Further Exploration into the Formation and Maintenance of Neuronal Assemblies through Synaptic Plasticity

Si Jia Li, Sanjeev Janarthanan, and Hiro Saito

December 10, 2019

1 Introduction

Memory formation is still a big mystery to us, but we have figured out that one of the mechanisms behind it is called spike timing dependent plasticity, or STDP. This idea is that changes in synaptic strength are related to spike timings, where if spontaneous spikes occur before the output of a neuron, the synapse gets stronger, but if it occurs after the output then the synapse gets weaker. These are timing dependent because the closer these spikes are to the output, the greater the change in synapse strength.

For our project we looked at "Formation and maintenance of neuronal assemblies through synaptic plasticity" by Ashok Litwin-Kumar and Brent Doiron, where they investigate how memories are created and maintained. The paper we studied demonstrated that stimulus of neural networks leads to interconnected ensembles of neurons. In addition, they showed how spontaneous dynamics actually reflect the previously experienced stimuli. Also, they showed that surface level variability is enough for spontaneous assembly reactivation, which reinforces the stability of these groups of neurons. This shows how it is the interactions between spontaneous dynamics and STDP that helps stabilize

memories.

Using the methods used in the paper, we reproduced their results, and found some of our own results.We wanted to see what effect stimulation frequency, population sizes, and learning rates had on synaptic strength when training new models. We also wanted to see how changing the learning rates affected already trained models, as the authors of the paper provided us with an already trained model. Finally, we investigated applying dimensional reduction to our model using PCA. This form of dimensional reduction is useful for reducing the complexity of neural networks and deducing which components are most significant [1]. Williamson et. al use dimensionality reduction in the hopes of figuring out how neuronal networks give rise to brain function, which is what we hope to do in order to build on the work of Litwin-Kumar et. al.

2 Equations

The paper used equations to model both membrane potential dynamics and the dynamics of synaptic conductance's. In the following section, we will describe these equations and define all the variables.

2.1 Membrane Potential Dynamics

$$\begin{aligned}\frac{d}{dt}V_i^X(t) = &\frac{1}{\tau^X}(E_L^X - V_i^X(t)) + \Delta_T^X exp(\frac{V_i^X(t) - V_{T,i}^X(t)}{\Delta_T^X})) \\ &+ \frac{g_i^{XE}(t)}{C}((E^E - V_i^X(t)) + \frac{g^{XI}}{C}(E^I - V_i^X(t)) \\ &- \frac{w_i^X(t)}{C}\end{aligned} \tag{1}$$

This equation governs the membrane voltage dynamics, and would trigger a spike if the voltage reached a certain threshold. This complicated equation is broken down into a few separate parts that all add together. V_i^X is referring to the voltage of neuron i in population X. τ^X is the resting membrane time constant of the population. E_L^X is the

resting potential of excitatory neurons in the population. $\Delta_X L$ is the EIF slope factor of the excitatory neurons, which stands for Exponential integrate-and-fire. This means that they neurons adapt their current and thresholds to the current situation. This is in contrast to inhibitory neurons, which are modelled as simple non-adapting integrate-and-fire neurons. As the thresholds can change, $V_{T,i}^X$ is the threshold of the neuron at the moment. g_i^{XE} is the conductance of the excitatory neuron i in population x, while g^{XI} denotes the same thing but for inhibitory neurons. C stands for the capacitance of the neurons, while w_i^X is the current through the neuron.

$$\frac{d}{dt}V_{T,i}^E(t) = \frac{1}{\tau_T}(V_T - V_{T,i}^E(t)) \tag{2}$$

This equation governs the voltage threshold of the neurons. The only new variable here is τ_T, which is the adaptive threshold time scale. For excitatory neurons, after the spike the voltage is reset, and unable to change for a certain amount of time, which is the refractory period of the neuron. After spiking, for excitatory neurons have the threshold reset to the current potential plus an additional amount A_T, which is defined in the code. For inhibitory neurons, the threshold does not change as mentioned with the earlier equation.

$$\frac{d}{dt}w_i^E(t) = \frac{1}{\tau_w}(a_w(V_i^E(t) - E_L^E) - w_i^E(t)) \tag{3}$$

This equation describes how the current in excitatory neurons change, where τ_w is the spike triggered adaptation time scale, and a_w is the subthreshold adaptation.

$$g_i^{XY}(t) = F^Y(t) * (J_{ext}^{XY} s_{ext}^{XY}(t) + \Sigma j J_{ij}^{XY} s_j^Y(t)) \tag{4}$$

In the equation for the conductance, $F^Y(t)$ is the synaptic kernel for input from population Y. In this case, $*$ represents convolution instead of multiplication. J is the synaptic strength, either of this neuron or external neurons, for population Y to population X. s is input in the form of spike trains. A spike train here is an independent homogeneous

Poisson process with a rate defined in the code.

2.2 Synapse Dynamics

2.2.1 Excitatory Neurons

$$\begin{aligned}\frac{d}{dt}J_{ij}^{EE}(t) = &- A_{LTD}s_j^E(t)R(u_i^E(t) - \theta_{LTD}) \\ &+ A_{LTP}x_j^E(t)R(V_i^E(t) - \theta_{LTP})R(v_i^E(t) - \theta_{LTD})\end{aligned} \tag{5}$$

This equation defines how the synaptic strengths change. Overall, the synaptic strength is bound between a min and max, written in the code. A_{LTD} is the factor which controls long term depression, while A_{LTP} is the factor that governs long term potentiation. R is just a linear-rectifying function, keeping the LTD fom increasing synaptic strength and vice versa for LTP. θ_{LTD} is the threshold value for LTD or LTP depending on the subscript. $u_i^E(t)$ and $v_i^E(t)$ represent the membrane voltage with low-pass filtered with time constants. In this equation, x_j^E represents the spike train s_j^E with a low pass filtered time constant as well.

2.2.2 Inhibitory Neurons

$$J_{ij}^{EI} \longleftarrow J_{ij}^{EI} + \eta(y_i^E(t) - 2r_0\tau_y) \tag{6}$$

$$J_{ij}^{EI} \longleftarrow J_{ij}^{EI} + \eta y_j^I(t) \tag{7}$$

These two equations overall govern the inhibitory STDP rules. η is the synaptic plasticity learning rate, controlling how much STDP affect the inhibitory neurons. Equation (6) is implemented if the presynaptic fired, and equation (7) is used if the postsynaptic neuron is fired. r_0 is the rate at which the inhibitory plasticity tries to balance the postsynaptic neurons, and τ_y is the time constant for low-pass filtered spike trains. This time constant is also used in y_i^X, which represents the spike train s_i^X utilizing τ_y.

In the paper, the learning rates A_{LTD}, A_{LTP} and η have been set to fixed values throughout the simulation. One of the objectives of this project is to examine how varying these parameters affects the stability and dynamics of pre-trained and new neuron ensembles.

3 Reproduction of Results

A key takeaway from this paper is that the spontaneous dynamics of a trained network reflect past stimuli. As such, we chose to reproduce a figure that represents this conclusion. Figure 1 contains two raster plots, the one on the left represents an untrained network and the one on the right a trained network. One can see that the spontaneous dynamics of these plots vary greatly. The untrained network has random spontaneous dynamics while the trained network has random activity which mirrors the evoked stimuli. The lines of neuronal firing which are grouped into bands are a result of the stimulus.

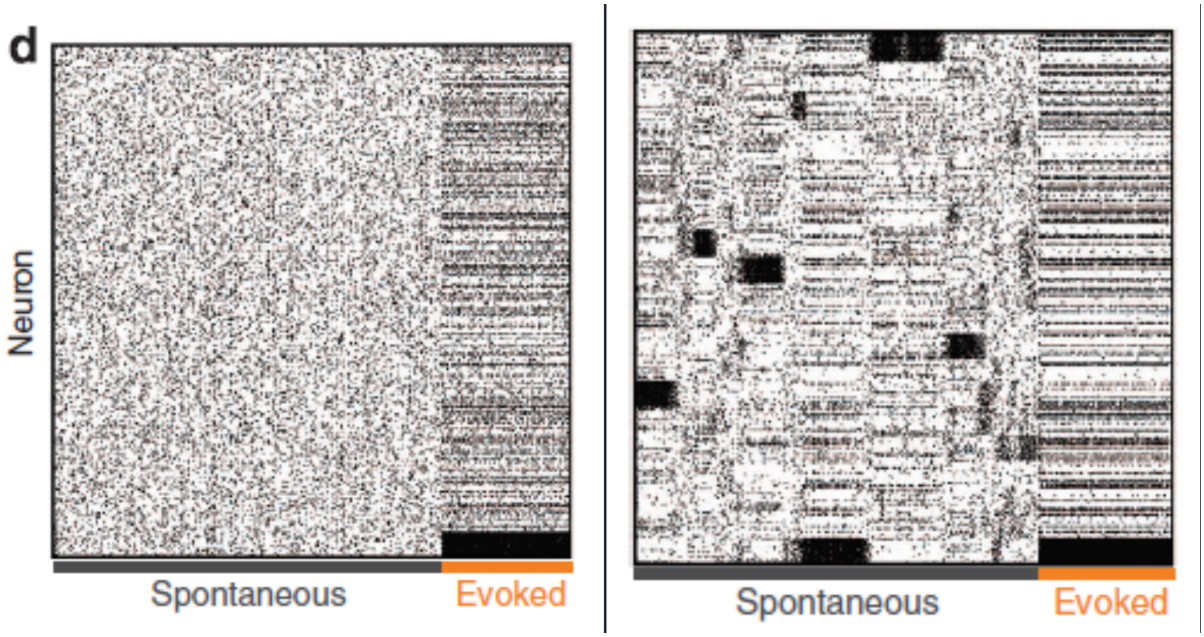

Figure 1: Left: A raster plot of an untrained network from the paper. Each row of a raster plot represents a neuron and adjacent rows are neighboring neurons. Each black dot represents when a neuron fires. Right: A raster of plot of a trained network from the paper. The spontaneous dynamics of this network reflected the evoked stimuli. This is seen in the bands of neurons firing together as well as clustering. Adapted from original paper

To reproduce these results, we used the source code provided by the authors to first create an untrained network. Then, we used the parameters for a trained network, provided by the authors, to create a trained network. We then compared these two raster plots to see if the trained plot contained patterns similar to evoked stimuli. As we expected, Figure 2 shows clusters of neuronal firing that appear as a result of a stimulus being applied to an untrained network. Our findings corroborate a major finding from the paper that the spontaneous dynamics of a trained network are similar in nature to the stimulus applied to the network.

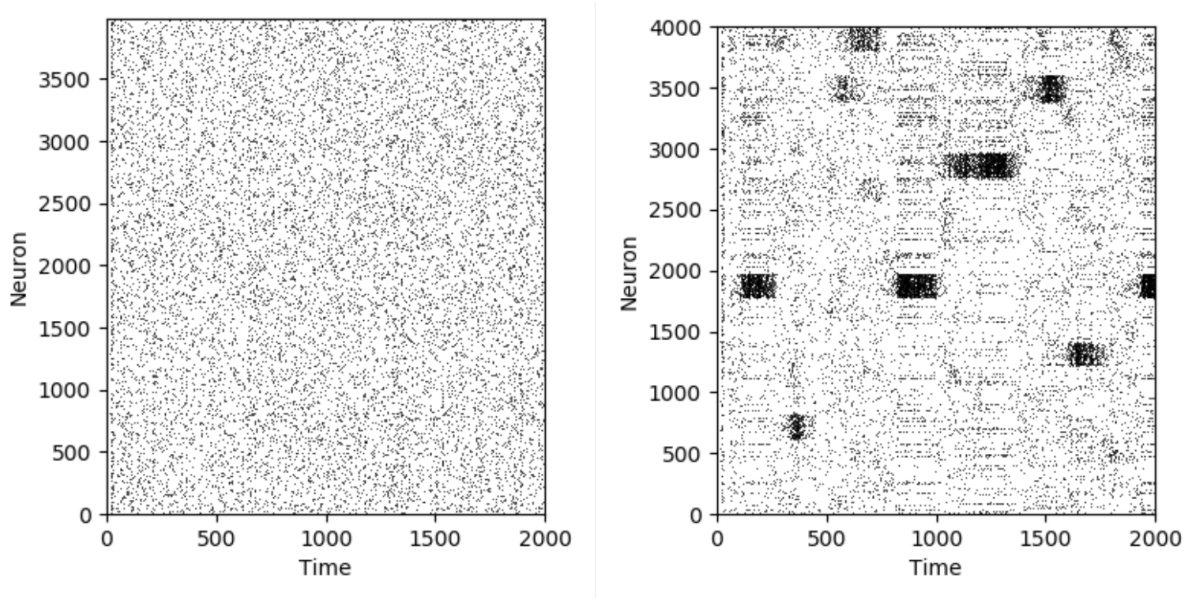

Figure 2: Left: A raster plot of an untrained network that we produced.The spontaneous dynamics of this network are random. Right: A raster of plot of a trained network using parameters provided by the authors. The spontaneous dynamics of this network reflected the evoked stimuli. This can be seen in clusters of neurons firing (black rectangles).

4 Results

4.1 Stability of the Learned Neuronal Assemblies

4.1.1 Stimulation Protocol

Once we investigated how different factors influence the formation of neuronal assemblies. We asked how the stability of the learned structure could be affected by the learning rates in equations 5 and 6.

To address this question, we used the trained assembly that had been provided by authors of the paper, Litwin-Kumar et al. We use their their original trained to make our comparisons independent of the initial trained assemblies. This model had 4000 excitatory neurons and 1000 inhibitory neurons. The excitatory neurons were randomly selected into 20 assemblies and trained to their maximum connection weights.

To disrupt the stability of this trained architecture, we randomly defined a second set of populations. We then stimulated the first assembly within this set of newly defined populations. The simulation was on for 5 seconds within a total of 10 seconds of simulation time. The stimulation frequency was 10 kHz. In Figure 3, the firing patterns are grouped assemblies. The stimulation increased the spiking activity between 2 and 7 seconds during the simulation for both the new and trained assemblies. The integration time of the simulation was 0.1 ms. Despite special notice, we used the same stimulaton parameters as those in the paper. The trained assemblies exhibit clustered group firing both before and the firing, indicating that training on new assemblies did not significantly disrupt stability of the preformed assemblies.

To record the effect of stimulation , we recorded the average connectivity strength (J^{EE}) for every cluster in both the old and new populations. We recorded the average weight once every second. Figure 3 bottom left clearly shows, in response to the stimulation, the formation of the first assembly within the new populations. Other assemblies remain at the initial assigned weights. For the preformed assemblies, the weights did not change due to the stimulation, in agreement of the spontaneous structured activities

before and after the stimulation (Figure 3 top right).

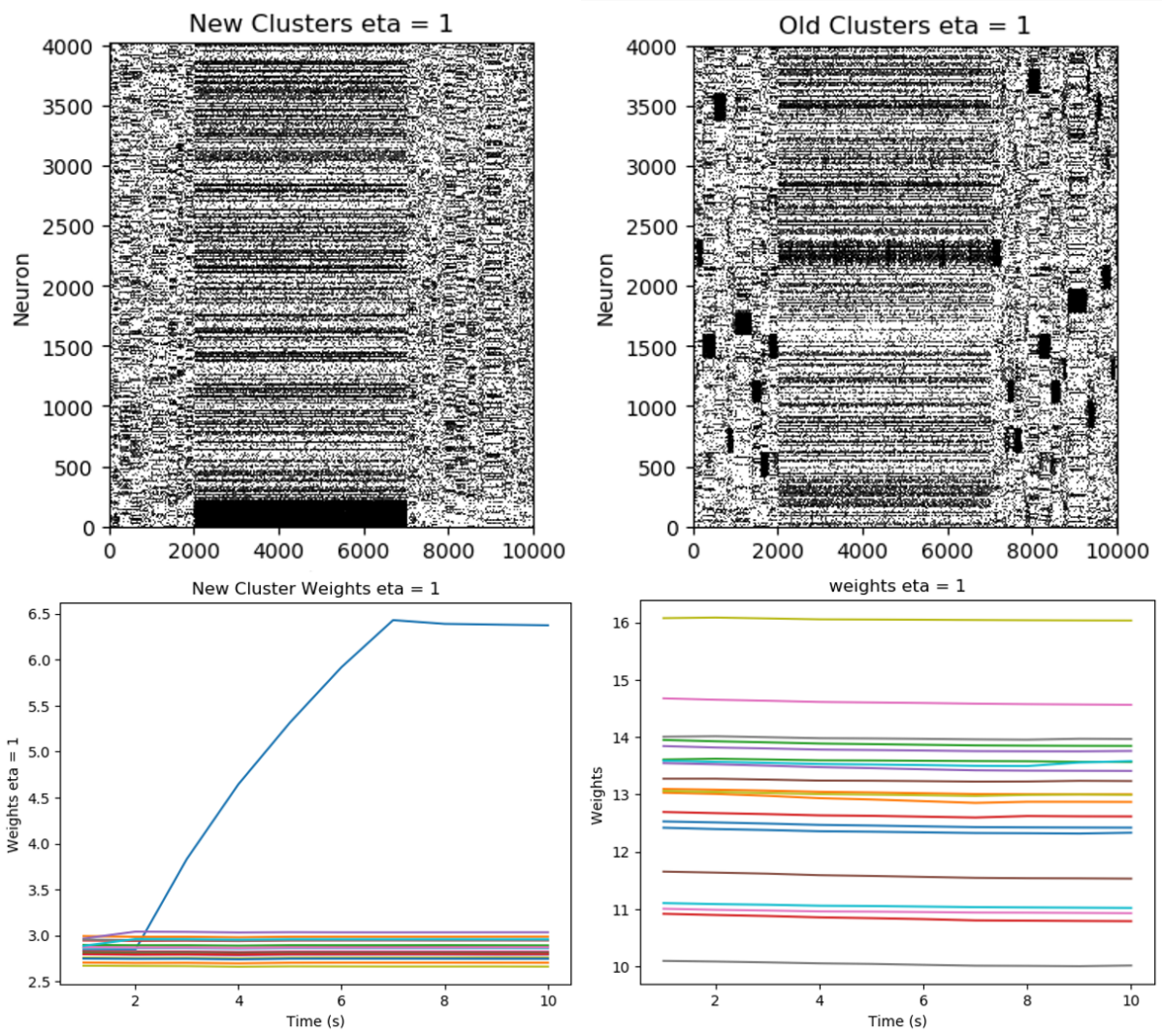

Figure 3: Top row: the spike timings of the trained and new assemblies. Bottom row: within assembly weight change for the trained and new assembles

We used a different stimulation scheme from the original authors. It is important to note that the original authors used an alternating stimulation pattern to retrain the model. We only retrained one of the newly defined assembly as the target pattern More importantly, the original authors trained the model for few thousands of seconds. Given the computation time constraint, we only trained the model for maximum of 100 seconds. Even with a short amount of time, we expect to explore the rich dynamics of this STDP model.

With the stimulation and definition of population in place, we examined the effects

of the learning rates on the stability of the trained and new structures, respectively. For each simulation, we used the same definition of the pre-trained and new assemblies.

4.1.2 Effect of Inhibitory STDP on Neuronal Stability

As described above, we varied the inhibitory learning rate (η in equation 6) with three orders of magnitude. Figure 4 summarizes the effect of η on the weight change. Across the learning rate range (first row), η did not affect the connectivity between the pre-trained excitatory neuronal connections. This effect can be easily understood from equation 6 where η only affects the connectivity between excitatory and inhibitory neuronal connections. However, increasing the inhibitory learning rate leads to lowered final average in-cluster weights for the new stimulated neuronal assembly (bottom row of Figure 4). Taken together, these results suggest that the inhibitory STDP learning rate only affects formation of new neuronal assemblies.

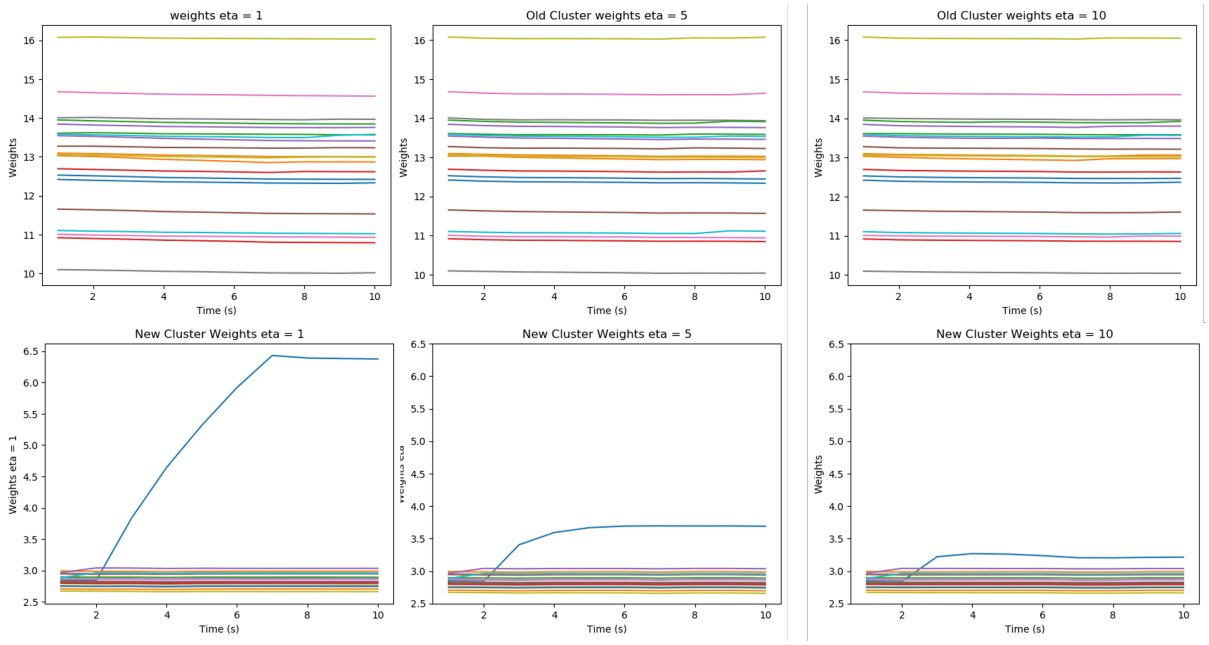

Figure 4: In-cluster connectivity weight change with increasing inhibitory learning rate for pre-trained populations (top row) and new populations (bottom row)

4.1.3 Effects of Excitatory STDP on Neuronal Stability

We then examined how changing the excitatory learning rates affect the stability of existing neuron structures and formation of new clusters. Keeping the inhibitory learning rate as the default value, we first examined the effect of long term depression rate (A_{LTD} in equation 6). Figure 5 shows the average connectivity weight change as a result of the 10 kHz stimulation for 5 seconds during the 10 seconds simulation.

The stimulation had opposite effects on stability the preformed and new assemblies. Even before the stimulation started, the preformed assembles drastically decreased the average connectivity weight faster as the A_{LTD} increased. Comparing the slopes for the change in the weights (stimulation was turned on at 2 seconds), the stimulation slowed the deterioration of the preformed neuronal assemblies.

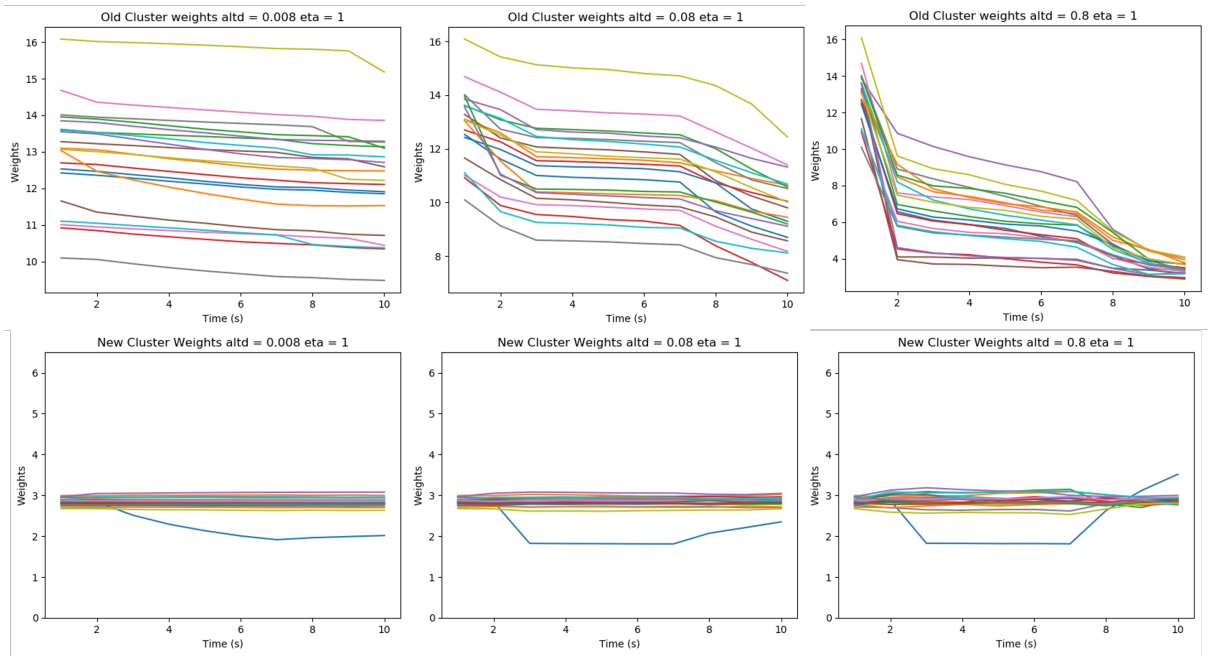

Figure 5: In-cluster connectivity weight change with increasing long term depression learning rate for pre-trained populations (top row) and new populations (bottom row)

To the opposite, only the stimulated neuronal assembly in the new populations decreased to the minimum connectivity weight as defined in the simulation. The decrease in the average connectivity weight can be understood with equation 5, where the increasing A_{LTD} amplifies the effect of long term depression. Interestingly, once the stimulation was

off at 7 seconds, the average weight for the new stimulated population climbed up (Figure 5 bottom middle and right).

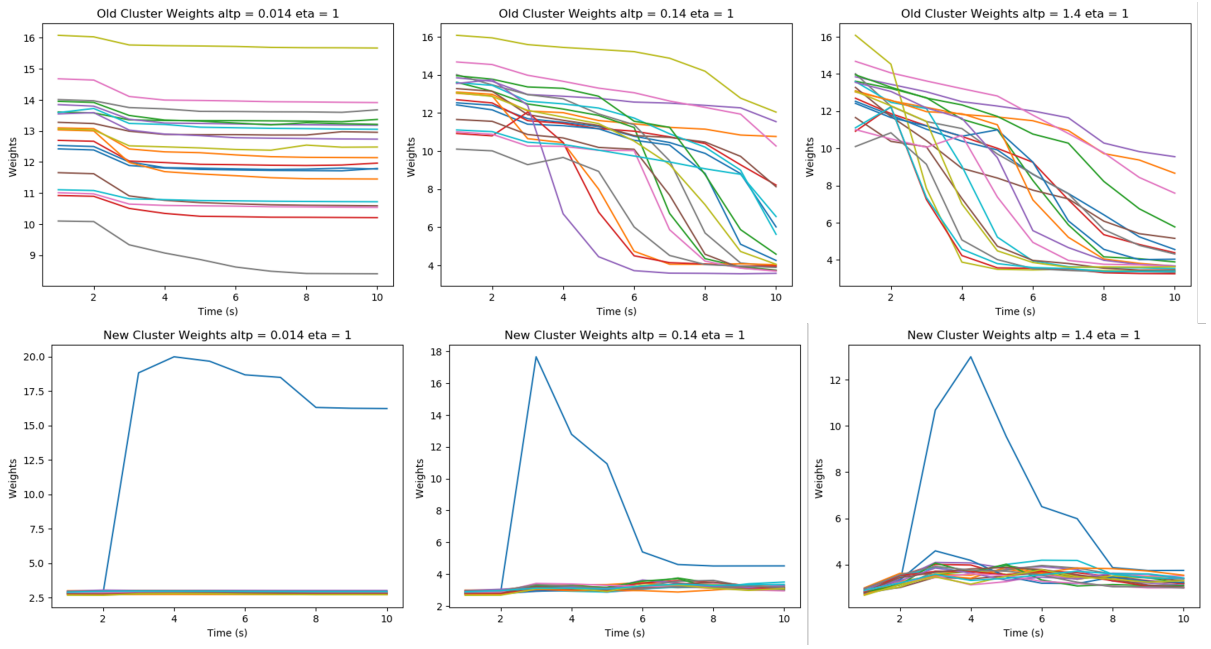

Figure 6: In-cluster connectivity weight change with increasing long term potentiation learning rate for pre-trained populations (top row) and new populations (bottom row)

In addition to the learning rate for LTD, we also investigated the effects of long term potentiation rate on the network stability. Again, the stimulation seemed to have opposite effects on the preformed and new assemblies. For the new assembly, the stimulation initially increased the average connection weight for the target assembly; afterwards, the stimulation actually caused the connection weight to decrease. This is very counter-intuitive but highlights the rich dynamics the STDP model can exhibit. Without further investigation, we can presume that the after the stimulated network reached certain configuration, the membrane potentials of the neurons in the stimulated assembly were locked into a value between the LTD threshold and LTP threshold (θ_{LTD} and $theta_{LTP}$ in equation 5). As a result, the stimulation only activated the LTD mechanism, leading to rapid decline in the excitatory connectivity weights. Similarly in the case of performed assemblies, the stimulation decreased average connectivity weights.

4.2 Changing Network Parameters

4.2.1 Neuronal Population Size

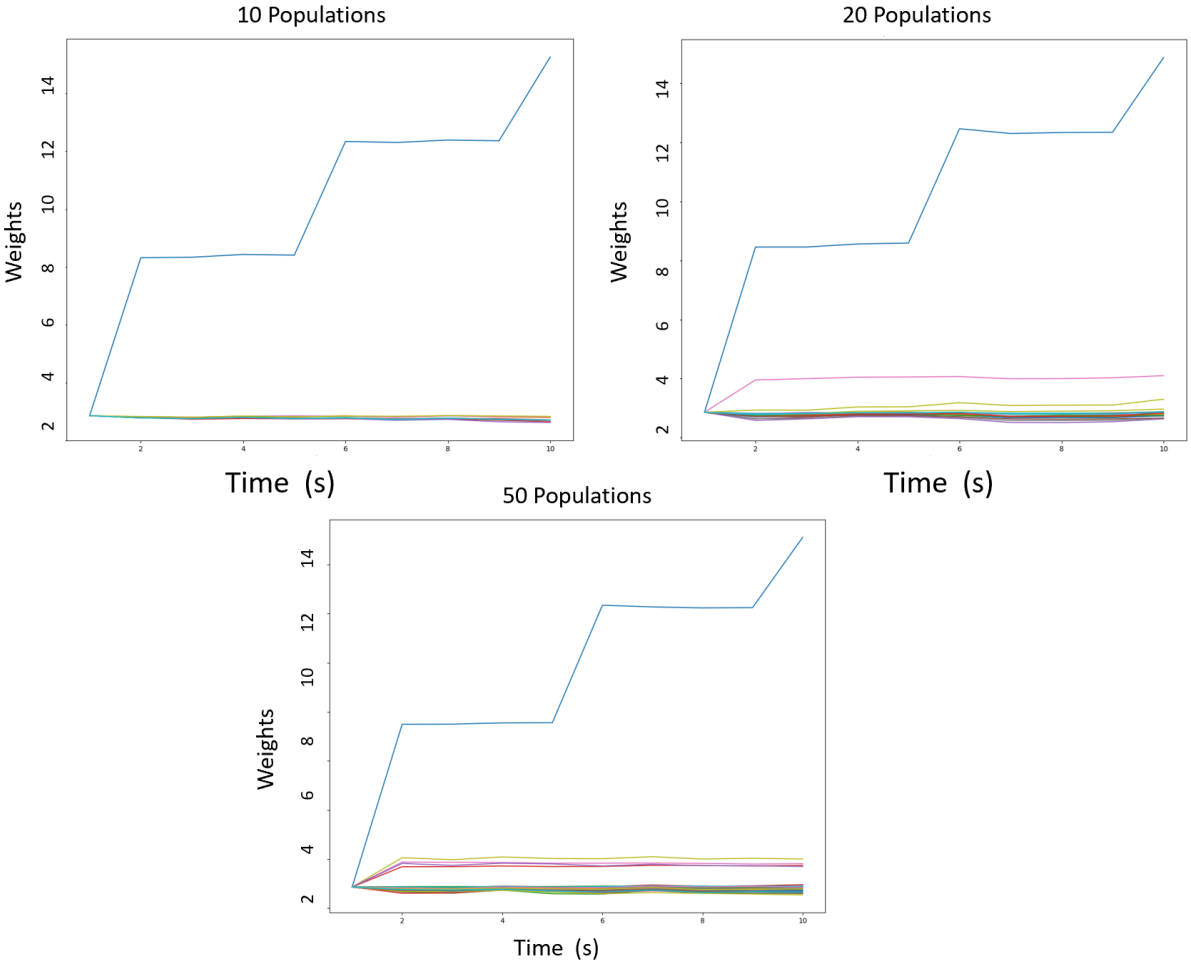

Figure 7: Top row: both of these figures show the average weights of all the populations in an untrained network over the course of the stimulus (10 seconds). The figure on the left contains 10 neuronal populations and the figure on the right contains 20. Bottom row: this plot contains 50 populations.

After looking at the stability of the network we wanted to look into what other variables affected the structure of the network. The first variable we looked at was the size of each neuronal population. When a random network is generated, neurons are randomly grouped into different populations. Neurons in a population are neighbors with the other neurons in that population and will synapse onto each other. It is important to note that a neuron can be a part of multiple populations. As such, the average weight

of connections in a population separate from the one being stimulated can increase. We stimulated one population of a randomly generated network for a total of 10 seconds, but with a 1 second pause between each 1 second stimulus of 10 kHz. We looked at a smaller network consisting of 500 neurons, 400 excitatory and 100 inhibitory, with neuronal populations consisting of 300 neurons. Figure 7 shows us that as you increase the number of populations, more populations separate from the stimulated population will see an increase in their average synaptic strength. With more neuronal populations, a greater percentage of neurons will be neighbors and synapse onto each other. This leads to more stronger synapses across populations. At 50 populations, five different populations almost double the average weight of their connections.Looking at the relative scale of the impact of changing this parameter, the number of neuronal populations doesn't have a major impact on the network architecture unless there is a dramatic increase.

4.2.2 Stimulation Frequency

Next, we looked at how changing the frequency of the stimulus would affect the network. Again, we looked at a smaller network of 500 neurons, 400 excitatory and 100 inhibitory, with 20 neuronal population comprised of 300 neurons. Stimuli of 2, 6, and 10 kHz were applied to a randomly generated network for a period of 10 seconds with 1 second of stimulus followed by a 1 second pause. As you increase the stimulus frequency, the most noticeable change is in the stimulated population. With a 2 kHz stimulus the stimulated population only has an average weight of connection of around 4, but this value jumps to 16 for a 10 kHz stimulus. Additionally, as the frequency increases, there is a slight increase in the average synaptic strength of a minimal amount of populations .

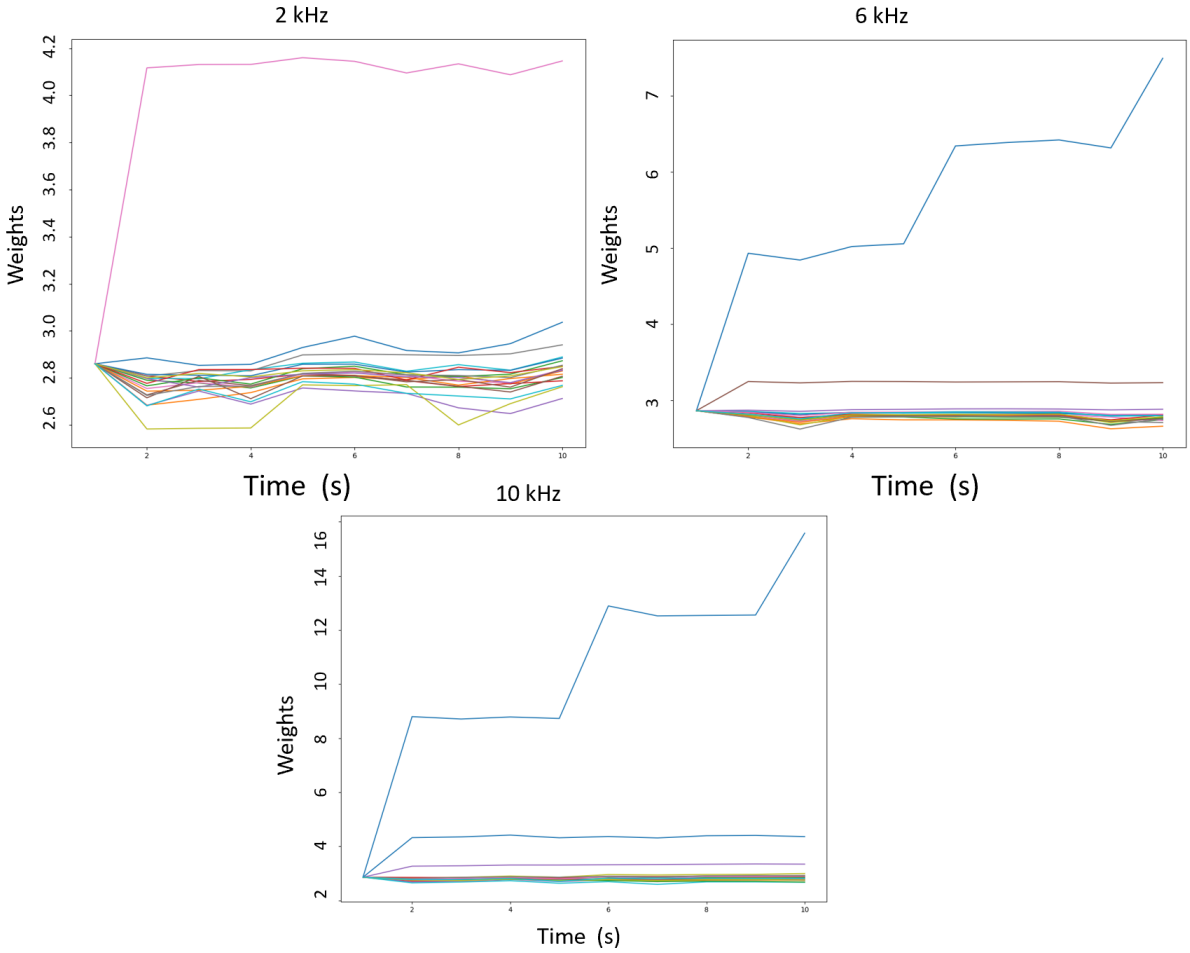

Figure 8: Top row: both of these figures show the average weights of all the populations in an untrained network over the course of the stimulus (10 seconds). The figure on the shows the results of a 2 kHz stimulus, the one on the right a 6 kHz stimulus. Bottom row: this plot shows 10 kHz.

4.3 Dimensionality Reduction

Finally, we applied dimensionality reduction to the model through principle component analysis (PCA). To accomplish this we created a matrix where each row represented a neuron and the columns represented time (each column is one timestep). Each entry was either a 0 or a 1, with 0 meaning that the neuron did not fire and 1 that it did fire. This matrix can be thought of as a translation of a raster plot. We then fitted a PCA model to this matrix. As such, we used PCA to model the variability in the firing of each neuron. This was done for a network consisting of 5000 neurons, 4000 excitatory and 1000 inhibitory, with 20 neuronal populations of 300 neurons. We applied stimuli to a

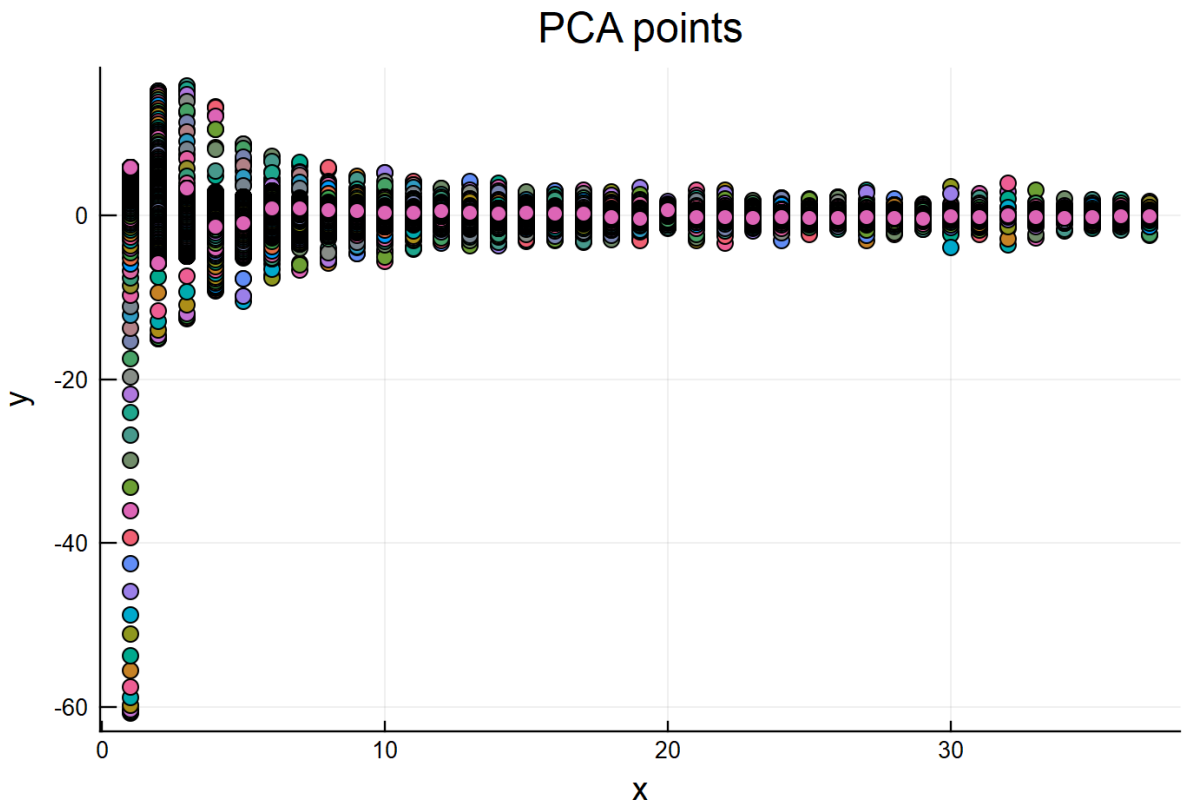

Figure 9: This is a scatter plot of the principle component scores of the untrained network. The axes represent the x and y components of the principle component scores. All 5000 neurons in the network are plotted.

randomly generated network for a period of 10 seconds with 1 second of stimulus followed by a 1 second pause. We also applied this stimuli to the trained network provided by the authors. Figure 9 shows the principle component scores from PCA performed on the untrained network.

We found the number of components for the untrained network and trained network to be 41. The ratio of variances preserved in the principal subspace was 0.99 for both generated PCA models. These results suggested that training a network did not increase its complexity. Our null hypothesis was that each neuronal population would explain the variance in firing. If this were true, PCA would've resulted in 20 components. However, since there are more components, there are other significant factors in the network that affect neuronal firing. Further investigation is needed to determine what exactly these factors are.

5 Conclusions

With this paper we have been able to confirm results from the Litwin-Kumar et. al's work, "Formation and maintenance of neuronal assemblies through synaptic plasticity," as well as build on their results. First, we were able to reproduce the paper's raster plots demonstrating that the spontaneous dynamics of a trained network reflect evoked stimuli. This provides evidence for the idea that STDP helps reinforce the memory of a stimuli through spontaneous dynamics.

We then looked at the behavior of stability in the network. The learning rates affect both the stability of pre-formed structures and the formation of newly formed structures. From our results we can see that excitatory STDP is much more important than inhibatory STDP, as η seems to limit how much a new population can grow in weight, while not affecting old networks at all. However, increasing both long term depression and long term potentiation seems to have a negative affect on the synaptic weights on both new and old systems. This shows that LTD and LTP share a delicate balance with synaptic weights.

Next, we changed a few of the network parameters. When the number of populations increased, more populations would see an increase in the average synaptic strength as one population was stimulated. As the frequency of the stimulus increased, the stimulated population saw a huge increase in its average synaptic strength and a few other populations saw far smaller increases in the weight of their connections. Both of these results indicate that large changes are needed in either of these parameters in order to have a meaningful effect on the network.

Finally, we used dimensionality reduction to study the complexity of our network. Through PCA we were able to see that both an untrained and trained network can be simplified to 41 significant components. More analysis is needed to ascertain what theses components represent and what they tell us about the activity of the brain.

References

[1] Ryan C. Williamson, Benjamin R. Cowley, Ashok Litwin-Kumar, Brent Doiron, Adam Kohn, Matthew A. Smith, and Byron M. Yu; *Scaling Properties of Dimensionality Reduction for Neural Populations and Network Models*. PLOS Computational Biology 12 (12), e1005141.

6 Appendix

In this project, we inherited the original model and parameters in the programming language Julia. To support our analysis, we implemented a weight calculating function within the simulation loop. We here first show the code to stimulate a network which was supplied by the authors. (assigning values to parameters is excluded for the sake of brevity)

```
    v = zeros(Ncells) #membrane voltage
    nextx = zeros(Ncells) #time of next external excitatory input
    sumwee0 = zeros(Ne) #initial summed e weight, for normalization
    Nee = zeros(Int,Ne) #number of e->e inputs, for normalization
    rx = zeros(Ncells) #rate of external input
    for cc = 1:Ncells
        v[cc] = vre + (vth0-vre)*rand()
        if cc <= Ne
            rx[cc] = rex
            nextx[cc] = rand(expdist)/rx[cc]
            for dd = 1:Ne
                sumwee0[cc] += weights[dd,cc]
                if weights[dd,cc] > 0
                    Nee[cc] += 1
                end
            end
        else
            rx[cc] = rix
            nextx[cc] = rand(expdist)/rx[cc]
        end
    end

    vth = vth0*ones(Ncells) #adaptive threshold
    wadapt = aw_adapt*(vre-vleake)*ones(Ne) #adaptation current
    lastSpike = -100*ones(Ncells) #last time the neuron spiked
    trace_istdp = zeros(Ncells) #low-pass filtered spike train for istdp
    u_vstdp = vre*zeros(Ne)
    v_vstdp = vre*zeros(Ne)
    x_vstdp = zeros(Ne)

    Nsteps = round(Int,T/dt)
    inormalize = round(Int,dtnormalize/dt)

    N_steps_save_weights = Nsteps ÷ Nskip
    #weights_time = zeros(Float32,N_steps_save_weights,Ncells,Ncells)
    weights_time = zeros(Float32,size(popmembers,1),N_steps_save_weights)
    w_t = 1
    weights_time[:,w_t] = calc_cluster_weights(popmembers,weights)

    println("starting simulation")

    #begin main simulation loop
    for tt = 1:Nsteps
        if mod(tt,Nsteps/100) == 1  #print percent complete
            print("\r",round(Int,100*tt/Nsteps))
        end
        t = dt*tt
        forwardInputsE[:] .= 0.
        forwardInputsI[:] .= 0.
```

```
        #check if we have entered or exited a stimulation period
        tprev = dt*(tt-1)
        for ss = 1:size(stim)[1]
            if (tprev<stim[ss,2]) && (t>=stim[ss,2])  #just entered stimulation period
                ipop = round(Int,stim[ss,1])
                for ii = 1:Nmaxmembers
                    if popmembers[ipop,ii] <= 0
                        break
                    end
                    rx[popmembers[ipop,ii]] += stim[ss,4]
                end
            end

            if (tprev<stim[ss,3]) && (t>=stim[ss,3]) #just exited stimulation period
                ipop = round(Int,stim[ss,1])
                for ii = 1:Nmaxmembers
                    if popmembers[ipop,ii] <= 0
                        break
                    end
                    rx[popmembers[ipop,ii]] -= stim[ss,4]
                end
            end
        end #end loop over stimuli

        if mod(tt,inormalize) == 0 #excitatory synaptic normalization
            for cc = 1:Ne
                sumwee ::Float64  = 0.
                for dd = 1:Ne
                    sumwee += weights[dd,cc]
                end

                for dd = 1:Ne
                    if weights[dd,cc] > 0.
                        weights[dd,cc] -= (sumwee-sumwee0[cc])/Nee[cc]
                        if weights[dd,cc] < jeemin
                            weights[dd,cc] = jeemin
                        elseif weights[dd,cc] > jeemax
                            weights[dd,cc] = jeemax
                        end
                    end
                end
            end
        end #end normalization

        #update single cells
        spiked = zeros(Bool,Ncells)
        for cc = 1:Ncells
            trace_istdp[cc] -= dt*trace_istdp[cc]/tauy

            while(t > nextx[cc]) #external input
                nextx[cc] += rand(expdist)/rx[cc]
                if cc < Ne
                    forwardInputsEPrev[cc] += jex
                else
                    forwardInputsEPrev[cc] += jix
```

```
236             xerise[cc] += -dt*xerise[cc]/tauerise + forwardInputsEPrev[cc]
237             xedecay[cc] += -dt*xedecay[cc]/tauedecay + forwardInputsEPrev[cc]
238             xirise[cc] += -dt*xirise[cc]/tauirise + forwardInputsIPrev[cc]
239             xidecay[cc] += -dt*xidecay[cc]/tauidecay + forwardInputsIPrev[cc]
240
241             if cc < Ne
242                 vth[cc] += dt*(vth0 - vth[cc])/tauth;
243                 wadapt[cc] += dt*(aw_adapt*(v[cc]-vleake) - wadapt[cc])/tauw_adapt;
244                 u_vstdp[cc] += dt*(v[cc] - u_vstdp[cc])/tauu;
245                 v_vstdp[cc] += dt*(v[cc] - v_vstdp[cc])/tauv;
246                 x_vstdp[cc] -= dt*x_vstdp[cc]/taux;
247             end
248
249             if t > (lastSpike[cc] + taurefrac) #not in refractory period
250                 # update membrane voltage
251                 ge = (xedecay[cc] - xerise[cc])/(tauedecay - tauerise);
252                 gi = (xidecay[cc] - xirise[cc])/(tauidecay - tauirise);
253
254                 if cc < Ne #excitatory neuron (eif), has adaptation
255                     dv = (vleake - v[cc] + deltathe*exp((v[cc]-vth[cc])/deltathe))/taue + ge*(erev-v[cc]
256                     v[cc] += dt*dv;
257                     if v[cc] > vpeak
258                         spiked[cc] = true
259                         wadapt[cc] += bw_adapt
260                     end
261                 else
262                     dv = (vleaki - v[cc])/taui + ge*(erev-v[cc])/C + gi*(irev-v[cc])/C;
263                     v[cc] += dt*dv;
264                     if v[cc] > vth0
265                         spiked[cc] = true
266                     end
267                 end
268
269                 if spiked[cc] #spike occurred
270                     spiked[cc] = true;
271                     v[cc] = vre;
272                     lastSpike[cc] = t;
273                     ns[cc] += 1;
274                     if ns[cc] <= Nspikes
275                         times[cc,ns[cc]] = t;
276                     end
277                     trace_istdp[cc] += 1.;
278                     if cc<Ne
279                         x_vstdp[cc] += 1. / taux;
280                     end
281
282                     if cc < Ne
283                         vth[cc] = vth0 + ath;
284                     end
285
```

```
#istdp
if spiked[cc] && (t > stdpdelay)
    if cc < Ne #excitatory neuron fired, potentiate i inputs
        for dd = (Ne+1):Ncells
            if weights[dd,cc] == 0.
                continue
            end
            weights[dd,cc] += eta*trace_istdp[dd]
            if weights[dd,cc] > jeimax
                weights[dd,cc] = jeimax
            end
        end
    else #inhibitory neuron fired, modify outputs to e neurons
        for dd = 1:Ne
            if weights[cc,dd] == 0.
                continue
            end
            weights[cc,dd] += eta*(trace_istdp[dd] - 2*r0*tauy)
            if weights[cc,dd] > jeimax
                weights[cc,dd] = jeimax
            elseif weights[cc,dd] < jeimin
                weights[cc,dd] = jeimin
            end
        end
    end
end #end istdp


#vstdp, ltd component
if spiked[cc] && (t > stdpdelay) && (cc < Ne)
    for dd = 1:Ne #depress weights from cc to cj
        if weights[cc,dd] == 0.
            continue
        end

        if u_vstdp[dd] > thetaltd
            weights[cc,dd] -= altd*(u_vstdp[dd]-thetaltd)
            if weights[cc,dd] < jeemin
                weights[cc,dd] = jeemin

            end
        end
    end
end #end ltd

#vstdp, ltp component
if (t > stdpdelay) && (cc < Ne) && (v[cc] > thetaltp) && (v_vstdp[cc] > thetaltd)
    for dd = 1:Ne
        if weights[dd,cc] == 0.
            continue
        end

        weights[dd,cc] += dt*altp*x_vstdp[dd]*(v[cc] - thetaltp)*(v_vstdp[cc] - thetaltd);
        if weights[dd,cc] > jeemax
            weights[dd,cc] = jeemax
```

Code to plot average weights of connections and to create raster plots

```
using PyPlot
include("sim.jl")


PyPlot.clf()
doplot = false

#plot average connection weights over time
figure
display(plot(1:size(weights_time,2),transpose(weights_time[:,:])))
xlabel("Time")
ylabel("Weights")
title("10 kHz--Untrained Network")
savefig("weight_change_2kHz_100s.png",dpi=150)
display(gcf())

#raster plots
if doplot
    Npop = size(popmembers,1)
    Nmaxmembers = size(popmembers,2)
    println("creating plot")
    figure(figsize=(4,4))
    xlim(0,T)
    ylim(0,sum(popmembers.>0))
    ylabel("Neuron")
    xlabel("Time")
    tight_layout()

    #plot raster with the order of rows determined by population membership
    rowcount ::Int64  = 0
    for pp = 1:Npop
        print("\rpopulation ",pp)
        for cc = 1:Nmaxmembers
            if popmembers[pp,cc] < 1
                break
            end
            global rowcount+=1
            ind = popmembers[pp,cc]
            vals = times[ind,1:ns[ind]]
            y = rowcount*ones(length(vals))
            scatter(vals,y,s=.3,c="k",marker="o",linewidths=0)
        end
    end
    display(gcf())
    print("\rdone creating plot")
    savefig("output.png",dpi=150)
end
```

Code to run PCA and generate scatter plot

```
using MultivariateStats, DelimitedFiles, Plots

times = readdlm("times.csv", ',')

times_rows = size(times, 1)
times_cols = size(times,2)

#fire_times is a matrix where each row is a neuron and each progressive column is one timestep ahead.
#Each entry represents a time when that neuron fired.
fire_times = zeros(times_rows, times_cols)
for neuron = 1:times_rows
    for time = 1:times_cols
        if times[neuron, time] > 0
            fire_times[neuron, time] ::Int64  = 1
        else
            fire_times[neuron, time] ::Int64  = 0
        end
    end
end

M = fit(PCA, fire_times)
Y1 = transform(M, fire_times)

Plots.scatter(Y1, title = "PCA points", xlabel = "x", ylabel = "y", legend = false)
```

Code to run stability analysis is the rest of the appendix