Computation NeuroScience, Homework 3

장지원 202211167

[Exercise 3.1]

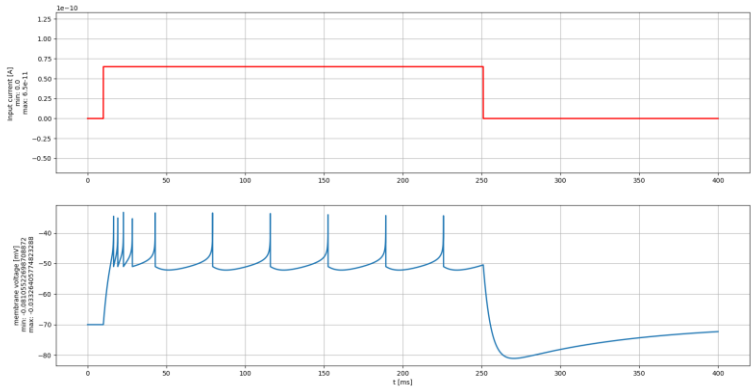
다음 model은 AdEx model(Exponential Integrate-and-Fire model with a single adaptation current)이다.

$$\begin{bmatrix} \tau_m \frac{du}{dt} & = & -(u-u_{rest}) + \Delta_T exp(\frac{u-\vartheta_{rh}}{\Delta_T}) - Rw + RI(t) \\ \tau_w \frac{dw}{dt} & = & a(u-u_{rest}) - w + b\tau_w \sum_{t^{(f)}} \delta(t-t^{(f)}) \end{bmatrix}$$

```
MEMBRANE_TIME_SCALE_tau_m = 5 * b2.ms
MEMBRANE_RESISTANCE_R = 500*b2.Mohm
V_REST = -70.0 * b2.mV
V_RESET = -51.0 * b2.mV
RHEOBASE_THRESHOLD_v_rh = -50.0 * b2.mV
SHARPNESS_delta_T = 2.0 * b2.mV
ADAPTATION_VOLTAGE_COUPLING_a = 0.5 * b2.nS
ADAPTATION_TIME_CONSTANT_tau_w = 100.0 * b2.ms
SPIKE_TRIGGERED_ADAPTATION_INCREMENT_b = 7.0 * b2.pA
```

위 식에서의 default parameter는 위와 같다. 이 때 reset은 위의 AdEx model에서 존재하지 않는 parameter인데, 이는 firing 이후에 어디까지 내려올지를 결정하는 parameter이다.

이를 바탕으로 default를 이용해 시뮬레이션을 해보았다. 이 때 input으로 65pA를 10~250ms동안 인가해 주었다. 결과는 밑의 그림과 같다. 초반에는 점차 adaptation 되어서 firing이 간격이 전보다 delay되는 모습을 보이고 있고(initial bursting), 점차 시간이 지나며 firing 간격이 일정해지는 형태(tonic)를 보인다.



3.1.1

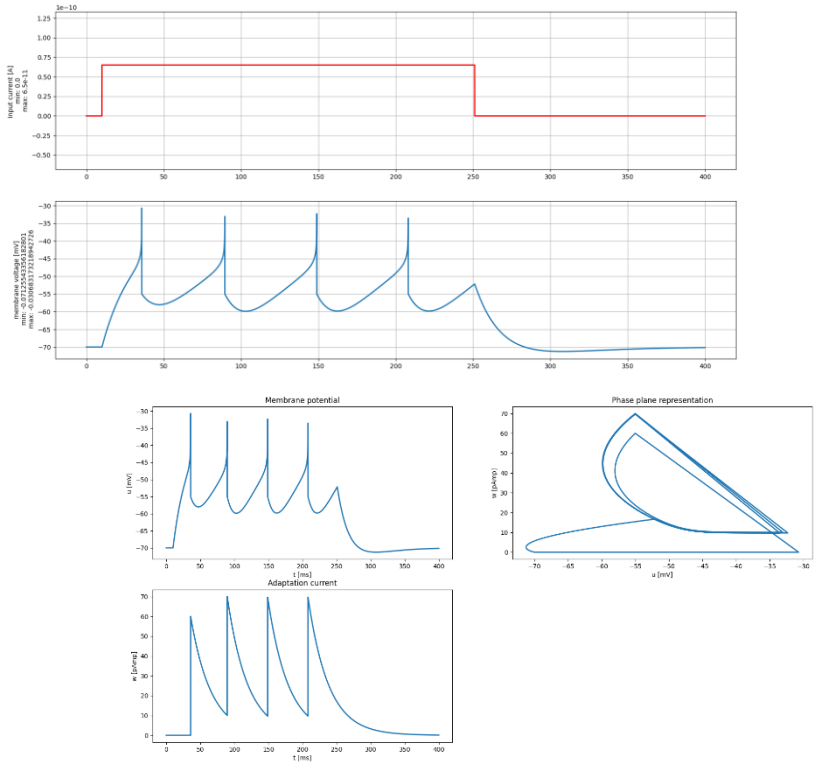
여러 가지 firing pattern을 만들어 보자. 위의 default는 initial bursting pattern을 만들고 있다. parameter 값을 바꿔서 tonic, adapting, initial bursting, irregular 등의 여러가지 pattern을 만들어 보자.

: 각 pattern을 만들기 위한 parameter 값은 밑의 표를 참고하였다. 전류는 default를 실행할 때와 같이 65pA를 10~250ms동안 인가해 주었다.

Type	τ_m(ms)	a(ns)	τ_w(ms)	b(pA)	u_r(mV)
Tonic	20	0.0	30.0	60	-55
Adapting	20	0.0	100	5.0	-55
Init. brust	5.0	0.5	100	7.0	-51
Bursting	5.0	-0.5	100	7.0	-46
Irregular	9.9	-0.5	100	7.0	-46
Transient	10	1.0	100	10	-60
Delayed	5.0	-1.0	100	10	-60

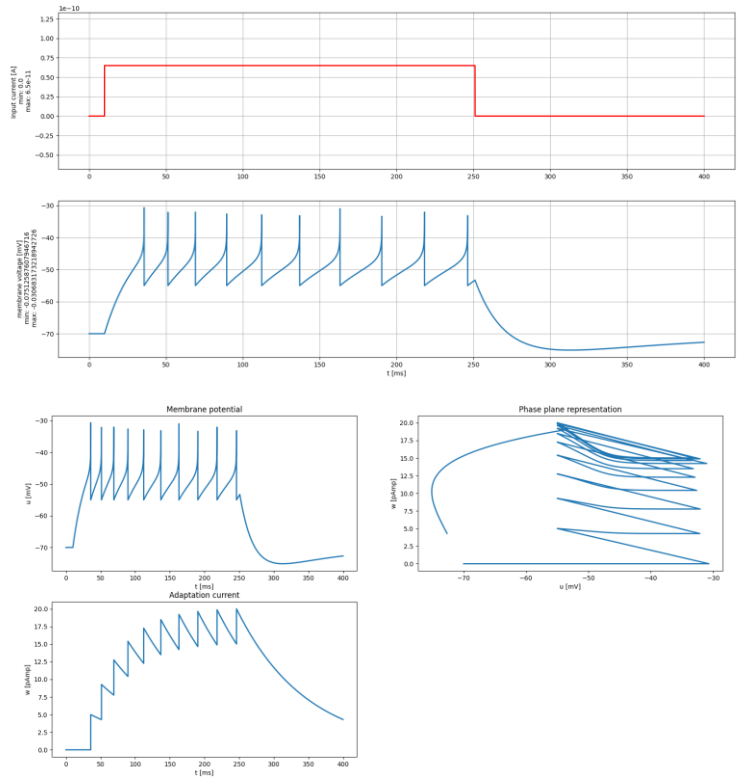
결과는 다음과 같았다. 위와 같이 input과 firing을 확인할 수 있는 plot을 출력해 보았고, u와 w, 그리고 u-w phase plane을 확인해볼 수 있는 plot도 추가로 출력해 보았다. 두 번째 plot은 `AdEx.plot_adex_state()` 함수를 사용해서 만들었다.

a. Tonic



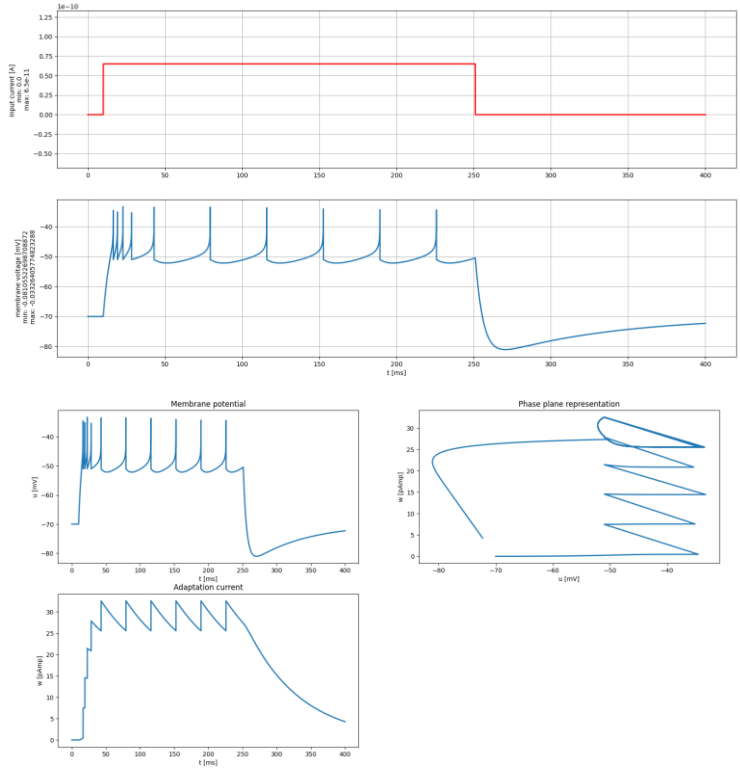
b. Adapting

adaptation에서는 b값을 작게 준다.

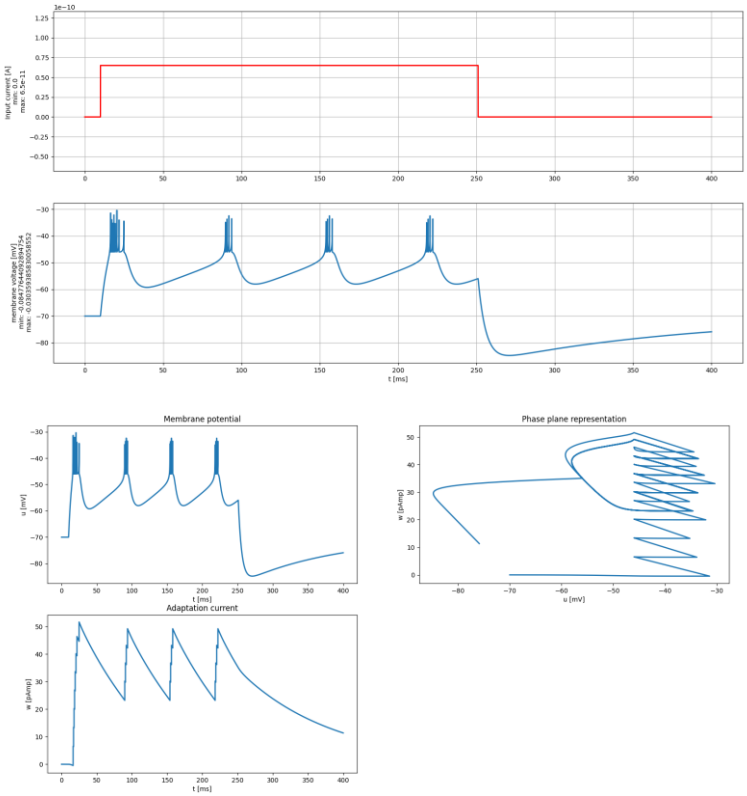


c. Init. burst

Init. burst에서는 τ_m의 값을 줄여 주었다. 이는 u의 dynamics를 빠르게 하겠다는 의미이다.

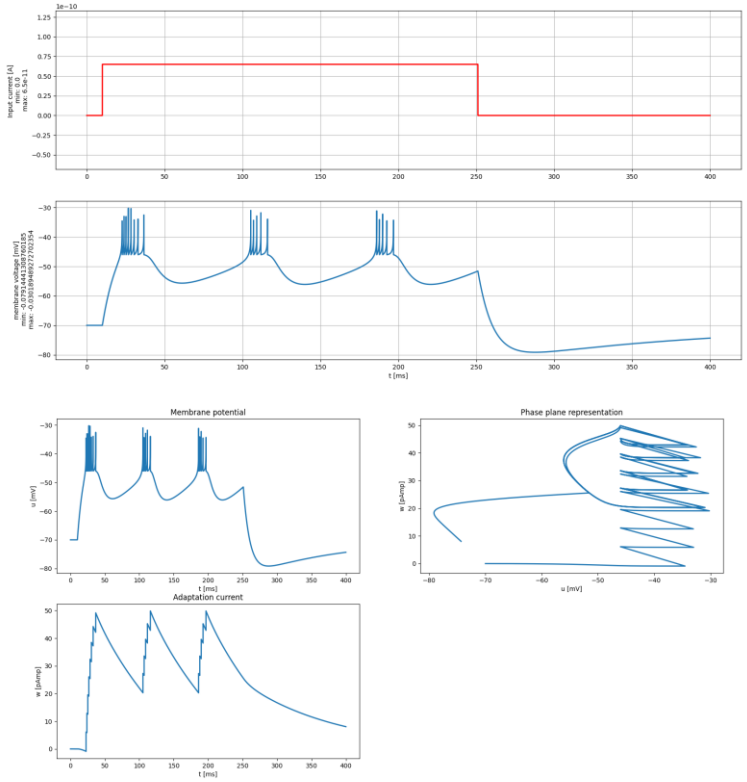


d. Bursting

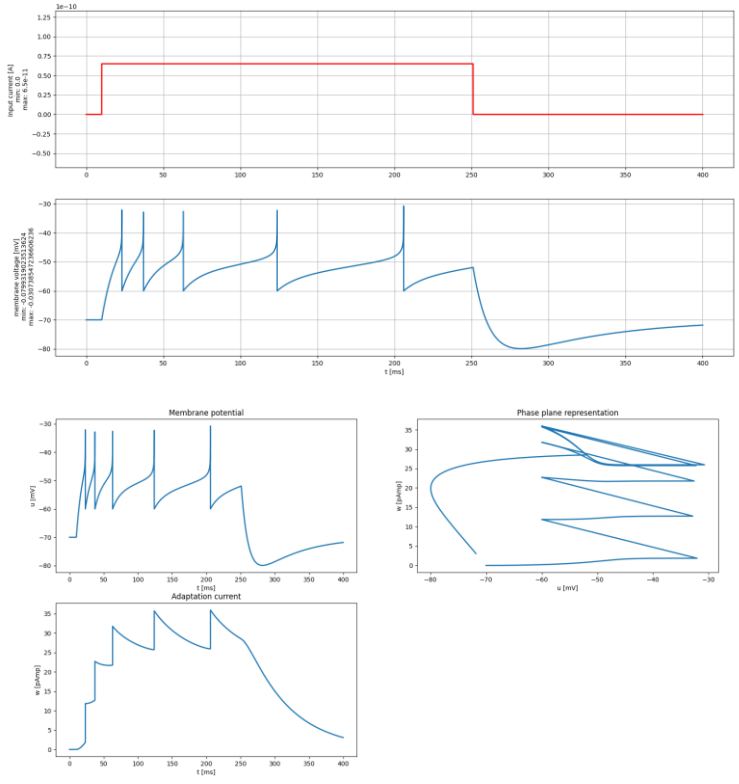


e. Irregular

Bursting에서 보다 τ_m의 값을 늘려 주었다. 이는 u의 dynamics를 천천히 하겠다는 의미이다.

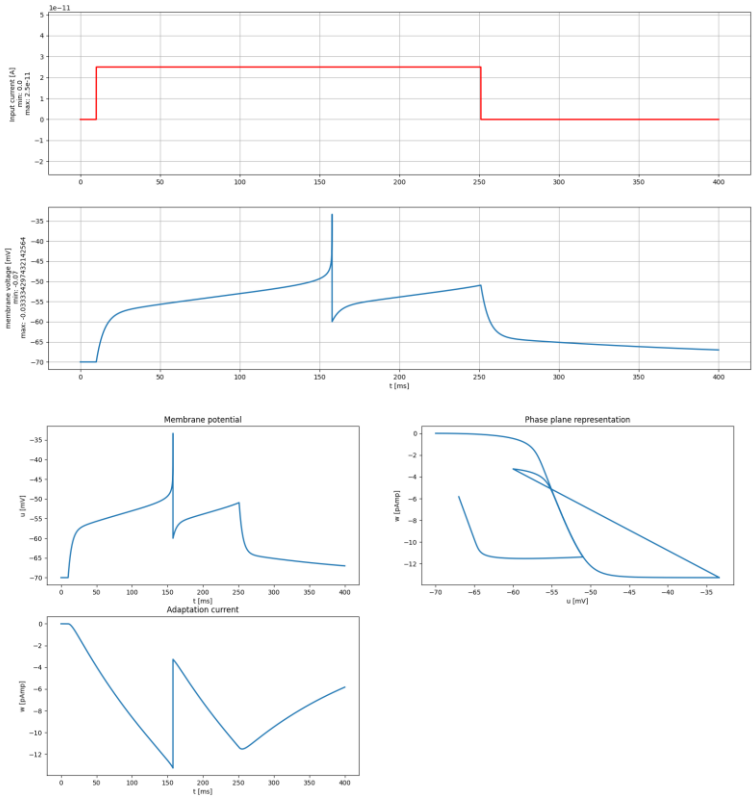


f. Transient



g. Delayed

Delay일 때는 25pA로 다른 pattern들보다 적은 input을 인가해 주었다. 너무 큰 input을 준다면 delay되지 않을 수 있다. 이 때는 한 번만 firing 된다.



우리는 이 Exercise를 통해서 AdEx model의 각 parameter가 각 pattern에 어떤 영향을 끼치는 지 이해해 볼 수 있다.

[Exercise 10.1]

network of sparsely connected Leaky-Integrate-And-Fire neurons에 대한 시뮬레이션을 진행해 볼 것이다.

10.1.1

a. N_E, N_I, C_E, C_I, W_{EE}, W_{EI}, W_{IE}, W_{II} 가 어떤 값들인지 설명해 보아라.

N_E: number of excitatory neuron *(default: 5000)*

N_I: number of inhibitory neuron *(default = N_E/4)*

C_E: N_E*connection probability *(connection probability는 뉴런이 connect 되어 있을 확률이다, default = 0.1)*

C_I: N_I*connection probability

W: Synaptic strength

b. W의 단위는 무엇인가?

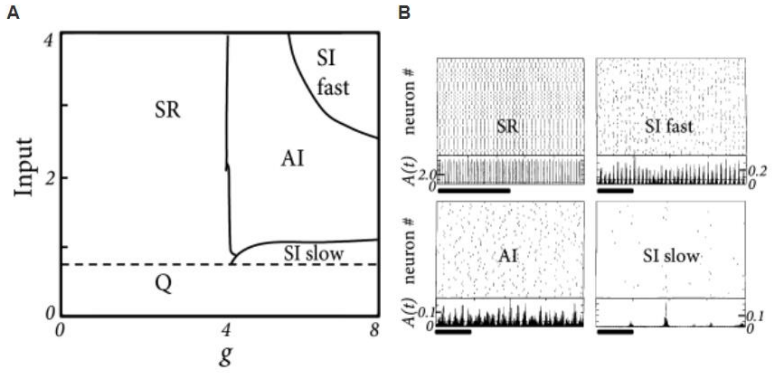
: W의 voltage로 전위의 단위이다.

c. $v_{threshold}$구해보아라.

: 이것은 밑의 식과 같이 정의할 수 있고, default 값을 넣어서 계산하면 20Hz가 나온다. `LIF_spiking_network.FIRING_THRESHOLD()` 함수를 사용하면 default 값(20Hz)이 출력된다.

$$\nu_{threshold} = \frac{u_{thr}}{N_{extern} w_0 \tau_m}$$

d. 밑의 그림에서 1이 의미하는 게 무엇인지 설명하라. 그리고 수평선이 의미하는 게 무엇인가?



: 1은 threshold일 때의 input 값을 의미한다. 수평선 밑은 Q(정지 상태)를 의미한다.

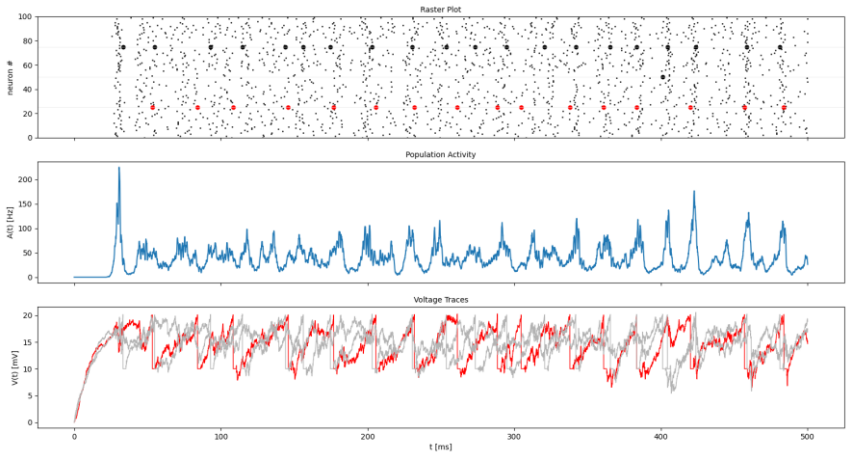
e/f. 500ms 동안 시뮬레이션을 돌려 보고, 해당 시간 동안의 network activity를 plot으로 그려 보아라. 이 때 single neuron의 firing rate는 얼마인가?

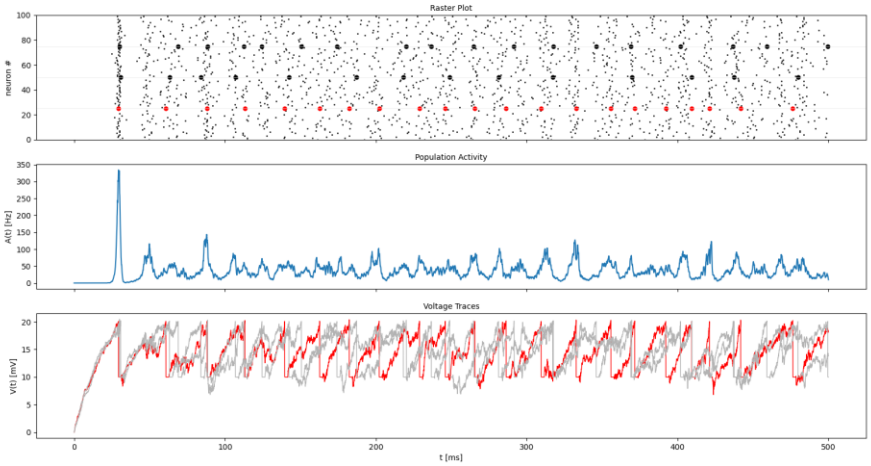
이 문제에서는 poisson_input_rate를 바꿔가며 plot을 그려볼 것이다.

1. poisson_input_rate = 12일 때

: plot은 밑과 같이 나오고, SR인 것처럼 보인다. 하지만 완벽히 Synchronize되지는 않은 것을 확인할 수 있는데, 이는 우리가 g값을 4.0(balanced)으로 주었기 때문이다. 두 번째 plot은 g를 3.0으로 설정했을 때의 plot이다. g를 3.0 정도로 주면 4.0으로 주었을 때보다 더 Synchronize되는 것을 확인할 수 있다.

전체 spike는 122,011개가 나왔다. 이를 6250(number of neuron)으로 나누고, 이 값을 duration(0.5ms)로 나눠 준다면, single neuron의 firing rate를 구할 수 있다, single neuron의 firing rate는 대략 39 정도이다.

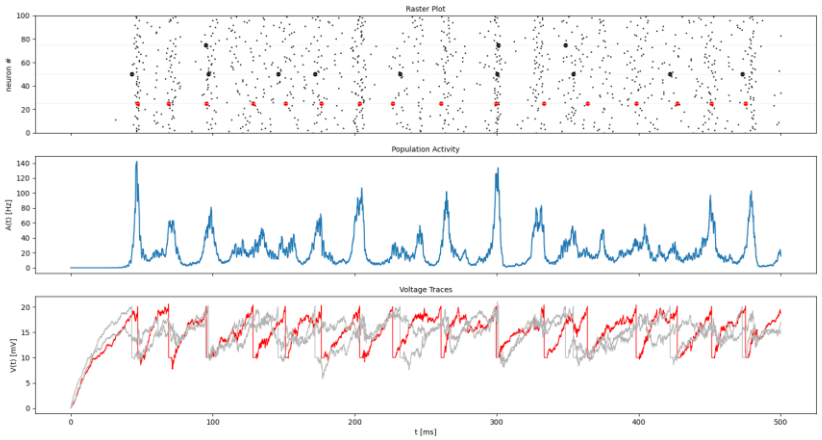




2. poisson_input_rate = 10일 때

: poisson_input_rate가 10인 경우는 threshold 값일 때를 의미한다. plot은 밑과 같이 나온다. threshold일 때는 Q(정지 상태)가 되어야 하지만, 아직은 SR 영역에 있으며, 정지 상태가 되지 않은 것을 확인할 수 있다.

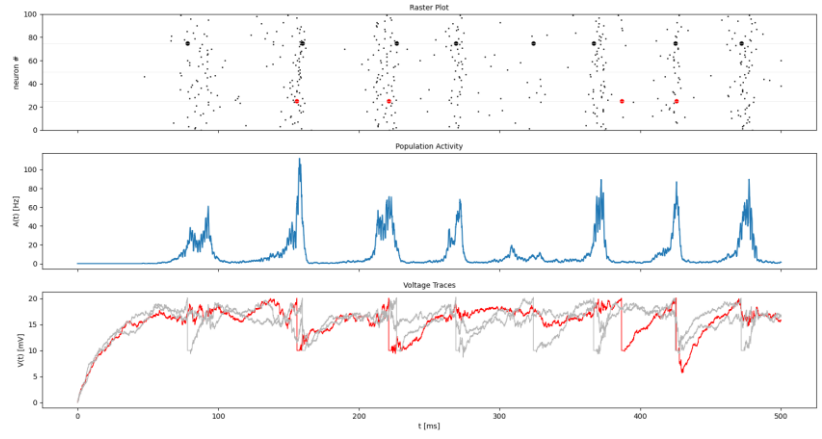
전체 spike는 66,953개가 나왔다. 이를 6250(number of neuron)으로 나누고, 이 값을 duration(0.5ms)로 나눠 준다면, single neuron의 firing rate를 구할 수 있다, single neuron의 firing rate는 대략 21 정도이다.



3. poisson_input_rate = 9일 때

: plot은 밑과 같이 나온다. 아직까지는 SR 영역에 있고, 정지 상태가 되지 않은 것을 확인할 수 있다.

전체 spike는 30,378개가 나왔다. 이를 6250(number of neuron)으로 나누고, 이 값을 duration(0.5ms)로 나눠 준다면, single neuron의 firing rate를 구할 수 있다, single neuron의 firing rate는 대략 9 정도이다.

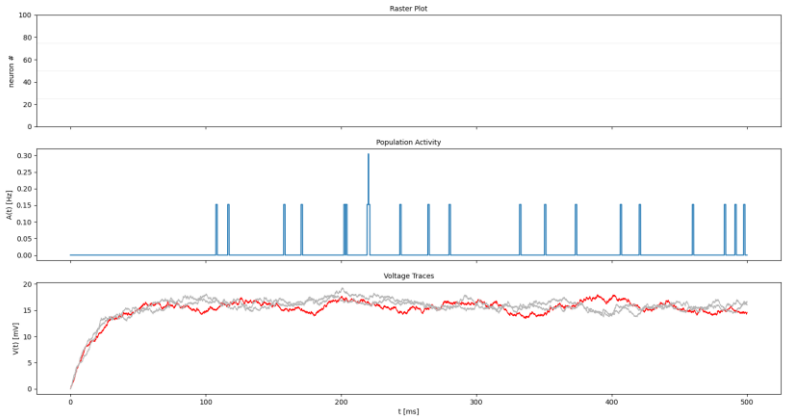


4. poisson_input_rate = 8일 때

: plot은 밑과 같이 나온다. input_rate가 8정도 되었을 때, 정지 상태가 됨을 확인할 수 있다.

전체 spike는 20개가 나왔다. 이를 6250(number of neuron)으로 나누고, 이 값을 duration(0.5ms)로 나눠 준다면,

single neuron의 firing rate를 구할 수 있다, single neuron의 firing rate는 대략 0 정도이다.



[Exercise 7.4]

Hopfield network에 대해서 알아보자.

7.4.1

N = 14X14일 때 얼마나 많은 pattern(K)을 저장할 수 있는가? C_{store}를 이용해서 설명하라. 구한 값, K는 밑의 문제에서 활용된다.

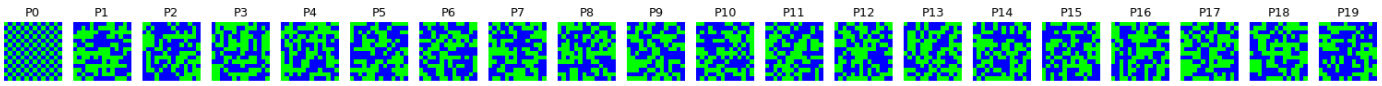
$$C_{\text{store}} = \frac{M^{\max}}{N} = \frac{M^{\max} N}{N^2}$$

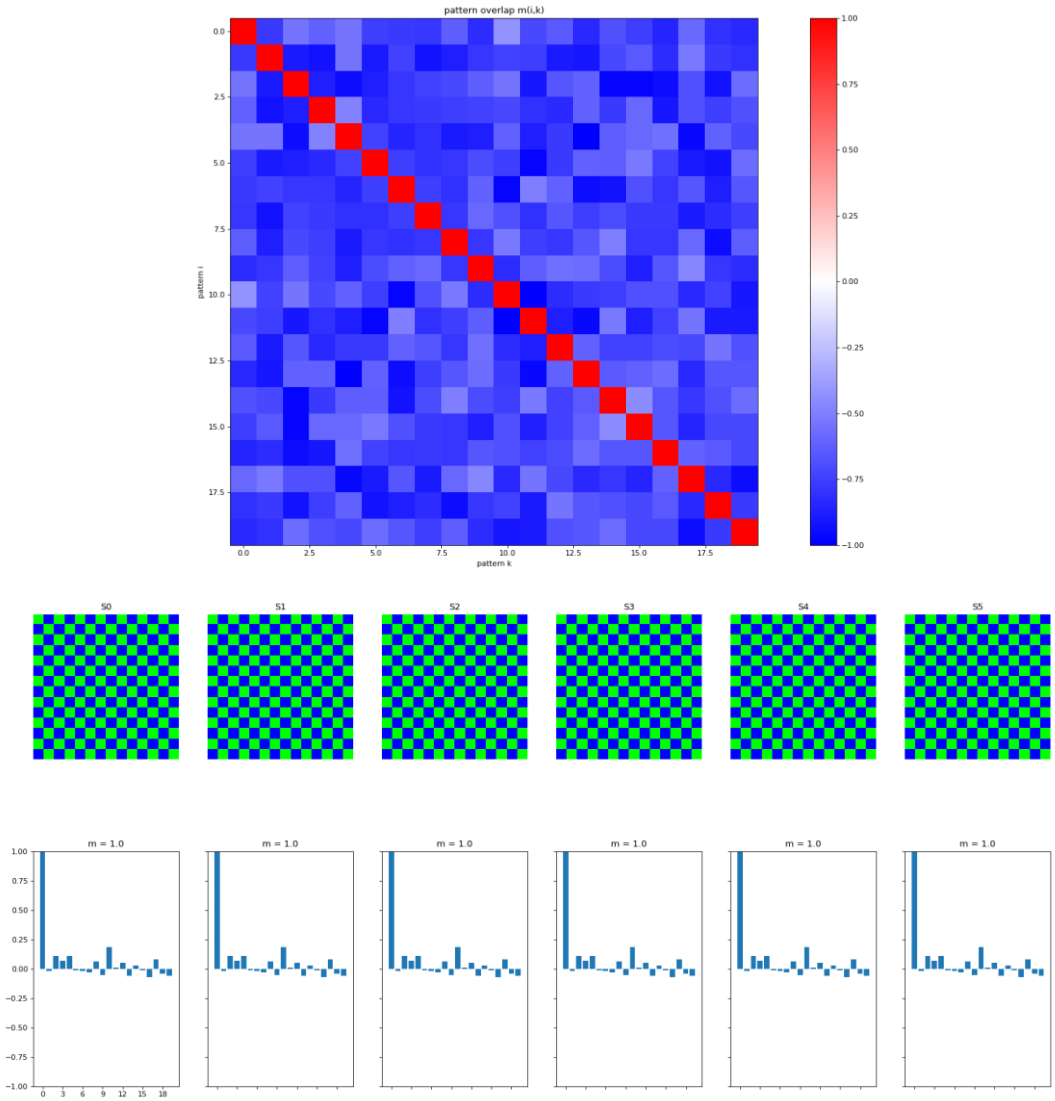
: Test book에 따르면 P_{error} = 0.001일 때의 C_{store}는 0.105 정도이다. 따라서 N이 14X14일 때는 M^{max}가 20정도가 된다. 이 때 M^{max}는 prototype의 개수, 즉 pattern을 저장할 수 있는 개수가 되므로 N = 14X14일 때는 20개 정도의 pattern을 저장할 수 있다고 말할 수 있다.

7.4.2

14X14 형태의 checker board와 K-1(19)개의 pattern을 생성해 보고, network를 checker board로 initializing한 후, 다섯 번의 반복 network evolve를 진행해 보자.

: 결과(첫 번째 실행에서의 결과)는 다음과 같다. Prototype은 매 실행마다 random하게 생성되기 때문에, 밑의 실행 외에도 2번의 실행을 추가로 해보았다(총 3번의 실행을 진행했다). 매 실행에서의 결과는 동일하게 나왔다. 즉 K=20일 때는 5번의 evolve만으로 checker board를 거의 잘 찾아가는 것이다. 물론 prototype이 checker board와 유사하게 생긴 경우에는 못 찾아갈 수도 있을 것이다.

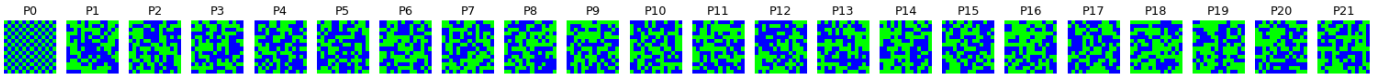


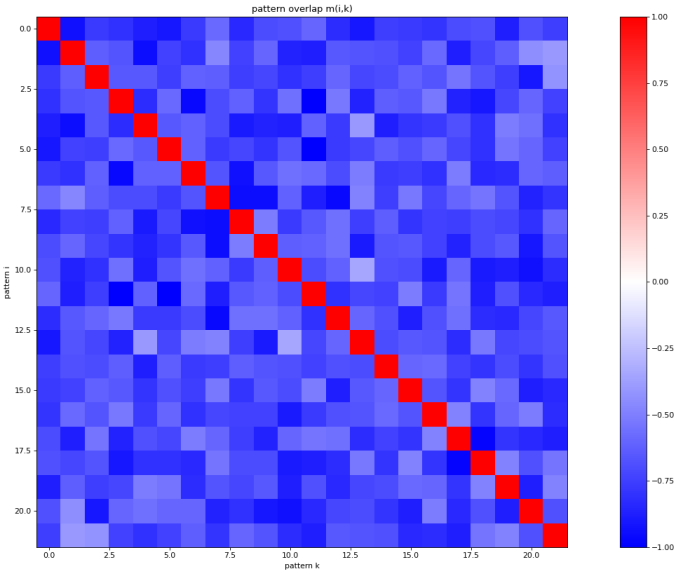


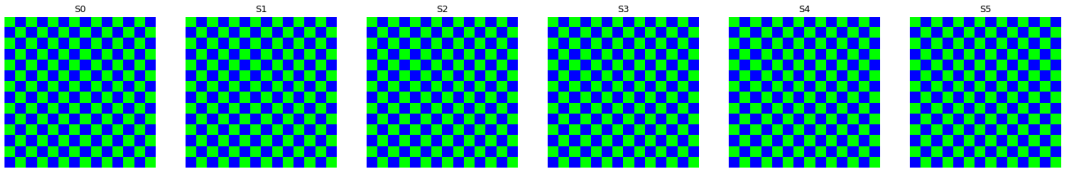
7.4.3

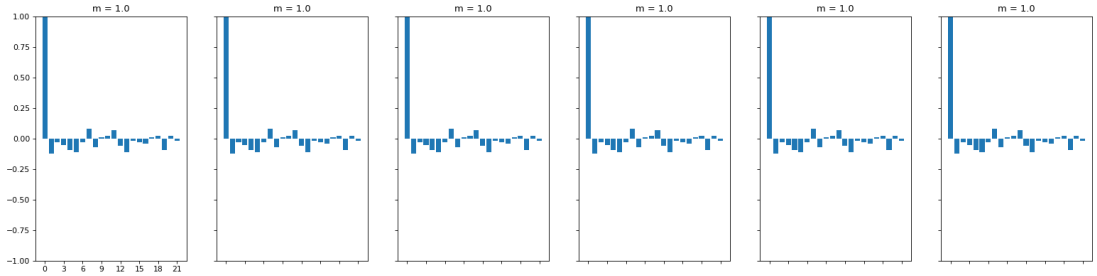
Pattern의 수가 K*11, K*15일 경우에는 무엇을 관찰할 수 있는가?

: Pattern의 수가 K*11=22개일 때는 다음과 같은 결과(첫 번째 실행에서의 결과)가 나온다. 밑의 실행 외에도 2번의 실행을 추가로 해보았다(총 3번의 실행을 진행했다). 매 실행에서의 결과는 동일하게 나왔다. 즉 K=22일 때는 5번의 evolve만으로 checker board를 거의 잘 찾아가는 것이다. 물론 prototype이 checker board와 유사하게 생긴 경우에는 못 찾아갈 수도 있을 것이다.

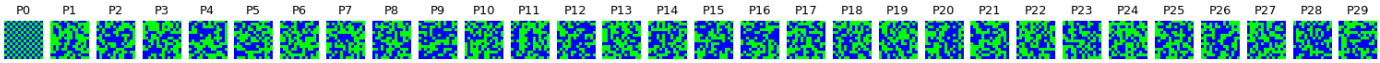


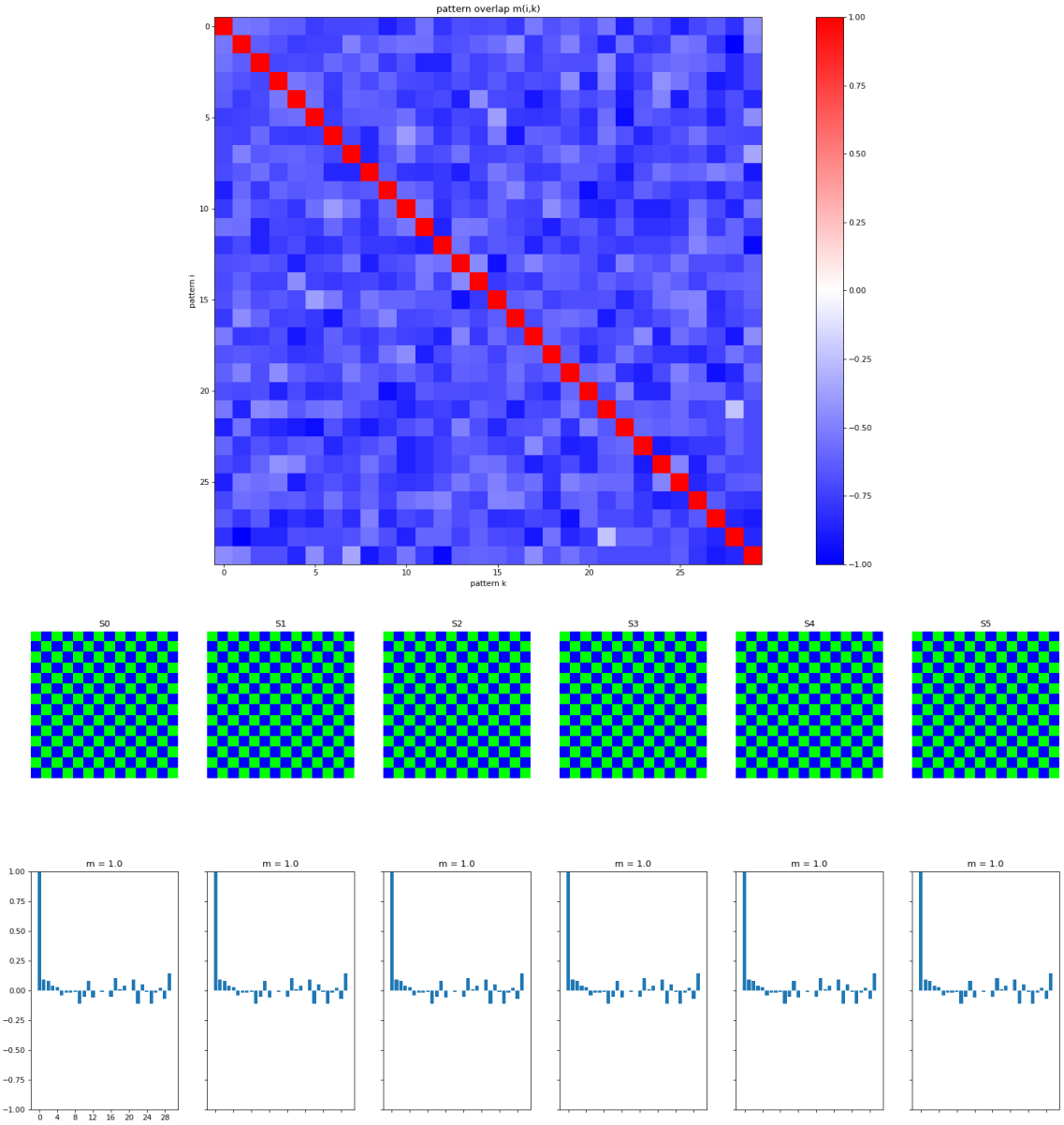




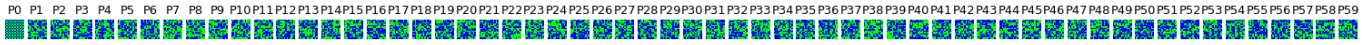


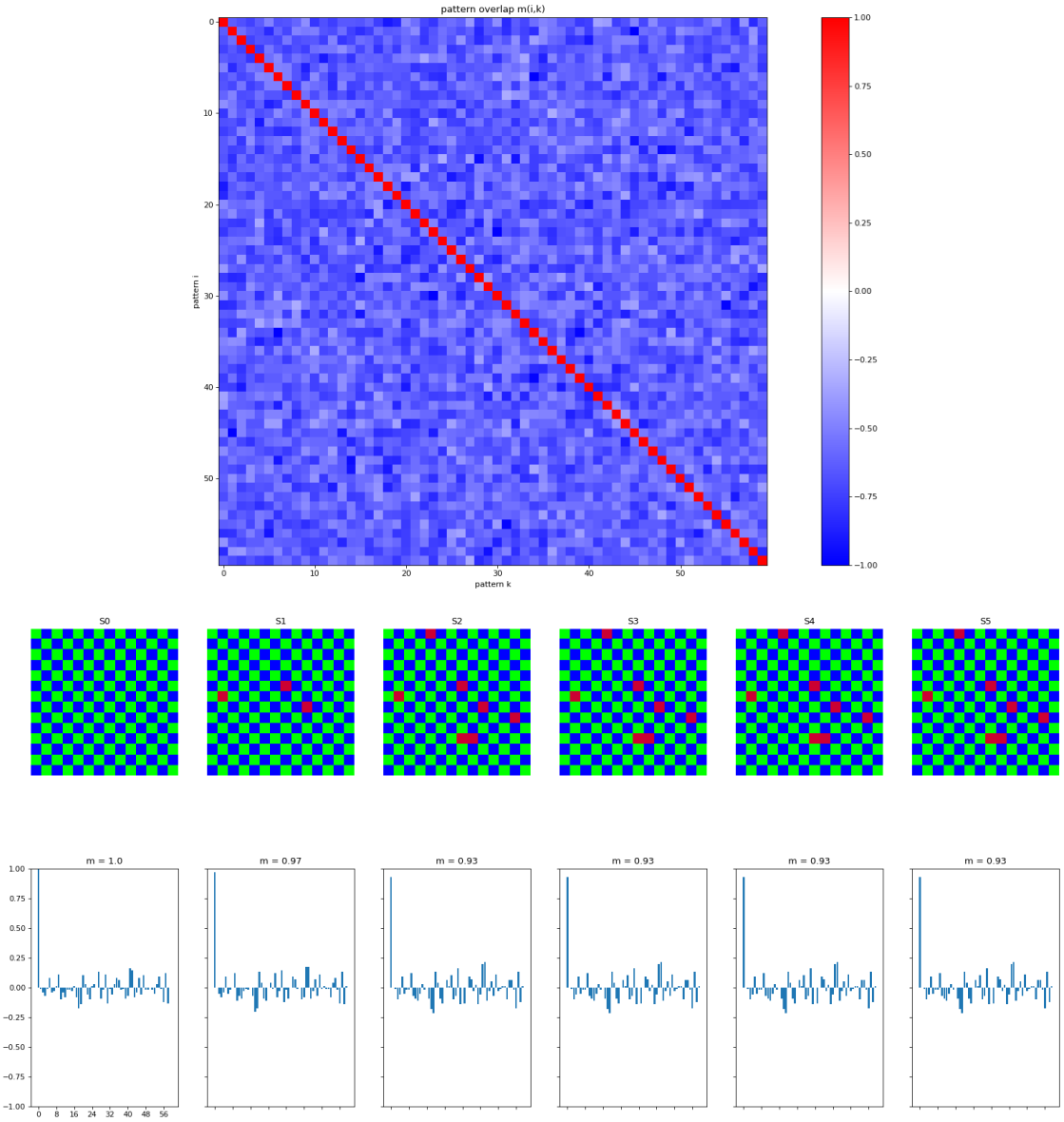
: Pattern의 수가 K*15=30개일 때는 다음과 같은 결과(첫 번째 실행에서의 결과)가 나온다. 밑의 실행 외에도 2번의 실행을 추가로 해보았다(총 3번의 실행을 진행했다). 첫 번째와 두 번째 실행의 결과는 동일하게 나왔고, 마지막 실행에서는 다른 결과가 나왔다. 즉 K=30일 때는 5번의 evolve만으로 checker board를 대부분 잘 찾아가지만 낮지 않은 확률로 찾지 못하는 경우가 있는 것이다.





: 추가로 pattern의 수를 60으로 하여 한 번 더 실행을 진행해 보았다. 이 경우에서는 모든 실행에서 checker board를 찾아가지 못했다.





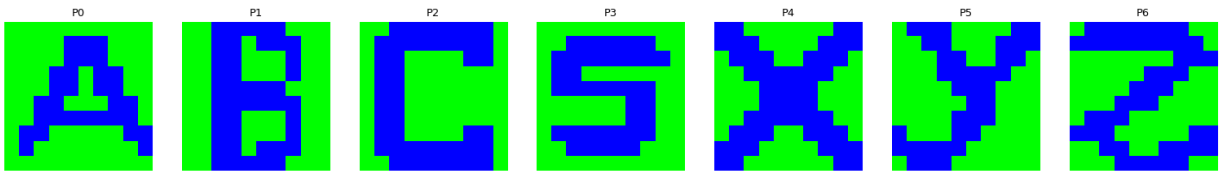
[Exercise 7.5]

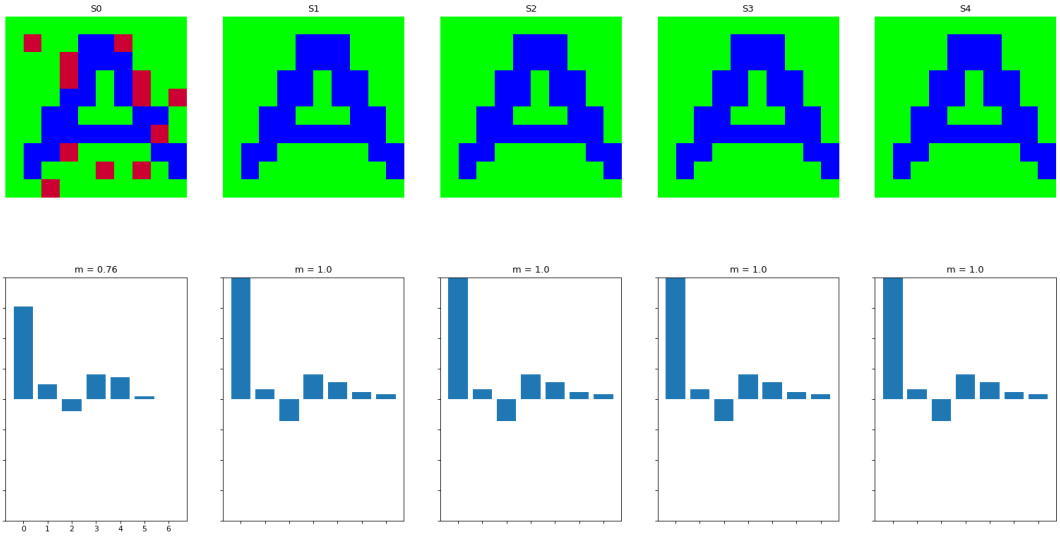
이 파트에서는 랜덤이 아닌 pattern을 가지고 시뮬레이션을 진행해 볼 것이다.

7.5.1.

Pattern 을 알파벳(`letter_list = ['A', 'B', 'C', 'S', 'X', 'Y', 'Z']`)으로 하여서 시뮬레이션을 해보자.

: 이 경우에는 noise를 0.2정도로 주어도 잘 찾아가는 것을 확인할 수 있다. 결과는 다음과 같다.

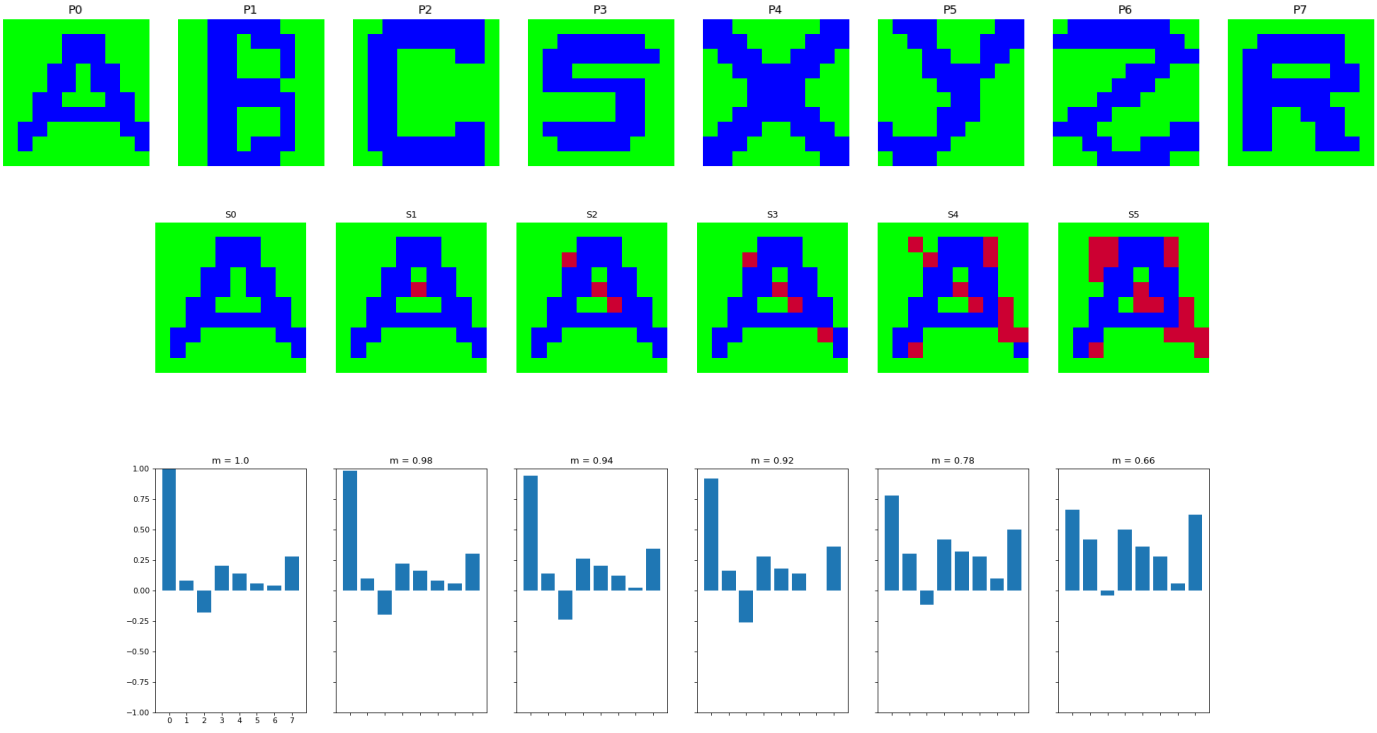




7.5.2

letter list에 'R'을 추가해 보고, 'A'가 여전히 fixed point인지 확인해 보자. (이 때 'R'은 'A'와 유사하게 생겼기 때문에 'A'를 찾는데 영향을 끼칠 것이다.

: fixed point인지 아닌지 확인하기 위해서 noise 0.0으로 주고 [A, B, C, S, X, Y, Z, R] list에 대해서 시뮬레이션 해보았다. R이 있을 때는 A가 fixed point가 되지 않음을 확인할 수 있다. 위에서 언급했듯이 R prototype이 A가 유사하게 생겼기 때문에 이가 영향을 끼치는 것이다.



7.5.3

small set을 만들어보고, 모든 letter가 fixed point인지 확인해 보자. 그리고 Capacity와의 불일치에 대해서 설명해 보라.

: 먼저 [A, B, C, S, X] list에 대하여 확인해 보았다. 이 경우에는 5개의 letter가 모두 fixed point가 되는 것을 확인할 수 있었다. 다음으로는 이 list에 'Y' letter를 추가해서 확인해 보았다. 이 경우도 마찬가지로 6개의 letter가 모두 fixed point가 되는 것을 확인할 수 있었다. 이후 list에 'Z' letter를 추가해서 확인해 보았다. 이 경우에서는 C와 B가 fixed point되지 않은 것을 확인할 수 있었다. 이는 밑 결과에서 확인할 수 있다. **7.4.1**에 따르면 10개(10X10이므로)의 letter까지는 커버할 수 있어야 하는데, 그러지 못하는 것을 확인할 수 있다. 이는 알파벳을 prototype으로 설정하면 랜덤으로 prototype을 생성할 때보다 서로의 유사성이 올라가기 때문일 것이라 예측할 수 있다.