2022-01-27

- Met with Braden about rotation
  - Discussed project based on Cueva and Wei, 2018
  - Also proposed my idea of working on motor timing task
  - Important steps:
    * Read Cueva and Wei, 2018 in detail
    * Think about how to reproduce results from paper

2022-02-02

- DONE Code up simple simulation
- DONE Create GitHub repo for Brinkman lab work
  - https://github.com/conormcgrory/rnn_spatial
- DONE Create conda environment for Brinkman lab work

2022-02-03

- DONE Create notebook showing environment simulation
- DONE Review Python 3.10 page (specifically pattern matching feature): https://docs.python.org/3/whatsnew/3.10.html
- DONE Write movement simulator that reflects at boundary instead of resampling
- Animal motion simulation
  - *Problem*: If variance of Brownian motion is too small, rejection sampling procedure used to enforce boundary takes forever (think about it: if all angles within three standard deviations lead the animal into the wall, then it will take more than 100 samples on average to get one that leads away)
  - *Software design question*: Does object-oriented methodology make sense here? Or should we just use functions?

2022-02-04

- DONE Look to see if anyone has build code for simulating rat motion
  - Did cursory online search; didn't find anything
  - Asked Camille; waiting to hear back

2022-02-05

- DONE Write basic code for PathIntegratorRNN
  - Wrote simple RNN implementation (without noise or decay terms)
  - Need to decide on data format and length for MotionSimulation before doing more work

- DONE Change boundary behavior in simulation to sample from uniform distribution
- DONE Figure out what Cueva and Wei are doing with speed in simulations

2022-02-07

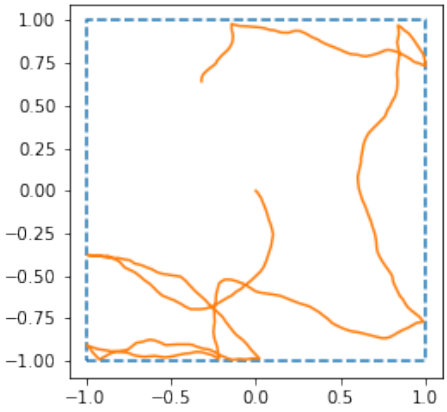
- DONE Write status report for week

2022-02-08

- Met with Braden today
  - Discussed problems determining mechanism Cueva and Wei use for simulating animal trajectories
    - Braden suggested trying run-and-tumble models originally used for bacteria motion
    - Found Python library for this: https://github.com/Deux9/runtumble
  - Asked question about "leak term" in RNN equation from paper
    - Braden says this is totally standard in neuroscience literature, even though it's less common in ML
  - Goals for week:
    - Get network working on current simulated trajectories

2022-02-09

- DONE Add different speeds to trajectory simulations
- DONE Change `RectangleBoundary` to `SquareBoundary`
- DONE Change trajectory from (current direction, current position) to (current direction, next position)
- DONE Write batch optimization routine for spatial RNN
  - Going to first try to use CG method in scipy.optimize instead of Hessian-free approach used by Cueva and Wei
  - If this doesn't work, I'll try Hessian-free package: https://pythonhosted.org/hessianfree/builtins.html
- Got crude trajectory simulation working
  - Angle is sampled from Brownian motion process
  - Speed is sampled from uniform distribution over [0.0, 0.1, 0.2, 0.3, 0.4]
  - When trajectory collides with boundary, angle is sampled from uniform distribution over [0, 2 * pi] until angle is found that respects boundary



*

2022-02-15

- Met with Braden
  - Project at this point has two parts: replicating Cueva and Wei model, and using it to answer scientific questions about spatial navigation
  - Replicating Cueva and Wei Model
    - Figured out general structure of mini-batch approach
    - Need to learn about Hessian-free optimization and implement it
  - Scientific questions
    - Big one: How does RNN perform error correction during path integration?
    - Hardcastle et al.: Boundaries are used for error correction
    - Could this be true of "landmarks" in general?
    - Idea: Train network in environment with one boundary shape (e.g. square) and test in another (e.g. triangle)
    - Idea: Create network that takes landmark information as second input, see if cell responses are similar near boundaries and landmarks
- DONE Read Hessian-free optimization paper
  - Hinton lecture: https://www.youtube.com/watch?v=K2X0eBd-0lc

2022-02-16

- DONE Install PyTorch in conda environment
  - Needed to create new conda env 'spatial-rnn' with python 3.9 to successfully install PyTorch. Once PyTorch supports 3.10, I can switch back to earlier env 'spatial_rnn'
- DONE Implement simple path integration network in PyTorch

  - Resources:
    * https://www.cpuheater.com/deep-learning/introduction-to-recurrent-neural-networks-in-pytorch/
    * https://github.com/gabrielloye/RNN-walkthrough/blob/master/main.ipynb
    * https://blog.floydhub.com/a-beginners-guide-on-recurrent-neural-networks-with-pytorch/
  - Simplified network has no noise or decay terms; otherwise is same as Cueva and Wei network
  - Wrote basic PyTorch code, need to debug

2022-02-21

- DONE Write script to generate trajectory data

2022-02-22

- DONE Write status report

2022-02-23

- DONE Refactor simulation code
- DONE Add code for comparing predicted motion to actual motion
  - Git commit: 08516f6

2022-02-25

- DONE Write Dataset class for simulation data
  - https://pytorch.org/tutorials/beginner/basics/data_tutorial.html
  - http://www.feeny.org/custom-pytorch-dataset-class-for-timeseries-sequence-windows/
  - Commit: 679ac4a

2022-03-01

- Meeting with Braden
  - Explained that RNN is not training successfully on data
  - Talked about strategies for getting it to work
  - Need to do some research to see if path integration/dead reckoning is something people have trained RNNs to do previously (I'm assuming yes?)
  - Talked about ResNet (residual network architecture)
    * Need to see if PyTorch has a ResNet option
- DONE Try to get code working

  - Things to try
    * Make motion slower and straighter
    * Increase number of trials (need to check original paper for their numbers)
    * Change speed distribution to weight zero more heavily
    * Check to see if weights are blowing up
    * Check parameter initialization in RNN
    * Check to see if we can add leak term
    * Make sure there are bias components to input
  - Wrote scripts! Commit: `92a1399`
- DONE Research RNN path integration
  - Found one interesting Ganguli paper that replicates Cueva and Wei results: Sorscher et al., 2019
- DONE Write status report

2022-03-05

- DONE Get access to server
  - Messaged Braden; need to talk to Tushar
  - Got server access using old account (from Park Lab rotation)

2022-03-08

- DONE Set up cluster for RNN training
  - Got cluster access
  - Installed `conda` on cluster and created environment with CUDA enabled
- DONE Write simulation and training scripts for cluster and test locally
  - Wrote and tested scripts!
  - Commit: `92a1399`

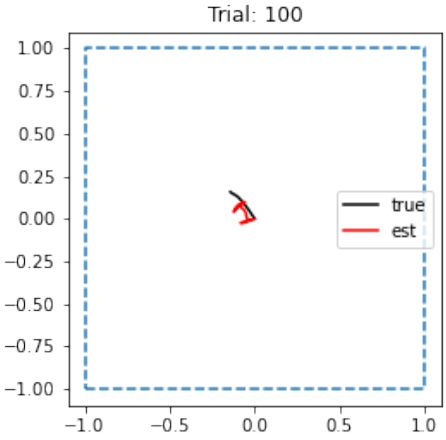
2022-03-10

- DONE Check that RNN code is using Float32 instead of Float64
  - Checked code – for both input and output tensors, dtype is 'float32'

2022-03-25

- DONE Re-run RNN with altered speed distribution
  - Changed code (commit: 64d999e)
  - Ran simulation successfully
  - Kicked off RNN training
  - Finished RNN training

    * Loss isn't very low, but trajectories aren't long enough to see whether network has learned path integration better or worse
    * Need to rerun with larger time step

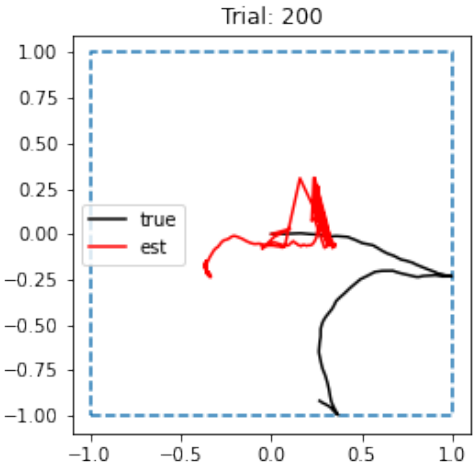


    *

- DONE Send email to Xiuxin
  - Sent email; CC'd Braden

2022-03-29

- DONE Run RNN training with altered speed distribution, but longer time step
  - Changed timestep from 0.01 to 0.1 – this makes the trajectories long enough to hit boundary at least once
  - Ran trajectory simulation
  - Started RNN training
  - Finished RNN training
  - Network not learning path integration task at all – unclear what's wrong here



–

2022-03-30

- DONE Look at code from Chris Cueva
  - Code is written in MATLAB and well-documented
  - Code has a lot of flexibility: multiple nonlinearities and loss functions are supported
  - Hessian-free optimization is coded up manually (`conjgrad.m`)
  - Trajectory simulation code is in `generateINandTARGETOUT.m`
    - What's this? http://www.walkingrandomly.com/?p=2945
    - Angle and speed seem to be chosen similarly to what we're doing, with some slight differences at boundary (instead of uniformly sampling, they keep Gaussian sampling of angular velocity, but increase variance if a certain number of samples fail to get the animal back in boundary)
    - NOTE: Gaussian noise is added to input (was this mentioned in paper?)
  - RNN code is in `rnn.m`
    - Weight matrices are initialized more carefully than what we're doing
    - Look at orthogonal random matrix initialization from Saxe et al., 2014 – this apparently speeds up training at same level as layer-wise pretraining
    - During training, they also use gradient clipping (Pascanu et al., 2013)
    - Training is very complicated! They seem to be using a damping parameter?
    - Question: Why are they computing certain steps on the entire

training set? Didn't the paper say they were using minibatches?

2022-04-01

- Meeting w/Braden
  - Interesting observation about code: Forward pass equation seems to be missing factor of sqrt(dt); this hasn't caused issues because dt=1
  - Goal 1: Run Chris's code; make sure it works
    - Try zeroing out network noise (`bhneverlearn`)
    - If this doesn't affect result, then we can ignore it in our model
    - We want to SIMPLIFY!!!
  - Goal 2: Port over important features from Chris's code to ours
    - Weight initialization
    - Look up: Gaussian orthogonal ensemble (also look at paper)
    - ResNet component
    - Add noise to input data
    - Add noise to network
  - Note: Good SDE book: Van Kampen

2022-04-03

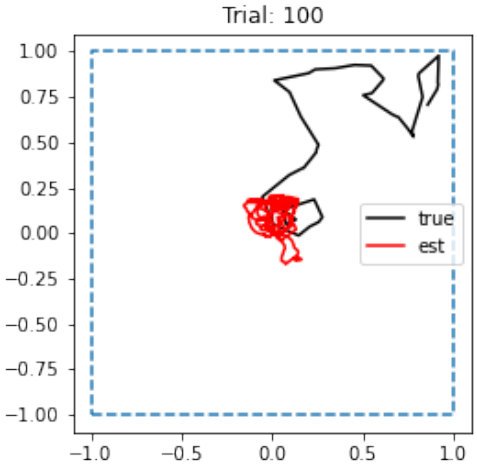
- DONE Figure out how RNN function works in PyTorch
  - Main obstacle: Need to understand how to implement `forward()` method for RNNs – examples online don't make sense
- DONE Add weight initialization to PyTorch code
  - Orthogonal weight initialization is supported in PyTorch! (`torch.nn.init.orthogonal_`)
  - Sparse weight initialization also supported1 (`torch.nn.init.sparse`)
  - According to paper, initialization used for recurrent weights W^{rec} doesn't affect results
    - However, they used different initializations for each environment: zero-mean Gaussian initialization for hexagonal environment, and **orthogonal initialization for square and triangular environments**
  - Bias and output weights W^{out} were initialized to zero
  - According to paper, input weights W^{in} were initialized as zero-mean Gaussians with variance $1/N_{in}$
    - However, this **isn't** what they're doing in the code!
    - Need to read code and check commented-out lines to see what's going on here
    - Still haven't looked at this! Just using Gaussian initialization for now
  - Commit: `446553a`
- DONE Add noise to inputs in PyTorch code

  - Checked code; turns out they aren't actually adding noise
  - Variable `noisamplitude_input` is set to zero, so no noise is added
- DONE Set trajectory parameters in PyTorch code to be equal to those from Chris's code
  - Number of timesteps (`numT` in their code, `N_STEPS` in ours): 450
  - Number of trials per epoch (`numtrain` in their code): 500
    - Code samples 500 trials for each "epoch"
    - This is the mini-batch that they use to compute the gradient and perform an update
    - They also sample test trials, but they don't seem to use these to perform update
  - Number of epochs (`numepoch` in their code): 1900
    - This is not the same as our idea of epochs
    - In their training routine, they only train on each batch once
  - Number of total trials(`N_TRIALS`): 500 * 1900 = 950000
  - Boundary is square with `x_min, y_min = -1` and `x_max, y_max = 1`
    - In our code, this is copied by setting `BOUNDARY_TYPE = 'square'` and `BOUNDARY_HEIGHT = 2.0`
  - Density of non-zero speed values (`density` in their code): 0.1
    - Speed values are sampled using `sprand()` function and then divided by 5
    - I'm assuming this is some sort of sparse uniform sampling function?
    - Full command: `speed = sprand(numtrials, numT, density)/5`
    - Need to figure out how to do this in our code
  - Angle is updated using commands (approximate):
    - `angularvelocity = randn() / 3`
    - `angle = angle(t-1) + angularvelocity`
    - In our code, this same operation is done by the `smp_direction_step()` method
    - Equivalent would be setting `std_brownian = 1/3`
    - This means that `np.sqrt(time_step) * std_norm` has to equal 1/3
  - Position is updated by adding speed to current coordinates
    - This is equivalent to setting `time_step = 1`
    - Therefore, `std_norm` has to be equal to 1/3

2022-04-05

- DONE Figure out how to modify `MotionSimulation` class to generate batches for training
  - Worry about test set later
  - Might need to make `MotionSimulation` a subclass of PyTorch's `Dataset`?

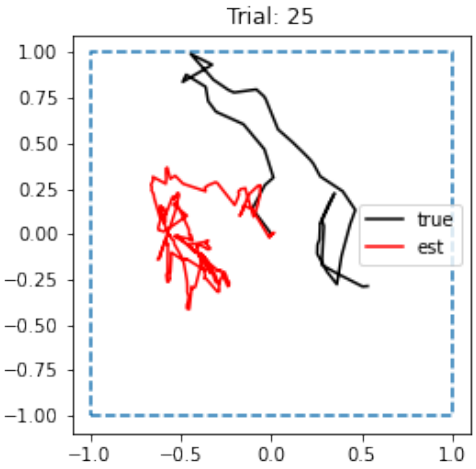
        * Decided not to do this – doesn't work well unless all trials are stored somewhere
    - Commit: `25d52b1`
- DONE Think about removing `boundary_height` parameter from boundary objects in code
    - Thought about this, but might table for now because if it's removed, there won't be any clear reason why `SquareBoundary` should even be a class instead of just a single function, and I don't want to do any more refactoring
- DONE Test RNN with new parameters and training routine
    - Started training run on Dirac
    - Run finished. Loss didn't decrease at all; not sure why



    -

2022-04-08

- DONE Try training network with velocity changed to Cartesian coordinates
    - Commit: `7d41fc9`
    - Checked trajectories: Still make sense
    - Kicked off training run on cluster
    - Finished training. Loss decreased, but not by that much
    - Created test dataset in Cartesian coordinates: `data/sim_2022_04_12`
    - Performance did not seem better than test with polar coordinates



–

2022-04-11

- DONE Run through RNN code to make sure I'm not making any obvious errors
  - Checked order of inputs/outputs, `batch_size=True` setting
  - No obvious mistakes

2022-04-12

- Met with Braden
  - Two next steps
    - Run MATLAB code
    - Get `forward()` method working

2022-04-13

- Downloaded RNN training run with learning rate set to 0.01 instead of 0.001. Didn't work very well; loss didn't decrease as much as previous run
- Found other codebases for grid cell simulation:
  - https://github.com/ganguli-lab/grid-pattern-formation
  - https://github.com/neuroailab/mec
  - Need to look at these to see what they're doing
- DONE Run MATLAB code from Chris
  - Code works, but is taking an extremely long time on my laptop
  - Need to figure out how to run it on cluster
    - Cluster has MATLAB

        * Need to replace plot commands with commands to save figures to file
- DONE Look at Python codebases for problem
  - Codebase 1: https://github.com/ganguli-lab/grid-pattern-formation
    * Written in PyTorch
    * One key difference: Hidden units are being initialized to zero
    * `batch_first` option in RNN isn't set to `True`
    * This doesn't make sense!
    * Another difference: Rayleigh distribution is used for velocity
  - Codebase 2: https://github.com/neuroailab/mec
    * More general framework from OpenAI
    * Uses TensorFlow

2022-04-14

- Double-checked RNN code; weight initialization made no difference
- DONE Initialize hidden weights to zero
  - Added code for initializing hidden units
  - Added L2 regularization on all weights
  - Removed bias for input, recurrent and output weights
  - Started training network on cluster
  - Trained model still had biases – not sure why. Might have been problem with git push/pull order. Trying again.
  - Downloaded model. Need to check if biases are still present.
  - Biases were removed for model (`test_2022_04_14.pt`), but performance was still bad. Going to try to increase speed density next.

2022-04-15

- DONE Run RNN with polar input and speed set using Rayleigh distribution

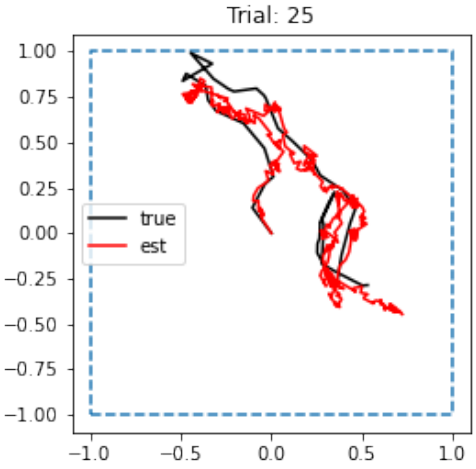
2022-04-18

- DONE Run Ganguli code
  - Tried setting up locally – didn't work. Code needs to be run with CUDA on cluster.
  - Need to create environment for code on cluster
  - Can currently run code via `main.py` script on command line
  - Should eventually figure out how to connect to Jupyter server running on cluster
  - Initial run failed – `main.py` was trying to import `LSTM` from `model.py`
  - Second run failed – `num_epochs` argument not recognized
  - Third run failed – `Trainer` was looking for a missing file in `models` directory

  - This code does not seem up-to-date

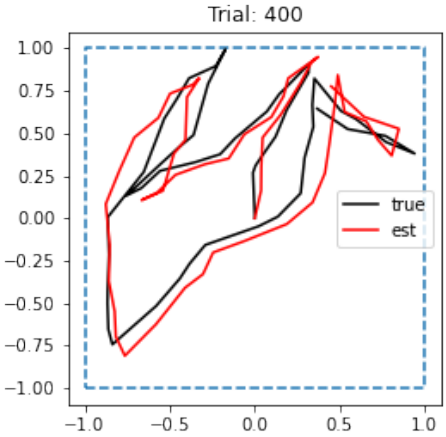
2022-04-19

- DONE Check that simulation data is being generated correctly
  - Wrote notebook (`test_data.ipynb`) that checks that generated trajectories can be integrated
  - Commit: `[main 6f3a369]`
- DONE Change trajectory simulation to use Rayleigh distribution for speed and re-run simulation (Cartesian)
  - Make sure to check Ganguli's code to make sure I'm using roughly the same parameters as he is
  - Commit: `[main b072ea2]`
    - **This commit actually didn't correctly implement sampling!** Use this one instead: `[main 3e8d7d9]`
  - Didn't make significant difference in loss
- DONE Run RNN code with learning rate set to 1e-4 (this is what Ganguli used)
  - NOTE: This code was using the original speed sampling protocol, **not** the Rayleigh distribution
  - Commit: `[main 050fcbd]`
  - Model: `test_2022_04_19_02.pt`
  - Wow! Even after 5 epochs, effect on loss is amazing!
  - Training loss decreased almost to zero!
  - Next step: Run on polar coordinates, make trajectories more smooth

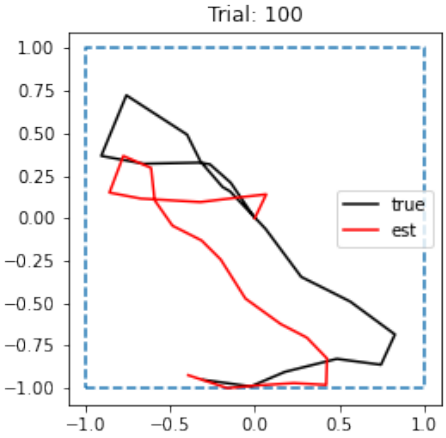


  -
- DONE Run RNN with gradient clipping
  - Maybe putting this back will help?
  - This appeared to help slightly when I tried it on my local machine

  - Want to look more into this later
- DONE Run RNN with new learning rate on polar input with sampling from Rayleigh distribution
  - Commit: `[main 3e8d7d9]`
  - If this doesn't work, I should revert to the old model of speed sampling and make sure lowering learning rate improves performance on polar trajectories too. After that works, I can then figure out how to tune the Rayleigh speed sampling to make that work.
  - Loss decreased significantly, and network learns path integration pretty well!

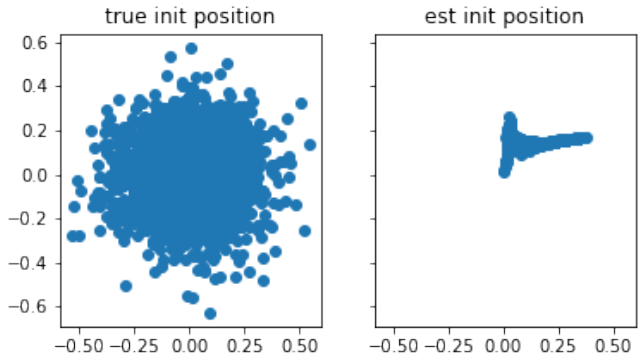


    *
  - However, network can sometimes get integration wrong at beginning of path:
    * Could this be because of hidden state initialization?



    *
  - Found bias in initial position!

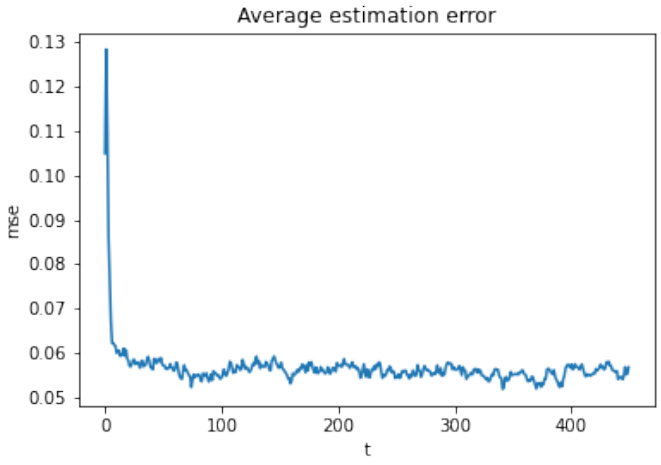
    * Why is this happening?
    * Maybe answer is to start all sequences with zero velocity, zero position entry?



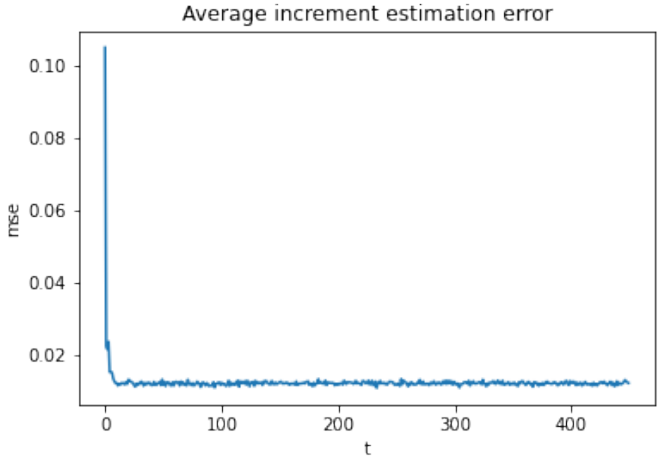
    *

2022-04-20

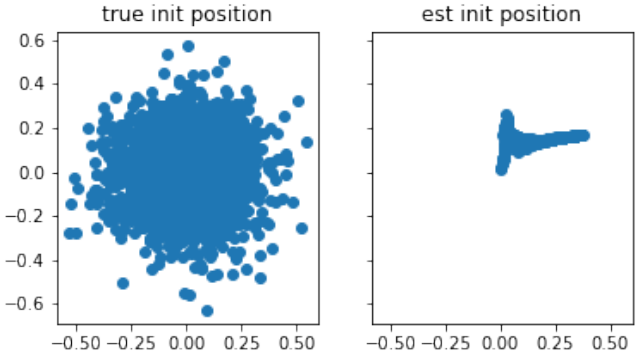
- DONE Figure out why network messes up initial position
  - The problem: Average error is extremely large for the first couple of trajectory points



    *
  - By looking at the error between the increments at each time point, we can see that **the first time point is the major problem**:

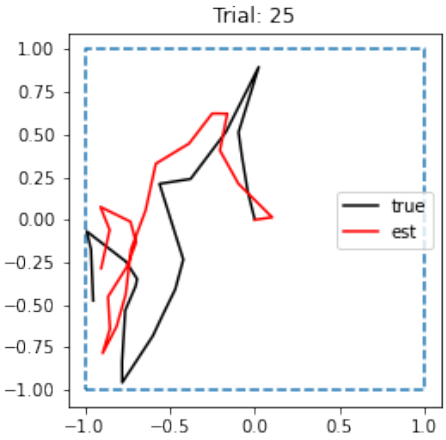


  - This is due to a significant bias in the first time point:

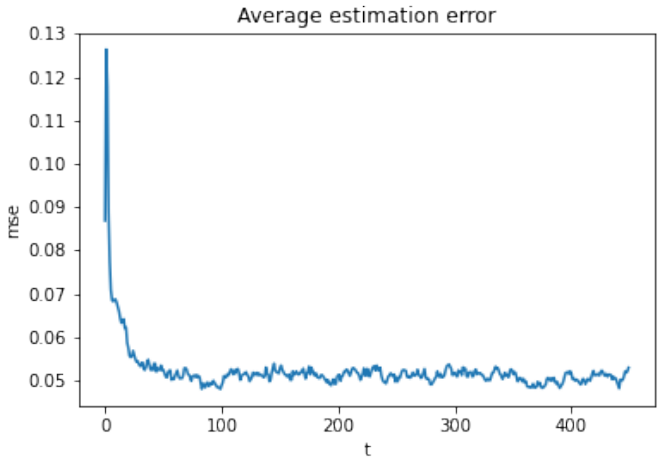


  - Possible answer: Initializing hidden unit activities to zero causes problem
    - Maybe adding a zero-velocity entry at the beginning of each trajectory would help this?
  - Another idea: Lack of bias terms causes problem
    - Checked Cueva and Wei, 2018: in model, recurrent units have biases
    - Maybe adding these back in will help?
  - Another idea: Polar coordinates cause problem
    - Evidence: This doesn't occur with Cartesian velocity inputs
- DONE Run RNN with bias term added to hidden layer
  - To match Cueva and Wei paper, bias needs to be initialized to zero
  - PyTorch RNN only has option for adding bias to input and recurrent

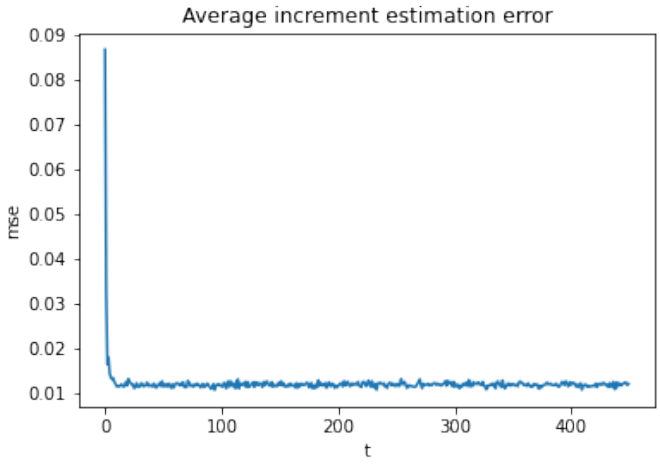
weights, so I have to add both
- Commit: `[main e3e3de6]`
- Launched on cluster
- Model slightly improved, but not by that much



*
- Avg. error

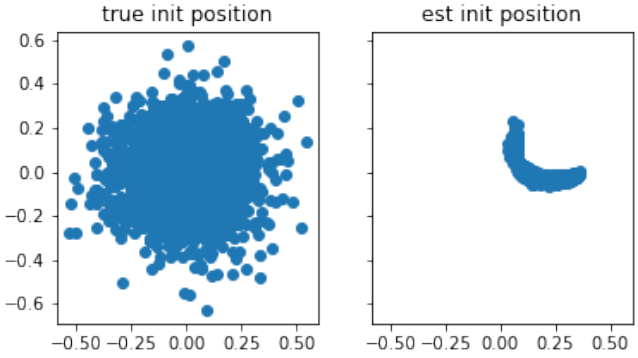


*
- Incremental error



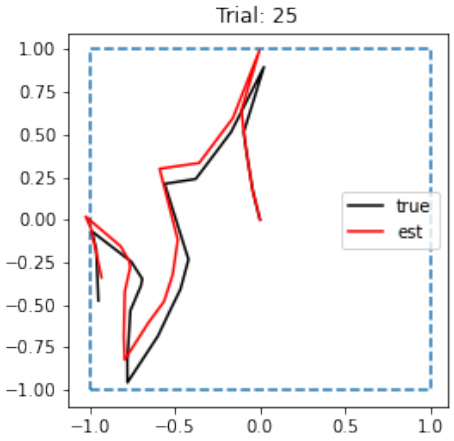
*

- Init position



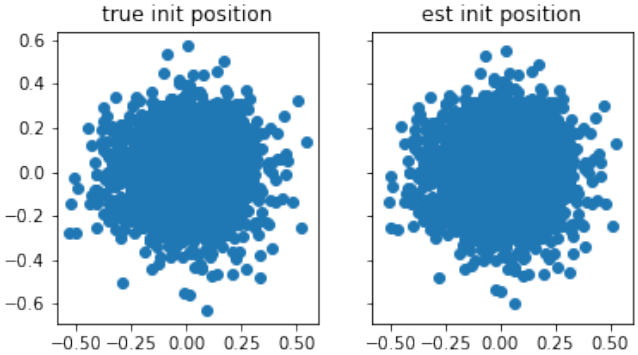
*

- DONE See if errors on first timestep are a problem with Cartesian coordinates
  - Commit: `[main da2ff09]`
  - Launched on cluster
  - Errors on first timestep are **not** a problem with Cartesian coordinates
  - Overall performance is better:



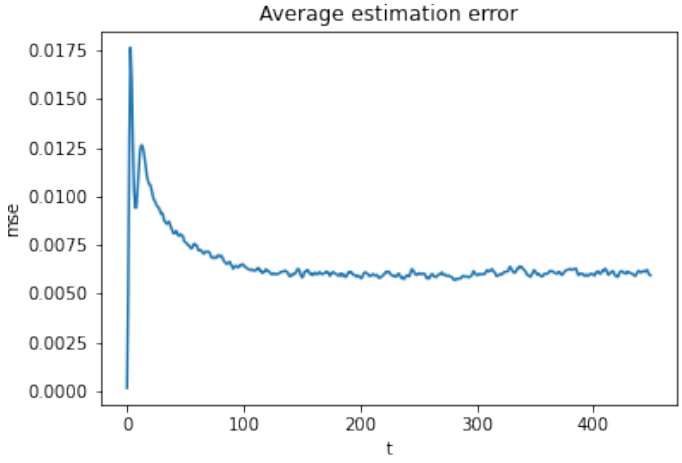
*

– No noticeable bias in initial position:



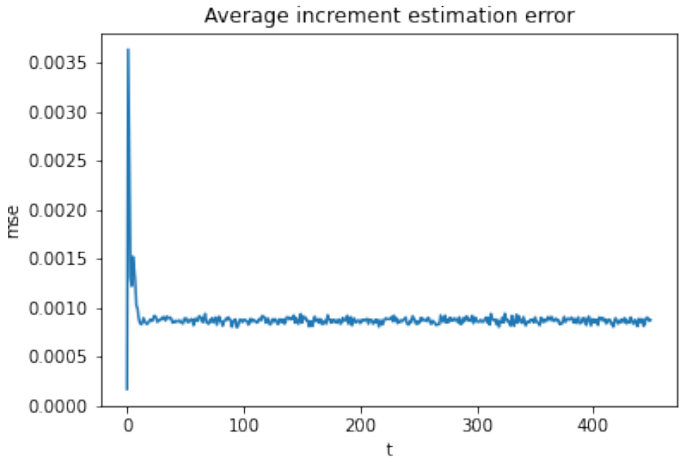
*

– Initial avg. error and incremental error are much smaller:



*

- Incremental error



*

2022-04-21

- DONE Refactor RNN codebase
  - Finished!
  - Commit: `[param-refactor 1032b1d]`

2022-04-22

- DONE Compute activations of hidden units in RNN
  - Trying to adapt Ganguli lab code: https://github.com/ganguli-lab/grid-pattern-formation/blob/master/visualize.py

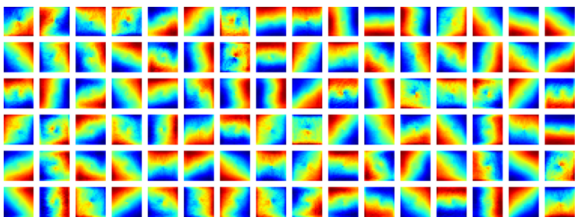
  - Also could use scipy function: https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.binned
  - Adapted code (still need to clean up)

2022-04-25

- DONE Clean up cell activity plotting code
  - Commit: `[param-refactor 1391e1c]`
  - Still need to simplify `compute_ratemaps()` function more

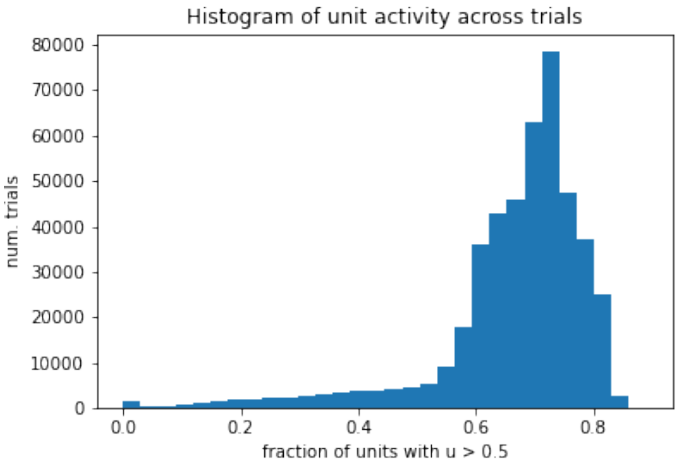
2022-04-26

- Meeting with Braden
  - Idea: Try training the network for longer, see if grid cells emerge
  - Question: Why are Ganguli et al using cross-entropy loss?
    * Idea: Cross-entropy loss might perform the same function as the metabolic penalty from Cueva and Wei does
  - Roadmap:
    1. Run current code for more batches; see if error goes down further and grid cells emerge
    2. Clean up grid cell plotting code adapted from Ganguli
    3. Add metabolic penalty to loss function
    4. Try to get both Ganguli and Cueva and Wei code to run
- DONE Run RNN training for 3000 batches
  - Commit: `[param-refactor 88d598e]`
  - Started run on cluster
  - Finished! Downloaded model: `test_2022_04_26.pt`
  - Loss was still decreasing at batch 3000; going to try to run for 5000 batches
  - Final loss: 0.0106
  - Ratemaps have clearly different structure:

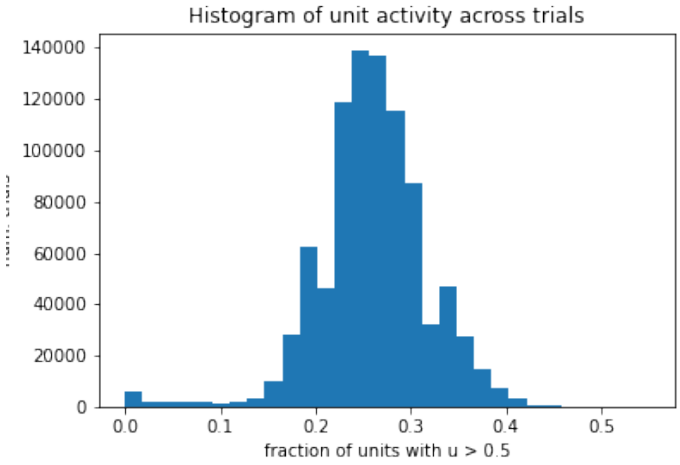


    *
- DONE Add metabolic penalty to RNN loss function
  - Going to need to replace off-the shelf loss with function written from scratch
  - Loss function from Cueva and Wei
    * Component 1: Mean squared error, with average taken over number of points (num. trials * num. points per trial * 2)

        * Component 2: L2 regularization on input and output weights (not recurrent weights!)
        * Component 3: Metabolic cost, computed as mean squared activation across time and num. batches
    - It seems that they're also doing contracting autoencoder (CAE) regularization on the recurrent weights in the code
        * Not mentioned in the paper!
        * Need to figure out what this means, and if it's necessary
    - Regularization parameters
        * Weights (L2): 0.5 (`rnn.m`, line 145)
        * Metabolic (h): 0.1 (`rnn.m`, line 152)
- Sparseness of hidden unit activity in RNN
    - Looked at activity of hidden units in trained RNN
    - Question: Are these representations sparse? Are only a few hidden units active at any one time, or are they all active at once?
    - Plotted histogram of fraction of hidden units whose activity (absolute value of output) exceeded 0.5 for model trained on 1900 batches (`models/test_2022_04_26.pt`)
        * This indicates that most hidden units are active most of the time; **hidden representation is not sparse**

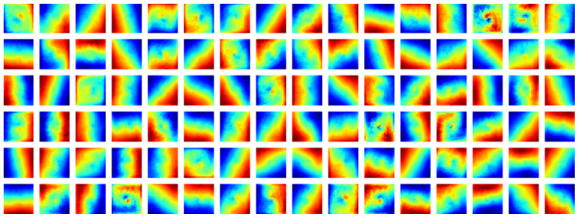


        *
    - Also plotted same histogram for model trained on 3000 batches (`models/test_2022_04_26_02.pt`)
        * Unit activities for this model are **more sparse than model trained on fewer batches**



*

- DONE Run RNN training for 5000 batches
  - Commit: `[param-refactor 9b7a2d6]`
  - Launched on cluster
  - Finished. Final loss: 0.0051
  - Loss still decreasing! Going to try to run for 8000 batches next
  - Hidden units have similar tuning as 3000-batch run:

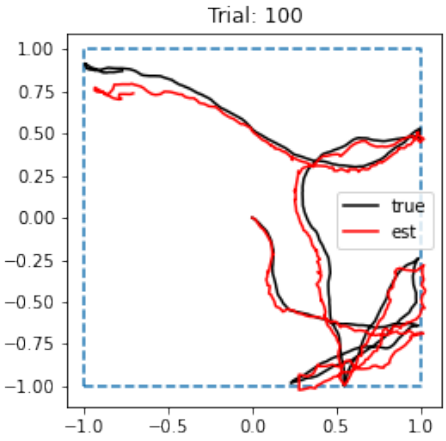


*

- DONE Add parameter saving to `train_model.py`
  - Need to implement `save_model()` and `load_model()` functions in `model.py`
  - Commit: `[param-refactor 0d956ca]`
- DONE Simplify `compute_ratemaps()` function to only use one batch

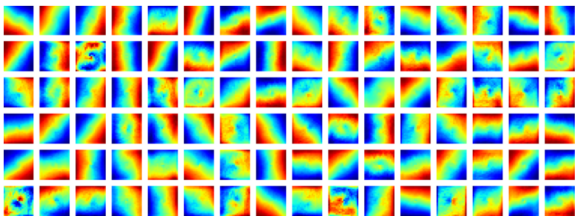
2022-04-27

- DONE Run RNN for 8000 batches
  - Commit: `[param-refactor 0d956ca]`
  - Launched on cluster
  - Loss seemed to stop decreasing! Final loss: 0.0028
  - Performance:



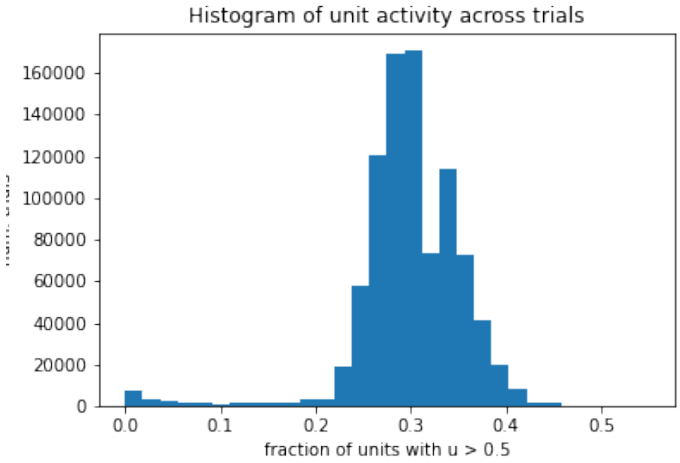
*

– Tuning:



*

– Activity:



*

- DONE Update old parts of codebase
  - To update:
    - `examine_rnn_model.ipynb`

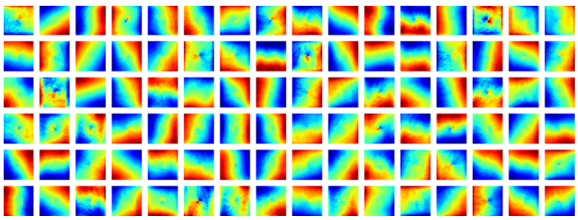
    * `motion_simulation.ipynb`
    * `sample_trajectories.ipynb`
    * `test_data.ipynb`
    * `test_motion.ipynb`
  - Renamed a bunch of notebooks
  - Updated initial position bias notebook
  - Updated trajectory generator notebook
  - Commit: `[param-refactor 91c92aa]`
- DONE Merge `param-refactor` back into `main`

2022-04-28

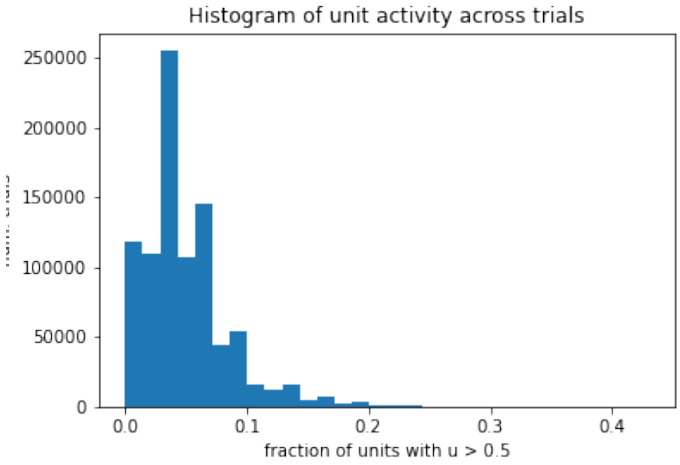
- TODO Ask Josue about "boomerang" posterior
  - This is what I'm seeing for initial position in polar coordinates
  - How are normal distributions combined to create this?

2022-04-30

- DONE Run RNN training with new loss function
  - Commit: `[main 6671917]`
  - Launched on cluster!
  - Successfully ran. Loss was still decreasing after 5000 batches
  - Performance same as model with old loss function
  - Tuning is very similar to model with old loss function:
    * No grid-like tuning structure



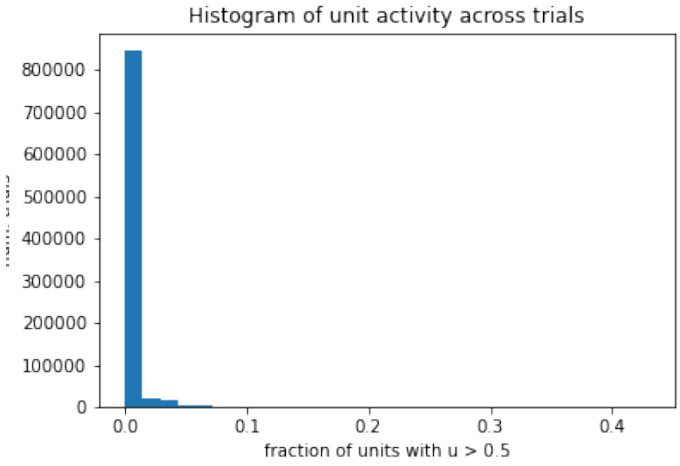
    *
  - Activity histogram shows (as expected) shift to the left:



*

2022-05-04

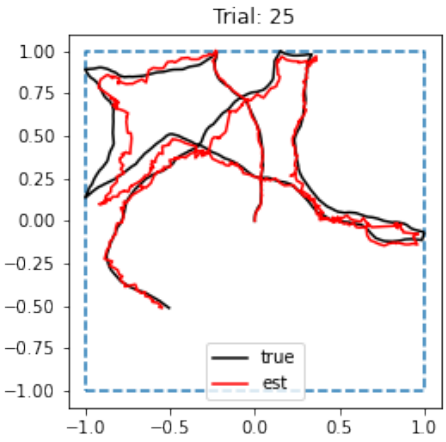
- DONE Run training with new loss function for 8000 batches
  - Launched on cluster
  - Finished. Loss seemed to converge
  - Tuning and performance are unaffected
- DONE Train RNN with $\lambda_h = 0.5$
  - Launched on cluster
  - Finished. Loss might have still been decreasing.
  - Performance and cell tuning unaffected
  - Cell activity histogram is more decreased



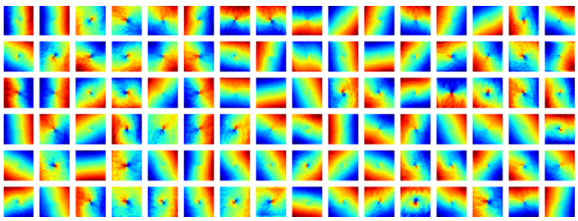
*

  - Need to increase λ_h more to see if this is the cause
  - Because performance isn't degraded, λ_h can be increased further

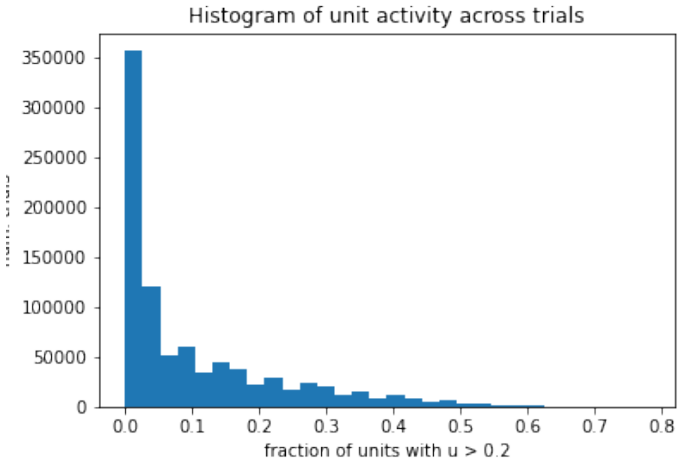
- DONE Train RNN with $\lambda_h = 1.0$
  - Launched on cluster
  - Finished running. Loss definitely converged.
  - Model stored at `models/20220510_01`
  - Performance seems normal



    *
  - Tuning is same as before



    *
  - Activity is sparse

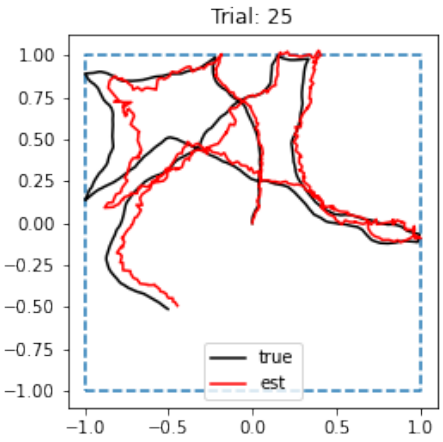


*

  - Conclusion: Need to increase λ_h even more!
- DONE Add functionality to `Trainer` class that computes MSE for each run
- DONE Save MSE curve for each run in `runinfo.json` file

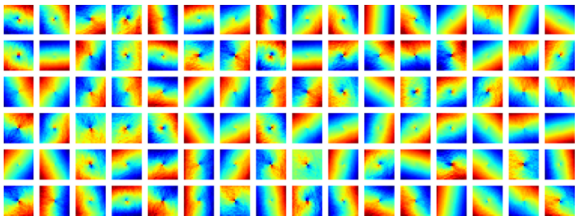
2022-05-12

- DONE Train RNN with $\lambda_h = 2.0$
  - Run finished. Loss clearly converged.
  - Performance is normal



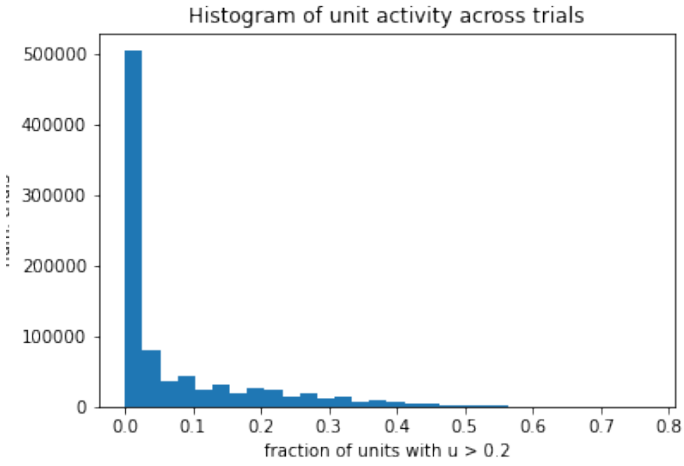
*

  - Tuning is the same



*

- Activity is even more sparse



*

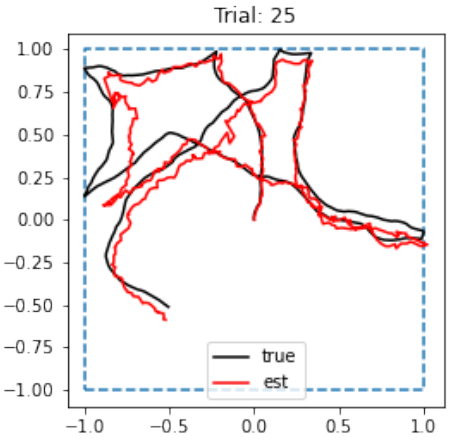
- Need to run with even greater metabolic penalty

- TODO Add CUDA acceleration to RNN training code
- TODO Add MSE trace to `Trainer` class
- DONE Add network checkpoints to `Trainer` class
- TODO Sanity check: Make sure I'm not using fewer data points than Ganguli code
- TODO Sanity check: Test performance on single trajectory rotated 90 degrees four times
  - This is to make sure theta is symmetrical
- TODO Add different boundary collision handling to code
  - Direction angle continues Brownian motion process uninterrupted
  - If current speed and direction will take agent outside of boundary, speed is set to zero
- TODO Compute RNN performance as function of trajectory timestep and see if this correlates with boundary collisions (or smoothed version of them)

2022-05-14

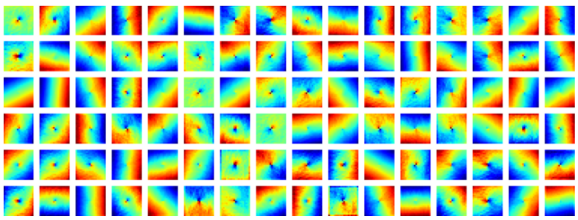
- DONE Run RNN with $\lambda_h = 4.0$

– Run finished. Loss seemed to converge.
– Performance is normal:



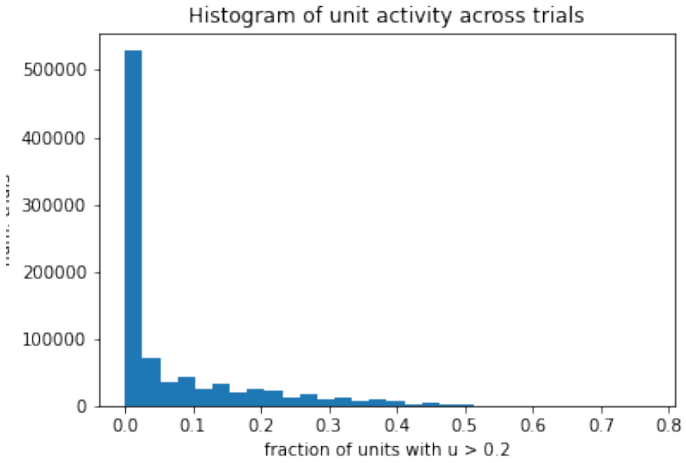
*

– Tuning is the same as before, with more pronounced center in fields:



*

– Activity



*

– Need to increase λ_h more!

2022-05-20

- DONE Plot coverage of trajectory in `inspect_model_tuning.ipynb`
- DONE Try with $\lambda_h = 100$
- DONE Add argument parsing
- DONE Save loss values
  - Maybe both λ_h and λ_w need to be increased jointly to prevent compensation by network?

2022-05-30

- DONE Write overview notebook
- DONE Send email to Braden
- TODO Add project notes to `rnn_spatial` repo