# Neuron 1 Documentation. Program version 0.56 (1.0) started Jan 20 2008.

Last updated 2 April 2008.

**The synapse.**

The synapse is the most critical part of the simulation: what happens there really makes all the difference – and there are, of course, a great many of them.

## The synapse objects

There are three synapse classes, synapse, dynsynapse and shuntsynapse: synapse is the parent class of the other two. All of them import syndata.h, which contains the basic data structures describing the synapse, syndata, syndatatype, and *syndataptr, and  the adaptivity structure, STDP, STDPinfo.

## Methods (in synapse class).

Firstly those used in setting up the synapse, or interrogating it for values:

`- init`
initialise on setup: set delay to 0, set the number of pre-synaptic spikes to 0,
`- setWeight: (float) w ;`
Sets weight to w.
`- setAlpha: (float) alph alphareset: (bool) alphareset;`
set alpha and alpha resetting for alpha function.
`- setSynapseType: (int) s ;`
set the type of synapse. Used to syntype in syn1, but not otherwise.
`- setPresynNeuron: (Neuron *) NNN number: (int) preno ;`
set the identity of the presynaptic neuron. Preno is the actual number of the presynaptic neuron.
`- setPostsynNeuron: (Neuron *) NNN number: (int) postno;`
set theidentity of the postsynaptic neuron, and its number.
`- setDelay: (double)d : (int) dsteps ;`
set the synaptic delay in seconds and timesteps.
`- setResetContributionOnInputSpike: (BOOL) resetval ;`
YES implies only 1 spike contributes
`- setAdaptive: (BOOL) adaptive:  (STDPinfo) adaptparams ;`
set the synapse to be adaptive, and fill in all the information about the details from adaptparams.
`- computeSynapseEffects: (double) timestep;`
Note: over-ridden in ShuntSynapse object: danger
Computes the value of the alpha function for each timestep, then places these values into an array for use by getSynapticContribution. Uses alpha function
        (alpha * alpha) * t * exp(-alpha * (t)) ;
additionally, if alpha = 0, then make whole effect occur at next timestep (use this for synapses from input neurons).
`- (float) getWeight ;`
returns current value of weight
`- (int) getSynapseType ;`
1=non-dynamic 2=dynamic
`- (syndatatype) getSynapseData ;`

returns whole structure describing synapse configuration

`- getPresynNeuron ;`

returns id of presynaptic neuron

`- getPostsynNeuron ;`

returns id of postsynaptic neuron

Methods used when applying the synaptic effect.

`- presynFires: (float) t_fire : (int) stepfire;`

Invoked (by method fire in Neuron object) when a presynaptic spike arrives at time t_fire (sample time sepfire). The number of presynaptic spikes is counted, and the times placed in the arrays lastspiketimes and lastspikesteps. This done in such a way that the 0'th element is the most recent. However, resetContributionOnInputSpike is true, only one spike (the most recent) is counted).

`- (float) getSynapticContribution: (double) current_time: (int) stepno ;`

This method is called from the neuron object (methods calcSynContribution and calcMultSynContribution) to find out the amount of activation contributed by this synapse. If there are no presynaptic spikes, 0 is returned. Otherwise, the contribution of the presynaptic spikes is summed. If a presynaptic spike has just arrived, the method applySTDPPresynaptically is invoked. Because an alpha function is being used, the actual contribution fro presynaptic spikes is different each time step.

`- applySTDPPresynaptically : (float) presyntime ;`

If the synapse is adaptive this method is invoked whenever a presynaptic spike arrives. The weight will not change if: LTP winner is set, and the postsynaptic STDP is already applied; if the delay between the last spike postsynaptically and this current presynaptic spike is too large (greater than (MAXPOSTSYNDT / adaptvalues.postpreTau)); if the adaptation is nearest neighbour (NN) and the postsynaptic STDP is already applied; if this adaptation would result I the weight changing sign, and this is not permitted. Weight change may be additive or multiplicative. Weights are bounded. (Note that for multiplicative synapses, this bound should have a modulus less than or equal to 1).

`- postsynFires: (float) t_fire: (int) stepfire ;`

invoked (by method fire in Neuron object) when a postsynaptic spike occurs. Used for resetting alpha functions and STDP adjustment. No weight change is made if: it would result in a disallowed zero-crossing. Given that there are presynaptic spikes, changes may be made. These may be nearest neighbour (in which case only the most recent presynaptic spike will be used) or classic (in which case all eligible ones will be used).

**Dynamic synapses, and how the dynamic synapse alters the effective weight.**

This is implemented in the presynFires method of dynsynapse.

Each dynamic synapse has two additional parameters, rechargerate and utilrate. Each dynamic synapse has a local variable, level, associated with it. This is initially 1. The variable is used to compute the effective weight by multiplying the original weight by it. Each time a presynaptic spike arrives at a dynamic synapse, level is used in weight computation, and then is decremented multiplicatively by multiplying it by exp(-utilrate). In addition, level gradually finds its way back to 0 by being incremented arithmetically by (current_firing_time - last_firing_time) * rechargerate (but is limited to 1) before application of level to weight. In this way, utilrate defines how quickly the effective weight decreases in response to rapid presynaptic firing, and rechargerate how quickly it recovers.

**Shunting Synapses.**

Have not been fully tested.