This document specifies the component specifications of the build_a_brain package. Each module is defined as well as the classes and functions within the modules.

Module: neural_classes

class neuron()

<u>Overview:</u>

- Defines the single unit neuron which is updated at each timestep of the simulation. This is a leaky integrate and fire neuron model.

<u>Functions:</u>

__init__(self, tau):

Initializes variables:

v0: float:	resting membrane potential
v: float:	current membrane potential
tau: float:	leakiness constant that defines how quickly the neuron spontaneously loses voltages
spiked: bool:	defines if the neuron recently spiked and can receive inputs or is in it's refractor period
refractory: int:	counter variable that defines how long ago the neuron spiked

update_potential(self, dt):

Updates the potential of the neuron based on the Euler equation. Checks if the neuron is in a refractor period. If it is, iterate the refractory counter. If it is not, update the neuron's potential.

dv: float: change in membrane potential at this timesstep

add_noise(self, input):

Adds a volage to the membrane potential if it is not in a refractory period.

input: float: voltage to input

set_inital_potential(self, input):

set the initial voltage of the neuron without checking if the neuron spiked.

input: float: voltage to input

input_spikes(self, spikes):

updates the current membrane potential based on an input presynaptic spike by a factor of 5mV if the neuron is not in a refractory period.

Spike: int: 1 if spike, 0 if no spike

reset_potential(self):

Sets the current membrane potential to -100mV, the spiked Boolean to true, and the refractory period to 5.

class neural_layer():

Overview:

- Defines a signle layer in the network

Functions:

__init__(self, num_neurons,layer_id,weights):

Initializes the layer variables:

layer_id: int: layer identifier in the model

neruons: list of neuron() objects: initializes neurons for this layer with random tau constants between 1 and 5

weights: lxlxnxc matrix of ints: l is the number of layers in the network,n is the number of neurons in each layer, and c is the number of connections each neuron has to adjacent layers. The int specifies which neuron this neuron is currently connected to in layer a to layer b.

set_initial_inputs(self, inputs):

calls the set_initial_input function of each neuron in the layer and initializes the current potential of that neuron based on the inputs

inputs: list of floats: Each index in the list corresponds to a neuron in the layer and defines the potential to set it to.

feed_forward(self, inputs, in_layer_id):

updates the current membrane potentials of each neuron in the layer based on the inputs. The inputs define if the presynaptic neurons fired. The connections to each presynaptic neurons defined by in_layer_id are updated by calling the input_spikes function for the neuron in which it is connected to defined by the layer.weights.

Inputs: lxn matrix of ints: 1 if spiked and 0 if not. L is the number of layers and n is the number of neruons in each layer.

add_noise(self,inputs):

adds a noise the layer. Calls the add_noise() function for each neuron in this layer

Inputs: list of floats: length is the number of neurons, and each index corresponds to a neuron in this layer.

driving_stimulus(self,drive_neurons):

adds a driving stimulus to neurons in this layer defined by the drive_neurons list. The stimulus is a random float between 0 and 20.

drive_neurons: list of ints: holds the index number of the neurons in this layer to receive the stimulus.

reset_potentials(self,inputs):

resets the neuron potentials of neurons in this layer depending of if these neruons spiked at this timepoint in the simulation. Calls the reset_potential() function on neurons that spiked.

Inputs: list of bool: length of the list is the same as number of neurons in the layer. True if spiked and False if not.

voltages(self):

resturns a list of the voltages for each neuron in the layer.

class neural_net():

Overview:

- Defines the neuron network

Functions:

__init__(self,num_layers,layer_sizes,connectivity):

Initializes the network variables:

Connectivity: lxl matrix of floats: defines the percentage of neuronal connections from neurons in layer 1 to neurons in layer b

weights: lxlxnxc matrix of ints: defines the neuron conections of each neuron in each layer that are generated randomly based on the connectivity percentage defined in connectivity.

Layers: list of list of layer objects: layers are generated by passing layer_sizes and weights to the layer() class.

feed_forward(self,inputs):

updates the neuron potentials based on input spikes from presynaptic connections. Calls the feed_forward() command on each layer which feeds in layer inputs and layer_id.

inputs: lxlxn matrix of ints: 1 if spiked and 0 if not. L is layer size, n is number of neruons in that layer

add_ noise(self,inputs):

adds stimulus to neuron potentials in the network by calling add_noise() on each layer.

Inputs: lxn matrix of floats: l is layer size and n is number of neurons in the layer.

reset_potentials(self,inputs):

resets the potentials of each neuron in the network by calling reset_potentials() on each layer

inputs: lxn matrix of bools: True if this neuron spiked, False if this neuron did not spike.

Voltages(self):

Returns all voltages of each neuron in the network in the form of a lxn matrix of floats.

Weights(self):

Returns the weights of each neuron in the network in the form of a lxlxnxc list of ints . L is layer size, n is number of neurons in the layer and c is a list that is length of number connections this neuron has to the neurons in another layer.

Module: neural_simulation

Function run_simulation(root, progress, num_steps, layer1_size, layer2_size, layer3_size, layer4_size, layer5_size, connectivity_matrix, driving_layer, driving_neuron_nums):

This function builds the neural network based on defined inputs and then runs the simulation over a defined number of steps each at 1ms. If root == -1, then no gui will be updated. First builds the network by creating a neural_net class object. Runs the simulation over time defined by num_steps and performs the following actions:

1) Check if neruons spiked
2) Reset potential of spiked neurons
3) Feed forward inputs of spiked neurons to postsynaptic connections
4) Add stimulus to driving layer
5) Update potentials based on the leakiness loss of voltage

Returns net, spikes, voltages

Spikes is a txlxn matrix of ints: 1 if spiked, 0 if not. t is time steps, l is layers, n is neurons.

Voltages is a txlxn matrix of floats: voltages at each timestep where t is time steps, l is layers, n is neurons.

Root: tk object: the gui object, can be entered as -1 if no gui is used.

Progress: tk progress bar object: the progress bar that shows runtime of the simulation displayed on the gui, can be entered as -1 if no gui is used.

Num_steps: int: number of time steps in the simulation

Layer_sizes: int: number of neurons in each layer (1-5)

Connectivity_matrix : lxl matrix of floats: defines the connectivity of each layer to the Others

Driving_layer_num: int: layer_id number that defines which layer is reciving constant Inputs

Driving_neuron_nums: float: percentage of neruons in the driving layer that are reciving constant inputs.

Function plot_raster(ax,root,progress,net,all_spikes,t):

Plots the raster of spikes over time for each neuron in the network color-coded by layer. If not gui is being used, root and progress can be input as -1.

Ax: matplotlib ax object: defines the axes to plot the raster on

Root: tkiner object: defines the gui to plot the raster on.
Progress: tkiner progress bar object: defines the progress bar on the gui to be updated
Net: net class object: defines the simulated network to be plotted which is an output of run_simulation()
all_spikes: txlxn matrix of ints: 1 or 0 based on if spiked or no spike at each timepoint which is an output of the run_simulation() function.
T: int: number of timesteps the simulation was run for.

Module: interface:
Function build_network_interface():

Builds the gui interface which allows the user to input parameters of layer size, layer connectivity, stimulation layer, stimulation strength and run time. First initializes the gui. A button is available "default connectivity" which will auto populate connectivity fields. Once all fields are filled out, the run simulation button will run the run_simulation() function followed by the plot_raster() function with the function inputs being taken from the user inputs in the gui. If all fields are not populated, or populated with incorrect values, an error will be displayed on the gui.

Interactions to accomplish use cases:

<u>Use Case: use the interactive interface to build and visualize the raster of a neural network.</u>

For this use case, the objective of the user is to use the toy model to understand how different network configurations provide different spiking dynamics. The user will interact with the system by either running the build_network_interface() command in the command line, or in a jupyter notebook.

The simple interaction with the system will generate the gui and guide the user to input values that are needed to run the simulation.

The output will be a raster plot in the gui of the spiking activity of the network.

<u>Use Case: Generation of neural networks with defined layer sizes and connectives.</u>

For this use case, the objective of the user is to build a neural network of defined configuration that defines connections between each neuron and other neurons in the network The user can run the net = neural_net(…) command in a jupyter notebook to create a net object that is defined by their inputs to the function.

This interaction with the system is best deployed in a jupyter notebook since the network structure is of most importance. The network structure can be accessed by the attributes of the class that are defined in the documentation above.

<u>Use Case: Simulate neural activity over time of a defined network architecture.</u>

For this use case, the objective of the user is to is to understand neural dynamics in a network, but this interaction with the package gives more informative outputs than a simple plot such as voltages and spikes over time. The user will use the net, spike, voltages = run_stimulation(...) command in a jupyer notebook to generate matrices of membrane voltages and spikes over time.

This use case is more in depth and for users with experience in python and working with spiking data from neural networks.