گزارش سری 3

بامداد رفعتی - 99222041

برای پیاده سازی بخش اول پروژه (شبیه سازی STDP) به چند کلاس احتیاج داریم.

کلاس TypeOfNeuron: شمارنده ای است که مشخص می کند نورون ما مهاری است یا تحریکی.

کلاس LIFClass: کلاس LIF پیاده سازی شده در پروژه 1 با پارامترهای زیر است:

```
u_rest=-60,
resistance=9,
capacitance=0.7,
time=100,
dt=0.1,
u_reset=-65,
u_start=-80,
```

توابع این کلاس:

تابع run: برای جلو رفتن و استارت شدن زمان اجرای برنامه است.

تابع u-t-plot: برای رسم نمودار پتانسیل زمان.

تابع check_for_spikes: نقاط زمانی که در آن پتانسیل از پتانسیل آستانه بیشتر شده است را مشخص می کند.

کلاس LIFClassWithSingleStep: همان کلاس لیف نرمال، ولی با تابع جریان ورودی single step.

کلاس NeuronCluster: کلاس اتصال نورون ها به یک دیگر با تابع from_neuron_group.

حال برای بخش اول پروژه، کلاس NormalSTDP را با کران بالای amplify_val_pos و کران پایین amplify_val_neg برای مدیریت تغییر وزن با توابع زیر پیاده سازی می کنیم:

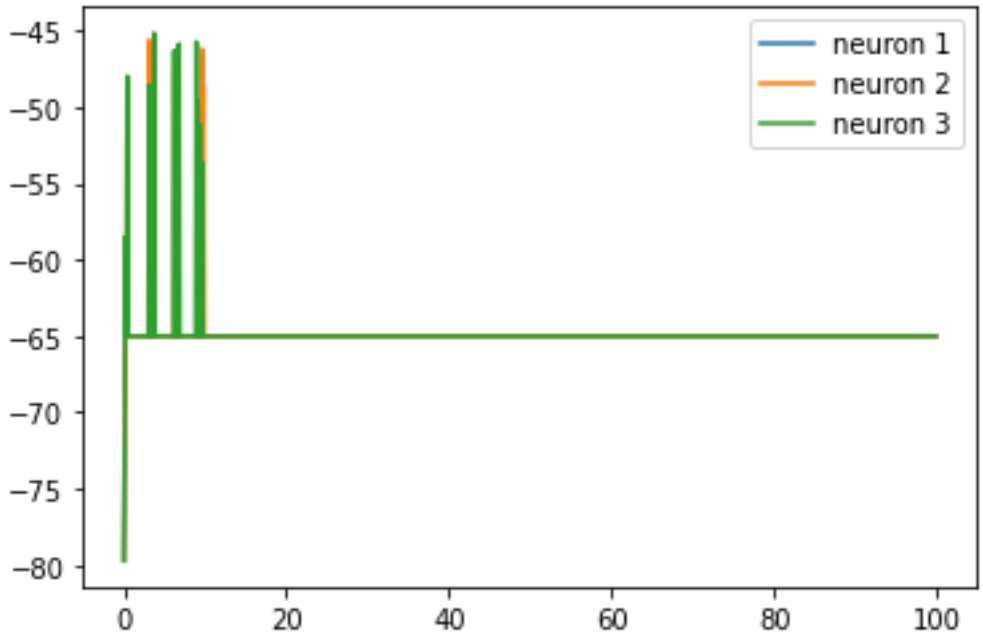
تابع learning process: تابع تغییر وزن بر اساس این که اول نورون پیشین یا نورون پسین اسپایک زده است.

تابع plot_weight: تابع رسم تغییرات وزن بین دو نورون در طول زمان.

وزن های اولیه بین 3 نورون به صورت زیر است:

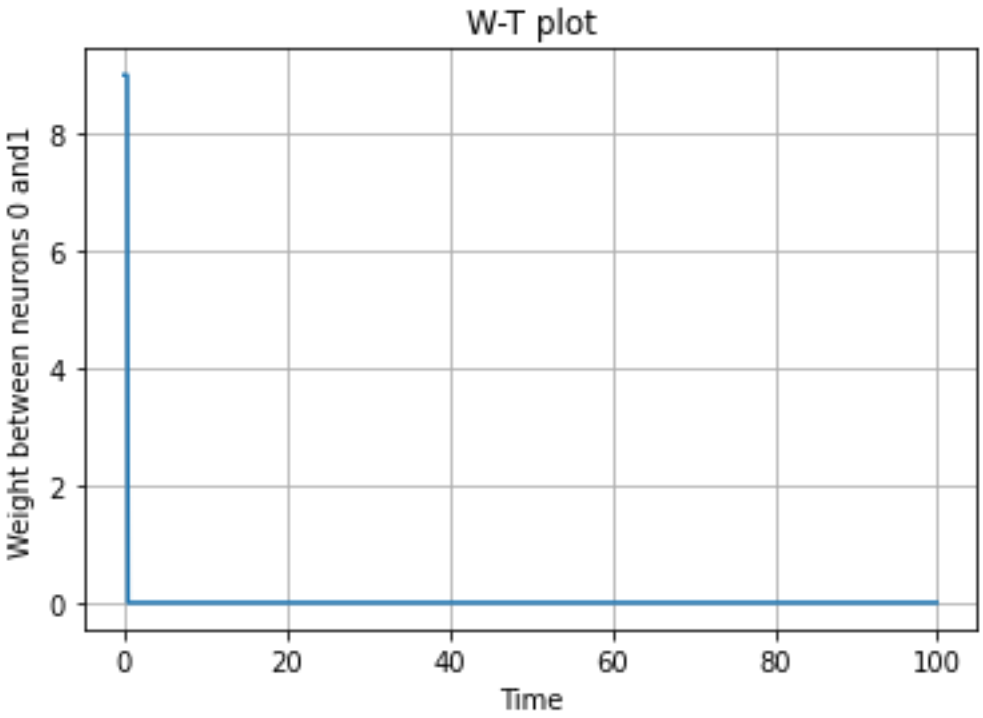
[[0, 9, 14], [13, 0, 11], [11, 14, 0]]

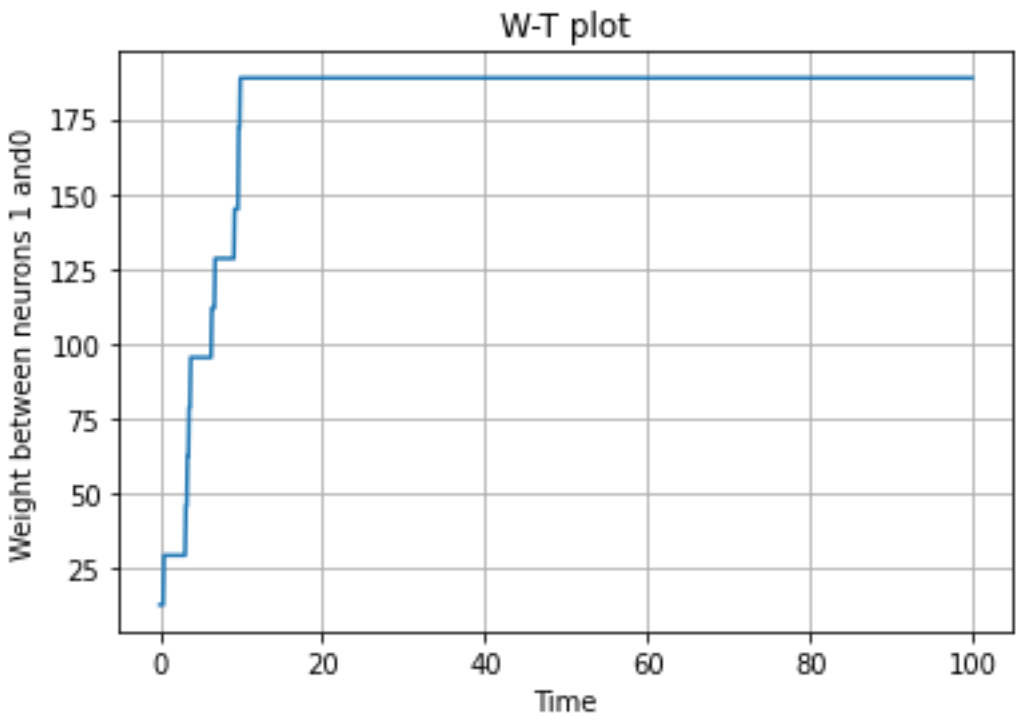
نمودار زیر، نمودار تایم اسپایک های هر نورون (پتانسیل_زمان) است.



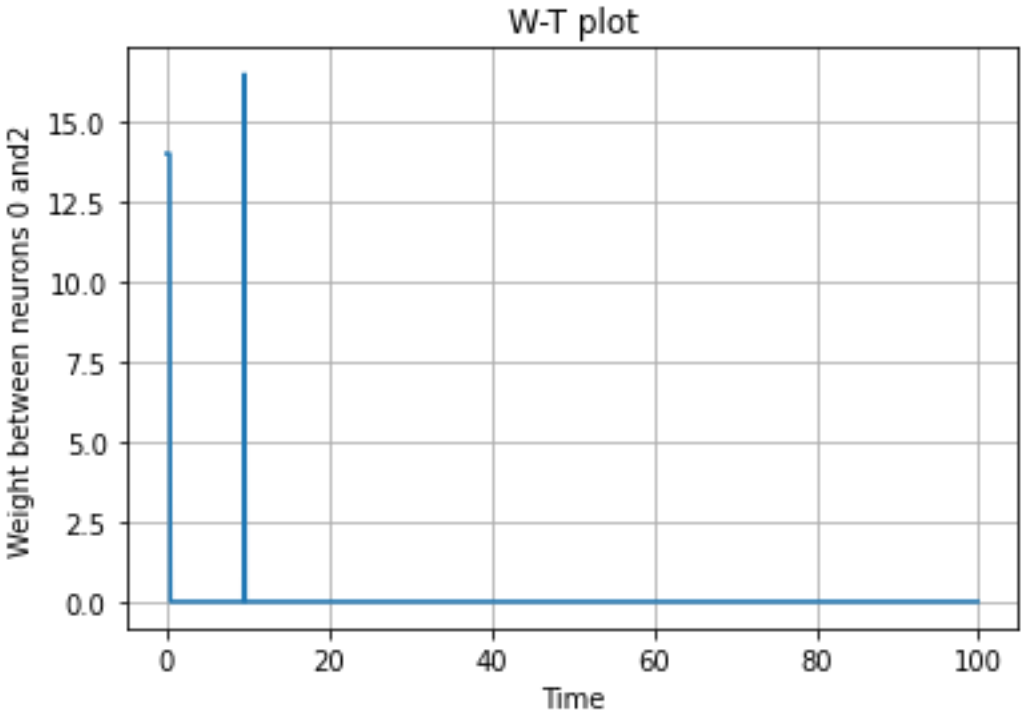
حال تغییرات وزن بین هر دو نورون را در واحد زمان بررسی می کنیم.

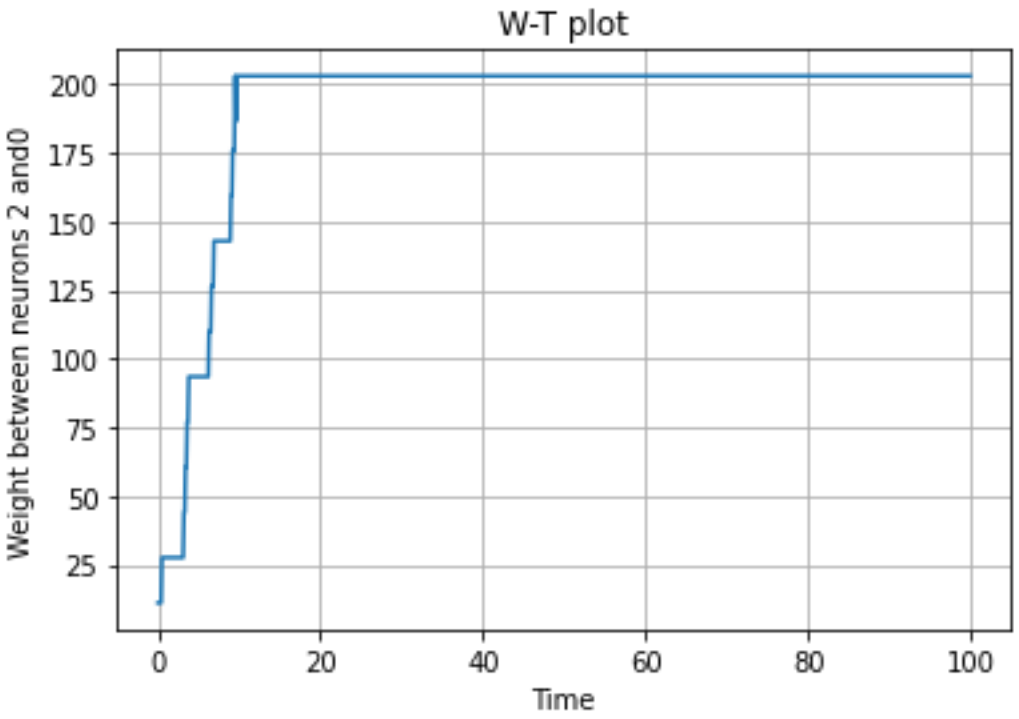
تغییرات وزن بین نورون 0 و 1: (جابجایی ندارد، پس هر جفت دو نمودار دارد)



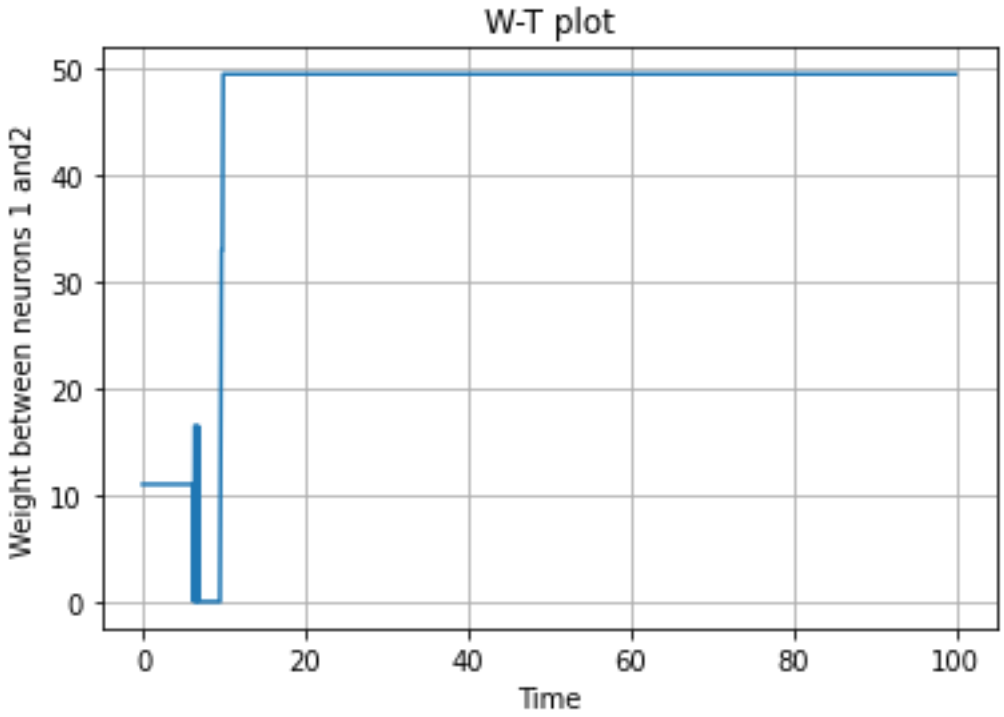


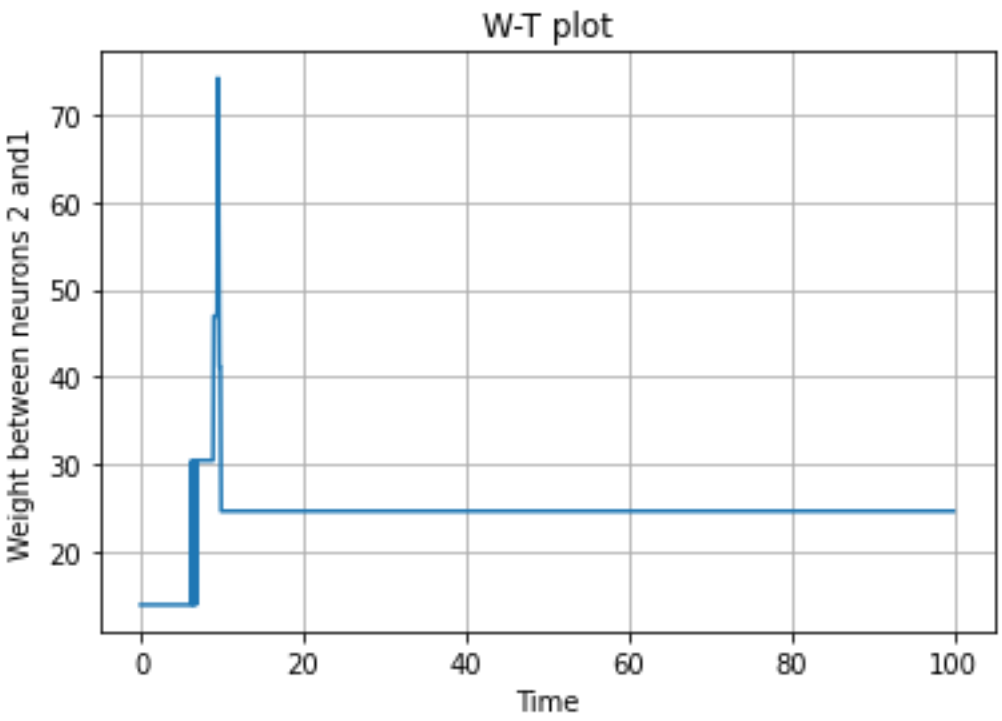
تغییرات بین نورون های 0 و 2:





تغییرات نورون های 1 و 2:





از این تغییرات نتیجه می گیریم که با اسپایک نورون پیشین تحریکی قبل از نورون پسین، وزن بین دو نورون افزایش می یابد و برعکس.

حال برای پیاده سازی بخش دوم پروژه، از کلاس LIFSNN استفاده می کنیم. یک شبکه نورونی اسپایکی که مدل نورون های آن LIF است و با پیشامد مطلوب، وزن بین دو نورون بیشتر می شود.

این کلاس آرگومان هایی چون ثابت تاو، ابعاد شبکه، میزان دوپامین و در نتیجه reward دریافتی دارد.

تابع form_the_network، شبکه عصبی ما را شکل می دهد.

تابع fit با استفاده از تابع learning_process، داده ها را به خورد شبکه عصبی می دهد و با استفاده از توابع activity_history، تغییرات لایه های مختلف را ذخیره می کند.

تابع prediction، برای پیشبینی و تست داده های تست به کار می رود.

تابع جریان استفاده شده در این شبکه:

```
[37] const = lambda x: 190 * (math.sqrt(math.fabs(math.sin(x))))
```

داده ها از فایل اکسل خوانده شده و به 4 دسته داده های آموزشی و تست فیچر ها و نتایج تقسیم شده است.

پس از آموزش شبکه، مشاهده می کنیم که دقت پیشبینی روی داده های آموزشی 0.8 و روی داده های تست 0.5 است.