Programming project report: Neuron Network

Aline Brunner, Claire Payoux, Florence Crozat et Justin Mapanao

Lausanne, December 17, 2020
Supervised by Maxime Emschwiller

Abstract

This report discusses the programming project done in the context of the *Projet en informatique (pour SV)* course, given by Prof. Jacques Rougemont. The project models a Neuron Network, based on Izhikevich's model[1], with the programming language C++. We explain which difficulties we faced and why we chose the implementations we used. We also present some results, examples of command and some tendencies we discovered to obtain certain types of results.

Contents

1 Introduction and objectives

This project simulate a neuron network, following the simplified model described by Eugene Izhikevich. We coded this model with the programming language C++. We are using gtest to test the program and Doxyfile to generate our documentation. One can launch the simulation from a terminal and there is multiple parameters to play with.

The model contains two main types of neurons : the inhibitory ones and the excitatory ones. There are seven sub types of neurons. The evolution of the neurons is defined by their nature but also by their connections. The simulation makes the network evolve according to the current received by the neuron depending on its connections during a given number of time steps.

2 Class Hierarchy

Our program is based on four basic classes: Simulation, Network, Neuron and Random.

The **Simulation** class handles the parameters given by the user, using the library *Tclap*. It creates the network of neurons according to the user specified-parameters and deals with the time steps. This class also generates the outputs.

The **Network** class creates the collection of neurons and connects them. On one side it stores the neurons and on the other side their connections, which are characterized by the neurons targeting them and the intensity of the connections. This class also updates the neurons and calculates the synaptic current that will be transferred to each neuron at each step of time.

The **Neuron** class models the neuron itself. It is defined by four parameters, specific to each type of neuron. The neuron state (firing or not) depends on the changes of the membrane potential and the recovery variable.These evolve during each step of the simulation. As a pure virtual class, the Neuron is inherited by two sub-classes **ExcitatoryNeuron** and **InhibitoryNeuron**

The **Random** class provides some distributions, which are very useful to model our neurons and the connections.

The file *main* creates the simulation and runs it.

3 Parameters and commands

3.1 Parameters and user

Our model allows the user to choose several parameters to modify the simulation. These are detailed with the command line:

"./neuron_network -h" after the executable is made in the build folder.

- The option **-c** gives the user the choice to create additional files where we can see the parameters characteristics and the evolution of a sample of each type of neuron during the simulation.
- The option **-N** allows the user to select the total number of neurons in the network.
- The option **-t** selects the duration of the simulation, in milliseconds.

- The option **-l** changes the mean connectivity of the neurons, (the number of connections it can make).
- The option **-L** represents the mean intensity of a connection, specific to each connection.
- The option **-d** gives an adjustable parameter to make the neuron parameters a, b, c and d vary in the interval "parameter" * [1−d, 1+d].
- The option **-p** is the proportion of excitatory neurons within the network.
- The option **-T** allows one to give more specific proportions for all types of neurons, for instance " IB:0.5,RS:0.2,FS:0.3 ".
- The option **-m** selects the generation model of the neurons. It can be basic (b, Poisson distribution of the number of connections between the mean given), constant (c,all neurons have the same number of connections to them) or over-dispersed (o,Poisson distribution of the inverse of the mean number of connections).
- The option **-o** allows the user to choose the name and path of the basic output file, which is a matrix of 0 and 1 indicating the firing state of the neuron.

The output files can be analysed using the Rscript file given to us with the command (with default parameters for the output file)

Rscript ../Rasterplots.R spikes.txt samples.txt parameters.txt

When the optional output files are not generated, the command to see the spikes as a plot showing the firing of neurons as a function of the time for each neuron is reduced to

Rscript ../Rasterplots.R spikes.txt

If the basic output file name is given by the user and the optional files are also asked, their names will be modified accordingly : NAME.txt for the main file, *NAME_samples.txt* and *NAME_parameters.txt* for the optional files. Thus to run the sript these names should be used instead

Rscript ../Rasterplots.R NAME.txt NAME_samples.txt NAME_parameters.txt

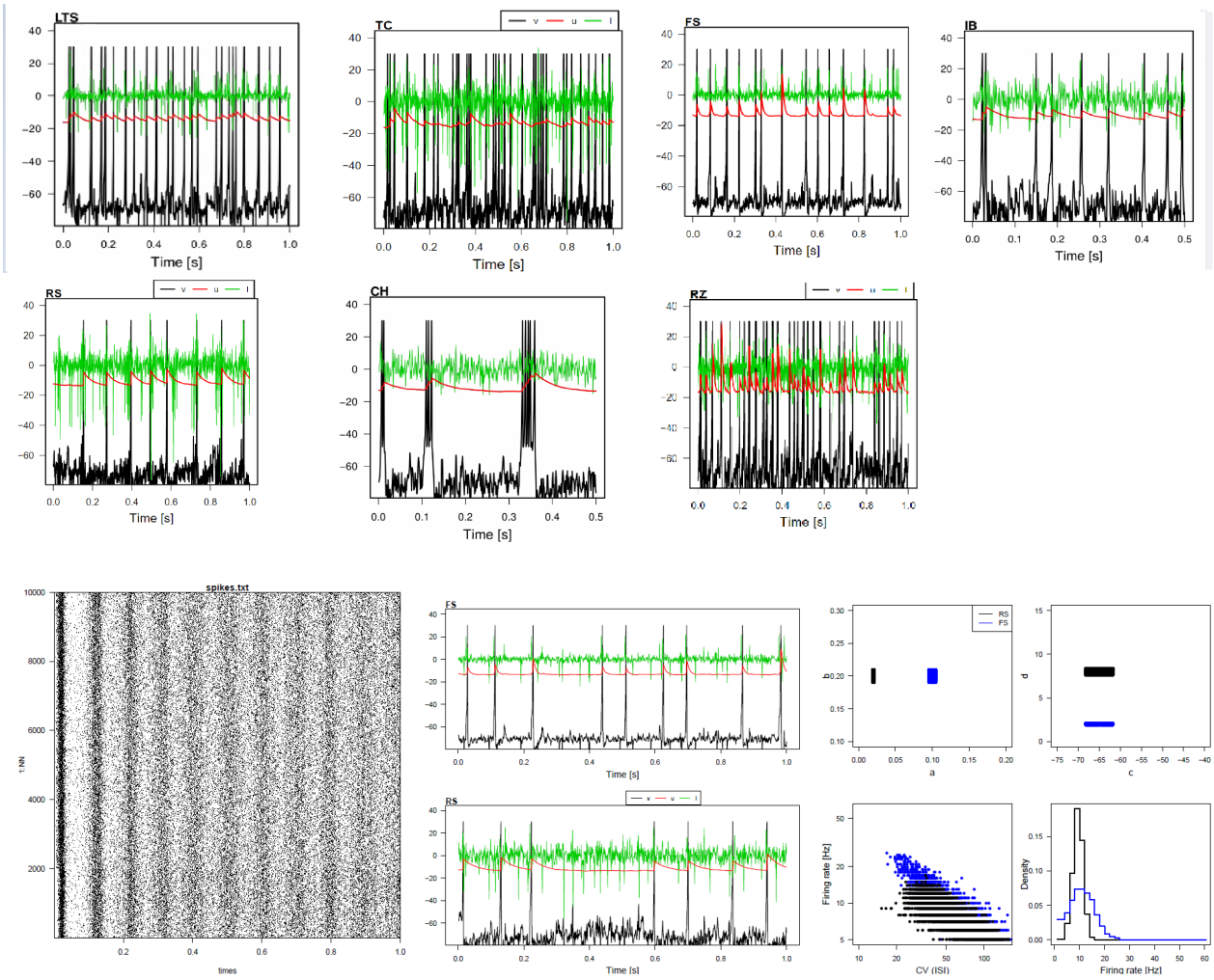
3.2 Main commands and results

The optional file samples.txt allow us to plot the values u and v for a neuron of each type and then obtain a firing pattern, which is shown in the picture below. However these graphs are made for one neuron chosen among ten thousand, so they are subject to random variation and the most important to see is the firing pattern on the spikes plot.

For the basic command line "./neuron_network -c -p 0.8 -t 1000 -L 20 -N 10000", we can obtain the following plots, whose significations are given in the Tendencies paragraph.

4 Conceptualisation

We chose to model our network by two vectors. The first contains a list of pointers to neurons. The second contains a map representing the connections of each neuron (pointers to neurons that influences the neuron of the corresponding index within the first container) and the intensity of the connection. We didn't separate the inhibitory neurons from the excitatory ones



in two specific vectors, because they form the entire network together. However, the Neuron class is polymorphic, so the functions are specific to the sub-type of neurons.

4.1 Connect the neurons

The number of connections that each neuron owns is determined by the mean connectivity (*lambda*) and the model chosen. Depending on the model, all neurons have either the same number of connections (*constant model*) or a distribution based on the mean connectivity (*basic and overdispersed models*). Since we use a random distribution, we must take the minimal value between the result obtained and the number of other neurons in the network. Otherwise, the network would have attempted to create more connections than there are neurons.

We connect the neurons by browsing the vector and choosing one at random, which will target the neuron corresponding to the current index. If the neuron chosen is the neuron targeted or if it is already connected to the target, we try to take the next neuron in the list (and so on until a neuron is available): this trick helps to avoid infinite loops, by throwing an error if all neurons are connected. However the program rarely has to use this trick if the mean number of connections between the neurons is small comparing to the number of neurons in the network. If the number of connections is large enough compared to the number of neurons (e.g. nine connections per neuron, for a network of ten neurons), it uses this trick more. However

in most simulation cases, it makes the program quicker and is not really a problem, as the neurons are created randomly and one is not more favored than another.

4.2 Output files

We chose to take the main output file as an attribute in the Simulation class, since we have to write in it at each step during the simulation. As the other files are optional, we decided not to take them as attributes. The parameter file only needs to be opened once (at the end of the simulation). While the sample file has to be modified at each step.

4.3 Errors handling

All the possible errors done by the user while giving the command lines are handled in the Simulation class. Certain inconsistencies can be easily corrected, so the program just displays a warning message (for example if the given neuron number is smaller than the mean connectivity we decrease it to match the number of neurons and we still launch the simulation) and corrects the mistake.

We first wanted to create our own Error class, to make a difference between errors that have to stop the program and errors that only have to be signaled but can easily be corrected. We then changed our minds, because by creating a new class, we were loosing some interesting features of the actual exceptions. As our project is not very big, we decided to handle the errors with the basic std::exception class. Furthermore, the program is stopped if we throw an error, so we couldn't handle our warnings with this strategy.

4.4 Testing

Some tests were created in order to test all non trivial functions, to control that they work as expected and produce the right results. It is mostly checking if the variables after launching the function have indeed evolved (update for Neuron, readLine in Simulation, synapticCurrent in Network). However to test the creation of the network we were interested in many aspects, namely if the size of the vector containing all neurons corresponds to the size of the vector containing their connections and intensity, and to the size given to create the network. We also searched in the vector of connections for each neuron if it is not connected to itself. Furthermore, to avoid segmentation faults, we also checked that no nullptr is contained in all vectors. To insure that these tests do not take too much time, the number of neurons in the network given is small.

4.5 Limitations

We tried to launch the simulation with a number of neurons near the reality of our brain (10^9). Unfortunately it exceeds the virtual memory (for one of our computer with 16GB RAM + 32GB swap), so we decided to put a maximum of 10^7 on the product of the number of neurons and the mean connectivity, to ensure that the computer can manage the desired network.

The neuron types RZ and TC were considered as excitatory. However, the model does not explicitly indicate them as such which is why they do not follow the theoretical patterns of Izhikevich well and we cannot observe their specific patters which were described by Izhikevich.

5 Tendencies

By testing many commands we could notice that there are some tendencies that are visible, for each type of neurons and for the entire simulation.

5.1 General "rules"

To obtain a good pattern, it is better to run the simulation with a ratio of 20% to 30% inhibitory neurons . If we have less excitatory, the neurons won't be firing as often as with a good ratio and some types won't be firing at all.

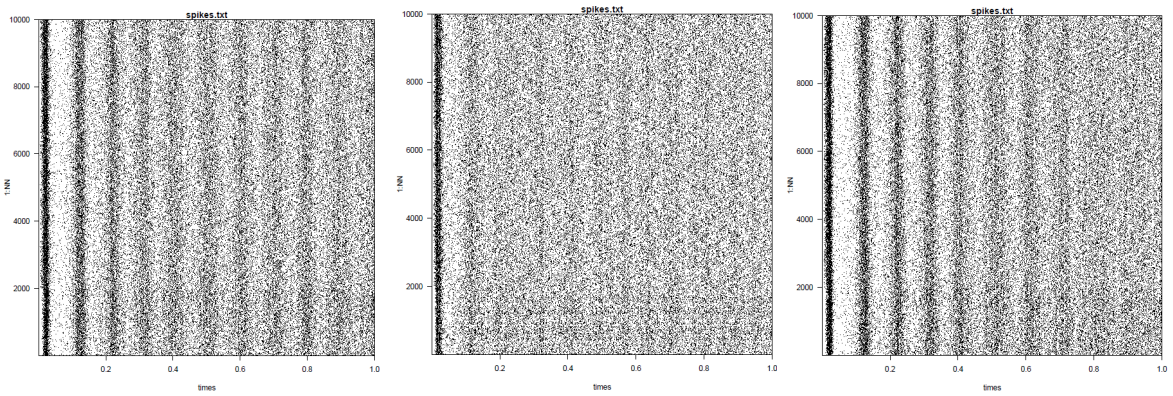
To increase the interactions between the neurons, it is better to increase the intensity than the connectivity.

5.2 Output plots

Plot 1: Representation of the neurons that are firing with a dash. This is the most representative plot of what happens in the network. We can distinguish the alpha rhythms well at the beginning and then the frequencies become higher as it tends to the gamma rhythms. The longer we let the simulation run, the more diluted the global frequency is. The best intensity to see these tendencies is around 11-12.

Plot 2: Representation of the evolution of the variable v of the neuron, which is the membrane potential over time, and every peak corresponds to a shot. Different patterns may appear depending on the neuron type (cf. Izhikevich) and some random variation (since only one neuron in the network is shown). The evolution of variable u shows peaks every time the neuron fires. The intensity for each step of time depends on the nature of the neuron (inhibitory or not) and the connected neurons firing.

Plot 3: Information about the single neurons. The two top graphs plot the parameters of the different types. In the bottom left graph, we can see the firing rate given the Coefficient Variable (interspike interval), which is the interspike interval deviation over the mean interspikes. The last graph shows the density of neurons given the firing rates, which gives information about the distribution of the firing frequency according to the neuron types. We can notice that the RS type has low deviation. The types, RZ and TC, have a low interspike time and a the highest firing rate. The RS and IB have a high interspike interval and a low deviation. We can also notice that the LTS fire more often than the FS. The FS type has a high deviation. This corresponds more or less to what we expect, except for RZ and TC.



(a) basic (-b) **(b)** over-dispersed (-o) **(c)** constant (-c)

Figure 1: Different models of connection obtained with the same command line but changing the model "./neuron_network -c -p 0.8 -t 1000 -L 20 -N 10000"

References

1. IZHIKEVICH, E. M.: Simple Model of Spiking neurons. *IEEE TRANSACTIONS ON NEURAL NETWORKS*. [N.d.], vol. 14, no. 6.
2. ROUGEMONT, J.: *PROJET 2: RÉSEAU DE NEURONES*. [N.d.]. Instructions.
3. *Image : Réseau De Neurones Artificiels*. [N.d.]. Available also from: https://www.freepng.fr/png-4j8gg0/.