به نام خدا

دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

درس شبکه‌های عصبی و یادگیری عمیق

تمرین اول

نام و نام خانوادگی	عرفان باقری سولا – محمد قره حسنلو
شماره دانشجویی	810198461 - 810198361
تاریخ ارسال گزارش	1401.12.26

فهرست

جدول‌ها

پاسخ 1. شبکه عصبی Mcculloch-Pitts

۱-۱. ماشین متناهی قطعی (FDA)

الف)

جدول انتقال حالت متناسب با شبکه نورون ها را در ادامه مشاهده می کنیم.

جدول 1. جدول انتقال حالت

پذیرفتن	حالت بعدی		ورودی	حالت کنونی	
0	0	0	0	0	0
0	1	0	1	0	0
0	0	1	0	1	0
0	1	0	1	1	0
1	1	1	0	0	1
0	1	0	1	0	1
1	1	1	0	1	1
1	1	1	1	1	1

باید توجه کرد که در تمام نورون ها هر سه ستون سمت چپ جدول ورودی خواهد بود. هر یک از ستون های سمت راست نیز خروجی متناظر یک نورون خواهد بود.

ب)

برای راحتی کار ابتدا جدول صحت را برای خروجی هر نورون رسم می کنیم.

C \ AB	00	01	10	11
0	0	1	1	1
1	0	0	0	1

شکل 1: نمودار صحت خروجی اول

C \ AB	00	01	10	11
0	0	0	1	1
1	1	1	1	1

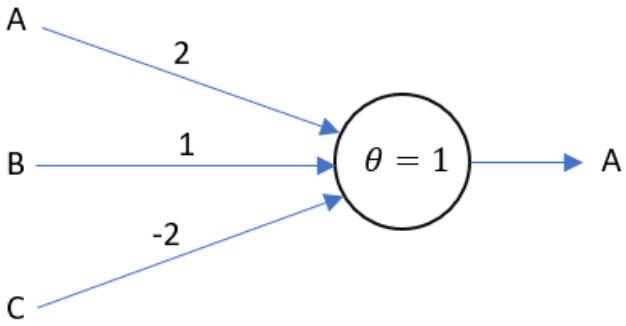
شکل 2: نمودار صحت خروجی دوم

C \ AB	00	01	10	11
0	0	0	1	1
1	0	0	0	1

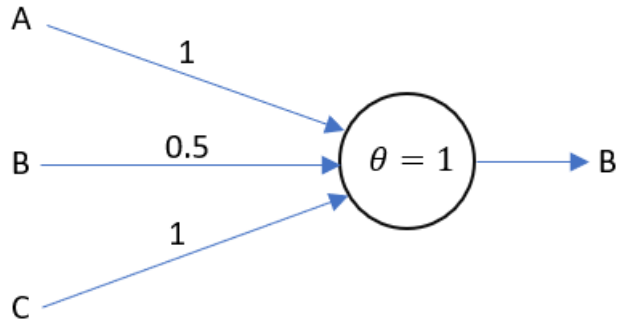
شکل 3: نمودار صحت خروجی سوم

با توجه به نمودار ها واضح است که خروجی ها به صورت خطی جدا پذیر هستند، پس انتظار داریم بتوانیم هر کدام را فقط با یک نورون پیاده سازی کنیم. همچنین هیچ دو خروجی دقیقا یکسان نیستند پس کمترین تعداد نورون ممکن برای پیاده سازی همان 3 تاست. در ادامه هر نورون را مشاهده خواهیم کرد.

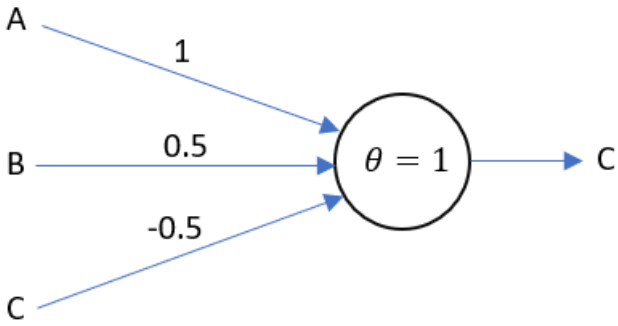
چون محدودتی روی وزن ها در صورت سوال طرح نشده است، وزن ها ر را طوری انتخاب می کنیم که threshold تمام نورون ها کمترین عدد صحیح مثبت یعنی 1 باشد.



شکل 4: نورون متناظر با خروجی اول

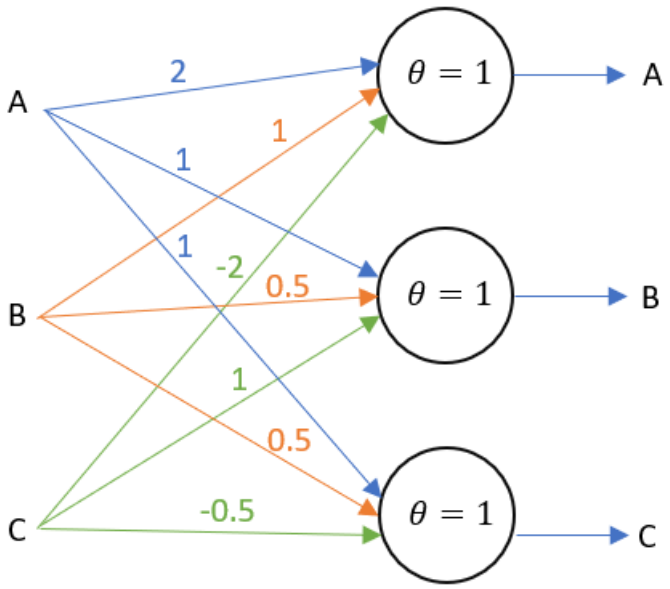


شکل 5: نورون متناظر با خروجی دوم



شکل 6: نورون متنظر با خروجی سوم

ج)



شکل 7: شبکه ادغام شده

د)

شبکه را به صورت یک کلاس در پایتون تعریف می کنیم به این صورت که وزن ها را به صورت یک ماتریس دو بعدی می گیرد و در آینده با دادن آرایه ای از ورودی ها به آن، با ضرب ماتریسی مناسب و مقایسه خروجی ها با threshold ، خروجی نهایی شبکه را برای هر ورودی برمی گرداند.

```
net.forward([[0, 0, 0],
             [0, 0, 1],
             [0, 1, 0],
             [0, 1, 1],
             [1, 0, 0],
             [1, 0, 1],
             [1, 1, 0],
             [1, 1, 1]])
```

```
array([[0, 0, 0],
       [0, 1, 0],
       [1, 0, 0],
       [0, 1, 0],
       [1, 1, 1],
       [0, 1, 0],
       [1, 1, 1],
       [1, 1, 1]])
```

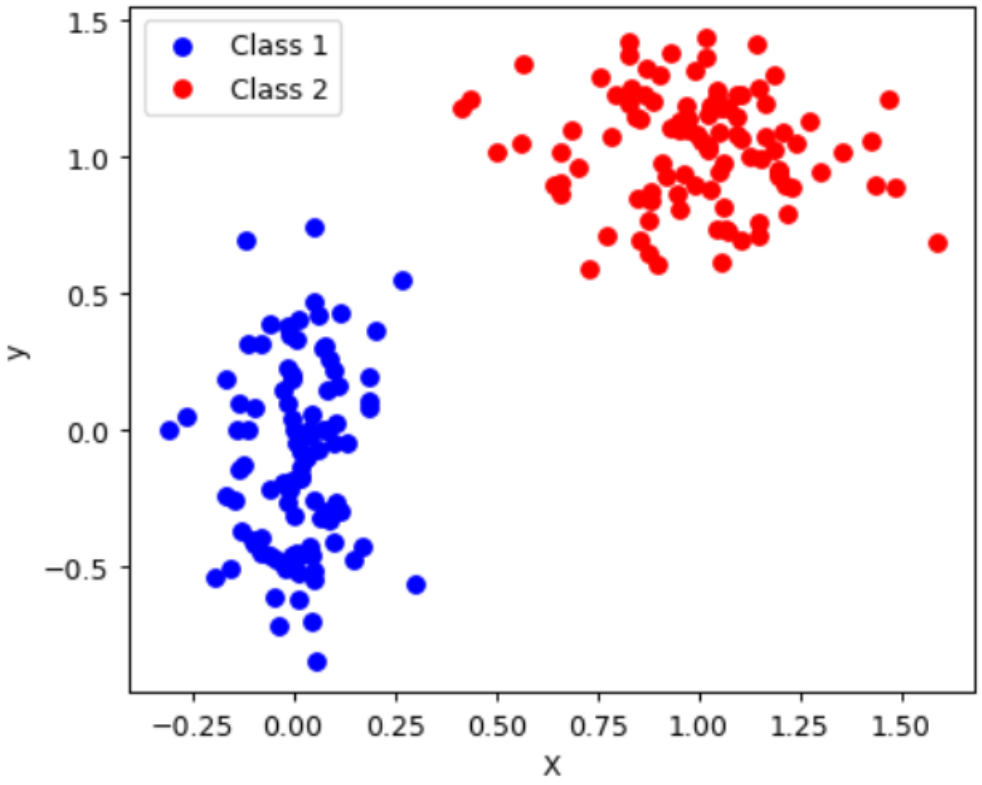
شکل 8: خروجی شبکه به ازای تمام حالت های ورودی

می بینیم که خروجی ها دقیقا با جدولی که در بخش الف داشتیم مطابقت دارد و شبکه به درستی کار می کند.

پاسخ ۲ – شبکه های Adaline و Madaline

۲-۱. Adaline

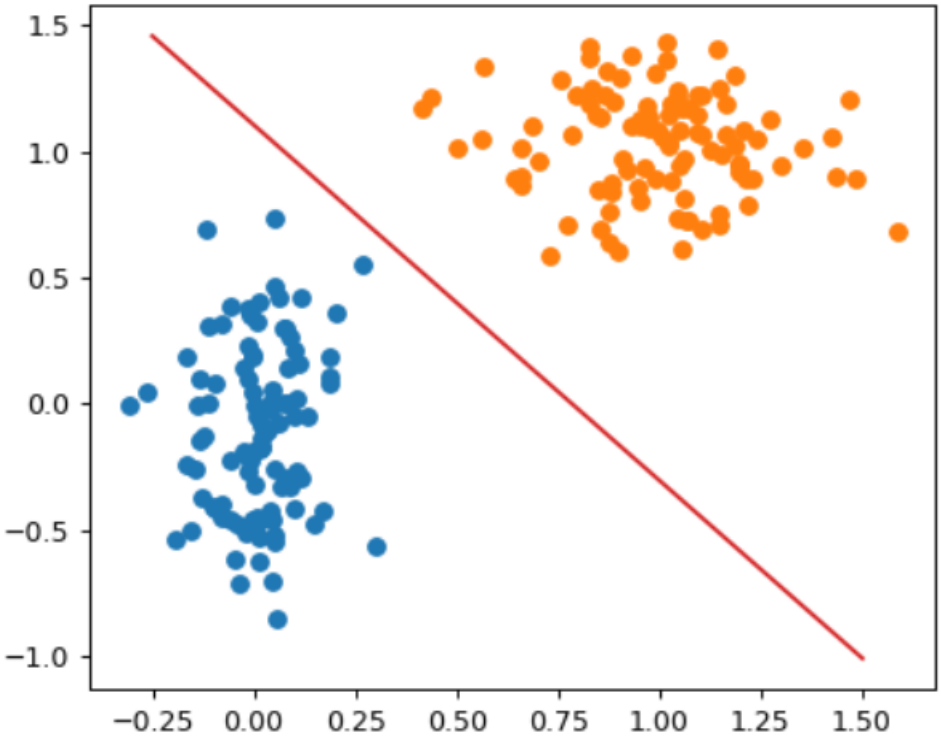
1- دو ماتریس میسازیم که در اولی(X1) ردیف اول دارای 100 داده که دارای میانگین 0 و انحراف معیار 0.1 بوده و ردیف دوم دارای 100 داده که دارای میانگین 0 و انحراف معیار 0.4 بوده میسازیم. برای دومی(X2) هم به همین صورت عمل میکنیم با این تفاوت که هر ردیف دارای میانگین 1 و انحراف معیار 0.2 است، میسازیم.



شکل 9: نمودار پراکندگی دسته داده های **X1** و **X2**

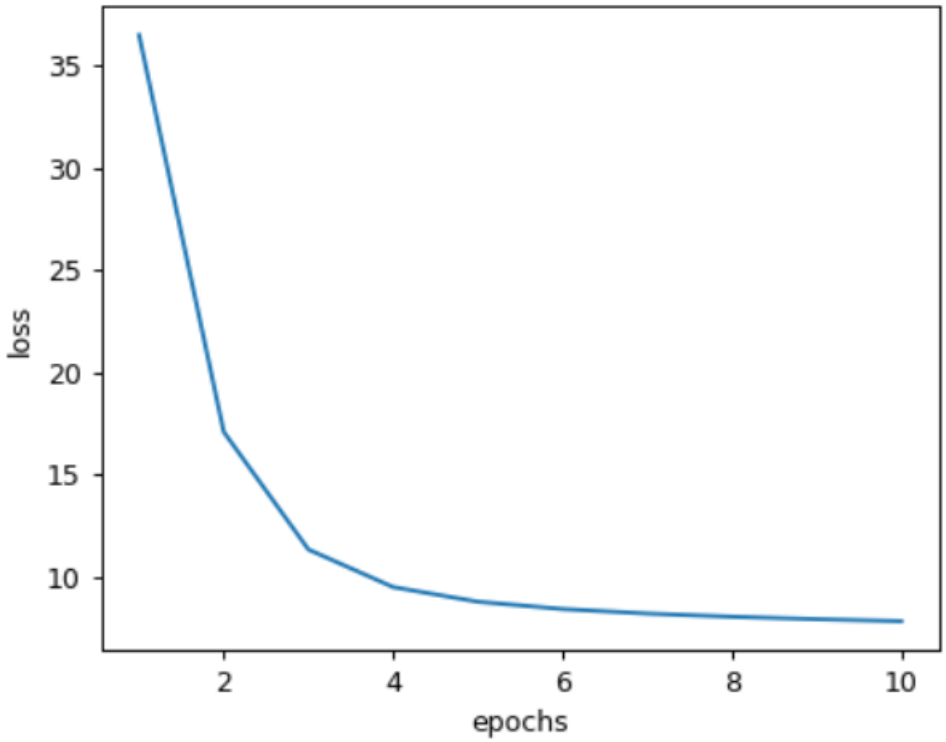
در شکل 9 ملاحظه میشود که دسته داده های تعریف شده را میتوان با یک خط adaline از هم جدا کرد و حتی با توجه به این شکل به دست امده میتوان loss را نزدیک به صفر رساند(بستگی به loss دارد که تعریف میکنیم.)

2- در اینجا یک مدل adaline که دارای دو ورودی و یک خروجی هست، تعریف میکنیم و داده ها به همان دلیلی که در قسمت قبل گفتیم(و همچنین به دلیل اینکه داده ها با هم تلاقی ندارند و دارای distrubtion نیست و داده های هر گروه به صورت دسته های مختلف در قسمت های مختلف فضا نیست) داده ها را با استفاده از adaline به خوبی میتوان از هم جدا کرد.



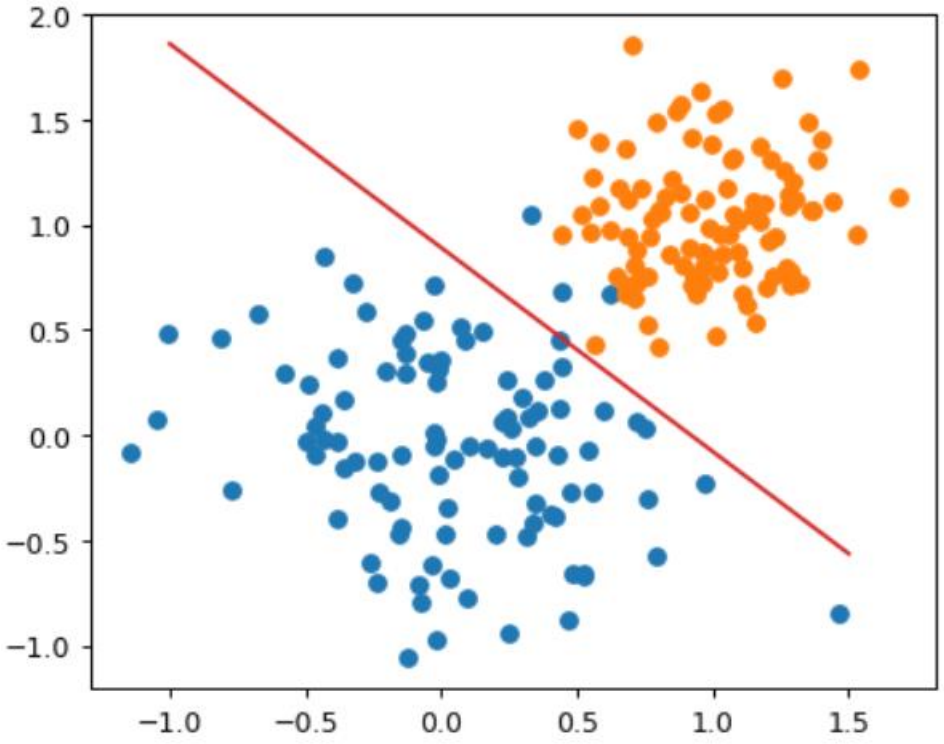
شکل 10: خط جداکننده حاصل از مدل **adaline** برای دیتاست های اول

خط حاصل از مدل را به صورت decision_line = -(clf.w[0] * t + clf.b) / clf.w[1] به دست می آید که شکل حاصل نشان میدهد این مدل به خوبی دو دسته داده ساخته شده را از هم جدا کرده است.



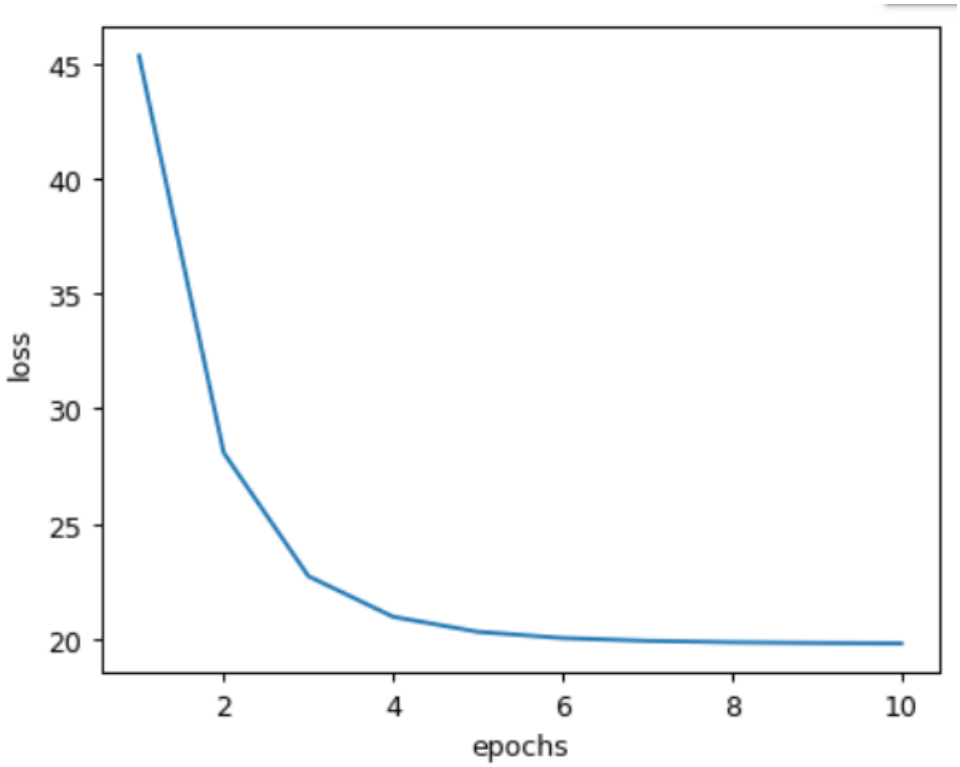
شکل 11: نمودار **loss** بر حسب هر **epoch** برای دیتاست های اول

3- نمودار توزیع حاصل از دو دسته داده جدید به صورت زیر میباشد:



شکل 12: خط جداکننده حاصل از مدل **adaline** برای دیتاست های دوم

همانطور که در شکا مشخص است، دو دیتاست پراکندگی بیشتری پیدا کرده و به هم نزدیک تر شده اند به طوری که با یک خط همه داده هارا نمیتوان از دیگری جدا کرد و نیاز به مدل هایی دارد که قوی تر از adaline عمل میکنند. یعنی این دو دیتاست با این مدل بایاس زیادی پیدا کرده و با مدل های قوی تر میتوان حتی همه داده هارا از هم جدا کرد.



شکل 13: نمودار **loss** بر حسب هر **epoch** برای دیتاست های دوم

در شکل بالا میتوان دید با تعداد epoch های برابر، دیتاست های دوم با loss بیشتری نسبت به دیتاست های اول به پایان رسیده اند که دلیل آن، قدرتمند نبودن adaline برای جداسازی همه داده ها از هر دو دسته است؛ چون در دسته دوم دیتاست ها قابل جداسازی با یک خط نیستند و با هم تلاقی دارند.

4- مقایسه های قسمت (د) در قسمت قبلی با توجه به نمودارها و نتایج مقایسه شده است.

Madaline .۲-۲

1- در MRI، وزن ها برای adaline های مخفی تنظیم میشود و باید یاد گرفته شود. وزن ها برای واحد خروجی فیکس هست و در فرایند آموزش نیازی به یادگیری ندارد و بر اساس نیاز مسئله آن را فیکس میکنیم. در حالی که در MRII یه متود برای تنظیم و یادگیری همه وزن ها در شبکه در نظر گرفته میشود.

Training Algorithm for MADALINE (MRI). The activation function for units Z_1, Z_2, and Y is

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

Step 0. Initialize weights:
Weights v_1 and v_2 and the bias b_3 are set as described;
small random values are usually used for ADALINE weights.
Set the learning rate α as in the ADALINE training algorithm (a small value).

Step 1. While stopping condition is false, do Steps 2–8.

Step 2. For each bipolar training pair, s:t, do Steps 3–7.

Step 3. Set activations of input units:

$$x_i = s_i.$$

Step 4. Compute net input to each hidden ADALINE unit:

$$z\_in_1 = b_1 + x_1 w_{11} + x_2 w_{21},$$

$$z\_in_2 = b_2 + x_1 w_{12} + x_2 w_{22}.$$

Step 5. Determine output of each hidden ADALINE unit:

$$z_1 = f(z\_in_1),$$

$$z_2 = f(z\_in_2).$$

Step 6. Determine output of net:

$$y\_in = b_3 + z_1 v_1 + z_2 v_2;$$

$$y = f(y\_in).$$

Step 7. Determine error and update weights:
If $t = y$, no weight updates are performed.
Otherwise:
If $t = 1$, then update weights on Z_J,
the unit whose net input is closest to 0,

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

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

If $t = -1$, then update weights on all units Z_k that have positive net input,

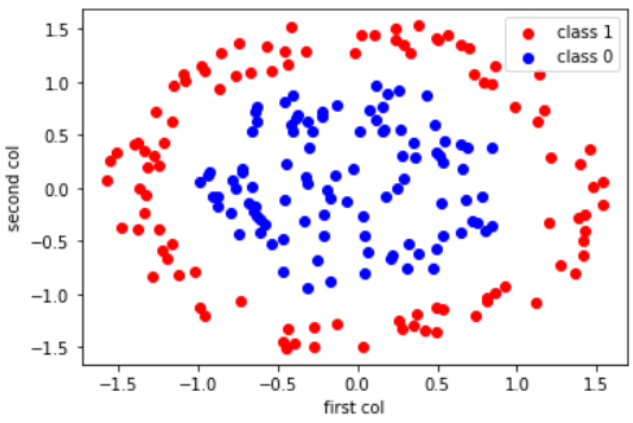
$$b_k(\text{new}) = b_k(\text{old}) + \alpha(-1 - z\_in_k),$$

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

Step 8. Test stopping condition.
If weight changes have stopped (or reached an acceptable level), or if a specified maximum number of weight update iterations (Step 2) have been performed, then stop; otherwise continue.

شکل 14: الگوریتم **MRI**

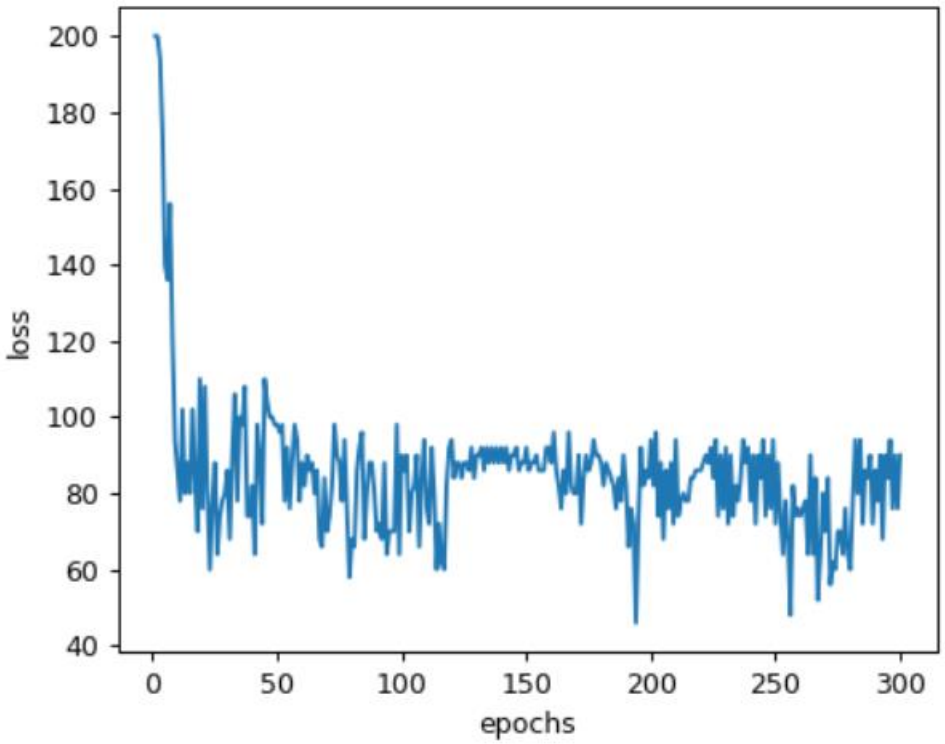
-2



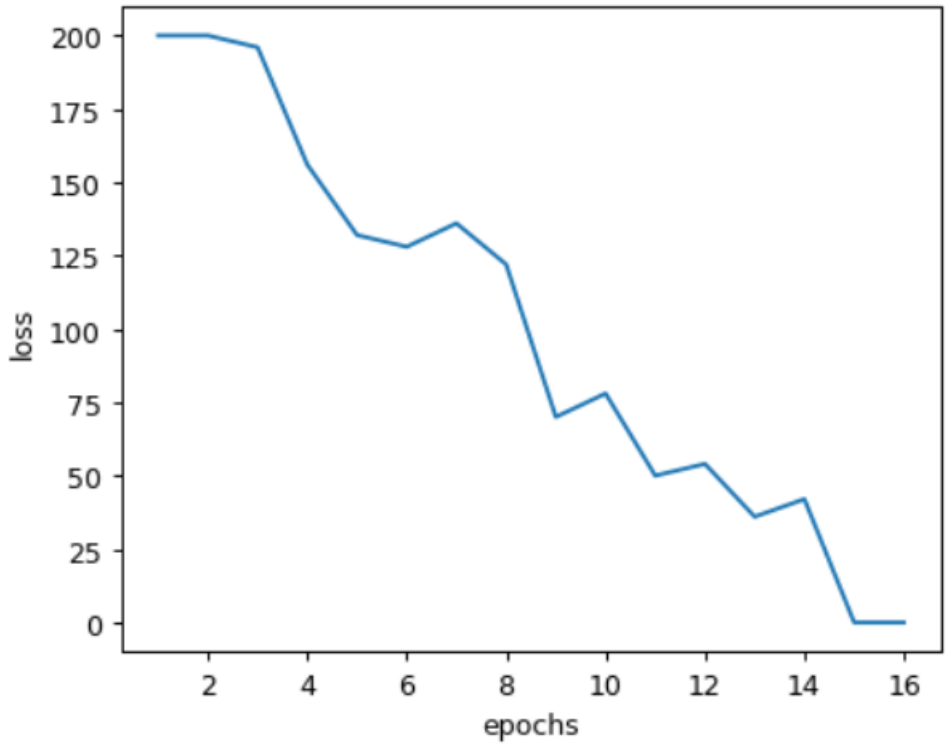
شکل 15: نمودار پراکندگی داده ها

همانطور که در شکل 15 قابل مشاهده است، قطعا نمیتوان با یک adaline داده های دو کلاس را از هم جدا کرد و به چندین adaline نیاز خواهیم داشت که در نتیجه نیاز به مدل madaline پیدا میکنیم.

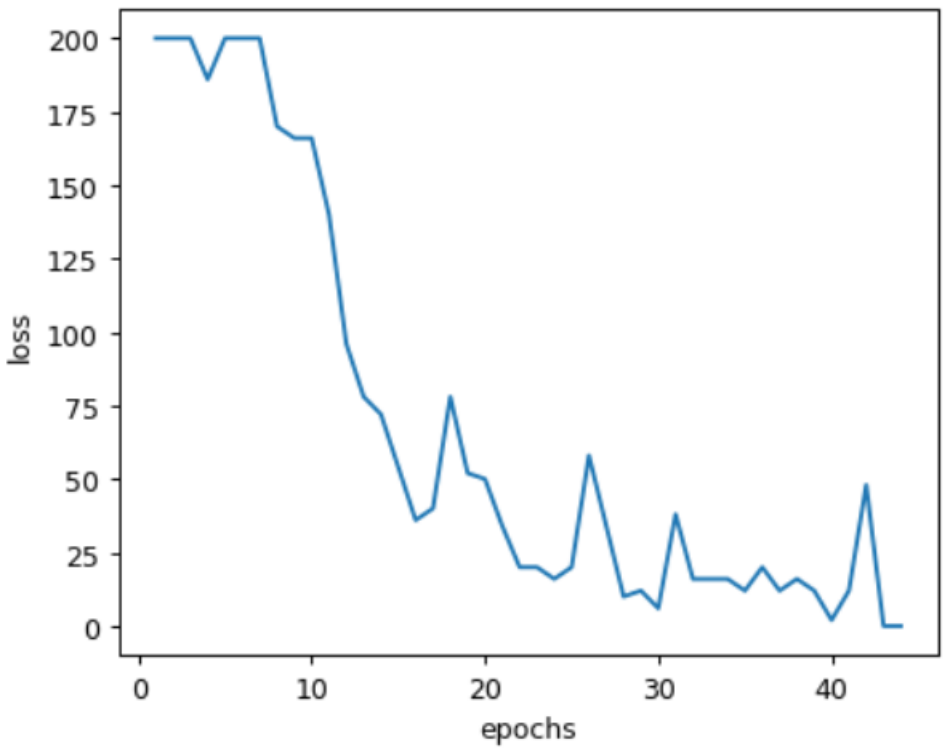
-3



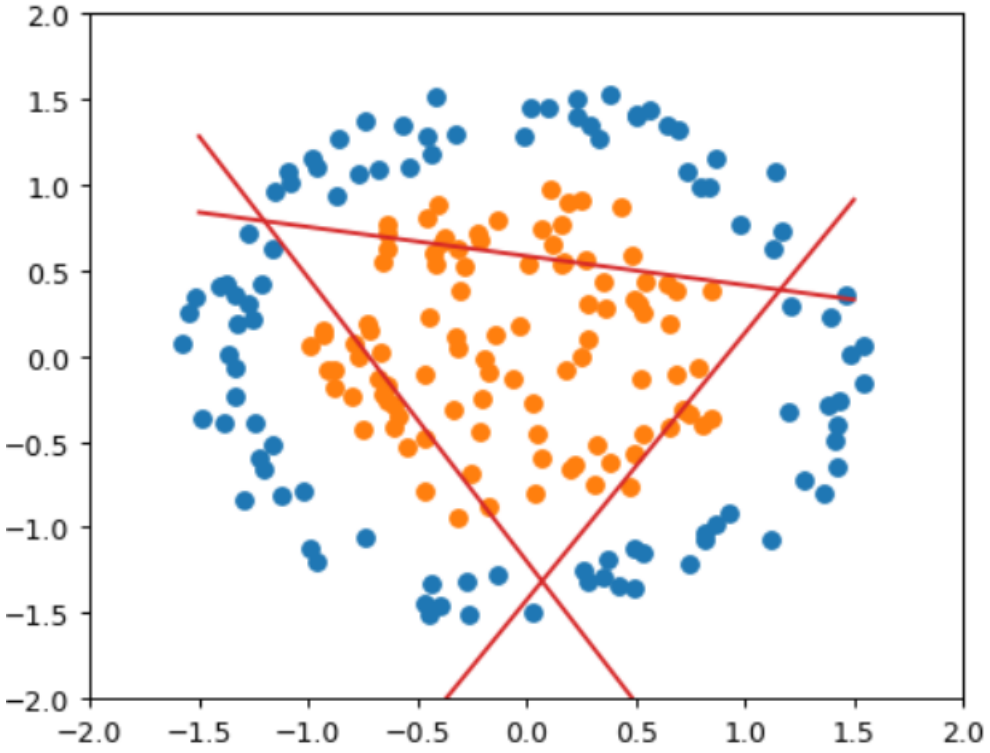
شکل 16: نمودار خطا بر اساس **epoch** برای 3 نورون



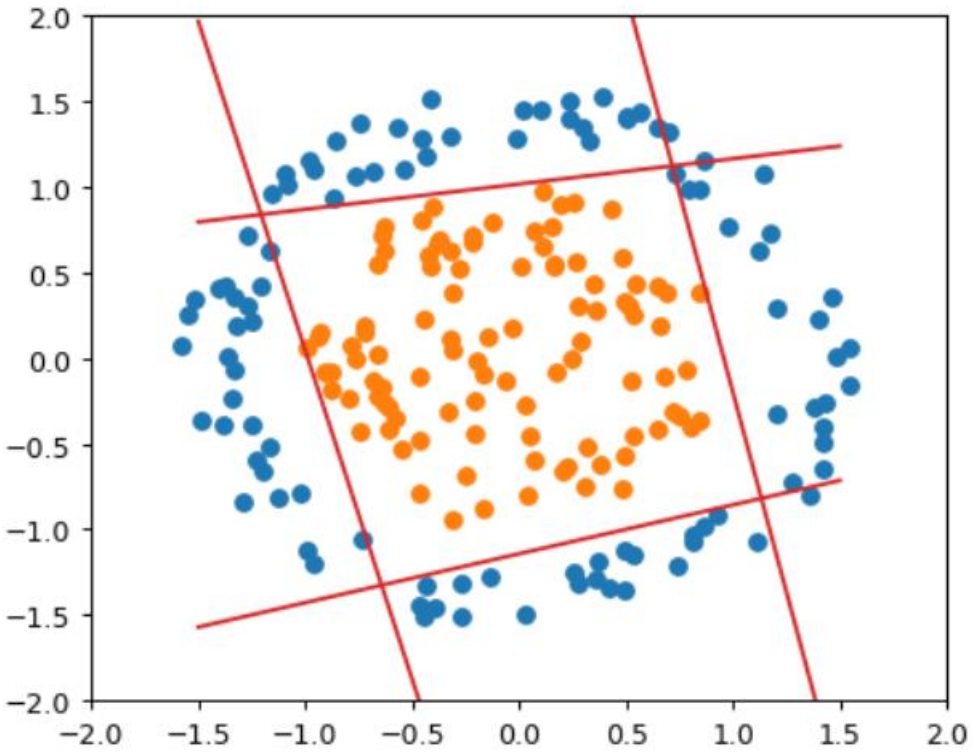
شکل 17: نمودار خطا بر اساس **epoch** برای 4 نورون



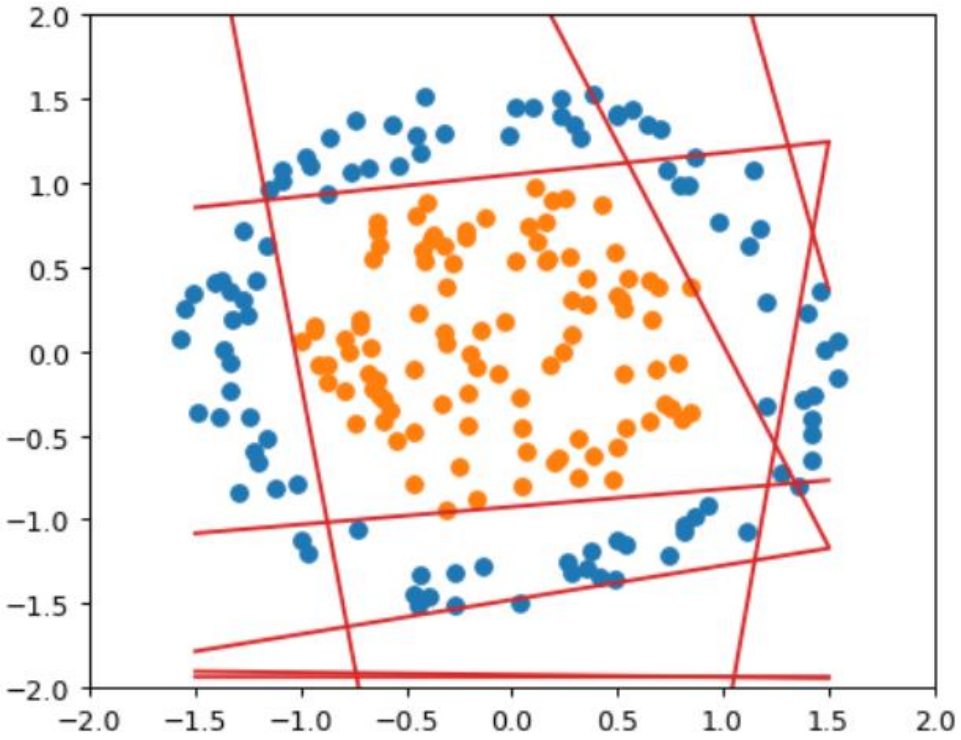
شکل 18: نمودار خطا بر اساس **epoch** برای 10 نورون



شکل 19: دقت جداسازی با 3 نورون



شکل 20: دقت جداسازی با 4 نورون



شکل 21: دقت جداسازی با 10 نورون

-4

همانطور که در شکل 16 مشخص است، Madaline با سه نورون از یک جایی به بعد نمیتواند مقدار خطا را کمتر کند، چون قدرت لازم را برای جداسازی بهترداده های دو کلاس ندارد؛ اما Madaline با چهار نورون خیلی بهتر میتواند عمل کند و با توجه به شکل 17 حتی میتواند loss را به صفر برساند. Madaline با ده نورون هم مانند Madaline با چهار نورون، loss را به صفر میتواند برساند و طبق شکل 18 میبینیم که به خوبی داده های دو کلاس را از هم جدا میکند اما به دلیل تعداد نورون بالا، دارای افزونگی میشود که در شکل 21 قابل مشاهده است.

```
print("Epoch numbers for Madaline with 3 Neurons: " + str(clf1.epochs))
print("Epoch numbers for Madaline with 4 Neurons: " + str(clf2.epochs))
print("Epoch numbers for Madaline with 10 Neurons: " + str(clf3.epochs))
```

```
Epoch numbers for Madaline with 3 Neurons: 300
Epoch numbers for Madaline with 4 Neurons: 16
Epoch numbers for Madaline with 10 Neurons: 44
```

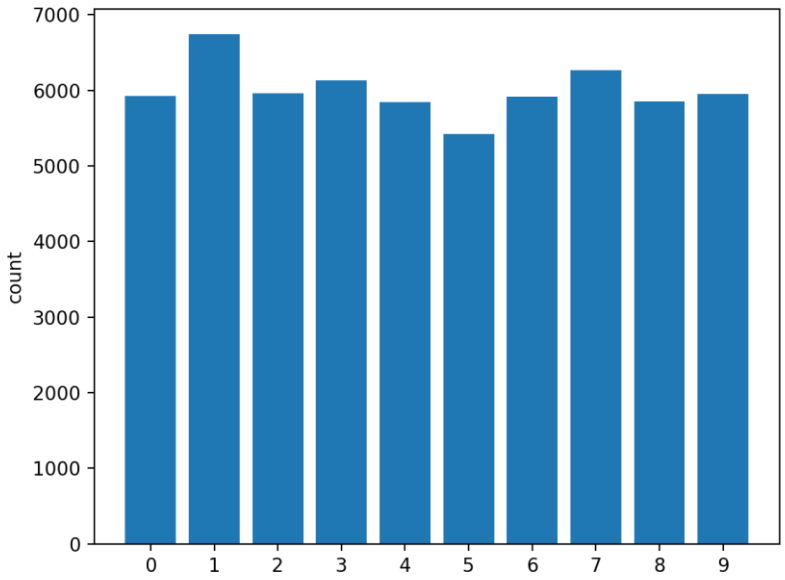
شکل 22: تعداد **epoch** ها برای سه مدل(3، 4 و 10 نورون)

برای تعداد epoch ها، مدل اول با سه نورون نتوانسته شرط عدم تغییر وزن ها را ارضا کند و با شرط دوم یعنی حداکثر تعداد تعیین شده(که 300 قرار دادیم)، به پایان رسیده است. اما مدل با 4 نورون با تعداد 16 و مدل با 10 نورون با تعداد 44 ایپاک توانسته اند شرط عدم تغییر وزن ها را ارضا کنند و نیازی به شرط دوم نشده است. تعداد ایپاک ها به دلیل عدم افزونگی در مدل با 4 نورون نسبت به مدل با 10 نورون کمتر شده است.

پاسخ ۳ – Auto-Encoders for classification

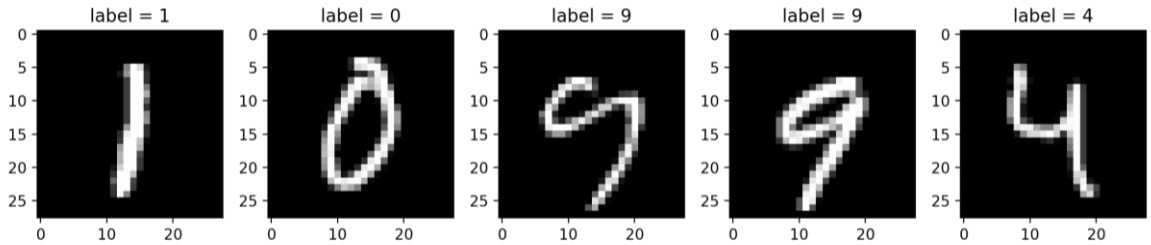
۳-۱. آشنایی و کار با دیتاست (پیش پردازش)

ابتدا با استفاده از کتابخانه keras داده ها را لود کرده و سپس با استفاده از لیبل داده های train نمودار خواسته شده را رسم می کنیم.



شکل 23: نمودار تعداد داده ها به ازای هر گروه برای **train**

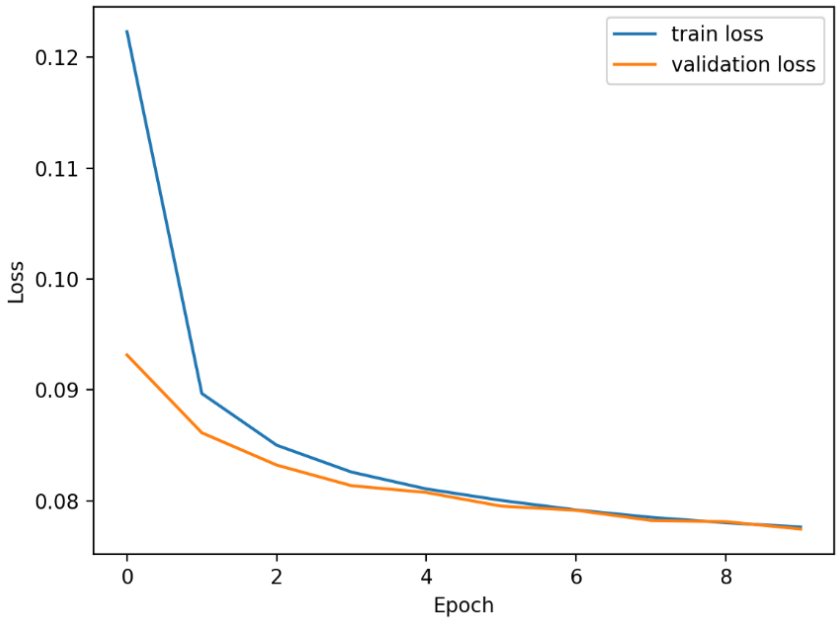
در ادامه پنج داده را به صورت رندوم مشاهده می کنیم. برای نرمالیزه کردن ورودی ها نیز از min-max نرمالیزیشن استفاده می کنیم به طوری که همه داده ها بین 0 و 1 باشند. با توجه به این نکته که داده های ورودی تصاویر سیاه و سفید هستند و کمترین مقدار پیکسل ها 0 و بیشترین مقدار 255 می باشد، نیازی به پردازش جداگانه نداریم و مستقیما داده ها را بر 255 تقسیم می کنیم.



شکل 24: تصویر5 داده رندوم

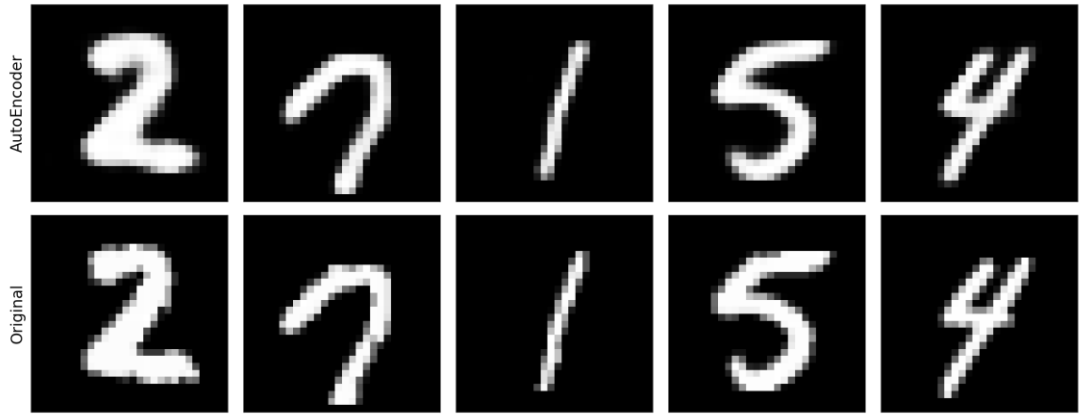
2-3. شبکه Auto-Encoder

شبکه گفته شده را پیاده سازی می کنیم به طوری که Encoder و Decoder دو مدل Sequential هستند و شبکه Auto-Encoder از اتصال این دو ساخته شده است. ترکیبات مختلفی برای activation function لایه ها را امتحان می کنیم که بهترین عملکرد مربوط به انتخاب sigmoid برای لایه آخر و Leaky-Relu برای بقیه لایه ها می باشد. چون انتظار داریم خروجی نیز به همانند خروجی در محدود 0 تا 1 باشد، عملکرد بهترین این توابع فعالسازی نیز به نوعی توجیه می شود. حال برای 10 epoch مدل را آموزش می دهیم. نتیجه به صورت زیر است:



شکل 25: نمودار **loss** و **validation-loss** برای **Auto-Encoder**

مشاهده می کنیم که شیب نمودار loss تا حد خوبی کاهش یافته و می توان فرآیند آموزش را متوقف کرد. در ادامه برای سنجش و اطمینان از عملکرد صحیح Auto-Encoder ، برای چند داده رندوم از داده های تست، ورودی و خروجی را رسم می کنیم. انتظار داریم که خروجی که Decoder از روی 30 ویژگی Encoder باز سازی می کند، تا حد خوبی شبیه تصویر اصلی باشد.

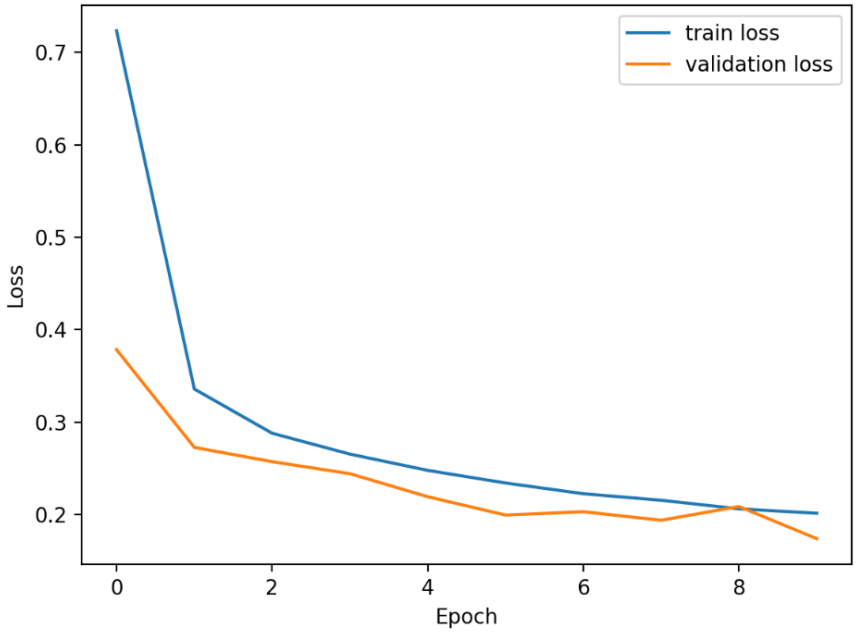


شکل 26: ورودی و خروجی شبکه **Auto-Encoder** برای 5 داده رندوم

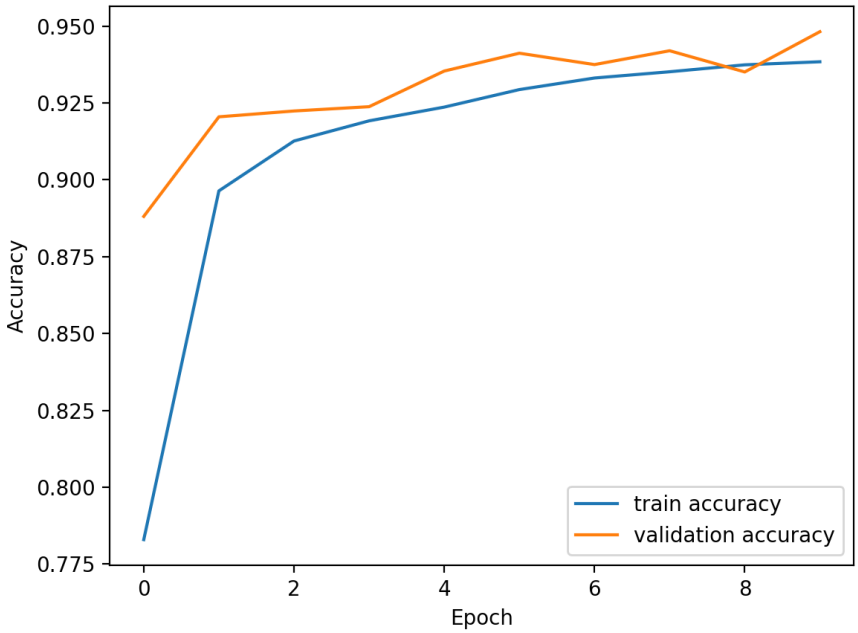
3-3. طبقه بندی

از Encoder بخش قبل استفاده می کنیم و تمام داده های آموزش و تست را با استفاده از آن به فضای ویژگی 30 تایی می بریم. سپس شبکه طبقه بند را همانند خواسته سوال به صورت دو لایه مخفی با 24 و 16 نورون و یک لایه خروجی با 10 نورون پیاده سازی می کنیم. (چون 10 کلاس داریم)

حال شبکه را با استفاده از داده ها در فضای ویژگی جدید در 10 epoch آموزش می دهیم. نمودار های خواسته شده را در ادامه مشاهده می کنیم:



شکل 27: نمودار **loss** و **validation loss** برای طبقه بند

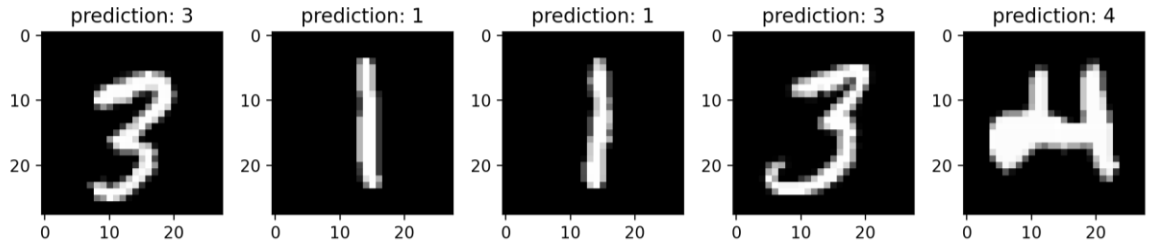


شکل 28: نمودار **accuracy** برای طبقه بند

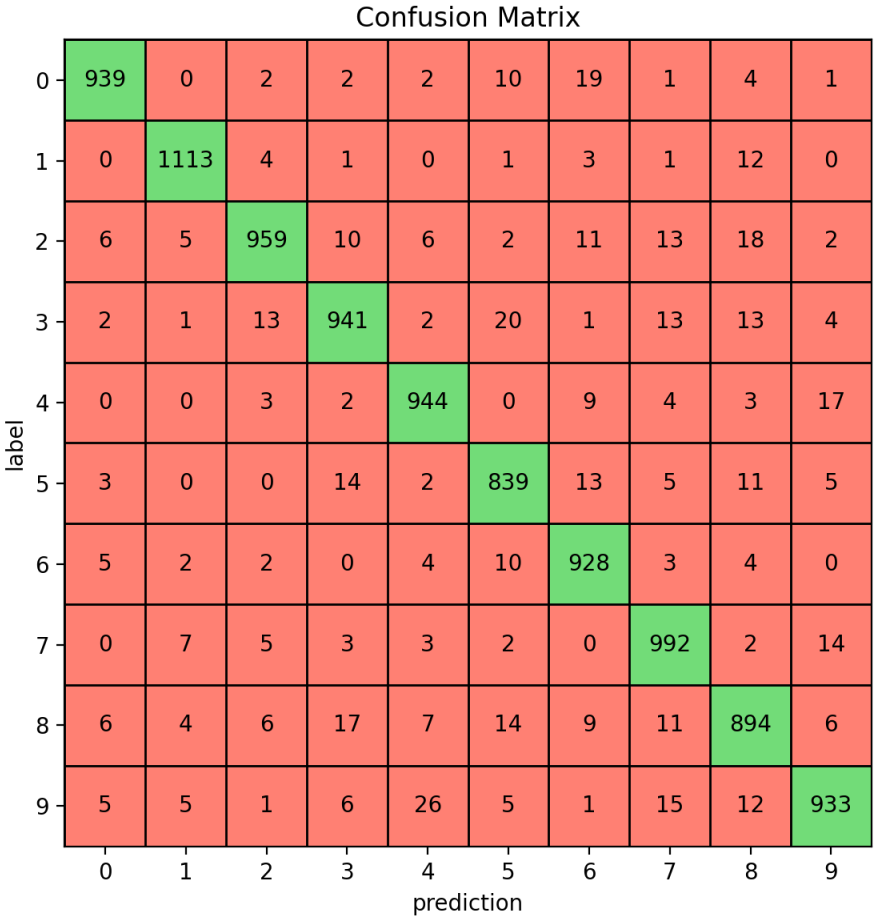
دقت طبقه پس از پایان فرآیند آموزش برابر 94.8٪ می باشد.

در نهایت یک تابع برای طبقه بند می نویسیم به طوری که با گرفتن ورودی از فضای اولیه (784 بعدی) ابتدا آن را با استفاده از Encoder به فضای ویژگی 30 بعدی ببرد و سپس با استفاده از طبقه بند آموزش داده شده لیبل آن را مشخص کند.

در ادامه خروجی مجموعه Encoder و طبقه بند را به صورت یکجا برای 5 داده رندوم مشاهده می کنیم.



شکل 29: خروجی **classifier** برای 5 داده رندوم از داده های تست



شکل 30: نمودار **confusion matrix** طبقه بند

به علت دقت خوبی که طبقه بند دارد، مشاهده می کنیم که اکثر داده ها روی قطر اصلی ماتریس قرار دارند و این یعنی درست طبقه بندی شده اند. اگر تجمع طبقه بندی های غلط را نیز بررسی کنیم، به نتایج جالبی می رسیم. برای مثال دو عدد 4 و 9 شباهت کلی به یکدیگر دارند و همانطور که در نمودار بالا می بینیم، به نسبت این اشتباه بیشتر رخ داده است.

پاسخ ۴ – پرسترون چند لایه

۴-۱. پیش پردازش

1- خواندن فایل csv. و فراخوانی تابع info.

```
df = pd.read_csv("sample_data/CarPrice_Assignment.csv")
df.head(10)
```

	car_ID	symboling	CarName	fueltype	aspiration	doornumber	carbody	drivewheel	enginelocation	wheelbase	...	enginesize	fuelsystem	boreratio	stroke	compressionratio	ho
0	1	3	alfa-romero giulia	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0	
1	2	3	alfa-romero stelvio	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0	
2	3	1	alfa-romero Quadrifoglio	gas	std	two	hatchback	rwd	front	94.5	...	152	mpfi	2.68	3.47	9.0	
3	4	2	audi 100 ls	gas	std	four	sedan	fwd	front	99.8	...	109	mpfi	3.19	3.40	10.0	
4	5	2	audi 100ls	gas	std	four	sedan	4wd	front	99.4	...	136	mpfi	3.19	3.40	8.0	
5	6	2	audi fox	gas	std	two	sedan	fwd	front	99.8	...	136	mpfi	3.19	3.40	8.5	
6	7	1	audi 100ls	gas	std	four	sedan	fwd	front	105.8	...	136	mpfi	3.19	3.40	8.5	
7	8	1	audi 5000	gas	std	four	wagon	fwd	front	105.8	...	136	mpfi	3.19	3.40	8.5	
8	9	1	audi 4000	gas	turbo	four	sedan	fwd	front	105.8	...	131	mpfi	3.13	3.40	8.3	
9	10	0	audi 5000s (diesel)	gas	turbo	two	hatchback	4wd	front	99.5	...	131	mpfi	3.13	3.40	7.0	

10 rows × 26 columns

شکل 31: نمایش ده داده ابتدایی دسته داده

این دیتاست دارای 25 ویژگی از ماشین همراه با قیمت میباشد که با استفاده از ویژگی ها در این دیتاست باید قیمت حدودی حدس زده شود.

```
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)
```

شکل 32: نمایش اطلاعات **dataframe** با **info**

در کل 205 دیتا در این دیتاست وجود دارد که جنس هر کدام میتواند از نوع int، float و object باشد.

2- تعداد داده های Nan بر اساس ستون

```
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 33: تعداد داده های ستون در هر دسته

با استفاده از تابع isna میتوان خانه هایی که NaN هستند را بر اساس True یا False برای هر خانه نمایش داد . با sum تعداد خانه هایی که NaN هستند را در یک ستون نمایش میدهد که در این دیتاست هیچ دیتای NaN وجود ندارد.

3- پاک کردن سه ستون CarName، car_ID و symboling

در ابتدا ستون CarName را در متغیر CompanyName ذخیره میکنیم و سپس با استفاده از drop، سه ستون نام برده شده را از دیتاست جدا میکنیم.

```python
def get_unique_company_names(CompanyName):
  CompanyName = [name.split(" ")[0] for name in CompanyName]
  CompanyName_set = set(CompanyName)
  return CompanyName_set

get_unique_company_names(CompanyName)
```

```
{'Nissan',
 'alfa-romero',
 'audi',
 'bmw',
 'buick',
 'chevrolet',
 'dodge',
 'honda',
 'isuzu',
 'jaguar',
 'maxda',
 'mazda',
 'mercury',
 'mitsubishi',
 'nissan',
 'peugeot',
 'plymouth',
 'porcshce',
 'porsche',
 'renault',
 'saab',
 'subaru',
 'toyota',
 'toyouta',
 'vokswagen',
 'volkswagen',
 'volvo',
 'vw'}
```

شکل 34: نام کمپانی های موجود در دسته داده

با استفاده از تابع شکل، کمپانی هایی که در این دیتاست استفاده شده اند را جدا میکنیم. برای این کار ابتدا کلمه اول هر ردیف از این ستون را جدا کرده و در متغیری ذخیره میکنیم. سپس به صورت set در می آوریم تا اسم کمپانی هایی که یکتا هستند به دست آید.

```
correct_dic = {'Nissan': 'nissan', 'maxda': 'mazda', 'porcshce': 'porsche', 'toyouta': 'toyota', 'vokswagen': 'volkswagen'}
for index, name in enumerate(CompanyName):
  first_word_of_string = name.split(" ")[0]
  if first_word_of_string in correct_dic:
    CompanyName[index] = CompanyName[index].replace(first_word_of_string, correct_dic[first_word_of_string])
get_unique_company_names(CompanyName)
```

```
<ipython-input-10-1ba53ba08afe>:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-
  CompanyName[index] = CompanyName[index].replace(first_word_of_string, correct_dic[first_word_of_string])
{'alfa-romero',
 'audi',
 'bmw',
 'buick',
 'chevrolet',
 'dodge',
 'honda',
 'isuzu',
 'jaguar',
 'mazda',
 'mercury',
 'mitsubishi',
 'nissan',
 'peugeot',
 'plymouth',
 'porsche',
 'renault',
 'saab',
 'subaru',
 'toyota',
 'volkswagen',
 'volvo',
 'vw'}
```

شکل 35: خروجی بعد از تصحیح نام های غلط کمپانی

اسم چند کمپانی(Nissan، maxda، porcshce، toyouta، vokswagen) اشتباه نوشته شده است. برای تصحیح، یک دیکشنری که این اسم های غلط را به اسم های درست map میکند تشکیل میدهیم. سپس به ازای هر ردیف از این ستون، اگر اسم غلط کمپانی آورده شده باشد، با اسم درست آن با استفاده از دیکشنری مذکور، جایگزین میکنیم.

4- نمایش ماتریس correlation

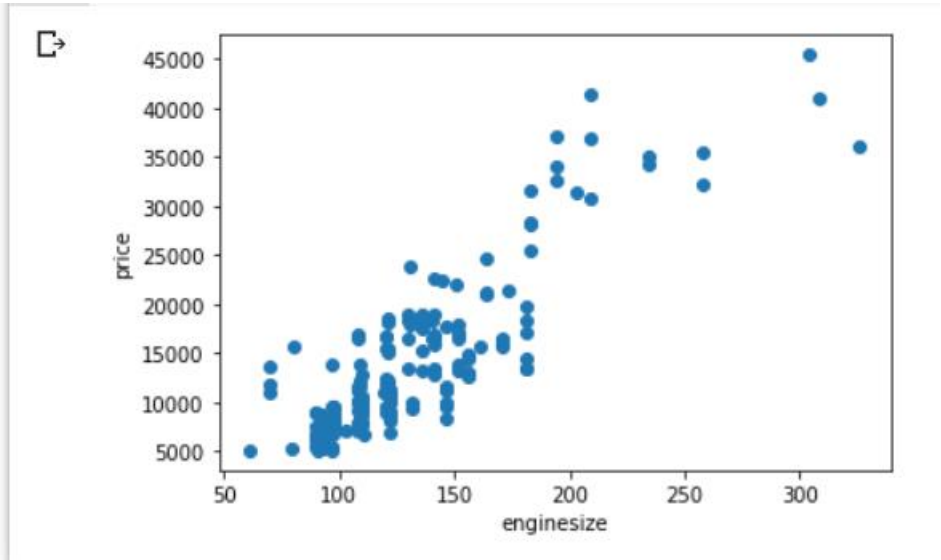
```
df.corr()
```

	wheelbase	carlength	carwidth	carheight	curbweight	enginesize	boreratio	stroke	compressionratio	horsepower	peakrpm	citympg	highwaympg	price
wheelbase	1.000000	0.874587	0.795144	0.589435	0.776386	0.569329	0.488750	0.160959	0.249786	0.353294	-0.360469	-0.470414	-0.544082	0.577816
carlength	0.874587	1.000000	0.841118	0.491029	0.877728	0.683360	0.606454	0.129533	0.158414	0.552623	-0.287242	-0.670909	-0.704662	0.682920
carwidth	0.795144	0.841118	1.000000	0.279210	0.867032	0.735433	0.559150	0.182942	0.181129	0.640732	-0.220012	-0.642704	-0.677218	0.759325
carheight	0.589435	0.491029	0.279210	1.000000	0.295572	0.067149	0.171071	-0.055307	0.261214	-0.108802	-0.320411	-0.048640	-0.107358	0.119336
curbweight	0.776386	0.877728	0.867032	0.295572	1.000000	0.850594	0.648480	0.168790	0.151362	0.750739	-0.266243	-0.757414	-0.797465	0.835305
enginesize	0.569329	0.683360	0.735433	0.067149	0.850594	1.000000	0.583774	0.203129	0.028971	0.809769	-0.244660	-0.653658	-0.677470	0.874145
boreratio	0.488750	0.606454	0.559150	0.171071	0.648480	0.583774	1.000000	-0.055909	0.005197	0.573677	-0.254976	-0.584532	-0.587012	0.553173
stroke	0.160959	0.129533	0.182942	-0.055307	0.168790	0.203129	-0.055909	1.000000	0.186110	0.080940	-0.067964	-0.042145	-0.043931	0.079443
compressionratio	0.249786	0.158414	0.181129	0.261214	0.151362	0.028971	0.005197	0.186110	1.000000	-0.204326	-0.435741	0.324701	0.265201	0.067984
horsepower	0.353294	0.552623	0.640732	-0.108802	0.750739	0.809769	0.573677	0.080940	-0.204326	1.000000	0.131073	-0.801456	-0.770544	0.808139
peakrpm	-0.360469	-0.287242	-0.220012	-0.320411	-0.266243	-0.244660	-0.254976	-0.067964	-0.435741	0.131073	1.000000	-0.113544	-0.054275	-0.085267
citympg	-0.470414	-0.670909	-0.642704	-0.048640	-0.757414	-0.653658	-0.584532	-0.042145	0.324701	-0.801456	-0.113544	1.000000	0.971337	-0.685751
highwaympg	-0.544082	-0.704662	-0.677218	-0.107358	-0.797465	-0.677470	-0.587012	-0.043931	0.265201	-0.770544	-0.054275	0.971337	1.000000	-0.697599
price	0.577816	0.682920	0.759325	0.119336	0.835305	0.874145	0.553173	0.079443	0.067984	0.808139	-0.085267	-0.685751	-0.697599	1.000000

شکل 36: ماتریس وابستگی

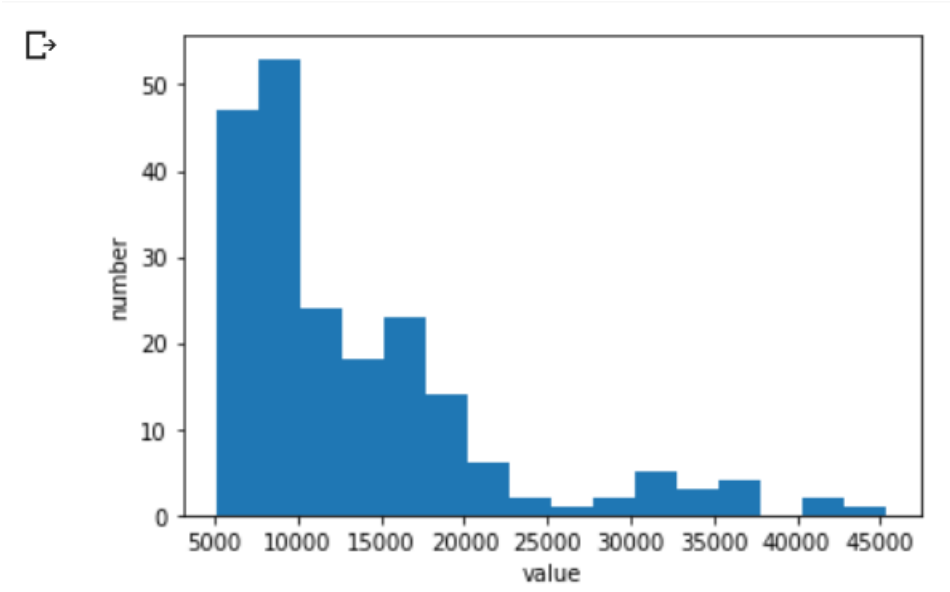
قبل از تبدیل داده های categorical به داده های عددی، ماتریس وابسنگی را نشان میدهیم و با توجه به این ماتریس، ستون قیمت بیشترین وابستگی را با ویژگی enginesize دارد؛ زیرا طبق این جدول بیشترین correlation قیمت با این متغیر میباشد.

5- نمودار توزیع قیمت و نمودار قیمت برحسب enginesize



شکل 37: نمودار قیمت بر اساس انداره موتور

نمودار قیمت بر حسب enginesize نشان دهنده وابستگی بین این دو ویژگی است و میتوان مثلا با یک خط این دو ویژگی را بر حسب دیگری به دست اورد و وابسته خطی دارند و میتوان یکی را بر حسب تابعی از دیگری نوشت.



شکل 38: نمودار توزیع قیمت

در نمودار توزیع قیمت، نشان میدهد هر حدود قیمتی شامل چند مقدار از دیتاست میشود و بین 5000 تا 10000 بیشترین تعداد را داراست که نشان میدهد دیتاست نسبت به زیاد بودن ماشین های لوکس بایاس ندارد.

6- تبدیل داده های categorical به داده های عددی

مقادیر دو ستون cylindernumber و doornumber به صورت عدد حروفی آمده اند و با استفاده از کتابخانه word2number میتوان مقادیر عددی حروفی را به عدد تبدیل کرد(مثلا تبدیل two به 2)

```
categorical_col = ['fueltype', 'aspiration', 'carbody', 'drivewheel',
                   'enginelocation', 'enginetype', 'fuelsystem']

# get the dummies and store it in a df
df = pd.get_dummies(data=df, columns=categorical_col, drop_first=True)

df.head(10)
```

	doornumber	wheelbase	carlength	carwidth	carheight	curbweight	cylindernumber	enginesize	boreratio	stroke	...	enginetype_ohcf	enginetype_ohcv	enginetype_
0	2	88.6	168.8	64.1	48.8	2548	4	130	3.47	2.68	...	0	0	
1	2	88.6	168.8	64.1	48.8	2548	4	130	3.47	2.68	...	0	0	
2	2	94.5	171.2	65.5	52.4	2823	6	152	2.68	3.47	...	0	1	
3	4	99.8	176.6	66.2	54.3	2337	4	109	3.19	3.40	...	0	0	
4	4	99.4	176.6	66.4	54.3	2824	5	136	3.19	3.40	...	0	0	
5	2	99.8	177.3	66.3	53.1	2507	5	136	3.19	3.40	...	0	0	
6	4	105.8	192.7	71.4	55.7	2844	5	136	3.19	3.40	...	0	0	
7	4	105.8	192.7	71.4	55.7	2954	5	136	3.19	3.40	...	0	0	
8	4	105.8	192.7	71.4	55.9	3086	5	131	3.13	3.40	...	0	0	
9	2	99.5	178.2	67.9	52.0	3053	5	131	3.13	3.40	...	0	0	

10 rows × 38 columns

شکل 39: خزوجی بعد از تبدیل داده های غیرعددی به داده های عددی

با استفاده از تابع get_dummies از کتابخانه pandas، داده های غیرعددی شناسایی شده را به به داده های عددی تبدیل میکنیم. داده های غیرعددی شناسایی شده شامل fueltype، aspiration، carbody، drivewheel، enginelocation، enginetype و fuelsystem است. در واقع تابع get_dummies به مانند one hot encoding عمل میکند و مقادیر موجود در ستون را به صورت یک ویژگی در نظر میگیرد و هر ردیف اگر آن ویژگی را داشته باشد، یک وگرنه صفر میگذارد. به این ترتیب با اعمال این تابع روی دیتاست، در مجموع 37 ستون خواهیم داشت.

در آخر price که ستون target میباشد را از دیتاست خارج میکنیم تا بعدا به عنوان ستون مقصد استفاده کنیم.

7- جداسازی دیتا به سه دسته train، validation و test

در سوال دسته validatoion خواسته نشده اما چون در قسمت بعد validation_loss را میخواهد، همینجا به دسته داده هارا تقسیم میکنیم.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(df, price, test_size=0.15, random_state=42)
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.175, random_state=42)
len(X_train), len(X_val), len(X_test)
```

```
(143, 31, 31)
```

شکل 40: تقسیم داده ها به سه دسته **train**، **validation** و **test**

در نهایت 143 دیتا برای train، 31 دیتا برای validation و 31 دیتا برای test خواهیم داشت. 70 درصد دیتاها متعلق به train، 15 درصد برای validation و 15 درصد برای test میباشد.

8- scale داده ها با استفاده از MinMaxScalar

```
from sklearn.preprocessing import MinMaxScaler
# define min max scaler
scaler = MinMaxScaler().fit(X_train)
# transform data
X_train = scaler.transform(X_train)
X_val = scaler.transform(X_val)
X_test = scaler.transform(X_test)
pd.DataFrame(X_train).head(10)
```

	0	1	2	3	4	5	6	7	8	9	...	27	28	29	30	31	32	33	34	35	36
0	0.0	0.320690	0.493103	0.428571	0.200000	0.441478	0.2	0.335938	0.757143	0.871429	...	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
1	1.0	0.313793	0.432759	0.171429	0.941667	0.205162	0.2	0.085938	0.364286	0.457143	...	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.244828	0.093103	0.209524	0.400000	0.053850	0.2	0.085938	0.264286	0.638095	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.334483	0.489655	0.342857	0.133333	0.245216	0.2	0.156250	0.450000	0.661905	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
4	0.0	0.244828	0.093103	0.209524	0.400000	0.060970	0.2	0.085938	0.264286	0.638095	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5	0.0	0.406897	0.544828	0.361905	0.350000	0.325768	0.2	0.296875	0.771429	0.680952	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0
6	0.0	0.875862	0.941379	0.971429	0.633333	0.843792	0.6	0.914062	0.900000	0.609524	...	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0
7	0.0	0.341379	0.422414	0.400000	0.266667	0.210948	0.2	0.156250	0.435714	0.719048	...	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0
8	1.0	0.734483	0.725862	0.628571	0.741667	0.534490	0.2	0.195312	0.657143	0.533333	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0
9	1.0	0.910345	0.948276	0.742857	0.416667	1.000000	0.4	0.734375	0.778571	1.000000	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0

10 rows × 37 columns

شکل 41: خروجی بعد از **scale** داده ها

این scalar به گونه ای عمل میکند که مینیمم و ماکسیمم را بین 0 و 1 اسکیل میکند.

$$x_{norm}^{(i)} = \frac{x^{(i)} - x_{min}}{x_{max} - x_{min}}$$

این کار را روی داده های train فیت میکنیم(یعنی مقادیر xmin و xmax بر این اساس به دست می آید) و سپس بر اساس مقادیر به دست آمده از این طریق با استفاده از فرمول فوق، روی داده های سه دسته موجود (train، validation و test) transform میکنیم.

Multi-Layer Perceptron .۲-۴

1 - ساخت سه مدل MLP ساده

```
input_dim = len(X_train[-1])
output_dim = 1

def model_with_1_hidden_layer():
  model = Sequential([
      Dense(128, activation='relu', input_shape=(input_dim,)),
      Dropout(0.1),
      Dense(1, activation='linear')
  ])
  return model

def model_with_2_hidden_layer():
  model = Sequential([
      Dense(256, activation='relu', input_shape=(input_dim,)),
      Dropout(0.1),
      Dense(32, activation='relu'),
      Dropout(0.1),
      Dense(1, activation='linear')
  ])
  return model

def model_with_3_hidden_layer():
  model = Sequential([
      Dense(256, activation='relu', input_shape=(input_dim,)),
      Dropout(0.1),
      Dense(64, activation='relu'),
      Dropout(0.1),
      Dense(8, activation='relu'),
      Dropout(0.1),
      Dense(1, activation='linear')
  ])
  return model
```

شکل 42: مدل های تعریف شده با 1، 2 و 3 لایه مخفی

در اینجا سه مدل که هر کدام به ترتیب 1، 2 و 3 لایه مخفی دارد را میسازیم و برای هر لایه مخفی، تابع فعال سازی relu و در لایه آخر را linear قرار میدهیم، چون مسئله از نوع رگرسیون است و این توابع فعالسازی در این نوع مسئله جواب میدهد. همچنین بعد از هر لایه مخفی برای جلوگیری از overfitting یک Dropout با احتمال 10٪ قرار داده ایم.

2 - بررسی loss و optimzer های مختلف

Loss Functions:

MSE:
is generally used when we have regression type of problem and target variable is continuous.
MSE = (1/n) * Σ(actual – forecast)^2
BINARY CROSS ENTROPY / LOG LOSS:
It is almost used for logistic regression problem. l=−(ylog(p)+(1−y)log(1−p))
MAE
MAE is another metric which is used to calculate the loss function.
MeanAbsolutePercentageError
Computes the mean absolute percentage error between y_true & y_pred.
loss = 100 * abs((y_true - y_pred) / y_true)
MeanSquaredLogarithmicError
Computes the mean squared logarithmic error between y_true & y_pred.
loss = square(log(y_true + 1.) - log(y_pred + 1.))

انواع مختلفی از loss وجود دارد که چند مورد از آنها در تصویر بالا آورده شده است. دو loss که در ادامه استفاده میشود، MeanAbsolutePercentageError و MeanSquaredLogarithmicError میباشد فرمول آنها در تصویر بالا قابل مشاهده است. هر دو از اختلاف ایجاد شده بین target و prediction استفاده میکند. یکی از دلایل عدم استفاده از MSE این بود که چون با مسئله رگرسیون مواجه هستیم، توان دو میتواند مقدار loss را بسیار بزرگ کند و با استفاده از sgd با NaN در محاسبه epoch ها در loss مواجه میشدیم. از log loss نمیتوان در این مسئله استفاده کرد؛ چون با مسئله رگرسیون مواجه هستیم و این loss function برای مسائل logistic بهتر است. بقیه loss های نام برده شده قابل استفاده در مسئله رگرسیون است.

sgd:

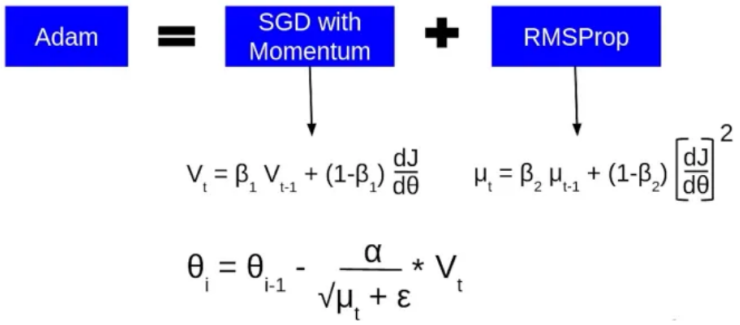
Stochastic gradient descent (often abbreviated SGD) is an iterative method for optimizing an objective function with suitable smoothness properties.

Compared to Gradient Descent, Stochastic Gradient Descent is much faster, and more suitable to large-scale datasets. But since the gradient it's not computed for the entire dataset, and only for one random point on each iteration, the updates have a higher variance.

Adam:

In SGD with momentum it resolved the issue of getting stuck at local minima using the weighted sum of previously accumulated gradient. In RMSProp it resolved the problem of same learning rate for all the parameters using sum of square gradients.

Now, we will look at the most commonly and widely used optimizer i.e. ADAM. It combines both SGD with momentum to resolve local minima problem and RMSProp which uses sum of square of previous gradients to resolve same learning rate issue.



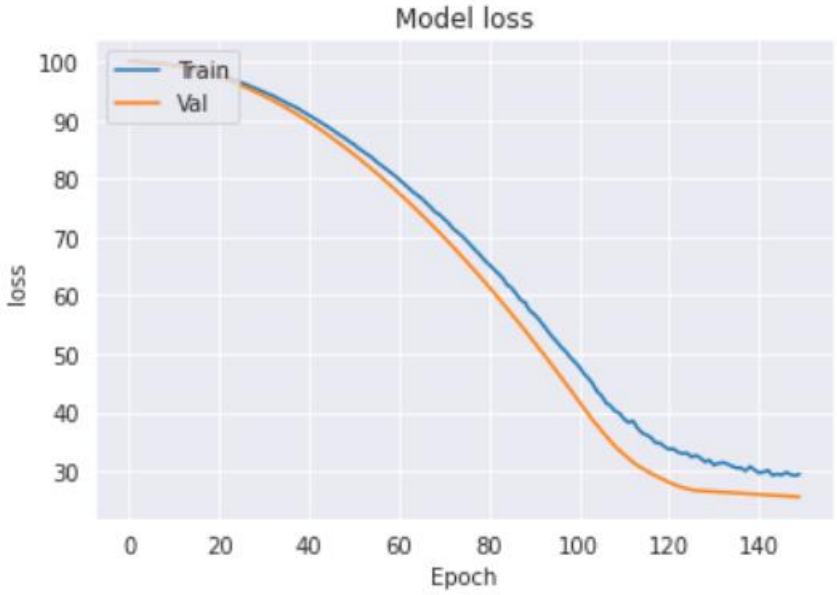
sgd ازفرمول $\theta=\theta-\eta\cdot\nabla\theta J(\theta;x(i);y(i))$ استفاده میکند که با اینکه با مشتق مواجه هستیم و برای شبکه های عصبی بزرگ ممکن است طول بکشد و همچنین ممکن است در لایه های اول با vanishing مواجه شود، اما در اینجا عملکرد نسبتا خوبی از خود نشان داد. همچنین sgd نسبت به gradient descent، سریع تر است و برای دیتاهای بزرگتر بهتر است ولی با توجه به ماهیت آن ممکن است با واریانس بیشتری مواجه بشویم.

adam در واقع ایده های دو optimzer به نام های sgd و RMSProp استفاده میکند که مشکل گیر کردن در local minima که در sgd رخ میداد و همچنین یکسان بودن learningr rate در epoch ها که در RMSProp رخ میداد را حل کرده و این دو را با هم ترکیب میکند.

3- آموزش مدل با adam و MeanAbsolutePercentageError

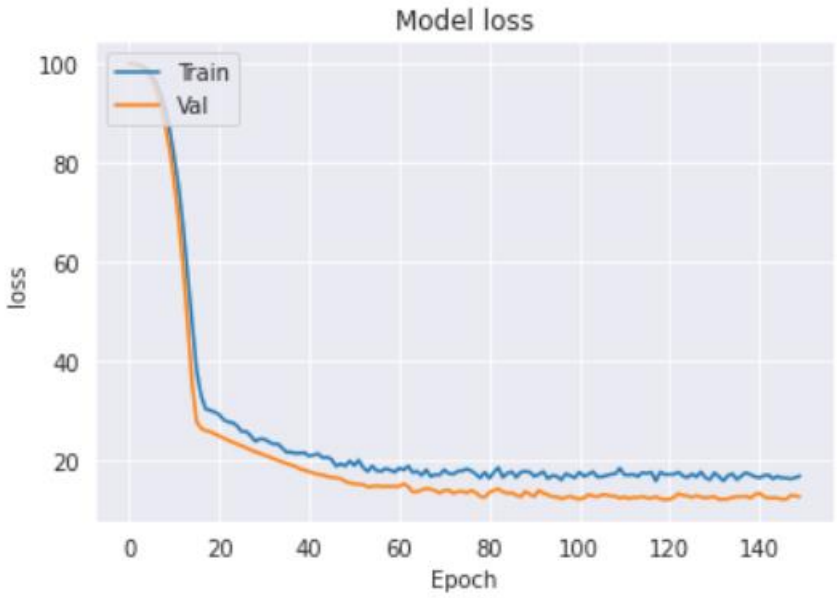
a. یک لایه مخفی:

در این قسمت با یک لایه مخفی، مدل را آموزش میدهیم. تعداد batch را 8 در نظر گرفته و در 150 epoch آموزش انجام میشود.



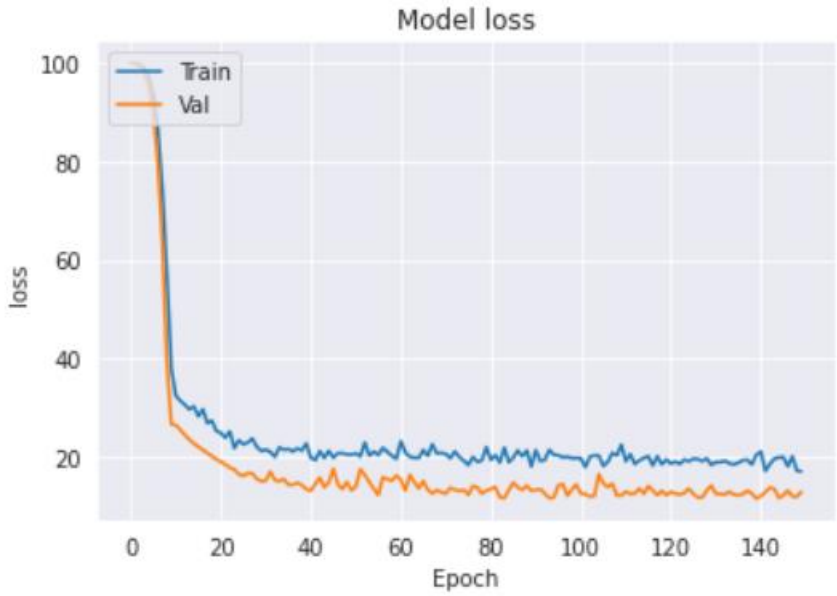
شکل 43: نمودار خطا با 1 لایه مخفی + **adam + MeanAbsolutePercentageError**

b. دو لایه مخفی:



شکل 44: نمودار خطا با 2 لایه مخفی + **adam + MeanAbsolutePercentageError**

c. سه لایه مخفی:



شکل 45: نمودار خطا با 3 لایه مخفی + **adam + MeanAbsolutePercentageError**

همانطور که در شکل های بالا قابل مشاهده است، خطا در مدل با سه لایه مخفی نسبت به دو تای دیگر با نرخ بالاتری به صفر نزدیک میشود. مدل با یک لایه مخفی کم ترین نرخ را برای رسیدن به خطای صفر داراست و دلیل آن میتواند به خاطر سادگی مدل نسبت به دو مورد دیگر باشد.

R2 Score

The R2 score is a very important metric that is used to evaluate the performance of a regression-based machine learning model. It is pronounced as R squared and is also known as the coefficient of determination. It works by measuring the amount of variance in the predictions explained by the dataset.
R2 Score is between 0 and 1, the closer to 1, the better the regression fit.

$$R2 = 1 - \frac{\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{N}(y_i - \bar{y})}, \qquad \bar{y} = \frac{1}{N}\sum_{i=1}^{N} y_i$$

معیار R2 Score: این معیار ازمهم ترین معیار ها برای مسائل رگرسیون میباشد که مقدار آن بین 0 و 1 قرار میگیرد و هر چه به یک نزدیک تر باشد یعنی این مدل نتیجه ارزیابی بهتری را داشته است. فرمول آن در قسمت بالا قابل مشاهده است.

```
[46] predictions1 = model1.predict(X_test)
     predictions2 = model2.predict(X_test)
     predictions3 = model3.predict(X_test)

     1/1 [==============================] - 0s 62ms/step
     1/1 [==============================] - 0s 79ms/step
     1/1 [==============================] - 0s 76ms/step
```

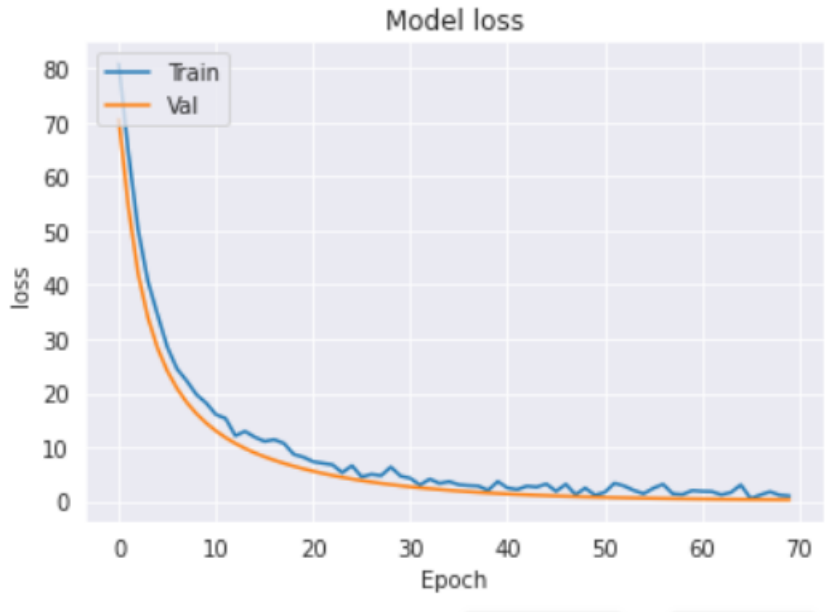
```
from sklearn.metrics import r2_score
print("model1 with 1 hiddenlayer-> r2_score: " + str(r2_score(y_test, predictions1)))
print("model2 with 2 hiddenlayer-> r2_score: " + str(r2_score(y_test, predictions2)))
print("model3 with 3 hiddenlayer-> r2_score: " + str(r2_score(y_test, predictions3)))

model1 with 1 hiddenlayer-> r2_score: 0.3009783352857245
model2 with 2 hiddenlayer-> r2_score: 0.5568922754329007
model3 with 3 hiddenlayer-> r2_score: 0.5915051599193155
```

شکل 46: **R2 Score** برای سه مدل با لایه های متفاوت

با توجه به نتایج بالا، مدل سوم که تعداد لایه های مخفی آن سه تا بود، عملکرد بهتری نسبت به دوتای دیگر از خود نشان داده است که دلیل آن میتواند یادگیری ویژگی های سخت تر به دلیل مدل پیچیده تر باشد.

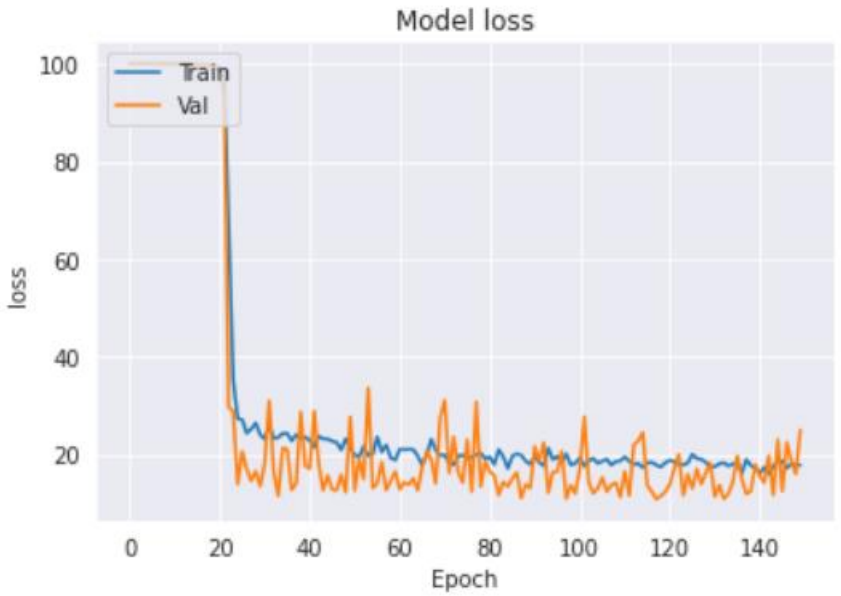
4- سه مدل دیگر با optimizer و loss متفاوت

a. مدل با adam و MeanSquaredLogarithmicError:



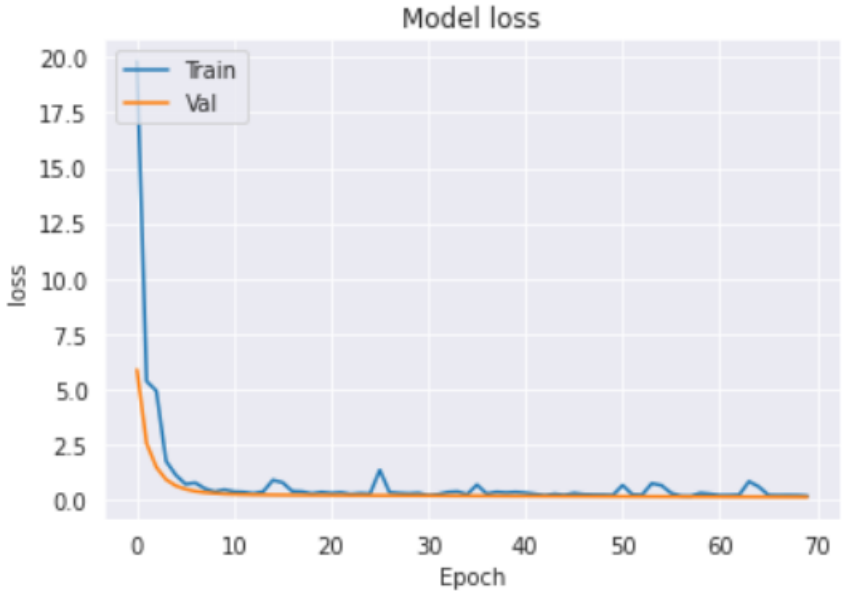
شکل 47: نمودار خطا با 3 لایه مخفی + **adam + MeanSquaredLogarithmicError**

b. مدل با sgd و MeanAbsolutePercentageError:



شکل 48: نمودار خطا با 3 لایه مخفی + **sgd + MeanAbsolutePercentageError**

c. مدل با sgd و MeanSquaredLogarithmicError:



شکل 49: نمودار خطا با 3 لایه مخفی + **sgd + MeanSquaredLogarithmicError**

```
[168] predictions4 = model4.predict(X_test)
      predictions5 = model5.predict(X_test)
      predictions6 = model6.predict(X_test)

      1/1 [==============================] - 0s 92ms/step
      1/1 [==============================] - 0s 100ms/step
      1/1 [==============================] - 0s 93ms/step
```

```
print("model with adam + mean_absolute_percentage_error: " + str(r2_score(y_test, predictions2)))
print("model with adam + mean_squared_logarithmic_error: " + str(r2_score(y_test, predictions4)))
print("model with sgd + mean_absolute_percentage_error: " + str(r2_score(y_test, predictions5)))
print("model with sgd + mean_squared_logarithmic_error: " + str(r2_score(y_test, predictions6)))
```

```
model with adam + mean_absolute_percentage_error: 0.5731648367289317
model with adam + mean_squared_logarithmic_error: 0.07250071900865962
model with sgd + mean_absolute_percentage_error: 0.7457766003850628
model with sgd + mean_squared_logarithmic_error: 0.3697821942468238
```

شکل 50: **R2 Score** برای سه مدل با **loss** و **optimizer** های متفاوت

با توجه به نتایجی که به دست آمد، sgd + MeanAbsolutePercentageError با R Score ٪74 بهترین مدل برای دیتاست داده شده انتخاب شد.

5- پیش بینی 5 داده test با استفاده از مدل انتخاب شده

در این قسمت با استفاده از تابع random، 5 داده از دیتاست test انتخاب کرده با استفاده از مدل انتخاب شده که دارای 3 لایه مخفی بوده و با sgd + MeanAbsolutePercentageError کار میکند، پیش بینی میکنیم.

```
abs(y_pred - y_test_sample)
```

```
array([3147.4921875 , 1435.22070312,  557.80908203, 1151.53076172,
       1538.01367188])
```

شکل 51: نتایج مدل روی داده های جدید

اختلاف هر مقدار پیش بینی شده با مقدار اصلی آن به صورت بالا شده است که تخمین نسبتا خوبی را میتوان از مدل train شده برای داده های جدید به دست آورد.