STEP ONE: CREATE V1 NETWORK WITH REALISTIC POPULATION PROBABILITIES

The code for V1 and LGN networks (and corresponding connections) are in homework.py

V1 Network: this is a network of 10,000 LIF neurons, with four populations (excitatory, SST, PV, and VIP). 85% of the neurons are excitatory, and 15% are inhibitory. Of this 15% inhibitory, 4.4% are PV, 3.2% SST, 7.4% are VIP. These population fractions were based on the fractions in the fractions seen in L2/L3 here.

- 85% excitatory; 15% inhibitory
- Based on L2/L3 of the paper → 4.3% PV 3.1% SST 7.4% VIP (=14.8%)
    - To make it ~ 15%, adding .1 to 4.3 and 3.1

STEP TWO: CONNECT V1-V1 NEURONS WITH REALISTIC CONNECTION PROBABILITIES

V1 Network Connections: the neurons of the V1 network were then connected via add_edges function, with distance_connector used as its connection rule. The params file for each neuron pair is instantaneous (instantaneousInh.json when the source is an LIF inhibitory population and instantaneousExc.json when the source is the LIF excitatory population). The values for the delay parameter were chosen based on the values seen in the Systematic Integration paper's **v1_v1_edge_models.csv**. Below is a table summarizing all the edges, and the parameters used. Below the table is documentation of the connection rule used.

TAR/SRC	EXC (pyr)	PV	SST	VIP
EXC	D_weight_min:**0.0345** D_weight_max:**0.107** D_max: Nsyn_min: **3** Nsyn_max: **7** Delay:**1.6** Params_file:**instantaneous.json**	D_weight_min:**0.2645** D_weight_max:**0.54** D_max: Nsyn_min:**3** Nsyn_max:**7** Delay:**1.2** Params_file:**instantaneous.json**	D_weight_min:**0.21** D_weight_max:**0.44** D_max: Nsyn_min:**3** Nsyn_max:**7** Delay:**1.5** Params_file:**instantaneousExc.json**	D_weight_min:**0.09** D_weight_max:**0.3** D_max: Nsyn_min:**3** Nsyn_max:**7** Delay:**1.5** Params_file:**instantaneousExc.json**
PV	D_weight_min:**0.2** D_weight_max:**0.46** D_max: Nsyn_min:**3** Nsyn_max:**7** Delay: **0.9** Params_file:**instantaneousInh.json**	D_weight_min:**0.28** D_weight_max:**0.47** D_max: Nsyn_min:**3** Nsyn_max:**7** Weight_max:**0.0034** weight_sigma:**50.0** Delay: **1.6** Params_file:**instantaneousInh.json**	D_weight_min:**0.03** D_weight_max:**0.18** D_max: Nsyn_min:**3** Nsyn_max:**7** Weight_max:**0.0017** weight_sigma:**50.0** Delay:**1.2** Params_file:**instanteneousInh.json**	D_weight_min:**0** D_weight_max:**0.06** D_max: Nsyn_min:**3** Nsyn_max:**7** Delay:**1.2** Params_file:**instanteousInh.json**
SST	D_weight_min:**0.125** D_weight_max:**0.353** D_max: Nsyn_min:**3** Nsyn_max:**7** Delay:**1.5**	D_weight_min:**0.0763** D_weight_max:**0.27** D_max: Nsyn_min:**3** Nsyn_max:**7** Delay:**1.5**	D_weight_min:**0.009** D_weight_max:**0.08** D_max: Nsyn_min:**3** Nsyn_max:**7** Delay:**1.5**	D_weight_min:**0.21** D_weight_max:**0.56** D_max: Nsyn_min:**3** Nsyn_max:**7** Delay:**1.5**

	Params_file:instantaneousInh.json	Params_file:instantaneousInh.json	Params_file:instantaneousInh.json	Params_file:instantaneousInh.json
VIP	D_weight_min:**0.01** D_weight_max:**0.14** D_max: Nsyn_min:**3** Nsyn_max:**7** Delay:**1.5** Params_file:**instantaneousInh.json**	D_weight_min:**0.0** D_weight_max:**0.09** D_max: Nsyn_min:**3** Nsyn_max:**7** Delay:**1.5** Params_file:**instantaneousInh.json**	D_weight_min:**0.05** D_weight_max:**0.31** D_max: Nsyn_min:**3** Nsyn_max:**7** Delay:**1.5** Params_file:**instantaneousInh.json**	D_weight_min:**0.0** D_weight_max:**0.045** D_max: Nsyn_min:**3** Nsyn_max:**7** Delay:**1.5** Params_file:**instantaneousInh.json**

```python
def distance_connector(source, target, d_weight_min, d_weight_max, d_max,
nsyn_min, nsyn_max):
    # Avoid self-connections.
    sid = source.node_id
    tid = target.node_id
    if sid == tid:
        return None

    # first create weights by euclidean distance between cells
    r = np.linalg.norm(np.array(source['positions']) -
np.array(target['positions']))
    if r > d_max:
        dw = 0.0
    else:
        t = r / d_max
        dw = d_weight_max * (1.0 - t) + d_weight_min * t

    # drop the connection if the weight is too low
    if dw <= 0:
        return None

    # filter out nodes by treating the weight as a probability of connection
    if random.random() > dw:
        return None

    # Add the number of synapses for every connection.
    tmp_nsyn = random.randint(nsyn_min, nsyn_max)
    return tmp_nsyn
```

Documentation:

This function is passed as the connection rule when creating edges between V1-V1 nodes. It connects the source and target neurons by calculating a weight (based on the distance b/w source and target) that is treated as probability of connection.

Parameters:

- D_weight_min: min probability of connection
- D_weight_max: max probability of connection
- D_max: d_max is max euclidean distance b/w cells (160.0 i vs 300.0 e)
- Nsyn_min/nsyn_max:# of synapses!! No need to change!!

The d_weight_min and d_weight_max were calculated using the Allen Synaptic Physiology Dataset. I made a copy of the jyputer notebook, and calculated the 95% confidence intervals for each neuron pair that was present in my network.

STEP THREE: LGN NETWORK

The LGN network has 500 excitatory neurons of one population class tON. The sources of the LGN inputs were distributed retinotopically through function generate_lgn_positions, which randomly assigns X and Y coordinates using numpy's random.uniform.

In order for the LGN neurons to provide spikes to V1 neurons based on the positioning of the source and target neurons, the V1 neurons' positions were distributed in 2D space:

```python
def convert_to_2d(target):
    # x & z in cortex --> translation in visual space
    x_coordinates = [target['positions'][0]]
    z_coordinates = [target['positions'][2]]

    # convert x and z to linear degrees: tan(x) * (180/pi)
    X = np.tan(0.07 * np.array(x_coordinates) * np.pi / 180.) * 180.0 / np.pi
    Y = np.tan(0.04 * np.array(z_coordinates) * np.pi / 180.) * 180.0 / np.pi
    return np.column_stack((X, Y))
```

Documentation

This function transitions the 3D positions of the V1 neurons to visual space. The x and z axises in the cortex translate to the visual space The coordinates were converted to radians, and then to linear degrees. The function returns the positions as a 2D column stack.

Now that the LGN network has been created with 2D positions, and we have a functionality to convert the target nodes to 2D – we can connect the neurons so that LGN inputs can provide spikes to each V1 neuron according to that neuron's position in 2D. The connection_rule between the LGN nodes and the V1 populations is defined by select_source_cells

```python
def select_source_cells(sources, target, nsources_min=10, nsources_max=30,
d_max=300, nsyns_min=3, nsyns_max=12):
    # for every source cell, limited number of presynaptic targets
    total_sources = len(sources)
    nsources = np.random.randint(nsources_min, nsources_max)

    # randomly select sources to synapse to target node
    selected_sources = np.random.choice(total_sources, nsources,
    replace=False)

    syns = np.zeros(total_sources) # initialize synapse array to 0's


    for index in selected_sources:
        node = sources[index]
        # compute the euclidean distance between source node and target node
        dist = np.linalg.norm(np.array(node['positions']) -
        np.array(convert_to_2d(target)))

        # create synapses b/w nodes if within maximum distance
        if dist <= d_max:
            syns[index] = np.random.randint(nsyns_min, nsyns_max)
```

```
   return syns
```

Documentation

This function is passed as the connection rule when creating edges between LGN and V1 nodes. It connects the source and target neurons by randomly selecting source LGN nodes, checking if they are within maximum distance of the target V1 node, and then randomly selecting a number of synapses for that source-target.
Parameters:

- Sources: entire LGN network
- Target: populations of V1 network (LIF_exc, SST, PV, VIP)
- N_sources_min: minimum number of sources that target cell can receive spikes from
- N_sources_max: maximum number of sources that target cell can receive spikes from
- D_max: d_max is max euclidean distance b/w cells (160.0 i vs 300.0 e)
- Nsyn_min/nsyn_max:# of synapses!!

STEP FOUR: CREATING LGN SPIKES INPUT

Used statistics of LGN firing from the Neuropixels dataset to provide inputs into the V1 model. The full source code for the data cache can be found in LGN_DataCache.py. Individual sessions were filtered out based on genotype and LGN as one of the ecephys structures.

```
# filter out sessions by genotype (SST, PV, VIP, WT)
sst_sessions = sessions[(sessions.full_genotype.str.find('Sst') > -1) & \
                       (sessions.session_type == 'brain_observatory_1.1')& \
                        (['LGd' in acronyms for acronyms in
                        sessions.ecephys_structure_acronyms])]

def compute_firing_rate(sessions, stimulus):
   firing_rates = [] # instantiate list of firing rates


   for count in range(len(sessions)):
       session_id = sessions.index.values[count]
       session = cache.get_session_data(session_id)

       # filter out units with high snr
       units_with_very_high_snr = session.units[session.units['snr'] > 4]
       session_rates = units_with_very_high_snr[stimulus]

       # session rates is all the firing rates of each unit in a session
       session_avg = statistics.mean(session_rates)
       firing_rates.append(session_avg)

   # firing_rates is a total list of firing rates of all units of all
   sessions
   typical_firing_rate = statistics.mean(firing_rates)
   return typical_firing_rate
```

Documentation:

This function collects the firing rates of a filtered subset of sessions from the Visual Coding Neuropixels dataset

in response to a specified stimulus. It's purpose is to filter and compute typical firing rates of specified neurons. It averages over all units of a session, and then all sessions to find a typical firing rate of a given genotype in response to a given stimulus.
Parameters:

- Sessions: Pandas dataframe; In the context of this exercise, pass in pre-filtered sessions of LGN neurons that provides input to the neuron types of our V1 model (sst, pv, vip, wt)
- Stimulus: String that is the metric name for stimulus in question; in the context of this exercise 'firing_rate_fl' (chosen stimulus is full field flash)
    - Drifting gratings firing rate: firing_rate_dg
    - Static gratings firing rate: firing_rate_sg
    - nature scenes firing rate: firing_rate_ns
    - Dot motion firing rate: firing_rate_dm
    - Full field flashes firing rate: firing_rate_fl
    - Gabors firing rate: firing_rate_rf

The typical firing rates (typical sst firing rate : 8.683619967260256; typical pv firing rate: 7.109125520976934; typical vip firing rate: 6.655023111202979) as computed with compute_firing_rate were then used to create an h5 file (lgn_spikes.h5) of LGN spikes using BMTK's PoissonSpikeGenerator.

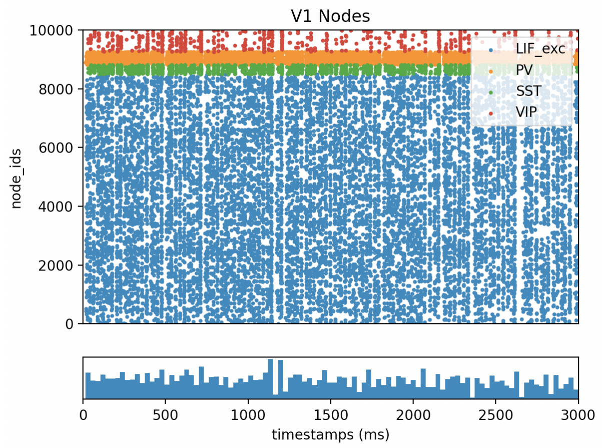
STEP FIVE: BUILDING NETWORK FOR SIMULATIONS

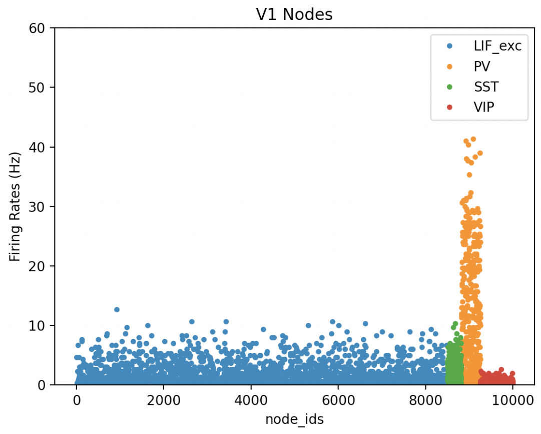
To run simulations, a PointNet environment was set up, and config.json, simulation_config.json, and circuit_config.json were created. However, I altered the run_pointnet.py to build the environment using the config.json file instead of simulation_config since I consistently received errors. The config.json file was edited to incorporate all of the necessary parameters.

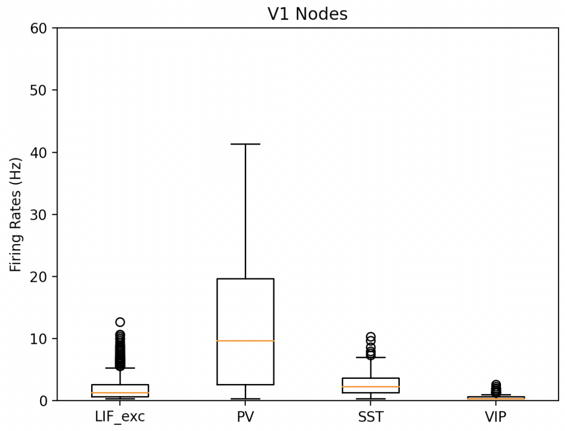
```
from bmtk.simulator import pointnet

def main(config_file):
   configure = pointnet.Config.from_json(config_file)
   configure.build_env()

   network = pointnet.PointNetwork.from_config(configure)
   sim = pointnet.PointSimulator.from_config(configure, network)
   sim.run()

if __name__ == '__main__':
   main('config.json')
```

```
{
  "manifest": {
    "$BASE_DIR": "${configdir}",
    "$NETWORK_DIR": "$BASE_DIR/network",
    "$OUTPUT_DIR": "$BASE_DIR/output",
    "$COMPONENTS_DIR": "$BASE_DIR/components"
  },
  "target_simulator": "NEST",
  "run": {
    "tstart": 0.0,
    "tstop": 3000.0,
    "dt": 0.01
  },
  "inputs": {
    "LGN_spikes": {
      "input_type": "spikes",
      "module": "h5",
      "input_file": "$BASE_DIR/lgn_spikes.h5",
      "node_set": "LGN"
    }
  },
  "output": {
    "log_file": "log.txt",
    "output_dir": "$OUTPUT_DIR",
    "spikes_file": "spikes.h5",
    "quiet_simulator": true
  },
  "components": {
    "synaptic_models_dir": "$COMPONENTS_DIR/synaptic_models",
    "point_neuron_models_dir": "$COMPONENTS_DIR/point_neuron_models"
  },
  "networks": {
    "nodes": [
      {
        "node_types_file": "$NETWORK_DIR/LGN_node_types.csv",
        "nodes_file": "$NETWORK_DIR/LGN_nodes.h5"
      },
      {
        "node_types_file": "$NETWORK_DIR/V1_node_types.csv",
        "nodes_file": "$NETWORK_DIR/V1_nodes.h5"
      }
    ],
    "edges": [
      {
        "edge_types_file": "$NETWORK_DIR/V1_V1_edge_types.csv",
        "edges_file": "$NETWORK_DIR/V1_V1_edges.h5"
      },
      {
        "edge_types_file": "$NETWORK_DIR/LGN_V1_edge_types.csv",
        "edges_file": "$NETWORK_DIR/LGN_V1_edges.h5"
      }
    ],
    "gap_juncs": []
  }
}
```

STEP SIX: RESULTS (no perturbations)
The results of my network are as shown. When compared to the experimental data, this looks roughly correct. The LIF excitatory neuron population exhibits regular-spiking (RS) behavior, as displayed through the large amount of small firing rates between 0-5 Hz. The PV inhibitory neurons display fast-spiking (FS) behavior with firing rates ranging from 0-40 Hz. The SST and VIP inhibitory neuron populations receive little to no LGN input, and therefore have low firing rates.

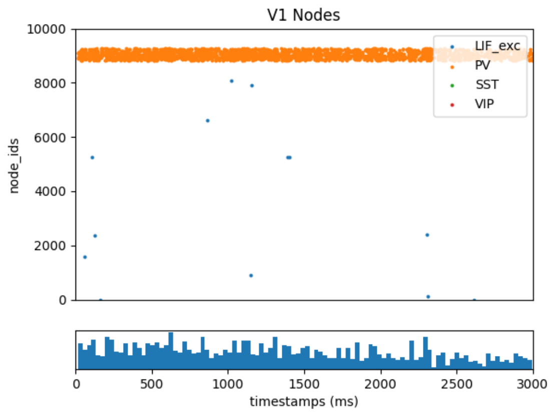


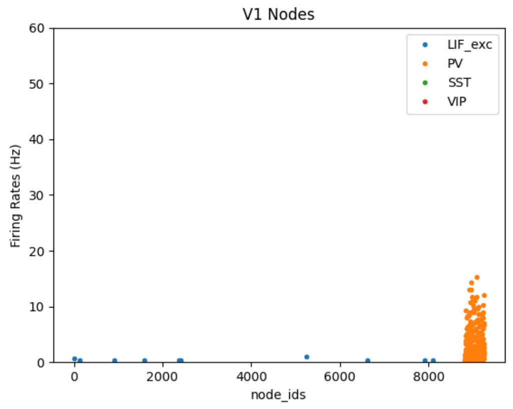


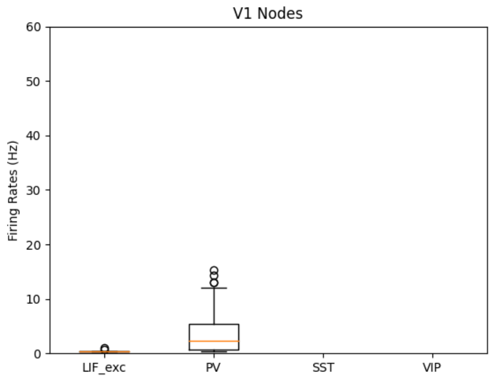


STEP SEVEN: PERTURBATIONS
Perturbation of neuronal activity has always played a key role in neuroscience research. Electrical stimulation provides added temporal control, it has been widely used for perturbing vision and evoking eye movements. I applied perturbations to the network by electrically stimulating each population one by one through a clamp with an absolute amplitude of 230.0 (- for inhibitory, + for excitatory). The source code can be found in perturbations.py.

Perturbation #1: Inhibitory perturbation of LIF_exc population

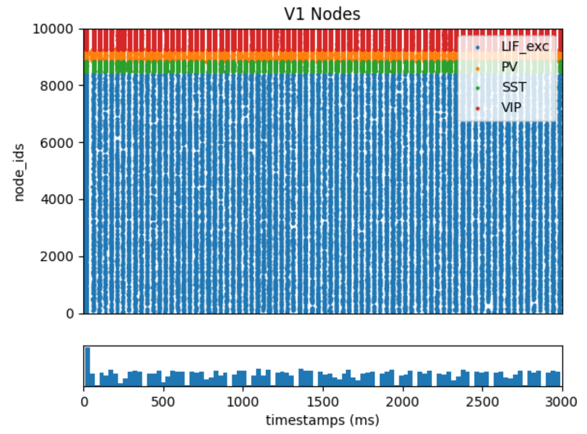


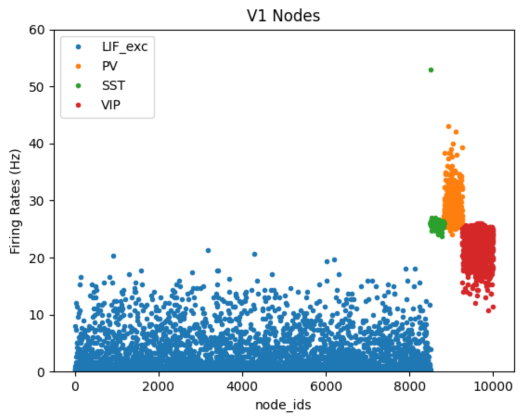


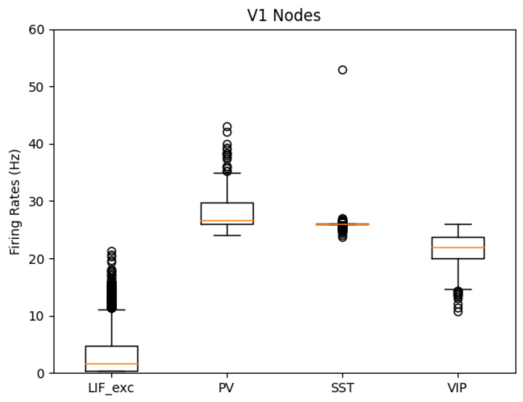


- An expected result of the inhibitory perturbation of the LIF excitatory neurons is that the population is basically shut down, with very few firing rates (all of which are of amplitudes between 0-1 Hz).
- The perturbation led to a depression of the inhibitory populations as well. The SST and VIP populations are entirely shut down, and the PV population's firing rates are decreased.
- This is due to the connections from the excitatory population to the inhibitory populations. Once the LIF_exc nodes are shut down, so are their subsequent connections to the VIP, SST, and PV populations, resulting in a drop of responses in each of these populations.

Perturbation #2: Excitatory Perturbation of LIF_exc population

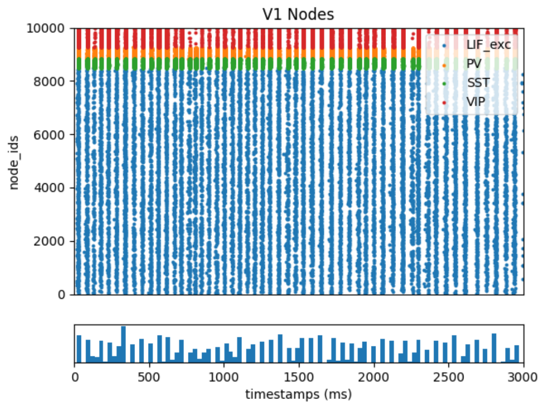


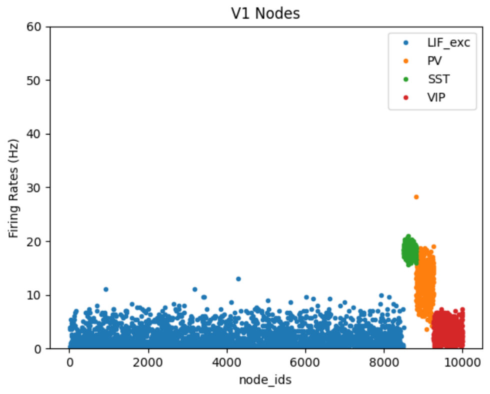


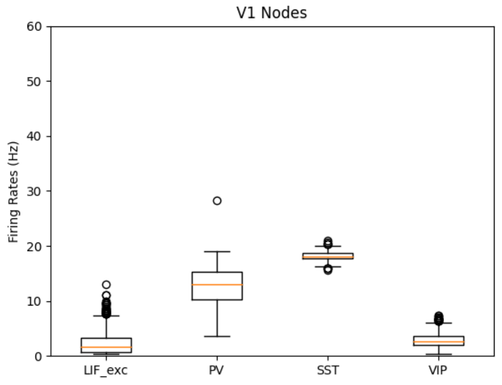


- There is an increase in the amount and amplitude of LIF_exc neurons that fire as a result of the excitatory perturbation.
- Additionally, the SST and VIP inhibitory populations exhibit strengthened firing (with an increase in the amount and amplitude of their firing rates). Out of all the perturbations performed, this is the strongest that the VIP population performs.
- While the range of the PV firing rates is now the higher half of the original range (0-50 Hz), the population displays weakened firing with less neurons firing.

Perturbation #3: Inhibitory Perturbation of PV population

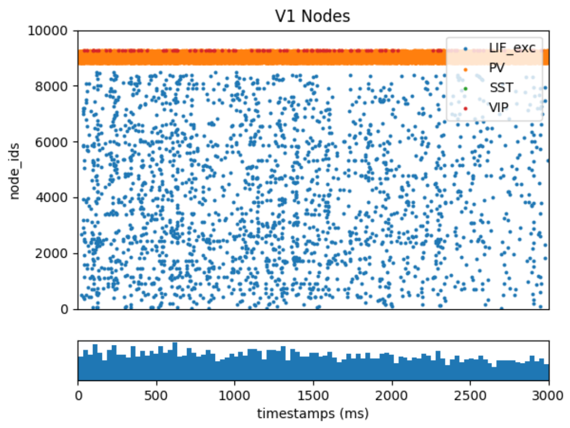


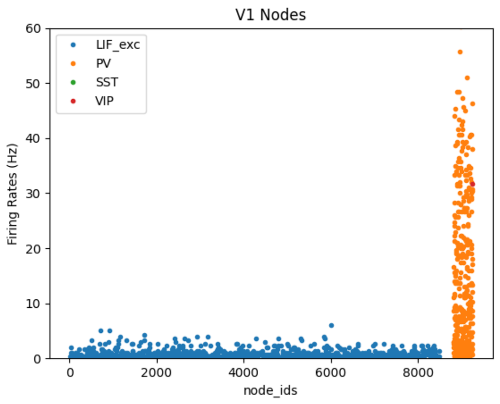


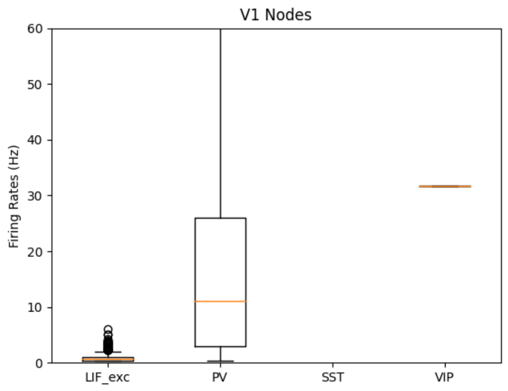


- The pattern in the raster plot indicates that spikes were temporally synchronized across all the populations.
- LIF_exc seems relatively unaffected (same as in simulation with no perturbations)
- The PV population's range of firing rates becomes shortened, with a smaller amount of nodes firing.
- There's an increase in the amplitude of SST firing rates, but a decrease in the amount of SST nodes that exhibit spikes.
- There is also an increase in the amount and amplitude of VIP firing rates.

Perturbation #4: Excitatory perturbation of PV population

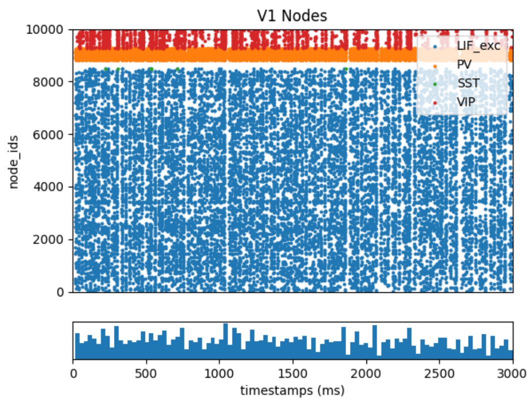


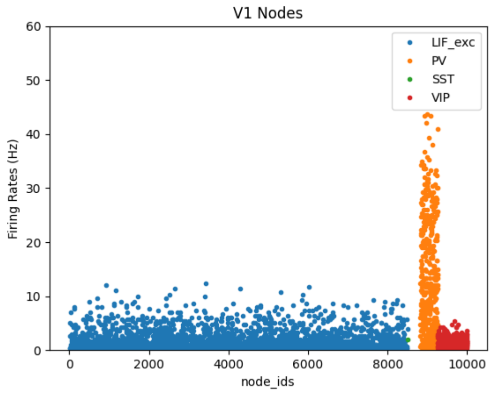


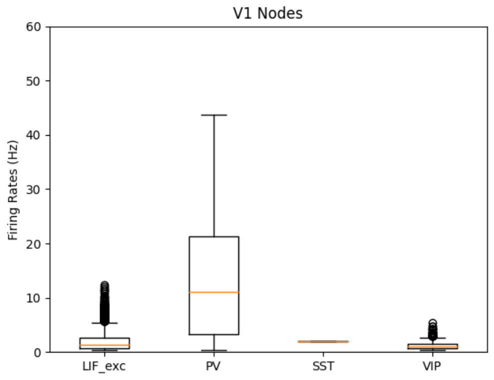


- PV nodes demonstrates increased firing rates (originally, the highest firing rate was 40 Hz versus here, where it goes to 60 Hz).
- Additionally, the SST population is shut down (displays no firing), and there was a decrease in the amount of VIP neurons that fired.
- There is one VIP neuron that spikes, but it spikes consistently leading to a single very high firing rate.

Perturbation #5: Inhibitory Perturbation of SST population

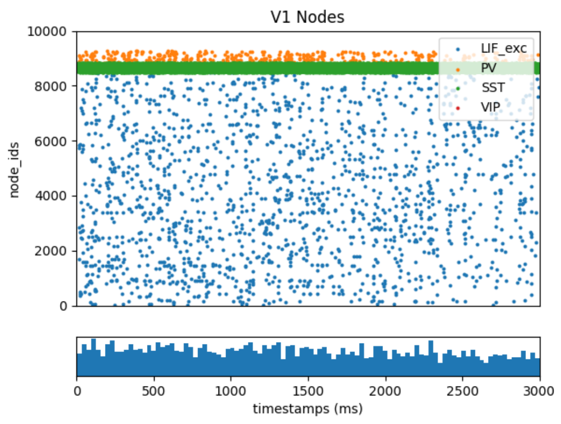


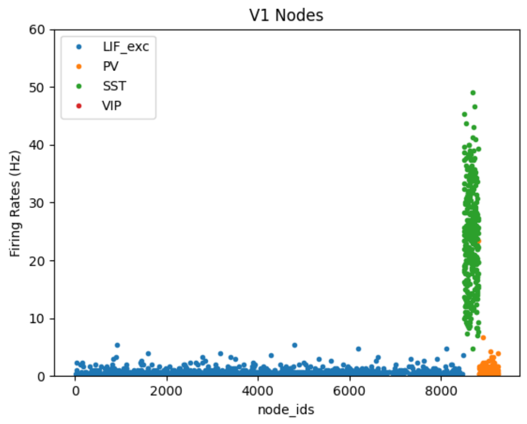


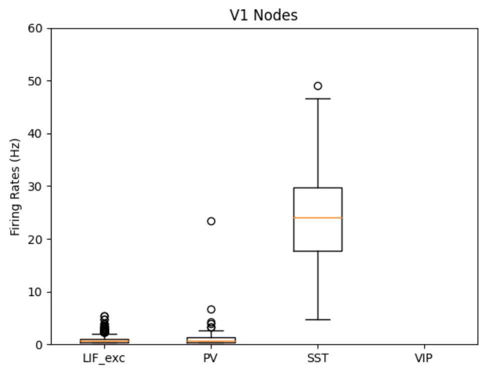


- The SST population is basically shut down as a result of its inhibitory perturbation, with only one node that spikes and a firing rate of 2-3 Hz.
- However, unlike the inhibitory perturbation of the PV population (where its inhibition led to improved firing rates of the other inhibitory populations), there is relatively no change in the behavior of the PV and VIP populations.

Perturbation #6: Excitatory Perturbation of SST population



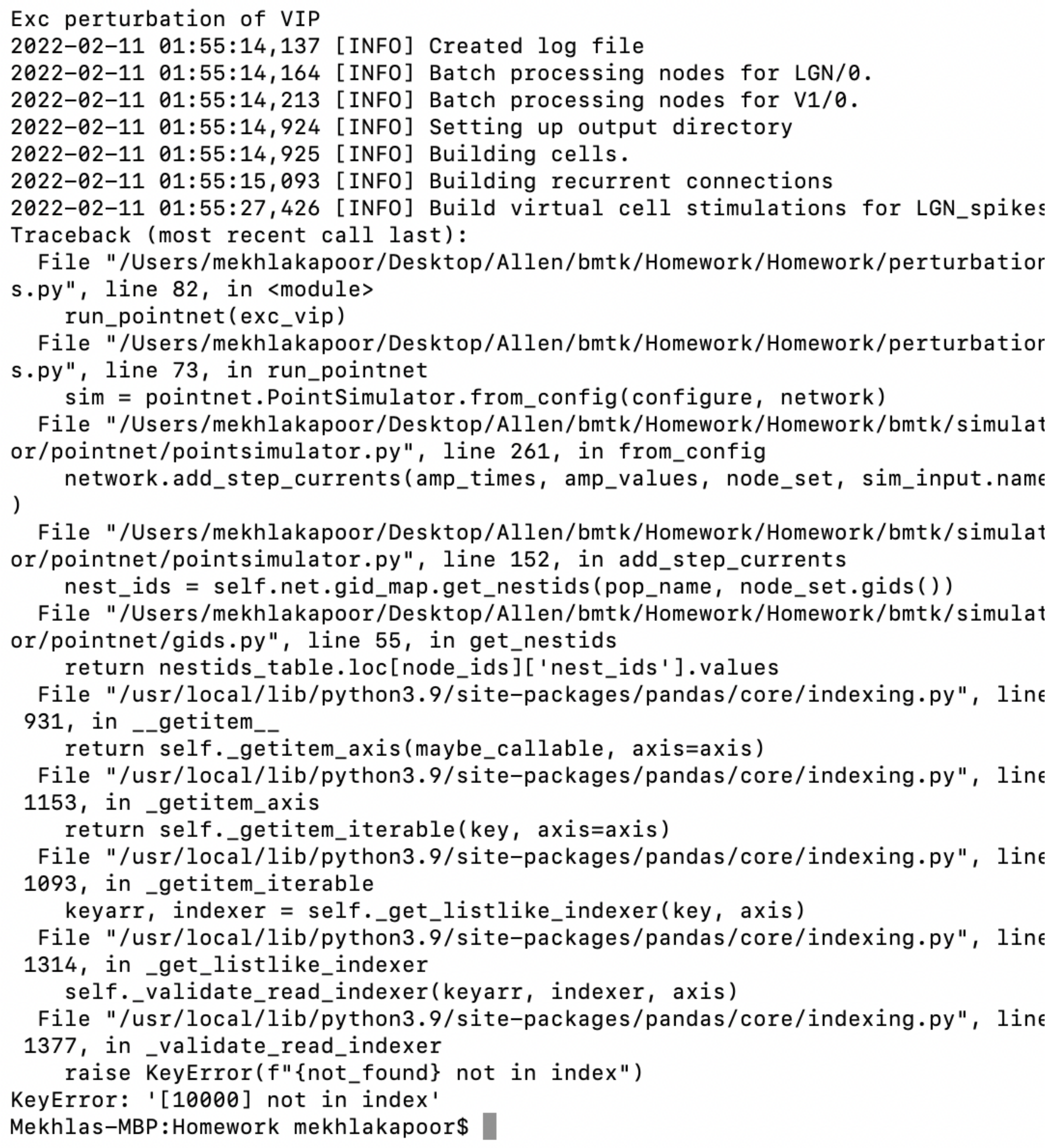




- There is a tremendous improvement in the amount and amplitude of SST firing rates.
- However this excitation of the SST population leads to a huge decrease in the rest of the populations; the LIF_exc and PV populations drop dramatically in amplitude and amount, and the VIP population is completely shut down and displays no firing rates.
- This indicates that the SST population strongly inhibits the other populations when strengthened.

NOTE: Excitatory and Inhibitory Perturbations of VIP population

- When perturbations are applied to the VIP population, no matter what the amplitude of said stimulation, it causes a shutdown of the population and leads to index errors when the lgn attempts to send spikes to it. The entire error message is below for reference. Additionally, this behavior occurs no matter what the amplitude of the perturbations is. This is most likely due to the extremely small connection probabilities that the VIP population has to the others, while their connections to VIP are much stronger. Through the simulations, the VIP population consistently performs weaker than the other inhibitory populations unless those are inhibited.



```
Exc perturbation of VIP
2022-02-11 01:55:14,137 [INFO] Created log file
2022-02-11 01:55:14,164 [INFO] Batch processing nodes for LGN/0.
2022-02-11 01:55:14,213 [INFO] Batch processing nodes for V1/0.
2022-02-11 01:55:14,924 [INFO] Setting up output directory
2022-02-11 01:55:14,925 [INFO] Building cells.
2022-02-11 01:55:15,093 [INFO] Building recurrent connections
2022-02-11 01:55:27,426 [INFO] Build virtual cell stimulations for LGN_spikes
Traceback (most recent call last):
  File "/Users/mekhlakapoor/Desktop/Allen/bmtk/Homework/Homework/perturbatior
s.py", line 82, in <module>
    run_pointnet(exc_vip)
  File "/Users/mekhlakapoor/Desktop/Allen/bmtk/Homework/Homework/perturbatior
s.py", line 73, in run_pointnet
    sim = pointnet.PointSimulator.from_config(configure, network)
  File "/Users/mekhlakapoor/Desktop/Allen/bmtk/Homework/Homework/bmtk/simulat
or/pointnet/pointsimulator.py", line 261, in from_config
    network.add_step_currents(amp_times, amp_values, node_set, sim_input.name
)
  File "/Users/mekhlakapoor/Desktop/Allen/bmtk/Homework/Homework/bmtk/simulat
or/pointnet/pointsimulator.py", line 152, in add_step_currents
    nest_ids = self.net.gid_map.get_nestids(pop_name, node_set.gids())
  File "/Users/mekhlakapoor/Desktop/Allen/bmtk/Homework/Homework/bmtk/simulat
or/pointnet/gids.py", line 55, in get_nestids
    return nestids_table.loc[node_ids]['nest_ids'].values
  File "/usr/local/lib/python3.9/site-packages/pandas/core/indexing.py", line
 931, in __getitem__
    return self._getitem_axis(maybe_callable, axis=axis)
  File "/usr/local/lib/python3.9/site-packages/pandas/core/indexing.py", line
 1153, in _getitem_axis
    return self._getitem_iterable(key, axis=axis)
  File "/usr/local/lib/python3.9/site-packages/pandas/core/indexing.py", line
 1093, in _getitem_iterable
    keyarr, indexer = self._get_listlike_indexer(key, axis)
  File "/usr/local/lib/python3.9/site-packages/pandas/core/indexing.py", line
 1314, in _get_listlike_indexer
    self._validate_read_indexer(keyarr, indexer, axis)
  File "/usr/local/lib/python3.9/site-packages/pandas/core/indexing.py", line
 1377, in _validate_read_indexer
    raise KeyError(f"{not_found} not in index")
KeyError: '[10000] not in index'
Mekhlas-MBP:Homework mekhlakapoor$ 
```