Appendix

A The Quantity Comparison Between Activations and Weights

As shown in Table 4, D_l/C_l is the size/channel of output of layer l and D_{l-1}/C_{l-1} is the size/channel of input to layer l; k is the kernel size. Normally, $D_l \approx D_{l-1}$ and $D_l \gg k$. Therefore, weights occupy greatly more memory overheads than the corresponding activations for FC; for LC, the weights also occupy more memory overheads than the corresponding activations because $(C_l \times k \times k) > (C_{l-1} + C_l)$. On the contrary, for CNN, the activations occupy greatly more memory overheads than the corresponding weights.

	Activations	Weights
FC	D_l+D_{l-1}	$D_l \times D_{l-1}$
LC	$C_l \times D_l$+$C_{l-1} \times D_{l-1}$	$C_l \times k \times k \times D_l$
CNN	$C_l \times D_l$+$C_{l-1} \times D_{l-1}$	$C_l \times k \times k$

Table 4: The quantity comparison between activations and weights for TinyFoA in various DNN architectures.

B Equations for TinyFoA and Other Algorithms

Steps	Forward Pass	Gradient Calculation
BP (Rumelhart, Hinton, and Williams 1986)	$\boldsymbol{h}_l = \sigma_l(\boldsymbol{W}_l \boldsymbol{h}_{l-1} + \boldsymbol{b}_l)$	$\delta \boldsymbol{W}_l = \left[(\boldsymbol{W}_{l+1}^T (\boldsymbol{W}_{l+2}^T ... (\boldsymbol{W}_L^T \delta \boldsymbol{h}_L \odot \sigma'_L) ... \odot \sigma'_{l+2}) \odot \sigma'_{l+1}) \right] \odot \sigma'_l \otimes \boldsymbol{h}_{l-1}^T$
DRTP (Frenkel, Lefebvre, and Bol 2021)	$\boldsymbol{h}_l = \sigma_l(\boldsymbol{W}_l \boldsymbol{h}_{l-1} + \boldsymbol{b}_l)$ $\boldsymbol{y}_{one-hot}$ is the one-hot-encoded label	$\delta \boldsymbol{W}_l = (\boldsymbol{B}_l^T \boldsymbol{y}_{one-hot}) \odot \sigma'_l \otimes \boldsymbol{h}_{l-1}^T$ where $\boldsymbol{B}_l$ is a fixed random connectivity matrix
PEPITA (Dellaferrera et al. 2022)	$\boldsymbol{h}_l = \sigma_l(\boldsymbol{W}_l \boldsymbol{h}_{l-1} + \boldsymbol{b}_l)$ $\boldsymbol{h}_l^{err} = \sigma_l(\boldsymbol{W}_l \boldsymbol{h}_{l-1}^{err} + \boldsymbol{b}_l)$	$\delta \boldsymbol{W}_l = (\boldsymbol{h}_l - \boldsymbol{h}_l^{err}) \otimes (\boldsymbol{h}_{l-1}^{err})^T$, where $\boldsymbol{h}_0^{err} = x + Fe$ e denotes the error and F denotes the fixed random matrix
FF (Hinton 2022)	$\boldsymbol{h}_l^{pos} = \sigma_l(\boldsymbol{W}_l \boldsymbol{h}_{l-1}^{pos} + \boldsymbol{b}_l)$ $\boldsymbol{h}_l^{neg} = \sigma_l(\boldsymbol{W}_l \boldsymbol{h}_{l-1}^{neg} + \boldsymbol{b}_l)$ $g_l^{(pos)} = (\boldsymbol{h}_l^{(pos)})^T \boldsymbol{h}_l^{(pos)}, g_l^{(neg)} = (\boldsymbol{h}_l^{(neg)})^T \boldsymbol{h}_l^{(neg)}$	$\delta \boldsymbol{W}_l = \boldsymbol{h}_l^{(pos)} \cdot \delta g_l^{(pos)} \odot \sigma'_l \otimes (\boldsymbol{h}_{l-1}^{(pos)})^T + \boldsymbol{h}_l^{(neg)} \cdot \delta g_l^{(neg)} \odot \sigma'_l \otimes (\boldsymbol{h}_{l-1}^{(neg)})^T$ $\delta g_l^{(pos)} = \text{Sigmoid}(g_l^{(pos)} - \theta) - 1$, $\delta g_l^{(neg)} = \text{Sigmoid}(g_l^{(neg)} - \theta) - 0$ where θ is the threshold
TinyFoA-V-BA-BW	$\boldsymbol{h}_l = \sigma_l(\boldsymbol{W}_l \boldsymbol{h}_{l-1} + \boldsymbol{b}_l)$ $\boldsymbol{a}_l = \boldsymbol{B}_l \boldsymbol{h}_l$	$\delta \boldsymbol{W}_l = (\boldsymbol{B}_l^T \delta \boldsymbol{a}_l) \odot \sigma'_l \otimes \boldsymbol{h}_{l-1}^T$ where $\boldsymbol{B}_l$ is a fixed random matrix

Table 5: The formulation of forward pass and gradient calculation for various algorithms, where σ_l is the ReLU activation function, $\boldsymbol{h}_0$ is the input $\boldsymbol{x}$. $\odot$ is the element-wise product and $\otimes$ is the Kronecker Product. For parameters update, $\boldsymbol{W}_l = \boldsymbol{W}_l - \eta \delta \boldsymbol{W}_l$ and $\boldsymbol{b}_l = \boldsymbol{b}_l - \eta \delta \boldsymbol{b}_l$, where η is the learning rate.

C The Derivation and Proof for Gradient and Parameters Update of TinyFoA

Let us consider a DNN with L layers. The input $\boldsymbol{x} \in \mathbb{R}^{D_x \times 1}$ and the target $\boldsymbol{y} \in \mathbb{R}^{N_c \times 1}$ are considered for training the DNN. The activations of the hidden layer l of the DNN are denoted as $\boldsymbol{h}_l$, where $\boldsymbol{h}_0 = \boldsymbol{x}$. For the hidden layer l, the activations $\boldsymbol{h}_l \in \mathbb{R}^{D_l \times 1}$ based on binary activations $\boldsymbol{h}_{l-1}^b \in \mathbb{R}^{D_{l-1} \times 1}$, binary weights $\boldsymbol{W}_l^b \in \mathbb{R}^{D_l \times D_{l-1}}$ and biases $\boldsymbol{b}_l \in \mathbb{R}^{D_l \times 1}$, are presented as follows:

$$\boldsymbol{h}_l = \sigma_l(\boldsymbol{W}_l^b \boldsymbol{h}_{l-1}^b + \boldsymbol{b}_l) = \sigma_l(\boldsymbol{z}_l). \tag{6}$$

For each $\boldsymbol{h}_l^b$, a fixed random matrix $\boldsymbol{B}_l \in \mathbb{R}^{N_c \times D_l}$ is employed to project the $\boldsymbol{h}_l^b \in \mathbb{R}^{D_l \times 1}$ to the output vector $\boldsymbol{a}_l \in \mathbb{R}^{N_c \times 1}$. Then, the projected vector $\boldsymbol{a}_l$ is used to calculate the error and the gradient, as follows:

$$\begin{aligned}
\boldsymbol{a}_l &= \boldsymbol{B}_l \boldsymbol{h}_l^b, \\
\boldsymbol{p}_l &= \sigma_{\text{output}}(\boldsymbol{a}_l), \\
\delta \boldsymbol{h}_l &= \frac{\partial \text{loss}_l(\boldsymbol{p}_l, y)}{\partial \boldsymbol{a}_l} \frac{\partial \boldsymbol{a}_l}{\partial \boldsymbol{h}_l^b} \frac{\partial \boldsymbol{h}_l^b}{\partial \boldsymbol{h}_l},
\end{aligned}$$

where $\boldsymbol{p}_l \in \mathbb{R}^{N_c \times 1}$ is the probability distribution vector of $\boldsymbol{a}_l$ and σ_{output} is the Softmax activation function. loss_l is the cross-entropy loss for the hidden layer l, and $\delta \boldsymbol{a}_l = \frac{\partial \text{loss}_l(\boldsymbol{p}_l, y)}{\partial \boldsymbol{a}_l}$ is the gradient of loss_l with respect to $\boldsymbol{a}_l$. $\delta \boldsymbol{h}_l \in \mathbb{R}^{D_l \times 1}$ is the gradient for updating weights $\boldsymbol{W}_l^b$ and biases $\boldsymbol{b}_l$.

For $\frac{\partial \text{loss}_l(\boldsymbol{p}_l, y)}{\partial \boldsymbol{a}_l}$, $\delta \boldsymbol{a}_l = \boldsymbol{p}_l - \boldsymbol{y}$ due to the cross-enropy loss. Additionally, $\frac{\partial \boldsymbol{a}_l}{\partial \boldsymbol{h}_l^b}$ is $\boldsymbol{B}_l^T$ and $\frac{\partial \boldsymbol{h}_l^b}{\partial \boldsymbol{h}_l}$ is $\mathbf{1}_{|\boldsymbol{h}_l| \leq 1}$ (means output 1 if $|\boldsymbol{h}_l| \leq 1$; otherwise output 0). Therefore, the gradient of $\text{loss}_l(\boldsymbol{p}_l, \boldsymbol{y})$ with respect to $\boldsymbol{h}_l$ is calculated as follows:

$$\begin{aligned}
\delta \boldsymbol{h}_l &= \frac{\partial \text{loss}_l(\boldsymbol{p}_l, y)}{\partial \boldsymbol{a}_l} \frac{\partial \boldsymbol{a}_l}{\partial \boldsymbol{h}_l^b} \frac{\partial \boldsymbol{h}_l^b}{\partial \boldsymbol{h}_l} \\
&= \boldsymbol{B}_l^T (\boldsymbol{p}_l - \boldsymbol{y}) \odot \mathbf{1}_{|\boldsymbol{h}_l| \leq 1}.
\end{aligned}$$

Moreover, considering the vertical layer-wise training, we only need to calculate the partial gradients $\delta \boldsymbol{W}_l^{(i,j)}$ to reduce the training memory overheads. We divide the input activation $\boldsymbol{h}_{l-1} \in \mathbb{R}^{D_{l-1} \times 1}$ and the output activation $\boldsymbol{h}_l \in \mathbb{R}^{D_l \times 1}$ into M slices, respectively. The sliced input activations are denoted as $\boldsymbol{h}_{l-1}^{(i)} \in \mathbb{R}^{\frac{D_{l-1}}{M} \times 1}$ and the sliced output activations are denoted as $\boldsymbol{h}_l^{(j)} \in \mathbb{R}^{\frac{D_l}{M} \times 1}$, where $i = 1, 2, ..., M$ and $j = 1, 2, ..., M$. Therefore, the gradient of $\text{loss}_l(\boldsymbol{p}_l, \boldsymbol{y})$ with respect to $\boldsymbol{h}_l^{(j)}$ is calculated as follows:

$$\begin{aligned}
\delta \boldsymbol{h}_l^{(j)} &= \frac{\partial \text{loss}_l(\boldsymbol{p}_l, y)}{\partial \boldsymbol{a}_l} \frac{\partial \boldsymbol{a}_l}{\partial \boldsymbol{h}_l^{b,(j)}} \frac{\partial \boldsymbol{h}_l^{b,(j)}}{\partial \boldsymbol{h}_l^{(j)}} \\
&= (\boldsymbol{B}_l^{(j)})^T (\boldsymbol{p}_l - \boldsymbol{y}) \odot \mathbf{1}_{|\boldsymbol{h}_l^{(j)}| \leq 1},
\end{aligned}$$

where $\boldsymbol{B}_l^{(j)} \in \mathbb{R}^{N_c \times \frac{D_l}{M}}$. $\mathbf{1}_{|\boldsymbol{h}_l^{(j)}| \leq 1} = 1$ if $|\boldsymbol{h}_l^{(j)}| \leq 1$ and $\mathbf{1}_{|\boldsymbol{h}_l^{(j)}| \leq 1} = 0$ otherwise.

Furthermore, we introduce the derivation for the sliced parameter (weights and biases) updates. The gradient of $\text{loss}_l(\boldsymbol{p}_l, \boldsymbol{y})$ with respect to $\boldsymbol{W}_l^{b,(i,j)}$ and $\boldsymbol{b}_l^{(j)}$ are calculated as follows:

$$\begin{aligned}
\delta \boldsymbol{W}_l^{b,(i,j)} &= \delta \boldsymbol{h}_l^{(j)} \frac{\partial \boldsymbol{h}_l^{(j)}}{\partial \boldsymbol{z}_l^{(j)}} \frac{\partial \boldsymbol{z}_l^{(j)}}{\partial \boldsymbol{W}_l^{b,(i,j)}} \\
&= (\boldsymbol{B}_l^{(j)})^T (\boldsymbol{p}_l - \boldsymbol{y}) \odot \mathbf{1}_{|\boldsymbol{h}_l^{(j)}| \leq 1} \odot \sigma_l'(\boldsymbol{z}_l^{(j)}) \otimes (\boldsymbol{h}_{l-1}^{b,(i)})^T \\
\delta \boldsymbol{b}_l^{(j)} &= \delta \boldsymbol{h}_l^{(j)} \frac{\partial \boldsymbol{h}_l^{(j)}}{\partial \boldsymbol{z}_l^{(j)}} \frac{\partial \boldsymbol{z}_l^{(j)}}{\partial \boldsymbol{b}_l^{(j)}} \\
&= (\boldsymbol{B}_l^{(j)})^T (\boldsymbol{p}_l - \boldsymbol{y}) \odot \mathbf{1}_{|\boldsymbol{h}_l^{(j)}| \leq 1} \odot \sigma_l'(\boldsymbol{z}_l^{(j)}) \cdot 1,
\end{aligned}$$

where $\odot$ is the Hadamard Product, and $\otimes$ is the Kronecker Product. $\sigma_l'(\boldsymbol{z}_l^{(j)}) = \mathbf{1}_{\boldsymbol{z}_l^{(j)} > 0}$ because σ_l is ReLU activation function here.

Finally, the gradient of $\text{loss}_l(\boldsymbol{p}_l, \boldsymbol{y})$ with respect to $\boldsymbol{W}_l^{(i,j)}$ is denoted as follow:

$$\begin{aligned}
\delta \boldsymbol{W}_l^{(i,j)} &= \delta \boldsymbol{W}_l^{b,(i,j)} \odot \mathbf{1}_{|\boldsymbol{W}_l^{(i,j)}| \leq 1} \\
&= \delta \boldsymbol{h}_l^{(j)} \odot \mathbf{1}_{\boldsymbol{z}_l^{(j)} > 0} \otimes (\boldsymbol{h}_{l-1}^{b,(i)})^T \odot \mathbf{1}_{|\boldsymbol{W}_l^{(i,j)}| \leq 1}
\end{aligned}$$

D Hyperparameter Details

In this section, we present the details of hyperparameters, including input size, category, batch size, epochs, learning rate, optimizer, the length of hidden layers, the activation function of hidden layers, and loss function, for all the algorithms and various datasets evaluated in this paper.

Table 6 demonstrates the hyperparameter details for all the algorithms, which means these hyperparameters apply to all four datasets in our experiments. Moreover, Table 7 demonstrates the hyperparameters details for these algorithms with respect to various datasets. Table 8 demonstrates the hyperparameters details for TinyFoA-LC.

Algorithms	Optimizer	Length/number of Hidden Layers	Activation Function	Loss	η
BP (Rumelhart, Hinton, and Williams 1986)	Adam	2000/4	ReLU	CCE	1e-4
DRTP (Frenkel, Lefebvre, and Bol 2021)	NAG	2000/4	Tanh	BCE	1e-4
PEPITA (Dellaferrera et al. 2022)	Mom	2000/3	ReLU	CCE	1e-3/1e-4
FF (Hinton 2022)	SGD	2000/4	ReLU	BCE	1e-3
TinyFoA-FC	Adam	2000/4	ReLU	CCE	1e-4

Table 6: The hyperparameter details for all the algorithms.

Hyperparameters	MNIST (LeCun 1998)	CIFAR-10 (Krizhevsky 2009)	CIFAR-100 (Krizhevsky 2009)	MIT-BIH (Mark et al. 1982)
Input Size	28x28x1	32x32x3	32x32x3	13x13x1
Category	10	10	100	5
Batchsize	100	100	100	100
Epochs	100	100	100	100
η of PEPITA	1e-3	1e-3	1e-3	1e-4

Table 7: The hyperparameter details for all the algorithms with respect to various datasets.

Hyperparameters	Channel	kernel	Stride	Epochs	Optimizer	Activation Function	Loss	η	Batchsize
TinyFoA-LC	[16,32,64,64]	3	1	200	Adam	ReLU	CCE	1e-4	100

Table 8: The hyperparameter details for TinyFoA-LC.