علیرضا آزادبخت

۹۵۲۲۲۰۰۳

STDP AND SPIKING NEURAL NETWORKS

PROJECT 3

تمرین شماره یک:

در این تمرین هدف پیاده سازی قانون یادگیری stdp برای دو نورون پست سینپتیک و پیری سینبتیک بود از آنجایی که کتابخانه هایی که در این زمینه معرفی شده اند دارای سینتکس خیلی دشواری بودند و اگر لازم بود که از آن ها استفاده کنیم نهایتا مجبور به کپی کردن نمونه کد های موجود بودیم تصمیم گرفتیم که پیاده سازی این سری از تمرین هارا نیز از صفر پیاده سازی کنیم.
اگر به رابطه های یادگیری نگاه کنیم متوجه میشویم که تنها زمان فایر کردن دو نورون اهمیت دارد ولی پتانسیل ارسالی آن ها اهمیتی ندارد

$$\frac{dy_i}{dt} = -\frac{y_i}{\tau_-} + \sum_f \delta(t - t_i^f). \quad \frac{dx_j}{dt} = -\frac{x_j}{\tau_+} + \sum_f \delta(t - t_j^f).$$

The values of x_j and y_i determine the weight change:

$$\frac{dw_{ij}}{dt} = -A_-(w_{ij}) y_i(t) \sum_f \delta(t - t_j^f) + A_+(w_{ij}) x_j(t) \sum_f \delta(t - t_i^f).$$

مراحل پیاده سازی خیلی روبه جلو انجام شد ولی بجای اینکه یک نورون واقعی مانند LIF قرار دهیم از یک تابع random_fire_pattern_generator استفاده کردیم این تابع به صورت رندوم و با در نظر گرفتن بازه هایی برای آماده سازی دوباره (cool down) و ملاک های مشابه زمان فایر کردن های دو نورون را تعیین میکند، به کمک این تابع ۲ الگوی فایر ایجاد میکنیم و هنگام شبیه سازی در بازه های زمانی که فایر ها اتفاق می افتند قانون یادگیری را انجام میدهیم

```
def get_fire_pattern(simulation_time=1000):
    fire_pattern = [0] * simulation_time
    cool_down = 0
    for i in range(simulation_time):
        if cool_down == 0 and random.random() > 0.5:
            fire_pattern[i] = 1
            cool_down = math.floor(10 + random.random() * (20 + random.random() * 5))
        else:
            if cool_down > 0:
                cool_down -= 1
    return fire_pattern
```

در ابتدای شبیه سازی وزن بین دو نورون را یک در نظر میگیریم و رفته رفته در هر استپ زمانی مقادیر x , y , w , delta_w را اپدیت میکنیم:

```
for i in range(len(timer)):
    # # X
    x[i] = x[i - 1] + ((-x[i - 1]) / tau_plus + x_fire_pattern[i])
    if x[i] < 0:
        x[i] = 0

    # # Y
    y[i] = y[i - 1] + ((-y[i - 1]) / tau_minus + y_fire_pattern[i])
    if y[i] < 0:
        y[i] = 0

    # # W
    delta_w[i] = (-a_minus * A_minus(w[i - 1]) * y[i - 1] * x_fire_pattern[i - 1]) + (a_plus * A_plus(w[i -
1]) * x[i - 1] * y_fire_pattern[i - 1])

    w[i] = w[i - 1] + delta_w[i]
```

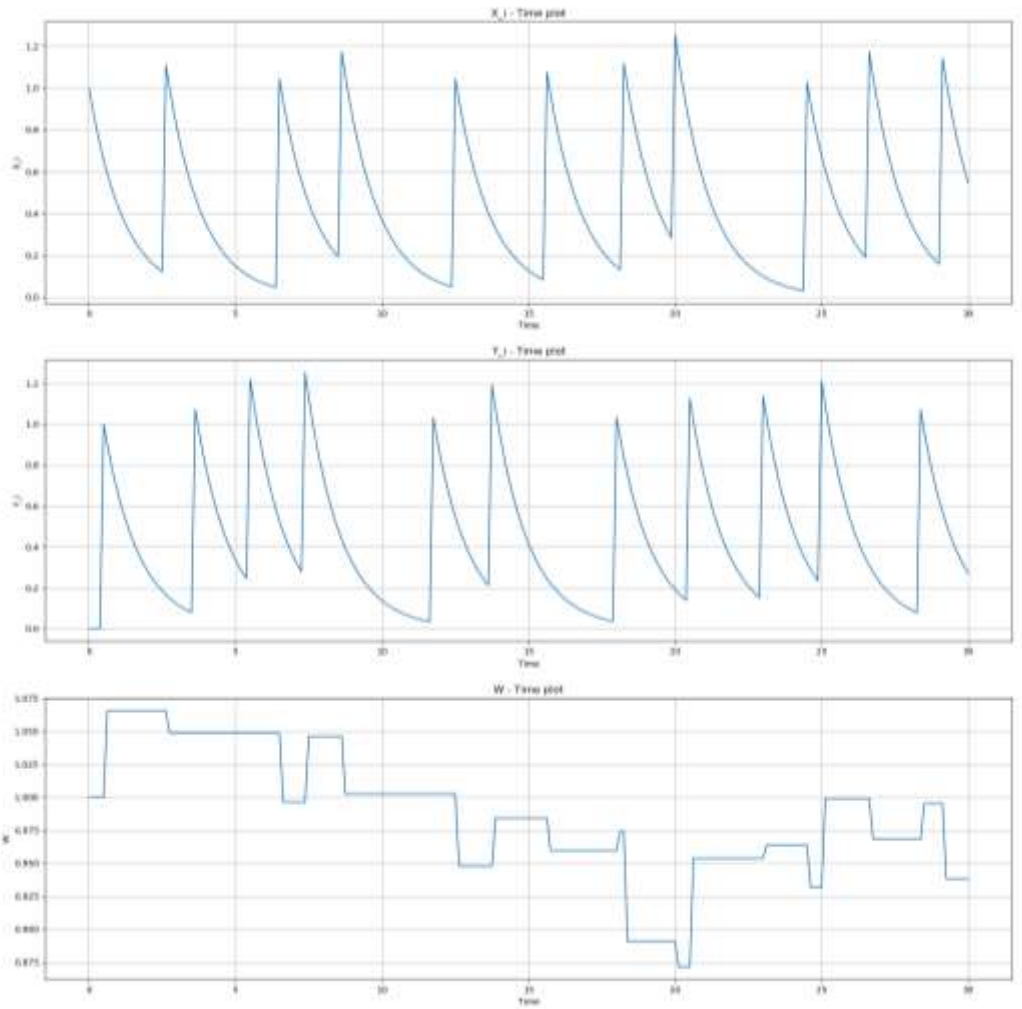
پارامتر های ورودی را در حالت های مختلف شبیه سازی کردیم (grid search) و نمودار هایی که به نمودار های مرجع شبیه سازی های کتابخانه های معروف وجود داشت را گزارش کردیم.

تابع بکار رفته در A از نوع hard bound است.

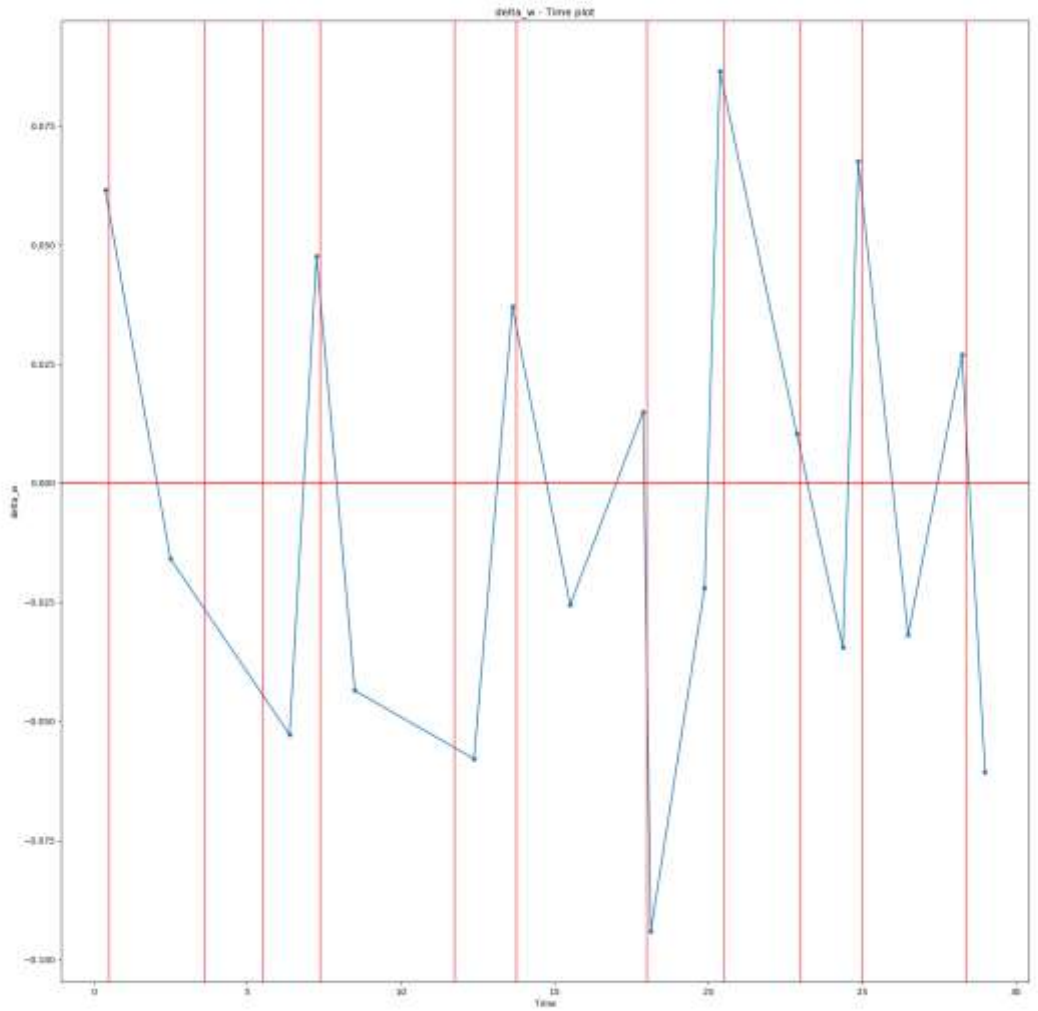
```
tau_plus = 10
tau_minus = 10

# a_minus = 0.1
# a_plus = -a_minus*tau_minus / tau_plus * 1.05
a_minus = 0.1
a_plus = 0.1
```

```
def A_plus(w):
    if w > 1:
        return 0
    return 1


def A_minus(w):
    if w == 0:
        return 0
    return 1
```

نتایج بدست آمده به شکل زیر میباشند:



از روی نقاط نوک تیز نمودار های x,y مشخص میشود که الگو فایر کردن نورون ها در چه حالتی قرار داشته اند الگو های مشاهده شده بقدر کافی راضی کننده هستند مشاهده میکنیم که وقتی نورون پیری سینپتیک فایر میکند مقدار وزن کاهش می یابد و هنگامی که نورون پست سینپتیک فایر میکند مقدار وزن افزایش می یابد. تفاوتی که بین نمودار وزن ها و نمودار های متداول وزن هایی که گزارش میشود وجود خط های شیب دار است این به این دلیل است که ساختار اپدیت وزن ها در استپ بعدی اعمال میشود بخاطر همین اندکی انحراف در این خط ها وجود دارد و مستقیما خط عمودی نیستند



نمودار delta_w خیلی شهود خوبی پیدا نکرد و موفق به درست کردن آن هم نشدیم ولی بعضی از نکات اصلی این نوع نمودار ها همچنان دیده میشوند. خطوط عمودی قرمز زمان فایر کردن نورون های پست سینپتیک را نمایش میدهد و مشاهده میکنیم که در اکثر نقاط بعد از فایر کردن اختلاف وزن ها به شدت منفی میشود که با ایده کلی قانون یادگیری هماهنگ است ولی در بعضی قسمت ها مشاهده میکنیم که خلاف این اتفاق می افتد و اگر بررسی کنیم متوجه میشویم که در این نقاط تعداد فایر کردن های نورون پیری سینپتیک بیشتر بوده و اثر کاهش وزن حاصل از فایر نورون پست سینپتیک را خنثی میکند و باعث میشود که در نمودار این اختلاف با ایده اصلی بوجود بیاید. به طور کلی این نمودار نزدیک به نتایج مورد نیازمان هست و حدس میزنیم که مشکل حاصل بخاطر این هست که ما اثر اتصال دو نورن را در نظر نگرفتیم و در صورت فایر کردن نورون ها پتانسیل قبلی به نورون بعد منتقل نمیشود هرچند ممکن است سناریویی اتفاق بیافتند که مشابه الگو هایی که ما ایجاد کردیم باشد ولی به هر حال بخشی از مسئله محسوب میشود که ما در نظر نگرفتیم و در تمرین دوم و سوم این موضوع را حل کرده ایم و نتایج دلچسب تری بدست آوردیم.

تمرین شماره دو:

در این تمرین ۱۰ نورون ورودی داریم که از روش قبل برای اینکه بتوانیم الگو های مورد نیاز را کنترل کنیم و بتوانیم در زمان مناسب الگو ها را ارسال کنیم ۱۰ نورون اولیه را نورون واقعی نگرفتیم و تنها زمان اسپایک زدن آن هارا در نظر میگیریم.

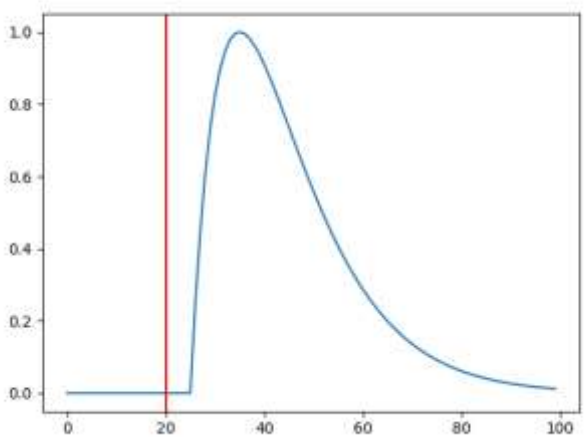
دو الگویی که برای شبکه در نظر گرفتیم الگوی [1, 2, 1, 1, 2, 0, 0, 0, 0, 0] [0, 0, 0, 0, 0, 1, 0, 0, 1, 2] که الگوی شماره یک هر ۱۰ استپ و الگوی شماره ۲ هر ۵ استپ به شبکه داده میشود و در زمان های غیر از این زمان ها یکی از نورون های ورودی به طور رندوم اسپایک میزند این روند تا ثانیه ۵۰۰ ادامه پیدا میکند به اصطلاح تا این ثانیه شبکه را تحت قانون یادگیری stdp که در تمرین قبل پیاده کردیم اموزش میدهیم بعد از این ثانیه چند ثانیه به شبکه جریان ورودی نمیدهیم تا پتانسیل ها کاهش پیدا کنند و سپس در یک زمان خاص الگو شماره یک و بعد از چند ثانیه الگو شماره دو و بعد از چند ثانیه جریان رندوم به شبکه میدهیم سعی کردیم جریان رندوم که میدهیم را منصفانه در نظر بگیریم که شانس اسپایک زدن آن ها بیشتر شود.

این فرایند را تحت کد زیر پیاده سازی کردیم:

```
def get_input_spikes(i, freq_pattern1=10, freq_pattern2=5):
    if i % freq_pattern1 == 0:
        state = 4
    elif i % freq_pattern1 == 1:
        state = 3
    elif i % freq_pattern2 == 0:
        state = 2
    elif i % freq_pattern2 == 1:
        state = 1
    else:
        state = 0

    if i in [2300, 2302, 2304]:
        state = 4
    elif i in [2301, 2303, 2305]:
        state = 3
    elif i in [2400, 2402, 2404]:
        state = 2
    elif i in [2401, 2403, 2405]:
        state = 1
    elif i in [2500, 2501, 2502 , 2503,2504]:
        state = 0
    elif i > 2000:
        return [0] * n

    if state == 4:
        # pattern 2
        return [0, 0, 0, 0, 0, 1, 1, 1, 1, 1]
    elif state == 3:
        # pattern 2
        return [0, 0, 0, 0, 0, 0, 1, 0, 0, 1]
    elif state == 2:
        # pattern 1
        return [1, 1, 1, 1, 1, 0, 0, 0, 0, 0]
    elif state == 1:
        # pattern 1
        return [0, 1, 0, 0, 1, 0, 0, 0, 0, 0]
    elif state == 0:
        # Random firing
        a = [0] * n
        fire = math.floor(random.random() * n)
        a[fire] = 1
        return a
```

دو نورون خروجی را از کلاس نورون هایی که در پروژه شماره دو انجام دادیم تهیه کردیم در هر استپ زمانی طبق فرمول $w_i \varepsilon(t - t_j^{(a)} - d_i)$ جریان ورودی دریافتی از هر نورون ورودی را محاسبه میکنیم و در نهایت با پتانسیلی که بدست میاوردیم نورون های خروجی را اپدیت میکنیم و طبق اینکه آیا فایری اتفاق افتاده یا نه وزن ها را طبق قانون یادگیری stdp اپدیت میکنیم.

در این قسمت تابع اپسیلون هنگام پیاده سازی کمی با مشکل های مهندسی کار روبرو هستیم چون این تابع نیاز دارد که هیستوری فایر های قبل را داشته باشد تا بتواند طبق دیلی نورون ها و تابع نمایی درون فرمول آن اثر وزن ها را اعمال کند این موضوع را با ذخیره سازی تمام هیستوری فایر های قبلی حل کردیم که توضیحات پیاده سازی آن از این گزارش خارج است و بیشتر بحث های مهندسی پشت آن وجود دارد.



نتیجه نهایی تابع spike response function مشابه حالت بالا شد (در ثانیه ۲۰ فایر اتفاق افتاده و دیلی ۵ ثانیه ای اعمال میشود)

```
def spike response function(s, time constant controlling=10):
    h = 1
    if s <= 0:
        h = 0
    result = (s / time_constant_controlling) * math.exp(1 - s / time_constant_controlling) * h
    if result < 0.01:
        return 0
    return (s / time_constant_controlling) * math.exp(1 - s / time_constant_controlling) * h
```

مراحل آپدیت پتانسیل نورون های خروجی:

```
# output neuron 1
input temp = 0
for i in range(n):
    for past_fire_time in update_queue[i]:
        input_temp += w1[i] * spike_response_function(current_time - past_fire_time - d1[i])
fired = output1.update(j=current_time, input=input_temp)
```

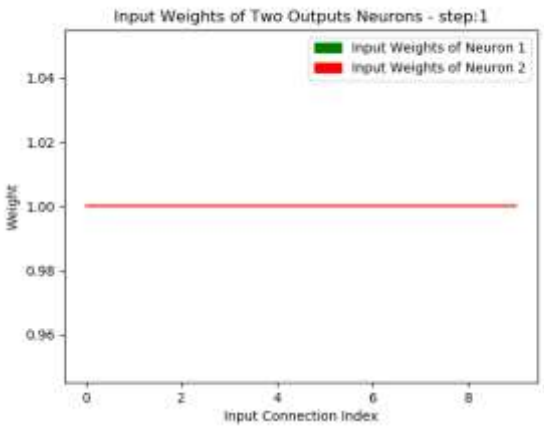
و بعد از این مرحله قانون یادگیری را برای تمام وزن های ورودی اعمال میکنیم:

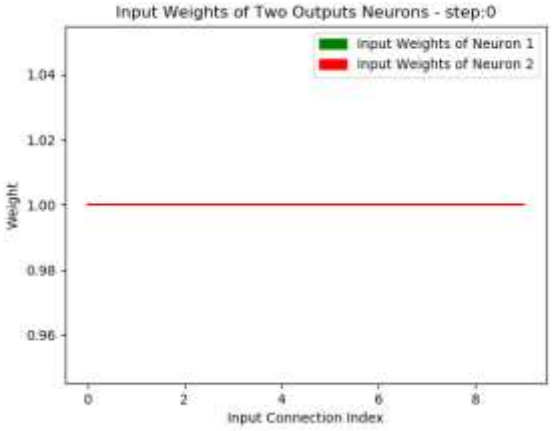
```
for i in range(n):
    ## X
    x1[i][current time] = x1[i][current time - 1] + ((-x1[i][current time - 1]) / tau plus + fire pattern[i])
    if x1[i][current time] < 0:
        x1[i][current_time] = 0
    ## Y
    y_fire_pattern = 0
    if fired > 0:
        y_fire_pattern = 1
    y1[i][current time] = y1[i][current time - 1] + ((-y1[i][current time - 1]) / tau minus + y fire pattern)
    if y1[i][current time] < 0:
        y1[i][current_time] = 0

    ## W
    delta_w = (-a_minus * A_minus(w1[i - 1]) * y1[i][current_time - 1] * fire_pattern[i]) + (
                a plus * A plus(w1[i - 1]) * x1[i][current time - 1] * y fire pattern)
    w1[i] = w1[i] + delta w
    if w1[i] < 0:
        w1[i] = 0.01
```

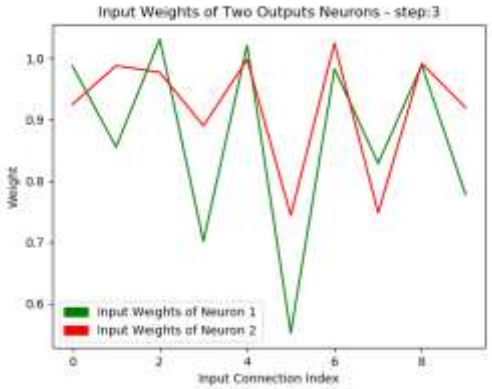
نکته ای که در نظر گرفتیم اتصالات ممکن است در حین یادگیری منفی یا مساوی صفر شوند در این صورت کل یادگیری متوقف میشود برای جلو گیری از این حالت عدد 0.01 را به عنوان کوچکترین مقادیری که وزن ها میتوانند بگیرند در نظر میگیریم

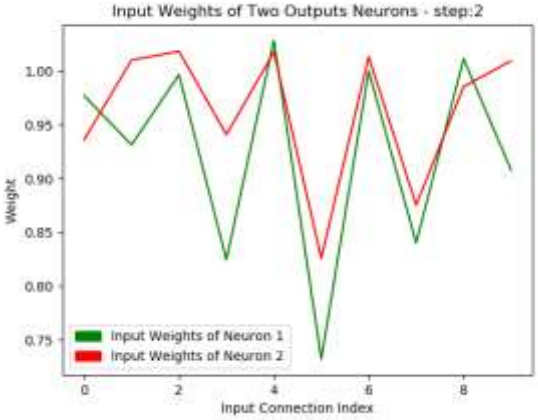
در زیر ۲ سناریو تغیرات وزن شبکه را در نظر میگیریم، هر نقطه افقی در نمودار نشان دهنده مقدار یکی از وزن های اتصالی است و هر نمودار متعلق به وزن های ورودی یکی از نورون های خروجی در نظر گرفته میشود. روند تغیرات وزن نورون یکبار با وزن های آغازین رندوم و یک بار با وزن های آغازین یکسان دنبال میکنیم

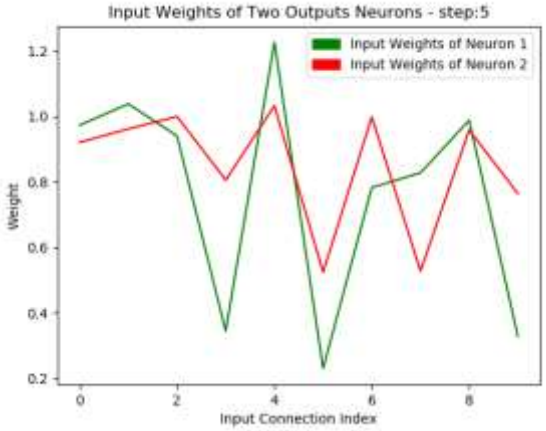
وزن های آغازین یکسان:

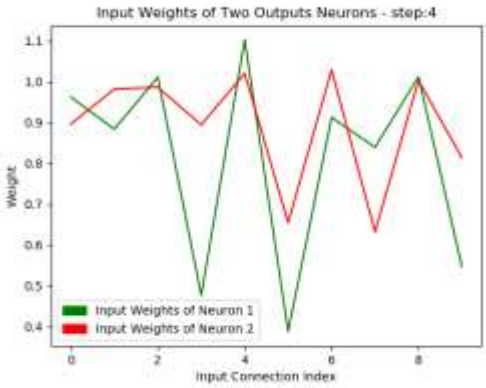


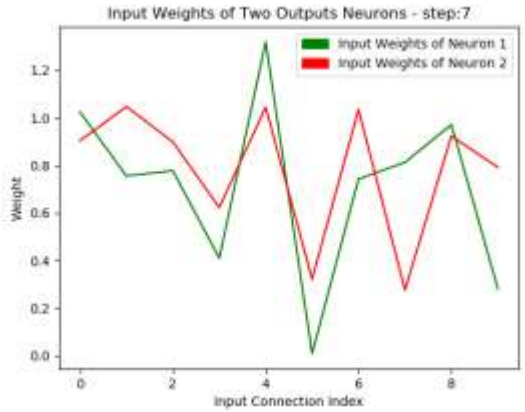


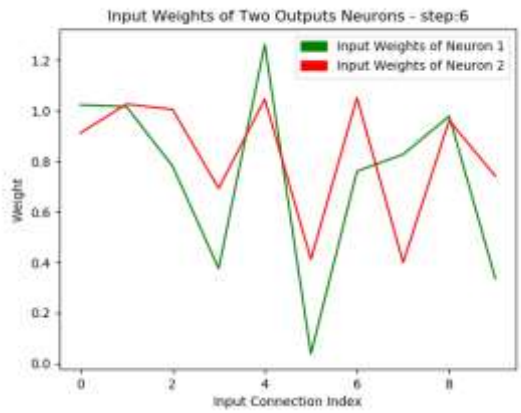


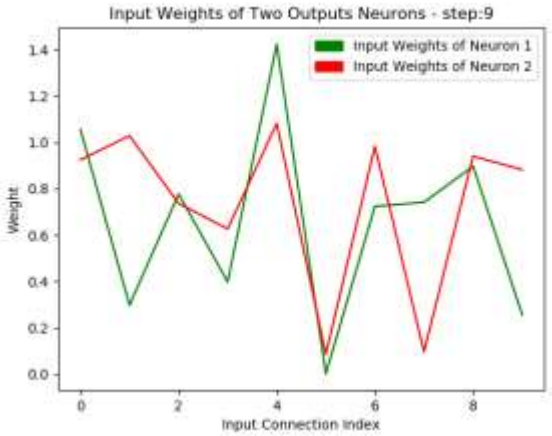


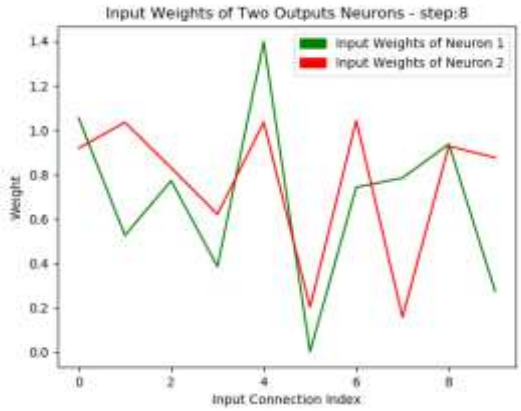


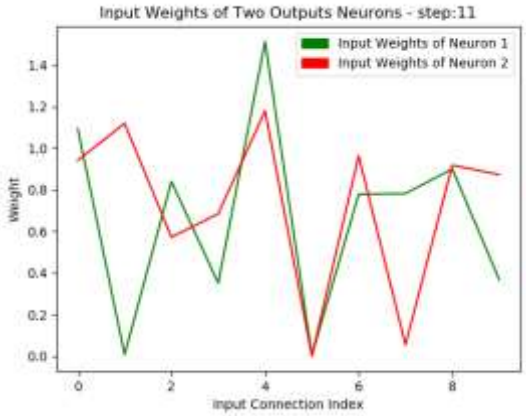


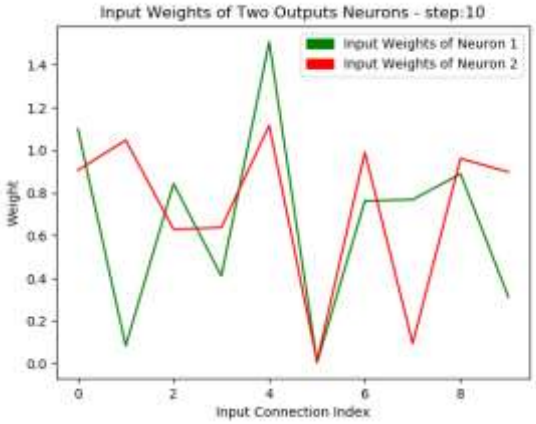








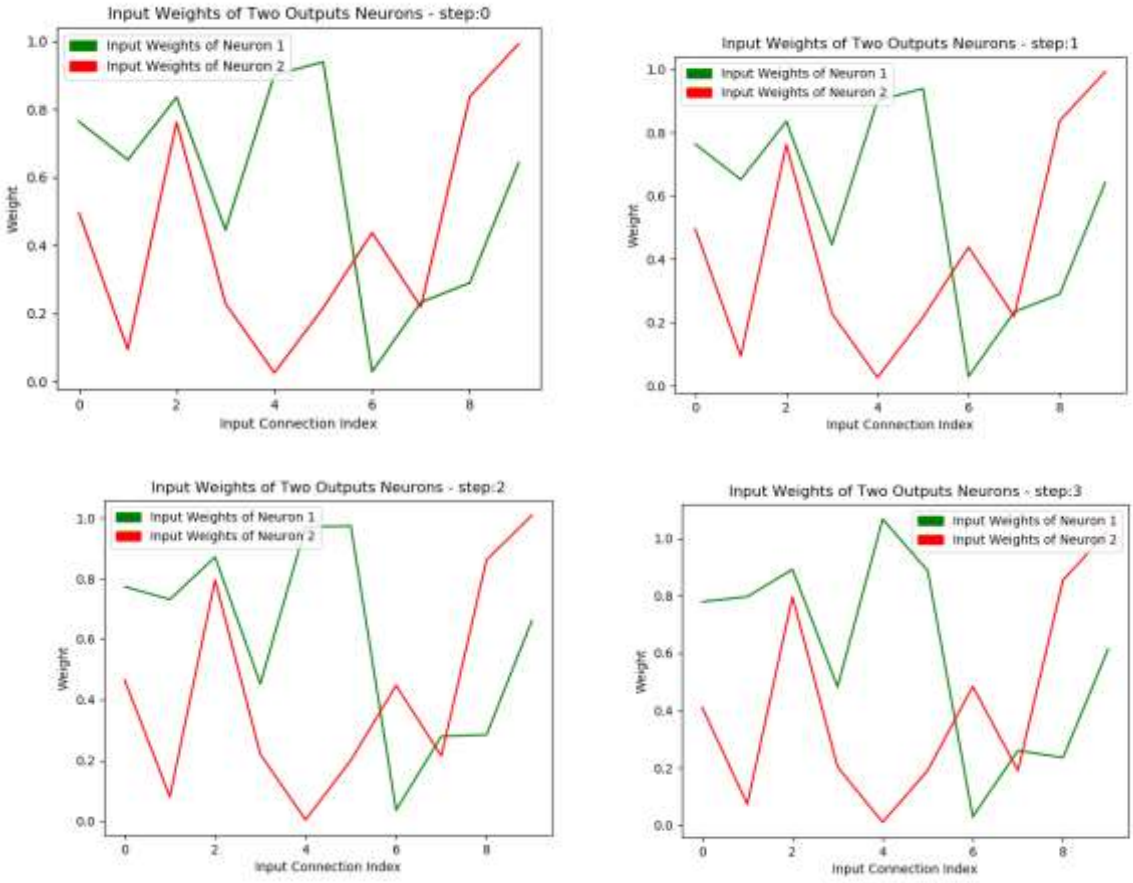


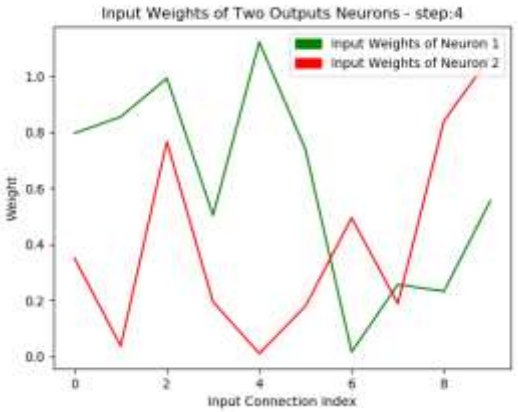


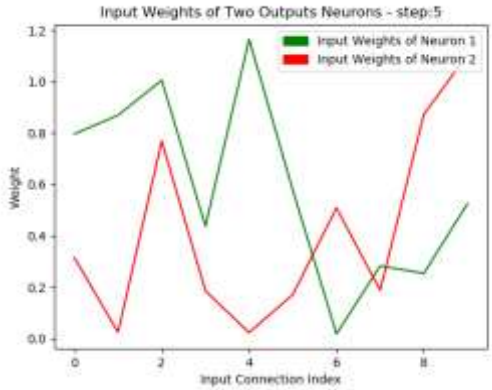
در این حالت به طور شهودی حرکت مدل را در فضای پارامتر ها به خوبی مشاهده میکنیم و میبینیم که شبکه در حال یادگیری بعضی از الگو ها هست و در بخش اول و اخر بعضی وزن ها بشدت کم و بعضی به شدت افزایش می یابد چیزی که مشاهده کردیم به خاطر اینکه وزن های اولیه یکسان بود ولی دیلی ها رندوم بودند هر دو نورون به سمتی رفتند که وزن های یکسانی داشته باشند هنگامی که این آزمایش را با دیلی های یکسان انجام دادیم هر دو نورون وزن های خیلی نزدیک به هم پیدا کردند و نمودار هایشان تقریبا روی هم افتاد همانطور که در بعضی از قسمت های این آزمایش هم مشاهده میشود قسمت های همپوشانی،

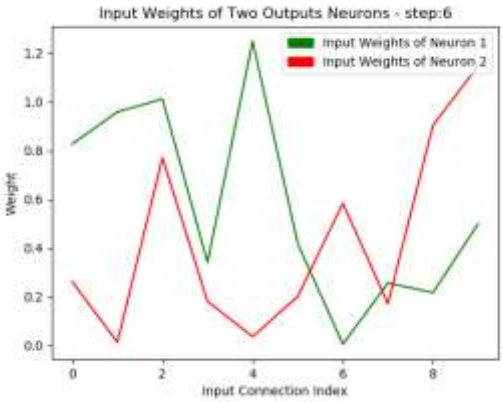
اتصالاتی هستند که دیلی های نزدیک به هم دارند. نکته بعدی که دیدیم و در نمودار های بعدی نیز نشان میدهیم هر دو نورن تنها یک پترن را یاد گرفتند که اگر غیر از این بود باید هر یک به فضای پارامتر به سمت یادگیری یکی از الگو ها میرفت در اینجا توجیهی که میکنیم این است که هنگامی که ما میخواستیم الگو های ورودی را بچینیم همیشه الگو دوم بعد از ۵ ثانیه از الگوی اول میاید، یک الگوی کلی تر هر دو این الگو ها را در بر میگیرد و انگار یک الگوی کلی شامل ۲ الگو و مقداری نویز هستیم که هر 10 ثانیه یکبار تکرار میشوند . آزمایش دیگری که انجام دادیم این بود که این الگوی بزرگتر را از بین ببریم و انتخاب اینکه بعد از چند ثانیه کدام الگو اجرا شود را رندوم در نظر گرفتیم و نتایج بدست آمده تفاوت چندانی نداشت و تنها همپوشانی کمتری مشاهده کردیم ولی همچنان در یک فضای نزدیک به هم وزن ها حرکت میکردند.

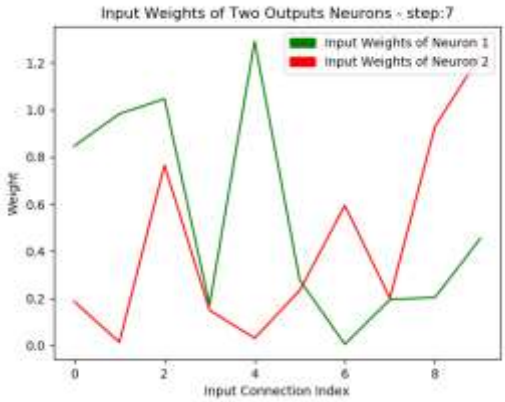
وزن های آغازین رندوم:

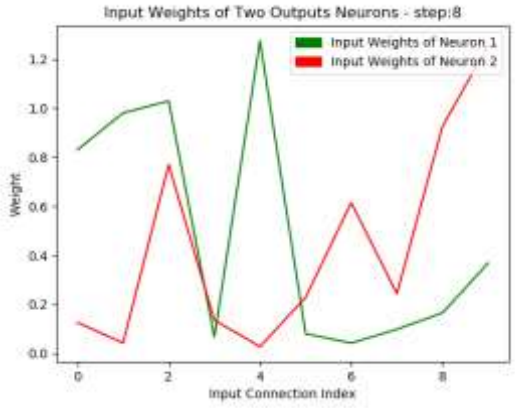


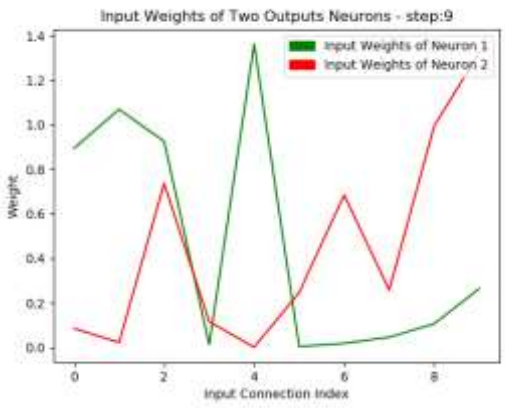

Input Weights of Two Outputs Neurons - step:4
Input Weights of Neuron 1
Input Weights of Neuron 2
Weight
Input Connection Index

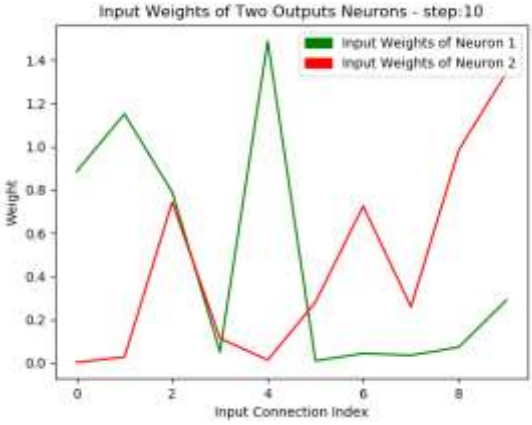

Input Weights of Two Outputs Neurons - step:5
Input Weights of Neuron 1
Input Weights of Neuron 2
Weight
Input Connection Index

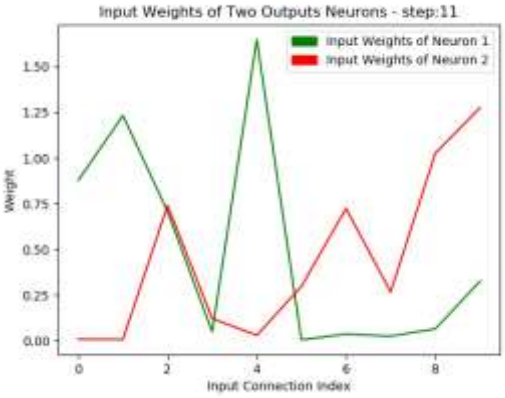

Input Weights of Two Outputs Neurons - step:6
Input Weights of Neuron 1
Input Weights of Neuron 2
Weight
Input Connection Index


Input Weights of Two Outputs Neurons - step:7
Input Weights of Neuron 1
Input Weights of Neuron 2
Weight
Input Connection Index





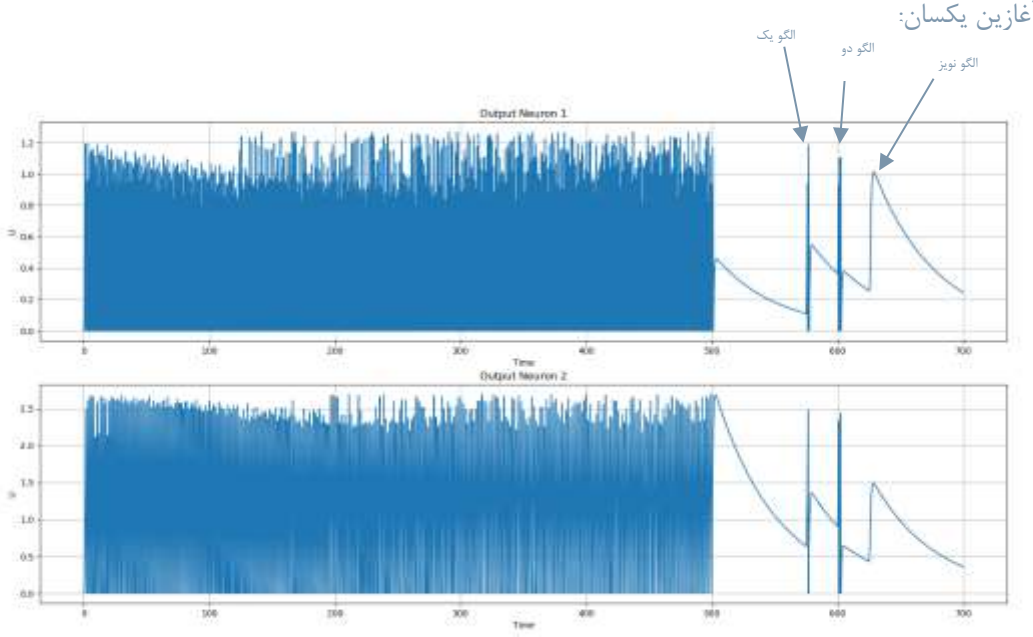




از قبل شنیده بودیم که در شبکه های عصبی مصنوعی استفاده از وزن های اولیه یکسان اشتباه است و این موضوع را در این بخش هم مشاهده کردیم هنگامی که وزن های اولیه را به طور رندوم وزندهی کردیم مشاهده کردیم که دو نورون در فضا به خوبی از هم فاصله گرفتند و میتوان گفت تا حدودی هر نورون مسولیت یاد گیری یکی از الگو ها را در نظر گرفته، حتی با وجود اینکه در این آزمایش مشکل الگوی بزرگ تر که مطرح کردیم را حل نکردیم ولی نتایج تا حدی قابل قبول است و هر نورون یک الگو را حساس شده است حال کلمه حساس شده است را در آزمایشات زیر توجیه میکنیم.

برای اینکه دید بهتری به نحوه عملکرد نورون های خروجی و فرایند یادگیری داشته باشیم پتانسیل نورون های خروجی را مورد بررسی قرار دادیم

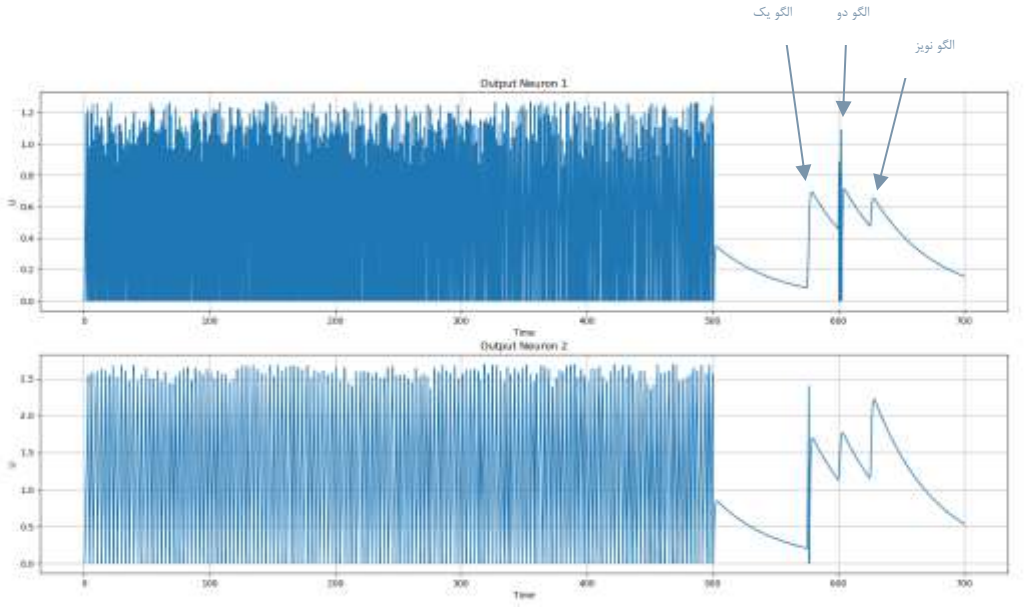
وزن های آغازین یکسان:



هر دو نورون در ابتدای فرایند یادگیری چون وزن های یکسانی در محیط داشتند به سرعت با کوچکترین نویزی شروع به نوسان و اسپایک زدن میکردند ولی رفته رفته بعد از اعمال قانون یادگیری stdp در نورون شماره یک بعد از ثانیه ۱۲۰ به بعد و نورون دو از ثانیه ۲۰۰ به بعد شدت اسپایک ها کاهش پیدا میکند و اگر با دقت بیشتری نگاه کنید بازه های بین اسپایک ها رفته رفته منتظم تر میشود و به عدد وقفه ی ۵ ثانیه ای که بین دو الگو بود نزدیک میشود و میبینیم که رفته رفته شبکه به الگو های داده شده حساس میشود و هنگامی که آن نورون هارا مشاهده میکند اسپایک میزند و در حضور نویز فعالیت کمتری دارد. و رفته رفته فعالیت شبکه کاهش پیدا میکند و فقط اطراف بازه های زمانی الگو ها فعالیت میکند. در بالا تر ادعا کردیم که هر دو نورون بعد از مدتی در فضای وزن ها به هم نزدیک میشوند و هر دو نورون تقریبا یک چیز را یاد میگیرند این موضوع در نموردار بالا هم مشخص است: از دو منظر یک اینکه در یک نورون خروجی بین دو فاصله زمانی که الگو مخالف اجرا میشود همچنان پتانسیل به طور چشم گیری بالا میرود و با احتمال کمتری اسپایک میزند ولی همچنان پتانسیل افزایش میابد. از منظر دیگر بعد از ثانیه ۵۰۰ دیگر به شبکه جریان داده نمیشود و در لحظه ۵۰۰ ابتدا الگوی شماره یک و سپس الگو شماره دو و سپس در مدت کوتاهی نویز به شبکه داده میشود همان طور که مشاهده میکنید هنگامی که هر دو الگوی اولیه وارد میشوند هر دو نورون خروجی باهم اسپایک میزنند این بدان معنی است که هر دو نورون هر دو الگو را یاد گرفته اند.

نکته دیگر این است که شبکه نسبت به نویز وارد شده هم واکنش نشان نمیدهد پتانسیل آن افزایش میابد ولی در شرایط منصفانه به سطح اسپایک زدن نمیرسند.

وزن های آغازین رندوم:

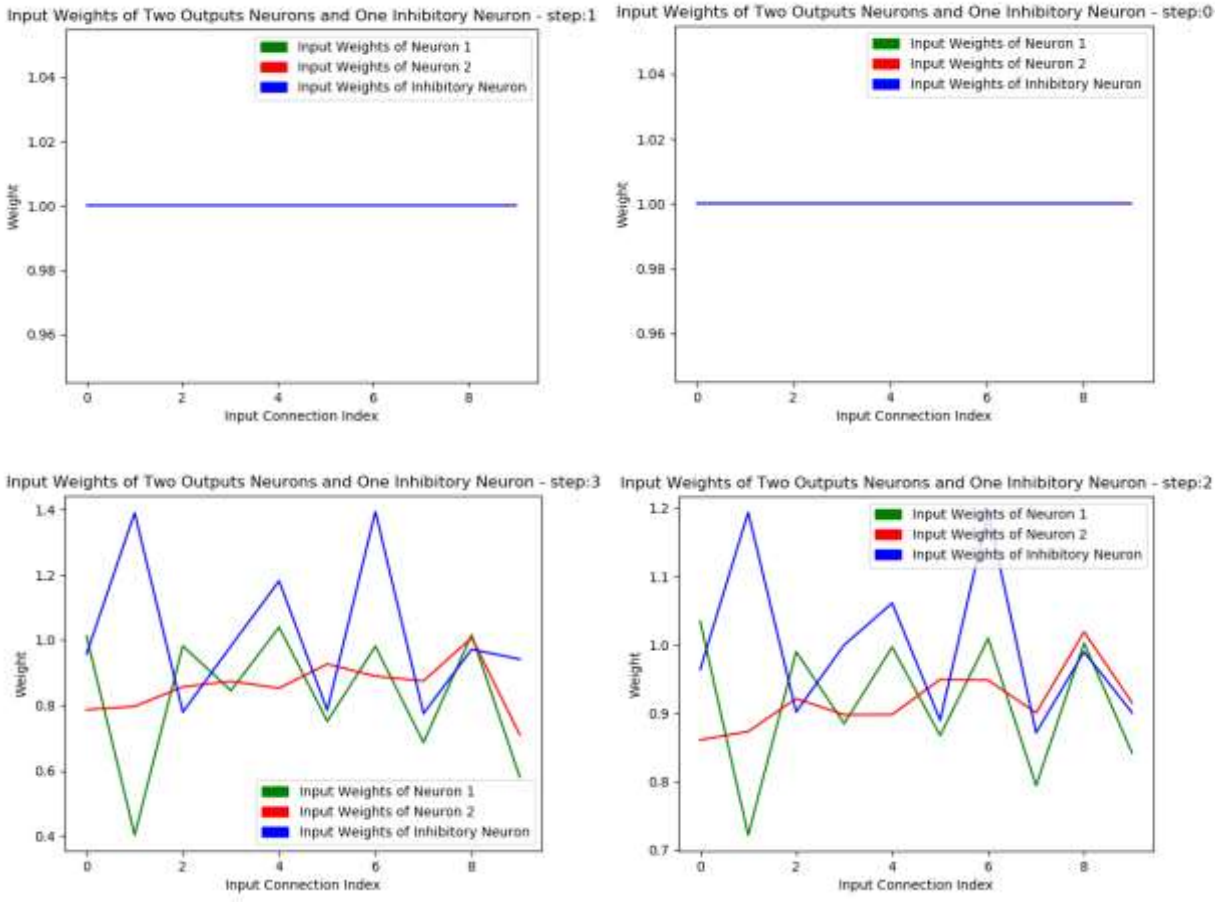


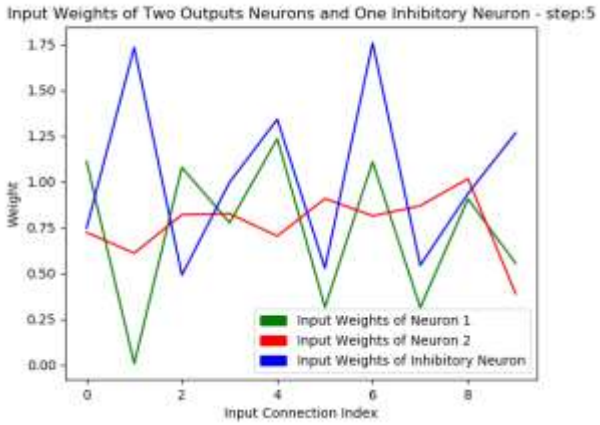
در این حالت دیگر کاهش رفته رفته فعالیت شبکه و حساس شده به الگو ها خیلی مشهود نیست ولی همچنان اندکی این حساسیت دیده میشود. ولی نکته خیلی جالب تر این است که همانطور که در قسمت تست (از ثانیه ۵۰۰ به بعد) مشاهده میکنیم که نورون اول به الگو دوم حساس شده و نورون دوم به الگوی اول حساس شده و آن را اموزش دیده است و این همان چیزی بود که از مشاهده وزن های خروجی احساس میکردیم اتفاق میافتد چون وزن های ورودی دو نورون در فضا از هم فاصله گرفتند این دونورون رفتار متفاوتی نشان میدهند و هر یک یکی از الگو هارا یاد گرفته ولی یک سری مشکلات نسبت به روش قبلی میبینیم نورون شماره ۲ درست است که به الگوی شماره یک حساس شده ولی در فرایند یادگیری نتوانسته به نویز های ورودی مقاومت پیدا کند و حتی حساسیت خیلی قوی ای مشابه حالت قبل نسبت به الگو دو پیدا نکرده (تنها یک اسپایک زده است در حالت قبل بیشتر اسپایک زده میشد) این اتفاق را اینطور توجیه میکنیم که در حالت قبل هر دونورون هر دو الگو را اموزش میدیدند و این بدان معنی است که در زمان یکسان داده ورودی متعلق به یک الگوی کلی بیشتری در روش قبل دیده میشد ولی در این روش تعداد داده های اموزشی برای هر نورون نصف میشود و اگر زمان آموزش افزایش یابد این موضوع بر طرف میشود و میتوانیم شبکه های مقاوم در برابر نویز خوبتری بسازیم که هر یک از نورون ها به یکی از الگو ها حساس باشد.

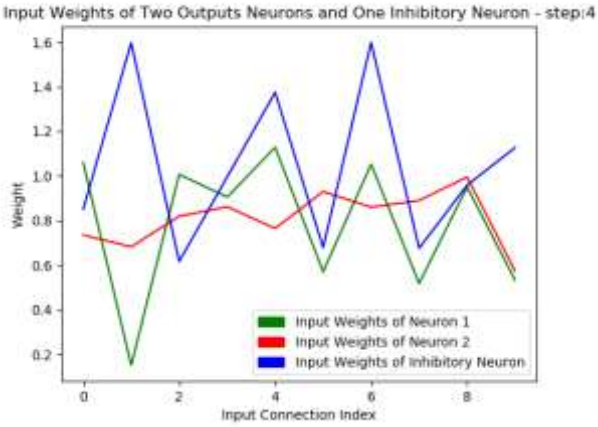
تمرین شماره سه:

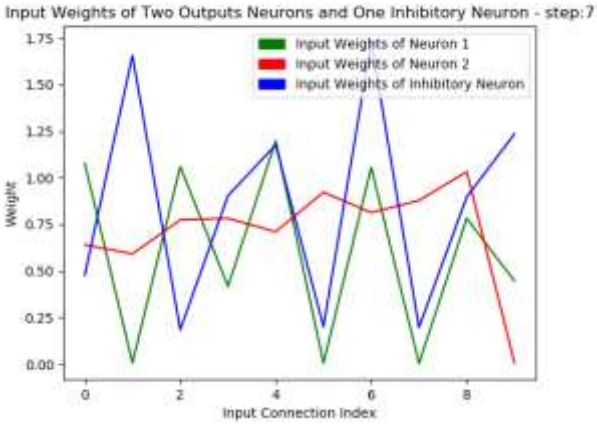
این تمرین کاملا مشابه قبلی هر کاری برای دو نورون خروجی انجام میدادیم را برای ۳ نورون انجام میدهیم که نورون جدید یک نورون مهاری است که به دو نورون خروجی متصل است در تمرین دو، دیلی اتصالات رندوم بود ولی در این حالت اتصال نورون های خروجی اصلی ۱ می باشد و دیلی اتصال نورون مهاری ۰ میباشد و این نورون با دو اتصال با دیلی ۰ به دو نورون خروجی متصل است و هنگام فایر کردن این ۲ نورون را اینهیبیت میکند بر روی تمامی اتصالات قانون یادگیری stdp را پیاده سازی کرده ایم و عملیات اینهیبیت کردن را از بخشی که در پروژه قبلی پیاده سازی کرده بودیم برداشتیم آزمایشات قبلی را برای این شبکه جدید هم پیاده سازی کردیم.

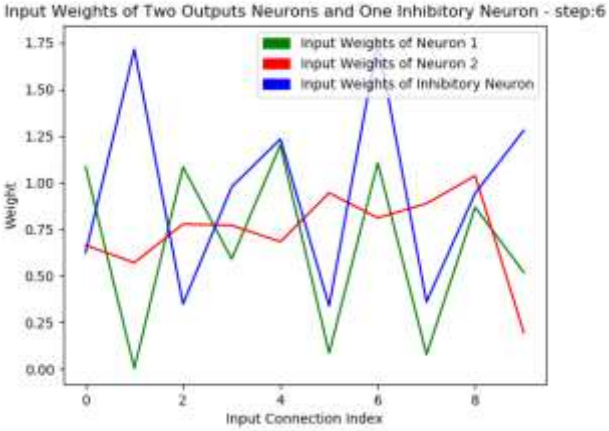
وزن آغازین یکسان:

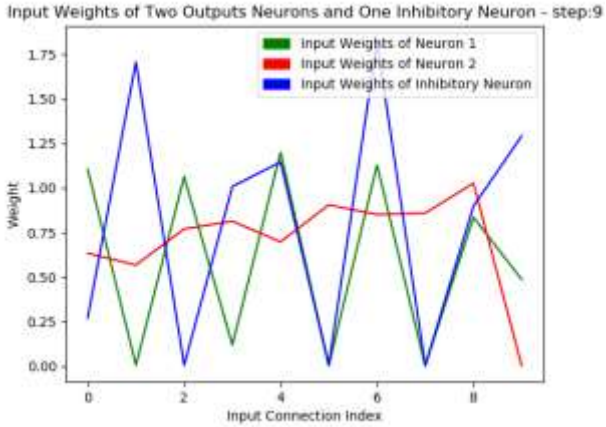


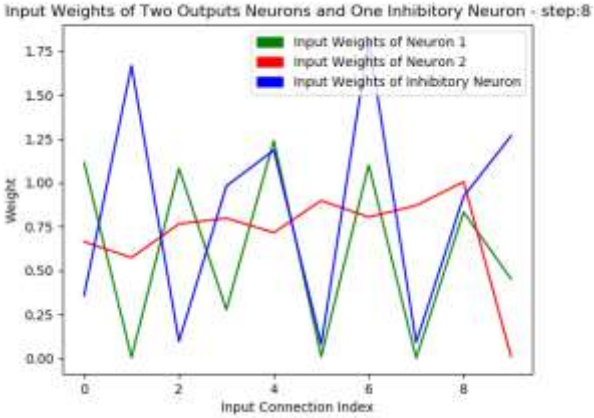

Input Weights of Two Outputs Neurons and One Inhibitory Neuron - step:5
Input Weights of Neuron 1
Input Weights of Neuron 2
Input Weights of Inhibitory Neuron
Weight
Input Connection Index

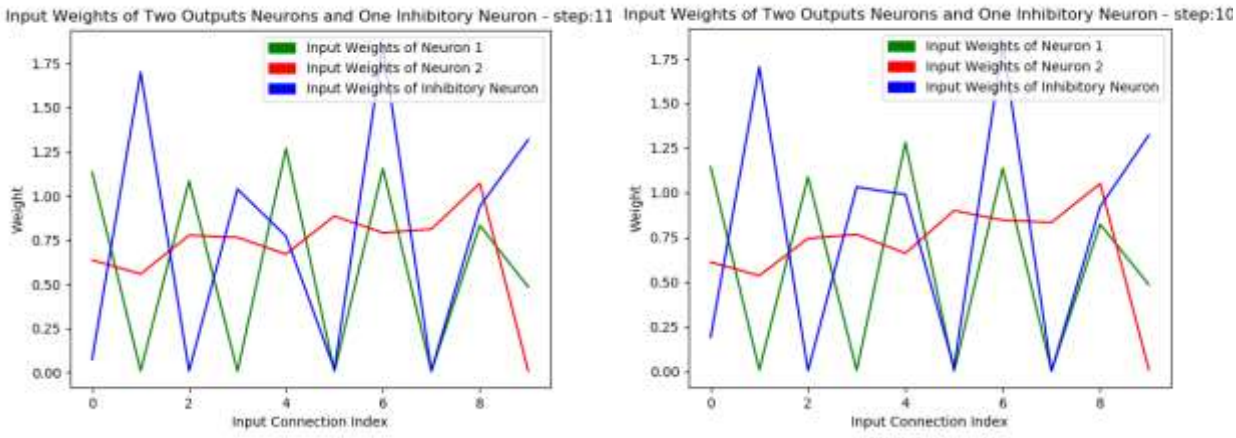

Input Weights of Two Outputs Neurons and One Inhibitory Neuron - step:4
Input Weights of Neuron 1
Input Weights of Neuron 2
Input Weights of Inhibitory Neuron
Weight
Input Connection Index


Input Weights of Two Outputs Neurons and One Inhibitory Neuron - step:7
Input Weights of Neuron 1
Input Weights of Neuron 2
Input Weights of Inhibitory Neuron
Weight
Input Connection Index


Input Weights of Two Outputs Neurons and One Inhibitory Neuron - step:6
Input Weights of Neuron 1
Input Weights of Neuron 2
Input Weights of Inhibitory Neuron
Weight
Input Connection Index

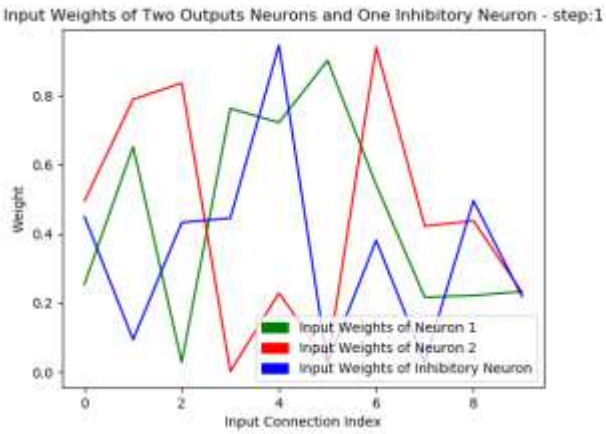

Input Weights of Two Outputs Neurons and One Inhibitory Neuron - step:9
Input Weights of Neuron 1
Input Weights of Neuron 2
Input Weights of Inhibitory Neuron
Weight
Input Connection Index

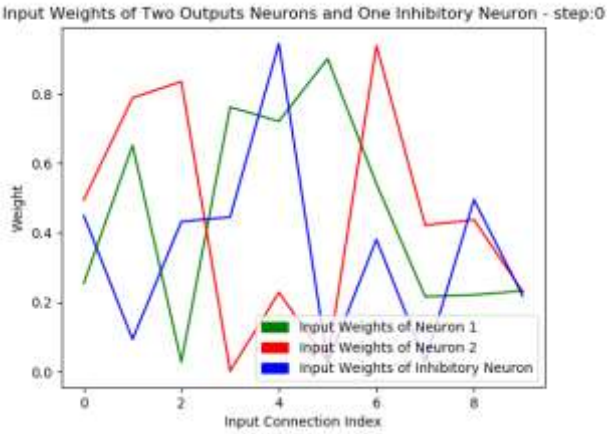

Input Weights of Two Outputs Neurons and One Inhibitory Neuron - step:8
Input Weights of Neuron 1
Input Weights of Neuron 2
Input Weights of Inhibitory Neuron
Weight
Input Connection Index

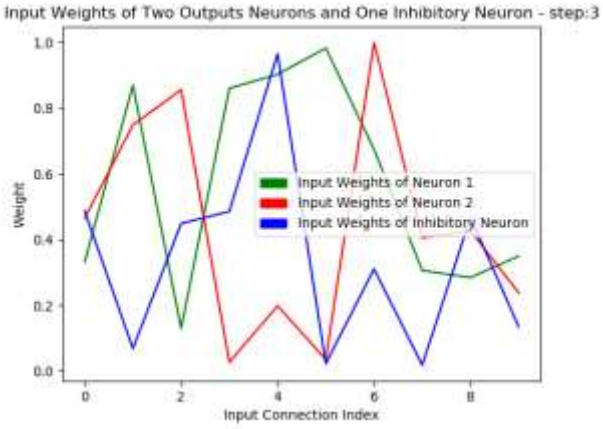


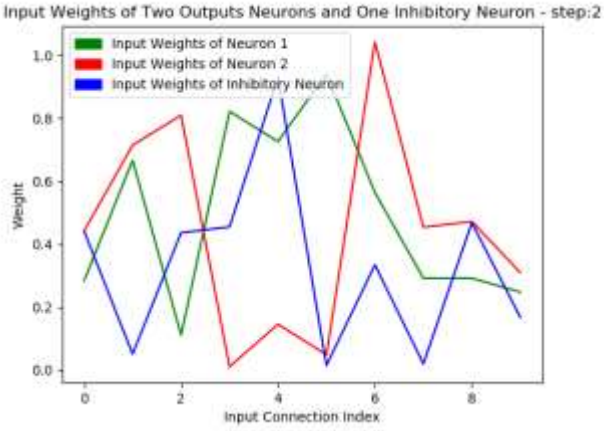
اولین نکته ای که مشهود است این است که از یادگیری نورون قرمز رنگ(شماره ۲) جلو گیری میشود و نمیتواند به قدر کافی از حالت اولیه خود جا بجا شود و نکته دیگر این است که نورون مهاری خود نیز شروع به فعالیت در راستای یادگیری کرده است و در بعضی از نقاط فضا الگویی مشابه الگو نورون شماره ۲ را به خود گرفته و در خیلی از اتصالات به مقدار صفر میرسند حال اگر به اینکه این نورون های صفر شده کدام ها هستند توجه کنید معلوم میشود که این اتصالات اطراف نورون هایی هستند که در الگو ها مقدار ۲ دارند و دو بار پشت سر هم می آیند و همچین افتادن این اتفاق در شبکه کاملا منطقی است زیرا نورون نویز های ما میتوانند با احتمال یک دهم در هر استپ زمانی فعال شوند ولی در این دو الگو یکی از نورون ها دوبار پشت سر هم اتفاق میافتند و اگر شبکه وزن این نورون را زیاد کند و بقیه را کم کند مشابه این است که اثر نویز را کم کند و به نویز مقاوم شود و فقط به نورون هایی که اطلاعات در خود دارند توجه کند (واقعا این حرکتشون جالب بود!! و جالب تر اینکه این اتفاق با قانون stdp افتاد) این اتفاق فقط در الگو شماره دو اینگونه بوده و الگوی شماره یک هر دو نورون مهاری و نورون سبز ما به نحوی متفاوت شروع به یادگیری الگو میکنند و از همین وزن ها میتوان حدس زد که نورون مهاری هر دو الگو را یاد گرفته و اثر یادگیری در نورون قرمز را به کلی از بین برده و نورون سبز هم دقت بالایی نخواهد داشت. حال مشابه حالت قبل بعد از اینکه تغیرات وزن با مقادیر آغازین رندوم را هم بررسی کردیم این نکته را در نمودار پتانسیل های نورون ها بررسی میکنیم.

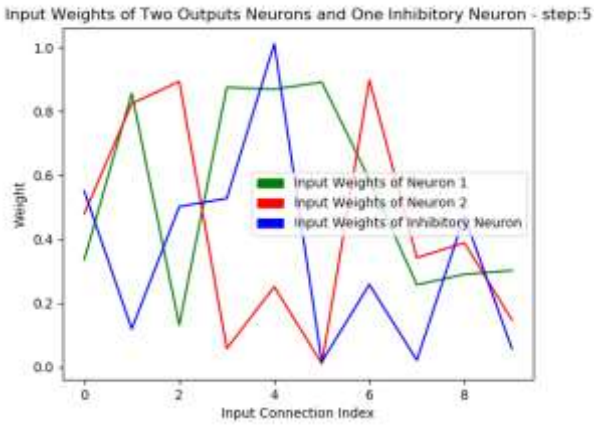
مقادیر آغازین رندوم:

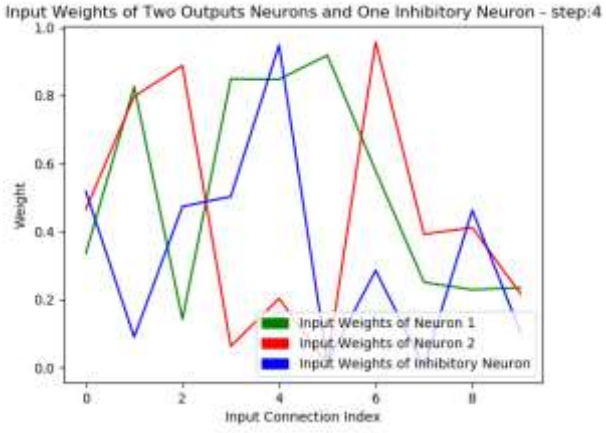


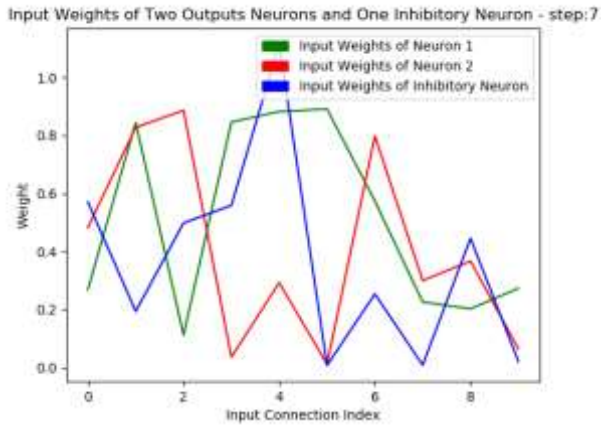


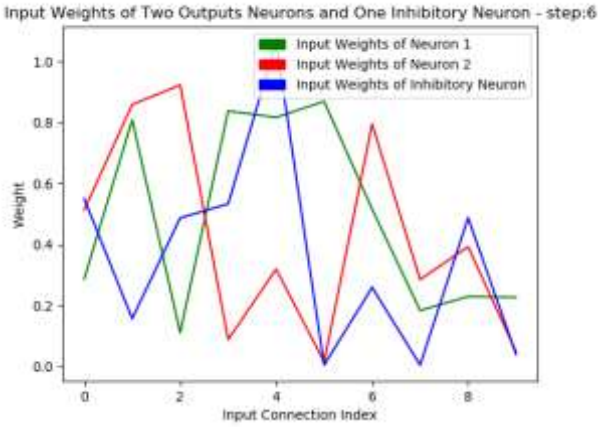


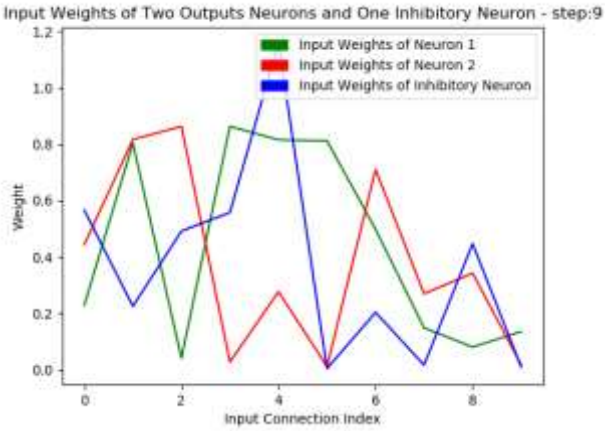


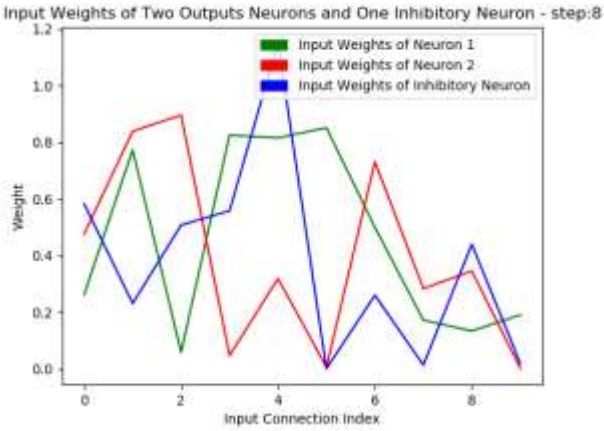


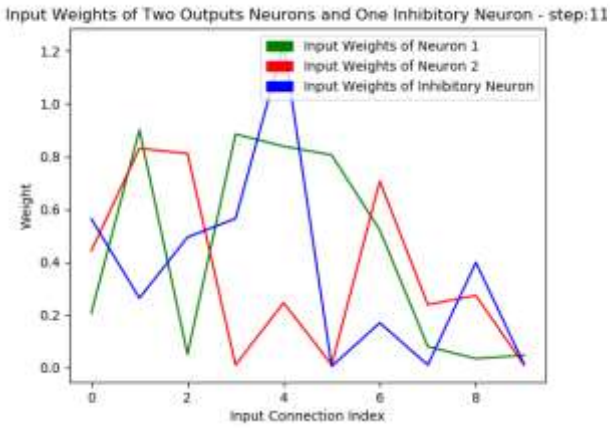


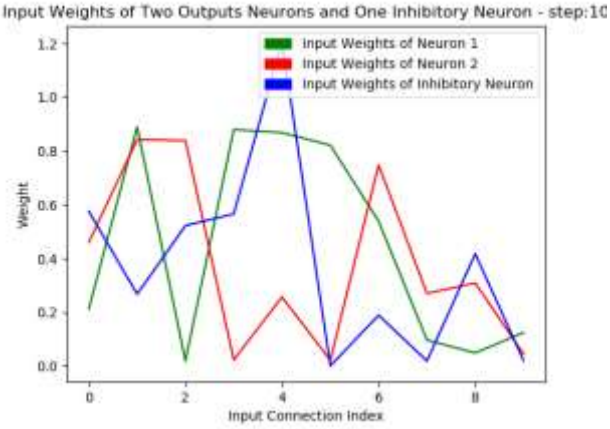








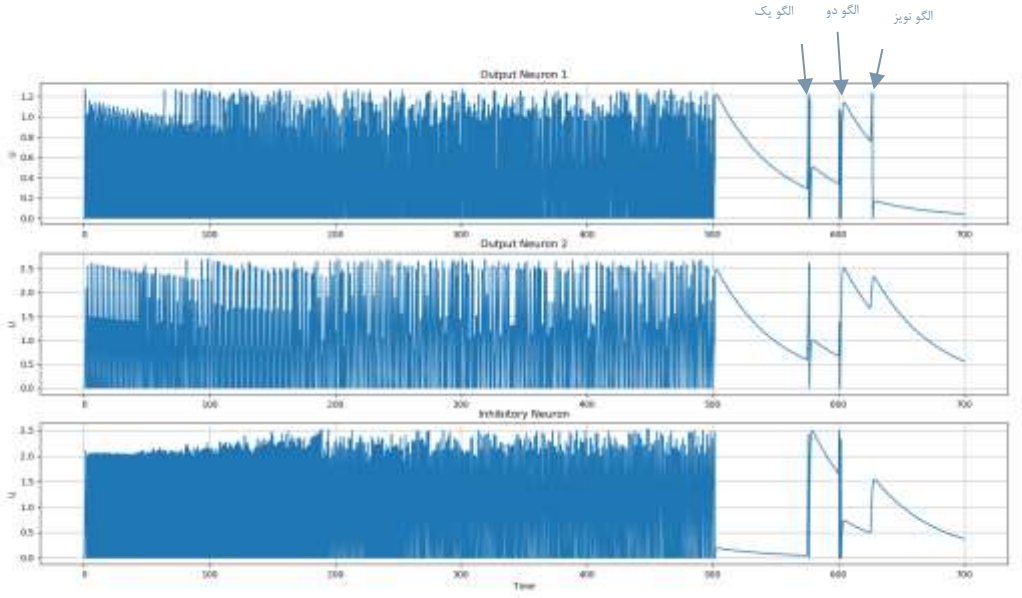




در این آزمایش اثر نکاتی که در مرحله قبل دیدیم هم دیده میشود ولی نکته ی جالب تر، هر سه نورون به نورون ورودی شماره ۴ توجه دارند این نورون برای شناسایی الگو شماره یک استفاده میشود و تغیرات زیادی روی این وزن اتفاق نیافتاده است و ب طور کلی میتوانیم بگوییم اثر قانون یادگیری stdp در این حالت کمتر دیده میشود این طور توجیه میشود که از شانس خوب وزن های اولیه نورون مهاری حالتی مشابه حالت های لرن شده قبلی دارد و میتواند اثر هر دو الگو را با وزن های اولیه خود هم ببیند و به آن

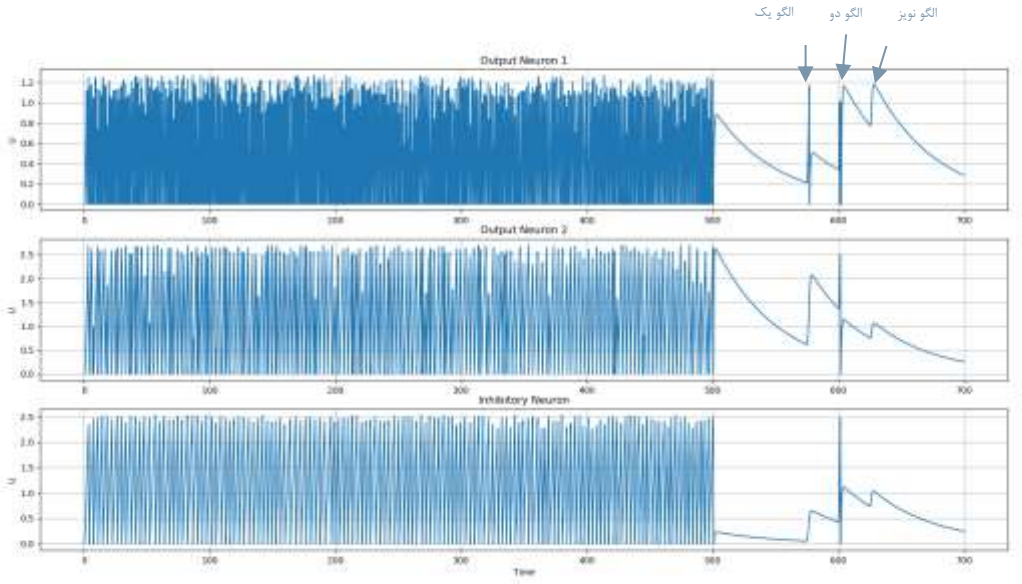
ها حساس باشد و نه تنها خودش خیلی در فضای پارامتری جابجا نمیشود بلکه مانع این میشود که نورون های دیگر هم آموزش کاملی داشته باشند و وزن های آن ها نیز از حالت اولیه خیلی نتواند جابجا شود.

پتانسیل نورون های خروجی در حالت وزن های اولیه یکسان:



وقتی میگوییم نورونی به خوبی آموزش دیده است به این معنی است که به الگو های (اطلاعات داخلی داده ها) حساس باشد و به نویز هم مقاوم باشد، چیزی که در این ۳ نمودار میبینیم تمام اتفاقاتی که در تمرین دو گفتیم اتفاق افتاده، بعد از یک مدت تناوب نورون ها بر روی بازه زمانی ای میشود که الگو ها تکرار میشوند ولی اتفاقی که در این شبکه افتاده است نورون مهاری بهتر از دو نورون دیگر آموزش دیده و در فرایند یادگیری خود بیشتر نوسان داشته و در نتیجه وزن های بهتری را کشف کرده و به نویز مقاومت پیدا کرده و بعد از هر اسپایک دو نورون دیگر را مهار کرده و جلوی اسپایک زدن آن ها را تا حدی گرفته و چون دیلی این نورون ۰ است سریع تر این کار را در اوایل آموزش که وزن ها یکسان است انجام میدهد و نمیگذارد بقیه نورون ها نوسان کنند و وزن های آن ها بیشتر تغییر کند و یادگیری در آن ها صورت بگیرد و تنها خودش آموزش پیدا میکند و همانطور که میبینیم ۲ نورون دیگر همچنان با نویز کار میکنند که در مقایسه این نورون ها با نورون های تمرین دو نتیجه میگیریم که آموزش در این ۲ نورون کمتر اتفاق افتاده است. حتی نورون شماره یک همچنان با نویز ها اسپایک میزند و این نشانه عدم آموزش درست است. میتوانیم نتیجه بگیریم خاصیت مهاری نورون مهاری یک نوع رقابت در شبکه ایجاد کرده و نورونی که، دیلی کمتری دارد زودتر آموزش میبیند و جلوی آموزش بقیه را میگیرد.

پتانسیل نورون های خروجی در حالت وزن های اولیه رندوم:



نتیجه ای که مشاهده میکنید خیلی وابسته به وزن های ورودی اولیه است به همین دلیل خیلی با تصوری که داشتیم همخوانی ندارد ولی همچنان میتوان مشاهده کرد که نورون شماره یک به هیچ وجه آموزش کاملی نداشته و همچنان با وجود اینکه اسپایک های زیادی ام زده است ولی همچنان به نویز مقاوم نیست و حتی الگو ها را هم به درستی یاد نگرفته است ولی چیزی که مشاهده میکنیم نورون مهاری الگو شماره دو را به درستی یاد گرفته و به نویز هم مقاوم شده است ودوباره مشاهده میکنیم که نورون مهاری بخاطر کارکردی که در شبکه دارد جلوی یادگیری کامل دو نورون باقی را گرفته و با آن ها رقابت میکند با توجه به اینکه اسپایک های کمی هم نسبت ب نورون های دیگر در فرایند آموزش زده ولی مشاهده میکنیم که این نورون با مقادیر اولیه خود هم به الگو ها حساس بود و نوساناتی که داشته کاملا منظم و در بازه های درست اتفاق افتاده و به خاطر دیلی کمتری که داشته جلوی یادگیری الگو ها را از نورون های دیگر میگیرد و باعث میشود نورون شماره یک مثلا کاملا نویز را یاد بگیرد و نتواند به طور چشمگیری در فضای پارامتر جابجا شود و اطلاعات مهمی که در زمان الگو ها باید مشاهده میکرد را بخاطر اینکه نورون مهاری جلوی آن را میگیرد نمیبیند و نمیتواند آموزش ببیند و همچنان نویز ها را یاد میگیرد.