operator<< . inherited

Neuron

id, layer

Connection*

vector

vector<Conn*> conns

OR ~~vector<Conn2 *conns~~ X

Sigmoid

neuronType

in, out, lambda

activate( )

Spiking

neuType

thresh

in_current*

spike

spike_train.

Izhi

v, u, a, b, c, d

LIF

C1

C2

C3 ONE COPY

node1 [*C1, *C2, *C3]

node2 [*C1, *C2]

Connection

c_id, in_node, out_node

weight, enable,

conn_type

vector: receptor

outputs from in_node

SpikingConn

static spikeFactor

Neuron — Conn — Neuron — Conn

Network<Ti, TA, To> : NetworkBase

id

~~vector~~ neurons

vector { in_neurons, hid_ne..., out_ne..., conns

Net(id, i_num, h_num, o_num)

copyNet copyconstructor

copy neurons

copy connections

neuron add conns

conn add neurons

Network(const Net &net)

① neurons(net.neurons) X

② for(auto &n: net.neurons){

neurons.push_back(n);

X

copying the pointer

③ Copying the Neurons

for(auto &n: neurons){

newNeuron(*n)

neurons.push_back( )

Copying conns.

conns.push_back(new C(*c

for(&c: conns) {

for(&n: neurons) {

if(n->id == c->in->id)

c->in = n;

n->out_conn.push_back(

Node n1 n2 n3 *C1 *C2

*n1 *n2 *n3

vector<Neuron*> neurons

Neuron/Connection

Do NOT do copies in these classes

SpikeNet::run( )

- load inputs (sensor)

for t: TIMESTEPS

{

- hid_nodes.

- tmp += sen_val × weight × spikeFactor

- load(tmp)

- step( )

- out_nodes

- tmp += spike × weight × spikefactor

- load(tmp)

- step( )

}

HybridNet::run( )

- load sensor inputs.

for TIMESTEPS

{

- hid_nodes.

- tmp += ...

- load(tmp)

- step( )

↳ spike_train.

}

- out_nodes.

- tmp += rate · weight

- activate

in_n C out_n

Run a network.

- load inputs
- run.
  - Sigmoid_net
  - Spiking_net
  - Hybrid_net

load_inputs (accumulate inputs, non-input_layer)
activate.
transmit output to connections.

for (timesteps)
  load inputs from connections.
  step(t)
  transmit spikes.
end.

- hidden_neurons.

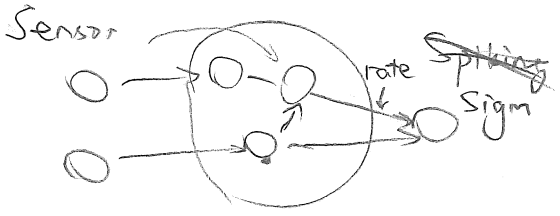
for (timesteps)
  load inputs.
  step(t)
  transmit spikes.
end

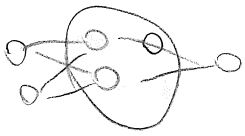
for (&c: out_conn)
  if(c is NOT spiking
    transmit (rate)
end

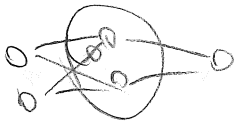
- output_neurons

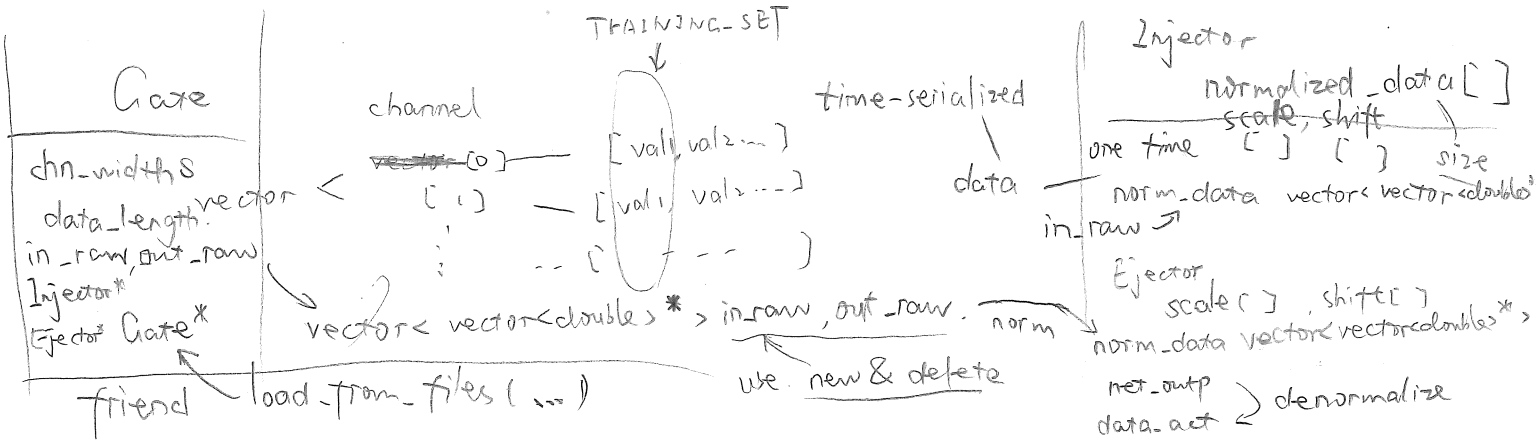
for (&c: out_conn)
  if c -> out_node == spik
    transmit spike
  end

load inputs.
activate.
transmit.



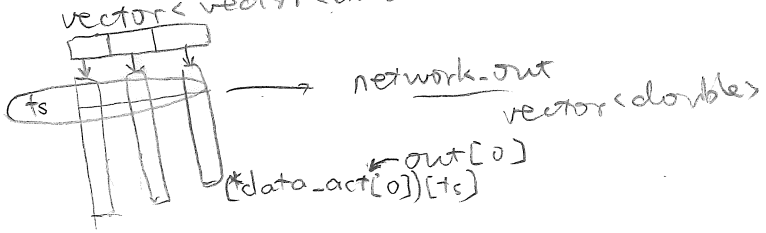






Ejector

data_act
vector< vector<double>*>



gate

- load_from_file → raw_data.
- load_factors_file → norm_factors.
- init/Ejector / Injector → allocate memory

for (ts : data_length)
get_norm_inject_data (ts) → load to net
net.run ( )
get_net_out_to_ejector (ts)

evaluate → cal_mse using net_out & data_norm
cal net fit

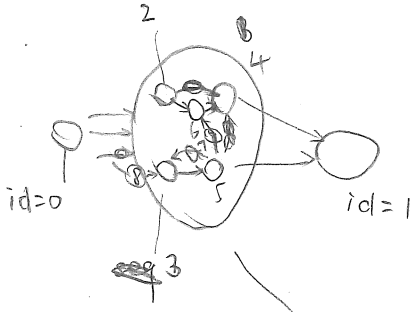
Data Generator

time interval
timesteps
width, owidth
mapping_func (system dynamic)
savefile

generate ( )
t = 0; timesteps × intv.
for (&d : imps)
(*d)( )

Add a neuron.

pos_to_insert



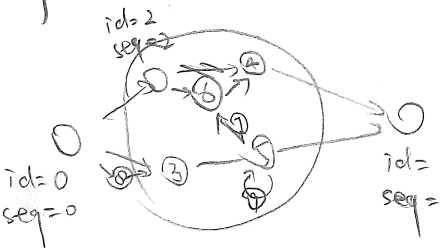
hid → ②
③ push_back
④ push_back
⑤
else
⑥ if (Onode_id > Inode_id)
insert_before_Onode
⑦ else (Onode_id < Inode_id)
insert_after_Inode
⑧ if (Inode == in_neuron.
insert before_Onode

if (Onode == out_neuron

do not use id to order them.

sequence: 2, 8, 3, 6, 4, 5, 7 (9)

use seq instead



seq 1 2 3 4 5

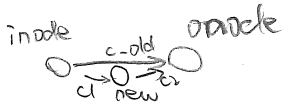
assign seq after add_neuron.
seq begins from 1

if (out == out_neuron
push_back

if (in_seq < out_seq)
insert_before_out
else (in_seq >= out_seq)
insert_after_in.

including conn_to_itself

add_neuron



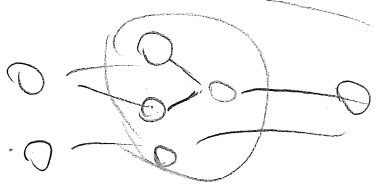
new.add_inConn(c1)
new.add_outConn(c2)

~~c1.add_add~~

c1.inode.add_outConn(c1)
c2.onode.add_inConn(c2)

inode.remove_outConn(c_old)
onode.remove_inConn(c_old)

add_conn



inode_size = 2
hnode_size = 4
onode_size = 1

inode: ishift(0, 5) (ith-1)
onode: oshift(2, 6) (all-1)

if(ishift < 2) then (oshift (2,5)

No conn from input to output

add_neuron.

add_neuron_in2out (next_cid) innov. next_nid. -> ++next_nid

pop.

new_nid -> 6.

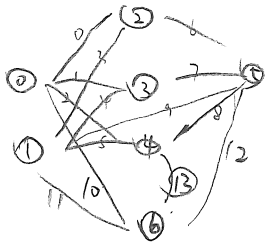
new_cid -> 10. next_cid.

11, 12

next_cid += 2 + onode

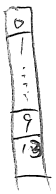
inode

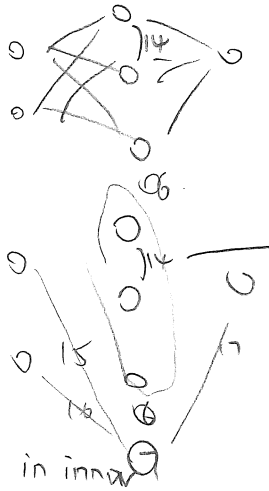
add_conn.(next_cid)



next_cid=13.

next_cid=14.





next_cid=15.

next-cid =17.

1. get local next_nid

2. find if existed in innov

true: use that nid, cid in innov

false: use new nid, cid.

add innov -> (nid, cid)

pos to insert neuron. -> easy.

conn. ->

---

add_neuron (next_nid _ next_cid)

1. find conn_to_mutate.
2. create_neuron & conns.
3. check innov
   - set id.
   - push new innov if any.
4. find pos to insert neuron.