2018 ■ Vol. 14 ■ no. 1

Spike Neural Models Part II: Abstract Neural Models

Melissa G. Johnson[a], ✉ and Sylvain Chartier[a]

[a]University of Ottawa

Abstract ■ Neurons are complex cells that require a lot of time and resources to model completely. In spiking neural networks (SNN) though, not all that complexity is required. Therefore simple, abstract models are often used. These models save time, use less computer resources, and are easier to understand. This tutorial presents two such models: Izhikevich's model, which is biologically realistic in the resulting spike trains but not in the parameters, and the Leaky Integrate and Fire (LIF) model which is not biologically realistic but does quickly and easily integrate input to produce spikes. Izhikevich's model is based on Hodgkin-Huxley's model but simplified such that it uses only two differentiation equations and four parameters to produce various realistic spike patterns. LIF is based on a standard electrical circuit and contains one equation. Either of these two models, or any of the many other models in literature can be used in a SNN. Choosing a neural model is an important task that depends on the goal of the research and the resources available. Once a model is chosen, network decisions such as connectivity, delay, and sparseness, need to be made. Understanding neural models and how they are incorporated into the network is the first step in creating a SNN.

Keywords ■ Neural models, spiking neural networks, leaky integrate and fire, Izhikevich's model.

Acting Editor ■ Denis Cousineau (Université d'Ottawa)

✉ mjohn140@uottawa.ca

MGJ: 0000-0000-0000-0000; *SC*: 0000-0000-000-0000

10.20982/tqmp.14.1.p001

Introduction

This paper extends the first tutorial on neural models (Johnson & Chartier, 2017) by describing models from the remaining two categories of spiking neural network nodes: 1) general representation neural model, and 2) generic threshold-fire neural model. Because both these categories are abstractions of a biologically-based model, one might ask why we would want abstract models: shouldn't models be as biologically accurate as possible? This is a legitimate question, especially considering projects like the Human Brain Project (http://www.humanbrainproject.eu/) which is trying to simulate the entire human brain in a super computer.

Pragmatic reasons for abstraction include: limited time, money, and computational resources, and lack of knowledge (Levy & Bechtel, 2013, 2; Kaplan, 2011). Time, money, and computational resources are all interrelated and need to be taken into consideration during the design phase. System type affects speed of simulations, but money affects the system type one can afford. Biological detail adds complexity which makes simulations take longer to run, regardless of the computer system. Therefore considerations of the budget, current equipment, and the time allotted for the simulation may make it necessary to sacrifice biological realism for abstraction. As for lack of knowledge, there are functions performed by neurons for which the mechanisms are not know; yet these functions are still modelled. For example, Hodgkin and Huxley did not know about ion channels when developing their model. The workings of these channels were abstracted into the differential equations used in the model. While ion channels are known now; other information, such as how glial cells influence learning and cognition is not (Fields et al., 2014); but their influence is still part of the function of the neuron model. While these pragmatic reasons are widely recognized reasons for abstractions, there are other reasons that make abstraction beneficial, and necessary, in modelling.

SNNs can look at specific biological and cognitive questions such as how signals are transmitted or how learning is possible, without incorporating every biological detail of neurons into the network. In fact, trying to create a completely biologically realistic SNN may be impossible because the complexity of the model would be too overwhelming to be understood and impossible to analysis (Chirimuuta, 2014). When biological details are abstracted, the resulting models can also be generalized beyond just the one specific situation. The Hodgkin-Huxley model is fitted precisely to the giant squid neuron, but while that neuron has one specific firing pattern, different neurons have different patterns. Abstracting some of the precise fitting allows the Hodgkin-Huxley model to display these different neuron firing patterns (Guckenheimer & Labouriau, 1993) such as the distinct class of pyramidal neurons that have a "chattering" pattern (Gray & McCormick, 1996).

Neural models have been generalized into type I and type II models. Type I models are integrators; their output frequency is directly related to the input frequency. For these models, the higher the frequency of input, the higher the frequency of output. Type II models are resonators; they only fire once a certain frequency of input is reached. Both these categories can be generalized into specific properties and mathematical equations, and are one of the best means of understanding the brain (Chirimuuta, 2014). This tutorial focuses specifically on type I models but for more information on type I and type II models see St-Hilaire and Longtin (2004) or Gerstner, Kistler, Naud, and Paninski (2014).

No matter the model used for simulating neurons, a good understanding of the model, its uses and shortcomings, is necessary. In the first tutorial, the biologically based Hodgkin-Huxley (HH) model was discussed. In this tutorial, two other models are presented: Izhikevich's model and the Leaky Integrate and Fire (LIF) model. Izhikevich's model is part of the general representation category so while it does model biological behaviours, the parameters have no biological basis. The LIF model is part of the generic threshold-fire category so it is not biologically realistic in any way except that it can integrate input and fire at some threshold. These two neural model sections are followed with a brief introduction to SNN and how the neural models fit into the network. All Matlab code used in this tutorial and the integration of the LIF model are presented in the appendices.

Izhikevich's Model

There is a lot of information in explicit models which may not be necessary for a given simulation. If accurate action potentials are necessary but how they are derived is not, a general representation model is more efficient and less complicated. General representation models preserve the functionality but lose the biological realism. One such model is Izhikevich's model (Izhikevich, 2003).

Izhikevich's model is based on the bifurcation and normal form reduction of the Hodgkin-Huxley model.[1] The computation used to derive Izhikevich's model is beyond the scope of this paper, but for more information on bifurcation of the Hodgkins-Huxley model, see Guckenheimer and Labouriau (1993). A defining characteristic of Izhikevich's model is its ability to show the different firing dynamics, such as chattering (described shortly), while still being a simple two-dimensional model. Izhikevich's model contains two ordinary differential equations (Eq. 1 and 2) and an auxiliary equation (Eq. 3).

$$\frac{dv}{dt} = 0.04v^2 + 5v + 140 - u + I \tag{1}$$

$$\frac{du}{dt} = a\,(bv - u) \tag{2}$$

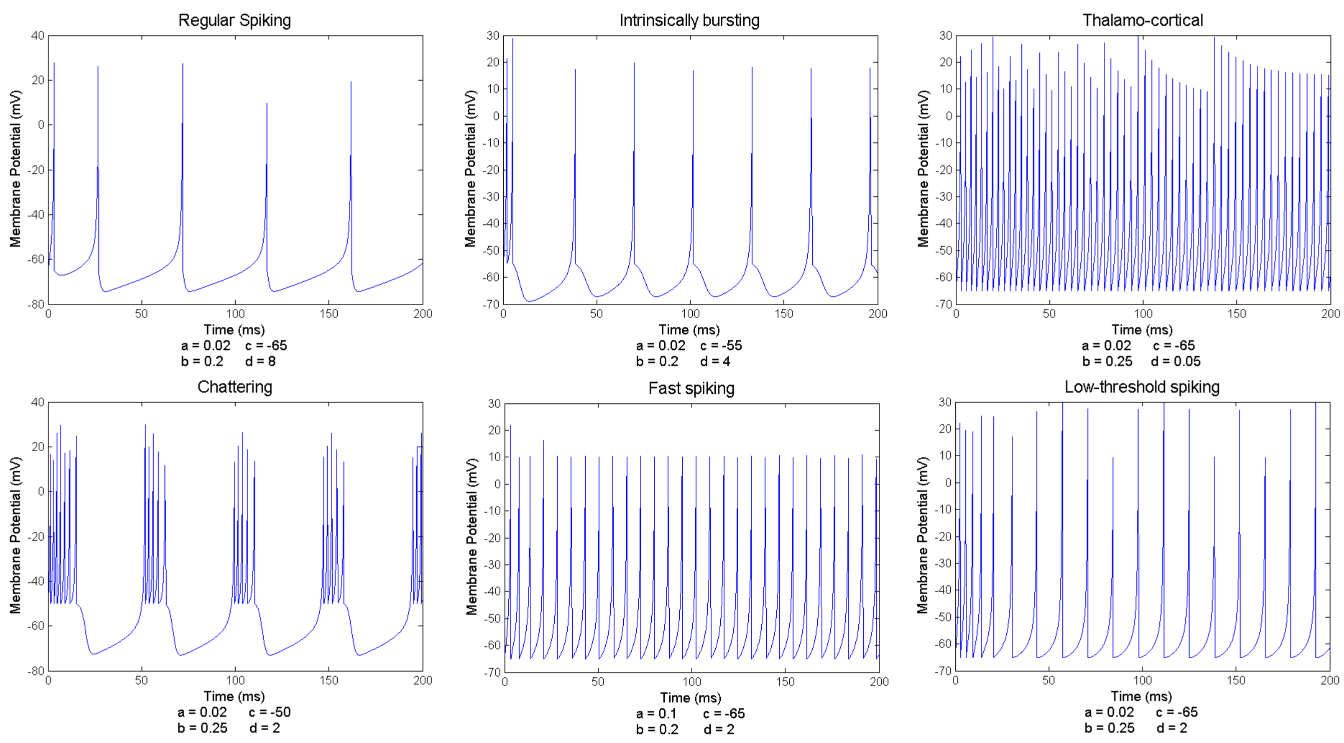
$$\text{if } v \geq 30\ mV, \text{ then } \begin{cases} v \leftarrow c \\ u \leftarrow u + d \end{cases} \tag{3}$$

In these equations, v represents the membrane potential and u represents membrane recovery. These two variables are intertwined so that v is dependent on u to produce a spike train but the value of u is dependent on the value of v to determine how fast the membrane recovers. The values of v and u are influenced by the parameters a, b, c, and d which describe respectively: the time scale of the recovery variable, the sensitivity of the recovery to subthreshold fluctuations of the membrane potential, the after-spike reset value of the membrane potential, and the after-spike reset of the recovery variable. The constants in equations 1, 2, and 3 (e.g. 0.04, 5, and 140 in Eq. 1) were obtained by fitting the equations to the spike initiation dynamics of the cortical neuron (Izhikevich, 2003).

The different firing patterns (see Figure 1) are produce by the interplay between the parameters a, b, c, and d. Regular spiking is the most common firing pattern and is defined as spiking at regular but increasingly distant intervals with constant input; this is the firing pattern displayed by the Hodgkin-Huxley model. Intrinsically bursting and chattering neurons are other types of excitatory neuron firing patterns while fast spiking and low-threshold spiking are patterns that cortical inhibitory interneurons pro-

[1] Bifurcation is the study of how the flow, or behaviour, changes in dynamic systems as parameters change. For example, how u in Equation 1 affects how the voltage produces spikes. There are whole books on this topic such as Kuznetsov (2013).

Figure 1 ■ Different types of spiking patterns based on varying the parameters a, b, c, and d. Input is set to 10 mV in all cases.



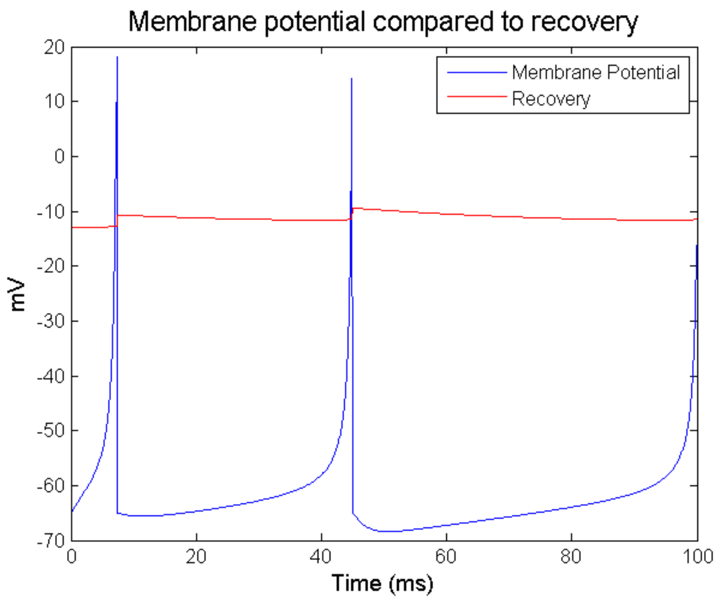
duce. The model can also duplicate the firing patterns of the thalamo-cortical neuron.

To understand how the model is able to produce the different firing patterns, one needs to understand the role of the v and u and how the parameters a, b, c, and d affect v and u. The membrane potential and the parameter c are the easiest to understand. The membrane potential is simply the voltage across the membrane; when this value reaches a pre-determined threshold, a spike is said to occur. The membrane potential is graphed in spike trains, as seen in Figure 1. The parameter c is the reset value, or the value the membrane potential returns to after a spike occurs. In the Hodgkin-Huxley model an explicit reset value was not needed because resetting the membrane potential after an action potential happens via ionic currents. In Izhikevich's model there is no implicit reset mechanism. The auxiliary equation (Eq. 3) explicitly resets the voltage after a spike occurs so that the membrane potential does not continue to increase. For example, in Figure 1, immediately after a spike, Chattering resets to -50 mV while Fast Spiking resets to -65 mV, which are the values of c for those two spike patterns.

When the membrane potential increases, the recovery parameter decreases (see Figure 2). The recovery parameter is important to make sure that there is a refractory period after a spike and to affect timing between action potentials. Parameter d plays a role in changing the possible timing between spikes. After a spike, when the neuron needs that recovery period, u is increased again via Equation 3. This increase in u is done by adding parameter d to u. The smaller the parameter d, the shorter the recovery time. Note that d should never be negative or it means that recovery reduces constantly, eventually making it automatically increase v and the neuron will always spike. This will be discussed more in the “Parameters” subsection of “Working with a Model”.

Parameters a and b both affect how the recovery parameter changes. The parameter a is a multiplicative value for whatever the change is, allowing the recovery parameter to change faster or slower depending on its value. How the recovery changes is determined by the interaction between u, v, and b. If $bv > u$, then the change of u will be positive and when $bv < u$, the change is negative. Assuming v and u are negative, u actually provides help in spiking – the larger the negative, the more it will increase v to potential spiking criteria. Therefore, a large b means a large

Figure 2 ■ Changes of membrane potential (v) and membrane recovery (u). Parameters are: $a = 0.02$, $b = 0.2$, $c = -65$, and $d = 2$ with a constant input of 5 mV over 100 ms.



negative change to the recovery, recovery is quicker and spikes occur closer together (see Figure 3). Note though, that this dynamic can change if voltage is not negative (see the subsection “Parameters” in the section “Working with a Model”). Overall, the larger the parameters a and b are, the quicker the model will be able to produce spikes again after firing.

Izhikevich’s model is a simple model capable of complex spike patterns. This makes it an excellent choice when the spike dynamics are important. It is also fairly easy to understand and fast to run in a simulation, but there is still complexity in parameter selection and understanding how the parameters work together.

Leaky Integrate and Fire

The simplest models are those from the third category of neural models: generic threshold models. Generic models don’t mimic biological neurons but display basic voltage change of the membrane potential as it builds to action potentials or reduces back to resting value. The LIF model belongs to this category and is one of the most used models in the literature (Mihalas & Niebur, 2009). The popularity of the LIF rests on the fact that the model is very simple and quick to use yet still shows membrane potential changes dependent on input and time. Because of its simplicity, LIF is also very easy to modify to get the exact functionality needed for the task (Gerstner & Naud, 2009).

Despite its use, the LIF has many issues. For one, it is not actually a spiking model because it never spikes. In the model, a spike is assumed to occur once voltage reaches a certain value, the spike threshold. Some simulations explicitly display the spike via modifications to the code; these spikes are manually built in for visual display purposes only and are not the product of the equation. The LIF Matlab code in Appendix B provides the option for displaying the spike or not depending on one’s preference (see Figure 4 for the different display options).

Another important neuron function that is missing in LIF is adaptation – the increasing interval between spikes seen in the regular spiking pattern. In the LIF model, if the input current is constant then spiking will occur at a constant interval (see Figure 4).

The LIF model is based on the basic electric circuit which has a linear resistor (I_R) and a capacitor (I_C). This means that the input current is split into two components:

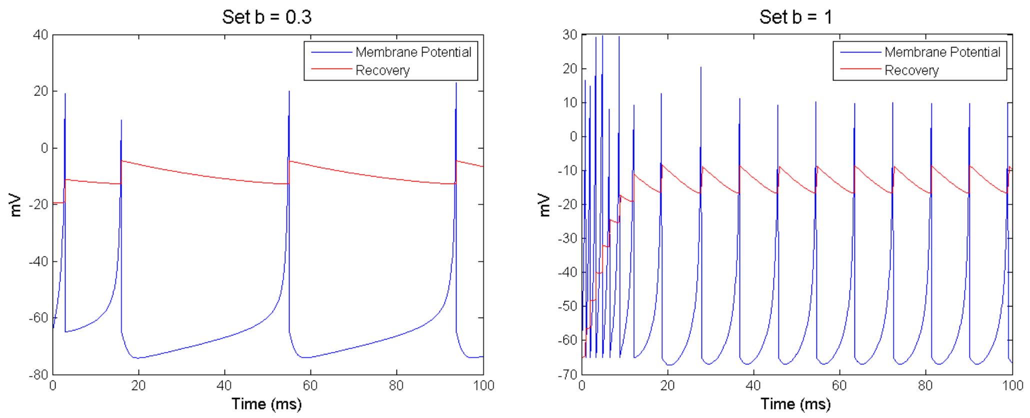
$$I\left(t\right) = \ I_R\left(t\right) \ + I_C\left(t\right) \tag{4}$$

where t is time. Based on Ohm’s law,

$$I_R\left(t\right) = \frac{v\left(t\right)}{R} \tag{5}$$

where $v(t)$ is the voltage as a function of time as the voltage travels across the resistor and R is the resistance from the resistor.

Figure 3 ■ Changes to recovery and membrane potential for different values of b. The other parameters are $a = -0.02$, $c = -65$, $d = 8$; the input current is 5 mV.



I_c can be rewritten based on the definition of a capacity

$$I_C(t) = C\frac{dv(t)}{dt} \tag{6}$$

where $\frac{dv(t)}{dt}$ is the change in voltage with regard to time. In terms of LIF, $v(t)$ is the membrane potential, therefore $\frac{dv(t)}{dt}$ is the change in membrane potential over time. C is a multiplicative value of the change to represent the capacitance, or the effect of the capacitor.

Using Equations 5 and 6, Equation 4 can be written as:

$$RC\frac{dv(t)}{dt} = RI(t) - v(t) \tag{7}$$

Both R and C are constants and together, RC, is the membrane time constant, τup.

$$\tau\frac{dv(t)}{dt} = RI(t) - v(t) \tag{8}$$

In the previous neural models, the differential equations were too difficult to integrate so the Matlab code in the appendices uses Euler's method (see Johnson & Chartier, 2017, for information on Euler's method), but the LIF is a single ordinary derivative equation (ODE) and therefore can be integrated to yield a precise result (see Appendix C). The integration of Equation 8 yields

$$v(t) = v_{rest}e^{-\frac{t-t_f}{\tau}} + \frac{R}{\tau}\int_0^{t-t_f} e^{-\frac{s}{\tau}} I(t-s)\,ds \tag{9}$$

where t is the current time and t_f is the time the model last reached threshold (or "fired"). There is an auxiliary equation to force the reset of the membrane potential after firing:

$$\text{if } v \geq \vartheta, \text{ then } v = v_{reset} \tag{10}$$

where v_{reset} is a reset value. The reset value does not have to be the same as the resting value, v_{rest}. For example, to show hyperpolarization after spike this reset value is lower than the resting potential.

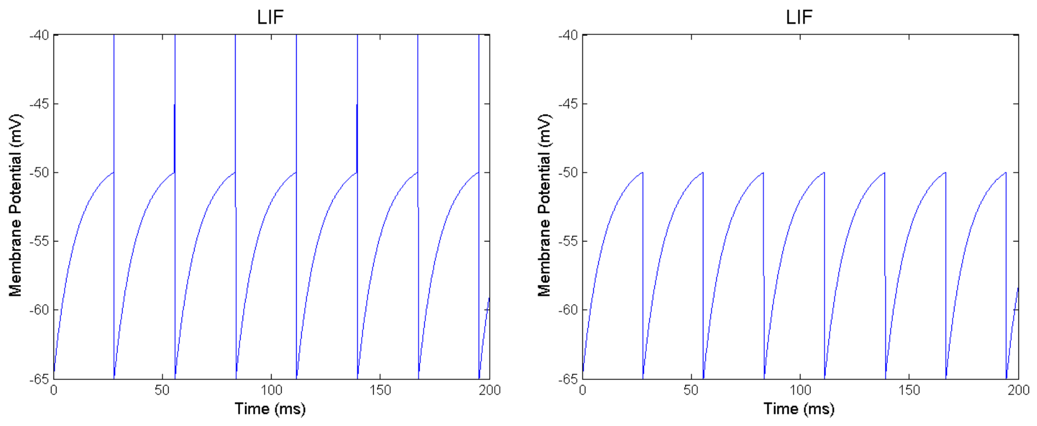
In many cases, including the code used in this tutorial, the input, $I(t)$, is not a continuous function but discrete time constants representing either "spiking" or "not spiking". This difference allows for further integration of the equation via piecewise integration (see Appendix C). Using piecewise integration, the final equation actually requires the previous membrane potential and the time difference since then to calculate the new membrane difference. The final integration for a discrete time constant input is:

$$v(t) = v_{rest} + RI(t) + (v_{t-1} - (v_{rest} + RI(t)))\,e^{-\frac{dt}{\tau}} \tag{11}$$

Notice the inclusion of dt (the time step); this represents the time difference between the last time the voltage was checked and the current time, and replaces the difference between current time and last firing ($t - t_f$). The final parameters in the LIF equation are the resting value, resistance and membrane time constant; which are normally set to 1, input; which is a function of time, and the time step used in the calculations. The auxiliary equation may also use the resting value instead of a reset value.

To overcome some of the missing features of neurons, there have been many modifications to the simple LIF. For example, there is the adaptive LIF which adjusts the frequency of firing so that spikes don't occur at equal intervals but slowly increasing intervals (Liu & Wang, 2001).

Figure 4 ■ LIF with and without spikes; there is no adaptation to the voltage, therefore spikes are equal distance apart. Time set to 200 ms, input current 1.6 mV, $R = 10$, $\tau = 10$, resting potential=reset=-65, threshold=-50



There are also many different integrate and fire models, such as the exponential model and the quadratic model; these models use the same premise of integrating input until a threshold to fire (see Chapter 5 in Vogels, Rajan, & Abbott, 2005). These models can both spike and produce more realistic timing. There is a lot of room for modifications, adaptations, and fitting in this category because of its simplicity.

Working with a Model

Three distinct models have been presented in two tutorials: Hodgkin-Huxley in tutorial I, and now Izhikevich's and LIF. When designing a simulation, the researcher needs to choose a model that is appropriate for the research but without unnecessary complexity that would limit understandability, replicability, and resource availability. Models must be carefully chosen at the onset of the network development. Models also have parameters that need to be optimized for the simulation; selecting parameters that are too big, too small, or incorrect, can lead to issues with your simulation. Finally, all the models discussed use differential equations, therefore methods to solve these equations need to be considered before coding the simulation.

Choosing the Correct Model

The three models discussed in these two tutorials are just meant to give a general understanding of how models work; there are many different neuron models in the literature. Choosing a model that is a good fit for the research requires understanding what the research is trying to show, understanding computational and time limits, and understanding the neural model.

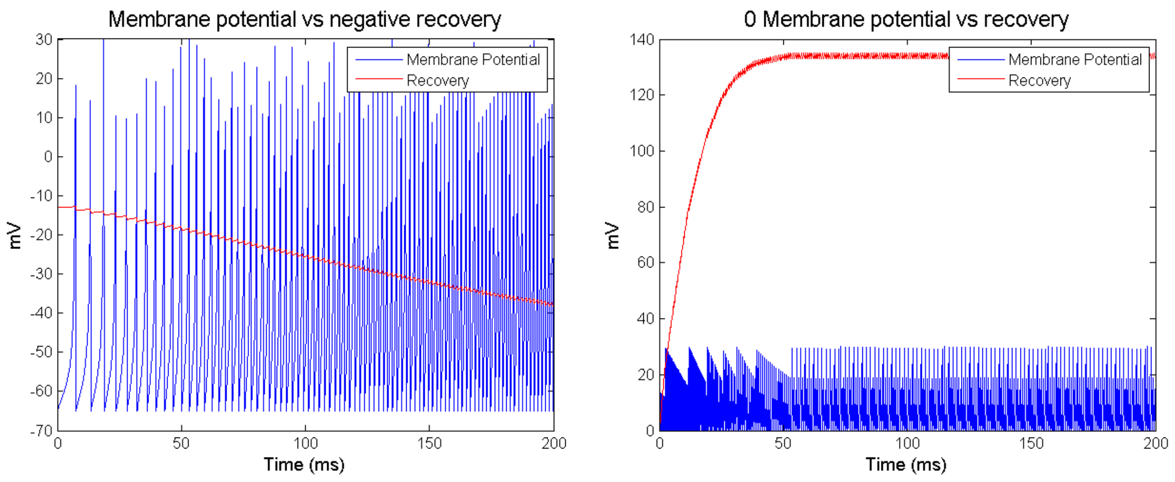
The first thing to consider is what the research is trying to determine. For example, if the point of the model is to see how neurochemicals affect size of the spike, a more biological model is necessary. Modelling spike time dependent plasticity (STDP) just needs action potentials so a generic threshold and fire model will suffice. For more information on choosing a models see (Izhikevich, 2004).

Computational resources and run time are also considerations that need to be addressed. The more complex the neuron, the more equations used, the more computational resources are needed and therefore the longer it will take to run the simulation. For example, as mentioned in Tutorial I, on my basic laptop, running the code for Hodgkin-Huxley, simulating 200 ms takes approximately 0.27 seconds (longer than the time being simulated), Izhikevich's model takes approximately 0.004 seconds, and LIF takes approximately 0.0007 seconds. While that might appear to be quite fast, that is a single neuron for 200 ms; simulations can have hundreds of neurons and run for much longer periods of time. The time needed to run the whole simulation can be a factor. Keeping the simulation simple is not only easier to program, but also means getting results quicker.

Parameters

Quick results are important, but accurate and useful results are more important. The parameters selected for use in the equations can have a huge effect on the final results. For example, in Izhikevich's model, the values of the parameters a, b, c, and d produce drastically different spike patterns, and while Figure 1 shows some accurate results, not all possible results are useful. For example, what happens if the parameter d is set as negative or the reset is set to 0? As seen in Figure 5, both these cases produce the unwanted result of perpetual spiking. This result is not useful

Figure 5 ■ Incorrectly selecting parameters in Izhikevich's model. Parameters are: $a = 0.02$, $b = 0.2$, $c = -65$ (left) or $c = 0$ (right), and $d = -1$ (left) or $d = 2$ (right). Left has constant input of 5mV of input while right has no input.



and would make the simulation meaningless. Understanding what the model's parameters mean and do will reduce the risk of producing inaccurate results.

Another important parameter that is often used in modelling but has not been explicitly discussed here is the time step, dt. Because of the difficulty in integrating Hodgkin-Huxley and Izhikevich's models, some form of integration estimation is required, which comes with its own parameter decisions. Frequently, Euler's method is used (see Johnson & Chartier, 2017), which, while having its limits is often sufficient compared to more precise, and complex, methods. In Euler's method, the time step, dt, determines how accurate the solution is. If the value for dt is too large, information is lost, but if the value of dt is too small the simulation takes a long time to run. In Figure **??**, the dt parameter is modified to show how it affects Izhikevich's model. The dt parameter affects the Δv and Δu variables such that the larger the change in time, the more information is lost in the parameter changes. This in turn affects the voltage and the recovery variables, how they change, and how the neuron model functions.

Even in simple models parameter selection is important to avoid unintended consequences culminating into bad results. For example, in LIF, if the spike threshold, ϑ, is too large, the model will never spike (see Figure 7). These are just some of the problems that incorrect parameters can produce. All parameters need to be tested carefully before running the full simulation.

Spiking Neural Networks

Most research considers how neurons work together. To get results from groups of neurons, they need to be able to interact, which is where the network part of the neural network is important. While the network part of SNNs is beyond the scope of this paper, there are a few things to consider.

Neurons are connected to each other, but not necessarily every neuron to every neuron. In some cases, neurons may have low connectivity, or sparse connectivity, where the neuron is only connected to a few other neurons. In other cases, neurons may have high connectivity, or be densely connected, such that they are connected to many other neurons. A network can include both the dense and sparse conditions at the same time. When a network has both conditions, the neurons with high connectivity are often considered "hubs" and are important for traversing the network in an efficient manner.

As with the neuron models, these connections are not physical but values and equations. For example, in Table 1, when the 6^{th} presynaptic neuron in the network (N_{61}) spikes, it affects the postsynaptic neuron N_{32} but no other neurons. Conversely, presynaptic neuron N_{31} affects postsynaptic neurons N_{22}, N_{42}, and N_{62}.

Table 1 shows all excitatory connections, but in the brain there are both inhibitory and excitatory neurons: is this going to be reflected in the network? Inhibitory and excitatory differences could be shown by using positive and negative numbers in the connection matrix. Table 1 also contains only zeros and ones, but not all connections are the same strength. N_{11} might have a larger

Figure 6 ■ Regular spiking pattern for Izhikevich's model with different time step selections: 0.1 (top left), 1.1 (top right), and 2.1 (bottom). Graphs show how voltage and recovery (top) and the change of their respective differential equation used in calculating voltage and recovery (bottom) depending on the time step used.

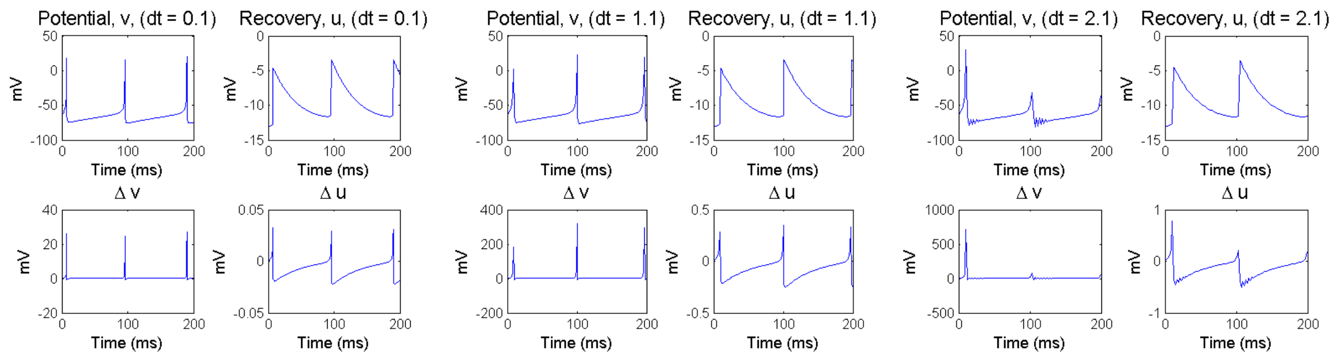


Table 1 ■ Example table of possible connections between neurons. Presynpatic neurons fire, and their action potential affects postsynpatic neurons.

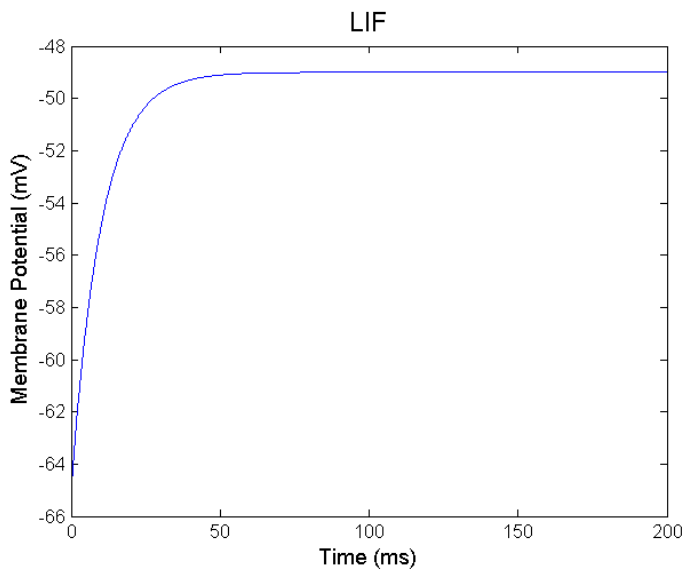
	Postsynaptic Neurons					
Presynaptic Neuron	N_{12}	N_{22}	N_{32}	N_{42}	N_{52}	N_{62}
N_{11}	0	1	0	0	1	0
N_{21}	1	0	1	0	1	0
N_{31}	0	1	0	1	0	1
N_{41}	0	0	1	0	1	1
N_{51}	1	1	0	1	0	0
N_{61}	0	0	1	0	0	0

effect on N_{22} than on N_{52}. How are these strengths going to be taken into consideration and represented? Are the strengths between neurons changing over time, and if so, how? Changes in network connections are often done following Hebb's rule which is paraphrased as "what fires together wires together" (Markram, Gerstner, & Sjöström, 2011). This means that if N_{11} fires than N_{22} fires their connection strength should increase. How neural connection strength increases (or decreases) is often a logarithmic function with its own rules and equations. As mentioned in tutorial 1, these tutorials are only looking at point neurons; but there are other more complex features that could be modelled to produce interesting behaviours. One such feature is axon length which can produce time differences, or delays, between when the presynaptic neuron fires and when the postsynaptic neuron receives the action potential (Maass, 1996). Because neurons have different axon length, it is possible that a presynaptic neuron that fires slightly after another neuron reaches the postsynaptic beforehand; or that a presynaptic neuron that fires before the postsynaptic fires reaches the postsynaptic neuron afterwards.

All the models covered here are spiking models, but what does the spike mean? Height and frequency may be important for coding information in the brain, along with spike pattern, but how that coding is achieved is still under debate in the neuroscience community (Ainsworth et al., 2012). There are two overall coding methods: rate coding which includes spike-count rate and time-dependent rate, and temporal coding which includes not only the rate aspect but the time aspect of coding (see Brette, 2015; Borst & Theunissen, 1999, for more information).

Considering all the above issues, SNNs models can become overwhelming and complex very quickly. With all the computations needed because of the nodes, the connections between nodes, and changing of connection strengths, simulations can take a long time to run. Picking an appropriate model of a neuron is an important step in developing the network.

Figure 7 ■ LIF with spike threshold of -45 mV, resting & reset at -65 mV, and a constant input of 1.6 mV will never spike.



Conclusion

The first tutorial covered how neural models are classified, how biological neurons work, and the Hodgkin-Huxley model. This tutorial reviews why neural models are often abstraction of biological models, Izhikevich's model, LIF model, parameter selection, and the basics of how neural networks are assembled. There is still a lot of work to do in spiking neuron models and related networks, but having a basic understanding of the neuron and how to model them is the first step in creating networks.

Authors' note

Correspondence concerning this article should be addressed to Melissa G. Johnson, School of Psychology, University of Ottawa, 75 Laurier Ave E, Ottawa ON, Canada, K1N 6N5. This research was supported in part by Natural Sciences and Engineering Research Council (NSERC) of Canada grants to S. Chartier.

References

Ainsworth, M., Lee, S., Cunningham, M. O., Traub, R. D., Kopell, N. J., & Whittington, M. A. (2012). Rates and rhythms: a synergistic view of frequency and temporal coding in neuronal networks. *Neuron*, *75*, 572–583. doi:10.1016/j.neuron.2012.08.004

Borst, A. & Theunissen, F. E. (1999). Information theory and neural coding. *Nature Neuroscience*, *2*(11), 947–957. doi:10.1038/14731

Brette, R. (2015). Philosophy of the spike: rate-based vs. spike-based theories of the brain. *Frontiers in Systems Neuroscience*, *9*, 151. doi:10.3389/fnsys.2015.00151

Chirimuuta, M. (2014). Minimal models and canonical neural computations: the distinctness of computational explanation in neuroscience. *Synthese*, *191*(2), 127–153. doi:10.1007/s11229-013-0369-y

Fields, R. D., Araque, A., Johansen-Berg, H., Lim, S.-s., Lynch, G., Nave, K.-A., & Wake, H. (2014). Glial biology in learning and cognition. *The Neuroscientist*, *20*(5), 426–431. doi:10.1177/1073858413504465

Gerstner, W., Kistler, W. M., Naud, R., & Paninski, L. (2014). *Neuronal dynamics: from single neurons to networks and models of cognition*. Cambridge: Cambridge University Press.

Gerstner, W. & Naud, R. (2009). How good are neuron models? *Science*, *326*(5951), 379–380. doi:10.1126/science.1181936

Gray, C. M. & McCormick, D. A. (1996). Chattering cells: superficial pyramidal neurons contributing to the generation of synchronous oscillations in the visual cortex. *Science*, *274*(5284), 109–113. doi:10.1126/science.274.5284.109

Guckenheimer, J. & Labouriau, J. S. (1993). Bifurcation of the hodgkin and huxley equations: a new twist. *Bulletin of Mathematical Biology*, *55*(5), 937–952. doi:10.1007/BF02460693

St-Hilaire, M. & Longtin, A. (2004). Comparison of coding capabilities of type i and type ii neurons. *Journal of*

Computational Neuroscience, *16*(3), 299–313. doi:10.1023/B:JCNS.0000025690.02886.93

Izhikevich, E. M. (2003). Simple model of spiking neurons. *Neural Networks, IEEE Transactions on*, *14*(6), 1569–1572. doi:10.1109/TNN.2003.820440

Izhikevich, E. M. (2004). Which model to use for cortical spiking neurons? *IEEE Transactions on Neural Networks*, *15*(5), 1063–1070. doi:10.1109/tnn.2004.832719

Johnson, M. G. & Chartier, S. (2017). Spike neural models part i: the hodgkin-huxley model. *The Quantitative Methods for Psychology*, *13*(2), 105–119. doi:10.20982/tqmp.13.2.p105

Kaplan, D. M. (2011). Explanation and description in computational neuroscience. *Synthese*, *183*(3), 339–373. doi:10.1007/s11229-011-9970-0

Kuznetsov, Y. A. (2013). Elements of applied bifurcation theory Science & Business Media: Springer.

Levy, A. & Bechtel, W. (2013). Abstraction and the organization of mechanisms. *Philosophy of Science*, *80*, 241–261. doi:10.1086/670300

Liu, Y.-h. & Wang, X.-j. (2001). Spike-frequency adaptation of a generalized leaky integrate-and-fire model neuron. *Journal of Computational Neuroscience*, *10*(1), 25–45. doi:10.1023/A:1008916026143

Maass, W. (1996). Lower bounds for the computational power of networks of spiking neurons. *Neural Computation*, *8*(1), 1–40.

Markram, H., Gerstner, W., & Sjöström, P. J. (2011). A history of spike-timing-dependent plasticity. *Frontiers in Synaptic Neuroscience*, *3*, 4–4. doi:10.3389/fnsyn.2011.00004

Mihalas, S. & Niebur, E. (2009). A generalized linear integrate-and-fire neural model produces diverse spiking behaviors. *Neural Computation*, *21*(3), 704–718. doi:10.1162/neco.2008.12-07-680

Vogels, T. P., Rajan, K., & Abbott, L. F. (2005). Neural network dynamics. *Annual Review of Neuroscience*, *28*(1), 357–376. doi:10.1146/annurev.neuro.28.061604.135637

Appendix A: Matlab code for Izhikevich Model

While it is generally recommended that one codes with functions and procedures, because of the simplicity of Izhikevich's model, and to save space, time, and make it easier to follow, that was not done in the below code. Instead, the code can be written as follows in one Matlab script.

First, parameters and initial variables are declared, including Izhikevich's models parameters, length of time, time step, and input:

```
% Parameters a, b, c, and d are based on Izhikevich's work
% Change these to display different spike patterns.
spikeType = 'Regular spiking'; % Just used for a heading of a graph and a reminder of what is being graphed
    .
a = 0.02; % change these 4 lines to produce the different spike patterns; see Figure 1 for different values.
b = 0.2;
c = -65;
d = 8;
maxTime = 200; % ms; how long to run the program
dt = 0.1; % time step
t = [0:dt:maxTime]; % vector of all time steps to run
Input = ones(1,length(t))*10; % mV; input, to change the constant input, change the multiplication number
    (10); or change all values to produce non-constant input

% setting the initial to values of membrane potential and recovery
v(1) = c % using auxiliary, after spike voltage is c
u(1) = v(1)*b % recovery is equal to the sensitivity (b) of the voltage (v)
```

With all the variables initialized, the next step is to calculate the membrane potential and recovery using Euler's method. This is done in a loop for all the time set in maxTime.

```
% Euler's method; for each time step, use differential equations
for i = 1:length(t)
  v(i+1) = v(i) + (0.04*v(i)^2 + 5*v(i) + 140 - u(i) + Input(i))*dt ; % equation 1
  u(i+1) = u(i) + (a*(b*v(i+1)-u(i)))*dt; % equation 2
  % equation 3 (auxiliary)
  if v(i+1) $>$= 30
    v(i+1) = c;
    u(i+1) = u(i)+d;
  end;
end
```

At this point, the vector v contains all the voltage values over time, and the vector u contains recovery values over time. These are used for the different graphs produced in this tutorial.

Figure 1 shows how the membrane changes over time and can be graphed with the following code:

```
figure('Color', 'white')
plot(t,v(1:end-1))
xlabel('Time (ms)', `fontsize', 12);
ylabel('Membrane Potential (mV)', `fontsize', 12);
title(spikeType, `fontsize', 14);
```

Most of the graphs show both membrane potential and recovery.

```
figure('Color', 'white')
plot(t,v(1:end-1), 'b', t,u(1:end-1), 'r') % the voltage is set to blue −− 'b', while the recovery is red
    −− 'r'
legend('Membrane Potential', 'Recovery');
xlabel('Time (ms)', 'fontsize', 12);
ylabel('mV', 'fontsize', 12);
title('Membrane potential vs Recovery', 'fontsize', 14);
```

Finally, Figure **??** is a plot that shows multiple parameters side by side for the same time period.

```
figure('Color', 'white')
subplot(2,2,1)
plot(t,v(1:end-1))
xlabel('Time (ms)', 'fontsize', 12);
ylabel('mV', 'fontsize', 12);
title(`Potential, v, (dt=0.1)', 'fontsize', 14);
subplot(2,2,2)
plot(t,u(1:end-1))
xlabel('Time (ms)', 'fontsize', 12);
ylabel('mV', 'fontsize', 12);
title(`Recovery, u, (dt=0.1)', 'fontsize', 14);
subplot(2,2,3)
plot(t,deltav*dt)
xlabel('Time (ms)', 'fontsize', 12);
ylabel('mV', 'fontsize', 12);
title('\Delta v', 'fontsize', 14);
subplot(2,2,4)
plot(t, deltau)
xlabel('Time (ms)', 'fontsize', 12);
ylabel('mV', 'fontsize', 12);
title('\Delta u', 'fontsize', 14);
```

Appendix B: Matlab code for Leaky Integrate & Fire

Like Izhikevich's model, LIF is a simple model that can be easily written in one script file. The first step is to initialize all the parameters being used.

```
dt = 0.1;          % ms; time step
```

```
endTime = 100;        % ms; amount of time that is shown in the graphs
t = [0:dt:endTime];      % ms vector; time points
Input = ones(1,length(t))*1.6;  % mv; input from presynaptic spikes and/or electrode
% membrane info
R = 10;          % MOhm; membrane resistance
tau = 10;         % ms; membrane time constant; RC
% spike  info
uresting = -65;    % mV; resting membrane potential
ureset = -65;     % mV; reset after spike; normally lower than resting
threshold = -55;   % mV; spike threshold
spike = NaN;       % mV; LIF doesn't actually fire; this forces  it .  Set  to  NaN to not  force  spike ,  otherwise  set
    it  to  a  value  higher  than  threshold
v(1)  = uresting;  % Initial voltage value
```

Once the values are initialized, loop though the time to get the new voltage value based on Equation 11.

```
i = 2;  % i is each change in  time ;  the   first   time  is   initial   values  set  above
while i $<$= length(t)

  v(i)  = uresting + Input(i).*R + (v(i-1)-(uresting + Input(i).*R))*exp(-dt/tau);  %
    Equation 11
  % if  potential  is  higher  than  threshold ,  spike  and reset  voltage  (Equation  10)
  if  (v(i)  $>$ threshold)
    if ~(isnan(spike))  % this if statement just displays the spike when requested
      v(i)  = spike;
      i = i + 1;
    end
    v(i)  = ureset;  % auxiliary from Equation 10
   end
  i = i + 1;
end  % while i
```

The only plot of LIF was the basic voltage over time plot.

```
set(gcf,'color','w');
plot(t,v)
title('LIF', 'fontsize', 14)
xlabel('Time (ms)', 'fontsize', 12);
ylabel('Membrane Potential (mV)', 'fontsize', 12)
```

Appendix C: Integrating LIF

When reading about the LIF model, there seem to be many different equations which can be confusing. The reason for the different equations is because the final integration is affected by the input. This appendix will cover how to derive the final equation based on three different types of input: constant input, continuous input, and spiking input.

Please note that in the LIF model, the function resets after firing. This means that while time, t, is a continuous increasing function, within the model it is not; it gets reset to 0 at firing, or is shown as $t - t_f$ – time since last fire as seen in equation 9. For the sake of comprehension, the work below assumes last fire occurred at $t_f = 0$. This assumption makes no difference to the final solution. For readability, the functions $v(t)$ and $I(t)$ are written with their parameters beside as subscript, v_t and I_t. This helps to avoid confusion between brackets used in the equation and brackets for functions.

Before looking at the three different input types, there are a few steps that are always done. From Equation 8:

$$\tau \frac{dv}{dt} = RI_t - v_t$$

This is a first-order linear differential equation (ODE), and can be put into the form $\frac{dy}{dx} + P(x)\,y = Q(x)$

$$\frac{dv}{dt} + \frac{1}{\tau} v_t = \frac{1}{\tau} R v_t$$

At this point, an integrating factor, $N(t) = e^{\int P(x)dx}$, is needed.[2]

$$N(t) = e^{\int \frac{1}{\tau} dt}$$

$$N(t) = e^{\frac{t}{\tau}}$$

Multiple both sides of the ODE by the integrating factor

$$e^{\frac{t}{\tau}} \frac{dv}{dt} + e^{\frac{t}{\tau}} \frac{1}{\tau} v_t = e^{\frac{t}{\tau}} \frac{1}{\tau} R I_t$$

Let

$$f(t) = e^{\frac{t}{\tau}} \text{ and } g'(t) = \frac{dv}{dt}$$

Therefore

$$f'(t) = \frac{1}{\tau} e^{\frac{t}{\tau}} \text{ and } g(t) = v_t$$

Using integration by parts $(f(t)\,g(t))' = f(x)\,g'(x) + f'(x)\,g(x)$

$$\frac{d}{dt}\left(e^{\frac{t}{\tau}} v_t\right) = e^{\frac{t}{\tau}} \frac{1}{\tau} R I_t$$

The above equation is the starting point for the following LIF solutions.

Constant Input

Constant input is the easiest solution and the least useful – it is very rare that there is a constant input going into neurons. But, constant input does provide a nice base for understanding the math.

$$\frac{d}{dt}\left(e^{\frac{t}{\tau}} v_t\right) = e^{\frac{t}{\tau}} \frac{1}{\tau} R I_t$$

Assuming constant input means that $I_t = I$ for all time.

$$\frac{d}{dt}\left(e^{\frac{t}{\tau}} v_t\right) = e^{\frac{t}{\tau}} \frac{1}{\tau} R I$$

Note: For the integration, we want to go from when the neuron last fired ($t_f = 0$) to current time t. Because of the use of the symbol t inside and outside the integration, there is a potential for confusion over what is bounded and what is not. Therefore, for integration, the variable s replaces the bounded t. This is particularly important when looking at continuous input.

$$e^{\frac{t}{\tau}} v_t = \int_0^t e^{\frac{s}{\tau}} \frac{RI}{\tau} ds + C$$

Properties of integrals states: $\int k f(x)\,dx = k \int f(x)\,dx$ where k is a constant

$$e^{\frac{t}{\tau}} v_t = \frac{RI}{\tau} \int_0^t e^{\frac{s}{\tau}} ds + C$$

$$e^{\frac{t}{\tau}} v_t = \frac{RI}{\tau} \left[\tau e^{\frac{s}{\tau}}\right]_0^t + C$$

[2] Integrating factor is normally designated by I(x) but the function I is already used for input and our formula is based on time, t. Therefore we are using $N(t)$ to designate the integrating factor.

$$e^{\frac{t}{\tau}} v_t = RI \left[e^{\frac{s}{\tau}}\right]_0^t + C$$

$$e^{\frac{t}{\tau}} v_t = RI \left(e^{\frac{t}{\tau}} - e^{\frac{0}{\tau}}\right) + C$$

$$e^{\frac{t}{\tau}} v_t = RI \left(e^{\frac{t}{\tau}} - 1\right) + C$$

$$v_t = e^{-\frac{t}{\tau}} RI \left(e^{\frac{t}{\tau}} - 1\right) + Ce^{-\frac{t}{\tau}}$$

$$v_t = RI \left(1 - e^{-\frac{t}{\tau}}\right) + Ce^{-\frac{t}{\tau}}$$

To solve for C, we know that the starting value, $v(0)$, without input, is v_{rest}.

$$v_{rest} = RI \left(1 - e^{-\frac{0}{\tau}}\right) + Ce^{-\frac{0}{\tau}}$$

$$v_{rest} = C$$

Therefore, the final solution when input is constant is

$$v_t = v_{rest} e^{-\frac{t}{\tau}} + RI \left(1 - e^{-\frac{t}{\tau}}\right)$$

To account for different firing times, this can be written as:

$$v_t = v_{rest} e^{-\frac{t-t_f}{\tau}} + RI \left(1 - e^{-\frac{t-t_f}{\tau}}\right)$$

Continuous Input

In simulations and experimentations, input might be a function of time such as $I(t) = cos(t)$. Equation 8 can still be integrated, to a point, even if the input function is not known. This solution is what is most commonly seen in books and articles on LIF (Gerstner et al., 2014, for example). Starting with Equation 8:

$$\tau \frac{dv}{dt} = RI_t - v_t$$

After the integrating factor is applied

$$\frac{d}{dt}\left(e^{\frac{t}{\tau}} v_t\right) = e^{\frac{t}{\tau}} \frac{1}{\tau} RI_t$$

$$e^{\frac{t}{\tau}} v_t = \int_0^t e^{\frac{s}{\tau}} \frac{RI_s}{\tau} ds + C$$

In the above, to make sure the bounded integrating variable is not confused with the variable t, the bounded variable is changed to s.

$$e^{\frac{t}{\tau}} v_t = \frac{R}{\tau} \int_0^t e^{\frac{s}{\tau}} I_s ds + C$$

$$v_t = \frac{R}{\tau} e^{-\frac{t}{\tau}} \int_0^t e^{\frac{s}{\tau}} I_s ds + Ce^{-\frac{t}{\tau}}$$

This is where it was really important to change the bounded (integrating) variable from t to s. Note here that $e^{-\frac{t}{\tau}}$ is a constant because t is a constant value. Therefore, by properties of integrals: $\int kf(x)\,dx = k \int f(x)\,dx$ where k is a constant

$$v_t = \frac{R}{\tau} \int_0^t e^{\frac{s}{\tau}} e^{-\frac{t}{\tau}} I_s ds + Ce^{-\frac{t}{\tau}}$$

$$v_t = \frac{R}{\tau} \int_0^t e^{\frac{s-t}{\tau}} I_s ds + Ce^{-\frac{t}{\tau}}$$

Now consider that s is bound between 0 and t, yet t is always a constant value regardless of s. Therefore $s-t=-s$.

$$v_t = \frac{R}{\tau}\int_0^t e^{\frac{-s}{\tau}} I_{t-s} ds + Ce^{-\frac{t}{\tau}}$$

We cannot simplify any further because the input function $I(t)$ is unknown.
To solve for C, we assume that at $v(0) = v_{rest}$.

$$v_{rest} = \frac{R}{\tau}\int_0^0 e^{\frac{-s}{\tau}} I_0 ds + Ce^{-\frac{0}{\tau}}$$

By properties of integrals $\int_a^a f(x) = 0$

$$v_{rest} = \frac{R}{\tau}(0) + C$$
$$v_{rest} = C$$

Therefore,

$$v_t = v_{rest}e^{-\frac{t}{\tau}} + \frac{R}{\tau}\int_0^t e^{\frac{-s}{\tau}} I_{t-s} ds$$

To get solution shown in equation 9, which accounts for time being continuous and spike firing happens during time t, replace t with $t - t_f$ for everything except input (which does not reset).

$$v_t = v_{rest}e^{-\frac{t-t_f}{\tau}} + \frac{R}{\tau}\int_0^{t-t_f} e^{-\frac{s}{\tau}} I_{t-s} ds$$

Spiking Input

The above continuous input is not often programmed for simulations. Instead, input is usually discrete values to represent spikes (or not spiking). In this case, input is constant for small periods of time, represented in the code as dt. So, from 0 to dt, input is a constant value $I(dt)$, from time dt to $2dt$, input is a constant value $I(2dt)$. This allows the ODE to be solved using piecewise integration. From original equation 8:

$$\tau\frac{dv}{dt} = RI_t - v_t$$

Applying integration factor to get:

$$\frac{d}{dt}\left(e^{\frac{t}{\tau}}v_t\right) = e^{\frac{t}{\tau}}\frac{1}{\tau}RI_t$$

Piecewise integration looks specifically at the bound range; area under the curve from a to b (0 to t). This means that the integration is always the difference from 0. In reality though, the difference is adjusted by the resting value. This means that using piecewise integration, to get the final solution, v_t will be wrong by v_{rest}. To account for a starting value, v_{rest}, we need to replace v_t with $v_t - v_{rest}$.

$$e^{\frac{t}{\tau}}(v_t - v_{rest}) = \int_0^t e^{\frac{s}{\tau}}\frac{RI_s}{\tau} ds$$

$$e^{\frac{t}{\tau}}(v_t - v_{rest}) = \frac{R}{\tau}\int_0^t e^{\frac{s}{\tau}} I_s ds$$

Note: this is the solution for time t.
Based on our discrete input, we actually have:

$$e^{\frac{t}{\tau}}(v_t - v_{rest}) = \begin{cases} \frac{R}{\tau}\int_0^t e^{\frac{s}{\tau}} I_{dt} ds & 0 < t \le dt \\ \frac{R}{\tau}\int_0^t e^{\frac{s}{\tau}} I_{2dt} ds & dt < t \le 2dt \\ \vdots & \vdots \\ \frac{R}{\tau}\int_0^t e^{\frac{s}{\tau}} I_{(k+1)dt} ds & k\ dt < t \le (k+1)\,dt \end{cases}$$

where each input, I_t, is a constant value for the time period lasting length dt. Because there are an infinite amount of time periods, to solve this integration, we use the same trick that proofs by induction use: assume we have solved the integration for up to time t (equation above); then solve for the next time step, $t + dt$, where from t to dt we know that the input is constant.

$$e^{\frac{t+dt}{\tau}}(v_{t+dt} - v_{rest}) = \frac{R}{\tau}\int_0^{t+dt} e^{\frac{s}{\tau}} I_s ds$$

By properties of integrals: $\int_a^c f(x) = \int_a^b f(x) + \int_b^c f(x)$ where $a < b < c$

$$e^{\frac{t+dt}{\tau}}(v_{t+dt} - v_{rest}) = \frac{R}{\tau}\left(\int_0^t e^{\frac{s}{\tau}} I_s ds + \int_t^{t+dt} e^{\frac{s}{\tau}} I_s ds\right)$$

From above we know that $e^{\frac{t}{\tau}}(v_t - v_{rest}) = \frac{R}{\tau}\int_0^t e^{\frac{s}{\tau}} I_s ds$

$$e^{\frac{t+dt}{\tau}}(v_{t+dt} - v_{rest}) = e^{\frac{t}{\tau}}(v_t - v_o) + \frac{R}{\tau}\int_t^{t+dt} e^{\frac{s}{\tau}} I_s ds$$

We know $I(s)$ is constant from t to $t + dt$, therefore let that value equal $I(s) = I(t + dt)$

$$e^{\frac{t+dt}{\tau}}(v_{t+dt} - v_{rest}) = e^{\frac{t}{\tau}}(v_t - v_o) + \frac{R}{\tau}\int_t^{t+dt} e^{\frac{s}{\tau}} I_{t+dt} ds$$

$$e^{\frac{t+dt}{\tau}}(v_{t+dt} - v_{rest}) = e^{\frac{t}{\tau}}(v_t - v_o) + \frac{RI_{t+dt}}{\tau}\int_t^{t+dt} e^{\frac{s}{\tau}} ds$$

$$e^{\frac{t+dt}{\tau}}(v_{t+dt} - v_{rest}) = e^{\frac{t}{\tau}}(v_t - v_o) + \frac{RI_{t+dt}}{\tau}\left[\tau e^{\frac{s}{\tau}}\right]_t^{t+dt}$$

$$e^{\frac{t+dt}{\tau}}(v_{t+dt} - v_{rest}) = e^{\frac{t}{\tau}}(v_t - v_o) + RI_{t+dt}\left[e^{\frac{s}{\tau}}\right]_t^{t+dt}$$

$$e^{\frac{t+dt}{\tau}}(v_{t+dt} - v_{rest}) = e^{\frac{t}{\tau}}(v_t - v_o) + RI_{t+dt}\left(e^{\frac{t+dt}{\tau}} - e^{\frac{t}{\tau}}\right)$$

$$(v_{t+dt} - v_{rest}) = e^{\frac{t}{\tau}} e^{-\frac{t+dt}{\tau}}(v_t - v_o) + RI_{t+dt}\left(e^{\frac{t+dt}{\tau}} - e^{\frac{t}{\tau}}\right)e^{-\frac{t+dt}{\tau}}$$

$$(v_{t+dt} - v_{rest}) = e^{-\frac{dt}{\tau}} v_t - e^{-\frac{dt}{\tau}} v_o + RI_{t+dt} - RI_{t+dt}e^{-\frac{dt}{\tau}}$$

$$v_{t+dt} = e^{-\frac{dt}{\tau}} v_t - e^{-\frac{dt}{\tau}} v_o + RI_{t+dt} - RI_{t+dt}e^{-\frac{dt}{\tau}} + v_o$$

$$v_{t+dt} = v_{rest} + RI_{t+dt} + (v_t - (v_o + RI_{t+dt}))e^{-\frac{dt}{\tau}}$$

which is equivalent to Equation 11 by replacing t by $t - 1$ and $t + dt$ with t (instead of assuming current step, solve next; assume previous, solve current).

$$v_t = v_{rest} + RI_t + (v_{t-1} - (v_0 + RI_t))\, e^{-\frac{dt}{\tau}}$$

Citation

Johnson, M. G. & Chartier, S. (2018). Spike neural models part II: Abstract neural models. *The Quantitative Methods for Psychology*, *14*(1), 1–16. doi:10.20982/tqmp.14.1.p001

Received: 04/09/2017 ~ Accepted: 03/10/2017