# Neurons population

گزارش کار پروژه ی بررسی جمعیت نورونی

توسط : کتایون کبرائی – ۹۹۲۲۲۰۸۴

استاد مربوطه : استاد خردپیشه

## موضوع پروژه

در این گزارش ما به ابتدا به بررسی ارتباط بین دو نورون از نوع های تحریکی-تحریکی، تحریکی-مهاری و مهاری-مهاری میپردازیم. در بخش بعدی ما ارتباط بیش از دو نورون برای مثال هشت نورون را بررسی میکنیم. و در نهایت در بخش اخر ارتباط بین چند جمعیت نورونی را بررسی خواهیم کرد.

# بخش اول : بررسی ارتباط بین دو نورون

در این بخش در اولین گام کلاسی از یک نورون از نوع ال آی اف تعریف میکنیم. این مدل نورون همان طور که میدانیم باید شاخصه هایی مثل نوع جریان ورودی و اندازه جریان ورودی برای بدست اوردن مقدار پتانسیل، مقاومت و ظرفیت خازنی، گام های زمانی، و در نهایت نوع نورون که میتواند مهاری یا تحریکی باشد، را ورودی بگیرد و با استفاده از انها تعریف شود.

In [63]:
```
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,
                 neuron_type = "excitatory", save_name="none"):
        self.i = i
        self.time_interval = time_interval
        self.dt = dt
        self.u_rest = u_rest
```

***نکته :*** *برای اینکه هرکدام از متغیر های ورودی استفاده شده در توابع به درستی کار کند، برای انها در ابتدا مقداری را پیش فرض قرار میدهیم.*

در بخش بعدی شروع به ساختن توابع مورد نیاز نورون ساده میکنیم. اولین تابع، تابع تعریف کننده لیستی است که طی بازه های زمانی مشخصی مقدار انرژی پتانسیل سلول بدست امده و در ان ذخیره می شود تا این بخش جزو ویژگی های نورون شود و در بخش های بعدی مسئله بتوان از ان استفاده کرد. فرمول به این صورت است که هر بار از ابتدا بازی زمانی به اندازه دلتا ثانیه به زمان اولیه اضافه میشود و انرژی پتانسیل ان زمان بدست می اید. حال اگر این انرژی پتانسیل از مقدار ترشلد بیشتر شد نورون اسپایک زده و به حالت استراحت اولیه دوباره ست میشود.

```
    def init_potencials(self):
        self.timer = np.arange(0, self.time_interval + self.dt, self.dt)
        u = [self.u_rest for i in range(len(self.timer))]
        self.i_init = [self.i_func(5, j) for j in self.timer]
        const = self.R * self.C

        for t in range(len(self.timer)):
            u[t] = u[t-1] + (((-u[t-1] + self.u_rest) + self.R * self.i_init[t]) * self.dt)/const
            if u[t] >= self.threshold or u[t] < self.u_rest:
                u[t] = self.u_rest
            self.u = u
            self.cur_time = t
```

توابع بعدی نیز به ترتیب تابع Times که در ان مقادیر بازه های زمانی تعریف و مشخص می شود، تابع رسم نمودار پتانسیل مربوط به همان نورون مشخص و نمودار رسم جریان ورودی به همان نورون مشخص و نمودار رسم اف بر حسب جریان ورودی خواهند بود.

در گام دوم برای این بخش مسئله ما ارتباط بین دقیقا دو نورون را ایمپلیمنت میکنیم. پس ابتدا برای ان کلاس مشخصی تعریف میکنیم که ویژگی های خاص خود مثل نورون های موجود در مجموعه که در اصل اشیایی از کلاس قبلی(کلاس نورون پایه) هستند، ارتباطات انها که با استفاده از لیستی مشخص میشود، وزن نورون های مهاری و تحریکی در این بخش و تاخیر هرکدام و ... را دارد.

```
In [64]: class First_neuron_group:
             def __init__(self, neurons, connections, ex_weight = 2, inh_weight = 2, delay = 1, counter = 500):
                 self.neurons = neurons
                 for i in neurons:
                     self.neroun_action.append(i.start())
                 self.conections = conections
```

حال یکی از مهم ترین توابع یعنی تابع اکشن یا عمل ارتباط بین دو نورون را ایمپلیمنت میکنیم. در این تابع در لیست نورون های موجود در مجموعه یکی یکی میگذریم و چک میکنیم که ابتدا نورون با نورون های دیگر ارتباطی دارد یا خیر. پس از ان نوع ارتباط را بررسی میکنیم و اگر اسپایکی وجود داشت ان را به لیست اسپایک های نورون های مهاری یا نورون های تحریکی اضافه میکنیم و با انها جمع میزنیم.

```
    def action(self):
        length = len(self.neurons)
        self.spikes_effect = [[0] * length for i in range(self.counter)]
        for t in range(self.counter):
            for i in range(len(self.actions)):
                action_info = next(self.actions[i])
                if action_info == True :
                    for j in self.connections[i]:
                        if self.neurons[i].type == "excitatory":
                            self.ex_spikes.append(i + 1)
                            self.ex_spikes_times.append(t)
                            if t+self.delay < self.counter:
                                self.spikes_effect[t + self.delay][j] += self.ex_weight

                        if self.neurons[i].type == "inhibitory":
                            self.inhibitory_spikes.append(i + 1)
                            self.inhibitory_spikes_time.append(t)
                            if t+self.delay < self.counter:
                                self.spikes_effect[t+self.delay][j] += self.inh_weight

            for i in range(len(self.neurons)):
                self.neurons[i].u[self.neurons[i].current_time] += self.spikes_effect[t][i]
```

***نکته :*** *در هنگام بدست اوردن مجموعه اسپایک ها در هر مرحله با توجه به تاخیر نورون مورد نظر، میزان اثر نورون را بدست می اوریم.*

بعد از تعریف کردن عمل اکشن و بدست اوردن تاثیر نورون های تعریف شده حال میتوانیم توابع رسم نمودار پتانسیل نورون ها و نمودار رستر اسپایک هارا رسم کنیم.

## بخش مثال ها :

*نمونه اول : ارتباط بین دو نورون تحریکی همراه با دو جریان متفاوت*

ابتدا باید دو نوع مدل جریان را تعریف کنیم که برای مثال میتواند دو جریان ثابت با مقادیر متفاوت باشد.

در بخش بعدی دو شئی از کلاس نورون هردو از نوع تحریکی و به هرکدام یکی از جریان ها را میدهیم.

```
In [9]: i1 = lambda x: 4
        i2 = lambda x: 3
        neuron1 = Neuron(i_func=i1)
        neuron2 = Neoron(i_func=i2)
```

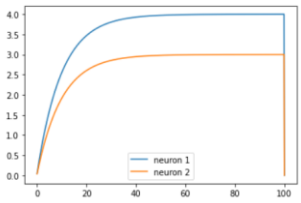
در گام بعدی باید این دو نورون تعریف شده را به کلاس ارتباط بین دو نورون دهیم و سپس تابع اکشن را برای شکل گیری ارتباط صدا کنیم.

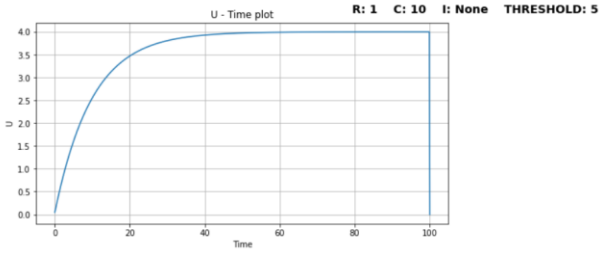
```
neurons = [neuron1, neuron2]
connections = [[1], [0]]
first_neuron_group = First_neuron_group(neurons, connections)
neurons_group.action()
```

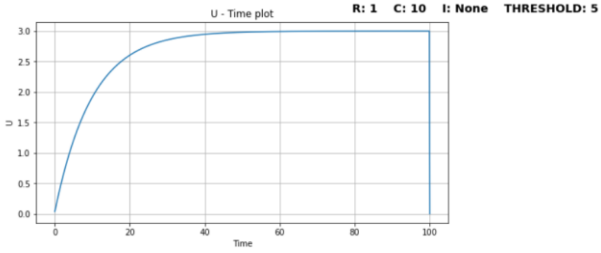
در نهایت برای رسم نمودار های مورد نظر توابع رسم را صدا کنیم:

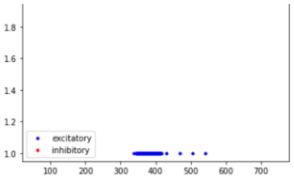
```
In [10]: first_neuron_group.neurons_u_plot()
         first_neuron_group.raster_plot()
         neuron1.plot_U_t()
         neuron2.plot_U_t()
```

نمودار های ان به ترتیب برای این مدل خواهند شد :



*نمونه دوم : ارتباط بین یک نورون تحریکی و یک نورون مهاری*

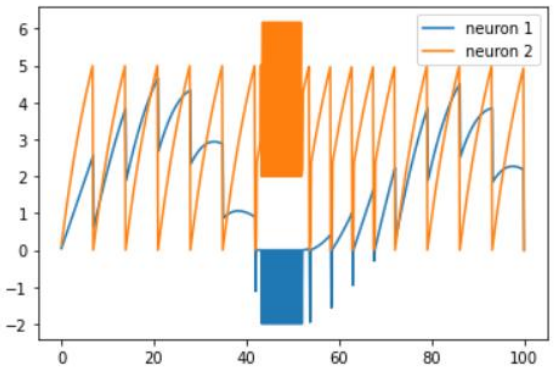
مانند مثال قبل دوباره دو نورون این بار یکی مهاری و یکی دیگری تحریکی با جریان های متفاوت تعریف میکنیم.

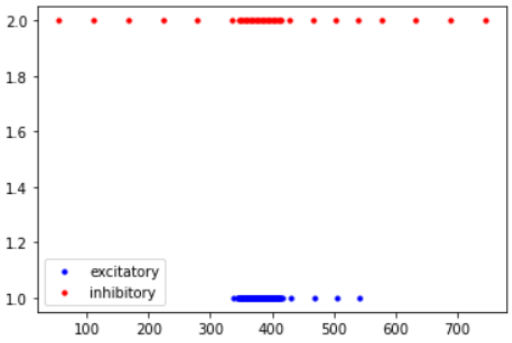
```
i1 = lambda x: 4 * (math.sin(x/5) + 0.3)
i2 = lambda x: 10
neuron1 = neuron(i_func=i1)
neuron2 = neuron(i_func=i2 , neuron_type='inhibitory')
```

سپس دوباره مدل نورون ها رو به کلاس کانکشن بین شان میدهیم و تابع اکشن را برای شان فراخوانی میکنیم.

```
neurons = [neuron1, neuron2]
connections = [[1], [0]]
first_neuron_group = First_neuron_group(neurons, connections)
first_neuron_group.action()
```

در نهایت برای توابع رستر و پتانسیل نورون ها خواهیم داشت:



*نمونه سوم : ارتباط بین دو نورون مهاری*

ابتدا باید این دو نورون و جریان متفاوت شان را تعریف کنیم:

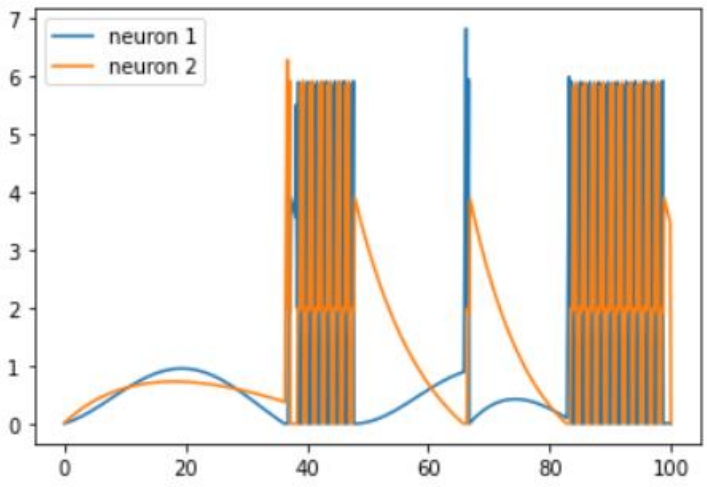
```
i1 = lambda x: (math.sin(x/8) + 0.3)
i2 = lambda x: (math.cos(x/24))
neuron1 = neuron(i_func=i1, neuron_type='inhibitory')
neuron2 = neuron(i_func=i2, neuron_type='inhibitory')
```

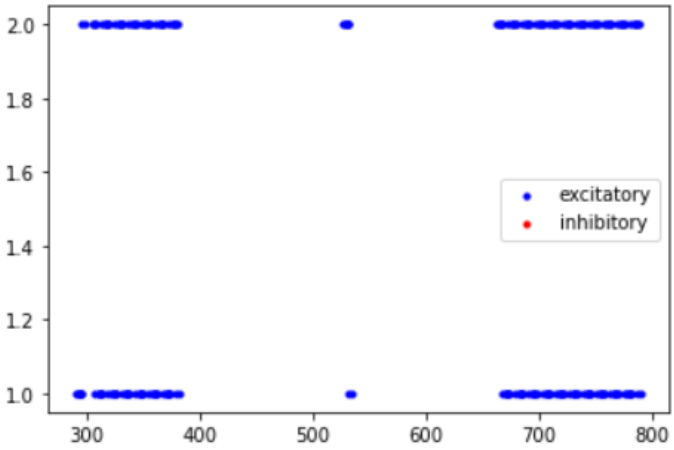
سپس این دو نورون مهاری را به تابع های اکشن از کلاس ارتباط دو نورونی میدهیم تا ارتباط صورت گیرد.

```
neurons = [neuron1, neuron2]
connections = [[1], [0]]
first_neuron_group = First_neuron_group(neurons, connections)
first_neuron_group.action()
```

در نهایت هم توابع مورد نظرشان را رسم میکنیم:

```
first_neuron_group.u_plot()
first_neuron_group.raster_plot()
```






همان طور که مشاهده می شود در لحظاتی که نورون تحریکی اسپایک میزند پتانسیل نورون نورون های در ارتباط با ان نورون افزایش می یابد.  دقیقا عکس این اتفاق برای نورون مهاری خواهد افتاد.

# بخش دوم : بررسی ارتباط بین چندین نورون مختلف

در این بخش در گام اول متغیر های مربوط به این بخش تمرین را به عنوان ورودی میگیریم که شامل نورون های ورودی و کانکشن بین شان که با استفاده از لیستی مشخص میشود، تعداد نورون ها و احتمال ارتباط گرفتن بین نورون ها و ... میباشد.

```
In [76]: class second_neuron_groups:
             def __init__(neurons, neurons_count, ex_count, inh_count, ex_prob, inh_prob, i = 0):
                 self.neurons = neurons
                 self.neurons_count = neurons_count
                 self.inh_count = in_count
```

در گام بعدی تابع اصلی این کلاس که همان اکشن و عمل ارتباطی بین تعدادی از نورون ها است را تعریف میکنیم.

```
    def action(self):
        counter = 0
        ex_conn_count = math.ceil(neurons_count * ex_prob)
        inh_conn_count = math.ceil(neurons_count * inh_prob)
        for i in range(self.inh_count):
            self.population.append(Neuron(neuron_type = "inhibitory"))
            counter += 1
        for j in range(counter):
            neuron = neuron(neuron_type="inhibitory")
            neurons.append(neuron)
            connections.append(random.sample(range(neurons_count), inh_conn_count))
        for i in range(self.ex_count):
            self.population.append(Neuron(neuron_type = "excitatory"))
            counter += 1
        for k in range(counter):
            neuron = neuron(neuron_type="excitatory")
            neurons.append(neuron)
            connections.append(random.sample(range(neurons_count), ex_conn_count))
        neurons_group = First_neuron_group(neurons, connections)
        return neurons_group
```

***نکته :*** *همان طور که مشاهده می شود ما از کلاس قبلی برای بررسی دو به دو نورون های موجود در مجموعه استفاده کردیم.*

## بخش مثال ها :

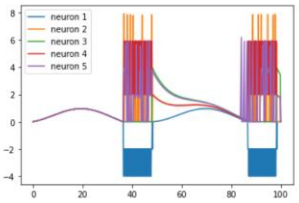
ابتدا باید هشت مدل نورون تحریکی و دو مدل نورون مهاری تعریف کنیم به طوری که همه نورون ها روی هم اثر بگذارند اما پیش از ان به لیستی از جریان های مختلف هم نیاز است تا هر بار به نورون های مختلف داده شود.

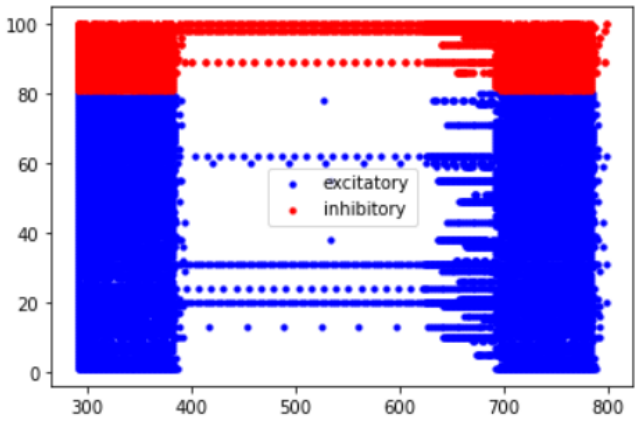
```
In [40]: Is = []
for i in range(100):
  I = lambda x: math.sin(x/8) + 0.3
  Is.append(I)
```

بعد از مشخص کردن نورون های مورد نظر و فرستادن انها به کلاس بخش دوم تابع اکشن فراخوانی میشود.

```
second_neuron_groups = Second_neuron_groups(100, 80, 20, 0.01, 0.02, Is)
second_neuron_groups.action()
```

در ادامه هم توابع مورد نظر برای مدل ما رسم میشود.



# بخش سوم : بررسی ارتباط بین چند جمعیت نورونی

در این بخش ابتدا کلاسی تشکیل داده و متغیرهای مورد نظر ان مثل نورون های ورودی و ارتباط بین این نورون ها در کنار وزن اسپایک هر کدام از نورون های مهاری و تحریکی، تاخیر اثر اسپایک ها و اندازه ترشلد سلول، را در ان قرار میدهیم و تعیین میکنیم.

```
In [82]: class Third_neuron_groups:
             def __init__(self, neurons, connections, ex_weight = 2, inh_weight = 2, delay = 1, threshold = 20, effect = 2,
                          counter = 800):
                     self.neurons = neurons
                     for i in neurons:
                         self.neroun_action.append(i.start())
                     self.conections = conections
                     self.ex_weight = ex_weight
                     self.inh_weight = inh_weight
                     self.delay = delay
                     self.counter = counter
                     self.actions = []
                     self.spikes = []
                     self.ex_spikes_times = []
                     self.ex_spikes = []
                     self.inh_spikes_times = []
                     self.inh_spikes = []
                     self.spikes_effect = []
                     self.connected_neuron_groups = []
                     self.threshold = threshold
                     self.effect = effect
```

در قسمت بعدی تابع اکشن و عمل ارتباط بین چند جمعیت نورونی را تعریف میکنیم.

```
    def action(self):
            length = len(self.neurons)
            self.effect = [[0] * length for i in range(self.counter)]
            for t in range(self.counter):
                  for i in range(len(self.actions)):
                        action_info = next(self.actions[i])
                        if action_info == True :
                             for j in self.connections[i]:
                                    if self.neurons[i].neuron_type == "excitatory":
                                            self.ex_spikes.append(i + 1)
                                            self.ex_spikes_time.append(t)
                                            if t+self.delay < self.counter:
                                                   self.effect[t + self.delay][j] += self.ex_weight

                                    if self.neurons[i].neuron_type == "inhibitory":
                                            self.inh_spikes.append(i + 1)
                                            self.inh_spikes_time.append(t)
                                            if t+self.delay < self.counter:
                                                    self.effect[t+self.delay][j] += self.inh_weight
                  for i in range(len(self.neurons)):
                        self.neurons[i].u[self.neurons[i].current_time] += self.effect[t][i]
```

از طرفی در این تابع از تابعی به نام کانکشن هم استفاده میکنیم که هر بار با صدا شدن ان گروه نورونی ورودی ان به مجموعه گروه های نورونی دیگر کانکت میشود.

```
def connect(self, neuron_group):
        self.connected_neuron_groups.append(neuron_group)
```

در نهایت هم توابع رسم نمودار پتانسیل این کلاس و نمودار رستر ان را قرار میدهیم.

```
def neurons_u_plot(self, neurons_count=5):
    for i in range(min(neurons_count, len(self.neurons))):
        plt.plot(list(map(lambda j: j * self.neurons[i].dt, range(len(self.neurons[i].u)))), self.neurons[i].u)


def raster_plot(self):
    plt.scatter(self.excitatory_spikes_time, self.excitatory_spikes, color = "red", s = 10)
    plt.scatter(self.inhibitory_spikes_time, self.inhibitory_spikes, color = "blue", s = 10)
    plt.legend(["excitatory", "inhibitory"])
```

## بخش مثال ها :

در این بخش باید سه جمعیت نورونی هرکدام شامل ۱۰ نورون یکی مهاری و دوتا تحریکی تعریف کنیم.

ابتدا باید لیستی از جریان های ورودی درست کنیم تا به نورون های مورد نظر بدهیم.

```
ex1_Is = []
for i in range(100):
  I = lambda x: 3*i
  ex1_Is.append(I)
ex2_Is = []
for i in range(100):
  I = lambda x: random.random() * 5.5
  ex2_Is.append(I)
inh_Is = []
for i in range(100):
  I = lambda x: 3
  inh_Is.append(I)
```

حال میتوانیم سه جمعیت نورونی را پیاده سازی کنیم و تابع اکشن برای ایجاد کانکشن بین شان را صدا کنیم.

```
neuron_group1 = Third_neuron_groups(100, 100, 0, 0.01, 0, ex1_Is)
neuron_group2 = Third_neuron_groups(100, 100, 0, 0.01, 0, exc_I_2)
neuron_group3 = Third_neuron_groups(100, 0, 100, 0, 0.1, inh_Is)
neuron_group1.connect(neuron_group3)
neuron_group2.connect(neuron_group3)
neuron_group1.action()
neuron_group2.action()
neuron_group3.action()
```

سپس در این مرحله تابع های رسم نمودار های مربوطه را قرار میدهیم.

```
neuron_group1.neurons_u_plot()
neuron_group2.neurons_u_plot()
neuron_group3.neurons_u_plot()
```

که نمودار های ان خواهد شد: