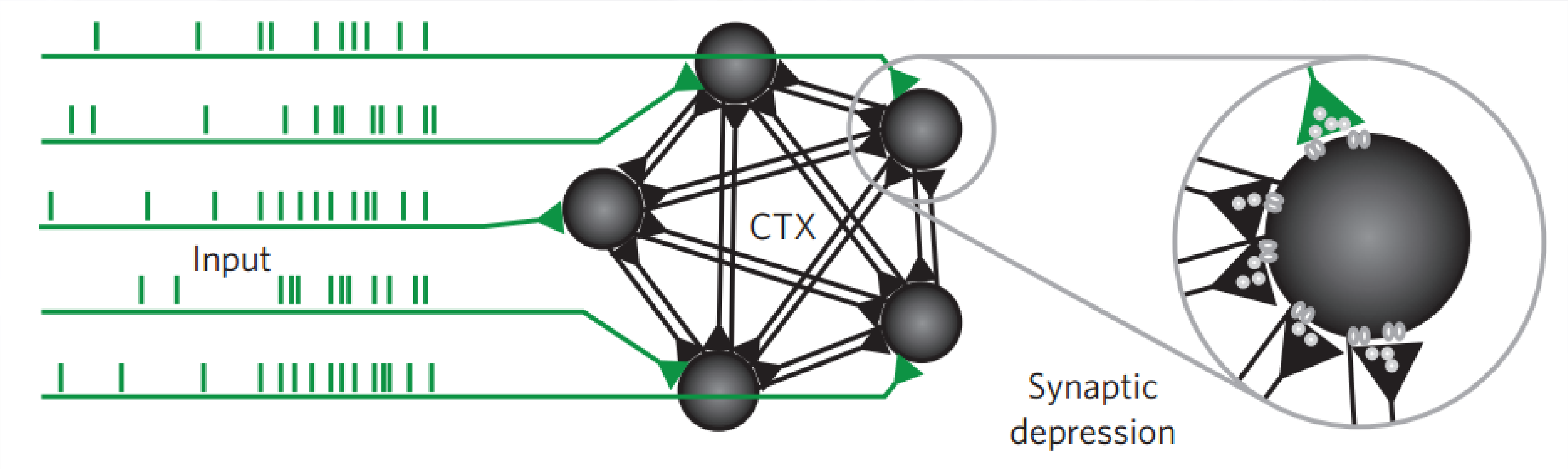
自组织临界性

• 小组成员： 杨楚雪　田伟柏　王誉晨　谢尚恩

1 模型设定

RESEARCH BACKGROUNDS



动力学方程

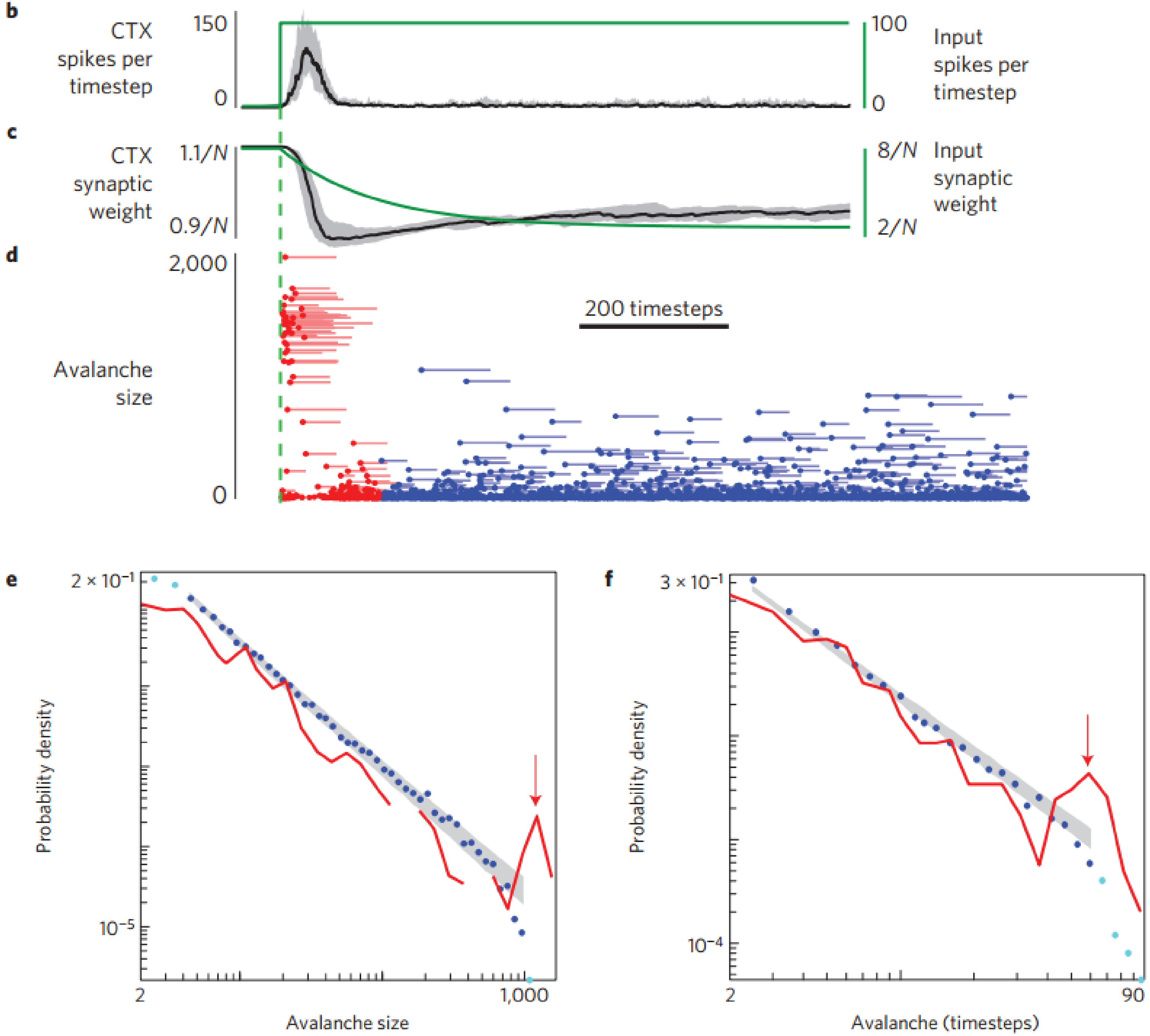
$$p(t+1)=\Omega_i(t)\sigma_i(t)+\sum_{j=1}^{N}W_{ij}(t)s_j(t)$$

$$W_{ij}(t+1)=W_{ij}(t)+\frac{1}{\tau_r}\left(W_{ij}^{o}-W_{ij}(t)\right)-\frac{1}{\tau_d}W_{ij}(t)s_j(t)$$

$$\Omega_i(t+1)=\Omega_i(t)+\frac{1}{\tau_r}\left(\Omega_i^{o}-\Omega_i(t)\right)-\frac{1}{\tau_d}\Omega_i(t)\sigma_i(t)$$

参数设置

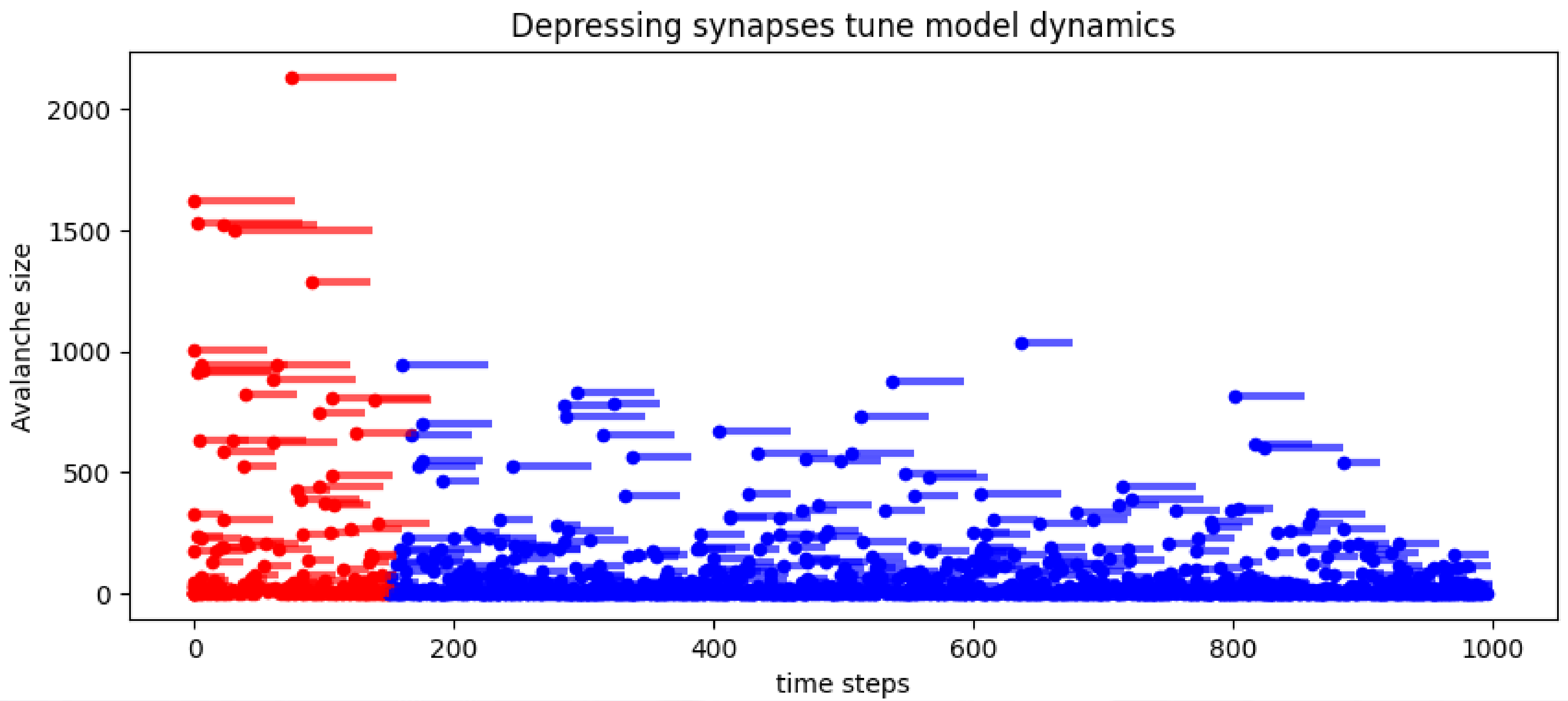
```
N = 1000
inhibitory_rate = 0.2
time_step = 900
pre_time = 20
r_pre = 0.00005
r_sta = 0.1
tau_r = 400
tau_d = 20
eva_max = 1.5
```


b
CTX spikes per timestep
150
0
100
0
Input spikes per timestep
c
CTX synaptic weight
1.1/N
0.9/N
8/N
2/N
Input synaptic weight
d
2,000
Avalanche size
0
200 timesteps
e
2 × 10⁻¹
10⁻⁵
Probability density
2
1,000
Avalanche size
f
3 × 10⁻¹
10⁻⁴
Probability density
2
90
Avalanche (timesteps)

2 实验结果

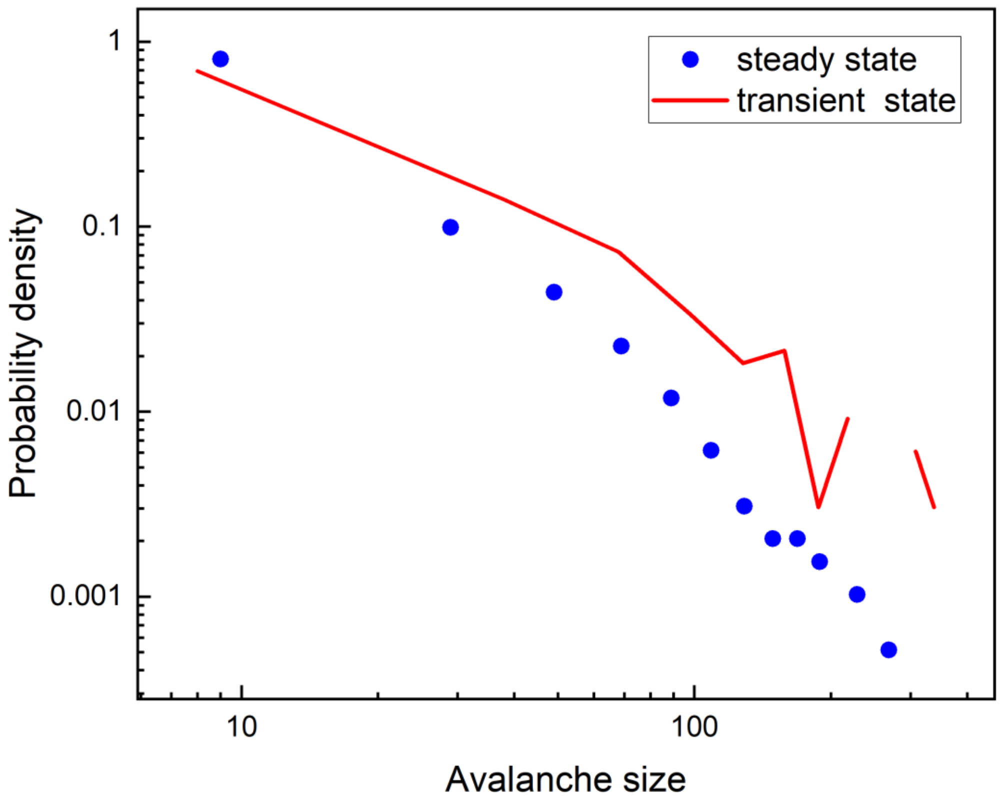
RESEARCH BACKGROUNDS

雪崩示意图W=1.5

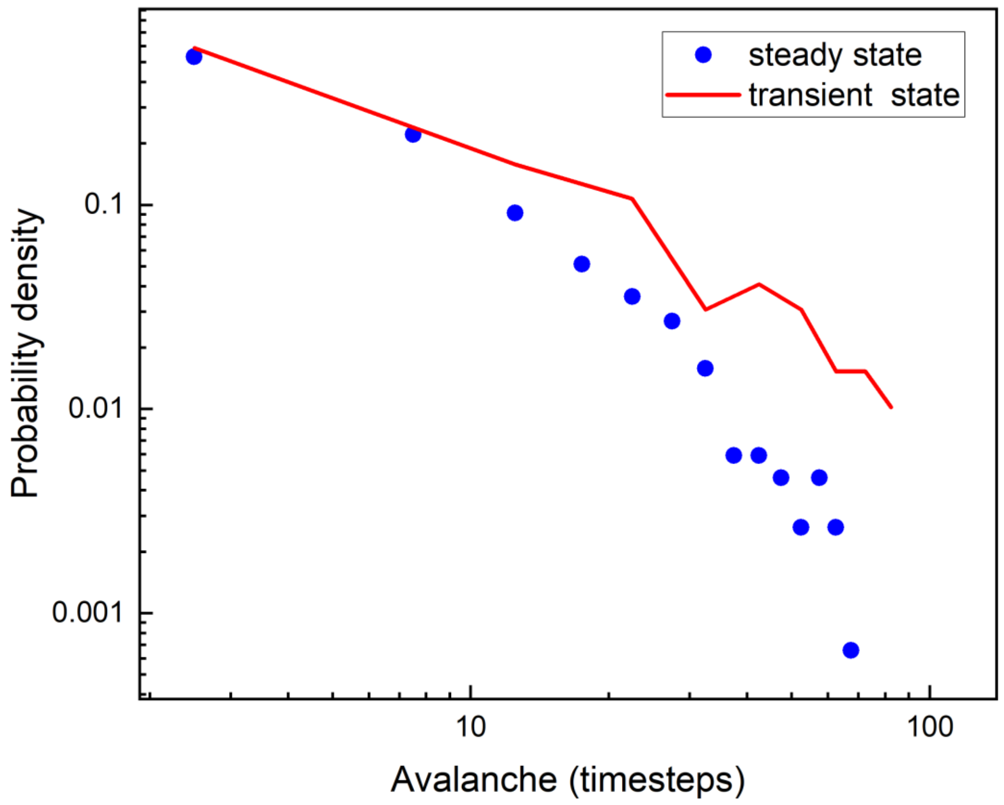


Depressing synapses tune model dynamics
Avalanche size
2000
1500
1000
500
0
0
200
400
600
800
1000
time steps

幂律W=1.5



steady state
transient state
Probability density
1
0.1
0.01
0.001
10
100
Avalanche size



steady state
transient state
Probability density
0.1
0.01
0.001
10
100
Avalanche (timesteps)

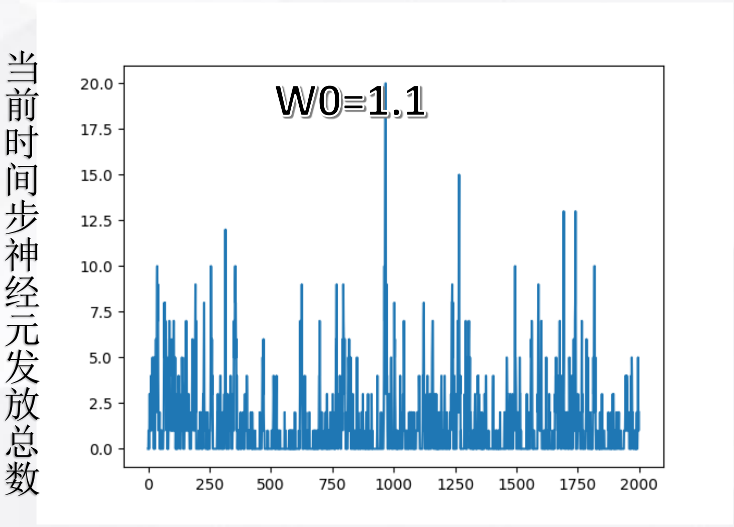
3

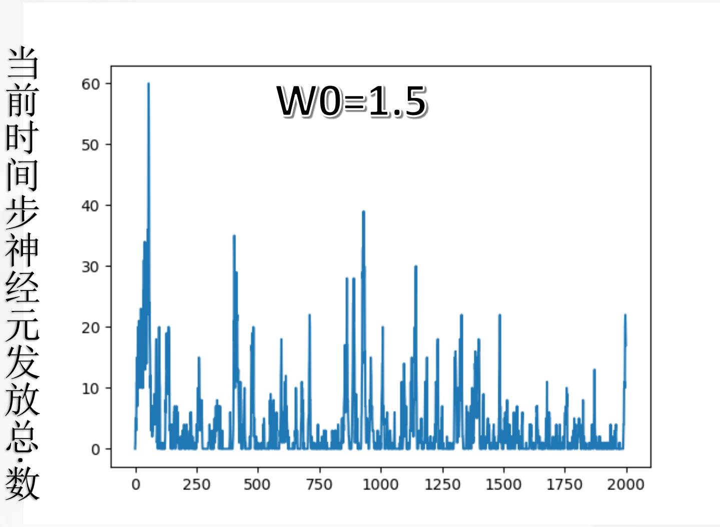
参数讨论

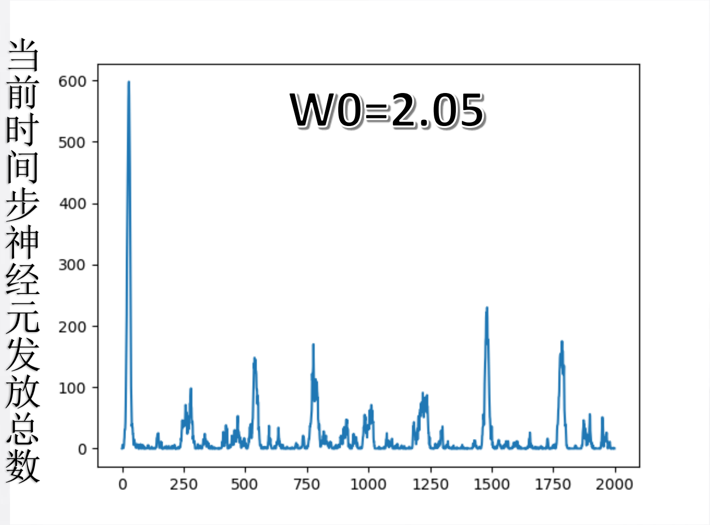
RESEARCH BACKGROUNDS

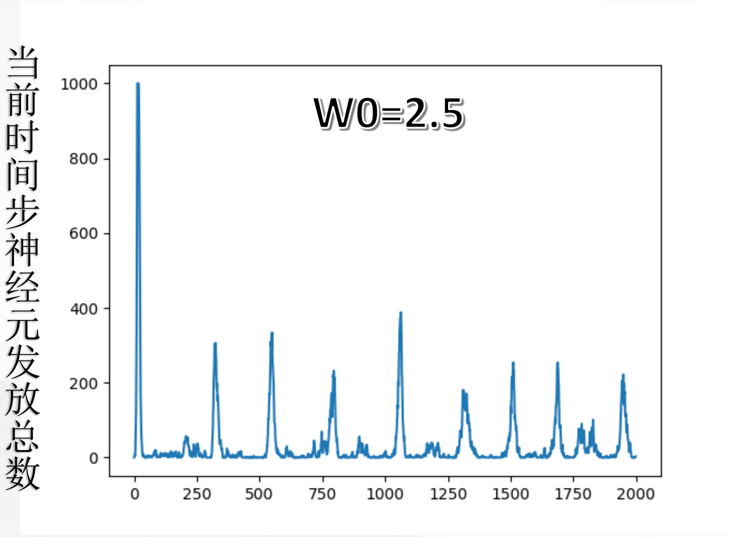
权重初始化
最大特征值的影响

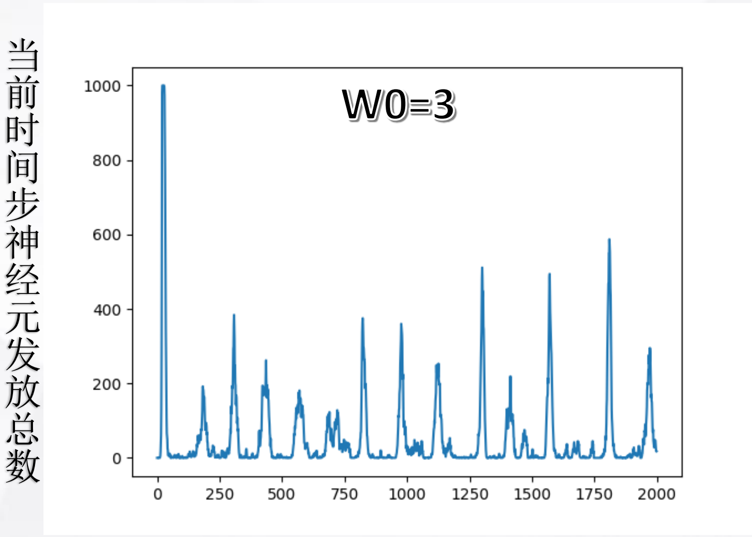
神经元激活的时域图





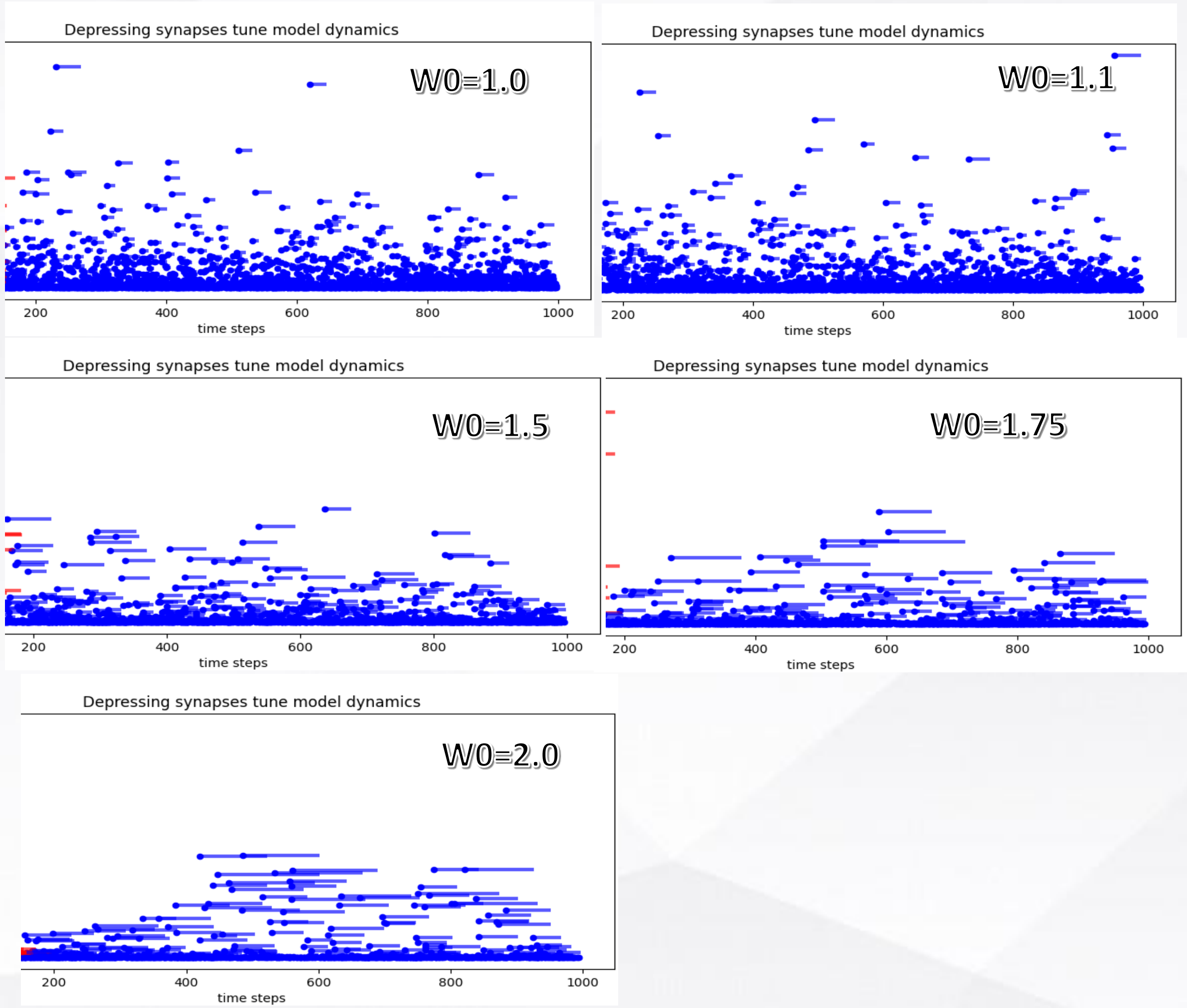






稳态响应

纵坐标：一次雪崩从开始到结束所有神经元累计发放次数



横着的尾巴：一次雪崩从开始到结束持续时间

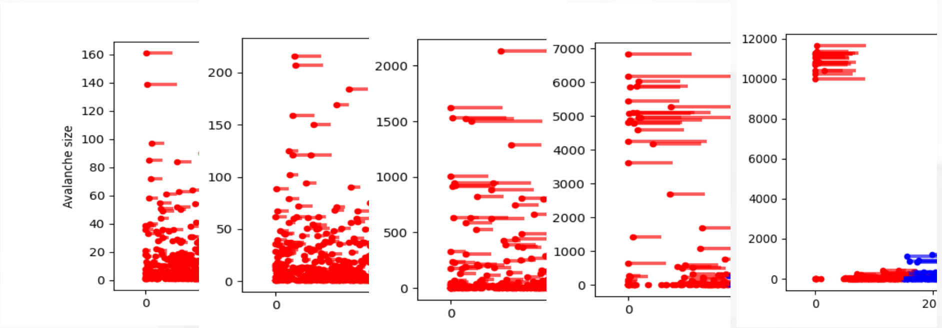
权重初始化
最大特征值的影响

神经元激活的示意图

示意图提供的信息量

- 收集了峰面积，峰宽信息
- 研究其统计性质
- 观察发现瞬态稳态的区别

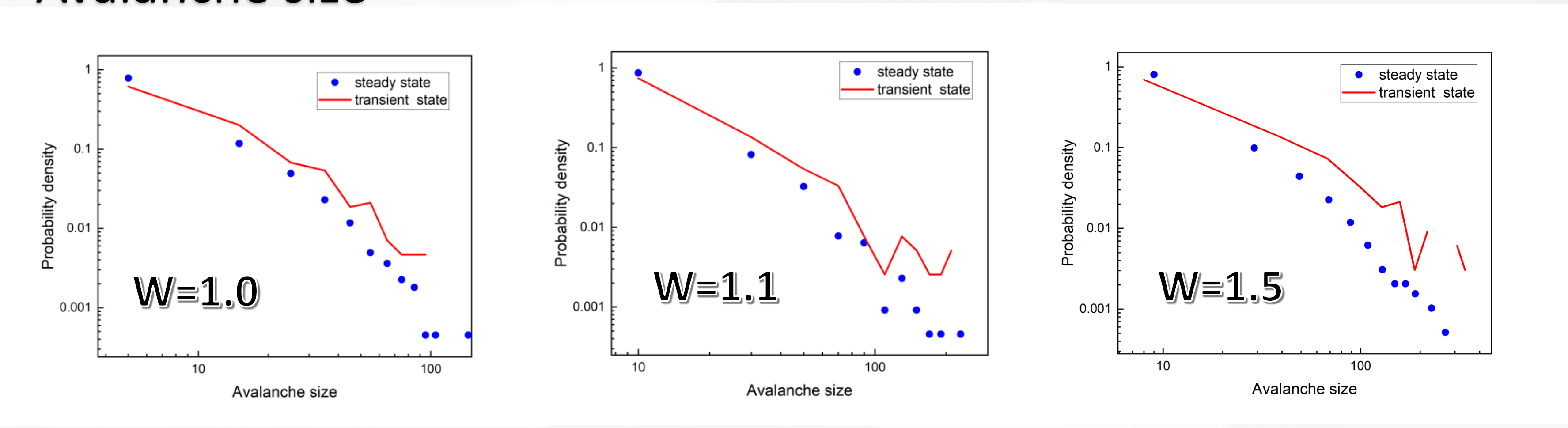
瞬态响应



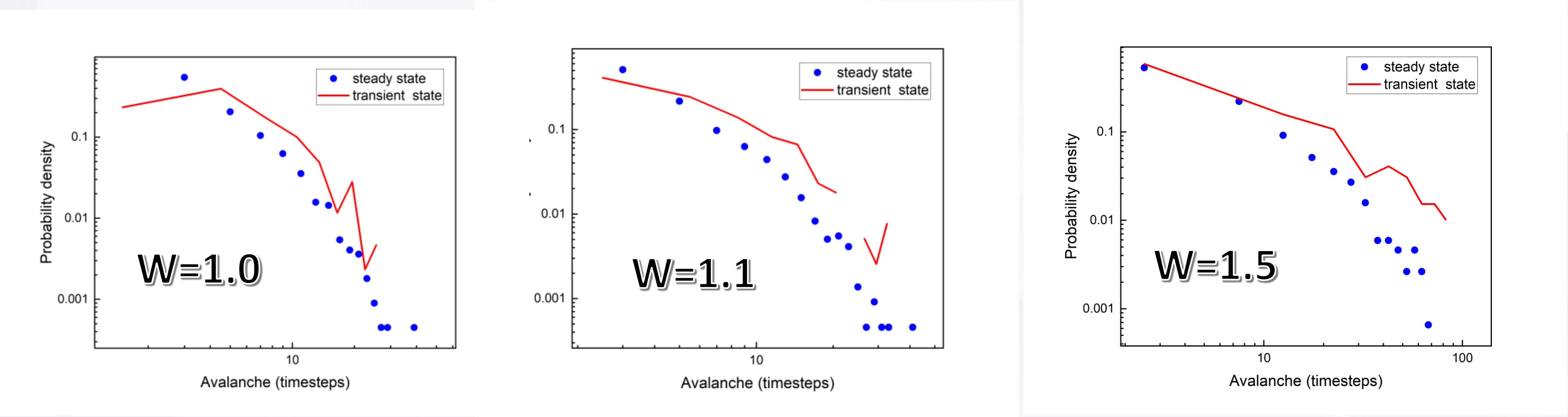
W0=1.0 W0=1.1 W0=1.5 W0=1.75 W0=2.0

神经元雪崩幂律图

Avalanche size



Avalanche duration



幂律图做法

- 频率直方图采用了双对数坐标

幂律图寓意

- 揭示了雪崩规模越大出现频率越低
- 神经元不应期不满足这样的规律
- 在瞬态向稳态过渡时期出现了自组织临界性

参数影响

- 改变了权重初始化最大特征值这个参数，发现便没有影响幂律的结论

技术细节

W初始化时

20%的抑制和80%的正常神经元不需要打乱顺序

编程时

注意雪崩size的定义

一个好的peak数组： 峰面积peak[1][1]

峰宽peak[1][2]

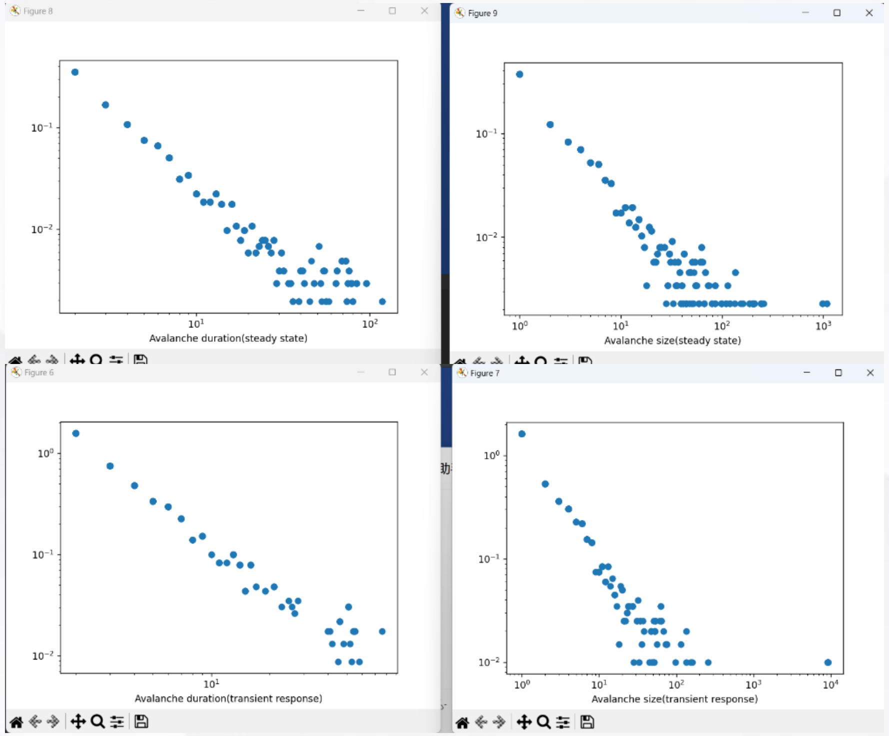
峰起点peak[1][3]

统计时

进行20次独立重复实验不然画不出雪崩示意图

统计时

要注意频率直方图的bin



4

附录代码

RESEARCH BACKGROUNDS

```python
import random
import numpy as np
import matplotlib.pyplot as plt

def img_probe(array,rank):
    import matplotlib.pyplot as plt
    img = array

    plt.imshow(img, cmap='Greys')
    plt.title('{}'.format(rank))
    plt.show()

'''
系统参数设定
'''
N = 1000
inhibitory_rate = 0.2
time_step = 900
pre_time = 20
r_pre = 0.00005
r_sta = 0.1
tau_r = 400
tau_d = 20
eva_max = 1.5

'''
数组的建立
'''
p_list = []
p0 = np.zeros([N,1])
p_list.append(p0)

s_list = []
s0 = np.zeros([N,1])
s_list.append(s0)
```

```python
W_list = []
W0 = []
for j in range(N):
    if j < int(inhibitory_rate * N)+1:
        W0_i = -1 * np.random.rand(N)
    else:
        W0_i = np.random.rand(N)
    W0.append(W0_i)
random.shuffle(W0)
W0=np.array(W0)

eva = np.linalg.eigvals(W0)
W0 = W0*eva_max/max(abs(eva))
W_list.append(W0)


sigma_list=[]
for t in range(time_step):
    if t < pre_time:
        sigma = np.random.binomial(1, r_pre, N).reshape(N,1)
    else:
        sigma = np.random.binomial(1, r_sta, N).reshape(N,1)
    sigma_list.append(sigma)


Omega_list = []
Omega0 = 8/N * np.ones([N,1])
Omega_list.append(Omega0)


def s_activation(p):
    s=np.zeros([len(p),1])
    for i in range(len(p)):
        if p[i]>1:
            s[i]=1
        elif p[i]<0:
            s[i]=0
        else:
            s[i]=np.random.binomial(1, p[i], 1)
    return s
```

```python
'
"
动力学方程:同步触发
'''

# 初始化
p = p0
s = s0
Omega = Omega0
W = W0

# 备注：t==0,意味着p[1],s[1]
for t in range(time_step):
    # p更新
        p = Omega * sigma_list[t] + np.dot(W,s)
    p_list.append(p)

    # W更新
        W = W + (W0 - W) / tau_r - np.dot(W, s) / tau_d
    W_list.append(W)

    Omega = Omega + (Omega0 - Omega) / tau_r - Omega * sigma_list[t] / tau_d
    Omega_list.append(Omega)

    # s更新
        s = s_activation(p)
    s_list.append(s)


    # 时间步更新
        t+=1


p_density = np.average(p_list,axis=1)
s_density = np.average(s_list,axis=1)

# plt.plot(s_density)
# plt.title('{}'.format('s'))
# plt.show()

img_probe(s_list, 's')
```

```python
def generate_peak_array(data):
    peak = []
    end_index = []
    start_index = []

    for i in range(len(data)-1 -1):          # 后面的减1是为了防止数组指标溢出
            if data[i] == 0 and data[i + 1] != 0: # 当前值为0，下一个值不为0，表示一个峰的开始
                start_index.append(i)
    for j in range(len(data)-1 -1):          # 后面的减1是为了防止数组指标溢出
            if data[j] != 0 and data[j + 1] == 0: # 当前值不为0，下一个值为0，表示一个峰的结束
                 end_index.append(j+1)
    if data[len(data)-1] != 0:
        end_index.append(len(data)-1)

    if len(start_index) == len(end_index):
        print("peak search completed")

    for k in range(len(start_index)):
        peak_area = sum(data[start_index[k]:end_index[k]]) # 最后一个峰没发放完少一
            peak_width = end_index[k] - start_index[k] - 1
        peak_start = start_index[k]
        if peak_area != 0:
            peak.append([peak_area, peak_width, peak_start])
    return peak
```

```python
def plot_peak(peak):
    # 计算时间序列
      # time = [row[2] + row[1] for row in peak]
    # peak_area = [row[0] for row in peak]
    # plt.plot(time, peak_area)

    # 绘制图形

      plt.xlabel('time steps')
    plt.ylabel('Avalanche size')
    plt.title('Depressing synapses tune model dynamics')

    # 绘制峰面积保持一段时间的横线
      for i in range(len(peak)):
       x = peak[i][2]
       y = peak[i][0]
       w = peak[i][1]
       plt.hlines(y, x, x + w, color='r', linewidth=3, alpha=0.65)

    # 绘制散点
      cumulative_width = 0
    for i in range(len(peak)):
       x = peak[i][2]
       y = peak[i][0]
       plt.scatter(x, y, color='r', s=20)
       cumulative_width += peak[i][1]

    plt.show()
```

THANKS

感谢