به نام خدا

دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

درس شبکه‌های عصبی و یادگیری عمیق

تمرین اول

نام و نام خانوادگی	حمید نعمتی – مهرداد نوربخش
شماره دانشجویی	810100492 - 810100495
تاریخ ارسال گزارش	۱۴۰۱.۰۷.۰۱

فهرست

شکل‌ها

پاسخ 1. شبکه عصبی McCullough-Pitts

۱-۱. ضرب کننده باینری دو بیتی

الف) در اسلاید ها و کتاب با ساخت انواع گیت‌های منطقی با نورون‌های MP اشنا شدیم.

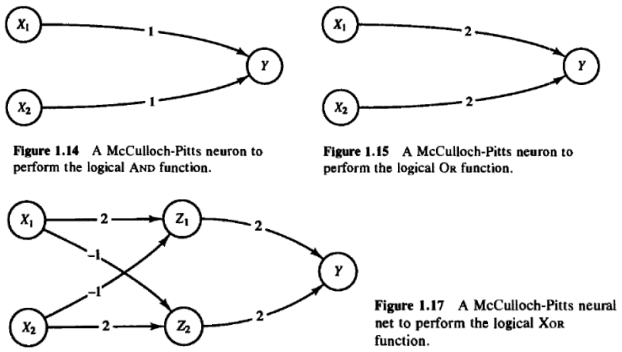


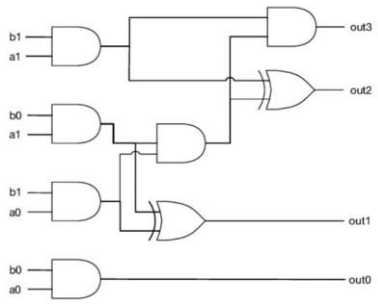
Figure 1.14 A McCulloch-Pitts neuron to perform the logical AND function.

Figure 1.15 A McCulloch-Pitts neuron to perform the logical OR function.

Figure 1.17 A McCulloch-Pitts neural net to perform the logical XOR function.

شکل 1- گیت‌های منطقی ساده **AND** و **OR** و **XOR** ساخته شده با نورون‌های **MP**

با ترکیب این دانش با انچه در درس مدار منطقی خوانده‌ایم میتوان یک ضرب کننده‌ی دوبیتی ساخت که در شکل زیر آمده است:



شکل 2- مدار سازنده‌ی ضرب کننده دو بیتی

طبق شکل بالا بیت صفر خروجی(کم ارزش ترین بیت) وقتی یک میشود که بیت صفر هر دو ورودی یک باشد. بیت اول خروجی زمانی یک میشود که یا بیت صفر ورودی اول صفر و بیت یکم ورودی دوم یک شود یا بیت صفر ورودی دوم صفر و بیت یکم ورودی اول یک شود. بیت دوم خروجی زمانی یک میشود که بیت صفر ورودی اول یک و بیت یکم ورودی دوم یک شود یا بیت اول هر دو ورودی یک باشد. بیت پر ارزش خروجی هم زمانی یک است که بیت صفر و یک هر دو ورودی یک باشد.

برای اطمینان از درستی شکل بالا جدول درستی را هم بررسی میکنیم. بعدا ازین جدول برای بررسی خروجی خود استفاده میکنیم:

x_0	x_1	x_2	x_3	f_0	f_1	f_2	f_3
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0
0	0	1	0	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	0	0	1
0	1	1	0	0	0	1	0
0	1	1	1	0	0	1	1
1	0	0	0	0	0	0	0
1	0	0	1	0	0	1	0
1	0	1	0	0	1	0	0
1	0	1	1	0	1	1	0
1	1	0	0	0	0	0	0
1	1	0	1	0	0	1	1
1	1	1	0	0	1	1	0
1	1	1	1	1	0	0	1

شکل 3- جدول درستی ضرب کننده دو بیتی

ب) آدرس کولب حاوی کدها:

https://colab.research.google.com/drive/1-GEm1OQKmGfMrOcUEKRff-tFBABA1pvx?usp=sharing

ابتدا کلاسی برای نورون MP میسازیم:

```
[22] import numpy as np
     import itertools

     class MccullochPitts:
         def __init__(self, pos_weight, neg_weight, excitatory_list, activation_threshold=0):
             """
             A connection path is excitatory if the weight on the path is positive; other-
             wise it is inhibitory. All excitatory connections into a particular neuron have
             the same weights.
             (Laurene V. Fausett p27)
             """
             self.pos_weight = pos_weight
             self.neg_weight = neg_weight
             self.activation_threshold = activation_threshold
             self.excitatory_list = excitatory_list

         def activation_function(self, g):
             """
             The activation of a McCulloch-Pitts neuron is binary. That is, at any time
             step, the neuron either fires (has an activation of 1) or does not fire (has an
             activation of 0).
             Each neuron has a fixed threshold such that if the net input to the neuron
             is greater than the threshold, the neuron fires.
             (Laurene V. Fausett p26, 27)
             """
             return g >= self.activation_threshold

         def forward(self, inputs: list):
             assert(len(inputs) == len(self.excitatory_list))
             sum = 0
             for input, excitatory in zip(inputs, self.excitatory_list):
                 if excitatory:
                     sum += input * self.pos_weight
                 else:
                     sum += input * self.neg_weight
             return self.activation_function(sum)
```

شکل 4- کلاس نورون **MP**

حال گیت‌های منطقی را بازسازی میکنیم طبق تعریف:

```
[23] class AndGate(MccullochPitts):
         def __init__(self):
             super().__init__(pos_weight=1, neg_weight=0, excitatory_list=[1, 1], activation_threshold=2)

     class OrGate(MccullochPitts):
         def __init__(self):
             super().__init__(pos_weight=1, neg_weight=0, excitatory_list=[1, 1], activation_threshold=1)

     class XorGate():
         def forward(self, inputs: list):
             z1 = MccullochPitts(pos_weight=2, neg_weight=-1, excitatory_list=[1, 0], activation_threshold=2).forward(inputs)

             z2 = MccullochPitts(pos_weight=2, neg_weight=-1, excitatory_list=[0, 1], activation_threshold=2).forward(inputs)

             y  = MccullochPitts(pos_weight=2, neg_weight= 0, excitatory_list=[1, 1], activation_threshold=2).forward([z1, z2])

             return y
```

شکل 5- گیت‌های منطقی **AND** و **OR** و **XOR** ساخته شده توسط نورون **MP**

حال تابعی میسازیم که خود ضرب کننده‌ی دوبیتی را شبیه‌سازی کند:

```
[24] def binary_multiplier(a1, a0, b1, b0):

         tmp0 = AndGate().forward([a0, b0])
         tmp1 = AndGate().forward([a0, b1])
         tmp2 = AndGate().forward([a1, b0])
         tmp3 = AndGate().forward([a1, b1])
         tmp2_2 = AndGate().forward([tmp1, tmp2])

         Out0 = tmp0
         Out1 = XorGate().forward([tmp1  , tmp2])
         Out2 = XorGate().forward([tmp2_2, tmp3])
         Out3 = AndGate().forward([tmp2_2, tmp3])

         return Out0, Out1, Out2, Out3
```

شکل 6- ضرب کننده‌ی دوبیتی ساخته شده شبیه سازی شده با گیت های منطقی

سپس با کمک itertools یک حلقه‌ی تودرتو میسازیم که دقیقا همان ورودی‌های ما را شبیه سازی کند:

itertools.**product**(**iterables*, *repeat=1*)

Cartesian product of input iterables.

Roughly equivalent to nested for-loops in a generator expression. For example, `product(A, B)` returns the same as `((x,y) for x in A for y in B)`.

The nested loops cycle like an odometer with the rightmost element advancing on every iteration. This pattern creates a lexicographic ordering so that if the input's iterables are sorted, the product tuples are emitted in sorted order.

To compute the product of an iterable with itself, specify the number of repetitions with the optional *repeat* keyword argument. For example, `product(A, repeat=4)` means the same as `product(A, A, A, A)`.

شکل 7- تعریف تابع **product** در کتابخانه **itertools**

حال که ورودی‌ها را ساخته‌ایم و ضرب کننده‌ی دوبیتی را هم ساخته ایم وقت آن است که خروجی ها را بگیریم و با جدول درستی بالا مقایسه کنیم.

در اینجا O0 پرارزش‌ترین بیت و O3 کم ارزش ترین بیت است. به عبارت ساده تر بیت سمت چپ خروجی چاپ شده ارزش بیشتری را داراست، هم در ورودی هم در خروجی.

مبینیم که خروجی چاپ شده دقیقا با جدول درستی بالا همخوانی دارد پس ضرب کننده دوبیتی ما به درستی کار میکند.

```
[25] binaries = list(itertools.product([0, 1], repeat=4))
     print(" A1 A0 B1 B0 -> O0  O1  O2  O3")
     for b in binaries:
         Out0, Out1, Out2, Out3 = binary_multiplier(*b)
         print(b, "  ", Out3*1, ' ', Out2*1, ' ',Out1*1,  ' ',Out0*1)
```

```
 A1 A0 B1 B0 -> O0  O1  O2  O3
(0, 0, 0, 0)    0   0   0   0
(0, 0, 0, 1)    0   0   0   0
(0, 0, 1, 0)    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   0   0   1
(0, 1, 1, 0)    0   0   1   0
(0, 1, 1, 1)    0   0   1   1
(1, 0, 0, 0)    0   0   0   0
(1, 0, 0, 1)    0   0   1   0
(1, 0, 1, 0)    0   1   0   0
(1, 0, 1, 1)    0   1   1   0
(1, 1, 0, 0)    0   0   0   0
(1, 1, 0, 1)    0   0   1   1
(1, 1, 1, 0)    0   1   1   0
(1, 1, 1, 1)    1   0   0   1
```

شکل 8- خروجی ضرب کننده‌ی دوبیتی روی تمام ورودی های ممکن

تمام خواسته‌های سوال یک انجام شد و در اینجا سوال یک به پایان میرسد.

پاسخ ۲ – AdaLine and MadaLine

۲-۱. AdaLine

الف)

از کتابخانه numpy استفاده میکنیم و دو دسته داده با توزیع نرمال شدح داده شده میسازیم:

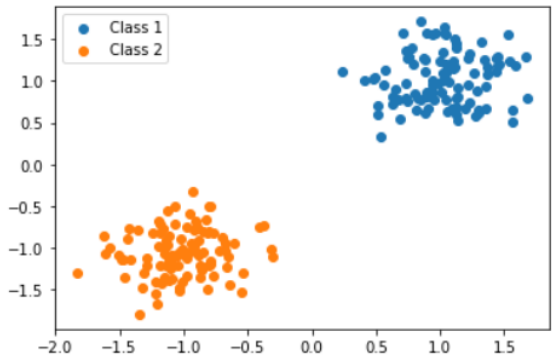
2.1

```
[ ]  import matplotlib.pyplot as plt
     import numpy as np
```

a

```
np.random.seed(0)

c1_x = np.random.normal(loc=1 , scale=0.3, size=100)
c1_y = np.random.normal(loc=1 , scale=0.3, size=100)
c2_x = np.random.normal(loc=-1, scale=0.3, size=100)
c2_y = np.random.normal(loc=-1, scale=0.3, size=100)

plt.scatter(x=c1_x, y=c1_y, label="Class 1")
plt.scatter(x=c2_x, y=c2_y, label="Class 2")

plt.legend()
```

```
<matplotlib.legend.Legend at 0x7f11c0e05fd0>
```



شکل 9- نمودار پراکندگی دو دسته داده شرح داده شده

ب)

ابتدا طبق اسلایدها و کتاب درسی کلاس نورون AdaLine را تعریف میکنیم. همانطور که دیده میشود از Shuffle بهره گرفته ایم که فرایند آموزش را تسریع میبخشد. از tqdm هم استفاده کردیم که تعداد epoch هایی که اموزش طول میکشد را ببینیم. مقدار$1e-10$ را به عنوان stop_tolerance به صورت تجربی بعد ار چند بار آزمایش انتخاب کردیم. Learning rate را هم بعد از چند با تست شدن های مختلف از اعداد 0.01 تا 1 به مقدار 0.1 رسیدیم. سایر قسمت‌های کد شبیه کتاب هستند:

```python
from tqdm.auto import tqdm
from sklearn.utils import shuffle

class Adaline():
    def __init__(self, in_features=2, lr=0.1, stop_tolerance=1e-10):
        """
        Initialization (Laurene V. Fausett p82)
        """
        self.weights        = np.random.random(in_features)
        self.bias           = np.random.random()
        self.lr             = lr
        self.stop_tolerance = stop_tolerance
        self.loss_history   = []

    def forward(self, xi):
        """
        Compute net: y_in = b + E xi.wi
        """
        net = self.bias + np.dot(xi, self.weights)

        return net

    def step(self, xi, net, target):
        """
        Update bias and weights, i = 1, . . . , n:
        b(new)  = b(old)  + a(t − y_in).
        wi(new) = wi(old) + a(t − y_in)xi . (y_in=net)
        """
        grad          = target  - net
        self.bias    += self.lr * grad
        self.weights += self.lr * grad * xi

        return max( abs(self.lr * grad), max(abs(self.lr * grad * xi)) )

    def calc_loss(self, targets, nets):
        return np.mean( 1/2 * ( np.array(targets) - np.array(nets) )**2 )
```

```python
    def train(self, X, Y, max_epochs=10):
        """
        If the largest weight change that occurred in Step 2 is
        smaller than a specified tolerance, then stop; otherwise continue.
        """
        self.loss_history = []
        max_change = 10e12
        X, Y = shuffle(X, Y, random_state=0)

        for epoch in tqdm(range(max_epochs)):
            targets, nets = [], []

            for x, y in zip(X, Y):
                net             = self.forward(x)
                step_max_change = self.step(x, net, y)
                max_change      = min(max_change, step_max_change)

                targets .append(y)
                nets    .append(net)

            self.loss_history.append(self.calc_loss(targets, nets))
            if max_change < self.stop_tolerance:
                print(max_change, self.stop_tolerance)
                break

    def activation_function(self, g):
        return 1 if g >= 0 else -1

    def predict(self, X):
        Y = []
        for x in X:
            net    = self.forward(x)
            output = self.activation_function(net)
            Y.append( output )

        return np.array(Y)
```

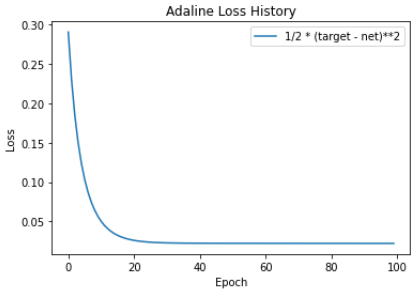
شکل 10- پیاده سازی **AdaLine**

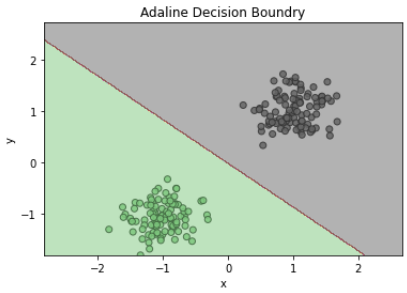
```python
def plot_loss(self):
    plt.plot(self.loss_history, label="1/2 * (target - net)**2")
    plt.legend()
    plt.title("Adaline Loss History")
    plt.xlabel("Epoch")
    plt.ylabel("Loss")
    plt.show()

def plot_scatter(self, X, Y):
    plt.scatter(
        X[:, 0],
        X[:, 1],
        c          = Y,
        cmap       = 'Accent',
        edgecolors = "k",
        alpha      = 0.85,
    )
    h = 0.01  # step size in the mesh
    xx, yy = np.meshgrid(np.arange(min(X[:, 0]) - 2, max(X[:, 0]), h) + 1,
                         np.arange(min(X[:, 1]) - 1, max(X[:, 1]), h) + 1)

    # Plot the decision boundary
    Z = self.predict(np.c_[xx.ravel(), yy.ravel()])

    # Put the result into a color plot
    Z  = Z.reshape(xx.shape)
    cm = plt.cm.Accent
    plt.contourf(xx, yy, Z, cmap=cm, alpha=0.5)
    plt.title("Adaline Decision Boundry")
    plt.xlabel("x")
    plt.ylabel("y")
    plt.show()
```

شکل 11- کد لازم برای چاپ خروجی مورد نظر سوال

```python
X, Y = prepare_data(c1_x, c1_y, c2_x, c2_y)

adaline = Adaline(in_features=2, lr=5e-4)
adaline.train(X, Y, max_epochs=100)
adaline.plot_loss()

plt.show()
adaline.plot_scatter(X, Y)
```

```
100% 100/100 [00:00<00:00, 283.84it/s]
```





شکل 12- نمودارهای پراکندگی و **Loss** روی داده‌های

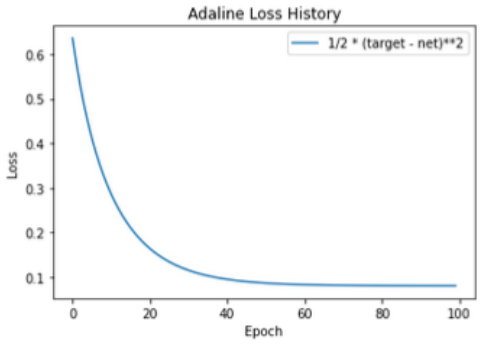
میبینیم که داده‌ها به خوبی جدا شده‌اند. از این اتفاق درمیابیم که اگر داده‌ها خطی جدایی پذیر باشند (با اینکه برخلاف شبکه پرسپترون در این روش اثبات نمیشود که حتما خط جداکننده یافت شود) حرکت

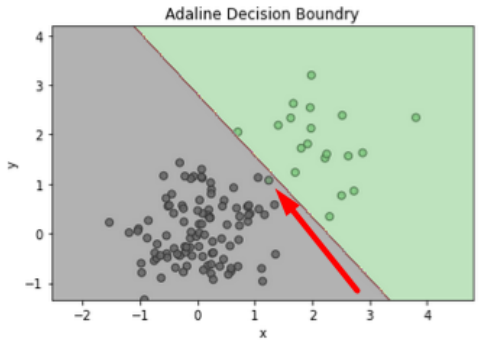
کردن وزن‌ها در خلاف جهت گرادیان $(t - net)$ میتوان یادگیری موفقی داشت و Loss را کاهش داد. پس روش یادگیری Delta روش خوبی برای مسائل ساده است. ثابت میشود قاعده Delta برای مسائلی که تعداد اعضای هر گروه **برابر** باشد و پراکندگی **متقارن** باشد جواب را پیدا میکند.

ج)

```
np.random.seed(0)

c1_x = np.random.normal(loc=0 , scale=0.6, size=100)
c1_y = np.random.normal(loc=0 , scale=0.6, size=100)
c2_x = np.random.normal(loc=2 , scale=0.8, size=20 )
c2_y = np.random.normal(loc=2 , scale=0.8, size=20 )

X2, Y2 = prepare_data(c1_x, c1_y, c2_x, c2_y)

adaline = Adaline(in_features=2, lr=5e-4)
adaline.train(X2, Y2, max_epochs=100)
adaline.plot_loss()

plt.show()
adaline.plot_scatter(X2, Y2)
```

```
100%|██████████| 100/100 [00:00<00:00, 279.07it/s]
```





شکل 13- نمودار **Loss** و پراکندگی داده‌ها برای مدل **AdaLine** در داده‌های نابرابر

میبینیم زمانی که داده‌ها نابرابر باشد و پراکندگی گروه‌ها با هم متفاوت باشد یافتن خط جداساز ممکن است رخ ندهد (حتی درصورت وجود). این یعنی مدل AdaLine فقط در شرایط خاصی درست عمل میکند زیرا که به دنبال کمینه کردن به صورت least mean square (LMS) عبارت $error = (t - h)$ نیست و عبارت $(t - net)$ را کمینه میکند. ولی چون که اولین نورنی بود که برای اموزش از گرادیان استفاده کرد و کارایی آن را در فرایند آموزش نشان داد، بیان آن یک هدف آموزشی عالی میباشد.

علاوه بر پراکندگی، نقطه شروع و شیوه‌ی initialization بسیار مهم است در یافتن جواب.

برای یافتن جواب در روش‌های گرادیانی یافتن مقدار خوب learning rate یک مسئله حیاتی میباشد و شرط یافتن جواب است. در پرسپترون خطی این مسئله را نداشتیم.

استفاده از تابع فعال سازی tanh حل کننده موضوع $(t - net)$ است و چون مشتق پذیر است در همه نقاط میتوان از $(t - h)$ استفاده کرد.

MadaLine .۲-۲

الف)

MR1 روش اصلی آموزش MadaLine است. در این روش فقط لایه مخفی آموزش میبیند و وزن‌های لایه خروجی ثابت هستند. در روش MR2 وزن‌های لایه خروجی هم قابل تغییر اند. ما MR1 را توضیح میدهیم:

1. وزن‌ها را initialize میکنیم و learning rate را انتخاب میکنم.
2. تا زمانی که شرط توقف ارضا نشده ادامه بده:
   - برای هر s, t قدم‌های زیر انجام شود
   - X=s
   - محاسبه net برای همه AdaLine ها
   - محاسبه out برای هر AdaLine
   - از out های به دست امده out کل حساب شود
   - Error حساب شود و وزن‌ها به روز شوند.
     - اگر t=1 بود انگاه Z_j هایی که خروجی net انها به 0 نزدیک بوده را به روز کن.
     - اگر y=-1 بود آنگاه Z_k هایی که net آنها net مثبت دارند.
   - شرط توقف را بررسی کن.

در MadaLine خروجی AdaLine ها OR و یا AND میشود. در MR1 عمل OR اتفاق افتاده است.

ب)

ابتدا کلاس MadaLine را تعریف میکنیم:

```python
class Madaline:
    def __init__(self, in_features=2, hidden_units=3, lr=0.1, stop_tolerance=1e-10):
        """
        Initialize weights:
        Weights v i and v2 and the bias b3 are set as described;
        small random values are usually used for ADALINE weights.
        """
        self.weights        = np.random.random([in_features, hidden_units])
        self.output_weights = np.array([1/hidden_units] * hidden_units)
        self.bias           = np.random.random(hidden_units)
        self.output_bias    = (hidden_units - 1)/hidden_units
        self.lr             = lr
        self.stop_tolerance = stop_tolerance
        self.loss_history   = []

        self.hidden_units   = hidden_units
        self.in_features    = in_features

    def activation_function(self, g):
        return ((g >= 0) - 0.5) * 2

    def forward(self, x):
        """
        Compute net input to each hidden ADALINE unit
        Determine output of each hidden ADALINE unit
        Determine output of net
        """
        x    = x.reshape(1, len(x))
        z_in = self.bias + x @ self.weights
        z    = self.activation_function(z_in)
        y_in = self.output_bias + z @ self.output_weights
        y    = self.activation_function(y_in)[0]

        return y, z_in, y_in

    def predict(self, X):
        Y = []
        for x in X:
            y, z_in, y_in = self.forward(x)
            Y.append(y)
        return np.array(Y)

    def eval(self, X, Y):
        preds = []
        for x, t in zip(X, Y):
            y, z_in, y_in = self.forward(x)
            preds.append(y)
        from sklearn.metrics import classification_report
        print(classification_report(Y, preds))
```

```python
    def step(self, y, z_in, t, x):
        z_in = z_in[0]  # flatten
        if t != y:  # If t = y, no weight updates are performed
            # print(z_in, t, y)
            if t == 1:  # If t = 1, then update weights on Zj, the unit whose net input is closest to 0
                close_zero_idx = np.argmin(abs(z_in))
                self.weights[:, close_zero_idx] += self.lr * (1 - z_in[close_zero_idx]) * x
                self.bias[close_zero_idx] += self.lr * (1 - z_in[close_zero_idx])
            elif t == -1:  # If t = −1, then update weights on all units Zk that have positive net input
                for h in range(self.hidden_units):
                    if z_in[h] > 0:
                        self.weights[:, h] += self.lr * (-1 - z_in[h]) * x
                        self.bias[h] += self.lr * (-1 - z_in[h])
            return True
        return False

    def train(self, X, Y, max_epochs=80):
        """
        If the largest weight change that occurred in Step 2 is
        smaller than a specified tolerance, then stop; otherwise continue.
        """
        self.loss_history = []
        X, Y = shuffle(X, Y, random_state=0)
        for epoch in tqdm(range(max_epochs)):
            targets, nets, changes = [], [], []
            for x, t in zip(X, Y):
                y, z_in, y_in = self.forward(x)
                changes.append(self.step(y, z_in, t, x))
                targets.append(t)
                nets.append(y_in)
            self.loss_history.append(self.calc_loss(targets, nets))
            if not any(changes):
                print("Early Stopping")
                break

    def calc_loss(self, targets, nets):
        return np.mean(1/2 * (np.array(targets) - np.array(nets))**2)
```

شکل 14- پیاده سازی **MadaLine**

سپس کدی برای کشیدن نمودارها مینویسیم:

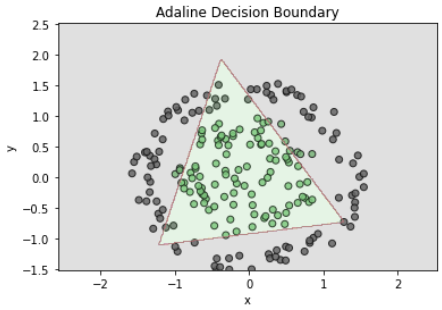
```python
    def plot_loss(self):
        plt.plot(self.loss_history, label="1/2 * (target - net)**2")
        plt.legend()
        plt.title("Adaline Loss History")
        plt.xlabel("Epoch")
        plt.ylabel("Loss")
        plt.show()

    def plot_scatter(self, X, Y):
        plt.scatter(
            X[:, 0],
            X[:, 1],
            c = Y,
            cmap       = 'Accent',
            edgecolors = "k",
            alpha      = 0.85,
        )
        h = 0.01  # step size in the mesh
        xx, yy = np.meshgrid(np.arange(min(X[:, 0]) - 2, max(X[:, 0]), h) + 1,
                             np.arange(min(X[:, 1]) - 1, max(X[:, 1]), h) + 1)
        # Plot the decision boundary
        Z = self.predict(np.c_[xx.ravel(), yy.ravel()])
        # Put the result into a color plot
        Z  = Z.reshape(xx.shape)
        cm = plt.cm.Accent
        plt.contourf(xx, yy, Z, cmap=cm, alpha=0.2)
        plt.title("Adaline Decision Boundary")
        plt.xlabel("x")
        plt.ylabel("y")
        plt.show()
```

شکل 15- کد لازم برای چاپ خروجی مورد نظر سوال

حال برای حالت ۳ و ۴ و ۸ نورون MadaLine را اجرا میکنیم:

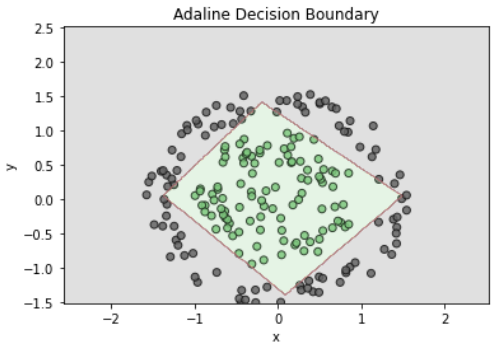
```python
for n in [3, 4, 8]:
    print(f"### Hidden Units: {n} ###")
    madaline = Madaline(in_features=2, hidden_units=n, lr=0.2)
    madaline.train(X=np.array(df[[0, 1]]), Y=np.array(df[2]), max_epochs=100)
    madaline.eval (X=np.array(df[[0, 1]]), Y=np.array(df[2]))
    madaline.plot_scatter(X=np.array(df[[0, 1]]), Y=np.array(df[2]))
```

```
### Hidden Units: 3 ###
100% 100/100 [00:00<00:00, 188.65it/s]
              precision    recall  f1-score   support

        -1.0       0.92      0.91      0.91       100
         1.0       0.91      0.92      0.92       100

    accuracy                           0.92       200
   macro avg       0.92      0.92      0.91       200
weighted avg       0.92      0.92      0.91       200
```



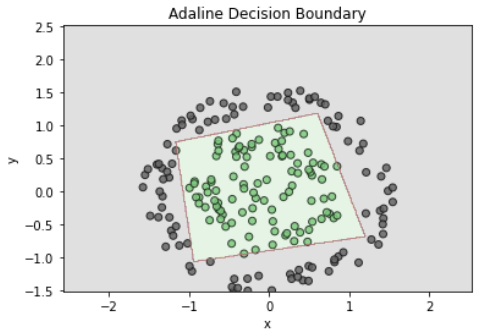
شکل 16- **MadaLine** برای ۳ نورون

```
[ ] ### Hidden Units: 4 ###
30% 30/100 [00:00<00:00, 143.55it/s]
Early Stopping
              precision    recall  f1-score   support

        -1.0       1.00      1.00      1.00       100
         1.0       1.00      1.00      1.00       100

    accuracy                           1.00       200
   macro avg       1.00      1.00      1.00       200
weighted avg       1.00      1.00      1.00       200
```



شکل 17- **MadaLine** برای ۴ نورون

```
### Hidden Units: 8 ###
4% 4/100 [00:00<00:02, 46.80it/s]
Early Stopping
              precision    recall  f1-score   support

        -1.0       1.00      1.00      1.00       100
         1.0       1.00      1.00      1.00       100

    accuracy                           1.00       200
   macro avg       1.00      1.00      1.00       200
weighted avg       1.00      1.00      1.00       200
```



شکل 18- **MadaLine** برای ۸ نورون

ج)

با توجه به نتایج قسمت ب میبینیم که با افزایش تعداد نورون‌ها فضاهای پیچیده‌تری میتوان ساخت، همچنین سرعت یادگیری بالاتر میرود، همچنین دقت بالاتر میرود. این بدین معناست که پیچیده کردن شبکه بهایی است که باید برای دقت بهتر و فیت شدن به فضاهای پیچیده‌تر پرداخت کنیم. در مورد سرعت یادگیری چون مسئله پیش رو ساده است پیچیده کردن شبکه سرعت یادگیری را بالا میبرد. ولی در مسائل

ذاتا پیچیده افزایش تعداد نورون‌ها باعث بیش برازش و حتی کند شدن آموزش میشود. با ۳ نورون هیچگاه دقت به ۱۰۰٪ نمیرسد درصورتی که ۱۰۰ ایپاک فرصت داشته. ۴ نورون در ۳۰ ایپاک توانسته به دقت مطلوب برسد. در حالت ۸ نورون در ۴ ایپاک به دقت مطلوب رسیده ایم یعنی بهبود تقریبا ۸ برابری در صورتی که تعداد نورون‌ها فقط ۲ برابر شده است.

پاسخ ۳ – Restricted Boltzmann Machine

کدهای مربوط به این سوال و سوال 4 در این آدرس زیر موجود است:

(A

ابتدا دیتاست های مربوطه را می خوانیم و از هر کدام 5 خانه ی اول و 5 خانه ی آخر را به کمک دستور head و tail مشاهده می کنیم. همچنین اگر تنها دیتافریم را در یک سلول فراخوانی کنیم به طور خودکار 5 خانه ی اول و 5 خانه ی آخر نمایش داده می شود.

```
movies
```

	movieId	title	genres
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
1	2	Jumanji (1995)	Adventure|Children|Fantasy
2	3	Grumpier Old Men (1995)	Comedy|Romance
3	4	Waiting to Exhale (1995)	Comedy|Drama|Romance
4	5	Father of the Bride Part II (1995)	Comedy
...	...	...	...
9737	193581	Black Butler: Book of the Atlantic (2017)	Action|Animation|Comedy|Fantasy
9738	193583	No Game No Life: Zero (2017)	Animation|Comedy|Fantasy
9739	193585	Flint (2017)	Drama
9740	193587	Bungo Stray Dogs: Dead Apple (2018)	Action|Animation
9741	193609	Andrew Dice Clay: Dice Rules (1991)	Comedy

9742 rows × 3 columns

شکل 19- دیتاست **movies**

دیتاست شامل نام فیلم ها و ژانر هر کدام از آن ها است و هر فیلم یک Id مخصوص به خود را دارد که به کمک آن قابل شناسایی است.

ratings

	userId	movieId	rating	timestamp
0	1	1	4.0	964982703
1	1	3	4.0	964981247
2	1	6	4.0	964982224
3	1	47	5.0	964983815
4	1	50	5.0	964982931
...	...	...	...	...
100831	610	166534	4.0	1493848402
100832	610	168248	5.0	1493850091
100833	610	168250	5.0	1494273047
100834	610	168252	5.0	1493846352
100835	610	170875	3.0	1493846415

100836 rows × 4 columns

شکل 20- دیتاست **ratings**

در این دیتاست نیز امتیازی که هر کاربر به هر فیلم مشاهده می شود. هر کاربر نیز یک Id مخصوص به خود را دارد. امتیازات نیز از 0 تا 5 هستند که 5 بالاترین امتیاز و بیشترین رضایت از آن فیلم است.

```
print('Movies: ', movies.shape)
print('Ratings: ', ratings.shape)
```

```
Movies:  (9742, 3)
Ratings:  (100836, 4)
```

شکل 21- ابعاد دیتاست های **movies** و **ratings**

دیتاست movies و ratings به ترتیب شامل 9742 و 100836 ردیف هستند و هر کدام از آن ها 3 و4 ستون نیز دارند. با استفاده از دستور زیر ستونی به نام List Index در دیتاست movies ایجاد می کنیم.

```
movies['List Index'] = movies.index
```

مقدار این ستون مقادیر index دیتاست movies است. از آن جایی که Id فیلم ها در دیتاست شامل تمام Id ها نیست و ترتیب ندارد نمی توانیم از movieId به عنوان ایندکس استفاده کنیم چرا که می بینیم

دیتاست شامل 9742 فیلم است اما ما فیلمی با Id 193581 نیز داریم. بنابراین از ستون List Index به عنوان ایندکس برای فیلم ها استفاده می کنیم.

(B

به کمک دستور merge بر روی movieId دو دیتاست را با هم ادغام می کنیم. عملیات ادغام به صورت inner اتفاق می افتد که در آن از اشتراک movieId در دو دیتاست استفاده می شود و ترتیب مطابق دیتاست سمت چپ که در اینجا movies است حفظ می شود. نتیجه در شکل زیر آمده است.

df

	movieId	title	genres	List Index	userId	rating	timestamp
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	0	1	4.0	964982703
1	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	0	5	4.0	847434962
2	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	0	7	4.5	1106635946
3	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	0	15	2.5	1510577970
4	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	0	17	4.5	1305696483
...	...	...	...	...	...	...	...
100831	193581	Black Butler: Book of the Atlantic (2017)	Action|Animation|Comedy|Fantasy	9737	184	4.0	1537109082
100832	193583	No Game No Life: Zero (2017)	Animation|Comedy|Fantasy	9738	184	3.5	1537109545
100833	193585	Flint (2017)	Drama	9739	184	3.5	1537109805
100834	193587	Bungo Stray Dogs: Dead Apple (2018)	Action|Animation	9740	184	3.5	1537110021
100835	193609	Andrew Dice Clay: Dice Rules (1991)	Comedy	9741	331	4.0	1537157606

100836 rows × 7 columns

شکل 22- نتیجه ی ادغام دیتاست **movies** و **ratings**

(C

ستون های title، genres و timestamp اضافی هستند و آن ها را از دیتاست حذف می کنیم. این ستون های اطلاعات اضافه ای در مورد مسئله به ما نمی دهند که بتوانیم برای پیش بینی از آن استفاده کنیم. هر فیلم با یک Id مختص به خود شناسایی می شود در نتیجه عنوان و ژانر آن به ما اطلاعات اضافه تری نمی دهد. ممکن است در مسائل دیگر و مدل های دیگر بتوان از ژانر فیلم های مشاهده شده توسط کاربر استفاده کرد اما ما در اینجا تنها پیش بینی را بر اساس امتیازهای کاربران انجام می دهیم. همچنین timestamps نیز در این مسئله کاربردی ندارد چرا که زمانی که کاربر به فیلم امتیاز داده است برای ما اهمیتی ندارد و خود امتیاز مهم است.

(D

کاربران را بر اساس userId گروه بندی می کنیم.

```
[105] users = df.groupby('userId')

[106] users.ngroups

      610
```

شکل 23- دستور **groupby**

می بینیم که در مجموع 610 گروه داریم به این معنی که 610 کاربر مختلف داریم.

(E

با توجه به مقاله ی ضمیمه شده، می دانیم که ورودی ماشین RBM باید برای هر کاربر شامل یک لیست باشد که در هر ایندکس از آن، امتیاز کاربر به آن فیلم وجود داشته باشد و در دیگر خانه ها نیز مقدار آن 0 باشد. برای مثال اگر کاربر 1، به فیلمی که در ایندکس 0 در دیتاست movies قرار دارد امتیاز 4 داده باشد، اولین خانه ی لیست مربوطه، یعنی ایندکس 0 نیز مقدار 4 دارد. البته امتیازات نرمال شده اند. در نهایت نیز به ازای هر کاربر یک لیست داریم و خروجی نهایی لیستی از لیست ها است. این عملیات در قطعه کد زیر برای هر کاربر انجام شده است.

```
train_X = []
for userId, userMovies in users:
  userMovies['rating'] = userMovies['rating']/5
  userMoviesList = [0]*movies.shape[0]
  for i,movie in userMovies.iterrows():
    userMoviesList[int(movie['List Index'])] = movie['rating']
  train_X.append(userMoviesList)
```

شکل 24- آماده سازی ورودی برای **RBM** و نرمال سازی امتیازات

train_X[0] نشان دهنده ی لیست امتیازات مربوط به کاربر اول است.

```
print(train_X[0])

[0.8, 0, 0.8, 0, 0, 0.8, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
```

شکل 25- لیست امتیازات کاربر اول

می بینیم که در این لیست کاربر اول به فیلم موجود در ایندکس 0 در دیتاست movies امتیاز 0.8 (4) داده است و فیلم مربوط به ایندکس 1 را نیز مشاهده نکرده است و امتیاز آن 0 است. طول این لیست به اندازه ی تعداد فیلم های موجود یعنی 9742 است. از آن جایی که 610 کاربر نیز داریم بنابراین train_X شکلی به صورت (610,9742) دارد.

(F

در این قسمت به سراغ پیاده سازی RBM می رویم. مدل خواسته شده را با 20 لایه ی پنهان و 9742 لایه ی آشکار می سازیم. 9742 تعداد فیلم های موجود در دیتاست است. از تابع sigmoid برای فعال سازی در لایه ی آشکار و پنهان؛ و از MAE به عنوان cost function استفاده کرده ایم. همچنین از objective function معرفی شده در مقاله که Contrastive Divergence نام دارد استفاده کرده ایم. جزئیات این تابع در مقاله توضیح داده شده است و از تکرار آن صرف نظر می کنیم.

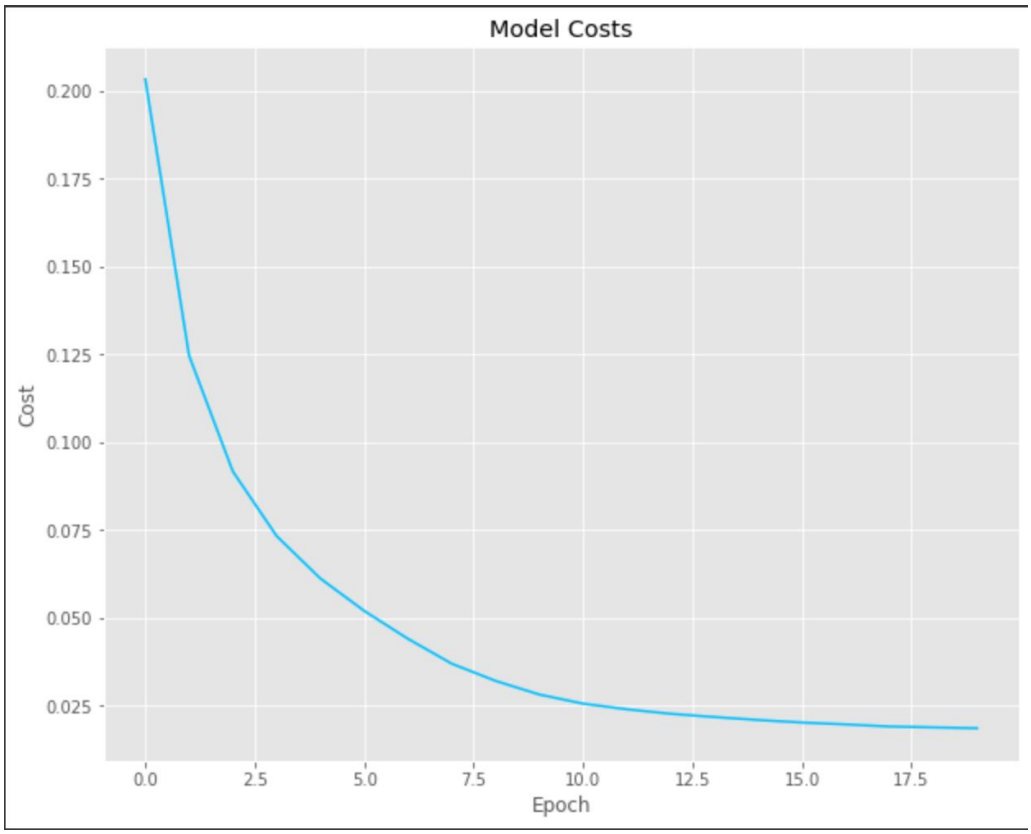
پس از آن که ساخت graph به پایان رسید و شبکه ی مورد نظر تکمیل شد، یک session ایجاد می کنیم و به کمک آن operationهای تعریف شده را اجرا می کنیم.

```
visible_units = len(movies)
hidden_units = 20
W = tf.placeholder(tf.float32, [visible_units, hidden_units])
v_bias = tf.placeholder(tf.float32, [visible_units])
h_bias = tf.placeholder(tf.float32, [hidden_units])
x = tf.placeholder("float", [None, visible_units])
#forward pass
_h = tf.nn.sigmoid(tf.matmul(x, W) + h_bias)
h = tf.nn.relu(tf.sign(_h - tf.random_uniform(tf.shape(_h))))
#backward pass
_v = tf.nn.sigmoid(tf.matmul(h, tf.transpose(W)) + v_bias)
v = tf.nn.relu(tf.sign(_v - tf.random_uniform(tf.shape(_v))))
h1 = tf.nn.sigmoid(tf.matmul(v, W) + h_bias)

#cd
positive_gradient = tf.matmul(tf.transpose(x), h)
negative_gradient = tf.matmul(tf.transpose(v), h1)
CD = (positive_gradient - negative_gradient) / tf.to_float(tf.shape(x)[0])

#cost function
update_w = W + CD
update_vb = v_bias + tf.reduce_mean(x - v, 0)
update_hb = h_bias + tf.reduce_mean(h - h1, 0)
objective = tf.reduce_mean((x - v)*(x - v))
```

شکل 26- کد مربوط به پیاده سازی **RBM**

(G

حال مدل تعریف شده را به مدت 20 epoch آموزش می دهیم. مقدار batch_size را نیز 128 در نظر می گیریم.در نهایت مقدار هزینه را در هر دور محاسبه می کنیم و نمودار cost را رسم می کنیم. نتیجه در شکل زیر نمایش داده شده است.



شکل 27- نمودار **cost** بدست آمده از **RBM**

می بینیم که با افزایش تعداد epoch ها هزینه نیز کاهش یافته است.

(H

یوزر شماره 75 را انتخاب می کنیم. باید توجه کنیم که 75 ایندکس مربوط به کاربر در لیست train_X است و userId فرد متفاوت است. حال لیست مربوط به این کاربر را به مدل می دهیم و عملیات feed و reconstruct را انجام می دهیم و در نهایت یک لیست از امتیازاتی که برای فیلم ها به کاربر پیشنهاد شده است دریافت می کنیم. نتیجه به صورت زیر است.

```
rec_movies
```

```
array([[0.21169454, 0.09618857, 0.03903125, ..., 0.00387926, 0.00333613,
        0.00385023]], dtype=float32)
```

شکل 28- خروجی مدل برای امتیازات کاربر **75**

حال اگر بخواهیم نام فیلم ها را نیز بدست آوریم باید این لیست را با دیتاست movies انطباق دهیم. امتیازات پیشنهاد شده توسط مدل را به عنوان یک ستون با نام rec به دیتاست movies اضافه می کنیم و در نهایت دیتافریم جدید را بر اساس امتیاز پیشنهاد شده به صورت نزولی مرتب می کنیم و 15 سطر اول آن را مشاهده می کنیم. نتیجه در شکل زیر نمایش داده است.

```
all_movies_for_user.head(15)
```

	movieId	title	genres	List Index	rec
2226	2959	Fight Club (1999)	Action|Crime|Drama|Thriller	2226	0.521025
224	260	Star Wars: Episode IV - A New Hope (1977)	Action|Adventure|Sci-Fi	224	0.484522
1939	2571	Matrix, The (1999)	Action|Sci-Fi|Thriller	1939	0.479027
2078	2762	Sixth Sense, The (1999)	Drama|Horror|Mystery	2078	0.474231
257	296	Pulp Fiction (1994)	Comedy|Crime|Drama|Thriller	257	0.458953
2145	2858	American Beauty (1999)	Drama|Romance	2145	0.438831
3141	4226	Memento (2000)	Mystery|Thriller	3141	0.431368
277	318	Shawshank Redemption, The (1994)	Crime|Drama	277	0.424541
510	593	Silence of the Lambs, The (1991)	Crime|Horror|Thriller	510	0.422451
3638	4993	Lord of the Rings: The Fellowship of the Ring,...	Adventure|Fantasy	3638	0.418379
314	356	Forrest Gump (1994)	Comedy|Drama|Romance|War	314	0.404948
461	527	Schindler's List (1993)	Drama|War	461	0.401245
4909	7361	Eternal Sunshine of the Spotless Mind (2004)	Drama|Romance|Sci-Fi	4909	0.390051
907	1206	Clockwork Orange, A (1971)	Crime|Drama|Sci-Fi|Thriller	907	0.371169
4137	5952	Lord of the Rings: The Two Towers, The (2002)	Adventure|Fantasy	4137	0.356655

شکل 29- **15** فیلم که بیشترین امتیاز پیشنهاد شده را دارند (برای کاربر **75**)

می بینیم که بیشترین امتیاز پیشنهاد شده مربوط به Fight Club است که با توجه به اینکه از معروف ترین فیلم های دنیا است و کاربران زیادی امتیاز بالا به آن داده اند، در اینجا نیز امتیاز بالا برای پیشنهاد دریافت کرده است.

نکته ی قابل توجه در این لیست این است که ممکن است این لیست شامل فیلم هایی باشد که کاربر قبلا مشاهده کرده است بنابراین اگر بخواهیم تنها فیلم هایی که کاربر ندیده است را پیدا کنیم، ابتدا باید userId این کاربر را پیدا کرده و سپس فیلم هایی که به آن ها امتیاز داده است را پیدا کنیم و آن ها را از این لیست حذف کنیم.

پاسخ ۴ – MLP

(A

ابتدا فایل csv مربوطه را می خوانیم و به کمک دستور info اطلاعات مربوط به دیتاست را بررسی می کنیم. نتیجه در شکل زیر نمایش داده شده است.

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	...	grade	sqft_above	sqft_basement	yr_built	yr
0	7129300520	20141013T000000	221900.0	3	1.00	1180	5650	1.0	0	0	...	7	1180	0	1955	
1	6414100192	20141209T000000	538000.0	3	2.25	2570	7242	2.0	0	0	...	7	2170	400	1951	
2	5631500400	20150225T000000	180000.0	2	1.00	770	10000	1.0	0	0	...	6	770	0	1933	
3	2487200875	20141209T000000	604000.0	4	3.00	1960	5000	1.0	0	0	...	7	1050	910	1965	
4	1954400510	20150218T000000	510000.0	3	2.00	1680	8080	1.0	0	0	...	8	1680	0	1987	

5 rows × 21 columns

شکل 30- 5 ردیف ابتدایی از دیتاست **houses**

```
houses.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
```

شکل 31- خروجی تابع **info** بر روی دیتاست **houses**

می بینیم که دیتاست شامل اطلاعات خانه ها است و فیچرهایی از جمله bedrooms (تعداد اتاق خواب ها)، bathrooms (تعداد سرویس های بهداشتی)، price (قیمت خانه) و ... دارد. همچنین از 21613 ردیف موجود در دیتاست، تمام آن ها در ستون های مختلف، non-null هستند و نوع هر کدام از فیچرها نیز مشخص شده است.

(B

به کمک دستور isna و isnull تعداد داده هایی که nan و یا null هستند را بدست می آوریم. نتیجه در شکل های زیر نمایش داده شده است.

```
houses.isna().sum()
```

```
id               0
date             0
price            0
bedrooms         0
bathrooms        0
sqft_living      0
sqft_lot         0
floors           0
waterfront       0
view             0
condition        0
grade            0
sqft_above       0
sqft_basement    0
yr_built         0
yr_renovated     0
zipcode          0
lat              0
long             0
sqft_living15    0
sqft_lot15       0
dtype: int64
```

شکل 32- خروجی دستور **isna**

```
houses.isnull().sum()
```

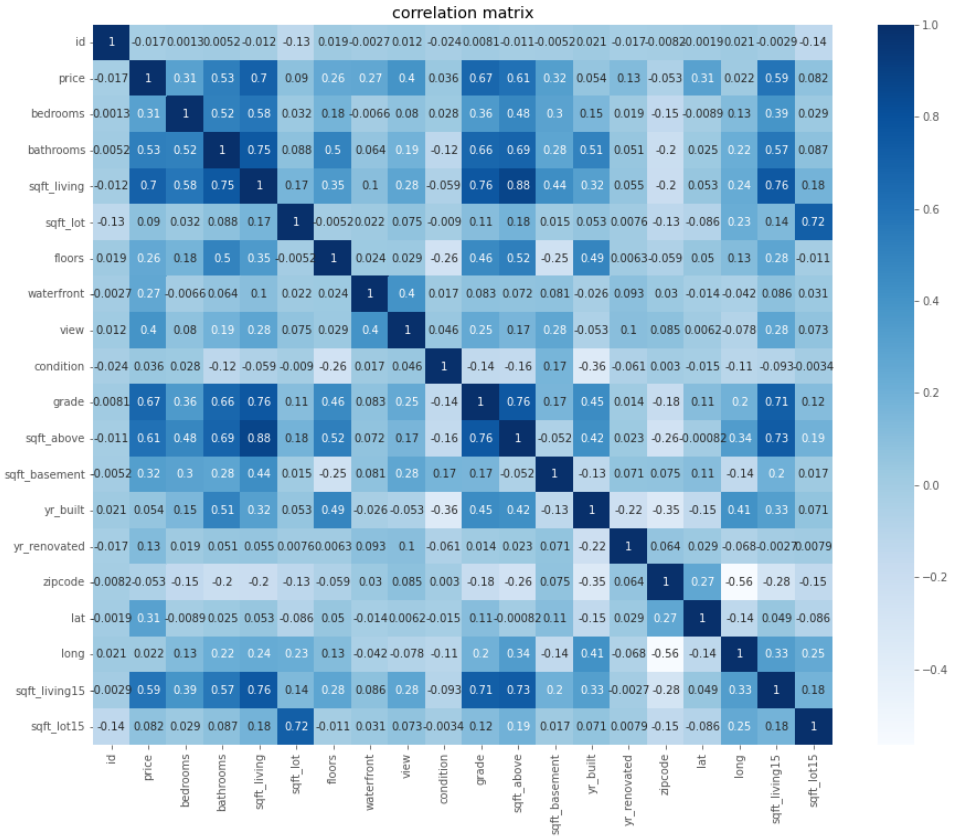
```
id               0
date             0
price            0
bedrooms         0
bathrooms        0
sqft_living      0
sqft_lot         0
floors           0
waterfront       0
view             0
condition        0
grade            0
sqft_above       0
sqft_basement    0
yr_built         0
yr_renovated     0
zipcode          0
lat              0
long             0
sqft_living15    0
sqft_lot15       0
dtype: int64
```

شکل 33- خروجی دستور **isnull**

.می بینیم که هیچ داده ی nan و یا null در دیتاست وجود ندارد

(C

به کمک دستور corr ماتریس correlation را محاسبه می کنیم و نتیجه را رسم می کنیم.



شکل 34- ماتریس **correlation** برای دیتاست **houses**

می دانیم که در ماتریس correlation، هر چقدر مقدار قدر مطلق یک خانه، به 1 نزدیک تر باشد به این معنی است که فیچرهای متناظر با آن خانه همبستگی بیشتری با یک دیگر دارند و اگر مقدار آن منفی باشد همبستگی معکوس است به این معنی که با افزایش یکی دیگری کاهش می یابد؛ و اگر مقدار آن مثبت باشد به معنای همبستگی مثبت است به این معنی که با افزایش یکی، دیگری نیز افزایش می یابد. می دانیم که correlation بالا لزوما به معنی رابطه ی علت و معلولی نیست و صرفا ارتباط میان متغیرها را نشان می دهد.

از آن جایی که متغیر هدف ما price است، این ردیف را از جدول استخراج می کنیم و به طور جداگانه بررسی می کنیم.

```
cor_target = corr_matrix["price"]
cor_target
```

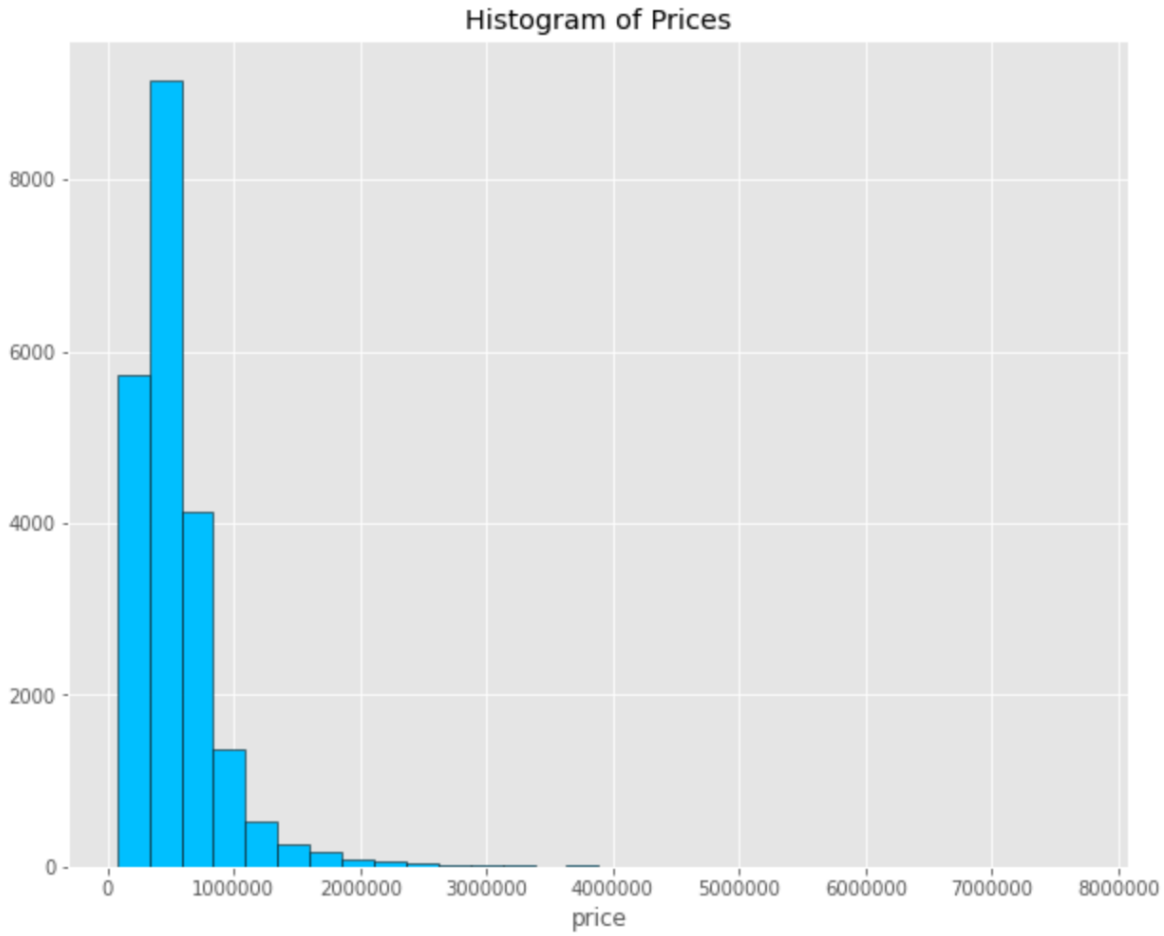
```
id              -0.016762
price            1.000000
bedrooms         0.308350
bathrooms        0.525138
sqft_living      0.702035
sqft_lot         0.089661
floors           0.256794
waterfront       0.266369
view             0.397293
condition        0.036362
grade            0.667434
sqft_above       0.605567
sqft_basement    0.323816
yr_built         0.054012
yr_renovated     0.126434
zipcode         -0.053203
lat              0.307003
long             0.021626
sqft_living15    0.585379
sqft_lot15       0.082447
Name: price, dtype: float64
```

شکل 35- مقادیر **correlation** برای متغیر **price**

می بینیم که همبستگی price با خودش 1 است که این امر بدیهی است. پس از آن بیشترین همبستگی با متغیر sqft_living مشاهده می شود که مقدار correlation میان آن ها 0.7 است و با یکدیگر ارتباط مثبت دارند.

(D

در این قسمت نمودار توزیع قیمت را به کمک هیستوگرام رسم می کنیم. نتیجه در شکل زیر نمایش داده شده است. در رسم نمودار تعداد bins ها 30 در نظر گرفته شده است.



شکل 36- نمودار توزیع قیمت خانه ها

می بینیم که نمودار توزیع قیمت، نموداری right skewed است به این معنی که میانگین قیمت ها، مقدار بیشتری از میانه و مد آن ها دارد. قیمت بیشتر خانه ها کمتر از 1 میلیون واحد است و تعداد خانه هایی که قیمتی بیشتر از 2 میلیون و 500 هزار واحد دارند ناچیز است.

حال نمودار ارتباط میان price و sqft_living را رسم می کنیم. sqft_living مساحت خانه را بر اساس معیار square feet نمایش می دهد.

شکل 37- نمودار ارتباط میان **price** و **sqft_living**

می بینیم که با افزایش sqft_living، به طور کلی قیمت نیز افزایش یافته است که منطقی است. در واقعیت نیز معمولا انتظار داریم وقتی مساحت یا زیربنای یک خانه افزایش پیدا کند، قیمت آن نیز افزایش پیدا کند هر چند که این عامل به تنهایی در قیمت تاثیرگذار نیست.

(E

داده ی date موجود در دیتاست به صورت string ذخیره شده است که به کمک دستور to_datetime ابتدا آن را به نوع datetime تبدیل می کنیم سپس به کمک دستورات زیر ماه و سال را استخراج کرده، به دیتاست اضافه می کنیم و ستون date را حذف می کنیم.

```
datetime_df = pd.to_datetime(houses['date'])
houses['year'] = datetime_df.dt.year
houses['month'] = datetime_df.dt.month
houses = houses.drop(columns=['date'])
```

شکل 38- استخراج ماه و سال از **date**

(F

پیش از آن که داده ها را به train و test تقسیم کنیم، ستون های zipcode و id را نیز از دیتافریم حذف می کنیم چرا که این دو ستون اطلاعاتی را در مورد قیمت خانه به ما نمی دهند و صرفا مقادیر عددی هستند که برای هر خانه یونیک می باشد. سپس داده ها را به کمک تابع train_test_split در sklearn به دو قسمت train و test تقسیم می کنیم.

(G

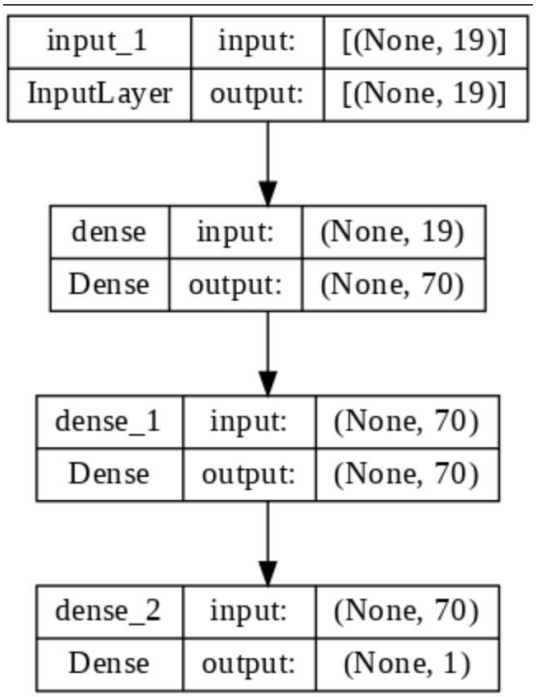
برای scale کردن از MinMaxScaler استفاده می کنیم. ابتدا scaler را بر روی داده های train فیت می کنیم و سپس برای جلوگیری از data leakage، داده های test را به کمک این scaler، transform می کنیم. به این ترتیب داده های test در فرایند فیت شدن scaler شرکت نمی کنند. لازم به ذکر است که برای target variable که در اینجا price می باشد نیز همین کار را تکرار می کنیم. scale کردن خروجی الزامی نیست اما با توجه به اینکه اعداد خروجی بزرگ هستند، بهتر است که scale را برای خروجی نیز انجام دهیم.

```
[78] scaler = MinMaxScaler()
     x_train = scaler.fit_transform(x_train)
     x_test = scaler.transform(x_test)

[79] target_scaler = MinMaxScaler()
     y_train = target_scaler.fit_transform(y_train.reshape(-1,1))
     y_test = target_scaler.transform(y_test.reshape(-1,1))
```

شکل 39- **scale** داده های ورودی و خروجی

(H

یک مدل MLP با دو لایه پنهان ایجاد می کنیم. از آن جایی که در ورودی پس از حذف ستون های ذکر شده در قسمت F، 19 ستون(فیچر) داریم، در لایه ی اول سایز ورودی را 19 در نظر می گیریم. در 2 لایه ی میانی نیز در هر لایه 70 نورون و در نهایت یک نورون در لایه ی خروجی در نظر می گیریم. تابع فعال ساز تمام لایه ها نیز relu در نظر گرفته شده است. ساختار شبکه ی ساخته شده را به کمک دستور plot_model مشاهده می کنیم.



شکل 40- شبکه ی **MLP** ساخته شده

پس از ساخت مدل آن را کامپایل و train می کنیم.

I و H)

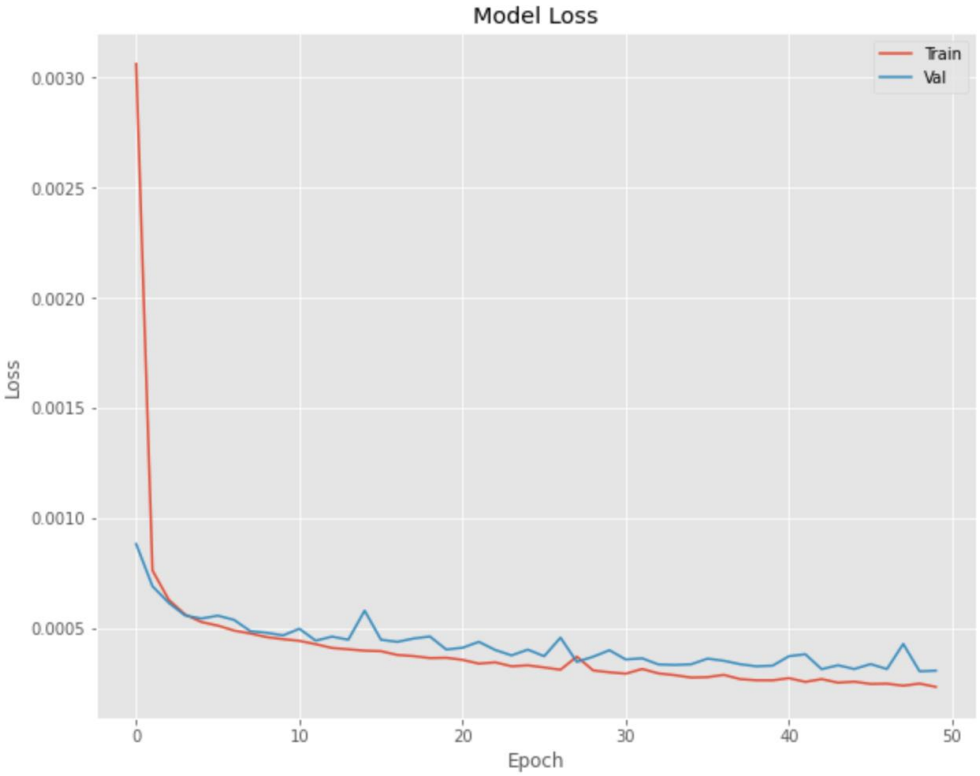
در این قسمت دو Optimizer و Loss Function مختلف را بررسی می کنیم و برای هر کدام نمودار Loss را رسم می کنیم. در هر کدام از حالت ها مدل را برای 50 epoch آموزش داده ایم و همچنین batch_size را نیز 128 در نظر گرفته ایم. همچنین از داده های test به عنوان validation در فرایند آموزش استفاده می کنیم.

Adam و MSE:

در حالت اول از Adam به عنوان Optimizer و از mse به عنوان Loss Function استفاده می کنیم. می دانیم که mse رایج ترین loss function مورد استفاده است و از رابطه ی زیر محاسبه می شود که در آن y_i مقدار اصلی متغیر هدف و $\hat{y}_i$ پیش بینی مدل برای آن خروجی است.

$$\mathrm{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

حال نمودار loss را رسم می کنیم.



شکل 41- نمودار **loss** بدست آمده با استفاده از **Adam** و **MSE**

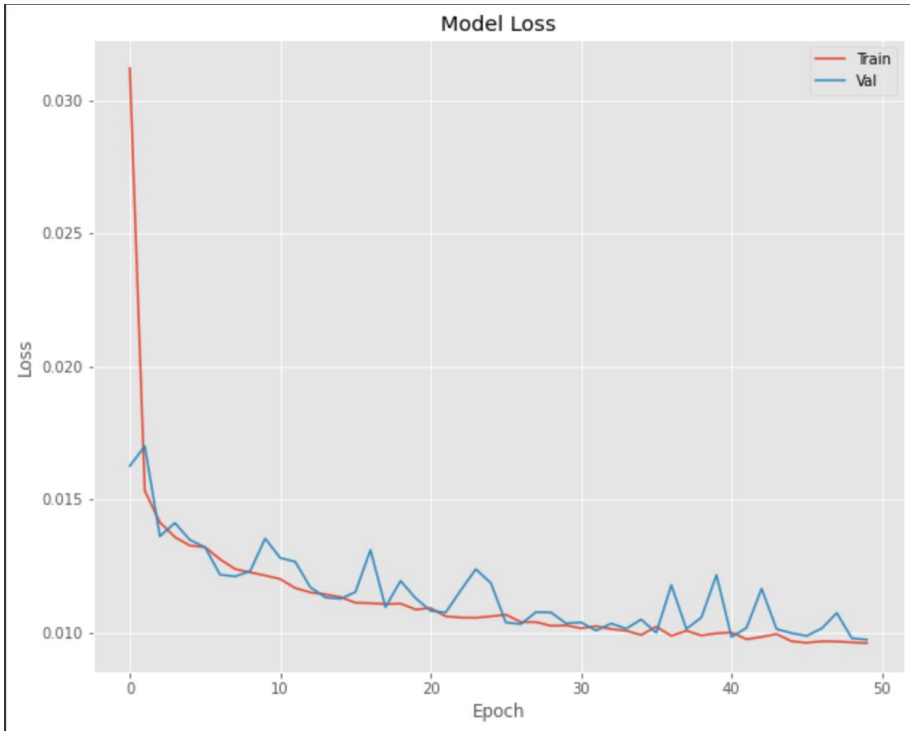
با توجه به نمودار می بینیم که مدل نه underfit شده و نه overfit و loss بدست آمده از دیتای train و test در نقاط انتهایی نمودار به یکدیگر نزدیک شده است به این معنی که مدل generalization مناسبی دارد.

Adam و MAE

در این قسمت از MAE به عنوان loss function استفاده می کنیم.

$$\mathrm{MAE} = \frac{1}{n}\sum_{j=1}^{n}|y_j - \hat{y}_j|$$

اگر دیتای ما توزیعی شبیه به گوسی داشته باشد اما تعدادی outlier نیز داشته باشیم، در این صورت استفاده از mae مناسب است چرا که نسبت به outlier ها robust تر است. مدل را آموزش می دهیم و نتیجه را در شکل زیر مشاهده می کنیم.

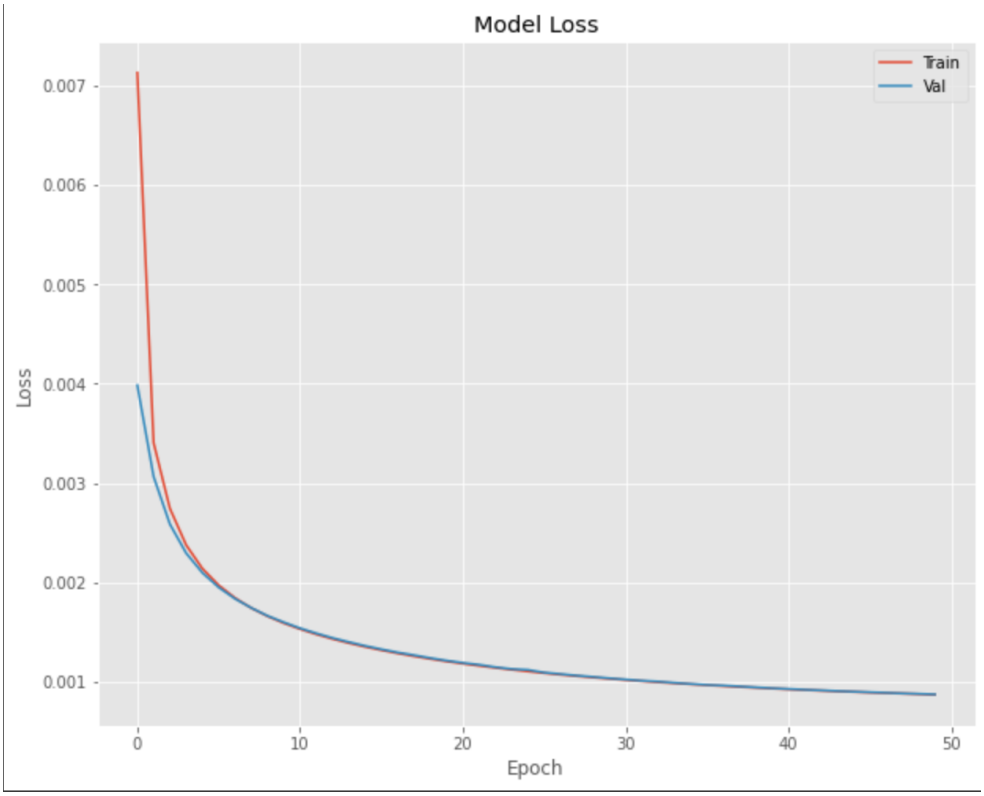


شکل 42- نمودار **loss** بدست آمده با استفاده از **Adam** و **MAE**

می بینیم که در این حالت مقداری حرکات نویزی در loss مربوط به validation مشاهده می شود که ممکن است ناشی از کاهش generalization مدل بر روی دیتای test باشد اگر چه مدل بر روی دیتای train به خوبی fit شده است. همچنین از آنجایی که نقطه ی شروع اولیه ی مدل به صورت رندوم است، ممکن است با اجراهای متفاوت نتایج متفاوتی نیز حاصل شود.

SGD و MSE

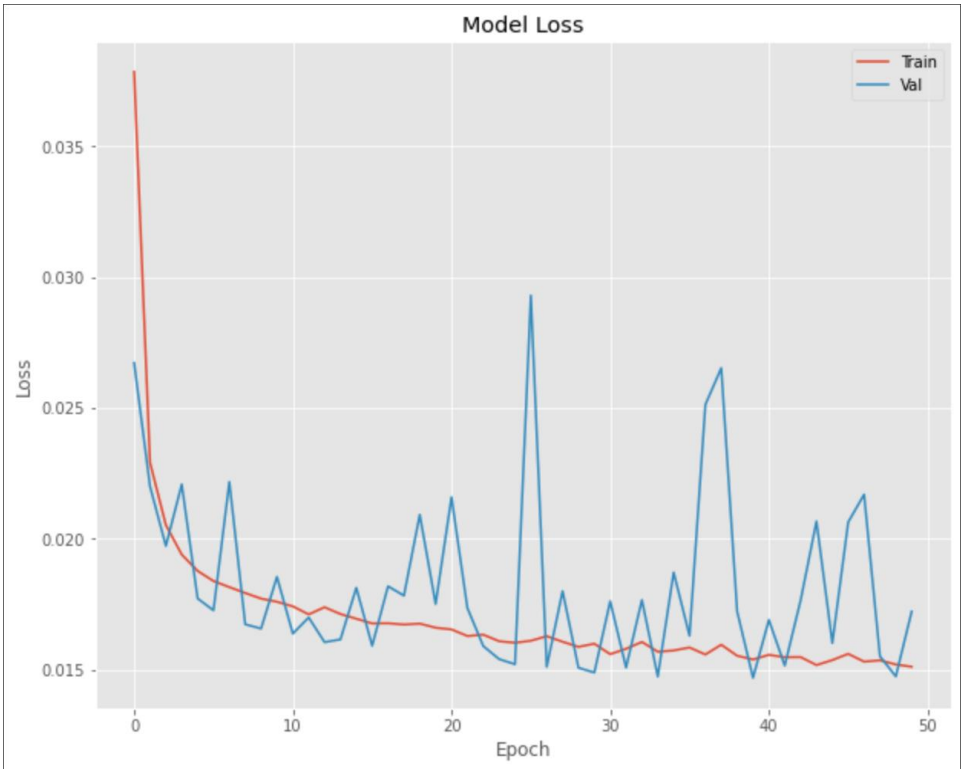
در این حالت از SGD به عنوان optimizer استفاده می کنیم. SGD برای دیتاست های بزرگ بهتر است و generalization بهتری ارائه می دهد اگر چه Adam سریعتر عمل می کند.



شکل 43- نمودار **loss** بدست آمده با استفاده از **SGD** و **MSE**

می بینیم که در این حالت دو نمودار از یک نقطه به بعد عملا منطبق هستند و مدل به خوبی fit شده است و همانطور که انتظار داشتیم generalization مناسبی نیز در مدل داریم.

SGD و MAE



شکل 44- نمودار **loss** بدست آمده با استفاده از **SGD** و **MAE**

در این حالت می بینیم که نمودار loss برای دیتای test بسیار نویزی است و مدل generalization ضعیفی دارد. ممکن است بتوانیم با تغییر دیگر پارامترها مانند نرخ یادگیری، مدل را بهتر کنیم اما همچنان این مشکل وجود خواهد داشت.

با مقایسه ی این نمودارها می توانیم نتیجه بگیریم که با توجه به دیتای ما، MAE به خوبی MSE عمل نمی کند و همچنین استفاده از SGD نتیجه ی بهتری نسبت به Adam ارائه می کند. بنابراین برای قسمت بعدی از مدلی که با استفاده از SGD و MSE آموزش دیده شده است استفاده می کنیم.

(K

در این قسمت ابتدا 5 نمونه از دیتای test را به صورت تصادفی انتخاب می کنیم سپس با استفاده از مدل قیمت را برای آن ها پیش بینی می کنیم.

```
[83] samples = random.sample(list(enumerate(x_test)),5)

[84] samples_index = list(map(lambda x: x[0], samples))
     samples_x = list(map(lambda x: x[1], samples))
     samples_y = [y_test[i] for i in samples_index]

[85] y_pred = model.predict(np.array(samples_x))
     y_pred
```

شکل 45- انتخاب **5** نمونه تصادفی از دیتای **test**

پس از آن که نمونه های تصادفی را از x_test انتخاب کردیم، باید y متناظر با آن ها را نیز انتخاب کنیم تا بتوانیم نتایج را مقایسه کنیم. برای این کار index مربوط به هر x را نیز بدست آورده ایم و از آن برای انتخاب y متناظر کمک گرفته ایم. در نهایت نیز نمونه ها را به مدل داده و نتیجه را پیش بینی کرده ایم.

	y_pred	y_samples
0	0.050412	[0.044214117029651015]
1	0.071973	[0.04408291786932564]
2	0.011405	[0.02702702702702703]
3	0.012707	[0.027814221988979274]
4	0.058975	[0.0402781422198898]

شکل 46- نتیجه ی مقایسه ی دیتای نمونه و پیش بینی مدل

می بینیم که نتیجه در بعضی نقاط نزدیک به مقدار واقعی و در بعضی نقاط با آن فاصله دارد. برای آن که دید بهتری داشته باشیم، MSE و RMSE را برای این نقاط محاسبه می کنیم.

```
res_mse = mean_squared_error(samples_y,y_pred)
print('MSE: ',res_mse)
print('RMSE: ',np.sqrt(res_mse))
```

```
MSE:  0.0003276199524701737
RMSE:  0.0181002749280273
```

شکل 47- نتیجه ی **MSE** و **RMSE**

می بینیم که مقدار RMSE تقریبا 0.18 است به این معنی که مقادیر پیش بینی شده به طور میانگین 0.18 واحد از مقادیر اصلی فاصله دارند که این مقدار، مقدار مناسبی است و مدل به خوبی به دیتای ما fit شده است. می دانیم که اگر RMSE 0 باشد بهترین نتیجه حاصل می شود. ممکن است بتوانیم با استفاده از تغییراتی در مدل این مقدار را از 0.18 نیز کمتر کنیم.