# June 2nd, 2023

* Use SOFM for basis of drawing images
  + SOFM neurons are “fast” neurons (from scaffolded attractors paper)
    - Quickly converge to scaffolding for a shape to be drawn
    - How to keep this unsupervised?
      * Get self organized network to acquire outputs that correspond to useful shapes
        + Some simple RL?
  + Output/activity neurons are “slow” neurons that actually execute behavior
    - Slowly converge to a sequential behavior
  + Questions to look at:
    - Does a developmental SOFM speed up learning of the scaffolding patterns?
    - Does a developmental SOFM improve the recall of the activity patterns?
    - Are the results more robust to noise if the scaffolding neurons are trained developmentally?
    - Would this network be able to draw new shapes that are shown by pulling from previous experience with other shapes?
  + Input: image to draw
  + Output: sequence of drawing signals executed by the activity neurons (i.e. a generated drawing)
  + How to make the architecture recurrent? Would it make sense to wrap the output of the activity layer back into the SOFM?
    - Or maybe the activity layer output wraps back into the activity layer input in addition to the SOFM input
  + What would developmental learning do here?
    - Initially, system can draw simple, random doodles
    - Initial learning step reinforces the good, smooth doodles
    - Reinforced ones produce new children in the developmental stage
    - Each SOFM neuron corresponds to which output neuron could be activated
      * Which neurons are activated and when
      * Need attractor dynamics so multiple neurons can be activated at once
    - Need more than one layer for drawing shapes
    - But just learning smooth drawing might only need 1 layer
  + 3d self-organized network?
    - Every point in the network corresponds to movement in a certain direction
      * 3rd dimension specifies delay
  + Fast attractor creates a context in which slow attractor emerges to draw a shape
  + Furqan paper never closed the loop (to enforce good shapes)
    - 1. Need to define “good shape” and “bad shape”
    - 2. What kinds of shapes
    - 3. How are those shapes related to those in early developmental stage
  + Define more complex:
    - Finer levels of control? More variation?
  + Teaching a non-developmental system to do only the good behaviors is a tough problem
    - Benefit of developmental learning
      * Prune off behaviors that will not be useful in future, encourage useful behaviors
      * Over a few levels of development
  + Writing letters (spiking neurons paper)
  + Bump could be a bias on slow attractor to create sequential activity
    - Need a reinforcement loop to enforce good “bumps”
  + replace fast attractor with SOFM from furqan paper, keep slow attractors
    - More complex shapes requires a way for higher level patterns to form
      * Second layer of SOFMs that sequentially produce bumps
    - More control of simple shapes gives more exact drawings of straight lines in different directions
    - Each SOFM bump corresponds to a simple shape, and a dynamical series of bumps in the SOFM corresponds to a more complex shape
    - Goal of dev:
      * Read out simple shapes, gradually be able to produce combinations of shapes
        + Turns SOFMs into
        + Look ato 2010 icdm paper
* Simpler system than Furqan paper
  + Look into maintaining fast and slow neuron system or changing
  + Output: system that draws lines/doodle
    - Small number of possible outputs
      * Small group of neurons that act together to produce a result (x, y)
      * Each output neuron would readout a temporal output pattern
        + Could simplify to having one drawing output neuron
        + Or multiple output neurons that draw a given shape

Focus on control part of system, not on output part

* + Learning process reinforces smooth outputs
  + Timings are learned parameters (output by lower level network)
  + Output signals wouldn’t be square waves
  + Stimulate system from underlying neural system
  + Every set of parameters would produce a shape as the output
  + Output neurons would have differential equation for controlling activation/deactivation
    - Needed to add delay (done by creating attractor network)
  + Fast neurons activate sequentially to create spatiotemporal pattern
    - Could simplify things here
    - Were previously unable to get enough variation in the delays
    - Specify more directly what the output neurons should do

# June 8th, 2023

* Realized in scaffolding attractors paper that everything is in one layer
  + Was previously envisioning a SOFM scaffolding layer, and a (possibly recurrent) activity layer
    - Both must be recurrent to be attractors and have dynamics
* Should SOFM be organizing the inputs or the response patterns?
  + Scaffolding (SOFM) still fast neurons
    - SOFMs produce a fixed pattern
  + Activity neurons have to be slow so they produce activity over a period
    - Slow neurons produce spatiotemporal output
* What are the inputs?
  + Handwritten digits – if model sees a “1”, it draws a “1”
    - Read “1” -> set up scaffolding for the “1” activity pattern -> execute the “1” activity pattern
  + Handwritten shapes – same idea as above
  + Combinations of shapes/patterns – given an mess of different overlapping, noisy, shapes, different areas of the SOFM will be activated (so, multiple activation bumps). Train those bumps to provide the scaffolding to draw the different shapes in a clean way, separating them out and extracting from the noise
    - Same idea applies with numbers/letters
    - Kind of like decoding a “captcha” and reproducing the characters
  + Needs a prompt (from outside or inside the system)
* What is the (intended) effect of development?
  + Produce more complex shapes?
  + Produce more specific shapes (i.e. differentiate between rectangle & square, circle & ellipse, etc.)
  + Something to think about if keeping the layers separate: Are we developing the SOFM scaffolding layer? The activity layer? Both?
* What will the system do?
  + Needs a prompt
  + Prompt will cause a bump once the sofm layer reaches equilibrium
    - How the brain stores and recalls memory
  + Brain has priors for drawing simple shapes, could skip the step of developing the simple shape ability
  + Show an input shape, hidden layer (SOFM?) takes input and outputs a bias to the (ring?) activity layer to produce the correct activity
    - Maybe need an extra layer in between
      * Translates the static signal from SOFM into a sequence of active inputs to the activity layer
        + Why not include this in the ring? Ring is there to produce the output robustly (can’t give it too much responsibility)
      * 2010 ICDL paper & Averbeck work key insight:
        + Pattern that specifies the sequence in the ring layer doesn’t have to be a dynamical pattern
* development

1. First stage - Control itself in drawing simple lines/curves

* + - Babbling stage
    - Prune out useless shapes, and learn to only produce useful shapes
    - Could short-circuit this stage and make network able to produce simple shapes automatically

2. Eventually try to draw more complex shapes

* Solid idea:
  + SOFM -> “input/what” layer (1 active neuron, says what shape to draw)
  + 2nd layer -> “how/code” layer (encodes what shape to draw)
    - 1. How is the code generated and stabilized? Fixed pattern or dynamical pattern?
      * Does the ring layer have to deactivate the code neurons that are exciting it?
      * The dominant code bump needs to inhibit the activation of the other bumps
  + Ring layer -> pen draws in certain directions
* Where is the (developmental) learning?
  + Ring layer, code layer, and input layer start small
    - Only simple shapes are indicated, coded for, and drawn
      * Undesirable simple shapes are weeded out
  + All layers grow in the same way to provide more precise movement
  + But where is their scope for learning?
    - Input layer to the code layer
      * SOFM as an attractor network requires learning
    - Code layer to the ring layer
      * May need to be learned too
    - Everything after the code layer can be fixed but growing
      * While system grows more complex, harder to control
        + But learning simpler shapes early helps later complex learning
      * Probably don’t focus on learning how to go code -> activity just yet
        + Want to be able to control the speed/duration of activity and pen up/down
  + Most developmental learning will be input -> code
  + Can maybe have the code -> ring be learned early on and then have it be established (later on)
* Assumptions:
  + Drawing starts at a certain point
  + Shapes are simple enough to be decomposed very simply
  + What if same motion required multiple times for a sequence?
    - No reason a given direction should only be represented by one neuron
* Implementation:
  + Start with ring layer, play around with bumps on the ring, inputs into ring, and see how we can control it
    - Look at the equations
      * How can we switch from one bump to another? (code layer)
  + What equations are needed
* Also look at modern stuff in this paper (who’s quoting Averbeck?)
* Next steps:
  + Averbeck papers
  + Who’s citing averbeck
  + Control systems / writing systems
  + Hubfield network slides
  + Recent attractor network work by AM (used so many different ways)

Follow-up email:

I think we can build a first version of the code-ring network without using any attractors at all (though we may add them later to add more robustness). Basically, the ring will just be a passive layer of neurons with each neuron tuned to its position in the ring. The size of the ring thus indicates resolution. In the code layer, each neuron has a gaussian projection of weights to the ring, centered on a specific ring neuron. So a code layer neuron that points to the "north" neuron in the ring will also have smaller connections to its neighboring neurons. We can even imagine the code layer to be a ring with basically a 1-d retinotopic projection to the ring layer, though the code ring will need to have width as well with a row of neurons at a particular angular position having the same projection to the ring, so the code layer would be a 2-d strip of neurons formed into a ring.

The code layer neurons will need to have real-valued outputs with a significant dynamic range (though it is possible to do this with binary neurons by replacing each neuron with a neuron group and activate various fractions of the group.) Each ring neuron will have an intrinsic dynamic response that causes its activity to rise, stabilize and then fall. The key thing is to write the differential equations for the ring neurons so that the activation timing and duration ring neurons can be modulated by the code layer pattern - somewhat like the slow neurons in the scaffolded model were modulated by the fast ones.

Let's first focus on building this network. Then we'll get to the problem of learning the codes.

# June 16th, 2023

* Is code layer 2-d so we get sequential/temporal dynamics stored spatially? – want multiple neurons to be giving the same response. Only one neuron isn’t as flexible (depends on how we turn neurons off – duration matters in both ring and code). Old model made it hard to turn a neuron off and back on
  + Might be other ways to do this, but redundancy in systems is good – extra robust
  + Code neurons need variable amounts of activity – how is real-valued activity represented?
    - Real-valued neurons – but typical activation functions don’t give enough range
    - Binary neurons – multiple neurons act as an assembly to decide how much the group is activated
      * Attractors are easier to implement with binary neurons
        + Attractors add robustness – once system learns to write an “A”, that needs to be memorized (as an attractor!)

Alternatively could use an SOFM to memorize (to replace the code layer)

Needs to be able to have multiple bumps, and multiple different levels (which cause different durations in the ring layer)

* We are contemplating having multiple ring neurons activated at once – even more complicated?
* Need to be able to control both order and duration in code layer
* Need to map control layer to ring so ring neurons are activated in specific order and specific durations
  + Attractors might help with controlling these, but want to keep code layer spatial, not temporal
    - Which translates to spatiotemporal pattern in ring layer
* Look at ICDL paper again
* Should code neurons be inter-connected yet? If so, only connected within their “group”? Or also connected to adjacent groups?
  + Code layer starts as unconnected network, and ring is an unconnected network
    - Work out how does activity pattern in code layer translate to dynamic pattern in ring layer?
      * Need to control order of activity in ring layer and duration of activity in each part of the ring
* Start by picking apart ICDL paper and modify from that
  + Except output isn’t a spiking layer
* Start by setting up system to respond to one pattern
  + Create output system first, then figure out the code system
  + Use kind of equations from scaffolding attractors and synergies paper (since ICDL is spiking network)
    - Standard continuous time hupfield network equation
* Then figure out codes
* Then figure out development

# June 23rd, 2023

* Implemented a very simple, discretized drawing output mechanism
* Takes in a sequence of activation vectors for the ring layer and outputs the doodle drawn
  + Currently using real activation values – can switch to binary
* Simply just sequentially adding vectors, no differential equations or anything yet

# July 6th, 2023

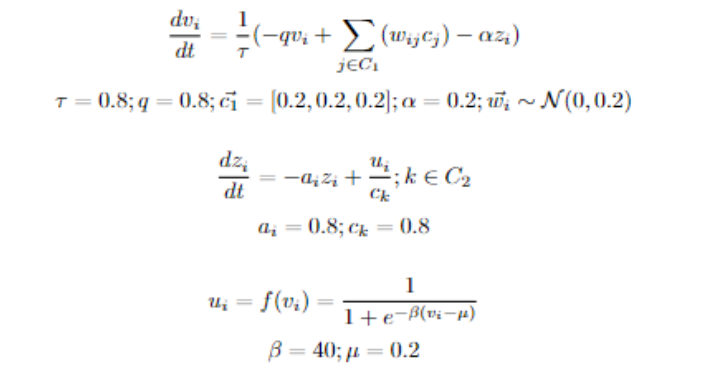
* Should R neurons have overlapping activation times? Or should the most activated draw first, and once it’s done, then the second most activated draws, etc.
  + No reason for no overlap
  + ICDL, most active goes first while inhibiting others, then un-inhibits the others
  + Could we inhibit from C rather than from R neurons?
  + R neurons go off because they run out of energy then need to go off
    - Want to be able to control this
    - Might need more mechanisms/signals to do this
      * Or can skip this, but constrains results significantly
  + Could have 2 C layers – one keeps R neurons on for certain amounts of time, and one decides the order or R activations
  + C just needs a static pattern
  + Once an R neuron is inhibited, need it to be able to be re-excited (by using a band, not just a ring)
  + Could alternatively make R a spatiotemporal attractor network (similar to scaffolded attractors)
    - Hard to control what scaffolded attractor looks like
    - They didn’t control the order of activations so this would be tricky
* M <- C connections – why do we need weights from C to M? Does this take the place of reciprocal connections within the C layer if we only had the C layer?
  + Replacing fast attractors with a systematic learning process
* If the M -> C weights will be copied to the C -> M weights, how could M and C neurons be different types?
* Will C neurons be inter-connected? Or indirectly connected through M to achieve network dynamics?
* We still need to think about inputs for the figure 1 network, right?
* Goal: Need to show that we can learn simple patterns from a bunch of possibilities, then grow to learn more options that span from the good patterns as development occurs
* SOFM learns good codes
* Measuring smoothness:
  + Look at change in angle between successive points on doodle
* Instead of writing system:
  + Replace R with another attractor network
  + Produces a spatiotemporal pattern
    - some good, some bad
  + network learns to produce only good spatiotemporal patterns
    - as network grows, more likely to create more good patterns
* R is our main challenge
  + Could turn it into an attractor
    - Now R has a mind of its own
    - Makes C -> R more complicated
  + Right now R is passive (except for the lateral inhibition)
    - But no positive connectivity
    - Just does what C says
* Turning R into strip lets us re-activate
* Turning C into strip lets us activate R to different degrees
  + But probably want C neurons to be real-valued, but storing real-values as attractors is difficult
    - Attractors want to be binary
  + Lets try to do real-valued C since it’s bidirectional and M is real-valued
  + Start with C-R network
* My decisions:
  + Make R not an attractor
  + Make C real-valued
  + Start with C-R network
  + Make github tasks to break down the problem
* AM suggestions:
  + Look at scaffolded attractors paper
    - neurons already arranged in a ring
    - make ring thicker so we can re-activate
    - make strip on code layer
    - this model can only produce binary-ish patterns
  + ignore ICDL paper stuff for now
  + look at attractors lecture – schedule outlook time for this
  + start without R as an attractor, just use scaffolded attractors equations without input (so R neurons only inhibit each other, not excite each other)
    - see where this gets us
  + set durations of R to be constant for now, but later can change this
    - could get into dendritic computing – using dendrites instead of wires
      * have weights modulated by input
      * depends on weight, modulated by some signal (which comes from another set of neurons)

# July 14th, 2023

* synergies paper
* suppose code activity between 0 and 1, but each neuron is binary
  + if spiking neurons, change spiking rate
  + if real-valued neurons, how to make it stop at 0.5, instead of going all the way to 0 or 1?
    - Use a module of several neurons – use a “band” for the code layer
* Have 3 terms that reduce activity of R neurons
  + Try them out and see if we need both z and -q\*v\_i
  + Maybe take out z term?
    - Might need something with z part to enable neuron to turn off after some time
* Need something that will turn off neuron depending on desired duration
  + Should this come from code layer?
  + Do we need 2 code layers in parallel?
    - One provides input to keep neurons excited for durations
    - And one provides input to decide order of activity
    - Could consider each R neuron as an excitatory-inhibitory pair, where each half of the pair only receives input from one of the code layers
    - This is a competition between excitation and inhibition
  + Or maybe instead use dendritic computation – let’s try this
    - Not have a second term that inhibits neurons, but have the inhibition work on modulating the weights rather than other terms in the equations
    - Make g dependent on input from other code layer and time?
      * Second code layer (C\_2) is a gate to the dendrite of C\_1 to R
* Start with equations for a ring layer, imagining fixed inputs from code layers
  + How do we want to configure the R neurons?
  + Start with one R neuron:
    - No lateral connections because only one R neuron – so the third term doesn’t exist here
    - Want a neuron that turns on, then turns off after some time that depends on code input
  + Then later we’ll add lateral inhibition, which ensures that there’s only one neuron at a time
  + \*\*\* write down equations \*\*\*

# July 21st, 2023

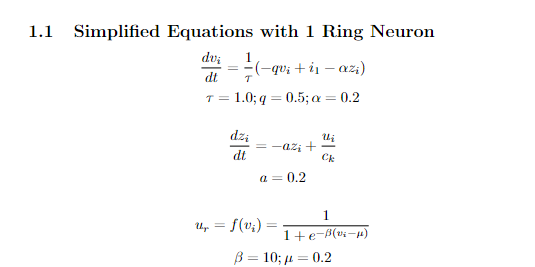
* Should the gates control each C -> R connection?
* If resource depletion is controlled by the gate for a specific C -> R, do we activate only one R at a time? Or only one C at a time?
  + Just because a given code neuron has an active signal doesn’t mean its resource should be depleting. Only when its corresponding ring neuron is active due to its output to the ring neuron?
  + How do we determine if a given code -> ring connection is causing activity at the current moment? By whether the gate is open.
    - But then shouldn’t the other gates be closed while a given gate is open?
    - Lateral inhibition of the gates?
* Should resource depletion rate depend on the code neuron’s output?
  + Stronger code output -> slower depletion?
* ICDL – gate was controlling the duration of neuron activation
  + Was a general input into the response neurons
  + Deactivation variable was dependent of gating inputs
  + 2 inputs into R layer, one from E, one from M
    - Not clear in ICDL paper, but are in thesis
  + E->R decides order
  + M->R decides duration
* Make the fall-off of the output be steeply gradual
  + Depletion rate of the resource is being gated!!! \*\*\*
* Can make each M neuron activate only one C neuron, and only one C neuron activates one R neuron
* Scaffolded attractors paper might work
  + But make deactivation of ring neurons
* New variable which is output state of code2 neuron
  + This will be the weight for deactivation rate of the ring neuron
* First thing to try:
  + Implement a single R neuron with “input” from C1 and C2
  + How are those inputs incorporated into ring neuron equations
  + C1 in the activation variable
  + C2 in the deactivation variable
  + But would need a third state variable, which is a resource variable
    - Changes over time in response to C2
    - Resource becomes weight of deactivation of ring neuron
  + Change values of C1 and C2 to see if we get desired effect
  + Then create multiple ring neurons and give lateral inhibition, and see if they can control the sequencing and duration of the ring
  + Ring has v and u in ICDL [he means u = z in scaff attr paper, like u is the deactivation?)
    - As u keeps rising, u can get high enough to deactivate v
    - V is driven by excitatory input from other neurons
    - U is driven by v
    - As v remains positive, u keeps growing
    - But as u grows, v’s activation leaks out (decelerates)
    - Can give u a variable gain in 2 ways:
      * Control sensitivity of u
      * Control sensitivity of u in v
    - Add input from c2 neuron into z equation
    - Or make b\_i controllable by input from c2 neuron
  + Can also apply modulation to alpha
  + Keep first, second and last term of eq 3
  + Keep zi equation, but make b\_i depend on c2 output
  + A graph of a number of numbers

    Description automatically generatedCan use eulers method implementation

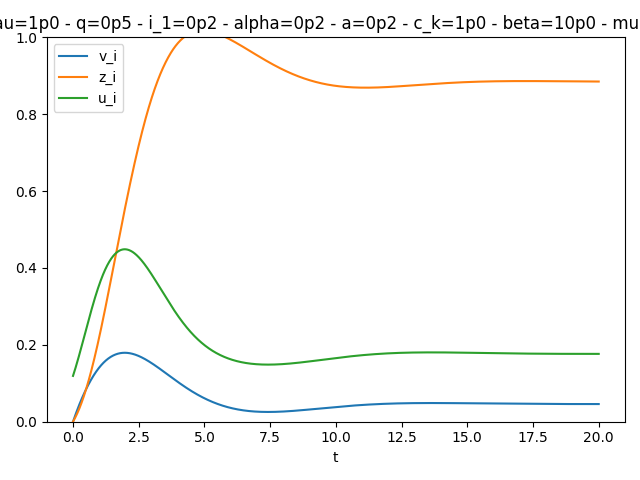
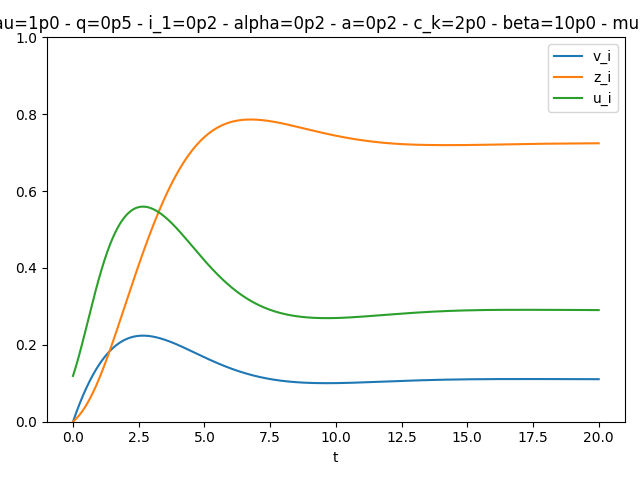
# July 28th, 2023

* take out weights and replace sum term for now in v\_i equation
* Start with v, z, u all set to 0
* Plot u\_i with different values for c\_k and i\_1 (which is same as c\_j, just without weights – input to ring i)
* V might rise too high, which is determined by interplay between activation and deactivation
  + V depends on activation and deactivation values
  + Can change act/deact params and sigmoid params to keep v from rising too far, and deactivates v faster/slower
  + If beta is small, the sigmoid is less steep
  + If beta large, sigmoid Is very steep
    - Sigmoid will only play a role in activation/deactivation once v has risen very high
    - Also depends on value of mu
* Set mu s.t. it deactivates the neuron once it reaches the peak that we want
  + Set mu small so it gets deactivated as soon as activation occurs, but not too soon to stifle all activation
  + Beta small so dependence is almost linear
* Want u to plateau for awhile
* Fix beta and mu, and play around with the two inputs (i\_1 and c\_k)
* Start off v and z and u at 0.0
* Lateral inhibition:
  + Add up all inputs from other (or all) neurons and multiply them by a negative gain
  + And that negative gain is added to all neurons – lateral inhibition is non-specific
  + Weight between any two neurons is the same (except possibly itself)
* Alternative:
  + Could add a fixed inhibitory term just like i\_1 to see the effect (try this later)

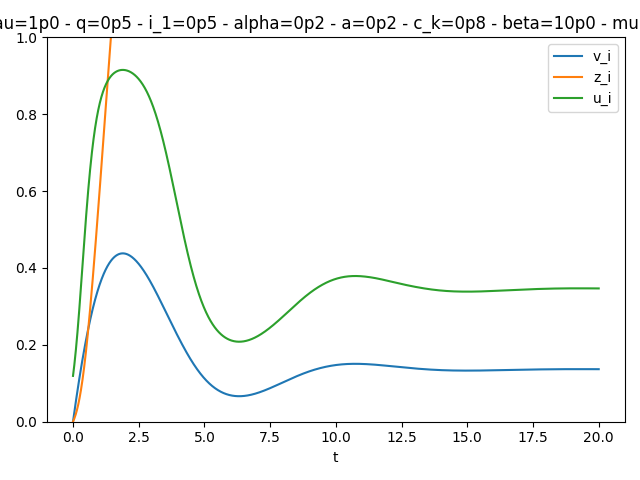
# August 4th, 2023



* Looking at changing (input from first Code layer) and (input from second Code layer)
* A graph of a number of numbers

  Description automatically generatedWith and (as a baseline example):
* Increasing to 1.0, and then to 2.0:
* So, it appears that increasing makes taper off sooner, allowing for less of a peak in the amount of deactivation () and an overall flatter deactivation curve
  + This leads to a more gradual decrease in both and since there is less of a spike in deactivation
  + We also see a higher level-off point for both and with an increase in
* A graph of a number of numbers

  Description automatically generatedLooking at the baseline again to compare results with varying ( and )
* A graph of a number of numbers

  Description automatically generated with medium confidenceIncreasing to 0.5, and then to 1.0 while holding :
  + From this, we see that increasing increases the amount of plateau-time for , and overall increases both the peak and level-off point of and
* Some things to consider:
  + is very hard to control with these equations; the deactivation values increase rapidly as the neuron activates. Maybe we need a gain parameter on in the deactivation equation?
  + With these equations, won’t come back down to 0; it tends to level off relatively high (depending on the other parameters)
    - Getting back down to 0.0 is probably a behavior we need, right? So the neuron can be reactivated?

# August 11th, 2023

* We want controlling only the slope of and therefore the speed at which hits 1.0.
  + For a constant , the duration of activation should be constant no matter what
  + For the rate of increase of , we want it to decrease at the same rate – want the net width to always be the same for constant
  + Ideas:
    - Include in the equation
    - Need to make sigmoid steeper (increase ) so we get a square wave for
* Also need to make sure returns to 0 after activation – use steeper sigmoid!
* We want to control the duration of the square wave
  + And for constant , the wave should have the same starting time of activity
* Next steps:
  + Increase steepness of sigmoid and maybe adjust
  + Plot for different , want all pulses to have same width
  + Add another plot for and

# August 15th, 2023

* Plot v vs u over time
* Given these equations, if we want v and u to return to 0, we can change mu and beta to where the settling point of v is pretty low (close to 0)
  + If that settling point is low enough, where z is close to 0, that’s all we can do
* Plot this with something with a phase plot
  + u-v is my phase/state space
  + any time series can be plotted as a trajectory in phase space
* when I increases, the settling point for v increases – do we want that?
  + No-we want the settling point to be close to 0 for v
* Its okay if trajectory drops negative briefly
* Around my intersection point determines if the equilibrium is stable or not – need to run the thing
  + If unstable: there will be some periodic cycle or something
* Uv term is common in neural models – intuition becomes harder
* \*\* plot trajectory over my nullclines – where trajectory is a time-plot (gif) on u-v plane
  + Run sims with different values of I and see what happens
* Worst case: make a parameter dependent on a variable
  + Conductance of membrane depends on voltage in some existing neural model
    - Like making rho a function of v
      * Could be algebraic, could be differential (unfortunately)
    - Completely neuro-scientific
    - U is kinda like this already (voltage dependent resetting variable)
* Can also plot time series like before but plotting in u-v space will show dynamics better

# August 25th, 2023

* Look at trajectory gifs with new params
  + c still affects z peak
* how do we keep largest input from reactivating its output neuron?
  + Will have to rely on lateral inhibition
  + But also with high u value after convergence that’s not a bad thing
  + Ratio between rho and c is keeping u high
    - That means if c changes, we settle at different place
    - C being larger means neuron remains on for longer
    - That means z/c is going to be smaller, so u settles lower
  + U settles to lower value after longer firing
    - Need u to resist neuron from firing again
* Do we need u to help lateral inhibition or be active in it?
* Possible multiple ring neuron equations:

A black board with math equations

Description automatically generated

* A black background with white text

  Description automatically generated
* Say we do z+c/c – since c is fixed, as long as z active, it affects deactivation. But when z goes to 0, that term goes to 1 (too high)
  + Add some fraction of c to the numerator?
  + Z+gamma \* c / c – like adding a fixed balancing term to the right side of equation
    - When z becomes 0, balances out to gamma
  + If gamma is a function of rho, where it settles down depends only on one parameter
  + Would affect how fast v is rising now that u is decaying slower
  + Slopes of v become shallower
* ICDL used a gate to do this (only shown in thesis)
  + Each output neuron has a gate – when I comes in, it’s effective only while population is active
    - Population was recurrently connected – so self-sustaining activity (after I switched off)
* Could divide I by other ring neurons – interpose a layer between I and ring neurons
  + That middle layer does the inhibiting
* Could put a weight on I – could be z-dependent
* Dendrites can be inhibited by other synapses
* Will the ring neuron be able to sustain activity?
* Multiply I by (1-gamma\*normalized z’s)
  + X / (x+y)
  + When a neuron wins, it inhibits all others by gamma
  + During transitionary period, want the sum of all activity to not exceed gamma?
  + Want a threshold function
    - Contain the inhibition

# September 1st, 2023

A black background with white text

Description automatically generated

A black background with white text

Description automatically generated

A black background with white text

Description automatically generated

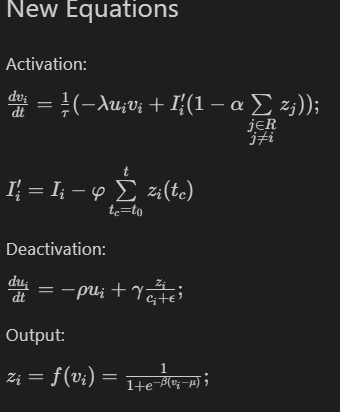
* Check that single neuron still working with z+c/c
* might be deactivating everyone
* Now that +c is added to u equation
* Then if that’s okay, check lateral inhibition term
* Lateral inhibition useless because other neurons z is 0
* Don’t use z for inhibition, use v?
* Some neurons might have access to synapses of other neurons (that’s v)
* In the brain inhibitory and excitatory neurons are different, here we’re using the same ones
* Not biologically realistic
* Could use a different sigmoid for inhibitory
* Part - but that includes new params
* Maybe need to increase alpha
* We wanted to rely on u being high to hold first neuron down long enough that second neuron could activate
* Maybe we shouldn’t normalize by number of neurons in ring
* Expect only one neuron to be active at a time
* Step 0 - check 1 neuron with new u equation
* Step 1 - Get rid of division term in v
* Step 2 - increase alpha (close to 1)
* Step 3 - if that doesn’t help, change z to v
* Step 4 - include i in summation
* We don’t want high I to result in low u settling point
* Ideally want u to settle to 0
* If not possible, want u to settle higher for higher I
* Original paper didn’t rely on other neurons
* Once a neuron i was turned on, I\_i was set to 0
* Because recurrent input from i to i
* Then once z\_i goes back to 0, i remains off
* Implies a second set of neurons
* Last resort: put a phenomenological latch like the original paper
* Could also restrict range of big I
* Can inhibition keep down an I of 1 and an I of 0.1
* If not, we need to reconsider the range of I
* I\_i is already a synaptic input
* 1-alpha says there’s a modulation on that synapse
* Two sources of synapse modulation:
* Synapse has a limited resource (icdl paper)
* Can make resource recovery time super long so things never really reactivate
* Assume subpopualtion of inter neurons which are inhibiting the synapse (GLIEL CELLS?)
* GLIEL cells Receiving input from other neurons
* Could replace sum term with a sigmoid/threshold of the gliel cells
* Assume a mechanism of holding gliel cells output to 1 until deactivation
* Kinda like the latch
* Adds a new equation for new GLIEL cells ?
* Or just put in an ad hoc term for it and say that’s an assumption
* Still have issue of zj not activating quick enough
* What if we include neuron i in the summation
* If u remains high, i won’t get reactivated when everything is inactive
* But don’t want to rely on u because realistically want it to rest at 0
* Lots of dynamics change. Prob won’t get what we want without a +zi term on dv equation
* Then becomes a competition between neuron powering its own activation and deactivation
* Historetic function??
* When inhibition Hits 1, remain at 1

Test single neuron and send email

# September 15th 2023

* Discharging I is not difficult
* Need to be able to keep the neuron down after activation
* Without turning off the I value
* The thesis explains the gating neuron, which is set to 0 once the neuron is turned on
  + Then relies only on internal dynamics to remain active
* Discharging I by cumulative z of the same neuron over time sounds doable maybe?
  + Do we want it to depend on how long the neuron is on over time or not
  + Need a gain param on the cumulative z
  + For neurons that will be active not as long, we are only discharging a smaller value
    - But maybe not true because duration may be long
* Still want the 1 – sum(z) of the other neurons to perform the competition between neurons
  + The one becoming active the fastest should dominate
* AM gone 23-1st

# September 22nd, 2023



* With these equations, is becoming far too negative, and is dependent on the length of integration
  + So, the longer we integrate, the further v drops

A graph of a function

Description automatically generated

* Potential solution: apply a non-steep sigmoid to the cumulative z to ensure doesn’t drop too far negative

A screenshot of a black screen

Description automatically generated

A graph of a number of lines

Description automatically generated with medium confidence

* is still difficult to control because is still dropping to (
* Use 1-sum for I’ eqn to have it converge to 0
  + Put I’ on right side of I’ eqn
  + Amount of decrease of I’ should depend on I’
  + When I’ -> 0, wan no further reduction
* I’(t\_0) = I and decays to 0 while z remains high
  + Time-dependence of I’ is accumulation of z
* Want continuous time equation, not a summation
* Want z to drain I’ until I’ = 0
* ;
  + To make nullcline dependent on length of time
  + If phi same for all neurons, duration wont depend on c
  + Phi must be high enough so neurons can drain fast enough
* Try c in dI’ eqn but still will be tricky
* Why no competition in previous graphs?
  + Because I’ is serving 2 purposes:
    - 1. Vehicle for competition
      2. And resource draining
* Need to make sure alpha high enough so one neuron can stifle others (alpha=0.9ish)
* At the start, all neurons should be competing
  + Highest I activates first
  + Needs to stifle others
  + When next neuron needs to activate, I’ should be shut down
* Maybe we can get rid of c
* Amount of time to get I’ (or z) to a threshold depends on how high I’ started, i.e., I\_i
  + Now I\_i has an effect on duration
  + Keep c in du eqn and try in dI’
* Maybe we can accept the interaction between I\_i and c\_i but it’s not very clean
  + Ideally want duration to only depend on c
  + And have zeroing ot of I\_i’
  + Set phi s.t. it depends on I\_i
    - No matter where I\_i’ starts, it gets down in same amount of time
      1. (with a constant c)
    - Want duration to only depend on c\_i
* Find a new eqn for dI’ s.t. we get duration unaffected by I\_i
  + But prob still need some function of I\_i
  + Or try some fixed phi
  + Of function of phi = g(I\_i)
* Try
  + Will need to refactor doodle so it has I\_i’ in state
  + \* CHECK THIS – should RHS have I’ or I?

# October 6th, 2023

* Changed equations to be accurate to what we are expecting

A screenshot of a black screen

Description automatically generated

A graph of a number of colored lines

Description automatically generated

* 1st order linear diff eq – time constant says how long to deplete to 1/e (?) of something
  + I’ will deplete to a level dependent on I
* Simulate 1 neurons with different I, even when c is the same the neuron will decar at different times
  + I-dependency
* I’ is enforcing competition in a way that’s proportional to I
* Don’t want I to control discharging of neuron
  + This happens with
  + Put I-dependence on u?
    - Scale c or gamma s.t. effect of I is nullified
    - Also give I a role in deactivation
* Increasing c gives increased duration
  + When I higher, scale c s.t. it’s effectively lower
    - Divide c by I?
    - Make gamma depend on I?
      1. When I high, make u increase faster
* Check this in eq’ns by:
  + Add and see if equal-c neurons have same duration
  + Maybe another function of I’?
  + Grid search with more I’s and c’s and with more neurons

# October 13th, 2023

A screenshot of a black screen

Description automatically generated

* Lateral inhibition constant (alpha) has to be pretty high (>= 1.0) to make sure only one neuron active at a time
* If c is low, the short duration leaves I’ with some resource left in the tank so that will result in several short bursts of activations (see green and pink below)

A graph of colored lines

Description automatically generated

* May be helpful to guarantee that I values are distant enough from each other that neurons don’t activate on top of one another
  + Or this behavior can be learned
* Next step: much larger ring and have it create the output drawings
  + Then develop metric for good vs bad drawings
* Goal: system babbles, then gradually learns to produce better drawings
* Thesis – should be 60-70 pages double-spaced, with plenty of figures
  + Ch 1 – intro and goals and motivations
  + Ch 2 – background
    - Motor control, developmental learning, can look at papers Ali wrote and cited to start
  + Ch 3 – math description of model
  + Ch 4 – results
  + Ch 5 – summary and conclusions and future work
  + \* must ask permission from publishers when using figures from another paper \*
    - Except thesis is uncopyrighted so that is okay
    - Asking permission is an automated process
  + Use latex – furqan’s thesis is in latex and ali will be sending the files to use as a template

# October 18th, 2023

* Furqan showed activity as a heatmap
  + 36 neurons x 1000 timesteps of activity
* More rounded transition in drawings:
  + Get momentum in mapping between activity to pen
  + Little kinks between transitions of neurons
  + Want it so all lines are smooth, and we’re learning, say, to draw untangled drawings
  + Can this be a learned behavior?
  + Double check the code under # draw outpu
  + Non-holonomic constraints - continuous closed circuit of the governing parameters, by which the system may be transformed from any given state to any other state
    - If a robot is turning, it has to have a turn radius
* Add momentum at time t-1 to calculation of current direction?
* And have a gain < 1.0
  + With a term with dot prod value from previous step
* Remember particle swarm optimization – that had momentum:
  + velocity(t) = alpha \* velocity(t-1) + beta \* other stuff
* smooth a time series by averaging over the neighborhood, but with diminishing weights for previous values
* or we could define a new vector:
  + discounted running avg of the directions (this could be second term in momentum eqn)
* longer lines may take longer to die out
* y(t) = alpha\* x(t) + (1 – alpha) x(t – 1)
* make inertial term very small
* creating drawings:
  + GIFs are too inefficient for every single run
  + Just draw overtime without savings images
* Next steps:
  + Double check mapping from activation from activity to pen and the overall plotting/scaling/etc
  + Speed up plotting without savings imgs/gifs
  + Replace jupyter output at each step
  + Get table with text showing properly
  + Then add momentum term

# October 27th, 2023

* Try a simple momentum for-loop:
  + Check vectorization (and convolution thing)
  + Try alpha=0.5
* Later on, use RL to penalize little kinks?
* 1st step – civilize drawing network to learn smoother functions, reduce tendency to product less smooth drawings
  + Depends on quality metric
* 2nd step - Now need learn the map
  + Train SOFM to only learn if output drawing is sufficiently good
    - So every neuron in SOFM will correspond to a good pattern in C & D layers
  + The C & D vectors will be inputs into SOFM
  + Can either learn or not learn, or can make it a continuous spectrum based on quality
  + Every SOFM neuron will be tuned to a specific drawing, and nearby neurons will be tuned to similar output drawings
    - Then can try interpolating between nearby/similar drawings to get a combo of drawings
  + SOFM will need to be large
  + SOFM learning will be 2-ways – Code and Duration
  + Say SOFM is 20x20 -> 400 neurons
  + And C = 36 neurons, D = 36 neurons
  + Input vectors will be length 72 for all 400 SOFM neurons
  + And weights will be identical for SOFM -> C/D and SOFM <- C/D
  + During learning, SOFM -> C/D will be present but not active
    - Babbling will start in code layer, create a drawing in output, and if output drawing is good enough, that babbling will be reinforced in the map
  + During inference, give a random input to SOFM neuron and it’ll send signal to C/D to create a similar drawing to active SOFM neuron
  + Then can interpolate SOFM activity to get mixtures of drawings
    - A bump in the map, or 2 different neurons, etc.
* Babbling – may need to limit search space so possible babble inputs are sparse
* Evaluation function – may be hard to find good drawings
  + Or maybe we should accept a lot of drawings
    - This would tax the map’s capacity though
  + So if we make constraints tight, only a few patterns will be learned and map can fit it all
    - Learning from babbles will take longer though
  + Example: no crossing in drawings
  + Example: sharpness of turns
* Next steps:
  + Double check momentum function, why does alpha have such a little effect?
  + Evaluation Function
    - Heavily penalize crossings
    - Probably too tough to enforce the drawing to connect the ends of it
    - Encourage very smooth drawings
    - Simple option:
      1. Include some desired patterns in babbling
      2. And have evaluation function enforce the desired babbles
    - Evaluate entire drawing, but possibly evaluate by going step-by-step
    - Could have sub-objectives:
      1. Step-by-step scoring
      2. How close end is to beginning
      3. Penalize sharp turns
      4. Overall outputted drawing

# November 3rd, 2023

* Verified that vectorization formula is accurate
* Now checking a comparison between alpha=0.1 and alpha=0.9
* How to make curves smoother:
  + Unsmooth curves because switching on and off of neurons has little overlap
  + With high momentum, new neuron’s activation is effectively one step later
  + Try much higher momentum
  + Or maybe make sigmoid less steep
    - But will affect other equations
* With slow drawing (high momentum) we get less drawings:
  + Depends on what we want outputs to be
* Evaluation function:
  + Smooth curves based on Menger curvature (radius of circle between 3 points)?
    - Define a good range of curvature
    - Gaussian-like function centered around some middle value of curvature
  + First developmental step:
    - Draw some simple shapes
      1. Except space of possible shapes is very large
    - Or draw just one shape (e.g. circle)
      1. Reinforce based on if any features show that are similar to parts of a circle
    - Or maybe 2 shapes that are very different (e.g. circle and triangle)
      1. And reward when shape is even close to similar
    - Simplify evaluation process by starting from some specific point and aim for a circle with one specific radius (based on menger curvature)
  + First step:
    - Figure out how many steps needed for a circle given my momentum value
  + Can bias the babbling by what the map is learning, then that closes the RL loop
  + But how to we develop the network beyond better circlces?
  + How do we define actual metric?
    - Require it to draw some approximately closed curve
    - Don’t reinforce if it wanders off
    - Can bias system to avoid wandering around to limit search space
  + What do we need for circle to turn in on itself?
  + Simpler shapes?
    - Averbeck used polygons, not circles
      1. Because they have definite directions of movement
      2. Then it might not be necessary to specify the size of the shape
    - Would require different codes on duration layer, not sequence layer
  + Maybe specify a class of desired shapes rather than one specific shape
  + Suppose we say:
    - Lines that cross are not good
    - Don’t care about what kind of line drawn
    - And then put some value on directional variation
      1. Reward more varied directions, without crossing
    - But what would happen in SOFM?
      1. First, might produce few curves with the good property, those are reinforced
      2. Then start to bias future inputs with those good curves, then more of those curves will start to show up
    - Ideally want SOFM capable of producing many doodles that don’t cross
      1. This will happen without biasing the babbling inputs
    - But with biasing, we might get similar/same doodles in SOFM
    - Or we can reinforce from SOFM to input generator
      1. But then we might learn variations of just one shape
    - What are we enabling here with developmental learning?
      1. Drawing of corners?
      2. Curves?
    - Can we constrain to a certain amount of direction changes?
      1. To promote drawing of corners
      2. But what’s the next stage?
      3. Develop from corners to:
         * Smooth curves following same direction changes
         * Or polygons
    - Make any number of turns, don’t cross, all turns must be in same (clockwise) direction
      1. Shows that a curve can become part of a closed shape
  + Start by activating only a few neurons for babbling trials
    - Limits space of possible patterns
  + Maybe development changes momentum levels
    - Start with low momentum, and get sharp curves
    - As system becomes more complicated, momentum issue should become less important
  + If curve crosses: no learning
    - If it changes directions too much, no learning
    - And only turn on a few neurons in each trial
    - And could constrain durations to start all around one point (plus some noise)
  + Detecting intersections with discrete plotting is tough

# November 10th, 2023

* Checking for intersections:
  + Need all prev points available
  + Therefore the vectors of movements at all timesteps
  + If new vector crosses any of those vectors, then
* Keep collection f all line segments drawn, then check if new line segments intersects any of the old line segments
* Constraints to limit search space:
  + For now, keep durations all the same
    - So give one constant c value
  + Not allowing neurons to re-fire
  + Bias the babbling signal
  + Should we constrain that nearby ring neurons must activate each other?
    - So given active neuron A, the next neuron cannot be more than 30 degrees from it
      1. Motor constraints
  + No crossing
  + Reduce the number of time -> reduce the number of neurons active in an episode
* Example combo:
  + Limit to only 7 rings being active
  + Keep durations the same
  + And the next neuron to activate cannot be more than 2 neurons away from the current
  + This might constrain search space enough give us decent collection of good examples in the babbling stage
* First thing to try:
  + Limit to 5 segments drawn – as a hard rule just for now?
  + Max angular distance between active neurons is 30 degrees or so
  + Set durations to all the same value
  + Then apply the SOFM and see the different specialized short smooth curves reinforced in the map
    - More similar curves closer together in map
  + Then in developmental learning:
    - More neurons in ring, allows finer constraints
* Ultimately want what is learned by the infant to become the basis of something learned later that’s more complicated
* Something to learn?
  + To avoid sharp turns, want sequentially active neurons to be nearby
    - But learning this would take forever
* Have SOFM generate babbles, and then that biases future babbles
  + Removes need for many constraints
  + But still need to specify (un)desired behavior
* Over time, random input into code layer becomes less random and more influence from the code
* Start by randomly generating activity on SOFM, and if that random pattern has a strong connection to the code layer, then that will force the code layer into a similar pattern
  + Plus whatever outside noise we are adding
  + But if the randomly activated neuron on SOFM hasn’t developed a strong connection to the code layer yet, then the pattern will be messy
  + And as we grow code layer/SOFM, then more complex versions of same curves then can be learned
* Extra addition?
  + Whatever output, that is encoded with another SOFM that’s connected to the initial SOFM
    - To close the loop?
  + Maybe not a new SOFM, but something like a convolutional layer?
    - Something that converts image into pattern
  + Or maybe SOFM anyways, since image SOFM can be connected to the motor SOFMs
    - Possibly need to allow multiple hills on the visual SOFM?
* Very powerful idea to use maps to encode things
* And when using convolutional filters, we just specify that it’s convolutional
  + Remember output of convolutional layer is a feature map

# November 20th, 2023

* How to limit to 5 neurons? Based on how much time we run it for?
  + If so, then how to we also limit the angle? Just skip any invalid activations?
* Make nearby neurons more likely to be activated, rather than hard definition
  + Trying to capture that sudden changes in direction/movement are difficult for a body (take a lot of energy)
  + Jerk – is very expensive movement
    - Mimicking this aspect, while also using this to limit our search space
  + Probably want to increase the limit for the number of active neurons if we do this
* Probabilisticness would not be part of the learning
* The learning we’re doing is in SOFM
  + It will learn that when we do a very jumpy combo of neurons, those shapes wont be rewarded (so they’ll not be reinforced)
* Balance between random babbling and deliberate movement:
  + Inputs driving the code to generate movements start out randomly (from SOFM)
    - Initial weights from map to system are random
  + Have a alpha vs (1-alpha) for input from SOFM vs babbling
    - As learning proceeds, babbling importance goes down
      1. Ideally, this would be learned, but maybe we just need to control this
  + Possibly could achieve 1. By saying initially both weights to and from SOFM are random
    - So if we init a random signal to SOFM, we’d get a random shape drawn
* So we will want to gradually decrease the weight of the babbling, and increase the weight of the input from the SOFM in generating better drawings over time
* Still need metric for determining good drawings
* Bidirectional weights to/from map and code:
  + When weights rewarded, the SOFM learns that code pattern
  + SOFM only learns from code layer when reward signal comes in based on the output
    - Or we can use a steep sigmoid for this to get continuous quality of drawings
    - Probably better to allow more gradual/gentle learning
  + So when SOFM receives the reward signal, whatever pattern it sees on the code will get reinforced (scaled by the quality of the drawing outputted)
  + Then what happens to the weights going from SOFM to code layer?
    - Make them identical to the weights from code to SOFM, so they’re chained together
* Output SOFM can potentially come later
  + Output SOFM would become tuned to particular shapes, then connected back to the input SOFM
    - And showing the input SOFM a shape from the output SOFM would cause the system to draw a similar shape
  + Right now the activity is converted into a motor behavior, not a visual signal
    - So ring neurons are only producing movement
    - So the ring pattern does not tell us what shape is being drawn
    - So we would need another system to convert visual output into code
    - Basically would need a simple mirror system
      1. Mirror – activated both by doing an action, and seeing that action being done
      2. Or could have a mirror system provide input into the SOFM
* Easy way to increase complexity - could replace SOFM with one activity peak to have an SOFM with multiple peaks
  + Much more complicated, much more similar to nature