به نام خدا

دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

درس شبکه‌های عصبی و یادگیری عمیق

تمرین اول

نام و نام خانوادگی	محمد ناصری – مریم عباس‌زاده
شماره دانشجویی	810100406 - 810100486
تاریخ ارسال گزارش	1401.08.07

فهرست

پاسخ 1. شبکه عصبی Mcculloch-Pitts 1

۱-۱. ضرب کننده باینری دو بیتی 1

پاسخ ۲ – AdaLine and MadaLine 6

۲-۱. **AdaLine** 6

۲-۱. **MAdaLine** 9

پاسخ ۳ – عنوان پرسش سوم به فارسی 13

۳-۱. عنوان بخش اول 13

پاسخ ۴ –MLP 14 14

۴-۱. **Multi Layer Perceptron** 14

شکل‌ها

شکل 1 - خروجی z3 2
شکل 2 - خروجی z2 2
شکل 3 - خروجی z1 3
شکل 4 - خروجی z0 3
شکل 5 - خروجی کد برای همه موارد ممکن 5
شکل 6 - برنامه سوال دوم۱ 6
شکل 7 - برنامه سوال دوم۲ 7
شکل 8 - نمودار نقاط و خط جداکننده adaLine 7
شکل 9 - نمودار خطای adaLine 7
شکل 10 - نمودار نقاط و خط جداکننده adaLine ۲ 8
شکل 11 - نمودار خطای adaLine ۲ 8
شکل 12- قوانین آپدیت کردن وزن 10
شکل 13 - نمودار پراکندگی داده خام 10
شکل 14- نمودار پراکندگی نقاط با ۳ خط جداکننده MadaLine 11
شکل 15- نمودار پراکندگی نقاط با ۴ خط جداکننده MadaLine 11
شکل 16 - نمودار پراکندگی نقاط با ۸ خط جداکننده MadaLine 11
شکل 18 - نمودار توزیع قیمت..... 14
شکل 19 - نمودار قیمت و 'sqft_living'..... 15
شکل 20 - قطعه کد مربوط به تبدیل فیچر دیتا به فیچرهای سال و ماه..... 15
شکل 21 - قطعه کد مربوط به تقسیم داده ها به داده های تست و ترین..... 15
شکل 22 - MinMax Scaler First 16
شکل 23 - MinMax Scaler Second 16
شکل 24 - نمودار loss و validation loss (MSELoss, Adam)..... 16
شکل 25 - نمودار loss و validation loss (Adam, HuberLoss)..... 17
شکل 26 - نمودار loss و validation loss (SGD, MSELoss)..... 18
شکل 27 -نمودار loss و validation loss (SGD, HuberLoss)..... 19
شکل 28 - قطعه کد مربوط به رسم نمودار 20

شکل 29 - قطعه کد مربوط به پیشبینی قیمت.....20

جدول‌ها

جدول 1. جدول حالت ضرب دوبیتی 1
جدول 2 - جدول دقت و تعداد epoch برای MadaLine 12

پاسخ ۱. شبکه عصبی Mcculloch-Pitts

۱-۱. ضرب کننده باینری دو بیتی

هدف این سوال این است که به کمک نورون Pitts-Mcculloch توسعه یافته یک ضرب کننده باینری ساخته شود، که دو ورودی دو بیتی را گرفته و آن ها را ضرب کند. برای این کار به دو ورودی دو بیتی (در واقع چهار نورون برای همه ورودیها) نیاز داریم .همچنین چهار بیت خروجی (چهار نورون) مورد نیاز است. لازم به توجه است که تمامی نورون های ورود ی و خروجی باینری هستند.

برای حل این مساله در ابتدا جدول حالت ضرب دوبیتی را تشکیل میدهیم تا فرمولهای خروجی بدست بیاید.

جدول 1. جدول حالت ضرب دوبیتی

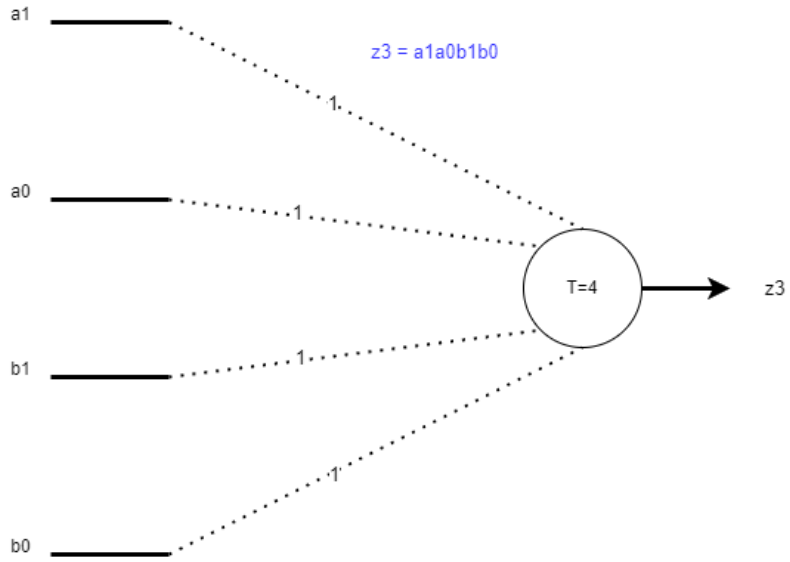
a1	a0	b1	b0	z3	z2	z1	z0
0	0	0	0				
0	0	0	1				
0	0	1	0				
0	0	1	1				
0	1	0	0				
0	1	0	1				1
0	1	1	0			1	
0	1	1	1			1	1
1	0	0	0				
1	0	0	1			1	
1	0	1	0		1		
1	0	1	1		1	1	
1	1	0	0				
1	1	0	1			1	1
1	1	1	0		1	1	
1	1	1	1	1			1

از روی جدول بدست آمده با تشکیل جداول کارنو به ازای هر بیت خروجی Z مقادیر زیر حاصل می‌شود.

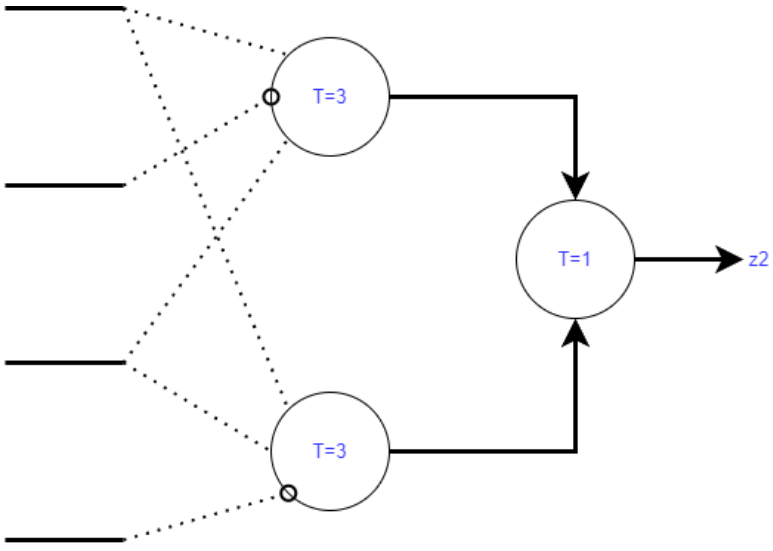
$$z_3 = a_1 a_0 b_1 b_0$$
$$z_2 = a_1 \overline{a_0} b_1 + a_1 b_1 \overline{b_0}$$
$$z_1 = a_1 \overline{a_0} b_0 + a_1 \overline{b_1} b_0 + \overline{a_1} a_0 b_1 + a_0 b_1 \overline{b_0}$$
$$z_0 = a_0 b_0$$

الف) رسم شبکه هر خروجی

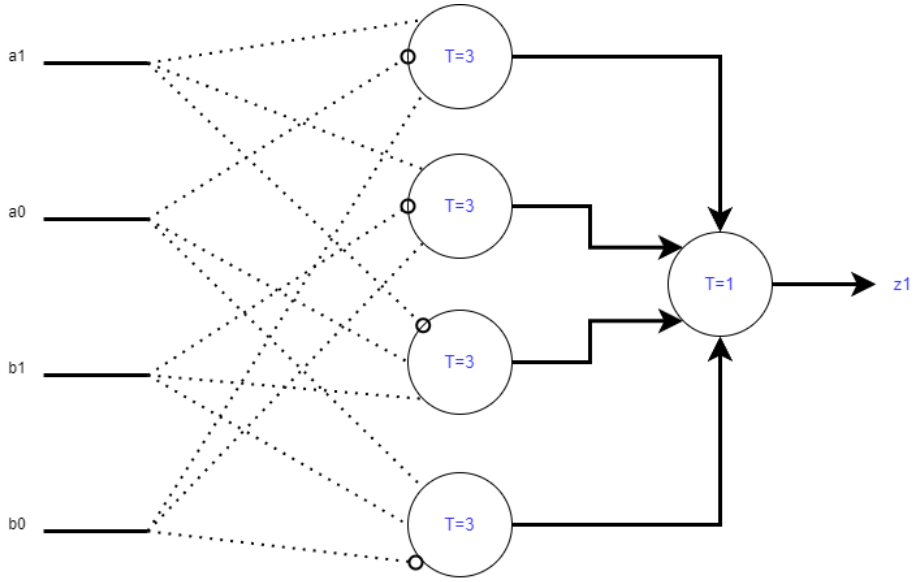
برای رسم شبکه‌ها وزن هر نورون را به صورت پیش‌فرض برابر ۱ در نظر می‌گیریم و رسم می‌کنیم.



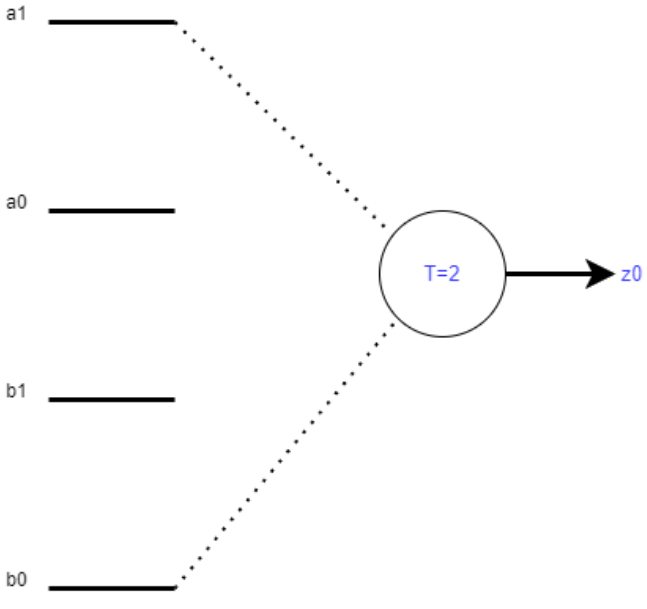
شکل 1 - خروجی **z3**



شکل 2 - خروجی **z2**



شکل 3 - خروجی **z1**



شکل 4 - خروجی **z0**

لازم به ذکر است در نمودارهای بالا خطوط متصل کننده که با دایره در انتها مشخص شده‌اند دارای وزن (۱-) می‌باشند. در مرحله بعد با توجه به شبکه‌های به‌دست آمده، آن‌ها را پیاده‌سازی می‌کنیم.

ب) پیاده‌سازی شبکه‌ها

با توجه به شبکه‌های به‌دست آمده در ابتدا تعریفی برای هر یک از **gate**ها انجام می‌دهیم. در ادامه به کمک گیت‌ها دقیقا مطابق شبکه‌های تصاویر بالا ورودی‌ها را به گیت داده و خروجی را ثبت میکنیم که خروجی نهایی به شکل زیر خواهد بود.

```
def threshold_gate(dot: int, T: float) -> int:
    '''Returns the binary threshold output'''
    if dot >= T:
        return 1
    else:
        return 0
```

```
def and_gate(inputs, weights) -> int:
  dot_product = np.array(inputs) @ np.array(weights)
  return threshold_gate(dot_product, sum(weights))
```

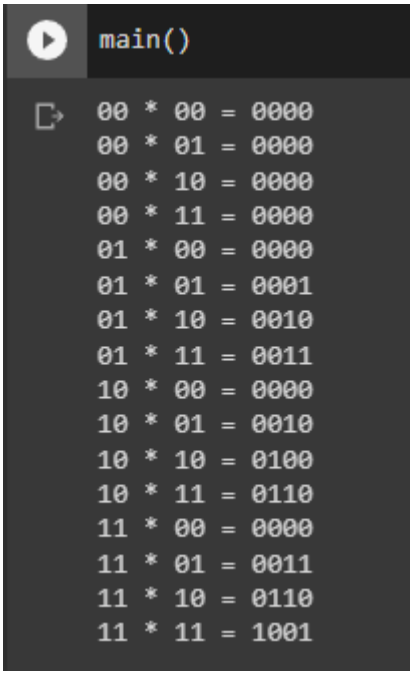
```
def or_gate(inputs, weights) -> int:
  dot_product = np.array(inputs) @ np.array(weights)
  return threshold_gate(dot_product, 1)
```

```
def not_gate(input, weight=-1):
  dot_product = input * weight
  return threshold_gate(dot_product, 0)
```

```
def two_bit_multiplier(a_1, a_0, b_1, b_0):
  z_3 = and_gate([a_1, a_0, b_1, b_0], [1, 1, 1, 1])

  z_2 = or_gate([
      and_gate([a_1, not_gate(a_0), b_1], [1,1,1]),
      and_gate([a_1, b_1, not_gate(b_0)], [1,1,1])
    ],[1,1])

  z_1 = or_gate([
      and_gate([a_1, not_gate(a_0), b_0], [1,1,1]),
      and_gate([a_1, not_gate(b_1), b_0], [1,1,1]),
      and_gate([not_gate(a_1), a_0, b_1], [1,1,1]),
      and_gate([a_0, b_1, not_gate(b_0)], [1,1,1]),
    ], [1,1,1,1])

  z_0 = and_gate([a_0, b_0], [1,1])

  return z_3, z_2, z_1, z_0
```

شکل 5-پیاده سازی شبکه ضرب دوبیتی

پس از پیاده‌سازی و اجرا، تمامی حالات ضرب دوبیتی را بررسی و نمایش می‌دهیم:



شکل 6 - خروجی کد برای همه موارد ممکن

پاسخ ۲ – AdaLine and MadaLine

۲-۱. AdaLine

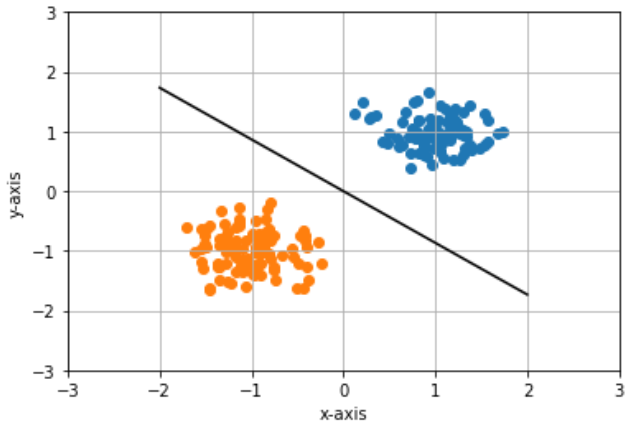
الف) برنامه‌ی مربوط به این بخش، در فایل به نام HW1_Q2_A.py ذخیره شده است. منظور از step، مرحله‌ای از محاسبات است که در آن، هر یک از الگوهای دوبعدیِ (x,y)، در به‌روز کردن ضرایب مساله نقش دارند. در واقع هر epoch، متشکل از چند step است. برای اینکه منحنی خطا، معنادار باشد، در رسم آن، محور افقی را epochها در نظر گرفته‌ایم و نه stepها. مقدار خطا را در پایان هر epoch، برابر میانگین تمام $0.5(t-net)^2$ ها فرض کرده‌ایم. تصاویر برنامه و نتایج آن را در زیر مشاهده می‌کنید و پس از تصاویر تحلیل هر دو مورد با هم قرار گرفته است (پیاده‌سازی در فایل HW1_Q2_A/B):

```
10  import numpy as np
11  import math
12  import matplotlib.pyplot as plt
13
14  data = np.zeros((200,3))
15  x1 = np.random.normal(1,0.3,100)
16  y1 = np.random.normal(1,0.3,100)
17  x2 = np.random.normal(-1,0.3,100)
18  y2 = np.random.normal(-1,0.3,100)
19
20  for i in range(100):
21      data[i,0] = x1[i]
22      data[i,1] = y1[i]
23      data[i,2] = 1
24
25  for i in range(100):
26      data[i+100,0] = x2[i]
27      data[i+100,1] = y2[i]
28      data[i+100,2] = -1
29  data
30
31  w1,w2,b,alpha = np.random.normal(0,0.1,4)
32  alpha = math.fabs(alpha)
33  temp = np.ones((200,1))
34  epoch=0
35  error = []
36
37  while(epoch < 500):
38
39      for i in range(200):
40          net = w1*data[i,0]+w2*data[i,1]+b
41          w1 += alpha*(data[i,2]-net)*data[i,0]
42          w2 += alpha*(data[i,2]-net)*data[i,1]
43          b += alpha*(data[i,2]-net)
44
45      for i in range(200):
46          temp[i] = math.fabs(w1*data[i,0]+w2*data[i,1]+b-data[i,2])**2*0.5
47
48      error.append(temp.sum()/200)
49
50      epoch += 1
```

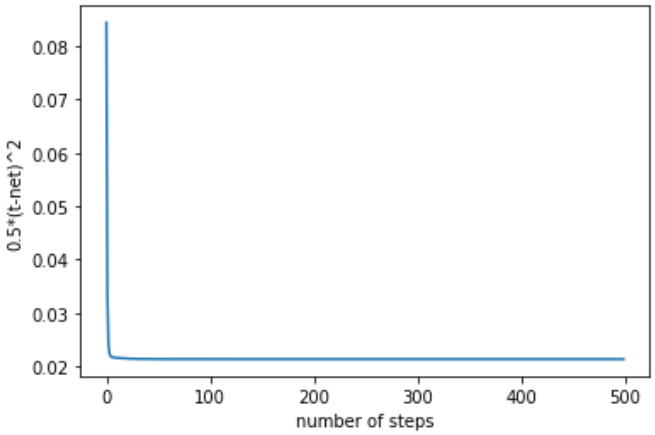
شکل 7 - برنامه سوال دوم۱

```
x = np.linspace(-2,2,3)
y = -(w1*x+b)/w2
plt.plot(x,y,'k')
x1 = data[:100,0]
y1 = data[:100,1]
x2 = data[100:,0]
y2 = data[100:,1]
plt.scatter(x1,y1)
plt.scatter(x2,y2)
plt.grid()
plt.xlim(-3,3)
plt.ylim(-3,3)
plt.xlabel('x-axis')
plt.ylabel('y-axis')
plt.figure()
plt.plot(error)
plt.xlabel('number of steps')
plt.ylabel('0.5*(t-net)^2')
```

شکل 8 - برنامه سوال دوم۲



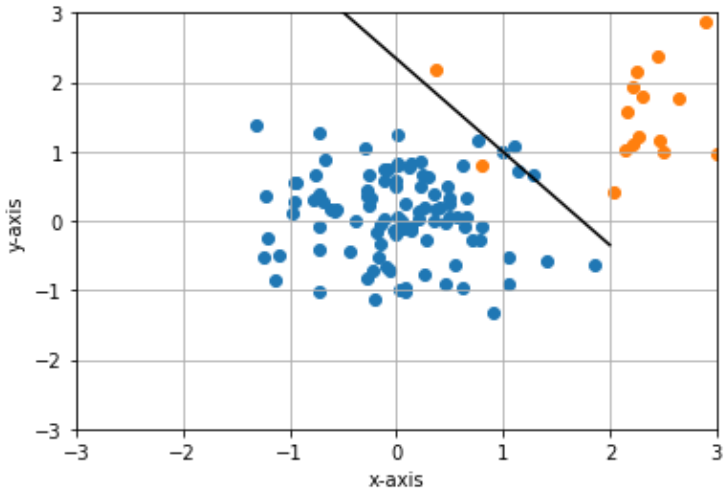
شکل 9 - نمودار نقاط و خط جداکننده **adaLine**



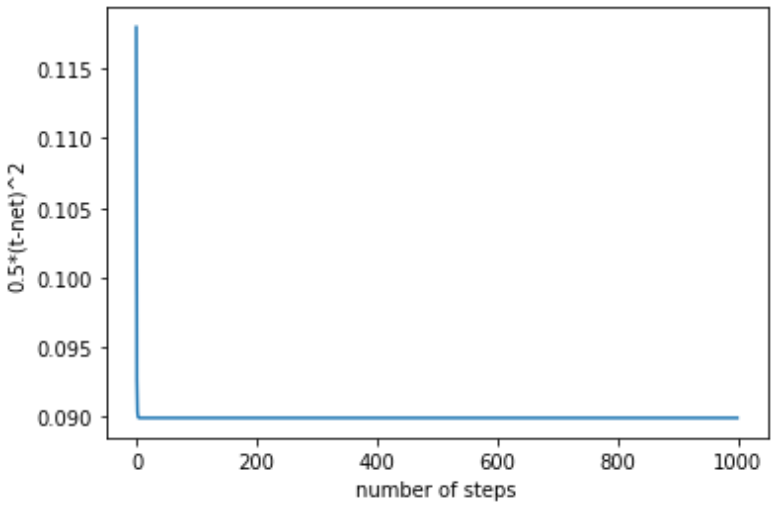
شکل 10 - نمودار خطای **adaLine**

ب) حالت دوم

نتایج آن را در زیر مشاهده می‌کنید:



شکل 11 - نمودار نقاط و خط جداکننده **adaLine** ۲



شکل 12 - نمودار خطای **adaLine** ۲

تحلیل: در حالت قسمت اول سوال، نقاط به شکل بهتر و دقیق‌تری توسط خط از هم جدا می‌شوند این می‌تواند به این دلیل باشد که علی‌رغم اینکه فاصله مراکز داده‌ها در دو حالت برابر است ولی مقدار انحراف معیار مجموعه داده‌ها در حالت دوم به مراتب بیشتر از حالت اول است و این باعث می‌شود تا داده‌ها از دو مجموعه با یکدیگر تداخل بیشتری داشته باشند و دقت دسته‌بندی کاهش و cost نسبت به حالت قبل افزایش داشته باشد. با توجه به همین توضیحات برای حالت دوم به دلیل تصادفی بودن داده‌های دو الگو، ممکن است در نهایت جایگاه نسبی آن‌ها به گونه‌ای باشد که به شکل خطی جداپذیر نباشند (مانند شکل۸) و در نتیجه این روش به حل دقیق و درست منجر نشود. یک راه حل می‌تواند این باشد که در مورد طول و عرض هر کدام از کلاس ها، داده‌هایی را که بیش از حد، از میانگین‌شان فاصله دارند، در الگوریتم وارد نکنیم. به بیان دقیق‌تر اگر $X_i \sim N(\mu, \sigma^2)$ باشد (در اینجا X_i می‌تواند مقدار طول یا عرض هر کدام از گروه‌های موردنظر باشد)، و مقدار مشاهده شده‌ی آن خارج از بازه‌ی $(\mu - 2\sigma, \mu + 2\sigma)$ قرار بگیرد، داده‌ی iام را از دیتاست خارج کنیم. البته این کار به قیمت از دست رفتن بخشی از داده‌های مساله می‌شود و حتی باز هم تضمینی بر این نیست که داده‌ها خطی جداپذیر شوند، اما احتمال بروز این پدیده را پایین می‌آورد.

MAdaLine .۱-۲

برای پیاده‌سازی این قسمت از تعریف صفحه ۹۰ کتاب فاست استفاده کردیم. در کتاب الگوریتم مربوطه به صورت زیر تعریف شده است (پیاده سازی در فایل HW1_Q2_C):

مقداردهی اولیه وزن‌ها: مقادیر وزن v_1 و v_2 و مقدار بایاس b_3 مقداردهی می‌شوند.

برای هر مجموعه داده نتیجه حاصل از اعمال activation ورودی‌ها را به‌دست می‌آوریم. (x=s)

به‌دست آوردن مقدار net (مجموع ضرب وزن‌ها در ورودی‌ها بعلاوه بایاس):

$$z_{in} = wx + b$$

به‌دست آوردن خروجی پس از اعمال تابع activation:

$$z = f(z_{in}) = \begin{cases} 1 & if\ z \geq 0 \\ -1 & if\ z < 0 \end{cases}$$

محاسبه مجموع ضرب وزن‌ها در نتایج مرحله قبل بعلاوه بایاس و خروجی آن بعد از اعمال activation

$$y_{in} = vz + b$$
$$y = f(y_{in})$$

آپدیت کردن وزن‌ها:

If $t = 1$, then update weights on Z_J,
the unit whose net input is closest to 0,

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

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

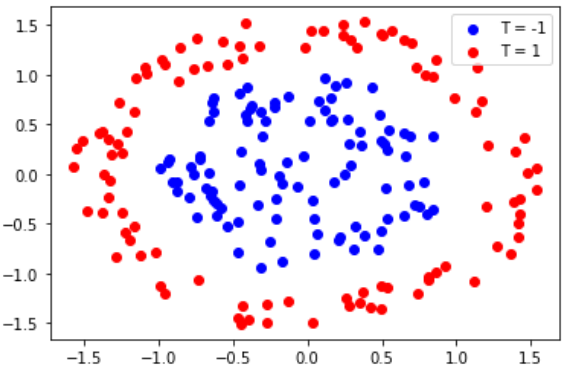
If $t = -1$, then update weights on all units
Z_k that have positive net input,

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

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

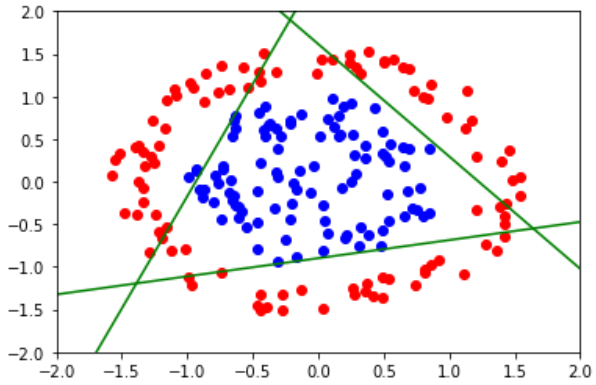
شکل 13- قوانین آپدیت کردن وزن

شرط توقف: اگر وزن‌ها تغییری نکردند یا تعداد تکرارها از حد مجاز بیشتر شد.



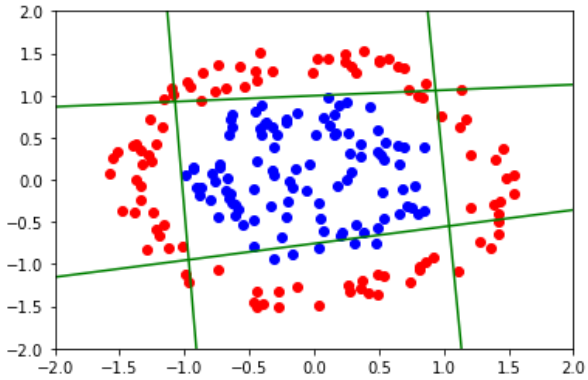
شکل 14 - نمودار پراکندگی داده خام

الف)



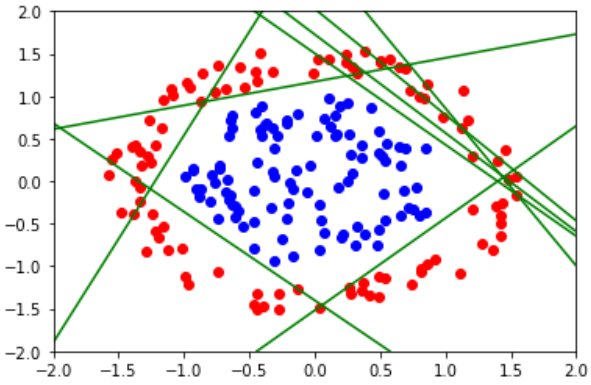
شکل 15- نمودار پراکندگی نقاط با ۳ خط جداکننده **MadaLine**

ب)



شکل 16- نمودار پراکندگی نقاط با ۴ خط جداکننده **MadaLine**

ج)



شکل 17 - نمودار پراکندگی نقاط با ۸ خط جداکننده **MadaLine**

همینطور تعداد epoch و دقت هر مرحله تست در جدول زیر آورده شده است.

جدول 2 - جدول دقت و تعداد **epoch** برای **MadaLine**

	# of Iterations	**Accuracy**
۳ خط جداکننده	470	**0.93**
۴ خط جداکننده	170	**0.96**
۸ خط جداکننده	22	**1**

تحلیل: با مقایسه نتایج بالا می بینیم که افزایش تعداد نورون ها باعث کاهش تعداد تکرار لازم الگوریتم برای همگرایی و همچنین افزایش دقت می شود، دلیل آن این است که از آنجا که داده‌ها به صورت دایره‌های هم‌مرکز هستند، با تعداد نورون (خطوط) بیشتر آزادی عمل بیشتری برای مرز بندی بین دو کلاس داریم. همچنین در قسمت 8 نورون مشاهده می‌شود که جداسازی با ۵ خط انجام شده و خطوط بعدی تاثیری ندارند فلذا چون خطا بعد آن صفر می شود و وزن سایر خط ها تغییر نمی‌کند و به عبارتی به آن ها نیاز نیست.

پاسخ ۳ – عنوان پرسش سوم به فارسی

۳-۱. عنوان بخش اول

متن نمونه

.

پاسخ ۴ – MLP

۴-۱. Multi Layer Perceptron

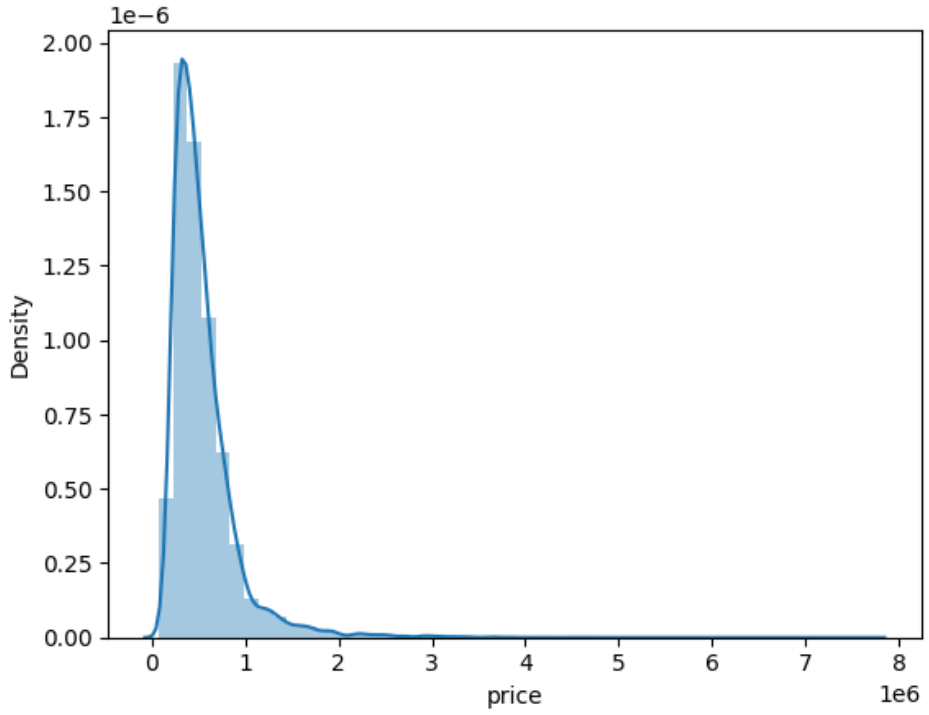
کد مربوط به این قسمت به کمک pytorch نوشته شده و در فایلی به نام HW1-Q4.ipynb ذخیره شده است. بخش‌های این قسمت به ترتیب انجام شده است و نمودارهای لازم و کدهای مربوط به هر بخش به همراه تحلیل‌های مورد نیاز در زیر آورده شده است.

A) در این بخش دیتاست را به کمک کتابخانه پانداس می‌خوانیم و به کمک تابع info() اطلاعات مورد نیاز برای شناخت داده را بدست می‌آوریم. با توجه به اطلاعات بدست آمده: دیتاست مورد استفاده ما دارای 21 ستون(features) و 21613 سطر (sample) است. داده های ما عموما از جنس int64 و float64 هستند.

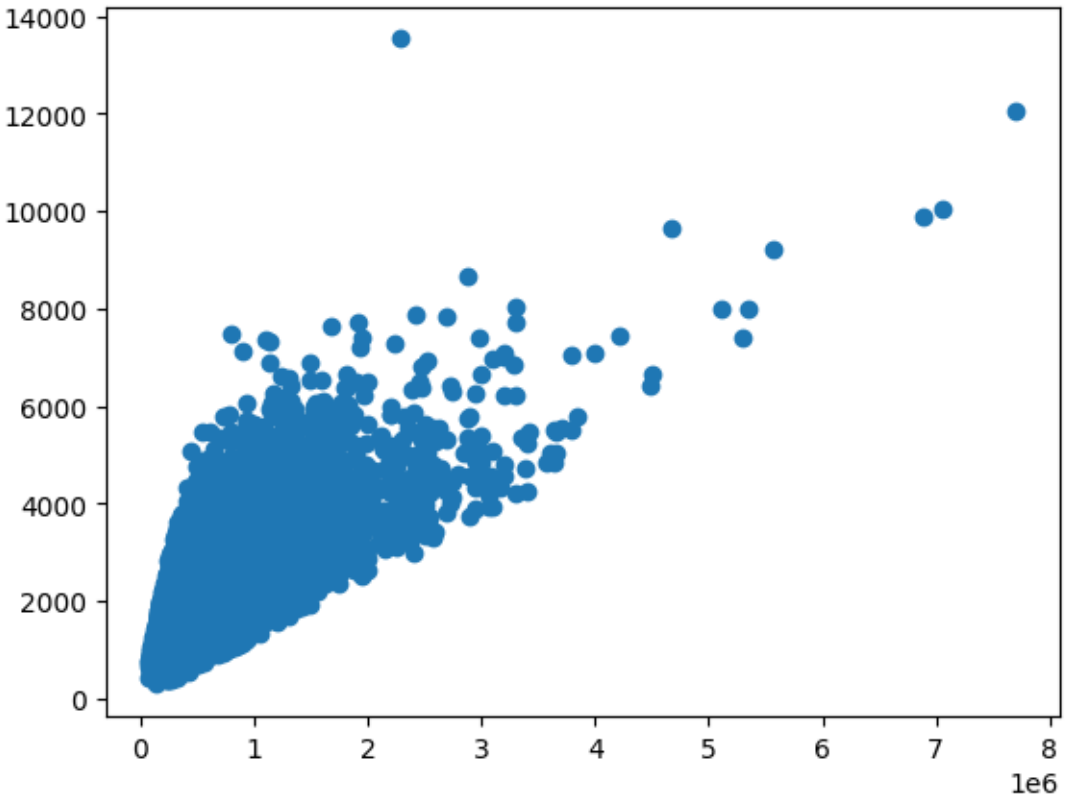
B) به کمک دستور isnull().sum() تعداد داده های NaN برای هر ستون را می‌توان بدست آورد که برای دیتاست مورد استفاده ما برای هر ستون این مقدار برابر 0 بدست آمد.

C) Correlation Matrix را به کمک تابع corr() رسم می‌کنیم، ویژگی "sqft_living " بیشترین همبستگی را با ویژگی قیمت دارد. با میزان همبستگی " 0.702035"

D) با توجه به نمودار زیر، ویژگی قیمت دارای توزیع نرمالی نمی‌باشد و کمی چولکی به راست دارد.



شکل 18 – نمودار توزیع قیمت



شکل 19 – نمودار قیمت و **'sqft_living'**

با توجه به نمودار فوق همانطور از مقدار همبستگی این دو فیچر انتظار میرفت، نمودارشان دریک راستا در حال افزایش است (رابطه ی خطی دارند).

E) به کمک قطعه کد زیر ستون date را به دو ستون سال و ماه تقسیم کرده و از دیتاست حذف می‌کنیم.

```
year = []
month = []
for item in Data.date:
    year.append(item[:4])
    month.append(item[4:6])
Data['Year'] = year
Data['Month'] = month
Data_new = Data.drop('date', axis=1)
```

شکل 20 – قطعه کد مربوط به تبدیل فیچر دیتا به فیچرهای سال و ماه و حذف آن

F) در این بخش برای جدا کردن داده های آموزش و تست تابعی به نام get_splits را در کلاس HousesDataset ایجاد کردیم، این کلاس داد هارا برای ورودی مدلمان آماده می‌کند. ٪20 داده‌ها را به داده‌های تست و 80 درصد آن‌ها را به داده‌های ترین اختصاص داده ایم

```
def get_splits(self, n_test=0.2):
    # determine sizes
    test_size = round(n_test * len(self.X))
    train_size = len(self.X) - test_size
    # calculate the split
    return random_split(self, [train_size, test_size])
```

شکل 21 – قطعه کد مربوط به تقسیم داده‌ها به داده‌های تست و ترین

G) در ابتدا به کمک کتابخانه‌ی skilearn و قطعه کد زیر، داده‌های ترین و تست را scale کردیم،

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

شکل 22- **minmax Scaler First**

H) اما با توجه به اینکه اگر داده های تست و ترین را به صورت جدا و با اسکیل داده های ترین بخواهیم اسکیل کنیم، ممکن است در داده‌های تست داده‌هایی خارج از محدوده‌ی min,max داده های ترین وجود داشته باشد که در اینصورت ignore می‌شوند. بنابراین در ابتدا تمام فیچر هارا با هم اسکیل کردیم، برای داده‌های تارگت نیز از log داده‌ها استفاده کردیم.

```
x_scaler = MinMaxScaler()
x = x_scaler.fit_transform(x)
y = y.to_numpy()
y = y.reshape(1, -1)
y = np.log(y)
```

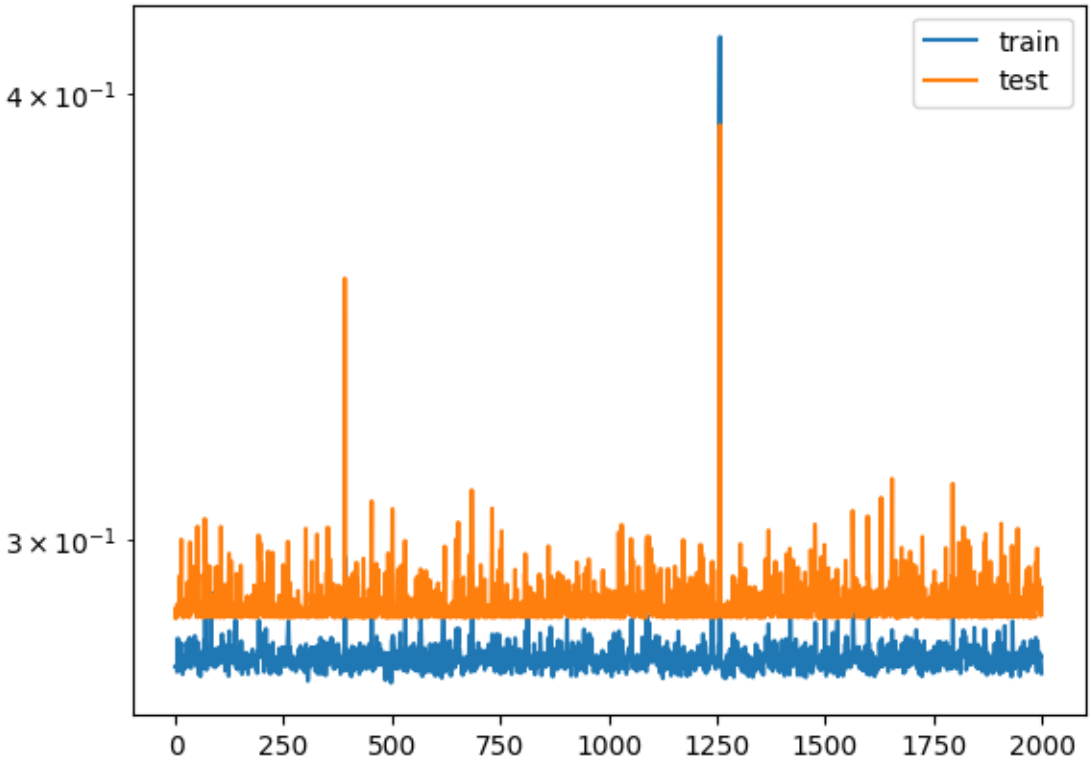
شکل 23 – **minmax Scaler Second**

I) در این قسمت یک مدل MLP با دو لایه پنهان و دو لایه ورودی و خروجی طراحی شده است که ورودی لایه ورودی به تعداد فیچر ها است که در این مسئله به خصوص برابر با 20 است. از آنجایی که مسئله از نوع رگرسیون است، خروجی لایه آخر برابر با 1 قرار داده شده است. تعداد

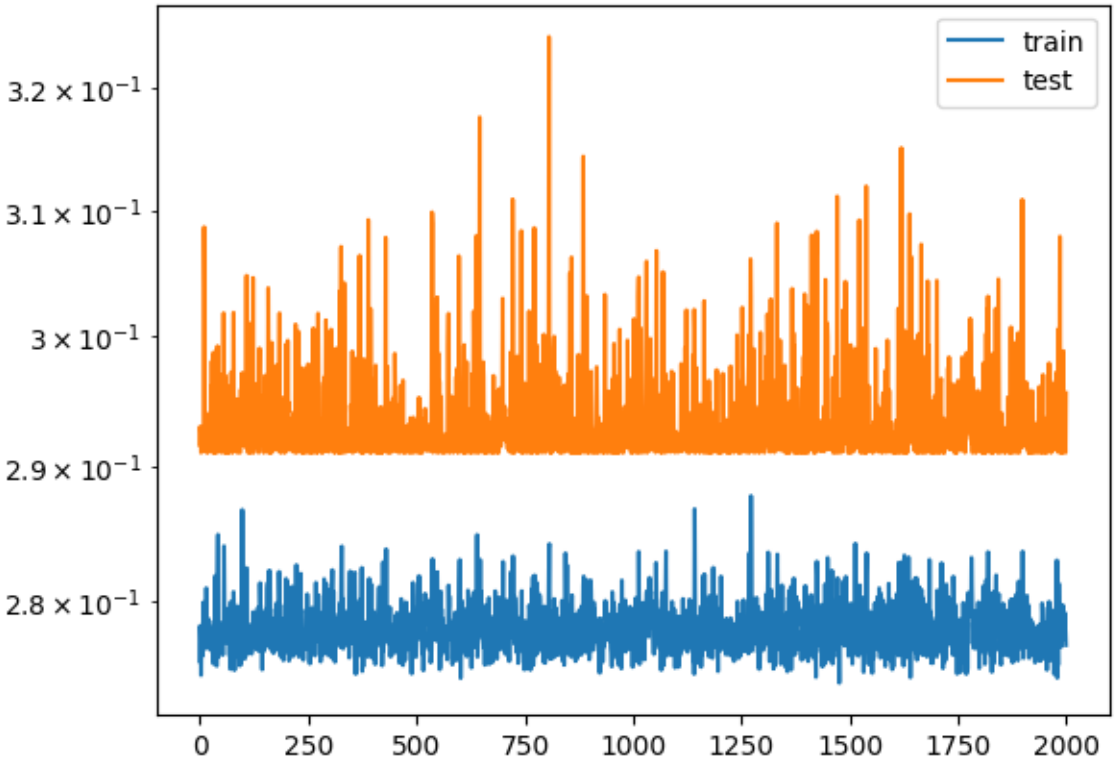
نورون های لایه های مخفی 32در نظر گرفته شده اند. تعداد epochs برابر با 2000 در نظر گرفته شده است.

دو optimizer : Adam , SGD و loss function : MSELoss , HuberLoss را در نظر گرفته ایم که نمودار های هر کدام به ترتیب زیر آورده شده اند.

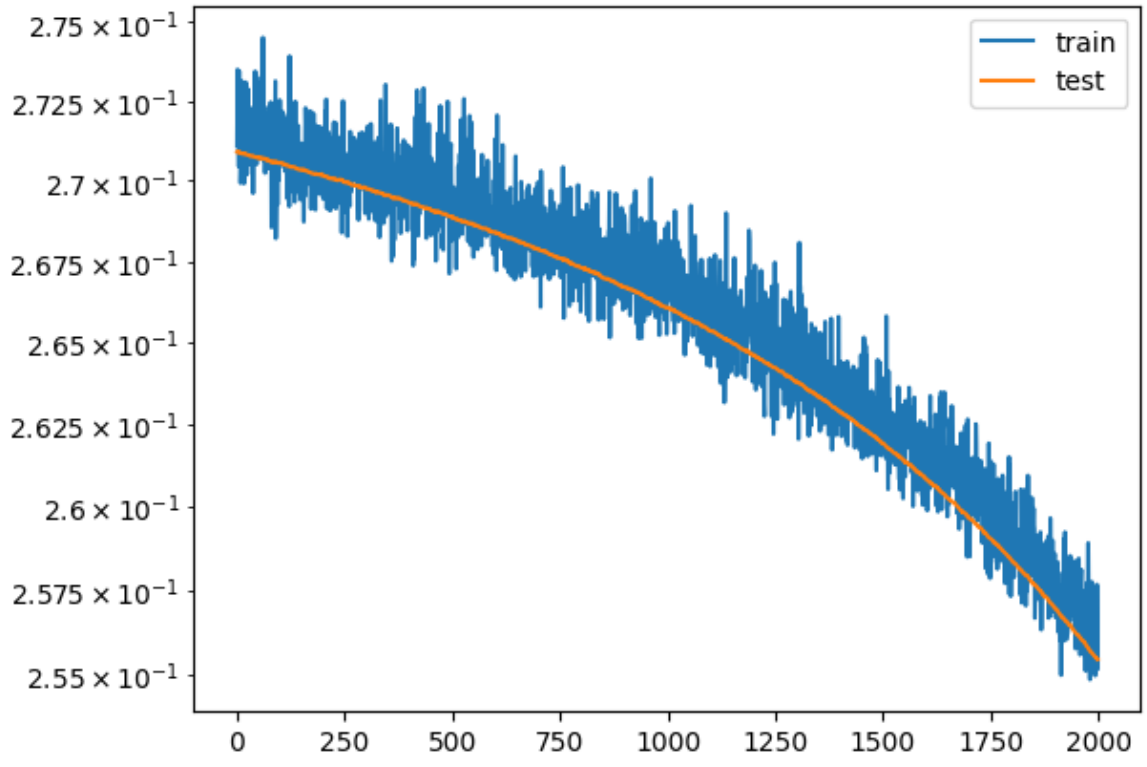
Adam , MSELoss
Adam, HuberLoss
SGD, MSELoss
SGD, HuberLoss



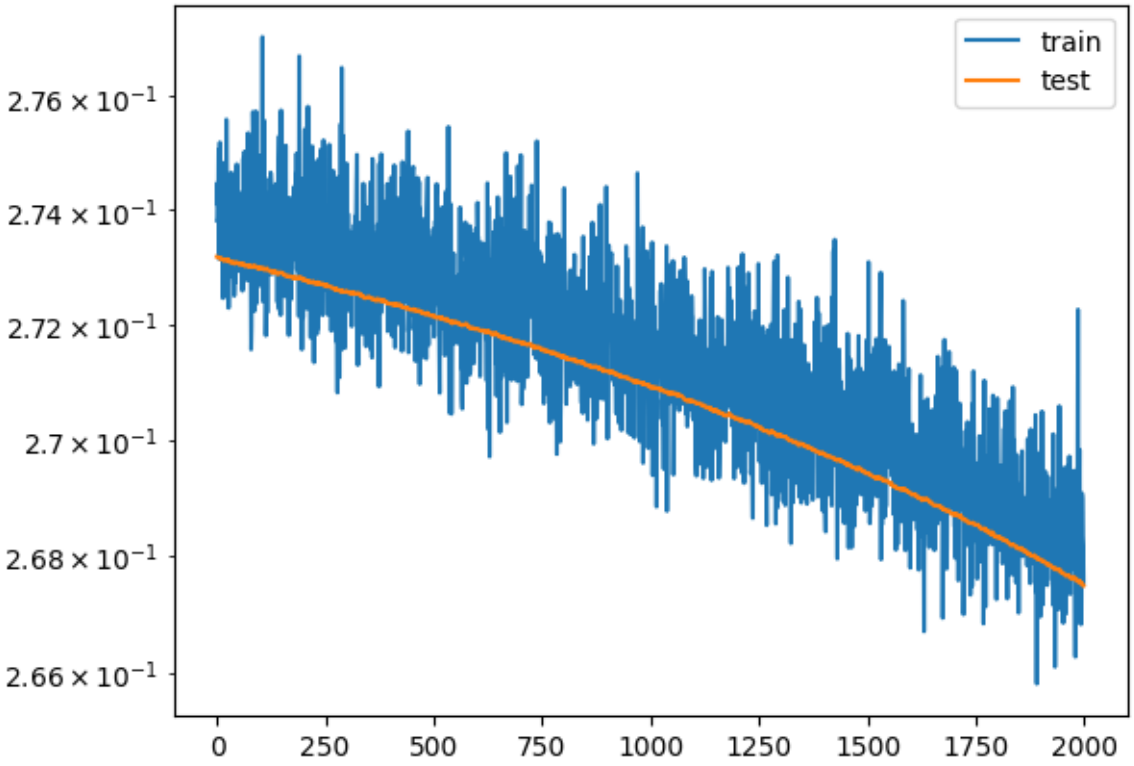
شکل **24** - نمودار **loss** و **validation loss** (**Adam , MSELoss**)



شکل **25** - نمودار **loss** و **validation loss** (**Adam, HuberLoss**)



شکل **26**- نمودار **loss** و **validation loss** (**SGD, MSELoss**)



شکل **27**- نمودار **loss** و **validation loss** (**SGD, HuberLoss**)

با توجه به نتایج بدست آمده برای دیتاست مسئله و مدل ایجاد شده optimizer: Adam و criterion: MSELoss عملکرد بهتری داشته است.

برای رسم نمودار ها از قطعه کد زیر استفاده شده است:

```
def Plot_MLP(train_dl, test_dl, optim, criter, n_epochs):
    #plot Loss and validationLoss plot
    criterion = criter
    optimizer = optim
    train_losses = np.zeros(n_epochs)
    test_losses = np.zeros(n_epochs)
    for it in range(n_epochs):
        train_loss = []
        for i, (inputs, targets) in enumerate(train_dl):

            optimizer.zero_grad( )
            outputs = model(inputs)
            loss = criterion(outputs, targets)
            loss.backward()
            optimizer.step()

            train_loss.append(loss.item())
        train_loss = np.mean(train_loss)
        test_loss = []
        for i, (inputs, targets) in enumerate(test_dl):
            outputs = model(inputs)
            loss = criterion(outputs, targets)
            test_loss.append(loss.item())
        test_loss = np.mean(test_loss)
        train_losses[it] = train_loss
        test_losses[it] = test_loss

        print(f'epoch {it+1}/{n_epochs}, train loss: {train_loss:.4f}, test loss: {test_loss:.4f}')
    plt.semilogy(train_losses, label='train')
    plt.semilogy(test_losses, label='test')
    plt.legend()
    plt.show()
```

شکل 28 – قطعه کد مربوط به رسم نمودار

برای پیشبینی قیمت خانه توسط مدل طراحی شده از قطعه کد زیر استفاده شده است:

```
# make a class prediction for one row of data
def predict(row, model):
    # convert row to data
    row = Tensor([row])
    # make prediction
    yhat = model(row)
    # retrieve numpy array
    yhat = yhat.detach().numpy()
    return yhat
for i in range(5):
    data = Data_new.loc[random.randint(0, 21613)]
    y = data['target']
    x = data.drop('target')
    yhat = predict(x, model)
    print('Predicted: %.3f' % yhat, y)
```

شکل 29 – قطعه کد مربوط به پیشبینی قیمت

قیمت واقعی و قیمت پیشبینی شده بر حسب Log بدست آمده در زیر آمده است.

:Adam , MSELoss

```
Predicted: 13.102 y:13.64709190633931
```

```
Predicted: 13.102 y:13.444446876573442
Predicted: 13.102 y:12.736553826954605
Predicted: 13.102 y:13.42984807715229
Predicted: 13.102 y:13.422467969854667
```

:Adam, HuberLoss

```
Predicted: 13.045 y:13.616937660584268
Predicted: 13.045 y:13.502168738731916
Predicted: 13.045 y:13.122363377404328
Predicted: 13.045 y:13.199324418540456
Predicted: 13.045 y:12.206072645530174
```

: SGD, MSELoss

```
Predicted: 13.062 y:13.742939865129438
Predicted: 13.048 y:13.058358047428417
Predicted: 13.046 y:13.681979165339751
Predicted: 13.053 y:13.304601597392535
Predicted: 13.040 y:12.871334622600584
```

:SGD, HuberLoss

```
Predicted: 13.057 y:13.681979165339751
Predicted: 13.048 y:13.071070083016778
Predicted: 13.049 y:12.969212197910155
Predicted: 13.055 y:13.62918097977278
Predicted: 13.047 y:12.611371073081239
```

.