Python interface to the 'point neuron simulation'

Author: Yongwang SHI

This project is the homework project of CS902, lectured by Prof. Chaojun LU

This interface is currently under development, use with care!

Main features:

- Support the commonly used features of the 'point neuron simulation'
- One for all resolution using Python
- Various modes: rm(simulate and remove data files automatially), read(read from data file and proceed filtering and visulizing), nameX(keep data files and return the filenames instead of data)
- lab-neu: automatically generate tasks, distribute to servant machines/processes, simulate and filer in parall, collect data together
- [BUG] With matplotlib to show running plot of the voltage trace interactively
- [Developing] Smart filter: dig out precious data with custom-designed quries

Overview

There are 3 modules and 3 demos:

Modules:

- gen_neu.py: an interface to 'point neuron simulation'
- lab_neu.py: generate tasks and assign to servant machines. Including
    - time_cost(): compute the expecting time_cost of an task, in unit "per operation"
    - class Task: abstraction of an task, identified by task.tid, parameter for simualtion is stored in task.pm
    - class task_generator: effectively and conveniently generator tasks according to what user's command
    - class lab: waiting for connection from servants, assign tasks and collect data
- servant.py: (class servant_machine) connect to center machine and recive task, simulate, filer(in developing) and return to the center machine

Demos:

- gen_neu_demo.py: how to use gen_neu
- server_demo.py: how to use lab_neu, including generate tasks and start the 'lab'
- servant_demo.py: how to use class servant_machine

Usage

about gen_neu:

```
# Get defalut parameters
pm = parameters()

# Customize parameters
    # basic ones
pm['simulation-method'] = 'simple' # it could be simple, SSC(spike order corrction)
                                   # and (NOt TESTED YET:) IF-jump, etc(refer to
                                   # https://github.com/bewantbe/point-neuron-network-simulator)
pm['neuron-model'] = 'HH-PT-GH' # it could be HH-PT-GH, LIF-G, LIF-GH, DIF-GH, DIF-single-GH, etc
                                # refer to the link above
pm['net']           # the adjacent matrix of the neruon net(system)
                    # you can fill in a path or a list of list (matrix)
                    # by default it's -, which means all neuron connected

pm['t'] = 100       # the lasting time of simulation
pm['dt'] = 1.0/32.0 # the duration of a small time step
pm['stv']= pm['dt'] # the duration of one data record
pm['nE'] = 1        # number of excititory neurons (neurons that gives a positive inplus)
pm['nI'] = 0        # number of inhibitory neurons (neurons that gives a negative inplus)

    # more advanced para
pm['pr'] = 0.1      # poisson rate (the probability of a random input,
                    # to simulate the complicated network connection of human neuron system
pm['ps'] = 0.014    # poisson strength, not to be too large
pm['scee'] = 0.05   # the effect of one fired 'e'xcititory neuron giving to a connected 'e' one
pm['scei'] = 0.05   # the effect of one fired 'e'xcititory neuron giving to a connected 'i' one
pm['scie'] = 0.05   # the effect of one fired 'i'xcititory neuron giving to a connected 'e' one
pm['scii'] = 0.05   # the effect of one fired 'i'xcititory neuron giving to a connected 'i' one


# call the simulator and run simulation.
result = gen_neu(pm,'v rm')  # the second parameter could be
# v: default, verbose, print process
# rm:default, remove data files (you can the content via return value)
# no-rm: force not to remove data files
#        NOTICE: rm is by default on, even if you do not specify it
#                if you want to keep the files, you MUST specify no-rm
# read:  do not do simulation but read existing files
#        useful when you 'no-rm' and clear the return value
# nameX: do not return data, only return filenames
#        in this mode, no-rm is forced on
# cmd:   do not do simultation but print the command line options
#        that "is" going to be executed, useful for test and debug


# after simulation
print "data has been stored in list 'result'"
print [(x[0],x[1]) for x in result] #x[2] is data itself, you can use it
```

```
plt.plot(result[0][2][0])  # show the voltage trace of the first neruron
plt.show()
```

about lab_neu:

```
from lab_neu import task_generator, lab
from gen_neu import parameters

# The following part shows how you can effectively and conveniently
# (re)produce many tasks,
# for example, you want to test serveral model each with serveral possion rate
# and if the possion rate is less than 0.4, you want to simulate for longer time(1000 ms)
# It could be a lot of boring work if you want to do it manually, but 'task_generator' will do it

pm = parameters()               # Create the default parameters for simulation
pm['t'] = 100                   # You can change the parameters by keys

tg = task_generator(pm)         # Use 'pm' as a ptototype of all tasks' parameters

# You can reproduce many tasks by changing some values of parameters
# tg.add(masks, target, range) will search in tg.pms for those coincide one of the 'masks'
# and replace the vlaue of key 'target' with values in 'range'
tg.add(tg.pms, 'neuron-model', ['HH-PT-GH','HH-GH'])
tg.add(tg.pms, 'pr', [x/100.0 for x in range(0,100)])

# You can filter the masks for more specific reproduction
masks = [dict(x) for x in tg.pms if x['pr']<0.4 and x['neuron-model']=='HH-PT-GH']
                                                # We DEEP copied some pms as masks
                                                # DEEEEEEP COPY, please
print "tasks generated: %d" % len(masks)
for mask in masks:
    mask['neuron-model'] = '*'
            # All values will be matched if you set the value of mask to be '*'
            # In other words, we ignored the difference of neuron-model
tg.add(masks,'t',[1000])


# Once you have told task_generator what you want to simulate,
# tg,generate will generate all tasks, it's a list of 'Task' instances
tasks = tg.generate()
print([(a.tid, a.pm['neuron-model'],a.pm['pr']) for a in tasks])

# Once you get all the tasks, you can run the 'lab'
HOST = "127.0.0.1"
PORT = 22222        # Servant machine will connect at this HOST and PORT
l = lab(tasks,HOST,PORT,gen_cmd = 'v rm')
        # gen_cmd is the modes of the interface
```

```
        # by default, it's v rm
        # v:          verbose process
        # rm:         remove data files after simulation
        # no-rm:      force not to remove data files, you must specific no-rm
        #             if you want to keep the files, since 'rm' is by default on
        # extro-data: reuturn extra-data, including conductance and gating variables.

# start assign work, make sure this is started first before all servants
l.start()
```

about servant.py:

```
from servant import Servant

PORT = 22222        # the same as what the lab_neu has opened
HOST = "127.0.0.1"  # the hostname of the machine running lab_neu
ADDR = (HOST,PORT)

sm = Servant(ADDR)  # initialize the servant machine
# At this point,  make sure lab_gen has started 'lab_gen.start()'
sm.start()          # Start working
```

For more, please refer to comments at the beginning of each file. For the interpertation of each neuron models and their parameters, refer to https://github.com/bewantbe/point-neuron-network-simulator

Bugs:

It is welcomed to tell me any bugs. The following is some bugs that have not repaired so far.

1. On linux, center machine recive broken message from the servant, though the buffer is large enough
2. If the center machine is not closed properly, the port can't be released and you need to reassign a PORT if you want to continue.

Programming Practice

This part is just for my hw project.

There are some programming skills that have been applied in this project.

本项目的背景：

计算神经学中常利用各种模型模拟人的神经网络。有时我们会尝试各种模型和参数，希望能解释某一现象或着预测可能会发生的现象。已有几个不错的模拟器，其中包括'point neuron simulation'。

但该模拟器使用C++编写，不是很便于使用。这个项目包括一个Python借口，用户可以在交互式环境中直接编辑各项参数，无需将其保存成参数文件，即可进行模拟；同时也能直接得到模拟结果。

此外，为了方便大量的模拟，这个项目包括一个任务生产器，能抽象地生成大批任务（比如模型为A，B和C，参数为pr = 0,0.01,...,0.99，pr>=0.4只需模拟100ms,但其他的要模拟1000ms，C需要额外参数arg=...，本项目能轻松高效地生成这样的复杂任务，只需简单的几行代码）。最后，此项目利用socket和消息驱动，实现了多终端、多线程模拟（Python的多线程无法真正利用多线程的计算资源，本项目可以通过在一个计算机上的多个端口运行来利用计算机的多线程资源）。

模块化，事件驱动编程：

由于（多个）远程机的进度的不确定性和并行编程的复杂性，本项目采用事件驱动的方法。中心机等待远程机发送连接、申请作业、进度汇报、申请提交等消息，中心机收到消息后决定如何响应。

相关代码：

```
def lab():              # 中心机程序
    def start(self):
        self.s.listen(5)
        while not self.all_finished():
            c, addr = self.s.accept()
            jmsg = json.loads(c.recv(1024))
            title = self.interprete(jmsg)   # 此处解读是哪个远程机，是什么类型的消息
            if title == 'hello':
                sid = None
            else:
                sid = jmsg['sid']
            self.logger('Event',sid,'start()','get connection')

                        # 此处对不同消息进行不同的处理，对不同的操作进行模块化

            {
                'hello': self.hello,
                'bye'  : self.bye,
                'error': self.error,
                'report': self.report,
                'pull-request':self.pull_request,
                'task-require':self.task_require,
                'task-process':self.task_process
            }[title](c,jmsg,sid)


            c.close()
        return self.tasks
```

面向对象编程：

本项目对任务、任务生成器、远程端、中心端进行了合理的抽象，并将其封装成类。

相关代码： 对任务的抽象

```
class task:
    tid= None
    pm = None
    status = 'not-assigned'
    assigned_to = []            # We may assign the same task to various servants in some cases
    time   = [None, None, None] # [expected_time_cost_cmp,expected_time_cost_abs,real_time_cost]
    data   = None               # Finished, data will contain the results from the servant

        (some functions)

# status of an task:
#    not-assigned
#    assigned
#    simulating
#    filtering
#    pulling
#    finished
#    failed
```

例子

参见 usage，或者pyfile*_demo.py

gen_neu 示意(gen_neu_demo.py)：

lab_neu 示意（server_demo.py和servant_demo.py配合运行）： 左：server_demo.py,右：servant_demopy

Credits:

Thanks to 'point neuron simulation' from which I copied the network files.