# TPN simulation: documentation

Leslie S. Smith

December 10, 2024

## 1 Data structures

There are currently seven Matlab structures in this simulation, *simulation, neuron, targets, apical, basal, shunts, IIneuron*. These are described below.

### 1.1 Structure: *simulation*

This describes simulation-wide information and parameters. It has 5 fields:

**N_TPNs** number of two point neurons

**N_IIs** number of inhibitory interneurons

**duration** duration of the simulation (seconds)

**timestep** simulation time step (seconds)

**simlength** duration of the simulation (time steps); calculated from simulation.duration and simulation.timestep

### 1.2 Structure: *neuron*

This describes information relevant to the two point neurons (TPNs). It is an array of structures, with one per TPN, It has 12 fields currently:

**refractoryperiod** refractory period of each TPN

**relrefperiod** relative refractory period of each neuron

**thresh_leap** amount by which the threshold increases (leaps) after a spike is generated

**thresh_decay** rate at which threshold decays after increasing

**thresh_value** initial value of the threshold

**maxnospikes** the maximum number of spikes that can be stored for this TPN

**TP_threshold** holds the threshold for the whole simulation duration for each TPN

**targets** structure holding information about the targets for the exon of this neuron (see below)

**thresh_increment** the incremental threshold added after a spike (computed from neuron.thresh_leap and neuron.thresh_decay as well as neuron(i).refractoryperiod, neuron(i).relrefperiod and simulation.timestep.

**th_inc_length** length of neuron(i).thresh_increment

**spikes** timing of spikes (in timesteps). Maximal number is neuron(i).maxnospikes

**targets** target for this TPN neuron's axon. Format (neuron type, neuron number, synapse type, delay (in seconds)).

### 1.3 Structure: *targets*

This is an array of structures inside each of the neuron(i) and IIneuron (i) structures, one per connection of the neuron's axon to another neuron. It has 4 fields:

**t_ntype** the type of neuron to which this axon is connected: TPN or II

**t_nno** the identity of the neuron to which the axon is connected

**to_syntype** the type of synapse to which the axon is connected (A, B, AS, BS, S)

**delay** the delay associated with this connection (in seconds)

**delaysamps** the duration in timesteps (calculated from duration above, and simulation.timestep.

### 1.4 Structure: *apical*

This is an array of structures, one per TPN, describing the apical area and apical spines. Each apical synapse works using the same values, except for the weight which is different for each synapse. Each structure has fields:

**n_apicalinputs** the number of apical inputs (synapses) on this neuron

**tau_apical** the time constant of the apical area

**C_apical** the capacitance of the apical area

**R_apical** the resistance of the apical area

**R_synap_dendrite** resistance of apical dendrite (apical to basal)

**C_apical_spine** the capacitance of the apical spine (for all the synapses)

**R_apical_spine** the resistance of the apical spike (for all the synapses)

**R_synap_spine**

**synapsemultiplier** used to allow weights to be in a unitary range, while making the voltages of an appropriate size.

**apicalinputs** the timing of the (external) apical inputs, in the form of (timestep, synapseno ) repeated

**apicalsynapseweights** an array of weight values, one per apical(i).n_apicalinputs

**apicalspikeno** maximal number of external apical spikes

**alpha_apical** array providing the alpha function for (all the) apical synapses

**apicalpostsynapse** array (number of apical synapses (apical(i).n_apicalinputs) by simulation.simlength) recording postsynaptic apical activation

**apicalfracleak** fractional leak from apical area: set from simulation.timestep / (apical(i).R_apical $*$ apical(i).C_apical)

**apicalspinefracleak** fractional leak from (all) apical spines: not used inTPN_runstep_v3, (set to 0),but planned to be used in v4.

### 1.5 Structure: *basal*

This is an array of structures, one per TPN, describing the basal area and basal spines. Each basal synapse works using the same values, except for the weight

**n_basalinputs** the number of basal inputs (synapses) on this neuron

**tau_basal** the time constant of the basal membrane

**C_basal** the capacitance of the basal area

**R_basal** the resistance of the basal area

**R_synba_dendrite** resistance of basal dendrite

**C_basal_spine** the capacitance of the basal spine

**R_basal_spine** resistance of basal spine

**R_synba_spine** resistance of basal dendrite (basal to apical)

**synapsemultiplier** used to allow weights to be in a unitary range, while making the voltages of an appropriate size.

**basalinputs** the timing of the (external) basal inputs, in the form of ( timestep, synapseno ) repeated

**basalsynapseweights** the basal synaptic weights, one per basal(i).n_basalinputs

**basalspikeno** maximal number of external basal spikes

**alpha_basal** array providing the alpha function for (all the) basal synapses

**basalpostsynapse** array (number of basal synapses (basall(i).n_basalinputs) by simulation.simlength) recording postsynaptic basal activation

**basalfracleak** fractional leak from basal area: set at simulation.timestep / (basal(i).R_basal $*$ basal(i).C_basal)

**basalspinefracleak** fractional leak from (all) basal spines: not used (set to 0) in TPN_runstep_v3,but planned to be used in v4.

### 1.6 Structure: *shunts*

This is an array of structures, one per TPN. It describes the different shunting synapses, which may be apical or basal. These multiplicative synapses act on the summed activation.

**noapicalshunts** : number of apical shunting synapses

**nobasalshunts** : number of basal shunting synapses

**apicalshuntinputs** : actual apical shunt inputs, in (time synapse_number) format, external and internal

**basalshuntinputs** : actual basal shunt inputs, in (time synapse_number) format, external and internal

**apicalshuntweights** : weights for apical shunt: $0 \leq$ value $\leq 1$ where 0 has no effect

**basalshuntweights** : weights for basal shunt: $0 \leq$ value $\leq 1$ where 0 has no effect

**apicalshuntduration** : time that apical shunt is active for (in seconds): shunting effect is spread over this duration

**basalshuntduration** : time that basal shunt is active for (in seconds): shunting effect is spread over this duration

### 1.7 Structure: *IIneuron*

This structure describes the inhibitory interneurons (IIs), which are basically leaky integrate and fire neurons. There is an array of these structures, one per II.

**refractoryperiod** : II LIF refractory period, during which it cannot fire

**relrefperiod** : relative refractory period, during which threshold is elevated

**C** capacitance of single compartment

**R** leakage resistance of single compartment

**II_threshold** : array to hold actual threshold

**thresh_leap** : amount by which threshold increases after firing (and after refractory period)

**thresh_decay** rate at which the elevated threshold decays

**thresh_value** initial threshold value

**th_increment** : array to hold threshold increment

th_inc_length length of th_increment

**spikes** array of spike times

**spikecount** number of spikes produced

**maxnospike** maximum number of spikes for this neurons

**tau** used in setting up alpha function

**alpha_synapse** : array from alpha function (which is the same for each synapse, thus for each neuron)]

**activation** array to hold activation

**inputs** the (external, currently) inputs to this II neuron (format: (time, synapse snumber)

**n_synapses** number of synapses on this II neuron

**alpha_synapse** alpha function for all the synapses on this II neuron

**weights** the weights on each synapse (should be n_synapses long)

**fracleak** leakage per time step, calculated from C, R and simulation.timestep.

**targets** target for this II neuron's axon. Format (neuron type, neuron number, synapse type, delay (in seconds)).

## 2 Spike format

Spikes may be externally or internally generated. External spikes are predefined, and arrive from outside the simulation. Internal spikes arise when a neuron fires: each firing may create a number of spikes on at different synapses, on different neurons.

### 2.1 External spike inputs

External inputs may be for either a TPN or an II. Those for a TPN may be apical, basal, apical shunting or basal shunting. There is, however, only one type of synapse on an II. Internally, they have the format ($N_i$ time $S_i$), where $N_i$ is the neuron number, time is the time of arrival of the spike (in seconds), and $S_i$ is the synapse number. These should be in time order (although there may be more than one spike at any specific time). This will help in implementing internal spikes.

### 2.2 Internal spikes

Internal spikes arise from spikes on any of the neurons. These will be coded internally a (target_type $N_i$ time $S_i$) where target_type may be apical, basal, apical shunting, basal shunting or inhibitory interneuron. Each spike may arrive at a number of different synapses, on a number of different neurons, each with their own delay.

## 3 Functions

There are a number of functions within this simulation.

### 3.1 runTPN_spines_shun_v4

This is the script that calls the functions. Currently implemented as a script, as this provides visibility for all the data structures after it has terminated. It calls *readnetwork*, *setupnetworkV2*, *setupinterconnection*, *setupweights*, *TPN_runstep_v4*, *II_runstep*, *createspikelist* and *spikeraster*. *setupnetworkV2* and *setupinterconnection* both call *readnetwork* which reads a text file into a table.

## 4 Operation

### 4.1 Synapse on a spine (on a TPN) (applies to both apical tuft and basal dendrites)

Logically, a presynaptic pulse arrives at a spine synapse, and results in the transfer of charge to the dendrite. This transfer of charge takes place over time, and is represented by the alpha function. The total charge is proportional to the weight associated with the synapse.

Currently (TPNrunstep_v3) the alpha function is added to the depolarisation (voltage? current?) of the spine, and then these are summed to give the apical current. This is probably not the best idea, even if it basically works!

We could consider the synapse to be transferring charge to the dendrite to which it is attached. So we could just multiply the alpha function by the weight to get the time course of this charge, and simply add the charge up at the dendrite (i.e. consider the dendrite as a point).

While this is simpler, it loses the advantage of the isolation of the spine. Alternatively, consider the charge incoming at the spine (which is proportional to the weight): this takes place over time in a way characterised by the alpha function which describes the conductance and hence the current. This charges up a spine capacitance (C_apical_spine) (with leakage resistance (R_apical_spine)) and allows a voltage to be recorded at the spine over a time period rather longer than the alpha function. This voltage causes a transfer of charge (current over time) through the resistance of the body of the spine (R_synap_dendrite) into the dendrite. When we come to consider adaptivity, we would then have a voltage at the spine (where adaptation take place) as well as the voltage at the dendrite enabling a more nuanced approach to changing the weight (or other characteristic of the spine base synapse)[1].

Matlab functions have been written to help accomplish this in TPNrunstep_v4: setupAlphaFunction.m (used to create the alpha function as an array which sums to 1, appropriately when the alpha function describes the post-synaptic voltage, and used in v3), and setupAlphaFunctionV2.m (which is used in v4, where the alpha function describes the transfer of charge through the synapse, so instead of summing to 1, the integral (sum) of the function over time is 1

$$\sum_{i=1}^{\text{length}(\alpha)} (\alpha(i)\Delta t) = 1 \tag{1}$$

and implemented in alphafunctioneffect.m (used in V4 to calculate the voltage effect on a spine of the charge coming in, time step (simulation.timestep) by time step, considering the capacitance and leakage resistance at the spine containing the synapse).

While the units are now correct (the incoming charge is current times time, and the outgoing voltage is the voltage on a capacitor in response to this charge), we need to think about the actual size of these values. The alpha function integrates to 1, so that the total incoming charge is weight coulombs. Given a weight of (say) 3, and a capacitance of $10^-8$ farads (i.e. $0.01\mu F$), this leads to a voltage of

$$\frac{1}{C} \sum_{i=1}^{\text{length}(\alpha)} (\alpha(i)\Delta t)w \tag{2}$$

i.e. $3 * 10^8$v, independent of the timestep, and the leakage is set to 0, or rather less (about $1.2 * 10^8$, for a leakage of 0.01) if not.

## 4.2 Apical tuft to apical dendrite

The voltage on each spine-based dendrite passes a current through a resistor (R_synap_dendrite) to the dendrite, where this results in an increase in the voltage across a capacitance

[1]The spine itself is isolated from the dendrite, but adaptation at the spine has local values of both the depolarisation at the spine and at the dendrite to which it is attached. Thus (for example) a burst arriving at the spine synapse might cause a large increase the depolarisation at the spine, which could result in conformational (or other) longer term changes at the spine, This might be modulated by depoarisation at the dendrite (base of the spine).

(C_apical), which is concurrently being discharged by a leakage resistance (R_apical). The voltages from the different spines are summed. The apical dendrite is currently considered as a point.

If we need to subdivide the different inputs to the apical dendrite, and deal with them in a more complex way than simply summing them, then we will need to introduce some geometry to the apical dendrite.

### 4.3 Basal dendrite

The charges arriving from the spine-based dendrites are summed: again there is a capacitance (and a leakage resistance) associated with the basal dendrite (which is considered as a point currently). If we need to subdivide the different inputs to the basal dendrite, and deal with them in a more complex way than simply summing them, then we will need to introduce some geometry to the basal dendrite.

### 4.4 Oblique dendrite: from tuft to basal area

This transfers the charge (voltage?) from the apical area to the basal area, enabling the apical area to influence the basal area, and hence the possibility of spike production.The details of how this happens are not currently clear, and we characterise this using the equation from Reza and Ahsan.

$$\text{ahactiv}(t) = (B(t)^2 + 2B(t) + 2A(t)(1 + |B(t)|) \tag{3}$$

where $B(t)$ is the basal activation (voltage) and $A(t)$ is the apical activation (voltage), and ahactiv$(t)$ is the axon hillock activation. This results in a spike if the threshold is exceeded. On a spike being generated, the threshold is set to $\infty$ for the refractory period, then falls to the initial threshold + threshold_leap, and then exponentially decreases towards the initial threshold.

### 4.5 Inhibitory Interneuron (II)

Inhibitory interneurons are simple leaky integrate-and-fire neurons, characterised by a single compartment with a capacitance C and leakage resistance R. Incoming spikes a synapses release charge according to an alpha function and weight, and this charges the capacitor C (while leaking through R), resulting in a voltage across the capacitor. These voltages are summed, and a spike is caused when the threshold is exceeded. As for the TPN axon hillock, on a spike being generated, the threshold is set to $\infty$ for the refractory period, then falls to the initial threshold + threshold_leap, and then exponentially decreases towards the initial threshold.

## 5 Spikes and their targets

When a neuron spikes, the spike is transferred , with a delay, to 0 or more synapses. This is defined in the targets element of the neuron (i.e. the target element of the neuron

structure for TPNs, and the target element of the *IIneuron* for inhibitory interneurons). This element has the structure *targets*, described above.

When a spike occurs, the spike is added to the spikes list for the target neuron: this is *apicalinputs* in the *apical* structure, *basalinputs* in the *basal* structure, and *inputs* in the *IIneuron* structure. The list of incoming spikes is sorted into time (in *timesteps*) order. Currently, no differentiation is made between predefined external spikes and those incoming from other neurons. Indeed the same synapse may receive inputs from both external spikes and those from any of the other neurons. Whether this happens depends on the predefined external spikes and the content of the file from which the targets are read (targets.txt currently).

# 6 Testing and setting of parameters

Without testing, one cannot be sure that the system is behaving properly: indeed, even with testing one cannot prove this, but testing certainly improves belief that the system is behaving appropriately.

Testing should start simply: one TPN and one II, with some external inputs, and internal synapses of all types. This appears to function.