به نام خدا

دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

درس شبکه‌های عصبی و یادگیری عمیق

تمرین اول

نام و نام خانوادگی	آرمان عطاردی – پوریا شیخ الاسلامی
شماره دانشجویی	810101228 - 810100394
تاریخ ارسال گزارش	1401.12.28

فهرست مطالب

۱ شبکه عصبی Mcculloch Pitts

۱.۱ ماشین متناهی قطعی DFA

۱.۱.۱ الف)

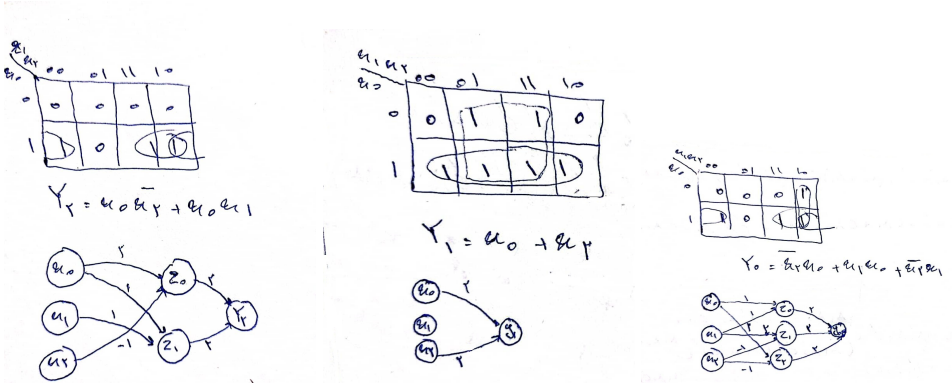
جدول ۱: جدول انتقال حالت DFA متناسب با شبکه عصبی

حالت کنونی	ورودی		حالت بعدی		پذیرفتن
X0	X1	X2	Y0	Y1	Y2
۰	۰	۰	۰	۰	۰
۰	۰	۱	۰	۱	۰
۰	۱	۰	۱	۰	۰
۰	۱	۱	۰	۱	۰
۱	۰	۰	۱	۱	۱
۱	۰	۱	۰	۱	۰
۱	۱	۰	۱	۱	۱
۱	۱	۱	۱	۱	۱

۲.۱.۱ ب)

با توجه به جدول بالا و با استفاده از جدول کارنو مدار ورودی به خروجی را بدست آورده و با استفاده از AND و OR آن را بازنویسی می کنیم. سپس با استفاده از نورون های شبکه عصبی مدار خود را می سازیم:

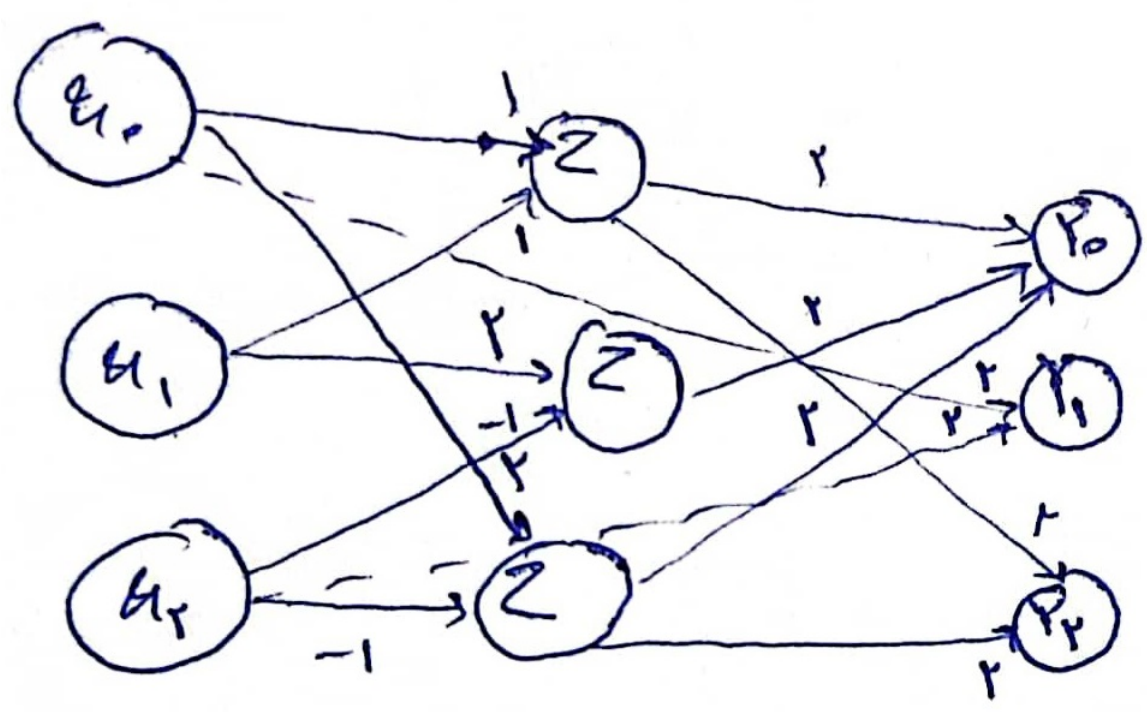
شکل ۱: مدار هر خروجی با استفاده از شبکه عصبی



(آ) شبکه Y0 (ب) شبکه Y1 (ج) شبکه Y2

۳.۱.۱ ج)

با ادغام کردن ۳ مدار بالا به مدار زیر می رسیم:



شکل ۲: DFA با استفاده از شبکه عصبی

توجه داشته باشید که threshold تمام مدار ها برابر ۲ می باشد.

۴.۱.۱ د)

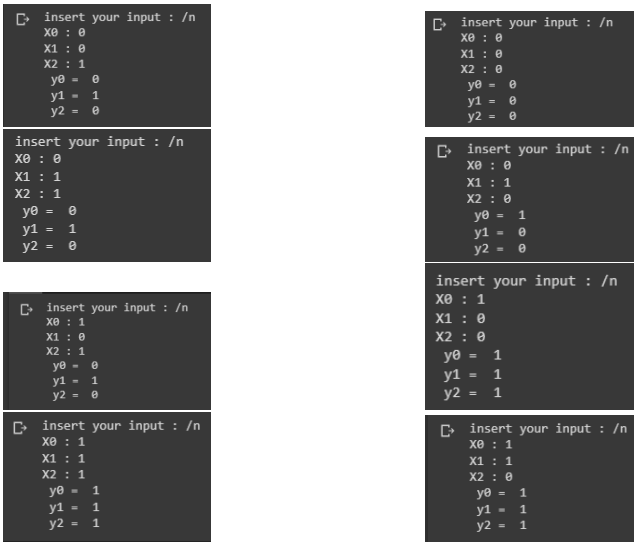
با استفاده از کد زیر کلاس نورون را تعریف کرده و سپس وزن ها و ورودی خود را مشخص می کنیم:

```
import numpy as np
class mp_neuron:
    def __init__(self , w0 , w1 , w2):
        self.w = np.array([[w0 , w1 , w2]])
        self.CONST_THRESHOLD = 2
        self.net =  1
        self.h = 0

    def setInput(self , x0 , x1 , x2):
        input = np.array([[x0],
                          [x1],
```

```
                    [x2]])
        self.net = np.sum(np.matmul(self.w , input))

    def getH(self):
        if self.net !=  1 :
            if self.net >= self.CONST_THRESHOLD : self.h = 1
            else : self.h = 0
        return self.h

    def getNet(self):
        return self.net

    def getW(self):
        return self.w

    def getThreshold(self):
        return self.CONST_THRESHOLD

z0 = mp_neuron(1 , 1 , 0 )
z1 = mp_neuron(0, 2 ,  1 )
z2 = mp_neuron(2 , 0 ,  1)
#--------------------
y0 = mp_neuron(2,2,2)
y1 = mp_neuron(2,0,2)
y2 = mp_neuron(2,0,2)
#--------------------
print("insert your input : /n")
x0 = int(input("X0 : "))
x1 = int(input("X1 : "))
x2 = int(input("X2 : "))
#----------------
z0.setInput(x0 , x1 , x2 )
z1.setInput(x0 , x1 , x2 )
z2.setInput(x0 , x1 , x2 )
#----------------
y0.setInput(z0.getH() , z1.getH() , z2.getH())
y1.setInput(x0 , x1 , x2)
y2.setInput(z0.getH() , z1.getH() , z2.getH())
#---
print(" y0 = " , y0.getH() , "\n y1 = " ,y1.getH() , "\n y2 = " ,y2.g
```

خروجی برنامه برای تمام ورودی های ممکن به شکل زیر خواهند بود:

شکل ۳: خروجی های شبکه عصبی طراحی شده به ازای تمام ورودی های ممکن



۲ شبکه های AdaLine and MadaLine

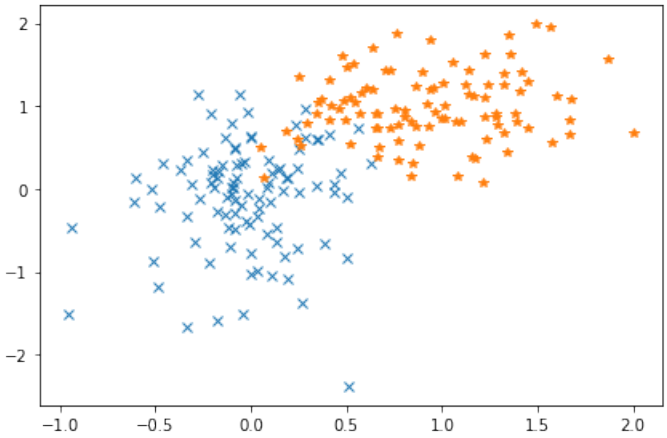
۱.۲ AdaLine

۱.۱.۲ الف

با توجه به داده‌های خواسته شده در این سوال و متغیرهای تصادفی ایکس و ایگرگ با میانگین داده شده و انحراف معیار مشخص نمودار پراکندگی یا همان اسکتر پلات را ترسیم خواهیم کرد به کمک پایتون در ابتدا برای این کار باید از ۱۰۰ داده تصادفی که به صورت اعداد گاووسی هستند استفاده کنیم برای این کار کد های مربوط به اعداد تصادفی گاووسی را در ابتدا در پایتون یا جوپیتر نوت بوک یا اسپایدر یا دیگر نرم افزار های برنامه نویسی پیاده سازی میکنیم و دو دسته جدا یعنی کلاس اول و کلاس دوم را می سازیم .
همانطور که مستحضر هستید میانگین اعداد یعنی مجموع اعداد تقسیم بر تعداد داده ها و در اینجا چون تعداد داده ها ۱۰۰ و میانگین خواسته شده ۱ می‌باشد پس باید مجموع اعداد نیز ۱۰۰ در نظر گرفته شود . با استفاده از کتابخانه NumPy دو دسته داده را با کد زیر رسم کرده و نمایش می دهیم.

```
#creating the firt sets of data
mean1 = [0, 0]
cov1 = [[0.1 , 0], [0, 0.4]]
x1 , y1 = np.random.multivariate_normal(mean1, cov1 , 100).T
#---
#Creating the secong sets of data
mean2 = [1, 1]
cov2 = [[0.2 , 0], [0, 0.2]]
x2  , y2= np.random.multivariate_normal(mean2, cov2 , 100).T
```

نتیجه بدین شکل خواهد بود:



شکل ۴: داده های AdaLine

۲.۱.۲ ب)

حال نورون های Adaline را به صورت زیر تعریف می کنیم:

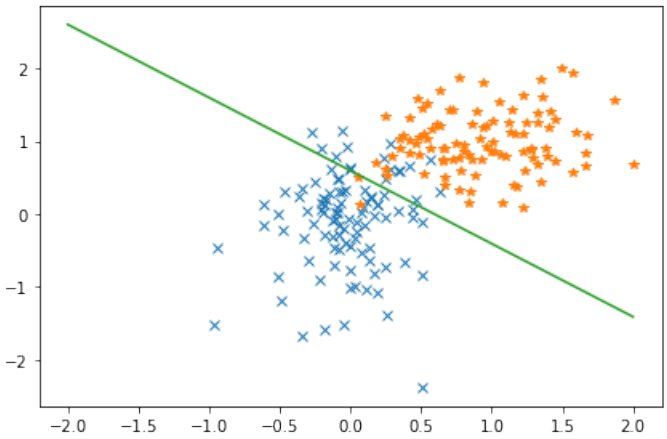
```
#Define a class with the charectristics of adaline neuron
class adaline_neuron:

    # one can change these constants
    def __init__(self):
        self.w = [0.002 ,0.00319 ]
        self.alpha = 0.0001
        self.threshold = 0.005
        self.net =  1
        self.h = 0

    def setInput(self , x0 , x1):
        input = np.array([[x0],
                          [x1]])
        self.net = np.sum(np.matmul(self.w , input)) + self.threshold

    def getH(self):
        if self.net !=  1 :
            if self.net >= self.CONST_THRESHOLD : self.h = 1
            else : self.h = 0
        return self.h

    def getNet(self):
        return self.net

    def getW(self):
        return self.w

    def getThreshold(self):
        return self.threshold

    def learn(self , sx , sy ,t ):
        i=0
        #start using the formula to find the right weights and biyas
        temp =0
        for i in range(200):
            self.setInput(sx[i] , sy[i])
```

```
            self.w[0] = self.w[0] + (self.alpha *
                                     (t[i]   self.net) * sx[i] )
            self.w[1] = self.w[1] + (self.alpha *
                                     (t[i]   self.net) * sy[i] )
            #find the right biyas
            self.threshold = self.threshold +
            (self.alpha * (t[i]   self.net))
            temp+=(1/2 *((t[i]   self.net)**2))
        return temp

    def isEnough(self , sx , sy , t ):
        CONST_ERROR_THRESHOLD = 1
        flag = False
        for i in range (100):
            self.setInput(sx[i] , sy[i])
           # print((t - self.net)**2)
            if ((t   self.net) **2) > CONST_ERROR_THRESHOLD :
                flag = True
        return flag
```

در کد بالا چندین متد تعریف شده که مهم ترین آنها متد learn می باشد. در این متد مقدار وزن و بایاس با توجه به ارور تعریف شده تعیین می شوند به طوری که مقدار آن را مینیمم کنند. همچنین مقدار خطا در هر ایپاک بدست آورده می شود و در انتها آنها را در یک نمودار نمایش می دهیم.
با تکرار متد آموزش به طور متوالی (در اینجا ۱۰۰ بار) به شکل زیر نتیجه قابل قبولی بدست می آوریم:

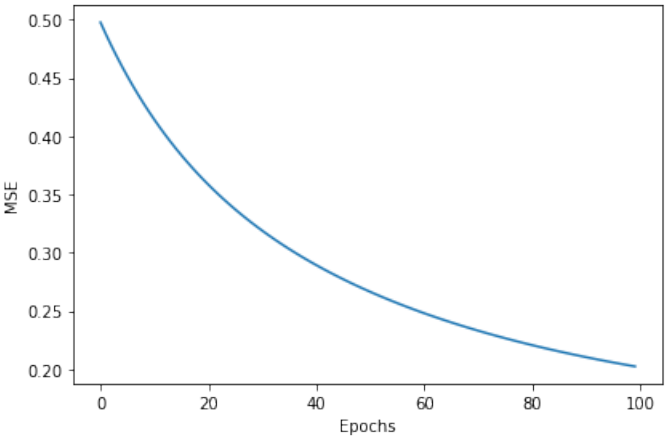
```
#Defining our neuron
ada_nur = adaline_neuron()
flag = True
ct=0
errorData01 = []

while(ct<100):
    error01 = ada_nur.learn(all_data[:,0] ,all_data[:,1] , all_data[:
    errorData01.append(error01)
    flag1 = ada_nur.isEnough(x1 , y1 , 1)
    if (not(flag1)) : break
    ct +=1
print("ct = " , ct)
```

نتیجه طبقه بندی بدین صورت می باشد:



شکل ۵: طبقه بندی با استفاده از AdaLine

همچنین نمودار خطا در طول ۱۰۰ ایپاک به شکل زیر است:

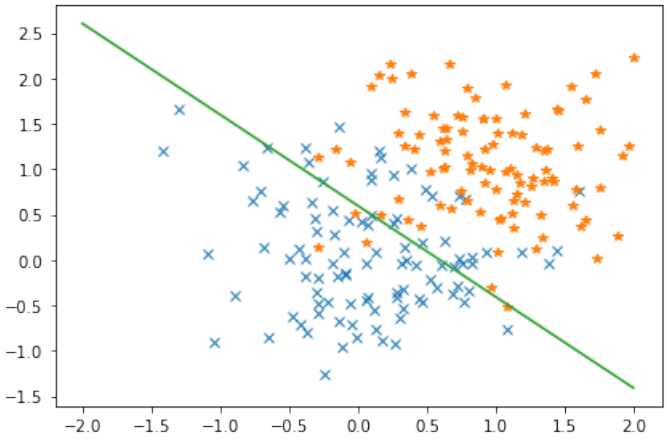


شکل ۶: خطای طبقه بندی AdaLine

با توجه به دو تصویر بالا می توان نتیجه گرفت که طبقه بندی به بهترین شکل انجام شده و مقدار خطادر طول ۱۰۰ ایپاک به ۰ نزدیک شده. البته با توجه با داده ها خطای ۰ امکان پذیر نمی باشد اما نورم AdaLine به گونه ای طبقه بندی را انجام داده که خطا به کمترین مقدار خود برسد.

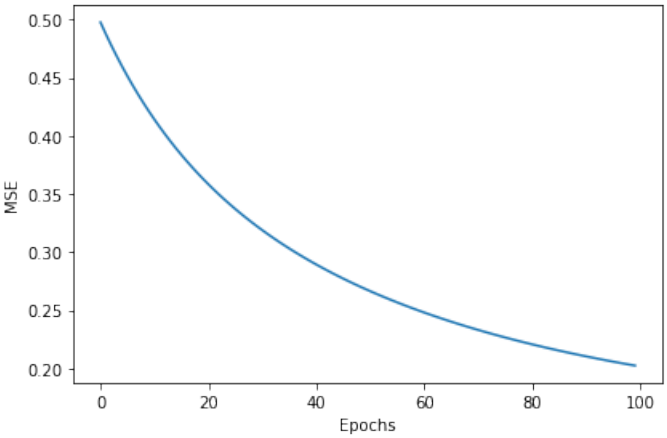
۳.۱.۲ ج

با تغییر مقادیر میانگین و واریانس در قسمت اول مراحل بالا را دوباره تکرار می کنیم.
نتایج به صورت زیر خواهد بود:



شکل ۷: طبقه بندی با استفاده از AdaLine

و نمودار خطا به شکل زیر است:



شکل ۸: طبقه بندی با استفاده از AdaLine

این بار نیز خطا ۰ نشده زیرا داده ها به صورت خطی از هم جدا پذیر نیستند اما نورون AdaLine توانسته به بهترین شکل ممکن آنها را جدا کند به صورتی که خطا به حداقل برسد.

۴.۱.۲ د

تفاوت داده های ب و ج در آنست که داده های قسمت ج به شدت درهم تنیده هستند و به همین دلیل زمان بیشتری برای جداسازی آنها نیاز است. همچنین در انتهای جداسازی مقدار Loss بیشتر از حالت ب است.

MadaLine ۲.۲

الف) ۱.۲.۲

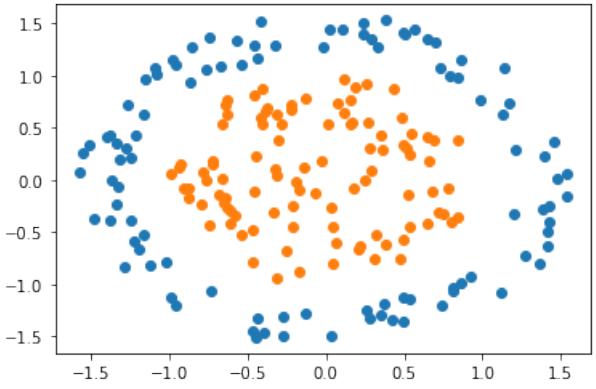
الگوریتم MRI به صورت زیر است: تابع فعال سازی به شکل تابع sign است. در این الگوریتم ابتدا مقادیر وزن ها و بایاس را به صورت رندوم یک عدد کوچک انتخاب می کنیم. سپس مراحل زیر را آنقدر تکرار می کنیم تا یا وزن ها دیگر تغییر نکنند یا حلقه به تعداد مشخصی اجرا شود:
ورودی ها را اعمال کنید و مقادیر خروجی هر نورون را مشخص کنید.
سپس مقدار net و خطا را محاسبه کرده و به شکل زیر وزن ها را بروزرسانی کنید
اگر مقدار هدف برابر مقدار خروجی بود هیچ کدام از نورون ها بروزرسانی نمی شوند در غیر این صورت
اگر مقدار هدف برابر ۱ بود وزن نورونی که خروجی آن به ۰ نزدیک تر است را بروزرسانی کن و اگر مقدار هدف برابر -۱ بود وزن همه نورون هایی که خروجی آن مثبت نیست را بروزرسانی کن.

ب) ۲.۲.۲

با استفاده از کد زیر داده ها را خوانده و رسم می کنیم:

```
#Read data from csv
headers = ['X', 'Y', 'T']
df = pd.read_csv('/content/MadaLine.csv' , names=headers)
df = df.reset_index()  # make sure indexes pair with number of rows

#Apply data into lists
xData01 =[]
yData01 = []
xData02 =[]
yData02 = []

for index, row in df.iterrows():
    if row['T'] == 1:
        xData01 = xData01 + [row['X']]
        yData01 = yData01 + [row['Y']]
    else:
        xData02 = xData02 + [row['X']]
        yData02 = yData02 + [row['Y']]
plt.scatter(xData01 , yData01)
plt.scatter(xData02 , yData02)
```

نتیجه بدین صورت خواهد بود:

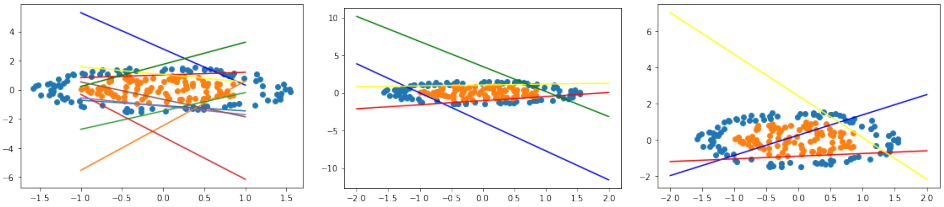


شکل ۹: داده های MadaLine

۳.۲.۲ ج)

داده ها را با ۳، ۴ و ۱۰ نورون با استفاده از کد های پیوست شده از هم جدا می کنیم. نتیجه به صورت زیر خواهد بود:

شکل ۱۰: تقسیم بندی داده ها با استفاده از MadaLine



(آ) ۳ نورون (ب) ۴ نورون (ج) ۱۰ نورون

همچنین تعداد ایپاک ها و دقت آنها به شرح زیر است:

	Accuracy	Epochs
3 Neurons	74.5%	500
4 Neurons	100%	130
10 Neurons	100%	40

طبیعتا با ۳ نورون نمی توانستیم به دقت صد در صد برسیم همچنین تعداد ایپاک ها برای رسیدن به دقت قابل قبول بسیار زیاد بود اما با ۴ نورون توانستیم دقت صد در صد را کسب کنیم. با اضافه کردن ۶ نورون دیگر با آنکه بار محاسباتی بیشتر شد اما در ایپاک کمتر (۴۰ ایپاک) توانستیم به همان دقت برسیم. البته تعداد ایپاک ها به صورت دستی انتخاب شده و برای انتخاب آنها از مقدار زیاد به کم حرکت کردیم و تا جایی که دقت افت نکند ادامه دادیم.

۴.۲.۲ د)

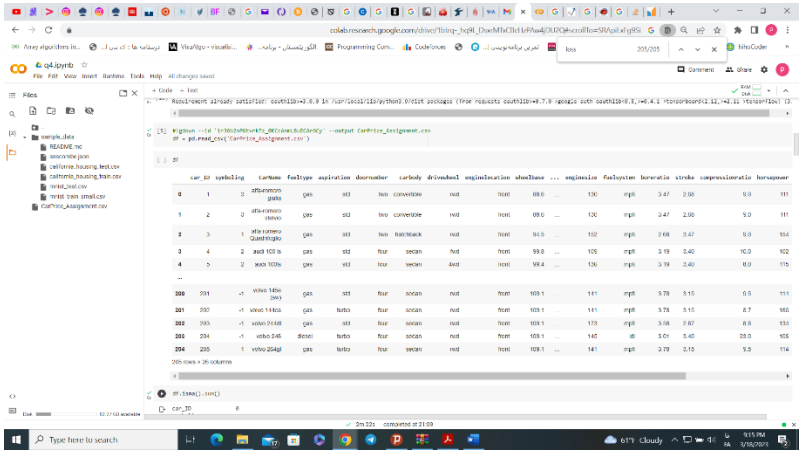
نتیجه بدست آمده قابل پیشبینی بود. با افزایش تعداد نورون ها تعداد ایپاک های لازم برای جداسازی کاهش پیدا می کند زیرا Redundancy افزایش پیدا کرده و چندین نورون جداسازی ها را برای یک قسمت انجام می دهند. طبیعتا کمتر از ۴ نورون نمی تواند این داده ها را از هم جداکنند.

Auto-Encoder for classification ۳

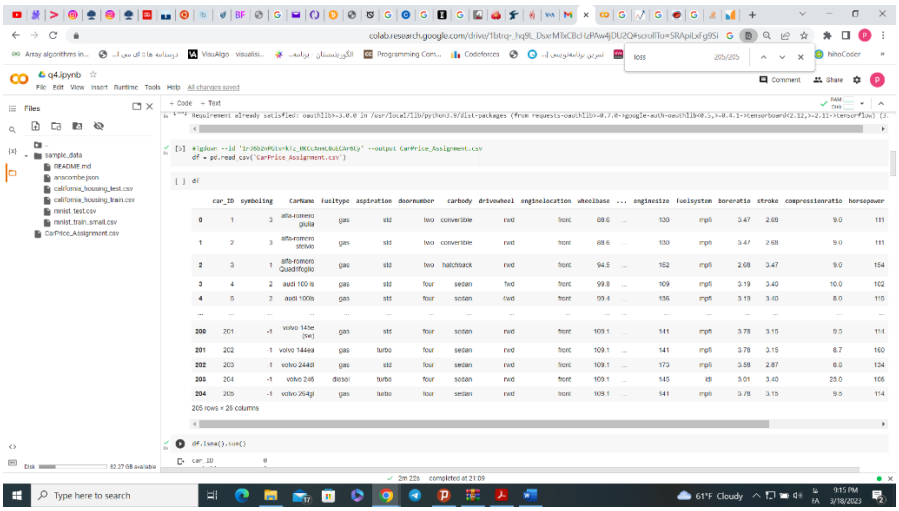
پاسخ ۴ – Multi–Layer Perceptron

۴-۱. پیش پردازش

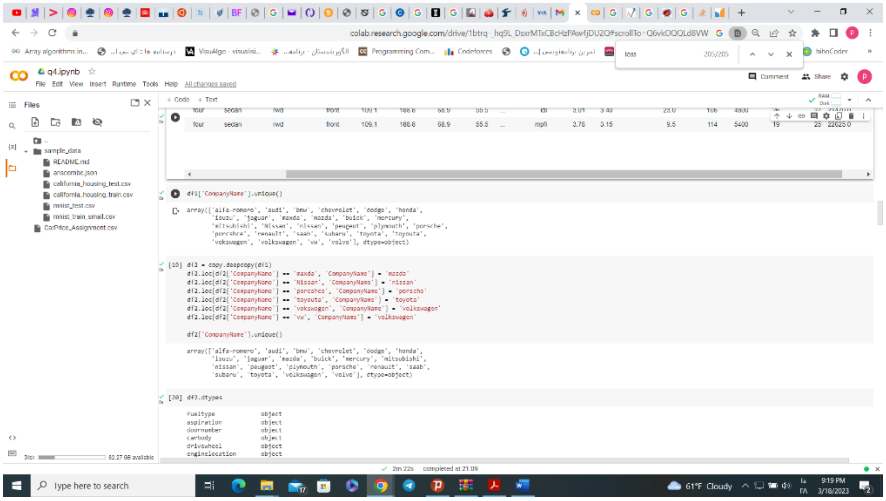
I. ابتدا با استفاده از pandas فایل دیتاست را میخوانیم. که کد این بخش به همراه چند خط از دیتاست نشان داده شده است. لازم به ذکر است پیشتر اقدام به فراخوانی کتابخانه مربوط و نصب پکیج های لازم گردید.



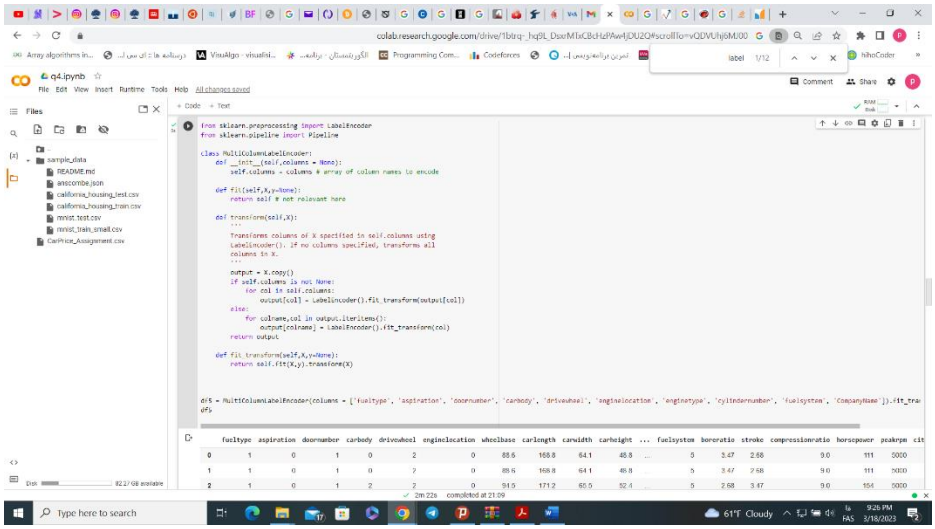
II. سپس با استفاده از دستور isnan اقدام به یافتن سلول های خالی کردیم و ستون های مربوطه را نشان دادیم. همانطور که در تصویر مشاهده میشود. همینطور در ادامه برای راحتی کار اقدام به deep copy از دیتاست کردیم.



III. سپس با توجه به نام های شرکت در دیتاست موجود با استفاده از دستور unique اقدام به یافتن و جایگذاری با نام های درست کردیم.

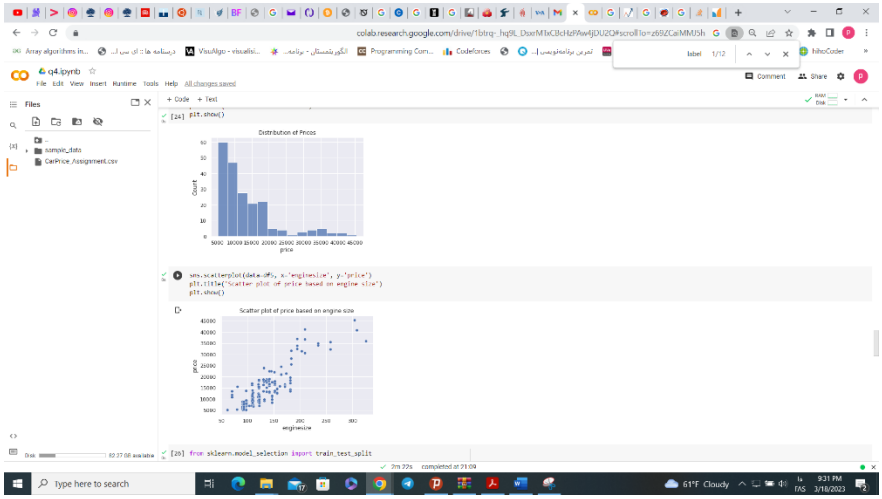


IV. برای انجام اینکار به جای استفاده از pd.get_dummies از label encoder استفاده میکنیم دلیل آن این است که اولی خروجی one hot تولید میکند که در فرایند رسم ماتریس کورلیشن به ضرر ماست و تعداد ستون ها را زیاد میکند.

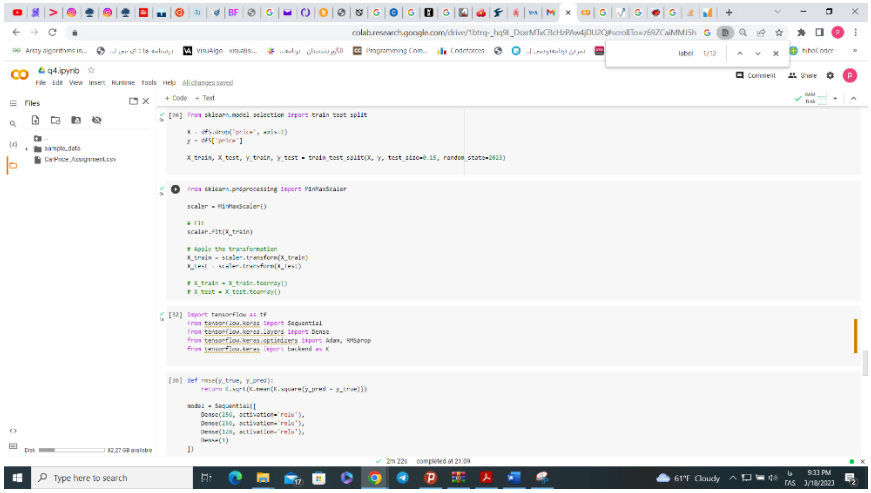


V. طبق انچه مشاهده شد engine size بیشترین ارتباط را با قیمت دارد. که بخش از آن در تصویر مشاهده میشود.

VI. در شکل نمودار سایز موتور بر حسب قیمت رسم شده است.



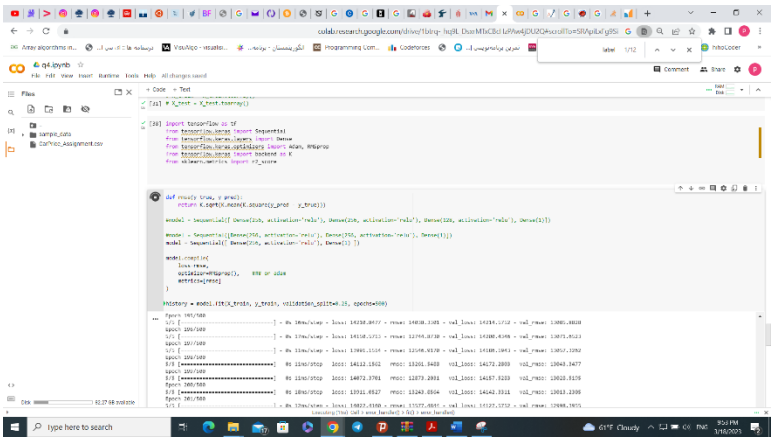
VII. مطابق با خواست سوال انجام شده است.

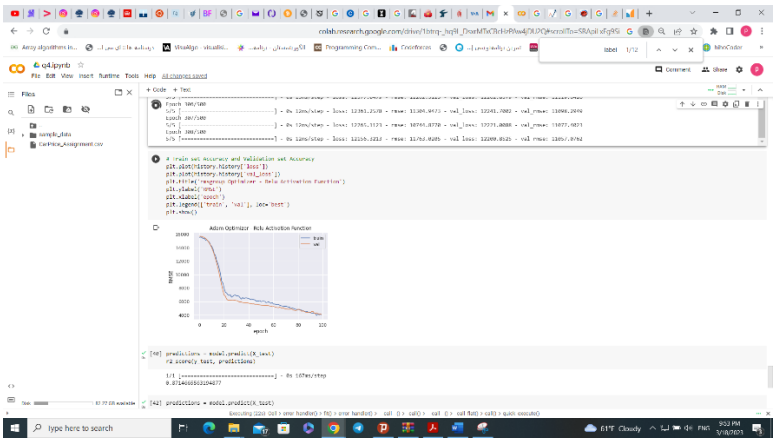


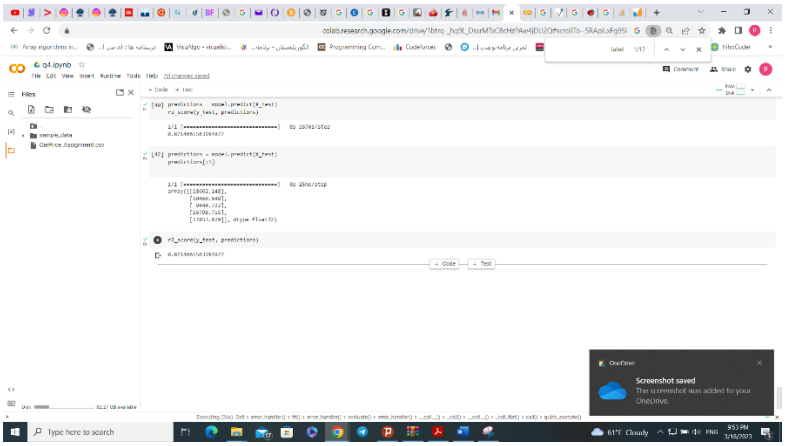
VIII. مطابق با خواست سوال انجام شده است. همانطور که در تصویر پیشین دید میشود.

MLP .۲-۴ .

شمای کلی کد برای عدم تکرار در ابتدا آورده شده است و سپس صرفا به تغییرات نتایج اشاره میشود.







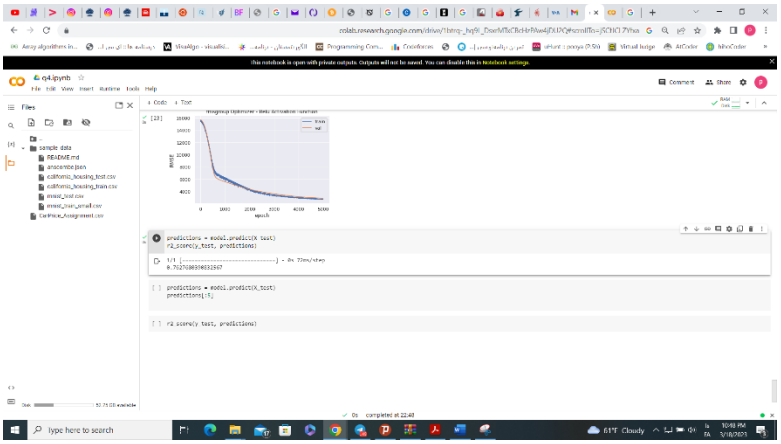
I. با توجه به صورت سوال به همان ترتیب عمل کردیم.

II. برای انجام از دو بهینه ساز adam و rmsprop استفاده کردیم و برای تابع loss نیز از rmse و r2-score.

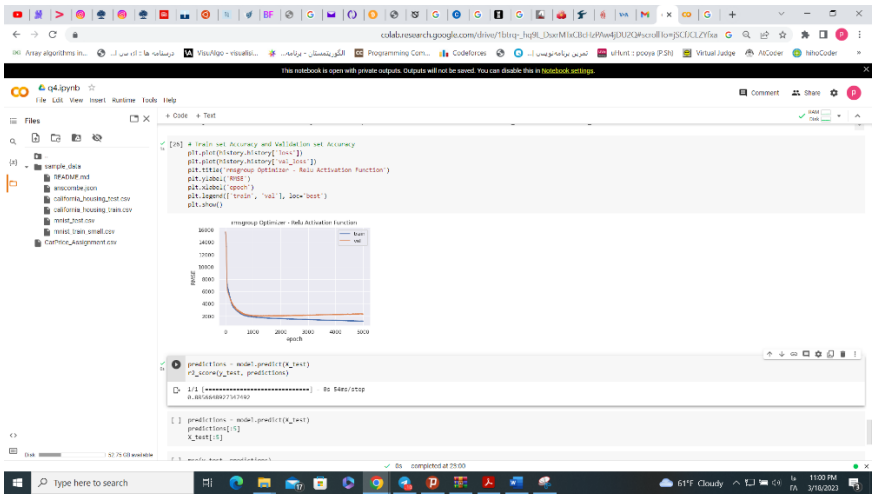
III. معیار r2-score در واقع همان ضریب تعیین است که نسبتی از واریانس متغیر وابسته است که از متغیرهای مستقل قابل پیشبینی باشد. این معیار در بحث های آماری بیشتر برای هدف پیشبینی خروجی آینده استفاده میشود که شامل یک رگرسیون خطی ساده است همینطور معمولا بین 0 و 1 قرار میگیرد. البته ممکن است مقادیر منفی نیز تولید شود چنین مواردی هنگامی پدیدار میشود که پیشبینی های مورد مقایسه با خروجی های متناظر از فرایند برازش مدل حاصل از آن داده ها مشتق نشده باشد. بهترین مقداری که میتواند بگیرد همان 1 است و مقدار 0 نشان دهنده این است که مدل همیشه میانگین خروجی بدون توجه به ورودی را گزارش میکند.

برای شروع ما برای تابع rmse loss و برای optimization، rmsprop را در نظر گرفتیم که در ادامه از mse و adam استفاده شدند. لازم به ذکر است patch-size=1 در نظر گرفته شده است.

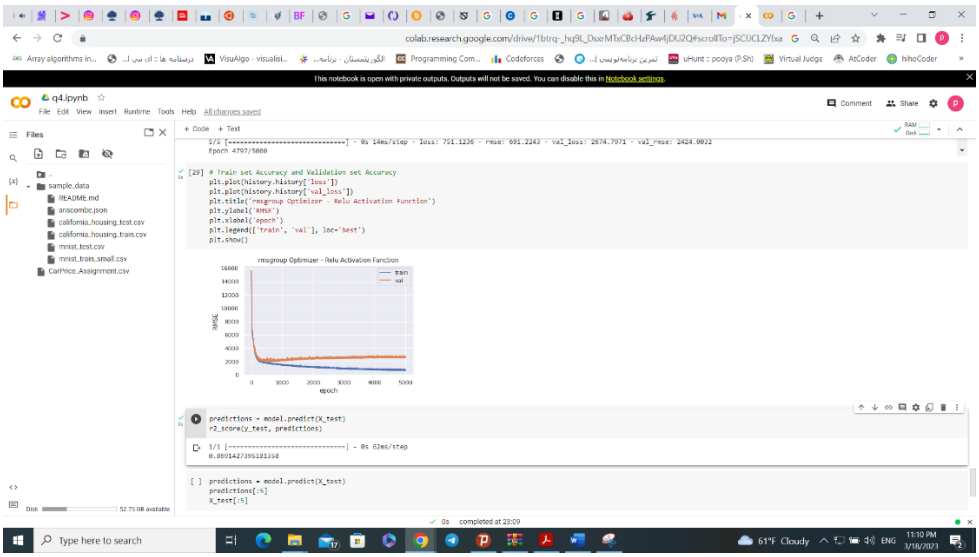
در ادامه به ترتیب نتایج خروجی اورده شده است. ابتدا برای مدل با یک لایه پنهان r2-score 0.76 به دست آمد.



برای مدل با دو لایه پنهان نتایج زیر به دست آمد همچنین میزان r2-score، 0.88 شد.



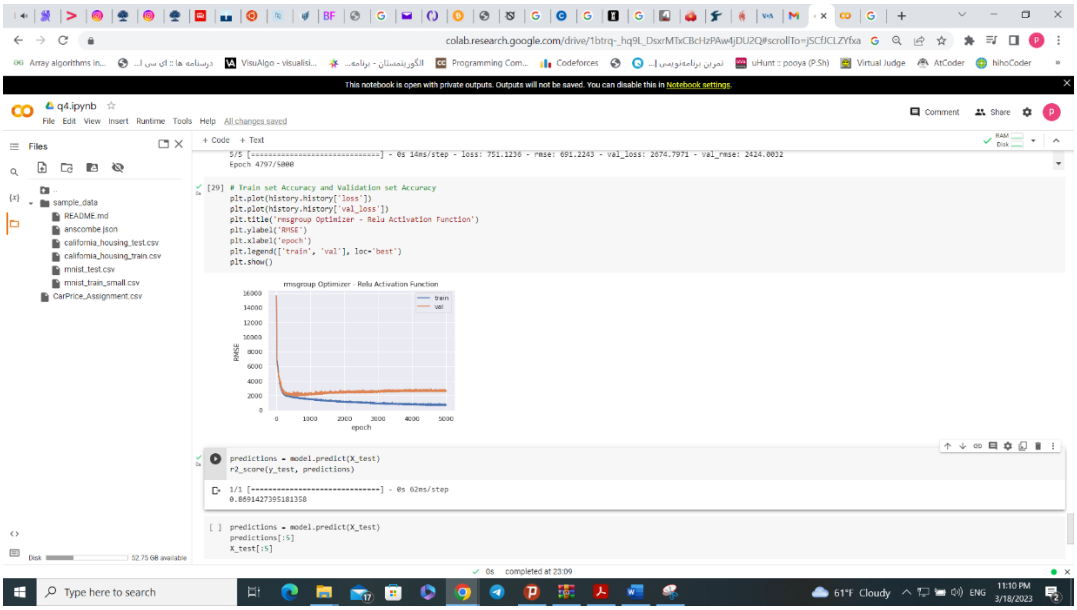
برای مدل با سه لایه پنهان نیز r2-score، 0.86 به دست آمد که نتایج قابل ملاحضه است.



با توجه به انچه گفته شد مدل با دو لایه پنهان انتخاب میشود. داریم:

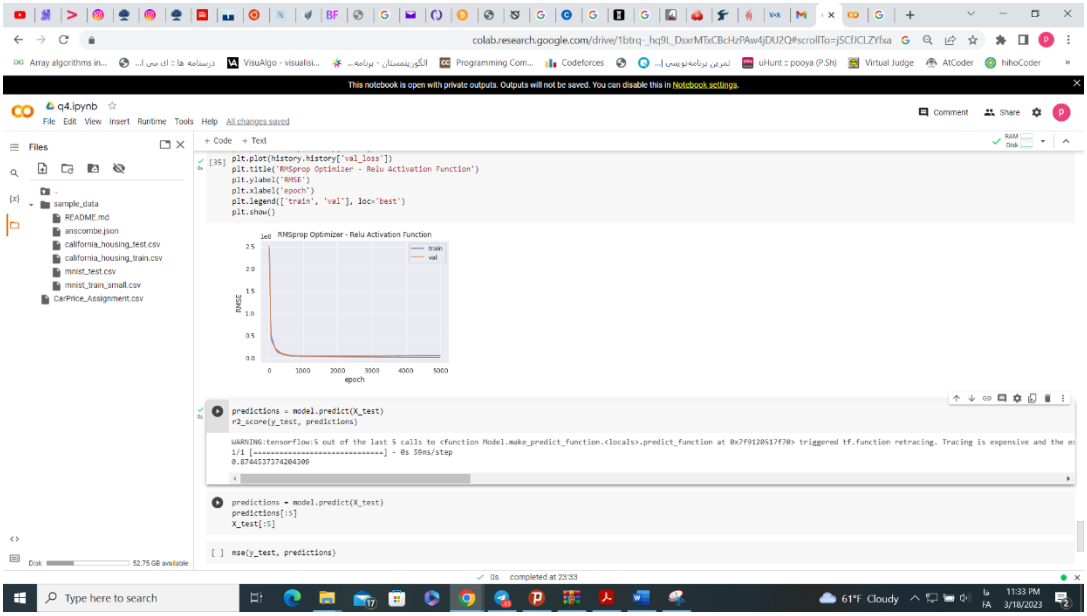
اکنون با توجه به صورت سوال برای مدل دومی انچه خواسته شده است را رسم میکنیم:

IV.ابتدا تابع بهینه ساز را تغییر میدهیم:



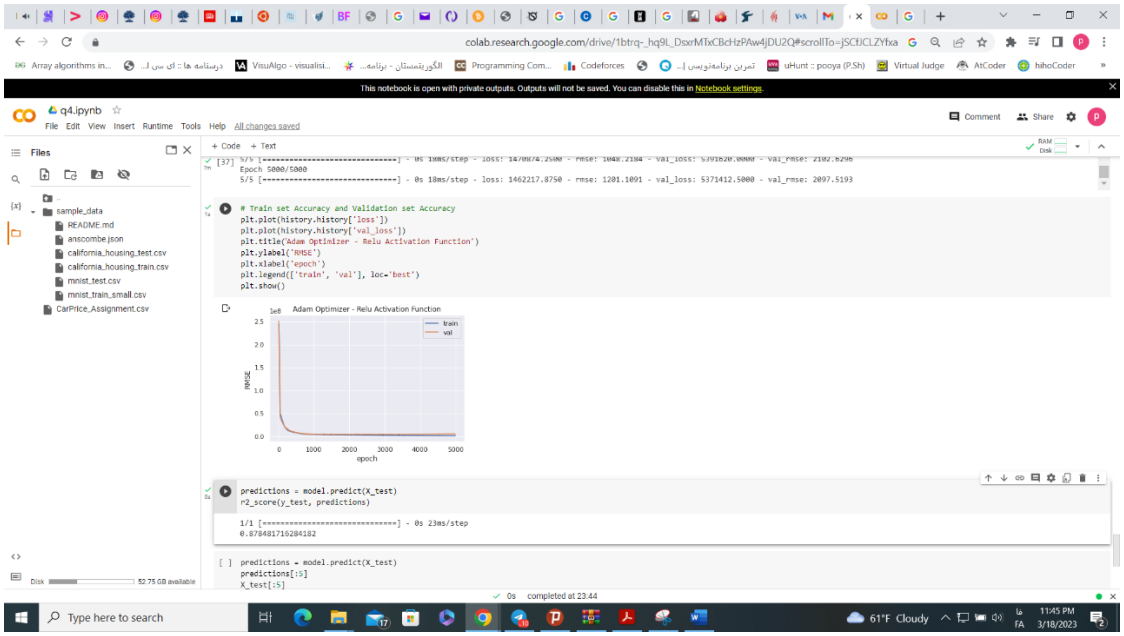
که در نتیجه آن r2-score، 0.88 میشود.

سپس برای تابع loss، mse ابتدا با تابع بهینه ساز rmsprop:



که در نتیجه آن r2-score، 0.87 میشود.

و سپس با تابع adam:



که در نتیجه آن r2-score، 0.878 میشود.

نتایج قابل مشاهده اند.

V. نتایج پیش بینی شده به همراه مقادیر واقعی آن اورده شده است همچنین برای مشاهده ی میزان از معیار mean-squared-error استفاده شد که نتایج آن در ادامه آورده شده است.