

Problem Set 7, CNS 185 1999

Handed out: 11 Nov 99
Due: 18 Nov 99
Points: 11 points out of 100 for the term

7.1 Models of Spiking Neurons (6 points)

A mysterious fact about biological neurons is that they communicate with each other using very short-lived, all-or-nothing events, known as *action potentials*, or (colloquially) “spikes”. One approximation technique modelers use is to approximate the exact firing pattern of a neuron by a continuous variable, the mean “firing rate” of the neuron. This is accurate when the currents flowing into cells change slowly. This facilitates the development of a number of mathematically tractable models of neural computation which give us insight into how large-scale computations (such as associative memories, classification by feed-forward networks, and learning) can be performed by networks of neurons. However, such models vastly oversimplify real neurons. What is different about computation with spikes compared to computation with continuously varying analog values? Though not precisely defined, this is a deep question and we will only be able to scratch the surface here.

Previously, we looked at the Hodgkin-Huxley model of a spiking neuron. This is a biologically accurate model of a squid neuron and with minor changes, similar models can be made of other neurons (e.g. cortical cells) whose membrane properties (e.g., voltage-gated ionic currents) are known.

These types of models, however, are computationally intensive and are specific to the particular neurons they are modeling. In this problem set, we will look at two of the more abstract models for spiking neurons. At the end of this section, we will see how these two models stack up to the Hodgkin-Huxley model we used in problem set 6. As before, most the MATLAB code you will need has been provided and you only have to plug in few lines of code here and there.

7.1.1 The Poisson Model (3 points)

An important starting point for discussions of neural spike trains is a probabilistic or statistical model of spike generation. This model pre-supposes that spike trains are very noisy and represent the inputs to a neuron only in a statistical sense. That is, given a knowledge of the input to a neuron, its output spike train can be determined only probabilistically. A very common statistical description of spike trains is the *Poisson process*. Even though, most neural spike trains are far from Poisson, the underlying point process is very simple and has applicability far beyond neural spike trains.

Let us say that a neuron is firing at a constant rate λ *i.e.*, the average number of spikes to be found in a period of T is λT . Furthermore, we will imagine that the exact firing times are otherwise “random”. We make this notion precise by saying that the probability of firing at a given instant is the same as the probability of firing at any other instant and that these events are statistically

independent. The random process defined by these two assumptions gives rise to the Poisson process.

In computer simulations, one way to generate a Poisson process is to discretize a period of time T into N bins s_i , $i = 1 \dots N$, each representing a portion $\delta = T/N$ of that time (e.g. $T = 30$ sec, $N = 30000$, and $\delta = 1$ msec). We now choose an appropriate value for p_δ and independently set each $s_i = 1$ with probability p_δ , and $s_i = 0$ otherwise. The vector \vec{s} now represents the firing of the cell as a function of (discretized) time; where ever \vec{s} is 1, a spike occurred.

1. Find an expression for p_δ in terms of λ and δ such that the average firing rate in this simulation is λ spikes per unit time. Sometimes there is no such p_δ . When?

An alternative method for generating spike trains is to randomly pick the arrival time t_1 of the first spike, then randomly pick the time until the next spike arrives, t_2 , and so on, until our time is up, i.e. until $\sum_i t_i > T$ (at which point we may discard the final time). In other words, the first spike occurs as time t_1 , the second spike occurs at time $t_1 + t_2$, and the k^{th} spike occurs at time $\sum_k t_k$. To do the simulation properly, we must choose the right probability distribution on the inter-spike intervals, t_i .

2. Consider the discretized time model, parameterized by δ and p_δ (consider T to be arbitrarily large). Find an expression for the probability distribution on inter-spike intervals, $Pr(t_i = k\delta)$, as a function of k and p_δ .

Now consider T to be a fixed and finite time. We decrease δ and increase N , always keeping $T = N\delta$. In effect, in the limit we are making a transition from discretized time to continuous time. Give a formula for the *probability density*:

$$\frac{Pr(T - \frac{1}{2}\delta < t_i < T + \frac{1}{2}\delta)}{\delta}$$

and compute its limit as $N \rightarrow \infty$.

3. What is the expected value of the inter-spike interval (ISI) for both the continuous and discrete cases? What is the most probable ISI? How do these values relate to λ , if at all? Would the most probable ISI be the same for a real neuron? Explain.

Neurons that always fire at a constant rate are boring. We have previously argued (and will argue again, below) that their firing rate might depend upon their (time-varying) input $I(t)$. In this homework, we will use $\lambda(t) = \lambda_{max} \sigma(I(t)) = \lambda_{max} / (1 + e^{-I(t)})$. We can easily accommodate this variation in our original Poisson model by choosing $f(t) = 1$ (i.e. fire a spike) with probability p_δ calculated by $\lambda = \sigma(I(t))$. This is known as an inhomogeneous Poisson process.

4. Find a MATLAB expression for the spike train output \vec{s} of a Poisson neuron given an input \vec{I} . With $\lambda_{max} = 100$ Hz, create one instantiation of a random spike train based on $\vec{I} = 2$ (constant input), for $1 \text{ msec} \leq t \leq 1000 \text{ msec}$. Plot the signal \vec{I} and the response \vec{s} using MATLAB's `subplot` command.
5. Use the program `gensig.m` to create a one-second span of white-noise "stimulus waveform" $I(t)$ ($1 \text{ msec} \leq t \leq 1000 \text{ msec}$) with slowly-varying input activity (e.g., bandwidth = 10 Hz, mean = 0, standard deviation = 4). Set $\lambda_{max} = 100$ Hz and create a random spike train based on this random \vec{I} . Plot the signal \vec{I} and the response \vec{s} using `subplot`.

7.1.2 The Integrate & Fire Model (3 points)

The variable-rate Poisson process (above) does not say anything about the mechanism of spike generation; it represents a phenomenological model used to approximate the irregular nature of real spike trains. Let's now look a little more closely (but still not too closely) at how spikes could be generated within a cell.

We have seen in lecture that an interesting and versatile model for spiking in single neurons is the *integrate-and-fire* (I&F) model. In its simplest form, it assumes a neuron capacitively integrates current injected into its soma. The current might arise due to post-synaptic potentials due to spikes from other cells, or from an experimenter's electrode. Either way, we denote the net current coming into the cell by $I(t)$ and write,

$$C \frac{dV}{dt} = I(t) \quad (1)$$

$$C \frac{dV}{dt} = -\frac{V}{R} + I(t) \quad (2)$$

where the first equation is the perfect integration case (corresponding to the perfect I&F model), and the second equation adds a resistive leakage term (corresponding to the leaky I&F model). When the membrane voltage V reaches some threshold V_T , the cell fires an action potential and instantly resets V to zero. This description leads to the equivalent circuit shown below.

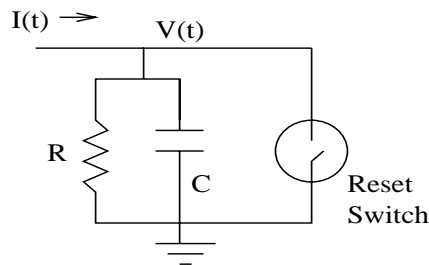


Figure 1: An integrate-and-fire circuit.

The “reset switch” symbol handles the nonlinear, discontinuous firing process. An additional modification is to have the switch clamp the voltage to 0 for a period t_{ref} after every spike; t_{ref} is known as the *refractory period*, because the cell cannot spike during that time, and any input current which arrives during this interval, is shunted to ground.

Leaky integrate-and-fire systems, even the simplistic one described above, are difficult to analyze directly. It is difficult to even write the full dynamics for V explicitly in terms of familiar functions. However, the simulations are fairly straightforward.

1. Fill in the Euler integration line in `iafsim.m`. Run `iafsim` on the same two stimulus waveforms you used for the Poisson model, and compare the results. Because the I&F model is using biologically plausible units, the input waveforms you were using for the previous section will be of the wrong order of magnitude. To make them work for the I&F model, you should multiply the waveforms by a factor of 10^{-10} (i.e. the current of 2A will now become 200pA). Try four models: to model a vanilla integrate-and-fire model (with no leak conductance), use `R = Inf`, `C = 100pF`, `Tref = 0`, `Vt = 15mV`. Add leak by setting `R = 100`. Add a refractory period by additionally setting `Tref = 5msec`. Finally, also add noise by adding an

independent 300 Hz, mean 0, standard deviation 100pA Gaussian current to the input signal (a deterministic step current). Hand in your plots.

2. Create four plots of spiking frequency vs input current, one for each model. Vary the constant input current from 0 to 500pA, or choose a reasonable range for your parameters.
3. Now look back at the $f - I$ curves you generated in problem set 6 for the Hodgkin-Huxley model. How are these curves different? (We are more concerned with qualitative properties than with the precise values which can be modified by tweaking the model parameters.)
4. When is the I&F model a good approximation to the H-H model? When is it not?

7.2 Using Neurons to Multiply (5 points)

In this section, we will explore how neurons can implement the multiplication operation by exploiting their stochastic nature. Many of the complex operations that the brain performs rely on multiplication as a fundamental computational element. While it is hypothesized that the brain must have multiple mechanisms for performing this operation, no clear example has been worked out yet. Here, we will examine two possible ways in which multiplication may be accomplished. For more details on these methods as well as other plausible scenarios, see section 21.1 in *Biophysics of Computation*.

7.2.1 Coincidence Detection (3 points)

The first mechanism relies on coincidence detection between two statistically independent spike trains by a leaky integrate-and-fire unit with time constant τ . Imagine two input streams, carrying spikes at a mean rate of f_A and f_B , that converge onto a leaky integrate-and-fire unit. We would like to compute the product $f_A f_B$. Say that the unitary EPSP (Excitatory Post Synaptic Potential) of amplitude V_0 from either input is large but does not, by itself, exceed the voltage threshold, V_T . For this, two EPSPs are required (i.e. $2V_0 < V_T$). However, we do not want two inputs from the same neuron to cause the receiving cell to reach threshold V_T . The time Δt between successive spikes from the same spike train must be large enough that the voltage increase caused by the first one will decay sufficiently that when the second EPSP happens, the two will sum to some value smaller than V_T .

1. In terms of V_0 , V_T , and perhaps other necessary constants, what is the minimal Δt which will meet this requirement?

The probability of the joint occurrence of two statistically independent events is the product of the probabilities of the individual events: $P(A \cap B) = P(A)P(B)$. Recall that in a Poisson process $p_\delta = \lambda\delta$ where p_δ is the probability of a spike in a time bin, λ is the spiking rate, and δ is the size of the time bin. So the probability of a spike is proportional to the firing frequency. Thus, the probability of concurrent spikes from two statistically independent Poisson units, A and B, (e.g., the probability of both units spiking in the same bin) is $p_{\delta A}p_{\delta B}$, which is proportional to the product of their rates, $f_A f_B$.

We will now run a simulation to see if the firing rate of a leaky output unit will be approximately proportional to the the product of the two input rates. Use an integrate-and-fire neuron with two Poisson inputs A and B. For the I&F unit, set $R=100 \text{ M}\Omega$, $C=100 \text{ pF}$, $V_T = 15 \text{ mV}$. For the inputs,

A and B, vary the firing rates f_A and f_B between 1 and 60 Hz between trials. For each spike from the Poisson trains, inject current into the I&F neuron such that the peak EPSP will be 10 mV. The post synaptic current should be an exponential time course, starting at I_s and decaying as $e^{-t/\tau}$. Actually, to make your lives easier, your caring TA's have written a function that does most of this for you: `coincidence.m`.

- Plot the firing rate of the I&F neuron versus the product of f_A and f_B for a range of frequencies between 1 Hz and 60 Hz. What is the constant of proportionality for the relationship $f_{IF} \propto f_A f_B$? (i.e. what is the value of k if $f_{IF} = k * f_A * f_B$?) If the relationship you find is not linear, explain why.
- Now set f_B to a constant value and vary f_A from 1 Hz to 100 Hz. Plot the firing rate of the I&F neuron versus f_A . Is this relationship linear? Why or why not?

Notice that this mechanism only works in the presence of sufficient jitter. If the input spikes are too regular, the response depends on the exact phase relationship between the two trains.

7.2.2 Linear Summation of Linear Threshold Neurons (2 points)

A second mechanism for multiplication by neurons is based on the linear summation of noisy linear threshold (LT) neurons. The firing rate output of a single LT neuron is zero up to a current threshold I_T and linear in the current input with slope R for larger values (see Figure 2A). Figure 2B depicts a population of such cells whose thresholds are drawn from a uniform probability distribution (the I_T value for any one cell remaining constant in time).

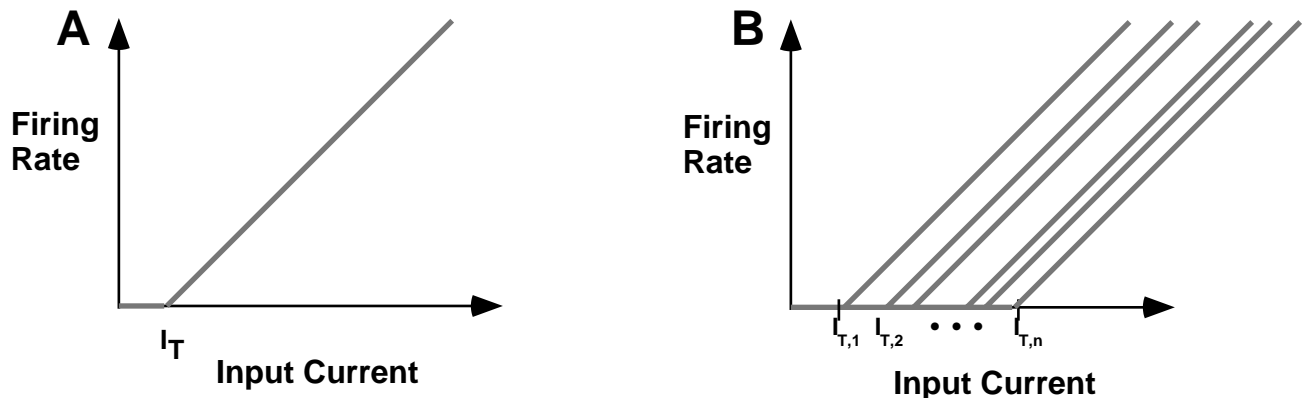


Figure 2: (A) The $f - I$ curve for a single Linear Threshold unit, and (B) for a population with a uniform distribution of I_T .

When a current input I_{Input} is applied to such a population and the output from all cells is summed, the result is a quadratic function of I_{Input} for a certain range of inputs. We can get an intuition of why this works by plotting the firing rates of the different units side-by-side, as shown in Figure 3B. Taking the sum of the firing rates is equivalent to calculating the area under the triangle depicted in the figure. As the input current increases, the triangle's area will grow quadratically. This method of taking an integral through random samples is known as a Monte Carlo sampling procedure. Keep in mind that the range of I_T 's you choose, and the number of units you use will significantly affect your results.

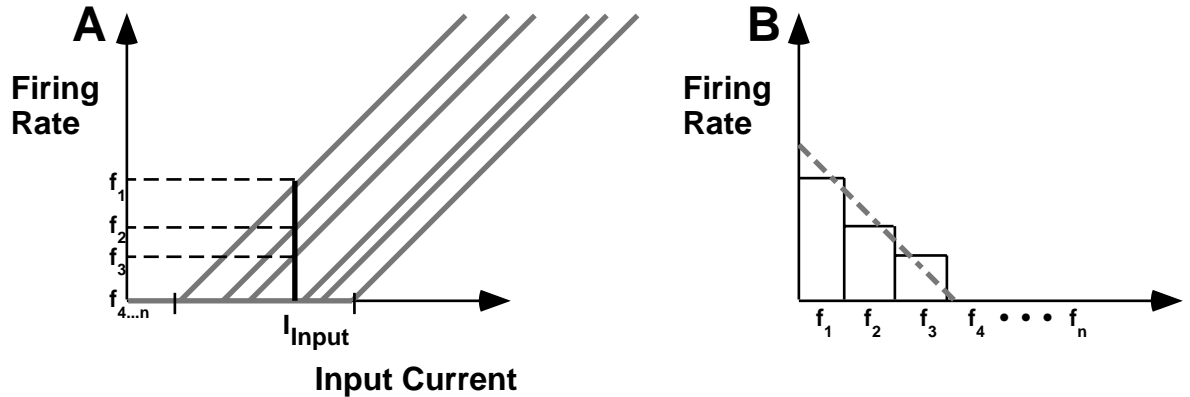


Figure 3: (A) The firing rates for the population of units, given a constant input current I_{Input} ; (B) The sum of these firing rates can be thought of as the area under the triangle specified by the origin and the dotted line, which is a quadratic function in I_{Input} .

Computing xy requires two populations of such cells with jittered thresholds, one evaluating $(x+y)^2$ and one $(x-y)^2$. Subtracting these two yields the desired result. For simplicity, we will only look at the properties of a single such population.

1. Run a simulation using 50 to 100 cells and hand in your plots. The function `LTsum.m` is provided for your convenience. Plot the sum of the population activity versus the input current. Pick a range of current thresholds, and make your input currents vary over the same range. The sum should converge to a quadratic function of the input.
2. What happens if all the I_T values are smaller than the input current? What if they are all larger? For the multiplication to work as described above, how should the range of possible I_T relate to the range of possible input currents?
3. How does the number of neurons used affect your results? If the population were suddenly halved, could some modification be made to the circuit to counter the change in the population output? What if the number of neurons doubled?
4. How does scaling R (the constant that relates the input I to the output Y of each neuron) affect the sum of the population output? You may wish to use the Matlab function `polyfit` to fit a quadratic function to your simulated data and look at the coefficients of the polynomial fits.