Adjoint propagation of error signal through modular recurrent neural networks for bio-plausible learning

Zhuo Liu[1], Xu Meng[1], Yousheng Wang[1], Hao Shu[1], Linmiao Wang[1], Wei Wang[2], Tao Chen[1,*]

[1] *School of Microelectronics, University of Science and Technology of China, Hefei 230026, Anhui, China*

[2] *Digital Intelligence Centre, Shenzhen Power Supply Bureau, China Southern Power Grid, Shenzhen 518000, Guangdong, China*

[*]Correspondence to: tchen@ustc.edu.cn

A General Formulation

Here, we describe the detailed process of inference and learning of the MR-RNN model within our adjoint propagation framework. Each RNN module in our AP framework can be treated as an independent dynamical system with states $u_{\mathrm{L}(i)}$, internal weights $W_{\mathrm{L}(i)}$, bias $b_{\mathrm{L}(i)}$, nonlinear activation function $f_{\mathrm{L}(i)}$, inference signals from the $(i-1)$-th Layer $W_{\mathrm{L}(i-1,i)}u^{\beta,t}_{\mathrm{L}(i-1),\mathrm{SI}}$ and error feedback from the $(i+1)$-th Layer $E_{\mathrm{L}(i+1,i)}e_{\mathrm{L}(i+1),\mathrm{SE}}$. Its dynamics can be described as:

$$\begin{aligned} u^{\beta,t+1}_{\mathrm{L}(i)} =& W_{\mathrm{L}(i)} \cdot f_{\mathrm{L}(i)}(u^{\beta,t}_{\mathrm{L}(i)}) + b_{\mathrm{L}(i)} + \\ & W_{\mathrm{L}(i-1,i)} \cdot f_{\mathrm{L}(i-1)}(u^{\beta,t_e}_{\mathrm{L}(i-1),\mathrm{SI}}) - \beta E_{\mathrm{L}(i+1,i)} \cdot e_{\mathrm{L}(i+1),\mathrm{SE}} \end{aligned} \tag{S1}$$

where $\beta = 0$ in the inference phase and $\beta = 1$ in the learning phase. The i-th layer converges to a stable point $u^{\beta,t_e}_{\mathrm{L}(i)}$ in sufficient iteration steps t_e. Note that the subscript $\cdot_{\mathrm{L}(i),\mathrm{SE}}$ means the only the state of the SE neurons are non-zero for unified formulation. Given a pair of input and target (x, y), the network responds to the input $u^{\beta,t_e}_{\mathrm{L}(0),\mathrm{SI}} = u_{\mathrm{L}(0)} = x$ and generates the prediction $u_{\mathrm{L}(3)}$ in the inference phase. The dynamics of each layer of RNN is described by:

$$u^{0,t+1}_{\mathrm{L}(i)} = W_{\mathrm{L}(i)} \cdot f_{\mathrm{L}(i)}(u^{0,t}_{\mathrm{L}(i)}) + b_{\mathrm{L}(i)} + W_{\mathrm{L}(i-1,i)} \cdot f_{\mathrm{L}(i-1)}(u^{0,t_e}_{\mathrm{L}(i-1),\mathrm{SI}}) \tag{S2}$$

The i-th layer converges to a stable point $u^{0,t_e}_{\mathrm{L}(i)}$ with sufficient iterations. Then it passes the inference signals at its SI neurons to the RI neurons of the next layer. This process goes on and on until the signal reaches the output $u_{\mathrm{L}(3)}$:

$$u_{\mathrm{L}(3)} = W_{\mathrm{L}(2,3)} \cdot f_{\mathrm{L}(2)}(u^{0,t}_{\mathrm{L}(2),\mathrm{SI}}) + b_{\mathrm{L}(3)} \tag{S3}$$

where $f_{\mathrm{L}(3)}$ is the nonlinear activation function of the output layer. The error at the output layer is $e_{\mathrm{L}(3)} = f_{\mathrm{L}(3)}(u_{\mathrm{L}(3)}) - y$, which is fed to the L(2)'s RE neurons. Then the network enters the learning phase. In this phase, the dynamics is governed by:

$$\begin{aligned} u^{1,t+1}_{\mathrm{L}(i)} =& W_{\mathrm{L}(i)} \cdot f_{\mathrm{L}(i)}(u^{1,t}_{\mathrm{L}(i)}) + b_{\mathrm{L}(i)} + \\ & W_{\mathrm{L}(i-1,i)} \cdot f_{\mathrm{L}(i-1)}(u^{0,t_e}_{\mathrm{L}(i-1),\mathrm{SI}}) - E_{\mathrm{L}(i+1,i)} \cdot e_{\mathrm{L}(i+1),\mathrm{SE}} \end{aligned} \tag{S4}$$

Again, with sufficient iterations, L(2) converges to stable points $u^{1,t_e}_{\mathrm{L}(i)}$. The error of this layer is $e_{\mathrm{L}(i)} = f(u^{0,t_e}_{\mathrm{L}(i)}) - f(u^{1,t_e}_{\mathrm{L}(i)})$, which teaches the forward weights and relays the error to lower index layer. It is worth pointing out that the initial condition can be the neural states at the end of the inference phase (EPE error in the main text) or at the beginning of the inference phase (STE error). The order of layer-by-layer updating runs from lower index RNN to higher index RNN in the inference phase and reversed in the learning phase. A whole learning framework of two RNN modules without bias $b_{\mathrm{L}(i)}$ is given in Algorithm 1.

For the experiments illustrated in Fig. 5 in the main text, their process can be described by the pseudocode in Algorithm 2. Note that the RNN modules that are irrelevant in the the inference phase are set to zero-state (e.g. L(3) and L(4) for training on FMNIST) . In the learning phase, the error-perturbation is directly added to zero-state.

Algorithm 1: AP model without biases for computing

```
Data: A batch of input and target $(x, y)$,
      $\theta = (W_{L(0,1)}, W_{L(1,2)}, W_{L(2,3)}, W_{L(1)}, W_{L(2)}, E_{L(2,1)}, E_{L(3,2)})$
Result: $\theta$
1  Function Inference-phase($\theta, x$):
2      $u^{0,t_e}_{L(0),SI} = u_{L(0)} = x$;
3      for $i \leftarrow 1$ to $2$ do
4          for $t \leftarrow 0$ to $t_e - 1$ do
5              $u^{0,t+1}_{L(i)} \leftarrow W_{L(i)} \cdot f_{L(i)}(u^{0,t}_{L(i)}) + W_{L(i-1,i)} \cdot f_{L(i-1)}(u^{0,t_e}_{L(i-1),SI})$;
6          end
7      end
8      $u^{0,t_e}_3 = u_{L(3)} = W_{L(2,3)} f_{L(2)}(u^{0,t}_{L(2),SI})$;
9      $\Lambda_1 = \{u^{0,t_e}_{L(i)}\}, i = 1, 2, 3$;
10     return $\Lambda_1$;
11 Function Learning-phase($\theta, \Lambda_1, t$):
12     $e_{L(3)} = f_{L(3)}(u_{L(3)}) - y$;
13     for $i \leftarrow 2$ to $1$ do
14         for $t \leftarrow 0$ to $t_e - 1$ do
15             $u^{1,t+1}_{L(i)} \leftarrow W_{L(i)} \cdot f_{L(i)}(u^{1,t}_{L(i)}) + W_{L(i-1,i)} \cdot f_{L(i-1)}(u^{0,t_e}_{L(i-1),SI}) - E_{L(i+1,i)} \cdot e_{L(i+1),SE}$;
16         end
17         $e_{L(i)} = f_{L(i)}(u^{0,t_e}_{L(i)}) - f_{L(i)}(u^{1,t_e}_{L(i)})$;
18     end
19     $\Lambda_2 = \{e_{L(i)}\}, i = 1, 2, 3$;
20     return $\Lambda_2$;
21 Function Updating Weights($\theta, \Lambda_1, \Lambda_2$):
22     $\Delta W_{L(2,3)} \leftarrow -e_{L(3)} \cdot f_{L(2)}(u^{0,t_e}_{L(2),SI})^T$;
23     $\Delta W_{L(1,2)} \leftarrow -e_{L(2),RI} \cdot f_{L(1)}(u^{0,t_e}_{L(1),SI})^T$;
24     $\Delta W_{L(0,1)} \leftarrow -e_{L(1),RI} \cdot f_{L(0)}(u_{L(0)})^T$;
```

Algorithm 2: AP model in different configurations

```
Data: Two datasets $(x_1, y_1), (x_2, y_2)$, their AP configuration and connections
      parameters $(G_1, \theta_1)$, $(G_2, \theta_2)$, the number of training epochs for both tasks $n_{epoch}$
Result: $\theta_1$,$\theta_2$
1  Function Training-for-multi-task($\{(x_1, y_1), (G_1, \theta_1)\}$, $\{(x_2, y_2), (G_2, \theta_2)\}$):
2      for $epoch \leftarrow 1$ to $n_{epoch}$ do
3          Configuring network with $G_1$;
4          Inference-phase with $x_1$;
5          Training-phase with $(x_1, y_1)$;
6          updating parameters $\theta_1$;
7          Configuring network with $G_2$
8          Inference-phase with $x_2$;
9          Training-phase with $(x_2, y_2)$;
10         updating parameters $\theta_2$;
11     end
12     return $\theta_1$,$\theta_2$;
```

B Derivation of our model updates

We can derive the weight update rule following the principle of discrepancy reduction [1, 2], namely, reducing the discrepancy between neural states in the inference phase and learning phase. The argument is that when the network is trained, the error should be small, and discrepancy between the two phases are minimized. We define the total loss as:

$$\mathrm{Loss}(\theta) = \sum_{i=1}^{L} k_i \mathcal{L}_i(u^{0,t_e}_{\mathrm{L}(i)}, u^{\beta,t_e}_{\mathrm{L}(i)}) = \sum_{l=1}^{L} k_i (\|f_{\mathrm{L}(i)}(u^{\beta,t_e}_{\mathrm{L}(i)}) - f_{\mathrm{L}(i)}(u^{0,t_e}_{\mathrm{L}(i)})\|_p)^q \tag{S5}$$

Where $p = q = 2$, i.e. a square of Euclidean norm. k_i is a scalar that weights the contribution of a specific local loss to the total loss. Here $k_i = 1/2$. The error of each layer can be defined as the partial derivative of the loss to the activation:

$$\begin{aligned} e_{\mathrm{L}(i)} = \frac{\partial \mathrm{Loss}(\theta)}{\partial f_{\mathrm{L}(i)}(u^{0,t_e}_{\mathrm{L}(i)})} &= \frac{\partial \sum_{l=1}^{L} k_i (\|f_{\mathrm{L}(i)}(u^{1,t_e}_{\mathrm{L}(i)}) - f_{\mathrm{L}(i)}(u^{0,t_e}_{\mathrm{L}(i)})\|_p)^q}{\partial f_{\mathrm{L}(i)}(u^{0,t_e}_{\mathrm{L}(i)})} \\ &= \frac{\partial (\|f_{\mathrm{L}(i)}(u^{1,t_e}_{\mathrm{L}(i)}) - f_{\mathrm{L}(i)}(u^{0,t_e}_{\mathrm{L}(i)})\|_2)^2}{2\partial f_{\mathrm{L}(i)}(u^{0,t_e}_{\mathrm{L}(i)})} \\ &= f_{\mathrm{L}(i)}(u^{0,t_e}_{\mathrm{L}(i)}) - f_{\mathrm{L}(i)}(u^{1,t_e}_{\mathrm{L}(i)}) \end{aligned} \tag{S6}$$

Further, we can deduce the weight update rule from the gradients of the loss with respect to the weights:

$$\begin{aligned} \Delta W_{\mathrm{L}(i-1,i)} &= -\frac{\partial \mathrm{Loss}(\theta)}{\partial W_{\mathrm{L}(i-1,i)}} \\ &= -\frac{\partial \mathrm{Loss}(\theta)}{\partial f_{\mathrm{L}(i)}(u^{0,t_e}_{\mathrm{L}(i)})} \frac{\partial f_{\mathrm{L}(i)}(u^{0,t_e}_{\mathrm{L}(i)})}{\partial u^{0,t_e}_{\mathrm{L}(i)}} \Big(\frac{\partial W_{\mathrm{L}(i-1,i)} \cdot f_{\mathrm{L}(i-1)}(u^{0,t_e}_{\mathrm{L}(i-1),\mathrm{SI}})}{\partial W_{\mathrm{L}(i-1,i)}} \\ &\quad + \frac{\partial [W_{\mathrm{L}(i)} \cdot f_{\mathrm{L}(i)}(u^{0,t_e-1}_{\mathrm{L}(i)})]}{\partial W_{\mathrm{L}(i-1,i)}} \Big) \end{aligned} \tag{S7}$$

Since the spectral radius of $W_{\mathrm{L}(i)}$ is mall, we assume that the second term in the bracket can be omitted, and rewrite the forward weight update rule in the following form:

$$\Delta W_{\mathrm{L}(i-1,i)} \approx -\frac{\partial \mathrm{Loss}(\theta)}{\partial f_{\mathrm{L}(i)}(u^{0,t_e}_{\mathrm{L}(i)})} \frac{\partial f_{\mathrm{L}(i)}(u^{0,t_e}_{\mathrm{L}(i)})}{\partial u^{0,t_e}_{\mathrm{L}(i)}} \frac{\partial W_{\mathrm{L}(i-1,i)} \cdot f_{\mathrm{L}(i-1)}(u^{0,t_e}_{\mathrm{L}(i-1),\mathrm{SI}})}{\partial W_{\mathrm{L}(i-1,i)}}$$

$$= -e_{\mathrm{L}(i),\mathrm{RI}} \cdot f_{\mathrm{L}(i-1)}(u^{0,t_e}_{\mathrm{L}(i-1),\mathrm{SI}})^T \odot f'_{\mathrm{L}(i)}(u^{0,t_e}_{\mathrm{L}(i),\mathrm{RI}}) \tag{S8}$$

$$\approx -e_{\mathrm{L}(i),\mathrm{RI}} \cdot f_{\mathrm{L}(i-1)}(u^{0,t_e}_{\mathrm{L}(i-1),\mathrm{SI}})^T \tag{S9}$$

$$= -f_{\mathrm{L}(i)}(u^{0,t_e}_{\mathrm{L}(i),\mathrm{RI}}) \cdot f_{\mathrm{L}(i-1)}(u^{0,t_e}_{\mathrm{L}(i-1),\mathrm{SI}})^T + f_{\mathrm{L}(i)}(u^{1,t_e}_{\mathrm{L}(i),\mathrm{RI}}) \cdot f_{\mathrm{L}(i-1)}(u^{0,t_e}_{\mathrm{L}(i-1),\mathrm{SI}})^T \tag{S10}$$

where $\odot$ denotes element-wise multiplication. It is the same as the update rule of most discrepancy-based algorithms in conventional FNN. We can further drop the derivative of activation function, because previous studies have shown that the derivative can be omitted as long as the activation function is monotonically non-decreasing [1, 2, 3]. Expanding the error term, we arrive at the last row of the equation.

C The influence of parameters

We further investigate the influence of the hyper-parameters on the training, with the FMNIST dataset.

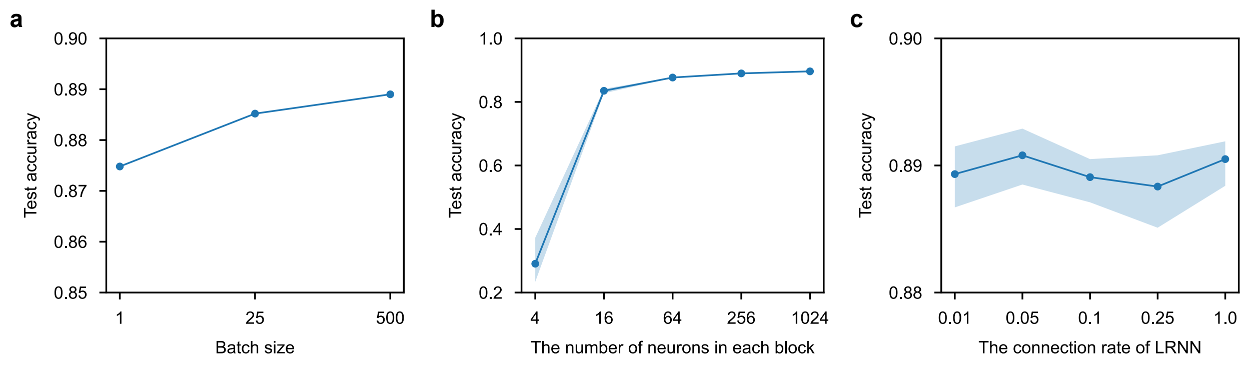

Figure S1: The test accuracy on FMNIST dataset with different parameters. a, Accuracy versus batch size. b, Accuracy versus the number of neurons in a RNN. The horizontal axis is number of neurons for each functional block (RI, SI, RE, SE). c, Accuracy versus the connecting rate of the RNN.

Batch size and Adam. In machine learning, it is common practice to train a network with batches of training data. Instead of computing the loss of one training sample and updating the weight, batch learning often compute the total loss of many samples, and then update weight. Usually batch learning can better approximate the gradient and accelerate the training. We have presented the results using batch learning in the main text. However, in physical hardware or biological networks, batch learning is non-local in time and demands storing the losses for the samples in a batch. Fig. S1(a) shows that the test accuracy increases by 1% with growing batch size. Importantly, even with batch size of 1 (without Adam optimizer[4]), the AP algorithm can reach 88.81% accuracy after 100 epochs. This result suggests that a suitable physical substrate can be trained with AP framework in a biologically plausible manner.

The number of neurons in each block of the RNN. In the main text, we usually set the number of neurons in each block (SI, RI, SE, RE) to be 256. To study how the size of RNN affects the performance, we have trained the two-layer MR-RNN model in the main test with different number of neurons for each functional block, ranging from 4 to 1024.Fig. S1(b) shows that increasing the block size improves the accuracy rate, however, when the block size is 64 or larger, the accuracy saturates. We attribute this saturation to the structural limit of the MR-RNN model. As discussed in the main text, further improvement on performance requires innovation in network structures, e.g., incorporating convolution layers.

The connection rate of RNN. We then examine the influence of connection rate of the RNN. The connection rate characterize the sparsity of the internal connections in a RNN, namely, the ratio of the number of existing connections to the number of all-to-all connections. With a block size of 256, the test accuracy for different connection rates are plotted in Fig. S1(c). It can be seen that sparse RNNs have comparable computational power with their dense counterpart, consistent with previous studies. Therefore, sparse RNNs are favorable for physical implementation in terms of stability and resource consumption.

References

[1] Alexander Ororbia and Ankur Mali. Biologically motivated algorithms for propagating local target representations. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4651–4658, 07 2019.

[2] Alexander G. Ororbia, Ankur Mali, Daniel Kifer, and C. Lee Giles. Backpropagation-free deep learning with recursive local representation alignment. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 9327–9335, Jun. 2023.

[3] Jan Melchior and Laurenz Wiskott. Hebbian-descent, 2019. Preprint at `https://arxiv.org/abs/1905.10585`.

[4] Jimmy Ba Diederik P. Kingma. Adam: A method for stochastic optimization. In *Proceedings of the International Conference on Learning Representations*. ICLR, 2015.