به نام خدا

مهرداد برادران

.

STDP/R-STDP Learning

.

استاد سعیدرضا خردپیشه

.

درس علوم اعصاب محاسباتی

.

شماره دانشجویی 99222020

در این پروژه سعی شده که فرآیند های یادگیری در مغز شبیه سازی بشود .

The basic idea underlying the proposed supervised R-STDP learning rule is the following: calculating the reward according to the supervised learning framework and strengthening a synaptic connection based on the combination effect of a dopamine reward and the STDP function, where STDP means strengthening a synaptic connection results in a faster buildup of the postsynaptic neuron potential when a presynaptic spike arrives, leading to the postsynaptic neuron firing earlier.

For this learning rule, the weight changes proposed by an STDP function are collected and a reward representing whether the output is higher or lower than the desired output is calculated after every simulation time window T. Then, this reward is used to change the synaptic connections of the network under the R-STDP learning rule.

The weights of the synaptic connections w_{ij}, where i and j are the indices of the pre-synaptic and the post-synaptic neurons, respectively, are updated after the simulation time window T and follow the equations:

$$w_{ij}(t) = w_{ij}(t - \Delta t) + \Delta w_{ij}(t) \qquad (5)$$

$$\Delta w_{ij}(t) = \eta \times r_{ij}(t) \times STDP_{ij}(t) \times g_{ij}(t). \qquad (6)$$

نورون های ورودی از طریق سیناپس هایی که قدرت اتصال متفاوتی دارند، به طور کامل به نورون خروجی متصل می شوند. بسته به رسانایی، وزن، نورون های پیش سیناپسی جریان پس سیناپسی (PSC) تولید می کنند که در گره های عصبی خروجی جمع شده و پتانسیل غشایی U(t) را افزایش می دهد. در مدل نورون (LIF)، پتانسیل به طور خود به خود با ثابت زمانی τ به صورت زیر تجزیه می شود.

$$\tau \frac{dU_j(t)}{dt} = -U_j(t) + \sum_{i=1}^{n} w_{ji} \times n_i(t) \qquad (1)$$

```
func = lambda x: 2000 * (math.sin(x) + 0.01)
```

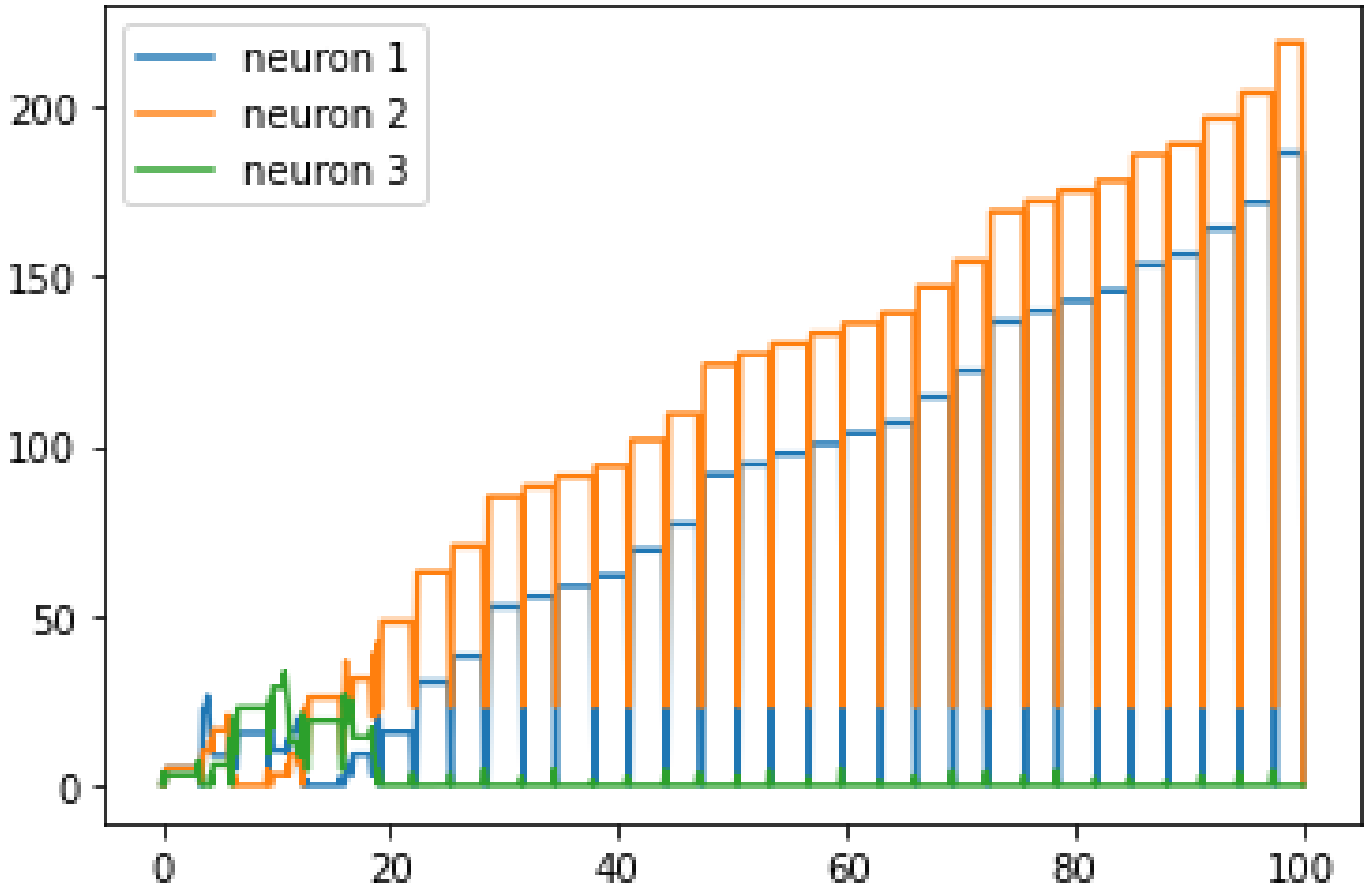
```
n1 = LIF(i_func=func)
n2 = LIF(i_func=func)
n3 = LIF(i_func=func)
```

```
connections = [[1, 2], [0, 2], [0, 1]]
```

```
nurons = [n1, n2, n3]
```

همانطور که در شکل بالا مشاهده میکنید برای جریان ورودی به نورون های جمعیت نورونی با fully connection میخواهیم خروجی هارو تخمین بزنیم .

به هر نورون تابع جریان سینوسی یکسانی وارد میکنیم .



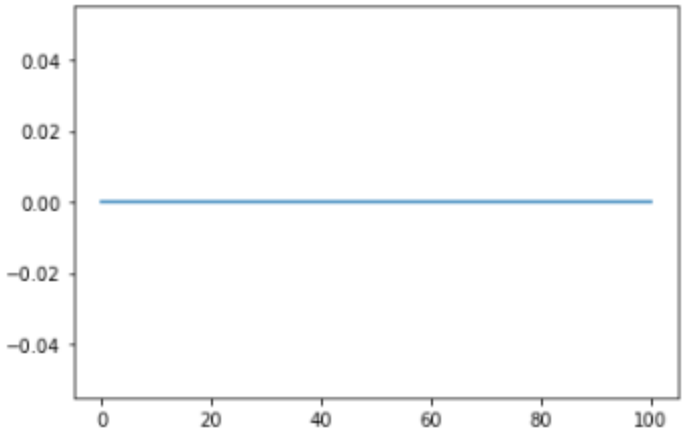
تغییرات پتانسیل را قبل فرایند یادگیری میتوان در شکل بالا مشاهده کرد که هرکدام به چ صورت پتانسیل خود را تغییر میدهند

کانکشن هر نورون با نورون های post-synaptic خودشون رو مشخص کردیم و در کانکشنز ذخیره کردیم تا تاثیراتی که مد نظر داریم نورون های جمعیت نورونی روی یکدیگر بگذارند.

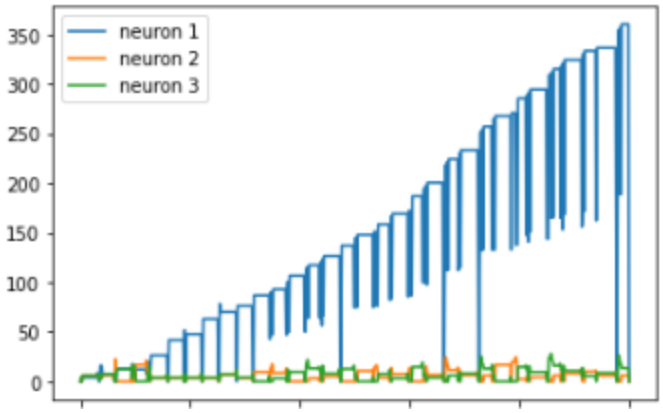
حال شبیه سازی را انجام داده و تغییرات وزن ها به صورت زیر هستند:

W_{11}:

```
learnin.w_plot(0 ,0)
```

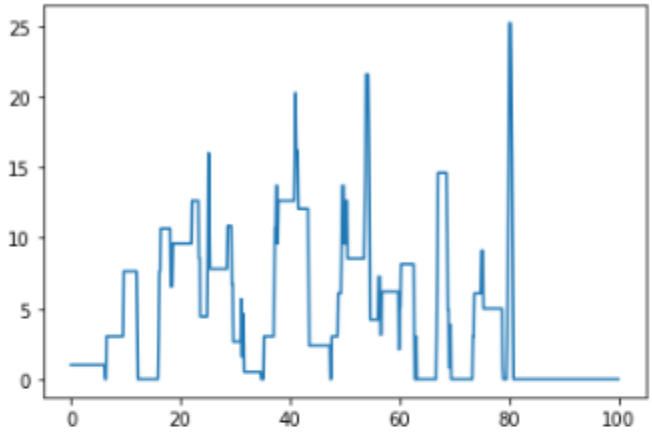



```
gp.neurons_u_plot()
```

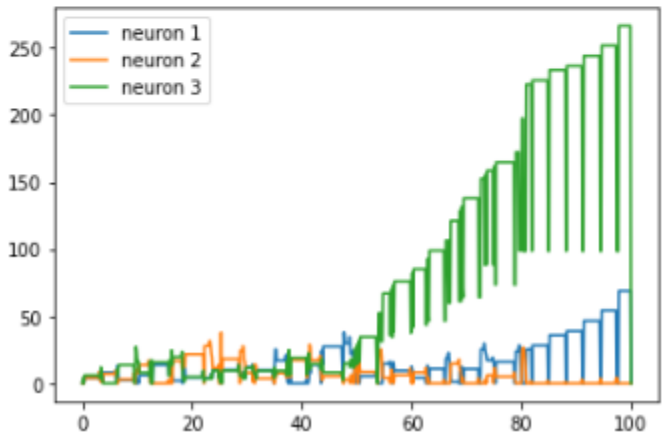


W_{12}:

```
learnin.w_plot(0 ,1)
```

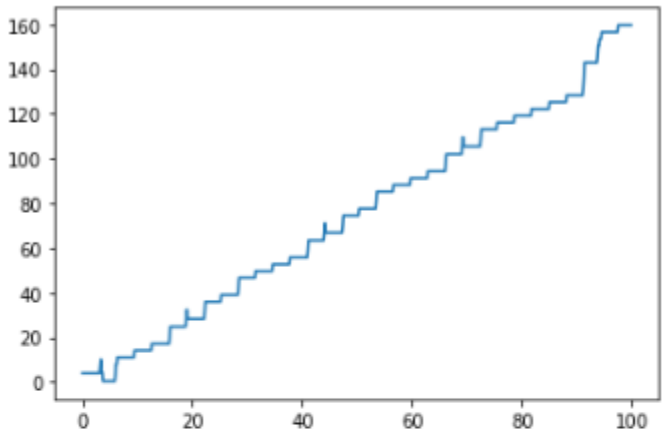



```
gp.neurons_u_plot()
```

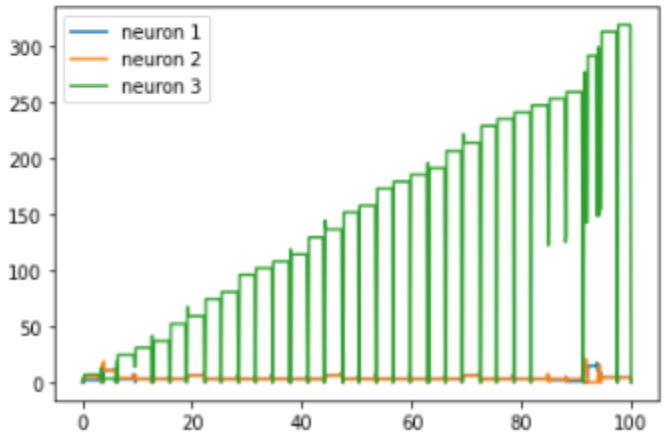


W_{13}:

```
learnin.w_plot(0 ,2)
```

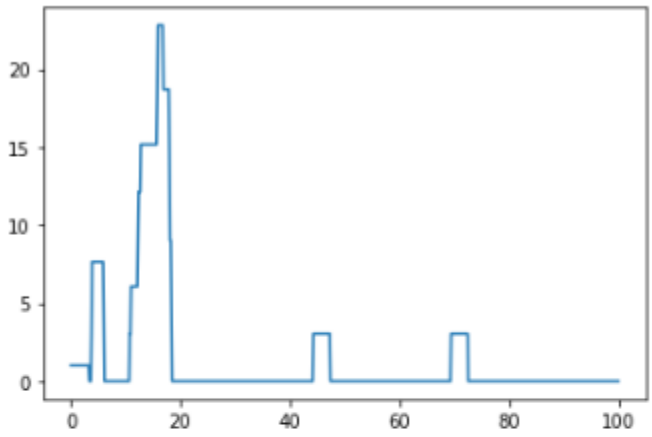



```
gp.neurons_u_plot()
```

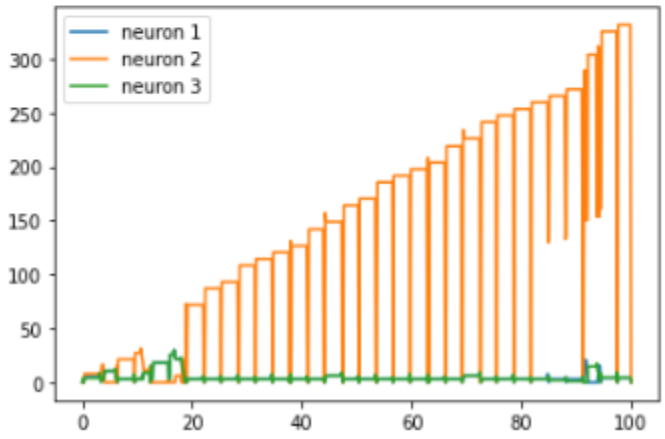


W_{21}:

```
learnin.w_plot(1 ,0)
```

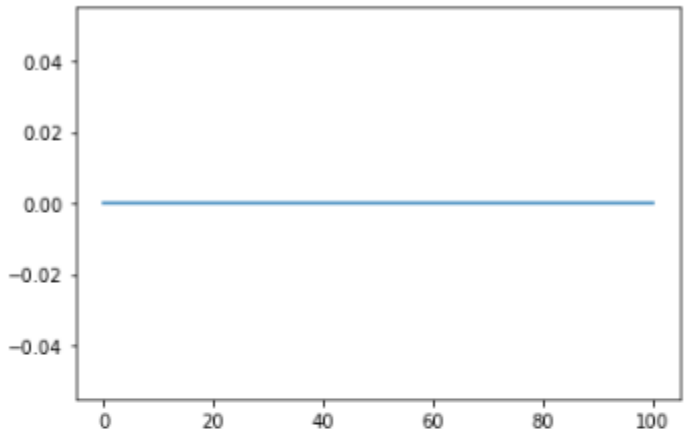



```
gp.neurons_u_plot()
```

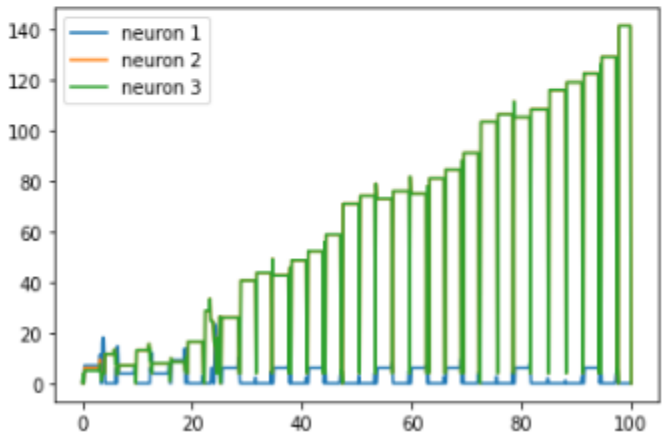


W_{22}:

```
learnin.w_plot(1 ,1)
```

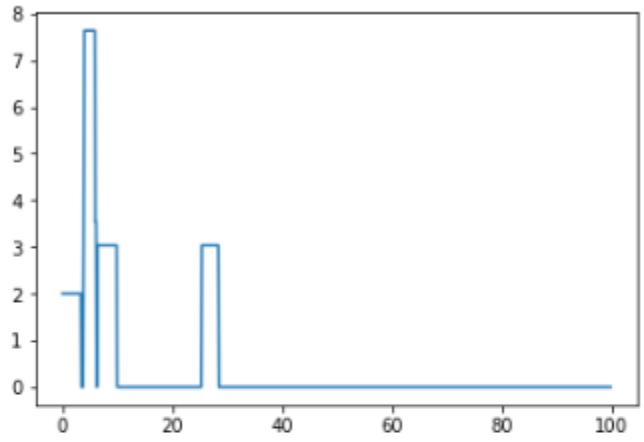



```
gp.neurons_u_plot()
```

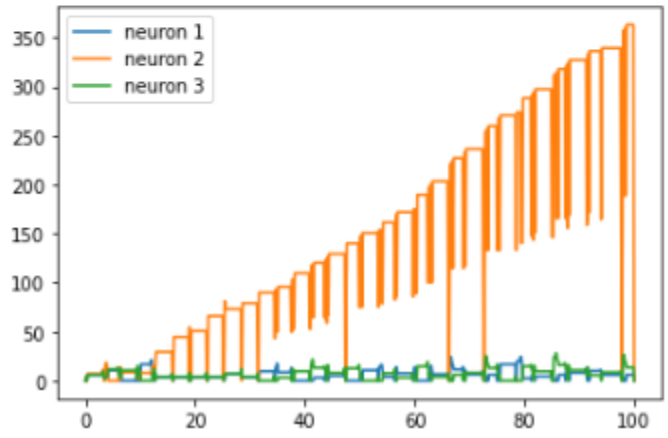


W_{23}:

```
learnin.w_plot(1 ,2)
```

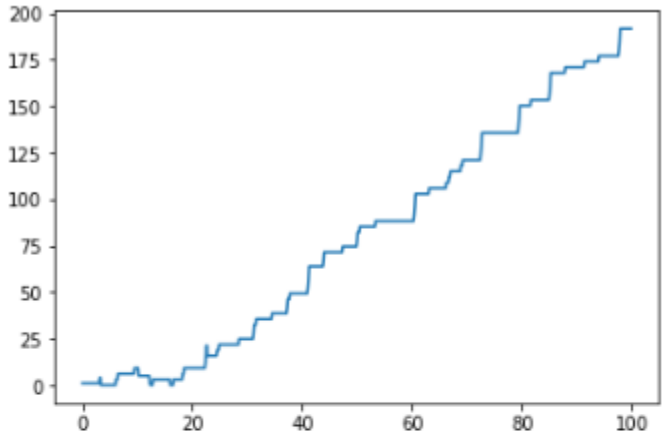



```
gp.neurons_u_plot()
```

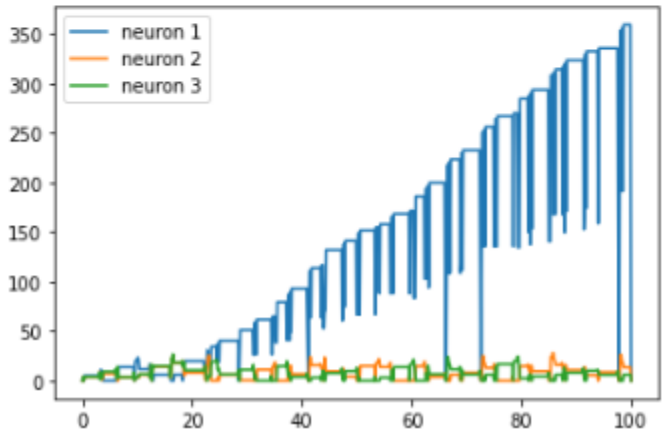


W_{31}:

```
learnin.w_plot(2 ,0)
```

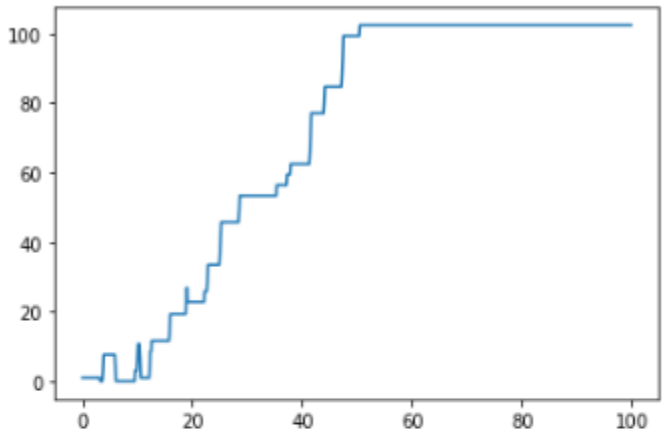



```
gp.neurons_u_plot()
```

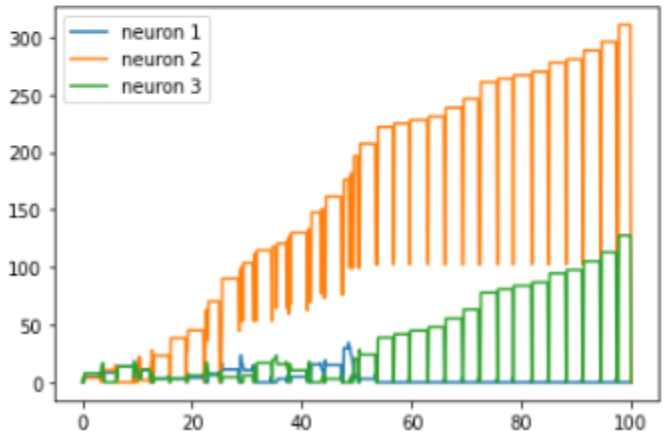


W_{32}:

```
learnin.w_plot(2 ,1)
```

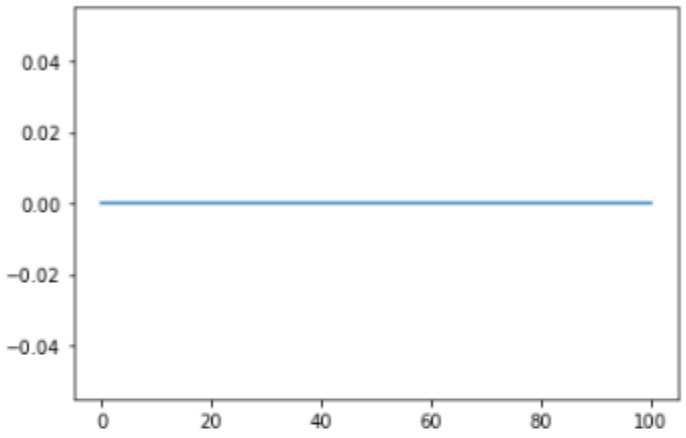



```
gp.neurons_u_plot()
```

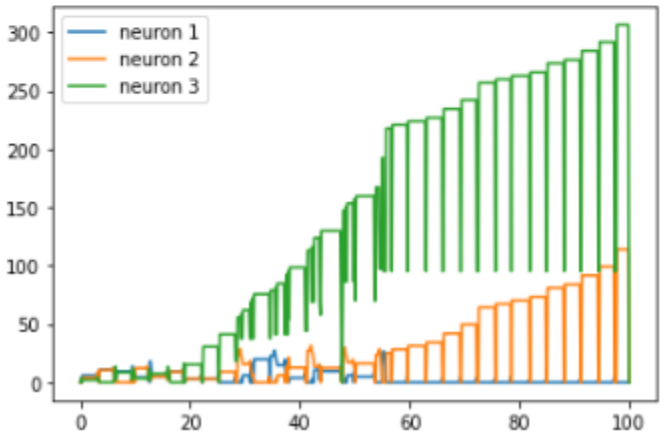


W_{33}:

```
learnin.w_plot(2 ,2)
```




```
gp.neurons_u_plot()
```



در این مرحله R-STDP را با ورودی های فایل اکسل آزمایش میکنیم .

حال نتیجه دقت مدل بر روی داده های train و test را میتوانید ببینید:

```
a = snn.predict(test_inputs,100, 0.125, 1)
```

```
0  0
0  0
0  0
0  0
0  0
0  0
1  0
0  0
0  0
0  0
```

```
count = 0
for i, el in enumerate(a):
    if el == test_outs[i]:
        count +=1
print(count/len(a))
print(a)
```

```
0.7
[1. 1. 1. 1. 1. 1. 0. 1. 1. 1.]
```

```
a = snn.predict(train_inputs,100, 0.125, 1)
```

```
0  0
0  0
1  0
0  0
0  0
0  0
0  0
0  0
0  0
0  0
```

```
count = 0
for i, el in enumerate(a):
    if el == train_outs[i]:
        count +=1
print(count/len(a))
print(a)
```

```
0.5
[1. 1. 0. 1. 1. 1. 1. 1. 1. 1.]
```