Backpropagation in Biological Neural Networks Yields Hebbian Dynamics

James Hazelden[1,3] Michael Ivanitskiy[5] Eli Shlizerman[1,2]
Daniel B Forger[3,4*]

1. University of Washington, Department of Applied Mathematics
2. University of Washington, Electrical & Computer Engineering
3. University of Michigan, Department of Mathematics
4. University of Michigan, Department of Computational Medicine and Bioinformatics
5. Colorado School of Mines, Department of Applied Mathematics

* forger@umich.edu

Abstract

Although deep neural networks (DNNs) were originally inspired by *in vivo* neural networks, their individual units do not much resemble the dynamics of neurons. Biological neurons have multivariate and time-varying internal state, and are thus capable of many dynamics not exhibited by traditional elements of DNNs, such as bursting, chaos, and bistability. Here, we apply techniques effective for training DNNs to training networks of more biologically accurate neurons. We develop an approach applicable to networks where each neuron is represented by biophysical (ionic) models of biologically realistic neurons (BNNs) whose individual neuron dynamics closely match the function of *in vivo* neurons. We provide evidence that BNNs learn more quickly than equivalently structured DNNs in the low-data regime. We observe that neurons with drastically different individual behaviors learn similarly and analyze the networks' loss landscape with respect to variation of physiological parameters. Notably, we show that our implementation of stochastic gradient descent through backpropagation behaves very similarly to Hebbian learning found in biological networks. This work further studies how biologically realistic neurons can solve complex tasks and provides new directions for interpreting machine learning systems.

1 Introduction

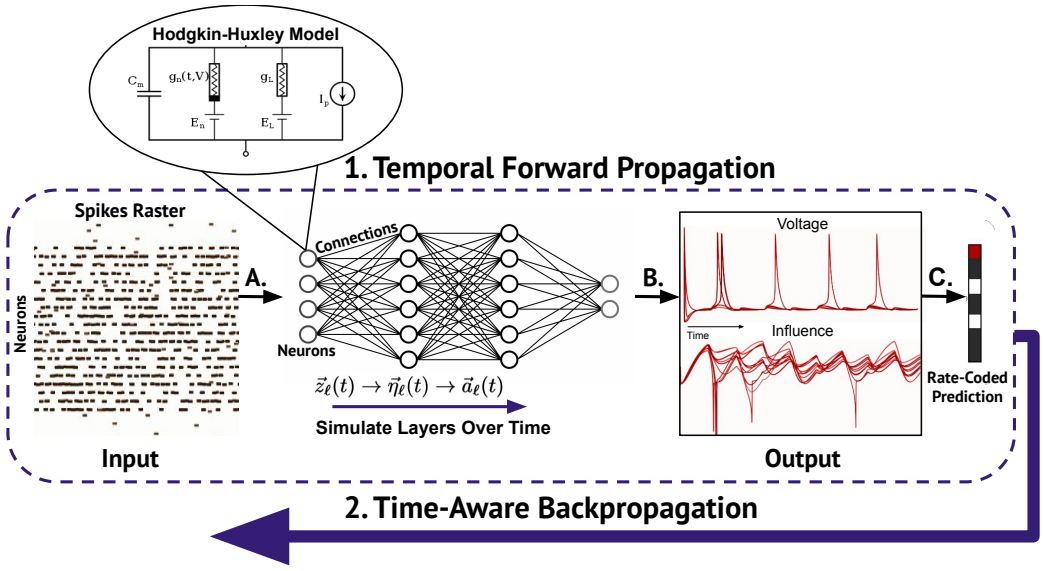

Figure 1: Time-aware training broken down into two stages per training sample: 1. Feed forward and 2. Backpropagation. A. An input in the form of a neuronal spiketrain is fed into a biological neural network (BNN) over some timeframe determined by the data or constraints. B. The BNN along with backpropagation relevant quantities are numerically simulated over the simulation timeframe. Neurons in the BNN have ionic dynamics determined by a system of ordinary differential equations (ODEs) in the Hodgkin-Huxley format, as shown in this example. C. The network output over time is quantified in some way–in our case, we use rate coding and select the neuron with the highest net output to "win" a classification task. 2. After feeding forward, the network average output over the simulation timeframe is backpropagated in a way that is "time-aware" in the sense that we consider all previous times' influence $s \leq t$ on the network state at each time t. This is contrasted with instantaneous learning rules which only factor in the current state at each time t without factoring in the influence of the past.

In recent years, artificial neural networks have provided solutions to innumerable tasks once thought to be the domain of only biological intelligence. These networks, for reasons primarily to do with the efficiency of implementation on existing computing hardware, differ significantly from their biological counterparts. Such networks build upon the perceptron neuron model proposed in 1958 by F. Rosenblatt, which models neurons as stateless non-linear functions with no temporal dynamics [7]. However, a fundamental problem with such networks is our current limited understanding of *how* the network learns to solve any given task – which, when applied to biological neural networks, is a core problem of neuroscience itself.

Machine Learning (ML) is often described as being inspired by the brain. Indeed, the idea of a perceptron, having inputs with varying learned weights that converge towards a "neuron," does bear some structural resemblance to real neuronal networks as do convolutional neural networks [7]. However, the "artificial" neurons (ANs) on which ML is based behave

very differently from how real neurons behave [1] [17]. The classical description of signal processing in a biological neuron, developed by Hodgkin and Huxley, shows that biological neurons (BNs) can be described accurately by non-linear differential equations [13]. While this can lead to "spiking" or "firing" behavior, signal processing by BNs is much more complex than even described by "spiking" artificial neural networks [4] [5] [6] [8] [10] [12] [9] [11] [12] composed of, for example, leaky integrate-and-fire units [19]. Moreover, as biophysical parameters (e.g., the equilibrium potentials of ions, average inputs, etc.) change, the signal processing by a single neuron can drastically change in ways not predicted by simple models [3].

The classical mechanism underlying learning in the brain is mainly through Hebbian learning. In the simplest description of Hebbian learning, weights between neurons only change when both the presynaptic and postsynaptic neurons fire. When the presynaptic neuron fires just before the postsynaptic neuron, the synaptic weight is regulated upward. When the presynaptic neuron fires just after the postsynaptic neuron, the synaptic weight is regulated downward. However, Hebbian learning is simply a localized learning rule based on correlations between neurons with no notion of loss or global results of a network in the brain. Approached using Hebbian learning to solve simple machine learning tasks typically achieves lesser accuracy on common ML benchmarks than backpropagation.

This raises fundamental questions: i) How can we use the efficient methods from ML to train networks of BNs? ii) Would these biological neuronal networks (BNN) have some advantages over artificial neural networks (DNNs) in terms of learning? iii) How does Hebbian learning on BNNs relate to backpropagation? iv) When trained, are BNNs robust to the kinds of parameter variation typically seen in biology? v) When these parameters are changed enough so that models show different dynamic behaviors, are similar solutions found? Here we address these questions by developing a mathematical algorithm that applies backpropagation to networks of Hodgkin-Huxley type neurons.

2 Methodology

We first introduce the methodology for a general network of neuron models that are modeled by ordinary differential equations (ODEs). Subsequently, the specifics of using the Hodgkin-Huxley (HH) and similar ionic models is outlined.

2.1 Computing Partial Derivatives

We consider a feedforward network of neurons with the following terminology at layer ℓ, taken at time t:

$$\ldots \xrightarrow{W_\ell} \vec{z}_\ell(t) \to \eta_\ell(t) \to \vec{a}_\ell(t) \xrightarrow{W_{\ell+1}} \ldots, \tag{1}$$

where W_ℓ denotes a matrix of weights associated with the synaptic connections into the layer ℓ, $\vec{z}_\ell(t)$ is the weighted input vector, $\vec{a}_\ell(t)$ is the unweighted output or activation of this layer and $\eta_\ell(t)$ is a solution of a system of non-autonomous ODEs for each neuron in the layer. Note that each neuron may be modeled by multiple interconnected variables, so for each neuron i in layer ℓ, $\eta_{\ell,i}$ is itself a vector of state variables. η_ℓ is modeled based on the current state as well as the input $\vec{z}_\ell$:

$$\frac{\mathrm{d}}{\mathrm{dt}}\eta_\ell(t) = f(\vec{\eta}_\ell(t'), \vec{z}_\ell(t)), \tag{2}$$

where t denotes the state of the system at time t and $t' \leq t$ is some prior time. Common experimental neuron models such as Fitzhugh, Morris-Lecar and Hodgkin-Huxley all belong to this class of neuron models [16]. In order to compute weight updates using gradient descent in a standard DNN, the loss is backpropogated through the network of neurons, iteratively applying the chain rule. Notably, Equation 2 indicates that the state η_ℓ depends on the instantaneous input to the neuron as well as the prior state, so to backpropogate through each layer, the gradients with respect to weighted input $z_\ell(t)$, and with respect to the previous state, $\eta_\ell(t')$ need to be computed.

In particular, let η denote a single neuron from the network and z denote this neuron's weighted input. Then, assuming continuous second partial derivatives of f,

$$\frac{\mathrm{d}}{\mathrm{dt}}\frac{\partial \eta}{\partial z} = \frac{\partial}{\partial z}\frac{\mathrm{d}}{\mathrm{dt}}\eta = \frac{\partial f}{\partial z} \tag{3}$$

This yields a differential equation for the derivative of η with respect to its weighted input. The right hand side can be explicitly computed since f is known (although, in practice, this is not necessary, as described in the implementation details).This quantity per neuron is numerically simulated along with the state η during the feed-forward phase of our method.

Additionally, in Equation 2, there is a dependence on the previous state of η. In particular, it is necessary during backpropagation to quantify the partial derivative of the state η at time t, with respect to the previous state $\eta(t')$, where $t' \leq t$. This is computed in a similar way as above:

$$\frac{\mathrm{d}}{\mathrm{dt}}\frac{\partial \eta_\ell(t)}{\partial \eta_\ell(t')} = \frac{\partial}{\partial \eta_\ell(t')}\frac{\mathrm{d}}{\mathrm{dt}}\eta_\ell(t) = \frac{\partial f}{\partial \eta_\ell(t')}. \tag{4}$$

These combined differential equations measure the partial derivatives with respect to weighted input at a particular time and the state at all previous times. Once these are computed, it remains to derive how they can be used for a learning rule. In our method, we numerically integrate the above equations over a simulation timeframe according to a rate-coding scheme, as detailed below.

2.2 Rate-Coded Time-Aware Learning

In using a model with underlying temporal dynamics, the question of how to factor time into the learning rule is important. It is clear that the neurons in the model must be simulated for some amount of time to infer how they behave. Adding further complexity, the method must be "time-aware", i.e. the influence of previous timesteps on the current state of the network must be considered. This is of much importance in real biological networks. As well, neuronal delays can mean that a downstream neuron firing can impact upstream neurons later in time. Our proposed method solves these two criteria by "rate-coding" over a simulation time frame and proposing a time-aware learning rule .

The rate-coding scheme is implemented by averaging the output of the last layer of neurons (output layer) of the network over some predetermined simulation timeframe. In particular, we numerically simulate the network over a particular timeframe and approximate

$$y := \frac{\int_0^\tau \vec{a}_L(t)dt}{\tau}, \tag{5}$$

where L is the final output layer and τ is the timeframe in milliseconds (this will either be determined by hyperparameter sweeping or by the data trained on). Once this quantity is

calculated, the loss function is applied to the total average output, y, in comparison to some desired output y^*. For example, for a classification task, y^* is a one-hot encoding of the target classification for each training sample. Our method is akin to rate-coding because the neuron that spikes for the longest amount of time during the simulated timeframe will "win" the classification task. Note that other metrics on the output of the network are also possible potentially encapsulating more of the information in the system than rate-coding.

Now, we detail the update rule that will compute the changes to the connectivity weights. Suppose w is a connectivity parameter in the output layer. Then,

$$\frac{\partial L}{\partial w} = \nabla_y L \cdot \frac{\partial y}{\partial w} = \frac{1}{\tau} \nabla_y L \cdot \int_0^\tau \frac{\partial \vec{a}_L(t)}{\partial w} dt. \tag{6}$$

Since the derivative is brought inside the integral, it suffices to backpropagate starting from the quantity $\frac{\partial a_L(t)}{\partial w}$. By the chain rule,

$$\frac{\partial \vec{a}_L(t)}{\partial w} = \frac{\mathrm{d}\vec{a}_L(t)}{\mathrm{d}\eta_L(t)} \cdot \frac{\partial \eta_L(t)}{\partial w}, \tag{7}$$

where $\frac{\mathrm{d}\vec{a}_L(t)}{\mathrm{d}\eta_L(t)}$ is a vector of component-wise derivatives. If $t' < t$ is the previous timestep considered during the feed-forward phase, the second term above can be expanded,

$$\frac{\mathrm{d}\eta_L(t)}{\mathrm{d}w} = \frac{\mathrm{d}\eta_L(t)}{\mathrm{d}\vec{z}_L(t)} \frac{\partial \vec{z}_L(t)}{\partial w} + \frac{\mathrm{d}\eta_L(t)}{\mathrm{d}\eta_L(t')} \cdot \frac{\partial \eta_L(t')}{\partial w}. \tag{8}$$

This holds since η_L is a function of the current weighted input as well as the previous state at time t', as in Equation 2. Since $z_L(t)$ is the weighted input, it is just a linear function of w. Furthermore, the rightmost partial is already computed because $t' < t$. The two remaining terms, $\frac{\mathrm{d}\eta_L(t)}{\mathrm{d}\vec{z}_L(t)}, \frac{\mathrm{d}\eta_L(t)}{\mathrm{d}\vec{\eta}_L(s)}$, are given by the differential equations 3, 4. Thus, the above can be computed, resulting in the derivative for all parameters in the output layer. For parameters in the hidden layers, the approach is repeated to backpropogate further into the network. While the equation above involves multiple components and appears complex to compute, the implementation turns out to be surprisingly efficient and straightforward, as detailed below. Furthermore, as in backpropagation on DNNs, it is in fact not necessary to compute all partial derivatives with respect to parameters. Instead, these can be computed from the vector $\frac{\mathrm{d}a_\ell}{\mathrm{d}z_\ell}$ using an outer product, since:

$$D_{W_\ell} a_\ell = \frac{\mathrm{d}a_\ell}{\mathrm{d}z_\ell} \cdot D_{W_\ell} z_\ell = \frac{\mathrm{d}a_\ell}{\mathrm{d}z_\ell} a_{\ell-1}^T,$$

where D denotes the total derived operator and $a_{\ell-1}^T$ is the transpose of $a_{\ell-1}$.

With the total derivative computed, weights are updated by a gradient descent rule with learning rate α. For example,

$$w \mapsto w - \alpha \frac{\mathrm{d}L}{\mathrm{d}w}.$$

Note that the second term in Equation 8 makes our approach "time-aware". In particular, the influence of previous states on the current state of the network is factored into the update rule. We believe this is paramount to designing a realistic learning rule for such neurons because neurons can have a delayed response to input stimuli not captured by a purely instantaneous perspective.

2.3 Implementation Details

Measuring all the aforementioned partial derivatives according to their own individual differential equations appears impractical and complex but in practice it can be done with ease using automatic differentiation packages such as PyTorch. In particular, such packages record the influence of each state variable on the output of the network as well as how the state at the previous timesteps influences the state at the next. Thus, in the implementation, there is no need to explicitly compute the partial derivatives that are needed for backpropagation because they are implicitly recorded by PyTorch (see supplement for more details on autodiff tree produced). Most of the code focuses on simulating the neuron model over a specified timeframe and measuring the averaged output of the network. Thus, the method fulfills the criteria of being time-aware without significant complexity in implementation.

We now outline the specific choice Hodgkin-Huxley (HH) for the neuron states η_ℓ used throughout this paper. In this case, the terminology at layer ℓ becomes:

$$\dots \xrightarrow{W_\ell} z_\ell \to \begin{pmatrix} \vec{V}_\ell \\ \vec{m}_\ell \\ \vec{n}_\ell \\ \vec{h}_\ell \\ \vec{y}_\ell \end{pmatrix} \to \vec{a}_\ell \xrightarrow{W_{\ell+1}} \dots \tag{9}$$

where, as above, W_ℓ is the matrix of incoming weights applied to the input to the layer and $\vec{z}_\ell$ denotes weighted input. $\vec{V}_\ell$ is the vector of voltages for each neuron in the layer, $\vec{m}_\ell, \vec{n}_\ell, \vec{h}_\ell$ are gating variables pertaining to the HH model for each neuron, $\vec{y}_\ell$ is a gating variable per neuron regulating synaptic input, and $\vec{a}_\ell$ is the output of the neuron, which is a function that indicates the output of a neuron from 0 to 1.

The model then has the equations:

$$\frac{d\vec{V}_\ell}{dt} = I_{app} - g_{Na}\vec{m}_\ell^3\vec{h}_\ell(\vec{V}_\ell - V_{Na}) - g_K\vec{n}_\ell^4(\vec{V}_\ell - V_K) - g_l(\vec{V}_\ell - V_l) - g_s\vec{y}_\ell(\vec{V}_\ell - V_s) \tag{10}$$

$$\frac{d\vec{\eta}_\ell}{dt} = \alpha_x(\vec{V}_\ell)(1 - \vec{\eta}_\ell) - \beta_x(\vec{V}_\ell)\vec{\eta}_\ell, \text{ for } x = m, n, h \tag{11}$$

$$\frac{d\vec{y}_\ell}{dt} = A_d\vec{z}_\ell(1 - \vec{y}_\ell) - A_r\vec{y}_\ell, \tag{12}$$

$$\vec{a}_\ell(\vec{V}_\ell) = \frac{1}{1 + \exp(-\frac{\vec{V}_\ell - V_t}{K_p})}, \tag{13}$$

where the functions α_x, β_x per gating variable are sigmoid like functions whose derivatives can be written in terms of their value (see supplement for full details). Using a sigmoid to measure output of neurons is standard practice in many biological models [16]. Note that no process such as softmax or normalization is applied to the final layer in the network. A variant of the model above is to use "gap-junctions" for connectivity. In this case, the variable $\vec{y}_\ell$ is not used and the weighted output from previous layers is fed into the neurons in the next layer via a direct input current. In our work, it was found that this style of connectivity resulted in facilitated faster and more accurate learning, so results below use gap junctions, although the synaptic gating variable approach is also supported in the code and is trainable with our method.

We numerically simulated the model using the trapezoidal rule which can be reduced to an explicit update for this model [21]. This rule allows for a stable update with a large

timestep. We used the Adam optimizer and a 2000 timestep simulation timeframe per batch of data. We optimized only weights without biases although factoring biases into the method may lead to better accuracy and could measure what baseline input currents are optimal for learning on different tasks.

To apply to the MNIST benchmark, we converted each input image to a spiketrain over the simulation timeframe according to a Poisson process. This conversion is somewhat standard for training on MNIST with spiking neural networks so we will not repeat the details here [2].

3 Results

3.1 Network architecture

We consider a relatively standard DNN architecture to solve the problem of classifying images of handwritten digits as 0-9, from the MNIST dataset [14]. Each image consists of 784 pixels which we convert to spiking inputs based on a Poisson process per pixel [2]. These inputs are then passed to 100 hidden neurons, which then project to 10 output neurons. Correct labels are encoded as a one-hot vector, and the loss is computed as the mean-squared-error between the output vector of the network and the vector with correct label. Output is measured by computing the mean output of the Hodgkin-Huxley spiking output neurons over time. Thus, we use a rate-coding scheme to measure network output. For each neuron, we use the Hodgkin-Huxley formalism, however, with a parameter set matching mammalian cortical neurons [3]. Matching biological networks, we pass the voltage from the output neuron into a simple model of synaptic transmission to determine the overall output of a neuron (which then varies between 0 and 1; see methods). Synapses carry outputs of neurons directly as input current to downstream neurons. While we also implemented synapses using gating variables, we found the former approach learned more efficiently, so we focus on it here. Details of our setup are shown in figure 1.

Our implementation of gradient descent through backpropagation uses direct backpropagation-through-time (BPTT), and the resultant computational graph from the backpropagation of derivatives is far more complex than for equivalent DNN architectures (See Supplemental Figure 3). The fundamental question we ask is how much each weight in the network influences network dynamics and the resultant loss. Since changes in the weight at previous timepoints can affect the output at future timepoints (see methods), we study the gradient of each weight over a time interval that is typically longer in time than that used in the loss function. To train the network, we use the same time interval to calculate the weight gradients and the loss functions.

3.2 Accuracy on MNIST Benchmark

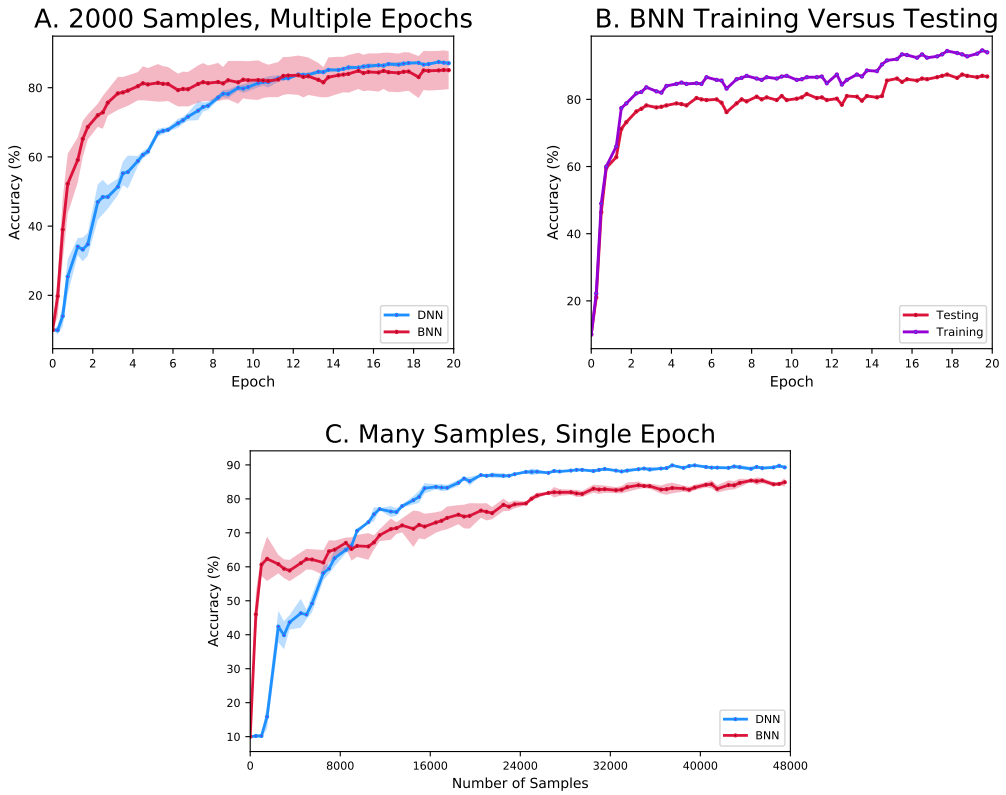

Figure 2: A. Accuracy of our technique compared to DNN trained on the same number of samples and with the same network structure (1 layer of 100 hidden neurons, 2000 samples per epoch). As can be seen, we can achieve comparable accuracy to a DNN. Furthermore, faster convergence over epochs is observed in almost all cases with our technique in comparison to the DNN. B. Comparison of training versus testing accuracy for the BNN model indicating negligible overfitting. C. Training on 48,000 samples for a single epoch. Again, faster convergence and similar final accuracy is observed. After many runs, we show the range where the network initialization and optimization seeds were randomized between runs.

We trained BNNs on a spiking analog of the MNIST dataset and compared to DNNs on this benchmark. Surprisingly, BNNs achieved greater accuracy during the early epochs of training on a small number of samples (2000 fixed samples per epoch) than DNNs (Figure 2a). We did not include learned biases in either model for simplicity. Both were able to achieve accuracies over 80% within 20 epochs training only on this subset of the data. A range of accuracies for different initializations and optimization seeds is also shown, indicating that for a wide range of initial conditions, high accuracies were quickly obtained for BNNs. Figure 2b indicates that learning is likely not resulting in overfitting. Finally, we trained for a single epoch on the full dataset in figure 2c, demonstrating comparable accuracy at around 90%. Taken together, this shows that our methodology can train BNNs in a comparable way

to DNNs.

3.3 Learning Mechanisms and Results

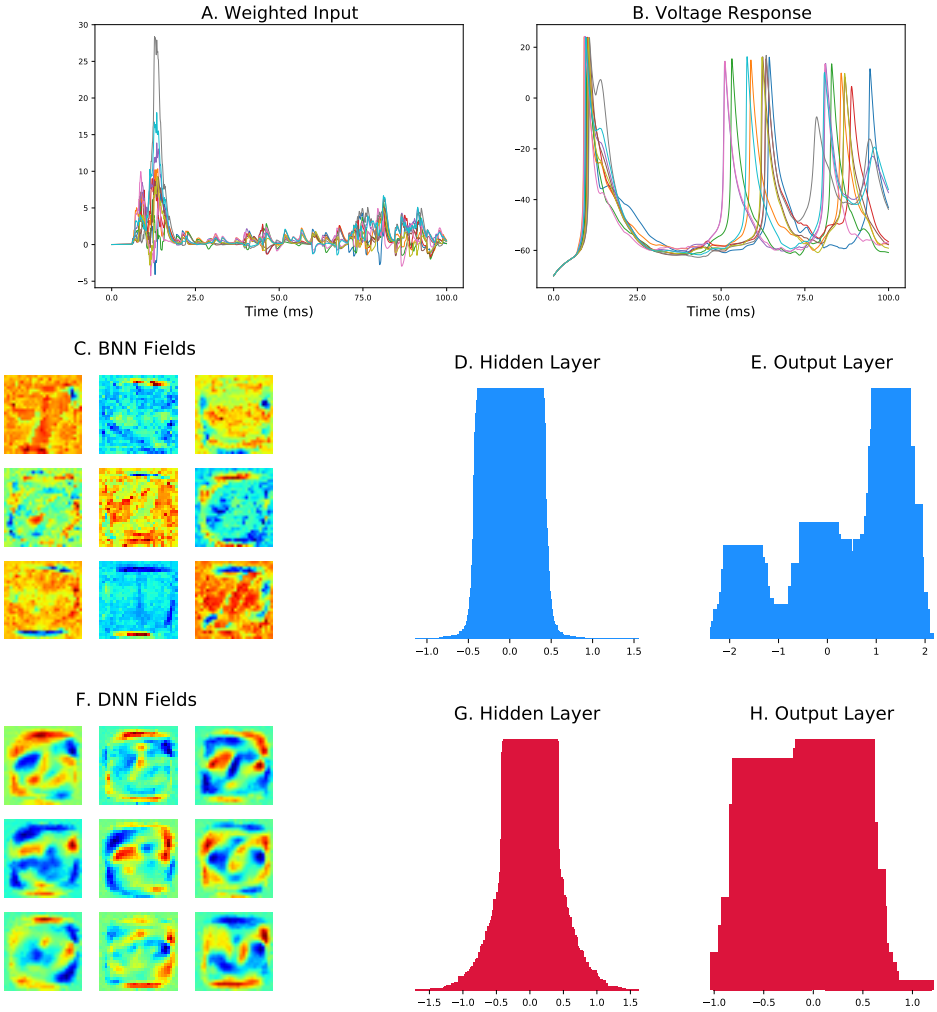

Figure 3: A. An example of typical weighted inputs to the ten neurons of the output layer for the trained model from Figure 2C. B The corresponding voltage trace induced by weighted input simulated over the simulation timeframe chosen Figure 2.C. Some example perceptive fields occurring in the hidden layer for this trained model. Notice features such as a 2 in the middle field and a 7 in the top left field. D-E. Histograms of hidden and output layer weights for this trained model. F-H. Corresponding plots of perceptive fields and histograms for the trained DNN model.

We next sought to analyze how learning works internally in BNNs with backpropagation, as well as the resulting dynamics. Weights in the hidden and output layers were initialized from a uniform distribution, as is standard practice. Figure 3a, and b shows weighted input to the output layer and output neuron traces given this input, respectively. We next plotted the "receptive" fields of the hidden layer neurons for the BNN (Figure 3C) and the DNN (Figure 3E) after training. Nine example receptive fields are shown with all 100 shown in Supplemental Figure 2. These fields represent the weight that each hidden layer neuron places on each pixel. The distribution of these weights is also shown for the BNN (Figure 3D) and DNN (Figure 3F). Surprisingly, the distribution of weights for the BNN was much less changed by the learning process than in the DNNs. The BNN receptive fields remained fairly uniform except for key select pixels which were greatly changed, whereas, in the receptive fields of the DNNs, the receptive fields show more weights that have changed. While the receptive fields of the DNN changed more gradually, showing several regions of inhibition or excitation, the receptive fields of the BNN were less coarse-grained, and thus more of the digits could be seen.

The BNN also resulted in a different distribution of learned weights in comparison to the DNN. Figures 3d, e and 3g, h show the distributions of the hidden and output layers for the BNN and DNN, respectively. For these later synapses, the trained BNN had a trimodal distribution interpretable as the "inhibition," "neutral or indifferent" or "excitation" weights seen in biophysical networks. This shows one key area where our training of BNNs matches what is often found in biological networks.

We next explored the properties of the fitted BNNs in Figures 3A, B. The initial spike in input and voltage is transient due to initial conditions and did not play a significant role in learning (e.g., initial gradients are low in Figure 4). As one can see, there is a very non-linear relationship between the weighted input and the output layer activities. Oscillations and voltage peaks of different heights and shapes are observed in the learned behavior. Some neurons (e.g., that shown in grey) have more peaks than other neurons. Furthermore, inputs can also extend the duration of an action potential or show depolarized lower amplitude membrane oscillations (DLAMOs) seen in some neuronal systems [18]. These offer some of the complex behaviors seen in the model.

3.4 Emergence of Hebbian-like Learning Rule

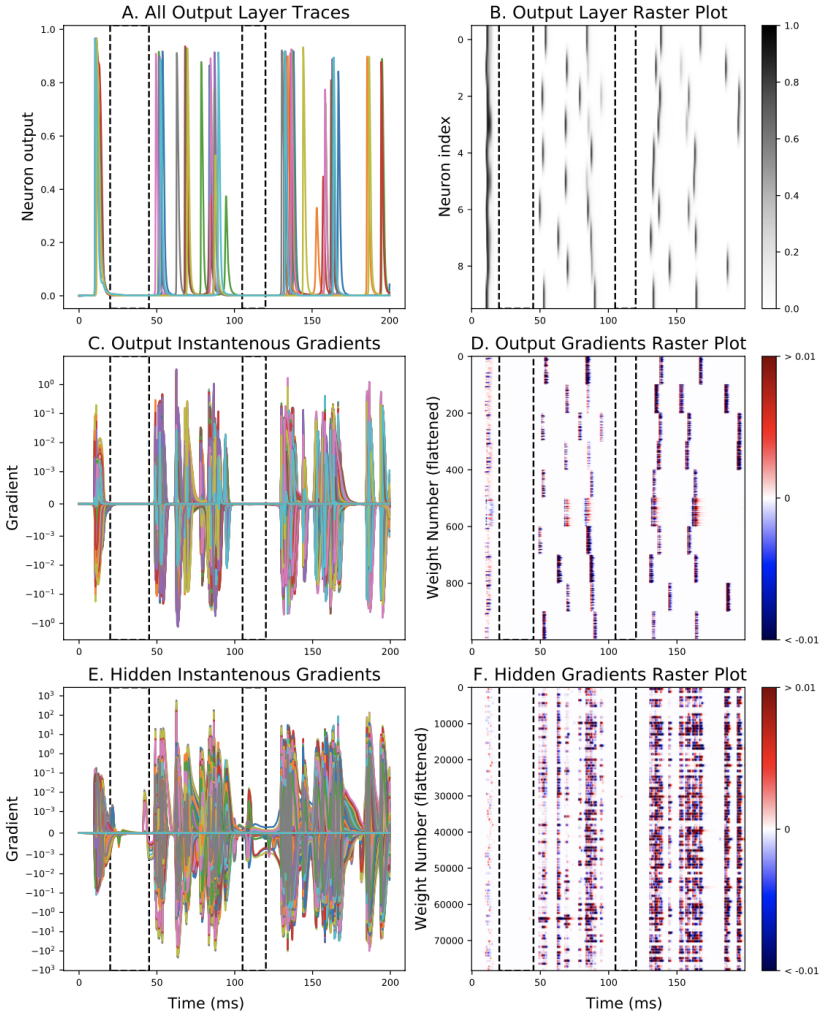

Figure 4: Comparison of instantaneous gradients with our method to neuron activity. As can be seen, gradients are almost always zero except when spiking happens. A-B. Show network output in a 200 ms timeframe. Note that A. shows output, not the voltage of neurons. C-D. Show the corresponding instantaneous gradients over this timeframe for all weights from the hidden to the output layer. Weights were flattened into a 2D array row-wise so that all weights going into a particular neuron are in consecutive rows. Note the correlation between spiking in the output layer in B. and changes to that specific neuron's input connections in D. E-F. Show the corresponding plots for C-D but for the hidden layer neurons. The dashed regions show areas where no neurons were active.

We next explored how our method results in learning in the BNN, by recalculating the gradient in a learned network but only for the mean activity in a small sliding time window

(1 mSec). In this way, we explored how prior changes in the weight can affect the spiking output of the network at a particular time (See Figure 4). In Figure 4B we plot the rasterized output over time of the 10 output neurons. In Figure 4C-D we plot the gradients of each of the synaptic output weights with respect to each output neuron in a flattened fashion, so that the weights to neuron 0 are shown first, then neuron 1, etc, in order of their post-synaptic target. Likewise, 4E-F show hidden layer gradients.

The gradients can be compared with Hebbian learning in that they determine how much individual weights are upregulated or downregulated at each gradinent descent step. Making this comparison, we observe the following interesting patterns, including:

- Gradients are near zero for most times and non-zero only when close to when the output neuron fires, mirroring Hebbian learning.
- The sign of the gradient switches just as the output neuron fires. This mimics Hebbian learning and spike timing dependent plasticity (STDP) where the sign of δ_t is inverse to the sign of the weight update, where δ_t is the difference between the time at which the presynaptic and postsynaptic neuron fire. That is, when the presynaptic neuron fires before the postsynaptic neuron, the connection is strengthened. The switching also bears resemblance to the "chirp" observed in the optimal signal to excite a neuron [3].
- In addition to dependence on voltage, the gradients also depend on state variables. For example, at the beginning of the simulation, gradients are very small even though similar action potentials are seen at later times in the simulation. This is because the additional variables (beyond the voltage) in the model are in initial states such that inputs do not have a large effect on producing the action potential.
- In general, the gradients were of smaller magnitude if the output of the neuron is of small magnitude as well. This is somewhat counterintuitive since there is a maximum output level, and one might imagine that further increases in the weight would yield fewer effects as the output neuron shows its maximum output.

In summary, learning in the BNN with our backpropagation method shows patterns similar to Hebbian learning. Unlike Hebbian learning, however, gradients are exactly estimated and used for up/down-regulation of weights. This could be partly the reason for the results to achieve better accuracy than what is typically seen with Hebbian learning.

Notably, the mathematical algorithm does not include any components that impose the emergence of Hebbian-style learning. These correlations with Hebbian learning thereby suggest that backpropagation and Hebbian-like learning observed in the brain could be related and backpropagation could emerge as biologically plausible Hebbian learning in the brain. However, the specific patterns of learning depend on many factors, including how many input neurons are firing, the average input to a neuron, the prior history (initial conditions) of a neuron, the timing of the firing of the postsynaptic neuron, and what the loss function is. Based on all of this, we hypothesized that the specific parameters of the neuron might greatly affect the performance of the model.

3.5 Robustness of the model

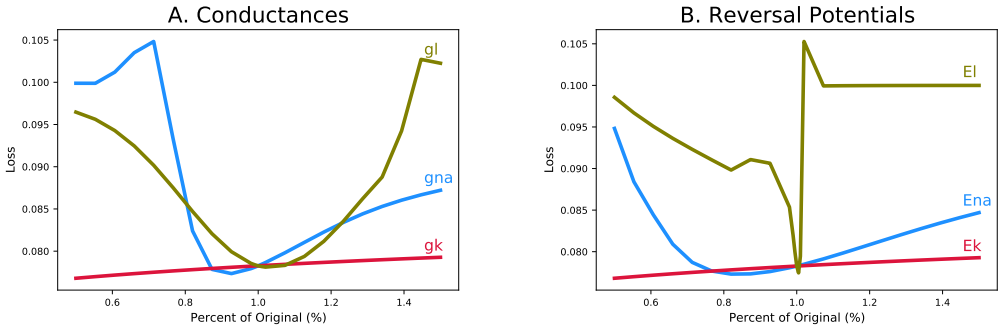

Figure 5: Evaluation of loss landscape of HH model after physiological variation. Individual biophysical parameters of each neuron in the network are perturbed by a percentage of their original value and the loss is evaluated on an MNIST validation batch. A shows loss landscape varying conductances in the HH model. B shows variations in the reversal potentials. See supplement for additional similar runs.

Real biological systems need to function accurately despite changing environmental conditions. For example, the extracellular concentration of ions could change which would affect the reversal potential in the model. Additionally, the conductances, which represent the maximum rate at which ions can flow through the membrane, depend on the total number of channels in a neuron, which is in constant flux. To study this, we computed the loss curves around variation of biological parameters by +-50%, shown in Figure 5. We found that for most of the biological parameters, the loss basins were relatively smooth and convex, implying robustness with respect to those biological parameters.

We however note the special case of E_l in the plot above. We can see that small changes to E_l have a massive effect on the loss. This makes sense under careful consideration. Indeed, the term E_l controls the resting potential of the Hodgkin-Huxley neuron. So, if we make it even slightly larger without changing the other parameters, this can drive the neuron into continuous spiking. On the other hand, making E_l smaller shifts the resting potential down and requires larger inputs to stimulate the neuron to spike. In this sense, we can think of the E_l constant as the "primary" reversal potential or bias term of the Hodgkin-Huxley neuron.

3.6 Learning in different models

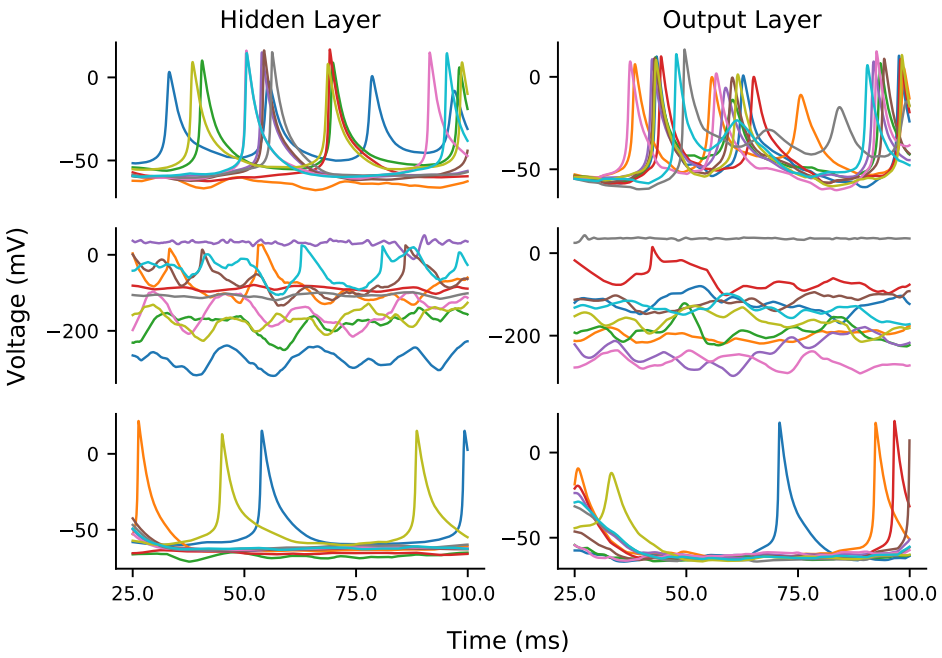

Figure 6: Representative voltage traces for three different trained models with distinctive parameters, as outlined below. The left column shows traces for every 10th neuron in the hidden layer and the right column shows all 10 output neurons' traces. The top plots are for the model with an applied current that causes more firing. The center plots are for a model where firing is suppressed.

While our model is largely robust to individual biological parameter changes, we note that some parameter changes can significantly change the dynamics of the model. We then tested learning in two additional models, one trained with a constant bias input current of 5 $\mu A/cm^2$ which puts it near the bistable regime (Figure 6, top) and one where a parameter was changed to prevent repetitive firing as seen in the original system Hodgkin and Huxley modeled (Figure 6 middle) [15]. This was contrasted with the original model we fit (Figure 6 bottom). This shows cases where spiking was enhanced (Figure 6 top), suppressed (Figure 6 middle), or our original parameters (Figure 6 bottom). All three models were able to achieve at least 80% accuracy (See Supplemental Figure 1), however, the model with the increased applied current learned slower than the other models. This is likely because the added current caused action potential to be more spontaneous and less dependent on the input to a neuron. We also note that the case where the neuron could not repetitively spike caused an encoding based not on spikes but the rest membrane potential (RMP, similar to average voltage) of the neuron. In this case, the RMP achieved values not typically seen in neurons.

We next explored the gradients, as we did in Figure 4, but with the new models (See Supplemental Figures 5 and 6). In general, the gradients for these two models were much

less than in the original model. In the case of the greater applied current, we see that when output neuron "0" fires, gradients backpropagate to all neurons that fired just before it in the hidden layer and input layer. The number of weights that actually change again is small. However, a more interesting behavior can be seen in the case of the modified Hodgkin-Huxley model used in Figure 6 (center). Output neuron "5" does not show the normal spiking behavior that other neurons do. In this case, the gradients do not show a Hebbian-type pattern. This shows how our Hebbian Learning behavior depends on the details of the biophysics of the model. Thus, as spiking behaviors are changed, different learning rules result.

4 Discussion

We present a novel method for training networks composed of neurons with ionic dynamics. Our approach differs from previous work in deep learning by using model neurons that exhibit far more complex dynamics and adds to the growing body of neuroscience literature investigating how biologically realistic neurons might be trained. Furthermore, we incorporate past-facing gradients that are hypothesized to capture more relative timing information between spikes in the network, as opposed to methods based purely on rate coding. In contrast with previous work on training SNNs, we provide more biological realism while requiring fewer "hacks" (such as surrogate gradient methods) to avoid the problem of non-differentiable voltage traces. Future work will allow for more investigation of biological plausibility and the connection to STDP-like rules, as well as into extending SGD to recurrent biological networks such as those occurring in nature. Additionally, our method may be applied to the training of neuron models implemented in neuromorphic hardware.

Machine learning is becoming an essential part of Neuroscience. Connection weights discovered by our method (Figure 3) might prove enlightening for both conventional ML problems and in the myriad uses of ML in Neuroscience, for example, in understanding the processing of visual images. The complex and diverse behavior unique to biologically realistic neuron models with action potential dynamics can be directly compared to recorded activity patterns from *in vivo* or *in vitro* neurons and can provide insight into how biological neurons can learn a vast number of tasks and behaviors.

Machine learning is becoming an essential part of neuroscience, and neuroscience has greatly inspired machine learning. Our work shows how mathematical principles from Machine learning can be used to train networks of model neurons that match the dynamics of biological neurons. When used to study specific biological problems, the parameters of our model will make predictions that can be tested in the lab. While we do not account for some aspects of neurophysiology (e.g., spatial aspects of neuronal processing), we show that there are many dynamics not captured by current methods in machine learning. As our biological models can learn quicker than standard networks used in ML in the MNIST problem. How this translates to other ML challenges presents an important area for future research to improve ML methods

Mathematically, there are some differences between our version of learning and Hebbian learning. For example, Hebbian learning considers the time just before and after a postsynaptic spike, whereas we consider the effect of a weight over a long time period on the loss function over a short period. The relationship between our gradients and optimal signals in ODEs needs to be further explored, perhaps in relation to the Poyntragin Maximum Principle [17]. Moreover, Hebbian learning only applies to pairs of neurons, whereas we see

some behaviors that backpropagate through the whole network. That backpropagation is somewhat rare as the number of times neurons at multiple levels fire together is rare, so practically focusing just on the effects between two connecting neurons might also offer an effective strategy.

Our work shows that applying backpropagation to biological neurons yields quick and sparse learning similar to that seen in Hebbian learning. More work needs to explore learning in biological networks to see how closely our predictions match learning in biological datasets. Hebbian learning is efficient on biological neurons, whereas backpropagation in DNNs is efficient on modern computer architectures. Careful future consideration of this work could help us better understand if evolution has developed learning mechanisms that are mathematically efficient, and to develop new methods and computer architectures that train BNNs.

From this work, we see that Hebbian learning can be interpreted as using a mathematically optimal learning procedure on a model built on ionic dynamics. Further work should also consider networks that are not fully connected, how to fit initial conditions and model parameters, additional structures (e.g., lateral inhibition, different types of synaptic dynamics), gradients that might be clipped biological, and this might help with learning, recurrent networks as well as datasets beyond MNIST.

4.1 Future Work

We hope that this work has opened up several avenues for further research. In particular, our work explores only networks in which all neurons are identical in their physiological parameters. In many biological neuronal networks, different neurons in the same network have very different firing regimes, roles, and behaviors [17]. Our work lays foundation for easily exploring such networks.

Additionally, we limited our network structures to only fully connected layers with a single hidden layer and only applied to the MNIST benchmark for simplicity. However, our method trivially scales up to other architectures and this is supported in our code. Indeed, adding layers has little effect on the overall time-complexity and number of neurons is more of a limiting factor. One could also vary the initial conditions of the model.

Applying to more directly applicable benchmarks (e.g. those using spiking recordings from real brain regions) and using more complex architectures such as encoder-decoder network or convolutional networks for latent space analysis and image processing, respectively, is a clear next step for our method. Using convolutions could make our method more efficient due to the reduced dimensionality of each layer's weights.

One final avenue of future research is applying our method to networks of recurrent biological neurons such as those occurring experimentally in nature. Being able to efficiently fit models of such regions clearly has many applications to computational neuroscience, specifically large-scale brain modelling.

5 Acknowledgements

This work was supported by NSF Grant 1714094 and HFSP RGP 0019/2018.

6 Data and Code Availability

No experimental data were used in this study.
The code for this study can be found at `https://github.com/knc-neural-calculus/HH-neurons-learning`.

References

[1] LeCun, Y., Bengio, Y. & Hinton, G. Deep learning. *Nature* **521**, 436–444 (2015).

[2] Pfeiffer, M. & Pfeil, T. Deep Learning With Spiking Neurons: Opportunities and Challenges. *Frontiers in Neuroscience* **12** (2018).

[3] Forger, D. B., Paydarfar, D. & Clay, J. R. Optimal Stimulus Shapes for Neuronal Excitation. *PLOS Computational Biology* **7**, e1002089 (2011).

[4] Tavanaei, A. & S., A. A Minimal Spiking Neural Network to Rapidly Train and Classify Handwritten Digits in Binary and 10-Digit Tasks. *International Journal of Advanced Research in Artificial Intelligence* **4** (2015).

[5] TL, Zhang, *et al.* A Plasticity-centric Approach to Train the Non-differential Spiking Neural Networks (2018). Preprint at `http://ir.ia.ac.cn/handle/173211/22081`.

[6] Kulkarni, S. R. & Rajendran, B. Spiking neural networks for handwritten digit recognition—Supervised learning and network optimization. *Neural Networks* **103**, 118–127 (2018).

[7] Rosenblatt, F. The Perceptron: A Probabilistic Model For Information Storage and Organization in the Brain *Psychological Review* **65**, 386–408 (1958).

[8] Tavanaei, A., Ghodrati, M., Kheradpisheh, S. R., Masquelier, T. & Maida, A. Deep learning in spiking neural networks. *Neural Networks* **111**, 47–63 (2019).

[9] Wade, J. J., McDaid, L. J., Santos, J. A. & Sayers, H. M. SWAT: A Spiking Neural Network Training Algorithm for Classification Problems. *IEEE Transactions on Neural Networks* **21**, 1817–1830 (2010).

[10] Bellec, G. *et al.* A solution to the learning dilemma for recurrent networks of spiking neurons *Nat Commun* **11**, 3625 (2020).

[11] Huh, D. & Sejnowski, T. J. Gradient Descent for Spiking Neural Networks. In Bengio, S. *et al.* (eds.) *Advances in Neural Information Processing Systems 31*, 1433–1443 2018).

[12] Lee, J. H., Delbruck, T. & Pfeiffer, M. Training Deep Spiking Neural Networks Using Backpropagation. *Frontiers in Neuroscience* **10** (2016).

[13] Hodgkin, A. L. & Huxley, A. F. A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of Physiology* **117**, 500–544 (1952).

[14] Deng, Li, The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine* **29**, 141–142 (2012).

[15] Clay, J. R., Paydarfar, D. & Forger, D. B. A simple modification of the Hodgkin and Huxley equations explains type 3 excitability in squid giant axons. *Journal of the Royal Society Interface* **5**, 1421–1428 (2008).

[16] Ermentrout, B. Terman, D. H. *Mathematical Foundations of Neuroscience* (Springer, 2010).

[17] Forger, D.B. *Biological Clocks, Rhythms, and Oscillations* (The MIT Press, Cambridge, MA, 2017).

[18] Belle, M., et al. Daily electrical silencing in the mammalian circadian clock. *Science* (2009).

[19] Izhikevich, E., Which Model to Use for Cortical Spiking Neurons?. *IEEE TRANSACTIONS ON NEURAL NETWORKS,* (2004).

[20] Stimberg, M., Goodman, D., Nowotny, T. Brian2GeNN: accelerating spiking neural network simulations with graphics hardware. *Nature* **Sci Rep 10**, (2020).

[21] *Multiplexing Visual Signals in the Suprachiasmatic Nuclei* (Cell Reports, 2017).