A APPENDIX

A.1 FULL DERIVATIVES OF THE BP ALGORITHM

We start from backpropagating the error signal from the PSC - a to the spike train - s. A causal impulse response only has value when $t > 0$ as the ϵ in (6). So the partial derivative of a neuron's output spike train on time t is the integration on the derivative of all future error with $t' > t$, which can be expressed by:

$$\frac{\partial L}{\partial s_i^{(l)}(t)} = \int_{t'=t}^{T} \frac{\partial L}{\partial a_i^{(l)}(t')} \frac{\partial a_i^{(l)}(t')}{\partial s_i^{(l)}(t)} dt' = \int_{t'=t}^{T} g_i^{(l)}(t')\epsilon(t'-t)dt' = \left(g_i^{(l)} * \tilde{\epsilon}\right)(t), \tag{23}$$

where T is the length of a simulation time window, and $\tilde{\epsilon}(t) = \epsilon(-t)$ is the time-reverse impulse response kernel of the synapse.

In Figure 4, we draw the full dependency in both (a) - forward and (b) - backward propagation under infinitesimal discrete time steps dt. Under which, the gradient on u can be described by:

$$\frac{\partial L}{\partial u_i^{(l)}(t)} = \frac{\partial L}{\partial s_i^{(l)}(t)} \frac{\partial s_i^{(l)}(t)}{\partial u_i^{(l)}(t)} + \frac{\partial L}{\partial u_i^{(l)}(t+dt)} \left(\frac{\partial u_i^{(l)}(t+dt)}{\partial s_i^{(l)}(t)} \frac{\partial s_i^{(l)}(t)}{\partial u_i^{(l)}(t)} + \frac{\partial u_i^{(l)}(t+dt)}{\partial u_i^{(l)}(t)} \right), \tag{24}$$

which does not have a closed form solution for two reasons:

• The derivative $\partial s_i^{(l)}(t)/\partial u_i^{(l)}(t)$ is zero when $u_i^{(l)}(t) \neq \vartheta$, and is ∞ when $u_i^{(l)}(t) = \vartheta$, which is ill defined.

• The complex temporal dependency of $u_i^{(l)}(t)$ and $u_i^{(l)}(t+dt)$ brings difficulty for calculation.

Following previous works (# cite): by introducing surrogate gradient as a substitution for $\partial s_i^{(l)}(t)/\partial u_i^{(l)}(t)$, we approx the first term in (24) as:

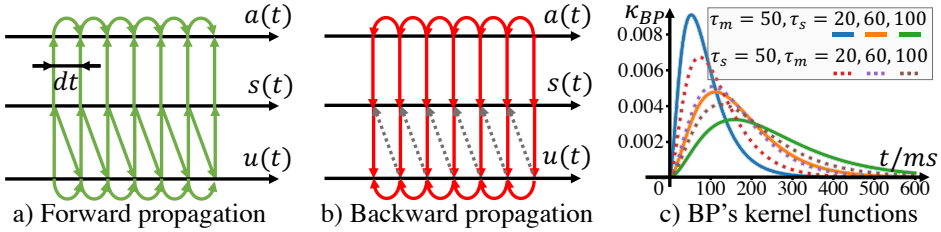
$$\frac{\partial L}{\partial s_i^{(l)}(t)} \frac{\partial s_i^{(l)}(t)}{\partial u_i^{(l)}(t)} \approx \frac{\partial L}{\partial s_i^{(l)}(t)} \sigma'(u_i^{(l)}(t)). \tag{25}$$

The other term in (24) describes the temporal dependency of the u. Define a membrane potential's impulse response kernel function $\zeta(t)$ as: $\zeta(t) = \frac{1}{\tau_m} e^{(-t/\tau_m)} H(t)$

By ignoring part of the dependency, as shown by the gray colored dash lines in Figure 4 (b), and applying the same trick as in (23), we have:

$$\begin{aligned} \frac{\partial L}{\partial u_i^{(l)}(t)} &\approx \int_{t'=t}^{T} \frac{\partial L}{\partial s_i^{(l)}(t')} \frac{\partial s_i^{(l)}(t')}{\partial u_i^{(l)}(t')} \frac{\partial u_i^{(l)}(t')}{\partial u_i^{(l)}(t)} dt \approx \int_{t'=t}^{T} \frac{\partial L}{\partial s_i^{(l)}(t')} \sigma'(u_i^{(l)})(t')\zeta(t'-t) \\ &= \left[\left((g_i^{(l)} * \tilde{\epsilon}) \cdot \sigma'(u_i^{(l)}) \right) * \tilde{\zeta} \right](t), \end{aligned} \tag{26}$$

where $\tilde{\zeta}(t) = \zeta(-t)$ is the time-reverse impulse response kernel of the membrane potential.



a) Forward propagation b) Backward propagation c) BP's kernel functions

Figure 4: (a) Full forward dependency between u, s, and a in the discrete simulation. (b) Full backward dependency in the discrete simulation. The dashed lines are usually been omitted when doing backpropagation. (c) BP's kernel functions with different time constants

Then the final step is to propagate the gradient from $u_i^{(l)}$ to $w_{ij}^{(l)}$ according to (1):

$$\begin{aligned}\frac{\partial L}{\partial w_{ij}^{(l)}} &= \int \frac{\partial L}{\partial u_i^{(l)}(t)} \frac{\partial u_i^{(l)}(t)}{\partial w_{ij}^{(l)}} dt \approx \int \left[\left((g_i^{(l)} * \tilde{\epsilon}) \cdot \sigma'(u_i^{(l)})\right) * \tilde{\zeta}\right](t) \cdot a_j^{(l-1)}(t) dt \\ &\approx \int \left((g_i^{(l)} \cdot \sigma'(u_i^{(l)})) * \tilde{\epsilon} * \tilde{\zeta}\right)(t) \cdot a_j^{(l-1)}(t) dt \\ &= \int \left(g_i^{(l)} \cdot \sigma'(u_i^{(l)})\right)(t) \cdot \left(a_j^{(l-1)} * \epsilon * \zeta\right)(t) dt \\ &= \int e_i^{(l)}(t) \cdot \left(s_j^{(l-1)} * \kappa_{\mathrm{BP}}\right)(t) dt,\end{aligned} \tag{27}$$

where $e_i^{(l)}(t) = (g_i^{(l)} \cdot \sigma'(u_i^{(l)}))(t)$, and $\kappa_{\mathrm{BP}}(t) = (\epsilon * \epsilon * \zeta)(t)$. We show the shape of κ_{BP}in Figure 4 (c), and calculate its close form expression as:

$$\kappa_{\mathrm{BP}}(t) = \left[\frac{\tau_m \cdot (e^{-\frac{t}{\tau_m}} - e^{-\frac{t}{\tau_s}})}{(\tau_m - \tau_s)^2} - \frac{t \cdot e^{-\frac{t}{\tau_s}}}{\tau_s(\tau_m - \tau_s)}\right] \cdot H(t). \tag{28}$$

We then provide the rules to further propagate the gradient to previous layers (take layer $(l-1)$ as an example). Following (1), we have:

$$g_j^{(l-1)} = \frac{\partial L}{\partial a_j^{(l-1)}(t)} = \sum_{i=1}^{N^{(l)}} w_{ij}^{(l)} \frac{\partial L}{\partial u_i^{(l)}(t)} = \sum_{i=1}^{N^{(l)}} w_{ij}^{(l)} \left[\left((g_i^{(l)} * \tilde{\epsilon}) \cdot \sigma'(u_i^{(l)})\right) * \tilde{\zeta}\right](t) \tag{29}$$

BP in hidden layers also following the similar steps: from $g_j^{(l-1)}$ to $\partial L / \partial u_j^{(l-1)}$ following (26), and further propagating to weights following (27).

A.2 Detailed Experimental Setups

A.2.1 Universal Spike Train Approximator

This section concludes all settings used in the 2-layers SNN experiment. All parameters used are summarized in Table 2.

Table 2: Parameters of the 2-layers network.

N_T	τ_m	τ_s	#inputs	#hidden	p_{in}	A^+
500ms	50ms	20ms	50	100	0.05	0.00004
τ_+	A^-	v_{rest}	ϑ	#iters	scale	bias
30ms	0	0	1	5000	0.3	0.01

The total simulation time N_t is 500ms. τ_m and τ_s are the time constants in (1) and (4). The total number of randomly generated inputs is 50. The number of hidden pyramidal cells is 100.

We sample the total 500ms by 1ms when doing this experiment, which means there are 500 time steps totally. As shown in Figure 5, the randomly generated input currents are produced by a two-steps process.

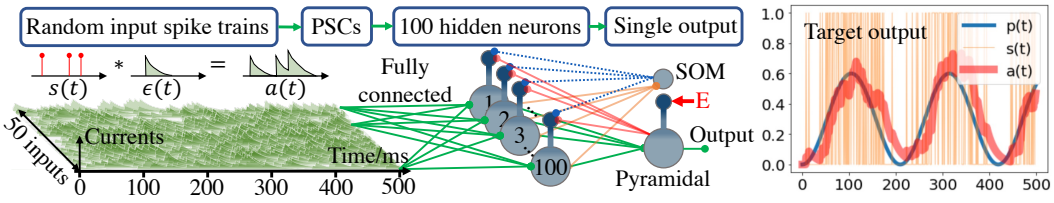


Figure 5: Visualization of all 50 input currents, the network architecture, and the process to generate the target output current

The first step is to generate spike trains $s(t)$ following Bernoulli distribution on each time step with probability $p_{in} = 0.05$. Then the input currents are defined using F-type synapses $a(t) = s(t) * \epsilon$ as in (6).

A^+, τ_+, and A^- are parameters of our STDP function. v_{rest} and ϑ are fixed to 0 and 1 in all experiment of our work. All weights are initialized by a scaled Gaussian distribution $w \sim (\text{scale} \times N(0, 1) + \text{bias})$.

The output target signal is defined by a sinusoidal probability function $p_{\text{target}}(t) = 0.3 + -0.3cos(0.03t)$, and $p_{\text{target}}(t)$ is converted to spike trains $s_{\text{target}}(t)$ following the Bernoulli distribution on each time step, further, $s_{\text{target}}(t)$ is converted to continuous currents following $a(t) = s(t) * \epsilon$. The reason we do such conversion is to guarantee that the single output neuron is possible to fit the target output signal perfectly.

A.2.2 MNIST

The experiment runs on a single RTX-3090 GPU. The MNIST dataset (LeCun, 1998) has 60,000 training images and 10,000 testing images. The training parameters are: batch size = 64, number of training epochs = 200, and learning rate = 0.0005 for the adopted AdamW optimizer (Loshchilov & Hutter, 2017). The images were converted to continuous-valued multi-channel currents. Moreover, data augmentations including RandomCrop and RandomRotation were applied to improve performance (Shorten & Khoshgoftaar, 2019).

A.2.3 CIFAR10

The experiment runs on a single RTX-3090 GPU. The CIFAR10 dataset (Krizhevsky et al., 2009) has 50,000 training images and 10,000 test images. We trained our SNN using NA for 1200 epochs with a batch size of 50 and a learning rate 0.0005 for the AdamW optimizer (Loshchilov & Hutter, 2017). The same input image coding strategy as the MNIST dataset was adopted. Moreover, data augmentations including RandomCrop, ColorJitter, and RandomHorizontalFlip (Shorten & Khoshgoftaar, 2019) were applied. The convolutional layers were initialized using the kaiming uniform initializer (He et al., 2015), and the linear layers were initialized using the kaiming normal initializer (He et al., 2015).

A.3 ARCHITECTURE ILLUSTRATION

We provide more figures to illustrate our proposed microcircuit architectures. Figure 6 (a) is another example of our spiking neural networks' architecture: Two types of cells, Pyramidal cells and Somatostatin (SOM) cells, are needed to realize the forward propagation and the local synaptic plasticity. This example three layers fully-connected (FC) neural network has 2-3-2 pyramidal cells in each layer. The input current signals are from four photoreceptor cells in this example simulating a vision related learning task.

(b) to (d) are the disassembled explanation of all synaptic connections. Each neuron has been indexed by its layer and the footnoted position in its layer. The pyramidal cells and SOM cells are one-on-one in each layer except for the 1^{st} layer, where no SOM cells are needed.

• (b) The connections in this sub-Figure (green colored solid arrows) are all feed-forward connections as the same as the weight connections in the more conventional non-spiking artificial neural networks (ANNs).

• (c) The output currents of the pyramidal cells in a layer are connected to the SOM cells in the next layer (orange colored solid arrows). The SOM cells use these signals to predict the firing activity of pyramidal cells in the next layer, so we name these connections as predict connections, and use the footnote p to represent all of the predict-related parameters. The adjustment of these predict connections requires all SOM cells to receive the one-to-one teaching current signal from each SOM cell's corresponding pyramidal cell (purple colored dashed arrows).

• (d) The top-down feed backward signals (red colored solid arrows), carrying the superposition of output current from the next layer and error information, are connected to the apical dendrites of previous layer's pyramidal cells. The next layer's output current signals should be canceled out by the local predicting signals that the next layer's SOM cells provided (blue colored dashed arrows),

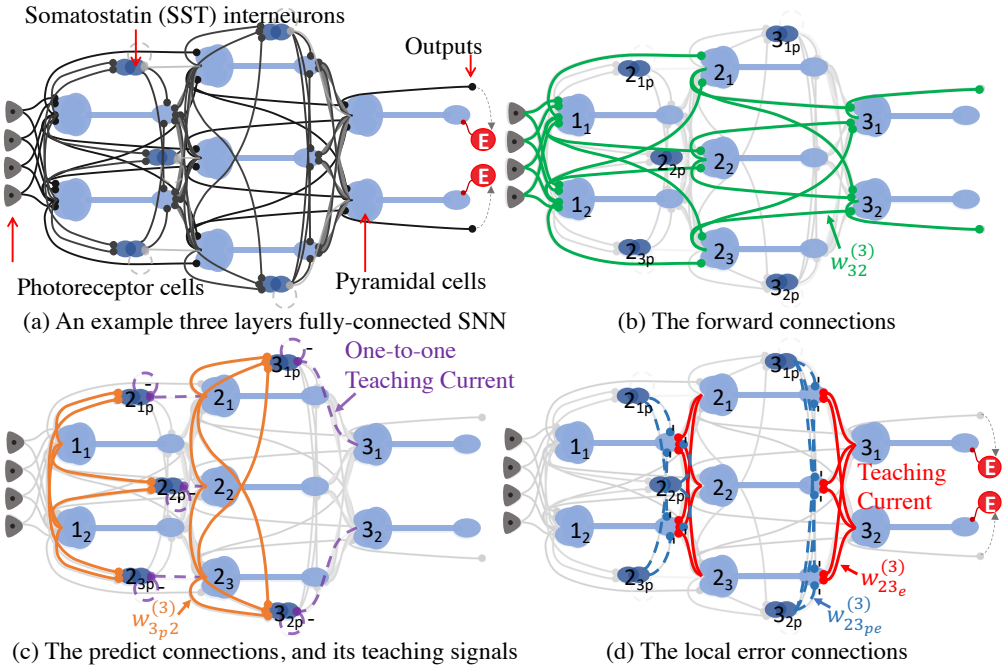


Figure 6: Our proposed Microcircuit architecture.

which will leave only error signals on the pyramidal cells' apical dendrites. Since these two types of connections together generate pyramidal cells' *error* information, we use the footnote e to mark all the error related variables. Importantly, the output layer's pyramidal cells needs additional error signals $e_i = a_i^{\text{target}} - a_i$ connected to its apical dendrites. Such error signals may come from higher to lower brain areas (Leinweber et al., 2017).

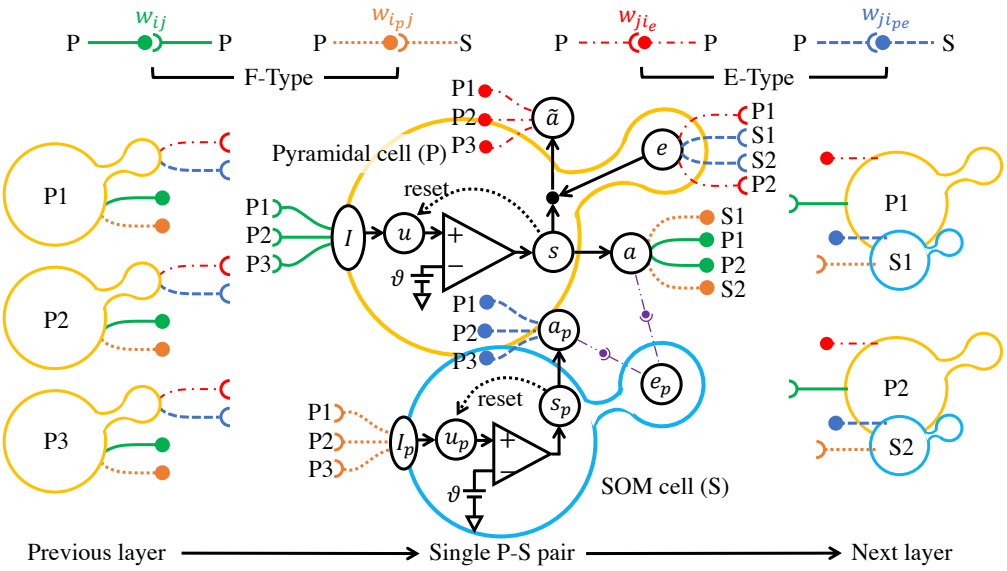


Figure 7: Our proposed Microcircuit architecture.

Figure 7 zoomed into one pair of Pyramidal-SOM cells, which has three input pyramidal cells from its previous layer, and forward connected to two pairs of Pyramidal-SOM cells in its next layer. Cells in a same layer are indexed, which is used to indicate synapses connections.