Assignment 1

CS 425/525: Brain Inspired Computing
Instructor: Konstantinos Michmizos, Fall 2020
Rutgers University

Seth Karten, Madhu Sivaraj, Alex Goodkind
Due Date: Oct 27th

Contribution

Everyone contributed towards the theoretical questions during a discussion. Seth took notes and wrote down the solutions agreed upon. Madhu was the leader on the programming. Seth helped tune the parameters for the programming models (from reading the original papers) and helped visualize the data. Seth also put the report together in LaTeX. Alex helped with the LIF part of the code.

Questions

What do you expect to happen if an IF neuron is fed a very low input current? An LIF neuron?

For an IF neuron, the neuron will fire once it has received enough total current from the inception of the neuron. If the duration of time is too short, the current will not accumulate enough to reach a high enough threshold that triggers a fire. If current is supplied for infinite time, it will be guaranteed to fire.

For an LIF neuron, it takes a very long time to fire. If the magnitude is too low, it does not fire at all because the current will leak out causing it to never accumulate enough to reach a threshold high enough to trigger a fire.

What do you expect to happen if an IF neuron is fed a larger input current? An LIF neuron?

For an IF neuron, it fires faster for a larger input current since the total voltage necessary to fire accumulates more quickly. The LIF neuron will spike quickly but slightly after the IF neuron since a LIF loses a small amount of voltage over time.

What are the limitations of an LIF neuron?

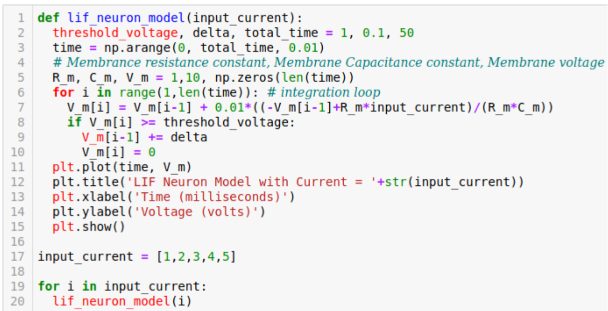
After each output spike in the LIF neuron, the membrane potential is reset. The model does not adapt and has no memory other than knowing when it last spiked (since the membrane potential is reset). It is also slow to process noise. If the LIF neuron does not receive enough signal to fire, it slowly reduces the membrane voltage until it forgets the signal. In reality, the neuron receives multiple inputs from different neurons in the network. The model only accounts for a single input current channel.

Programming

For questions 3.1, 3.2, 3.4, and 3.5, please see the figures on pages 3-5.

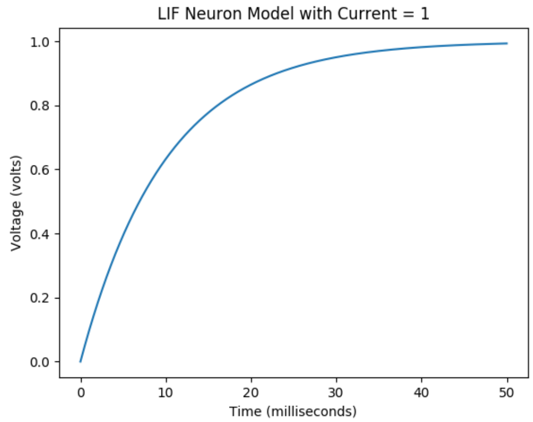
What happens to the firing rate as you continue to increase the input current? Why?

As the input current is increased, the firing rate increases. The input current is integrated into the total membrane voltage. So when the input current is higher, the total membrane voltage will reach the firing threshold faster, resulting in a faster firing rate.

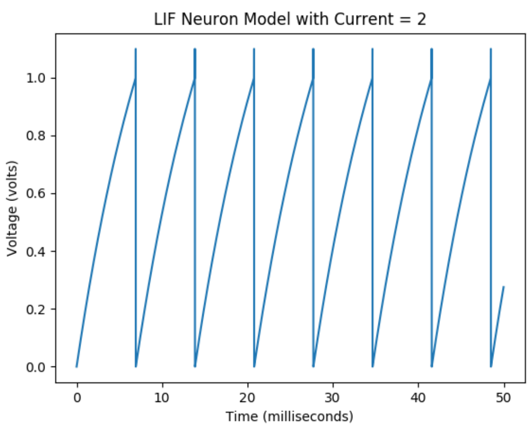


```python
def lif_neuron_model(input_current):
  threshold_voltage, delta, total_time = 1, 0.1, 50
  time = np.arange(0, total_time, 0.01)
  # Membrance resistance constant, Membrane Capacitance constant, Membrane voltage
  R_m, C_m, V_m = 1,10, np.zeros(len(time))
  for i in range(1,len(time)): # integration loop
    V_m[i] = V_m[i-1] + 0.01*((-V_m[i-1]+R_m*input_current)/(R_m*C_m))
    if V_m[i] >= threshold_voltage:
      V_m[i-1] += delta
      V_m[i] = 0
  plt.plot(time, V_m)
  plt.title('LIF Neuron Model with Current = '+str(input_current))
  plt.xlabel('Time (milliseconds)')
  plt.ylabel('Voltage (volts)')
  plt.show()

input_current = [1,2,3,4,5]

for i in input_current:
  lif_neuron_model(i)
```

(a) Code.



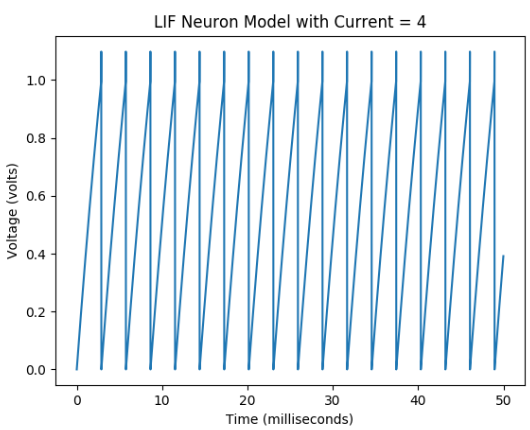
(b) Current=1mA.



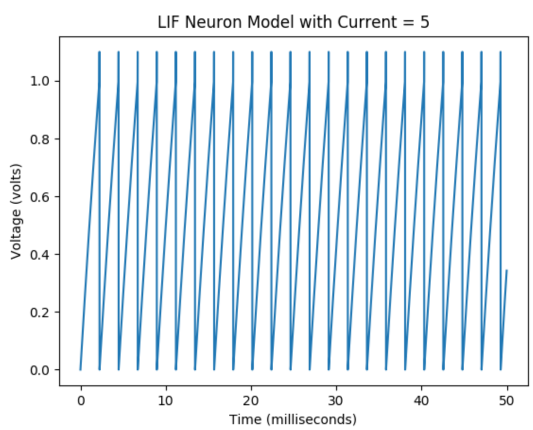
(c) Current=2mA.

[width=8cm]LIF3.png

(d) Current=3mA.



(e) Current=4mA.

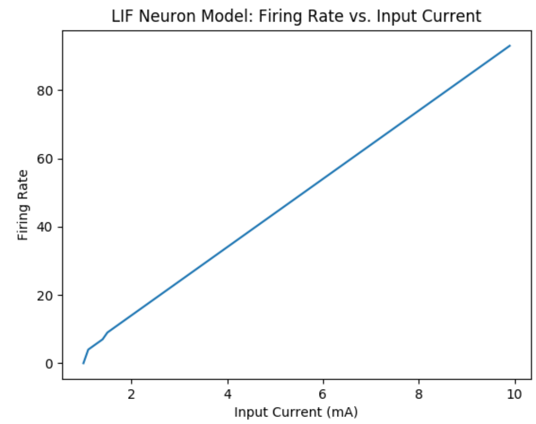


(f) Current=5mA.

Figure 1: Simulate an LIF neuron with different input currents and plot the membrane potential, showing (a) potential decay over time and (b) spiking behavior.

```python
def firing_rate_model(input_current):
  total_time, threshold_voltage, delta = 100, 1, 0.1
  time = np.arange(0, total_time, 0.01)
  # Membrance resistance constant, Membrane Capacitance constant, Membrane voltage
  R_m, C_m, V_m = 1,10, np.zeros(len(time))
  spikes = []
  for current in input_current:
    count = 0
    for i in range(1,len(time)): # integration loop
      V_m[i] = V_m[i-1] + 0.01 * ((-V_m[i-1]+R_m*current)/(R_m*C_m))
      if V_m[i] >= threshold_voltage:
        V_m[i-1] += delta
        V_m[i] = 0
        count += 1
    spikes.append(count)
  plt.plot(input_current, spikes)
  plt.title('LIF Neuron Model: Firing Rate vs. Input Current')
  plt.xlabel('Input Current (mA)')
  plt.ylabel('Firing Rate')
  plt.show()
input_current = np.arange(1,10,0.1)
firing_rate_model(input_current)
```

(a) Code.

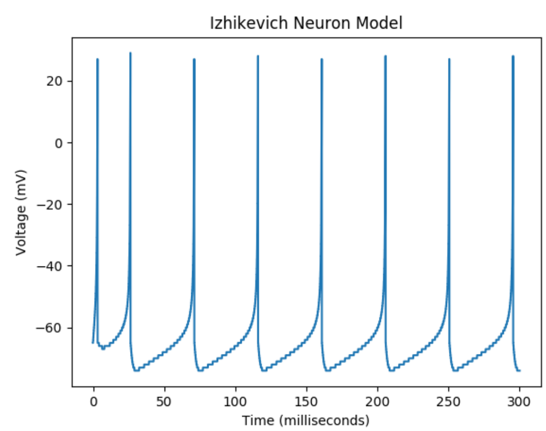


(b) Plot.

Figure 2: Plot the firing rate as a function of the input current.

```python
def izhikevich_model(input_current):
  threshold_voltage = 30
  # parameters from Izhikevich paper for regular spiking (RS) neurons
  # recovery, firing threshold, deep voltage reset, large spike reset
  a,b,c,d = 0.02, 0.2,-65,8
  total_time = 300
  time = np.arange(0,total_time, 0.01)
  V_m=np.full(len(time), c)    # Membrane voltage - start at reset voltage
  u = b*c   # u represents a membrane recovery variable
  v = c   # v represents the membrane potential of the neur
  for i in range(1,len(time)): # integration loop
    V_m[i] = v
    dv = 0.04*(v**2) + 5*v + 140 - u + input_current
    du = a*(b*v - u)
    v += dv*0.01
    u += du*0.01
    if v >= threshold_voltage:
      v = c
      u = u + d
  plt.plot(time, V_m)
  plt.title('Izhikevich Neuron Model')
  plt.xlabel('Time (milliseconds)')
  plt.ylabel('Voltage (mV)')
  plt.show()
input_current = 10
izhikevich_model(input_current)
```

(a) Code.

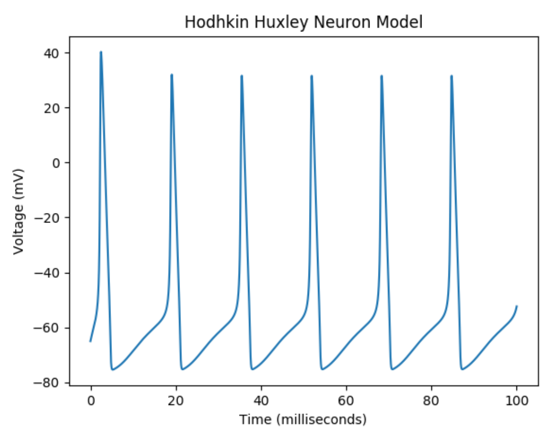


(b) Plot.

Figure 3: Simulate a neuron using the Izhikevich model.

```python
def hodgkin_huxley_model(input_current):
  total_time = 100
  time = np.arange(0,total_time, 0.01)
  V,n,m,h = np.zeros(len(time)),np.zeros(len(time)),\
            np.zeros(len(time)),np.zeros(len(time))
  V[0],n[0],m[0],h[0] = -65,0,0,1 # initialize voltage, n m h gating variables
  # params from Hodgkin-Huxley paper results and Dayan and Abbott
  normalize = 65  # normalization constant used across channel gating kinetics
  V_na = 115      # Na reversal potential
  V_k = -12       # K reversal potential
  V_l = 10.6      # Leak reversal potential
  G_na = 120      # Na max conductance (mS/cm^2)
  G_k = 36        # K max conductance (mS/cm^2)
  G_l = 0.3       # Leak max conductance (mS/cm^2)
  C_m = 1         # membrane capacitance (uF/cm^2)
  E_na = 50.0     # Na Nernst reversal potentials (mV)
  E_k = -77.0     # K Nernst reversal potentials (mV)
  E_l = -54.387 # Leak Nernst reversal potentials (mV)
  def alpha_n(V): # performs channel gating kinetics on input voltage
    return 0.01*(V+55.0)/(1.0-np.exp(-(V+55.0)/10.0))
  def beta_n(V): # performs channel gating kinetics on input voltage
    return 0.125*np.exp(-(V+normalize)/80.0)
  def alpha_m(V): # performs channel gating kinetics on input voltage
    return 0.1*(V+40.0)/(1.0-np.exp(-(V+40.0)/10.0))
  def beta_m(V): # performs channel gating kinetics on input voltage
    return 4.0*np.exp(-(V+normalize)/18.0)
  def alpha_h(V):   # performs channel gating kinetics on input voltage
    return 0.07*np.exp(-(V+normalize)/20.0)
  def beta_h(V):   # performs channel gating kinetics on input voltage
    return 1.0/(1.0+np.exp(-(V+35.0)/10.0))
  def I_na(V,m,h):   # Na membrane current function (uA/cm^2)
    return G_na*m**3*h*(V_na-V-normalize)
  def I_k(V, n): # K membrane current function (uA/cm^2)
    return G_k*n**4*(V_k-V-normalize)
  def I_l(V):   # Leak membrane current function (uA/cm^2)
    return G_l*(V_l-V-normalize)
  n[0]=alpha_n(V[0])/(alpha_n(V[0])+beta_n(V[0]))
  m[0]=alpha_m(V[0])/(alpha_m(V[0])+beta_m(V[0]))
  h[0]=alpha_h(V[0])/(alpha_h(V[0])+beta_h(V[0]))
  for i in range(1,len(time)): # integration loop
    V[i] = V[i-1] + 0.01 * ((input_current + I_na(V[i-1],m[i-1],h[i-1]) +\
                            I_k(V[i-1],n[i-1]) + I_l(V[i-1])) / C_m)
    n[i] = n[i-1] + 0.01 * (alpha_n(V[i])*(1.0-n[i-1])-beta_n(V[i])*n[i-1])
    m[i] = m[i-1] + 0.01 * (alpha_m(V[i])*(1.0-m[i-1])-beta_m(V[i])*m[i-1])
    h[i] = h[i-1] + 0.01 * (alpha_h(V[i])*(1.0-h[i-1])-beta_h(V[i])*h[i-1])
  plt.plot(time, V)
  plt.title('Hodhkin Huxley Neuron Model')
  plt.xlabel('Time (milliseconds)')
  plt.ylabel('Voltage (mV)')
  plt.show()
input_current = [7.5]
for i in input_current:
  hodgkin_huxley_model(i)
```

(a) Code.



(b) Plot.

Figure 4: Simulate a neuron using the Hodgkin-Huxley model.: We matched our parameters and equations to the parameters and gating functions found by Hodgkin and Huxley to match their experimental data.