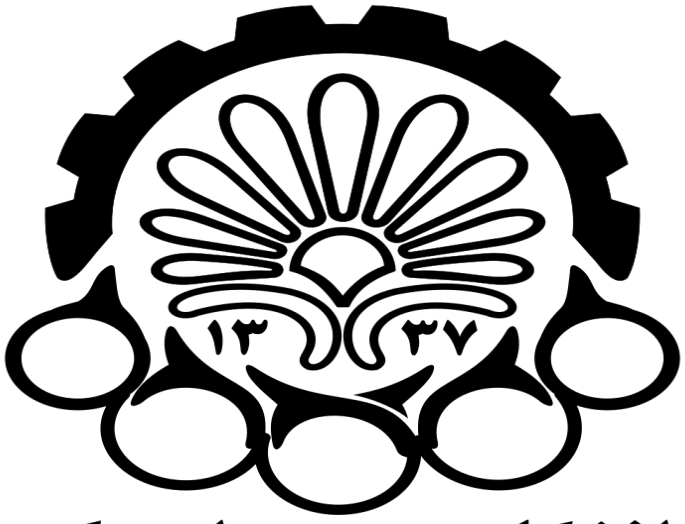


دانشگاه صنعتی امیرکبیر

(پلی تکنیک تهران)

درس رایانش عصبی و یادگیری عمیق

استاد صفابخش

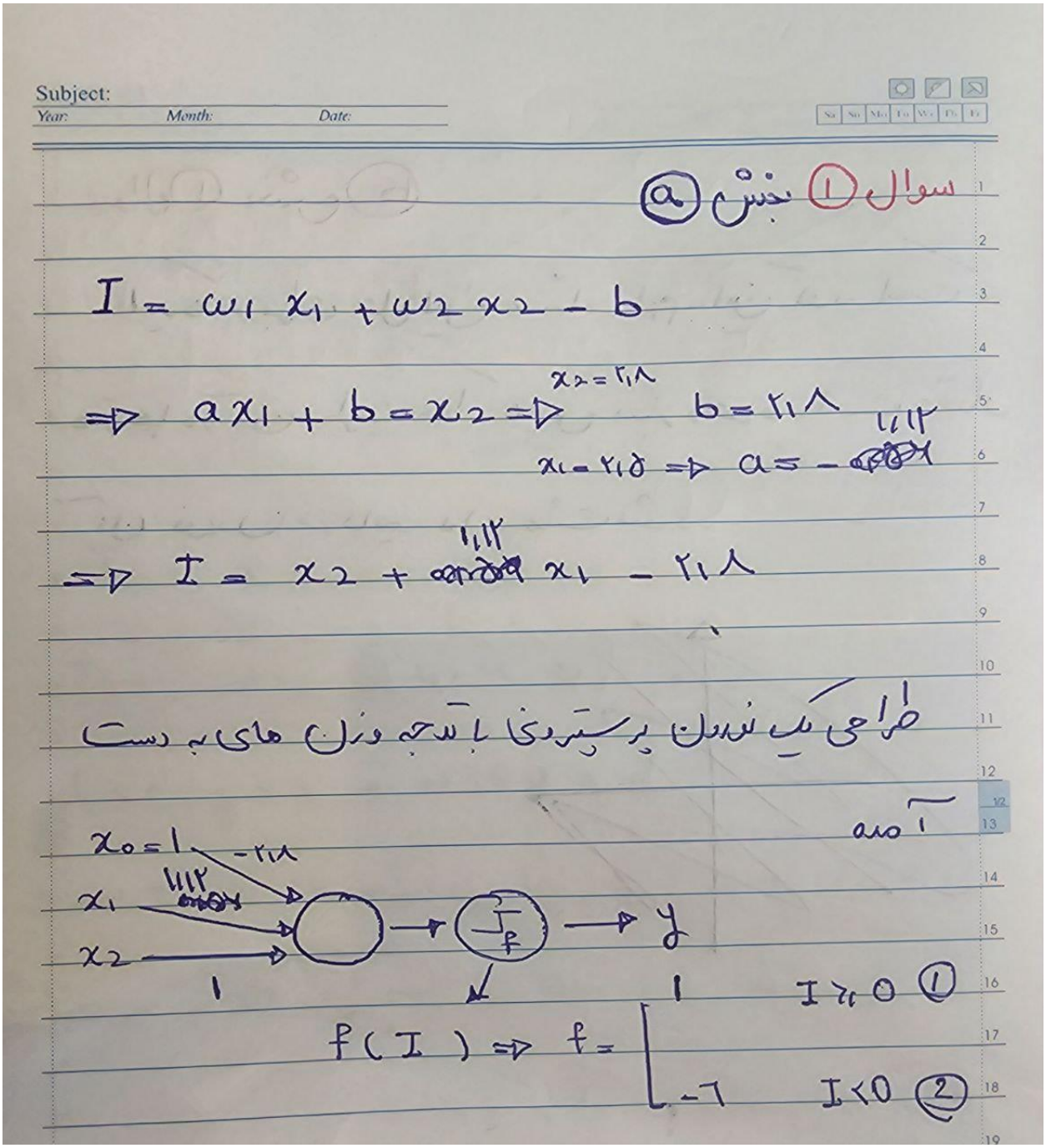
نیما پری فرد

۴۰۲۱۳۱۰۱۷

فهرست تمرین شماره ۱

تمرین ۱

a-۱:

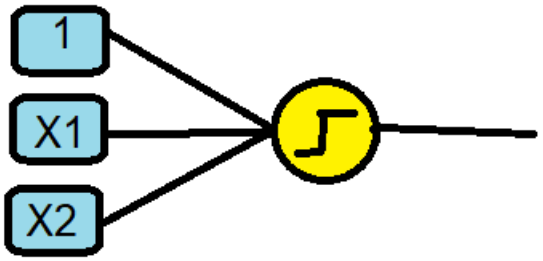

Subject:
Year: Month: Date:
سوال ① بخش ⓐ
I = w1 x1 + w2 x2 − b
a x1 + b = x2
x2 = 2.8
b = 2.8
x1 = 2.5
a = −1.12
I = x2 + 1.12 x1 − 2.8
طراحی یک نورون پرسپترونی با توجه وزن های به دست آمده
x0 = 1
−2.8
1.12
x1
x2
y
f(I)
f = 1
I ≥ 0 ①
−1
I < 0 ②

۱-b:

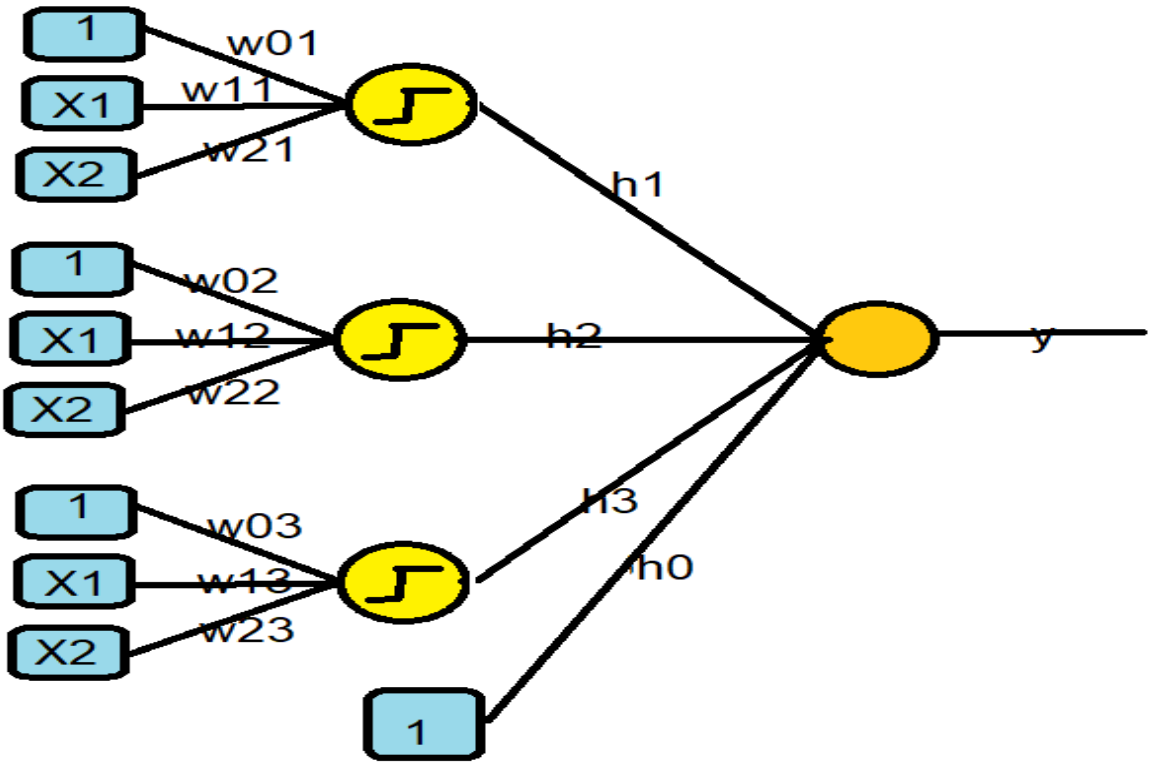
چون با یک خط نمی توان دو کلاس را از هم جدا کرد برای همین به صورت خطی جدایی پذیر نیست

برای حل این سوال نیاز به یک لایه مخفی داریم و در نهایت یک نورون در خروجی که خروجی این سه نوورن را ادغام کند.

برای حل این سوال به عنوان نورون پایه از نورون ۱۹۵۷ که دارای یک تابع خطی و تابع فعالیت پله بود استفاده می کنیم.



در نهایت معماری پیشنهاد شده به شکل زیر است.

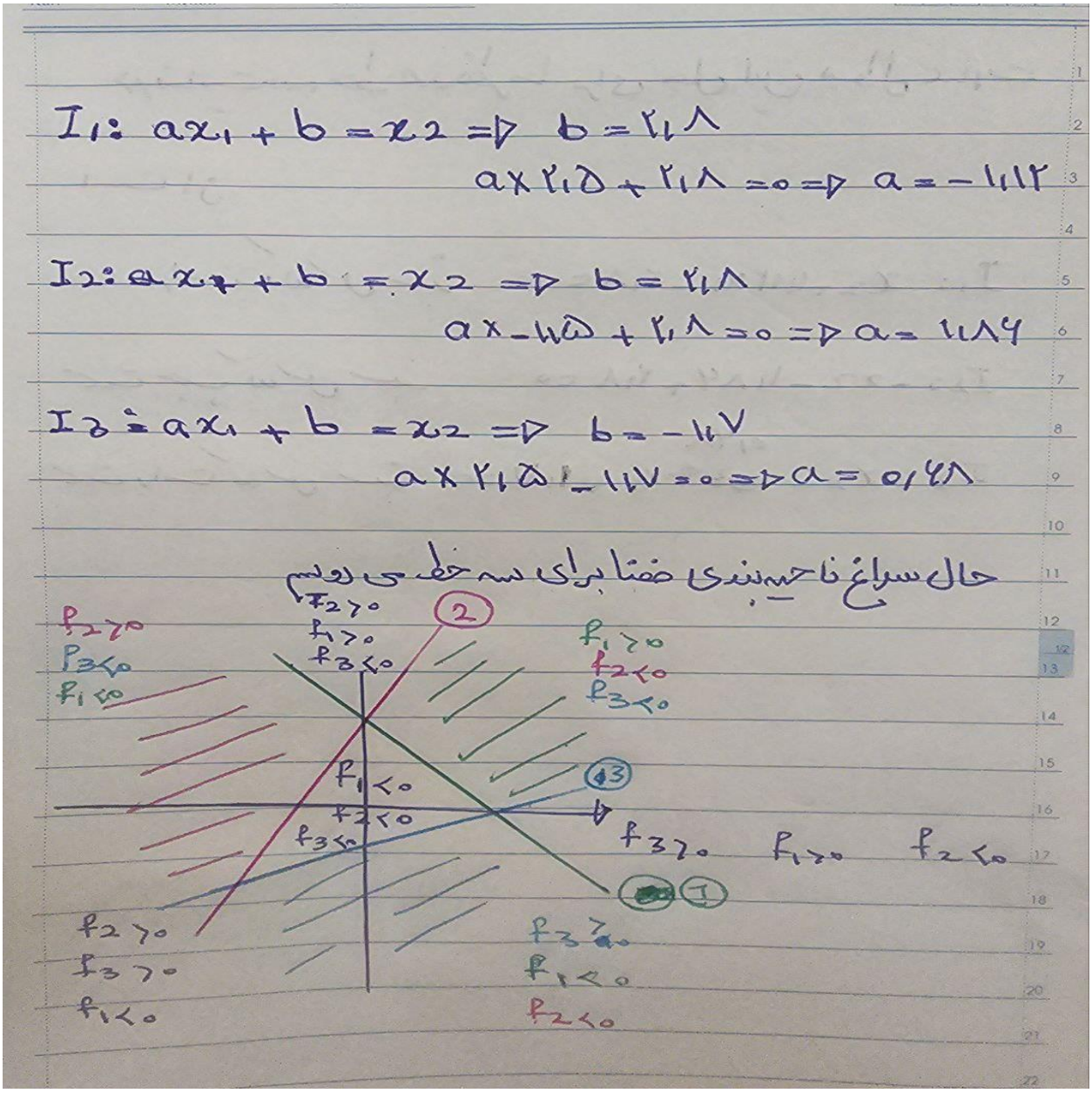


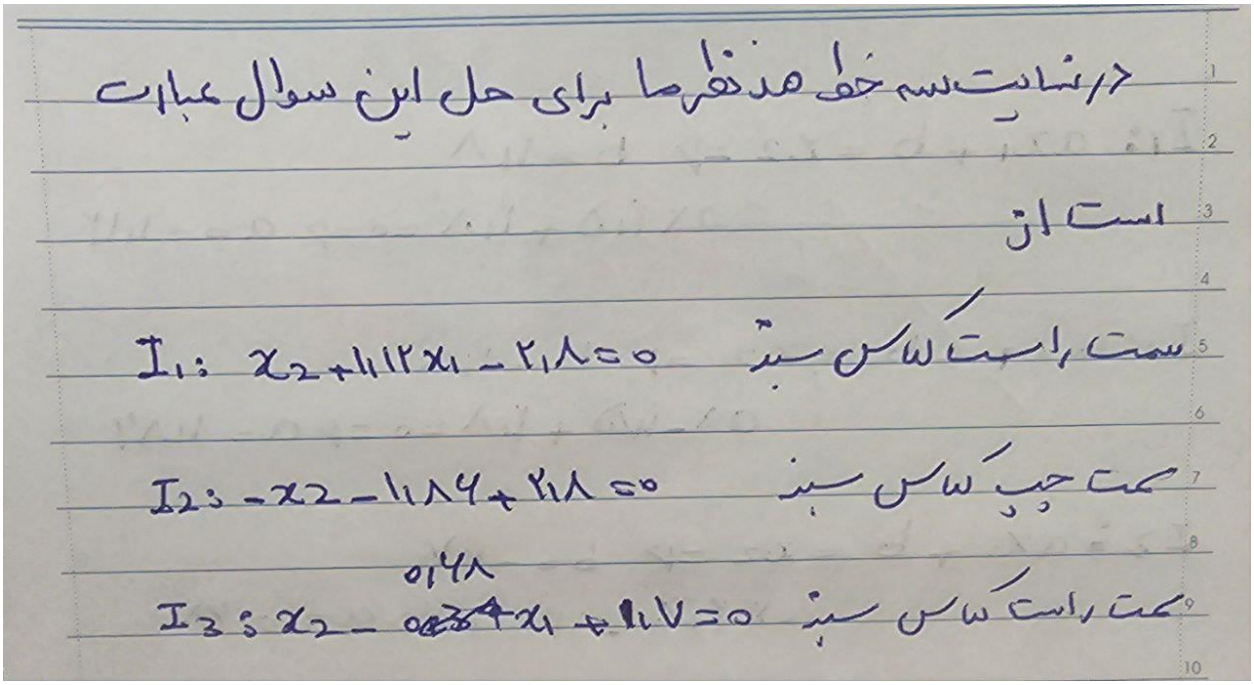
حال به دنبال پیدا کردن وزن های مناسب برای حل این سوال هستیم.

برای حل این سوال ایده بنده این است که اگر در محدوده کلاس آبی بیافتد هر سه خط کلاس مقدار -۱ خروجی بده.

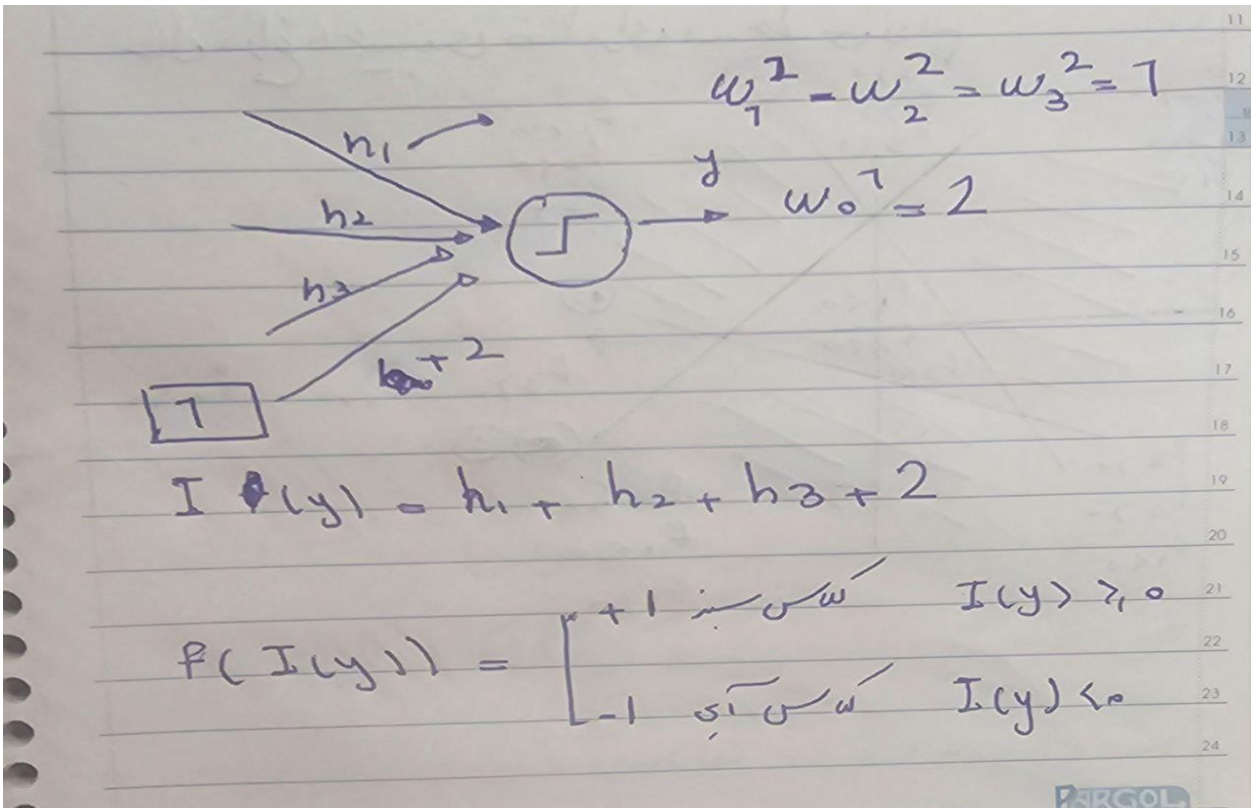
در نهایت وزن های h۱ و h۲ و h۳ جوری مقدار دهی کنیم که در نهایت خروجی y اگر کلاس سبز بود +۱ و اگر خروجی کلاس آبی -۱ خروجی بدهد.

حال در ابتدا سه خط جدا کننده در نورون های لایه مخفی را به دست می آوریم.





در این حالت خروجی برای فضایی که داده آبی در ان وجود دارد برابر با -۱ می شود و برای بقیه ناحیه بزرگتر از یک می شود.



۱-c:

Kernel Trick مفهومی در یادگیری ماشین است، به‌ویژه در زمینه‌ی ماشین‌های بردار پشتیبان (SVM). این امکان را فراهم می‌کند که داده‌های ورودی به صورت غیرخطی به یک فضای بعد بالاتر تبدیل شوند بدون اینکه مانند روش های دیگر افزایش بعد هزینه محاسباتی افزایش پیدا کند و ما دچار نفرین بعد شویم.

Kernel trick یک مرز تصمیم غیرخطی را با تبدیل ضمنی داده‌های ورودی به یک فضای بعد بالاتر پیدا می کند که در آن یک مرز تصمیم خطی می‌تواند ایجاد شود. این کار با استفاده از یک تابع هسته (Kernel Function) انجام می‌شود که ضرب داخلی بین جفت داده‌ها در فضای بعد بالاتر را بدون محاسبه تبدیل به صورت صریح محاسبه می‌کند. توابع هسته محبوب شامل تابع هسته خطی، تابع هسته چندجمله‌ای، تابع هسته گاوسی (RBF) و تابع هسته سیگموئیدی هستند.

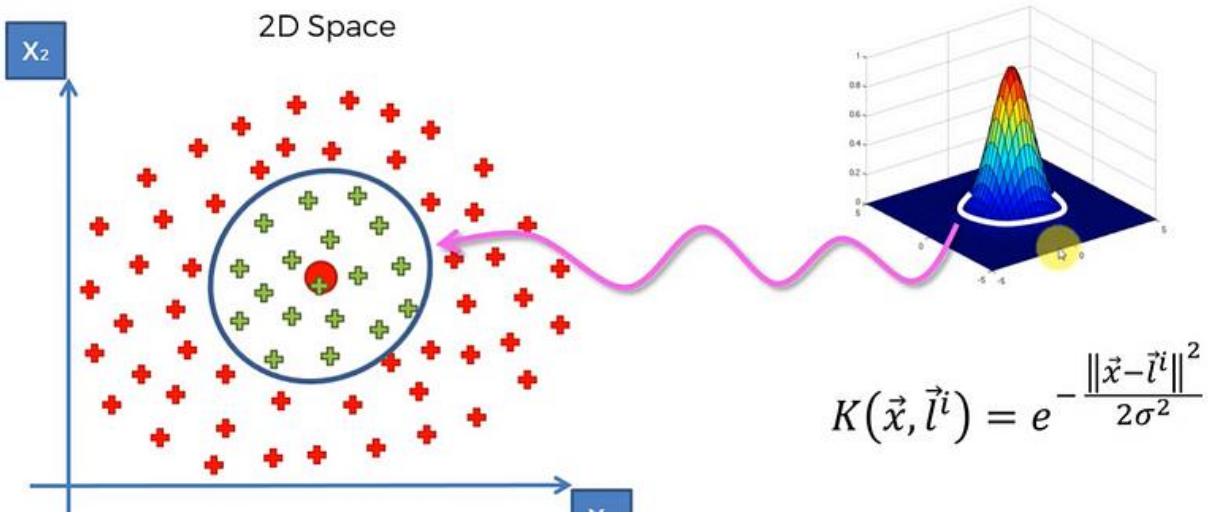
با استفاده از kernel trick، دسته بند می‌توانند به‌طور مؤثری داده‌ها را دسته‌بندی کنند که ممکن است در فضای ویژگی اصلی به صورت خطی جدایی‌پذیر نباشند و این باعث افزایش انعطاف‌پذیری و عملکرد آن‌ها در وظایف مختلف دسته‌بندی می‌شود.

شگرد کرنل روشی است که داده های ها را به بعد های بالاتر می برد بدون این که محاسبات ما بیش تر شود.

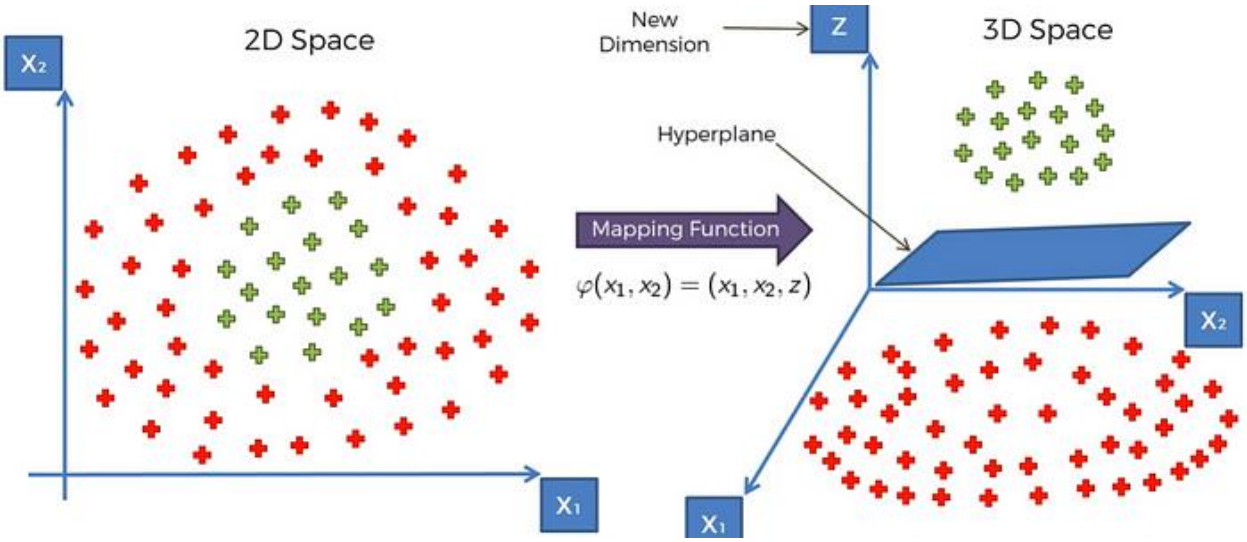
در این جا به نظر بنده بهترین کرنل rbf است چون می تواند به خوبی این شکل با استفاده از یک نورون و این کرنل از هم جدا کنیم.

برای مثال شکل زیر شبیه شکل ۱.b زیر است.

که با استفاده کرنل rbf همان طور که در شکل زیر مشاهده می کند شبیه شکل ماست و با افزایش به راحتی با استفاده از یک صفحه جدا شده است.



منبع این شکل و متن: https://medium.com/@suvigya/the-gaussian-rbf-kernel-in-۲۰۰۱۰aae۸۲۲c۱fb۲non-linear-svm-



$$K(\vec{x}, \vec{l}^i) = e^{-\frac{\|\vec{x}-\vec{l}^i\|^2}{2\sigma^2}}$$

تمرین ۲

برای پیاده سازی این تمرین در فایل perveptron.py یک نورون پرسپترونی را از صفر تا صد طبق نورون معرفی شده در اسلاید ها و سال ۱۹۵۷ پیاده سازی کردم.

```python
class Perceptron():
    new *
    def __init__(self, alpha, epochs):
        self.alpha = alpha
        self.epochs = epochs
        self.w = None
        self.train_errors = []
        self.validation_errors = []
        self.train_accuracies = []
        self.validation_accuracies = []
```

```
new *
def train(self, x_train, y_train, x_val=None, y_val=None):
    n_samples, n_features = x_train.shape
    self.w = np.random.rand(n_features)

    for epoch in range(self.epochs):
        # Forward pass
        y_pred_train = self.step_function(np.dot(x_train, self.w))

        # Calculate accuracy
        train_accuracy = accuracy_score(y_train, y_pred_train)
        self.train_accuracies.append(train_accuracy)

        # Update lists of errors
        train_error = 1 - train_accuracy
        self.train_errors.append(train_error)


        # Backpropagation
        error = y_train - y_pred_train

        if epoch % 10 ==0:
            print(f"Epoch {epoch} Error: {train_error}")

        # Update weights using gradient descent
        self.w += self.alpha * np.dot(x_train.T, error) / n_samples

        y_pred_val = self.step_function(np.dot(x_val, self.w))

        val_accuracy = accuracy_score(y_val, y_pred_val)
        self.validation_accuracies.append(val_accuracy)

        val_error = 1 - val_accuracy
        self.validation_errors.append(val_error)

3 usages  new *
def step_function(self, x):
    return np.where(x >= 0, 1.0, 0.0)
```

۲-a:

مشخصات پرسپترون آموزش داده شده:

۱- تابع خطا: پله

۲- بهینه ساز: گرادیان کاهشی

۳- معماری شبکه: از یک پرسپترون استفاده کردیم.

٤- تعداد گام آموزش: ۴۰

٥- اندازه دسته: برای آموزش کل داده به اموزش می دادم در واقعا اندازه برابر است با کل داده بنده.

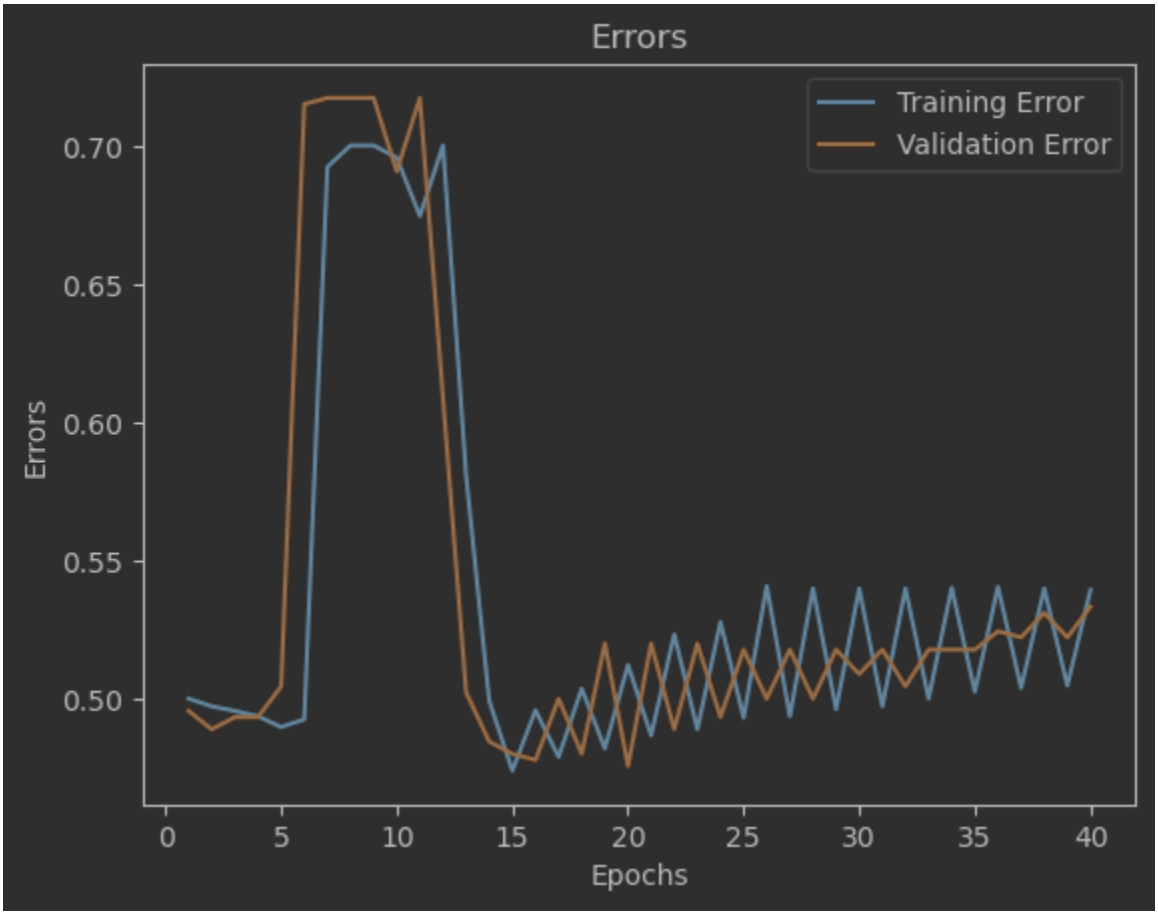
٦- آمارگان تفکیک داده: طبق متن گفته شده به نسبت ۸۰ ۱۰ ۱۰ داده را تقسیم کردم.

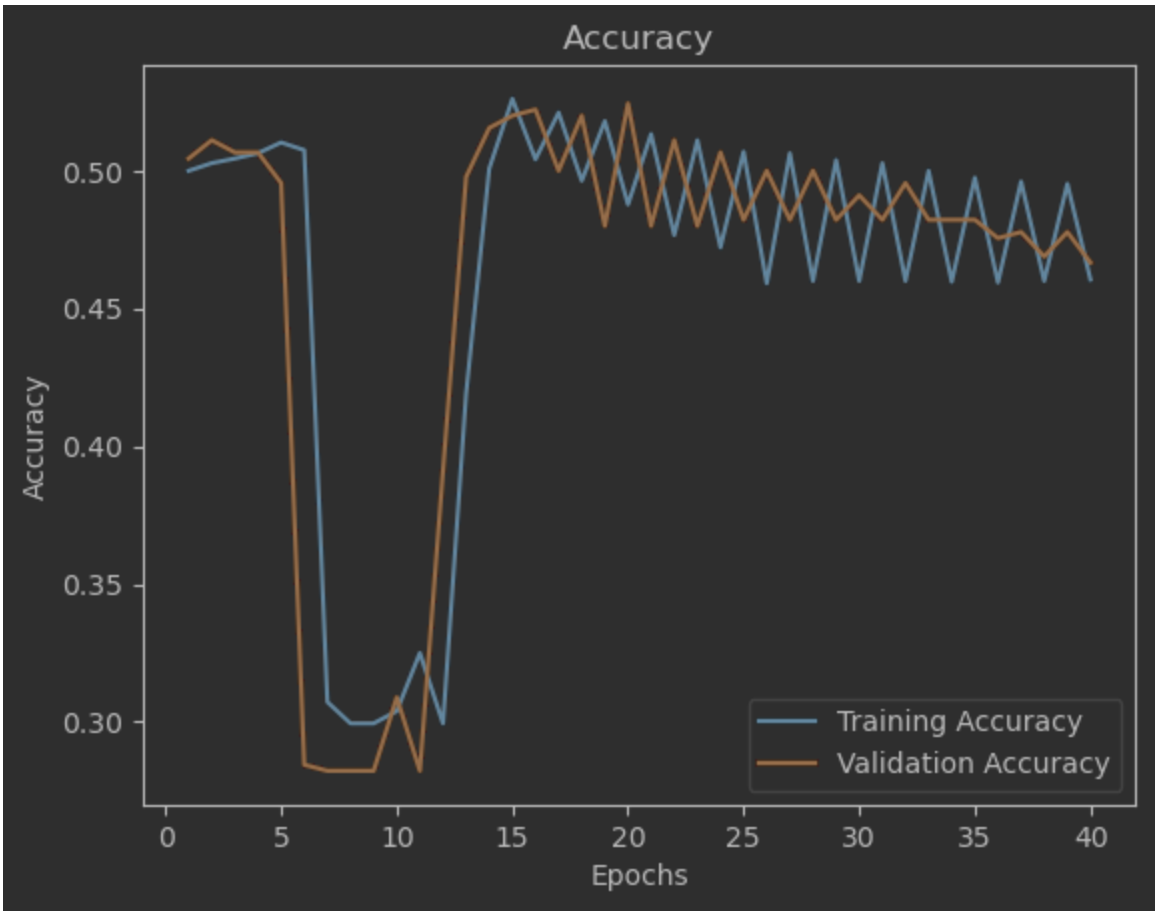
```
Train set: (3600, 3) (3600,)
Validation set: (450, 3) (450,)
Test set: (450, 3) (450,)
```

۷- پیش پردازش اعمال شده: در مورد اول پیش پردازشی اعمال نشده.

۸- نرخ یادگیری: ۰.۱

منحنی یادگیری و منحنی دقت:





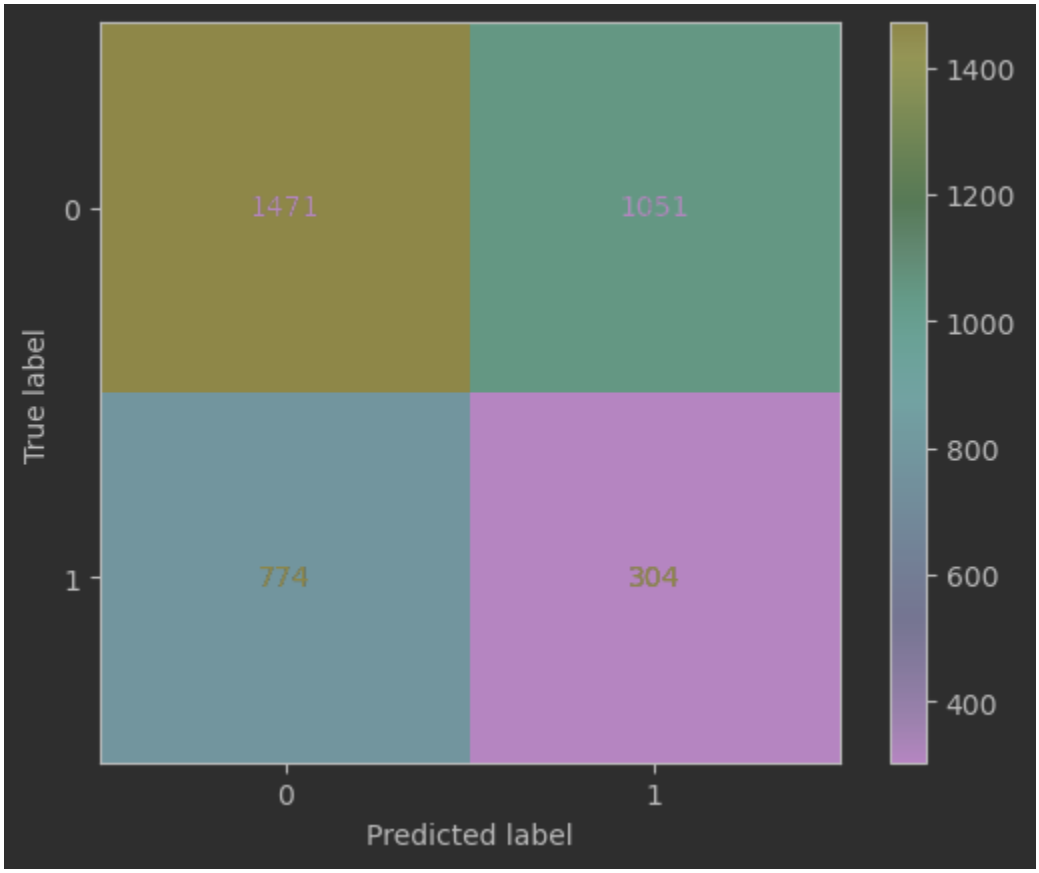
اگر بخواهم تحلیلی در مورد این نمودار ها بدهم علت زیگزاگ بودن این مدل می تواند این باشد توانایی این مدل این قدر است و داده جدایی پذیر خطی نیست و نمی تواند صفحه ای مناسب پیدا کند که داده ها از هم جدا کند

دقت و معیار f۱ به سه دسته داده مد نظر:

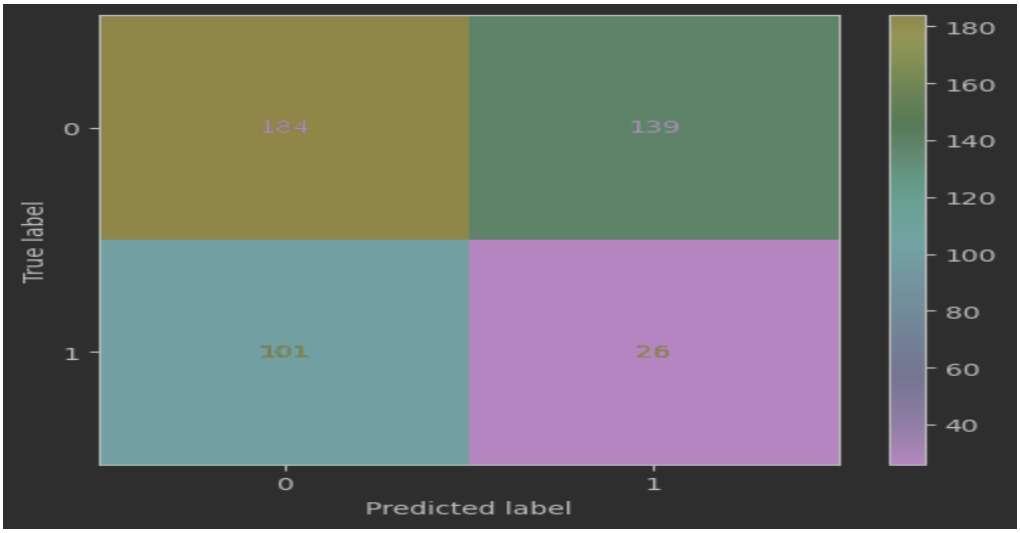
```
Accuracy for Train Data: 0.4931
F1-score for Train Data: 0.2499
Accuracy for Validation Data: 0.4667
F1-score for Validation Data: 0.1781
Accuracy for Test Data: 0.4600
F1-score for Test Data: 0.2430
```

ماتریس confusion برای سه دسته داده:

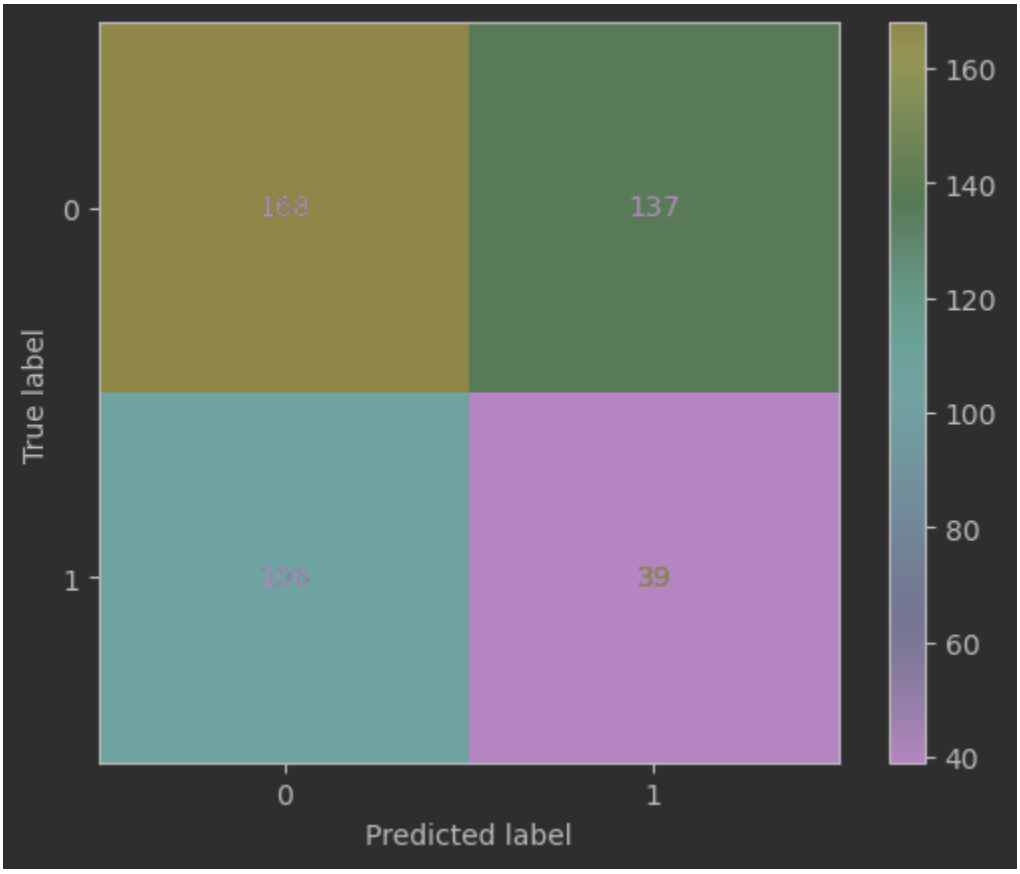
داده آموزش:



داده validation:



داده تست:

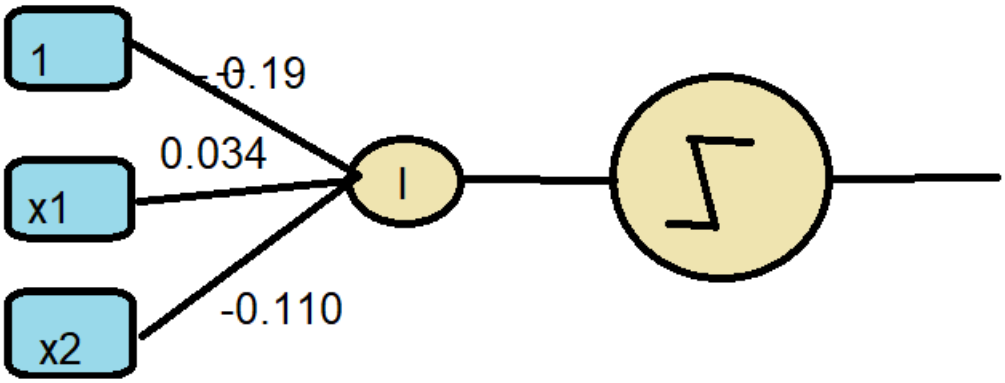


وزن های مدل آموزش داده شده:

```
weights of the perceptron after training


w0 : -0.19725401863209238
w1 : 0.034725863066346424
w2 : -0.11025430833193417
```

طرحواره نورون پرسپترونی:



۲-b:

۱- تابع خطا: پله

۲- بهینه ساز: گرادیان کاهشی

۳- معماری شبکه: از یک پرسپترون استفاده کردیم.

٤- تعداد گام آموزش: ۱۳۰

٥- اندازه دسته: برای آموزش کل داده به اموزش می دادم در واقعا اندازه برابر است با کل داده بنده.

٦- آمارگان تفکیک داده: طبق متن گفته شده به نسبت ۸۰ ۱۰ ۱۰ داده را تقسیم کردم.

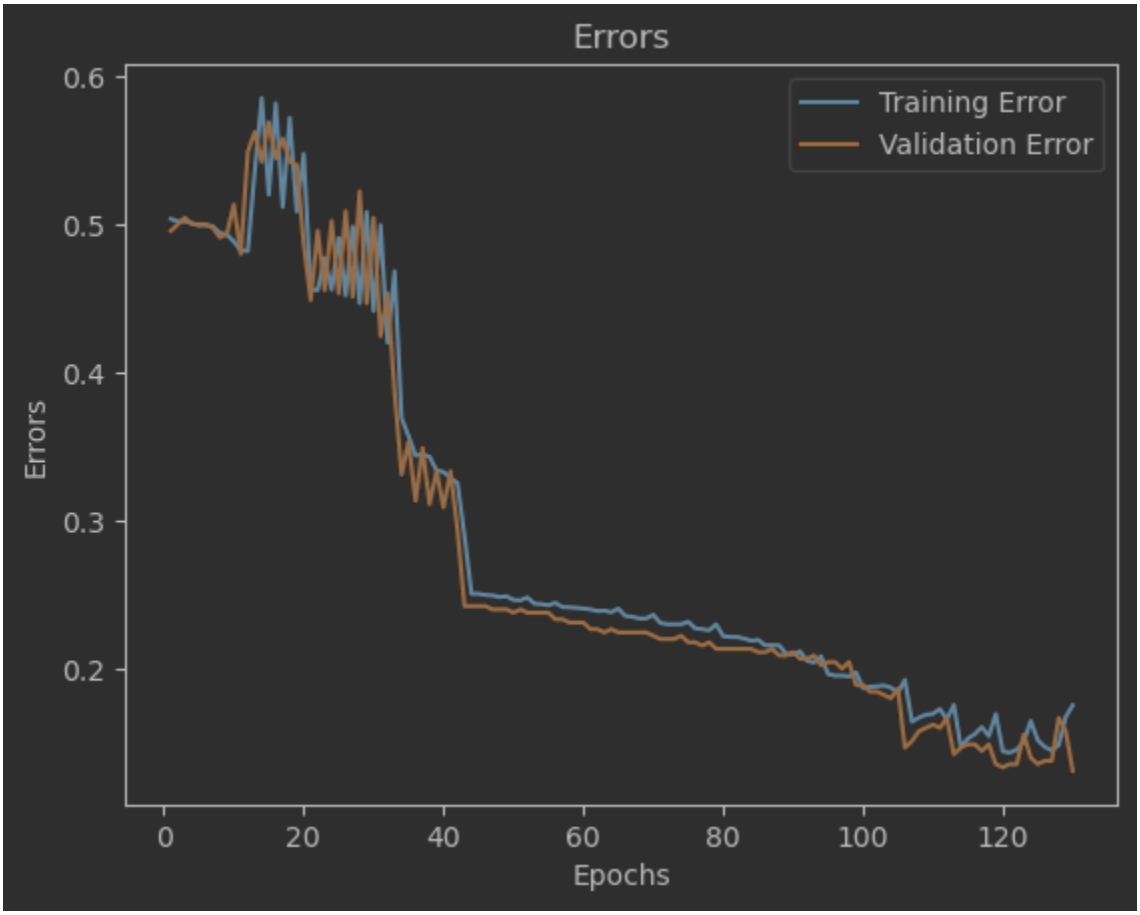
```
Train set: (3600, 7) (3600,)
Validation set: (450, 7) (450,)
Test set: (450, 7) (450,)
```

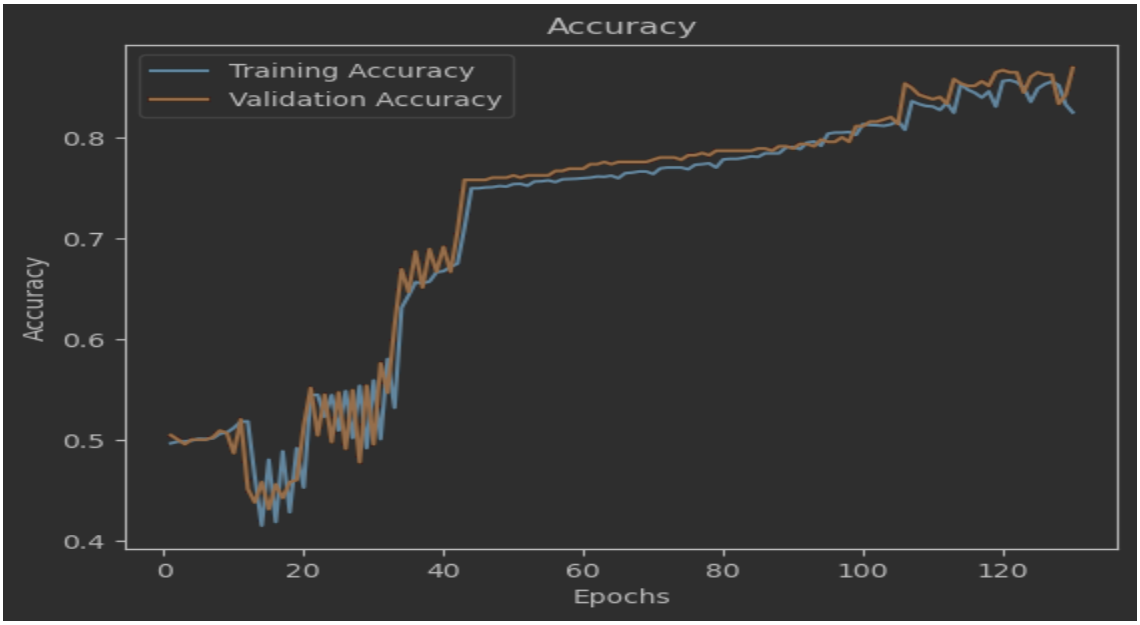
۷- پیش پردازش اعمال شده: تا توان سوم ویژگی ها اضافه شد.

	x1	x2	y	x1_2	x2_2	x1_3	x2_3
0	4.780310	2.698320	0	22.851364	7.280931	109.236602	19.646281
1	-2.273710	5.750180	0	5.169757	33.064570	-11.754529	190.127229
2	-4.817170	-2.209610	0	23.205127	4.882376	-111.783041	-10.788148
3	-9.526940	0.659360	0	90.762586	0.434756	-864.689709	0.286660
4	1.864400	-5.019510	0	3.475987	25.195481	6.480631	-126.468967
5	-2.850480	-8.056520	0	8.125236	64.907515	-23.160823	-522.928689
6	-5.971380	-1.058090	0	35.657379	1.119554	-212.923760	-1.184589
7	7.739690	0.972351	0	59.902801	0.945466	463.629112	0.919325
8	2.307780	6.916060	0	5.325849	47.831886	12.290887	330.808193
9	-1.859580	6.062970	0	3.458038	36.759605	-6.430498	222.872384

۸- نرخ یادگیری: ۰.۱

منحنی یادگیری دقت و صحت:





توان های ویژگی ها به مسئله اضافه شده و خب دقت بهتر شده است اما همچنان مسیر آموزش دارای حرکات زیگزاگی است که شاید به این خاطر است که ارتباط ویژگی ها را در نظر گرفته ایم.

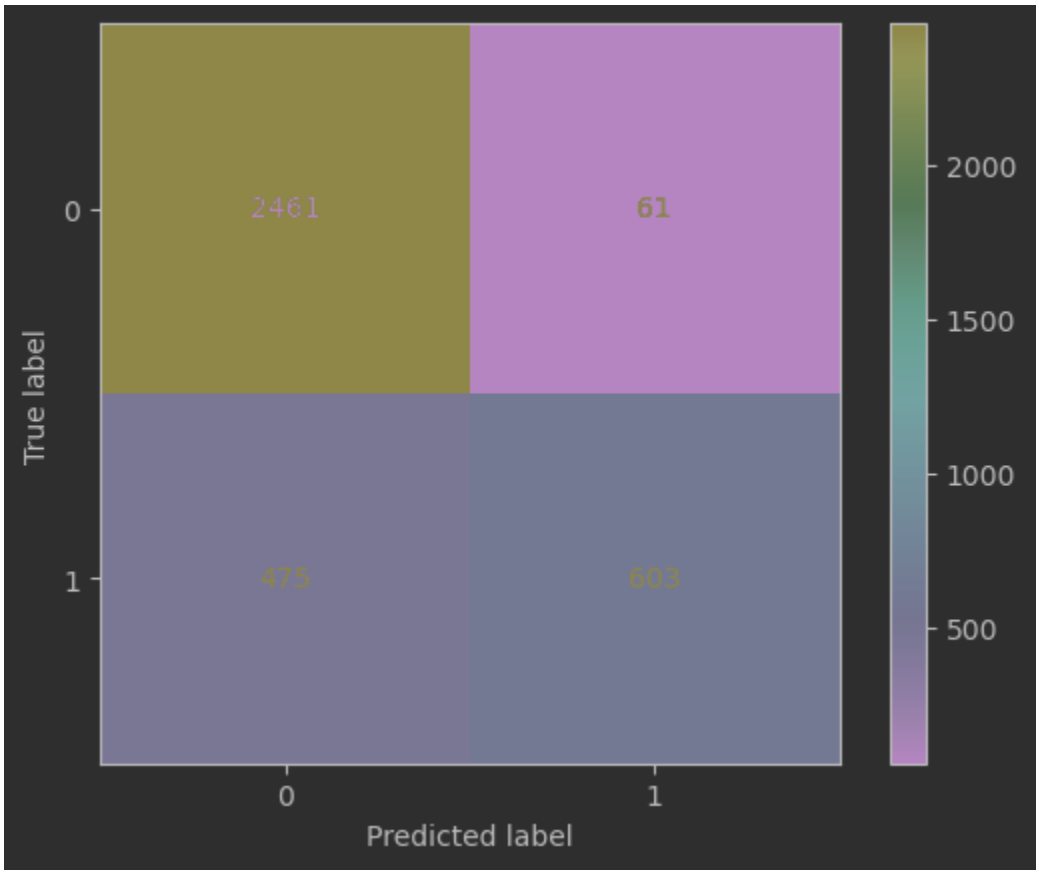
بعد تا گام ۱۳۰ پیش رفتم چن بعد آن اورفیت می شد و آموزش را همین جا نگه داشتم.

دقت و معیار f۱ به سه دسته داده مد نظر:

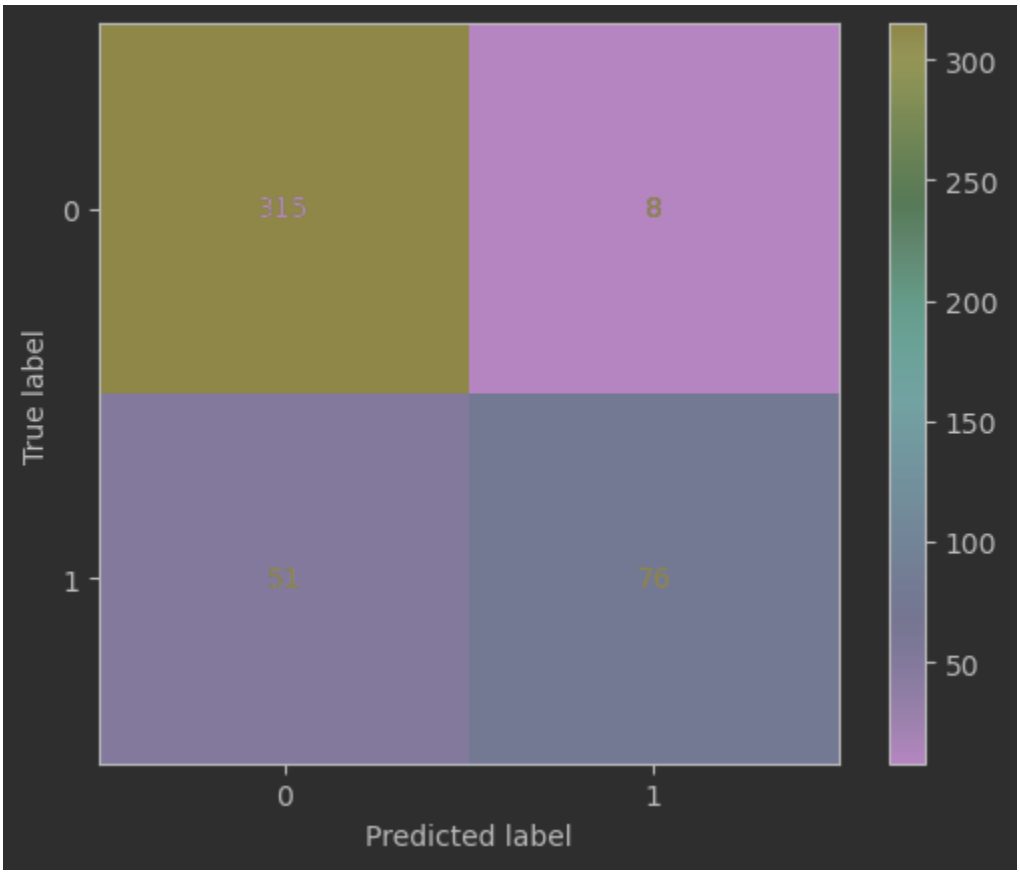
```
Accuracy for Train Data: 0.8511
F1-score for Train Data: 0.6923
Accuracy for Validation Data: 0.8689
F1-score for Validation Data: 0.7204
Accuracy for Test Data: 0.8622
F1-score for Test Data: 0.7373
```

ماتریس confusion برای سه دسته داده:

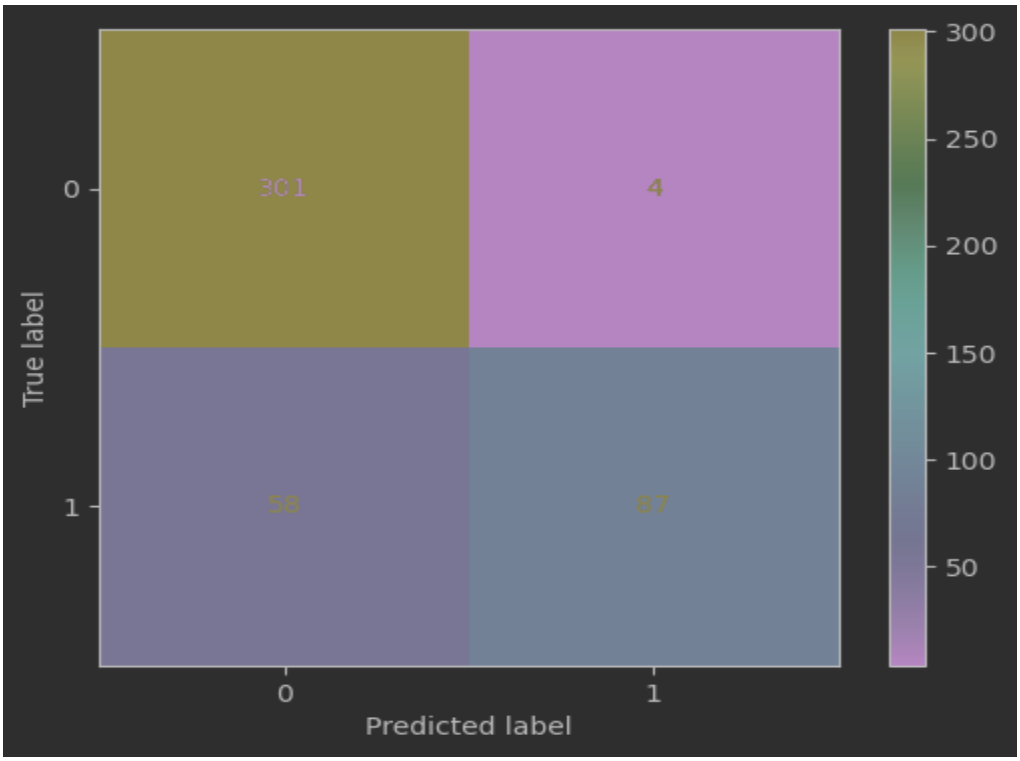
داده آموزش:



داده validation:



داده تست:



وزن های مدل آموزش داده شده:

```
weights of the perceptron after training


w0 : 0.49010267745795755
w1 : 0.015618092299819825
w2 : 0.4035930522431602
w3 : -0.03255889264873556
w4 : -0.08290794308412852
w5 : -0.00036837626156942314
w6 : -0.006550746375800094
```

۲-c:

۱- تابع خطا: پله

۲- بهینه ساز: گرادیان کاهشی

۳- معماری شبکه: از یک پرسپترون استفاده کردیم.

٤- تعداد گام آموزش: ۱۷۰

۵- اندازه دسته: برای آموزش کل داده به اموزش می دادم در واقعا اندازه برابر است با کل داده بنده.

٦- آمارگان تفکیک داده: طبق متن گفته شده به نسبت ۸۰ ۱۰ ۱۰ داده را تقسیم کردم.

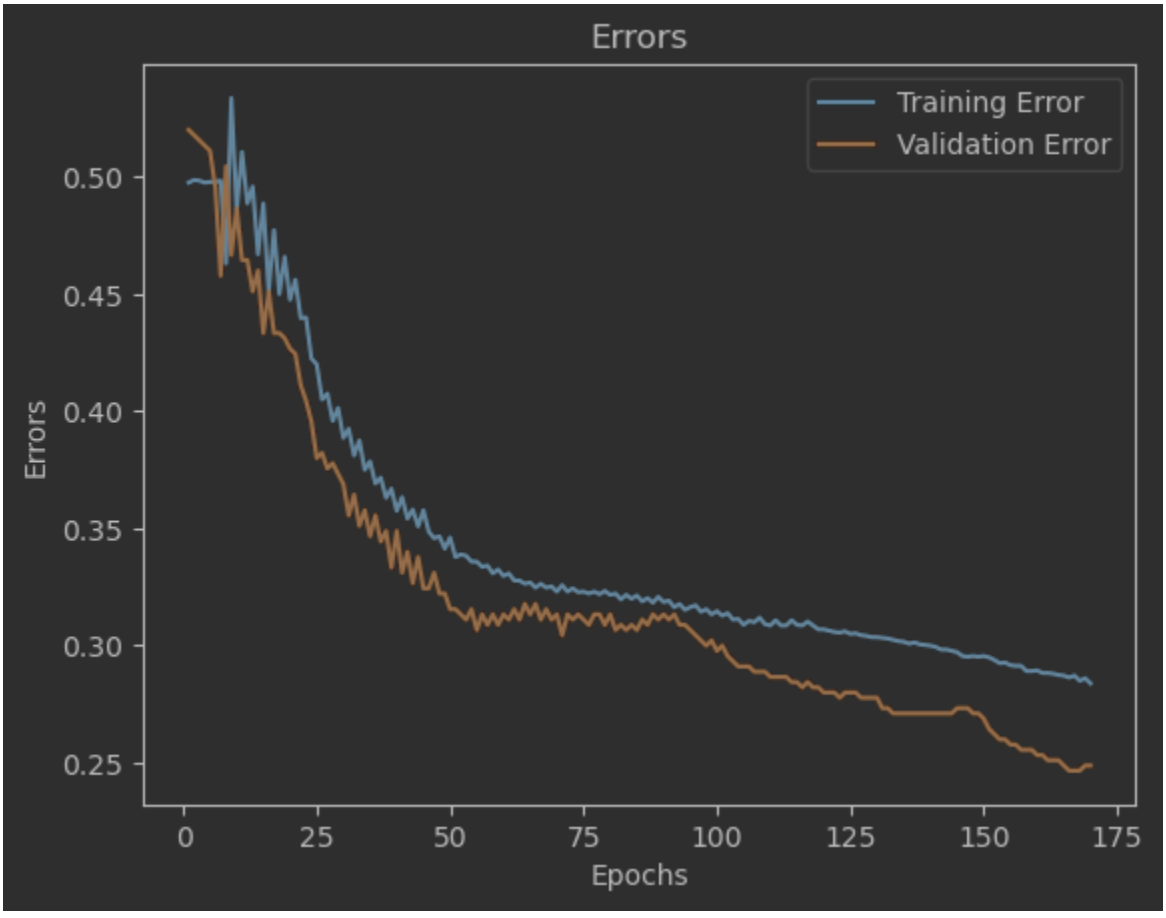
```
Train set: (3600, 13) (3600,)
Validation set: (450, 13) (450,)
Test set: (450, 13) (450,)
```

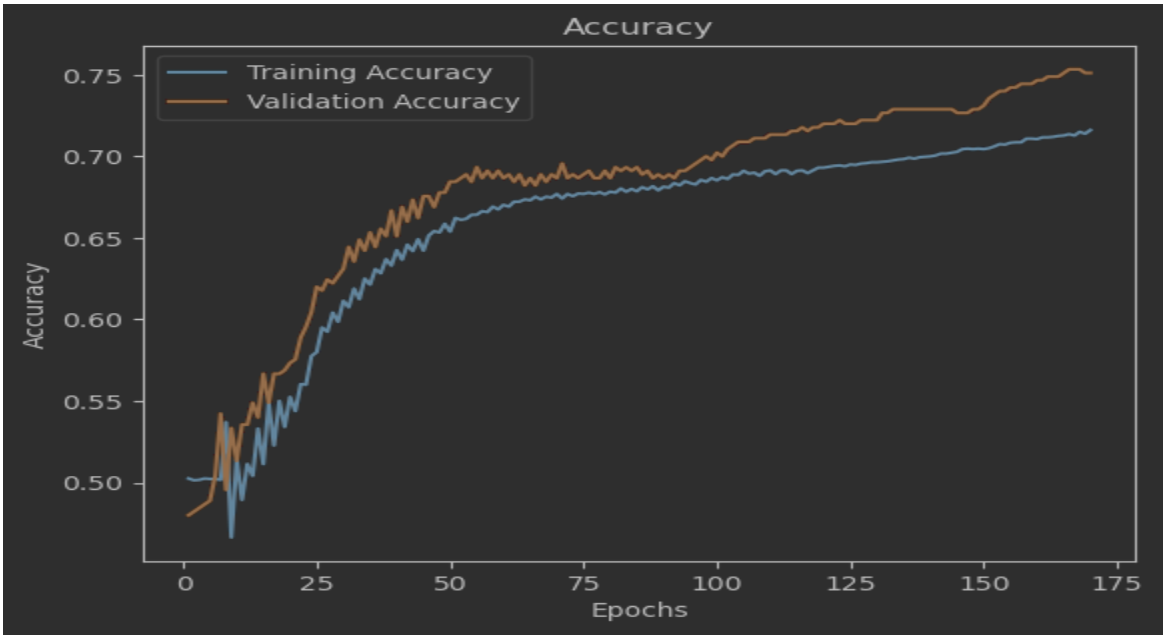
۷- پیش پردازش اعمال شده: تا توان سوم ویژگی ها اضافه شد.

x2_2	x1_3	x2_3	x1*x2	x1*x2_2	x2*x1_2	x2*x1_3	x1_2*x2_2	x2_2*x1_3
7.280931	109.236602	19.646281	12.898806	34.805106	61.660292	294.755309	166.379198	795.344145
33.064570	-11.754529	190.127229	-13.074242	-75.179244	29.727034	-67.590655	170.935798	-388.658433
4.882376	-111.783041	-10.788148	10.644067	-23.519237	-51.274280	246.996925	113.296162	-545.766875
0.434756	-864.689709	0.286660	-6.281683	-4.141891	59.845219	-570.141806	39.459543	-375.928701
25.195481	6.480631	-126.468967	-9.358374	46.974454	-17.447753	-32.529591	87.579172	163.282609
64.907515	-23.160823	-522.928689	22.964949	-185.017572	-65.461128	186.595637	527.388889	-1503.311479
1.119554	-212.923760	-1.184589	6.318257	-6.685285	-37.728716	225.292502	39.928377	-238.379743
0.945466	463.629112	0.919325	7.525695	7.317617	58.246549	450.810231	56.636090	438.345779
47.831886	12.290887	330.808193	15.960745	110.385470	36.833888	85.004510	254.745379	587.896291
36.759605	-6.430498	222.872384	-11.274578	-68.357427	20.965979	-38.987916	127.116103	-236.382564

۸- نرخ یادگیری: ۰.۰۰۰۱

منحنی یادگیری دقت و صحت:





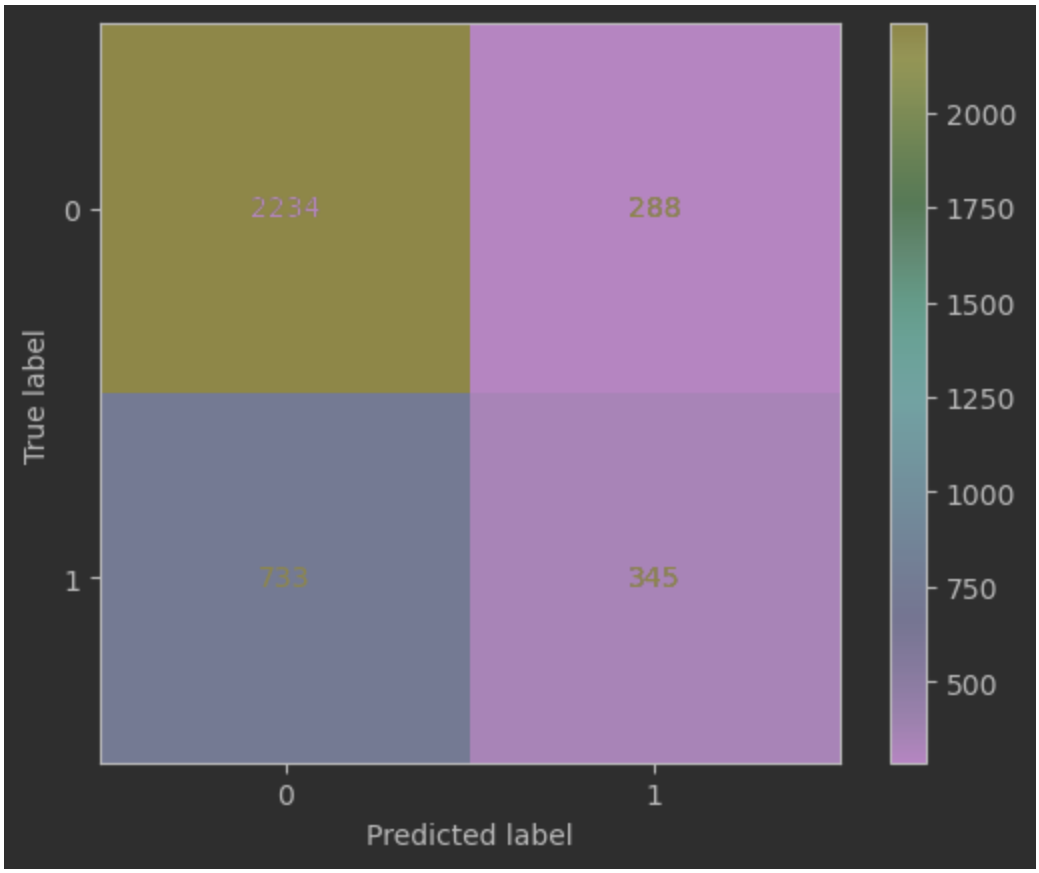
اگر بخوهیم کمی نمودار را تحلیل کنیم واضح است که فرم بهتری گرفته اما مشکلی دارد این است مسئله داده کافی برای اضافه کردن این تعداد ویژگی ندارد و با اضافه کردند ویژگی ها دچار نفرین بعد می شود.

دقت و معیار f۱ به سه دسته داده مد نظر:

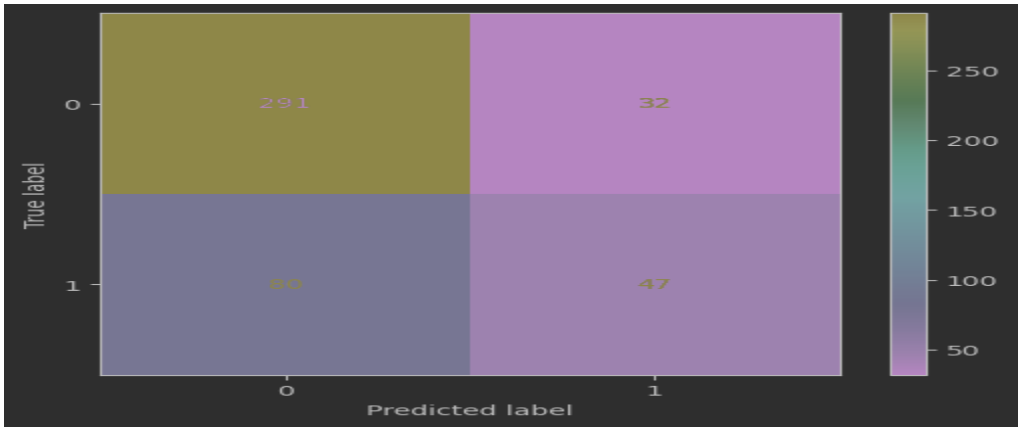
```
Accuracy for Train Data: 0.7164
F1-score for Train Data: 0.4033
Accuracy for Validation Data: 0.7511
F1-score for Validation Data: 0.4563
Accuracy for Test Data: 0.7044
F1-score for Test Data: 0.4242
```

ماتریس confusion برای سه دسته داده:

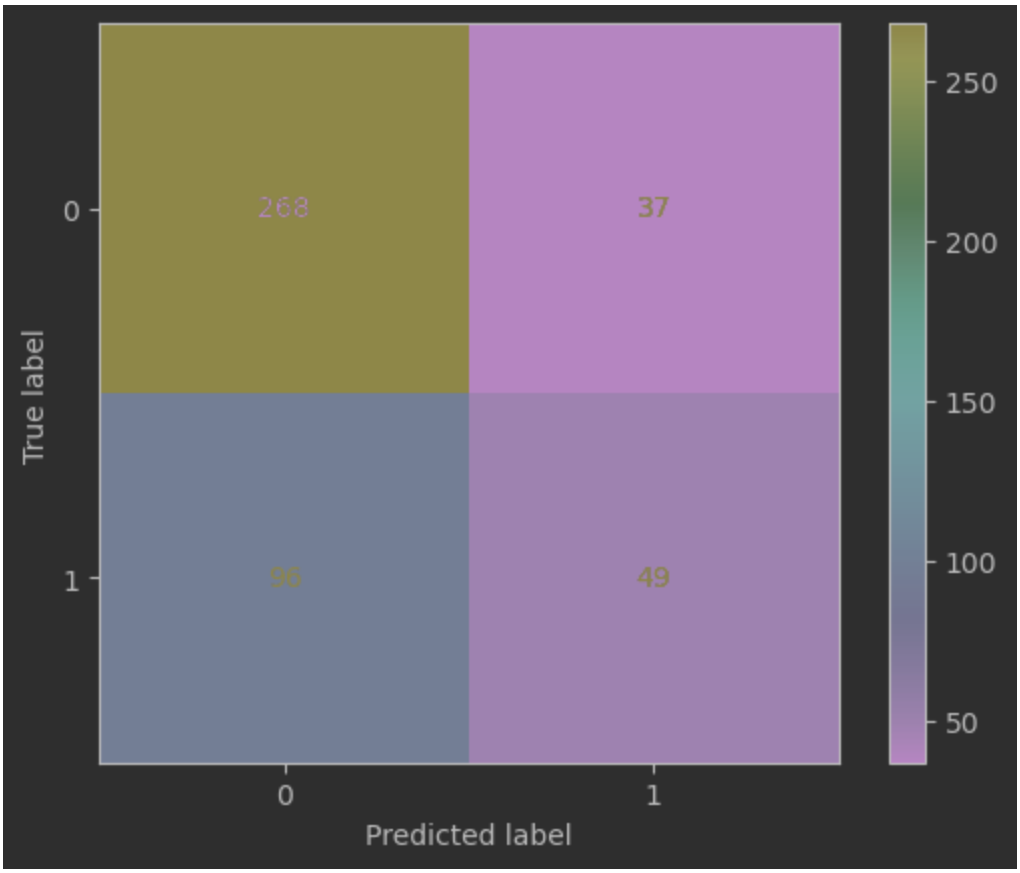
داده آموزش:



داده validation:



داده تست:



وزن های مدل آموزش داده شده:

```
weights of the perceptron after training


w0 : 0.5946940759488579
w1 : 0.98458946673593
w2 : 0.41188342795698063
w3 : 0.1696747851099382
w4 : 0.07086214179883416
w5 : 0.12432701305777609
w6 : 0.12706409439551647
w7 : 0.1609449283315463
w8 : 0.5783211297650717
w9 : 0.1694962849035778
w10 : 0.01860443368945021
```

با افزودن ضرب دو به دو این ویژگی‌ها، علاوه بر افزایش بعد، ویژگی‌هایی را نیز به مدل اضافه کردیم که نشان‌دهنده تعامل بین ورودی‌های اولیه ما است. به عبارت دیگر، با این کار، ما ارتباط و وابستگی بین ویژگی‌ها را بهتر مدل می‌کنیم. این اصطلاح را به گونه‌ای بیان کنیم که برداشت‌پذیری آن تشویق به ادامه کار کند.

می توان گفت در قسمت قبل ما توان های ویژگی ها را افزایش داده تا با استفاده از آن دقت بالا ببریم و در این جا دنبال ارتباط ویژگی های ساخته شده جهت افزایش دقت هستیم.در واقع این جا ما بد مسئله تا ۳ افزایش دادیم تا بررسی کنیم ایا در بعد ۳ جداپذیر خطی هست یا خیر.

تمرین ۳

۳-a:

سری تیلور یک ابزار ریاضی قدرتمند است که برای تقریب توابع با چند جمله‌ای استفاده می‌شود. گسترش سری تیلور یک تقریب چند جمله‌ای از رفتار تابع در اطراف یک نقطه مشخص فراهم می‌کند. با اضافه کردن جملات بالاتر در سری، می‌توان دقت بیشتری در تقریب تابع به دست آورد.

برای استفاده از سری تیلور برای تخمین یک تابع، معمولاً یک نقطه را انتخاب می‌کنیم که در آن تابع خوب رفتار کند، و سری را به یک نقطه معین قطع کرده وابسته به دقت مورد نظر و پیچیدگی تابع.

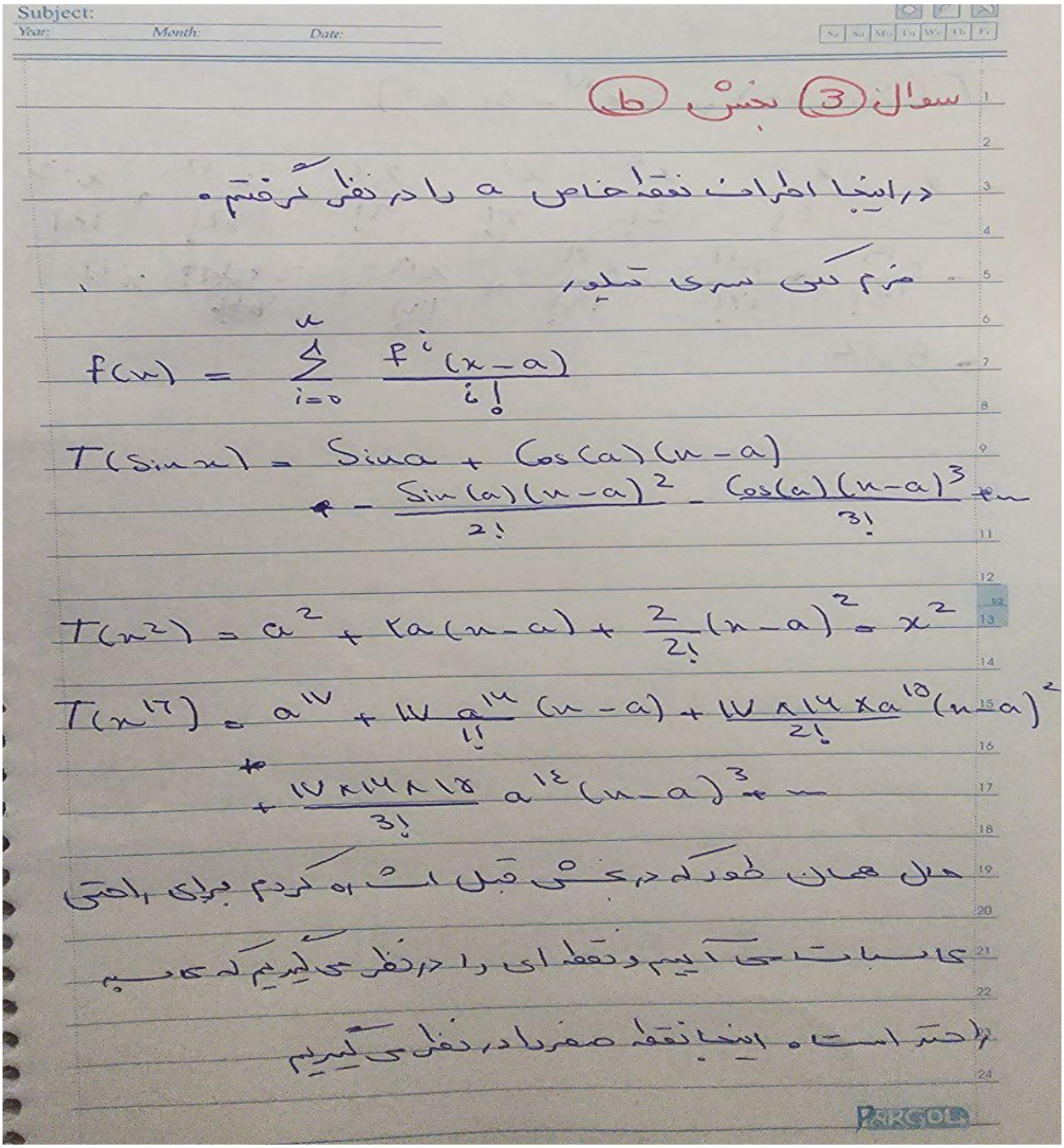
در شکل زیر از سری تیلور به این گونه استفاده می کنیم که میخواهیم مقدار تابع در نقطه x که نزدیک نقطه a است را حساب کنیم.

ثابت شده است که زمانی که n به سمت به نهایت می رود تابع ما همگرا می شود به مقدار واقعی.

$$f(x) \approx T_n(x) = f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f^{(2)}(a)}{2!}(x-a)^2 + \frac{f^{(3)}(a)}{3!}(x-a)^3 + \cdots + \frac{f^{(n)}(a)}{n!}(x-a)^n.$$

۳-b:

سری تیلور تقریبی از نقاط اطراف یک نقطه به ما می دهد.



سوال ③ بخش ط

در اینجا اطراف نقطه خاص a را در نظر گرفتم.

فرم کلی سری تیلور

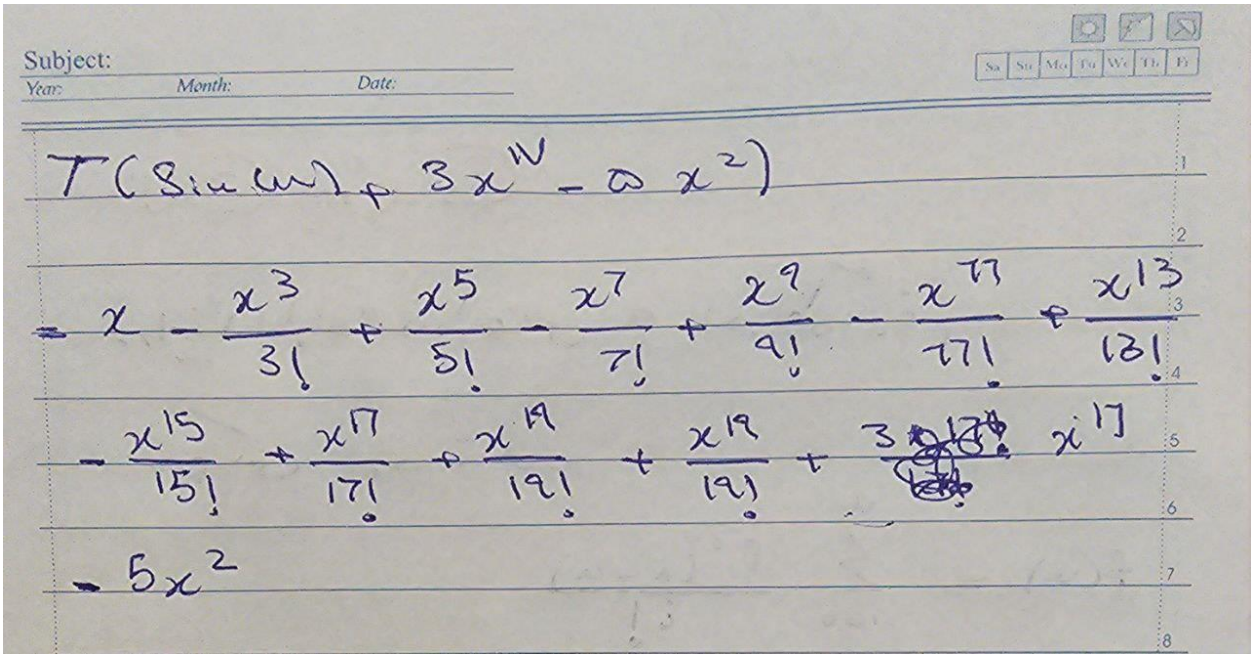
$$f(x) = \sum_{i=0}^{n} \frac{f^{i}(x-a)}{i!}$$

$$T(\sin x) = \sin a + \cos(a)(x-a) + \frac{-\sin(a)(x-a)^2}{2!} - \frac{\cos(a)(x-a)^3}{3!} + \dots$$

$$T(x^2) = a^2 + 2a(x-a) + \frac{2}{2!}(x-a)^2 = x^2$$

$$T(x^{17}) = a^{17} + \frac{17\,a^{16}}{1!}(x-a) + \frac{17\times16\times a^{15}}{2!}(x-a)^2 + \frac{17\times16\times15}{3!}a^{14}(x-a)^3 + \dots$$

حل همان طور که در بخش قبل اشاره کردم برای راحتی محاسبات سعی میکنیم نقطه ای را در نظر بگیریم که ساده تر است و اینجا نقطه صفر را در نظر میگیریم



c-۳:

برای پیاده سازی صفر تا صد پرسپترون برای مسئله رگرسیون را انجام دادم.

```python
class Perceptron():
    def __init__(self, alpha, epochs):
        self.alpha = alpha
        self.epochs = epochs
        self.w = None
        self.b = None
        self.train_errors = []
        self.validation_errors = []
```

در تابع train آموزش پرسپترون انجام می شود گرادیان محاسبه می شود و backpropagate می شود تا وزن ها به روز شوند.

```python
def train(self, x_train, y_train):
    n_samples, n_features = x_train.shape
    self.w = np.zeros(n_features)
    self.b = 0

    for epoch in range(self.epochs):
        # Forward pass
        y_pred_train = self.predict(x_train)

        # Compute errors
        train_error = self.rmse(y_train, y_pred_train)
        if epoch % 100 == 0:
            print(f"rmse error in epoch {epoch} is {train_error}")

        # Update lists of errors
        self.train_errors.append(train_error)

        # Backpropagation
        error = y_train - y_pred_train

        # Update weights and bias using gradient descent
        self.w += self.alpha * np.dot(x_train.T, error) / n_samples
        self.b += self.alpha * np.sum(error) / n_samples
```

با استفاده از دو تابع زیر منحنی خطا و منحنی مقایسه تابع به دست امده از سری تیلور و تابع اصلی انجام می شود.

```python
def plot_learning_curve(self, n):
    plt.plot(range(1, self.epochs + 1), self.train_errors, label='Training Error')
    plt.xlabel('Epochs')
    plt.ylabel(f"learning curve for Teylor Series with alpha {n} sentence")
    plt.title('Learning Curve')
    plt.legend()
    plt.show()

def plot_functions(self, x_range, target_function, title, dataset=None):
    plt.plot(x_range, target_function, label='Target Function', color='red')
    predicted_function =  self.predict(dataset)
    plt.plot(x_range, predicted_function, label='Predicted Function', color='blue')
    plt.xlabel('x')
    plt.ylabel('y')
    plt.title(f'Predicted vs. Target Function for {title}')
    plt.legend()
    plt.show()
```

این تابع فرمول به دست آمده به شکل بهتری بر می گرداند که در خروجی ها مشاهده می کنید.

```python
def print_formula(self, powers, k):
    formula = 'y = '
    for i, power in enumerate(powers):
        if power == 0:
            formula += f'{self.b:.5f}'
        else:
            formula += f'{self.w[i]:.5f}x^{{{power}}}'
        formula += ' + '
    if k < 17:
        formula += f'{self.w[-2]:.5f}x^{{{2}}}'
        formula += ' + '
        formula += f'{self.w[-1]:.5f}x^{{{17}}}'
    else:
        formula += f'{self.w[-1]:.5f}x^{{{2}}}'
    display(Math(formula))
```

اطلاعات مورد نیاز که در مورد شبکه در اولین صفحه سوال خواسته شده:

۱. تابع خطا: mse
۲. بهینه ساز: گرادیان کاهشی
۳. معماری شبکه: از یک پرسپترون استفاده کردیم.
٤. تعداد گام آموزش: ۱۰۰
٥. اندازه دسته: برای آموزش کل داده به اموزش می دادم در واقعا اندازه برابر است با کل داده بنده.
٦. آمارگان تفکیک داده: چون می خواستیم ظابطه یک تابع از طریق آموزش پرسپترون به دست آوریم همه داده ها داده آموزش در نظر گرفتم
۷. پیش پردازش اعمال شده: هر دفعه یکی از توان ها فرد را به داده اضافه می کردم.
۸. دیتاست: دیتاست ساخته شده به این صورت است داخل خود همیشه توان های ۲ و ۱۷ دارد و بعد از وارد حلقه می شود و هر یکی از درجه های فرد را به چند جمله ای اضافه می کند. دیتاست از - ۱ تا ۱ با طول گام های ۰.۰۰۰۱ ساخته شده است.
۹. نرخ یادگیری: ۱

```python
# Define the range and step size for x
x_range = np.arange(-1, 1, 0.0001)   x_range: ndarray (20000,)
target = np.array(np.sin(x_range) + (3*(x_range**17)) - (5 * (x_range**2)))   targe

# Generate the dataset and train the perceptron for each iteration
predictions = []   predictions: list (10)
weights = []   weights: list (0)

for i in range(1, 20, 2):
    print(f"Number of Sentence = {i}")   i: 19

    powers = np.arange(1, i + 1, 2)   powers: ndarray (10,)    i: 19
    dataset = np.column_stack([x_range ** power for power in powers])   dataset: nd
    dataset = np.column_stack((dataset, x_range**2))   dataset: ndarray (20000, 11)
    if i < 17:   i: 19
        dataset = np.column_stack((dataset, x_range**17))   dataset: ndarray (20000

    perceptron = Perceptron(alpha=1, epochs=100)   perceptron: <__main__.Perceptron

    # Train the Perceptron
    perceptron.train(dataset, target)   perceptron: <__main__.Perceptron object at
    # Print the formula of the function
    perceptron.print_formula(powers, i)   perceptron: <__main__.Perceptron object a

    # Plot the learning curve
    perceptron.plot_learning_curve(i)   perceptron: <__main__.Perceptron object at

    # Store the predictions
    predictions.append(perceptron.predict(dataset))   predictions: list (10)    per
    perceptron.plot_functions(x_range, target, f"powers up to {i}", dataset)   perc
```

```python
plt.figure(figsize=(16, 6))
plt.plot(x_range, target, label='Target Function', color='red')   x_range: ndarray (20000,)
for i, prediction in enumerate(predictions):   predictions: list (10)
    plt.plot(x_range, prediction, label=f'Prediction for {i*2+1} Sentences', linestyle='--')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Taylor Series Predicted Functions for Different Numbers of Sentences')
plt.legend()
plt.show()
```

تقریب سری تیلور به دست آمده بعد از آموزش پرسپترون با استفاده از یک جمله تقریب:

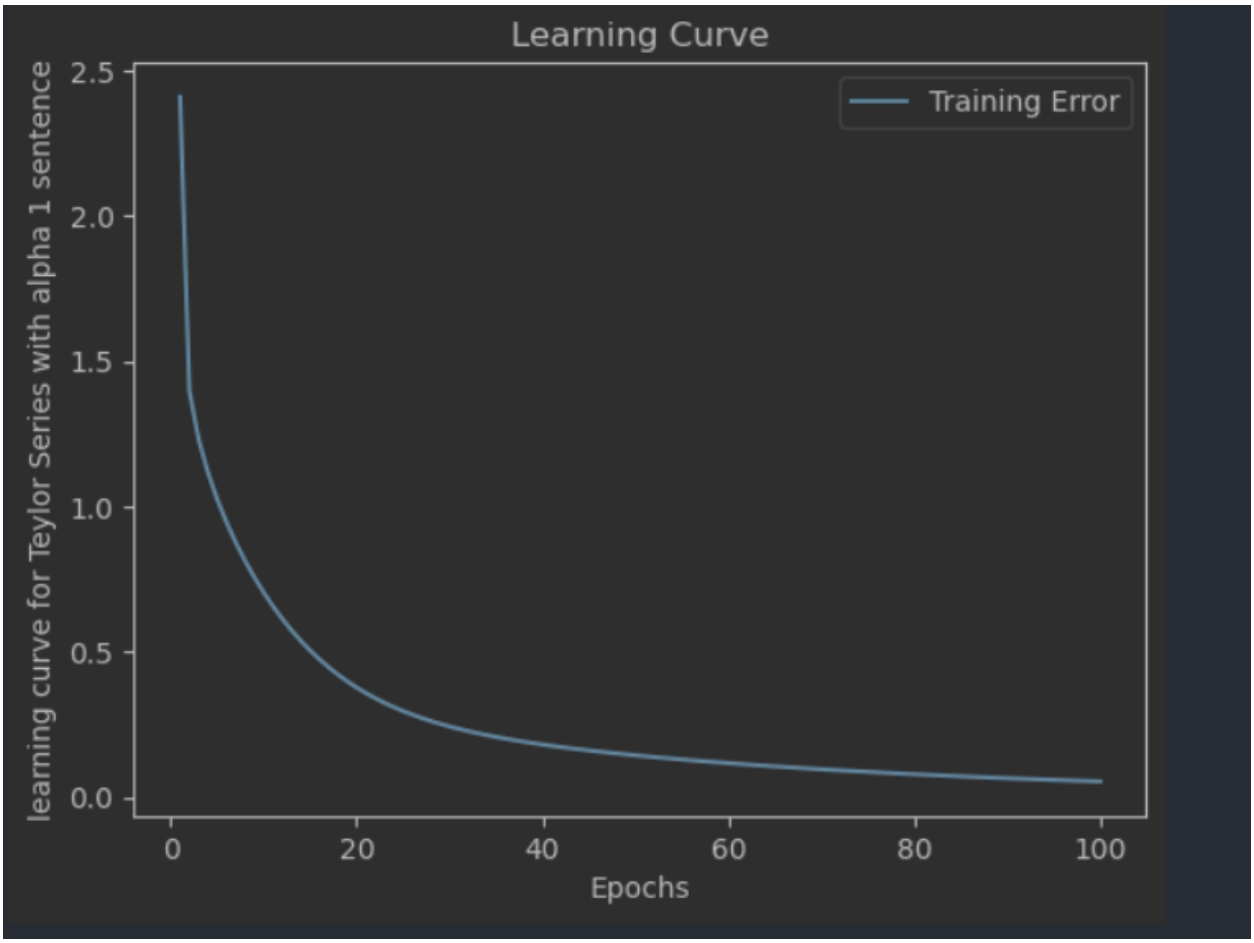
فرمول به دست آمده:

$$y = 0.98348x^{1} + -4.99902x^{2} + 2.51617x^{17}$$

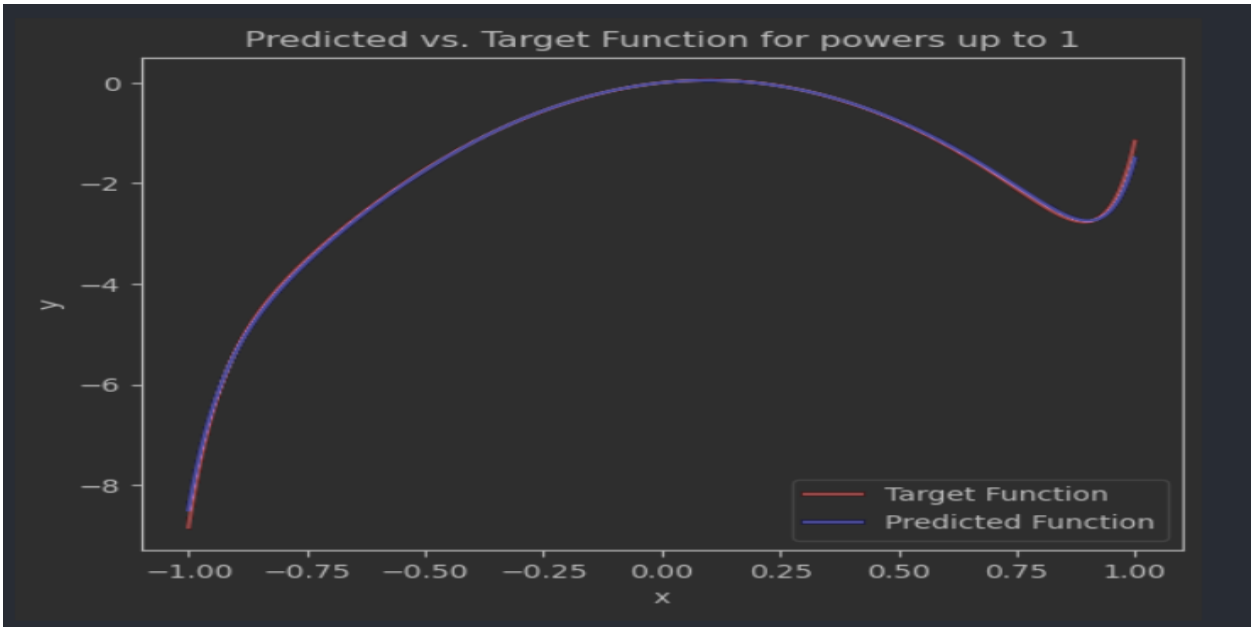
مشاهده فرآیند آموزش:

```
Number of Sentence = 1
rmse error in epoch 0 is 2.408741782869337
rmse error in epoch 10 is 0.6569528517477498
rmse error in epoch 20 is 0.358531030735672 44
rmse error in epoch 30 is 0.23559774547157683
rmse error in epoch 40 is 0.17728686984856007
rmse error in epoch 50 is 0.14159470352016215
rmse error in epoch 60 is 0.11543359895986911
rmse error in epoch 70 is 0.09479983184518515
rmse error in epoch 80 is 0.07816028751281248
rmse error in epoch 90 is 0.0646823220925977
```

منحنی خطا:



نمودار به دست آمده با استفاده از تقریب و مقایسه آن با نمودار اصلی:



تقریب سری تیلور به دست آمده بعد از آموزش پرسپترون با استفاده از دو جمله تقریب:

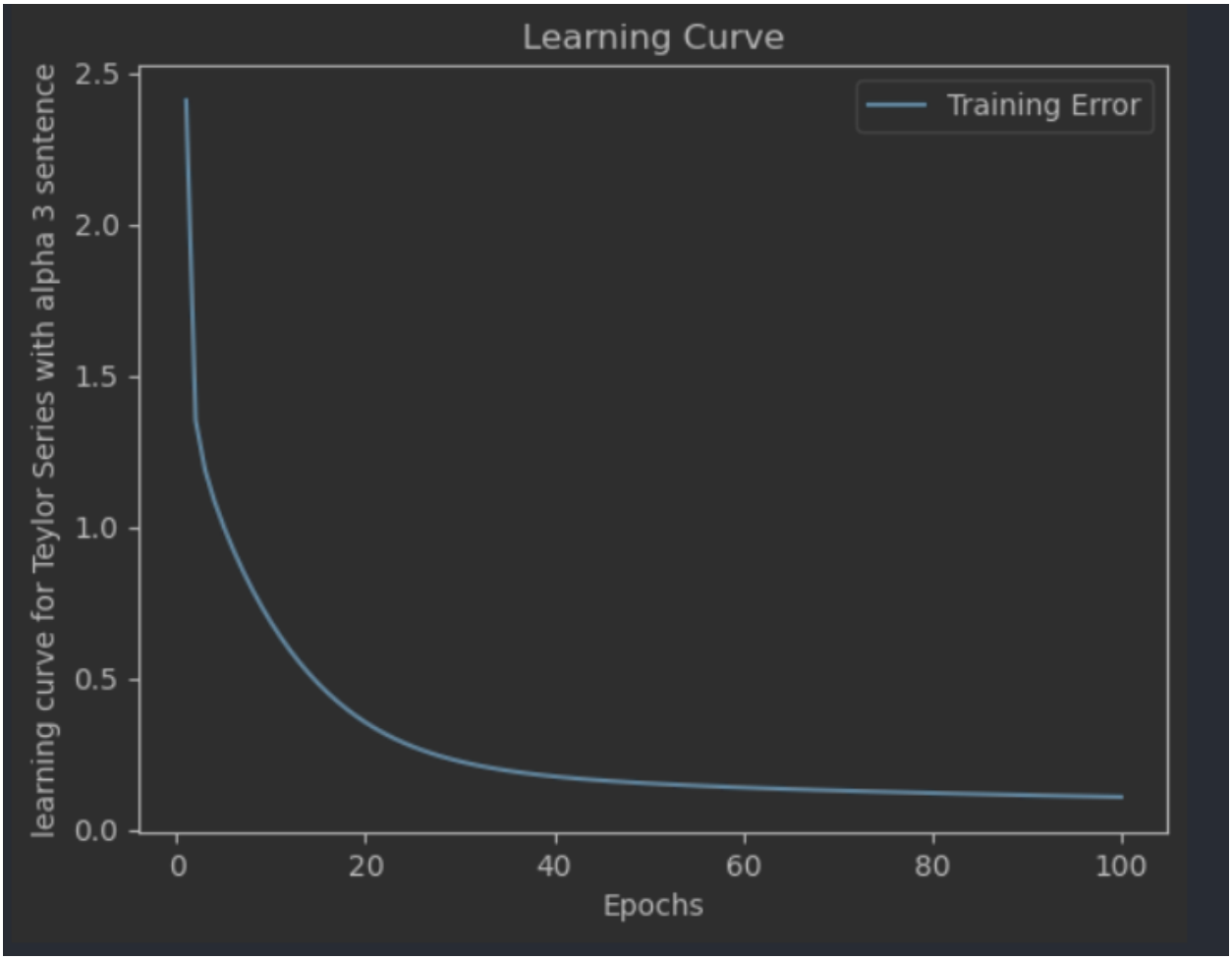
فرمول به دست آمده:

$$y = 0.59790x^{1} + 0.78679x^{3} + -4.99908x^{2} + 1.92524x^{17}$$

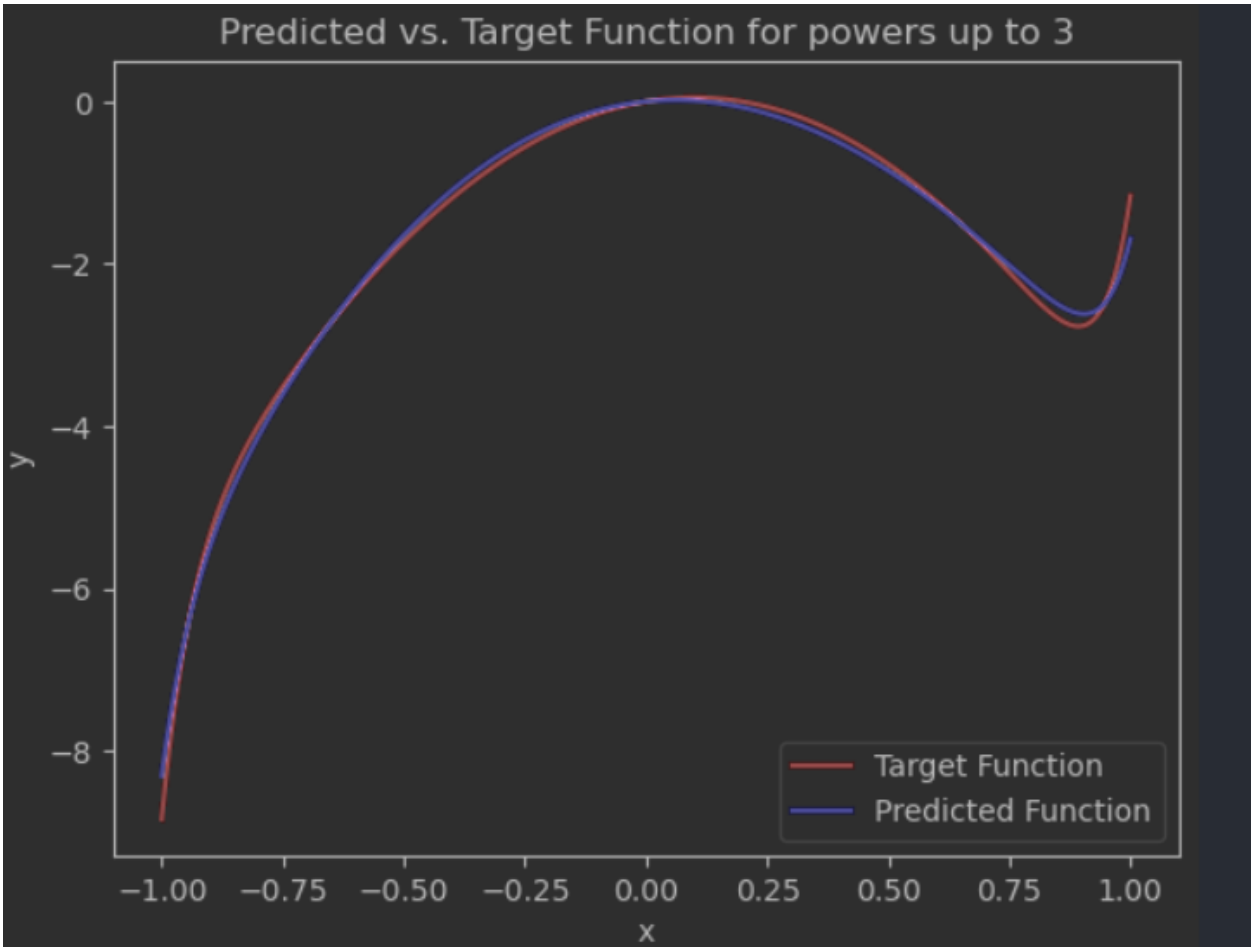
مشاهده فرآیند آموزش:

```
rmse error in epoch 0 is 2.408741782869337
rmse error in epoch 10 is 0.6353589619885601
rmse error in epoch 20 is 0.3339200033549791
rmse error in epoch 30 is 0.21712946257113 82
rmse error in epoch 40 is 0.1717806414996187
rmse error in epoch 50 is 0.150316889514136 83
rmse error in epoch 60 is 0.137026353053901 93
rmse error in epoch 70 is 0.1271930587348323
rmse error in epoch 80 is 0.119198274402693 82
rmse error in epoch 90 is 0.112335449405962 12
```

منحنی خطا:



نمودار به دست آمده با استفاده از تقریب و مقایسه آن با نمودار اصلی:



تقریب سری تیلور به دست آمده بعد از آموزش پرسپترون با استفاده از سه جمله تقریب:

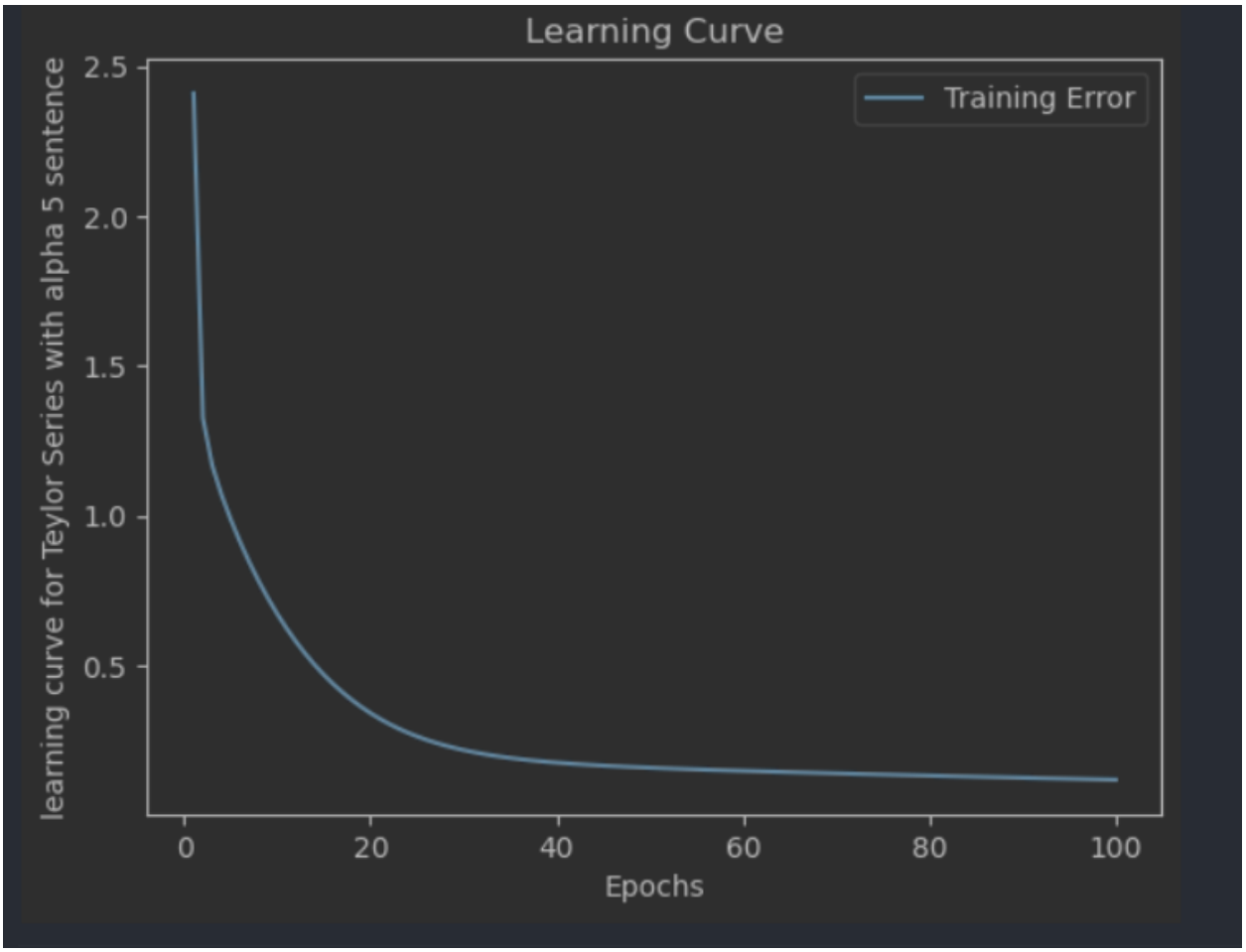
فرمول به دست آمده:

$$y = 0.64568x^{1} + 0.30148x^{3} + 0.68613x^{5} + -4.99911x^{2} + 1.58717x^{17}$$

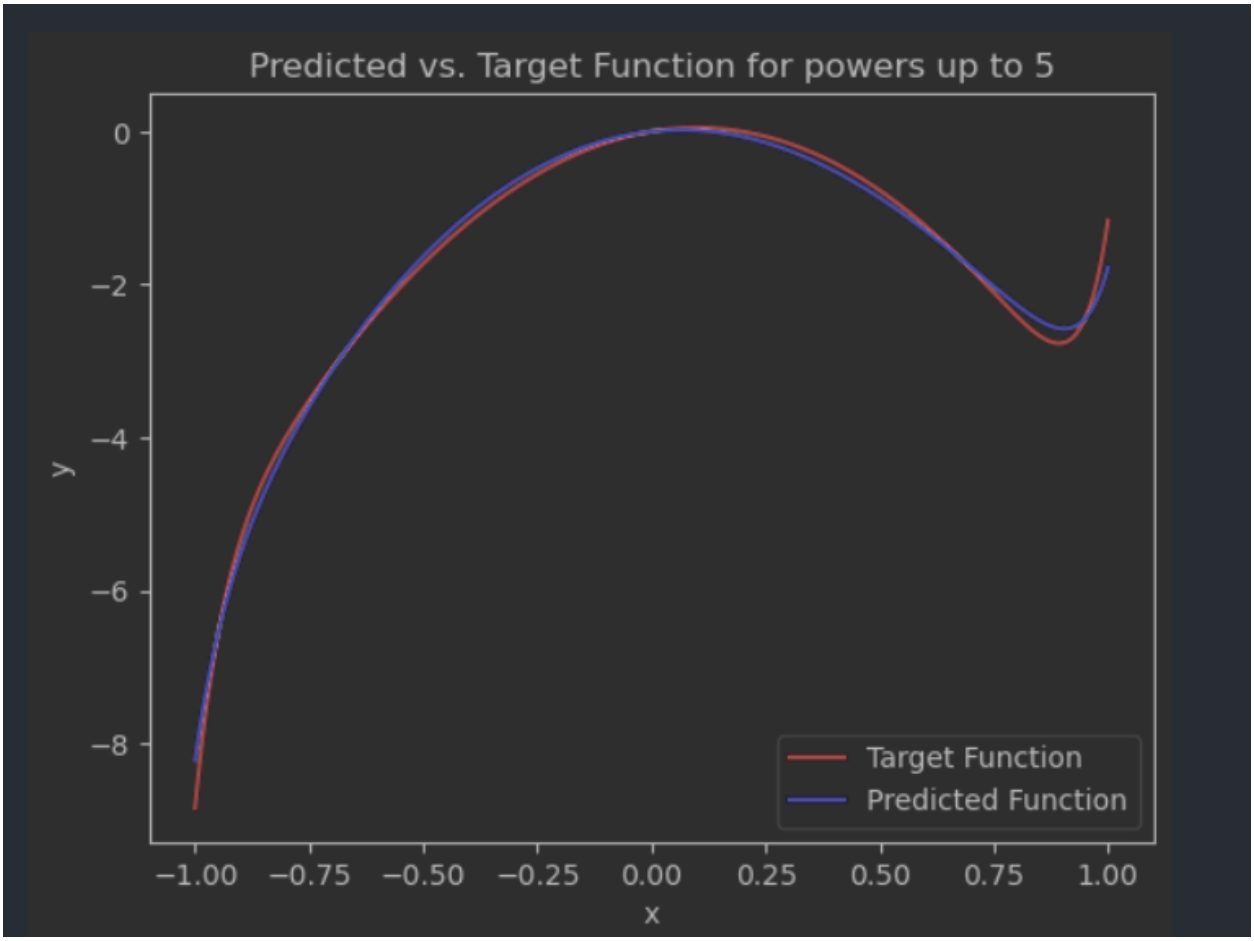
مشاهده فرآیند آموزش:

```
rmse error in epoch 0 is 2.408741782869337
rmse error in epoch 10 is 0.6229776518561342
rmse error in epoch 20 is 0.3220138487316318
rmse error in epoch 30 is 0.21119120570247102
rmse error in epoch 40 is 0.17346215655243533
rmse error in epoch 50 is 0.15757120795412527
rmse error in epoch 60 is 0.14748613527275478
rmse error in epoch 70 is 0.13917816469132474
rmse error in epoch 80 is 0.13165323389812283
rmse error in epoch 90 is 0.12463528421437274
```

منحنی خطا:



نمودار به دست آمده با استفاده از تقریب و مقایسه آن با نمودار اصلی:



تقریب سری تیلور به دست آمده بعد از آموزش پرسپترون با استفاده از چهار جمله تقریب:

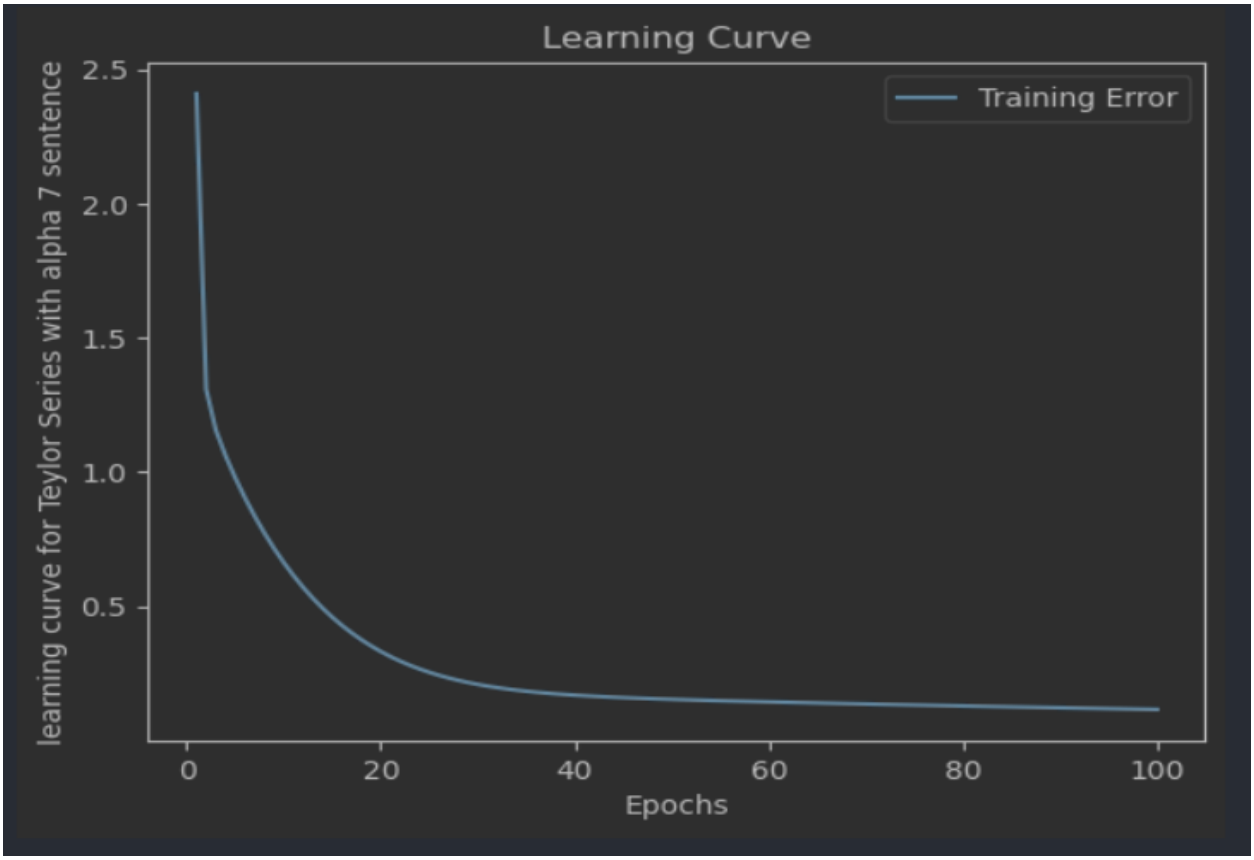
فرمول به دست آمده:

$$y = 0.70993x^{1} + 0.09265x^{3} + 0.38295x^{5} + 0.69599x^{7} + -4.99912x^{2} + 1.32364x^{17}$$

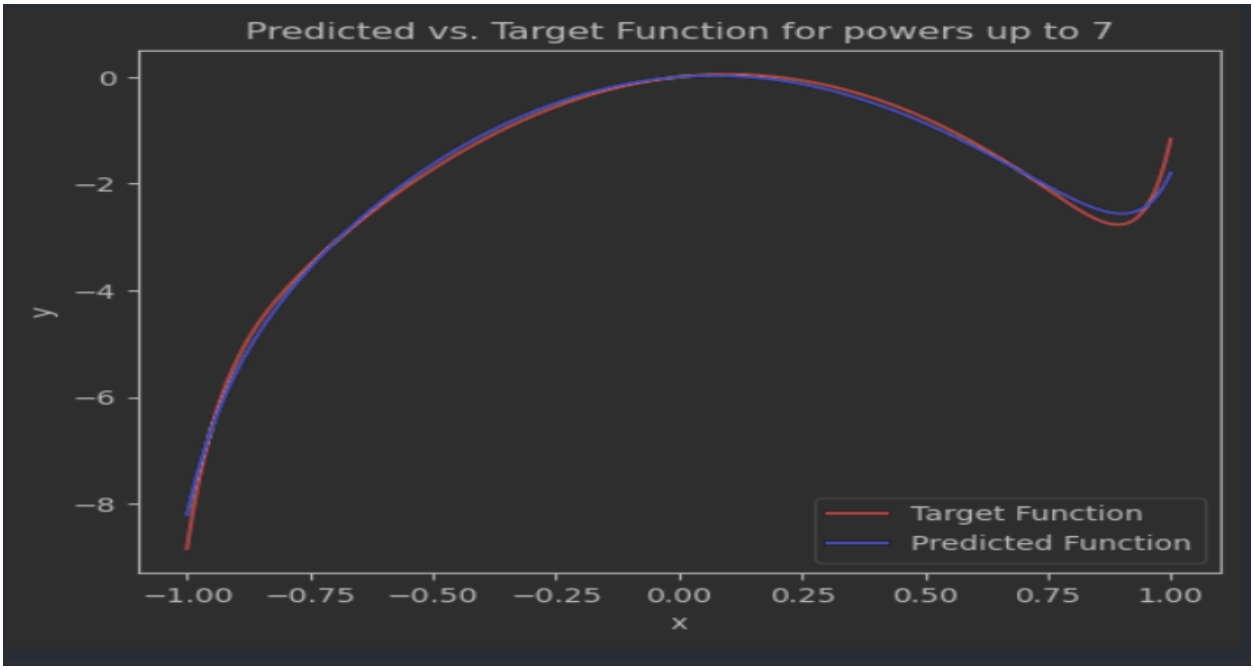
مشاهده فرآیند آموزش:

```
rmse error in epoch 0 is 2.408741782869337
rmse error in epoch 10 is 0.6147268632275317
rmse error in epoch 20 is 0.31294165913910704
rmse error in epoch 30 is 0.2031288580518074
rmse error in epoch 40 is 0.16725035829803814
rmse error in epoch 50 is 0.15275975605636616
rmse error in epoch 60 is 0.14352540335786 98
rmse error in epoch 70 is 0.135748559646074
rmse error in epoch 80 is 0.1285980866155242
rmse error in epoch 90 is 0.12188150265945617
```

منحنی خطا:



نمودار به دست آمده با استفاده از تقریب و مقایسه آن با نمودار اصلی:



تقریب سری تیلور به دست آمده بعد از آموزش پرسپترون با استفاده از پنج جمله تقریب:

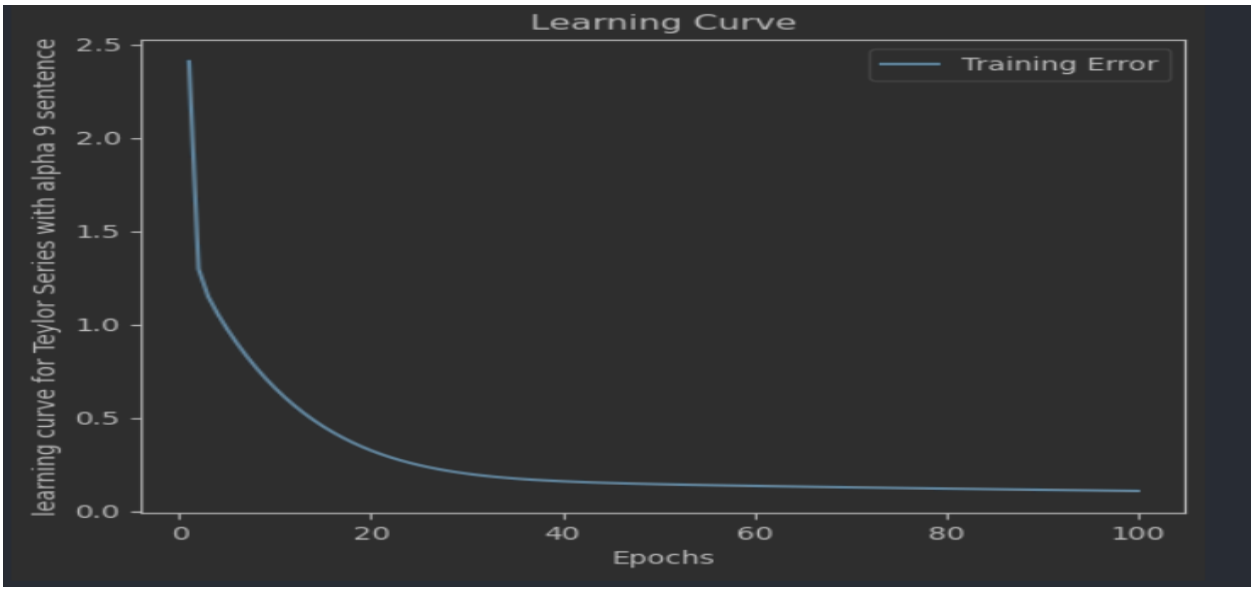
فرمول به دست آمده:

$$y = 0.76329x^{1} + -0.00428x^{3} + 0.20241x^{5} + 0.47646x^{7} + 0.69759x^{9} + -4.99910x^{2} + 1.10135x^{17}$$

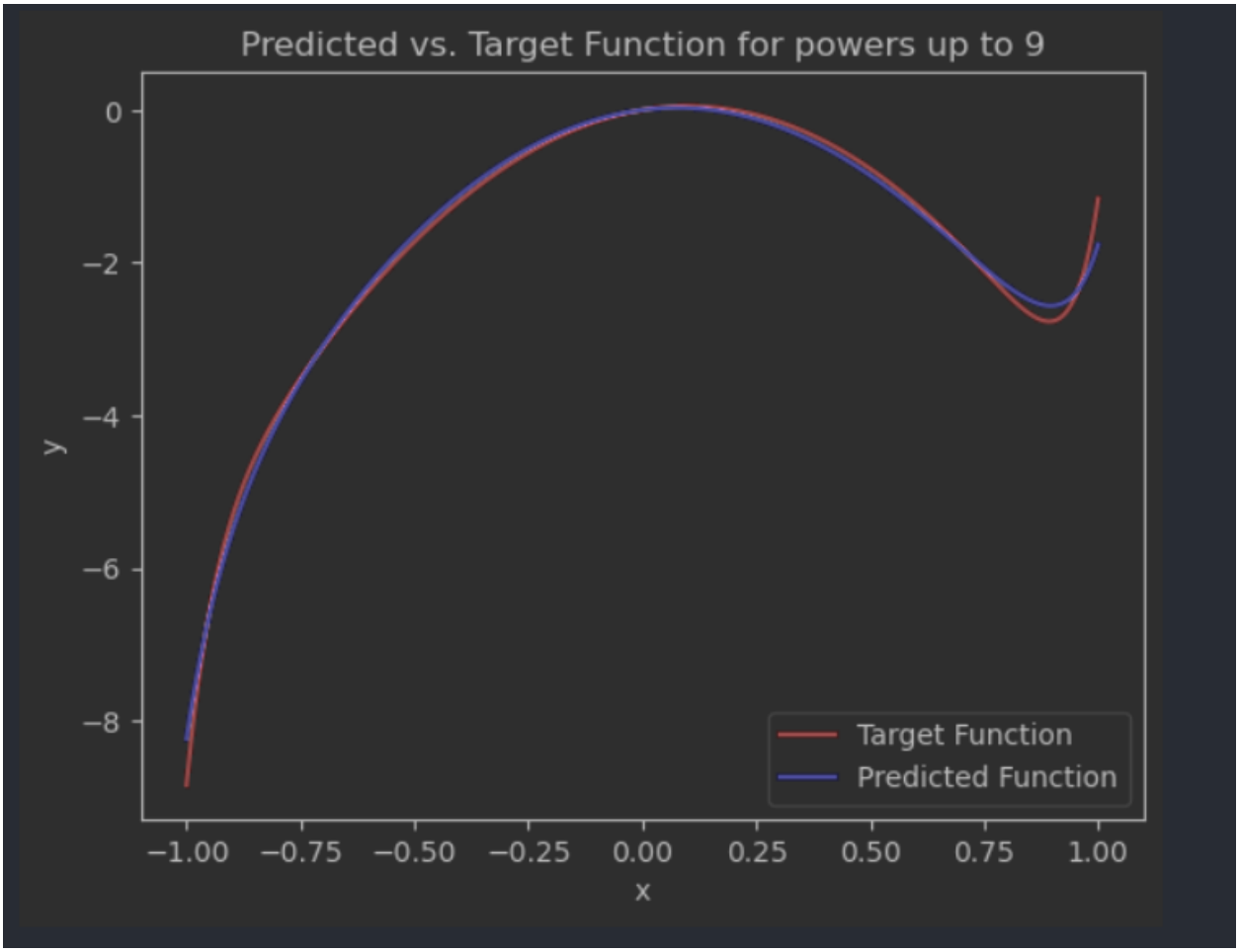
مشاهده فرآیند آموزش:

```
rmse error in epoch 0 is 2.408741782869337
rmse error in epoch 10 is 0.6086690113568569
rmse error in epoch 20 is 0.3048319466314165
rmse error in epoch 30 is 0.193187625475573 25
rmse error in epoch 40 is 0.1566997785537364
rmse error in epoch 50 is 0.14226835993580766
rmse error in epoch 60 is 0.1332514745548576
rmse error in epoch 70 is 0.12572926198575626
rmse error in epoch 80 is 0.11885648687209506
rmse error in epoch 90 is 0.11244082002678914
```

منحنی خطا:



نمودار به دست آمده با استفاده از تقریب و مقایسه آن با نمودار اصلی:



تقریب سری تیلور به دست آمده بعد از آموزش پرسپترون با استفاده از شش جمله تقریب:

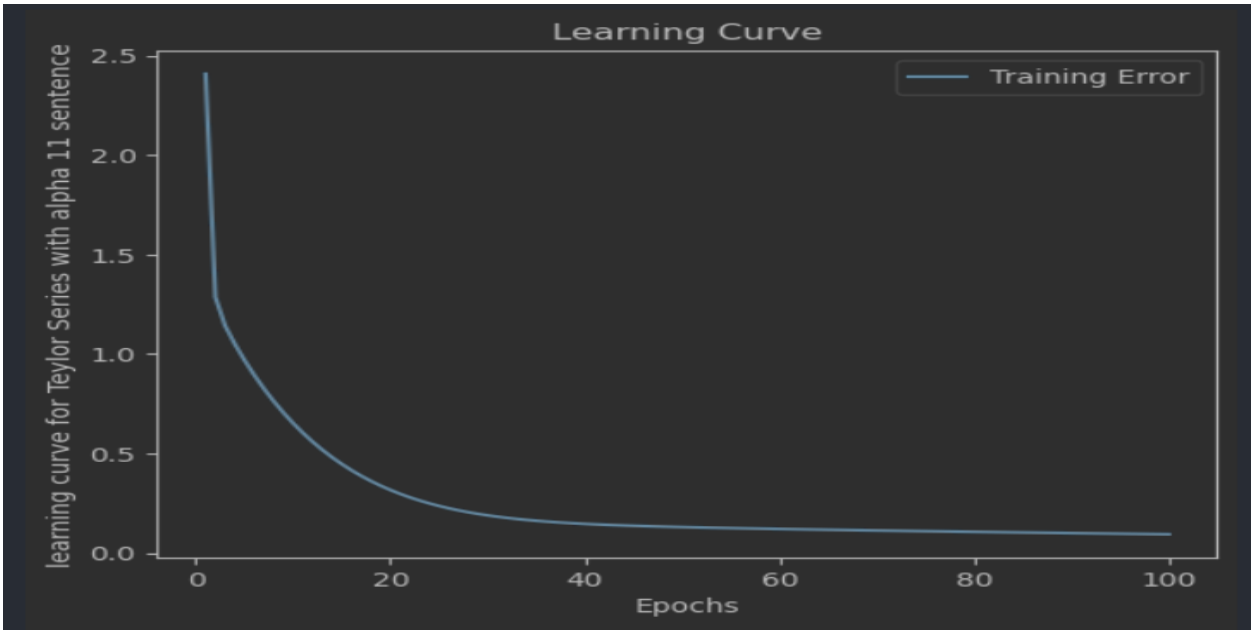
فرمول به دست آمده:

$$y = 0.80419x^{1} + -0.04427x^{3} + 0.09362x^{5} + 0.32685x^{7} + 0.52558x^{9} + 0.67586x^{11} + -4.99908x^{2} + 0.91470x^{17}$$

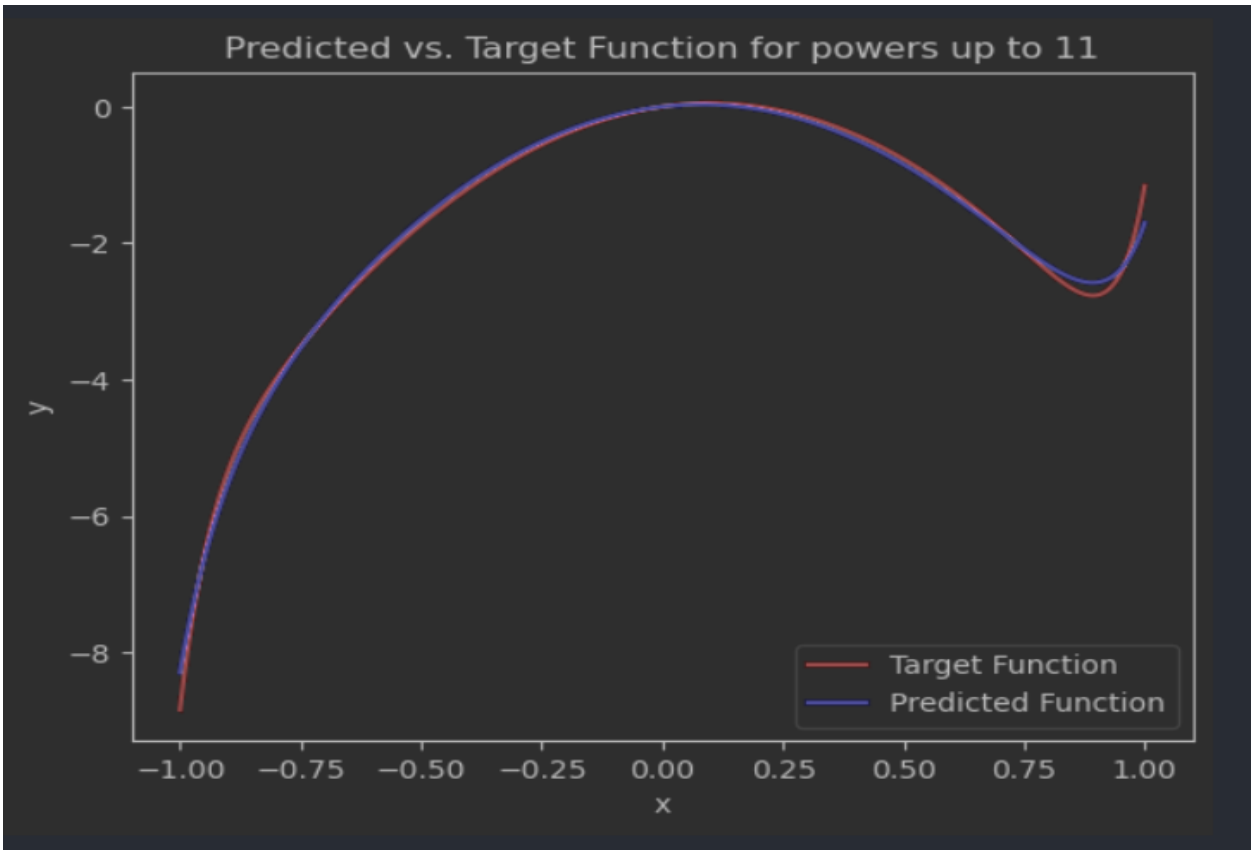
مشاهده فرآیند آموزش:

```
rmse error in epoch 0 is 2.408741782869337
rmse error in epoch 10 is 0.6039952464020457
rmse error in epoch 20 is 0.29754277188348965
rmse error in epoch 30 is 0.1827762307648987
rmse error in epoch 40 is 0.14454315903840428
rmse error in epoch 50 is 0.12954223130632084
rmse error in epoch 60 is 0.1203909483515619
rmse error in epoch 70 is 0.112891942976315 48
rmse error in epoch 80 is 0.10612690456987603
rmse error in epoch 90 is 0.09988213640816032
```

منحنی خطا:



نمودار به دست آمده با استفاده از تقریب و مقایسه آن با نمودار اصلی:



تقریب سری تیلور به دست آمده بعد از آموزش پرسپترون با استفاده از هفت جمله تقریب:

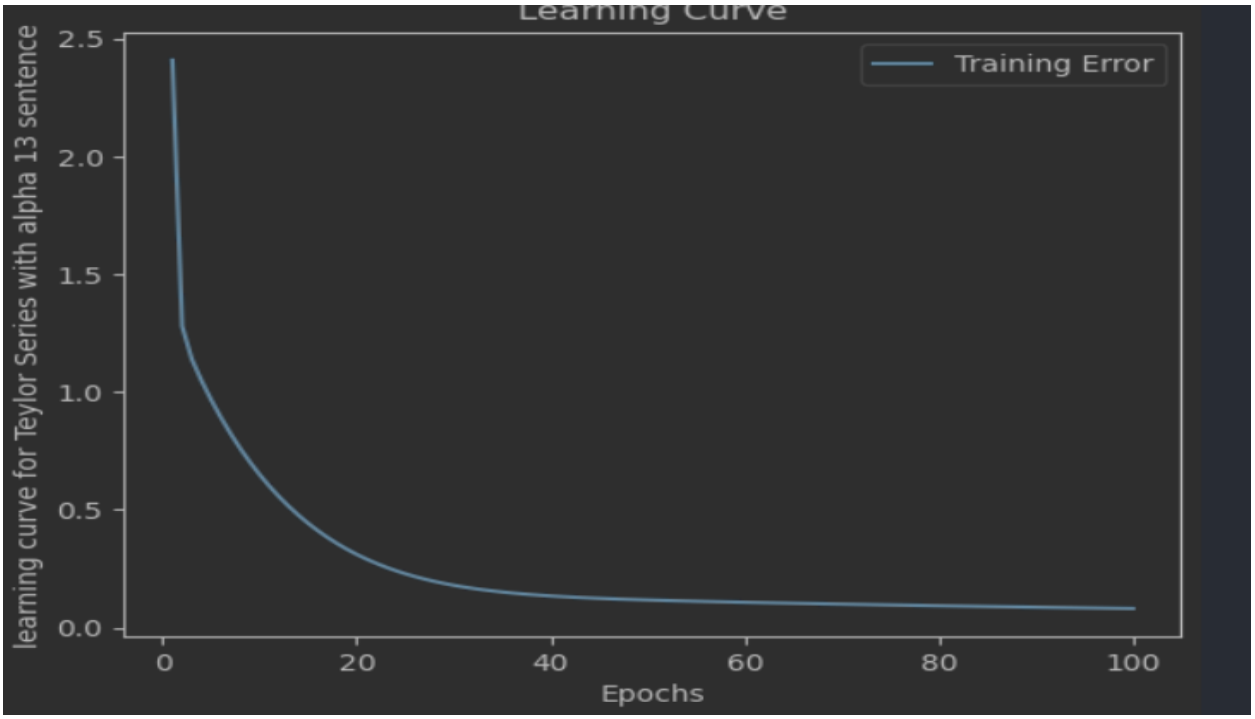
فرمول به دست آمده:

$$y = 0.83394x^{1} + -0.05349x^{3} + 0.03030x^{5} + 0.22584x^{7} + 0.40063x^{9} + 0.53634x^{11} + 0.63682x^{13} + -4.99905x^{2} + 0.76072x^{17}$$

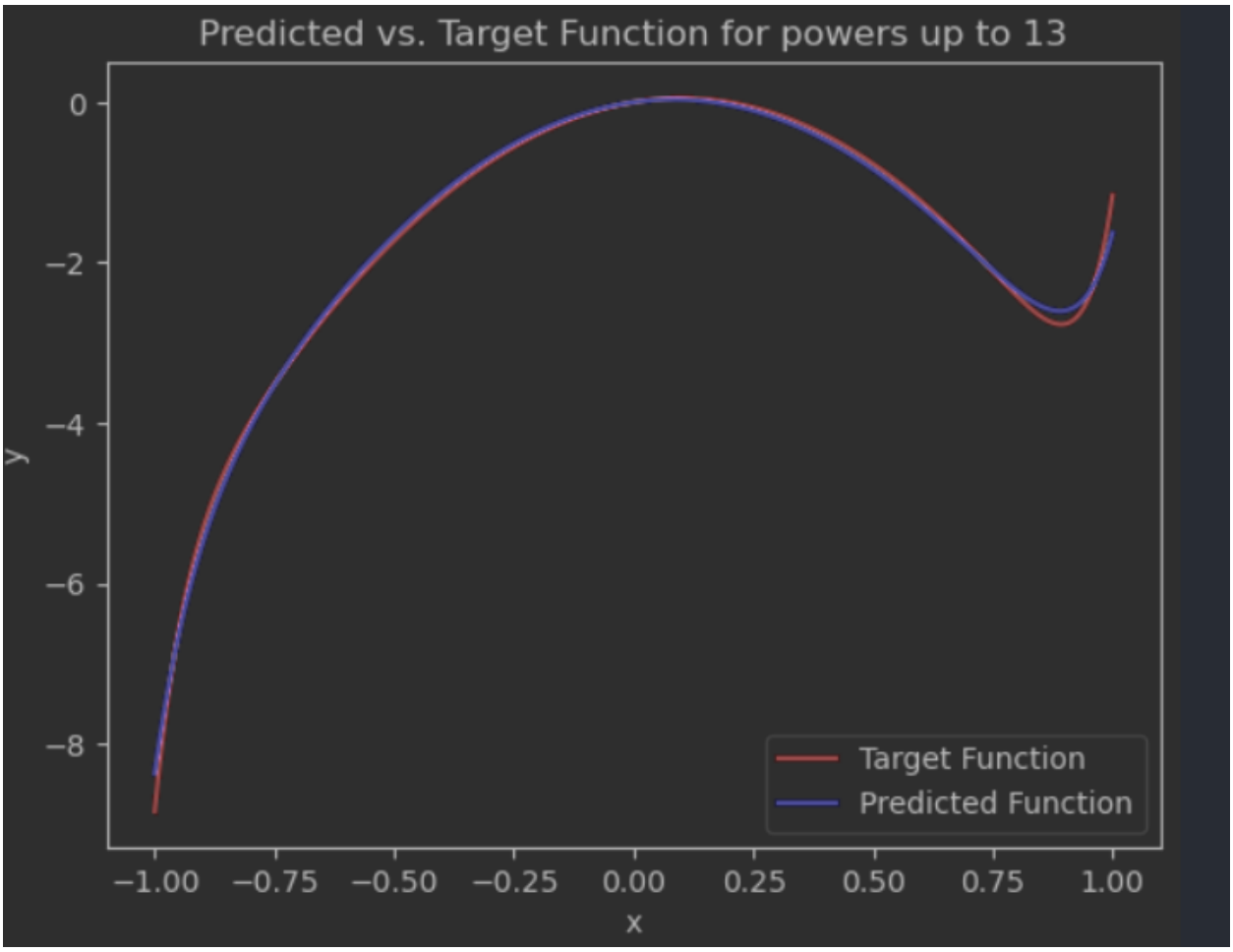
مشاهده فرآیند آموزش:

```
rmse error in epoch 0 is 2.408741782869337
rmse error in epoch 10 is 0.6002986568614261
rmse error in epoch 20 is 0.2911430472793322
rmse error in epoch 30 is 0.17281843995271182
rmse error in epoch 40 is 0.13229980246143222
rmse error in epoch 50 is 0.11639347461885606
rmse error in epoch 60 is 0.10690838597625058
rmse error in epoch 70 is 0.09929096412725484
rmse error in epoch 80 is 0.09252056463852457
rmse error in epoch 90 is 0.08635178052427006
```

منحنی خطا:



نمودار به دست آمده با استفاده از تقریب و مقایسه آن با نمودار اصلی:



تقریب سری تیلور به دست آمده بعد از آموزش پرسپترون با استفاده از هشت جمله تقریب:

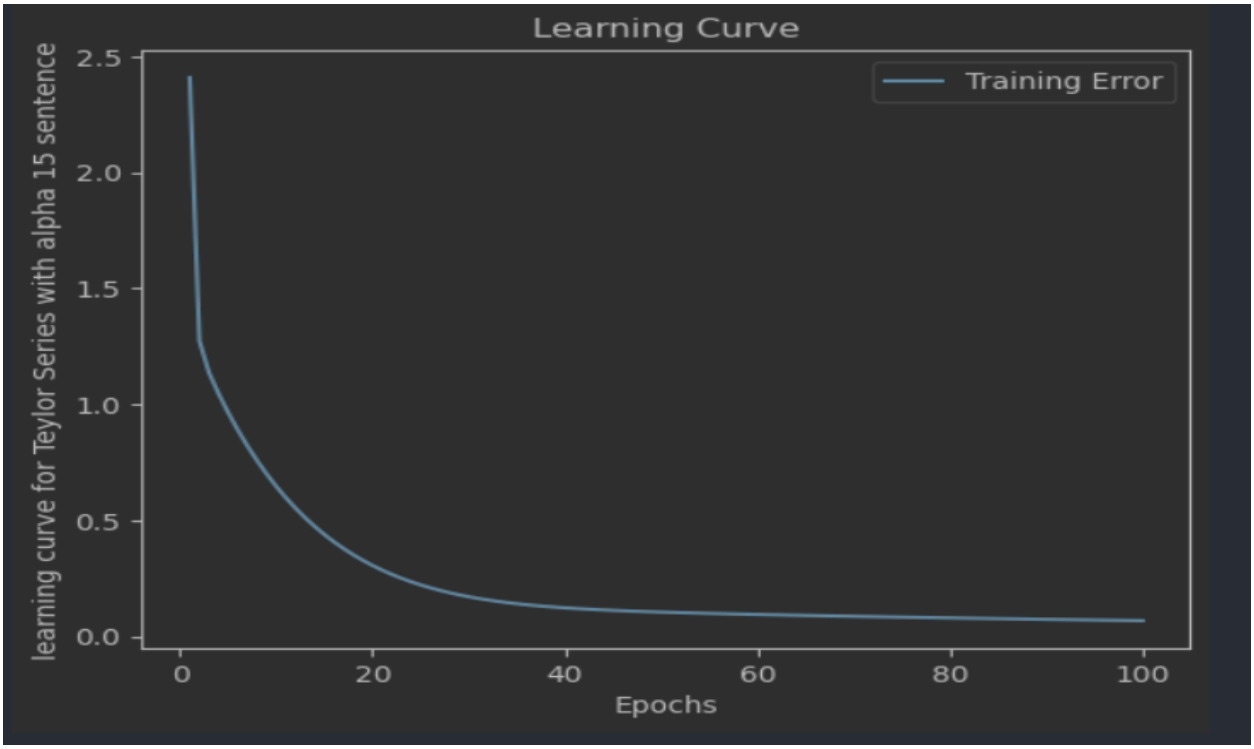
فرمول به دست آمده:

$$y = 0.85452x^{1} + -0.04641x^{3} + -0.00384x^{5} + 0.15903x^{7} + 0.31101x^{9} + 0.43138x^{11} + 0.52184x^{13} + 0.58811x^{15} + -4.99902x^{2} + 0.63571x^{17}$$

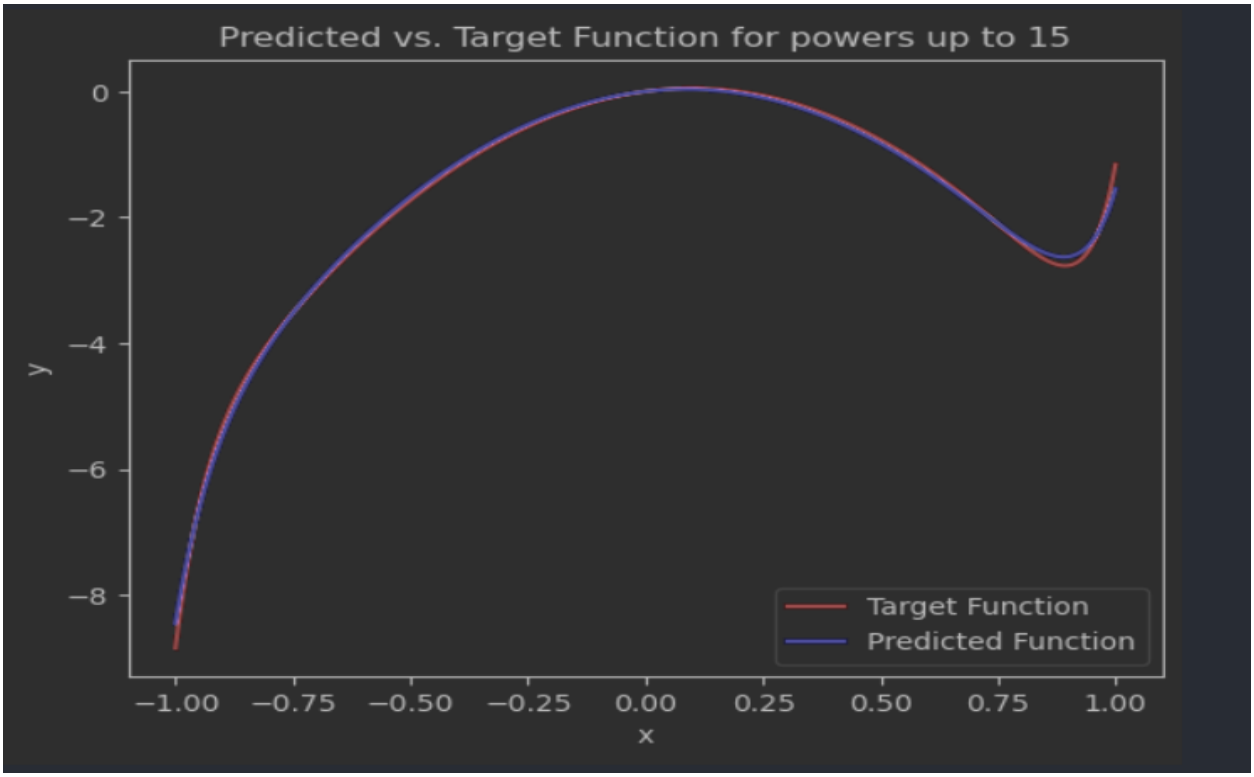
مشاهده فرآیند آموزش:

```
rmse error in epoch 0 is 2.408741782869337
rmse error in epoch 10 is 0.5973368080279624
rmse error in epoch 20 is 0.28565317109790633
rmse error in epoch 30 is 0.16380346862831477
rmse error in epoch 40 is 0.12079414149627014
rmse error in epoch 50 is 0.103821004736261
rmse error in epoch 60 is 0.09391726706806534
rmse error in epoch 70 is 0.08612838386415322
rmse error in epoch 80 is 0.07931062537955767
rmse error in epoch 90 is 0.0731791151074995
```

منحنی خطا:



نمودار به دست آمده با استفاده از تقریب و مقایسه آن با نمودار اصلی:



تقریب سری تیلور به دست آمده بعد از آموزش پرسپترون با استفاده از نه جمله تقریب:

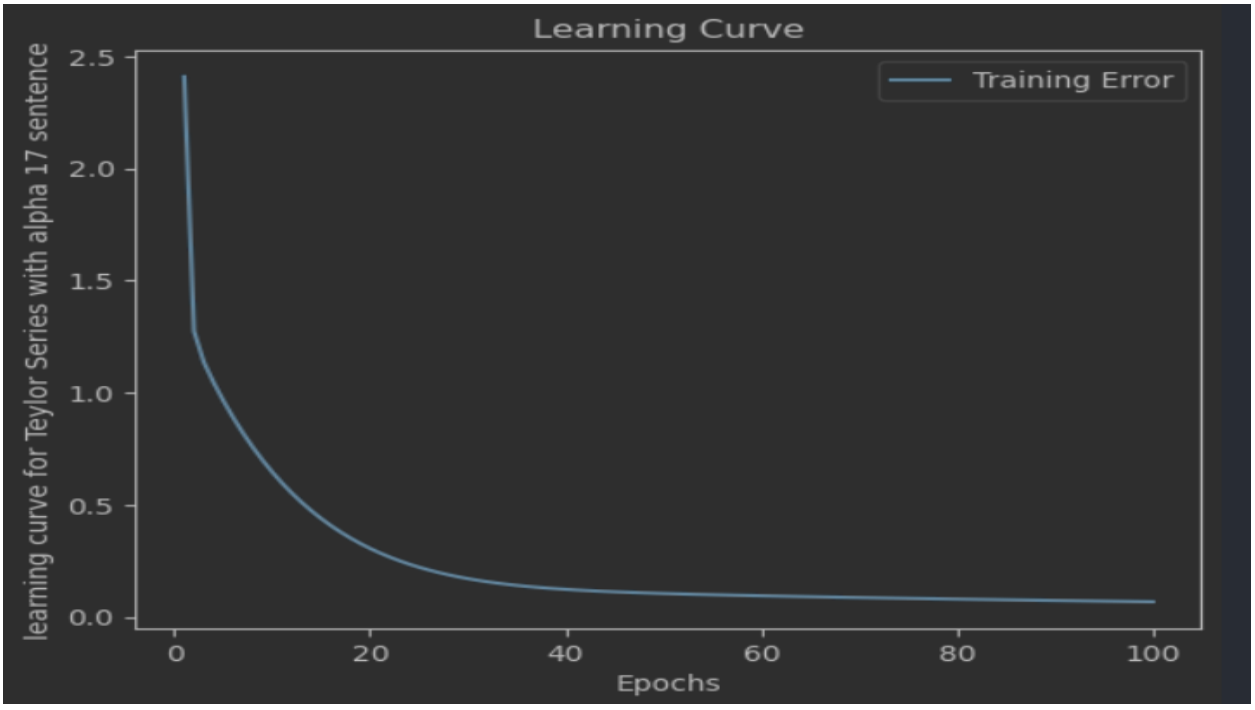
فرمول به دست آمده:

$$y = 0.85452x^{1} + -0.04641x^{3} + -0.00384x^{5} + 0.15903x^{7} + 0.31101x^{9} + 0.43138x^{11} + 0.52184x^{13} + 0.58811x^{15} + 0.63571x^{17} + -4.99902x^{2}$$

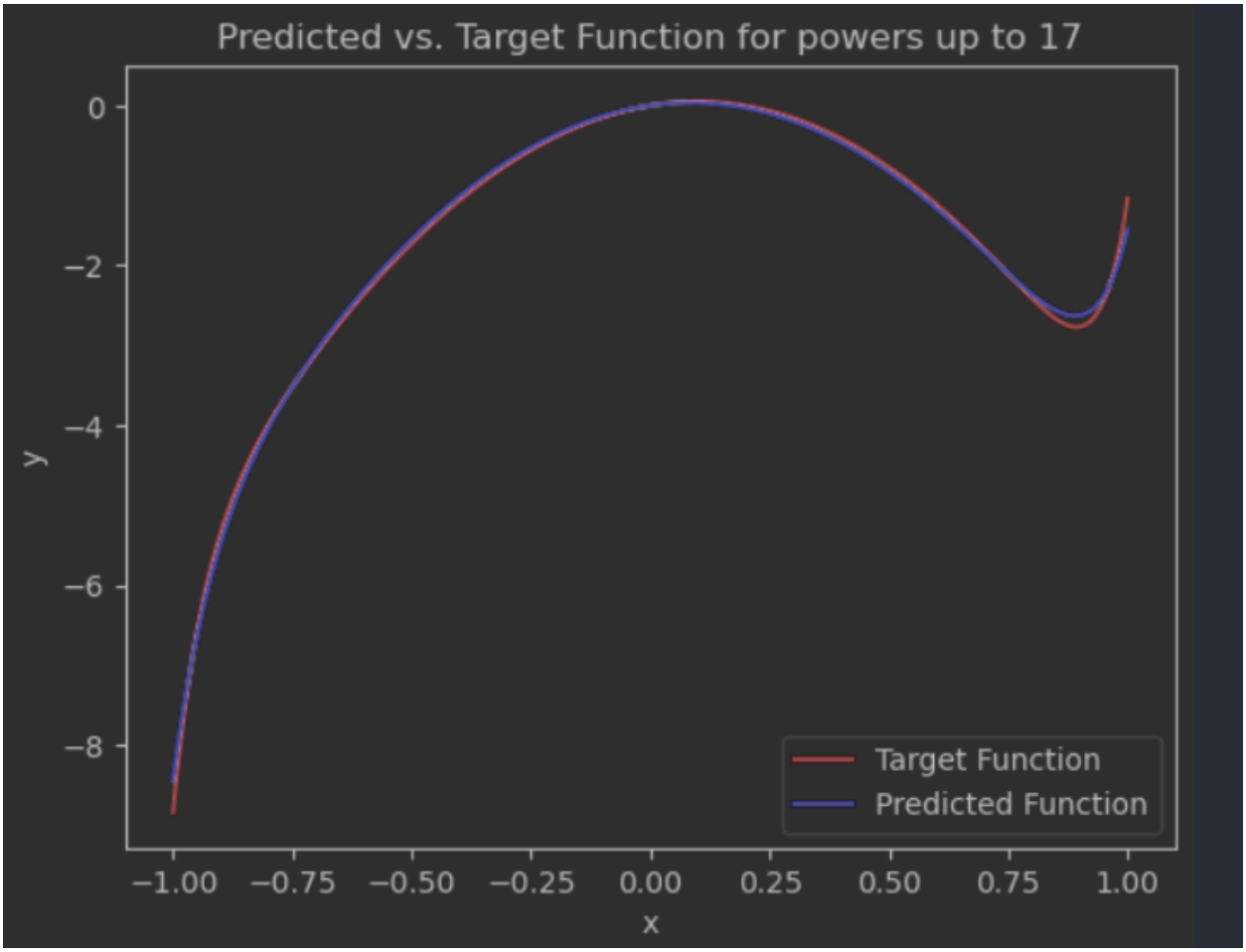
مشاهده فرآیند آموزش:

```
rmse error in epoch 0 is 2.408741782869337
rmse error in epoch 10 is 0.5973368080279624
rmse error in epoch 20 is 0.28565317109790633
rmse error in epoch 30 is 0.16380346862831477
rmse error in epoch 40 is 0.12079414149627014
rmse error in epoch 50 is 0.10382100473626102
rmse error in epoch 60 is 0.09391726706806534
rmse error in epoch 70 is 0.08612838386415324
rmse error in epoch 80 is 0.07931062537955769
rmse error in epoch 90 is 0.07317911510749951
```

منحنی خطا:



نمودار به دست آمده با استفاده از تقریب و مقایسه آن با نمودار اصلی:



تقریب سری تیلور به دست آمده بعد از آموزش پرسپترون با استفاده از ده جمله تقریب:

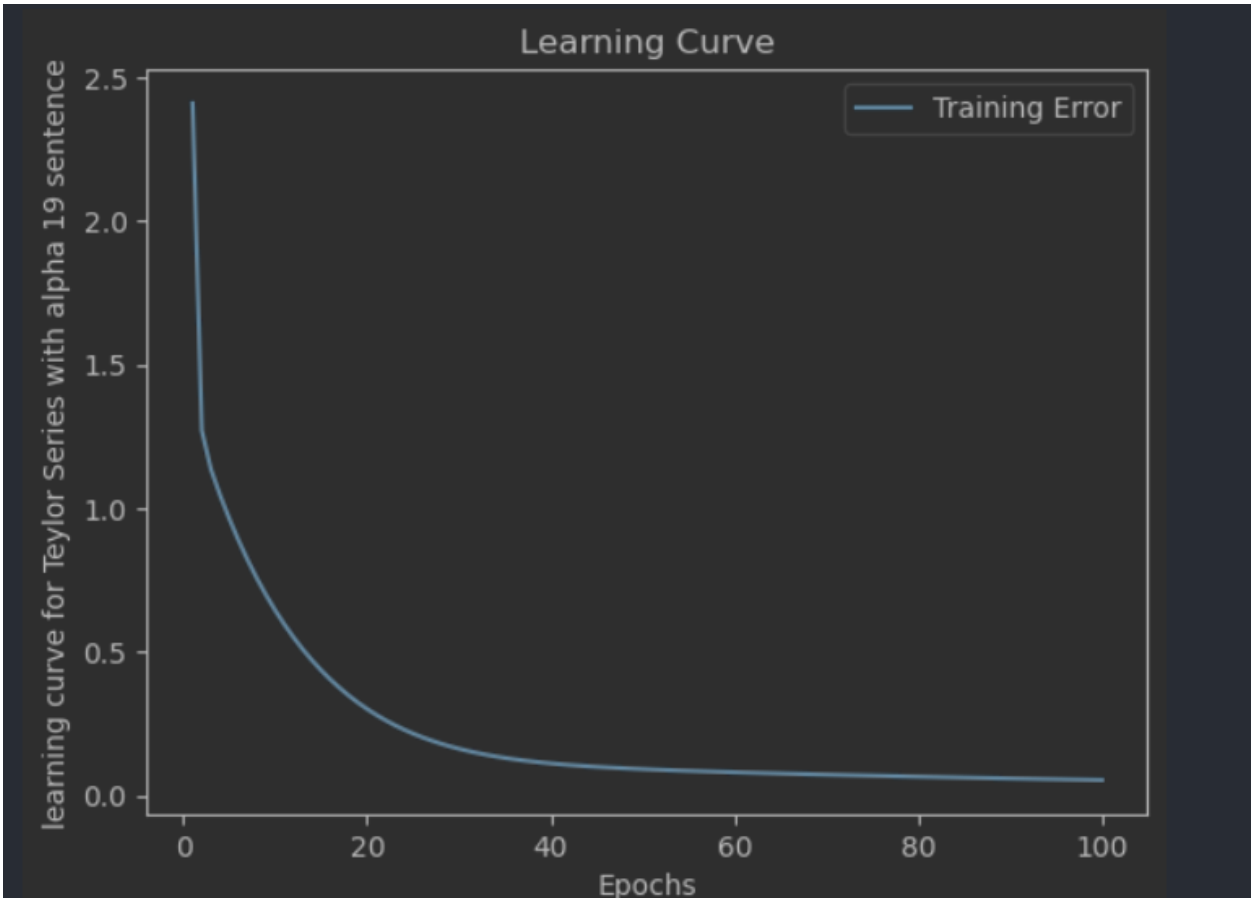
فرمول به دست آمده:

$$y = 0.86702x^{1} + -0.02503x^{3} + -0.01298x^{5} + 0.12056x^{7} + 0.24946x^{9} + 0.35262x^{11} + 0.43050x^{13} + 0.48773x^{15} + 0.52893x^{17} + 0.55794x^{19} + -4.99898x^{2}$$

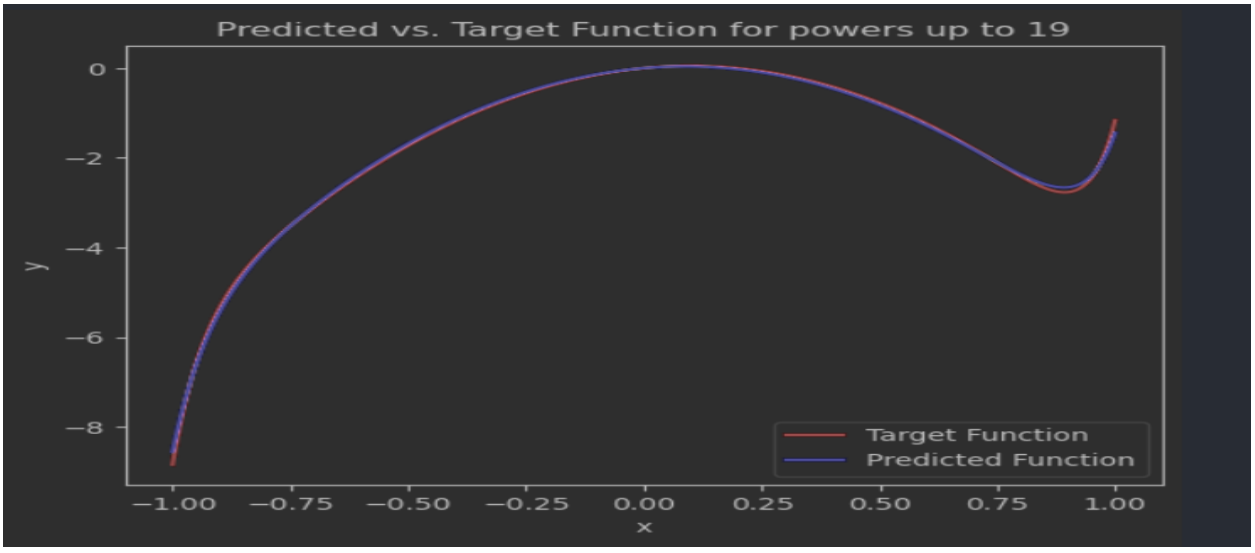
مشاهده فرآیند آموزش:

```
rmse error in epoch 0 is 2.408741782869337
rmse error in epoch 10 is 0.5947807260396835
rmse error in epoch 20 is 0.28052637023969695
rmse error in epoch 30 is 0.15491316241594782
rmse error in epoch 40 is 0.10892650808358324
rmse error in epoch 50 is 0.0905278239735584
rmse error in epoch 60 is 0.08000634588652795
rmse error in epoch 70 is 0.07191933259806661
rmse error in epoch 80 is 0.06495729249058335
rmse error in epoch 90 is 0.05877877248260785
```

منحنی خطا:



نمودار به دست آمده با استفاده از تقریب و مقایسه آن با نمودار اصلی:



در نهایت تجمیع تمام شکل های به دست امده بر روی یک نمودار: