



دانشگاه صنعتی امیرکبیر
(پلی تکنیک تهران)
دانشکده مهندسی برق

گزارش کار آزمایش دوم
آزمایشگاه مقدمه ای بر هوش محاسباتی

پیاده سازی پرسپترون

نگارش
ارشیا اسمعیل طهرانی
علی بابالو
پویا ابراهیمی

استاد راهنما
سرکار خانم موسوی

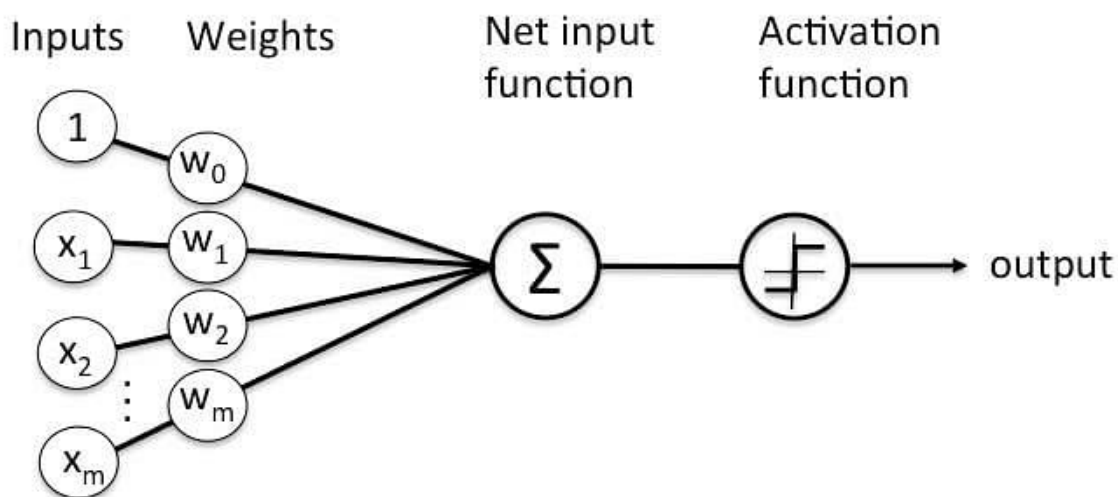
آبان ماه 1401

۱	پیش گزارش
۱	نقش perceptron در شبکه عصبی
۱	انواع perceptron
۲	مدل ساده از perceptron
۳	شباهت مدل artificial perceptron با ساختار perceptron در بدن انسان
۴	مقدمه
۴	الگوریتم یادگیری perceptron
۶	شرح آزمایش
۶	محیط پایتون
۱۹	تابع فعال ساز tanh
۲۰	تابع فعال ساز ReLU
۲۱	تمرین مربوطه
۲۱	محیط matlab
۲۸	بررسی به کار گیری الگوریتم Perceptron برای داده های زیر
۲۹	پیاده سازی گیت های And, Or, XOR
۲۹	پیاده سازی And
۳۴	پیاده سازی Or
۳۹	پیاده سازی XOR
۴۲	دیتاست Iris
۴۶	منابع و مواخذ

پیش گذارش

2-1- نقش Perceptron در شبکه عصبی

پرسپترون توسط فرانک روزنبلات در سال 1957 معرفی شد. او قانون یادگیری پرسپترون را بر اساس نورون اصلی MCP پیشنهاد کرد. پرسپترون الگوریتمی برای Supervised Learning از طبقه بندی کننده های باینری است. این الگوریتم نورون ها را قادر می سازد تا عناصر مجموعه آموزشی را در یک زمان یاد بگیرند و پردازش کنند.



شکل 1) یک مدل ساده از Perceptron

2-3-1- انواع Perceptron

تک لایه: پرسپترون تک لایه فقط می تواند الگوهای قابل جداسازی خطی را یاد بگیرد.
چند لایه: پرسپترون های چندلایه می توانند در مورد دو یا چند لایه که دارای قدرت پردازش بیشتری هستند یاد بگیرند.
الگوریتم پرسپترون وزن سیگنال های ورودی را برای ترسیم یک مرز تصمیم گیری خطی می آموزد.

توجه: Supervise Learning نوعی از یادگیری ماشینی است که برای یادگیری مدل‌ها از داده‌های آموزشی برچسب‌گذاری شده استفاده می‌شود. پیش‌بینی خروجی برای داده‌های آینده یا دیده نشده را ممکن می‌سازد. اجازه دهید در بخش بعدی روی قانون یادگیری پرسپترون تمرکز کنیم.

2-1-1. Perceptron در یادگیری ماشین

رایج ترین اصطلاح در هوش مصنوعی و یادگیری ماشین Perceptron (AIML) است. این مرحله ابتدایی یادگیری کدنویسی و فناوری‌های یادگیری عمیق است که شامل مقادیر ورودی، امتیازها، آستانه‌ها و وزن‌های پیاده‌سازی گیت‌های منطقی است. پرسپترون مرحله پرورش یک پیوند عصبی مصنوعی است. در قرن 19، آقای فرانک روزنبلات Perceptron را اختراع کرد تا محاسبات سطح بالا را برای شناسایی قابلیت های داده های ورودی یا هوش تجاری انجام دهد. با این حال، اکنون از آن برای اهداف مختلف دیگری استفاده می‌شود.

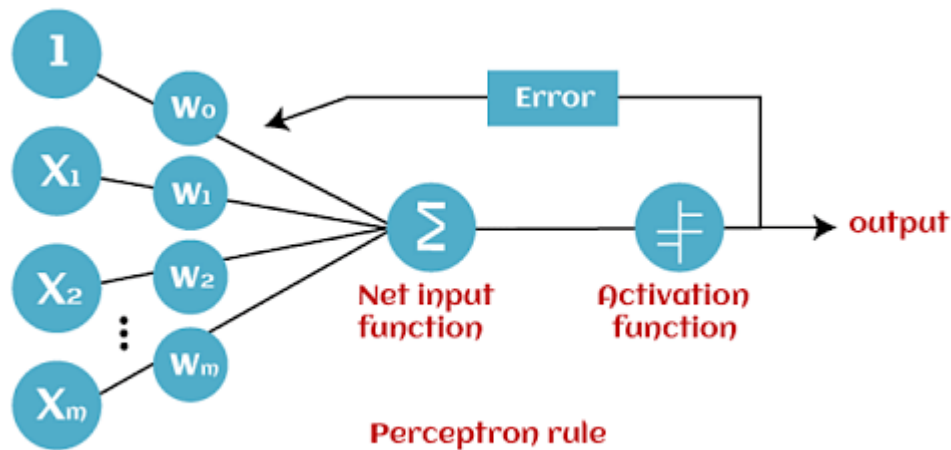
2-3-2- مدل پرسپترون در یادگیری ماشینی چیست؟

یک الگوریتم مبتنی بر ماشین که برای Supervised Learning، وظایف مرتب‌سازی باینری مختلف استفاده می‌شود Perceptron نام دارد. علاوه بر این، Perceptron همچنین نقش اساسی به عنوان یک نورون مصنوعی یا پیوند عصبی در تشخیص محاسبات داده های ورودی خاص در هوش تجاری دارد. مدل پرسپترون نیز به عنوان یکی از بهترین و خاص ترین انواع شبکه های عصبی مصنوعی طبقه بندی می‌شود. به عنوان یک الگوریتم یادگیری نظارت شده از طبقه‌بندی‌کننده‌های باینری، می‌توانیم آن را یک شبکه عصبی تک لایه با چهار پارامتر اصلی در نظر بگیریم: مقادیر ورودی، وزن‌ها و بایاس، مجموع خالص و تابع فعال‌سازی.

2-2- مدل ساده از Perceptron

پرسپترون یک پیوند عصبی تک لایه با چهار پارامتر اصلی در نظر گرفته می‌شود. مدل پرسپترون با ضرب تمام مقادیر ورودی و وزن آنها شروع می‌شود، سپس این مقادیر را برای ایجاد مجموع وزنی اضافه

می کند. علاوه بر این، این جمع وزنی برای به دست آوردن خروجی مورد نظر به تابع فعال سازی 'f' اعمال می شود. این تابع فعال سازی به عنوان تابع گام نیز شناخته می شود و با "f" نشان داده می شود.



شکل 2) یک مدل دیگر ساده از Perceptron

شباهت مدل Artificial Perceptron با ساختار perceptron در بدن انسان

Biological Neuron	Artificial Neuron
Cell Nucleus (Soma)	Node
Dendrites	Input
Synapse	Weights or interconnections
Axon	Output

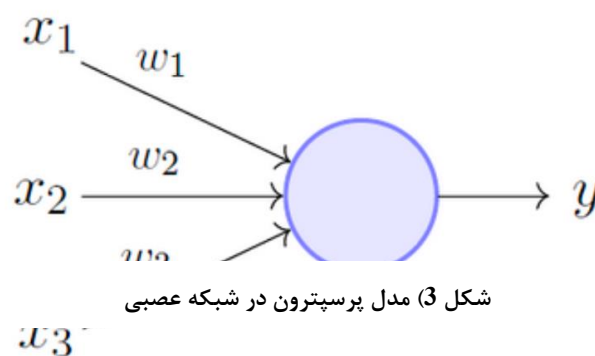
جدول 1) نورون بیولوژیکی در مقابل نورون مصنوعی

ورودی های پرسپترون مانند دندریت ها عمل می کنند و اطلاعات را دریافت می کنند. بدنه ی اصلی نورون ها مشابه تابع فعال ساز عمل می کند که پس از رسیدن به حد آستانه (Threshold) فعال شده و

سیگنال خروجی تولید می‌شود که معادل آکسون ها در نورون های بیولوژیکی می‌باشد. و سپس خروجی آکسون ها توسط سیناپس ها به نورون بعدی منتقل می‌شود.

مقدمه

در این آزمایش قصد داریم به پیاده سازی Perceptron در محیط های Python و MATLAB بپردازیم تا عمل Classification بر روی داده هایی انجام شود. مدل کلی Perceptron، به همراه رابطه خروجی آن، در ذیل آورده شده است.



خروجی پرسپترون
بر اساس ورودی های
آن در رابطه زیر توضیف شده است.

$$y = \begin{cases} 1 & \text{if } wx > 0 \\ 0 & \text{if } wx < 0 \end{cases}$$

وزن های مدل پرسپترون توسط الگوریتم یادگیری پرسپترون آپدیت می‌شوند که در ادامه به بررسی این الگوریتم پرداخته می‌شود.

2-3- الگوریتم یادگیری Perceptron

هدف این الگوریتم، پیدا کردن بهترین وزن های شبکه برای تشخیص صحیح کلاس داده های ورودی است.

ورودی شبکه:

P تا زوج دو تایی $\{(x_1, d_1) \dots (x_p, d_p)\}$ که بردار x مقادیر ورودی هر نمونه و بردار با ابعاد واحد d خروجی مورد نظر متناظر با آن بردار است.

خروجی شبکه:

مدل آموزش دیده (وزن های w مناسب) Perceptron می باشد.

مراحل:

1. مقادیر $Kmax > 0, Emax > 0, \eta > 0$ انتخاب می شوند.
 2. وزن ها به صورت مقادیر رندوم کوچک مقداردهی می شوند.
 3. شمارنده ها و متغیر خطا مقداردهی می شوند. $E = 0, k = 1, p = 1$
 4. آغاز سیکل یادگیری؛ $y = yp, d = dp, o = f(W^T y)$
 5. وزن ها بر اساس رابطه مقابل، بروز رسانی می شوند: $w = w + \frac{1}{2} \eta (d - o)(1 - o^2)y$
 6. مقدار خطا بر اساس رابطه ی روبرو محاسبه می شود. $E = E + \frac{1}{2} (d - o)^2$
 7. اگر $p < P$ باشد، آن گاه:
 8. $p = p + 1$ ، برو به مرحله 4
 9. در غیر این صورت:
 10. به مرحله 12 برو.
 11. پایان شرط.
 12. اگر $E < Emax$ باشد، آنگاه:
 13. یادگیری پایان یافته است.
 14. در غیر این صورت:
 15. مرحله بعدی یادگیری را شروع کن و به مرحله 17 برو.
 16. پایان شرط.
 17. اگر $K < Kmax$ باشد، آن گاه:
 18. $k = k + 1, E = 0, p = 1$
 19. در غیر این صورت:
 20. یادگیری پایان یافته است
 21. پایان شرط
-

شکل 4) الگوریتم یادگیری Perceptron

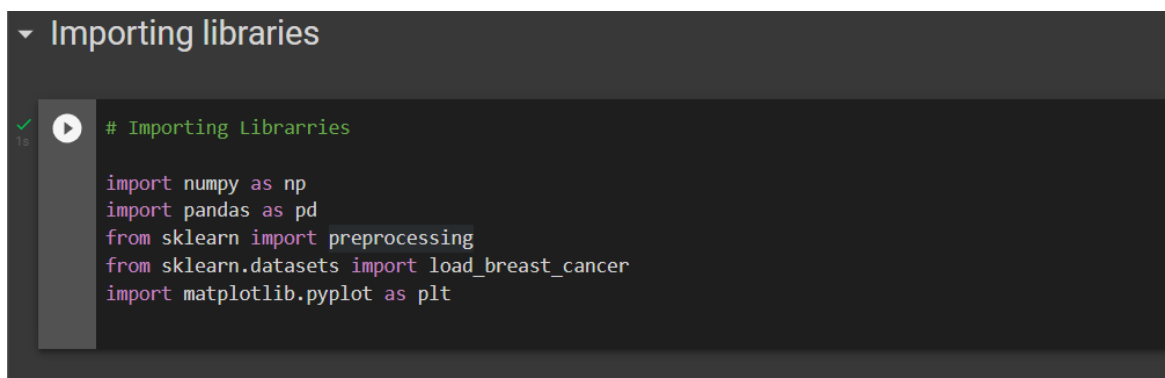
در این الگوریتم وزن های شبکه در ابتدا با استفاده از اعداد تصادفی مقدار دهی اولیه می شوند. سپس تا هنگامی که وزن ها همگرا نشده اند، وزن ها بر اساس تفاوت بین کلاس پیش بینی شده توسط شبکه، و کلاس اصلی داده ورودی آپدیت می شوند.

شرح آزمایش

در این قسمت به پیاده سازی Perceptron در محیط Python و MATLAB پرداخته می شود.

2-4- محیط Python

توضیحات ذیل مرتبط با ابتدای آزمایش تا تمرین اول می باشد. توجه شود که snippet های داخل گزارشکار مربوط به کد نویسی functional می باشد. بخش ماژولار و به صورت OOP در کد آمده است.



```
▼ Importing libraries

# Importing Libraries

import numpy as np
import pandas as pd
from sklearn import preprocessing
from sklearn.datasets import load_breast_cancer
import matplotlib.pyplot as plt
```

شکل 5) ماژول ها و کتابخانه های مربوطه را اضافه می کنیم.

توجه شود که از pandas برای dataframe استفاده می کنیم.

▼ Loading X, Y from load_breast_cancer() class

```
# Using data

data = load_breast_cancer()
X = data.data
Y = data.target

Feature = data.feature_names
df = pd.DataFrame(data=X, columns=Feature)
df['Target'] = Y

df
```

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.14710	0.2419	0.07871
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07017	0.1812	0.05667
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12790	0.2069	0.05999
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.10520	0.2597	0.09744
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430	0.1809	0.05883
...
564	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	0.1726	0.05623
565	20.12	20.25	131.00	1281.0	0.09730	0.13010	0.14120	0.09730	0.1750	0.05560

شکل 6) ورودی ها و خروجی ها از دیتاست استخراج می شوند.

همانطور که مشاهده می شود، در آخر در خروجی واضح است که چگونه دیتا استخراج شده است.

Functions

```

# Functions

def Rand(size):
    _Rand = np.random.rand(size)
    return _Rand

def Sigmoid(_input):
    _Sigmoid = (2 / (1 + np.exp(-_input))) - 1
    return _Sigmoid

def tanh(_input):
    return np.tanh(_input)

def ReLU(_input):
    if _input < 0:
        return 0
    else:
        return 0.01 * _input

def O(feature, weight, fcn):
    if fcn == 'sigmoid':
        return Sigmoid(np.dot(feature, weight))
    elif fcn == 'tanh':
        return tanh(np.dot(feature, weight))
    elif fcn == 'ReLU':
        return ReLU(np.dot(feature, weight))
    else:
        return -1

```

شکل 7) در ادامه تمامی توابع مربوطه را مشاهده میکنید.

به ترتیب توابع مربوط به تولید مقادیر رندم (در ادامه برای مقدار دهی رندم به وزن ها)، توابع فعال ساز Sigmoid، tanh، ReLU و همچنین تابع خروجی O نیز می باشد.

Hyper Parameters

```

[59] # Hyper_params

w = Rand(X.shape[1])
print(w)
learning_rate = 0.1
E_max = 1e6
E = 0
epoch = 50

[0.5769302  0.33222896 0.6105234  0.90179909 0.54135262 0.3630744
 0.92969223 0.71563389 0.21834187 0.86220284 0.18913979 0.22641869
 0.0131541  0.33814551 0.99341486 0.94735399 0.79660831 0.33526025
 0.76865511 0.19835019 0.56336076 0.3754508  0.96346875 0.775585
 0.22995294 0.87818812 0.94616958 0.80733258 0.05030582 0.55175481]

```

شکل 8) در این قسمت Hyperparameter ها مقدار دهی می شوند.

Without being shuffled and normalized

```
[109] n_learning = int(np.round(X.shape[0] * 1))

[60]
print(n_learning)
e_lst = []
for j in range(epoch):
    E = 0
    for i in range(n_learning):
        o = O(X[i], w, 'sigmoid')
        w += (0.5 * learning_rate * (Y[i] - o) * (1 - o ** 2) * X[i])
        E += (0.5 * (Y[i] - o) ** 2)
        # print(o)
    print(f"epoch_{j + 1} - input_{i+1}: {E}")
    if (E > E_max):
        break
    # e_lst.append(E / n_learning)
    e_lst.append(E)

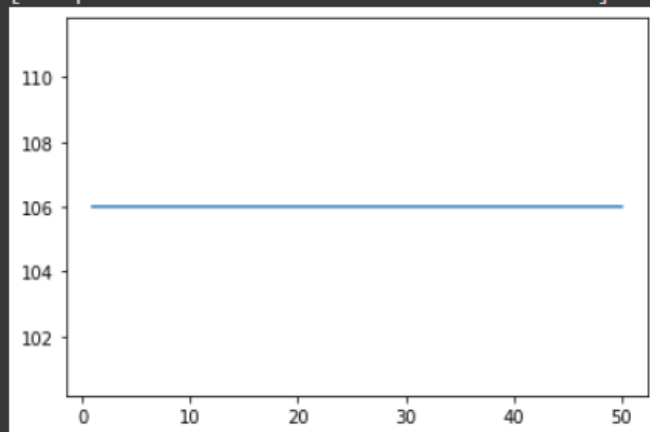
iteration = np.arange(1, epoch+1, 1)
plt.plot(iteration, e_lst)
```

شکل 9) در این قسمت، سیستم بدون در نظر گرفتن بایاس با همه ورودی ها آموزش داده می شود.

ملاحظه می کنید که با تابع فعال ساز sigmoid سیستم آموزش داده شده است.

در ادامه پس از epoch=50، ارور را مشاهده می کنید که بر روی مقدار 106 همگرا شده است.

```
epoch_50 - input_552: 103.0  
epoch_50 - input_553: 103.0  
epoch_50 - input_554: 103.0  
epoch_50 - input_555: 103.0  
epoch_50 - input_556: 103.0  
epoch_50 - input_557: 103.0  
epoch_50 - input_558: 103.0  
epoch_50 - input_559: 103.0  
epoch_50 - input_560: 103.0  
epoch_50 - input_561: 103.0  
epoch_50 - input_562: 103.0  
epoch_50 - input_563: 103.5  
epoch_50 - input_564: 104.0  
epoch_50 - input_565: 104.5  
epoch_50 - input_566: 105.0  
epoch_50 - input_567: 105.5  
epoch_50 - input_568: 106.0  
epoch_50 - input_569: 106.0  
[<matplotlib.lines.Line2D at 0x7f983b056390>]
```



شکل 10) پس از 50 بار آموزش دیدن بر روی همه ورودی ها، نمودار خطا بر حسب تکرار یا epoch به صورت فوق مشاهده می شود.

Normalizing

```
# normalizing

scaler = preprocessing.MinMaxScaler()
df = pd.DataFrame(X)
names = df.columns
d = scaler.fit_transform(df)
scaled_df = pd.DataFrame(d, columns=data.feature_names)
scaled_df.head()
```

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...
0	0.521037	0.022658	0.545989	0.363733	0.593753	0.792037	0.703140	0.731113	0.686364	0.605518	...
1	0.643144	0.272574	0.615783	0.501591	0.289880	0.181768	0.203608	0.348757	0.379798	0.141323	...
2	0.601496	0.390260	0.595743	0.449417	0.514309	0.431017	0.462512	0.635686	0.509596	0.211247	...
3	0.210090	0.360839	0.233501	0.102906	0.811321	0.811361	0.565604	0.522863	0.776263	1.000000	...
4	0.629893	0.156578	0.630986	0.489290	0.430351	0.347893	0.463918	0.518390	0.378283	0.186816	...

5 rows × 30 columns

شکل 11) در این تکه کد، با استفاده از توابع موجود در **pandas** و باقی توابع، ورودی ها نرمال سازی شده اند.

shuffling and resetting indexes

```
scaled_df = scaled_df.sample(frac=1.).reset_index(drop=True)
scaled_df.head()
```

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...
0	0.401297	0.330402	0.400180	0.256797	0.510698	0.315686	0.343486	0.401938	0.439899	0.168492	...
1	0.058592	0.371660	0.065510	0.025620	0.373928	0.340838	0.309513	0.107753	0.586869	0.687658	...
2	0.287236	0.139669	0.268952	0.164199	0.278866	0.055119	0.010682	0.043882	0.198485	0.109941	...
3	0.351129	0.584376	0.334877	0.213192	0.156360	0.100761	0.081443	0.086332	0.326768	0.092039	...
4	0.282503	0.213392	0.271923	0.157031	0.432157	0.184191	0.144213	0.167495	0.338384	0.310447	...

5 rows × 30 columns

شکل 12) در این تکه کد، عمل **shuffling** و **reset** کردن اندیس ها اتفاق افتاده است.

متد `sample` با `frac=1`، همه داده ها را به صورت تصادفی انتخاب کرده و متد `reset_index` با `drop=true`، اندیس جابه جا شده به علت تصادفی انتخاب کردن اندیس ها را `reset` میکند.

Train & Test parameter assignment

```
[135] train = scaled_df.sample(frac = 0.8).reset_index(drop=True)
test = scaled_df.drop(index = train.index).reset_index(drop=True)
print(train.shape, test.shape)
train.head()
test.head()
```

(455, 30) (114, 30)

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	wor radi
0	0.103744	0.140345	0.106489	0.049799	0.221901	0.208975	0.140300	0.108350	0.646970	0.414280	...	0.0739
1	0.307587	0.375719	0.308272	0.176331	0.442087	0.325502	0.249063	0.270328	0.333333	0.299705	...	0.2703
2	0.122391	0.209672	0.113468	0.057731	0.288977	0.065916	0.038707	0.082853	0.247980	0.296335	...	0.0917
3	0.305220	0.335475	0.290581	0.178961	0.341699	0.133427	0.137254	0.170875	0.271717	0.142165	...	0.3575
4	0.381419	0.237741	0.379656	0.231559	0.417080	0.358935	0.180904	0.305268	0.307071	0.394482	...	0.3141

5 rows x 30 columns

شکل 13) در این قسمت همانند شکل 12، داده های train و test به گونه ای که 80% کل داده ها با train و باقی داده ها به test اختصاص داده شده است، مقدار دهی می شوند.

Training with all data

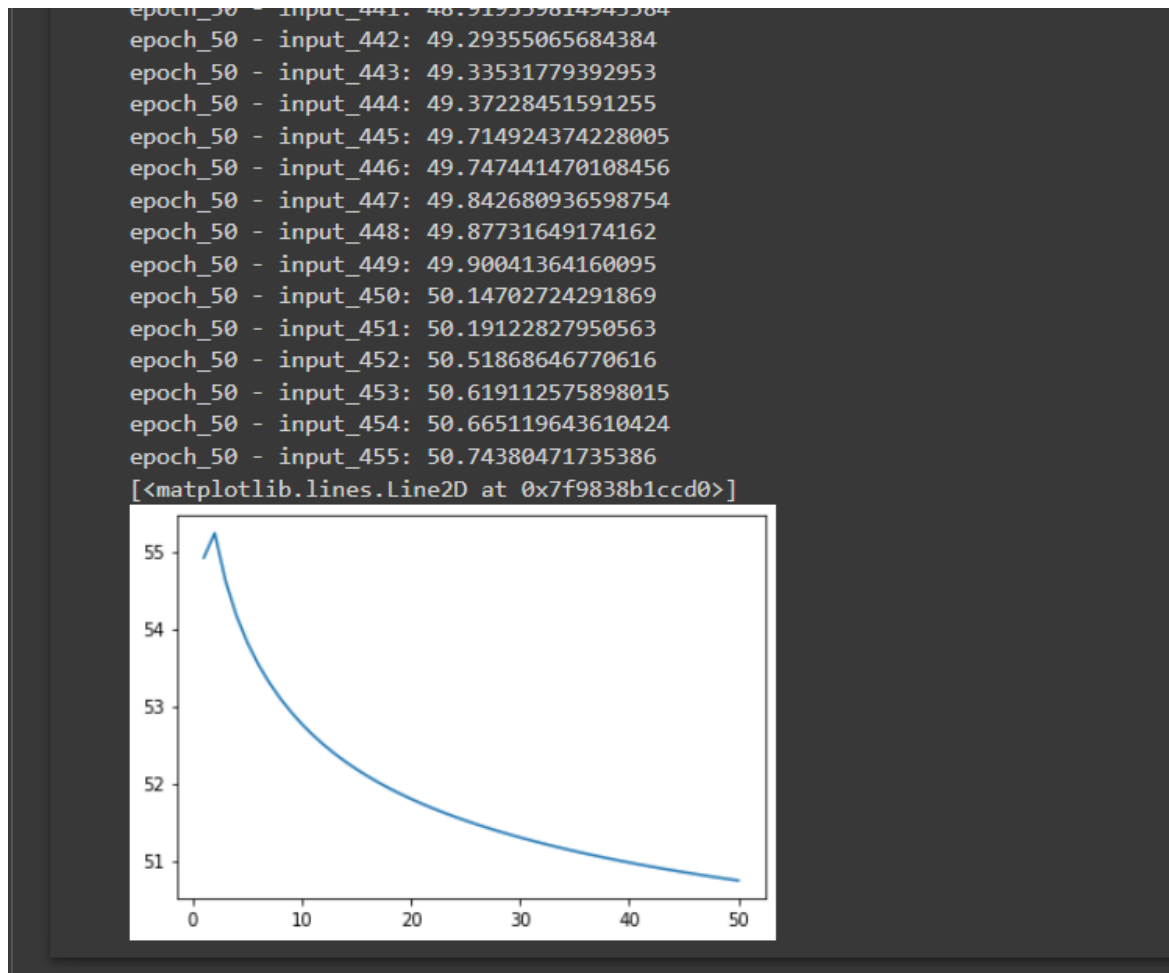
```
[136] e_lst_learn = []
for j in range(epoch):
    E = 0
    for i in range(n_learning):
        o = O(scaled_df.iloc[i], w, 'sigmoid')
        w += (0.5 * learning_rate * (Y[i] - o) * (1 - o ** 2) * scaled_df.iloc[i])
        E += (0.5 * (Y[i] - o) ** 2)
    # print(o)
    print(f"epoch_{j + 1} - input_{i+1}: {E}")
    # print(w)
    if(E > E_max):
        break
    e_lst_learn.append(E)

iteration = np.arange(1,epoch+1,1)
plt.plot(iteration, e_lst_learn)
```

```
epoch_50 - input_53: 8.123027953887929
epoch_50 - input_54: 8.210561957888787
epoch_50 - input_55: 8.422745242649771
epoch_50 - input_56: 8.647078689920129
epoch_50 - input_57: 8.719578151490328
epoch_50 - input_58: 8.804856362627898
epoch_50 - input_59: 8.842260770557596
epoch_50 - input_60: 9.076375221244467
epoch_50 - input_61: 9.176833647733059
epoch_50 - input_62: 9.420288414818769
epoch_50 - input_63: 9.467254657121066
epoch_50 - input_64: 9.60259006336289
```

شکل 14) در این قسمت، پس از نرمال شدن و شافل شدن متغیر ها، دوباره آموزش داده می شوند.

در نمودار شکل بعد ملاحظه می‌شود که با همان hyperparameter ها و همان تابع فعال ساز، به طور کلی بدون دست خوردن تنظیمات، اما وقتی ورودی ها نرماله سازی شده باشند، چقدر تفاوت در MSE ملاحظه می‌شود.



شکل 15) ملاحظه می‌شود که ارور به حد قابل توجه ای کاهش پیدا کرد.

test and train data

```
✓ [137] # train
    0s # test
      n_learning = train.shape[0]
      print(n_learning)
```

455

▼ Training with "train" Data

```
✓ [138] # Training:
    22s
      e_lst_learn = []
      for j in range(epoch):
          E = 0
          for i in range(n_learning):
              o = 0(train.iloc[i], w, 'sigmoid')
              w += (0.5 * learning_rate * (Y[i] - o) * (1 - o ** 2) * train.iloc[i])
              E += (0.5 * (Y[i] - o) ** 2)
          # print(o)
          print(f"epoch_{j + 1} - input_{i+1}: {E}")
          # print(w)
          if (E > E_max):
              break
          e_lst_learn.append(E)

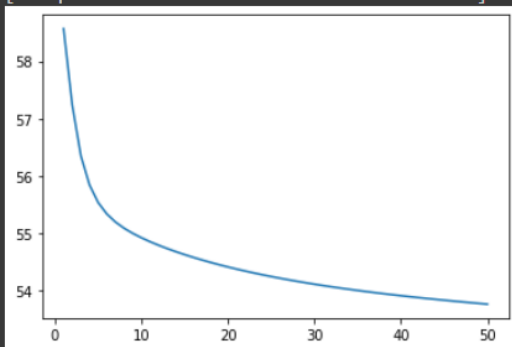
      iteration = np.arange(1, epoch+1, 1)
      plt.plot(iteration, e_lst_learn)
```

شکل 16) حال در این مرحله آموزش با ورودی داده های train انجام می شود و نتیجه را در شکل بعدی ملاحظه خواهید نمود.


```

epoch_50 - input_437: 51.7334783383262
epoch_50 - input_438: 51.81865655024501
epoch_50 - input_439: 51.83438049391286
epoch_50 - input_440: 51.85252267353462
epoch_50 - input_441: 51.891539915879
epoch_50 - input_442: 52.203559377133125
epoch_50 - input_443: 52.20512274185767
epoch_50 - input_444: 52.20768557715049
epoch_50 - input_445: 52.48573682148885
epoch_50 - input_446: 52.50912712947807
epoch_50 - input_447: 52.91790814777295
epoch_50 - input_448: 52.952207428495846
epoch_50 - input_449: 52.954493784620965
epoch_50 - input_450: 53.26402422101945
epoch_50 - input_451: 53.276743898146535
epoch_50 - input_452: 53.548232069260266
epoch_50 - input_453: 53.61489948648009
epoch_50 - input_454: 53.64767025242127
epoch_50 - input_455: 53.75258308142169
[<matplotlib.lines.Line2D at 0x7f9838a8c790>]

```



شکل 17) نتیجه به صورت فوق قابل مشاهده است.

در مراحل بعد، به سیستم بایاس هم اضافه می‌شود و نتایج با توجه به آن هم محاسبه می‌شود.

Adding bias

Problem: If we don't shuffle the dataset, the Error is lower than when, the dataset is shuffled

```

✓ [125] df_bias = scaled_df
    df_bias["bias"] = -1
    # df_bias = df_bias.sample(frac=1.).reset_index(drop=True)
    w_bias = Rand(df_bias.shape[1])
    n_learning = df_bias.shape[0]
    n_learning
    # w_bias.shape
    # df_bias.shape
    # df_bias

```

569

شکل 18) دیتا فریم با بایاس، به صورت فوق ساخته می‌شود.

نکته ای که حائز اهمیت است این است که در با همان تنظیمات و هایپر پارامتر ها، در صورت عدم شافل کردن داده ها، ارور کمتری به دست می آید تا زمانی که داده ها را شافل کنیم.

▼ Training complete dataset with bias

```
✓ [126] # with bias :
30s

# Training:

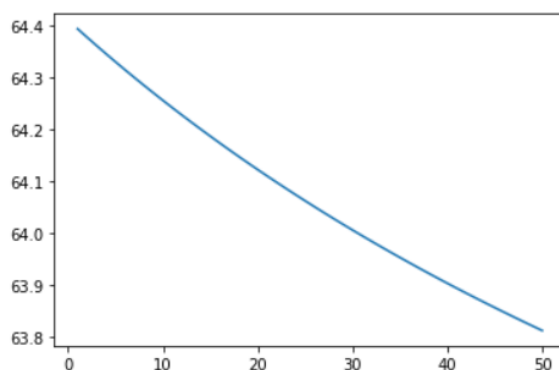
e_lst_learn = []
for j in range(epoch):
    E = 0
    for i in range(n_learning):
        o = O(df_bias.iloc[i], w_bias, 'sigmoid')
        w_bias += (0.5 * learning_rate * (Y[i] - o) * (1 - o ** 2) * df_bias.iloc[i])
        E += (0.5 * (Y[i] - o) ** 2)
    #     print(o)
    print(f"epoch_{j + 1} - input_{i+1}: {E}")
    #     print(w)
    if(E > E_max):
        break
    e_lst_learn.append(E)

iteration = np.arange(1,epoch+1,1)
plt.plot(iteration, e_lst_learn)
```

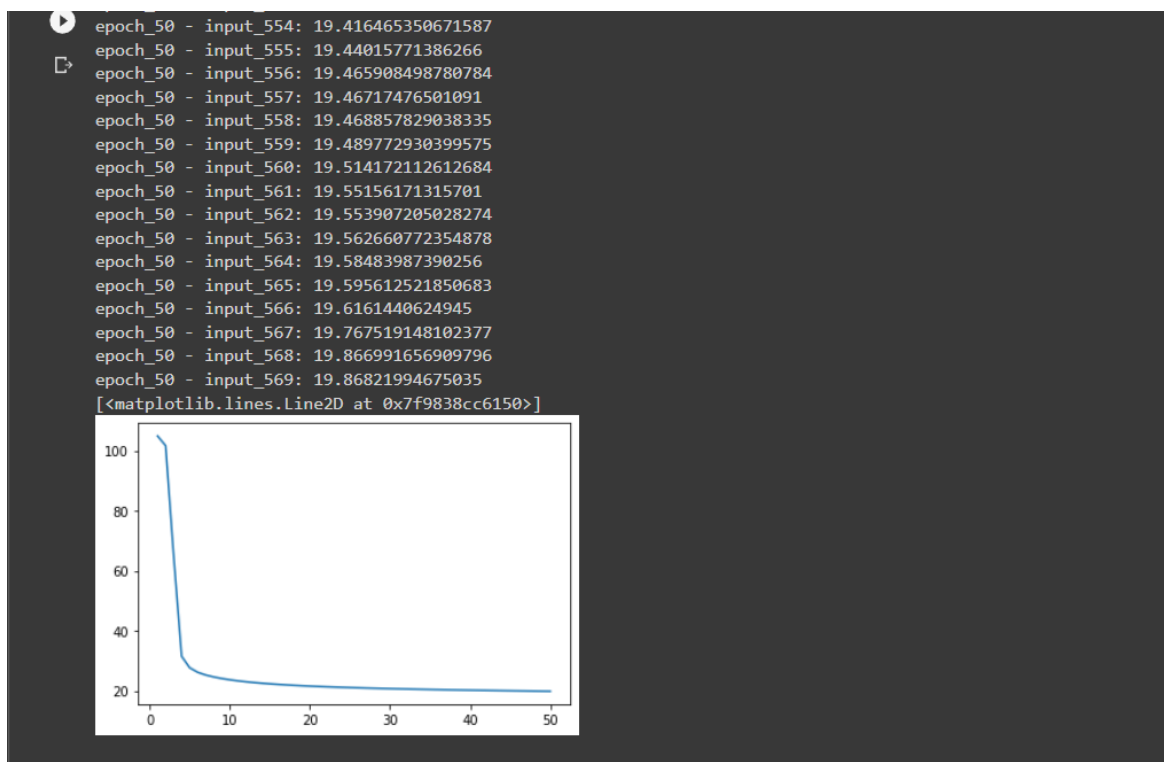
شکل 19) در این قسمت تمامی دیتا ست را به همراه بایاس آموزش داده ایم که نتایج به صورت شکل زیر قابل مشاهده است.

```
epoch_50 - input_568: 63.77001110057285
epoch_50 - input_569: 63.811785702320506
```

Out[36]: [matplotlib.lines.Line2D at 0x223713aaf40]



شکل 20-1 خروجی MSE با دیتاست نرمال شده و بایاس با $\eta = 1$



شکل 20-2) نتایج تحلیل MSE نشان می‌دهد که با همان تنظیمات و فقط به علت وجود پارامتر بایاس در شبکه، MSE چقدر کاهش پیدا کرد و تقریباً به $1/3$ مقدار بدون بایاس میل کرد.

```

Train and test assignment for model with bias

[141] df_bias = df_bias.sample(frac=1.).reset_index(drop=True)
      train = df_bias.sample(frac = 0.8).reset_index(drop=True)
      test = df_bias.drop(index = train.index).reset_index(drop=True)
      print(train.shape, test.shape)
      train.head()
      test.head()
      n_learning = train.shape[0]

(455, 31) (114, 31)

```

شکل 20) در این قسمت داده های train و test نیز تخصیص داده می‌شوند

روند کار به این صورت است که ابتدا کل داده ها شافل می‌شوند، سپس همانطور که پیش تر ذکر شده بود، با کمک توابع sample و reset_index، ترتیب اندیس ها به خورده و همینطور ورودی هایی به train و test تخصیص داده می‌شود.

▼ Training "train" datas with bias

```

# with bias :

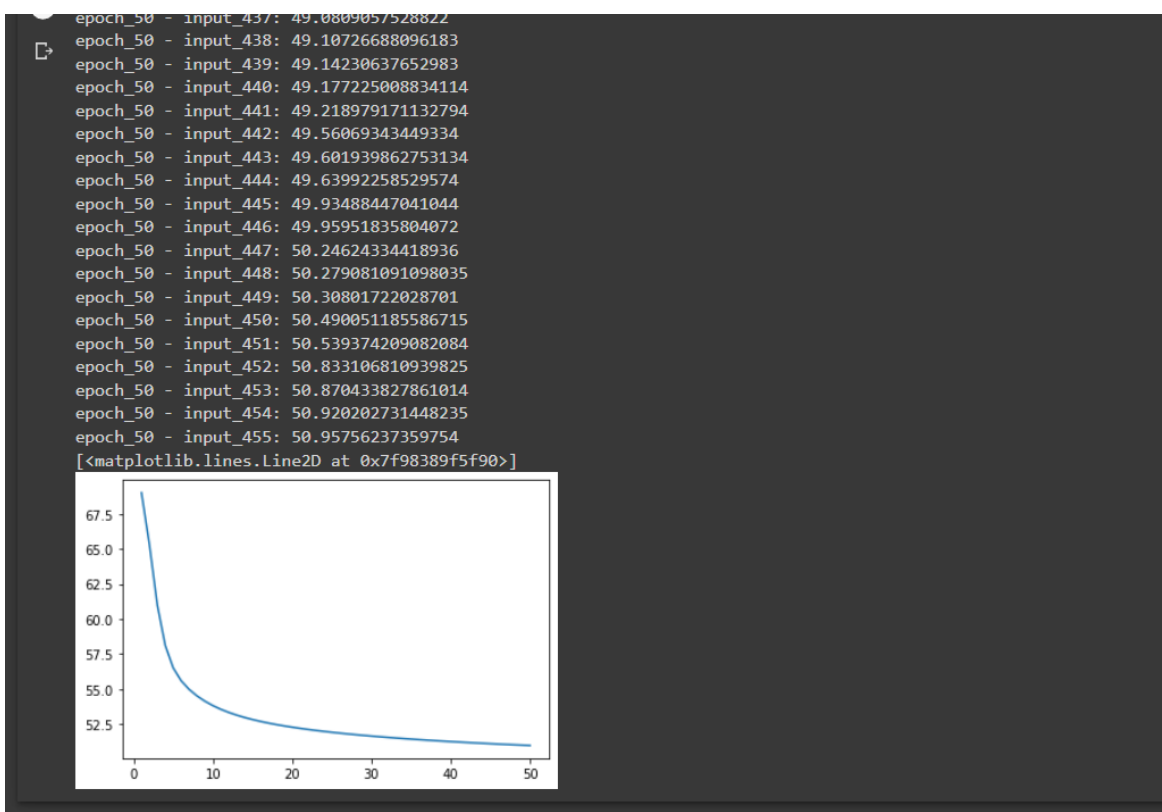
# Training:

e_lst_learn = []
for j in range(epoch):
    E = 0
    for i in range(n_learning):
        o = 0(train.iloc[i], w_bias, 'sigmoid')
        w_bias += (0.5 * learning_rate * (Y[i] - o) * (1 - o ** 2) * train.iloc[i])
        E += (0.5 * (Y[i] - o) ** 2)
    # print(o)
    print(f"epoch_{j + 1} - input_{i+1}: {E}")
    # print(w)
    if(E > E_max):
        break
    e_lst_learn.append(E)

iteration = np.arange(1,epoch+1,1)
plt.plot(iteration, e_lst_learn)

```

شکل 21) سپس در مرحله ی آخر، شبکه فقط با داده های train آموزش داده می شود و در ادامه نتایج قابل مشاهده است.



شکل 22) نتایج قابل مشاهده است.

بدیهی است که به علت کمتر بودن داده های train نسبت به کل دیتاست، ارور بیشتری (MSE) قابل مشاهده است.

در تحلیل اعمال انجام شده، شایان ذکر است که درست است که روی کل داده اگر شبکه آموزش داده شود دقت بسیار بالایی برای همان داده ها حاصل می شود. اما، ممکن است در configuration از تنظیمات، در صورتی که داده جدید به سیستم وارد شود، به علت overfitting روی داده ها، آن داده های جدید با خطای بسیار زیادی پیش بینی شوند. و راه حل در همین استفاده از داده های train برای آموزش و تست فرضی داده های test روی سیستم است.

حال برای حالت شبکه با بایاس، با همان پارامتر ها اما با توابع activation دیگر خواهیم داشت:

TANH -2-3-3

```
# with bias :

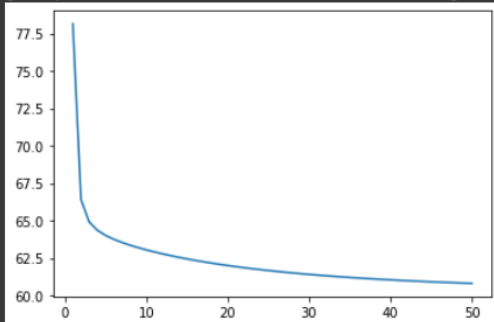
# Training:

e_lst_learn = []
for j in range(epoch):
    E = 0
    for i in range(n_learning):
        o = 0(df_bias.iloc[i], w_bias, 'tanh')
        w_bias += (0.5 * learning_rate * (Y[i] - o) * (1 - o ** 2) * df_bias.iloc[i])
        E += (0.5 * (Y[i] - o) ** 2)
    # print(o)
    print(f"epoch_{j + 1} - input_{i+1}: {E}")
    # print(w)
    if(E > E_max):
        break
    e_lst_learn.append(E)

iteration = np.arange(1,epoch+1,1)
plt.plot(iteration, e_lst_learn)
```

شکل 23) به صورت قابل مشاهده در شکل، پارامتر را به tanh تغییر میدهم

```
epoch_50 - input_560: 58.957950898158764
epoch_50 - input_561: 58.96489108241726
epoch_50 - input_562: 58.980455723723196
epoch_50 - input_563: 59.31450018821946
epoch_50 - input_564: 59.628199144772786
epoch_50 - input_565: 59.92554710215866
epoch_50 - input_566: 60.23629736429608
epoch_50 - input_567: 60.52713077986109
epoch_50 - input_568: 60.733159791149255
epoch_50 - input_569: 60.79986432393173
[<matplotlib.lines.Line2D at 0x7f98387b8fd0>]
```

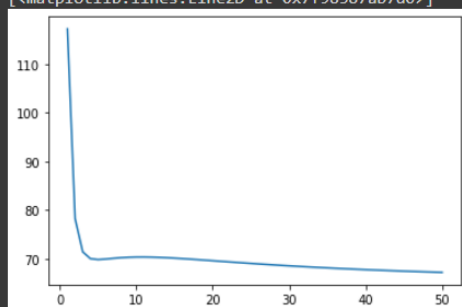


شکل 24) خروجی را ملاحظه می‌کنید که دقت و خطای بیشتری را نسبت به حالت تابع فعال ساز sigmoid ارائه می‌کند.

ReLU -2-3-4

خروجی برای تابع ReLU به صورت زیر می‌باشد

```
epoch_50 - input_557: 65.70710225344759
epoch_50 - input_556: 65.66365354736976
epoch_50 - input_557: 65.70710225344759
epoch_50 - input_558: 65.76839956690739
epoch_50 - input_559: 65.87937729922119
epoch_50 - input_560: 65.96340689496263
epoch_50 - input_561: 65.9919538142535
epoch_50 - input_562: 66.05855297831188
epoch_50 - input_563: 66.23171735189865
epoch_50 - input_564: 66.42529571648511
epoch_50 - input_565: 66.59143639266773
epoch_50 - input_566: 66.7536931964973
epoch_50 - input_567: 66.93488098269565
epoch_50 - input_568: 67.08265847823193
epoch_50 - input_569: 67.192143627501
[<matplotlib.lines.Line2D at 0x7f98387ab7d0>]
```



شکل 25) مشاهده می‌شود که MSE از دو حالت قبلی بیشتر است و نشان دهنده این است که با تنظیمات پارامترهای یکسان، MSE در حالت تابع فعال ساز sigmoid از همه آن‌ها کمتر است.

5-2- تمرین مربوطه

اثر اضافه کردن بایاس، در این است که سرعت همگرایی نیز بیشتر می‌شود و مدل به دقت بهتری می‌رسد.

محیط MATLAB

در MATLAB از همان ابتدا script به صورت Class نوشته شده است و از OOP استفاده شده است. شایان ذکر است که در MATLAB صرفاً syntax و برخی نکات جزئی با Python متفاوت است، و الگوریتم و روند کلی کار مشابه است. بنابراین توضیحات اضافه آورده نمی‌شود.

```
2 classdef Perceptron
3     methods(Static)
4         %%
5         function output_ = Sigmoid(input_)
6             output_ = 2 / (1 + exp(-input_)) - 1;
7         end
8         %%
9         function output_ = Rand(input_)
10            output_ = rand([1, input_]);
11        end
12        %%
13        function output_ = ReLu(input_)
14            if input_ <= 0
15                output_ = 0;
16            else
17                output_ = input_ / 10;
18            end
19        end
20        %%
21        function output_ = O(feature, weight, Activation_Function)
22            if Activation_Function == "Sgn"
23                output_ = sign(dot(feature, weight));
24            elseif Activation_Function == "Tanh"
25                output_ = tanh(dot(feature, weight));
26            elseif Activation_Function == "ReLu"
27                output_ = Perceptron.ReLu(dot(feature, weight));
28            else
29                output_ = Perceptron.Sigmoid(dot(feature, weight));
30            end
31        end
end
```

شکل 26) در این شکل مشاهده می‌شود که شبکه در داخل کلاس Perceptron نیز پیاده سازی شده است.

```

31 - end
32 - %%
33 - function [lst, weight] = perceptron(X, Y, Epoch, learning_rate, E_max, Activation_Function )
34 -     lst = zeros([1, Epoch]);
35 -     feature_size = size(X,2);
36 -     weight = Perceptron.Rand(feature_size);
37 -     dataset_size = size(X,1);
38 -     for i = 1:Epoch
39 -         E = 0;
40 -         for j = 1:dataset_size
41 -             o = Perceptron.O(X(j,:), weight, Activation_Function);
42 -             weight = weight + 0.5 * learning_rate * (Y(j) - o) * (1 - o ^ 2) * X(j, :);
43 -             E = E + 0.5 * (Y(j) - o) ^ 2;
44 -             if E > E_max
45 -                 break
46 -             end
47 -         end
48 -         lst(i) = E;
49 -     end
50 -     plot(1:Epoch, lst);
51 - end
52 - %%
53 - function output_ = Normalizing(input_)
54 -     input_ = input_ - min(input_);
55 -     output_ = input_ ./ max(input_);
56 - end
57 - end
58 - end

```

شکل 27) در ادامه شکل قبل، این شکل تکمیل کننده فایل "Perceptron.m" نیز می باشد.

پس از پیاده سازی class، در فایل "Breast_Cancer.m" باقی Script آورده شده است و خواسته ها نیز پیاده سازی گردیده است.

در ادامه محتویات داخل فایل ذکر شده قابل مشاهده است.

```

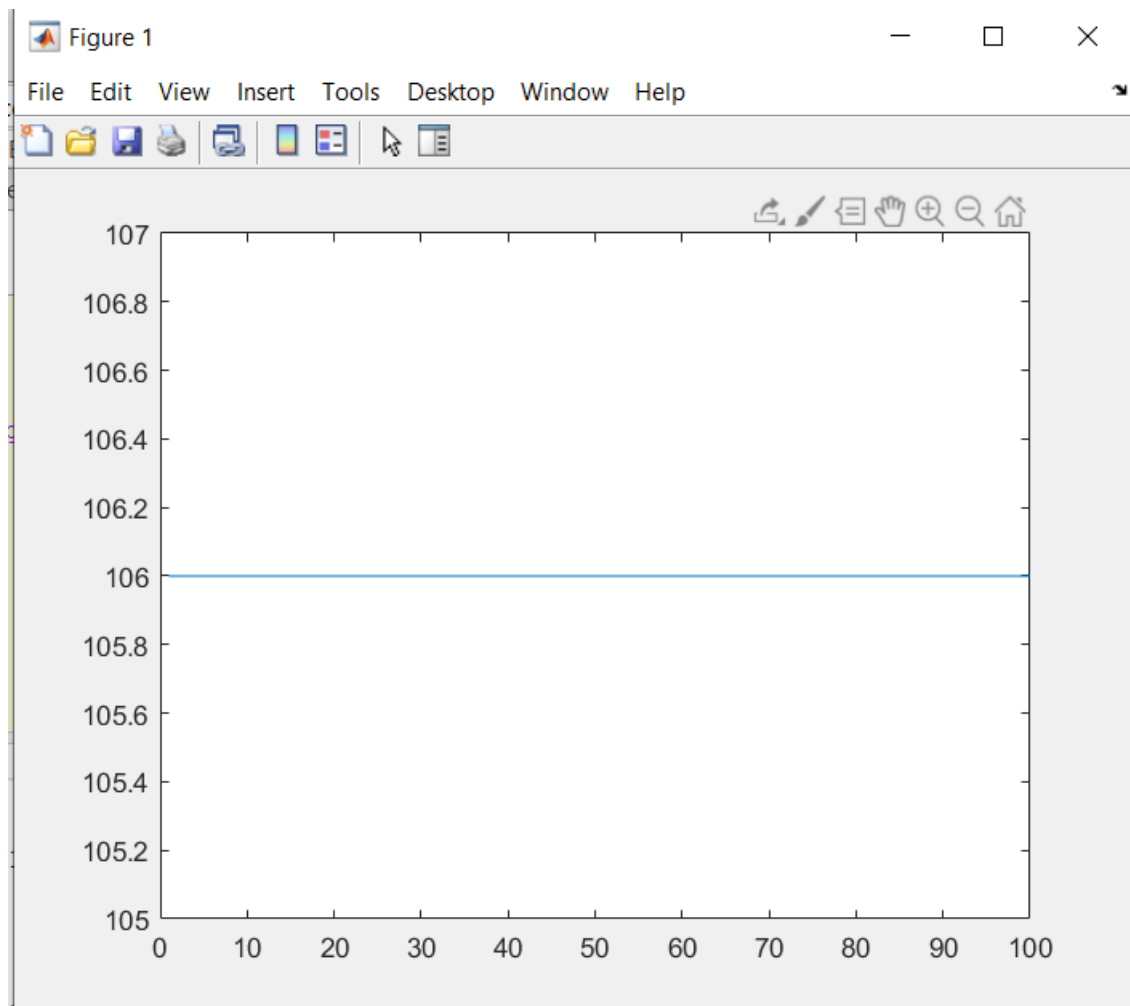
1  %% Importing data
2  data = readmatrix("data.txt");
3
4  %% Without Normalizing
5  X = data(2:end, 2:31);
6  Y = data(2:end, 32);
7  Perceptron.perceptron(X, Y, 100, 0.1, 1e6, "Sigmoid");
8
9  %% Normalizing
10
11  X_norm = Perceptron.Normalizing(X);
12  Y_norm = Perceptron.Normalizing(Y);
13
14  %% Sigmoid
15
16  Perceptron.perceptron(X_norm, Y_norm, 100, 0.1, 1e6, "Sigmoid");
17
18  %% Tanh
19
20  Perceptron.perceptron(X_norm, Y_norm, 100, 0.2, 1e6, "Tanh");
21
22  %% Sigmoid With Bias
23
24  X_bias = [X_norm, ones(569,1)];
25
26  Perceptron.perceptron(X_bias, Y_norm, 100, 0.1, 1e6, "Sigmoid");
27
28  %% Tanh with Bias
29
30  Perceptron.perceptron(X_bias, Y_norm, 100, 0.2, 1e6, "Tanh");

```

شکل 28) Script های استفاده از کلاس Perceptron

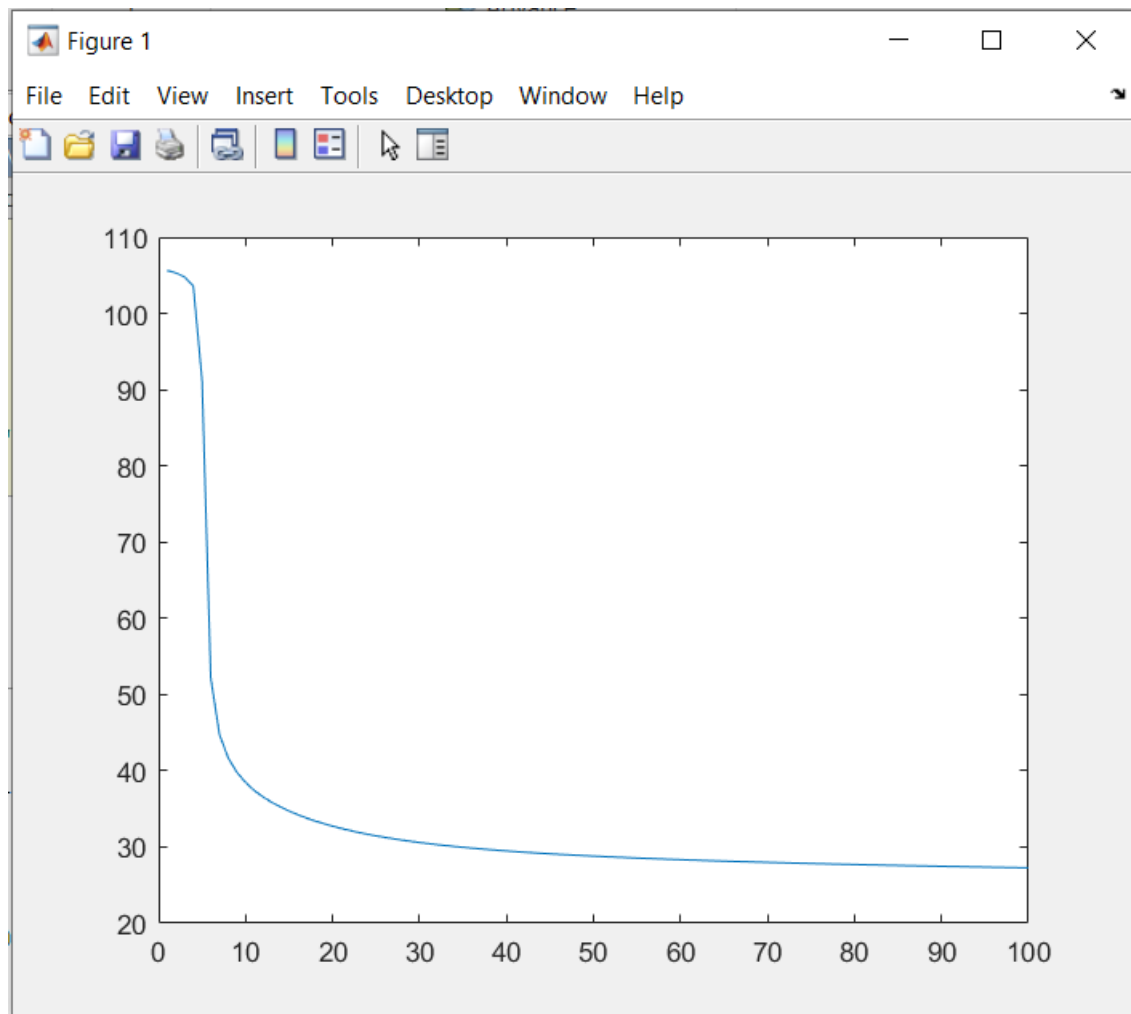
نتایج 2-3-5

بدون نرمال سازی MSE -1-1-1-1



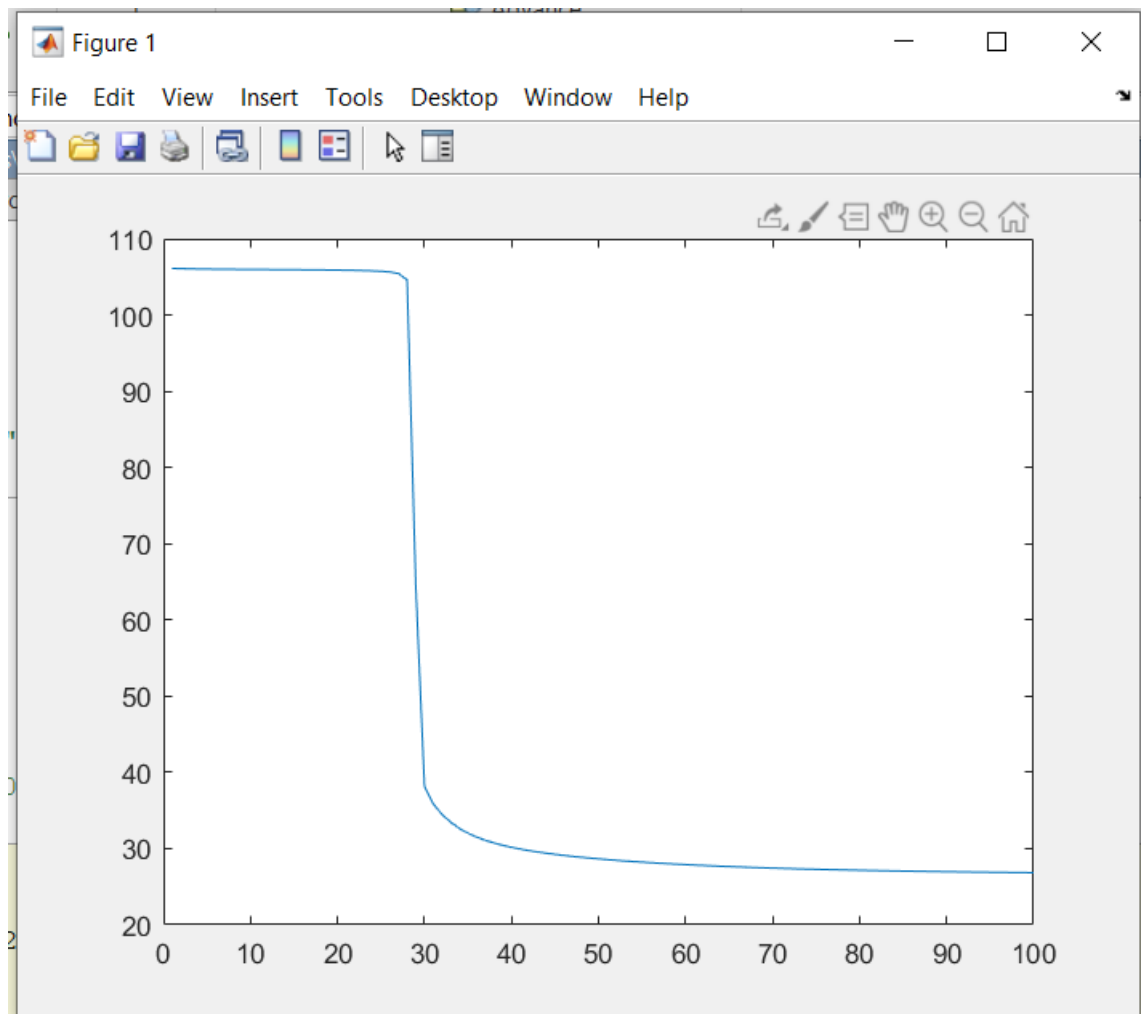
شکل 29 Perceptron بدون نرماله سازی

1-1-1-2 فعال سازی Sigmoid با نرماله سازی



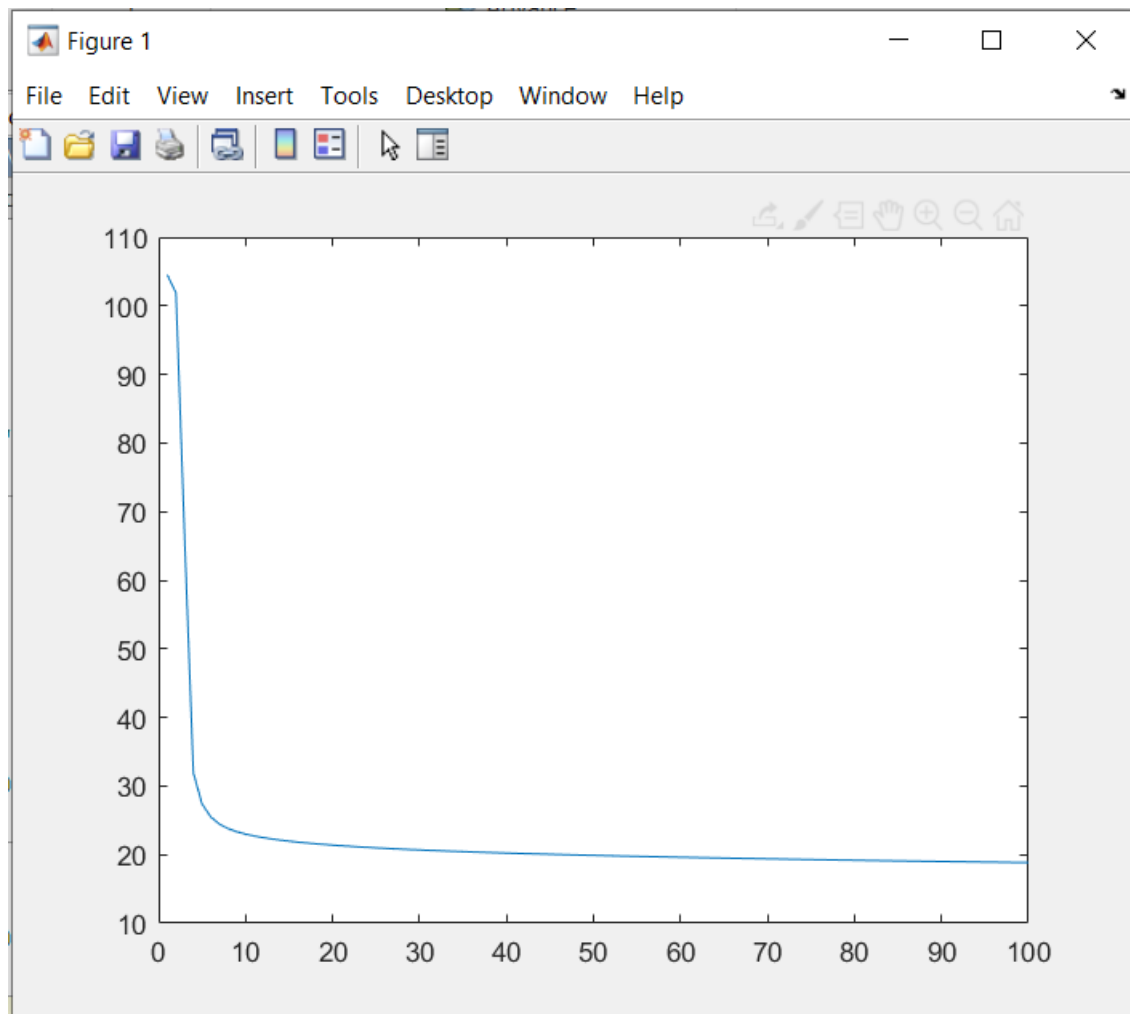
شکل 30) نمودار MSE برای فعال سازی sigmoid با نرمال سازی

1-1-1-3 فعال سازی tanh با نرماله سازی



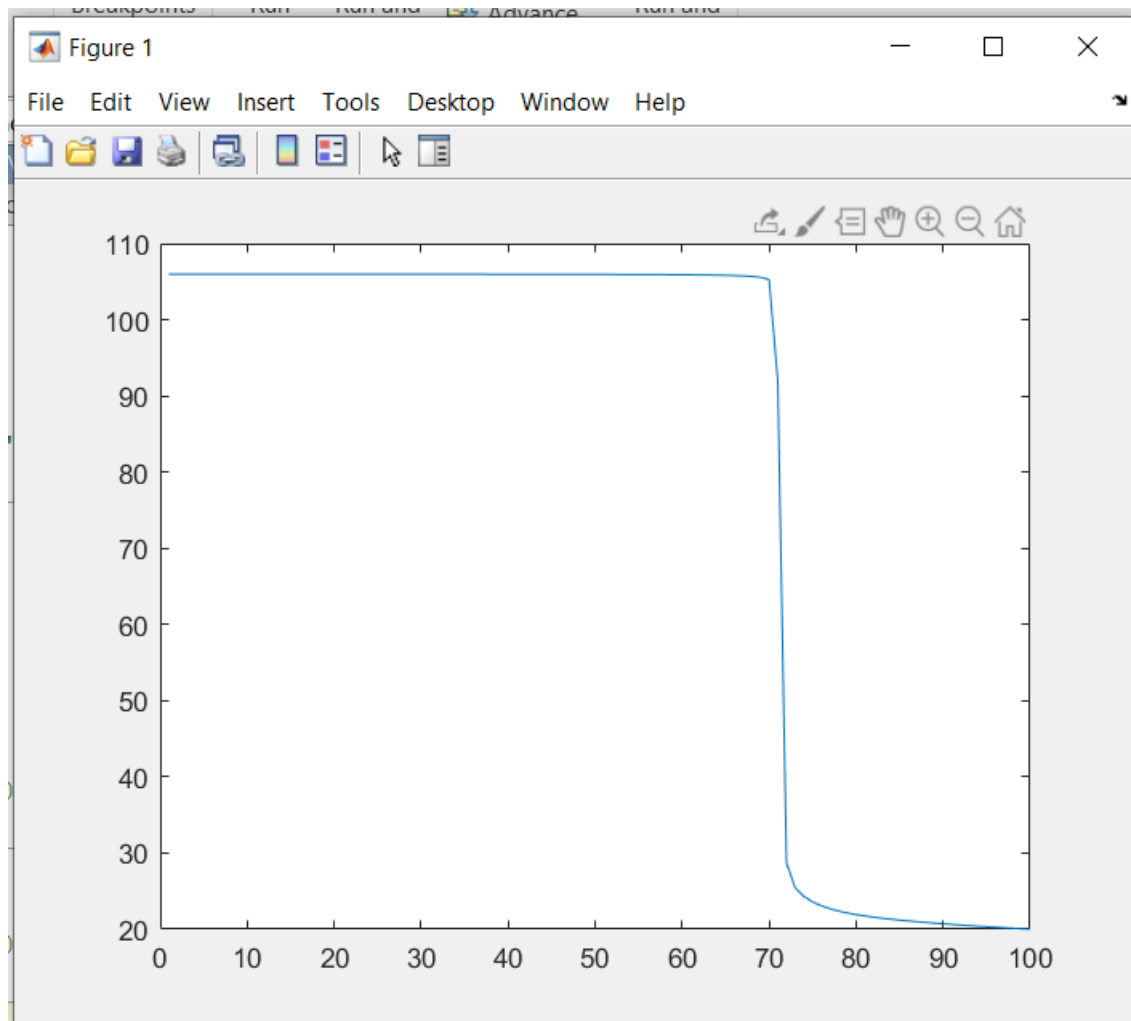
شکل 31 نمودار MSE برای فعال ساز tanh با نرماله سازی

1-1-1-4 فعال ساز sigmoid با بایاس و نرماله شده



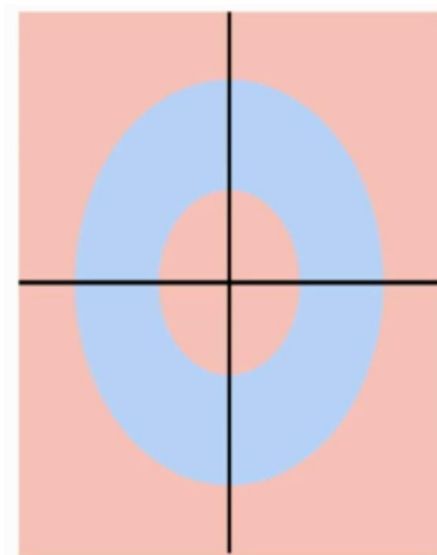
شکل 32 نمودار MSE برای فعال ساز Sigmoid با بایاس و نرماله شدن

فعال سازی tanh با بایاس و نرماله شدن -1-1-1-5



شکل 33) نمودار MSE برای فعال سازی tanh با بایاس و نرماله شدن

6-2- بررسی به کار گیری الگوریتم Perceptron برای داده های زیر



شکل 34 داده های گلبهی رنگ گروه a و آبی رنگ

گروه b

اگر بخواهیم تنها یک لایه Perceptron را به کار بگیریم، جداسازی داده ها در این حالت ناممکن است چرا که یک لایه ی Perceptron امکان جداسازی توابع غیرخطی را ندارد و مدل همگرا نمی شود. اما با به کار گیری چندین لایه، جداسازی این داده ها به سادگی قابل انجام است.

پیاده سازی توابع XOR، OR، و AND با Perceptron در

MATLAB

2-7- پیاده سازی AND

```
1 - X = [  
2 -     [0, 0];  
3 -     [0, 1];  
4 -     [1, 0];  
5 -     [1, 1];  
6 - ];  
7 - Y = [  
8 -     0;  
9 -     0;  
10 -    0;  
11 -    1;  
12 - ];  
13  
14 - weight = Perceptron.Rand(size(X, 2));  
15 - Epoch = 1000;  
16 - learning_rate = 0.5;  
17 - E_max = 1e6;  
18 - E = 0;  
19  
20 - for i = 1:10000  
21 -     E = 0;  
22 -     for j = 1:size(4)  
23 -         o = Perceptron.O(X(j,:), weight, "Sigmoid");  
24 -         weight = weight + 0.5 * learning_rate * (Y(j) - o) * (1 - o ^ 2) * X(j,:);  
25 -         E = E + 0.5 * (Y(j) - o) ^ 2;  
26 -         if E > E_max  
27 -             break  
28 -         end  
29 -     end  
30 - end
```

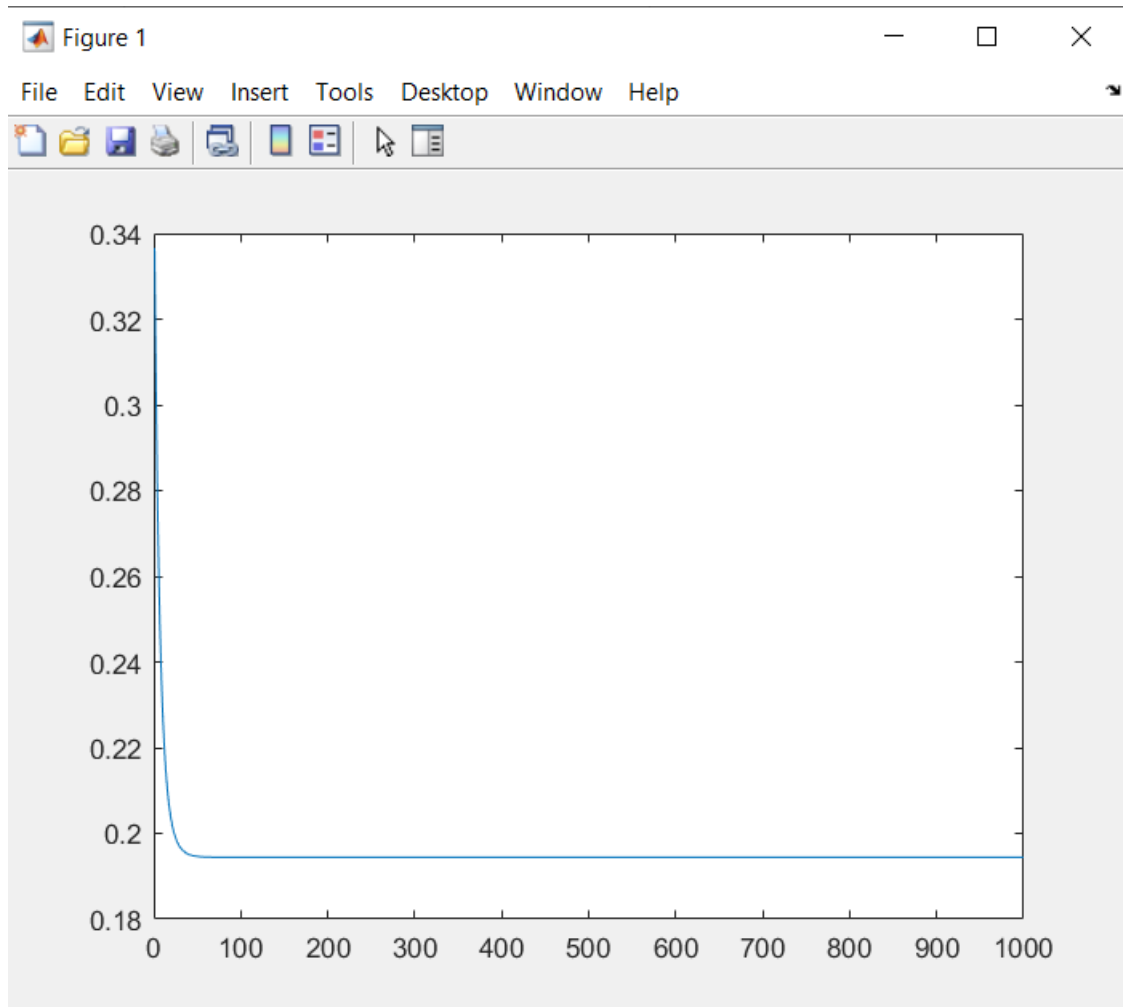
شکل 35) در فایل "And.m" نیز Script فوق موجود است و در آن از کلاس **Perceptron** نیز استفاده شده است.

همچنین در فایل "Modularized.m" نیز از این فایل ها استفاده شده است.

```
20 %% And  
21 - X_and = [  
22 -     [0, 0];  
23 -     [0, 1];  
24 -     [1, 0];  
25 -     [1, 1];  
26 - ];  
27 - Y_and = [  
28 -     0;  
29 -     0;  
30 -     0;  
31 -     1;  
32 - ];  
33  
34 - w_sigmoid_and = Perceptron.perceptron(X_and, Y_and, 1000, 0.1, 1e6, "Sigmoid");  
35 - w_tanh_and = Perceptron.perceptron(X_and, Y_and, 1000, 0.1, 1e6, "Tanh");  
36 - w_relu_and = Perceptron.perceptron(X_and, Y_and, 1000, 0.1, 1e6, "ReLu");  
37
```

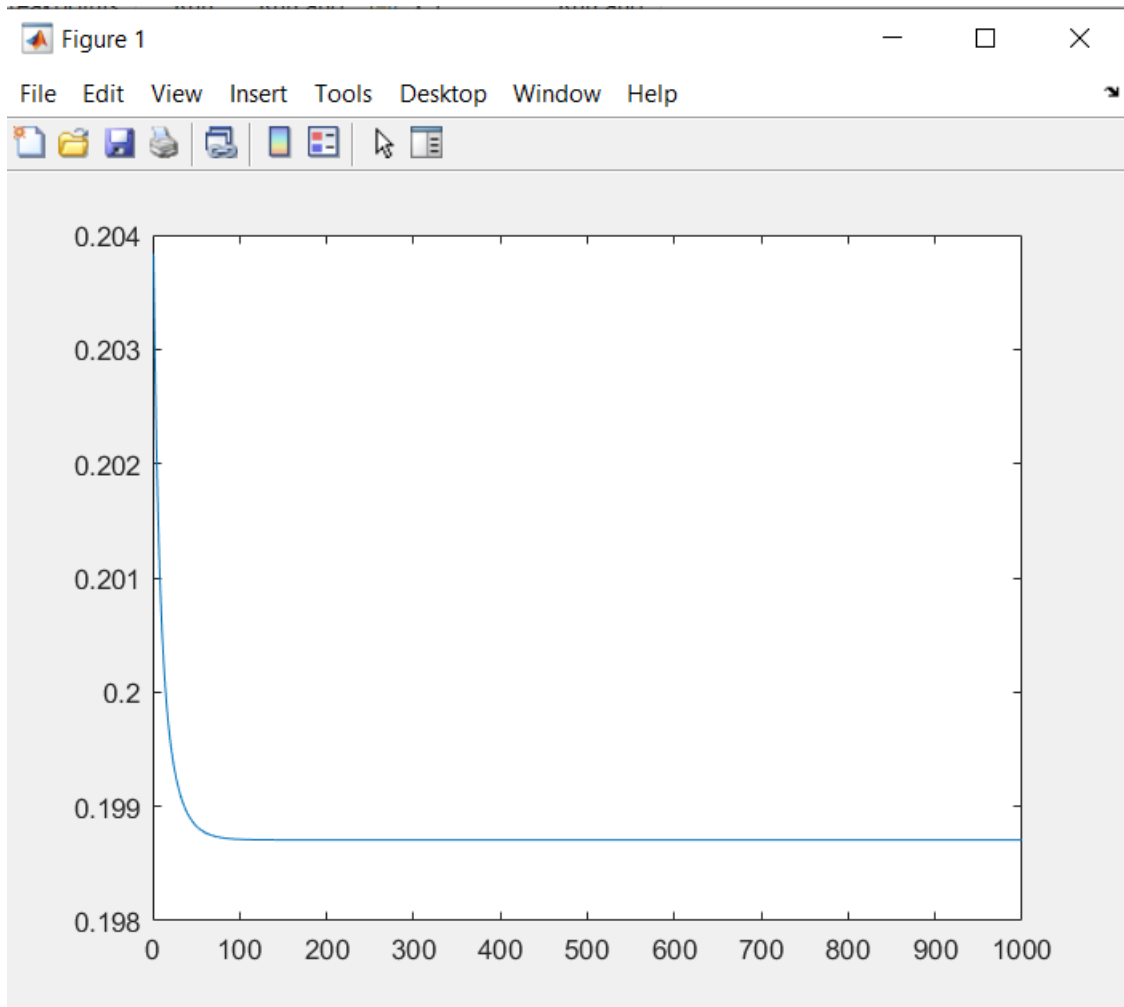
شکل 36) snippet مربوط به AND داخل کد

2-3-6- تابع فعال سازی sigmoid



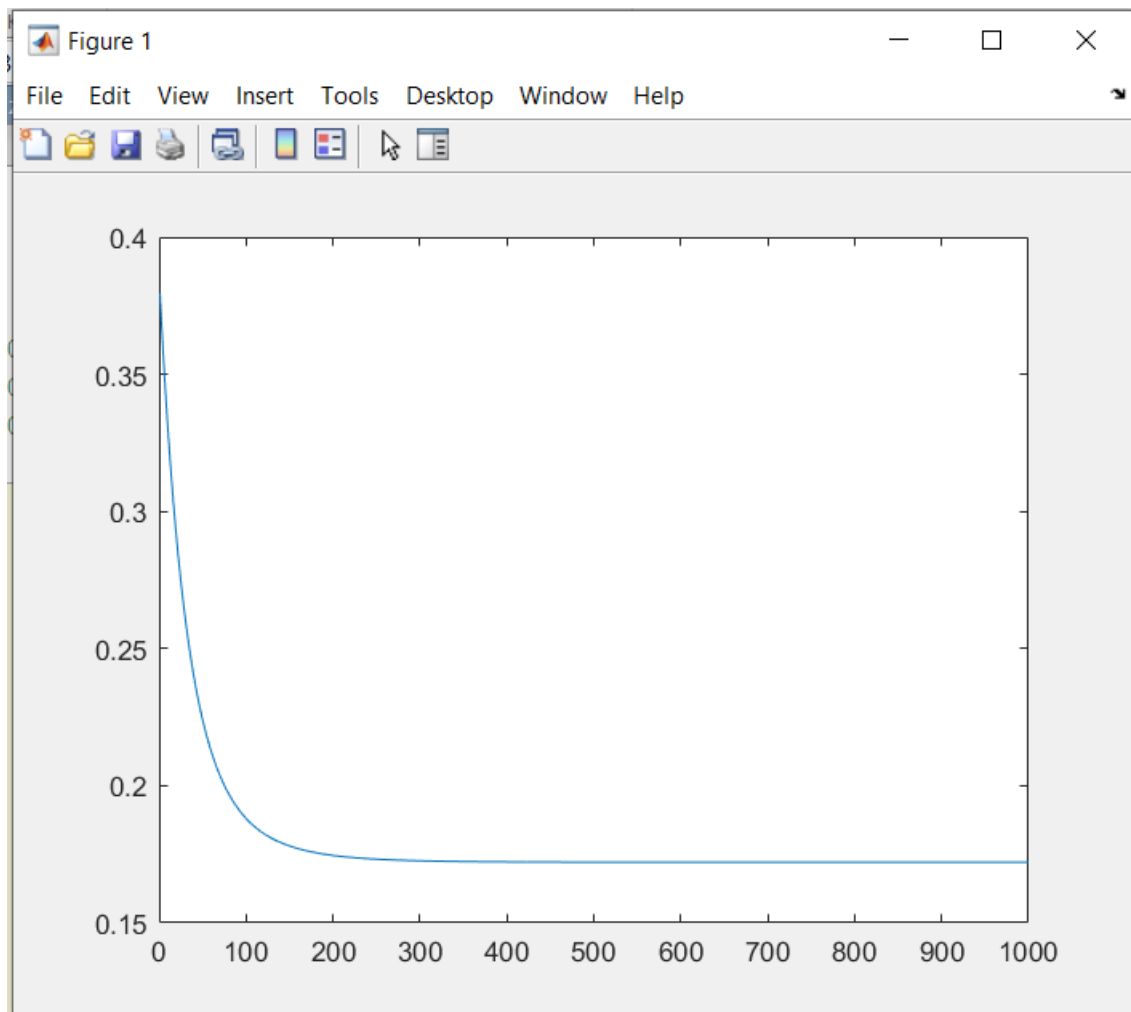
شکل 37 MSE با تابع فعال ساز sigmoid

2-3-7- Tanh - تابع فعال سازی Tanh



شکل 38 MSE با تابع فعال ساز Tanh

2-3-8- تابع فعال ساز ReLU



شکل 39) MSE با تابع فعال ساز ReLU

2-3-9- پیاده سازی در Python

به طور کلی تمرکز بر روی Script داخل MATLAB می باشد.

```
# AND:

import numpy as np

def Rand(size):
    _Rand = np.random.rand(size)
    return _Rand

def Sigmoid(input_):
    _Sigmoid = (2 / (1 + np.exp(-input_))) - 1
    return _Sigmoid

def O(feature, weight):
    _O = Sigmoid(np.dot(feature, weight))
    return _O

x = [[0, 0], [0, 1], [1, 0], [1, 1]]
y = [[0], [0], [0], [1]]

w = Rand(2)
learning_rate = 0.1
Emax = 1e6
E = 0
epoch = 20

elst = []
for i in range(20):
    E = 0
    for j in range(4):
        o = O(x[j], w)
        w += (0.5 * learning_rate * (y[j] - o) * (1 - o ** 2) * x[j])
        E += (0.5 * (y[j] - o) ** 2)
        if(E > Emax):
            break
    elst.append(E)
```

[5]

شکل 40) پیاده سازی AND در Python، شایان ذکر است که به علت تکرار مفاهیم، از توضیح بیشتر پرهیز شده است.

2-8- پیاده سازی Or

```
1 - X = [
2     [0, 0];
3     [0, 1];
4     [1, 0];
5     [1, 1];
6 ];
7 - Y = [
8     0;
9     1;
10    1;
11    1;
12 ];
13
14 - weight = Perceptron.Rand(size(X, 2));
15 - Epoch = 1000;
16 - learning_rate = 0.01;
17 - E_max = 1e6;
18 - E = 0;
19
20 - for i = 1:1000
21     E = 0;
22     for j = 1:size(X, 1)
23         o = Perceptron.O(X(j,:), weight, "Sigmoid");
24         weight = weight + 0.5 * learning_rate * (Y(j) - o) * (1 - o ^ 2) * X(j,:);
25         E = E + 0.5 * (Y(j) - o) ^ 2;
26         if E > E_max
27             break
28         end
29     end
30 - end
```

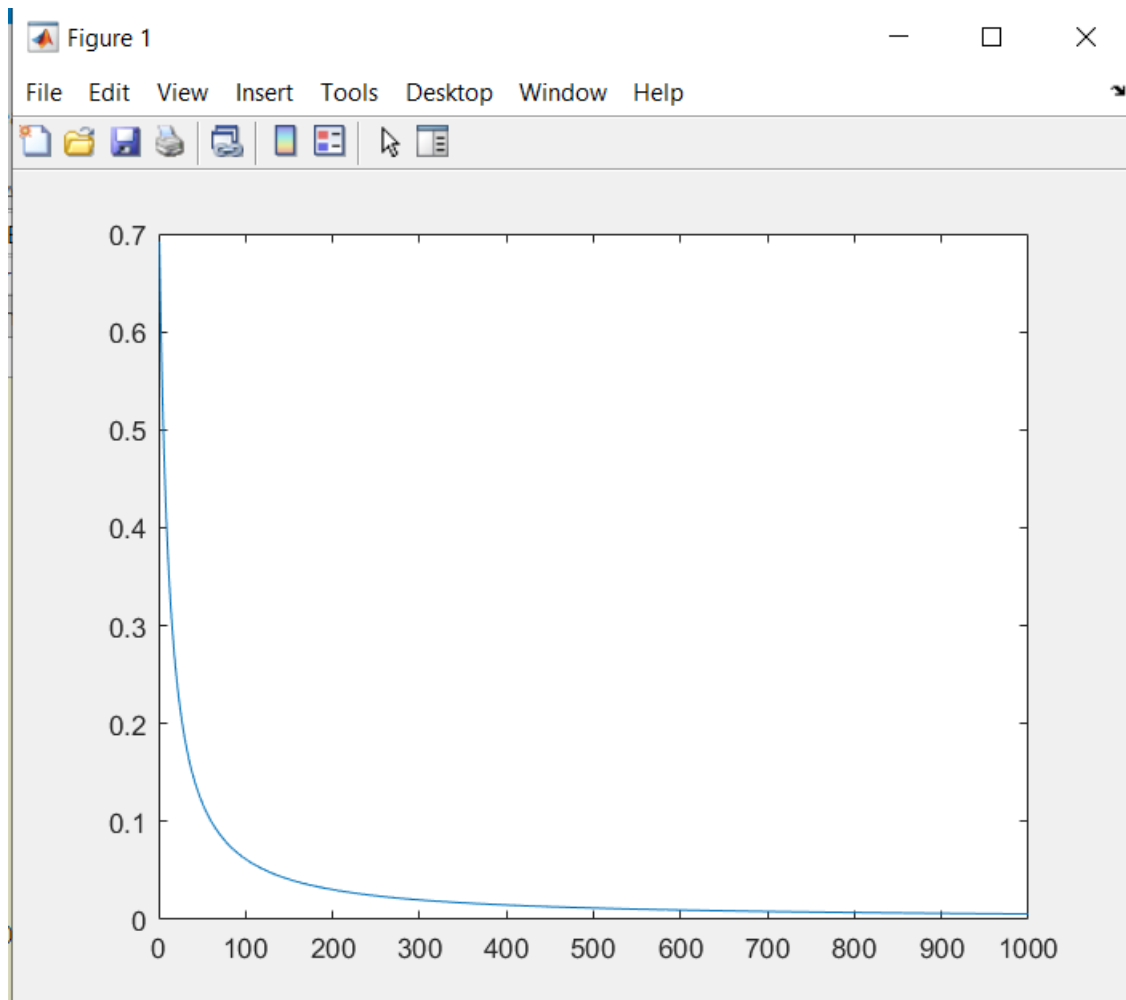
شکل 42) در فایل "Or.m" نیز Script فوق موجود است و در آن از کلاس **Perceptron** نیز استفاده شده است.

همچنین در فایل "Modularized.m" نیز از این فایل ها استفاده شده است.

```
2 %% Or
3 - X_or = [
4     [0, 0];
5     [0, 1];
6     [1, 0];
7     [1, 1];
8 ];
9 - Y_or = [
10    0;
11    1;
12    1;
13    1;
14 ];
15
16 w_sigmoid_or = Perceptron.perceptron(X_or, Y_or, 1000, 0.1, 1e6, "Sigmoid");
17 w_tanh_or = Perceptron.perceptron(X_or, Y_or, 1000, 0.1, 1e6, "Tanh");
18 - w_relu_or = Perceptron.perceptron(X_or, Y_or, 1000, 0.1, 1e6, "ReLU");
19
```

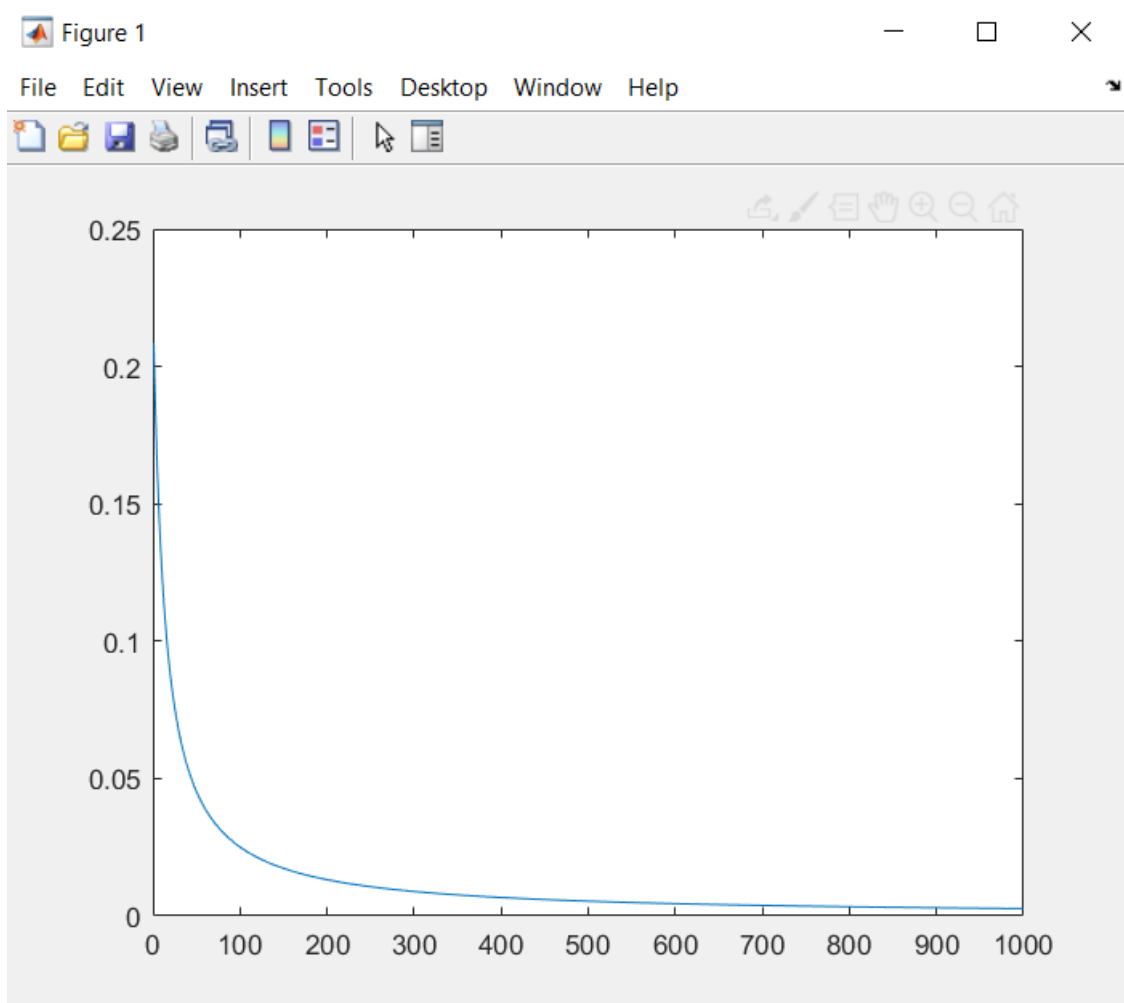
شکل 43) snippet مربوط به Or داخل کد

10-3-2- sigmoid سازی فعال سازی



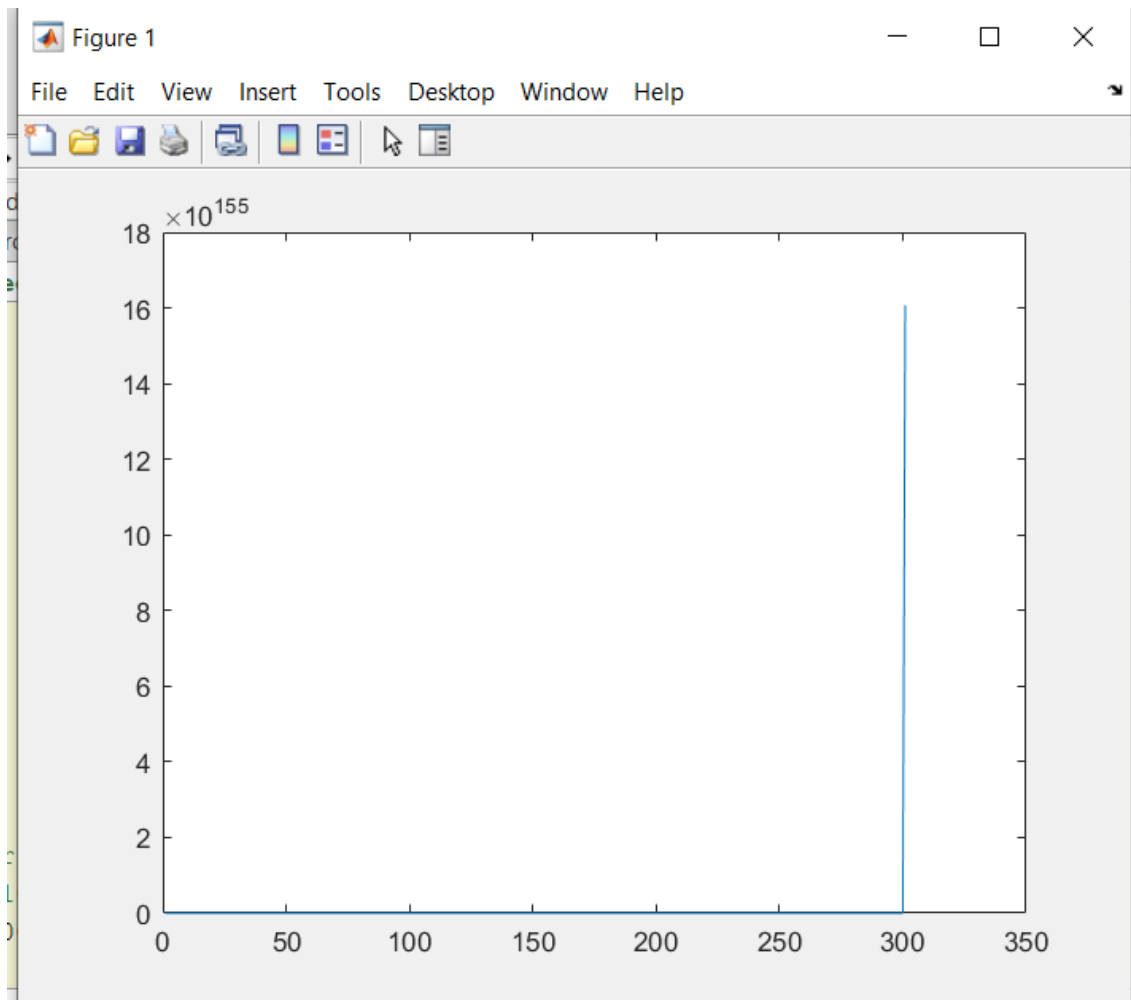
شکل 44) MSE با تابع فعال ساز sigmoid

11-3-2- Tanh تابع فعال سازی



شکل 45 MSE با تابع فعال ساز Tanh

2-3-12-تابع فعال ساز ReLU



شکل 46 MSE با تابع فعال ساز ReLU

با توجه به نمودار بالا تابع ReLU فعال ساز مناسبی برای گیت OR نیست.

2-3-13-پیاده سازی در Python

به طور کلی تمرکز بر روی Script داخل MATLAB می باشد.

```
# OR:

import numpy as np

def Rand(size):
    _Rand = np.random.rand(size)
    return _Rand

def Sigmoid(input_):
    _Sigmoid = (2 / (1 + np.exp(-input_))) - 1
    return _Sigmoid

def O(feature, weight):
    _O = Sigmoid(np.dot(feature, weight))
    return _O

x = [[0, 0], [0, 1], [1, 0], [1, 1]]
y = [[0], [1], [1], [1]]

w = Rand(2)
learning_rate = 0.1
E_max = 1e6
E = 0
epoch = 20

elst = []
for i in range(20):
    E = 0
    for j in range(4):
        o = O(x[j], w)
        w += (0.5 * learning_rate * (y[j] - o) * (1 - o ** 2) * x[j])
        E += (0.5 * (y[j] - o) ** 2)
        if(E > E_max):
            break
    elst.append(E)
```

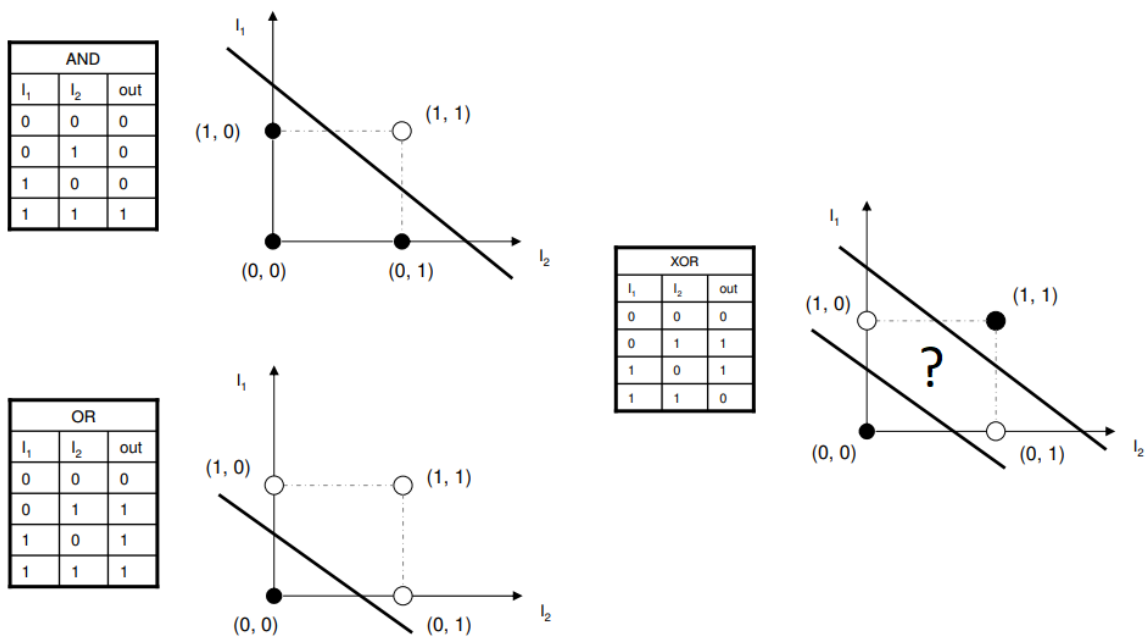
[4]

شکل 47) پیاده سازی Or در Python، شایان ذکر است که به علت تکرار مفاهیم، از توضیح بیشتر پرهیز شده است.

9-2- پیاده سازی XOR

در پیاده سازی XOR توسط Perceptron به چالش جدیدی بر می خوریم.

همانطور که در شکل زیر قابل مشاهده است، پس از تشکیل جدول درستی هر یک، و در نهایت رسم اعضای داخل ناحیه ها، ملاحظه می شود که:

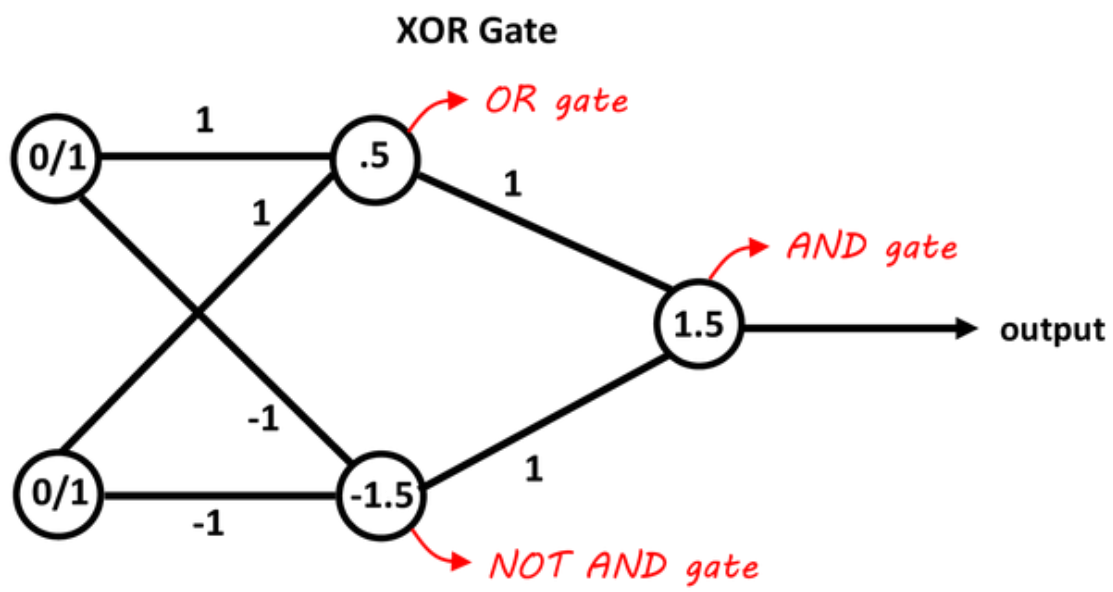


شکل 48) جدول درستی هر یکی از گیت ها و تشکیل نواحی اعضا

بنابراین یک Perceptron منفرد نمی تواند گیت XOR ما را جدا کند زیرا فقط می تواند یک خط مستقیم بکشد.

ترفند این است که متوجه شویم که می توانیم به طور منطقی دو Perceptron را روی هم قرار دهیم. دو Perceptron خطوط مستقیم را ترسیم می کنند و Perceptron دیگری که این دو سیگنال مجزا را در یک سیگنال واحد ترکیب می کند که فقط باید بین یک مرز صحیح / نادرست تفاوت قائل شود.

به طور شهودی تر، ما نمی توانیم از یک شبکه تک لایه استفاده کنیم و باید از تعداد لایه های بالاتر استفاده کنیم. همانند تمرین دوم، اگر بخواهیم تنها یک لایه Perceptron را به کار بگیریم، جداسازی داده ها در این حالت ناممکن است چرا که یک لایه ی Perceptron امکان جداسازی توابع غیرخطی را ندارد و مدل همگرا نمی شود. اما با به کارگیری چندین لایه، جداسازی این داده ها به سادگی قابل انجام است.



شکل 49) گیت XOR را می توان با ترکیب زیر از یک گیت NOT AND و یک گیت OR ایجاد کرد

دیتاست Iris

دیتاست Iris مخزن UCI شامل داده های مربوط به گیاهان است و هدف دسته بندی گونه ی گیاهی بر اساس داده ها ارائه شده می باشد. در این دیتاست اطلاعات ۳ گونه ی گیاهی موجود است و برای هر نمونه ۴ ویژگی ارائه شده است:

1. Sepal length in cm – طول گلبرگ
2. Sepal width in cm – عرض گلبرگ
3. Petal length in cm – طول کاسبرگ
4. Petal width in cm – عرض کاسبرگ

این دیتاست شامل ۳ گونه ی گیاهی است و مانند سوال دوم تمرین نمیتوان آن را با ۱ پرسپترون جداسازی کرد و برای دسته بندی داده ها به ۳ پرسپترون در یک لایه نیاز داریم (نیازی نیست شبکه ما چند لایه باشد). به همین علت به ۳ وکتور برای وزن نیاز داریم که در ادامه آن را توضیح خواهیم داد.

اولین نکته که باید ذکر کرد در این دیتاست مانند دیتاست های معمول در سطر اول فایل CSV. به نام گذاری ستون ها که همان feature های دیتای ماست نپرداخته و وقتی ما دیتاست را به عنوان ورودی میگیریم سطر اول که خود شامل دیتای ماست را به عنوان نام ستون ها در نظر میگیرد. برای حل این مشکل و از دست ندادن یک دیتا، ما میتوانیم ۲ کار انجام دهیم: راه حل اول به این صورت است که بصورت دستی دیتاست را تغییر دهیم و سطر اول آن را به نام گذاری feature بپردازیم اما چون دست بردن به دیتاست کار معقولی نیست از روش دوم استفاده کردیم. در این روش دیتا سطر اول را بصورت دستی به ابتدای دیتافریممان اضافه می کنیم که مراحل انجام این روش را در عکس پایین می بینیم.

```
In [27]: import pandas as pd
import numpy as np
import random
from sklearn import preprocessing

df = pd.read_csv("./assets/iris.data", sep=",")
df.columns = ['sepal length in cm', 'sepal width in cm', 'petal length in cm', 'petal width in cm', 'class']
# df set the first row of iris.data as the column's name so we'll miss a set of data
# we'll hardcode the first row of data
df.loc[-1] = [5.1, 3.5, 1.4, .2, 'Iris-setosa']
df.index = df.index + 1
df = df.sort_index()

x = df.iloc[:,4]
y = df.iloc[:,4]
```

شکل 50) تصویر مربوط به اضافه کردن دیتا سطر اول و گرفتن فیچرها و خروجی

در ادامه نیز توابع مورد نیاز که همان توابع استفاده شده در سوال مربوط به سرطان سینه و گیت ها می- باشد را اضافه می کنیم و با توجه به این که قبلا توضیحات مربوط به این توابع را دادیم از تکرار آنها خود داری می کنیم.

```
In [28]: # Functions

def Rand(size):
    _Rand = np.random.rand(size)
    return _Rand

def Sigmoid(input_):
    _Sigmoid = (2 / (1 + np.exp(-input_))) - 1
    return _sigmoid

def tanh(_input):
    return np.tanh(x)

def ReLU(_input):
    if _input < 0:
        return 0
    else:
        return 0.01 * _input

def O(feature, weight, fcn):
    if fcn == 'sigmoid':
        return Sigmoid(np.dot(feature, weight))
    elif fcn == 'tanh':
        return tanh(np.dot(feature, weight))
    elif fcn == 'ReLU':
        return ReLU(np.dot(feature, weight))
    else:
        return -1
```

شکل 51) توابع مورد نیاز نوروں ها

در ادامه به پارامترها و نرمال کردن دیتاها می پردازیم.

```
In [29]: # Hyper_params

# w = Rand(x.shape[1])
W = [Rand(x.shape[1] + 1) for i in range(3)] # +1 for bias
w = np.asarray(W)
learning_rate = 0.1
E_max = 1e6
epoch = 20
```

```
In [30]: # normalizing data:

scaler = preprocessing.MinMaxScaler()
df = pd.DataFrame(x)
names = df.columns
d = scaler.fit_transform(df)
scaled_df = pd.DataFrame(d, columns=df.columns)
x = scaled_df

# adding bias:
x['bias'] = -1

x.shape[0]
```

Out[30]: 150

شکل 52) تعیین پارامترها و نرمال کردن دیتاست

نکته ای که در رابطه به تعیین پارامترها مهم است و قبلا هم اشاره کردیم، در این دیتاست برای آموزش دیتاست به ۳ نوروں نیاز داریم که نتیجه میشود به ۳ وکتور وزن نیاز داریم. همانطور که سلول اول تصویر بالا مشخص است ما وزن را در یک لیست میریزن که این لیست یک ماتریس ۳*۵ است.

۳ سطر برای اینکه ما ۳ نورون داریم و ۵ ستون به این علت که داده های ما دارای ۴ نوع دیتای مختلف است که در بالا آن ها را توضیح دادیم و یک ستون برای بایاس نیز در ادامه به آن اضافه میکنیم پس هر نورون ما نیاز به ۵ وزن دارد که تشکیل یک ماتریس 5×3 می دهند، در ادامه نیز این لیست را یک npArray تبدیل میکنیم.

برای نرمال کردن داده هایمان نیز مانند گذشته از تابع Processing در کتابخانه sklearn استفاده می کنیم که مراحل نرمال سازی آن در تصویر بالا مشخص است و در انتها به دیتا های فیچرمان ستون بایاس را نیز اضافه می کنیم.

برای آموزش شبکه مان نیاز به ۳ حلقه داریم. حلقه اول مربوط به تعداد iteration هایمان است و در هر تکرار حلقه مقدار خطا را باید صفر کنیم. حلقه دوم مربوط به تعداد دیتا های دیتاستمان است که برابر با ۱۵۰ دیتا است، چون تعداد دیتا ها کم است همه آن را برای آموزش استفاده میکنیم، این حلقه همانطور که قبلا توضیح داده شده برای این است که برای هر دیتا از دیتاستمان باید وزن ها و خطا را آپدیت کنیم و حلقه آخر نیز برای نورون ها است که از نورون اول شروع میکنیم دیتا هارا به نورون میدهیم آن را آموزش می دهیم و خطا را برای آن حساب میکنیم بعد به سراغ نورون بعدی می رویم و همان کار هارا برای آن تکرار می کنیم.

نحوه عملکرد حلقه سوم بدین صورت است که ما چک می کنیم که آیا لیبل آن دسته ای که در آن قرار داریم ($y[i]$) با کلاس نورونی که برای هریک از گونه های گل قرار دادیم برابر است یا خیر و در این صورت متغیر فلگ را برابر ۰ یا ۱ قرار می دهیم. به عنوان مثال اگر در نورون مربوط به گونه versicolour قرار داشته باشیم و دیتایی که در حال بررسی آن هستیم لیبل setosa خورده باشید فلگ برابر ۰ می شود اما اگر در دیتا های مربوط به همان نوع گل versicolour قرار داشته باشیم فلگ برابر ۱ می شود. از این فلگ در محاسبه خطا و آپدیت کردن وزن ها استفاده می شود که در کد مشخص است.

```
In [36]: # Training:
e_lst_learn = []
_class = ["Iris-setosa", "Iris-versicolour", "Iris-virginica"]
for j in range(epoch):
    E = 0
    for i in range(len(x)):
        for z in range(3):
            label = y[i]
            if label == _class[z]:
                flag = 1
            else:
                flag = 0

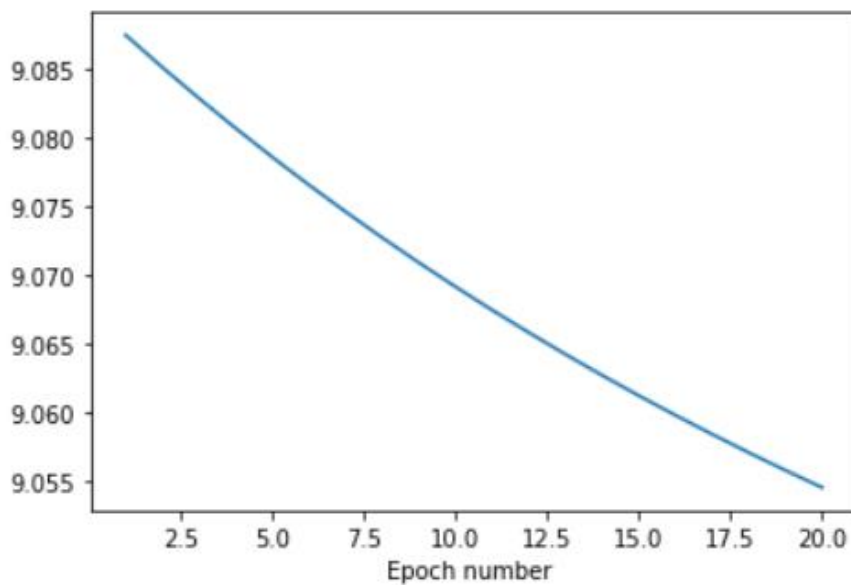
            o = 0(x.iloc[i], w[z], 'sigmoid')
            w[z] += (0.5 * learning_rate * (flag - o) * (1 - o ** 2) * x.iloc[i])
            E += (0.5 * (flag - o) ** 2)

        # print(o)
        # print(E)
        # print(w)
    e_lst_learn.append(E)

import matplotlib.pyplot as plt
iteration = np.arange(1, epoch+1, 1)
plt.plot(iteration, e_lst_learn)
plt.xlabel('Epoch number')
```

شکل 53) نحوه آموزش شبکه برای دیتاست Iris

در نهایت هم عملکرد مربوط این شبکه را مشاهده میکنیم که البته هرچه تعداد تکرارمان بیشتر باشد عملکرد مطلوب تر می شود.



شکل 54) ارور بر حسب تکرار

منابع و مواخذ

- https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_breast_cancer.html
- https://github.com/scikit-learn/scikit-learn/blob/36958fb240fbe435673a9e3c52e769f01f36bec0/sklearn/datasets/data/breast_cancer.csv
- [https://www.simplilearn.com/tutorials/deep-learning-tutorial/perceptron#:~:text=A%20Perceptron%20is%20a%20neural,value%20%E2%80%9Df\(x\).](https://www.simplilearn.com/tutorials/deep-learning-tutorial/perceptron#:~:text=A%20Perceptron%20is%20a%20neural,value%20%E2%80%9Df(x).)
- [https://en.wikipedia.org/wiki/Rectifier_\(neural_networks\)](https://en.wikipedia.org/wiki/Rectifier_(neural_networks))
- <https://www.geeksforgeeks.org/implementation-of-perceptron-algorithm-for-xor-logic-gate-with-2-bit-binary-input/>
- <https://flipdazed.github.io/blog/python%20tutorial/introduction-to-neural-networks-in-python-using-XOR>