



به نام خدا  
دانشگاه تهران  
دانشکده مهندسی برق و کامپیوتر



## درس شبکه‌های عصبی و یادگیری عمیق

### تمرین اول

نام و نام خانوادگی	مهسا راستی - مینا سیدی
شماره دانشجویی	810198393-810398084
تاریخ ارسال گزارش	1401.07.10

## فهرست

پاسخ 1. شبکه عصبی Mucclloch-Pitts	5
1-1. ضرب کننده باینری دو بیتی	5
پاسخ 2 - AdaLina and MadaLine	7
2-1. AdaLine	7
2-2. MadaLine	12
پاسخ 4 - MLP	17
4-1. Multiple Layer Perceptron	17

## شکل‌ها

- شکل 1. پیاده سازی نوروں M&P ..... 5
- شکل 2. ساختار ضرب کننده باینری دو بیتی ..... 5
- شکل 3. Thereshold و لیست ورودی‌های شبکه ..... 6
- شکل 4. پیاده سازی ضرب کننده باینری دوبیتی ..... 6
- شکل 5. خروجی‌های شبکه ضرب کننده باینری دوبیتی ..... 6
- شکل 6. پیاده‌سازی ساخت متغیرهای تصادفی اول ..... 7
- شکل 7. پیاده‌سازی ترسیم plot متغیرهای تصادفی اول ..... 7
- شکل 8. Plot متغیرهای تصادفی اول ساخته شده ..... 7
- شکل 9. پیاده سازی تابع activation\_func ..... 8
- شکل 10. پیاده سازی شبکه AdaLine ..... 9
- شکل 11. طراحی شبکه AdaLine ..... 9
- شکل 12. پیاده سازی ترسیم plot جداسازی داده‌ها ..... 10
- شکل 13. plot جداسازی داده‌های تصادفی با شبکه AdaLine ..... 10
- شکل 14. پیاده‌سازی ترسیم plot خطاها ..... 10
- شکل 15. plot خطاهای متغیر تصادفی اول ..... 10
- شکل 16. plot جداسازی داده‌های تصادفی دوم با شبکه AdaLine ..... 11
- شکل 17. plot خطاهای متغیر تصادفی دوم ..... 11
- شکل 18. پیاده سازی تابع activation\_func ..... 12
- شکل 19. پیاده سازی شبکه MadaLine ..... 13
- شکل 20. وارد کردن داده‌ها و ترسیم plot ..... 13
- شکل 21. ترسیم plot مربوط به داده‌های MadaLine ..... 13
- شکل 22. آموزش نوروں MadaLine ..... 14
- شکل 23. plot نوروں MadaLine با 3 نوروں ..... 15
- شکل 24. plot خطاهای نوروں MadaLine با 3 نوروں ..... 15
- شکل 25. plot نوروں MadaLine با 4 نوروں ..... 15
- شکل 26. plot خطاهای نوروں MadaLine با 4 نوروں ..... 15
- شکل 27. plot نوروں MadaLine با 8 نوروں ..... 16

- شکل 28. plot خطاهای نورون MadaLine با 8 نورون ..... 16
- شکل 29. وارد کردن داده به برنامه ..... 17
- شکل 30. نمایش داده‌های Nan ..... 17
- شکل 31. Correction Matrix داده‌ها ..... 18
- شکل 32. پیاده سازی ترسیم نمودار توزیع قیمت خانه و نمودار قیمت خانه و Sqft\_living ..... 18
- شکل 33. نمودار توزیع قیمت خانه ..... 19
- شکل 34. نمودار قیمت و Sqft\_living ..... 19
- شکل 35. جدا کردن ماه و سال ..... 19
- شکل 36. جدا کردن داده های آموزش و تست ..... 19
- شکل 37. Scale کردن داده‌های تست و آموزش ..... 20
- شکل 38. پیاده سازی MLP با 3 لایه ..... 20
- شکل 39. تارگت داده‌های train و test ..... 20
- شکل 40. Optimizer های Adam و ASGD ..... 20
- شکل 41. محاسبه مقدار Loss ..... 21
- شکل 42. پیاده‌سازی ترسیم Loss – Plot ..... 21
- شکل 43. نمودار Adam Optimizer و MSELoss ..... 21
- شکل 44. نمودار ASGD Optimizer و L1Loss ..... 22
- شکل 45. تفاوت مقدار پیشبینی شده قیمت خانه با مقدار واقعی ..... 22



## پاسخ ۱. شبکه عصبی Muccloch-Pitts

### ۱-۱. ضرب کننده باینری دو بیتی

در این بخش می‌خواهیم با کمک نورون Muccloch-Pitts توسعه یافته، یک ضرب کننده باینری بسازیم که ورودی دوبیتی را گرفته و آنها را در هم ضرب کند و خروجی چهاربیتی داشته باشد. برای این کار ابتدا نورون را به صورت زیر طراحی می‌کنیم:

$$net = x_1w_1 + \dots + x_iw_i + \dots + x_nw_n$$

$$h = \begin{cases} 1 & net \geq \theta \\ 0(-1) & net < \theta \end{cases}$$

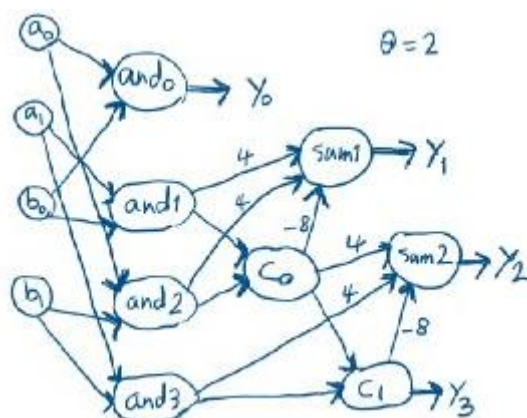
پیاده سازی این نورون، به صورت زیر انجام می‌شود:

```
def MP_neuron(theta, x1, x2, w1, w2, b, x3=0, w3=0):
    y = x1*w1 + x2*w2 + x3*w3 + b
    if(y >= theta):
        return 1
    else:
        return 0
```

شکل ۱. پیاده سازی نورون M&P

برای ساخت ضرب کننده باینری که شرایط گفته شده را داشته باشد، از ساختار زیر کمک

می‌گیریم:



شکل ۲. ساختار ضرب کننده باینری دو بیتی

ورودی‌های این نورون لیستی از همه حالت‌هایی از دو ورودی دوبیتی است که در هم ضرب خواهند شد:

```

0s ▶ List = [[0, 0, 0, 0], [0, 0, 1, 0], [0, 0, 0, 1], [0, 0, 1, 1],
              [0, 1, 0, 0], [0, 1, 0, 1], [0, 1, 1, 0], [0, 1, 1, 1],
              [1, 0, 0, 0], [1, 0, 0, 1], [1, 0, 1, 0], [1, 0, 1, 1],
              [1, 1, 0, 0], [1, 1, 0, 1], [1, 1, 1, 0], [1, 1, 1, 1]]
theta=2

```

شکل 3. Thershold و لیست ورودی‌های شبکه

ساختار نشان داده شده در تصویر بالا را به صورت زیر پیاده‌سازی می‌کنیم:

```

0s ▶ for i in range(len(List)):
      a0, a1, b0, b1 = List[i]

      and0 = MP_neuron(theta,a0,b0,1,1,0)
      and1 = MP_neuron(theta,a1,b0,1,1,0)
      and2 = MP_neuron(theta,a0,b1,1,1,0)
      and3 = MP_neuron(theta,a1,b1,1,1,0)

      c0 = MP_neuron(theta,and1,and2,1,1,0)
      c1 = MP_neuron(theta,and1,and2,1,1,0)

      sum1 = MP_neuron(theta,and1,and2,4,4,0,c0,-8)
      sum2 = MP_neuron(theta,c0,and3,4,4,0,c1,-8)

      y0 = and0
      y1 = sum1
      y2 = sum2
      y3 = c1

      print(a0, a1, ' * ', b0, b1, ' = ', y3, y2, y1, y0)

```

شکل 4. پیاده سازی ضرب کننده باینری دوبیتی

خروجی‌های شبکه به صورت زیر خواهد شد:

```

0 0 * 0 0 = 0 0 0 0
0 0 * 1 0 = 0 0 0 0
0 0 * 0 1 = 0 0 0 0
0 0 * 1 1 = 0 0 0 0
0 1 * 0 0 = 0 0 0 0
0 1 * 0 1 = 0 1 0 0
0 1 * 1 0 = 0 0 1 0
0 1 * 1 1 = 0 1 1 0
1 0 * 0 0 = 0 0 0 0
1 0 * 0 1 = 0 0 1 0
1 0 * 1 0 = 0 0 0 1
1 0 * 1 1 = 0 0 1 1
1 1 * 0 0 = 0 0 0 0
1 1 * 0 1 = 0 1 1 0
1 1 * 1 0 = 0 0 1 1
1 1 * 1 1 = 1 0 0 1

```

شکل 5. خروجی‌های شبکه ضرب کننده باینری دوبیتی

## پاسخ ۲ – AdaLina and MadaLine

### ۱-۲. AdaLine

ابتدا با کتابخانه random، دو دسته داده 100 تایی می‌سازیم که متغیر x و y دسته اول میانگین 1 و انحراف معیار 0.3 و دسته دوم میانگین -1 و انحراف معیار 0.3 داشته باشد:

```
# generating data
x1 = np.random.normal(1, 0.3, 100)
y1 = np.random.normal(1, 0.3, 100)
input0 = np.column_stack((x1, y1))
t0 = np.zeros(100) + 1

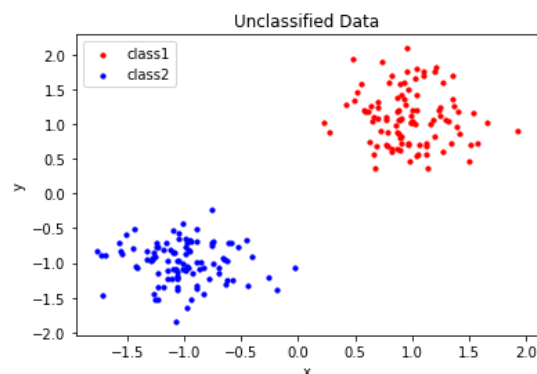
x2 = np.random.normal(-1, 0.3, 100)
y2 = np.random.normal(-1, 0.3, 100)
input1 = np.column_stack((x2, y2))
t1 = np.zeros(100) - 1
```

شکل 6. پیاده‌سازی ساخت متغیرهای تصادفی اول

سپس نمودار پراکندگی آنها را رسم می‌نماییم:

```
#plotting data sets
plt.figure()
ax1 = plt.axes()
ax1.scatter(x1, y1, s=10, c='r', marker="o", label='class1')
ax1.scatter(x2, y2, s=10, c='b', marker="o", label='class2')
plt.legend(loc='upper left')
plt.title("Unclassified Data")
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```

شکل 7. پیاده‌سازی ترسیم plot متغیرهای تصادفی اول



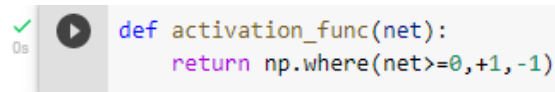
شکل 8. Plot متغیرهای تصادفی اول ساخته شده



ابتدا به معرفی شبکه AdaLine می‌پردازیم:

$$net = \sum_{i=1}^n x_i w_i + b$$
$$h = f(net) = \begin{cases} 1 & net \geq \theta \\ -1 & net < \theta \end{cases}$$

برای پیاده سازی این شبکه ابتدا یک تابع به نام activation\_func می‌سازیم تا وظیفه  $f(net)$  شبکه را انجام دهد:

A screenshot of a code editor showing a Python function definition. The function is named 'activation\_func' and takes 'net' as an argument. It returns a numpy array where elements are 1 if 'net' is greater than or equal to 0, and -1 otherwise. The code is: 

```
def activation_func(net):  
    return np.where(net>=0,+1,-1)
```

شکل 9. پیاده سازی تابع activation\_func

برای طراحی شبکه، کلاس AdaLine را می‌سازیم و در آن تابع fit و predict قرار می‌دهیم. در تابع fit، مطابق رابطه ذکر شده در معرفی شبکه AdaLine مقدار  $net$  و سپس  $f(net)$  را محاسبه می‌کنیم. سپس به محاسبه دلتا ( $\delta$ ) با نرخ آموزش ( $\alpha$ ) کوچک می‌پردازیم:

$$\delta = \alpha \times (Y - f(net))$$

سپس وزن‌ها و بایاس را در هر مرحله بروزرسانی می‌کنیم:

$$\omega_{new} = \omega_{old} + \alpha(t - net)x$$

$$b_{new} = b_{old} + \alpha(t - net)$$

در نهایت طبق خواسته سوال، مقدار خطا را با رابطه زیر محاسبه می‌کنیم:

$$loss_{new} = loss_{old} + \sqrt{Y^2 - f(net)^2}$$

```

class AdaLine:
    def __init__(self, learning_rate=0.1, epochs=100):
        self.learning_rate=learning_rate
        self.epochs=epochs
        self.activation_func=activation_func
        self.weights=None
        self.bias=None
        self.losses=[]

    def fit(self, X, Y):
        samples, inputs= np.shape(X)
        self.weights=np.zeros(inputs)
        self.bias=0

        for epoch in range(self.epochs):
            los=0
            for idx, x_i in enumerate(X):
                net = np.dot(x_i, self.weights) + self.bias
                y_predicted = self.activation_func(net)
                #update rule
                delta = self.learning_rate * (Y[idx] - y_predicted)
                self.weights += delta * x_i
                self.bias += delta

                los+=0.5*(Y[idx] - y_predicted)**2
            #error
            self.losses.append(los)

    def predict(self, X):
        net = np.dot(X, self.weights) + self.bias
        y_predicted = self.activation_func(net)
        return y_predicted

```

شکل 10. پیاده سازی شبکه AdaLine

برای جدا کردن دو دسته متغیر تصادفی تولید شده، ابتدا اعداد تولیدی را کنار هم قرار می‌دهیم. سپس شبکه AdaLine با نرخ یادگیری 0.01 و تعداد نمونه 100 ریاال طراحی می‌کنیم. سپس اعداد تولیدی را به تابع fit معرفی می‌کنیم:

```

train_X=np.concatenate((input0,input1))
train_Y=np.concatenate((t0,t1))

#question 2
#train the model
l=AdaLine(0.1,100)
l.fit(train_X,train_Y)

```

شکل 11. طراحی شبکه AdaLine

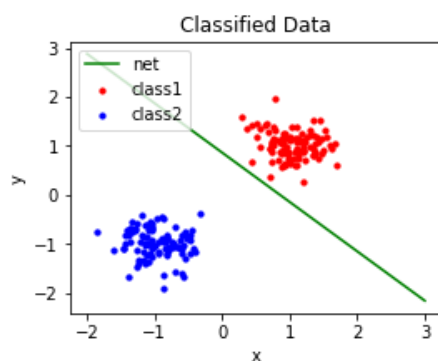
داده‌ها به صورت زیر از یکدیگر جدا می‌شوند:

```

✓ 1s #plotting results
plt.figure(figsize=(4, 3))
ax2 = plt.axes()
ax2.scatter(x1, y1, s=10, c='r', marker="o", label='class1')
ax2.scatter(x2, y2, s=10, c='b', marker="o", label='class2')
x=np.arange(-2,4,1)
ax2.plot(x,(-1.weights[0]*x-1.bias)/1.weights[1], c='g', label='net')
plt.legend(loc='upper left')
plt.title("Classified Data")
plt.xlabel('x')
plt.ylabel('y')
plt.show()

```

شکل 12. پیاده سازی ترسیم plot جداسازی داده‌ها



شکل 13. plot جداسازی داده‌های تصادفی با شبکه AdaLine

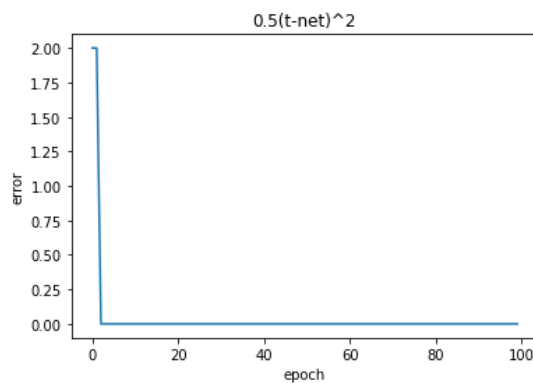
نمودار تغییرات خطاها به صورت زیر است:

```

✓ 2s #plotting errors
plt.figure()
plt.plot(l.losses)
plt.title("0.5(t-net)^2")
plt.xlabel('epoch')
plt.ylabel('error')
plt.show()

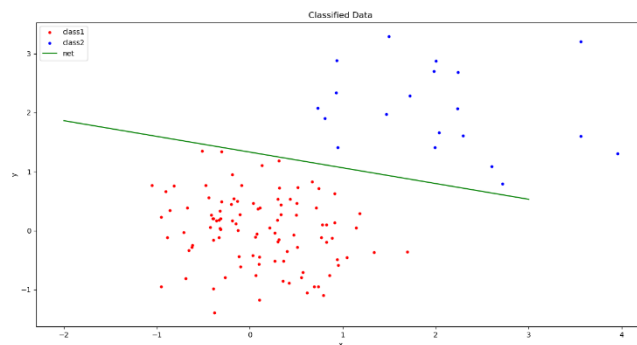
```

شکل 14. پیاده سازی ترسیم خطاها

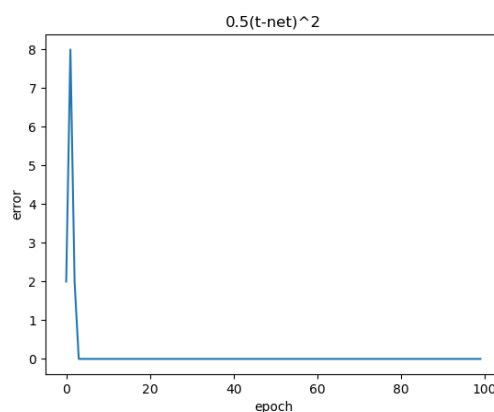


شکل 15. plot خطاهای متغیر تصادفی اول

عملیات قبلی بار دیگر با اعداد متفاوت تکرار می‌کنیم. این بار متغیرهای  $x$  و  $y$  دسته اول را با میانگین 0 و انحراف معیار 0.6 و دسته دوم را با میانگین 2 و انحراف معیار 0.8 به دست می‌آوریم. جدا سازی با شبکه AdaLine به صورت زیر درخواهد آمد:



شکل 16. plot جداسازی داده‌های تصادفی دوم با شبکه AdaLine



شکل 17. plot خطاهای متغیر تصادفی دوم

Plot خطاها برای متغیرهای تصادفی اول نشان می‌دهد که شبکه از همان ابتدا، به صورت کامل جداسازی دو دسته اعداد را انجام داده است زیرا این دو دسته همپوشانی با یکدیگر ندارند. در plot متغیرهای تصادفی دوم مشاهده می‌کنیم جداسازی به خوبی انجام نشده و error نوساناتی در هر آپک دارد. زیرا انحراف معیار دو دسته زیاد است و پراکندگی دو دسته از اعداد بگونه‌ای است که نمیتوان مرز مشخصی بین آنها قرار داد. در نتیجه جداسازی به صورت کامل انجام نمی‌شود. همچنین نوروں AdaLine برای داده‌هایی که پراکندگی متفاوتی دارند، نتیجه درستی نمی‌دهد به همین دلیل جداسازی به درستی انجام نمی‌شود.

## MadaLine .۲-۲

میدانیم نورون MadaLine، از ترکیب چند نورون AdaLine ساخته شده است. به همین دلیل تفاوت ساختاری زیادی با نورون AdaLine نخواهد داشت. همانند قسمت قبلی، تابع `activation_func` را پیاده‌سازی می‌کنیم:

```
def activation_func(net):
    result = []
    for i in range(len(net)):
        if net[i] >= 0:
            result.append(1)
        else:
            result.append(-1)
    return result
```

شکل 18. پیاده‌سازی تابع `activation_func`

برای طراحی شبکه MadaLine به ترتیب مقابل عمل می‌کنیم. همانند AdaLine، در تابع `fit` ابتدا مقدار `net` و  $f(net)$  را محاسبه می‌کنیم. اگر اختلاف  $f(net)$  و  $y$  برابر صفر نباشد، مقدار وزن‌ها و بایاس را برزورسانی می‌کنیم و درنهایت مقدار خطا را محاسبه می‌نماییم:

```
class MadaLine:
    def __init__(self, learning_rate=0.1, iterations=100):
        self.learning_rate = learning_rate
        self.iterations = iterations
        self.activation_func = activation_func
        self.hlayer_w = None
        self.hlayer_b = None
        self.and_layer_w = None
        self.and_layer_b = None
        self.err = []

    def fit(self, X, Y):
        for epoch in range(self.iterations):
            cost=0
            for i in range(len(X)):
                net_hlayer = np.dot(self.hlayer_w, X[i])+self.hlayer_b
                net_outlayer=np.dot(self.and_layer_w,self.activation_func(net_hlayer))+self.and_layer_b
                y_predicted=np.where(net_outlayer >= 0, +1, -1)

                # update rule
                error=Y[i]-y_predicted
                if error != 0 :
                    if Y[i]==1:
                        idw=np.argmax(abs(net_hlayer))
                        delta=Y[i]-net_hlayer[idw]
                        update=np.zeros((len(self.hlayer_w),2))
                        update[idw,0]=self.learning_rate * X[i][0]*delta
                        update[idw,1]=self.learning_rate * X[i][1]*delta
                        self.hlayer_w += update
                        self.hlayer_b[idw] += delta
```

```

        elif V[i]==-1:
            for j in range(len(net_hlayer)):
                if net_hlayer[j]>0:
                    delta=V[i]-net_hlayer[j]
                    update=np.zeros((len(self.hlayer_w),2))
                    update[j,0]=self.learning_rate * X[j][0]*delta
                    update[j,1]=self.learning_rate * X[j][1]*delta
                    self.hlayer_w += update_
                    self.hlayer_b[j] += delta

            cost += 0.5*(error**2)

        self.err.append(cost)

def init_paremters(self,neuron_num):
    random_gen1 = np.random.RandomState(10)
    self. hlayer_w = np.random.normal(0, 0.01, (neuron_num, 2))
    random_gen2 = np.random.RandomState(20)
    self.hlayer_b = np.random.normal(0, 0.01, neuron_num)
    self.and_layer_w= np.ones((neuron_num))
    self.and_layer_b= neuron_num-1

```

شکل 19. پیاده سازی شبکه MadaLine

ابتدا داده‌ها را وارد کرده و plot آن را ترسیم می‌کنیم:

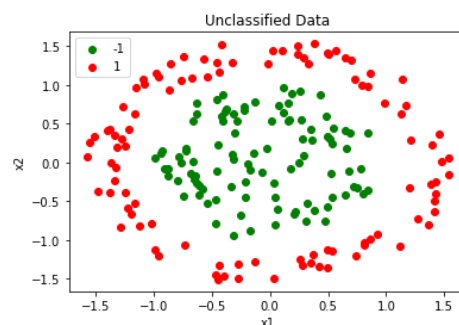
```

x1 = np.array(pd.read_csv("MadaLine.csv", usecols = [0]))
x2 = np.array(pd.read_csv("MadaLine.csv", usecols = [1]))
y = np.array(pd.read_csv("MadaLine.csv", usecols = [2]))

plt.figure()
ax1 = plt.axes()
x11=[]
x22=[]
y1=[]
y2=[]
for i in range(len(y)):
    if y[i] == 0:
        y1.append(x2[i])
        x11.append(x1[i])
    else:
        y2.append(x2[i])
        x22.append(x1[i])
ax1.scatter(x11, y1, c='g', marker='o', label='-1')
ax1.scatter(x22, y2, c='r', marker='o', label='1')
plt.legend(loc='upper left')
plt.title("Unclassified Data")
plt.xlabel('x1')
plt.ylabel('x2')
plt.show()

```

شکل 20. وارد کردن داده‌ها و ترسیم plot



شکل 21. ترسیم plot مربوط به داده‌های MadaLine

به ازای تعداد نورون 3 و 4 و 8، نورون MadaLine را آموزش داده و plot خطاهای آن را ترسیم می‌کنیم:

```

4s 4s for num_of_neurons in [3, 4, 8]:
    model=MadaLine(0.2,200)
    model.init_paremeters(num_of_neurons)
    model.fit(x,y)

    plt.plot(model.err)
    plt.title(num_of_neurons)
    plt.xlabel('epoch')
    plt.ylabel('error')
    plt.show()

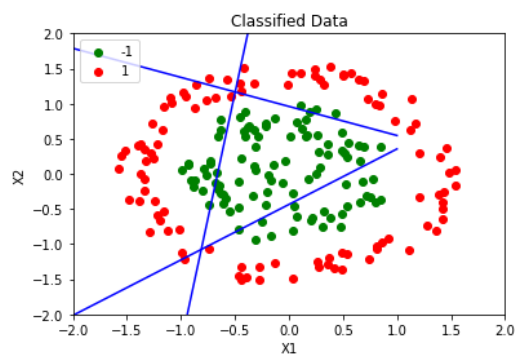
    domain=np.arange(-2,2,1)
    Line1=[]
    Line2=[]
    Line3=[]
    Line4=[]
    Line5=[]
    Line6=[]
    Line7=[]
    Line8=[]

    for i in range (len(domain)):
        Line1.append(-domain[i]*(model.hlayer_w[0][0]/model.hlayer_w[0][1])-model.hlayer_b[0]/model.hlayer_w[0][1])
        Line2.append(-domain[i]*(model.hlayer_w[1][0]/model.hlayer_w[1][1])-model.hlayer_b[1]/model.hlayer_w[1][1])
        Line3.append(-domain[i]*(model.hlayer_w[2][0]/model.hlayer_w[2][1])-model.hlayer_b[2]/model.hlayer_w[2][1])
        if num_of_neurons>3:
            Line4.append(-domain[i]*(model.hlayer_w[3][0]/model.hlayer_w[3][1])-model.hlayer_b[3]/model.hlayer_w[3][1])
        if num_of_neurons==8:
            Line5.append(-domain[i]*(model.hlayer_w[4][0]/model.hlayer_w[4][1])-model.hlayer_b[4]/model.hlayer_w[4][1])
            Line6.append(-domain[i]*(model.hlayer_w[5][0]/model.hlayer_w[5][1])-model.hlayer_b[5]/model.hlayer_w[5][1])
            Line7.append(-domain[i]*(model.hlayer_w[6][0]/model.hlayer_w[6][1])-model.hlayer_b[6]/model.hlayer_w[6][1])
            Line8.append(-domain[i]*(model.hlayer_w[7][0]/model.hlayer_w[7][1])-model.hlayer_b[7]/model.hlayer_w[7][1])

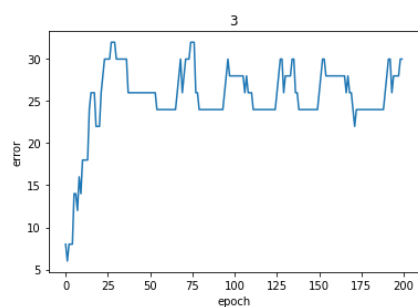
plt.figure()
ax2 = plt.axes()
x11=[]
x22=[]
y1=[]
y2=[]
for i in range(len(y)):
    if y[i] == -1:
        y1.append(x2[i])
        x11.append(x1[i])
    else:
        y2.append(x2[i])
        x22.append(x1[i])
ax2.scatter(x11, y1, c='g', marker='o', label='-1')
ax2.scatter(x22, y2, c='r', marker='o', label='1')
ax2.plot(domain,Line1,'b')
ax2.plot(domain,Line2,'b')
ax2.plot(domain,Line3,'b')
if num_of_neurons>3:
    ax2.plot(domain,Line4,'b')
if num_of_neurons==8:
    ax2.plot(domain,Line5,'b')
    ax2.plot(domain,Line6,'b')
    ax2.plot(domain,Line7,'b')
    ax2.plot(domain,Line8,'b')
plt.legend(loc='upper left')
plt.title("Classified Data")
plt.xlabel('X1')
plt.ylabel('X2')
plt.ylim([-2,2])
plt.xlim([-2,2])
plt.show()

```

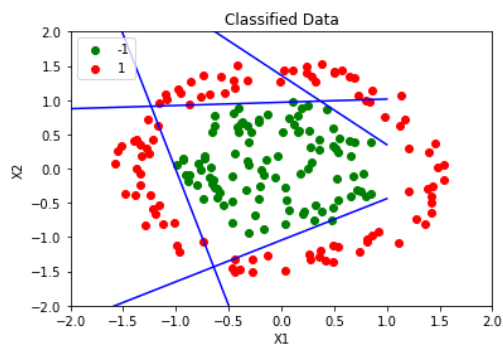
شکل 22. آموزش نورون MadaLine



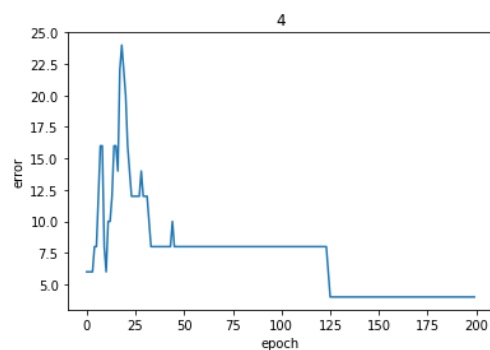
شکل 23. plot نورون MadaLine با 3 نورون



شکل 24. plot خطاهای نورون MadaLine با 3 نورون

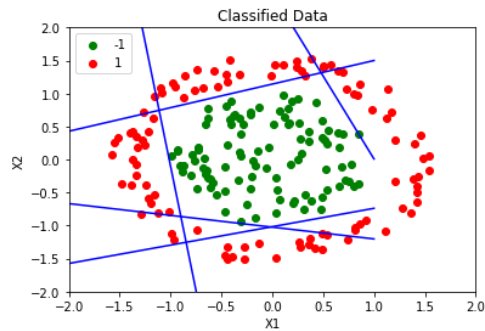


شکل 25. plot نورون MadaLine با 4 نورون

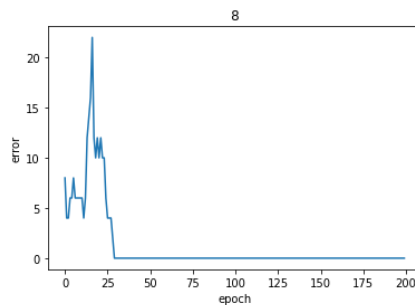


شکل 26. plot خطاهای نورون MadaLine با 4 نورون





شکل 27. plot نورون MadaLine با 8 نورون



شکل 28. plot خطاهای نورون MadaLine با 8 نورون

با افزایش تعداد نورون در شبکه، دقت جداسازی در اپک های کمتری افزایش می یابد. با 3 نورون، جداسازی به صورت درستی انجام نمی شود. به گونه ای که در اپکهای بالاتر نیز خطا صفر نمی شود. اما با 4 نورون در اپک 125 و با 8 نورون در اپک 25، خطا به سمت صفر میل می کند.

الگوریتم MRI، شکل اصلی شبکه MadaLine است که در آن وزن های لایه های پنهان تنظیم می شوند و وزن واحد خروجی ثابت است. اما در الگوریتم MR II، همه وزن ها را در net تنظیم می کند.

## ۴-۱. Multiple Layer Perceptron

با کمک کتابخانه Pandas، داده را وارد برنامه می‌کنیم:

```

✓ 0s #A
hdata=pd.read_csv("houses.csv")
print(hdata.info())

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21613 entries, 0 to 21612
Data columns (total 21 columns):
#   Column              Non-Null Count  Dtype
---  -
0   id                   21613 non-null  int64
1   date                 21613 non-null  object
2   price                21613 non-null  float64
3   bedrooms             21613 non-null  int64
4   bathrooms            21613 non-null  float64
5   sqft_living          21613 non-null  int64
6   sqft_lot             21613 non-null  int64
7   floors               21613 non-null  float64
8   waterfront           21613 non-null  int64
9   view                 21613 non-null  int64
10  condition            21613 non-null  int64
11  grade                21613 non-null  int64
12  sqft_above           21613 non-null  int64
13  sqft_basement        21613 non-null  int64
14  yr_built              21613 non-null  int64
15  yr_renovated          21613 non-null  int64
16  zipcode               21613 non-null  int64
17  lat                   21613 non-null  float64
18  long                  21613 non-null  float64
19  sqft_living15         21613 non-null  int64
20  sqft_lot15            21613 non-null  int64
dtypes: float64(5), int64(15), object(1)
memory usage: 3.5+ MB
None

```

شکل 29. وارد کردن داده به برنامه

داده‌هایی که Nan هستند به صورت زیر نمایش داده می‌شوند:

```

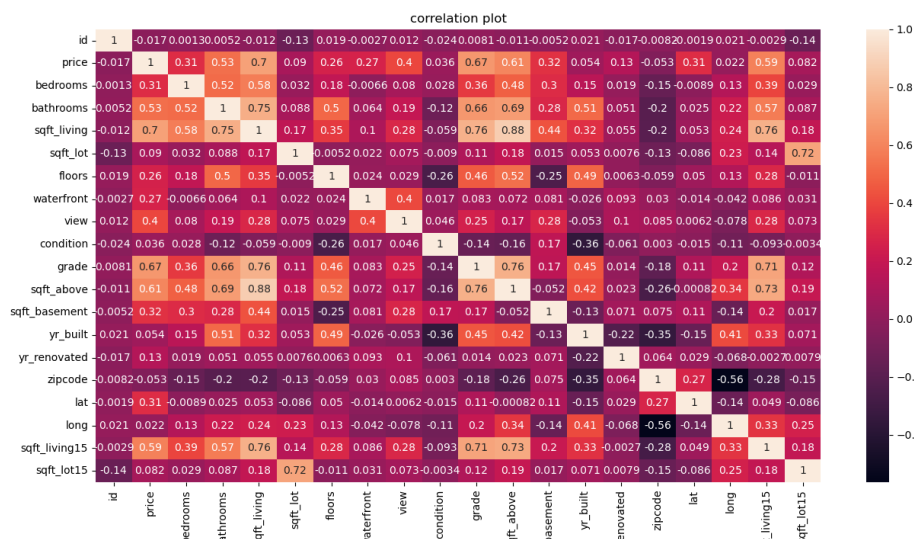
✓ 6s [41] #B
hdata.dropna(axis='columns',inplace=True)
invalid_rows = [index for index, row in hdata.iterrows() if row.isnull().any()]
print(invalid_rows)

```

شکل 30. نمایش داده‌های Nan

Correction Matrix داده‌ها به صورت زیر است:

```
#C
corr_matrix = hdata.corr()
sns.heatmap(corr_matrix, annot=True)
plt.title("correlation plot")
plt.show()
```



شکل 31. Correction Matrix داده‌ها

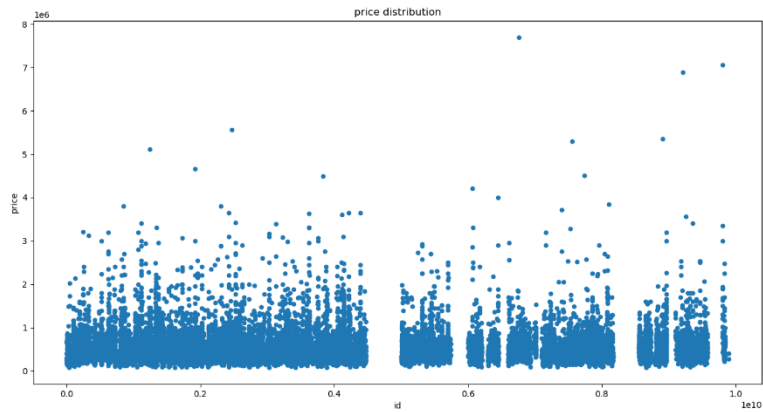
مشاهده می‌شود Sqft\_living بیشترین Correction را با price دارد.

نمودار توزیع قیمت و نمودار قیمت و Sqft\_living که بیشترین همبستگی را با price دارد به صورت زیر است:

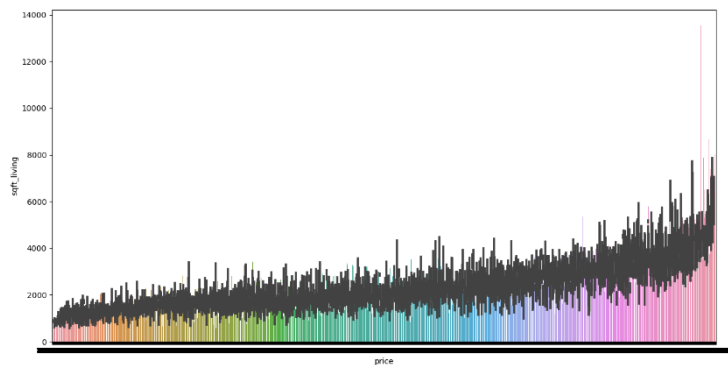
```
#D
hdata.plot.scatter('id', 'price')
plt.title("price distribution")
plt.show()

sns.barplot(x = 'price', y = 'sqft_living', data = hdata)
plt.xlabel("price")
plt.ylabel("sqft_living")
plt.show()
```

شکل 32. پیاده سازی ترسیم نمودار توزیع قیمت خانه و نمودار قیمت خانه و Sqft\_living



شکل 33. نمودار توزیع قیمت خانه



شکل 34. نمودار قیمت و Sqft\_living

بدین صورت ستون Date به دو ستون ماه و سال تبدیل می‌شود:

```
#E
hdata.drop('id', inplace=True, axis=1)
hdata['year'], hdata['month1'] = hdata['date'].str[:4], hdata['date'].str[4:]
hdata.drop('date', inplace=True, axis=1)

hdata['month'], hdata['trash'] = hdata["month1"].str.split("T", n = 1, expand = True)
hdata.drop('month1', inplace=True, axis=1)
hdata.drop('trash', inplace=True, axis=1)
```

شکل 35. جدا کردن ماه و سال

داده‌های تست و آموزش را از هم جدا می‌کنیم:

```
#F
train, test = train_test_split(hdata, test_size=0.2)
```

شکل 36. جدا کردن داده‌های آموزش و تست

داده‌های آموزش و تست را scale می‌کنیم:

```
✓ 0s #G
scaler = MinMaxScaler()
train_scaled = scaler.fit_transform(train)
test_scaled=scaler.transform(test)
```

شکل 37. Scale کردن داده‌های تست و آموزش

پیاده سازی MLP با 3 لایه را به صورت زیر انجام می‌دهیم:

```
✓ 0s #H
model=nn.Sequential(
    nn.Linear(20,100),
    nn.ReLU(),
    nn.Linear(100,50),
    nn.ReLU(),
    nn.Linear(50,1)
)
```

شکل 38. پیاده سازی MLP با 3 لایه

تارگت داده‌های آموزش و تست را با torch.tensor معرفی می‌کنیم:

```
targets = train_scaled[:, 0]
targets = torch.tensor(targets, dtype=torch.float32)
inputs = train_scaled[:, 1:]
inputs = torch.tensor(inputs, dtype=torch.float32)

test_targets = test_scaled[:, 0]
test_targets = torch.tensor(test_targets, dtype=torch.float32)
test_inputs = test_scaled[:, 1:]
test_inputs = torch.tensor(test_inputs, dtype=torch.float32)
```

شکل 39. تارگت داده‌های train و test

از Optimizerهای Adam و ASGD استفاده می‌کنیم:

```
for optim_loss in [0,1]:
    if optim_loss==0:
        optim = torch.optim.Adam(model.parameters(), lr = learning_rate)
        loss_func = nn.MSELoss()
    elif optim_loss==1:
        optim=torch.optim.ASGD(model.parameters(), lr = learning_rate)
        loss_func=nn.L1Loss()
```

شکل 40. Optimizerهای Adam و ASGD

برای Adam Optimizer از MSELoss و برای ASGD Optimizer از L1Loss استفاده می‌کنیم:

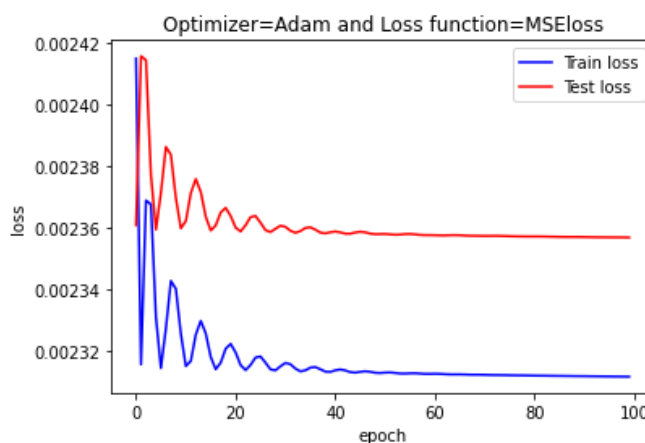
```
train_loss = []
test_loss = []
for epoch in range(epochs):
    optim.zero_grad()
    out = model(inputs)
    loss = loss_func(out, targets)
    loss.backward()
    train_loss.append(float(loss.item()))
    optim.step()
    # print(f'epoch',epoch)
    # print(f'train loss:',loss.item())
    with torch.no_grad():
        tests_out=model(test_inputs)
        los=loss_func(tests_out, test_targets)
        # print(f'test loss:',los.item())
        test_loss.append(float(los.item()))
```

شکل 41. محاسبه مقدار Loss

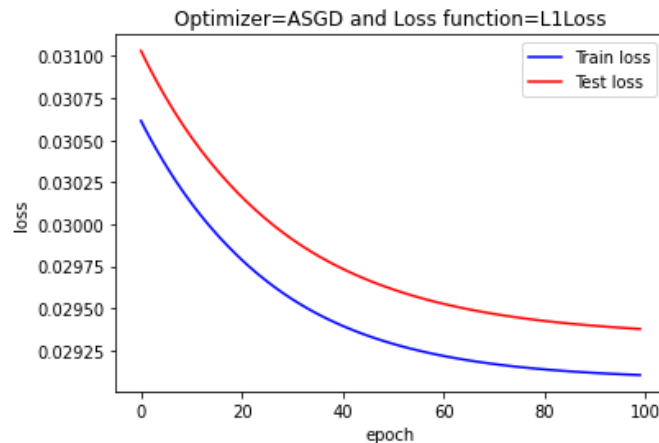
نمودار Loss بر حسب هر اپک را برای Optimizerها و Loss های مختلف رسم می‌کنیم:

```
plt.figure()
ax1 = plt.axes()
ax1.plot(train_loss,c='b',label="Train loss")
ax1.plot(test_loss,c='r',label='Test loss')
plt.legend()
plt.xlabel("epoch")
plt.ylabel("loss")
if optim_loss==0: plt.title("Optimizer=Adam and Loss function=MSEloss")
elif optim_loss==1: plt.title("Optimizer=ASGD and Loss function=L1Loss")
plt.show()
```

شکل 42. پیاده‌سازی ترسیم Plot - Loss



شکل 43. نمودار Adam Optimizer و MSELoss



شکل 44. نمودار Optimizer ASGD و L1Loss

مقدار Loss در MSELoss بسیار کوچکتر از L1Loss است. زیرا مقدار Loss بین 0 و 1 قرار دارد و در هر ایک به توان دوم رسیده و کوچکتر می‌شود. اما مقدار Loss در train و test در حالت استفاده از تابع L1Loss تفاوت کمتری در حالت استفاده از MSELoss دارد. در AdamOptimizer، مقدار Loss به صورت نوسانی کاهش می‌یابد اما در ASGDOptimizer به صورت نمایی کاهش پیدا می‌کند.

5 داده تصادفی از بین داده‌های تست انتخاب کرده قیمت خانه پیشبینی شده را با مقدار واقعی آن مقایسه می‌کنیم. تفاوت مقدار پیشبینی شده با مقدار واقعی به صورت زیر است:

```
#k
for i in range(5):
    indice=np.random.randint(0,len(test_inputs))
    print(f"test data",i)
    print(f"Real_Price = ",test_targets[indice],f"Predicted_Price = ",tests_out[indice])
    print(f"Differences = ",test_targets[indice]-tests_out[indice])
```

```
test data 0
Real_Price = tensor(0.0970) Predicted_Price = tensor([0.0510])
Differences = tensor([0.0461])
test data 1
Real_Price = tensor(0.0502) Predicted_Price = tensor([0.0514])
Differences = tensor([-0.0013])
test data 2
Real_Price = tensor(0.0234) Predicted_Price = tensor([0.0496])
Differences = tensor([-0.0262])
test data 3
Real_Price = tensor(0.0275) Predicted_Price = tensor([0.0533])
Differences = tensor([-0.0258])
test data 4
Real_Price = tensor(0.0433) Predicted_Price = tensor([0.0489])
Differences = tensor([-0.0056])
```

شکل 45. تفاوت مقدار پیشبینی شده قیمت خانه با مقدار واقعی