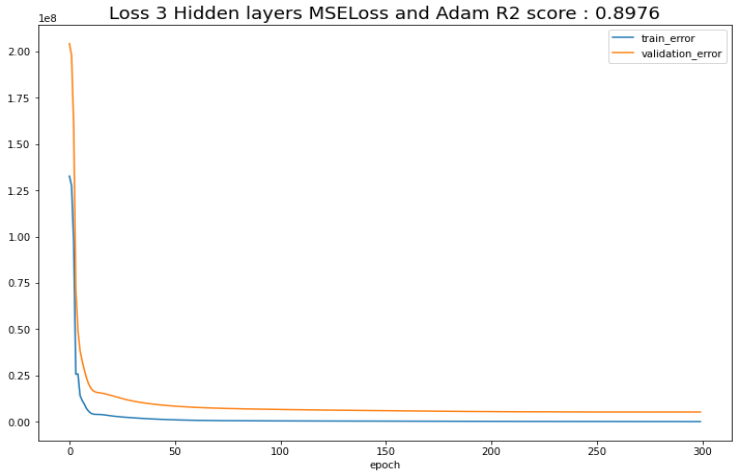
شکل 43. نتیجه آموزش مدل با تابع هزینه Huber و بهینه ساز Adam

آموزش مدل با تابع هزینه Huberو بهینه ساز RMSprop



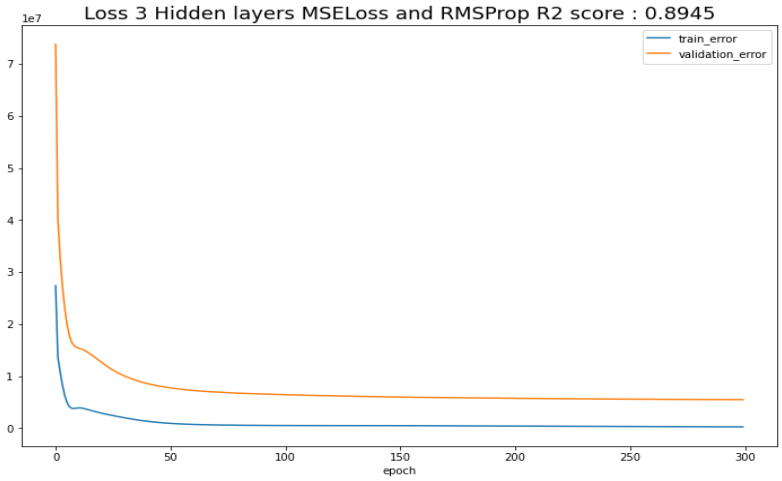
شکل 44. نتیجه آموزش مدل با تابع هزینه Huber و بهینه ساز RMSprob

آموزش مدل با تابع هزینه MSE و بهینه ساز Adam



شکل 45. نتیجه آموزش مدل با تابع هزینه MSEloss و بهینه ساز Adam

آموزش مدل با تابع هزینه MSE و بهینه ساز RMSprob



شکل 46. نتیجه آموزش مدل با تابع هزینه MSEloss و بهینه ساز RMSprob

VI. بررسی و تحلیل نتایج

همان طور که مشاهده شده برای شبکه عصبی MLP با سه لایه مخفی، 4 مدل آموزش داده شد. مدل های آموزش داده شده در تابع هزینه و بهینه‌ساز تفاوت داشتند. در جدول R^2 برای هر یک از حالت‌ها آورده شده است .

جدول 3. ارائه نتایج مدل با 3 لایه مخفی با توابع هزینه و بهینه‌سازهای مختلف

توابع هزینه و بهینه‌سازها	R^2
هزینه ***MSE*** و بهینه ساز ***Adam***	0.8976
هزینه ***MSE*** بهینه ساز ***RMSprop***	0.8945
هزینه ***Huber*** و بهینه‌ساز ***Adam***	0.8788
هزینه ***Huber*** و بهینه‌ساز ***RMSprop***	0.8455

با توجه به جدول بالا، تابع هزینه MSE با بهینه ساز Adam بهترین نتیجه را دارد و بیشترین واریانس توضیح داده شده توسط مدل نسبت به واریانس کل را دارد و به عبارتی بهترین پیشبینی برای قیمت خواهد داشت.

VII. تخمین 5 داده تست

در جدول زیر مقادیر داده‌های تست و پیشبینی قرار داده شده است و اختلاف آن حساب شده است. همچنین اختلاف بر اساس مقدار اصلی نرمالایز شده و درصد خطا نیز به دست آمده است.

جدول 4. نتایج پیشبینی مدل برای 5 داده تست

Y true (test)	Predict	Difference	Normalized Difference(%)
21485	21336.1816	148.8184	**0. 69%**
13845	11033.5361	2811.4639	**20.31%**
9639	11701.9346	-2062.9346	**21.40%**
10595	9553.1904	1041.8096	**9.83%**
6938	8048.4937	-1110.4937	**16.01%**

همان طور که مشخص است در بعضی از کیس‌ها حدود 20٪ خطا و در برخی کیس‌ها کمتراز 1٪ خطا داریم. زمانی که قیمت بسیار پایین است خطا ما بیشتر است. در کیس‌هایی که قیمت زیاد است خطای مدل ما بسیار کم است. برای مثال در قیمت حدود 21485 دلار خطای تخمین کمتر از 1٪ است. بهترین روش برای شناسایی نقاط ضعف مدل در برخی بازه‌های قیمت استفاده از روش Cross Validation است.