علیرضا آزادبخت

۹۵۲۲۲۰۰۳

POPULATIONS AND DECISION MAKING

PROJECT 2

تمرین شماره یک:

برای پیاده سازی تمرین شماره یک بعد از مطالعه کتابخانه های موجود تصمیم گرفتیم که پیاده سازی رو بدون استفاده از کتابخانه های موجود انجام بدیم برای همین کار از مدل LIF که در پروژه قبل پیاده سازی کرده بودیم استفاده کردیم و کلاسی برای نورون LIF با پارامتر های زیر ساختیم:

```
class Neuron:
    def   init  (self, type="exc", index=0, group = 0):  # exc/inh
        self.group = group
        self.index = index
        self.type = type
        self.time = 100
        self.steps = 0.25
        self.u_rest = 0
        self.r = 10
        self.c = 5
        self.threshold = 2 + random.random() * 2

        self.timer = np.arange(0, self.time + self.steps, self.steps)
        self.tm = self.r * self.c
        self.dt = self.steps
        self.i input = 0
        self.u = [self.u rest] * len(self.timer)

    def update(self, j, input):
        self.u[j] += self.u[j - 1] + (-self.u[j - 1] + self.r * input) / self.tm * self.dt
        result = 0
        if self.u[j] >= self.threshold:
            result = self.threshold - self.u_rest
            self.u[j] = self.u rest

            # print("fire :", self.type, self.index, result)
        return result

    def update u(self, value, j):
        if self.type == "exc":
            self.u[j] += value
        else:
            if (self.u[j] > (-1) * (self.u_rest / 3)):
                self.u[j] -= value
```

جمعیت نورونی که قصد پیاده سازی اش را داشتیم لیستی از ابجکت های این کلاس در نظر گرفتیم و نرون با تعدادی که نیاز بود با نوع Excitatory(exc)/Inhibitory(inh) تعریف کردیم و هنگام شبیه سازی در هر استپ زمانی dt تمام پتانسیل های نورون های جمعیت رو اپدیت میکردیم.

```
excitatory_N = 800
inhibitory_N = 200
N = excitatory_N + inhibitory_N
group = []
w = []
c = 0
for i in range(inhibitory_N):
    group.append(Neuron(type="inh", index=c))
    c += 1
for i in range(excitatory_N):
    group.append(Neuron(type="exc", index=c))
    c += 1

for i in range(N):
    temp = []
    for j in range(N):
        if i == j:
            temp.append(0)
        else:
            temp.append(random.random())
    w.append(temp)
```

نورون های جمعیت ما با وزن های w به هم متصلند و این وزن ها در ماتریسی ذخیره سازی شده اند و هنگامی که نورونی فایر میکند بر اساس وزن های مربوط به این نورون پتانسیل نورون های متصل رو افزایش میدهیم

```
y[i][g.index] = g.index
for k in range(N):
    g.update_u(w[g.index][k] * result, j=i)
```

هر یک از نورون ها حد فایر کردن متفاوتی دارند و مقداری تصادفی هنگام ساخته شدن نورون به حد فایر پایه اضافه میشود که نمودار های بهتری بگیریم.

```
self.threshold = 2 + random.random() * 2
```

به طور کلی در هر استپ زمانی پتانسیل همه نورون های جمعیت رو تحت جریان رندوم آپدیت میکنیم اگر نورونی طی این عملیات فایر کرد تمامی نورون های متصل رو با توجه به وزن مربوطه اپدیت میکنیم

```
for i in range(len(group[0].timer)):
    current_i = i_gen.get(i)

    yi[i] = current_i

    for g in group:
        result = g.update(j=i, input=current_i)
        if result > 0:
            y[i][g.index] = g.index
            for k in range(N):
                g.update_u(w[g.index][k] * result, j=i)
```

چیزی که مشاهده کردیم در نورون های مهاری اگر جلوی منفی شدن بیش از اندازه پتانسیل نورون هارو نگیریم کاملا همه چیز بهم میریزه بخاطر همین حد پایینی برای منفی شدن پتانسیل در نظر گرفتیم تا نورون های مهاری بیش از اندازه منفی نشوند

```
if self.type == "exc":
    self.u[j] += value
else:
    if (self.u[j] > (-1) * (self.u_rest / 3)):
        self.u[j] -= value
```

در هر استپ زمانی در نظر میگیریم که کدوم نرون ها فایر کردن و نمودار های مربوطه roysterرو طبق ایندکس اون نورون رسم میکنیم

حال ازمایشاتی که با جمعیت های نورونی مختلف انجام دادیم رو تحلیل میکنیم

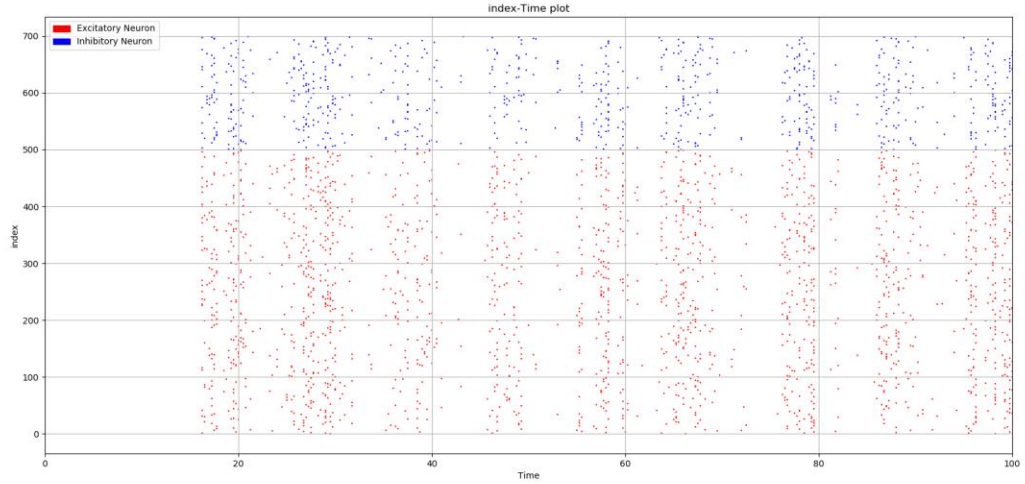
برای ساختن جریان رندوم بجای استفاده از تابع مولد پروژه قبل از تابع مولد صاف تری استفاده کردیم :

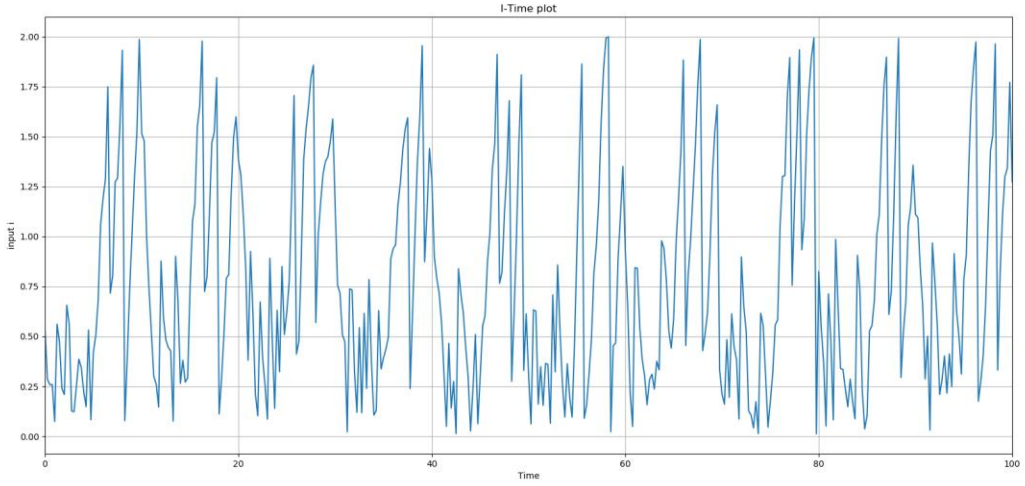
```
def get(self, iterate):
    if iterate % 20 == 0:
        self.going up = not self.going up
    temp = random.random()
    if self.going_up:
        temp = self.last_i + temp * (self.i / 2)
    else:
        temp = self.last_i - temp * (self.i / 2)
    if temp > self.i * 2 or temp < 0:
        temp = random.random() * (self.i)
    self.last i = temp
    return temp
```

نتایج زیر برای جمعیت های نورونی با وزن های اتصال یکسان و برابر میباشد:

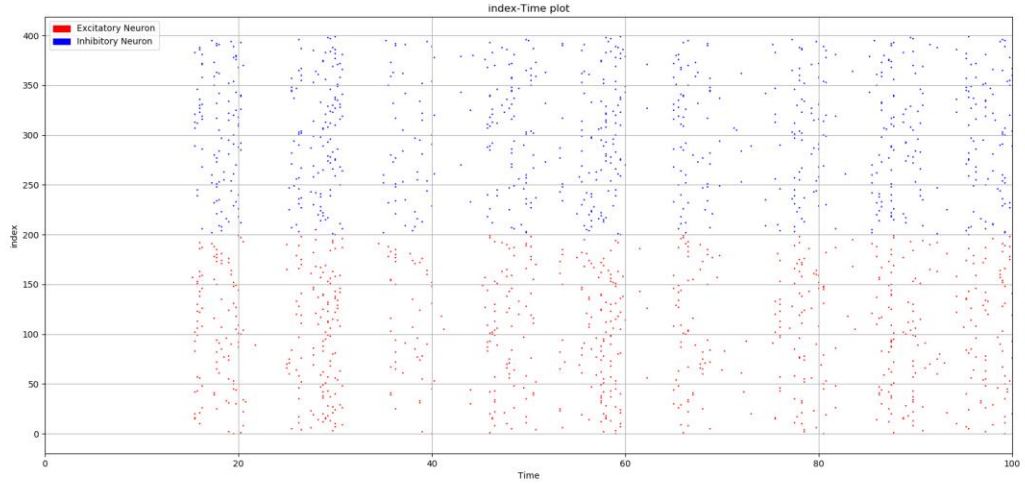
نکته قایل مشاهده اثر تعداد نرون های مهاری هست در نموداری که تعداد نرون های تحریکی و مهاری برابر هست تا حدی خنثی کردن اثر هر دو گروه رو میتونیم ببینیم که به نسبت تراکم نرون های فایر کرده کاهش پیدا میکنه ولی هرچی مهاری ها بیشتر میشن بنسبت تراکم فایر های بیشتر میشه

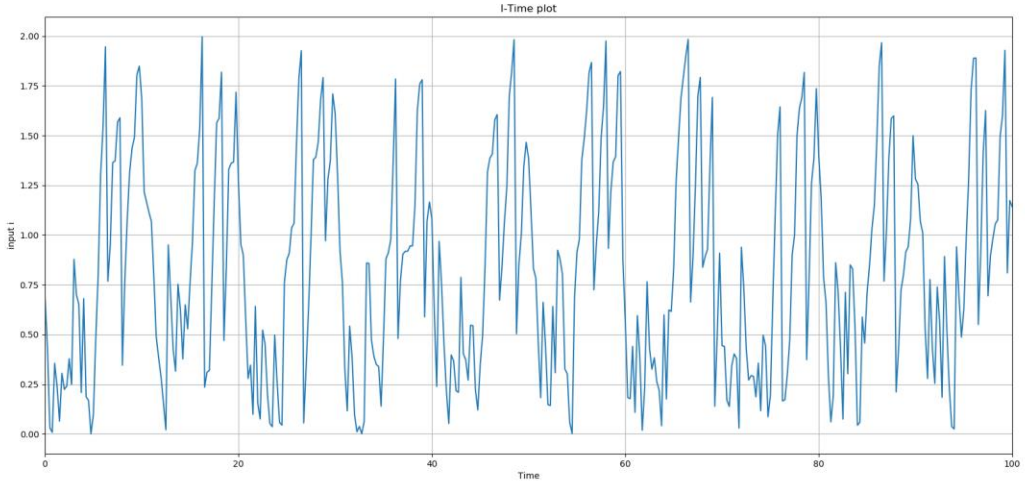
Population of 500 Excitatory Neurons and 200 Inhibitory Neurons



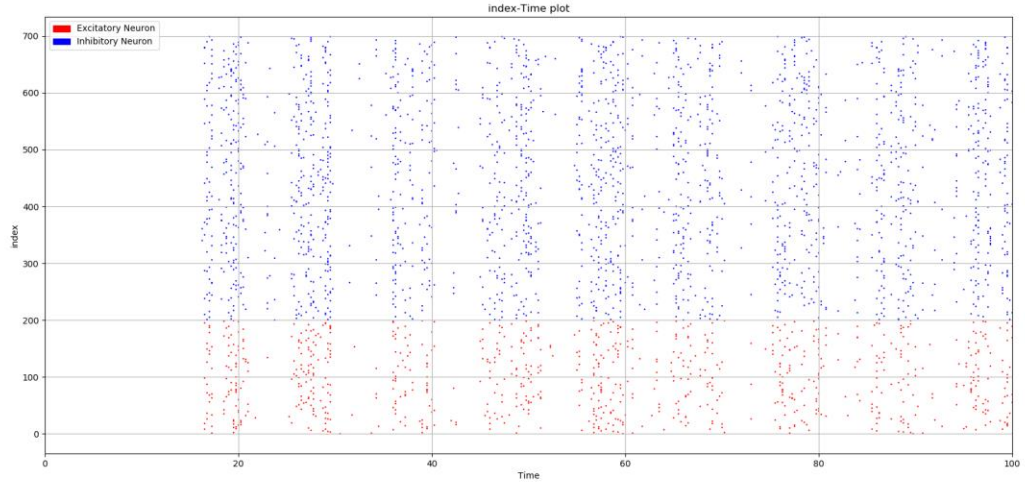


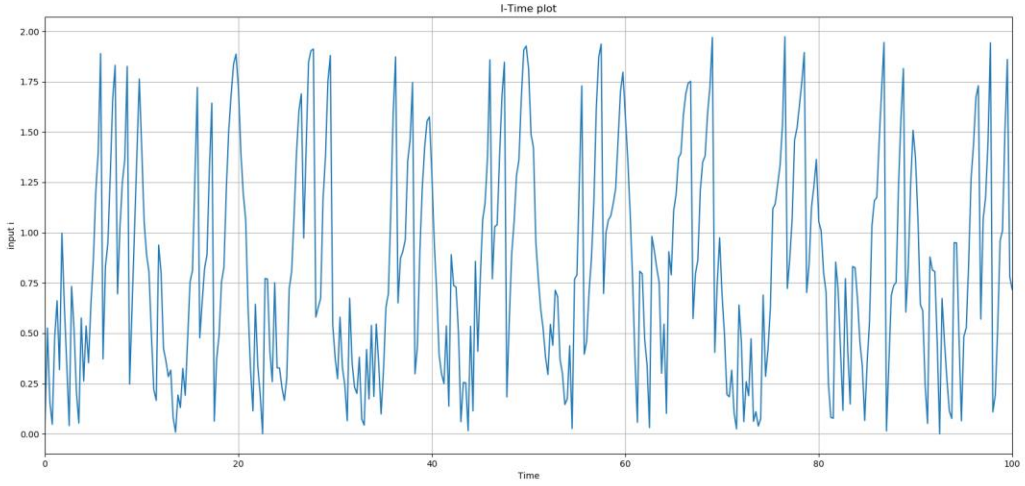
Population of 200 Excitatory Neurons and 200 Inhibitory Neurons



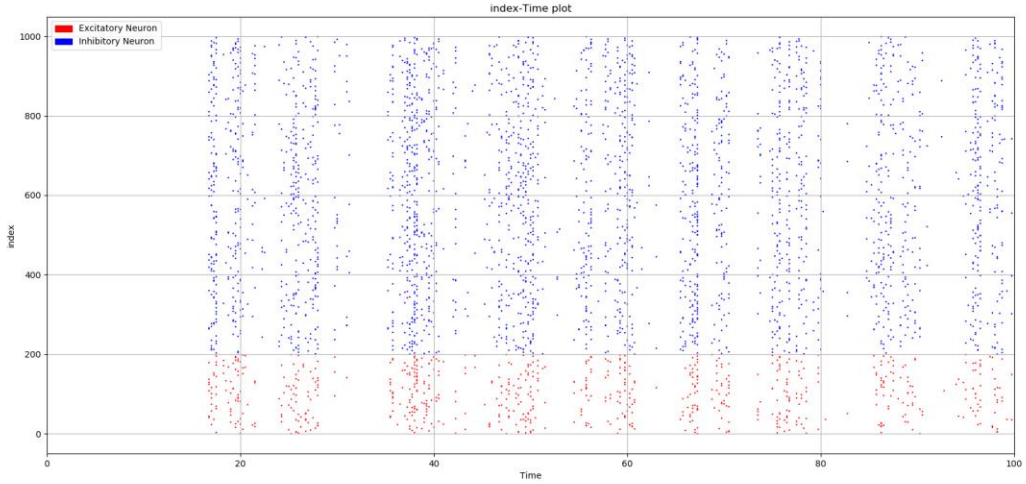


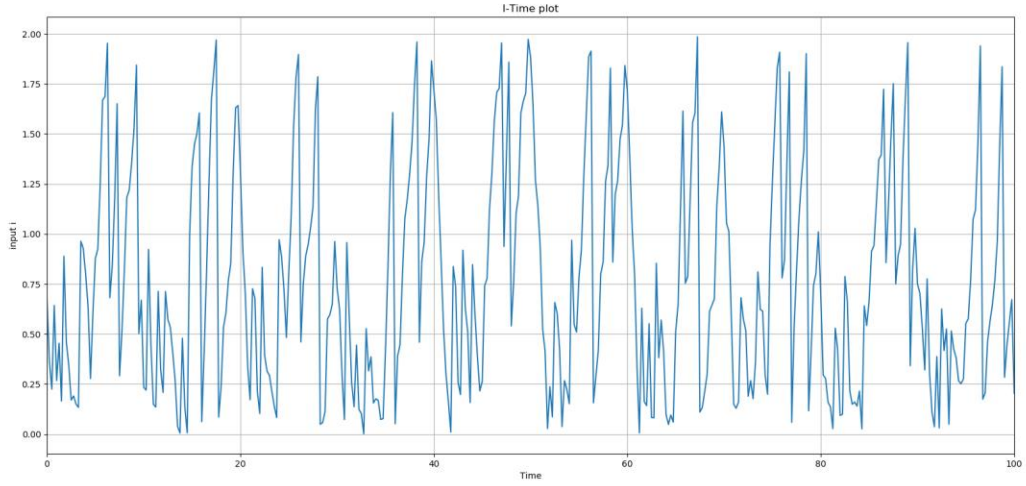
Population of 200 Excitatory Neurons and 500 Inhibitory Neurons



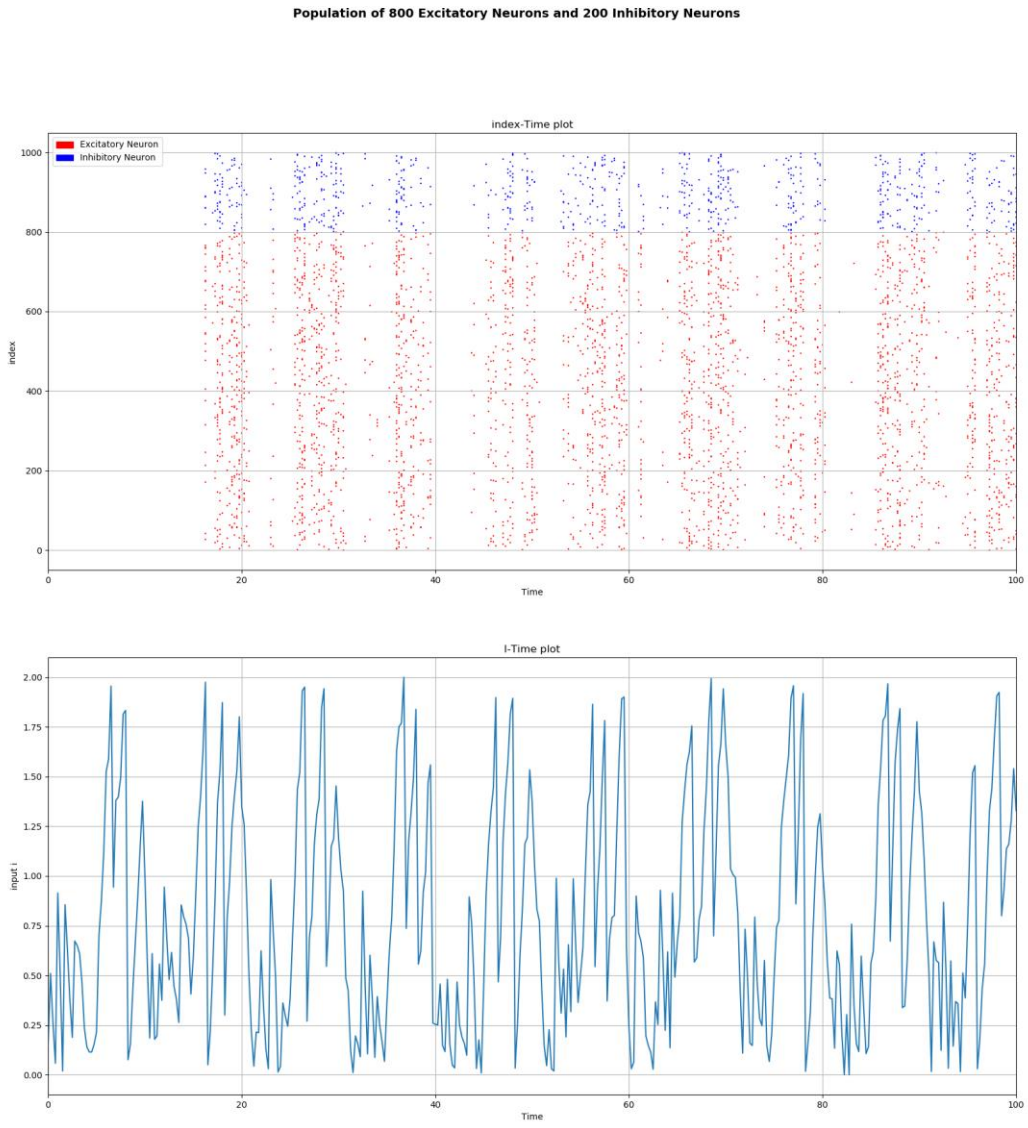


Population of 200 Excitatory Neurons and 800 Inhibitory Neurons

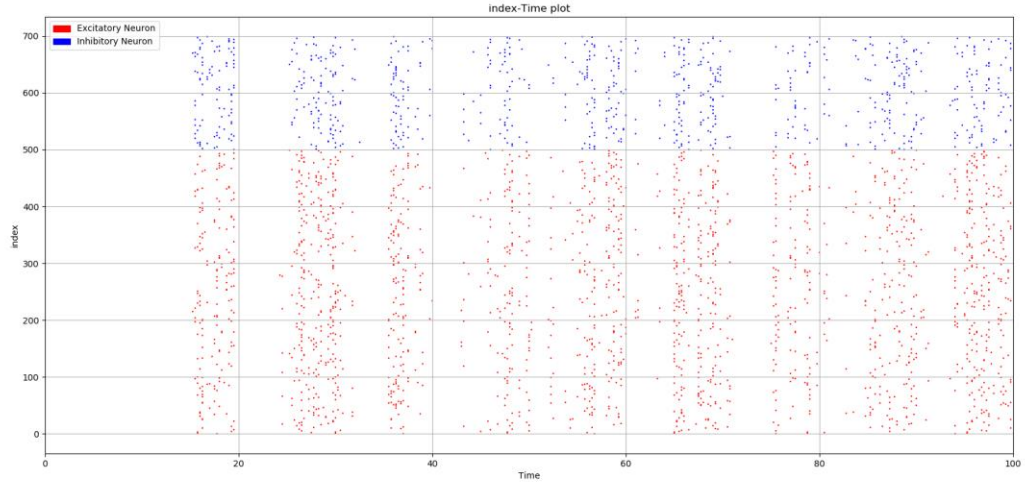


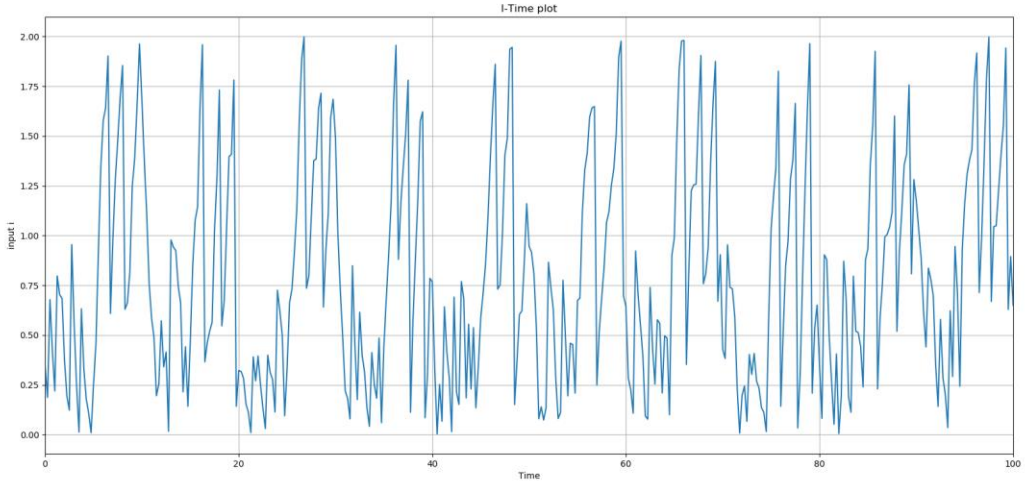


در آزمایش بعد اتصال نورون ها رو بصورت رندوم انجام دادیم و نتایج زیر رو مشاهده کردیم:

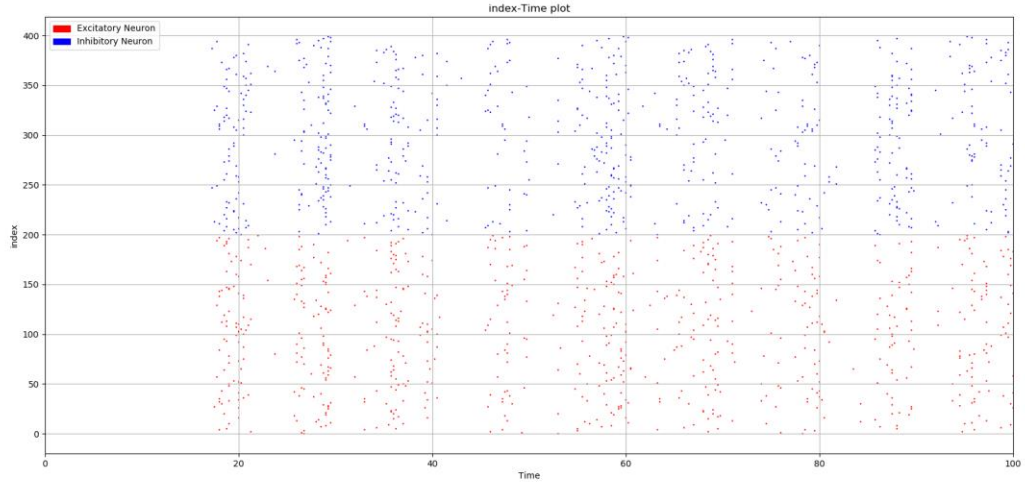


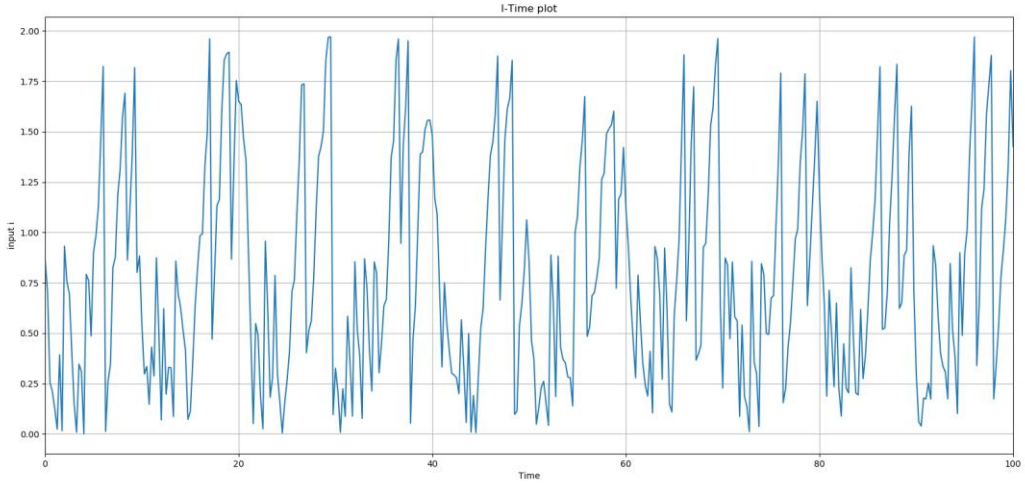
Population of 500 Excitatory Neurons and 200 Inhibitory Neurons



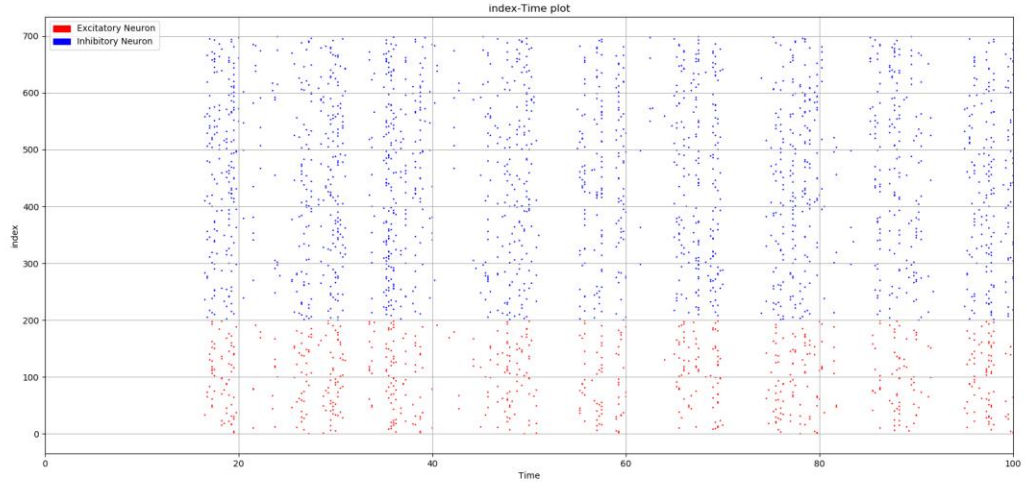


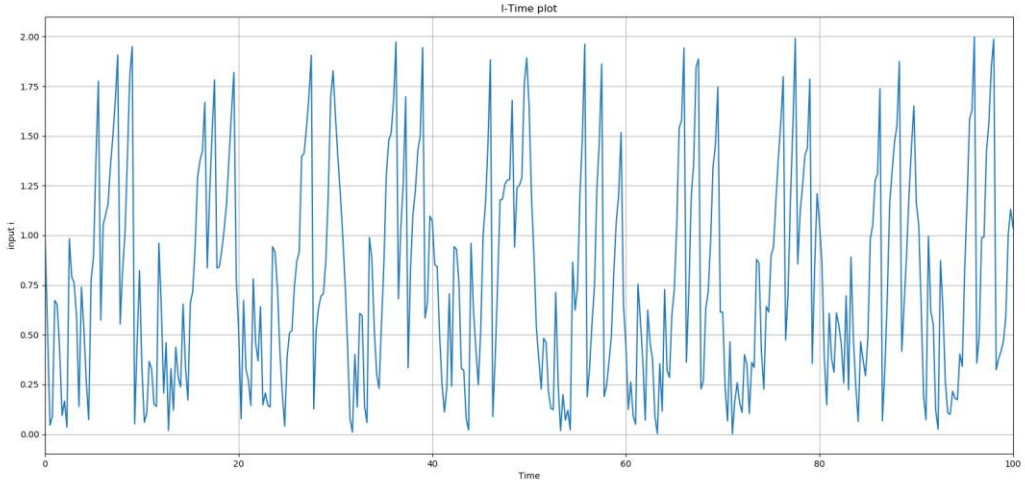
Population of 200 Excitatory Neurons and 200 Inhibitory Neurons



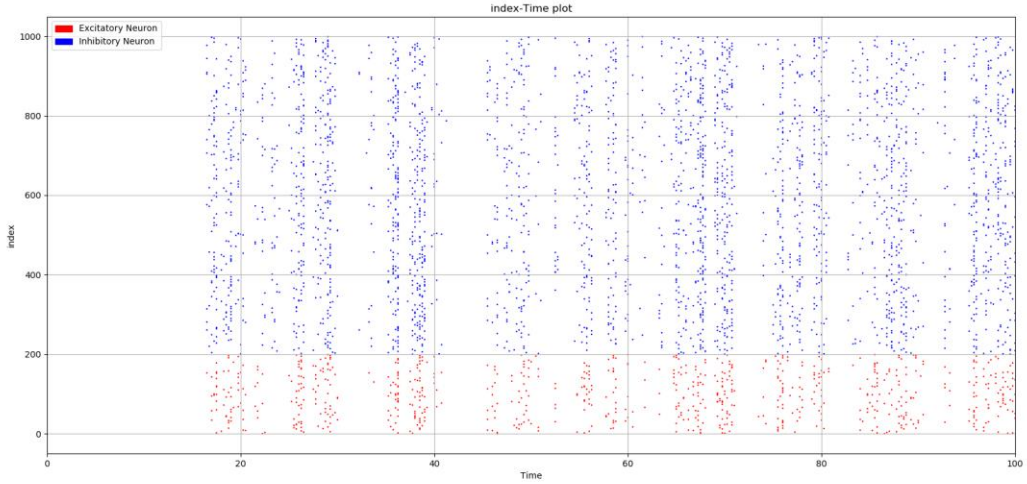


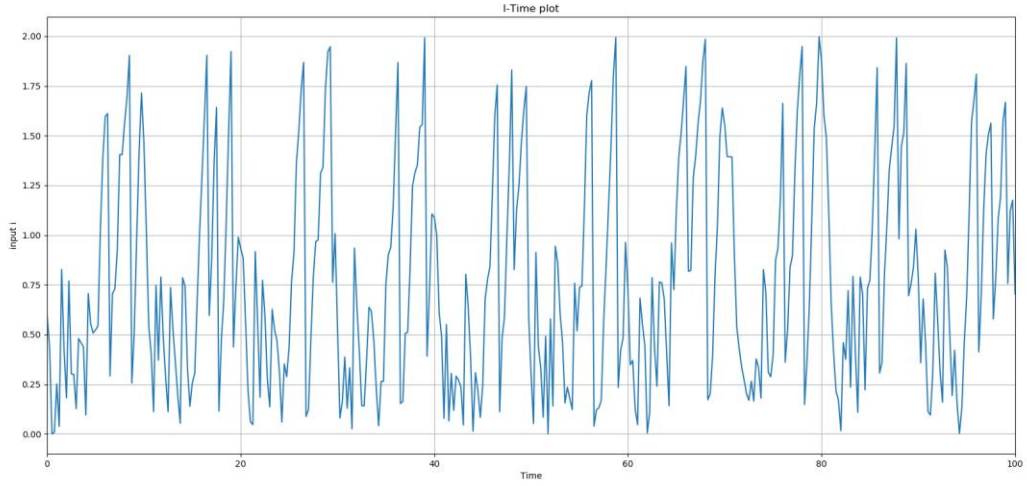
Population of 200 Excitatory Neurons and 500 Inhibitory Neurons



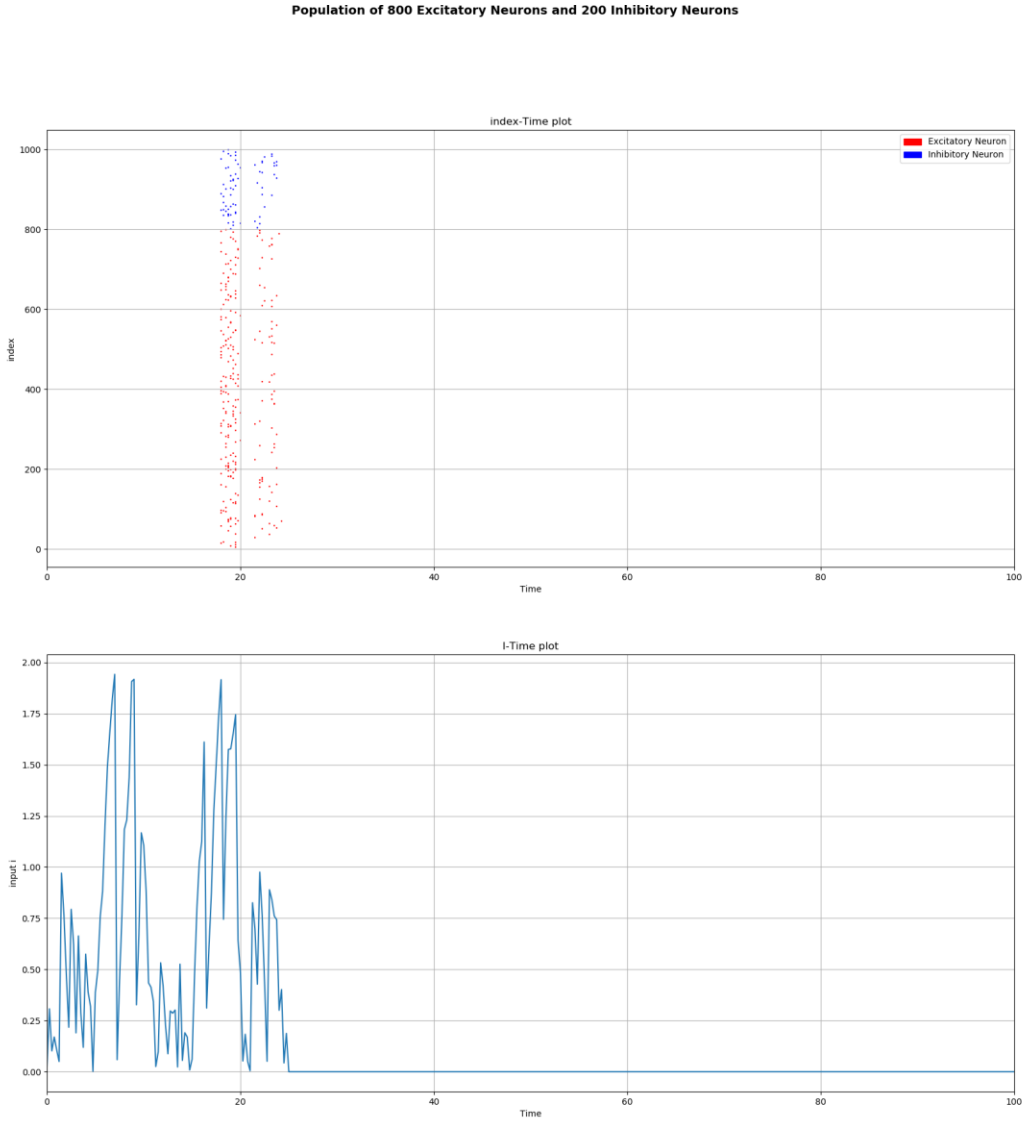


Population of 200 Excitatory Neurons and 800 Inhibitory Neurons

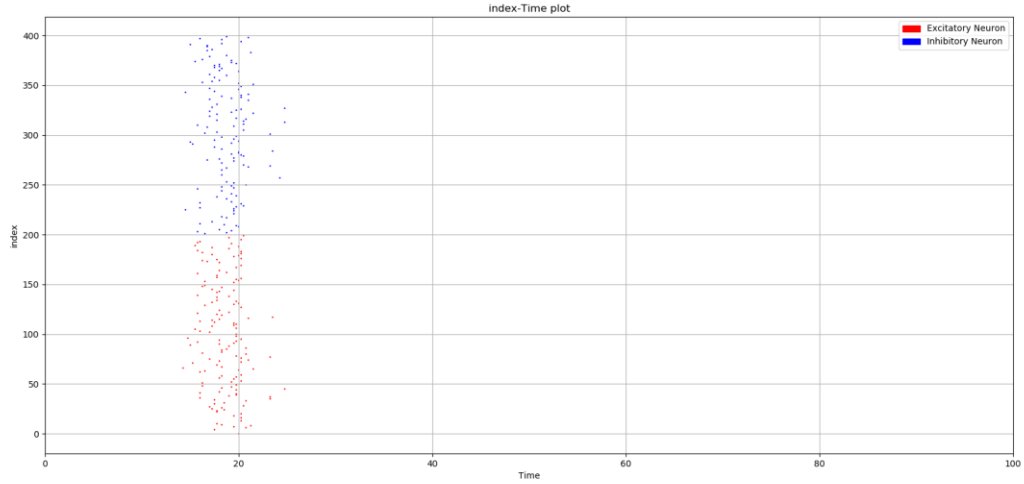


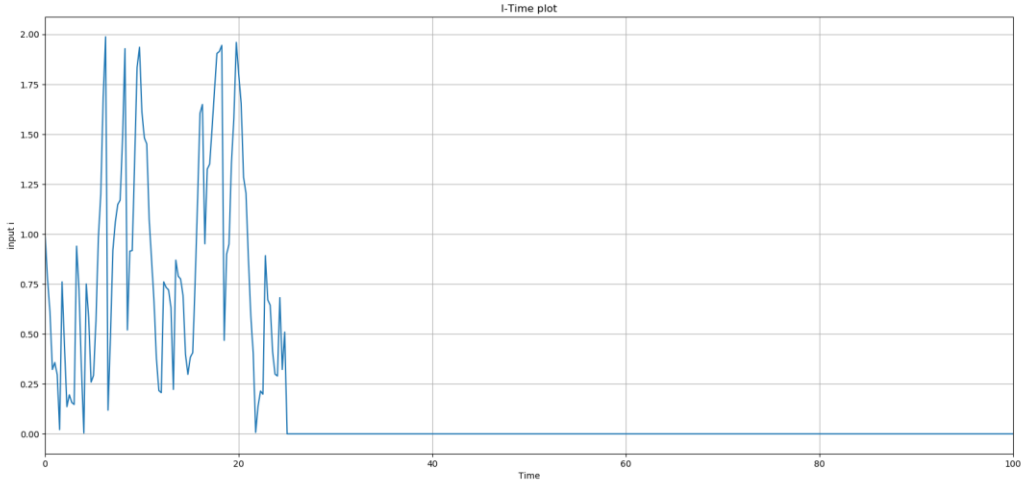


آزمایش بعدی که انجام دادیم اثر از بین رفتن جریان ورودی رو امتحان کردیم جریان ورودی بعد از یک مدت از بین میرود و اثر آن مورد بررسی قرار گرفت و به نتایج زیر رسیدیم :

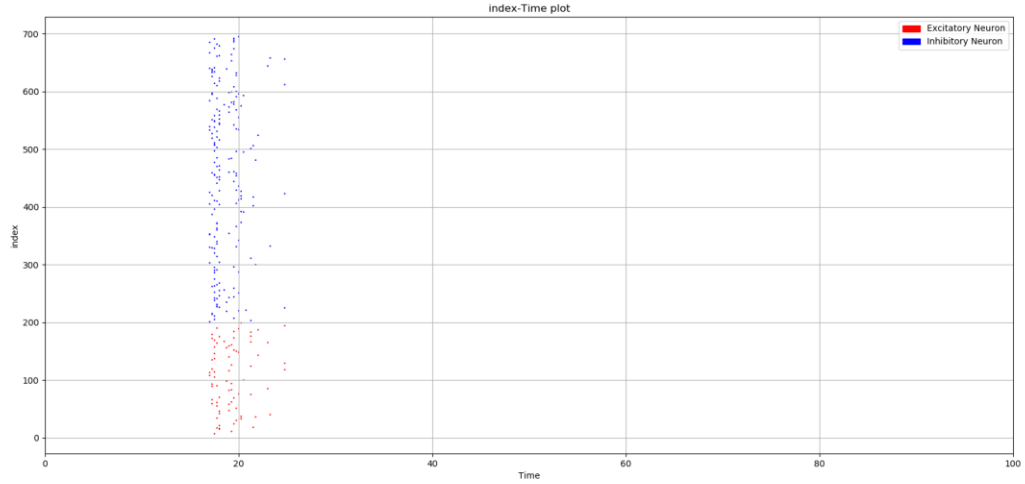


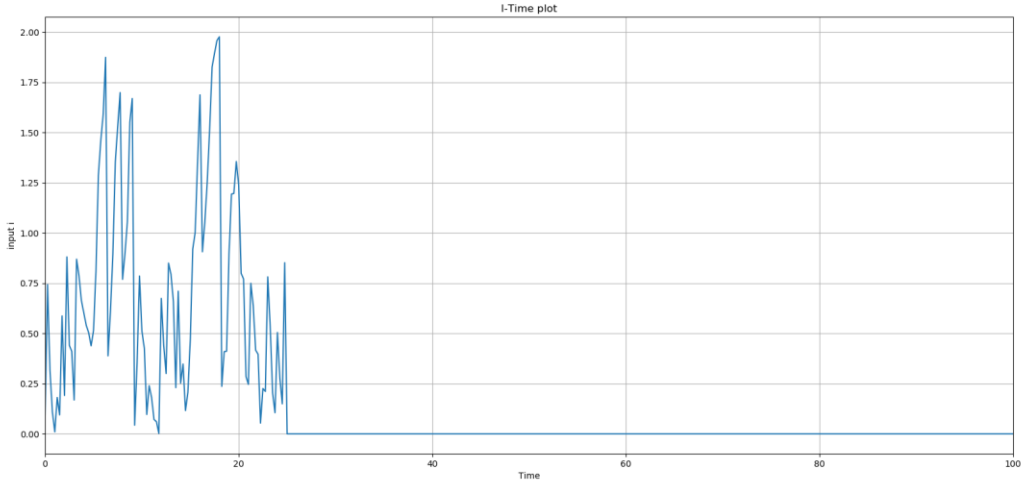
Population of 200 Excitatory Neurons and 200 Inhibitory Neurons



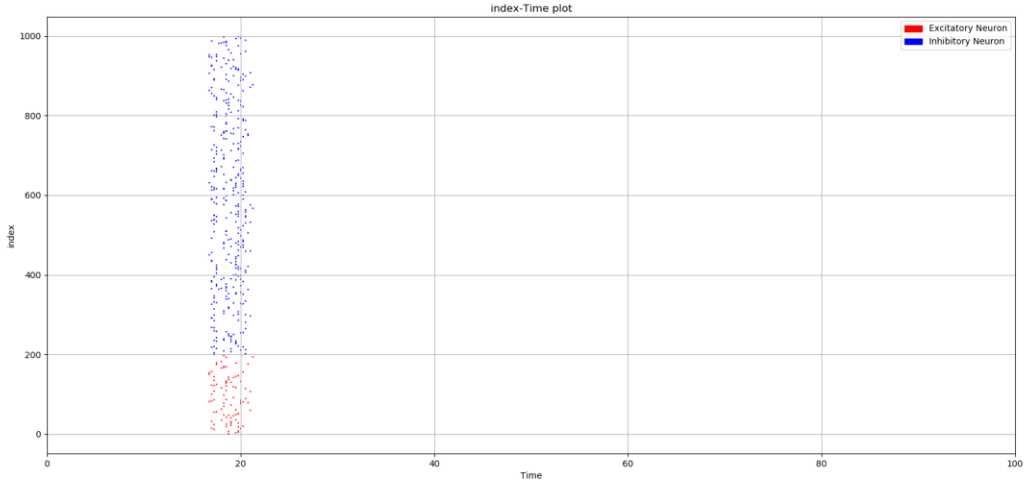


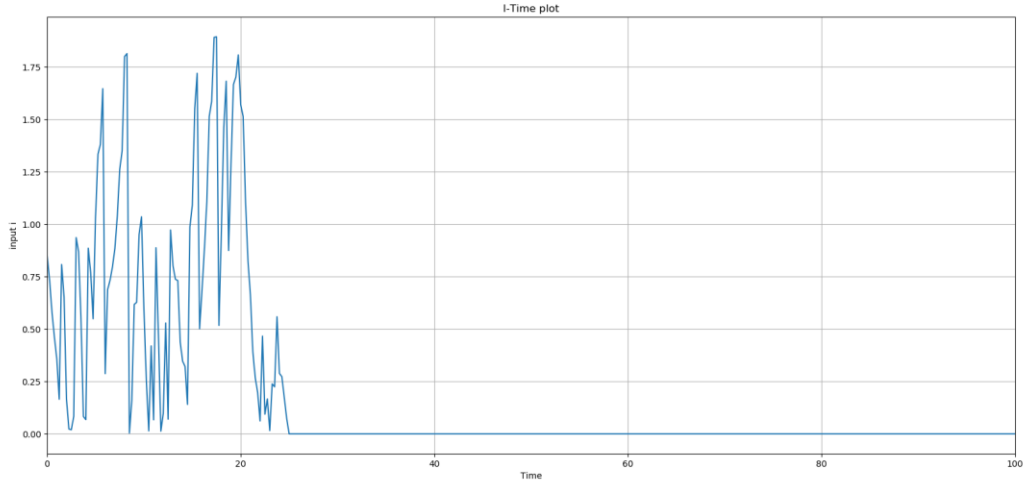
Population of 200 Excitatory Neurons and 500 Inhibitory Neurons

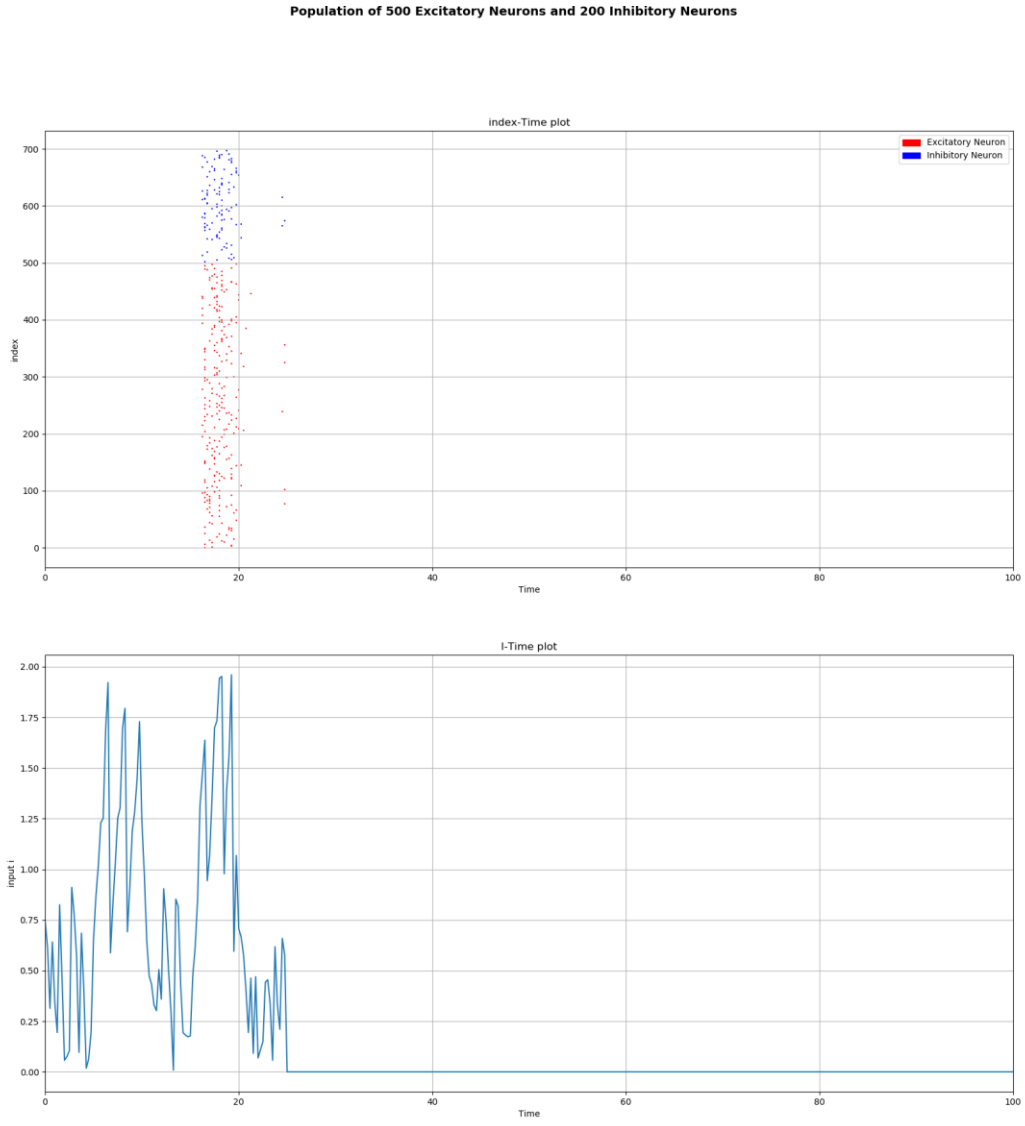




Population of 200 Excitatory Neurons and 800 Inhibitory Neurons







بعضی از نورون ها هنگامی که جریان رو به کم شدن میرود با جریان خیلی کم و با کمک ورودی از بقیه نورون ها فایر میکنند.

در آزمایشاتی که نورون های تحریکی بیشتر بودند شانس فایر کردن نورون ها بعد از کم شدن جریان ورودی افزایش پیدا میکند

تمرین شماره دو :

برای این تمرین کاملا مشابه تمرین قبل و از همان ساختار ها استفاده کردیم ولی این بار دو جمعیت نورونی exc و یک جمعیت inh ایجاد کردیم وموقع متصل کردن جمعیت ها از ماتریسی مشابه زیر استفاده کردیم:

0 0 0 0 4 0 0 0 0 4 2 2 2 2 0
0 0 0 0 0 0 0 0 0 0 2 2 2 0 2
0 0 0 0 0 0 0 0 0 0 2 2 0 2 2
0 0 0 0 0 0 0 0 0 0 2 0 2 2 2
0 0 0 0 0 0 0 0 0 0 0 2 2 2 2
0 0 0 0 0 3 3 3 3 0 0 0 0 0 4
0 0 0 0 0 3 3 3 0 3 0 0 0 0 0
0 0 0 0 0 3 3 0 3 3 0 0 0 0 0
0 0 0 0 0 3 0 3 3 3 0 0 0 0 0
0 0 0 0 0 0 3 3 3 3 0 0 0 0 0
2 2 2 2 0 0 0 0 0 0 0 0 0 0 4
2 2 2 0 2 0 0 0 0 0 0 0 0 0 0
2 2 0 2 2 0 0 0 0 0 0 0 0 0 0
2 0 2 2 2 0 0 0 0 0 0 0 0 0 0
0 2 2 2 2 0 0 0 0 0 0 0 0 0 0

مقادیری که تشکیل مربع های کوچیک تر داده اند گروه های جدا جدا هستند که از هم جدا هستند و دو گروه تحریکی با اتصالات با وزن های قرمز رنگ به گروه مهاری متصل شده اند و با این ایده با کمترین تغییر در کد تمرین قبل ۳ گروه مجزا ساختیم که از طریق گروه مهاری به هم متصل میشود برای این کار از کد زیر استفاده کردیم :

```
w = [[0 for x in range(N)] for y in range(N)]

for i in range(inhibitory_N):
    for j in range(inhibitory_N):
        if i != j:
            # w[i][j] = random.random()
            w[i][j] = 2
for i in range(inhibitory_N, inhibitory_N + excitatory_N1):
    for j in range(inhibitory_N, inhibitory_N + excitatory_N1):
        if i != j:
            # w[i][j] = random.random()
            w[i][j] = 3
for i in range(inhibitory_N + excitatory_N1, inhibitory_N + excitatory_N1 + excitatory_N2):
    for j in range(inhibitory_N + excitatory_N1, inhibitory_N + excitatory_N1 + excitatory_N2):
        if i != j:
            # w[i][j] = random.random()
            w[i][j] = 2

# w[excitatory_N1][0] = random.random()
# w[0][excitatory_N1] = random.random()
w[excitatory_N1][0] = 4
w[0][excitatory_N1] = 4

# w[excitatory_N1 + excitatory_N2][0] = random.random()
# w[0][excitatory_N1 + excitatory_N2] = random.random()
w[excitatory_N1 + excitatory_N2][0] = 4
w[0][excitatory_N1 + excitatory_N2] = 4
```

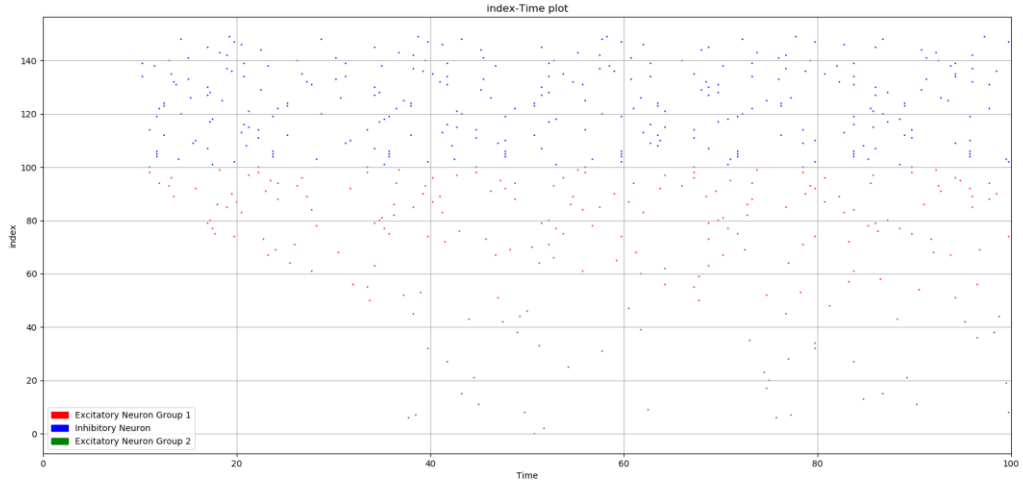
در بعضی از ازمایشات اتصالات بین نورون های را بطور رندوم ساختیم و نتایج را مشاهده کردیم

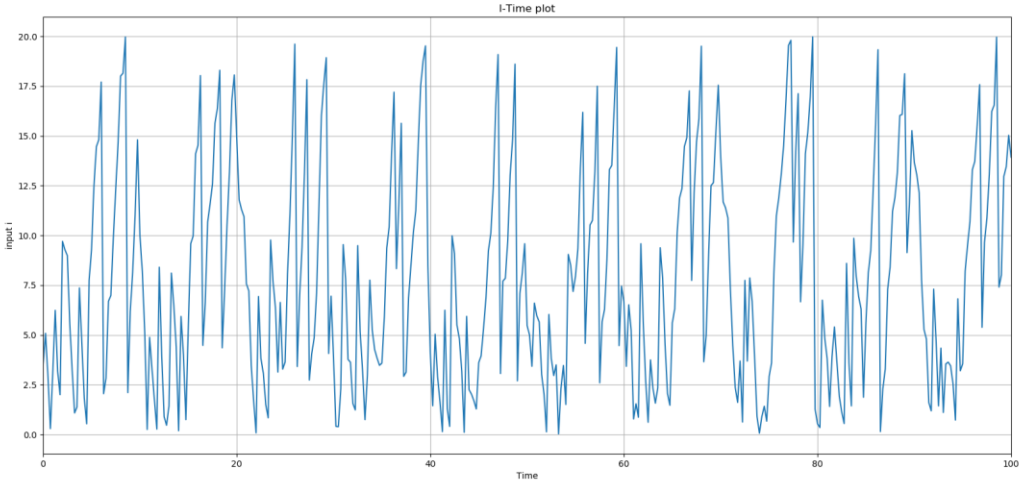
هنگام شبیه سازی به یکی از گروه ها به مقدار ۱،۰۲ برابر جریان ورودی مقدار دادیم و به جمعیت مهاری تنها اندک جریان ۰،۴ دادیم که تنها در حالت نرمال و فایر کردن با فرکانس پایین باشد و انتظار داشتیم که جمعیتی که جریان ورودی بیشتری میگیرد با فرکانس و چگالی بیشتری در نمودار فایر کند و تصمیم نهایی با جمعیت مورد نظر باشد که این ویژگی را در نتایج مشاهده کردیم:

```
for i in range(len(group[0].timer)):
    print(round(i / len(group[0].timer), 4) * 100, "%")
    current i = i gen.get(i)

    yi[i] = current_i

    for g in group:
        if g.type == "inh":
            current_i = 0.4
        if g.group == 2:
            current i *=1.02
        result = g.update(j=i, input=current_i)
        if result > 0:
            y[i][g.index] = g.index
            for k in range(N):
                g.update_u(w[g.index][k] * result, j=i)
```

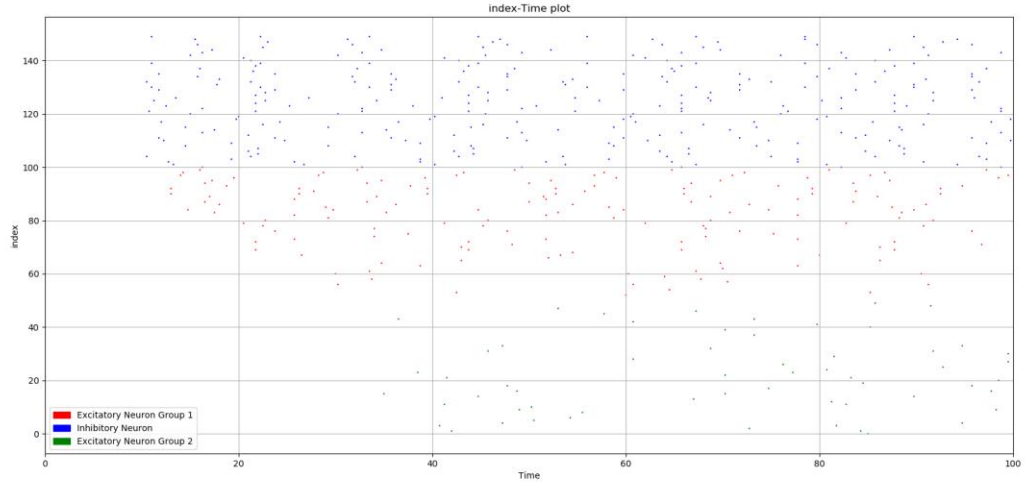
در دو نمودار زیر اتصال نرون ها ثابت بوده و تعمیم یافته جدول وزن های اول گزارش است ولی با ۱۵۰ نورون این دو آزمایش با جریان های های ورودی متفاوت انجام شده است :

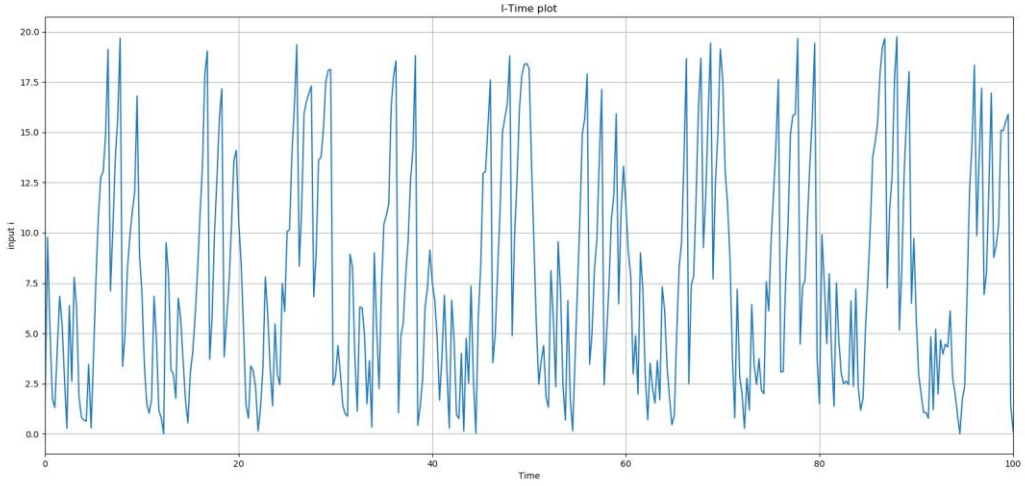
Population of 50 ,50 Excitatory Neurons and 50 Inhibitory Neurons



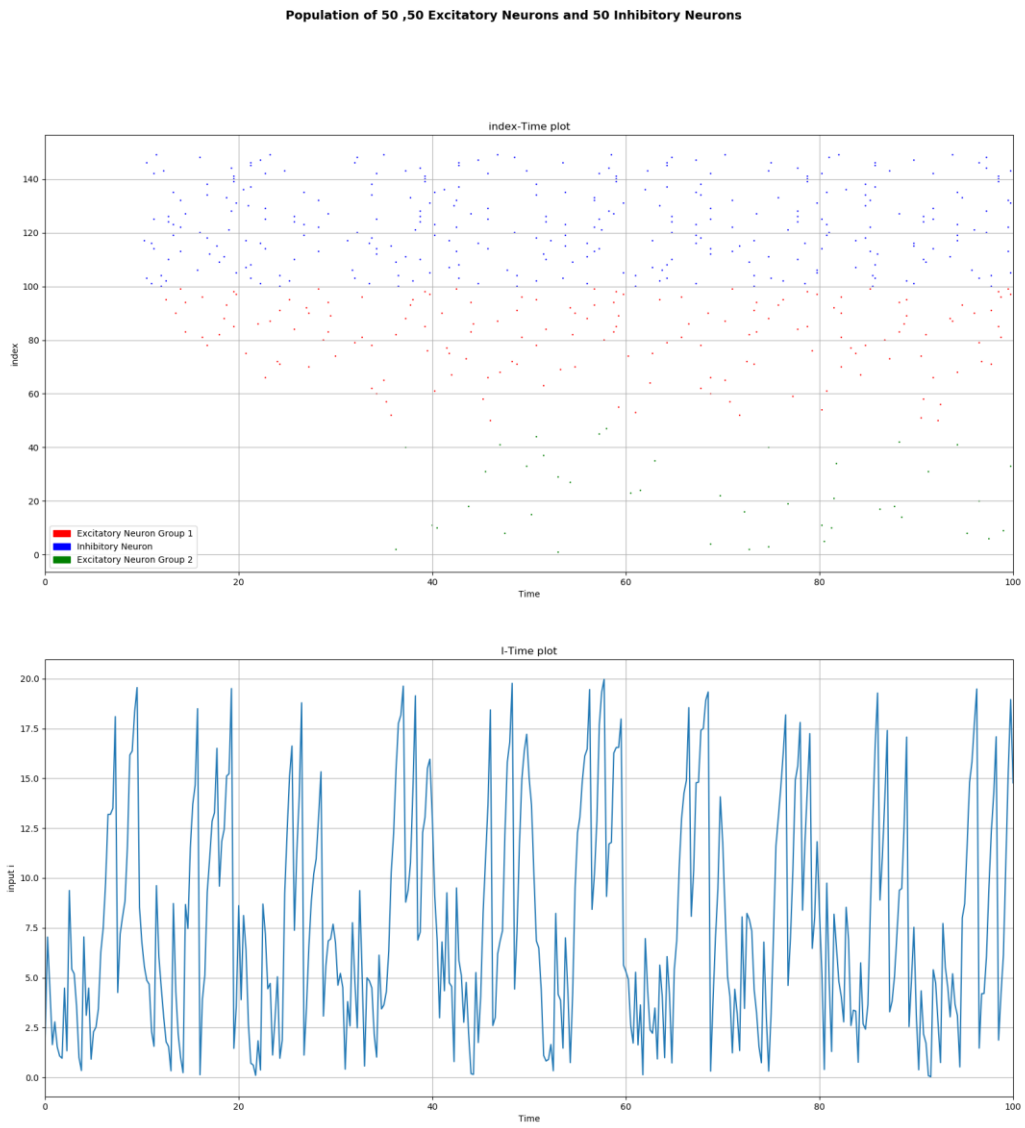


Population of 50 ,50 Excitatory Neurons and 50 Inhibitory Neurons





آزمایش فوق را با وزن های اتصال رندوم انجام دادیم و نتیجه زیر رابدست اوردیم



نتایج بدست آمده همانطور ک انتظار میرفت فرکانس فایر جمعیتی که ورودی کمتری میگرفت کمتر هست و در نهایت تصمیم با جمعیت قرمز رنگ میباشد و گروه مهاری قرمز غالب برگروه مهاری سبز میشود.