SCHOOL OF PHYSICS, ENGINEERING AND TECHNOLGY

Module ELE00162M: Machine Learning and Computational Intelligence

Laboratory for Neuromorphic Computing. Spiking Neural Networks: The Leaky Integrate-and-Fire (LIF) Neurone Model.

Introduction

This laboratory session supports the Neuromorphic topic within the module Machine Learning and Computational Intelligence. This laboratory provides opportunities to understand the operation of the LIF spiking neuron model, how standard numerical methods (forward and reverse Euler methods) can be used to implement the LIF model, and how the LIF model can be used to generate neuronal spike trains with different firing patterns.

Aims

The aims of this lab are to investigate:

- The basic operation of the LIF model.
- The use of the Euler numerical methods to implement a LIF neuron model.
- Using the LIF model to generate different responses, including realistic stochastic neuronal spike trains.

Getting started and using MATLAB

The lab uses MATLAB. Start MATLAB and create (or switch to) a directory where you can save your work. No additional MATLAB files are required for this laboratory, you will write your own scripts and functions to implement LIF spiking neurons.

For information regarding use of MATLAB on personal machines see
https://www.york.ac.uk/it-services/software/a-z/matlab/

For information regarding accessing MATLAB through the virtual desktop service see
https://www.york.ac.uk/it-services/services/vds/virtual-desktop/

LIF model background.

This section gives background to the Leaky Integrate-and-Fire (LIF) spiking neuron model. This neuron model forms the basis for the True North and SpiNNaker Neuromorphic computing platforms. A schematic of the LIF model is illustrated in figure 1, with current $I(t)$ creating potential difference $u(t)$ across the neuronal membrane.

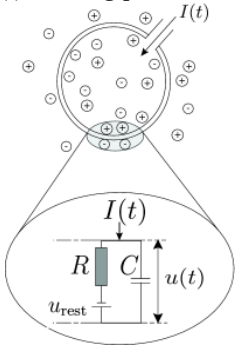

Figure 1. Schematic of the Leaky Integrate and Fire (LIF) neuron model.
From: https://neuronaldynamics.epfl.ch/online/

The LIF model represents the current across the neuronal membrane as the sum of resistive and capacitive currents. We can write an equation for this as

$$I(t) = C_m \frac{dV}{dt} + \frac{(V - E_L)}{R_m},$$

where V is the membrane potential ($u(t)$ in figure 1), C_m and R_m are the membrane capacitance and resistance, respectively and E_L is the leakage reversal potential, (u_{rest} in figure 1). The reversal potential is the potential at which the opposing forces on the ions from the concentration gradient and electrostatic repulsion cancel out, no net current flows across the resistive arm when the membrane potential is at the reversal potential.

If we multiply through by R_m and rearrange we get

$$\tau_m \frac{dV}{dt} = E_L - V + R_m I(t), \qquad (1)$$

where τ_m is the membrane time constant, $\tau_m = R_m C_m$. Equation (1) gives a differential equation in standard form to represent the neuron membrane potential, V. Over time V will relax exponentially to the value $E_L + R_m I(t)$ with time constant τ_m. Note that a time varying membrane potential, $V(t)$, is assumed in the representation in equation (1) as well as a time varying input current $I(t)$.

The LIF model incorporates a threshold. If the potential exceeds this pre-determined threshold there is a reset mechanism: At the next time step the voltage is instantaneously reset to a fixed value, usually the resting potential. Introducing discrete time indexing for the voltage, V, at time step n, as V_n, this condition is written as

$$\text{If } V_n \geq V_{th}, \quad V_{n+1} = E_L. \qquad (2)$$

If the membrane potential exceeds the threshold, the LIF neuron model is assumed to have generated an output spike and the membrane potential is instantaneously reset to the resting potential at the next time step.

Equations (1) and (2) define the LIF model. For a single neuron the three parameters R_m, C_m and E_L, have to be specified along with the threshold V_{th}.

The next section considers use of numerical methods to implement the LIF neuron as an algorithm,

LIF membrane potential numerical integration using forward and backward Euler methods.

This section considers background to the numerical methods required to implement the LIF neuron model in an algorithm. To solve the differential equation requires use of a numerical method. We will look at the forward and backward Euler methods, these are straightforward and widely used methods for numerically solving systems of differential equation.

We will consider the numerical integration of a first order differential equation, starting from the general form

$$\tau_V \frac{dV}{dt} = V_\infty - V \tag{3}$$

This describes a system which will relax exponentially towards the value V_∞ with time constant τ_V. From this an expression for the derivative is

$$\frac{dV}{dt} = \frac{V_\infty}{\tau_V} - \frac{1}{\tau_V} V \tag{4}$$

A finite difference equation can be constructed to advance the potential, *V*, from time step *n*, V_n, to time step $(n+1)$, V_{n+1}, using a time step or time increment *dt* and linear extrapolation using the derivative (slope).

$$V_{n+1} = V_n + dt \frac{dV}{dt} \tag{5}$$

The indexing on the derivative, which we need to specify, determines whether the forward or backward Euler method is used. An index of *n* gives the forward Euler method, using extrapolation in the forward direction. An index of $(n+1)$ gives the backward Euler method, using extrapolation in the backward direction. To solve both these we will write (4) in a more general form

$$\frac{dV}{dt} = A - BV \tag{6}$$

Here: $A = \frac{V_\infty}{\tau_V}$ and $B = \frac{1}{\tau_V}$.

Forward Euler. The finite difference equation for the forward Euler method is

$$V_{n+1} = V_n + dt\left(A - BV_n\right) \tag{7}$$

which yields the solution for V_{n+1} as

$$V_{n+1} = V_n\left(1 - dt\,B\right) + dt\,A \tag{8}$$

Backward Euler. The finite difference equation for the backward Euler method is

$$V_{n+1} = V_n + dt\left(A - BV_{n+1}\right) \tag{9}$$

which yields the solution for V_{n+1} as

$$V_{n+1} = \frac{V_n + dt\,A}{\left(1 + dt\,B\right)} \tag{10}$$

For the LIF model we have the following for *A* and *B*:

$$A = \frac{E_L + R_m I\left(t\right)}{\tau_m} \tag{11}$$

$$B = \frac{1}{\tau_m} \tag{12}$$

The first exercise considers how to implement the LIF neuron model using these methods.

Exercise 1 – Implementing a LIF neuron.

Many studies simplify the LIF model by using standard values for the parameters, these include a resting potential of 0V, a threshold of 1V and a resistance of 1Ω. While these are not biologically realistic they allow a more straightforward implementation and give similar results to using more biologically relevant parameters.

We will use the following parameters for the LIF neuron:

Time constant:	$\tau_m = 25\,\text{ms}.$
Resting potential:	$E_L = 0.$
Firing threshold	$V_{th} = 1.$
Input resistance:	$R = 1\,\Omega.$

The time constant is typical of that found experimentally when studying cortical neurons.

Task 1: Explore the use of the forward Euler integration scheme to calculate the neuron response for the two input conditions:

1. Input: $R_m I(t) = 0.8$
2. Input: $R_m I(t) = 1.2.$

Use a time step of $dt = 0.001\,\text{s}$, 1ms, for the Euler method up to a time limit of 250 ms.

The first input is not sufficient to exceed the threshold (Why not? – look at equation 1).
The second input will allow the membrane potential to exceed the threshold and thus generate output spikes.

The response of the membrane potential over time for these two input conditions is shown in figure 2. Generate a figure showing the same responses from your LIF model. The dots in the right panel are simply to indicate the timing of output spikes, these do not need to be included in your figure.

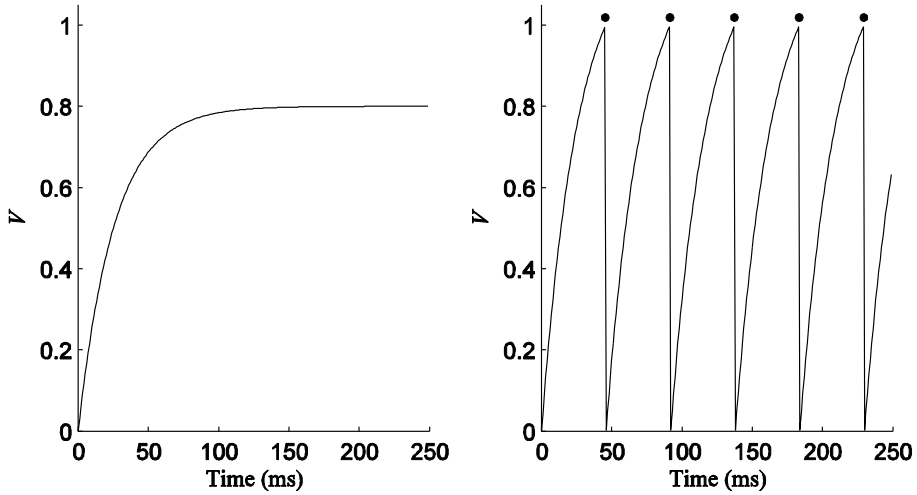

Figure 2. Response of LIF neuron model to constant inputs, (Left) $R_m I(t) = 0.8$, insufficient to generate spikes, (Right) Input of $R_m I(t) = 1.2$ generates spikes shown by dots.

If you have ideas about how to achieve this then try it now. If not some hints are below.

Hints
The solution involves integration over time where the membrane potential is updated from step *n* to step *n*+1 using equation (8) for the forward Euler method. A typical structure for this in MATLAB might look like:

```
for t_ind=2:t_steps

      % Forward Euler solution of LIF model.
      V(t_ind)=V(t_ind-1)*(1-B_fac)+A_fac;

      % Reset after each spike
      if (V(t_ind)>=v_thresh)
            V(t_ind)=v_init;
      end

end
```

This assumes you have defined the relevant variables and have a vector V to store the membrane potential values. Note that this only support a constant input through the term `A_fac` which relates to equation (11), don't forget to include *dt* in the variable `A_fac`.

Optional task: Comparison with backward Euler method.
If you are happy your script is working you could try using the backward Euler method. This should give very similar results to the forward Euler method. If you plot the voltage traces on the same graph the lines should superimpose.

What is the difference between the two methods?

The backward Euler method is usually considered as a more stable method, so that as the step size, *dt*, is increased it give more accurate results. You can see this if you increase the step size *dt* from 1ms to 2ms and higher values. Eventually the forward Euler method will break down while the backward Euler method will remain stable.

Exercise 2. Computing with spikes: Generating realistic spike trains.

Neuromorphic computing, like the human brain, relies on neuronal spike trains to convey information from neuron to neuron and communicate between brain regions. In this exercise we will consider how to generate realistic neuronal spike trains using the LIF model. Realistic in this case means that spike train characteristics are similar to those found in biological neural systems. A prominent feature of biological neuron spike trains is that they are stochastic in nature, i.e. there is considerable variability in the timing of individual spikes. In the LIF model similar behaviour is achieved using additional noise as input to the LIF neuron model from exercise 1. We will also consider how to create a statistical representation for a stochastic (or random) spike train using the statistical distribution of intervals between pairs of spikes.

Now we have a basic script from exercise 1, we can modify it to generate spike trains with more realistic characteristics. To do this requires the addition of noise to the step input. Noise is generated using the MATLAB function `randn`. This function generates samples from a normal distribution with zero mean and unit variance.

Task 1: Modify your LIF model to have the following features:

- Simulated time duration of 60 seconds.
- External current is not fixed but has a mean of $RI(t) = 0.5$ and standard deviation of $\sigma = 7$.
- Keep a record of firing times to construct a histogram of inter-spike-intervals.

If you have ideas about how to achieve this then try it now. If not some hints are below.

Hints

To achieve this you will have to modify your script so that `I_ext` is not fixed, but varies at each time step. One way you can achieve this is to incorporate an additional step into the simulation loop:

Define new variables at the start of the script:

```
I_ext_mag
I_ext_std
```

Incorporate the following into the appropriate place in the simulation loop:

```
% Generate noise sample
I_ext=I_ext_mag+I_ext_std*randn;
```

Note that the variable `A_fac`, related to equation (11) will have to be calculated each time step using `I_ext`.

This should have the effect of varying the external current applied to the neuron at each time step.

You will need to include a spike counter, say: `sp_count`. Initialise this to zero (before the `for` loop), and then add one to it each time the neurone fires.

You will also need to keep a record of firing times. This will need a vector. Create a vector to store spike times:
`sp_times=zeros(3000,1);`
This includes a generous number of storage values, the expected output rate should be around 25 spikes/sec, or 1500 spike times over 60 seconds.

Each time the neurone fires store the value of `t_ind` as an indication of the spike time. You can use your spike counter from above as an index here.

Once you have done the above modifications, run the new script.

Now you should have a vector of spike times, `sp_times`, which contains `sp_count` values, from which you wish to calculate the inter-spike-intervals.

To generate the intervals you need to create a sequence of values:

```
sp_times(2)-sp_times(1);
sp_times(3)-sp_times(2);
…
sp_times(sp_count)-sp_times(sp_count-1);
```

In other words we want to subtract: `sp_times(1:sp_count-1)`

From: `sp_times(2:sp_count)`
If we have `sp_count` individual spike times, then the number of intervals will be (`sp_count - 1`).

This should give the sequence of intervals.
Try this – does it give the expected result?

Task 2: Construct a histogram plot of the intervals. This will provide a useful visual summary of the behaviour of the spike train. The MATLAB command is: `hist`
What distribution would you expect for a random spike train? The distribution of intervals should be exponential.

Look at a section of the membrane potential – what does it look like? Does this seem reasonable?

Optional task: Other ISI distributions.
Once you have completed this task, you could try exploring the effects of altering the mean and standard deviation of the input $RI(t)$. As you increase the mean and reduce the standard deviation the firing of the LIF neuron will become more regular. You will see a shift in the shape of the ISI histogram plot, which will alter from an exponential distribution and become more like a normal distribution as the LIF neuron generates output spikes more regularly spaced in time.

End of script

Appendix 1. MATLAB plotting – a primer.

Suppose you have in MATLAB the following variables:

- `t` Single column vector of values.
- `V` A multi column matrix of values.

If V has 4 columns and you wanted to graph each column of `V` against `t` on a separate plot, one below the other, you could do it with the following MATLAB commands:

```
figure
subplot(4,1,1)
plot(t,V(:,1),'k')
subplot(4,1,2)
plot(t,V(:,2),'k')
subplot(4,1,3)
plot(t,V(:,3),'k')
subplot(4,1,4)
plot(t,V(:,4),'k')
```

This would generate four plots, but they would not have any labels, or titles. The string `'k'` specifies a black line should be used.

Note how the colon notation is used to select the appropriate rows or columns. Thus

- `V(:,1)` gives column 1.
- `V(1,:)` gives row 1.

The colon, :, means use all values in a specific dimension of the matrix. You can further refine this approach:

- `V(1:100,3)` gives rows 1 to 100 from column 3.
- `V(1:100,3:6)` gives rows 1 to 100 from column 3 to 6 in a 100 × 3 matrix.

You can label the axes using the commands:

```
xlabel('Time (sec)')
ylabel('V (mV)')
```

Replacing the text as appropriate.

A title can be inserted above each plot:

title('V_m')

This would produce the title: V_m

It shows how to use TeX compatible commands.

If you wanted to scale the axes of an individual plot (or subplot), to, say, *x* values in the range 0 to 200, and *y* values in the range -1 to +1 use the command:

```
axis([0 200 -1 1])
```

If you wanted to plot more than one line on a single graph, you can use successive `plot` commands. To ensure that the plots don't clear the results of previous plot commands, use the command:

```
hold on
```

Alternatively you can plot several columns in a matrix using a single plot command, which plots a separate line segment (in a different colour) for each column in the matrix.

If you want grid lines to help read values of the graph, use the command:

```
grid on
```

By default MATLAB includes a box round each plot. If you want to remove the box, leaving only the *x* and *y* axes, then use the command:

```
box off
```

If you have already created a plot (or subplot) in a figure, and want to add labels, set axes values, or any of the above commands then move the mouse over the plot (or subplot), click the left button (to select the current plot window) and type the command(s) you want in the MATLAB command window.