

نام و نام خانوادگی	ادیب رضائی - امیرمحمد خسروی
شماره دانشجویی	810198386 - 810198401
تاریخ ارسال گزارش	۱۴۰۱.۱۲.۲۲

	<p>به نام خدا</p> <p>دانشگاه تهران</p> <p>دانشکده</p> <p>مهندسی برق و</p> <p>کامپیوتر</p>	
<p>درس شبکه‌های عصبی و یادگیری عمیق</p> <p>تمرین اول</p>		

فهرست

1 پاسخ Mcculloch-Pitts Neural Network	1
۱-۱. ماشین متناهی قطعی	3
۲ پاسخ - AdaLine & MadaLine	9
۱-۲. Adaline	9
۲-۲. MadaLine	10
۳ پاسخ - Auto-Encoders for classification	16
۱-۳. آشنایی و کار با دیتاست (پیش‌پردازش)	16
۲-۳. شبکه Auto-Encoder	17
۳-۳. طبقه‌بندی	19
۴ پاسخ - Multi-Layer Perceptron	23
۱-۴. آشنایی و کار با دیتاست (پیش‌پردازش)	23
۱-۴. Multi-Layer Perceptron	31

شکل‌ها

1	شکل 1.1. شبکه y_1
2	شکل 1.2. شبکه y_0
3	شکل 1.3. شبکه Acc
4	شکل 1.4. شبکه‌های استفاده شده
4	شکل 1.5. جدول کارنو y_1
5	شکل 1.6. شبکه y_1 (روش دوم)
6	شکل 1.7. جدول کارنو y_0
6	شکل 1.8. شبکه y_1 (روش دوم)
7	شکل 1.9. جدول کارنو Acc
7	شکل 1.10. شبکه Acc (روش دوم)
8	شکل 1.11. شبکه نهایی
8	شکل 1.12. خروجی شبکه
9	شکل 2.1. دو دسته داده
9	شکل 2.2. جدا کردن دو دسته با شبکه عصبی AdaLine
10	شکل 2.3. داده های جدید و جدا ناپذیری آن توسط AdaLine
12	شکل 2.4. الگوریتم MRI پیاده سازی شده.
13	شکل 2.5. پراکندگی دیتا بارگزاری شده
13	شکل 2.6. MadaLine با ۳ نورون
14	شکل 2.7. Madaline با ۴ نورون
14	شکل 2.8. Madaline با ۱۰ نورون
16	شکل 3.1. نمودار تعداد بر حسب گروه
16	شکل 3.2. پنج داده رندوم
17	شکل 3.3. خلاصه مدل
18	شکل 3.4. آموزش مدل
18	شکل 3.5. نمودار loss و validation loss
19	شکل 3.6. خروجی مدل
20	شکل 3.7. خلاصه مدل طبقه‌بندی
20	شکل 3.8. آموزش مدل
21	شکل 3.9. نمودارهای accuracy
21	شکل 3.10. نمودارهای loss

22	شکل 3.11. مقادیر loss و accuracy برای داده تست
22	شکل 3.12. نمودار confusion matrix
23	شکل 4.1. فراخوانی تابع info روی dataframe
24	شکل 4.2. تعداد داده های nan برای هر ستون
25	شکل 4.3. ذخیره CompanyName و به عنوان یک ستون جدا و حذف ستون های carName, car_ID, symboling
26	شکل 4.4. پیدا کردن اسامی اشتباه تایپ شده و درست کردن آنها
27	شکل 4.5. درست شده اسامی کمپانی ها
27	شکل 4.6. تبدیل داده های توصیفی به داده های عددی
28	شکل 4.7. رسم correlation matrix
29	شکل 4.8. مرتب سازی correlation فیچر ها نسبت به price. فیچر enginesize بیشترین correlation را دارد.
29	شکل 4.9. نمودار توزیع قیمت
30	شکل 4.10. نمودار قیمت بر حسب enginesize
30	شکل 4.11. تقسیم داده ها به train , test
31	شکل 4.12. اسکیل کردن داده ها و شیب های نهایی آنها
31	شکل 4.13. مدل با یک لایه
32	شکل 4.14. مدل با دو لایه مخفی
33	شکل 4.15. مدل با سه لایه مخفی
34	شکل 4.16. رفتار gradient decent
37	شکل 4.17. شبکه عصبی با یک لایه مخفی
37	شکل 4.18. شبکه عصبی با دو لایه مخفی
38	شکل 4.19. شبکه عصبی با سه لایه مخفی
38	شکل 4.20. Adam & mae
39	شکل 4.21. SGD& mse
40	شکل 4.22. Adagrad & mae
40	شکل 4.23. پیش بینی قیمت ها

جدولها

1	جدول 1. جدول انتقال حالت
---	--------------------------

پاسخ 1. شبکه عصبی Mcculloch-Pitts

۱-۱. ماشین متناهی قطعی DFA

(الف)

جدول به صورت زیر خواهد بود. همانطور که مشخص است حالت‌های ورودی و خروجی به صورت باینری نوشته شده‌اند.

جدول 1.1. جدول انتقال حالت

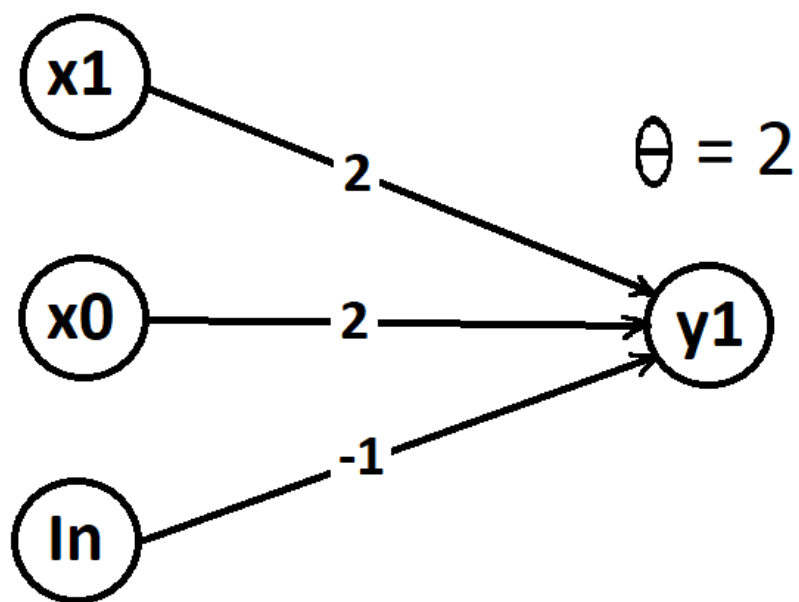
x1	x0	In	y1	y0	Acc
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	1	0	0
0	1	1	0	1	0
1	0	0	1	1	1
1	0	1	0	1	0
1	1	0	1	1	1
1	1	1	1	1	1

(ب)

برای هر خروجی شبکه را به صورت جداگانه رسم می‌کنیم. شبکه‌ها به صورت زیر خواهند بود.

شبکه برای y_1 :

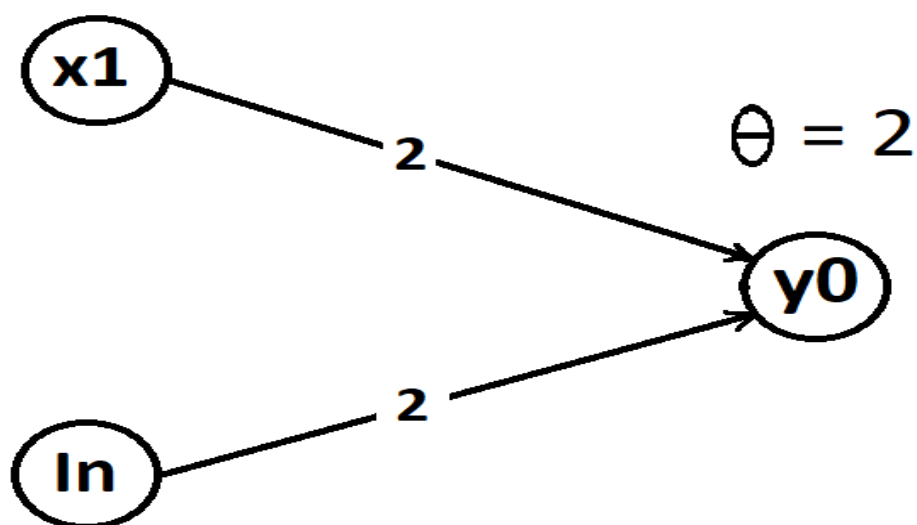
در این شبکه سه نورون ورودی و یک نورون خروجی داریم. با توجه به جدول هر زمان مقدار In یک باشد، تنها در صورتی y_1 یک می‌شود که هر دو x_1 و x_0 برابر یک باشند. ضمناً هر کدام از x_1 و x_0 اگر به تنهایی یک باشند خروجی یک خواهد بود. نتیجتاً شبکه مانند مثال بالا است که دارای threshold برابر 2 و وزن‌های 2، 2 و -1 است.



شکل 1.1. شبکه y_1

شبکه برای y_0 :

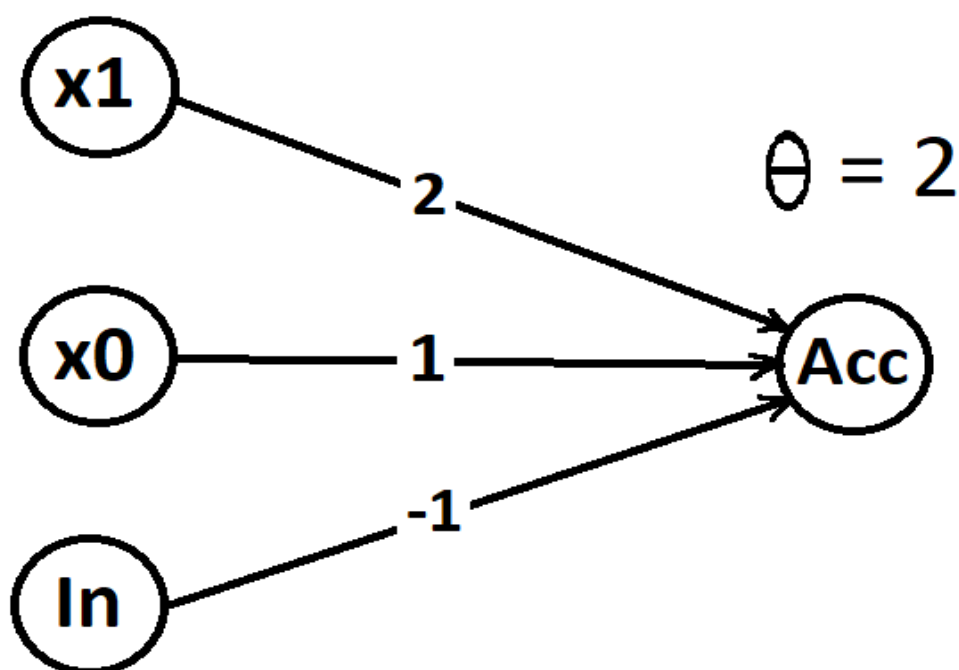
می‌توان گفت y_0 تنها به x_1 و In وابسته است و حال or کردن آنهاست. در نتیجه شبکه متناظر با آن همان شبکه or معرفی شده در اسلایدها خواهد بود. به این صورت که $threshold$ برابر با 2 و وزن‌ها نیز هر دو 2 هستند.



شکل 1.2. شبکه y_0

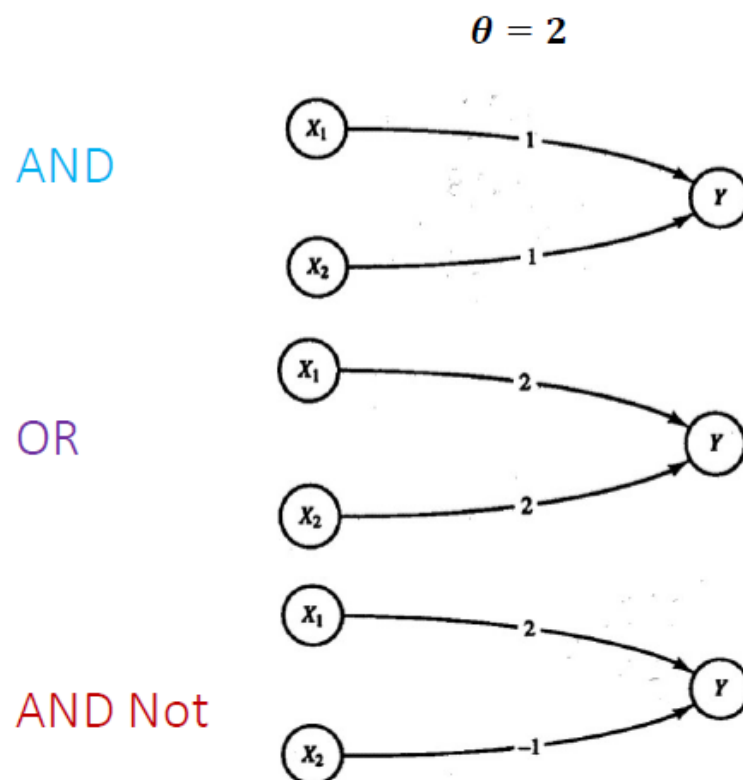
شبکه برای Acc:

در صورتی که x_1 به تنهایی یا x_1 به همراه x_0 برابر یک باشند، خروجی یک خواهد بود. در نظر گرفتن وزن 2 برای x_1 ، وزن 1 برای x_0 و وزن -1 برای ln می‌تواند شرایط گفته شده را ایجاد کند. در نتیجه شبکه برای Acc به صورت زیر خواهد بود:



شکل 1.3. شبکه Acc

یک روش دیگر برای این کار استفاده از شبکه‌های and, or و and not شده در صفحه درس است. برای پیدا کردن بهینه‌ترین شبکه‌ای که می‌توان با این شبکه‌ها ساخت می‌توان از جدول کارنو استفاده کرد. در زیر شبکه‌های استفاده شده و جداول کارنو مربوط به هر خروجی رسم شده‌اند.



شکل 1.4. شبکه‌های استفاده شده

ابتدا برای y_1 جدول کارنو را رسم میکنیم.

		$x_1 = 0$	
x_0	l_{nn}	0	1
	0	0	1
	1	0	0

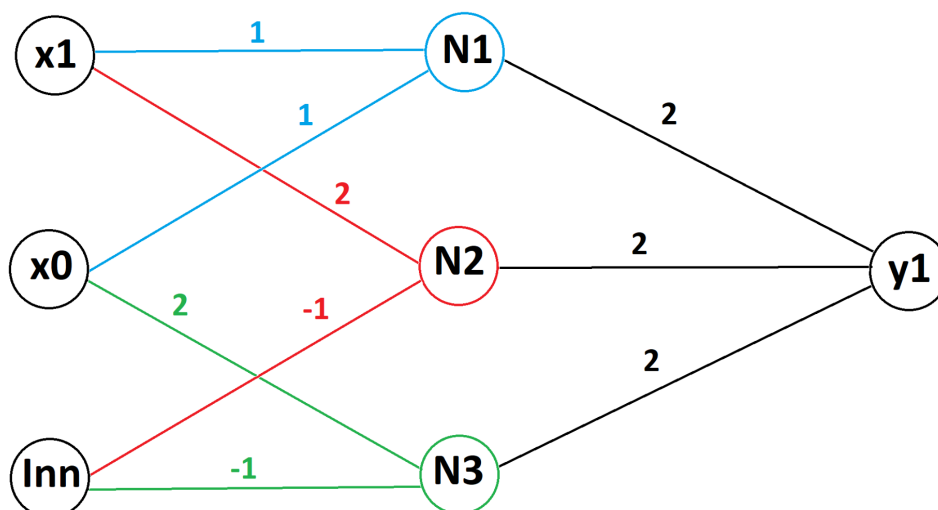
		$x_1 = 1$	
x_0	l_{nn}	0	1
	0	1	1
	1	0	1

$$y_1 = (x_1 \& \sim l_{nn}) \mid (x_1 \& x_0) \mid (x_0 \& \sim l_{nn})$$

شکل 1.5. جدول کارنو y_1

برای هر کدام از term های عبارت بالا نیاز به یک نورون داریم و در نهایت نیز برای ترکیب کردن آنها نیاز به یک نورون دیگر خواهد بود. نورون های متناظر با هر term (منظور از term یک عبارت درون پرانتز است) با رنگ مشابه با آن در زیر رسم شده است.

$$\Theta = 2$$



شکل 1.6. شبکه y_1 (روش دوم)

سه نورون $N1$ ، $N2$ ، $N3$ استفاده شده اند که هر کدام متناظر با یک term عبارت بالا است. $N1$ یک شبکه معادل and و $N2$ و $N3$ شبکه معادل and not هستند.

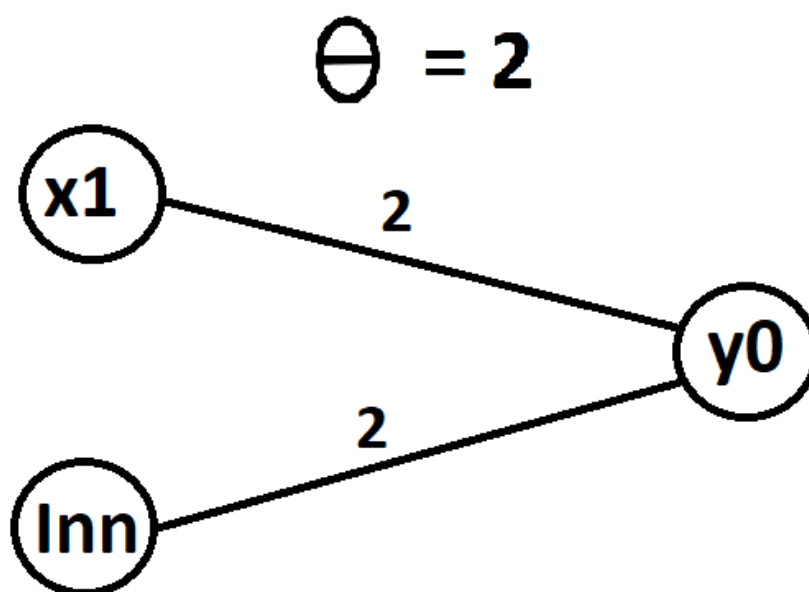
برای y_0 داریم:

		$x_1 = 0$		$x_1 = 1$	
x_0	l_{nn}	0	1	0	1
	0	0	0	1	1
	1	1	1	1	1

$$y_0 = (x_1) \mid (l_{nn}) = (x_1 \mid l_{nn})$$

شکل 1.7. جدول کارنو y_0

برای y_0 تنها نیاز به یک نورون داریم که شبکه معادل or را ایجاد کند. خواهیم داشت:



شکل 1.8. شبکه y_0 (روش دوم)

نهایتاً برای Acc خواهیم داشت:

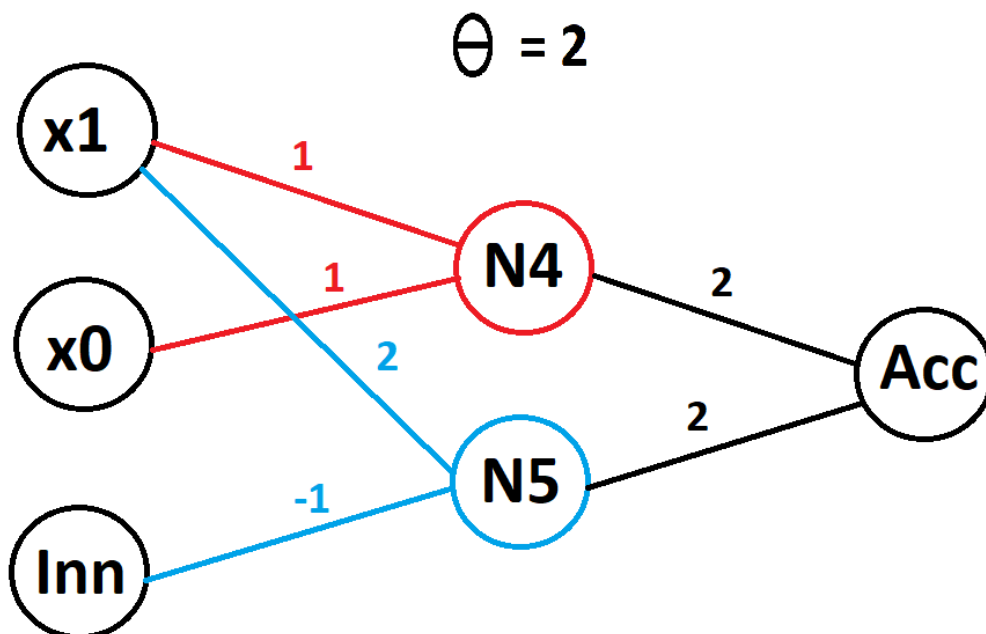
		$x1 = 0$	
Inn	$x0$	0	1
	0	0	0
	1	0	0

		$x1 = 1$	
Inn	$x0$	0	1
	0	1	1
	1	0	1

$$Acc = (x1 \& x0) | (x1 \& \sim Inn)$$

شکل 1.9. جدول کارنو Acc

برای این شبکه به یک شبکه and و یک شبکه not نیاز داریم که در شکل زیر استفاده شده‌اند تا Acc ایجاد شود.

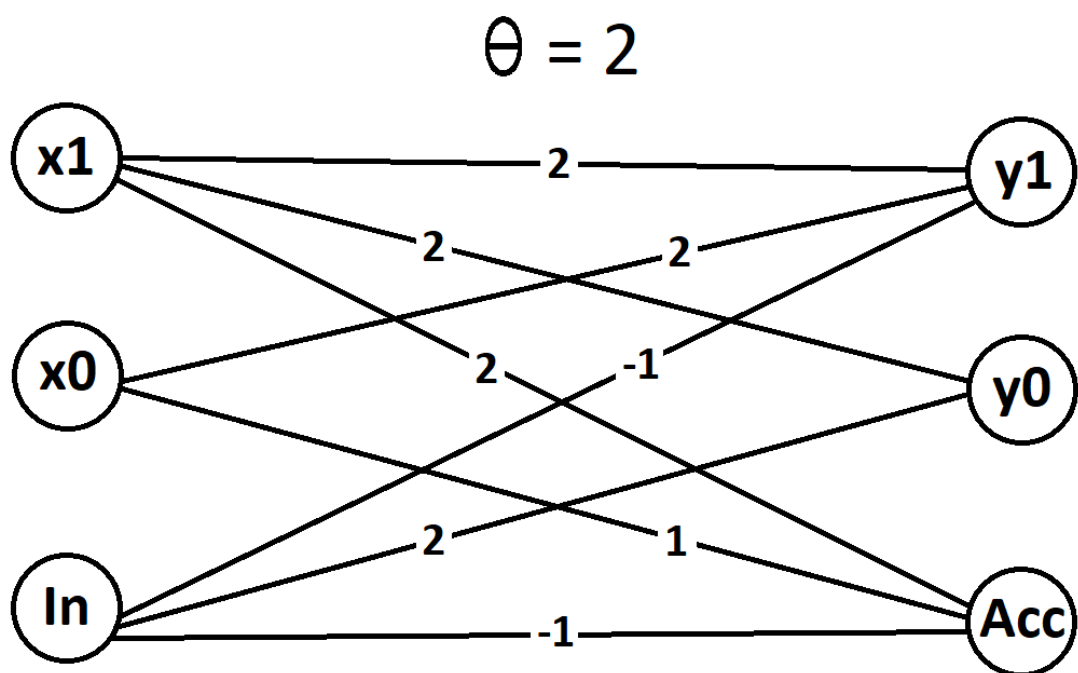


شکل 1.10. شبکه Acc (روش دوم)

(ج)

ترکیب شبکه‌های خروجی:

حال شبکه‌های بدست آمده از روش اول را با هم ترکیب و رسم می‌کنیم:



شکل 1.11. شبکه نهایی

(د)

شبکه طراحی شده در قسمت قبل را با زبان پایتون پیاده سازی می‌کنیم و همه حالات را به عنوان خروجی امتحان می‌کنیم. همانطور که در تصویر زیر دیده می‌شود، خروجی صحیح و مشابه جدول اولیه است:

✓
[25] printResult(inputs, outputs)

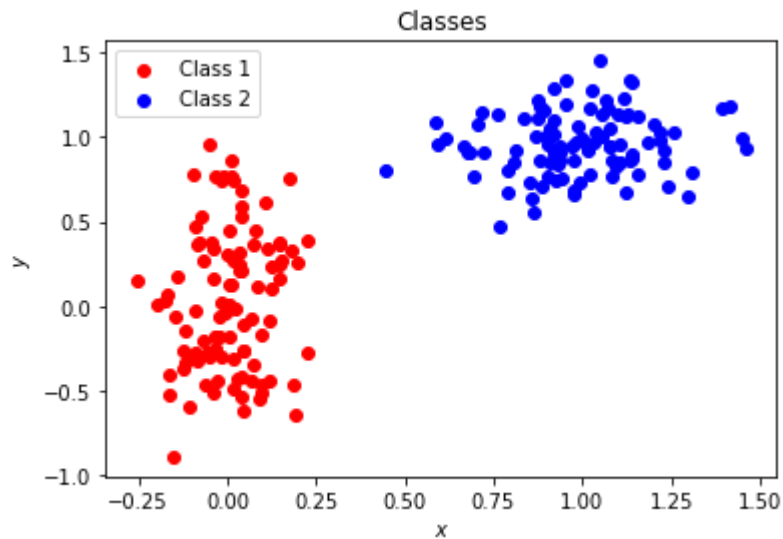
x1	x0	In		y1	y0	Acc
0	0	0		0	0	0
0	0	1		0	1	0
0	1	0		1	0	0
0	1	1		0	1	0
1	0	0		1	1	1
1	0	1		0	1	0
1	1	0		1	1	1
1	1	1		1	1	1

شکل 1.12. خروجی شبکه

پاسخ ۲ - AdaLine & MadaLine

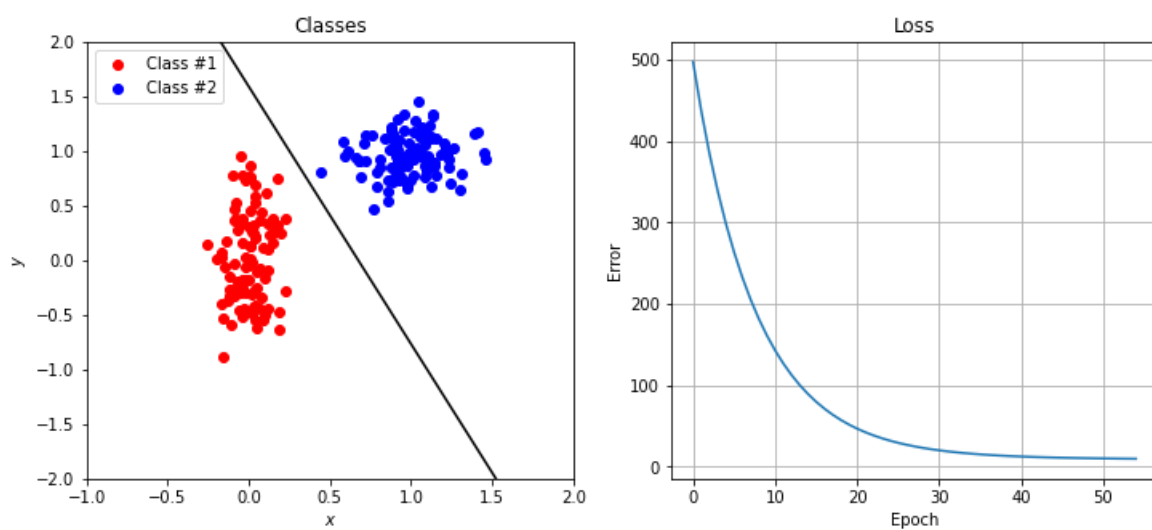
۲-۱. Adaline

(الف)



شکل ۲.۱. دو دسته داده

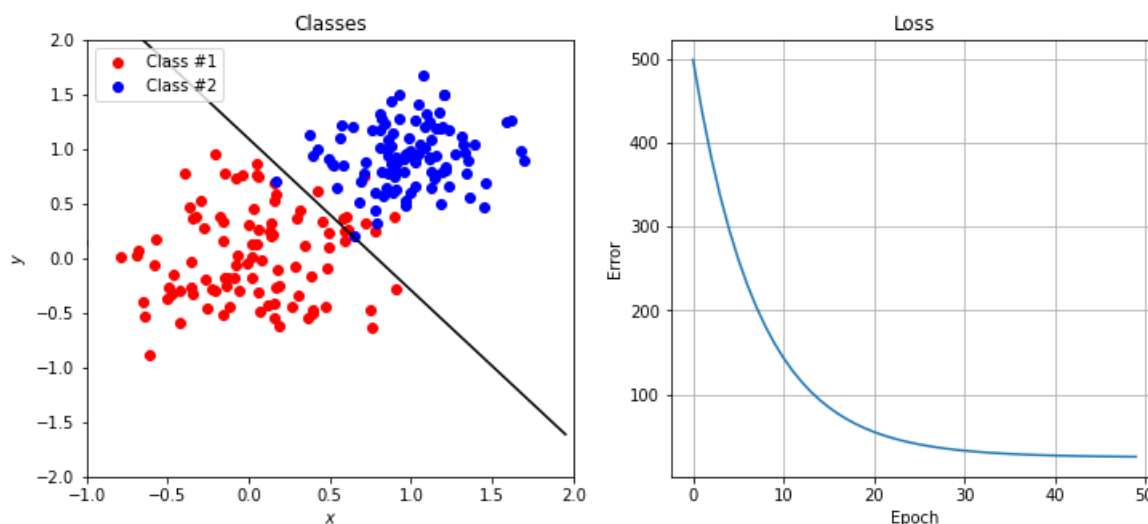
(ب)



شکل ۲.۲. جدا کردن دو دسته با شبکه عصبی AdaLine

مشاهده میشود این دو دسته به خوبی جدا شده اند. علت اصلی آن جدا پذیری این دو کلاس به صورت خطی است و شبکه عصبی AdaLine از عهده این کار بر می آید. خط رسم شده مارجین خوبی نیز دارد.

(ج)



شکل ۲.۳. داده های جدید و جدا ناپذیری آن توسط AdaLine

(د) مشاهده میشود این دو کلاس به درستی از هم جدا نشدند. علت اصلی آن جدا ناپذیری دو کلاس به صورت خطی است. این درحالی است که در بخش ج دو کلاس به طور خطی جدا پذیر بودند و خط رسم شده به درستی دو کلاس را از هم جدا کرد. اما خط رسم شده در جهت کمینه کردن ارور موجود رسم شده است و به طور تقریباً مناسبی همچنان عمل کرده است. این موضوع اما در Adaline تضمین شده نیست و ممکن است به درستی خط را رسم نکند. (در تعداد ایپاک های ثابت)

۲-۲. MadaLine

الف) روش های MRI و MRII متشکل از تعدادی نورون AdaLine است و مشابه یکدیگرند. در روش MRI وزن های لایه های پنهان تغییر میکند و وزن لایه آخر ثابت است و لایه آخر عملیات OR را انجام میدهد اما در MRII وزن تمامی لایه ها تغییر میکند.

در الگوریتم MRI زمانی وزن ها و بایاس ها آپدیت میشود که خطا رخ داده باشد. اگر خروجی مدل منفی ۱ باشد ولی target برابر ۱ باشد باید وزن نورون هایی که نزدیک تر به صفر هستند را آپدیت کرد تا ۱ شوند و در عملیات OR خروجی اصلاح شود همچنین اگر خروجی مدل ۱ باشد ولی target منفی ۱ باشد آنگاه باید وزن همه نورون های لایه مخفی با خروجی مثبت را آپدیت کنیم تا خروجی این نورون ها کمتر از صفر شوند و به منفی ۱ برسیم. قدم های پیاده سازی به شکل زیر است:

- 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:

$$z_in_1 = b_1 + x_1w_{11} + x_2w_{21},$$

$$z_in_2 = b_2 + x_1w_{12} + x_2w_{22}.$$
- 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:

$$y_in = b_3 + z_1v_1 + z_2v_2;$$

$$y = f(y_in).$$
- 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,

$$b_j(\text{new}) = b_j(\text{old}) + \alpha(1 - z_in_j),$$

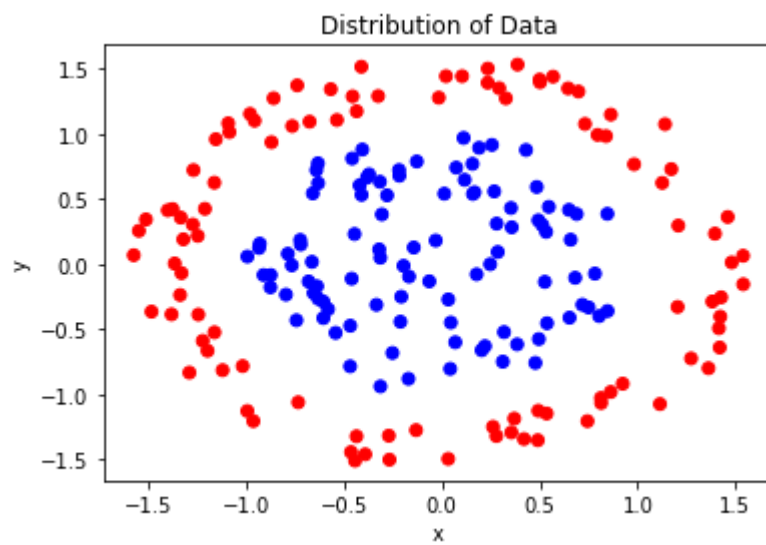
$$w_{ij}(\text{new}) = w_{ij}(\text{old}) + \alpha(1 - z_in_j)x_i;$$
 If $t = -1$, then update weights on all units Z_k that have positive net input,

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

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

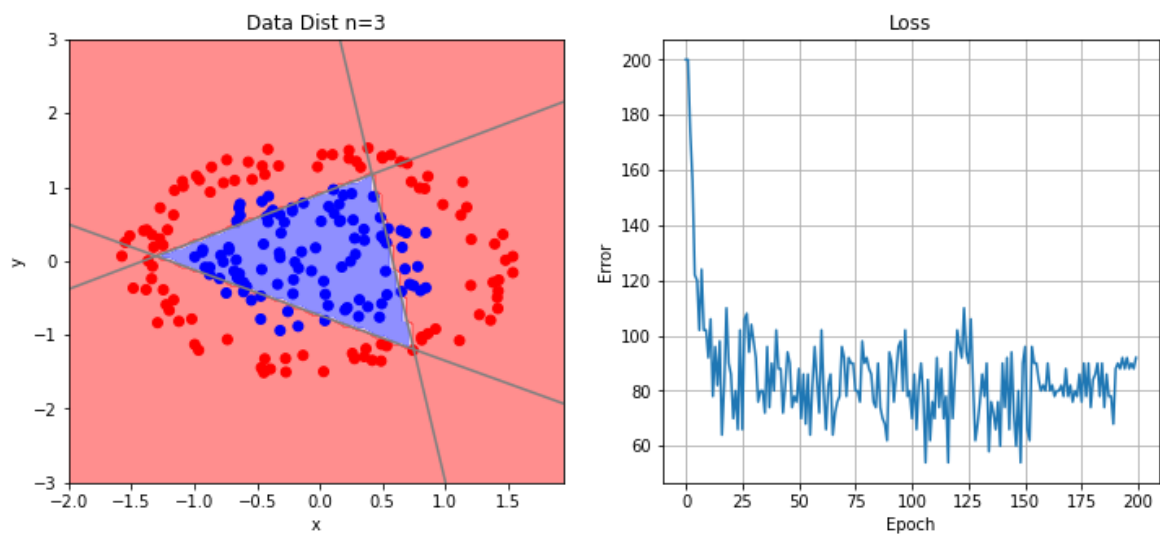
شکل ۲.۴. الگوریتم MRI پیاده سازی شده.

(ب)

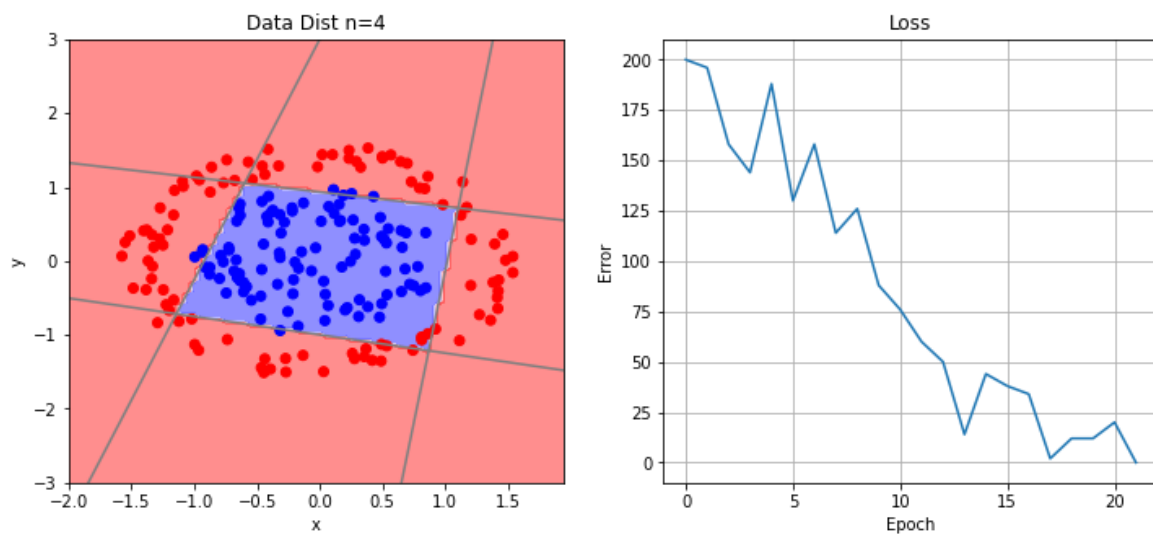


شکل ۲.۵ پراکندگی دیتا بارگزاری شده

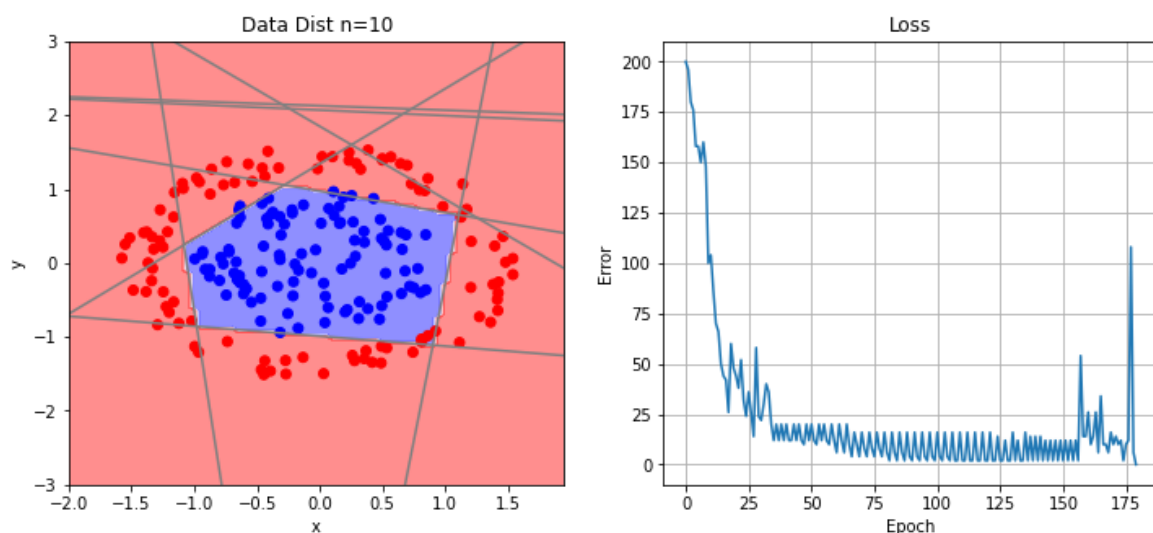
(ج)



شکل ۲.۶. MadaLine با ۳ نورون



شکل ۲.۷. Madaline با ۴ نورون



شکل ۲.۸. Madaline با ۱۰ نورون

در شکل ۲.۶ که شبکه MadaLine با ۳ نورون سعی بر جدا کردن دیتا کرده ملاحظه میشود ارور نهایی شبکه زیاد است. این به این علت است که حداقل تعداد خطوطی که نیاز است تا دو کلاس از هم جدا شوند بیشتر از ۳ است.

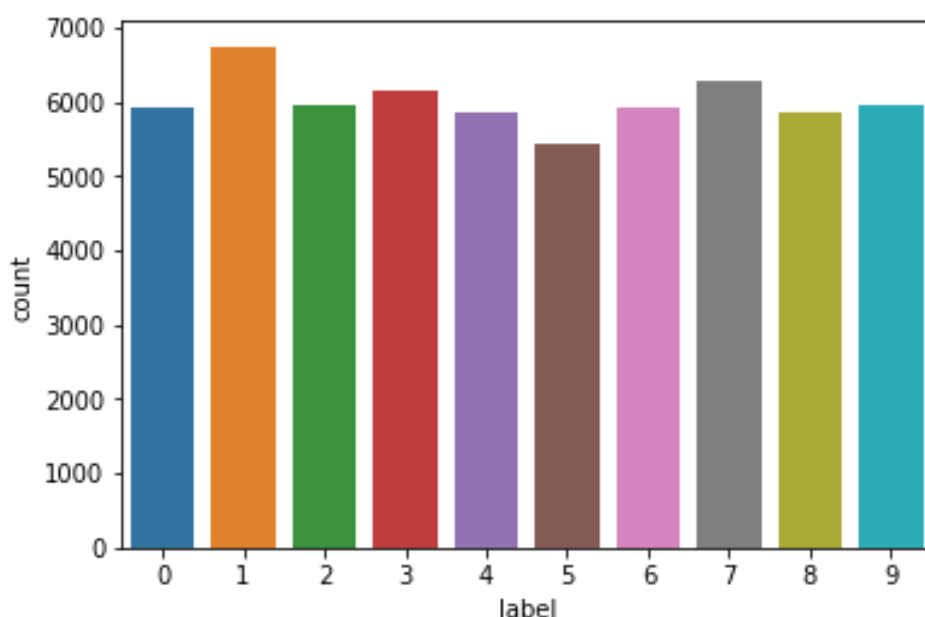
در شکل ۲.۷ ارور کمینه شده است و با دقت خوبی دو کلاس را با یک چهار ضلعی جدا کرده است.

در شکل ۲.۸ ۱۰ نورون بیشتر از حد نیاز بوده و یک سری خطوط بی ربط در جدایی این دو کلاس در شکل دیده میشود. شبکه نتوانسته است در این تعداد ایپاک جایگاه مناسبی برای این خطوط پیدا کند اما همچنان ارور تا حد خوبی کاهش یافته است اما پر نویز است.

پاسخ ۳ – Auto-Encoders for classification

۱-۳. آشنایی و کار با دیتاست (پیش‌پردازش)

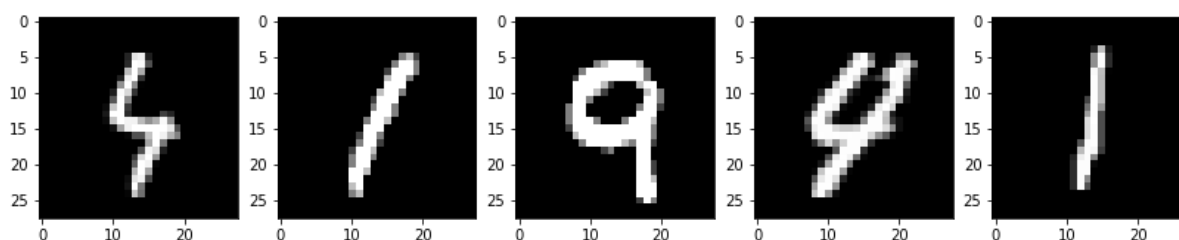
ابتدا با دستورات داده شده دیتاست مذکور را لود میکنیم. سپس با استفاده از labelهای داده‌های train نمودار تعداد را بر حسب گروه رسم میکنیم. نتیجه به صورت زیر است:



شکل 3.1. نمودار تعداد بر حسب گروه

همانطور که مشاهده میشود labelها تقریباً تعداد یکسانی دارند به این معنا که داده‌ها تنوع مطلوبی دارند.

حال 5 داده را به صورت رندوم رسم می‌کنیم. نتیجه:



شکل 3.2. پنج داده رندوم

سپس داده‌های را نرمالیز و آماده می‌کنیم تا در مراحل بعد استفاده شوند. ویژگی‌ها همان پیکسل‌ها خواهند بود.

۲-۳. شبکه Auto-Encoder

شبکه Auto-Encoder با معماری داده شده ایجاد می‌کنیم. برای بخش Encoder و Decoder به ترتیب از Relu و Sigmoid به عنوان Activation Function استفاده می‌کنیم.

در اینجا اندازه لایه‌هایی که به صورت optional در صورت پرسش داده شده‌اند را 300 در نظر می‌گیریم که میانگین 100 و 500 (اندازه FC‌های مجاور) است. در کل یک لایه ورودی، 4 لایه encode و همچنین 4 لایه decode خواهیم داشت که لایه آخر decode همان لایه خروجی است.

از Adam optimizer و همچنین برای loss از mse استفاده می‌کنیم و Auto-Encoder را compile می‌کنیم. summary کامپایل کردن مدل به صورت زیر خواهد بود:

```
Model: "model_11"
```

Layer (type)	Output Shape	Param #
input_6 (InputLayer)	[None, 784]	0
encoded_layer1 (Dense)	(None, 500)	392500
encoded_layer2 (Dense)	(None, 300)	150300
encoded_layer3 (Dense)	(None, 100)	30100
coded_layer (Dense)	(None, 30)	3030
dedoded_layer1 (Dense)	(None, 100)	3100
dedoded_layer2 (Dense)	(None, 300)	30300
dedoded_layer3 (Dense)	(None, 500)	150500
output_layer (Dense)	(None, 784)	392784

```

=====
Total params: 1,152,614
Trainable params: 1,152,614
Non-trainable params: 0
None

```

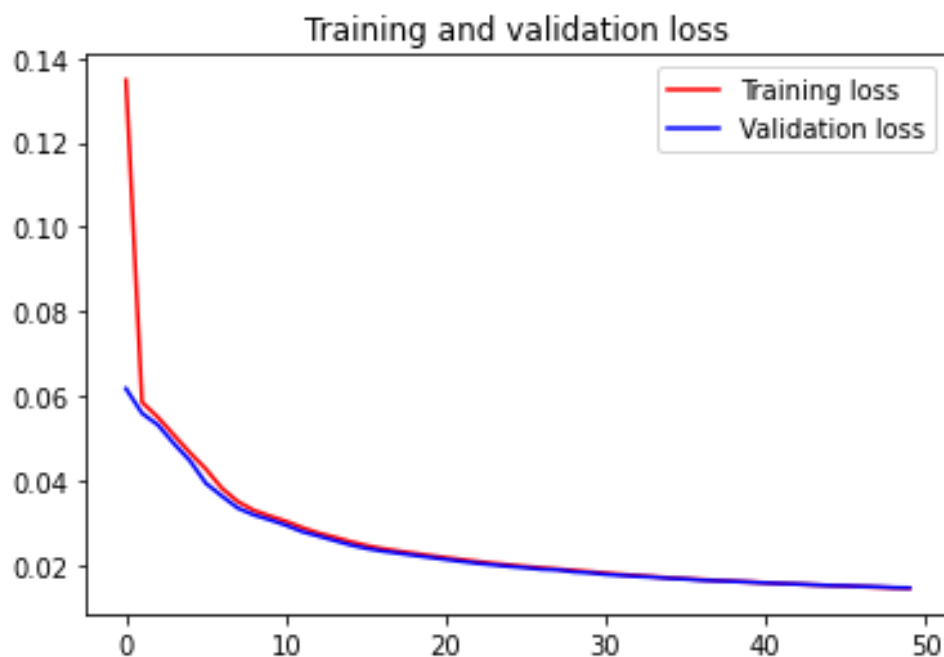
شکل 3.3. خلاصه مدل

نهایتاً مدل را با تعداد epoch برابر 50 و batch size برابر 256، train می‌کنیم. همانطور که در تصویر زیر مشاهده می‌شود، loss و val_loss به مرور کاهش می‌یابند:

```
Epoch 1/50
235/235 [=====] - 16s 60ms/step - loss: 0.1348 - val_loss: 0.0617
Epoch 2/50
235/235 [=====] - 13s 56ms/step - loss: 0.0585 - val_loss: 0.0558
Epoch 3/50
235/235 [=====] - 14s 59ms/step - loss: 0.0549 - val_loss: 0.0531
Epoch 4/50
235/235 [=====] - 14s 61ms/step - loss: 0.0507 - val_loss: 0.0486
Epoch 5/50
235/235 [=====] - 19s 80ms/step - loss: 0.0465 - val_loss: 0.0446
Epoch 6/50
235/235 [=====] - 22s 95ms/step - loss: 0.0426 - val_loss: 0.0392
Epoch 7/50
235/235 [=====] - 20s 84ms/step - loss: 0.0381 - val_loss: 0.0362
Epoch 8/50
235/235 [=====] - 17s 75ms/step - loss: 0.0349 - val_loss: 0.0334
Epoch 9/50
235/235 [=====] - 17s 72ms/step - loss: 0.0328 - val_loss: 0.0318
Epoch 10/50
235/235 [=====] - 19s 83ms/step - loss: 0.0315 - val_loss: 0.0306
Epoch 11/50
235/235 [=====] - 15s 64ms/step - loss: 0.0302 - val_loss: 0.0293
Epoch 12/50
235/235 [=====] - 12s 52ms/step - loss: 0.0288 - val_loss: 0.0278
Epoch 13/50
235/235 [=====] - 14s 61ms/step - loss: 0.0275 - val_loss: 0.0268
Epoch 14/50
235/235 [=====] - 13s 53ms/step - loss: 0.0265 - val_loss: 0.0257
Epoch 15/50
235/235 [=====] - 12s 53ms/step - loss: 0.0255 - val_loss: 0.0247
Epoch 16/50
235/235 [=====] - 12s 53ms/step - loss: 0.0245 - val_loss: 0.0238
```

شکل 3.4. آموزش مدل

اکنون نمودارهای loss و validation loss را رسم می‌کنیم. نتیجه به صورت زیر است:



شکل 3.5. نمودار loss و validation loss

حال از مدل استفاده می‌کنیم تا عکس‌های train و test را ابتدا encode و سپس decode کنیم. چند نمونه از عکس ورودی و خروجی Auto-Encoder متناظر با آن در ادامه نشان داده شده‌است.



شکل 3.6. خروجی مدل

همانطور که مشاهده می‌شود Auto-Encoder به خوبی عمل کرده و عکس‌ها را به نمونه‌های بهتری برای شبکه عصبی تبدیل می‌کند. از عکس‌های اولیه اطلاعات اضافه و ابعاد غیر ضروری حذف شده‌است (برای مثال اعداد 3 و 9 شکل بالا)

۳-۳. طبقه‌بندی

ابتدا قسمت encoder مدل Auto-Encoder ای که ساخته بودیم را جدا می‌کنیم و خروجی آن را به عنوان ورودی Classifier در نظر می‌گیریم. سپس فرمت labelها را به صورت one-hot در می‌آوریم. فرمت labelها به صورت زیر خواهد بود:

```
Original label: 5
```

```
After conversion to one-hot: [0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
```

حال داده‌های train را با نسبت 8 به 2 به دو قسمت train و validation تقسیم می‌کنیم. سپس مدلی Sequential با دو لایه مخفی که هردو از relu به عنوان activation function استفاده می‌کنند train می‌کنیم. لایه آخر نیز از تابع softmax استفاده خواهد کرد. summary مدلی که ساخته‌ایم به صورت زیر خواهد بود:

Model: "sequential_10"

Layer (type)	Output Shape	Param #
hidden_layer1 (Dense)	(None, 40)	1240
hidden_layer2 (Dense)	(None, 20)	820
output_layer (Dense)	(None, 10)	210

=====
Total params: 2,270
Trainable params: 2,270
Non-trainable params: 0
=====

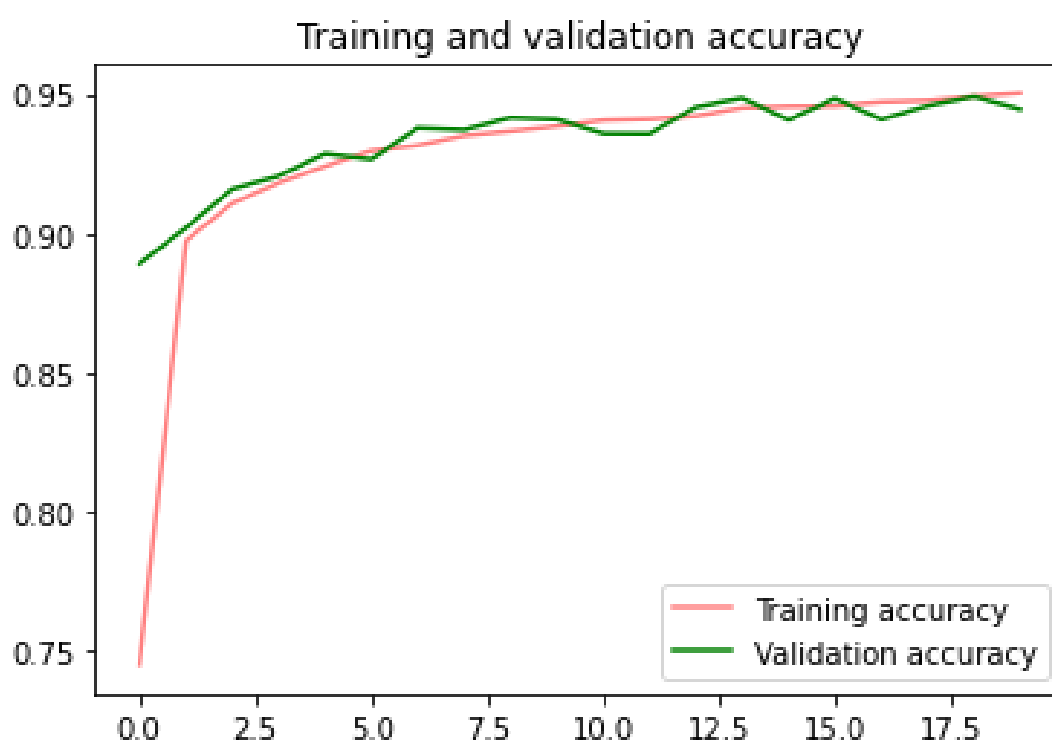
شکل 3.7. خلاصه مدل طبقه‌بندی

اکنون مدل ساخته شده را compile و سپس train می‌کنیم. تعداد batchها و epochها به ترتیب 64 و 20 است. همانطور که در شکل زیر مشاهده می‌شوند به مرور loss کاهش یافته و accuracy افزایش می‌یابد:

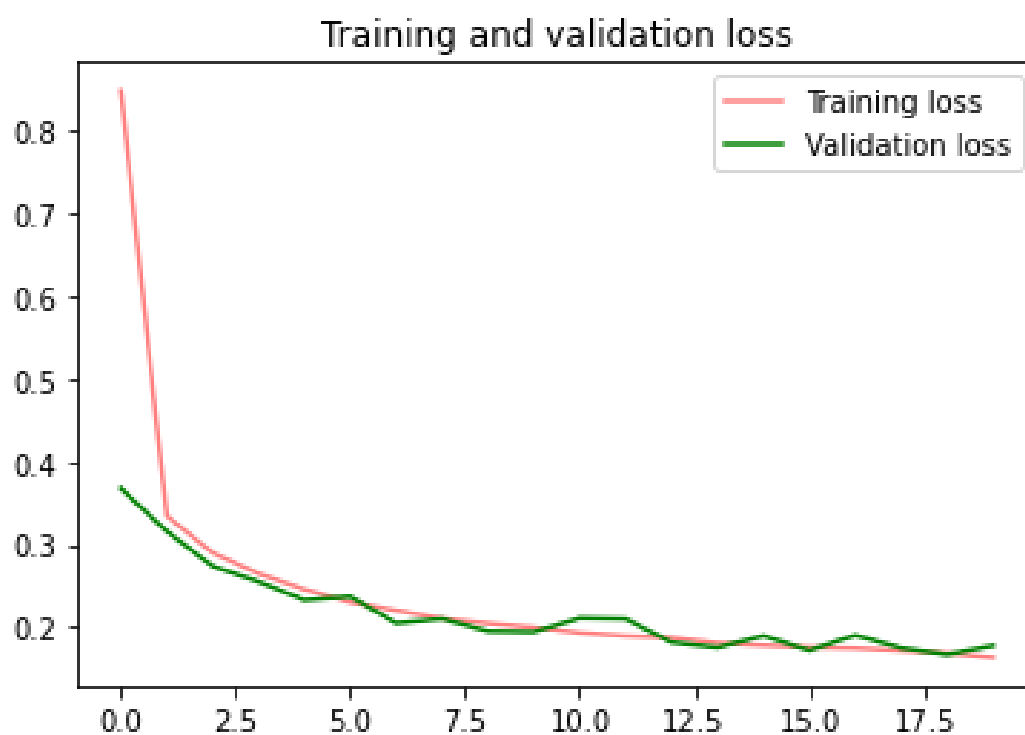
Epoch 1/20	750/750 [=====] - 4s 4ms/step - loss: 0.8489 - accuracy: 0.7454 - val_loss: 0.3677 - val_accuracy: 0.8892
Epoch 2/20	750/750 [=====] - 2s 3ms/step - loss: 0.3344 - accuracy: 0.8974 - val_loss: 0.3159 - val_accuracy: 0.9021
Epoch 3/20	750/750 [=====] - 2s 3ms/step - loss: 0.2898 - accuracy: 0.9109 - val_loss: 0.2736 - val_accuracy: 0.9160
Epoch 4/20	750/750 [=====] - 2s 3ms/step - loss: 0.2647 - accuracy: 0.9182 - val_loss: 0.2550 - val_accuracy: 0.9207
Epoch 5/20	750/750 [=====] - 2s 3ms/step - loss: 0.2457 - accuracy: 0.9241 - val_loss: 0.2327 - val_accuracy: 0.9287
Epoch 6/20	750/750 [=====] - 2s 3ms/step - loss: 0.2301 - accuracy: 0.9297 - val_loss: 0.2368 - val_accuracy: 0.9267
Epoch 7/20	750/750 [=====] - 3s 4ms/step - loss: 0.2200 - accuracy: 0.9317 - val_loss: 0.2048 - val_accuracy: 0.9379
Epoch 8/20	750/750 [=====] - 2s 3ms/step - loss: 0.2103 - accuracy: 0.9351 - val_loss: 0.2100 - val_accuracy: 0.9373
Epoch 9/20	750/750 [=====] - 2s 3ms/step - loss: 0.2044 - accuracy: 0.9367 - val_loss: 0.1951 - val_accuracy: 0.9416
Epoch 10/20	750/750 [=====] - 2s 3ms/step - loss: 0.1996 - accuracy: 0.9385 - val_loss: 0.1943 - val_accuracy: 0.9410
Epoch 11/20	750/750 [=====] - 2s 3ms/step - loss: 0.1931 - accuracy: 0.9406 - val_loss: 0.2107 - val_accuracy: 0.9359
Epoch 12/20	750/750 [=====] - 2s 3ms/step - loss: 0.1893 - accuracy: 0.9411 - val_loss: 0.2099 - val_accuracy: 0.9358
Epoch 13/20	750/750 [=====] - 3s 4ms/step - loss: 0.1873 - accuracy: 0.9421 - val_loss: 0.1816 - val_accuracy: 0.9453
Epoch 14/20	750/750 [=====] - 2s 3ms/step - loss: 0.1807 - accuracy: 0.9451 - val_loss: 0.1754 - val_accuracy: 0.9486
Epoch 15/20	

شکل 3.8. آموزش مدل

نمودارهای Loss، Validation Accuracy، Validation Loss و Accuracy به صورت زیر است:



شکل 3.9. نمودارهای accuracy



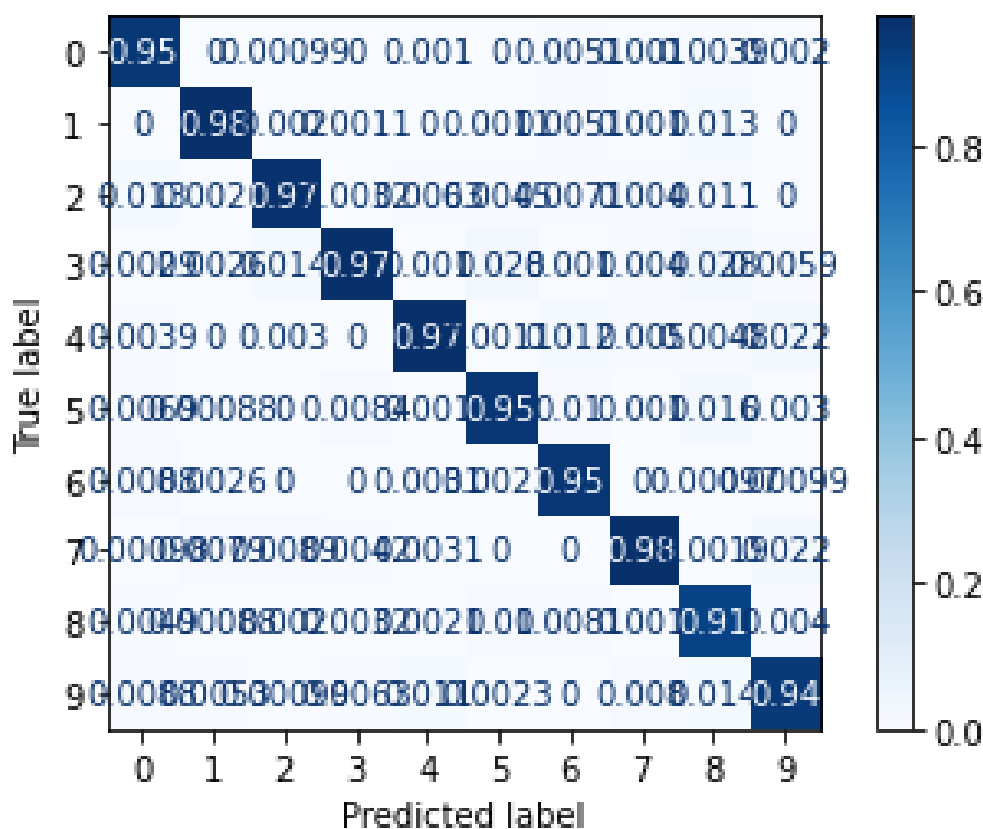
شکل 3.10. نمودارهای loss

دقت مدل روی داده‌های تست به صورت زیر است: (عدد سمت چپ loss و عدد سمت راست accuracy است)

[0.1823349893093109, 0.9417999982833862]

شکل 3.11. مقادیر loss و accuracy برای داده تست

نمودار confusion matrix به صورت زیر خواهد بود. همانطور که مشخص است با تقریب بالای 94 درصد همه اعداد درست تشخیص داده شده‌اند. اعدادی مثل 1، 2 و 7 آسانتر تشخیص داده شده‌اند زیرا درصد بیشتری از label‌های پیش‌بینی شده آن با label‌های اصلی مطابق است. از طرفی قابل درک نیز هست زیرا به سبب شکل متفاوتشان به ندرت با اعداد دیگر اشتباه گرفته میشوند. اما عددی مانند 8 درصد درست بودن label‌های پیش‌بینی شده آن برابر 91 درصد است که از دیگر اعداد مقدار کمتری است. به این معنا که تشخیص این عدد مقداری چالش برانگیز است. مجدداً می‌توان استدلال کرد دلیل این موضوع تشابه عدد 8 به اعداد 9، 0 و 6 است و اندکی محو شدن بخشی از این عدد باعث اشتباه گرفته شدن آن با این اعداد خواهد شد.



شکل 3.12. نمودار confusion matrix

پاسخ ۴ – Multi-Layer Perceptron

۴-۱. آشنایی و کار با دیتاست (پیش‌پردازش)

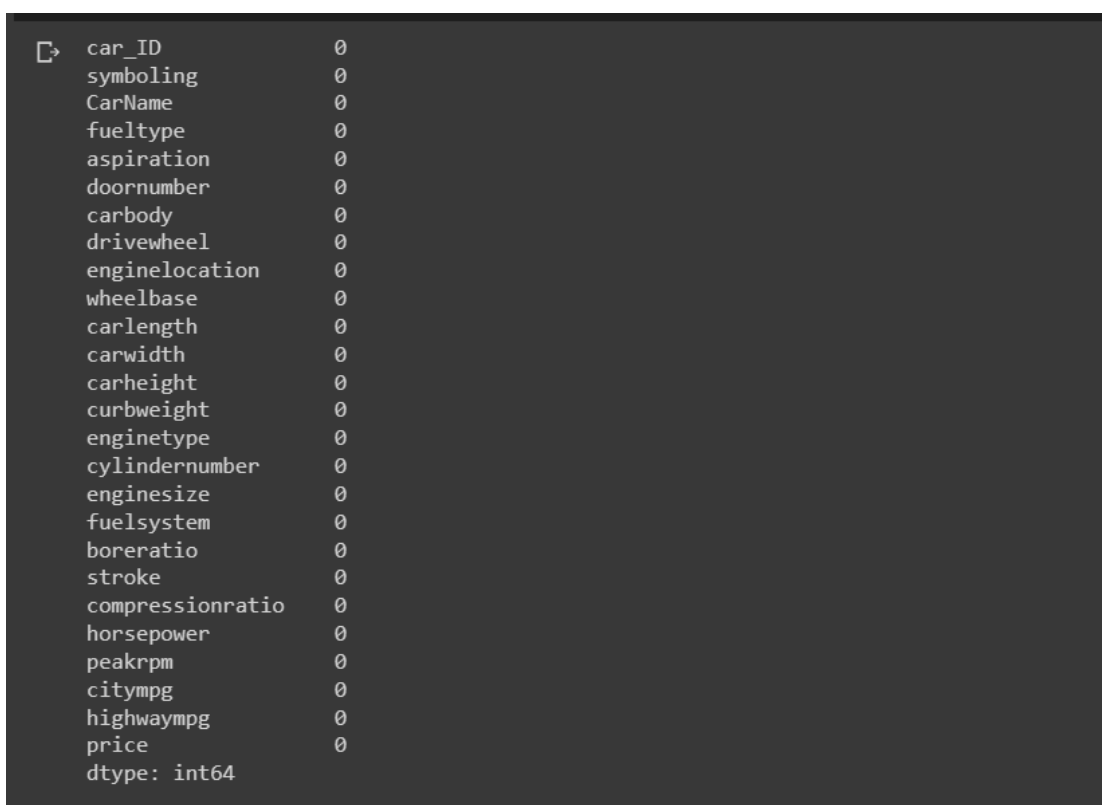
۱.

```
<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 ۴.۱. فراخوانی تابع info روی dataframe

همانطور که می‌بینیم، 26 ستون یا feature داریم که همه ستون‌ها با مقادیر غیر null پر شده‌اند. بعضی از ویژگی‌ها دارای تایپ int، بعضی float و بعضی دیگر object هستند. تعداد کل سطرها نیز 205 است.

۲.



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 ۴.۲. تعداد داده‌های nan برای هر ستون

```
[ ] new_cars_df = cars_df.copy()
new_cars_df[['CompanyName', 'CarName']] = cars_df['CarName'].str.split(' ', 1, expand=True)
new_cars_df.iloc[58]
```

```
car_ID          59
symboling       3
CarName         glc 4
fueltype        gas
aspiration      std
doornumber      two
carbody         hatchback
drivewheel      rwd
engineLocation  front
wheelbase       95.3
carlength       169.0
carwidth        65.7
carheight       49.6
curbweight      2500
enginetype      rotor
cylindernumber  two
enginesize       80
fuelsystem      mpfi
boreRatio       3.33
stroke          3.255
compressionratio 9.4
horsepower      135
peakrpm         6000
citympg         16
highwaympg      23
price           15645.0
CompanyName      mazda
Name: 58, dtype: object
```

Deleting CarName, car_ID and symboling columns:

```
[ ] new_cars_df.drop(['CarName', 'car_ID', 'symboling'], axis=1, inplace=True)
```

شکل ۴.۳. ذخیره CompanyName و به عنوان یک ستون جدا و حذف ستون های carName, car_ID, symboling

Finding wrong company names:

```
[ ] set(new_cars_df['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'}
```

As we can see we have these wrong words:

```
maxda, porcshce, toyouta, volkswagen
```

They have to be replaced with:

```
mazda, porsche, toyota, volkswagen
```

شکل ۴.۴. پیدا کردن اسامی اشتباه تایپ شده و درست کردن آنها

Now we replace them:

```
[ ] new_cars_df.replace({'CompanyName' : { 'maxda' : 'mazda',
                                             'porcshce' : 'porsche',
                                             'toyouta' : 'toyota',
                                             'vokswagen' : 'volkswagen' }}, inplace = True)

set(new_cars_df['CompanyName'])

{'Nissan',
 '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 ۴.۵. درست شده اسامی کمپانی ها

۴.

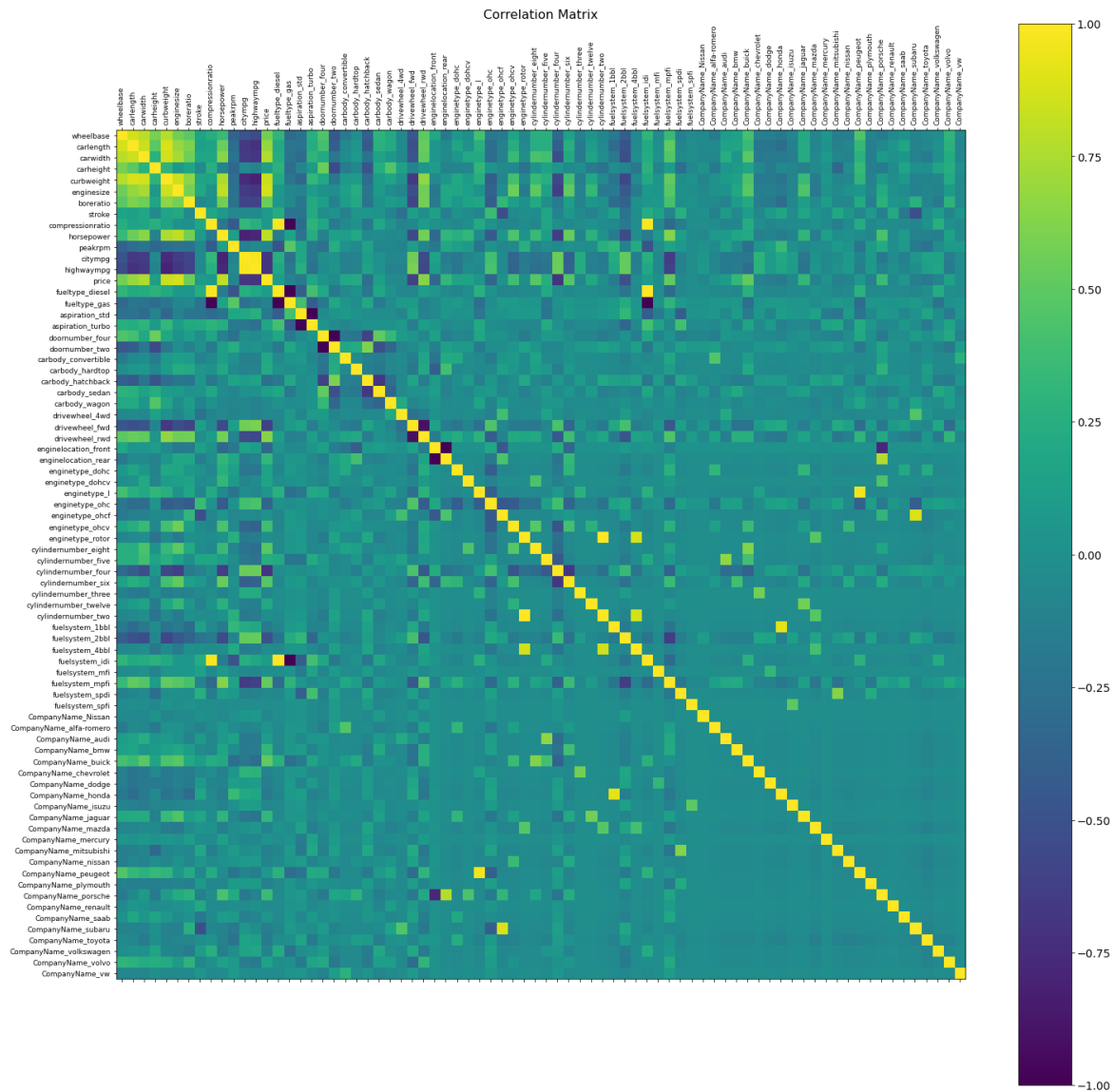
```
new_cars_df = pd.get_dummies(new_cars_df)
new_cars_df.head()
```

	wheelbase	carlength	carwidth	carheight	curbweight	enginesize	boreratio	stroke	compressionratio	horsepower	...
0	88.6	168.8	64.1	48.8	2548	130	3.47	2.68	9.0	111	...
1	88.6	168.8	64.1	48.8	2548	130	3.47	2.68	9.0	111	...
2	94.5	171.2	65.5	52.4	2823	152	2.68	3.47	9.0	154	...
3	99.8	176.6	66.2	54.3	2337	109	3.19	3.40	10.0	102	...
4	99.4	176.6	66.4	54.3	2824	136	3.19	3.40	8.0	115	...

5 rows × 76 columns

شکل ۴.۶. تبدیل داده های توصیفی به داده های عددی

٥.



شکل ٤.٧. رسم correlation matrix

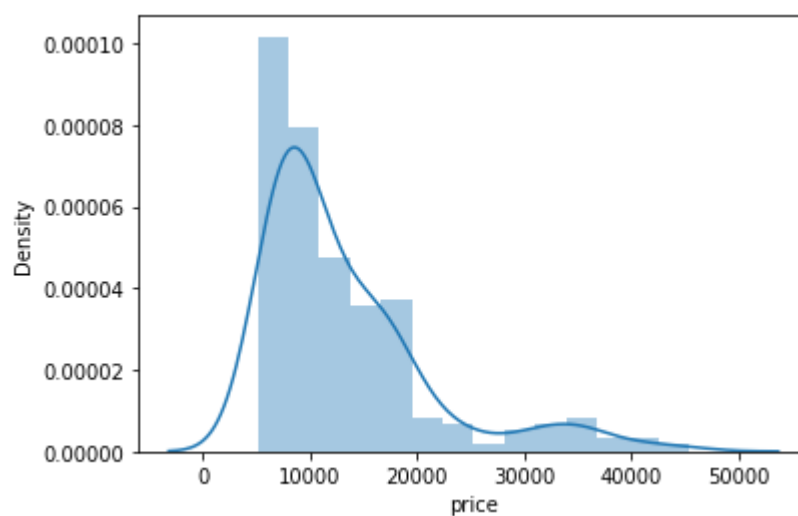
We see that **enginesize** is the most correlated among other features.

```
[ ] new_cars_df.corr().price.sort_values(ascending=False)
```

```
price            1.000000
enginesize       0.874145
curbweight       0.835305
horsepower       0.808139
carwidth         0.759325
...
fuelsystem_2bbl -0.501374
drivewheel_fwd  -0.601950
citympg          -0.685751
highwaympg       -0.697599
cylindernumber_four -0.697762
Name: price, Length: 76, dtype: float64
```

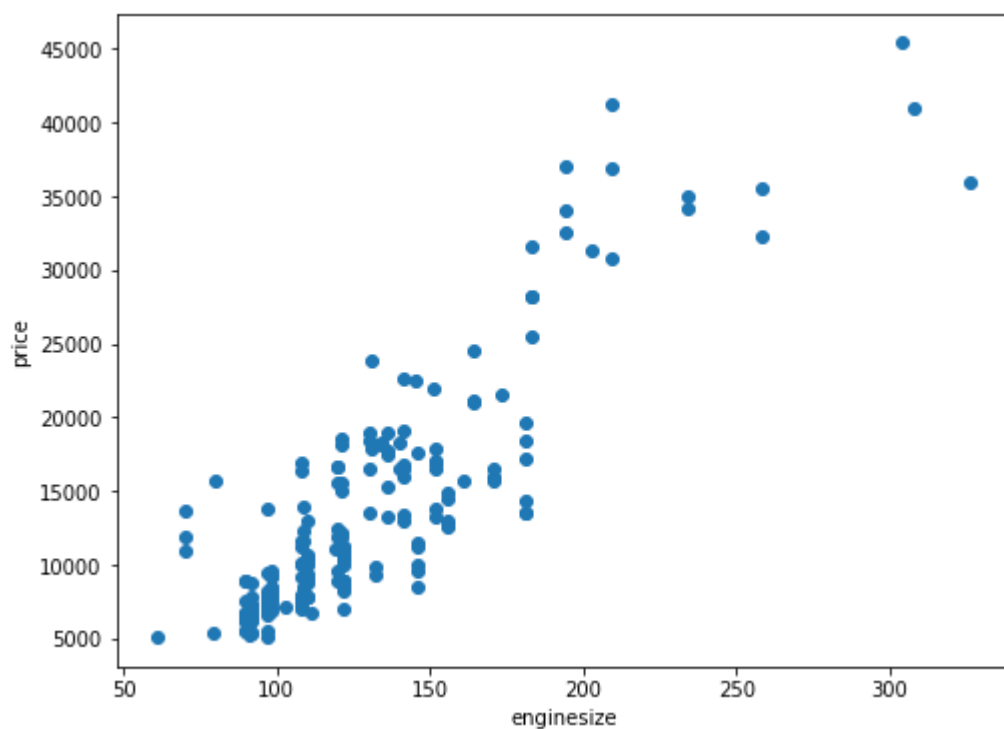
شکل ۴.۸. مرتب سازی correlation فیچر ها نسبت به price. فیچر enginesize بیشترین correlation را دارد.

۶.



شکل ۴.۹. نمودار توزیع قیمت

مشاهده میشود توزیع قیمت منحنی نرمال با چوله چپ است.



شکل ۴.۱۰. نمودار قیمت بر حسب engine size

مشاهده میشود correlation خیلی خوبی با price دارد و تقریباً روی خط $y=x$ قرار میگیرند.

۷.

```
k_train, x_test, y_train, y_test = train_test_split(df.drop('price', axis=1), y, test_size=0.15, random_state=100)
```

شکل ۴.۱۱. تقسیم داده ها به train , test

۸.

```
x_scaler = MinMaxScaler()
x_train = x_scaler.fit_transform(x_train)
x_test = x_scaler.transform(x_test)

x_train.shape, x_test.shape, y_train.shape, y_test.shape

((139, 75), (31, 75), (139,)), (31,)), (35, 75), (35,))
```

شکل ۴.۱۲. اسکیل کردن داده ها و شیب های نهایی آنها

۴-۱. Multi-Layer Perceptron

۱.

```
#model with one layer

model = Sequential()

model.add(Dense(HIDDEN_LAYER_SIZE, activation='relu', input_shape=(75,)))
model.add(Dense(1, activation='relu'))

model.compile(optimizer='adam', loss='mse', metrics=(coeff_determination,))
model.summary()
```

Model: "sequential_5"

Layer (type)	Output Shape	Param #
dense_10 (Dense)	(None, 512)	38912
dense_11 (Dense)	(None, 1)	513

```
=====
Total params: 39,425
Trainable params: 39,425
Non-trainable params: 0
```

شکل ۴.۱۳. مدل با یک لایه

```
#model with two layer

model = Sequential()

model.add(Dense(HIDDEN_LAYER_SIZE, activation='relu', input_shape=(75,)))
model.add(Dense(HIDDEN_LAYER_SIZE, activation='relu'))
model.add(Dense(1, activation='relu'))

model.compile(optimizer='adam', loss='mse', metrics=(coeff_determination,))
model.summary()
```

Model: "sequential_6"

Layer (type)	Output Shape	Param #
dense_12 (Dense)	(None, 512)	38912
dense_13 (Dense)	(None, 512)	262656
dense_14 (Dense)	(None, 1)	513

```
=====
Total params: 302,081
Trainable params: 302,081
Non-trainable params: 0
=====
```

شکل ۴.۱۴. مدل با دو لایه مخفی

```
#model with three layer

model = Sequential()

model.add(Dense(HIDDEN_LAYER_SIZE, activation='relu', input_shape=(75,)))
model.add(Dense(HIDDEN_LAYER_SIZE, activation='relu'))
model.add(Dense(HIDDEN_LAYER_SIZE, activation='relu'))
model.add(Dense(1, activation='relu'))

model.compile(optimizer='adam', loss='mse', metrics=(coeff_determination,))
model.summary()
```

Model: "sequential_7"

Layer (type)	Output Shape	Param #
dense_15 (Dense)	(None, 512)	38912
dense_16 (Dense)	(None, 512)	262656
dense_17 (Dense)	(None, 512)	262656
dense_18 (Dense)	(None, 1)	513
Total params: 564,737		
Trainable params: 564,737		
Non-trainable params: 0		

شکل ۴.۱۵. مدل با سه لایه مخفی

۲.

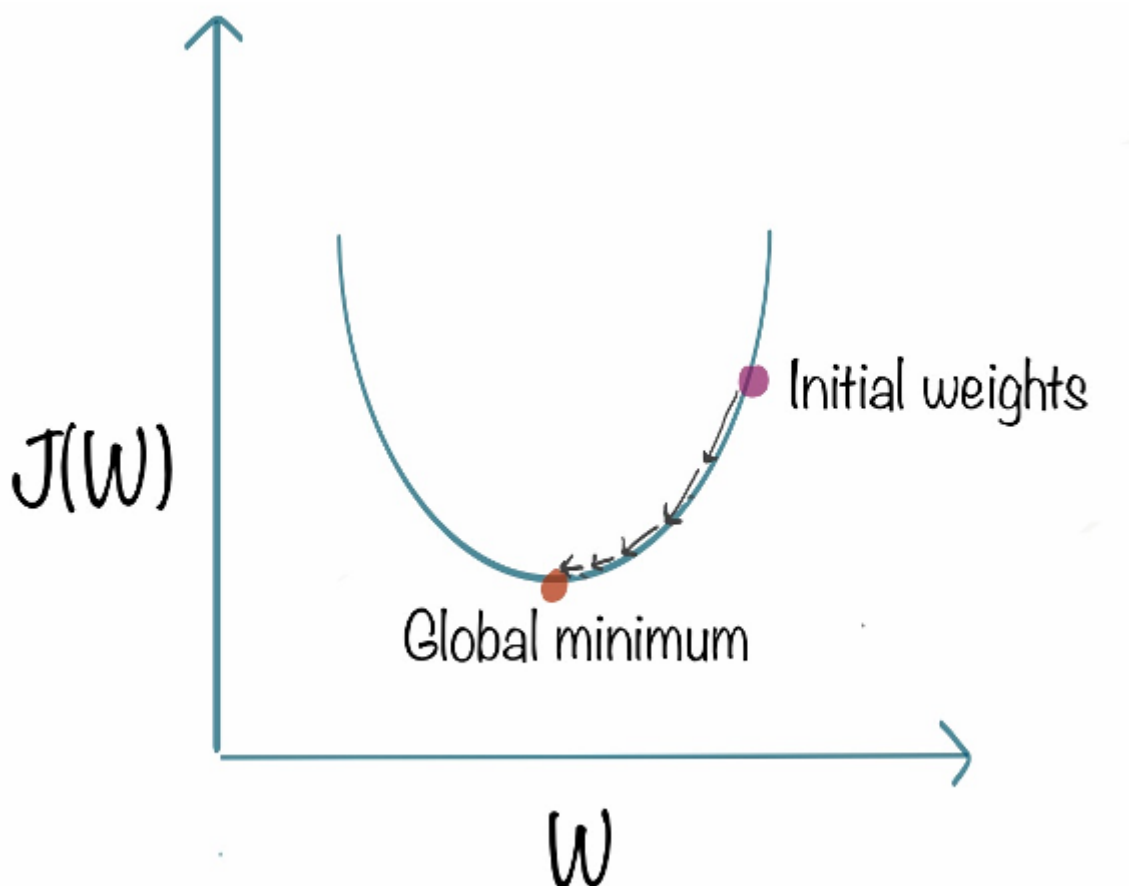
بررسی Optimizer ها:

Gradient Decent: این optimizer از روش گرادیان استفاده میکند تا local minimum های محلی را پیدا کند.

$$X_{new} = x(old) - \alpha * f'(x)$$

رابطه بالا نشان میدهد چگونه گرادیان عمل میکند. در اینجا alpha میزان گامی است که در هر مرحله برداشته میشود تا به لوکال مینیمم نزدیک تر شود.

در این روش با وزن های اولیه ابتدا شبکه مقدار دهی میشود سپس هزینه را محاسبه و در جهت کاستن هزینه حرکت میکند. وزن ها را به روز میکند. این کار را تا جایی ادامه میدهد که به لوکال مینیمم تابع هزینه برسد.



شکل ۴.۱۶. رفتار gradient decent

معایب این روش این است که برای دیتا های بسیار بزرگ هزینه محاسباتی سنگینی بر جای میگذارد. همچنین برای تابع های محدب خوب عمل میکند اما نمیتواند برای تابع های nonconvex به درستی عمل کند.

Adam: این optimizer اختصار یافته adaptive moment estimation است که نسخه پیشرفته تر stochastic gradient descent برای آپدیت وزن های شبکه عصبی میباشد. در این روش به جای استفاده از یک learning rate برای آپدیت وزن ها، adam به ازای هر وزن شبکه عصبی learning rate به آن اختصاص میدهد و آن را آپدیت میکند.

این optimizer مزایای متعددی دارد و به همین دلیل به عنوان optimizer پیش فرض از آن استفاده میکنند. از مزایای آن میتواند به سریع بودن اجرای آن، راحتی پیاده سازی، کم بودن نیاز به مموری و کم بودن نیاز به tune کردن آن میتوان اشاره کرد.

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) \left[\frac{\delta L}{\delta w_t} \right] \quad v_t = \beta_2 v_{t-1} + (1 - \beta_2) \left[\frac{\delta L}{\delta w_t} \right]^2$$

معادله بالا نشان دهنده نحوه کار adam می باشد. در اینجا β_1 و β_2 هرکدام نشان دهنده decay rate میانگین گرادیان های معادله می باشد.

این روش هم از down side هایی برخوردار است. adam بیشتر روی سرعت محاسبات تمرکز دارد اما stochastic gradient descent بیشتر روی data point ها تمرکز دارد و برای همین بهتر مدل را generalize میکند.

بررسی loss function ها:

Mean Squared Error: یکی از محبوب ترین loss function ها است که در آن میانگین مربع اختلاف خروجی های پیش بینی شده و خروجی های مدنظر محاسبه میشود.

$$MSE = \frac{1}{n} \sum_{i=1}^n (y^{(i)} - \hat{y}^{(i)})^2$$

در اینجا اختلاف به توان ۲ رسیده بنابراین دیگر مهم نیست که اختلاف ایجاد شده بالا یا پایین target value مورد نظر قرار گرفته باشد و فاصله آن در نظر گرفته میشود. از آن دست مقادیر با ارور های بزرگ خیلی بزرگ میشوند و میتوانند مشکل ساز باشند. همچنین mse یک تابع محدب است و یک global minimum واضح دارد و همین امر سبب میشود الگوریتم gradient descent روی آن خوب عمل کند.

Mean Absolute Error: در این تابع میانگین اختلاف مطلق بین مقادیر پیش بینی شده و مقادیر مطلوب محاسبه میشود.

$$MAE = \frac{1}{n} \sum_{i=1}^n |y^{(i)} - \hat{y}^{(i)}|$$

این تابع به عنوان تابع جایگزین mse استفاده میشود همانطور که اشاره شد mse در رابطه با outlier ها حساس است که میتواند روی عملکرد آن بسیار تاثیر گذار باشد. این تابع از این جهت خوب است که در حالاتی که outlier های زیادی وجود دارد میتواند این مشکل را بر طرف سازد.

مشکلی که این تابع برخوردار است این است که هرچه به صفر نزدیک تر میشود gradient descent دیگر روی آن به خوبی کار نمیکند چرا که مشتق تابع در نزدیکی صفر تعریف نشده است.

۳. معیار R2: نسبت تغییرات متغیر وابسته را که میتوان به متغیر مستقل نسبت داد را اندازه گیری میکند. این ضریب نشان میدهد که چند درصد تغییرات متغیر های وابسته در یک مدل رگرسیونی با متغیر مستقل تبیین میشود.

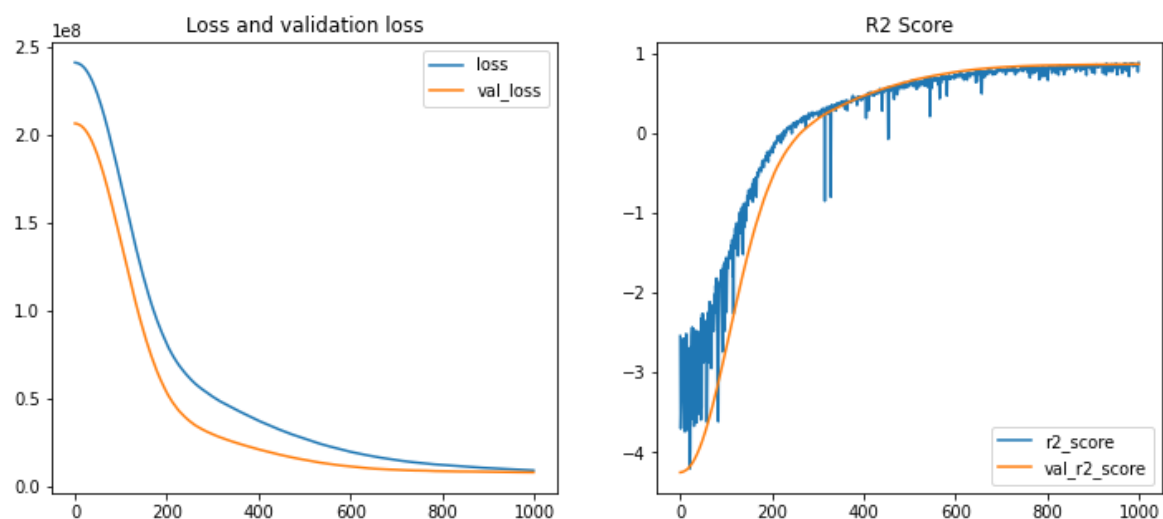
$$R - Squared = SS_{regression} / SS_{total}$$

در این رابطه SS regression معادل مجموع مربعات خطاها زمانی که از متغیر های مستقل استفاده شود است. همچنین SS total برابر مجموع توان دوم تمامی خطا هاست.

هر چه مقدار این ضریب به ۱ نزدیک تر باشد نشان دهنده نتایج بهتری است چرا که مقادیر بیشتری از اطلاعات توسط رگرسیون تولید شده توسط مدل توضیح داده میشود.

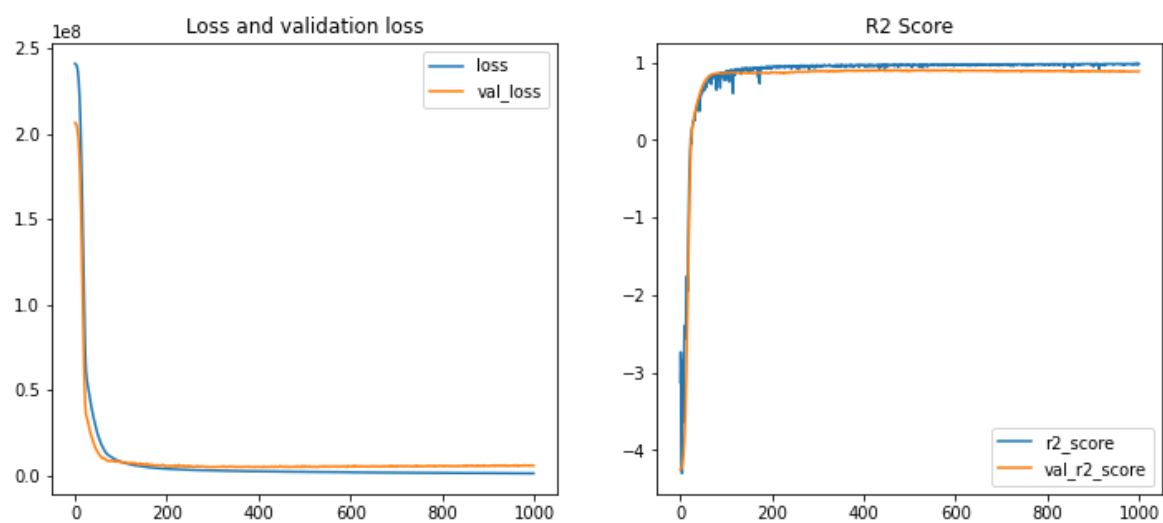
بررسی عملکرد شبکه های عصبی:

```
Epoch 993/1000
5/5 [=====] - 0s 25ms/step - loss: 8996516.0000 - coeff_determination: 0.8530 - val_loss: 7862176.0000 - val_coeff_determination: 0.8634
Epoch 994/1000
5/5 [=====] - 0s 22ms/step - loss: 8983632.0000 - coeff_determination: 0.8599 - val_loss: 7855783.5000 - val_coeff_determination: 0.8635
Epoch 995/1000
5/5 [=====] - 0s 20ms/step - loss: 8972360.0000 - coeff_determination: 0.8416 - val_loss: 7842597.5000 - val_coeff_determination: 0.8636
Epoch 996/1000
5/5 [=====] - 0s 29ms/step - loss: 8959098.0000 - coeff_determination: 0.8506 - val_loss: 7824908.0000 - val_coeff_determination: 0.8638
Epoch 997/1000
5/5 [=====] - 0s 19ms/step - loss: 8948338.0000 - coeff_determination: 0.8685 - val_loss: 7808563.0000 - val_coeff_determination: 0.8640
Epoch 998/1000
5/5 [=====] - 0s 19ms/step - loss: 8936800.0000 - coeff_determination: 0.7664 - val_loss: 7807283.0000 - val_coeff_determination: 0.8640
Epoch 999/1000
5/5 [=====] - 0s 17ms/step - loss: 8924420.0000 - coeff_determination: 0.8496 - val_loss: 7791309.5000 - val_coeff_determination: 0.8641
Epoch 1000/1000
5/5 [=====] - 0s 24ms/step - loss: 8912580.0000 - coeff_determination: 0.8833 - val_loss: 7771192.5000 - val_coeff_determination: 0.8643
```

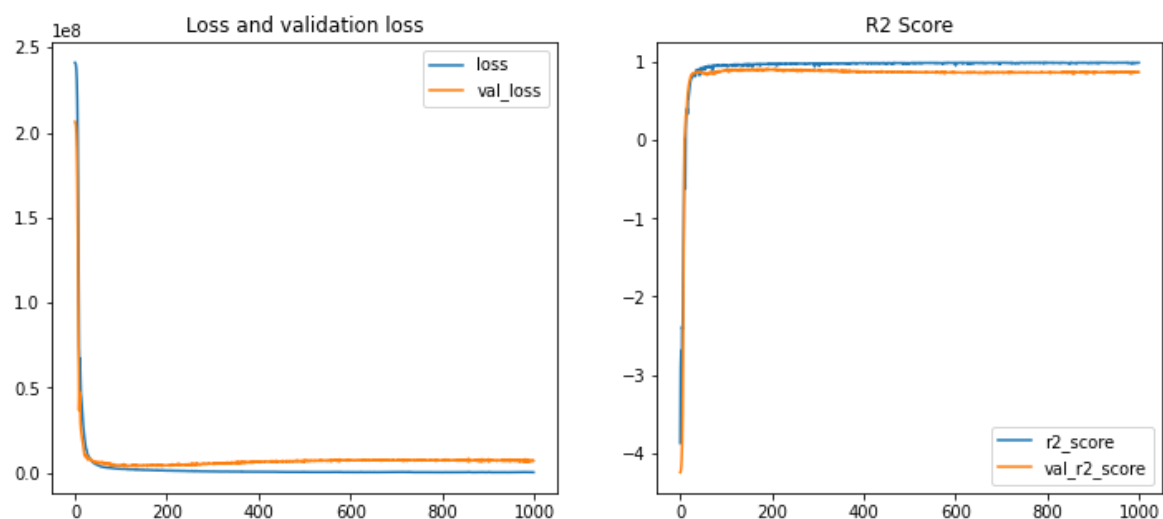
شکل ۴.۱۷. شبکه عصبی با یک لایه مخفی

```
Epoch 995/1000
5/5 [=====] - 0s 21ms/step - loss: 1050606.8750 - coeff_determination: 0.9805 - val_loss: 5720935.0000 - val_coeff_determination: 0.8829
Epoch 996/1000
5/5 [=====] - 0s 18ms/step - loss: 1041655.9375 - coeff_determination: 0.9847 - val_loss: 5687901.0000 - val_coeff_determination: 0.8840
Epoch 997/1000
5/5 [=====] - 0s 17ms/step - loss: 1038366.3750 - coeff_determination: 0.9772 - val_loss: 5778483.5000 - val_coeff_determination: 0.8817
Epoch 998/1000
5/5 [=====] - 0s 21ms/step - loss: 1040062.8750 - coeff_determination: 0.9855 - val_loss: 5743521.0000 - val_coeff_determination: 0.8825
Epoch 999/1000
5/5 [=====] - 0s 17ms/step - loss: 1039991.5000 - coeff_determination: 0.9743 - val_loss: 5641355.0000 - val_coeff_determination: 0.8850
Epoch 1000/1000
5/5 [=====] - 0s 18ms/step - loss: 1040988.3125 - coeff_determination: 0.9842 - val_loss: 5581193.0000 - val_coeff_determination: 0.8862
```



شکل ۴.۱۸. شبکه عصبی با دو لایه مخفی

```
Epoch 995/1000
5/5 [=====] - 0s 19ms/step - loss: 325531.3438 - coeff_determination: 0.9938 - val_loss: 7162619.5000 - val_coeff_determination: 0.8731
Epoch 996/1000
5/5 [=====] - 0s 25ms/step - loss: 330115.4688 - coeff_determination: 0.9949 - val_loss: 7255980.0000 - val_coeff_determination: 0.8724
Epoch 997/1000
5/5 [=====] - 0s 19ms/step - loss: 388738.4688 - coeff_determination: 0.9941 - val_loss: 6901606.5000 - val_coeff_determination: 0.8766
Epoch 998/1000
5/5 [=====] - 0s 25ms/step - loss: 402270.9688 - coeff_determination: 0.9904 - val_loss: 7420087.5000 - val_coeff_determination: 0.8700
Epoch 999/1000
5/5 [=====] - 0s 25ms/step - loss: 354298.7500 - coeff_determination: 0.9920 - val_loss: 6864436.0000 - val_coeff_determination: 0.8766
Epoch 1000/1000
5/5 [=====] - 0s 23ms/step - loss: 324037.3125 - coeff_determination: 0.9943 - val_loss: 7049467.5000 - val_coeff_determination: 0.8744
```

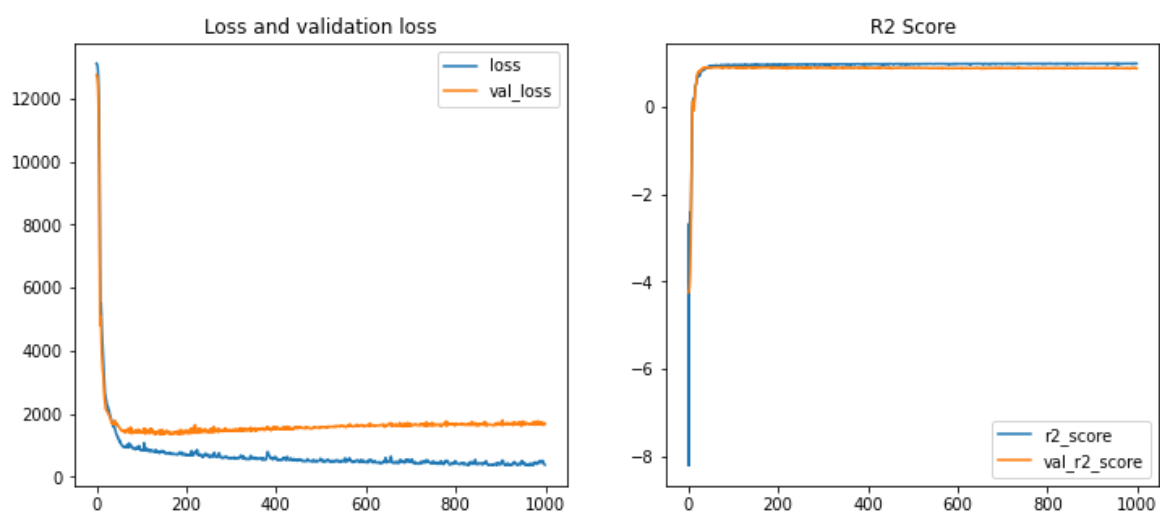


شکل ۴.۱۹. شبکه عصبی با سه لایه مخفی

ملاحظه میشود شبکه عصبی با سه لایه مخفی کمترین loss با ۳۲۴k و بیشترین ضریب r^2 با ۹۹.۴ درصد را داراست.

۴.

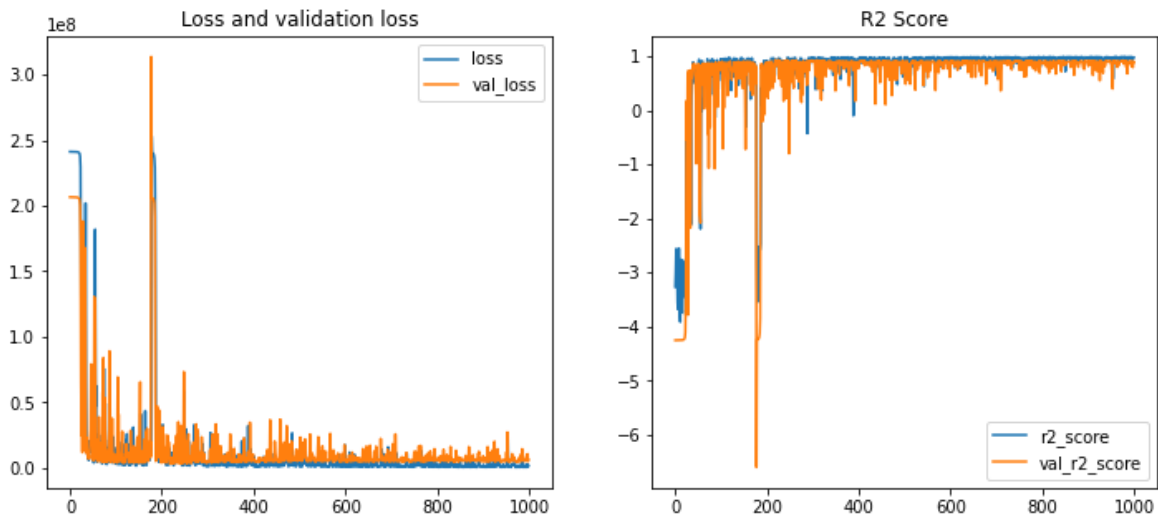
```
Epoch 999/1000
5/5 [=====] - 0s 25ms/step - loss: 402.1883 - coeff_determination: 0.9931 - val_loss: 1650.6147 - val_coeff_determination: 0.8905
Epoch 1000/1000
5/5 [=====] - 0s 20ms/step - loss: 379.3376 - coeff_determination: 0.9928 - val_loss: 1698.9769 - val_coeff_determination: 0.8839
```



شکل ۴.۲۰. Adam & mae

در این مدل با استفاده از adam شبکه به سرعت converge شده و همچنین از ضریب r^2 خوبی نیز برخوردار است. مشاهده میشود در میزان loss که در ایپاک های اولیه در validation set overfit رخ داده و خطای روی ولیدیشن افزایش و خطای train کاهش پیدا میکند. همچنین mae در مقایسه با mse در این optimizer بهتر عمل کرده است چرا که خطای نهایی آن برابر ۳۷۹ است که از جذر ۳۲۰k در mse کمتر است. این به این دلیل است که mse در رابطه با داده های outlier به خوبی عمل نمیکند.

```
Epoch 999/1000
5/5 [=====] - 0s 19ms/step - loss: 2299974.0000 - coeff_determination: 0.9610 - val_loss: 5048630.0000 - val_coeff_determination: 0.9118
Epoch 1000/1000
5/5 [=====] - 0s 26ms/step - loss: 1591140.6250 - coeff_determination: 0.9753 - val_loss: 6341249.5000 - val_coeff_determination: 0.8877
```



شکل ۴.۲۱. SGD& mse

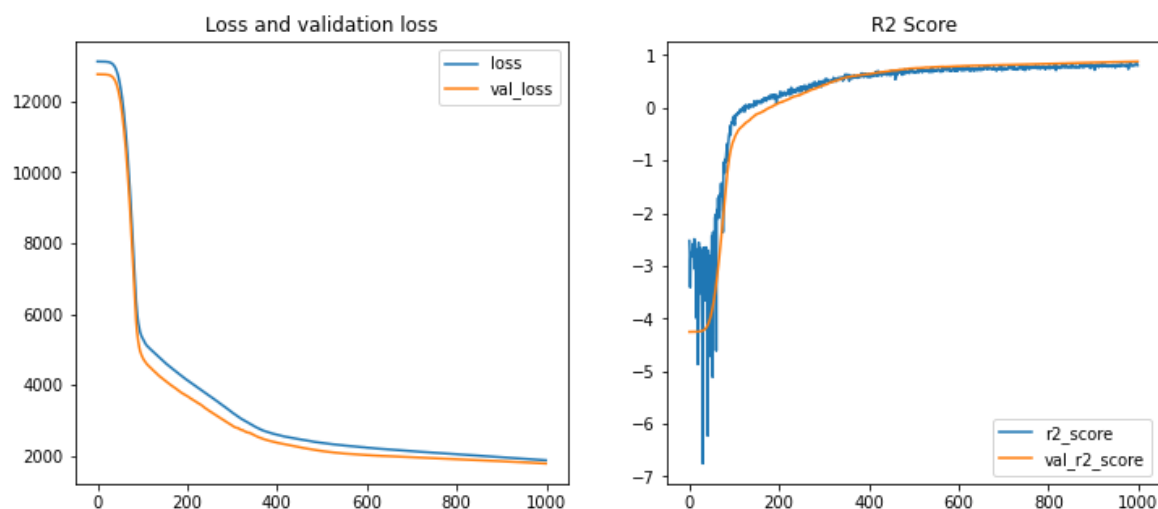
در شکل ۴.۱۷ مشاهده میشود noise زیادی در محاسبه خطا و حرکت sgd وجود دارد همچنین میزان خطای نهایی آن حدود ۵ برابر adam است اما از ضریب r^2 خوبی برخوردار است که این نشان میدهد بالا بودن r^2 همواره به معنی بهتر بودن مدل نیست. علت وجود نویز در نمودار به خاطر نحوه کارکرد sgd است. ممکن است learning rate برای این مدل بالا بوده باشد و روی local minimum به بالا و پایین رفته باشد برای همین از stability خوبی برخوردار نیست.

مدل SGD با loss function mae به درستی عملیات train را انجام نمیداد برای همین از mae و Adagrad در این بخش استفاده کردیم.

```

5/5 [=====] - 0s 20ms/step - loss: 1875.7964 - coeff_determination: 0.8338 - val_loss: 1783.1595 - val_coeff_determination: 0.8776
Epoch 1000/1000
5/5 [=====] - 0s 20ms/step - loss: 1874.0547 - coeff_determination: 0.8066 - val_loss: 1782.3228 - val_coeff_determination: 0.8782

```



شکل ۴.۲۲. Adagrad & mae

در این مدل یادگیری در ایپاک های اولیه به سرعت رخ داده اما به مرور سرعت کم شده تا آنجایی که حتی ۱۰۰۰ ایپاک هم برای یادگیری کامل آن کافی نبود و خطای نهایی آن ۶ برابر optimizer adam است. اما با نویز بسیار کمتری نسبت به sgd عمل میکند و نیاز به مقدار دهی learning rate نیست و به صورت adaptive حتی برای دیتاست های بزرگ به خوبی کار میکند. اما باید در ایپاک های بیشتری این روند انجام شود تا به convergence مناسبی برسد.

۵. پیش بینی قیمت ها با بهترین مدل Adam & mae:

```

y_pred: [8796.101], y_test: 7738.0
y_pred: [8757.019], y_test: 8495.0
y_pred: [10646.815], y_test: 8845.0
y_pred: [9370.099], y_test: 9298.0
y_pred: [7260.964], y_test: 7603.0

```

شکل ۴.۲۳. پیش بینی قیمت ها

ملاحظه میشود اختلاف قیمت ها از ۸۰ تومان تا ۱۰۰۰ تومان متغیر است که پیش بینی بسیار مناسبی برای این رنج های قیمتی میباشد.

