Computational Cognitive Neuroscience Assignment, Hopfield Networks, Schizophrenia and the Izhikevich Neuron Model

Oliver Goldstein

February 12, 2019

2.1) (1) 2.1) (2) Three memories were chosen and were subsequently all successfully recovered with full recall (proportion of correctly recovered states). Figure 1 details the performance of the network.

2.1 (3) In choosing a pattern, there did not seem to be an a priori, well established link between the shape of the memory encoded and the capacity of the network. Therefore, without any guiding principle on how the structure relates to association capabilities, I chose to implement randomised memories, even though manually playing around with the parameters revealed that some shapes were more easily encoded than others. Details regarding the implementation are provided here for the sake of reproducability. The implementation has unnormalised weights to keep it inline with [1]. In addition, the weights W_{ii} were set to zero $\forall i$. All results for the Hopfield network are shown for an average of ten runs in order to offset the effect of randomisation during asynchronous state updates and memory generation. Figures [2] [4] [1], display the so called error percentage, which is the percentage of states that differ from the target memory. An error percentage around 50% means that the state probably has little to no correlation with the target memory. Error percentages are related to the Hamming distance through the network size, for instance, an error percentage of ten on a network of 100 units, means ten states were different than the target memory. The degradation procedure probabilistically flips state values, though if a state value is actually flipped, it is always changed to the opposite value than that of the corresponding target memory. This procedure means that any degradation beyond that of 50% brings the state of the network towards the mirror of the state of the target memory.

Figure 2 demonstrates that a (bipolar) network with N neurons can indeed store approximately $0.138N$ patterns perfectly [2] [3], given that the memory itself has not been degraded[1]. However, I found the number of memories that were successfully encoded to be highly variable and dependent on the structure of the memories and the update procedure. It is clear that as the level of degradation increases beyond twenty percent of the memory state, the performance drastically begins to reduce. A network with 100 units, that has 50% of its state degraded, can barely hold three or four memories reliably. The axis which

[1]The following link provides a good overview of the probabilistic guarantees of the capacity of a Hopfield Network https://www.doc.ic.ac.uk/ ae/papers/Hopfield-networks-15.pdf

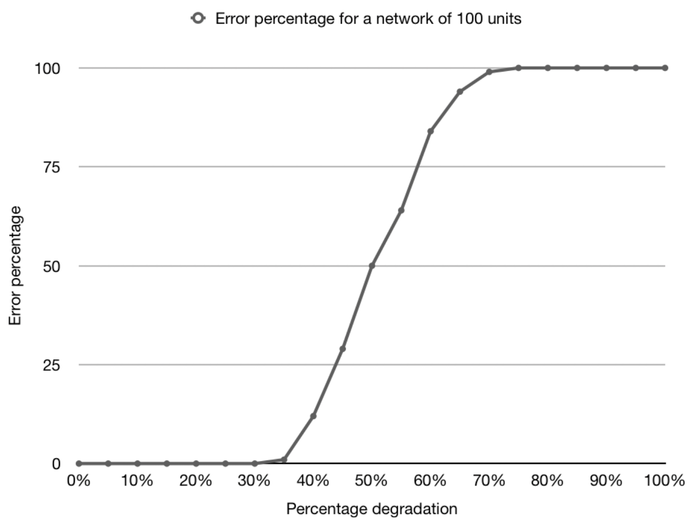

Figure 1: Error percentage for the first three memories, relative to the degradation of the memories. It is clear that beyond 50% degradation, the memory begins to correct to the mirror image of the network.

is varied around is the size of the network, which shows clearly that larger networks have more capacity, in a roughly linear pattern, which is demonstrated by 3. This is detailed in more detail by 4. These both show the capacity for zero degraded networks oscillates a little around 0.138*N, presumably dependent on the sources of randomness in the program.

2.2) (Check) To verify that stochastic update indeed works, Figure 2 shows a line for T=4 which roughly matches the deterministic update.

2.2) (1) A network of 100 units and 9 random memories is plotted for various pruning levels in 5. This shows that no amount of pruning is helpful in memory recovery.

2.2) (2) A computational theory which attempts to account for the qualia associated with mental health disorders (the hard problem of consciousness) is obviously beyond the scope of this question. The dynamics and thus the kind of computation which underlies schizophrenia is obviously a challenging theory to reconcile. Instead, I assume this question is asking whether some aspects of the largely empirically based computational and mathematical structure of a Hopfield network match some aspects of a mathematical structure placed over the qualitative nature of some of the symptoms of schizophrenia. One could then hypothesize that this computation is itself thought. In this case, the Hopfield network would have to, at the very least, account for delusions, hallucinations and Schneiderian symptoms. From this perspective, there are several interesting analogies to take note of. The first, is that there exist "parasitic foci", which are "localized areas of overpruned networks that tended to lock into specific non-memory activation patterns regardless of the flow pattern of surrounding neurons." These foci would, as the authors note, account for symptoms which are relatively independent of any external input information (ide fixe), such as hallucinations and delusions. These foci could then parasitically impact other configurations of the network in a self perpetuating cycle, resulting in delusional thought patterns. These obstructive dynamics, brought about by these foci, in the same vein of thinking, model intrusive thoughts and

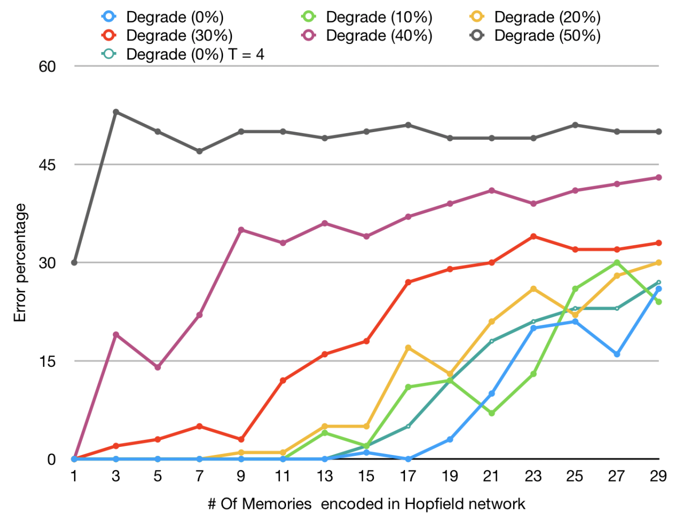

Figure 2: Error percentage versus number of memories encoded for various degradation levels of the memories.

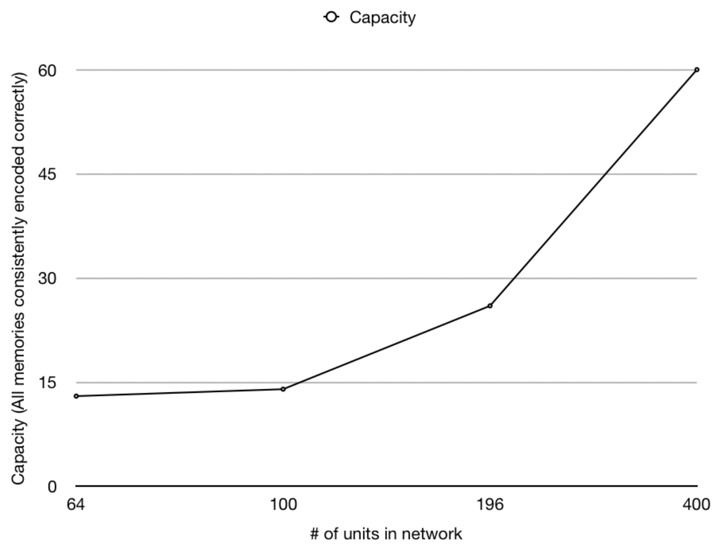

Figure 3: Maximum number of memories recalled with zero error (capacity), when zero degradation is applied.

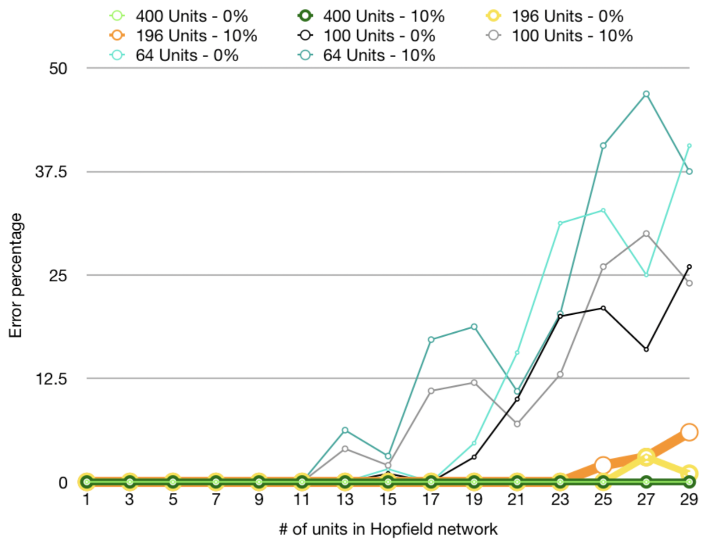

Figure 4: This demonstrates that as the size of the network increases, the number of memories that can be encoded increases.

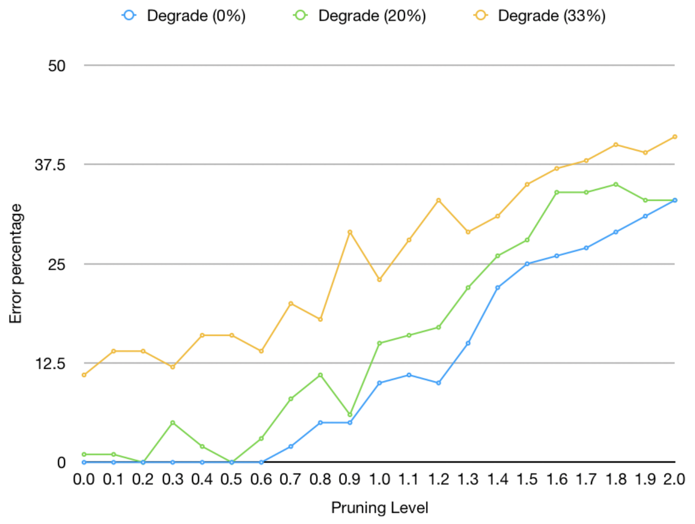

Figure 5: This demonstrates that an increase in the pruning level is associated with reduced network performance.

hallucinations, incoherent thought experiences and the dissolution of the feeling of free will (passivity experiences, loss of volition).

Overpruned networks produce fragments of different memories when presented with ambiguous input patterns. This is viewed, by the authors as a mapping from the network to the contamination Rorshach response. An extension of the theory is also provided to extend not just from the structure of experience to the network, but also from the network to the physical manifestation of the brain and a new theory underlying schizophrenia as the result of a failure to shut off pruning during adolescence. Overall, I feel that the model that the authors propose can account for the symptoms described, but only at one, very restricted and overly simplistic level of explanation. This is because the model lacks a deeper connection to the internal structure of some of the experiences themselves. For instance, delusions can be sub-divided into persecutory delusions, delusions of grandeur etc. It is as of yet unexplained as to how these kinds of thought fit into the framework provided. Another weakness of the explanation is that there seems to be a high degree of similarity between hallucinations, delusions and Schneiderian symptoms in the Hopfield network, which doesn't do justice to the dissimilarty of the qualitative experiences themselves, even from an empirical neuroscientific perspective (i.e. there is much more to say about the neuroscientific basis of visual hallucinations then a Hopfield network). The model also risks becoming too vague, potentially modelling other thought trains which are outside of the scope of schizophrenia, for instance OCD and ADHD.

2.2) (3) Pruning is obviously not the only way in which the Hopfield network can be degraded and have spurious minima introduced. One could simply randomise the weights sampled from a Gaussian or uniform distribution as an alternative strategy to degrade the network and introduce spurious minima. The temperature parameter (which creates a sample from a sigmoid function with scaled gradient) in some sense does this and modulating the parameter, led to a degradation in performance. The way in which factors such as dopamine modulation and stress can be accounted for largely rests on the work of [4]. Here, the authors claim that positive symptoms of schizophrenia are due to shallow basins of high firing rate attractor states. The authors claim this reduction is due to a reduction of GABA interneuron efficiency, which dictates how likely these attractor states result from the spontaneous attractor state. This theory is then "reversed" to explain negative symptoms, which is that the high firing rate attractor state is not itself deep enough. Seeman et al. [5] claims that life events or stressors are new inputs to the system, or a new memory so to speak and as the number of memories increases, the performance of the networks is more likely to degrade to spurious attractors. The paper even claims that stochastic updates using the temperature parameter above models dopaminergic activity. Following on from these ideas, it would be interesting to see if one could model D2 antagonists and D1 agonists using these temperature parameters. These could be modulated to vary how likely it is that the attractor states are entered. However, firing neurons would be needed to emulate the states from [4].

3) (1) Current injection applied was stochastic in one case and deterministic in the other. A current value of ten was applied in the deterministic case and ten multiplied by a standard normal distribution in the other. As the original paper had no units, I can only provide an arbitrary value of ten.

3) (2) I was able to replicate all modes except for the resonator mode. I suspect this was because the timing at which the bump in injected current applied was not perfected. I

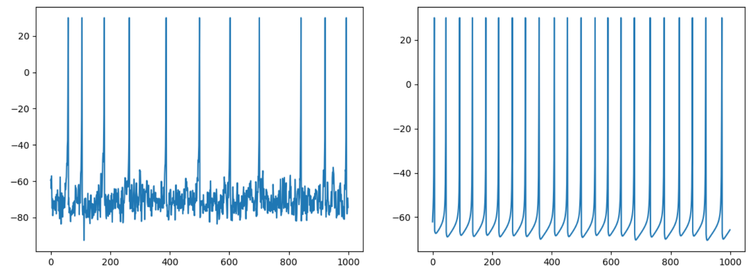

Figure 6: This demonstrates the response of the model to stochastic (left) and deterministic (right) current injections. Y axis is voltage (mv) and X axis is time.

did not optimise for this. I used different currents to try to achieve the effect including that specified in the paper, to no avail. The parameters are listed below.

4) (1) To visualise the dynamic modes, a raster plot was generated using MatLabs spy command. The modifiable parameters are w_e the excitatory weights, w_i the inhibitory weights, I_i the inhibitory input current and I_e the excitatory input current. Six new dynamic modes are presented at a population level, which I feel demonstrate a wide capability that the Izhikevitch model exhibits. One can observe fast spiking, regular spiking, inhibitory only activity, excitatory only activity, 5Hz oscillations and even activity with no particular pattern at all.

4 (2) Attractor dynamics can range in their expression, from points, lines, circles, plane, ring and even chaotic attractors. It is known that spiking neurons are involved in attractor dynamics and thus the Izhikevitch model could be used to implement these. These attractors should then be controllable such that different states can be moved between as per [6]. It is already clear, that cyclic attractors can be implemented as per Figure 8. However, it is also important to demonstrate the wide range of attractor networks that have been observed, for instance the ring model - a model that achieves contrast invariance, but also orientation selectivity matching empirical observations of neurons in the visual cortex. It would be useful to show that these effects can also occur with Izhikevitch neurons. A characteristic of the ring model is that activity persists even after the input current is removed, which means that the activity is self sustaining through recurrent connections. This framework is hypothesized to be a model of working memory. The neurons have a thalamic input and recurrent connections with excitatory connections between cells that are nearby and inhibitory connections with those that are further away. The ring model would exhibit the same firing pattern as shown in Figure. **??** from [6]. Implementing this same mechanism via modification of the S matrix from the Izhikevitch model yields the results as shown in Figure 9 whereby high current values were stimulated for half a second and during the second half of the second, the activity sustained itself.

4) (3) In order to model the effects of schizophrenia in this framework, it would be sufficient to modify the signal to noise ratio (by adding more noise to the current injection process) and modify the ratio of excitatory versus inhibitory neurons [7]. The ratio was changed to 1:1 rather than 4:1, resulting in attractors that were far more shallow, which emulates (slower) NMDA hypofunction and modulated GABA inhibition. This also made the attractors more unstable and spontaneous, especially in face of noise. By modifying the

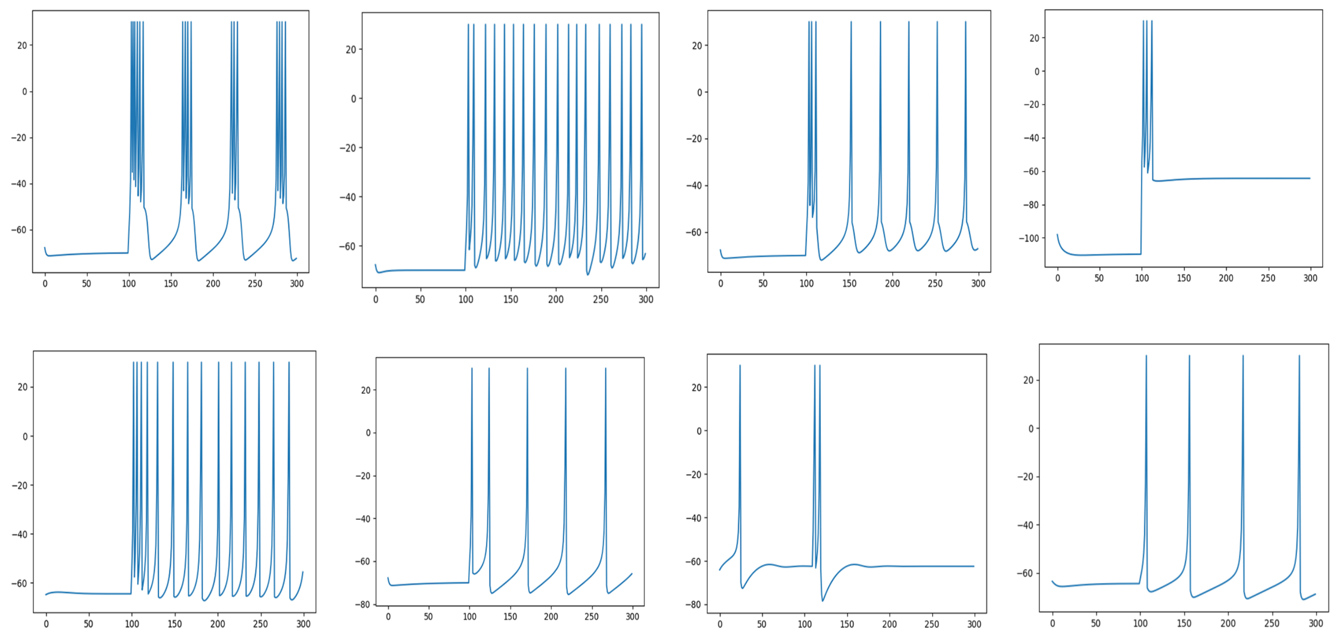

Figure 7: From top rightwards: chattering (CH), fast spiking (FS), instrinsic bursting (IB), Thalamo Cortical hyperpolarized (TH1). From bottom row rightwards: low spiking threshold (LTS), regular spiking (RS), resonator (RZ), Thalamo cortical depolarized (TH2). Y axis generally is from -80mv to 30mv (spike). The time simulated for is 300ms.

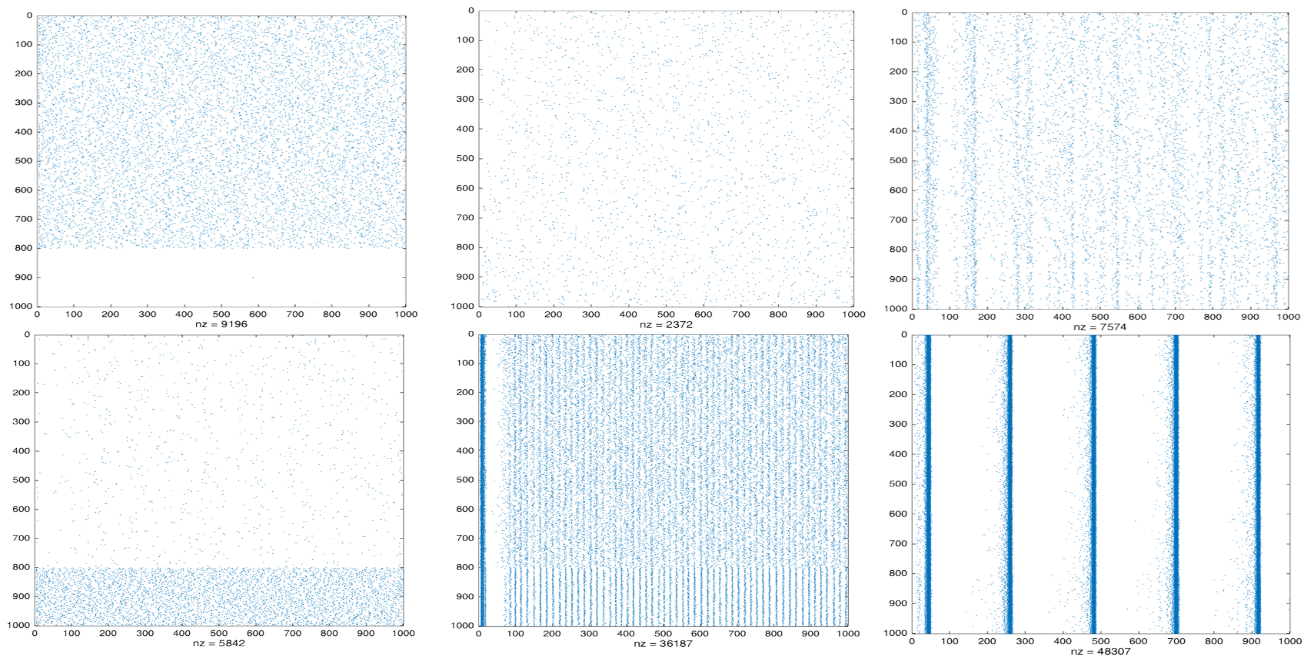

Figure 8: From top row rightwards: (w_e,w_i,I_e,I_w) (-0.5,1,10,-2), (0,0,4,2), (0.5,-1,5,2), From bottom row rightwards: (0.5,-1,5,10), (0.5,1,15,2), (0.6,-1,5,2)

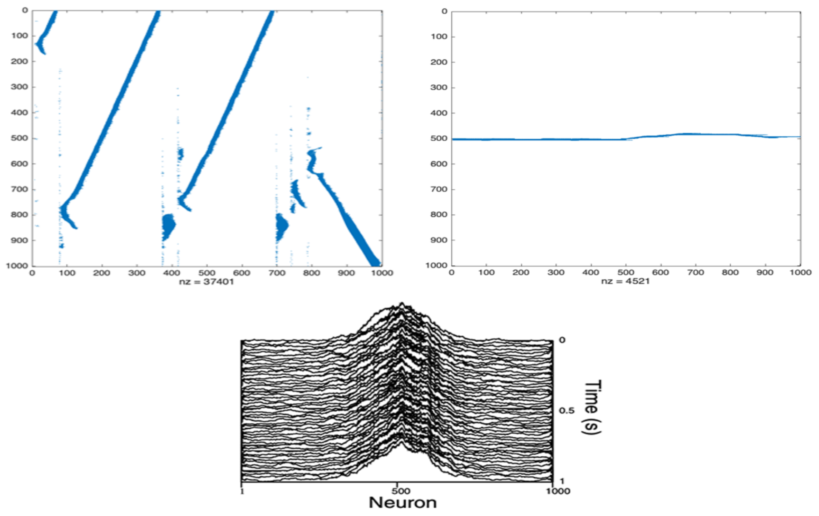

Figure 9: In the top left, the effect of increased noise and a 1:1 E/I balance. In the top right, a clear stable attractor pattern. The bottom shows a ring attractor from [6].

ratio of excitatory and inhibitory neurons and adding more noise into the process, the working memory ring like model implemented demonstrated strange behaviour. In fact various random attractors randomly formed and more than that the attractors moved on horizontally throughout neurons and sometimes randomly stopped. These may model continually changing trains of random thought. However, considering the vast literature on this topic and its complexity, I very much doubt whether this result is significant in any medical sense.

References

[1] Irwin Feinberg. Cortical pruning and the development of schizophrenia. 1990.

[2] ROBERTJ McEliece, EDWARDC Posner, EUGENER Rodemich, and SANTOSHS Venkatesh. The capacity of the hopfield associative memory. *IEEE transactions on Information Theory*, 33(4):461–482, 1987.

[3] John Hertz, Anders Krogh, and Richard G Palmer. *Introduction to the theory of neural computation.* Addison-Wesley/Addison Wesley Longman, 1991.

[4] Edmund T Rolls, Marco Loh, Gustavo Deco, and Georg Winterer. Computational models of schizophrenia and dopamine modulation in the prefrontal cortex. *Nature Reviews Neuroscience*, 9(9):696, 2008.

[5] Mary V Seeman. Neural networks and schizophrenia, 1994.

[6] Chris Eliasmith. A unified approach to building and controlling spiking attractor networks. *Neural computation*, 17(6):1276–1314, 2005.

[7] John W Olney and Nuri B Farber. Glutamate receptor dysfunction and schizophrenia. *Archives of general psychiatry*, 52(12):998–1007, 1995.

Appendix A Hopfield Network Code (Python)

```
from __future__ import print_function
import numpy as np
import math
from scipy.stats import bernoulli
import random
from scipy.spatial.distance import euclidean
import Again
import copy

size = 10

def generate_mem():
    return [[(−1 if random.randint(0,1) == 0 else 1) for i in range(0, size)] for j in range(0,size)]

def sign(value):
    if value > 0:
        return 1
    else:
        return −1

def print_matrix(matrix):
    for i in range(0, size):
        for j in range(0, size):
            string_state = ""
            if matrix[i][j] == 1:
                print(" 1", end = '')
            else:
                print(matrix[i][j], end = '')

        print("\n")

def hamming_distance(degraded_mem, original):
    dist = 0
    for row in range(0, len(degraded_mem[0])):
        for col in range(0, len(degraded_mem[1])):
            if degraded_mem[row][col] != original[row][col]:
                dist += 1
    return dist

def flip(memory, prob):
    bernoulli_var = bernoulli.rvs(size=1,p=prob)[0]
    if bernoulli_var == 1:
        if memory == 1:
            return −1
        else:
            return 1
    else:
        return memory

def degrade_memory(memory, p):
    degraded = copy.deepcopy(memory)
    for row in range(0, len(memory)):
        for col in range(0, len(memory)):
            degraded[row][col] = flip(memory[row][col], p)
    return degraded

T = [[[[0 for x in range(0,size)]
      for y in range(0, size)]
      for i in range(0, size)]
      for j in range(0, size)]
# pruning_factor = 0.6
# degradation_factor = 0.0
for degradation_factor in [0, 0.2, 0.33]:
    for pruning_factor in [2.0]:
        for mems_used in range(9,10):
            memories = [generate_mem() for i in range(0, mems_used)]
            for x in range(0,size):
                for y in range(0, size):
                    for i in range(0, size):
                        for j in range(0, size):
                            T_xy_ij = 0
                            for mem in range(0, mems_used):
                                T_xy_ij += memories[mem][i][j] * memories[mem][x][y]

                            # T_xy_ij /= len(memories)

                            T[x][y][i][j] = T_xy_ij
                            T[i][j][x][y] = T_xy_ij

            # Set self weights to zero

            for x in range(0, size):
                for y in range(0, size):
                    for i in range(0, size):
                        for j in range(0, size):
                            if (x,y) == (i,j):
                                T[x][y][i][j] = 0
                                T[i][j][x][y] = 0

            for x in range(0, size):
                for y in range(0, size):
                    for i in range(0, size):
```

```
            for j in range(0, size):
                if abs(T[x][y][i][j]) < (pruning_factor * euclidean([x,y], [i,j])):
                    # print(”Pruned!”)
                    T[x][y][i][j] = 0
                    T[i][j][x][y] = 0

total = 0
total_runs = 10
# Now set the state and determine input
for runs in range(0, total_runs):
    for mem in range(0, mems_used):

        degraded = degrade_memory(memories[mem], degradation_factor)
        # print_matrix(degraded)
        # print_matrix(memories[mem])
        for time in range(0, 20):
            for x in np.random.permutation(size):
                for y in np.random.permutation(size):
                    E = 0
                    for i in range(0, size):
                        for j in range(0, size):
                            E += T[i][j][x][y] * degraded[i][j]

                    prob_activation = 1/(1 + math.exp(-E/4))
                    bernoulli_var = bernoulli.rvs(size=1,p=prob_activation)[0]
                    if (bernoulli_var == 1):
                        degraded[x][y] = 1
                    else:
                        degraded[x][y] = -1
                    # degraded[x][y] = sign(E)

        # print_matrix(degraded)
        # print(hamming_distance(degraded, memories[mem]))
        total += hamming_distance(degraded, memories[mem])
print(”degradation_factor:” + str(degradation_factor))
print(”pruning_factor:” + str(pruning_factor))
if total_runs * mems_used == 0:
    print(0)
else:
    total /= (total_runs * mems_used)
    print(total)
```

Appendix B Izhikevitch Code (Python)

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

```
weight_excitatory = 0.5*random.random()
weight_inhibitory = -1*random.random()

# # Default:
# a = 0.02
# b = 0.2
# c = -65
# d = 2
# v = -65
# u = b * v

# # Regular spiking
# a = 0.02
# b = 0.2
# c = -65
# d = 8
# v = -65
# u = b * v

# # Intrinsically bursting
# a = 0.02
# b = 0.2
# c = -55
# d = 4
# v = -65
# u = b * v

# # Chattering
# a = 0.02
# b = 0.2
# c = -50
# d = 2
# v = -65
# u = b * v

# # Fast spiking
# a = 0.1
# b = 0.2
# c = -65
# d = 2
# v = -65
# u = b * v

# Low spiking threshold
# a = 0.02
# b = 0.25
# c = -65
# d = 2
# v = −65
# u = b ∗ v

# Thalamo cortical right
# a = 0.02
# b = 0.25
# c = −65
# d = 2
# v = −87
# u = b ∗ v
# I = [−100 for i in range(0, 100)]
# I = I + [0 for i in range(0, 200)]

# Thalamo cortical left
# a = 0.02
# b = 0.25
# c = −65
# d = 2
# v = −63
# u = b ∗ v
# I = [0 for i in range(0, 100)]
# I = I + [2 for i in range(0, 200)]

# # Resonator
# a = 0.1
# b = 0.26
# c = −60
# d = −1
# v = −60
# u = b ∗ v
# I = [0 for i in range(0, 10)]
# I = I + [5 for i in range(0, 100)]
# I = I + [10 for i in range(0, 10)]
# I = I + [5 for i in range(0, 180)]

# I = [0 for i in range(0, 100)]
# I = I + [10 for i in range(0, 200)]
time = []
voltage = []
for t in range(0, 300):

    time.append(t)
    v = v+0.5∗(0.04 ∗ (v ∗∗ 2) + 5 ∗ v + 140 − u + I[t])
    v = v+0.5∗(0.04 ∗ (v ∗∗ 2) + 5 ∗ v + 140 − u + I[t])
    u = u + a ∗ (b ∗ v − u)
    if v >= 30e-3:
        print(v)
        print("Fired")
        v = c
        u = u + d
        voltage.append(30)
    else:
        voltage.append(v)

plt.plot(time, voltage)
plt.show()
```

Appendix C   Izhikevitch Code Population (MatLab)

```
% Created by Eugene M. Izhikevich, February 25, 2003
% Excitatory neurons Inhibitory neurons
Ne=800;
Ni=200;

w_e = 0.5; % 0.5, 0.4, 0.6, 0.55, 1.5
w_i = -1.0;
I_e = 5; %5 8
I_i = 2;

re=rand(Ne,1); ri=rand(Ni,1);
a=[0.02*ones(Ne,1); 0.02+0.08*ri];
b=[0.2*ones(Ne,1); 0.25-0.05*ri];
c=[-65+15*re.^2; -65*ones(Ni,1)];
d=[8-6*re.^2; 2*ones(Ni,1)];
S=[w_e*rand(Ne+Ni,Ne), w_i*rand(Ne+Ni,Ni)];
v=-65*ones(Ne+Ni,1); % Initial values of v
u=b.*v; % Initial values of u
firings=[]; % spike timings
S = preprocess_weights(S);
neuron_count = Ne + Ni;
time_steps = 1000;
raster_plot = zeros(neuron_count, time_steps);

for t=1:time_steps % simulation of 1000 ms
I=[I_e*randn(Ne,1);I_i*randn(Ni,1)]; % thalamic input
if t < 100
for j=1:100
I(j) = 5;
end
end
fired=find(v>=30); % indices of spikes
```

```
firings=[firings; t+0*fired,fired];
raster_plot(fired,t) = 1;
v(fired)=c(fired); % Reset the voltage of the neurons that fire to c the reset voltage.
u(fired)=u(fired)+d(fired); % U also changes
I=I+sum(S(:,fired),2);
v=v+0.5*(0.04*v.^2+5*v+140-u+I); % step 0.5 ms
v=v+0.5*(0.04*v.^2+5*v+140-u+I); % for numerical
u=u+a.*(b.*v-u); % stability
end;
plot(firings(:,1),firings(:,2));

time = zeros(1, time_steps);
for x=1:time_steps
time(1, x) = x;
end

spy(raster_plot);
```

Appendix D Weight Preprocessing (MatLab)

```
function [S] = preprocess_weights(S)
for x=1:size(S,1)
    for y=1:size(S,1)
        prop = 0;
        if (x-y == 0)
            prop = 229.5;
        else
            prop = 27.5/(norm(x-y));
        end
        S(x,y) = prop - 12;
    end
end
```