Introduction

This simulation illustrates the basic properties of neural spiking and rate-code activation, reflecting a balance of excitatory and inhibitory influences (including leak and synaptic inhibition).

Orientation to the Software (ControlPanel and Views)

As this is the first simulation project in the textbook, we begin with some introductory orientation (see the Emergent documentation for more complete information). All of the major controls and parameters for the simulation are contained within the ControlPanel (NOTE: these links work when viewing this documentation within the emergent simulator, and not otherwise) object located in the middle of the 3 panels visible in the main project window (you can access it from the tab at the top of the middle panel). The right panel contains various 3D graphical displays of simulation data, including the network (NetView or Network View) and various graphs and grid-like displays (Graph view, Grid view).

In this simulation, there are two different ways to view the results, selectable by the tabs at the top of the right side of the window:
The **Network** tab shows the (very simple) network that is being simulated, with a single sending (input) neuron (at the bottom) that sends activation to the receiving neuron (at the top). We are primarily concerned with how the receiving neuron responds to the activation input from the sending neuron.

The **CycleOutputData** tab shows a graph of the receiving unit's main variables (see below for details) over time, in response to the sending activation.

We will see this single input being turned on and then off again, and observe the response of the receiving neuron. To see this, we can run the simulation.

At the bottom of the **ControlPanel** are 4 buttons: Init, Run, Step, Stop:

1. **Init** -- initializes the graph display and starts the simulation over from wherever it might have left off.
2. **Run** -- runs the full set of cycles of activation updating (updating of the equations that govern the behavior of the neural unit), displaying the results in the Network and CycleOutputData frames on the right hand side of the window.
3. **Step Cycle** -- runs one single cycle of activation updating (in more complex models, multiple levels of stepping will be available)
4. **Stop** -- if running, this will stop running.

Text like this indicates specific actions that you need to perform in the simulator. In this case, hit **Init** and **Run**, and look at the **Network** display.

You should see that very shortly after the input neuron comes on (indicated by the yellow color), the receiving neuron is activated by this input, firing a sequence of discrete action potentials or *spikes*. To get a better idea of the precise trajectory of this activation, it is much more convenient to use the **Graph View**, which displays the information graphically over time, allows multiple variables to be viewed at the same time, and even allows multiple runs (e.g., with different parameters) to be compared with each other.

**The Graph View**

- Press the **CycleOutputData** tab in the right panel to display the graph view display.

Only the excitatory and leak currents are operating here, with their conductances \( g_{\text{bar}.e}, g_{\text{bar}.l} \) and reversal potentials \( e_{\text{rev}.e}, e_{\text{rev}.l} \) as shown in the control panel.

- Press the **Init** button and then the **Run** button on the control panel to display a new graph.

This produces a plot using the current (default) parameters. You should see various lines plotted over 200 time steps (cycles) on the X axis.

Here is a quick overview of each of the variables -- we'll go through them individually next (see **GraphHelp** for more details on how to determine what is being graphed, and how to configure it):

- **net** (black line) = net input, which is the total excitatory input to the neuron \( \text{net} = g_e(t) \times g_{\text{bar}.e} \). \( g_e(t) \) is the proportion of excitatory ion channels open, and it goes from 0 prior to cycle 10, to 1 from 10-160, and back to 0 thereafter. Because \( g_{\text{bar}.e} = .3 \) (by default), the net value goes up to .3 from cycle 10-160. The timing of the input is controlled by the parameters **on_cycle** and **off_cycle** (the total number of cycles is controlled by **n_cycles**).

- **I_{net}** (red line) = net current (sum of individual excitation and leak currents), which is excitatory (upward) when the excitatory input comes on, and then oscillates as the action potential spikes fire. In general this reflects the net balance between the excitatory net input and the constant leak current (plus inhibition, which is not present in this simulation).

- **v_m** (blue line) = membrane potential, which represents integration of all inputs into neuron. This starts out at the resting potential of .3 (= -70mV in biological units), and then increases with the excitatory input. As you can see, the net current \( (I_{\text{net}}) \) shows the *rate of change* of the membrane potential. When \( v_m \) gets above about .5, a
spike is fired, and $v_m$ is then reset back to .3, starting the cycle over again.

- **act** (green line) = activation. This shows the amount of activation sent to other units -- by default in this model it is set to discrete SPIKE mode, so it is 0 except on the cycle when $v_m$ gets over threshold. When act_fun is set to the rate-code NOISY_XX1 function, it reflects the expected rate of spiking as a number between 0-1. Note - the green line is often hidden by the act_eq line- to see it, click on the CycleOutputData graph, then open the CycleOutputData tab in the ControlPanel then uncheck all the other lines that may be hiding it.

- **act_eq** (purple line) = rate-code equivalent activation -- computes a running average of spikes per cycle in SPIKE mode, providing a measure of the rate-code value corresponding to the current spiking behavior. If a rate code like NOISY_XX1 is being used, then it is the same as act, and is hidden by that value (all you see is the green line).

- **adapt** (orange line) = adaptation variable -- increases during spikes, and decays somewhat in between, building up over time to cause the rate of spiking to adapt or slow down over time.

---

**Software Tip:** You can click on any line in the graph log display using the red-arrow tool (click on the red arrow button in the upper-right corner of the 3D view window to see the exact values at the point where you clicked (as displayed in the graph caption at the bottom). You can also click on the light green outer frame, and then do the context menu (right mouse button or Ctrl + left mouse on mac) and select DataTable/Edit Dialog to obtain a spreadsheet view of all the data being plotted in the graph. You can then scroll through this data to find any value you might want to find.

---

Use the graph click on the black **net** line to verify that it goes to a value of .3 when the input goes on.

---

**Spiking Behavior**

The default parameters that you just ran show the spiking behavior of a neuron. This is implementing a modified version of the Adaptive Exponential or AdEx model, which has been shown to provide a very good reproduction of the firing behavior of real cortical pyramidal neurons. As such, this is a good representation of what real neurons do. We have turned off the exponential aspect of the AdEx model here to make parameter manipulations more reliable -- a spike is triggered when the membrane potential $V_m$ crosses a simple threshold of .5. (In contrast, when exponential is activated (with the spike_misc.ex flag), the triggering of a spike is more of a dynamic exponential process around this .5 threshold level, reflecting the strong nonlinearity of the sodium channels that drive spiking.)

At the broadest level, you can see the periodic green **act** spikes that fire as the membrane potential $v_m$ gets over the firing threshold, and it is then reset back to the rest level, from which it then climbs back up again, to repeat the process again and again. Looking at the overall rate of spiking as indexed by the spacing between spikes, you can see that it decreases over time -- this is due to the adaptation property of the AdEx model -- the spike rate adapts over time.

From the tug-of-war model, you should expect that increasing the amount of excitation coming into the neuron will increase the rate of firing, by enabling the membrane potential to reach threshold faster, and conversely decreasing it will decrease the rate of firing. Furthermore, increasing the leak or inhibitory conductance will tug more strongly against a given level of excitation, causing it to reach threshold more slowly, and thus decreasing the rate of firing.

This intuitive behavior is the essence of what you need to understand about how the neuron operates -- now let's see it in action.
Manipulating Parameters

Now we will use some of the parameters in the control panel to explore the properties of the point neuron activation function.

- Click on the ControlPanel and take a moment to familiarize yourself with the parameters (you can hover the mouse over the label for each parameter to view a brief description of what it is).

- **Software Tip:** All edit dialogs like the control panel have at least two buttons across the bottom: Apply (which applies any changes you have made to actually set the parameters) and Revert (which reverts to the previously applied values, which is useful if you have accidentally typed in a value that you didn't want). See the wiki Edit dialog for more information.

## Excitatory

First, we will focus on $g_{\text{bar} \cdot e}$, which controls the amount of excitatory conductance. In general, we are interested in seeing how the unit membrane potential reflects a balance of the different inputs coming into it (here just excitation and leak), and how the spiking rate responds to the resulting membrane potential.

- Increase $g_{\text{bar} \cdot e}$ from .3 to .4 (and press the Apply and Run buttons to see the effects). Then observe the effects of decreasing $g_{\text{bar} \cdot e}$ to .2 and all the way down to .1.

- **Software tip:** It is often useful to overlay different Runs on top of each other in the graph log, which will happen naturally. When you want to clear the log, press the Init button in the ControlPanel.

**Question 2.1:** Describe the effects on the rate of neural spiking of increasing $g_{\text{bar} \cdot e}$ to .4, and of decreasing it to .2, compared to the initial value of .3 (this is should have a simple answer).

**Question 2.2:** Is there a qualitative difference in the neural spiking when $g_{\text{bar} \cdot e}$ is decreased to .1, compared to the higher values -- what important aspect of the neuron's behavior does this reveal?

By systematically searching the parameter range for $g_{\text{bar} \cdot e}$ between .1 and .2, you should be able to locate the point at which the membrane potential just reaches threshold.

**Question 2.3:** To 2 decimal places (e.g., 0.15), what value of $g_{\text{bar} \cdot e}$ puts the neuron just over threshold, such that it spikes at this value, but not at the next value below it?

**Question 2.4 (advanced):** Using the equation for the equilibrium membrane potential from the Neuron chapter, compute the exact value of excitatory input required to just reach threshold, showing your math (note that: $g_1$ is a constant .3; $g_e$ is 1 when the input is on; inhibition is not present here and can be ignored) -- this should agree with your empirically determined value -- also be sure that you have turned off the ex flag so that the threshold is simple and deterministic at Vm of .5.
Leak

You can also manipulate the value of the leak conductance, $g_{\text{bar}.1}$, which controls the size of the leak current -- recall that this pulls the opposite direction as the excitatory conductance in the neural tug-of-war.

⇒ Press the Defaults button on the control panel to restore the default parameters, then manipulate the $g_{\text{bar}.1}$ parameter in .1 increments (.4, .5, .2 etc) and observe the effects on neural spiking.

**Question 2.5:** What value of $g_{\text{bar}.1}$ just prevents the neuron from being able to spike (in .1 increments) -- explain this result in terms of the tug-of-war model relative to the $g_{\text{bar}.e}$ excitatory conductance.

**Question 2.6 (advanced):** Use the same technique as in question 2.4 to directly solve for the value of $g_{\text{bar}.1}$ that should put the neuron right at its spiking threshold -- show your math.

Driving / Reversal Potentials

⇒ Press Defaults to restore the default parameters. Then manipulate the $e_{\text{rev}.e}$ and $e_{\text{rev}.l}$ parameters and observe their effects on the spiking rate.

You should see that decreasing $e_{\text{rev}.e}$ reduces the spiking rate, because it makes the excitatory input pull less strongly up on the membrane potential. Increasing $e_{\text{rev}.l}$ produces greater spiking by making leak pull less strongly down.

Rate Coded Activations

Next, we'll see how the discrete spiking behavior of the neuron can be approximated by a continuous rate-coded value. The blue act_eq line in the graphs has been tracking the actual rate of spiking to this point -- it goes up when a spike occurs and then decreases slowly in the interim, with the aggregate value over time becoming a closer reflection of the actual spiking rate. But the NOISY_XX1 activation function can directly compute a rate-code value for the neuron, instead of just measuring the observed rate of spiking. As explained in the Neuron chapter, this rate code activation has several advantages (and a few disadvantages) for use in neural simulations, and is what we typically use.

⇒ Press Defaults to start out with default parameters, then set the act_fun parameter to NOISY_XX1 instead of SPIKE, and Run with the various parameter manipulations that you explored above.

You should see that the green act line in the graph now rises up and then decreases slowly due to accommodation, without the discrete spiking values observed before. Similarly, the blue $v_m$ membrane potential value rises up and decreases slowly as well, instead of being reset after spiking.
**Question 2.7:** Compare the spike rates with rate coded activations by reporting the `act` values just before cycle 160 (e.g., cycle 155) for `g_bar.e = .2, .3, .4` with `act_fun = NOISY_XX1`, and the corresponding values of `act_eq` in the `act_fun = SPIKE` mode for the same `g_bar.e` values. Use the red arrow click mode to find values in the graph by clicking on the green act line at the corresponding cycle (155).

You should have observed that the `act` value tracks the actual `act_eq` spiking rate reasonably well, indicating that NOISY_XX1 is a reasonable approximation to the actual neural spiking rate.

⇒ To more systematically compare SPIKE and NOISY_XX1, do **Defaults** then hit the **SpikeVsRate Run** button in the control panel -- this will alternate between these two functions for a range of `g_bar.e` values, and plot the results in the **SpikeVsRate** graph (click to that graph to speed processing instead of watching each run in the CycleOutputData graph view).

The resulting graph shows the `g_bar.e` values on the X axis plotted against the NOISY_XX1 rate-code activation (`rate` line) and actual SPIKE rate (`spike`) on the Y axis. This indicates that the rate code function is a reasonable approximation of the spiking rate function, at least in capturing the actual spike rate. In terms of information processing dynamics in the network itself, discrete spiking is inevitably different from the rate code model in many ways, so one should never assume that the two are identical. Nevertheless, the practical benefits of using the rate-code approximation are substantial and thus we often accept the risk to make initial progress on understanding more complex cognitive functions using this approximation.

**Noise**

An important aspect of spiking in real neurons is that the timing and intervals between spikes can be quite random, although the overall rate of firing remains predictable. This is obviously not evident with the single constant input used so far, which results in regular firing. However, if we introduce noise by adding randomly generated values to the net input, then we can see a more realistic level of variability in neural firing. Note that this additional noise plays a similar role as the convolution of noise with the XX1 function in the noisy XX1 function, but in the case of the noisy XX1 we have a deterministic function that incorporates the averaged effects of noise, while here we are actually adding in the random values themselves, making the behavior stochastic.

• Change the variance of the noise generator (`noise_var` in the control panel) from 0 to .2, and press **Apply** and then **Run**. You should see the red `net` line is now perturbed significantly with the noise.

It can be difficult to tell from a single run whether the spike timing is random -- the unit still fires with some regularity.

⇒ Do many **Runs** on top of each other in the graph view.

Even with this relatively high level of noise, the spike timing is not completely uniform -- the spikes still form clusters at relatively regularly-spaced intervals. If you increase `noise_var` all the way to .5, the spikes will be more uniformly distributed. However, note that even with the high levels of variability in the specific spike timing, the overall `act_eq` rate of spiking recorded at the end of the input does not change that much. Thus, the rate code is a highly robust reflection of the overall net input.

In the brain (or large networks of simulated spiking neurons), there are high levels of variability in the net input due to variability in the spike firing of the different inputs coming into a given neuron. As measured in the brain, the statistics of spike firing are captured well by a *Poisson* distribution, which has variability equal to the mean rate of spiking, and reflects essentially the maximum level of noise for a given rate of spiking. Neurons are noisy.
Adaptation

Cortical pyramidal neurons exhibit the property of spike rate adaptation. We are now using a more advanced form of adaptation than the form from the original AdEx model, based on sodium-gated potassium channels (K_{na}), as determined by the \texttt{kna\_adapt} parameters shown in the control panel. You can explore the basic effect of adaptation by turning this on and off.

Hit \texttt{Defaults} and then turn the \texttt{adapt.on} button off and \texttt{Run} -- compare with it turned on.

You should observe that spiking is perfectly regular throughout the entire period of activity without adaptation, whereas with adaptation the rate decreases significantly over time. One benefit of adaptation is to make the system overall more sensitive to changes in the input -- the biggest signal strength is present at the onset of a new input, and then it "habituates" to any constant input. This is also more efficient, by not continuing to communicate spikes at a high rate for a constant input signal that presumably has already been processed after some point.

You may now close the project (use the window manager close button on the project window or \texttt{File/Close Project} menu item) and then open a new one, or just quit emergent entirely by doing \texttt{Quit emergent} menu option or clicking the close button on the root window.
Article Sources and Contributors