بسمه تعالی

تمرین اول مبانی هوش محاسباتی

استاد مربوطه

علی سبطی

محمد صادق نیکوفکر

شماره ی دانشجویی

9814162126

27.1.1400

گیت فردکین

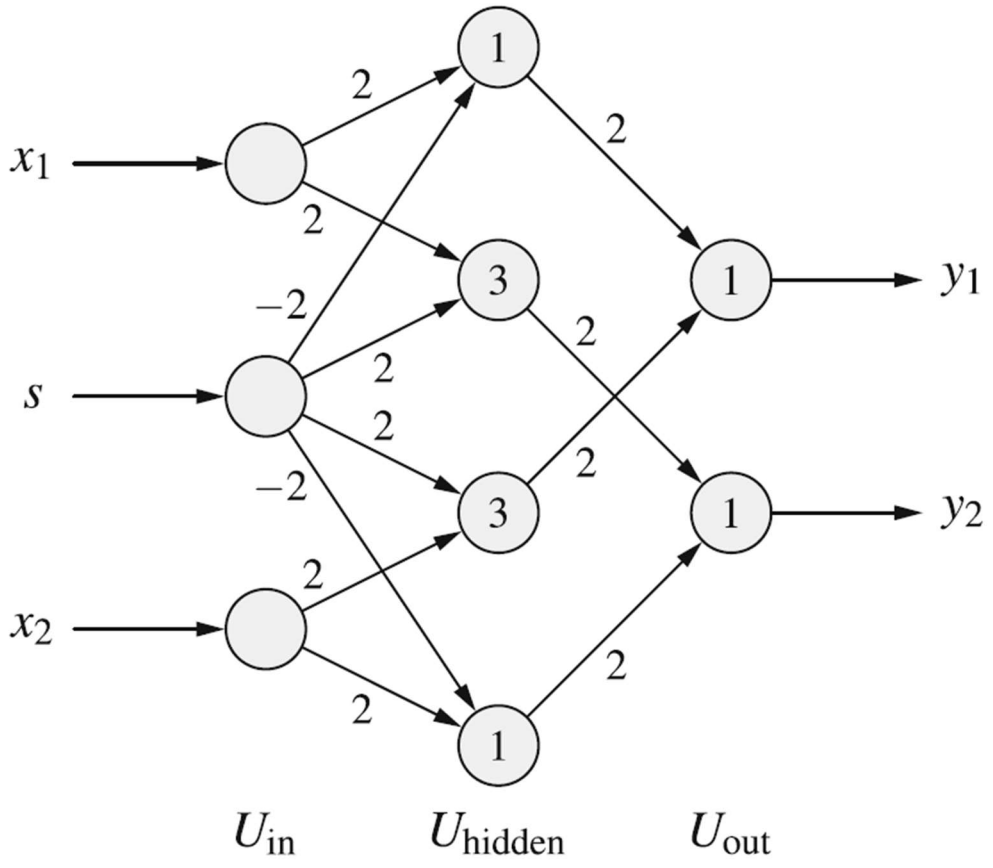
در این تمرین قصد داریم تا با استفاده از قانون دلتا ، شبکه ی نورونی (در این مثال شبکه ای با نورون) واستفاده از دیگر مفاهیم یادگیری ماشین ، گیت فردکین که با نام cswap هم شناخته میشود را آموزش داده و نتیجه را بررسی کنیم .

در گیت فردکین سه ورودی S , X1 , X2 را داریم که در نهایت منجر به تولید دو ورودی O1 , O2 میشود . روش کار این گیت به این صورت است که با استفاده از بیت کنترلی(S) مشخص میکند که در چه زمانی باید جای خروجی هارا تغییر داد . به عبارتی این بیت در صورت صفر بودن باعث میشود که ورودی X1 به خروجی O1 برود و ورودی X2 نیز به خروجی O2 منتقل شود اما درصورتی که این بیت برابر با یک باشد سواپ فعال میشود و جای این خروجی ها تغییر میکند به شکلی که X1 مقدار O2 را مشخص میکند و X2 مقدار O1 را تعیین میکند .

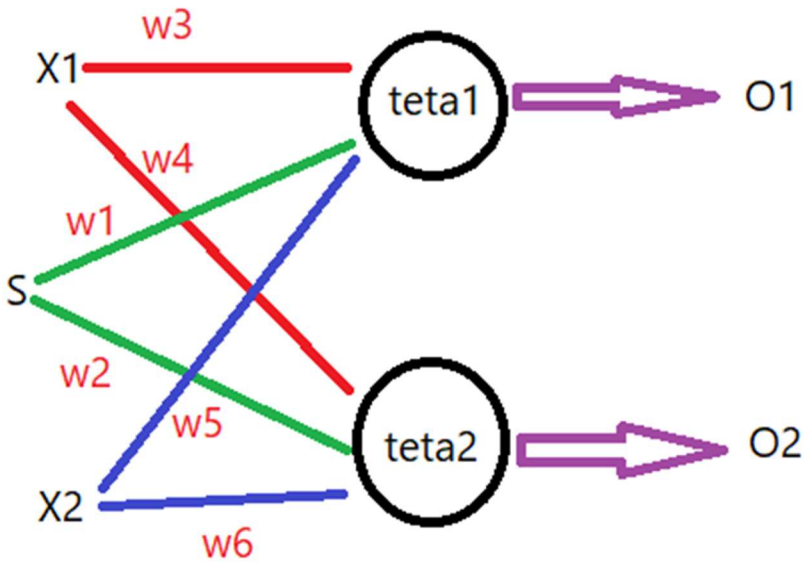
s	0 0 0 0 1 1 1 1
x_1	0 0 1 1 0 0 1 1
x_2	0 1 0 1 0 1 0 1
y_1	0 0 1 1 0 1 0 1
y_2	0 1 0 1 0 0 1 1

در حالت عادی گیت فردکین با طراحی مولتی لیر پرسپترون پیاده سازی میشود اما در این تمرین از ما خواسته شده تا اینکار را بدون لایه ی مخفی انجام بدهیم.میتوان عبارت جبری این گیت را بدین طریق نمایش داد :

((X1 xor X2) ^ s) xor (X1 / X2) = Y1 / Y2



نمایش گیت فردکینی که میخواهیم طراحی کنیم به شکل زیر است :



همانطور که در شکل مشخص است مدلی که قصد طراحی آن را داریم با مدل معمول تفاوت فاحشی دارد و این امر موجب میشود تا تغییراتی را برای حل مسئله در نظر بگیریم .

بنابر این طبق این مدل ما سه ورودی ، دو خروجی ، شش بردار وزن ، دو نورون و در نتیجه دو واحد اندازه گیری آستانه داریم .

برای پیاده سازی این مدل در برنامه نویسی بردار ها ، متغیر ها و مقادیر زیر را در نظر میگیریم :

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

o = [[0,0],[0,1],[1,0],[1,1],[0,0],[1,0],[0,1],[1,1]]

e = 1

teta1 = 1.8

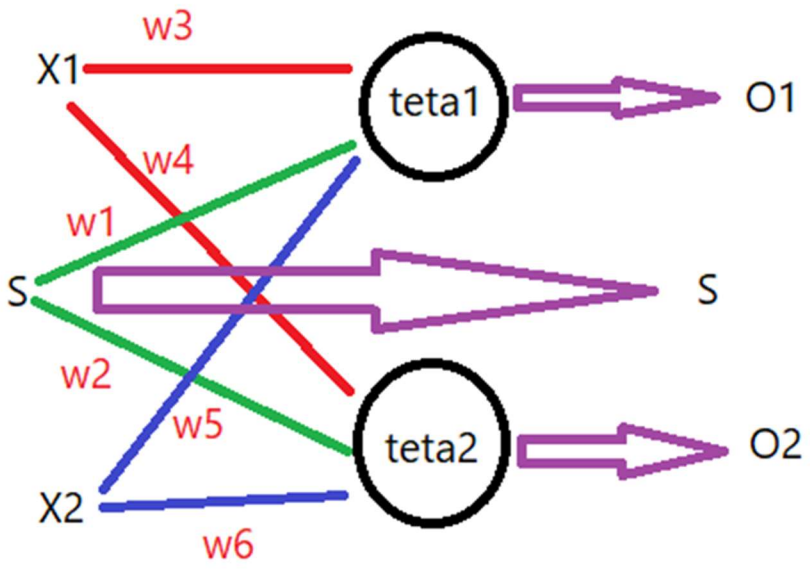
teta2 = 1.6

eta = 0.1

w = [[1,1.4,-3],[2,0.2,3.1]]

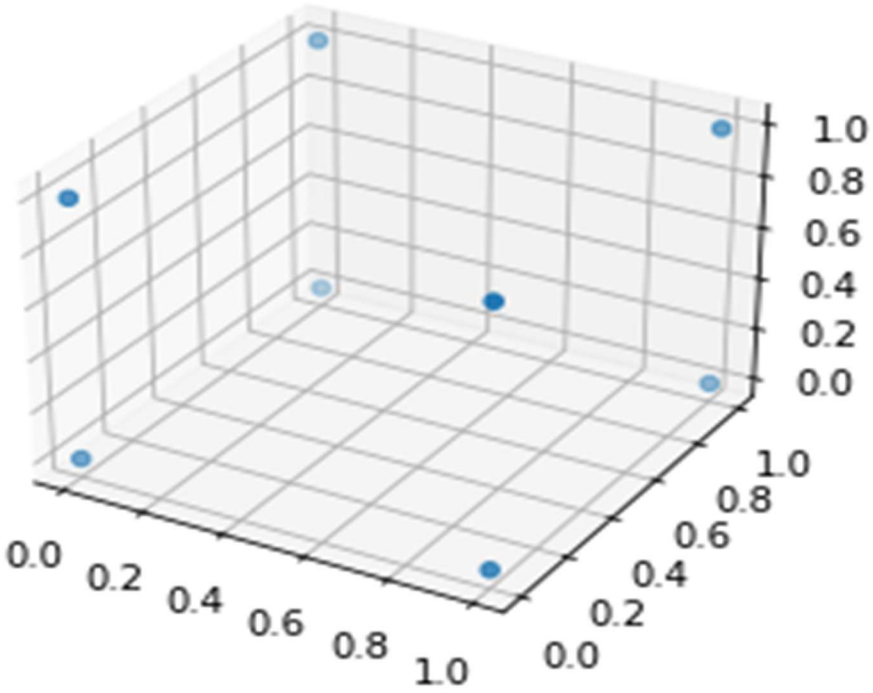
- بردار X یک ارایه (لیست) چند بعدی است که مقادیر سه ورودی(S , X1 , X2) ما را در بر دارد .
- در لیست O خروجی های مطلوب را نگه داری میکنیم .
- e مقدار خطا را برایمان مشخص میکند که با استفاده از آن متوجه میشویم تا با خروجی مطلوب چقدر فاصله داریم(در ابتدا برابر یک است تا وارد حلقه ی while بشویم)
- مقادیر تتا مشخص کننده تابع فعال سازی نورون های ما هستند . هر کدام از این ها مشخص میکنند که یکی از نورون ها ما نسبت به چه ورودی هایی حساس باشد و با دیدن آن ها فعال شود که با پیدا کردن مقادیر این دو ورودی بخشی از مدل را آموزش داده ایم .
- Eta مقدار نرخ یادگیری را مشخص میکند و تعیین میکند تا در زمان آموزش با چه اندازه گامی به دنبال جواب های مطلوب بگردیم و در نمونه جستجو کنیم .
- بردار w مجهول دیگر برنامه ی ما است که وزن های مرتبط با ورودی هارا مشخص میکند و زمانی که مقادیر صحیح این وزن ها و همچنین تتا های اشاره شده در بالا را پیدا کنیم به پاسخ مسئله رسیده ایم.

نمایش گیت فردکین حالت دومی که برای استفاده ی بهینه از لایبری های پایتون طراحی شده است(برای نمایش خروجی در حالت سه بعدی نیاز به سه متغیر خروجی داریم که جای x , y , z را پر میکنند اما در حالت فرض شده ی مسئله این پارامتر سوم را نداریم بنابر این آن را در فایل کد دوم اضافه میکنیم.)



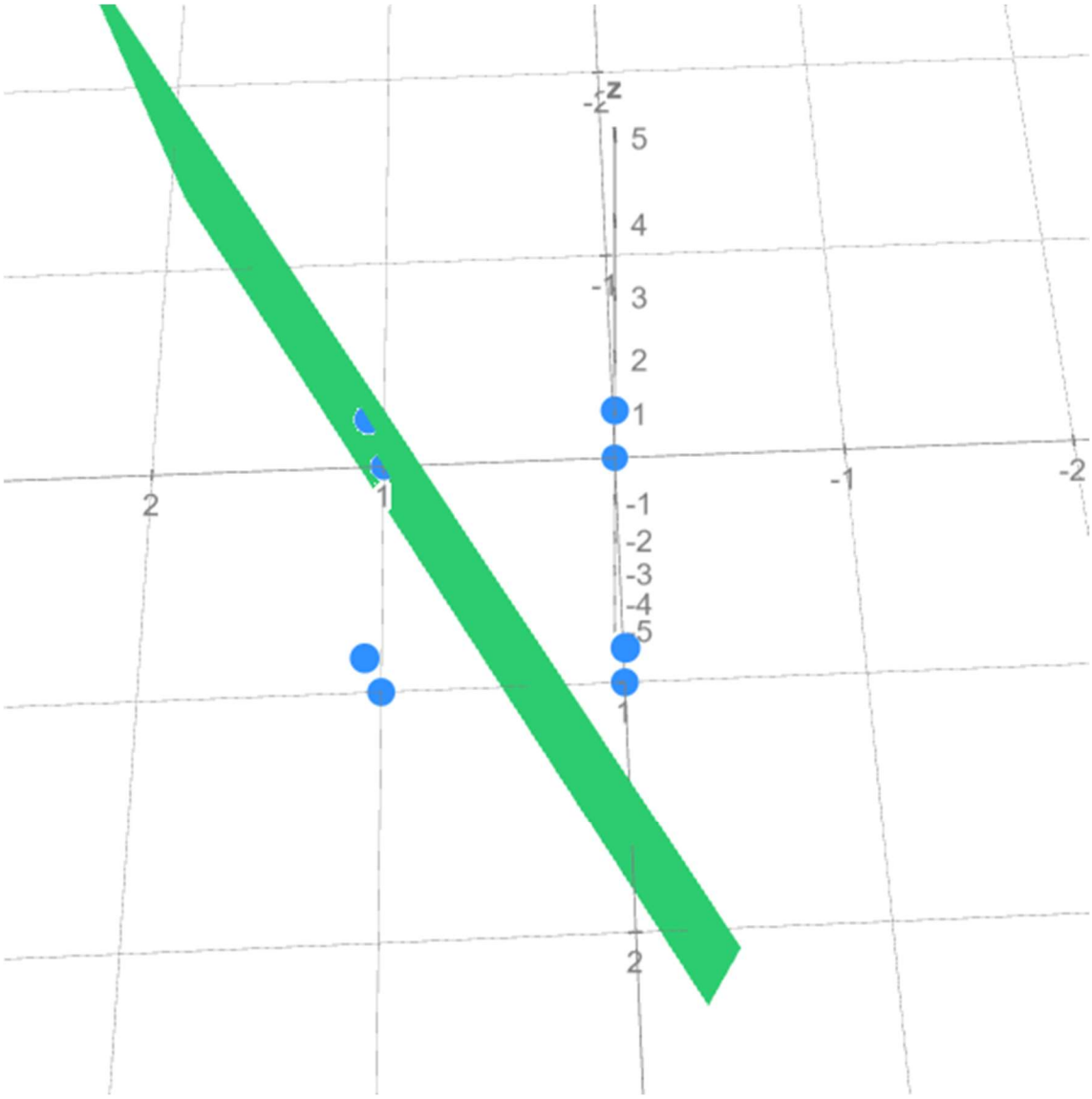
نمایش فضایی مسئله

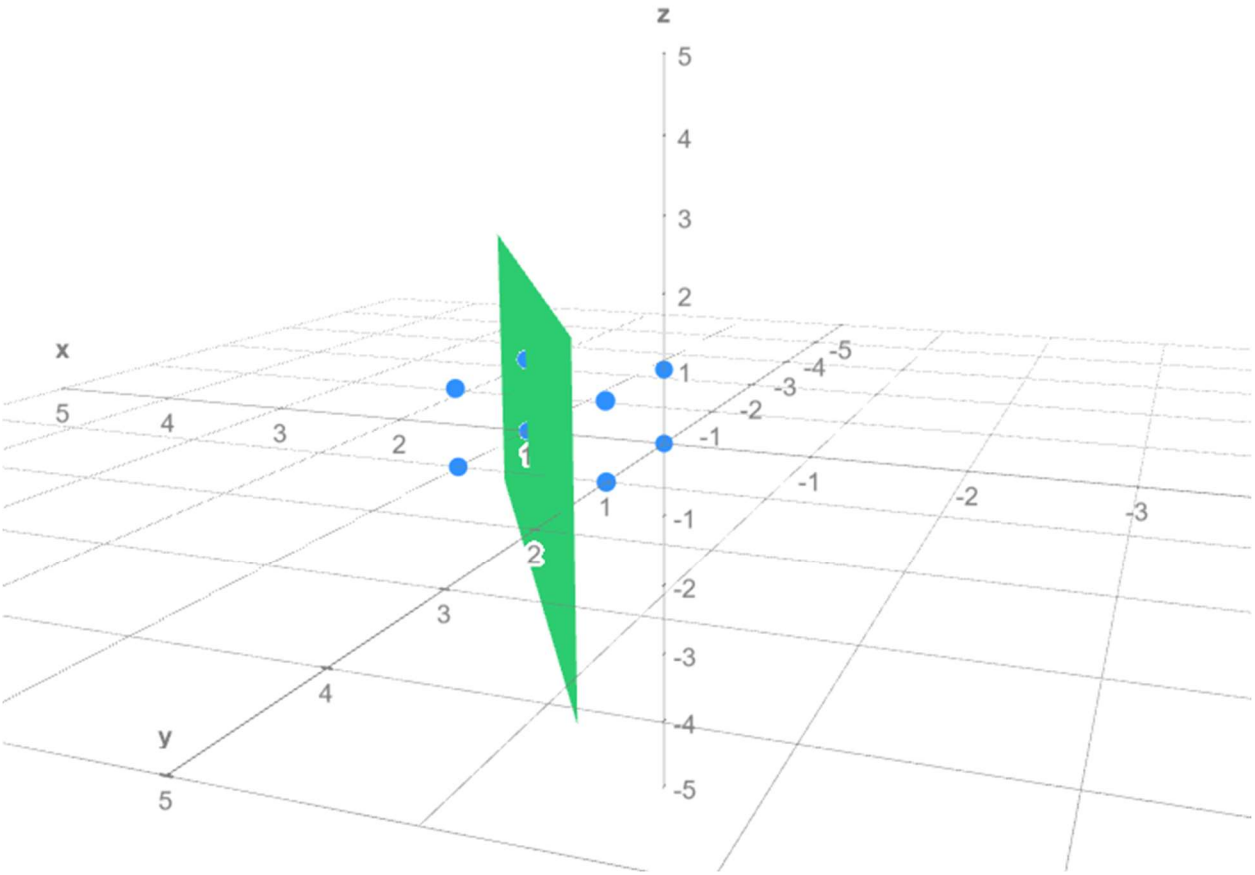
در تصویر زیر نمایش فضای مسئله را میبینید که به شکل سه بعدی (به علت وجود سه ورودی) با استفاده از لایبری matplotlib و در زبان پایتون پیاده سازی شده است . نقاط در این صفحه مختصات نشانگر هشت حالت مرتبط با ورودی های ما هستند و صفحاتی فرضی مشخص کننده ی مقادیر آموزش دیده شده هستند .



هر کدام از نقاط مشخص شده در تصویر نمایانگر X (لیست مقادیر ورودی X2, X1 , S) می باشد و در صورتی که بخواهیم خروجی را هم مدل کنیم شاهد صفحاتی خواهیم بود که مشخص کننده ی مرز میان نقاط با پاسخ صحیح و نقاط بدون پاسخ صحیح هستند . این صفحات با استفاده از مقادیر وزن ها و تتا ها مشخص و طبق فرمول خاصی محاسبه می شوند .

از آنجا که این فرمول برای محاسبه ی خط بدین شکل میباشد **y = (teta -w1*t)/w2** احتمال میدهم فرمول مشابه برای صفحه چیزی نزدیک به این فرمول باشد **(Y=(teta – WX1-WX2)/WS** یا **y[i] = (teta[i] – WX1*t – WX2*t)/WS** که در آن i مشخص کننده ی خروجی نظیر(برای حل مسئله مقادیر خروجی هر دو نورون در لیستی ریخته شده اند و این نشانگر مشخص کننده ی آن است که منظور از خروجی کدام یک است) و t خطی است که وقتی با مقادیر داده شده ضرب میشود صفحه ی مورد نظر را برای نمایش آماده میکند .


z
5
4
3
2
1
-1
-2
-3
-4
-5
2
1
-1
-2
2



برای مثال صحفه ی آورده شده در بالا مجموعه پاسخ هایی است که منجر به درست شدن چهار نقطه ی سمت چپ و عدم برابری خروجی ها با خروجی مطلوب نقاط سمت راست است . در نتیجه مقدار ارور 4 را به ما میدهد .

```
while e>0 :
    e=0
        .
        .
        .
        Print(epoch)
```

با استفاده از این حلقه شرط مد نظرمان که کاهش ارور ها تا رسیدن به صفر می باشد را چک میکنیم و در صورتی که تمام خروجی ها با تمام خروجی های مطلوب برابر باشد برنامه به پایان میرسد .

```
for i in range(8):
    .
    .
    .
```

در این حلقه یادگیری انجام میشود به این شکل که تمامی هشت نمونه ی ما در هر دوره بررسی و بر اساس آن آموزش داده میشود .

```
y=[None,None]
    wt = np.transpose(w)
    y =np.matmul(x[i],wt)
```

در اینجا لیست خروجی ها را معرفی کرده وبرای بدست آوردن مقادیرش ، ورودی هارا با استفاده از کتابخانه ی نامپای ، در ترانهاده ی بردار وزن ها ضرب میکنیم . دلیل استفاده از ترانهاده برابر کردن ستون های ماتریس ورودی و سطر های بردار وزن هاست که اگر اینگونه نباشد ضرب ماتریس امکان پذیر نیست .

```
if y[0] >= teta1:
    y[0] = 1
else:
    y[0] = 0
```

```
        if y[1] >= teta2:
            y[1] = 1
        else:
            y[1] = 0
```

در اینجا خروجی بدست آمده برای هر نورون را به صورت مجزا ، با استفاده از فانکشن فعال سازی نورون میسنجیم . به عبارتی دیگر در این بخش کد به دنبال این هستیم که بفهمیم مقدار نهایی خروجی بدست امده در ضرب قبلی منجر به فعال سازی نورون میشود یا نه .

```
    if y[0]!=o[i][0]:
        teta1=teta1 -eta*((o[i][0])-y[0])
        w[0] = w[0] + (eta*(o[i][0]-y[0])*np.array(x[i]))
        e = e + abs(o[i][0]-y[0])
    if y[1]!=o[i][1]:
        teta2=teta2 -eta*(o[i][1]-y[1])
        w[1] = w[1] + (eta*(o[i][1]-y[1])*np.array(x[i]))
        e = e + abs(o[i][1]-y[1])
```

در این بخش مقدار خروجی بدست امده را با خروجی مطلوب بررسی کرده و در صورت تفاوت ، وزن و تتا را برای رسیدن به کارکرد صحیح تغییر میدهیم و مقدار خطا را افزایش میدهیم . این تغییرات با استفاده از قانون دلتا اعمال میشوند که به صورت خلاصه میتوان گفت ، تفاوت مقدار خروجی بدست آمده با خروجی مطلوب با اعمال ضریب یادگیری تعیین شده را بر روی مقادیر تتا ، وزن ها و خطا اعمال میکنیم .

در نهایت با نشان دادن خروجی ها نتیجه را مشاهده میکنیم .

نتیجه

همانطور که در بالا گفته شد این مسئله را نمیتوان با استفاده از قانون تتا مدل کرد زیرا قانون تتا برای برخی مسائل خطی پاسخ میدهد در حالی که در اینجا پاسخ مسئله ی ما صفحه است و این فراتر از خط است و در نتیجه(تغییر دادن مقادیر اولیه ی وزن ها و تتا ها نیز تغییر زیادی در نتیجه ایجاد نمیکند) نمیتوان با استفاده ازین قانون ، مدلی برای این مسئله با خطای صفر بدست آورد .

بررسی روند نتایج

در ادامه و با دیدن نتایج که در ادامه آورده خواهد شد مشاهده میکنیم که همانطور که انتظار داشتیم در ابتدا الگوریتم با بررسی و آموزش مقادیر وزن ها و تتا هارا تغییر میدهد و در نمودار حرکت میکند و هر بار نتیجه ی متفاوتی به دست می آورد . برای مثال در اولین اپوک به هفت ارور برخورد میکند ، دیتای جدید را وارد و تست میکند و در اپوک دوم به شش ارور میرسد . در اپوک های بعدی مقادیر مختلفی برای ارور به دست می اورد تا جایی که به ارور هشت میرسد و گیر می افتد(در نمونه ی سوم در اپوک 33 ، در نمونه ی دوم در اپوک 4 و در نمونه ی سوم در اپوک 22) . پس از رسیدن به ارور هشت تمام اپوک های بعدی شبیه یکدیگر میشوند و دیگر به نتیجه نمیرسیم . از سه نمونه ی آورده شده میتوان نتیجه گرفت که مقدار نا مناسب eta باعث جهش های بزرگ نمونه می شود و شانس پیدا کردن پاسخ را کمتر میکند(از روی جواب میپرد) که نمونه ی دو نشانگر این موضوع است . همچنین مشاهده میکنیم که تغییرات تتا ها و وزن ها باعث تغییر نسبی زمان رسیدن به بن بست می شود و همچنین گاهی نمونه ای که مقادیر مناسب تری دارد به پاسخ نزدیک تر می شود (همانگونه که نمونه ی اول به مینیمم ارور 2 رسیده ولی نمونه ی چهارم نهایتا به ارور 4 دست پیدا کرده است).

نمونه ی اول با مقادیر ابتدایی زیر

teta1 = 1.8

teta2 = 1.6

eta = 0.1

w = [[1,1.4,-3],[2,0.2,3.1]]

.....
epoch:1
teta1:1.4999999999999998
teta2:1.8000000000000003
w:[array([1.1, 1.6, -2.7]), array([1.8, 0.2, 3.])]
y:[0. 1.]
e:7.0
.....
epoch:2
teta1:1.2999999999999996
teta2:1.8000000000000003
w:[array([1.2, 1.7, -2.4]), array([1.8, 0.3, 2.9])]
y:[0. 1.]
e:6.0
.....
epoch:3
teta1:1.1999999999999995
teta2:1.9000000000000004
w:[array([1.2, 1.8, -2.1]), array([1.7, 0.3, 2.8])]
y:[0. 1.]
e:6.0
.....
epoch:4
teta1:1.0999999999999994
teta2:1.9000000000000004
w:[array([1.2, 1.9, -1.8]), array([1.7, 0.4, 2.7])]
y:[0. 1.]
e:7.0
.....
epoch:5
teta1:1.0999999999999994
teta2:1.9000000000000004

w:[array([1.1, 1.9, -1.6]), array([1.7, 0.5, 2.6])]
y:[1. 1.]
e:6.0
.....
epoch:6
teta1:1.0999999999999994
teta2:2.0000000000000004
w:[array([1. , 1.9, -1.4]), array([1.6, 0.5, 2.5])]
y:[1. 1.]
e:5.0
.....
epoch:7
teta1:1.0999999999999994
teta2:2.0000000000000004
w:[array([0.9, 1.9, -1.2]), array([1.6, 0.6, 2.4])]
y:[1. 1.]
e:6.0
.....
epoch:8
teta1:0.9999999999999994
teta2:2.0000000000000004
w:[array([0.9, 1.9, -1.]), array([1.6, 0.7, 2.3])]
y:[1. 1.]
e:5.0
.....
epoch:9
teta1:0.9999999999999994
teta2:2.1000000000000005
w:[array([0.8, 1.9, -0.8]), array([1.5, 0.7, 2.2])]
y:[1. 1.]
e:5.0
.....
epoch:10
teta1:0.9999999999999994
teta2:2.1000000000000005
w:[array([0.8, 1.8, -0.7]), array([1.5, 0.8, 2.1])]
y:[1. 1.]
e:4.0
.....
epoch:11

teta1:0.9999999999999994
teta2:2.1000000000000005
w:[array([0.8, 1.7, -0.6]), array([1.4, 0.8, 2.1])]
y:[1. 1.]
e:4.0
.....
epoch:12
teta1:0.9999999999999994
teta2:2.0000000000000004
w:[array([0.8, 1.6, -0.5]), array([1.4, 0.9, 2.1])]
y:[1. 1.]
e:5.0
.....
epoch:13
teta1:0.9999999999999994
teta2:2.1000000000000005
w:[array([0.8, 1.5, -0.4]), array([1.3, 0.9, 2.])]
y:[1. 1.]
e:3.0
.....
epoch:14
teta1:0.9999999999999994
teta2:2.0000000000000004
w:[array([0.8, 1.4, -0.3]), array([1.3, 1. , 2.])]
y:[1. 1.]
e:5.0
.....
epoch:15
teta1:0.9999999999999994
teta2:2.0000000000000004
w:[array([0.8, 1.3, -0.2]), array([1.2, 1. , 2.])]
y:[1. 1.]
e:4.0
.....
epoch:16
teta1:0.9999999999999994
teta2:2.0000000000000004
w:[array([0.8, 1.2, -0.1]), array([1.1, 1. , 2.])]
y:[1. 1.]
e:4.0

.....
epoch:17
teta1:0.9999999999999994
teta2:1.9000000000000004
w:[array([8.00000000e-01, 1.10000000e+00, 1.52655666e-15]), array([1.1, 1.1, 2.
])]
y:[1. 1.]
e:5.0
.....
epoch:18
teta1:0.9999999999999994
teta2:2.0000000000000004
w:[array([0.8, 1. , 0.1]), array([1. , 1.1, 1.9])]
y:[1. 1.]
e:3.0
.....
epoch:19
teta1:0.9999999999999994
teta2:1.9000000000000004
w:[array([0.8, 0.9, 0.2]), array([1. , 1.2, 1.9])]
y:[1. 1.]
e:5.0
.....
epoch:20
teta1:0.9999999999999994
teta2:1.9000000000000004
w:[array([0.7, 0.9, 0.2]), array([0.9, 1.2, 1.9])]
y:[1. 1.]
e:4.0
.....
epoch:21
teta1:0.9999999999999994
teta2:1.9000000000000004
w:[array([0.6, 0.9, 0.2]), array([0.8, 1.2, 1.9])]
y:[1. 1.]
e:4.0
.....
epoch:22
teta1:0.8999999999999995
teta2:1.8000000000000003

w:[array([0.6, 0.9, 0.3]), array([0.8, 1.3, 1.9])]
y:[1. 1.]
e:6.0
.....
epoch:23
teta1:0.9999999999999994
teta2:1.9000000000000004
w:[array([0.5, 0.8, 0.3]), array([0.7, 1.3, 1.8])]
y:[1. 1.]
e:2.0
.....
epoch:24
teta1:0.8999999999999995
teta2:1.8000000000000003
w:[array([0.5, 0.8, 0.4]), array([0.7, 1.4, 1.8])]
y:[1. 1.]
e:6.0
.....
epoch:25
teta1:0.8999999999999995
teta2:1.8000000000000003
w:[array([0.4, 0.8, 0.4]), array([0.6, 1.4, 1.8])]
y:[1. 1.]
e:4.0
.....
epoch:26
teta1:0.8999999999999995
teta2:1.8000000000000003
w:[array([0.3, 0.8, 0.4]), array([0.5, 1.4, 1.8])]
y:[1. 1.]
e:4.0
.....
epoch:27
teta1:0.7999999999999995
teta2:1.7000000000000002
w:[array([0.3, 0.8, 0.5]), array([0.5, 1.5, 1.8])]
y:[1. 1.]
e:6.0
.....
epoch:28

teta1:0.8999999999999995
teta2:1.8000000000000003
w:[array([0.2, 0.7, 0.5]), array([0.4, 1.5, 1.7])]
y:[1. 1.]
e:2.0
.....
epoch:29
teta1:0.7999999999999995
teta2:1.7000000000000002
w:[array([0.2, 0.7, 0.6]), array([0.4, 1.6, 1.7])]
y:[1. 1.]
e:6.0
.....
epoch:30
teta1:0.7999999999999995
teta2:1.7000000000000002
w:[array([0.1, 0.7, 0.6]), array([0.4, 1.6, 1.7])]
y:[1. 1.]
e:6.0
.....
epoch:31
teta1:0.7999999999999995
teta2:1.7000000000000002
w:[array([1.38777878e-16, 7.00000000e-01, 6.00000000e-01]), array([0.4, 1.6,
1.7])]
y:[1. 1.]
e:6.0
.....
epoch:32
teta1:0.6999999999999995
teta2:1.7000000000000002
w:[array([1.38777878e-16, 7.00000000e-01, 7.00000000e-01]), array([0.4, 1.6,
1.7])]
y:[1. 1.]
e:7.0
.....
epoch:33
teta1:0.6999999999999995
teta2:1.7000000000000002

w:[array([1.38777878e-16, 7.00000000e-01, 7.00000000e-01]), array([0.4, 1.6, 1.7])]
y:[1. 1.]
e:8.0
.....
epoch:34
teta1:0.6999999999999995
teta2:1.7000000000000002
w:[array([1.38777878e-16, 7.00000000e-01, 7.00000000e-01]), array([0.4, 1.6, 1.7])]
y:[1. 1.]
e:8.0
.....
epoch:35
teta1:0.6999999999999995
teta2:1.7000000000000002
w:[array([1.38777878e-16, 7.00000000e-01, 7.00000000e-01]), array([0.4, 1.6, 1.7])]
y:[1. 1.]
e:8.0
.....
epoch:36
teta1:0.6999999999999995
teta2:1.7000000000000002
w:[array([1.38777878e-16, 7.00000000e-01, 7.00000000e-01]), array([0.4, 1.6, 1.7])]
y:[1. 1.]
e:8.0
.....
epoch:37
teta1:0.6999999999999995
teta2:1.7000000000000002
w:[array([1.38777878e-16, 7.00000000e-01, 7.00000000e-01]), array([0.4, 1.6, 1.7])]
y:[1. 1.]
e:8.0
.....
epoch:38
teta1:0.6999999999999995
teta2:1.7000000000000002

w:[array([1.38777878e-16, 7.00000000e-01, 7.00000000e-01]), array([0.4, 1.6, 1.7])]
y:[1. 1.]
e:8.0
.....
epoch:39
teta1:0.6999999999999995
teta2:1.7000000000000002
w:[array([1.38777878e-16, 7.00000000e-01, 7.00000000e-01]), array([0.4, 1.6, 1.7])]
y:[1. 1.]
e:8.0
.....
epoch:40
teta1:0.6999999999999995
teta2:1.7000000000000002
w:[array([1.38777878e-16, 7.00000000e-01, 7.00000000e-01]), array([0.4, 1.6, 1.7])]
y:[1. 1.]
e:8.0
.....

نمونه ی دوم با تغییر اتا به یک که باعث زودتر گیر افتادن در حلقه میشود (مقدار اتا مناسب نیست)

teta1 = 1.8
teta2 = 1.6
eta = 1
w = [[1,1.4,-3],[2,0.2,3.1]]

.....
epoch:1
teta1:0.8
teta2:2.6
w:[array([0. , 2.4, -1.]), array([1. , 1.2, 2.1])]
y:[1. 1.]
e:8.0
.....
epoch:2
teta1:0.8
teta2:1.6
w:[array([0. , 1.4, 0.]), array([1. , 2.2, 2.1])]

```
y:[1. 1.]
e:5.0
.............
epoch:3
teta1:0.8
teta2:2.6
w:[array([0. , 0.4, 1. ]), array([1. , 2.2, 1.1])]
y:[1. 1.]
e:5.0
.............
epoch:4
teta1:0.8
teta2:2.6
w:[array([0. , 0.4, 1. ]), array([1. , 2.2, 1.1])]
y:[1. 1.]
e:8.0
.............
epoch:5
teta1:0.8
teta2:2.6
w:[array([0. , 0.4, 1. ]), array([1. , 2.2, 1.1])]
y:[1. 1.]
e:8.0
.............
epoch:6
teta1:0.8
teta2:2.6
w:[array([0. , 0.4, 1. ]), array([1. , 2.2, 1.1])]
y:[1. 1.]
e:8.0
.............
epoch:7
teta1:0.8
teta2:2.6
w:[array([0. , 0.4, 1. ]), array([1. , 2.2, 1.1])]
y:[1. 1.]
e:8.0
.............
epoch:8
teta1:0.8
```

teta2:2.6
w:[array([0. , 0.4, 1.]), array([1. , 2.2, 1.1])]
y:[1. 1.]
e:8.0
.....
epoch:9
teta1:0.8
teta2:2.6
w:[array([0. , 0.4, 1.]), array([1. , 2.2, 1.1])]
y:[1. 1.]
e:8.0
.....
epoch:10
teta1:0.8
teta2:2.6
w:[array([0. , 0.4, 1.]), array([1. , 2.2, 1.1])]
y:[1. 1.]
e:8.0
.....

نمونه سوم با تغییر وزن ها و تتا ها

teta1 = 0.2
teta2 = -1.3
eta = 0.1
w = [[1,2,3],[3,2,1]]
.....
epoch:1
teta1:0.5
teta2:-0.8999999999999999
w:[array([0.8, 1.9, 2.9]), array([2.8, 1.9, 0.9])]
y:[1. 1.]
e:7.0
.....
epoch:2
teta1:0.7999999999999999
teta2:-0.5
w:[array([0.6, 1.8, 2.8]), array([2.6, 1.8, 0.8])]
y:[1. 1.]

e:7.0
.....
epoch:3
teta1:0.9999999999999999
teta2:-0.10000000000000003
w:[array([0.5, 1.7, 2.7]), array([2.4, 1.7, 0.7])]
y:[1. 1.]
e:6.0
.....
epoch:4
teta1:1.2
teta2:0.3
w:[array([0.4, 1.6, 2.6]), array([2.2, 1.6, 0.6])]
y:[1. 1.]
e:6.0
.....
epoch:5
teta1:1.4000000000000001
teta2:0.6
w:[array([0.3, 1.5, 2.5]), array([2. , 1.5, 0.5])]
y:[1. 1.]
e:5.0
.....
epoch:6
teta1:1.5000000000000002
teta2:0.7999999999999999
w:[array([0.2, 1.5, 2.4]), array([1.8, 1.4, 0.5])]
y:[1. 1.]
e:7.0
.....
epoch:7
teta1:1.6000000000000003
teta2:0.9999999999999999
w:[array([0.1, 1.5, 2.3]), array([1.6, 1.3, 0.5])]
y:[1. 1.]
e:7.0
.....
epoch:8
teta1:1.7000000000000004
teta2:1.2

w:[array([1.38777878e-16, 1.50000000e+00, 2.20000000e+00]), array([1.4, 1.2, 0.5])]
y:[1. 1.]
e:7.0
.....
epoch:9
teta1:1.7000000000000004
teta2:1.4000000000000001
w:[array([1.38777878e-16, 1.60000000e+00, 2.10000000e+00]), array([1.2, 1.1, 0.5])]
y:[1. 1.]
e:6.0
.....
epoch:10
teta1:1.7000000000000004
teta2:1.4000000000000001
w:[array([1.38777878e-16, 1.70000000e+00, 2.00000000e+00]), array([1.1, 1.1, 0.5])]
y:[1. 1.]
e:4.0
.....
epoch:11
teta1:1.8000000000000005
teta2:1.4000000000000001
w:[array([-0.1, 1.7, 1.9]), array([1. , 1.1, 0.5])]
y:[1. 1.]
e:5.0
.....
epoch:12
teta1:1.8000000000000005
teta2:1.4000000000000001
w:[array([-0.1, 1.7, 1.9]), array([0.9, 1.1, 0.5])]
y:[1. 1.]
e:6.0
.....
epoch:13
teta1:1.8000000000000005
teta2:1.4000000000000001
w:[array([-0.1, 1.7, 1.9]), array([0.8, 1.1, 0.5])]
y:[1. 1.]

e:6.0
.....
epoch:14
teta1:1.8000000000000005
teta2:1.4000000000000001
w:[array([-0.1, 1.7, 1.9]), array([0.7, 1.1, 0.5])]
y:[1. 1.]
e:6.0
.....
epoch:15
teta1:1.8000000000000005
teta2:1.3
w:[array([-0.1, 1.7, 1.9]), array([0.7, 1.1, 0.6])]
y:[1. 1.]
e:5.0
.....
epoch:16
teta1:1.8000000000000005
teta2:1.3
w:[array([-0.1, 1.7, 1.9]), array([0.6, 1.1, 0.6])]
y:[1. 1.]
e:6.0
.....
epoch:17
teta1:1.8000000000000005
teta2:1.3
w:[array([-0.1, 1.7, 1.9]), array([0.5, 1.1, 0.6])]
y:[1. 1.]
e:6.0
.....
epoch:18
teta1:1.8000000000000005
teta2:1.2
w:[array([-0.1, 1.7, 1.9]), array([0.5, 1.1, 0.7])]
y:[1. 1.]
e:5.0
.....
epoch:19
teta1:1.8000000000000005
teta2:1.2

w:[array([-0.1, 1.7, 1.9]), array([0.4, 1.1, 0.7])]
y:[1. 1.]
e:6.0
.....
epoch:20
teta1:1.8000000000000005
teta2:1.2
w:[array([-0.1, 1.7, 1.9]), array([0.3, 1.1, 0.7])]
y:[1. 1.]
e:6.0
.....
epoch:21
teta1:1.8000000000000005
teta2:1.0999999999999999
w:[array([-0.1, 1.7, 1.9]), array([0.3, 1.1, 0.8])]
y:[1. 1.]
e:5.0
.....
epoch:22
teta1:1.8000000000000005
teta2:1.0999999999999999
w:[array([-0.1, 1.7, 1.9]), array([0.3, 1.1, 0.8])]
y:[1. 1.]
e:8.0

.....
epoch:23
teta1:1.8000000000000005
teta2:1.0999999999999999
w:[array([-0.1, 1.7, 1.9]), array([0.3, 1.1, 0.8])]
y:[1. 1.]
e:8.0
.....
epoch:24
teta1:1.8000000000000005
teta2:1.0999999999999999
w:[array([-0.1, 1.7, 1.9]), array([0.3, 1.1, 0.8])]
y:[1. 1.]
e:8.0
.....

ضمیمه ها

ضمیمه های این تمرین شامل یک فایل ورد توضیحات ، یک فایل کد اصلی که بر اساس سه ورودی و دو خروجی نوشته شده است . و ضمیمه آخر فایل فرعی کد هست که برای استفاده از نمایش فضای سه بعدی نتیجه در آن از سه ورودی و سه خروجی که سومی همان S هست که به شکلی دست نخورده به خروجی اعمال میشود(نیازی به نورون ندارد و به طور مستقیم از S ورودی به S خروجی می رود.)