CN510: Principles and Methods of Cognitive and Neural Modeling

Simple Models of Point Neurons
Mathematical and Computational Methods

Lecture 3

Instructor: Anatoli Gorchetchnikov <anatoli@bu.edu>

The Current Prevailing View

Most neuroscientists agree on the following:

- The neuron is the basic signaling unit in the brain
- Different parts of the brain have different functional roles (e.g. auditory cortex, visual cortex, motor cortex, etc.)
- The different brain regions project to each other in a fairly precise fashion that is affected by experience
- A given brain region has roughly the same role across individuals
- Damage to one part of the brain may lead to long-term reorganization such that another region takes over the functionality of the damaged region, but there are limits to this type of adaptation
- Most non-trivial tasks involve several different brain regions interacting in a task-specific network

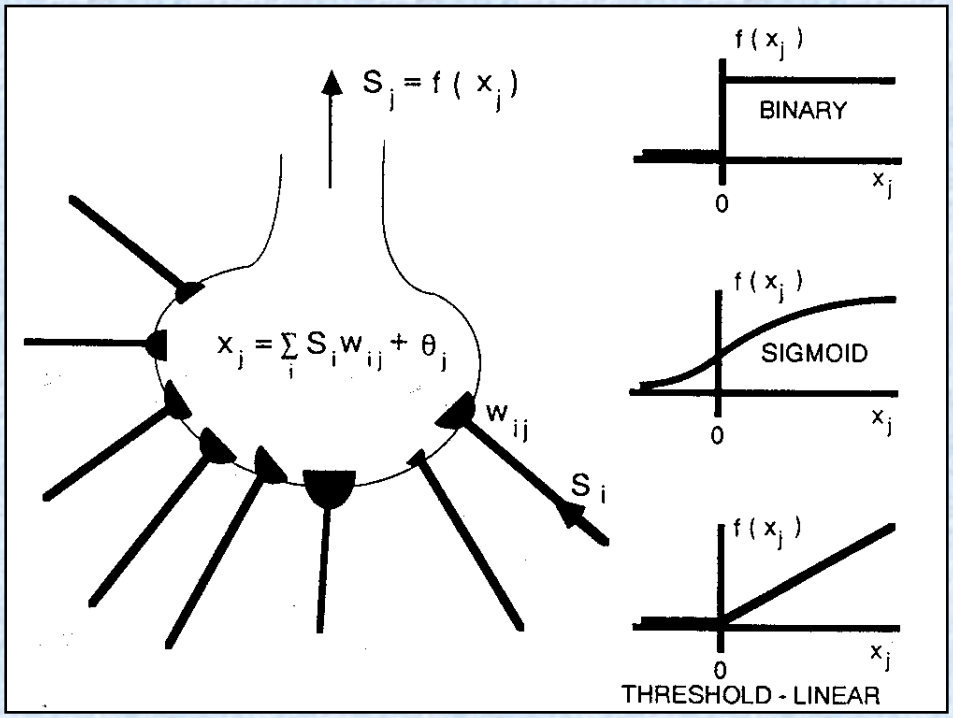
McCulloch-Pitts Neuron (1943)

"A logical calculus of the ideas immanent in nervous activity"

Networks can be configured to perform arbitrary logical functions

Proposed as a computer architecture

Weighted addition of its inputs $\sum_i S_i w_{ij}$ with optional threshold or bias value θ_j



$$\sum_i S_i w_{ij} = \vec{S} \cdot \vec{w} =$$

$$= \left\|\vec{S}\right\| \cdot \left\|\vec{w}_j\right\| \cos\left(\vec{S}, \vec{w}_j\right)$$

Energy Pattern

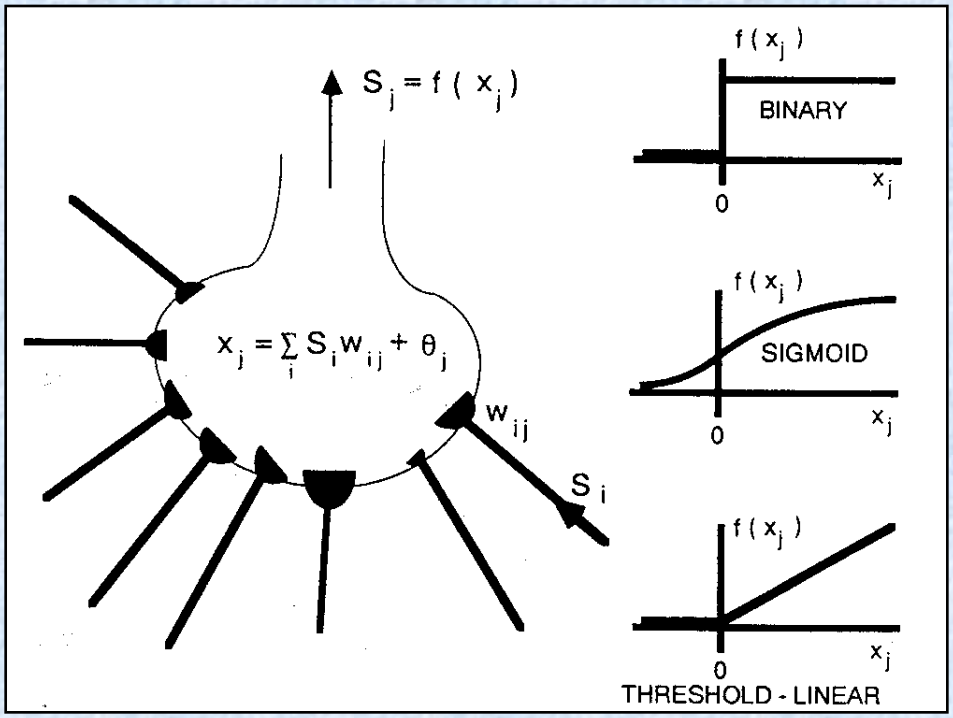
McCulloch-Pitts Neuron (1943)

This activation x_j representing membrane potential is often run through a signal (output) function $f\left(x_j\right)$

Output then acts as input to other neurons

Note that this scheme is equally applicable to spiking neurons given a certain signal function:

$$f\left(x_j\right)=\begin{cases}0 \; if \;\; x_j < x_\theta \\ 1 \; if \;\; x_j \geq x_\theta\end{cases}$$



$$\sum_i S_i w_{ij} = \vec{S} \cdot \vec{w} =$$

$$= \left\|\vec{S}\right\| \cdot \left\|\vec{w}_j\right\| \cos\left(\vec{S}, \vec{w}_j\right)$$

Energy Pattern

Simple Algebraic Model

The activity y_j of a neuron can be written as a simple algebraic equation:

$$y_j = \sum_i x_i w_{ij} + \theta_j$$

This neuron does not have any "memory" or trace of recent events, it only responds to a currently present inputs

Change of input leads to instantaneous change of neuronal activation – no temporal dynamics

Bad: Crude approximation, only valid when input changes on the time scale that is much slower than membrane potential change

Good: Very simple and computationally efficient, in most cases the system can be solved analytically

Simple Algebraic Model in Vector Terms

The activity $\boldsymbol{Y}$ of a neuronal population can be written as a vector of y_j:

$$Y = X \cdot W + \Theta$$

For linear signal function:

- Efficient linear algebra methods can be applied to solve the system for any input vector $\boldsymbol{X}$ given weight matrices $\boldsymbol{W}$ and biases $\boldsymbol{\Theta}$ even in the cases with many populations and recurrent connections in the network

For nonlinear signal function the solution becomes more complicated, but since in neural models these functions are non-decreasing, the unique solution usually exists

Why not stop here?

Simple Algebraic Model with Delays

Introducing axonal delays between neurons adds some dynamics to the system but also makes it much less tractable

$$y_j(t) = \sum_i x_i(t - n_i \Delta t) w_{ij}(t - \Delta t) + \theta_j(t)$$

This is a slightly simplified version of *CogExMachina* framework by HP Labs

Advantages:

- Some dynamics (and history) is included
- Computation fits well with the digital hardware

What if we want to change activation based on the previous state?

$$y_j(t) = y_j(t - \Delta t) + \sum_i x_i(t - n_i \Delta t) w_{ij}(t - \Delta t) + \theta_j(t)$$

But Is It Still an Algebraic Model?

$$y_j(t) = y_j(t - \Delta t) + \sum_i x_i(t - n_i \Delta t) w_{ij}(t - \Delta t) + \theta_j(t)$$

In other words what we compute on every step is the difference of activation, its change

Given the fixed time step we can write it as

$$\frac{\Delta y_j(t)}{\Delta t} = \sum_i x_i(t - n_i \Delta t) w_{ij}(t - \Delta t) + \theta_j(t)$$

But this is nothing more than a discrete Euler approximation of a continuous ordinary differential equation

$$\varepsilon \frac{dy_j(t)}{dt} = \sum_i x_i(t - d_i) w_{ij} + \theta_j$$

Why Differential Equations?

Temporal variations are important: many stimuli are coded as change in activity rather than its absolute level

Rates of change and oscillatory patterns are important: many actions are controlled by oscillation and dysfunction of oscillatory patterns leads to diseases

"A process can not be understood by stopping it. Understanding must move with the flow of the process..."

First law of Mentat.
Frank Herbert's "Dune"

Differential equations relate a rate of change of some process to other processes that change in time

What Can We Do with DE?

Exact analytical solution

- Perfect, but hard to obtain for non-trivial networks

Proof of existence of such a solution

- Good as a validation tool, but also hard to obtain

Qualitative analysis of behavior

- Predict the system behavior from the properties of equations

Numerical simulation

- Always works, but need to keep in mind accuracy, speed, and other hardware limits

Simplification

- Fix some slow variables, average out the change in some fast variables, analyze/simulate the resulting simpler system

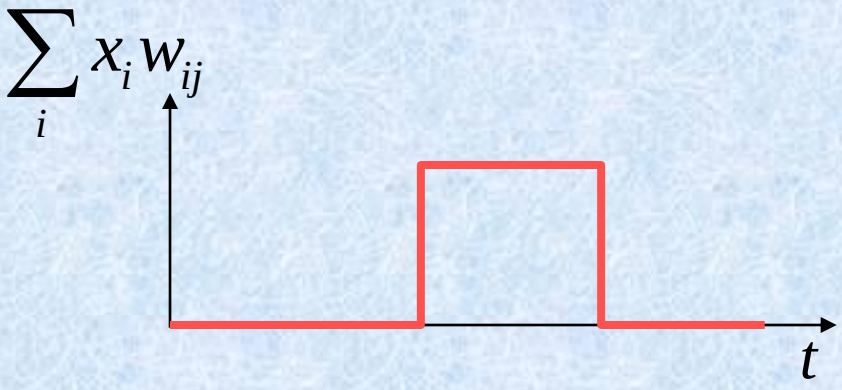
Integrator

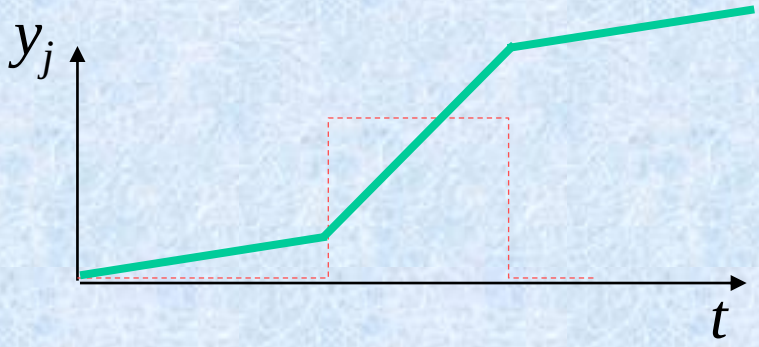
So, the activity y_j of a neuron can be written as a differential equation:

$$\varepsilon \frac{dy_j}{dt} = \sum_i x_i w_{ij} + \theta_j$$

Now activation changes with input, but it has some "memory" of what previous inputs did to the neuron

Here it grows with excitatory inputs and decays with inhibitory inputs

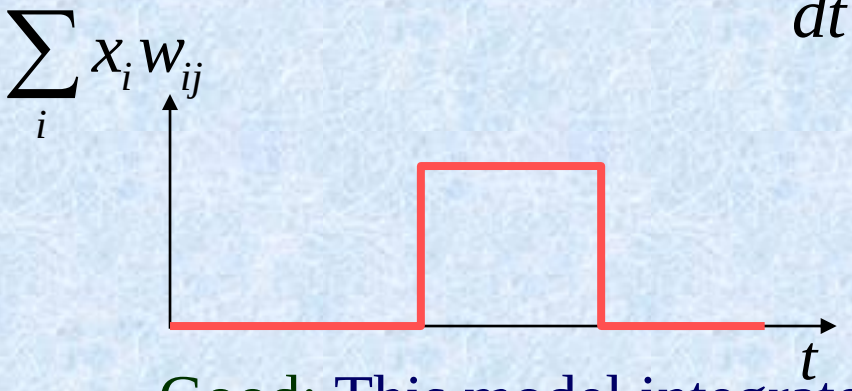




Integrator

Keeping explicit bias is not good, because it leads to constant activation change when there is no input – drop it

$$\varepsilon \frac{dy_j}{dt} = \sum_i x_i w_{ij}$$



Good: This model integrates inputs like real neurons do

Bad: Unlike real neurons, here the integration will continue indefinitely without bounds on activation

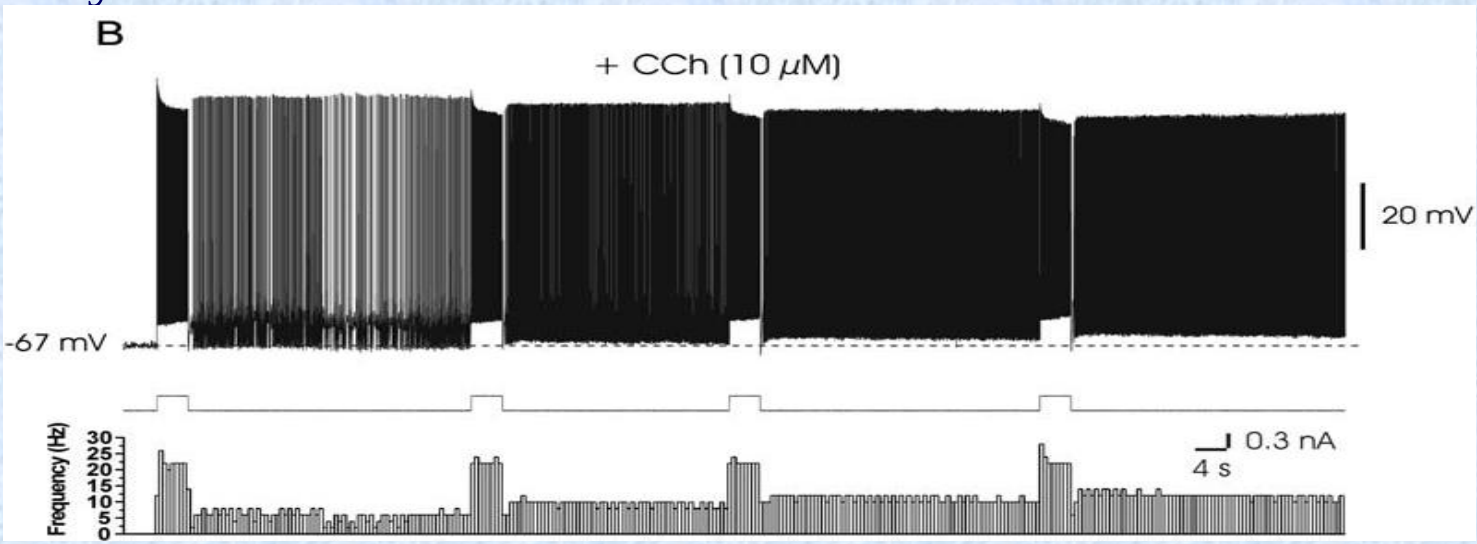
Also, the majority of neurons without inputs decay back to resting potential

Integrators in the Brain

Some cells (or cell parts) do show plateau potentials or firing rate patterns (equivalently: some cells act as integrators) in the presence of appropriate neuromodulators

e.g. alpha motorneurons in presence of serotonin

or entorhinal layer V pyramidal cells in presence of acetylcholine



but most cells return to resting potential after the input shuts off

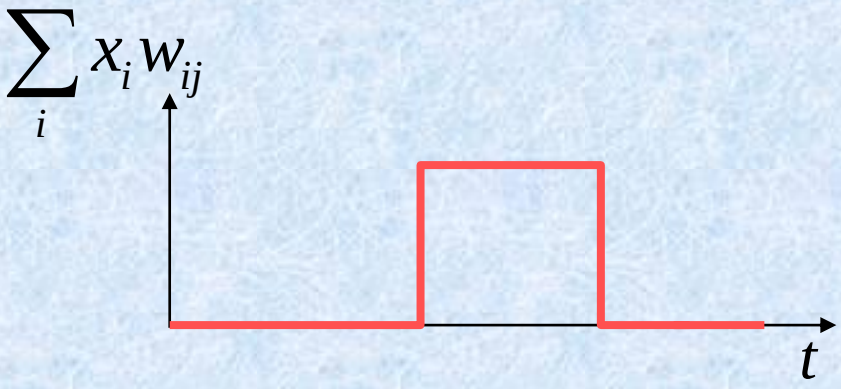
Leaky Integrator

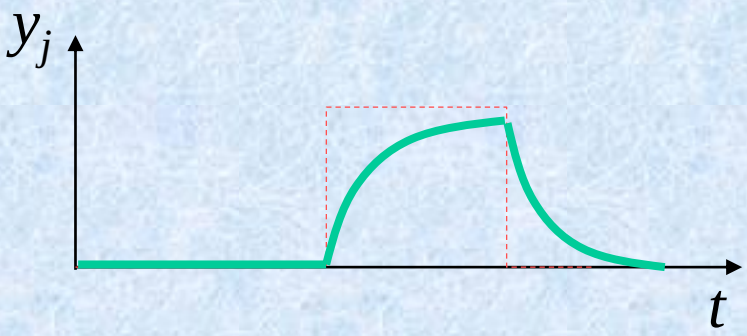
$$\varepsilon \frac{dy_j}{dt} = \sum_i x_i w_{ij}$$

We need to put an upper bound on y_j and make it decay to *0* without inputs

One shot for two rabbits:

$$\varepsilon \frac{dy_j}{dt} = -Ay_j + \sum_i x_i w_{ij}$$





As y_j grows Ay_j gets more powerful and balances the inputs

Leaky Integrator

$$\varepsilon \frac{dy_j}{dt} = -Ay_j + \sum_i x_i w_{ij}$$

is called leaky integrator

The input is integrated over time and the leak Ay_j counteracts this integration and causes the activation to go back toward *0* when the input is shut off

Note that this will work equally well for inhibitory (negative inputs) because then y_j becomes negative and $-Ay_j$ positive

Note also that here we assume *0* as the resting potential of the neuron:

- Advantage – simpler equations
- Disadvantage – need to convert the model output to compare with the experimental data

Leaky Integrator at Equilibrium

If there is no feedback in the network, we can often see what happens to the equation at equilibrium by setting $dy_j/dt = 0$:

$$\varepsilon \frac{dy_j}{dt} = -Ay_j + \sum_i x_i w_{ij} + \theta_j = 0$$

Solving for y_j yields:

$$y_j = \frac{1}{A}\left(\sum_i x_i w_{ij} + \theta_j \right)$$

Note that with $A = 1$, this is the same algebraic equation for a neuron's activity as in McCulloch-Pitts neuron

Note also that solving at equilibrium determines a critical point where $dy_j/dt = 0$ but unless it is a stable point the system will never get there

Leaky Integrator at Equilibrium

In many neural networks, activation equations are assumed to equilibrate very quickly relative to weight changes

$$0 < \varepsilon << 1$$

in the differential equation

$$\varepsilon \frac{dy_j}{dt} = -Ay_j + \sum_i x_i w_{ij} + \theta_j$$

In this case, the algebraic equation is a valid approximation to the differential equation

As long as the inputs x_i are changing slower than activations y_j

How can the feedback in the network interfere?

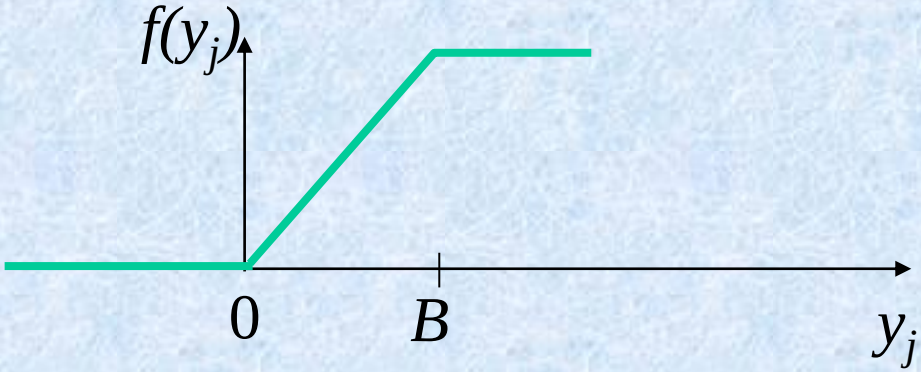
Leaky Integrator

$$\varepsilon \frac{dy_j}{dt} = -Ay_j + \sum_i x_i w_{ij}$$

The equilibration level of leaky integrator depends on the strength of the input

Might be a problem for strong inputs because it will push the neuron beyond reasonable membrane potential

Can be solved by choosing bounded output function $f(y_j)$:



Here the output is non-zero for any positive activation

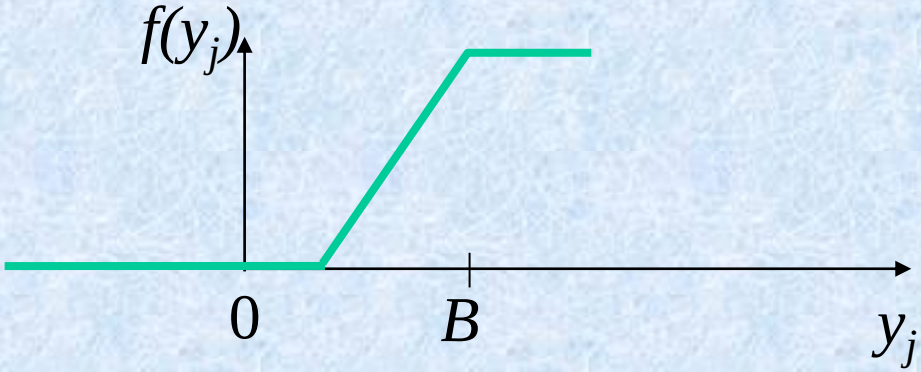
Leaky Integrator

$$\varepsilon \frac{dy_j}{dt} = -Ay_j + \sum_i x_i w_{ij}$$

The equilibration level of leaky integrator depends on the strength of the input

Might be a problem for strong inputs because it will push the neuron beyond reasonable membrane potential

Can be solved by choosing bounded output function $f(y_j)$:



Here the activation (and therefore input) should be above threshold to produce output

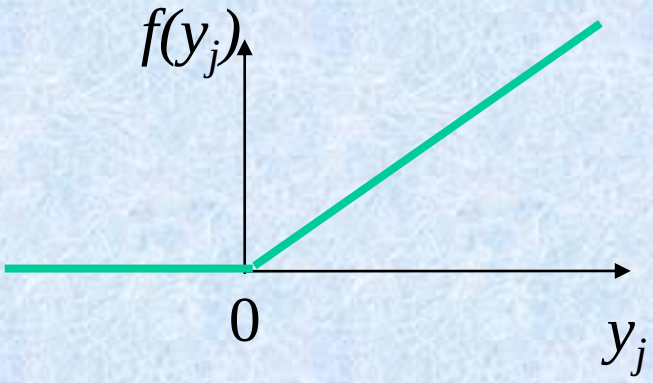
Leaky Integrator with Shunting Bound

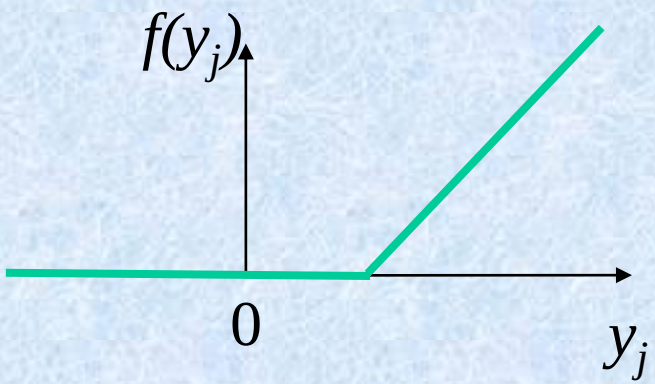
If we want the upper bound of activity to be independent on the input strength:

$$\varepsilon \frac{dy_j}{dt} = -Ay_j + \left(B - y_j\right)\sum_i x_i w_{ij}$$

Here $\left(B - y_j\right)$ term gets smaller as y_j grows

There is no need for bounding output function, we can use simple rectification or rectification with threshold





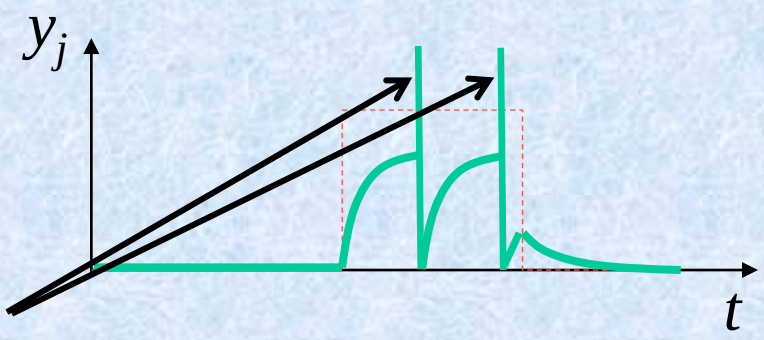
Leaky Integrate-and-Fire

$$\varepsilon \frac{dy_j}{dt} = -Ay_j + \sum_i x_i w_{ij} \quad \text{or} \quad \varepsilon \frac{dy_j}{dt} = -Ay_j + \left(B - y_j\right) \sum_i x_i w_{ij}$$

Leaky integrator is converted into leaky integrate-and-fire neuron by adding threshold condition as an output function:

If $y_j > \chi_j$, then emit spike and set $y_j = y_{Rj}$

χ_j is a spiking threshold and y_{Rj} is a refractory potential

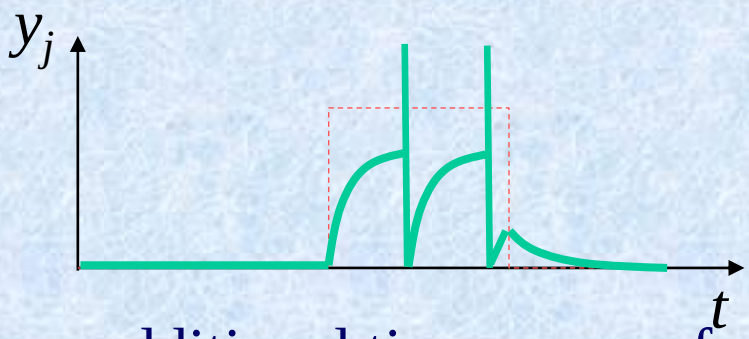


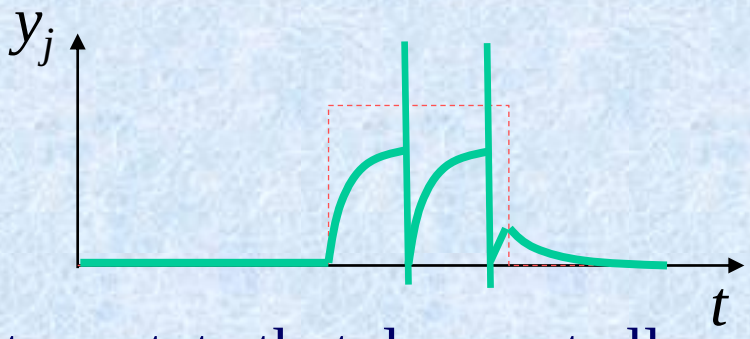
Note that “spikes” in the plot a only drawn for convenience, they are not part of dynamics

Leaky Integrate-and-Fire

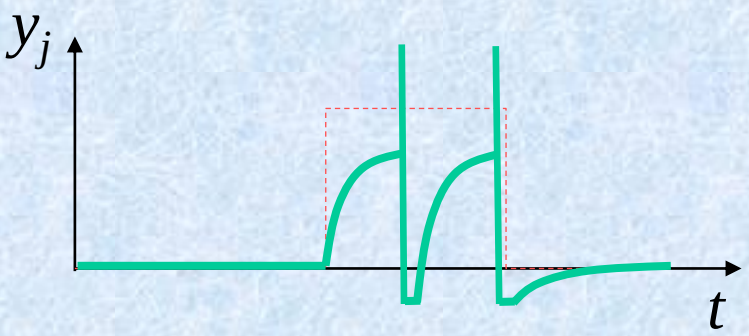
Variations of this neuron include:

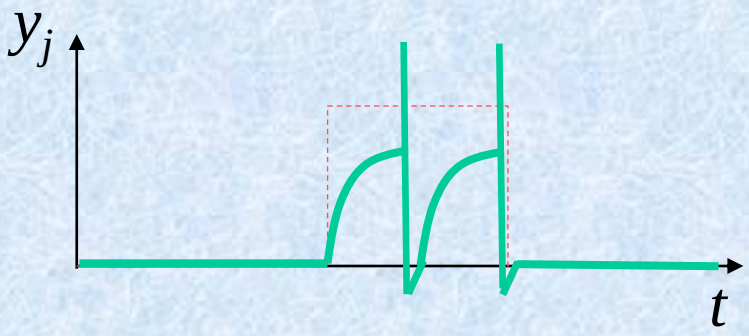
- refractory potential being equal to resting potential,





- additional timer on a refractory state that does not allow y_j to change immediately after spike,





- and many others

Initial Conditions

To solve $$\varepsilon \frac{dy_j}{dt} = -Ay_j + \sum_i x_i w_{ij} = -Ay_j + I$$

we need to specify initial conditions, e.g. $y_j(0)=y_0$

You will solve it in the homework…

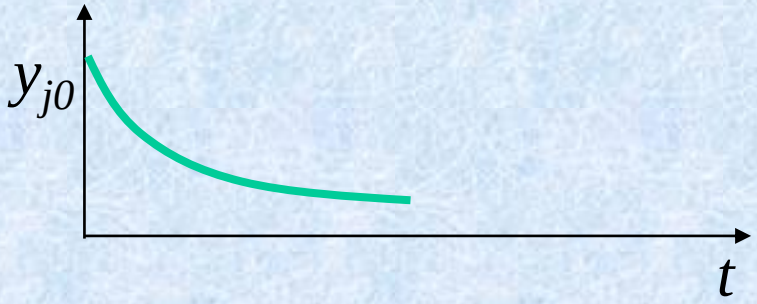
What good is it to solve it if we only can do it for simplest equations?

Rotter-Diesmann approach for spiking neurons uses exact solutions like this and combines them together to achieve fast numerical simulations of large networks in NEST

Rotter-Diesmann Approach

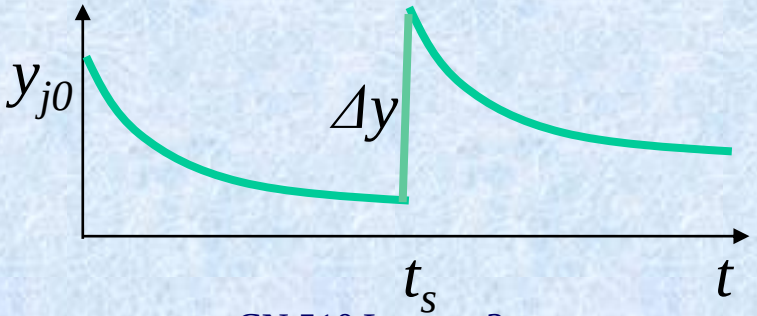
Let's say the analytical solution of neuronal activation without input spikes follows

$$y_j \qquad y_j = y_{j0} e^{-\frac{t - t_{j0}}{\tau}}$$

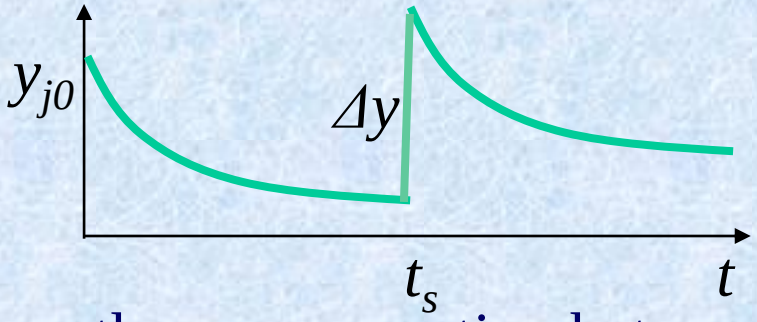


and each arriving spike changes activation instantaneously

$$y_j \qquad y_{js+} = y_{js-} + \Delta y$$



Rotter-Diesmann Approach



Trajectory follows the same equation between input spikes

In fact it is the same exponent, just shifted to the right

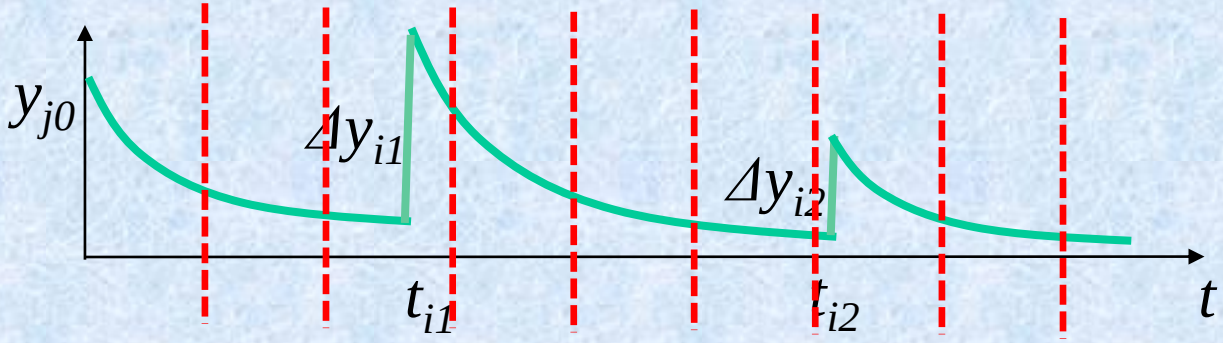
Then we can totally ignore the dynamics between arriving spikes

For each arriving spike compute $y_{j(this)} = y_{j(last)} e^{-\frac{t_{this} - t_{last}}{\tau}}$

Then set $y_{j(last)} = y_{j(this)} + \Delta y$ and wait for the next spike

Furthermore, for a set of different inputs use a set of Δy_i and you can get a trace of thousands of synaptic inputs with minimal computation

Rotter-Diesmann Approach



Usually you want to output the results of your simulation

Then you need to compute activation not only at the time of input arrival, but also on the time grid of your output

You will try a simple version (no input spikes for now) of this in the homework

Remember the difference: in analytic solution for every point you use the initial conditions at $t=0$, in iterative Rotter-Diesmann for each point you use a value in the previous point as a new initial condition

Issues with Numerical Simulations

Computers are digital:

Floating point representation: $Significant\ digits \times base^{exponent}$

- 32 bit fp number has 16.7 millions of possible values
- These values are packed closer together around 0 (negative exponent) and further apart as exponent grows

Immediate consequences:

- There are no infinitely small or infinitely large numbers in computer
- Smaller numbers are more precise
- Using equal operator on floating point numbers is asking for trouble: Is $2 \times 2 = 4$?

If *2* approximates to *2.000001*, representation for *4.000004* exists, and *4* approximates to *3.999999* then no!

Stability of Numerical Integration

As we do not have infinitely small numbers in computer, we must approximate differential with difference

$$\varepsilon \frac{dy_j}{dt} = -Ay_j + \sum_i x_i w_{ij}$$

changes to

$$\varepsilon \frac{\Delta y_j}{\Delta t} = -Ay_j + \sum_i x_i w_{ij}$$

Δt is finite, and we treat differential as constant over this time

The faster y_j changes, the more error we accumulate for the same Δt

If y_j changes too fast for the given Δt, then our numerical integration method is unstable for a given equation and Δt

Euler Methods

$$y_j(t) = y_j(t-1) + \Delta t\left(-Ay_j(t-1) + \sum_i x_i(t-1)\,w_{ij}(t-1)\right)$$

Forward because only uses values available from the previous time step

Backward methods estimate the value at the end of time step first, and then use it instead of the previous one

$$y_j(t) = y_j(t-1) + \Delta t\left(-Ay_j(t) + \sum_i x_i(t-1)\,w_{ij}(t-1)\right)$$

Requires a solution of a linear equation on every step: computationally much more intensive

Advantage: backward methods are much more stable

Stability of Numerical Integration

Variable time step integration methods reduce Δt for fast-changing parts of the solution, increase it for slow-changing parts

Disadvantage: if you simulate a large population of neurons and want them on the same clock, then the time step has to be the shortest from all neurons

As a result, overhead of adjusting the step kills the gain of using it

Stiff equations – contain some variables that lead to very fast changes in solutions, example – Hodgkin-Huxley equations

Matlab has several `odeXX` methods for various types of equations to ensure stability

Systems of Equations

Let's have two neurons: $\frac{dy}{dt} = -A_y y + w_{xy} x$

$$\frac{dx}{dt} = -A_x x + w_{yx} y$$

Here the general solution is

$$y(t) = a\exp(\lambda_1 t) + b\exp(\lambda_2 t)$$

$$x(t) = c\exp(\lambda_1 t) + d\exp(\lambda_2 t)$$

where a, b, c, d are constants (possibly complex) depending on parameters and initial conditions

λ_1 and λ_2 are eigenvalues of the matrix $M = \begin{pmatrix} -A_y & w_{xy} \\ w_{yx} & -A_x \end{pmatrix}$

Eigenvalues

$$Mv = \lambda v$$

Often finding vector v and scalar λ in this equation is beneficial for analysis, so they got special names: eigenvector and eigenvalue, respectively

To find them rearrange $(\lambda I - M)v = 0$

Here one solution is $v=0$; if $\lambda I - M$ is invertible then this is the only solution

Matrix is non-invertible if it has zero determinant

Thus to find the eigenvalues we need to find the roots of characteristic polynomial

$$p(\lambda) = \det(\lambda I - M) = 0$$

Eigenvalues

In our case this polynomial will be

$$p(\lambda) = \lambda^2 - \left(-A_y - A_x\right)\lambda + \left(A_y A_x - w_{yx} w_{xy}\right) = 0$$

Depending on the parameters these roots can be real or complex

Note that if the real parts of eigenvalues are positive the activations will grow without bounds

If they are negative the activations will die out

The only meaningful behavior (without other inputs) will be if they are zero, then the network will oscillate forever

Next Time

Critical points, stability analysis for points and systems of equations, 1D and 2D phase-plane analysis

Readings: Izhikevich, E.M. (2007). *Dynamical Systems in Neuroscience: The Geometry of Excitability and Bursting*. Cambridge, MA, MIT Press. Chapter 1.