STDP & RSTDP

گزارش کار پروژه بررسی مدل های STDP, RSTDP

توسط : کتایون کبرائی

استاد مربوطه : استاد خردپیشه

موضوع پروژه

در این پروژه ابتدا با استفاده از الهام گرفتن تمرین های قبلی یعنی بررسی مدل LIF و بررسی جمعیت های بین نورونی، دو نوع مدل دیگر را طراحی خواهیم کرد.

قسمت اول

در بخش اول ما نورون STDP را طراحی میکنیم. برای این کار ابتدا نیاز به کلاس نورون ساده خواهیم داشت که ویژگی های همیشگی مثل جریان، مقاومت و ... را داشته باشد. سپس توابع مربوط به ان مثل تعریف کردن مقادیر انرژی پتانسیل یا فرکانس ها و یا تولع رسم نمودار را برای این کلاس قرار می دهیم.

implementing basic neuron class

In [23]:
```
class Neuron:
    def __init__(self,i_func = 0, i = 5, time_interval = 100, dt = 0.1, u_rest = 0, R = 1, C = 10, threshold = 1,reset = True
                 ,save_name="none", u_spike = 20, u_reset = 0, tau = 0):
        self.i = i
        self.time_interval = time_interval
        self.dt = dt
        self.u_rest = u_rest
        self.R = R
        self.C = C
```

```
    def process(self, current_function, timespan, dt, reset=True):
        if reset:
            self.to_rest()
        size = math.ceil(timespan / dt)
        U = np.zeros(shape=(size, 2))
        spikes = []
        time = 0
        for index in range(len(U)):
            U[index, 1] = self.u
            U[index, 0] = time
            if self.u > self.threshold:
                spikes.append(time)
                self.reset()
            du = dt * (-1 * (self.u - self.u_rest) + 1e-3 * self.R * current_function(time)) / self.tau
            self.u += du
            time += dt
        return {'voltage': U, 'spikes': spikes}

    def frequency(self, current_range, timespan, dt):
        data = np.zeros(shape=(len(current_range), 2))
        for index in range(len(current_range)):
            self.to_rest()
            Func = lambda x: current_range[index]
            result = self.process(Func, timespan=timespan, dt=dt)
            result = result['spikes']
            data[index, 0] = current_range[index]
            if len(result) == 0:
                data[index, 1] = 0
            elif len(result) == 1:
                data[index, 1] = 1 / timespan
            else:
                data[index, 1] = (len(result) - 1) / (result[-1] - result[0])
        return data
```

در کلاس بعدی ارتباط بین نورون ها و جمعیت نورونی که در تمرین های گذشته هم داشتیم را پیاده سازی میکنیم. این کلاس با یک سری ویژگی های اضافه برای جمعیت نورونی تعریف میشود مثل نوع جمعیت که تحریکی یا مهاری است و یا نورون های داخل این جمعیت که در اصل اشیایی از کلاس نورون قبلی هستند.

In [24]:
```
class Population:
    def __init__(self, population_type, neurons, time_course, j=20):
        self.population_activity = []
        self.neurons = neurons
        self.connections = np.zeros((len(neurons), len(neurons)))
        self.j = j
        self.connection_type = self.fully_connection
        set_connection = self.connection_type
        self.population_type = population_type
        set_connection()
        self.time_course = time_course
        self.connection_history = deepcopy(self.connections.ravel())
```

در ادامه توابع مورد نیاز برای یک جمعیت نورونی را پیاده سازی میکنیم:

```
def fully_connection(self):
    self.connections = np.ones_like(self.connections) * (self.j / len(self.neurons))
    if self.population_type == 'inhibitory':
        self.connections = -1 * self.connections

def activity_history(self, time, dt, threshold):
    activity_list = np.zeros((len(self.neurons), 1))
    for idx in range(len(self.neurons)):
        activity_list[idx, 0] = self.activity_history_single(idx, time, dt, threshold)
    return activity_list

def activity_history_single(self, idx, time, dt, threshold):
    neuron = self.neurons[idx]
    S = 0
    activity = 0
    while self.time_course(S) > threshold:
        if (time - S) in neuron.spikes:
            activity += self.time_course(S)
        S += dt
    return activity

def single_step(self, input_current, self_activity, time, dt, time_course_threshold):
    inputs = self.connections.dot(self_activity)
    for i, neuron in enumerate(self.neuron_list):
        neuron.single_step(input_current + inputs[i, 0], time, dt)
    activity = self.calculate_activity_history(time + dt, dt, time_course_threshold)
    return activity

def reset(self, reset_connection=False):
    self.population_activity = []
    if reset_connection:
        set_connection = self.connection_type
        set_connection()
    for neuron in self.neurons:
        neuron.clear_history()
```


نکته : جمعیت پیاده سازی شده یک جمعیت **FULL_Connective** *می باشد.*

مثال های بخش اول

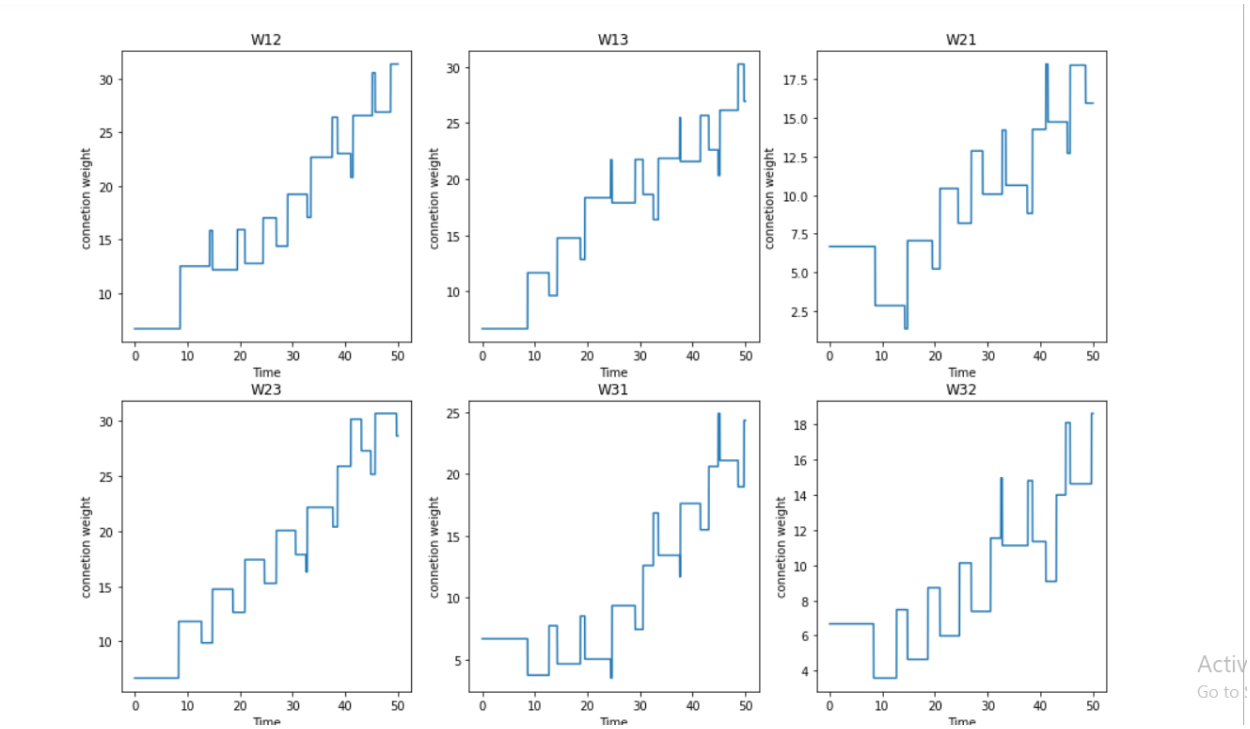
ابتدا باید سه مدل نورون را پیاده سازی کنیم و انها را در لیستی به کلاس جمعیت نورونی بدهیم. و

سپس سه جریان متفاوت را به عنوان ورودی به کلاس STDPدهیم.

```
In [16]: s = STDP("5000", "4000 * (math.sin(x) + 0.9)", "5000 * (math.cos(x) + 0.9)")
         s.weight_plotting()
```

Synaptic Weight Changes (STDP Rule)

و نمودار های ان به صورت زیر خواهد شد:



با توجه به نمودار ها میتوانیم تشخیص دهیم که وقتی نورون پست سیناپتیک فایر میکنید میزان وزن ها کاهش میابد و برعکس وقتی نورن پری سیناپتیک فایر میکند میزان وزن ها افزایش میابد.

قسمت دوم

در این قسمت باید ابتدا یک کلاس برای مدل SNN طراحی کنیم تا الگوریتم یادگیری در ان پیاده سازی شود و سپس دیتا ست مورد نظر را از منبع میخوانیم.

	test	Unnamed: 1	Unnamed: 2	Unnamed: 3	Unnamed: 4	Unnamed: 5	Unnamed: 6	Unnamed: 7	Unnamed: 8	Unnamed: 9	Unnamed: 10
0	inpput_neuron_number	train_1	train_2	train_3	train_4	train_5	train_6	train_7	train_8	train_9	train_10
1	1.0	1.0	2.0	3.0	2.0	1.0	1.0	0.0	0.0	0.0	2.0
2	2.0	1.0	1.0	1.0	0.0	1.0	0.0	1.0	0.0	0.0	2.0
3	3.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2.0
4	4.0	0.0	0.0	0.0	0.0	0.0	0.0	3.0	1.0	0.0	1.0
5	5.0	0.0	0.0	0.0	0.0	1.0	2.0	0.0	1.0	1.0	2.0
6	output_neuron_number	1.0	1.0	1.0	1.0	1.0	0.0	0.0	0.0	0.0	1.0
7	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
8	test	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
9	inpput_neuron_number	train_1	train_2	train_3	train_4	train_5	train_6	train_7	train_8	train_9	train_10
10	1.0	1.0	0.0	0.0	2.0	1.0	1.0	3.0	0.0	1.0	2.0
11	2.0	0.0	1.0	2.0	2.0	2.0	0.0	1.0	0.0	2.0	2.0
12	3.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	2.0	0.0	2.0
13	4.0	0.0	0.0	0.0	0.0	0.0	0.0	2.0	2.0	1.0	3.0
14	5.0	0.0	0.0	0.0	0.0	0.0	3.0	3.0	2.0	1.0	3.0
15	output_neuron_number	1.0	1.0	1.0	1.0	1.0	0.0	0.0	0.0	1.0	0.0

همان طور که مشاهده میشود برای وارد شدن داده ها به الگوریتم نیاز است که مقادیر در ۱۰۰۰۰ ضرب شوند.

حال اگر مدل را اجرا کنیم می بینیم که تا حد خوبی مدل یاد گرفته است.

```
accuracy of SNN on test data:  70.0 %
accuracy of SNN on train data:  80.0 %
```

و در نهایت برای نمودار های ان خواهیم داشت: