

در این نوت بوک به تمامی پرسش‌های تمرین عملی سری ۶ پاسخ داده شده است. همچنین نمودارها و گزارش‌های مربوطه بعد از هر بخش آورده شده و پاسخ‌های مورد نیاز به سوالات دستور کار در همین نوت بوک پاسخ داده شده است. برای اجرای کدها، کافیهست از اولین نوت بوک تا آخرین نوت بوک را به ترتیب اجرا کنید تا همه نتایج بدون هیچ مشکلی مجدداً تولید شوند. فایل pdfای که به همراه این نوت بوک فرستاده شده است، حاوی نکته اضافه‌ای نیست و صرفاً تبدیل شده این نوت بوک است لذا اگر در تصحیح فایل نوت بوک مشکلی وجود ندارد، توصیه می‌شود فایل pdf را نادیده بگیرید.

به منظور افزایش خوانایی و سادگی فهم کد، سعی شده است تا جای ممکن برای هر عملیات یک تابع تعریف شود و این تابع در پرسش‌های دیگر بازخوانی می‌شوند. مثلاً حلقه آموزش شبکه که روی epoch ها تعریف می‌شود، با نام train_multiple_epochs در بخش اول یک بار تعریف شده و در بخش‌های دیگر چندین بار فراخوانی می‌شود. ضمناً عملکرد هر تابع به صورت comment در بالای تعریف هر تابع نوشته شده است.

1 Q1 - Single layer network

1.1 generating dataset

```
In [1]: 1 import numpy as np
2
3 N = 100
4 x_min , x_max = -2, 2
5
6 # samples in the range -2 , 2
7 xs = (x_max - x_min) * np.random.random(size = N) + x_min
8
9 assert xs.max() < x_max and x_min < xs.min()
```

```
In [2]: 1 # 1-D not function
2 def NOT(x):
3     out = x.copy()
4     out[x >= 0] = 0.
5     out[x < 0] = 1.
6     return out
7 y_stars = NOT (xs)
8
9
```

1.2 Implementing one epoch

از توضیحات کتاب می‌دانیم که رابطه آپدیت پارامترها (params) برای وزن‌های لایه آخر به صورت زیر تعریف می‌شود:

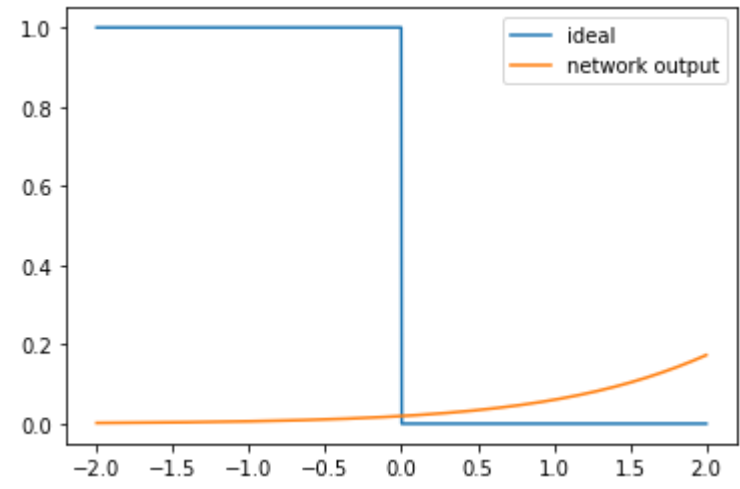
$$\Delta w_u^{(l)} = -\frac{\eta}{2} \nabla_{w_u} e_u^{(l)} = \eta \left(o_u^{(l)} - out_u^{(l)} \right) out_u^{(l)} \left(1 - out_u^{(l)} \right) in_u^{(l)}$$

```
In [3]: 1 # sigmoid function
2 def sigmoid(x):
3     return 1/(1+np.exp (-x))
4
5 # computing weight updates given a single data pair and network parameters
6 def get_weight_update_for_one_data(x,y_star , eta, params):
7
8     y,_,_ = get_network_output (x, params)
9
10    x_new = [x, -1]
11
12    delta_params = []
13    for i,p in enumerate (params):
14        delta_params.append (- eta * (y - y_star) * y * (1 - y) * x_new[i])
15
16    return delta_params
17
18 # computing network output given input data and params (both in batch mode and online mode)
19 def get_network_output (x, params):
20     w, theta = params
21     return sigmoid (w*x - theta), _ , _ # two extra parameters are added for consistency with 2 Layer perceptron
22
23 # computing total network error given the dataset and the network parameters
24 def get_total_error(xs, y_stars, params):
25
26     y, _ , _ = get_network_output(xs, params)
27     return ((y - y_stars) ** 2).sum()
28
29 # performing online learning on all datas for one epoch and updating params
30 def train_one_epoch(xs, y_stars, eta, params_init, ):
31     params = params_init.copy()
32
33     for x, y_star in zip (xs , y_stars):
34         error = get_total_error(xs, y_stars, params)
35
36         delta_params = get_weight_update_for_one_data(x , y_star, eta , params)
37
38         for i, delta_p in enumerate (delta_params):
39             params[i] = params[i] + delta_p
40
41     return error, params
42
43
44 eta = 1
45 w0 = theta0 = 3
46 params_init = [w0, theta0]
47
48 error, params = train_one_epoch(xs, y_stars, eta, params_init)
49
50 print ("Error after one epoch is : " , error)
```

Error after one epoch is : 51.59679952737811

همانطور که دیده می‌شود، با انجام صرفاً یک epoch، بهبود جدی حاصل نمی‌شود و خطا هنوز نسبتاً زیاد است

```
In [5]: 1 import matplotlib.pyplot as plt
2
3 # plotting the target funtion and the estimated function
4 def plot_in_range(params):
5
6     xs = np.linspace(start = x_min , stop = x_max, num=1000)
7     ys , _ , _ = get_network_output(xs, params)
8     y_stars = NOT (xs)
9     plt.plot (xs, y_stars, label = 'ideal')
10    plt.plot (xs, ys, label = 'network output')
11
12    plt.legend()
13    plt.show()
14
15 plot_in_range (params)
```



در شکل فوق، تابع هدف و رفتار فعلی شبکه با هم مقایسه شده اند. مطابق انتظارات، مشاهده می‌شود که با تنها انجام یک epoch رفتار تابع به درستی یادگرفته نشده است. در حالت ایده آل انتظار داریم که خروجی شبکه به طور کامل به تابع not نشان داده شده نزدیک شود و رفتار مشابهی از خود نشان دهد.

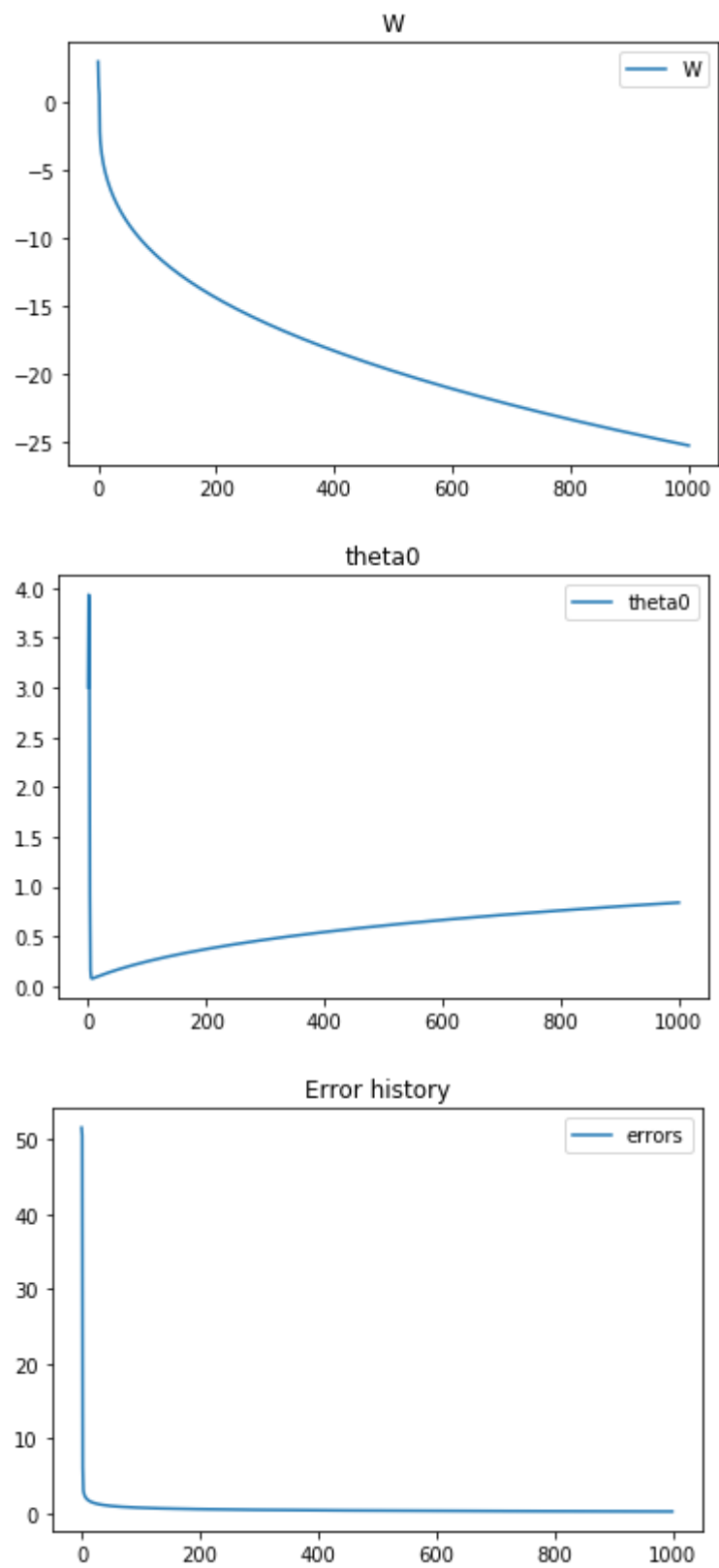
1.3 Training for 100 epochs

In [6]:

```
1 # Training for 'epochs' iterations given the initial parameters
2 def train_multiple_epochs (eta, epochs, params_init):
3
4     params_history = [[] for t in params_init]
5
6     errors = []
7     params = params_init.copy()
8
9     for epoch in range (epochs):
10         for t in range(len (params_history)):
11             params_history[t].append (params[t])
12
13         error, params = train_one_epoch(xs, y_stars, eta, params)
14         errors.append (error)
15
16
17     for t in range(len (params_history)):
18         params_history[t].append (params[t])
19
20
21     final_params = [ps[-1] for ps in params_history]
22
23     return errors, params_history, final_params
24
25 # plotting the history of wieghts and error during training
26 def plot_epochs (epochs, errors, params_history, params_names):
27     for parameter_history, parameter_name in zip (params_history, params_names):
28         plt.plot (parameter_history , label = parameter_name)
29         plt.title(parameter_name)
30         plt.legend()
31         plt.show ()
32
33     plt.plot (errors , label = "errors")
34     plt.title("Error history")
35     plt.legend()
36     plt.show ()
```

In [7]:

```
1 epochs = 1000
2 eta = 1
3 w0 = theta0 = 3
4 params_init = [w0, theta0]
5
6 errors, params_history, final_params = train_multiple_epochs (eta, epochs, params_init)
7
8 plot_epochs(epochs, errors, params_history = params_history, params_names = ['W' , 'theta0'])
```



همانطور که در نمودارهای فوق دیده می‌شود، با آموزش شبکه خطا به سرعت به صفر میل کرده است و این یعنی شبکه توانسته به خوبی رفتار داده‌ها را یاد بگیرد. در مورد مقدار آستانه می‌بینیم که این مقدار به سمت صفر میل کرده است. این رفتار مورد انتظار ما بود چرا که در داده‌های آموزش، صفر نقطه‌ای است که به عنوان آستانه مقدار خروجی را تغییر می‌دهد. در مورد وزن W باید گفت که این وزن ظاهراً به مقدار خاصی همگرا نمی‌شود بلکه با جلو رفتن در امر آموزش، دائماً مقدار آن منفی و منفی‌تر می‌شود. این مسئله را از چند زاویه می‌توان بررسی کرد. اول این که هر چه W منفی‌تر شود تابع سیگموید به یک تابع تیز نزدیک‌تر می‌شود و لذا رفتاری شبیه تابع پله را از خود نشان می‌دهد. (این مسئله را در بحث RL از اسلایدهای درس نیز دیده بودیم) لذا مطلوب‌ترین مقدار وزن برای صفر شدن خطا مقدار منفی بی‌نهایت است. البته مقدار وزن در اوایل آموزش با سرعت و شیب زیادی به سمت منفی شدن پیش می‌رود اما به مرور زمان شیب آن کم می‌شود. علت این مسئله را می‌توان در اشباع شدن تابع سیگموید جستجو کرد. در واقع وقتی تابع سیگموید به یک تابع پله شبیه می‌شود، مشتق آن در نواحی صاف، صفر خواهد بود و این مشتق صفر جلوی آپدیت شدن وزن‌ها با سرعت زیاد را می‌گیرند.

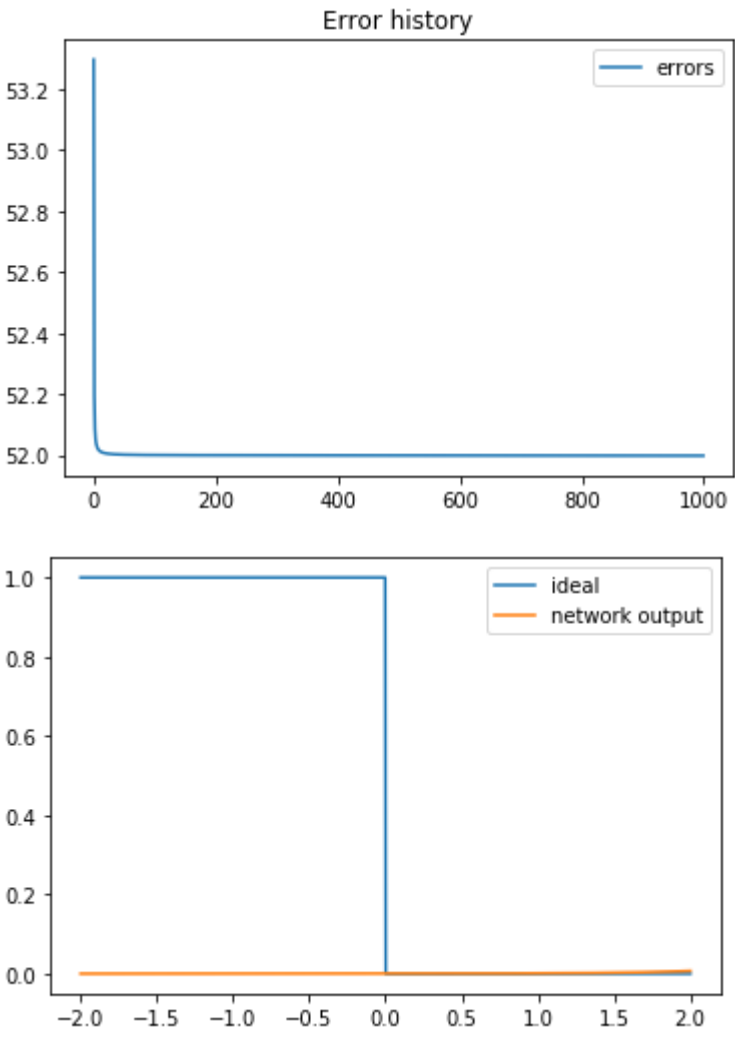
```
In [8]: 1 plot_in_range (final_params)
```



همانطور که در شکل فوق دیده می‌شود، حالا رفتار شبکه به رفتار ایده‌آل خیلی نزدیک شده است و این یعنی آموزش موفقیت آمیز بوده است.

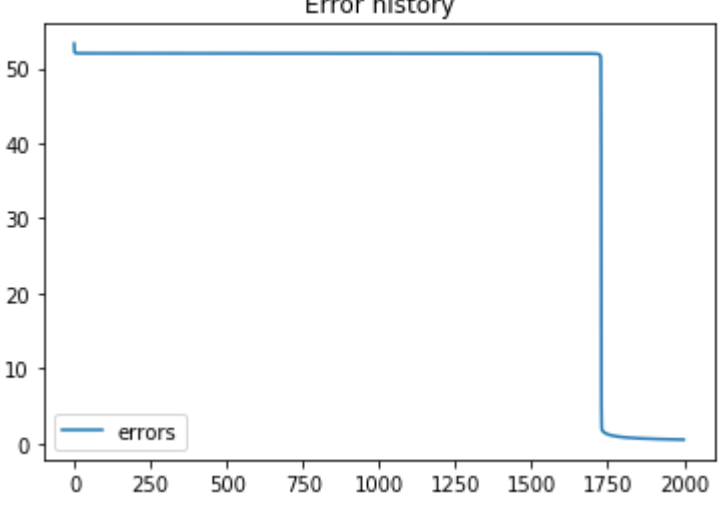
1.4 finding k*

```
In [9]: 1 epochs = 1000
2
3 k = 7
4 w0 = theta0 = k
5 params_init = [w0 , theta0]
6
7 errors, params_history, final_params = train_multiple_epochs (eta, epochs, params_init)
8 plot_epochs(epochs, errors, params_history = [], params_names = [])
9 plot_in_range (final_params)
```



به نظر می‌رسد نقطه $k = 7$ نقطه‌ای است که باعث می‌شود خطای آموزش بعد از ۱۰۰۰ اپیاک به صفر نرسد و در مقدار ۵۰ ثابت بماند. همچنین تابع شبکه هنوز با تابع هدف تفاوت جدی دارد.

```
In [10]: 1 epochs = 2000
2
3 k = 7
4 w0 = theta0 = k
5 params_init = [w0 , theta0]
6
7 errors, params_history, final_params = train_multiple_epochs (eta, epochs, params_init)
8 plot_epochs(epochs, errors, params_history = [], params_names = [])
9 plot_in_range (final_params)
```



با زیاد کردن تعداد ایپاک‌های آموزش، خطا مجدداً به صفر رسیده‌است و رفتار مورد نظر به خوبی یاد گرفته شده است.

2 Q2 - Double input

در این بخش به پرسش ۲ پاسخ داده شده است. توجه کنید که کدهای این قسمت نمی‌توانند مستقل از قسمت‌های فوق اجرا شوند چرا که برخی توابعی که در این بخش استفاده شده است را در بالا تعریف کرده‌ایم. بنابراین توصیه می‌شود کل نوت بوک را به ترتیب اجرا کنید.

2.1 generating dataset

```
In [11]: 1 import numpy as np
2
3 N = 100
4 x_min , x_max = -2, 2
5
6 # samples in the range -2 , 2
7 xs = (x_max - x_min) * np.random.random(size = [N,2]) + x_min
8
9 assert xs.max() < x_max and x_min < xs.min()
```

```
In [12]: 1 def f1(x):
2     out = np.zeros(x.shape[0])
3     out[x[:,0] >= 0] = 1.
4     out[x[:,0] < 0] = 0.
5
6     return out
7
8 y_stars = f1 (xs)
9
10
```

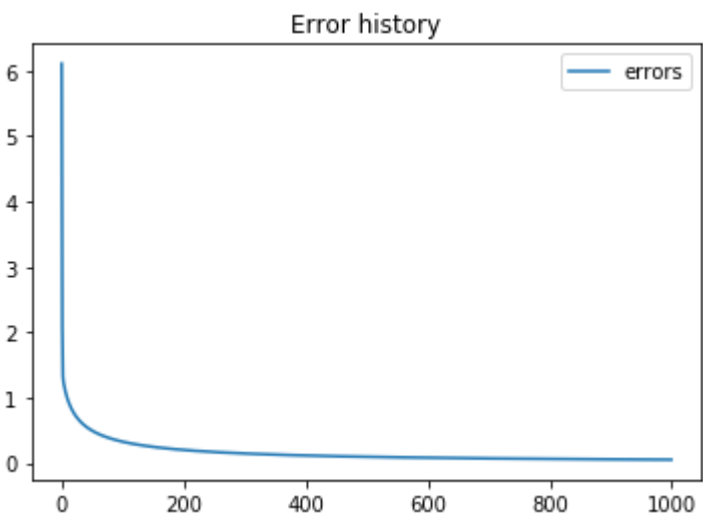
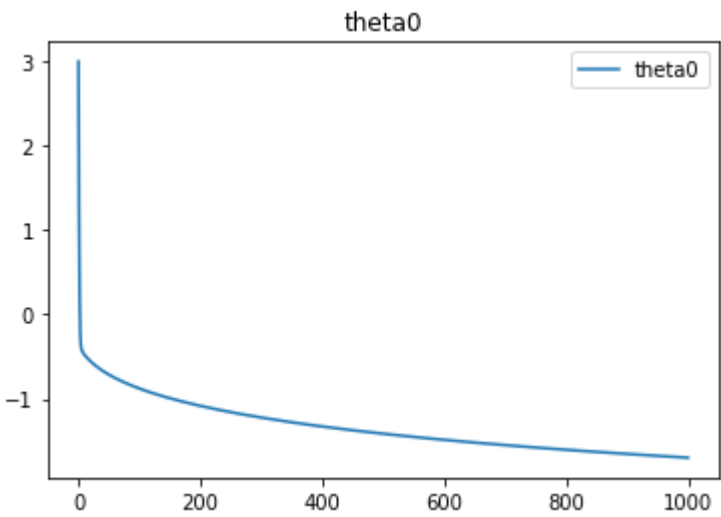
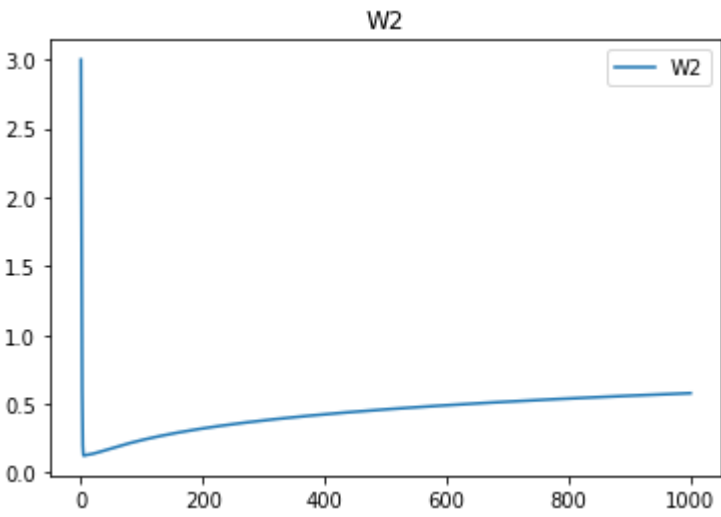
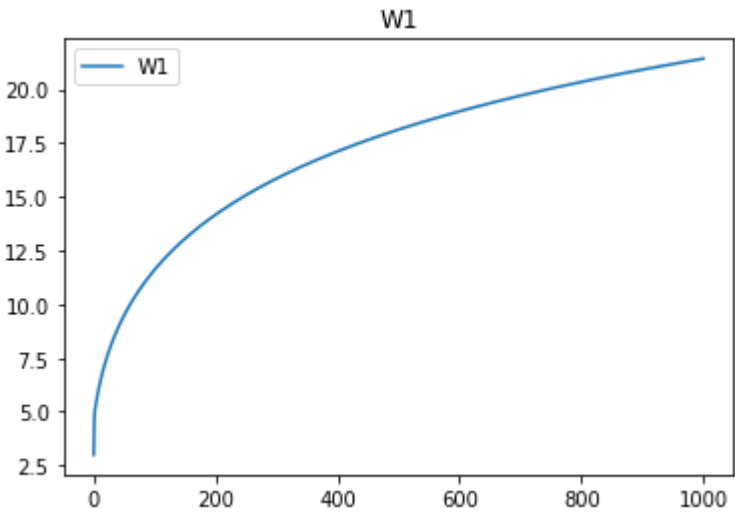
2.2 train loop

```
In [13]: 1 def sigmoid(x):
2     return 1/(1+np.exp (-x))
3
4 def get_weight_update_for_one_data(x, y_star, eta, params):
5
6     y , _ , _ = get_network_output (x, params)
7
8     x_new = [x[0], x[1] , -1]
9
10    delta_params = []
11    for i,p in enumerate (params):
12        delta_params.append (- eta * (y - y_star) * y * (1 - y) * x_new[i])
13
14    return delta_params
15
16 def get_network_output (x, params):
17     w1, w2, theta = params
18     if x.ndim == 2: #batch mode calculations
19         return sigmoid (w1*x[:,0] + w2*x[:,1] - theta) , _ , _
20     else:
21         return sigmoid (w1*x[0] + w2*x[1] - theta) , _ , _
22
23
24 eta = 1
25 w01 = w02 = theta0 = 3
26 params_init = [w01, w02, theta0]
27 error, params = train_one_epoch(xs, y_stars, eta, params_init)
28
29 print ("Error after one epoch is : " , error)
```

Error after one epoch is : 6.110533743656936

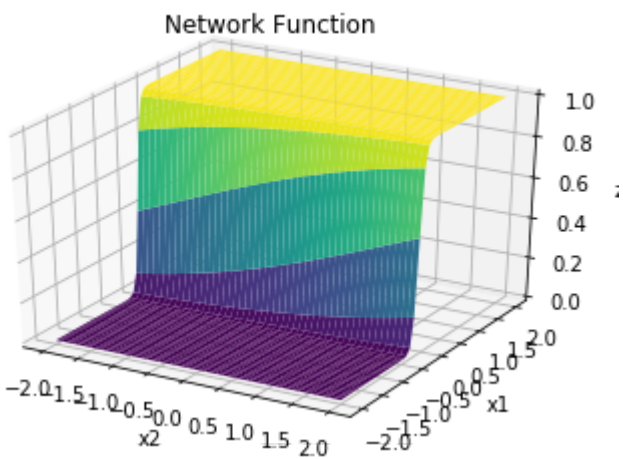
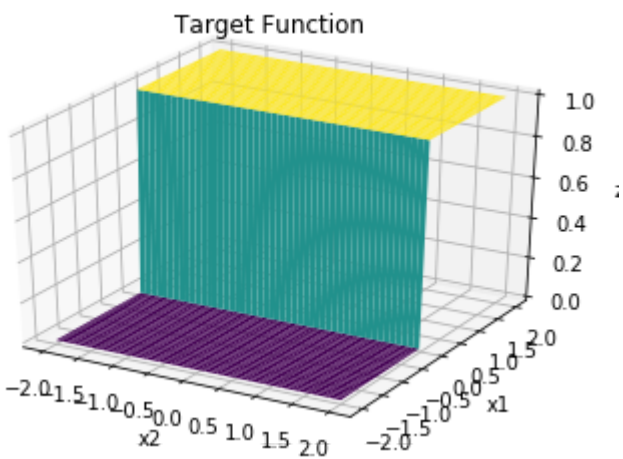
```
In [14]: 1 import matplotlib.pyplot as plt
2 from mpl_toolkits import mplot3d
3
4 def plot_in_range(params):
5
6     x1 = np.linspace(x_min, x_max, 50)
7     x2 = np.linspace(x_min, x_max, 50)
8
9     X1, X2 = np.meshgrid(x1, x2)
10
11    Z = X1.copy()
12    for i in range (len(x1)):
13        for j in range (len (x2)):
14            Z[i,j] = f1(np.array ([x1[i],x2[j]]))(None,:)
15
16    ax = plt.axes(projection='3d')
17    ax.plot_surface(X1, X2, Z, rstride=1, cstride=1,cmap='viridis', edgecolor='none')
18
19    ax.set_title ("Target Function")
20    ax.set_xlabel ("x2")
21    ax.set_ylabel ("x1")
22    ax.set_zlabel ("z")
23
24    plt.show()
25
26    Z = X1.copy()
27    for i in range (len(x1)):
28        for j in range (len (x2)):
29            Z[i,j], _ , _ = get_network_output(np.array ([x1[i],x2[j]]), params)
30
31    ax = plt.axes(projection='3d')
32    ax.plot_surface(X1, X2, Z, rstride=1, cstride=1,cmap='viridis', edgecolor='none')
33    ax.set_title ("Network Function")
34    ax.set_xlabel ("x2")
35    ax.set_ylabel ("x1")
36    ax.set_zlabel ("z")
37
38    plt.show()
39
40
41 # plot_in_range (params)
```

```
In [15]: 1 epochs = 1000
2 eta = 1
3 w01 = w02 = theta0 = 3
4 params_init = [w01, w02, theta0]
5
6 errors, params_history, final_params = train_multiple_epochs (eta, epochs, params_init)
7
8 plot_epochs(epochs, errors, params_history = params_history, params_names = ['W1' , 'W2' , 'theta0'])
```



همانطور که دیده می‌شود، در نمودار فوق مقدار خطا بازهم به سمت صفر میل کرده و آموزش شبکه با موفقیت صورت گرفته است. در شکل زیر رفتار مورد انتظار و رفتار یادگرفته شده توسط شبکه رسم شده‌اند که مشاهده می‌شود این دو خیلی به یکدیگر شباهت دارند.

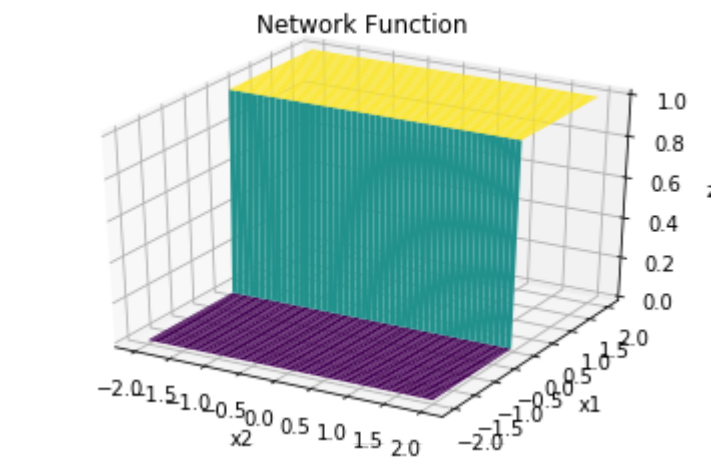
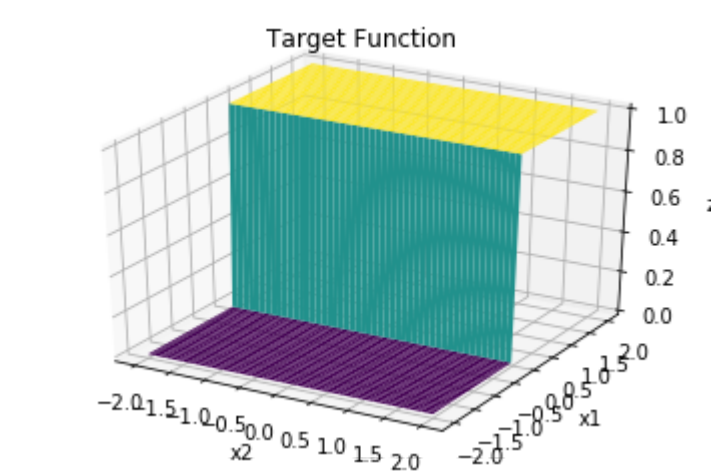
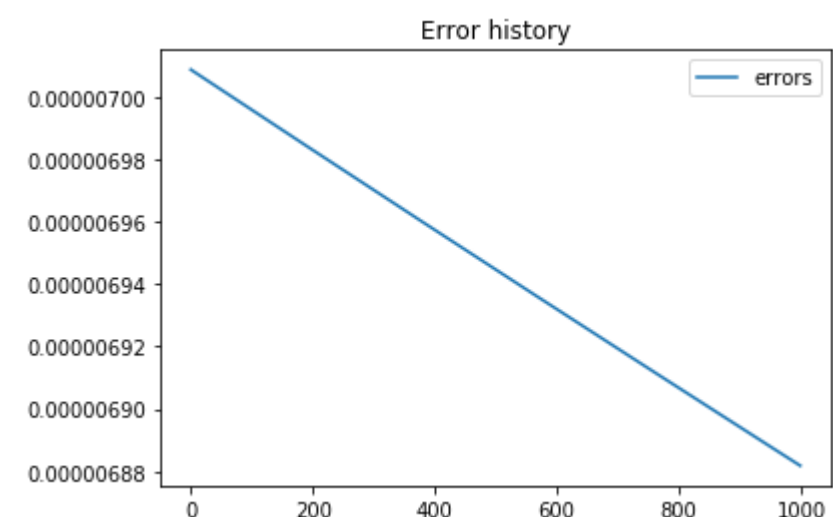
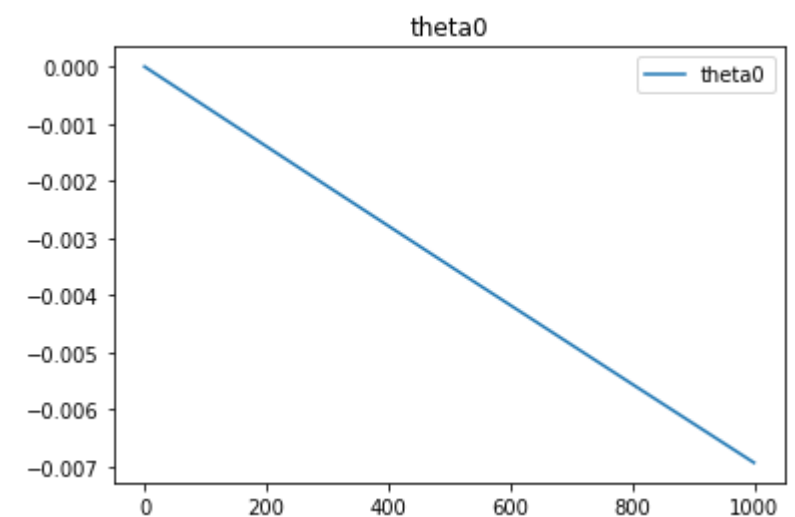
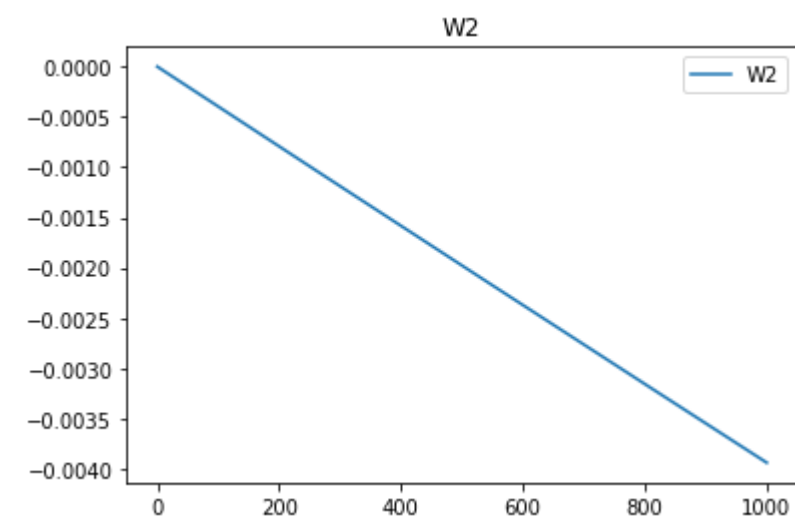
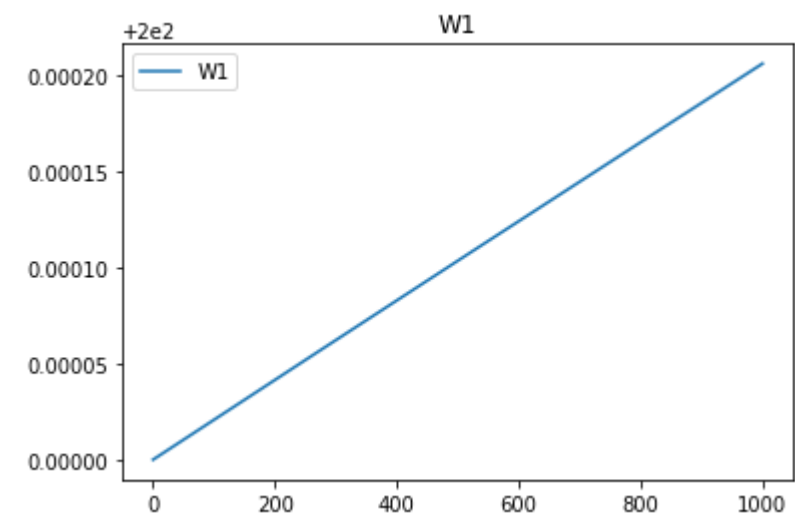
```
In [16]: 1 plot_in_range (final_params)
```



2.3 setting the initial conditions

In [17]:

```
1 epochs = 1000
2 eta = 1
3 w01 = +200
4 w02 = 0
5 theta0 = 0
6
7 params_init = [w01, w02, theta0]
8
9 errors, params_history, final_params = train_multiple_epochs (eta, epochs, params_init)
10
11 plot_epochs(epochs, errors, params_history = params_history, params_names = ['W1' , 'W2' , 'theta0'])
12
13 plot_in_range (final_params)
```



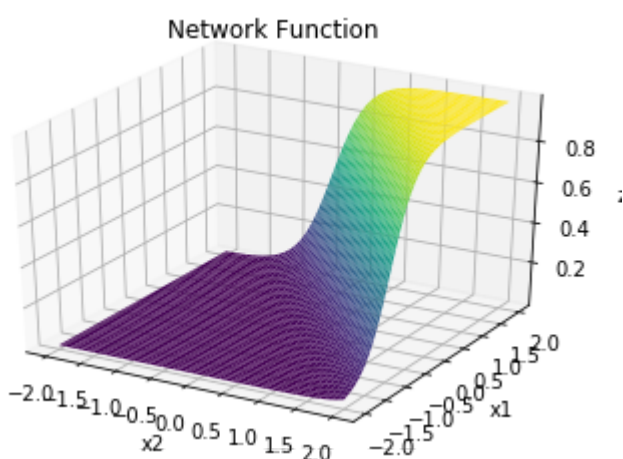
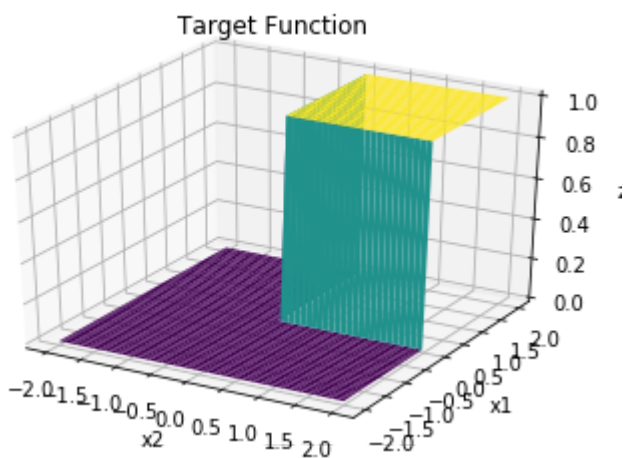
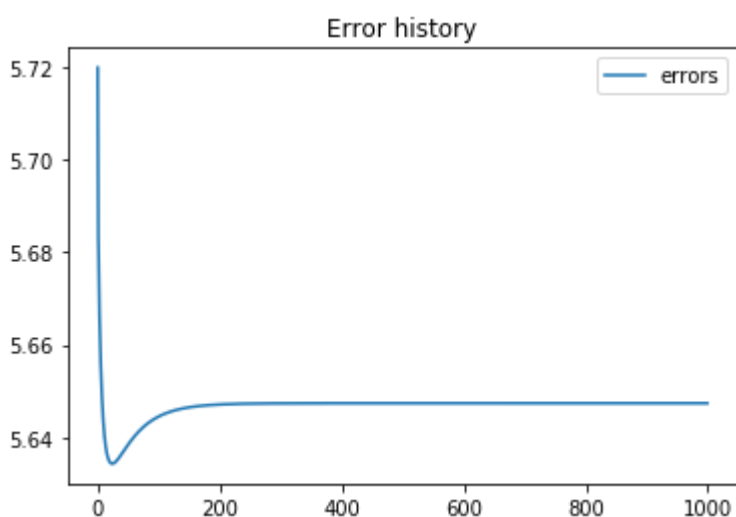
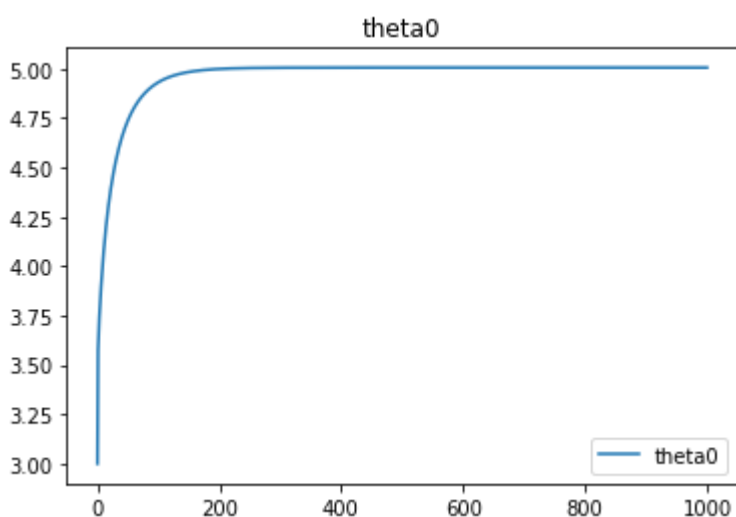
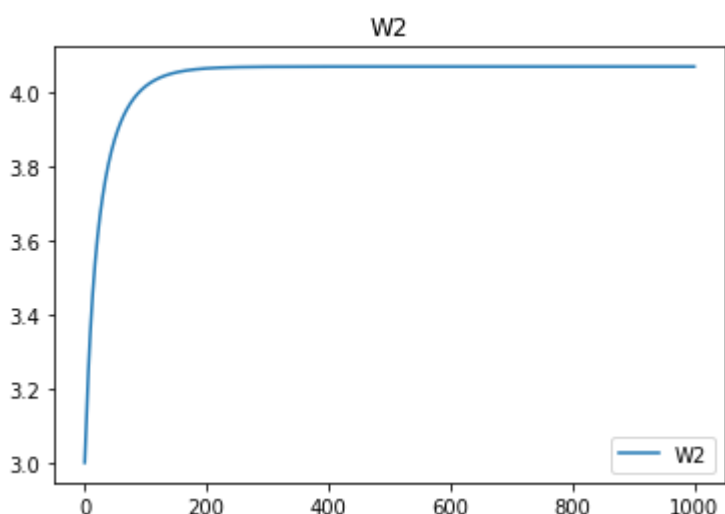
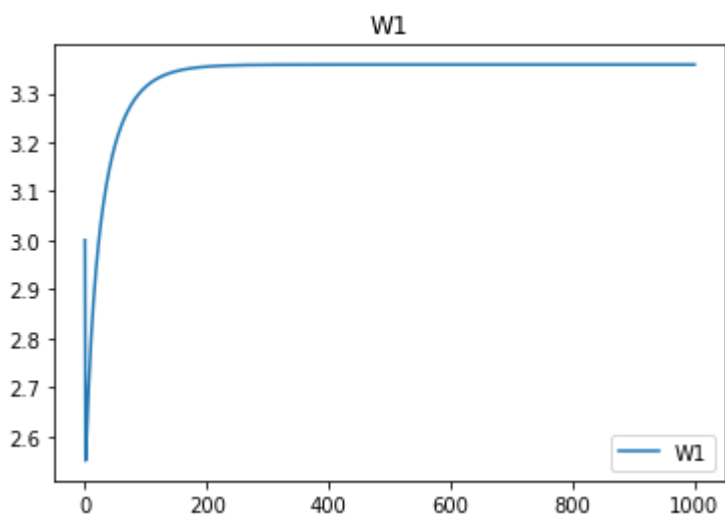
با توجه به این که تابع هدف باید از متغیر x_2 مستقل باشد، مقدار w_2 را صفر می‌گذاریم و انتظار داریم که مقدار w_2 در طول آموزش تغییری نداشته باشد. همانطور که مشاهده می‌شود، این مقدار تغییرات خیلی کمی داشته است که آن هم به نظر من به دلیل استفاده از رویکرد online learning است.

همچنین با توجه به این که در این سوال به دنبال یادگیری تابع پله روی متغیر اول هستیم، به نظر می‌رسد که بهتر است وزن w_1 را یک مقدار مثبت بزرگ قرار داده و مقدار آستانه را هم صفر در نظر بگیریم. همانطور که دیده می‌شود، با اعمال این شرایط اولیه مقدار اولیه خطا بسیار کم بوده و در نتیجه خیلی سریعتر به خطای کوچک رسیده‌ایم. همچنین نمودارهای خروجی رفتار شبکه، در نقطه آستانه شیب خیلی تندتری دارند و رفتار مورد نظر را خیلی بهتر یادگرفته‌اند

2.4 AND function


```
In [18]: 1 def AND(x):
2         out = np.zeros(x.shape[0])
3         out[(x[:,0] >= 0) & (x[:,1] >= 0)] = 1.
4         out[(x[:,0] < 0) | (x[:,1] < 0)] = 0.
5
6         return out
7
8     f1 = AND
9     y_stars = f1 (xs)
```

```
In [19]: 1 epochs = 1000
2     eta = 1
3     w01 = w02 = theta0 = 3
4     params_init = [w01, w02, theta0]
5
6     errors, params_history, final_params = train_multiple_epochs (eta, epochs, params_init)
7
8     plot_epochs(epochs, errors, params_history = params_history, params_names = ['W1' , 'W2' , 'theta0'])
9
10    plot_in_range (final_params)
```



به نظر می‌رسد در روند آموزش فوق، شبکه سعی کرده است تا جایی ممکن رفتار تابع AND را دنبال کند. اما همانطور که دیده می‌شود، به دلیل پایین بودن ظرفیت مدل، کم بودن عمق آن و همچنین کم بودن تعداد پارامترهای قابل آموزش، مدل نتوانسته است رفتار تابع را به طور ایده آل یادبگیرد و خطا را صفر کند. تغییر دادن تابع فعال‌سازی احتمالاً نمی‌تواند قدرت مدل را در این حالت ارتقا دهد. توجه کنید که به دلیل ساختار مدل، تابع فعال‌سازی روی

$$w_1x_1 + w_2x_2 - \theta$$

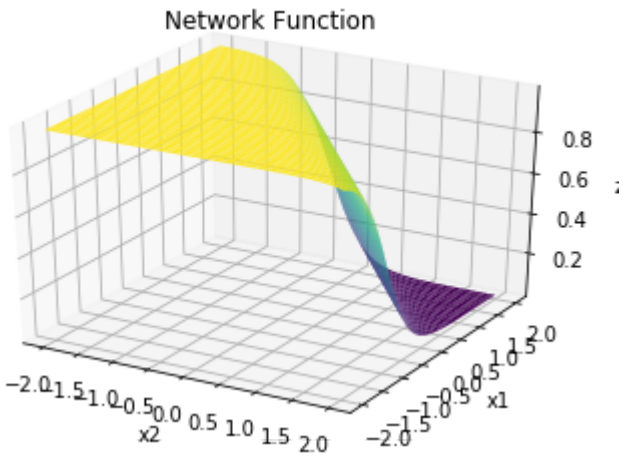
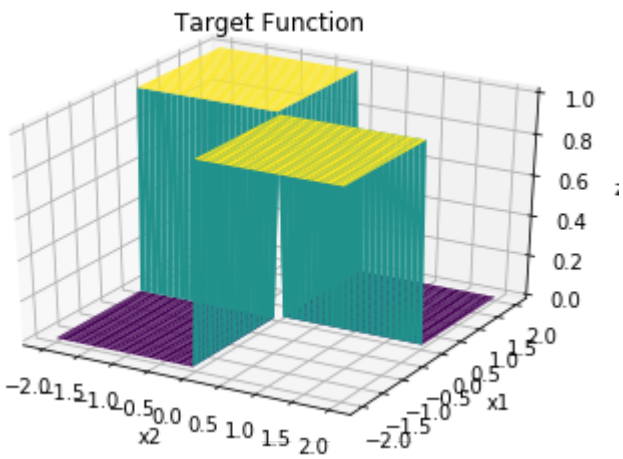
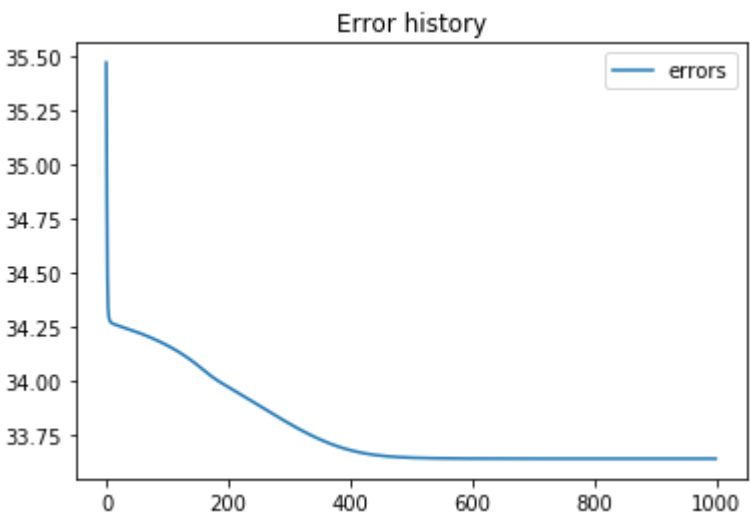
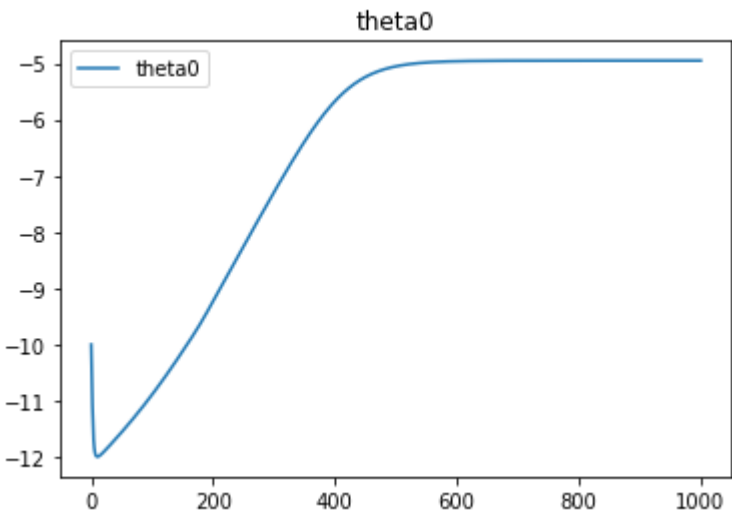
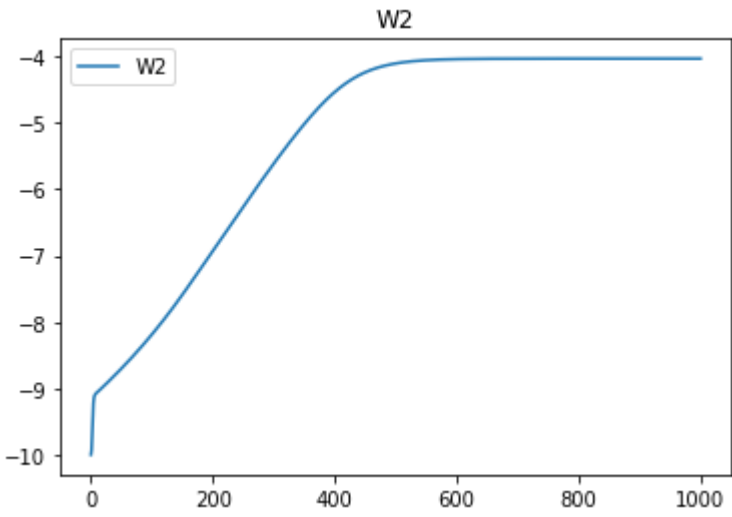
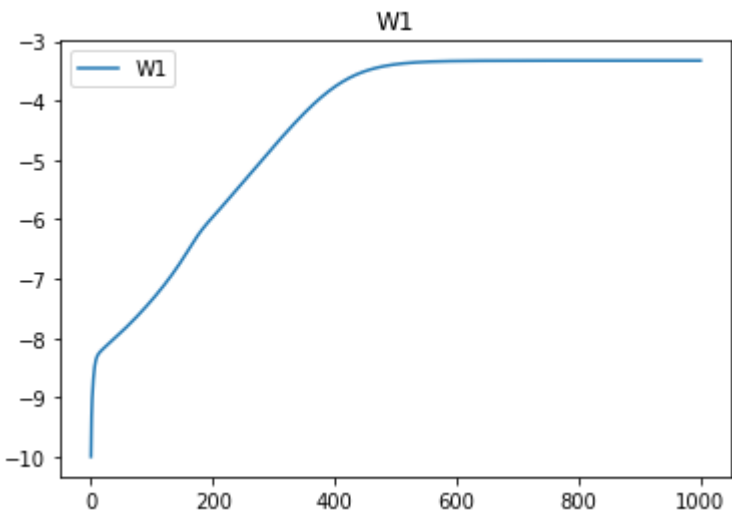
اعمال می‌شود. اعمال کردن هر نوع تابع فعالسازی روی این رابطه خطی، نهایتاً می‌تواند یک جداسازی خطی را به همراه داشته باشد. لذا به نظر می‌رسد با تغییر تابع فعالسازی هم قدرت جداسازی مدل از یک جداسازی صفحه خطی فراتر نخواهد رفت بلکه با عمیق کردن شبکه می‌توان انتظار داشت رفتارهای غیر خطی بهتر یادگرفته شوند .

2.5 XOR function

```
In [20]: 1 def XOR(x):
2         out = np.zeros(x.shape[0])
3         out[x[:,0]*x[:,1] <= 0] = 1.
4         out[x[:,0]*x[:,1] > 0] = 0.
5
6         return out
7
8 f1 = XOR
9 y_stars = f1 (xs)
```

In [21]:

```
1 epochs = 1000
2 eta = 1
3 w01 = w02 = theta0 = -10
4 params_init = [w01, w02, theta0]
5
6 errors, params_history, final_params = train_multiple_epochs (eta, epochs, params_init)
7
8 plot_epochs(epochs, errors, params_history = params_history, params_names = ['W1' , 'W2' , 'theta0'])
9
10 plot_in_range (final_params)
```



همانطور که دیده می‌شود، در این حالت مدل نتوانسته رفتار داده را اصلاً یاد بگیرد. در واقع همانطور که در بخش قبل نیز توضیح داده شد و همانطور که در اسلایدهای درس نیز توضیح داده شد، شبکه عصبی بدون لایه مخفی تنها می‌تواند یک جداسازی خطی انجام دهد اما تابع XOR یک رفتار غیرخطی دارد. در واقع بسته به این که نقطه شروع کجا باشد و شرایط اولیه پارامترها به چه صورتی باشد، مدل در یکی از بهینه‌های محلی گیر می‌کند و تنها می‌تواند برای دسته‌ای از داده‌ها پیش‌بینی درست را انجام دهد و برای بقیه داده‌ها رفتار اشتباهی را نشان می‌دهد.

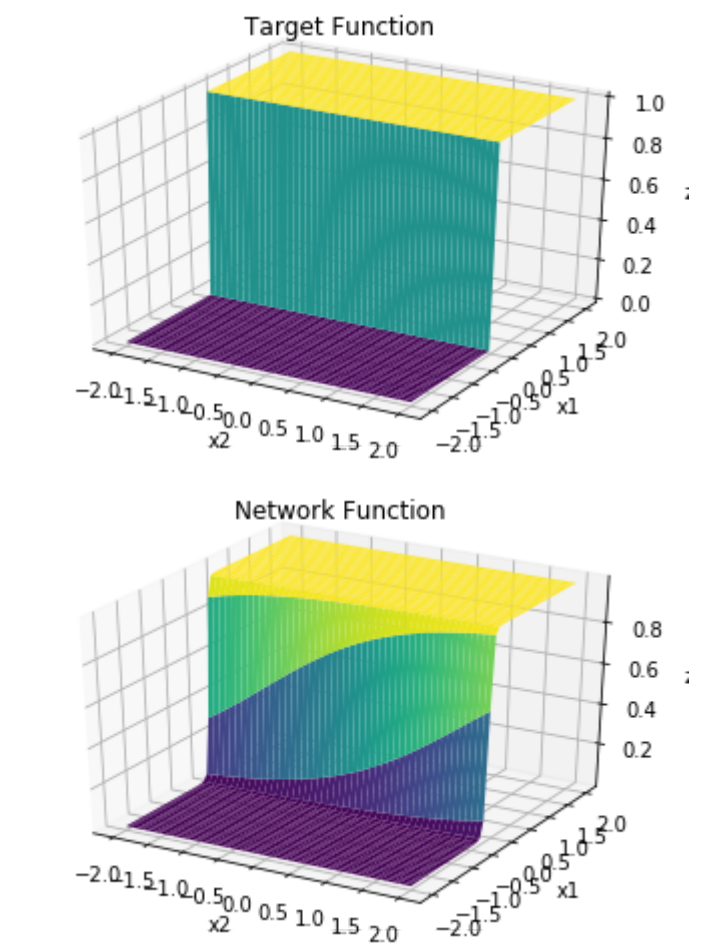
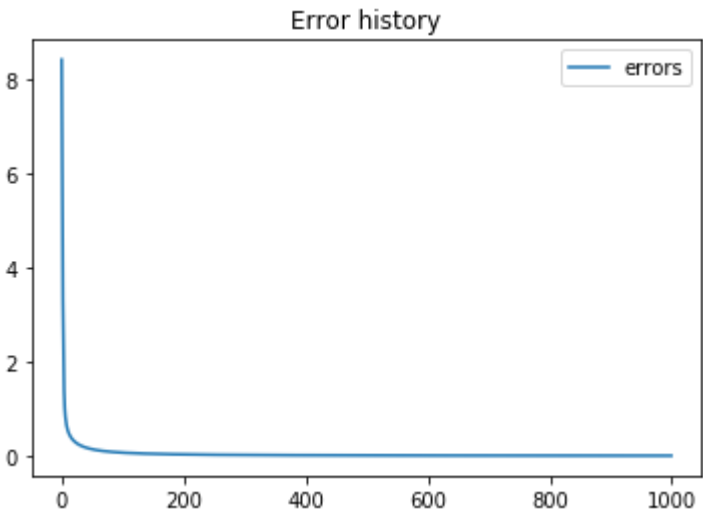
3 Q3 double layer perceptron

3.1 dataset generation

```
In [22]: 1 def f1(x):
2         out = np.zeros(x.shape[0])
3         out[x[:,0] >= 0] = 1.
4         out[x[:,0] < 0] = 0.
5
6         return out
7
8 y_stars = f1 (xs)
```

3.2 train loop

```
In [23]: 1 def sigmoid(x):
2         return 1/(1+np.exp (-x))
3
4 def get_weight_update_for_one_data(x,y_star , eta , params):
5
6     y,z1,z2 = get_network_output (x, params)
7
8     wp1, wp2, w11, w12, w21, w22, theta1, theta2, theta3 = params
9
10    delta_wp1 = - eta * (y - y_star) * y * (1 - y) * z1
11    delta_wp2 = - eta * (y - y_star) * y * (1 - y) * z2
12    delta_theta3 = - eta * (y - y_star) * y * (1 - y) * (-1)
13
14    delta_w11 = - eta * (y - y_star) * y * (1 - y) * wp1 * z1 * (1-z1) * x[0]
15    delta_w12 = - eta * (y - y_star) * y * (1 - y) * wp1 * z1 * (1-z1) * x[1]
16    delta_theta1 = - eta * (y - y_star) * y * (1 - y) * wp1 * z1 * (1-z1) * (-1)
17
18    delta_w21 = - eta * (y - y_star) * y * (1 - y) * wp2 * z2 * (1-z2) * x[0]
19    delta_w22 = - eta * (y - y_star) * y * (1 - y) * wp2 * z2 * (1-z2) * x[1]
20    delta_theta2 = - eta * (y - y_star) * y * (1 - y) * wp2 * z2 * (1-z2) * (-1)
21
22    delta_params = [delta_wp1, delta_wp2, delta_w11, delta_w12, delta_w21, delta_w22, delta_theta1, delta_theta2, delta_theta3]
23    return delta_params
24
25 def get_network_output (x, params):
26     wp1, wp2, w11, w12, w21, w22, theta1, theta2, theta3 = params
27     if x.ndim == 2:
28         z1 = sigmoid (w11*x[:,0] + w12*x[:,1] - theta1)
29         z2 = sigmoid (w21*x[:,0] + w22*x[:,1] - theta2)
30         return sigmoid (wp1*z1 + wp2*z2 - theta3), z1, z2
31     else:
32         z1 = sigmoid (w11*x[0] + w12*x[1] - theta1)
33         z2 = sigmoid (w21*x[0] + w22*x[1] - theta2)
34         return sigmoid (wp1*z1 + wp2*z2 - theta3), z1, z2
35
36
37 epochs = 1000
38 eta = 1
39 wp10 = wp20 = w110 = w120 = w210 = w220 = theta10 = theta20 = theta30 = 3
40
41 params_init = [wp10, wp20, w110, w120, w210, w220, theta10, theta20, theta30]
42 errors, params_history, final_params = train_multiple_epochs (eta, epochs, params_init)
43
44 plot_epochs(epochs, errors, params_history = [], params_names = [])
45 plot_in_range (final_params)
```

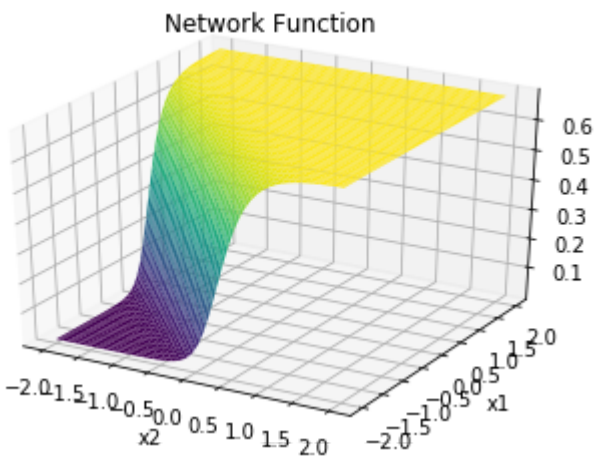
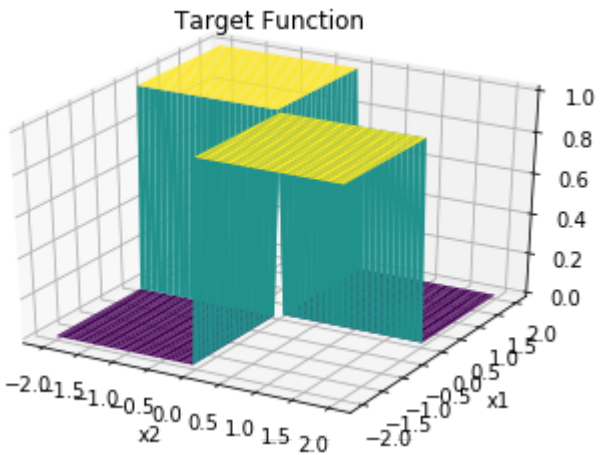
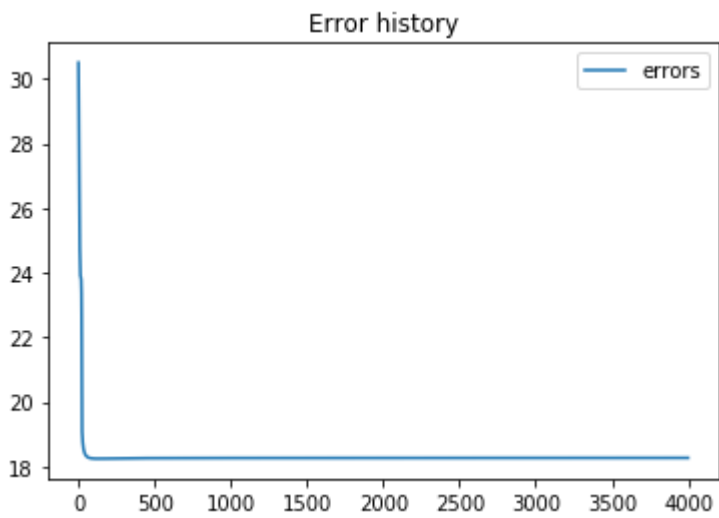


همانطور که دیده می‌شود، رفتار تابع مورد نظر به درستی یادگرفته شده است و خطا به صفر نزدیک شده است. به نظر می‌رسد تفاوت جدی در این مورد با بخش قبل وجود ندارد و در هر دو حالت شبکه موفق عمل کرده است.

3.3 XOR function

```
In [24]: 1 def XOR(x):
2         out = np.zeros(x.shape[0])
3         out[x[:,0]*x[:,1] <= 0] = 1.
4         out[x[:,0]*x[:,1] > 0] = 0.
5
6         return out
7
8 f1 = XOR
9 y_stars = f1 (xs)
```

```
In [25]: 1 epochs = 4000
2 eta = 1
3 wp10 = wp20 = w110 = w120 = w210 = w220 = theta10 = theta20 = theta30 = 3
4
5 params_init = [wp10, wp20, w110, w120, w210, w220, theta10, theta20, theta30]
6 errors, params_history, final_params = train_multiple_epochs(eta, epochs, params_init)
7
8 plot_epochs(epochs, errors, params_history = [], params_names = [])
9 plot_in_range (final_params)
```



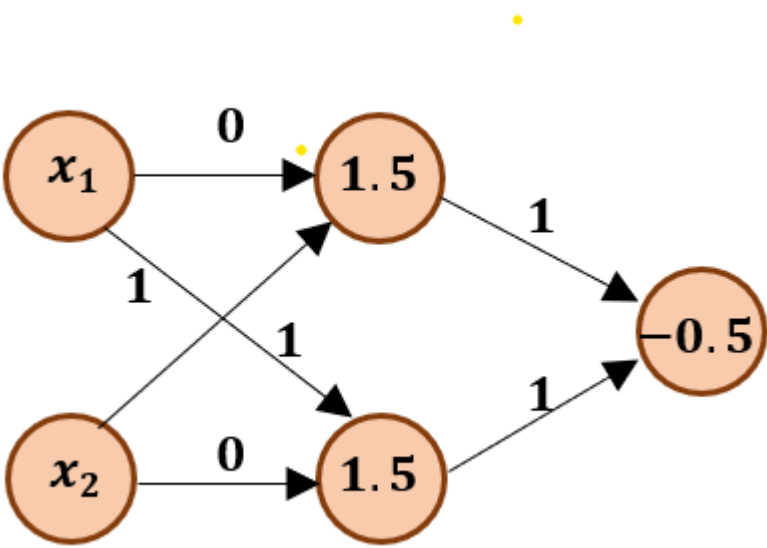
علی‌رغم این که انتظار داریم یک پرسپترون ۲ لایه قابلیت یادگیری تابع XOR را داشته باشد، اما در اینجا می‌بینیم که این اتفاق نیافتاده است. علت این مسئله شرایط اولیه نامناسب و نحوه آموزش شبکه است. در واقع نقطه شروع نامناسب در مسئله بهینه سازی تابع هدف سبب شده است که مدل در یک بهینه محلی گیر بیافتد و نتواند به بهینه سراسری برسد. همچنین نحوه آموزش باعث می‌شود که تابع هدف به شدت ناهموار باشد و لذا گیر کردن در بهینه‌های محلی به مراتب محتمل‌تر شود.

3.4 Calculalting New initital

پیاده‌سازی تابع XOR با کمک یک پرسپترون دو لایه در اسلایدهای درس نیز توضیح داده شده است. لذا در اینجا به طور خلاصه آن را توضیح می‌دهیم. برای پیدا کردن مقادیر مورد نظر، باید به رابطه بولین تابع XOR دقت کنیم:

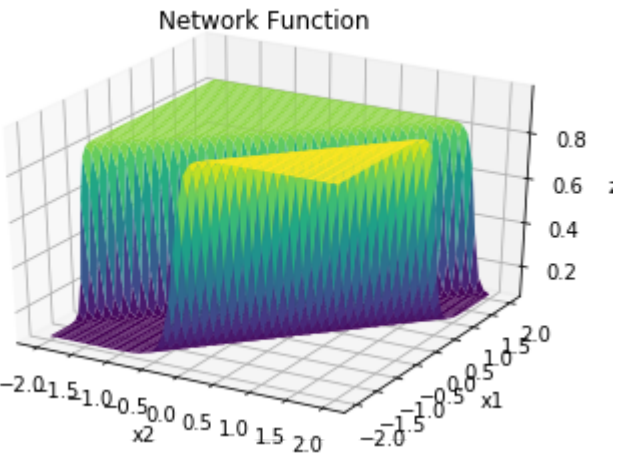
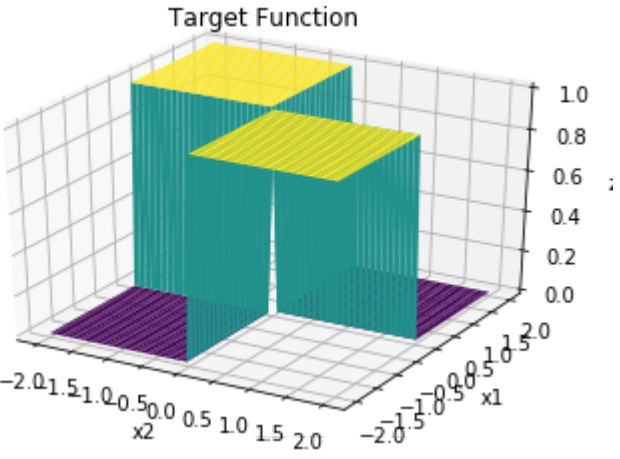
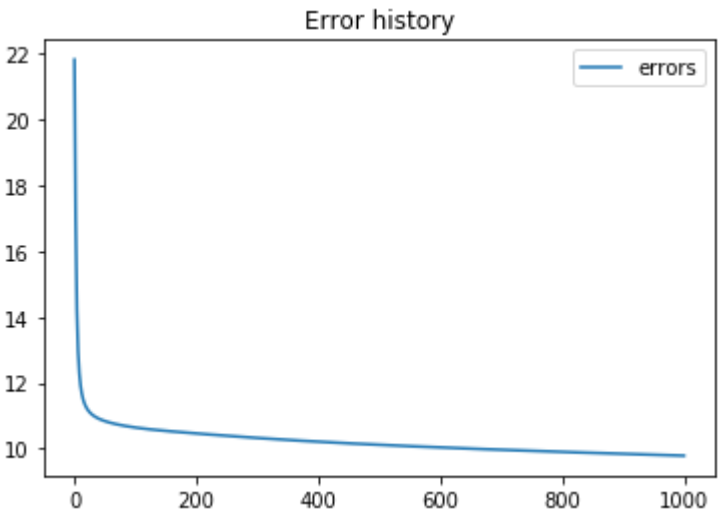
$$x_1 \oplus x_2 = \overline{x_1}x_2 + x_1\overline{x_2}$$

برای پیدا کردن مقادیر وزن‌ها، هدف آن است که لایه اول دو تابع AND را پیاده‌سازی نماید و حاصل این عبارت در لایه دوم با هم or شوند. لذا با در نظر گرفتن یک تابع آستانه تیز، مقادیر زیر به عنوان پاسخ مورد نظر پیشنهاد می‌شوند:



3.5 training with new initial conditions


```
In [26]: 1 epochs = 1000
2 eta = 1
3 wp10 = wp20 = 1
4 w220 = w110 = 1
5 w120 = w210 = 0
6 theta10 = theta20 = 3/2
7 theta30 = -1/2
8
9 params_init = [wp10, wp20, w110, w120, w210, w220, theta10, theta20, theta30]
10 errors, params_history, final_params = train_multiple_epochs(eta, epochs, params_init)
11
12 plot_epochs(epochs, errors, params_history = [], params_names = [])
13 plot_in_range (final_params)
```



به نظر می‌رسد به دلیل بهبود در نقطه شروع پارامترها، رفتار یادگرفته شده این بار بهبود جدی داشته است و خطا خیلی کمتر شده است. به عبارت دیگر نقطه شروع مناسب کمک کرده است تا مدل در بهینه‌های محلی گیر نیفتد و به نتایج بهتری برسد. البته هنوز ارور صفر نشده است و مدل قابلیت جدا سازی کامل را ندارد.