Learning Using a Single Forward Pass

Aditya Somasundaram * *as7458@columbia.edu*
Columbia University

Pushkal Mishra* *pumishra@ucsd.edu*
University of California San Diego

Ayon Borthakur* *ayon.borthakur@iitg.ac.in*
IIT Guwahati

Reviewed on OpenReview: *https://openreview.net/forum?id=EDQ8QOGqjr*

Abstract

We propose a learning algorithm to overcome the limitations of traditional backpropagation in resource-constrained environments: Solo Pass Embedded Learning Algorithm (SPELA). SPELA operates with local loss functions to update weights, significantly saving on resources allocated to the propagation of gradients and storing computational graphs while being sufficiently accurate. Consequently, SPELA can closely match backpropagation using less memory. Moreover, SPELA can effectively fine-tune pre-trained image recognition models for new tasks. Further, SPELA is extended with significant modifications to train CNN networks, which we evaluate on CIFAR-10, CIFAR-100, and SVHN 10 datasets, showing equivalent performance compared to backpropagation. Our results indicate that SPELA, with its features such as local learning and early exit, is a potential candidate for learning in resource-constrained edge AI applications.

1 Introduction

Backpropagation (BP) is a long-standing, fundamental algorithm for training deep neural networks (NNs) (Werbos, 1990; Rumelhart et al., 1986; LeCun et al., 1989; 2015). It is widely used in training multi-layer perceptrons (MLP), convolutional neural networks (CNN), recurrent neural networks (RNN), and now transformers. It is a loss minimization problem involving thousands, if not millions, of parameters. Data is first propagated through the network using forward passes, and the entire computational graph is stored. The difference (error) between the final layer output and the label is utilized to update the weight matrices of the network. The error is then propagated backward from the final layer using the chain rule of differentiation, which uses the stored computational graph to compute the associated gradients for each layer.

Careful observations of backpropagation drive home the point of no free lunch. Backpropagation works within a global learning framework, i.e., a single weight update requires knowledge of the gradient of every parameter (global/non-local learning problem) (Whittington & Bogacz, 2019). Backpropagation requires the storage of neural activations computed in the forward pass for use in the subsequent backward pass (weight transport problem) (Lillicrap et al., 2014; Akrout et al., 2019). For every forward pass, the backward pass is computed with the forward-pass updates frozen, which prevents online utilization of inputs (update locking problem) (Czarnecki et al., 2017; Jaderberg et al., 2017). Furthermore, there are several differences between backpropagation and biological learning, as mentioned in Section 2.

The constraints of backpropagation lead to significant challenges, such as high memory requirements. These limitations render backpropagation computationally expensive and unsuitable for applications in resource-constrained scenarios (example: on-device machine learning (ODL) Cai et al., 2020; Zhu et al., 2023). Efforts to mitigate these algorithmic constraints of backpropagation can lead to improved learning efficiency.

*AS & PM: joint first authors (order by coin toss); AB: PI. Work partly done at IIT Hyderabad.

We introduce and investigate a multi-layer neural network training algorithm — SPELA (Solo Pass Embedded Learning Algorithm) that uses embedded vectors as priors to preserve data structure as it passes through the network. Previous studies indicate that prior knowledge helps one learn faster and more easily (Goyal & Bengio, 2022; Wang & Wu, 2023). Although we do not claim complete biological plausibility, SPELA is built on the premise that biological neural networks utilize local learning (Illing et al., 2021) and neural priors to form representations. We introduce neural priors as symmetric vectors distributed on a high-dimensional sphere whose dimension equals the size of the corresponding neuron layer. Our backpropagation-free learning algorithm demonstrates a significant gain in computational efficiency, making it suitable for on-device learning (ODL) and brain-inspired learning features, such as early exit (for layered cognitive reasoning (Scardapane et al., 2020)) and local learning. In this paper, we make the following contributions:

- **Design:** We introduce SPELA. Next, we extend SPELA to convolutional neural networks. SPELA is a family of algorithms that use a single forward pass (with no backward pass) for training. During inference, output from any layer can be utilized for prediction. It makes an innovative use of embedded vectors as neural priors for efficient learning.
- **Evaluate:** Experiments conducted in this paper indicate that SPELA closely matches backpropagation in performance, and due to its computational efficiency, maintains an edge over it in resource-constrained scenarios. Moreover, SPELA can efficiently fine-tune models trained with backpropagation (transfer learning). In addition, extending SPELA to convolutional neural networks (CNNs) allows for complex image classification.
- **Complexity Analysis:** Theoretical bounds for peak memory usage show that SPELA can edge over backpropagation in the analyzed settings.

2 Related works

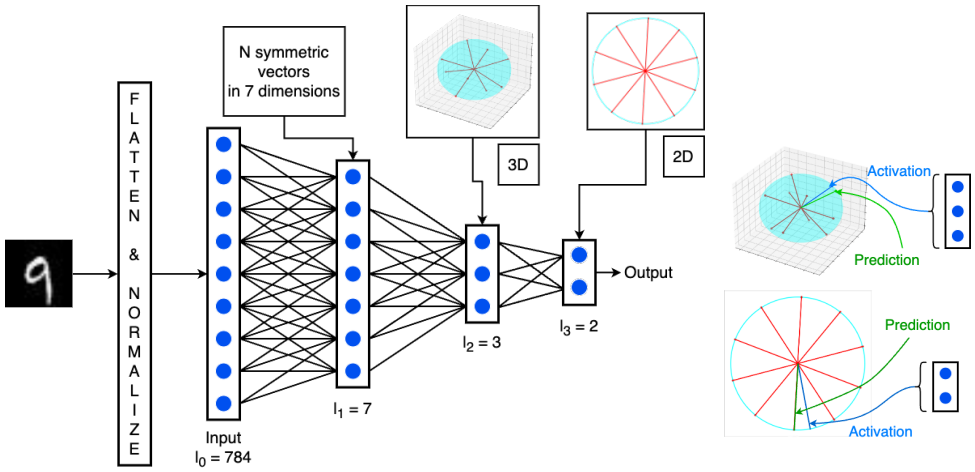
Recently, there has been significant interest in designing efficient training methods for multi-layer neural networks. Hinton (2022) presents the Forward-Forward (FF) algorithm for neural network learning. In FF, backpropagation is replaced with two forward passes: one with positive (real) data and the other using generated negative data. Each layer aims to optimize a goodness metric for positive data and minimize it for negative data. Separating positive and negative passes in time enables offline processing, facilitating image pipelining without activity storage or gradient propagation interruptions. These algorithms have garnered significant attention, and multiple modifications have been proposed in conjunction with applications in image recognition (Lee & Song, 2023; Pau & Aymone, 2023; Momeni et al., 2023; Dooms et al., 2024; Chen et al., 2024) and in graph neural networks (Park et al., 2024). In our experiments, SPELA can classify any layer without storing goodness memory, in contrast to the FF approach, which aggregates goodness values across layers. Furthermore, unlike FF, SPELA eliminates the need to generate separate negative data for training. Instead, it efficiently uses the available data without requiring additional processing.

Using forward and backward passes, Pehlevan (2019) introduced the concept of a non-negative similarity matching cost function for spiking neural networks to exhibit local learning and enable effective use of neuromorphic hardware. Lansdell et al. (2020) introduced a hybrid learning approach wherein each neuron learns to approximate the gradients. The feedback weights provide a biologically plausible way to achieve performance comparable to networks trained via backpropagation. Giampaolo et al. (2023) follows a similar strategy to our approach of dividing the entire network into sub-networks and training them locally using backpropagation (SPELA divides the network into sequential layers). Rather than backward propagation, Dellaferrera & Kreiman (2022) uses two forward passes and uses the global error to modulate the second forward-pass input. Recently, Li et al. (2025) proposed a diffusion-inspired denoising-based training of neural networks. We consider Dellaferrera & Kreiman (2022) as the closest match to our approach, but with one forward pass and layer-specific non-global error during training for SPELA. Inspired by pyramidal neurons, Lv et al. (2025) used distinct weights for forward and backward passes to train a multi-layer network in a biologically plausible manner. As detailed by Pau & Aymone (2023); Srinivasan et al. (2024), these algorithms still lack several desired characteristics of on-device learning.

3 Methods

3.1 Network Initialization and Learning Methods

The network is defined as follows: there are L layers, each containing l_i neurons followed by a nonlinear activation function (e.g Leaky ReLU). The weights of the network are initialized randomly. Each layer L_i (except the input layer) has N (number of classes in the given dataset) number of symmetric vectors, each of dimension l_i. These symmetric vectors are assigned a unique class. As the activation vector is also in the l_i dimensional space, we can measure how close the activation vector points to a particular symmetric vector using a simple cosine similarity function (Momeni et al., 2023). Based on cosine similarity, the network predicts the class assigned to the symmetric vector closest to the activation vector. These symmetric vectors remain fixed and are not updated during training. We describe the symmetric vector generation method in detail in Section A.1.



(a) Network Architecture (b) Prediction Method

Figure 1: (a) Network Architecture: Each layer possesses a distinct set of symmetric vectors. Here, the network is trained on MNIST-10, resulting in 10 symmetric vectors. (b) Prediction Method: Inference is performed using the closeness of activation and symmetric vectors. The activation is represented in blue, and the prediction is in green.

3.2 SPELA description

Algorithm 1 and 3 describe SPELA's training and testing methodologies, respectively. In this method, after passing a data point through layer i, we update layer i's weights and biases and then propagate the data point to layer $i+1$. Each layer updates its parameters as the data moves through the network using its local loss function. In this fashion, parallel training of all n layers becomes possible. Table 1 contrasts SPELA and other learning algorithms. Our algorithm exhibits the most favorable traits for the applications discussed in this work: a single forward pass for training, no backward pass, and a local loss function with no storage of activations.

3.3 Complexity Analysis

Updating the weights of the final layer in backpropagation requires one matrix-vector multiplication. Every other layer requires two matrix-vector multiplications to consider gradients from subsequent layers. SPELA can be visualized as cascading blocks of one-layer networks that perform classification at every junction. It can be viewed as a sequence of final layers from the backpropagation algorithm. Updating the weights of any layer using SPELA (as the weight updates are all analogous to the final layer of backpropagation) requires only one matrix-vector multiplication. For deep neural nets, the number of vector-matrix multiplications

Learning Methods	BP	FF	PEP	MPE	SPELA
Forward Pass	1	2	2	3	1
Backward Pass	1	0	0	0	0
Weight Update	1	2	1	1	1
Loss function	global	local	global	global	local
Activations	all	current	all	current	current

Table 1: Different learning algorithms are compared and contrasted with SPELA. PEP stands for PEPITA (Dellaferrera & Kreiman, 2022) and MPE for MEMPEPITA (Pau & Aymone, 2023).

Algorithm 1 Training MLP with SPELA

Given: An input (X), label (l), number of layers (K), and number of epochs (E)
Define: $\text{cos\_sim}(A, B) = \frac{A.B}{||A||.||B||}$ and $\text{normalize}(X) = \frac{X}{||X||}$ ▷ Dot product and normalization of vector
Set: $h_0 = x$
for $e \leftarrow 1$ to E **do** ▷ Iterate through epochs
  for $k \leftarrow 1$ to K **do** ▷ Iterate through layers
    $h_{k-1} = \text{normalize}(h_{k-1})$
    $h_k = \sigma_k(W_k h_{k-1} + b_k)$
    $\text{loss}_k = -\text{cos\_sim}(h_k, \text{vecs}_k(l))$ ▷ $\text{vecs}_k(.)$ is the set of symmetric vectors
    $W_k \leftarrow W_k - \alpha * \nabla_{W_k}(\text{loss}_k)$ ▷ Weight update using local loss
    $b_k \leftarrow b_k - \alpha * \nabla_{b_k}(\text{loss}_k)$ ▷ Bias update using local loss
  end for
end for

needed for weight updates is half of what backpropagation requires. Furthermore, this excludes other operations required by backpropagation and not by SPELA, such as transposing of weights. Both algorithms would need one vector-matrix multiplication for a forward pass. Considering this, SPELA requires about 0.67 as many matrix-vector multiplications as backpropagation (where the relative MACC for training is twice that of inference (Cai et al., 2020)). Regarding memory (see Table 2), we describe the complexities involved in variables that must be saved to calculate the weight updates. We do not include overhead memory complexities from storing weights, optimizer states, temporary variables, or other sources. At each computational step, SPELA trains only a single layer; hence, only that layer's activation needs to be stored. SPELA offers superior computational and memory complexity compared to backpropagation.

Algorithm	Forward pass complexity	Weight update complexity	Memory complexity
SPELA	N^2L	LN^2	N
BP	N^2L	$2LN^2$	LN

Table 2: The computation and memory complexities are shown above. The NN is considered to have L layers, each having N neurons. The complexity of a vector matrix multiplication is assumed to be $O(N^2)$. Here, for SPELA, it is assumed that activations are not stored.

3.4 The learning in SPELA

During SPELA learning, every layer can be considered a classifier head. Instead of moving the activation towards a one-hot encoded vector as in conventional training schemes, we focus on moving the activation vector towards a predefined symmetric vector. At each layer, output $o_i = \frac{h_i}{||h_i||}$, where $h_i = \sigma(z_i)$, where $z_i = W_i h_{i-1} + b_i$, where W_i and b_i are layer weights and bias, h_{i-1} is the previous layer activation output, $\sigma()$ is an activation such as Leaky ReLU. Also, assume that the symmetric vector assigned to the correct class is v. Then the loss is defined as the negative of cosine similarity: $L = -o_i^T \cdot v$, as $||o_i|| = ||v|| = 1$. Mathematically, we compute

$$\frac{\partial L}{\partial o_i} = \frac{\partial}{\partial o_i} -o_i^T \cdot v = -v \equiv g_o \text{ and } \frac{\partial L}{\partial h_i} = \frac{\partial o_i}{\partial h_i} g_o = \frac{1}{||h_i||}(I - o_i o_i^T) \equiv g_h$$

$$\frac{\partial L}{\partial z_i} = g_h \odot \sigma'(z_i) \equiv g_z, \text{ where } \sigma'(z) = \begin{cases} 1 & z > 0, \\ \alpha & z \leq 0 \end{cases}$$

$$\frac{\partial L}{\partial W_i} = g_z h_{i-1}^T \text{ and } \frac{\partial L}{\partial b_i} = g_z$$

The same process is identically applicable to all layers L in the network. This finally results in $W_i^t \leftarrow W_i^{t-1} - \gamma g_z h_{i-1}^T$ and $b_i^t \leftarrow b_i^{t-1} - \gamma g_z$, where γ is the learning rate and t is iteration step. In our implementation, the symmetric vector embeddings represent the layer-wise classifier head weights, which emulate cosine similarity computation, without getting updated during training. The output of this classifier head is utilized to compute a cross-entropy loss or a cosine loss (log(2 - cosine similarity). The derivation above assumes a simpler setting, allowing readers to understand weight updates.

3.5 SPELA for Convolutional Neural Network

For the Convolutional Neural Networks (CNNs) (Figure 3a), we use the interleaving of traditional CNN and multi-layer perceptron (MLP) layers to incorporate SPELA. The modifications are as follows: each kernel in the convolutional layer is assigned a certain number of *groups*. Each group has a nonzero number of classes. Each class belongs to exactly one group, with all classes evenly distributed across groups. Each CNN layer predicts the group assignment of an input class. For example, when the groups are (dog, cat, fish), (banana, boat, bug), and (football, airplane, phone), then the class 'dog' will belong to the first group. The number of groups and classes assigned to each group varies between kernels, with randomness facilitating our performance. Each class is given a score depending on what the kernel returns. If a kernel returns the first group, all the classes corresponding to the first group get a score of one. Similarly, if a kernel returns the second group, the corresponding classes are scored, and so on. After tallying the scores, the class with the highest cumulative score is selected as the CNN layer's output. This particular distribution of groups and classes in groups is random across kernels but is consistent across layers, one of the restrictions of our method. After obtaining the output of a particular CNN kernel, this slice of 2D data is flattened and projected down to a smaller dimension using a simple MLP. The classification is performed in the MLP precisely in a typical SPELA MLP setup. The data is pushed through the network after the classification. We keep the MLP as tiny as possible to mitigate learning in this perpendicular direction and focus more on training the CNN layer. Algorithm 2 details the layer-wise training procedure for CNNs.

4 Empirical Studies

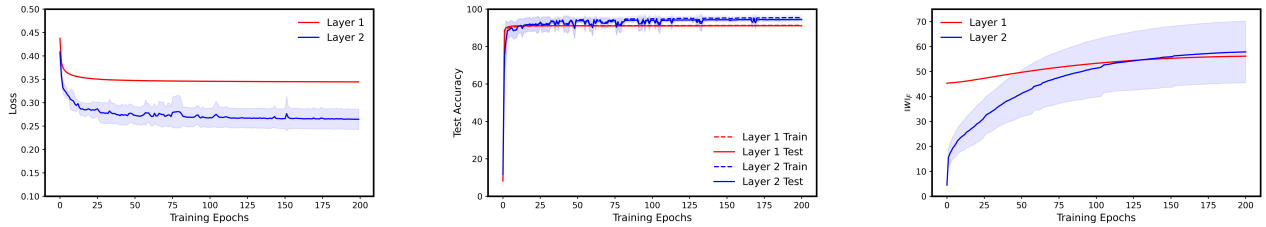
4.1 How does SPELA work?

We perform an in-depth analysis of SPELA's capacity. We first understand SPELA's learning dynamics on the standard MNIST 10 dataset. Following the design described in Dellaferrera & Kreiman (2022), we evaluate the performance of a $784 \rightarrow 1024 \rightarrow 10$ size SPELA network on MNIST 10. The network is trained by cosine loss defined as log(2 - Cosine Similarity). Figure 2a describes both layers of SPELA's epoch-specific decreasing training loss curves. The mean loss for layer 2 (0.26 after 200 epochs) is always lower than the corresponding layer 1 loss (0.34 after 200 epochs). Figure 2b describes the rise of test accuracy with the number of training epochs with SPELA. It is observed that SPELA training is effective at improving the accuracy from 11.60% without learning to 94.49% after 200 epochs of training. By design, SPELA can perform predictions at all layers (except the input layer), wherein the amount of available resources can determine the number of layers. Figure 2b also shows that the accuracy increases with the layer counts on MNIST 10 (91.18% accuracy for layer one vs. 94.49% accuracy for layer two after 200 epochs of training), thereby justifying the need for multiple layers to improve network performance. This also empowers SPELA with an early exit feature (Scardapane et al., 2020), enabling easy neural network distribution across hardware

Algorithm 2 Training and Inference from CNN i^{th} layer with SPELA

Given: κ number of classes, $C = \{1, 2, \ldots, \kappa\}$, n_i kernels and B_i is conv block from previous layer
Define: $S_i = 0$ is score for class $i, \forall i \in C$
Define: m groups such that:
$\rightarrow$ $G_i \subset C$
$\rightarrow$ $\bigcup_{i=1}^{m} G_i = C$
$\rightarrow$ $G_i \cap G_j = \phi \; \forall i \neq j$
for $j \leftarrow 1$ to n_i **do** ▷ Define MLP for each kernel
    MLP_j $\mathcal{N}(0,1)_{*\times d}$
end for
for $j \leftarrow 1$ to n_i **do** ▷ Kernel + MLP operation
    $o_j = \text{CNN}(B_i, k_j)$ ▷ k_j is the j^{th} kernel
    $o_j^{'} = \text{flatten}(o_j)$
    $o_j^{''} = \text{MLP}_j(o_j^{'})$
    $\text{loss}_j = -\text{cos\_sim}(o_j^{''})$
    Say the closest predicted group is G_m via cos_sim loss
    for c in G_m **do**
        $S_c = S_c + 1$
    end for
end for
for $j \leftarrow 1$ to n_i **do**
    $w_{k_j} \leftarrow w_{k_j} - \alpha * \nabla_{k_j}(\text{loss}_k)$ ▷ Kernel parameter update using local loss
    $w_{\text{MLP}_j} \leftarrow w_{\text{MLP}_j} - \alpha * \nabla_{\text{MLP}_j}(\text{loss}_k)$ ▷ MLP parameter update using local loss
end for
Prediction: $\arg\max_i \text{S}_i$

platforms and improving inference. Moreover, it is observed that SPELA reaches near maximum performance very quickly. SPELA achieves 74.88% performance after only 1 epoch of training on MNIST 10. Next, we analyze the effect of SPELA training on network parameters as in Figure 2c. The Frobenius norm of the layer weights demonstrates an increasing trend with learning, precisely $45.27, 4.49$ (before learning) to $56.16, 57.86$ (after 200 epochs) for layer 1, and layer 2, respectively.

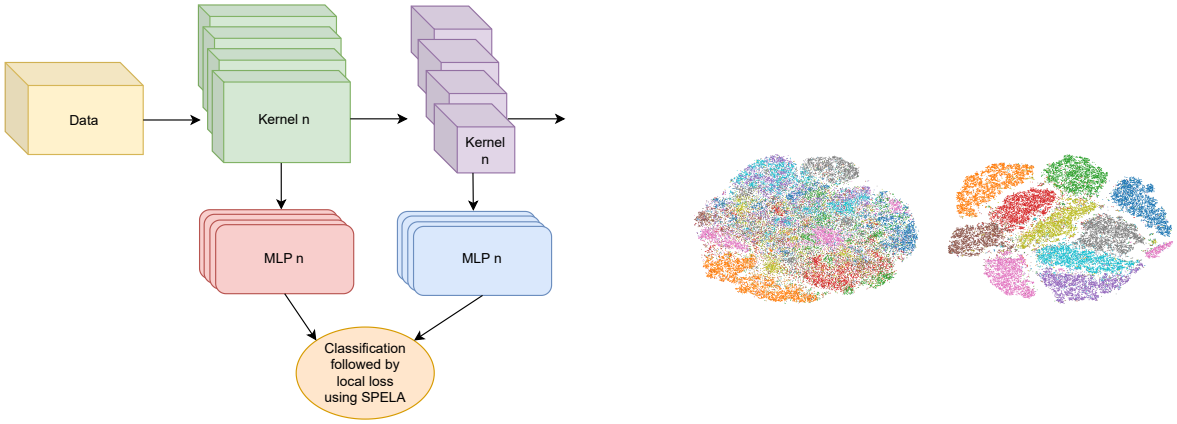


(a) Loss Curve w.r.t Epochs (b) Accuracy w.r.t Epochs (c) Norm of weights w.r.t Epochs

Figure 2: SPELA network behavior during learning to classify MNIST 10 digits. (a) Cosine training loss of SPELA to the progression of epochs. (b) Train and Test accuracy of both the hidden layer(1024 neurons) and the output layer(10 neurons) over epochs. (c) Evolution of Frobenius norm of the layer weights for SPELA with learning. The solid lines denote the mean, and the shades denote the standard deviation of five simulation runs.

Figure 3b establishes the representation learning capabilities of SPELA. A t-SNE embedding analysis of the output layer representation (10 neurons) before learning in Figure 3b exhibits a high degree of overlap of the MNIST 10 digit classes. However, after training with SPELA for 200 epochs of the network Figure 3c, we

observe 10 distinct clusters corresponding to MNIST classes in the t-SNE embeddings of the output layer representation.



(a) (b) Before Learning (c) After Learning

Figure 3: (a) Schematic diagram of SPELA Convolutional Neural Network. (b), (c) Two-dimensional t-SNE embeddings of the output layer(10 neurons) of a SPELA MLP network trained on MNIST 10 (b) Before any training, (c) After training for 200 epochs. The colors corresponded to different digits of MNIST 10.

SPELA can predict directly by comparing the layer output with the symmetric class embeddings. However, we parallelize this comparison using a classifier head with weights determined by the class embeddings. This design can use either a cosine loss (SPELA) or a cross-entropy loss (SPELA_CH) for layer training. We compare SPELA's performance wherever possible with existing work. Table 3 describes the key comparison results on MNIST 10, KMNIST 10, and FMNIST datasets (we selected these datasets as they are well suited for a multilayer perceptron classification task). Table 7 similarly describes the performance of SPELA on reasonably complex datasets such as CIFAR 10, CIFAR 100, and SVHN 10. Notably, across both tables, SPELA is the only training method that operates with a single forward pass; in contrast, backpropagation (BP), feedback alignment (FA), direct random target projection (DRTP), and PEPITA require either a forward pass followed by a backward pass, or two consecutive forward passes.

To ensure a fair comparison, we evaluate SPELA using two configurations: (i) a baseline matching the setup from previous work (Dellaferrera & Kreiman, 2022) (SPELA_CH_A and SPELA_A), and (ii) an optimized configuration (SPELA_CH_B and SPELA_B). Full experimental details are provided in Tables 19 and 20.

On the MNIST 10 dataset, SPELA achieves an accuracy of only 3.91% lower without any dropout than an equivalent backpropagation-trained network (94.49% vs. 98.40%). Interestingly, a $784 \rightarrow 1000$ SPELA achieves a 91.47% performance for the same training. Using the same network configuration, SPELA achieves 77.13% on KMNIST-10 and 85.08% on Fashion MNIST. Additionally, we observe that using cross-entropy loss (SPELA_CH) slightly improves performance on KMNIST-10 and Fashion MNIST compared to cosine loss (77.13% vs. 76.36%, and 85.08% vs. 85.00%, respectively). Conversely, on MNIST-10, cosine loss yields marginally better results (94.49% vs. 94.14%). Finally, incorporating dropout did not lead to performance improvements for SPELA across any of the datasets.

Keeping the input and output layer sizes fixed at 784 and 10, we varied the number of hidden layers (1024 neurons each) of SPELA from 1 to 9. Table 4 describes the results of such multilayer SPELA. We observe that a 4-hidden-layer SPELA performs best on MNIST (96.28%). Similarly, a 7-hidden layer and a 4-hidden layer perform best on KMNIST 10 and Fashion MNIST 10 respectively (83.57%, and 87.16%).

By design, SPELA does not require a softmax layer whose dimensions are usually equal to the number of classes in the dataset. Without this constraint, we explore the optimal network configuration, keeping the total number of neurons constant ($1024 + 10 = 1034$). Figure 4 and Table 8 clearly show that SPELA's performance depends on output layer size. When the output layer size is 5 (and the corresponding hidden layer

size is 1029), MNIST 10, KMNIST 10, and Fashion MNIST 10 performances are merely 53.19%, 35.66%, and 50.31% respectively. Conversely, a $784 \rightarrow 984 \rightarrow 50$ architecture, wherein the output layer size is five times that of the number of classes, achieves the best MNIST 10, KMNIST 10, Fashion MNIST 10 performances of 96.28%, 80.23% and 87% respectively. This work does not aim to compete with backpropagation (BP). Instead, our focus is on ANN training for applications where BP is computationally infeasible due to issues such as storage of forward pass activations (such as in the case of on-device learning with backpropagation (Cai et al., 2020)).

Model	**Architecture**	**# Epochs**	**MNIST 10**	**KMNIST 10**	**FMNIST 10**
BP_A	$784 \rightarrow 1024 \rightarrow 10$	100	98.56 ± 0.06	92.73 ± 0.11	88.72 ± 0.21
FA	$784 \rightarrow 1024 \rightarrow 10$	100	98.42 ± 0.07	-	-
DRTP	$784 \rightarrow 1024 \rightarrow 10$	100	95.10 ± 0.10	-	-
PEPITA	$784 \rightarrow 1024 \rightarrow 10$	100	98.01 ± 0.09	-	-
FF	$784 \rightarrow 4 \times (\rightarrow 2000)$	60	98.6	-	-
SPELA_CH_A	$784 \rightarrow 1024$	100	87.02 ± 0.2	59.73 ± 0.23	73.63 ± 0.52
SPELA_CH_A	$784 \rightarrow 1024 \rightarrow 10$	100	82.41 ± 5.16	72.52 ± 3.13	77.17 ± 6.11
SPELA_A	$784 \rightarrow 1024$	100	89.55 ± 0.06	64.27 ± 0.19	79.25 ± 0.11
SPELA_A	$784 \rightarrow 1024 \rightarrow 10$	100	90.27 ± 4.17	64.71 ± 6.96	72.58 ± 8.23
BP_CH_B	$784 \rightarrow 1024 \rightarrow 10$	200	98.40 ± 0.08	88.81 ± 2.03	91.89 ± 0.23
SPELA_CH_B	$784 \rightarrow 1024$	200	91.47 ± 0.06	68.7 ± 0.24	83.68 ± 0.04
SPELA_CH_B	$784 \rightarrow 1024 \rightarrow 10$	200	94.14 ± 1.22	77.13 ± 2.23	85.08 ± 0.58
SPELA_B	$784 \rightarrow 1024$	200	91.18 ± 0.09	68.7 ± 0.17	84.01 ± 0.10
SPELA_B	$784 \rightarrow 1024 \rightarrow 10$	200	94.49 ± 0.74	76.36 ± 2.79	85 ± 1.2

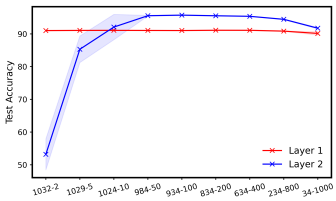
Table 3: Test accuracies (mean ± standard deviation) comparison of different learning methods on MNIST 10, KMNIST 10, and FMNIST 10 (Fashion MNIST 10) datasets. The accuracies of FA, DRTP, and PEPITA are as presented in Dellaferrera & Kreiman (2022). FF results are reported in Hinton (2022) for a network of 4 hidden layers with 2000 neurons each. We report both hidden layer and output mean accuracies (average of five runs) of SPELA. Table 19 and Table 20 describe all the relevant experiment details.

Dataset	**# 1**	**# 2**	**# 3**	**# 4**	**# 5**	**# 6**	**# 7**	**# 8**	**# 9**
MNIST 10	91.21 ± 0.11	95.7 ± 0.08	96.23 ± 0.07	96.28 ± 0.09	96.22 ± 0.05	96.26 ± 0.08	96.15 ± 0.06	96.16 ± 0.06	96.14 ± 0.08
KMNIST 10	68.70 ± 0.11	80.32 ± 0.27	83.11 ± 0.13	83.34 ± 0.06	83.47 ± 0.21	83.48 ± 0.19	83.57 ± 0.20	83.16 ± 0.2	83.23 ± 0.22
Fashion MNIST 10	83.98 ± 0.07	86.81 ± 0.09	87.14 ± 0.08	87.16 ± 0.06	87.16 ± 0.08	87.12 ± 0.07	87.07 ± 0.13	87 ± 0.15	86.95 ± 0.06

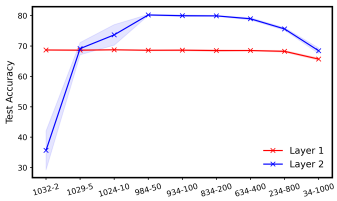
Table 4: Test accuracies (mean ± standard deviation) of SPELA for varying network depth. Columns indicate the number of 1024-neuron hidden layers in the network. For instance, column #1 denotes a $784 \rightarrow 1024 \rightarrow 10$ network, #2 denotes a $784 \rightarrow 1024 \rightarrow 1024 \rightarrow 10$ network. Each network with SPELA_B configuration is trained for 200 epochs and five runs.

4.2 Training Memory Comparison

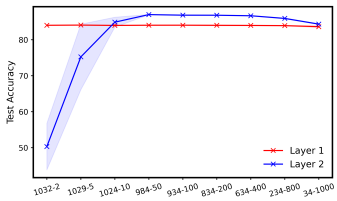
Next, we analyzed and compared SPELA's peak training memory consumption with an equivalent BP network. Training memory typically dominates model memory during learning; hence, we focus on peak training memory usage. Figure 5 presents the relative peak GPU memory consumption when the number of hidden layers varies from 1 to 9. For all batch sizes: 50, 100, 200, 400, 800, and 1000, the relative peak training memory consumed by SPELA training is almost the same, resulting in a flat curve. We use the training memory consumed by a $784 \rightarrow 1024 \rightarrow 10$ as a base for computing the relative peak training memory. Meanwhile, for a BP network, the relative peak training memory increases with the number of hidden layers



(a) MNIST 10



(b) KMNIST 10



(c) Fashion MNIST 10

Figure 4: Variation of SPELA's test performance for output layer sizes of 2, 5, 10, 50, 100, 200, 400, 800, and 1000. We keep the total number of network neurons fixed to 1034; hence, the corresponding hidden layer sizes are 1032, 1029, 984, 934, 634, 234, and 34. Throughout the experiments, the SPELA_B configuration is used for 200 training epochs. The solid lines denote the mean, and the shades denote the standard deviation of five simulation runs.

and batch size. Notably, the training memory is the residue after subtracting the model memory from the total peak memory. Due to SPELA's design, such as a per-layer classifier head which uses the vector embeddings as weights, the model memory of SPELA should be very slightly higher than that of BP, which we observe in Table 9 (for instance, 38.20 MB vs. 37.83 MB for a nine-hidden-layer layer network).

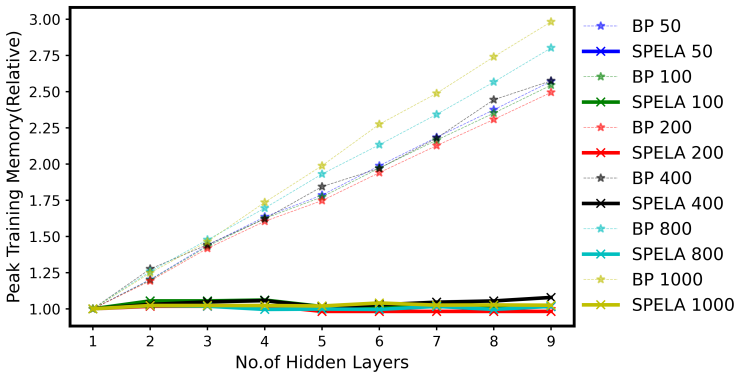
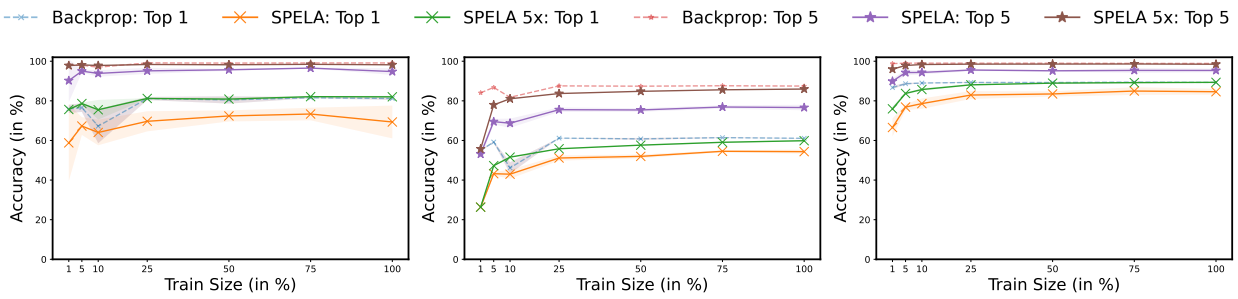


Figure 5: Variation of relative peak training memories(average of five runs) with layer depth for SPELA and BP. We use the peak training memory occupied by a $784 \rightarrow 1024 \rightarrow 10$ SPELA/BP network to report the relative peak training memory. The hidden layer sizes vary from 1-9 with 1024 neurons in each hidden layer, and for batch sizes of 50, 100, 200, 400, 800, and 1000.

4.3 Transfer Learning with SPELA

In real-world applications, on-device learning(ODL) helps mitigate the impact of phenomena such as data drift by enabling the on-device update of ML models. However, on-device learning must be performed under constraints such as memory and power for tiny ML applications. Consider a neural network (NN) with three layers: L_0, L_1, and L_2. Previous work on on-device learning using backpropagation shows that forward pass activation storage of such layers consumes significantly higher memory than neural network(NN) parameters (Cai et al., 2020). This assumes a network design requiring $L_0 \rightarrow L_1 \rightarrow L_2$ synapses as well as $L_0 \leftarrow L_1 \leftarrow L_2$ synapses. SPELA, on the other hand, would need only a unidirectional synaptic connection $L_0 \rightarrow L_1 \rightarrow L_2$ - implying SPELA would need fewer connection wires. These traits should make SPELA useful in tinyML applications. In this section, we examine the behavior of SPELA in the framework of tiny transfer learning, wherein the models are pre-trained with backpropagation and optimized using SPELA. We compute the top-1 and top-5 accuracies for varying degrees of training data(keeping the test dataset fixed). We follow the canonical transfer learning approach wherein the classifier head is replaced by a layer size equivalent to the number of classes. Moreover, as observed in Figure 4, since SPELA performs better when the output layer

is 5× the number of classes, we also evaluate performance on SPELA 5x. Figure 6, 8 and Tables 10, 11, 12, 13, 14, 15, describe the performance on the six datasets. Although the ResNet50 model is trained with backpropagation(BP) and should be the obvious training method, SPELA doesn't lag far behind on the six datasets. Moreover, similar to previous observations in Figure 4, SPELA 5x always outperforms SPELA. For instance, SPELA 5x achieves a performance of 98.20% (vs. 99.12% for a backpropagation network) on CIFAR 10 dataset, 85.95%(vs. 87.46% for a backpropagation network) on CIFAR 100 dataset, 98.48%(vs. 98.88% for a backpropagation network) on Pets 37 dataset, 82.96%(vs. 85.21% for a backpropagation network) on Food 101 dataset.



(b) CIFAR 10 dataset (c) CIFAR 100 dataset (d) Pets 37 dataset

Figure 6: Accuracy plots (after 200 epochs of fine tuning) of SPELA, SPELA 5x and Backpropagation trained networks for train dataset size percentages of 1, 5, 10, 25, 50, 75, and 100 during transfer learning(keeping the test dataset fixed). The solid lines denote the mean, and the shades denote the standard deviation of five simulation runs. SPELA 5x denotes a network with a classifier layer size 5× the number of classes.

4.4 Ablation Studies

Why not use alternate distance? Just as we try to orient the vectors to their corresponding embedded direction for correct classification, another question arises: Instead of classifying data in terms of closeness concerning angle (*cosine loss*), why not classify data in terms of closeness concerning distance (Euclidean loss)? Here, we run experiments by replacing our Cosine loss function defined by log(2 - cosine similarity) with the Euclidean norm loss function. Table 5 shows that when tested on MNIST 10, KMNIST 10, and Fashion MNIST 10 datasets, SPELA with Euclidean distance performs significantly lower than SPELA with cosine distance after training of 200 epochs (75.37% vs. 94.41% for MNIST 10, 52.71% vs. 75.05% for KMNIST 10 and 67.83% vs. 84.12% for Fashion MNIST 10).

Cer et al. (2018) justified that small angles have very similar cosines and proposed using angular similarity: $1 - \frac{\arccos(x)}{\pi}$, for classification. Table 5 shows that angular loss defined by log(2 - angular similarity) performs slightly better than cosine loss for KMNIST 10 (75.44% vs. 75.05%) and Fashion MNIST 10 (85.1% vs. 84.12%). On MNIST 10, angular loss performs almost at par with cosine loss (94.39% vs. 94.41%).

Randomizing the vector embeddings Symmetrically distributing the vectors is intuitive and axiomatic, as explained in Section A.1. However, the question remains: How much would the performance drop if we randomly choose these vectors? Accordingly, we rid ourselves of the complexity of finding a symmetric distribution and run experiments by drawing the embedded vectors from a Gaussian distribution $\mathcal{N}(\mathbf{0}, \mathbf{1})$ as well as from a uniform distribution $\mathcal{U}(-\mathbf{1}, \mathbf{1})$. Table 5 indicates that classification performance slightly drops for all three datasets when vectors are drawn at random. However, this performance drop is significantly lower than Euclidean distance in SPELA (MNIST 10: 4% vs. 19.54%, KMNIST 10: 6.65% vs. 22.34%, Fashion MNIST 10: 0.09% vs. 16.29%). In Appendix B.3.2, we discuss this observation further.

Except for cosine and angular similarity in all of these studies, for SPELA, the first layer classification is better than the subsequent layer in the early exit set-up.

Binarization (±1) of SPELA Anderson & Berg (2018) shows that binarization does not significantly change the directions of the high-dimensional vectors. As our algorithm tries to orient the activations

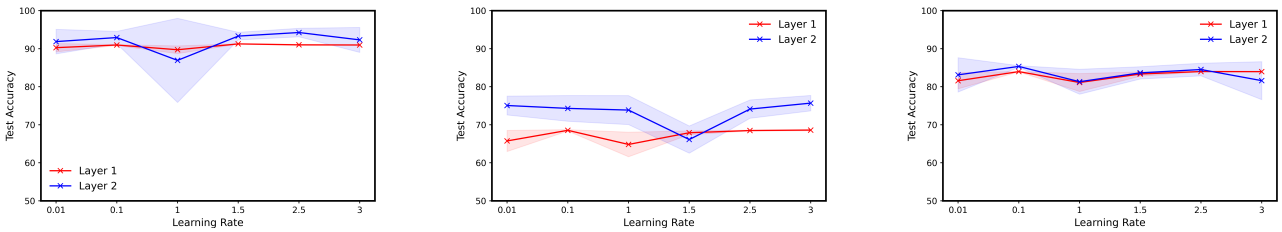
to a particular direction in high dimensions, it is safe to assume that the weight binarization should not significantly affect the performance. In Table 5, we show the results of our experiments involving the binarization of weights (here, we do not binarize the bias involved at each layer, only the weights to ± 1). Table 5 shows that our assumption is experimentally proven accurate. Binarization of weights while dealing with vectors in high dimensions does not change the relative angular positions significantly; hence, the accuracy does not drop significantly either. This modification could lead to a far more efficient algorithm that cuts down on memory and energy consumption, as well as the area occupied by a chip, while not sacrificing performance. Noteworthily, on KMNIST 10 and Fashion MNIST 10 datasets, SPELA (with ± 1 weights) performs better than an equivalent backpropagation trained network (MNIST 10:91.04% vs. 93.57%, KMNIST 10: 68.69% vs. 67.98%, Fashion MNIST 10: 84.06% vs. 76.38%).

Learning rate dependence of SPELA Next, we establish the learning rate dependence of SPELA's performance after 200 training epochs. Accordingly, Figure 7 and Table 17 describe the variation of test accuracies on MNIST 10, KMNIST 10, and Fashion MNIST 10 for learning rates of 0.01, 0.1, 1, 1.5, 2.5, and 3. We observe that MNIST 10 performs best (94.25%) for a learning rate of 2.5, KMNIST 10 (75.69%) for a learning rate of 3, and Fashion MNIST 10 (85.34%) for a learning rate of 0.1.

Table 22 provides the relevant experiment details for the above ablation studies.

Model	Architecture	MNIST 10	KMNIST 10	Fashion MNIST 10
SPELA-Cos	$784 \to 1024$	91.09 ± 0.12	68.74 ± 0.18	84 ± 0.11
SPELA-Cos	$784 \to 1024 \to 10$	94.41 ± 0.49	75.05 ± 3.47	84.12 ± 3.30
SPELA-Arccos	$784 \to 1024$	91.03 ± 0.06	68.39 ± 0.12	83.85 ± 0.08
SPELA-Arccos	$784 \to 1024 \to 10$	94.39 ± 0.91	75.44 ± 2.01	85.1 ± 0.71
SPELA-Euc	$784 \to 1024$	75.37 ± 0.34	52.71 ± 0.28	67.83 ± 0.77
SPELA-Euc	$784 \to 1024 \to 10$	48.16 ± 8.76	28.16 ± 8.14	51.82 ± 7.18
SPELA-RandNorm	$784 \to 1024$	90.91 ± 0.08	68.29 ± 0.21	84.03 ± 0.09
SPELA-RandNorm	$784 \to 1024 \to 10$	86.7 ± 9.76	68.4 ± 6.97	79.89 ± 6.25
SPELA-RandUnif	$784 \to 1024$	90.27 ± 0.16	67.37 ± 0.34	83.68 ± 0.04
SPELA-RandUnif	$784 \to 1024 \to 10$	84.12 ± 5.17	65.45 ± 4.69	73.08 ± 7.28
SPELA-Bin	$784 \to 1024$	91.04 ± 0.09	68.69 ± 0.17	84.06 ± 0.04
SPELA-Bin	$784 \to 1024 \to 10$	79.39 ± 12.67	61.80 ± 4.02	74.97 ± 4.09
BP-Bin	$784 \to 1024 \to 10$	93.57 ± 0.83	67.98 ± 4.80	76.38 ± 4.09

Table 5: Ablation study results of SPELA on MNIST 10, KMNIST 10, and FashionMNIST 10 datasets after 200 training epochs (and for five runs). SPELA-Cos implies SPELA with cosine distance, SPELA-Arccos implies SPELA with Arc cosine distance, SPELA-Euc implies SPELA with Euclidean distance, SPELA-RandNorm implies SPELA with random vectors from a normal distribution, SPELA-RandUnif implies SPELA with random vectors drawn from a uniform distribution, SPELA-Bin/BP-Bin implies with ± 1 weights.



(a) MNIST 10 (b) KMNIST 10 (c) Fashion MNIST 10

Figure 7: Variation of SPELA test accuracies (after 200 epochs of training) for learning rates of 0.01, 0.1, 1, 1.5, 2.5, and 3. Solid lines denote mean accuracies, and shades denote standard deviation over five runs.

4.5 SPELA Convolutional Neural Network

We evaluate the performance of our SPELA convolutional neural network (CNN) on image classification tasks. Accordingly, we here enlist the performance of SPELA CNN on complex datasets such as CIFAR 10, CIFAR 100, and SVHN 10. Table 23 summarizes the details of our experiments. Similar to Dellaferrera & Kreiman (2022); Liao et al. (2016), we compare the relative performance of SPELA CNN to reported results on previously proposed backpropagation alternatives (Table 23). Table 7 aims to showcase that our SPELA CNN can match these performances with only local learning and early exit capabilities. Of course, the performance of the CNN will improve by adding a global error akin to backpropagation. Although SPELA doesn't require a global error signal (either through backpropagation or as a second forward pass), a one-layer SPELA CNN attains a best performance of 45.21%. This is 5.21% lower than DRTP. Adding another layer to SPELA CNN improves performance to 56.33%, similar to a one-layer PEPITA. On the SVHN 10 dataset, a two-layer SPELA CNN archives a mean accuracy of 79.02%, which is a significant improvement over a vanilla SPELA performance on SVHN 10 (66.89%, Table 7).

Model	**CIFAR 10**	**CIFAR 100**	**SVHN 10**
BP	64.99 ± 0.32	34.20 ± 0.20	-
FA	57.51 ± 0.57	27.15 ± 0.53	-
DRTP	50.53 ± 0.81	20.14 ± 0.68	-
PEPITA	56.33 ± 1.35	27.56 ± 0.60	-
SPELA_CH_B CNN(1)	45.21 ± 12.67	19.62 ± 5.19	70.10 ± 16.68
SPELA_CH_B CNN(2)	52.4 ± 6.98	21.43 ± 5.94	79.02 ± 1.01
SPELA_B CNN(1)	44.35 ± 13.48	22.08 ± 4.53	78.14 ± 0.71
SPELA_B CNN(2)	56.59 ± 0.97	26.71 ± 0.79	76.28 ± 6.45

Table 6: Test accuracies (mean ± standard deviation) comparison of different CNN architectures on CIFAR 10, CIFAR 100, and SVHN 10 datasets. The accuracies of BP, FA, DRTP, and PEPITA are represented from Dellaferrera & Kreiman (2022). We report both hidden layer and output mean accuracies (average of five runs) of a SPELA convolutional neural network (CNN). SPELA CNN(1) and SPELA CNN(2) imply a network with a 1-layer and 2-layer CNN, respectively. BP, FA, DRTP, and PEPITA results are presented as the best-case scenario, which SPELA attempts to match under the constraint of equivalent CNN layers.

5 Discussion

We propose SPELA as a computationally efficient learning algorithm to train neural networks. For the experiments conducted, SPELA has a feature set comprising symmetric embedded vectors, local learning, early exit, and a single forward pass for training with no storage of activations, no weight transport, and no updated weight locking. As part of it, we perform detailed experiments to benchmark SPELA against existing works. In addition, we analyzed its suitability for transfer learning on backpropagation-trained image recognition networks. We perform ablation studies on SPELA to justify the design choices. Finally, we extend SPELA to convolutional neural networks (CNN) and benchmark it on equivalent image recognition tasks. Analyzing theoretical complexity (lower bound) and on-device implementation shows that SPELA is more efficient than backpropagation regarding memory utilization. Hence, this work can help guide the implementation of on-device learning in tiny microcontrollers such as ARM Cortex-M devices. Overall, we believe SPELA is helpful for a wide range of ML applications, wherein we care about training and testing efficiency regarding accuracy and memory. To conclude, we view this work as a starting point for developing efficient learning algorithms that can aid in applications where backpropagation presently has limitations. In the future, we will extend SPELA to the training and inference of advanced architectures such as deep convolutional neural networks and transformers. Although SPELA is biologically inspired, it lacks some key features in biology, such as the spiking behavior of neurons. Further work is necessary to integrate such features with SPELA.

References

Mohamed Akrout, Collin Wilson, Peter Humphreys, Timothy Lillicrap, and Douglas B Tweed. Deep learning without weight transport. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper_files/paper/2019/file/f387624df552cea2f369918c5e1e12bc-Paper.pdf.

Alexander G. Anderson and Cory P. Berg. The high-dimensional geometry of binary neural networks. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=B1IDRdeCW.

Han Cai, Chuang Gan, Ligeng Zhu, and Song Han. Tinytl: Reduce memory, not parameters for efficient on-device learning. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 11285–11297. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/81f7acabd411274fcf65ce2070ed568a-Paper.pdf.

Daniel Cer, Yinfei Yang, Sheng yi Kong, Nan Hua, Nicole Limtiaco, Rhomni St. John, Noah Constant, Mario Guajardo-Cespedes, Steve Yuan, Chris Tar, Yun-Hsuan Sung, Brian Strope, and Ray Kurzweil. Universal sentence encoder, 2018. URL https://arxiv.org/abs/1803.11175.

Xing Chen, Dongshu Liu, Jeremie Laydevant, and Julie Grollier. Self-contrastive forward-forward algorithm, 2024. URL https://arxiv.org/abs/2409.11593.

Henry Cohn and Abhinav Kumar. Universally optimal distribution of points on spheres. *Journal of the American Mathematical Society*, 20(1):99–148, 2007.

Wojciech Marian Czarnecki, Grzegorz Swirszcz, Max Jaderberg, Simon Osindero, Oriol Vinyals, and Koray Kavukcuoglu. Understanding synthetic gradients and decoupled neural interfaces. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, pp. 904–912. JMLR.org, 2017.

Giorgia Dellaferrera and Gabriel Kreiman. Error-driven input modulation: Solving the credit assignment problem without a backward pass. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato (eds.), *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 4937–4955. PMLR, 17–23 Jul 2022. URL https://proceedings.mlr.press/v162/dellaferrera22a.html.

Thomas Dooms, Ing Jyh Tsang, and Jose Oramas. The trifecta: Three simple techniques for training deeper forward-forward networks. *Transactions on Machine Learning Research*, 2024. ISSN 2835-8856. URL https://openreview.net/forum?id=a7KP5uo0Fp.

Fabio Giampaolo, Stefano Izzo, Edoardo Prezioso, and Francesco Piccialli. Investigating random variations of the forward-forward algorithm for training neural networks. In *2023 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–7, 2023. doi: 10.1109/IJCNN54540.2023.10191727.

Anirudh Goyal and Yoshua Bengio. Inductive biases for deep learning of higher-level cognition. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 478(2266):20210068, 2022. doi: 10.1098/rspa.2021.0068. URL https://royalsocietypublishing.org/doi/abs/10.1098/rspa.2021.0068.

Geoffrey Hinton. The forward-forward algorithm: Some preliminary investigations, 2022.

Bernd Illing, Jean Robin Ventura, Guillaume Bellec, and Wulfram Gerstner. Local plasticity rules can learn deep representations using self-supervised contrastive predictions. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=Yu8Q6341U7W.

Max Jaderberg, Wojciech Marian Czarnecki, Simon Osindero, Oriol Vinyals, Alex Graves, David Silver, and Koray Kavukcuoglu. Decoupled neural interfaces using synthetic gradients. In Doina Precup and Yee Whye Teh (eds.), *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pp. 1627–1635. PMLR, 06–11 Aug 2017. URL https://proceedings.mlr.press/v70/jaderberg17a.html.

Benjamin James Lansdell, Prashanth Ravi Prakash, and Konrad Paul Kording. Learning to solve the credit assignment problem. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=ByeUBANtvB.

Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, 1989. doi: 10.1162/neco.1989.1.4.541.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, May 2015. ISSN 1476-4687. doi: 10.1038/nature14539. URL https://doi.org/10.1038/nature14539.

Heung-Chang Lee and Jeonggeun Song. Symba: Symmetric backpropagation-free contrastive learning with forward-forward algorithm for optimizing convergence, 2023.

Qinyu Li, Yee Whye Teh, and Razvan Pascanu. Noprop: Training neural networks without back-propagation or forward-propagation, 2025. URL https://arxiv.org/abs/2503.24322.

Qianli Liao, Joel Z. Leibo, and Tomaso Poggio. How important is weight symmetry in backpropagation? In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI'16, pp. 1837–1844. AAAI Press, 2016.

Timothy P. Lillicrap, Daniel Cownden, Douglas B. Tweed, and Colin J. Akerman. Random feedback weights support learning in deep neural networks, 2014.

Changze Lv, Jingwen Xu, Yiyang Lu, Xiaohua Wang, Zhenghua Wang, Zhibo Xu, Di Yu, Xin Du, Xiaoqing Zheng, and Xuanjing Huang. Dendritic localized learning: Toward biologically plausible algorithm, 2025. URL https://arxiv.org/abs/2501.09976.

Ali Momeni, Babak Rahmani, Matthieu Malléjac, Philipp del Hougne, and Romain Fleury. Backpropagation-free training of deep physical neural networks. *Science*, 382(6676):1297–1303, 2023. doi: 10.1126/science.adi8474. URL https://www.science.org/doi/abs/10.1126/science.adi8474.

Namyong Park, Xing Wang, Antoine Simoulin, Shuai Yang, Grey Yang, Ryan A. Rossi, Puja Trivedi, and Nesreen K. Ahmed. Forward learning of graph neural networks. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=Abr7dU98ME.

Danilo Pietro Pau and Fabrizio Maria Aymone. Suitability of forward-forward and pepita learning to mlcommons-tiny benchmarks. In *2023 IEEE International Conference on Omni-layer Intelligent Systems (COINS)*, pp. 1–6, 2023. doi: 10.1109/COINS57856.2023.10189239.

Cengiz Pehlevan. A spiking neural network with local learning rules derived from nonnegative similarity matching. In *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 7958–7962, 2019. doi: 10.1109/ICASSP.2019.8682290.

David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, Oct 1986. ISSN 1476-4687. doi: 10.1038/323533a0. URL https://doi.org/10.1038/323533a0.

Edward B Saff and Amo BJ Kuijlaars. Distributing many points on a sphere. *The mathematical intelligencer*, 19:5–11, 1997.

Simone Scardapane, Michele Scarpiniti, Enzo Baccarelli, and Aurelio Uncini. Why should we add early exits to neural networks? *Cognitive Computation*, 12(5):954–966, June 2020. ISSN 1866-9964. doi: 10.1007/s12559-020-09734-4. URL http://dx.doi.org/10.1007/s12559-020-09734-4.

Ravi Srinivasan, Francesca Mignacco, Martino Sorbaro, Maria Refinetti, Avi Cooper, Gabriel Kreiman, and Giorgia Dellaferrera. Forward learning with top-down feedback: Empirical and analytical characterization, 2024.

Zihao Wang and Lei Wu. Theoretical analysis of the inductive biases in deep convolutional networks. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=N0KwVdaaaJ.

P.J. Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78 (10):1550–1560, 1990. doi: 10.1109/5.58337.

James C.R. Whittington and Rafal Bogacz. Theories of error back-propagation in the brain. *Trends in Cognitive Sciences*, 23(3):235–250, 2019. ISSN 1364-6613. doi: https://doi.org/10.1016/j.tics.2018.12.005. URL https://www.sciencedirect.com/science/article/pii/S1364661319300129.

Shuai Zhu, Thiemo Voigt, JeongGil Ko, and Fatemeh Rahimian. On-device training: A first overview on existing systems, 2023.

A Additional Methods

A.1 Electron simulation description

Figuring out N points on an evenly distributed circle is simple: dividing the circle into equal arcs and placing the points at their endpoints. For higher-dimensional spheres ($l_i \geq 3$), the problem is non-trivial and does not admit a closed-form solution. We use a well-known method of electron approximation to simulate N electrons in any D-dimensional space. We constrain each electron to a unit-radius ball and simulate the relative forces among electrons (Saff & Kuijlaars, 1997; Cohn & Kumar, 2007). At each iteration, we move each electron according to the direction and magnitude of the net force, respecting the unit-ball constraint. We define stability in terms of relative change in electrostatic potential. As the electrons shift around the ball, the electrostatic energy changes. As the electrons redistribute, their movements occur over progressively smaller distances. The relative change in electrostatic energy tends to zero with iterations. Once this relative change is less than a defined threshold, we claim to have a stable configuration and obtain an l_i-dimensional ball with a symmetric distribution of N electrons (see Fig. 1a).

A.2 Algorithms

Algorithm 3 Inference on MLP trained with SPELA

Given: An input (X) and number of layers K
Define: $\text{cos\_sim}(A, B) = \frac{A.B}{||A||.||B||}$
Set: $h_0 = x$
for $k \leftarrow 1$ to K **do** ▷ Passing data through all the layers
    $h_k = \sigma_k(W_k h_{k-1} + b_k)$
end for
for $i \leftarrow 1$ to N **do** ▷ N is the number of classes
    $\text{S}_i = \text{cos\_sim}(h_K, \text{vecs}(i))$ ▷ Similarity between activation vector and symmetric vectors
end for
Prediction: $\arg\max_i \text{S}_i$ ▷ Class corresponding to the maximum score is prediction

B Additional Results

B.1 How does SPELA work?

Model	Architecture	# Epochs	CIFAR 10	CIFAR 100	SVHN 10
BP	$3072 \rightarrow 1024 \rightarrow 10$	100	53.48 ± 0.36	27 ± 0.21	77.83 ± 0.48
FA	$3072 \rightarrow 1024 \rightarrow 10$/ $3072 \rightarrow 1024 \rightarrow 100$	100	53.82 ± 0.24	24.61 ± 0.28	-
DRTP	$3072 \rightarrow 1024 \rightarrow 10$/ $3072 \rightarrow 1024 \rightarrow 100$	100	45.89 ± 0.16	18.32 ± 0.18	-
PEPITA	$3072 \rightarrow 1024 \rightarrow 10$/ $3072 \rightarrow 1024 \rightarrow 100$	100	45.89 ± 0.16	18.32 ± 0.18	-
SPELA_B	$3072 \rightarrow 1024$	200	43.2 ± 0.18	19.77 ± 0.19	48.33 ± 5.
SPELA_B	$3072 \rightarrow 1024 \rightarrow 10$/ $3072 \rightarrow 1024 \rightarrow 100$	200	43.41 ± 1.03	21.24 ± 0.12	66.89 ± 2.09

Table 7: Test accuracies(mean ± standard deviation) comparison of different learning methods on CIFAR 10, CIFAR 100, and SVHN 10 datasets. The accuracies of FA, DRTP, and PEPITA are presented in Dellaferrera & Kreiman (2022). We report both hidden layer and output mean accuracies (average of five runs) of SPELA. For CIFAR 10 and SVHN 10, we use a $3072 \rightarrow 1000 \rightarrow 10$ network, and for CIFAR 100, we use a 3072-1024-100 network.

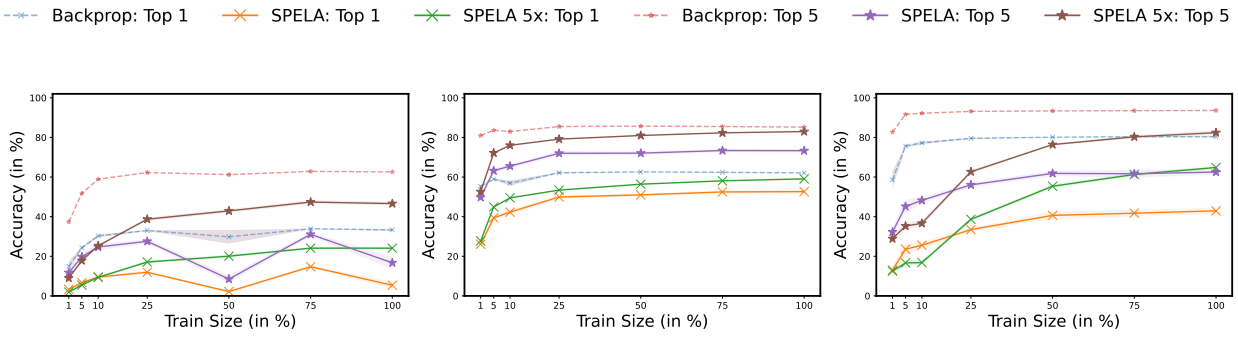
Dataset	a	b	c	d	e	f	g	h	i
MNIST 10	53.19 ± 4.73	85.31 ± 4.05	92.09 ± 3.87	95.56 ± 0.11	95.73 ± 0.13	95.55 ± 0.08	95.38 ± 0.07	94.46 ± 0.11	91.74 ± 0.22
KMNIST 10	35.66 ± 6.39	69.15 ± 2	73.67 ± 3.38	80.23 ± 0.18	79.98 ± 0.3	79.92 ± 0.18	78.96 ± 0.36	75.63 ± 0.42	68.44 ± 0.89
Fashion MNIST 10	50.31 ± 6.45	75.23 ± 9.1	84.88 ± 1.37	87 ± 0.05	86.82 ± 0.09	86.8 ± 0.06	86.66 ± 0.06	85.92 ± 0.12	84.33 ± 0.11

Table 8: Test accuracies (mean ± standard deviation) of SPELA for varying output layer depth, keeping the total neurons in the network fixed to 1034. The columns denote network configurations a: 1032-2, b: 1029-5, c: 1024-10, d: 984-50, e: 934-100, f: 834-200, g: 634-400, h: 234-800, i: 34-1000. The SPELA_B configuration is used throughout the experiments for 200 training epochs and five runs.

Model	# 1	# 2	# 3	# 4	# 5	# 6	# 7	# 8	# 9
BP	3.26	7.46	11.65	15.85	21.03	25.23	29.43	33.63	37.83
SPELA	3.30	7.54	11.78	16.02	21.24	25.48	29.72	33.96	38.20

Table 9: Model memory(in MB) occupied by a BP model and an equivalent SPELA model for the number of hidden layers varying from 1-9 (each of size 1024 neurons). The input and output layer sizes are 784 and 10, respectively.

B.2 Transfer Learning with SPELA



(b) Aircraft 100 dataset (c) Food 101 dataset (d) Flowers 102 dataset

Figure 8: Accuracy plots (after 200 epochs of fine tuning) of SPELA, SPELA 5x and Backpropagation trained networks for train dataset size percentages of 1, 5, 10, 25, 50, 75, and 100 during transfer learning(keeping the test dataset fixed). The solid lines denote the mean, and the shades denote the standard deviation of five simulation runs.

In these experiments, the networks are trained for 200 epochs with a learning rate of 0.1, and analysis is done on six datasets: Aircraft 100, CIFAR 10, CIFAR 100, Flowers 102, Food 101, and Pets 37 datasets (the numbers denote the number of classes in that dataset). Note that most of these datasets have a large number of classes. Table 21 describes the experimental details. A ResNet50 model pre-trained on ImageNet-1000(downloaded from PyTorch Hub) extracts features from the layer before the classifier head. These features then train a classifier head using backpropagation or SPELA.

Test size:	**1**	**5**	**10**	**25**	**50**	**75**	**100**
SPELA top1	58.79 ± 19.33	67.28 ± 4.50	64.03 ± 6.47	69.67 ± 5.10	72.38 ± 2.80	73.35 ± 3.17	69.32 ± 8.29
SPELA top5	90.23 ± 10.89	95.01 ± 0.69	93.84 ± 1.67	95.14 ± 1.55	95.68 ± 0.52	96.53 ± 0.20	94.78 ± 1.70
SPELA 5x top1	75.64 ± 0.54	78.64 ± 0.53	75.50 ± 4.97	81.21 ± 0.16	80.88 ± 0.73	82.11 ± 0.32	82.04 ± 0.50
SPELA 5x top5	97.81 ± 0.28	98.13 ± 0.24	97.84 ± 0.33	98.36 ± 0.16	98.22 ± 0.34	98.43 ± 0.05	98.20 ± 0.24
BP top1	75.93 ± 2.13	76.58 ± 2.18	67.22 ± 8.64	81.18 ± 0.80	80.47 ± 1.87	81.57 ± 0.27	81.18 ± 0.77
BP top5	98.25 ± 0.77	97.96 ± 1.70	96.99 ± 2.40	99.15 ± 0.07	99.06 ± 0.18	99.12 ± 0.12	99.12 ± 0.11

Table 10: Test accuracy for CIFAR 10 dataset.

Test size:	**1**	**5**	**10**	**25**	**50**	**75**	**100**
SPELA top1	26.35 ± 1.27	43.22 ± 0.91	42.97 ± 2.25	51.14 ± 1.37	51.97 ± 0.92	54.52 ± 0.54	54.35 ± 1.03
SPELA top5	53.14 ± 2.19	69.50 ± 1.41	68.71 ± 2.34	75.53 ± 1.15	75.40 ± 0.91	76.88 ± 0.85	76.62 ± 1.40
SPELA 5x top1	26.33 ± 0.96	47.26 ± 0.30	51.58 ± 0.26	55.81 ± 0.09	57.63 ± 0.19	59.07 ± 0.30	59.87 ± 0.26
SPELA 5x top5	55.74 ± 0.97	77.97 ± 0.31	81.04 ± 0.27	83.69 ± 0.12	84.80 ± 0.18	85.57 ± 0.06	85.95 ± 0.30
BP top1	55.00 ± 0.43	59.19 ± 0.40	46.16 ± 2.86	61.18 ± 0.09	60.79 ± 0.48	61.42 ± 0.20	61.08 ± 0.13
BP top5	84.05 ± 0.32	86.77 ± 0.25	81.79 ± 1.24	87.55 ± 0.09	87.40 ± 0.20	87.61 ± 0.07	87.46 ± 0.04

Table 11: Test accuracy for CIFAR 100 dataset.

Test size:	**1**	**5**	**10**	**25**	**50**	**75**	**100**
SPELA top1	66.54 ± 4.52	76.86 ± 1.57	78.60 ± 2.61	82.93 ± 1.99	83.48 ± 1.33	84.94 ± 1.96	84.60 ± 1.46
SPELA top5	89.89 ± 1.34	94.28 ± 1.41	94.33 ± 0.98	95.55 ± 0.57	95.16 ± 0.34	95.37 ± 1.24	95.34 ± 0.91
SPELA 5x top1	75.96 ± 1.34	83.73 ± 0.37	85.71 ± 0.66	88.08 ± 0.42	88.93 ± 0.11	89.19 ± 0.29	89.34 ± 0.20
SPELA 5x top5	95.97 ± 0.45	97.85 ± 0.23	98.34 ± 0.21	98.53 ± 0.07	98.57 ± 0.06	98.61 ± 0.09	98.48 ± 0.17
BP top1	86.71 ± 0.82	88.64 ± 0.44	88.99 ± 0.16	89.28 ± 0.23	89.26 ± 0.08	89.44 ± 0.25	89.29 ± 0.10
BP top5	98.79 ± 0.11	99.03 ± 0.06	99.02 ± 0.08	99.01 ± 0.09	98.99 ± 0.04	98.90 ± 0.05	98.88 ± 0.04

Table 12: Test accuracy for Pets 37 dataset.

Test size:	**1**	**5**	**10**	**25**	**50**	**75**	**100**
SPELA top1	3.32 ± 0.43	6.68 ± 0.54	9.53 ± 0.92	11.94 ± 0.70	2.18 ± 0.64	14.74 ± 0.60	5.38 ± 1.67
SPELA top5	11.76 ± 1.13	19.77 ± 1.63	24.70 ± 1.60	27.55 ± 1.15	8.51 ± 1.87	31.08 ± 0.94	16.72 ± 3.14
SPELA 5x top1	1.97 ± 0.41	5.47 ± 0.53	9.39 ± 0.49	17.10 ± 0.72	20.06 ± 0.87	24.11 ± 0.45	24.09 ± 0.64
SPELA 5x top5	9.00 ± 0.57	17.83 ± 0.67	25.31 ± 0.33	38.72 ± 0.98	42.92 ± 0.94	47.39 ± 1.01	46.61 ± 0.83
BP top1	14.94 ± 2.46	24.29 ± 0.56	30.29 ± 0.72	33.01 ± 0.25	29.82 ± 3.41	33.89 ± 0.16	33.32 ± 0.49
BP top5	37.38 ± 1.94	51.80 ± 0.74	58.93 ± 0.43	62.26 ± 0.20	61.22 ± 1.28	62.87 ± 0.35	62.59 ± 0.41

Table 13: Test accuracy for Aircraft 100 dataset.

Test size:	**1**	**5**	**10**	**25**	**50**	**75**	**100**
SPELA top1	26.09 ± 1.72	39.43 ± 0.91	42.28 ± 1.24	49.93 ± 0.89	50.99 ± 0.73	52.48 ± 0.77	52.65 ± 0.96
SPELA top5	49.77 ± 1.61	63.19 ± 1.15	65.53 ± 0.93	71.99 ± 1.25	72.06 ± 0.82	73.38 ± 0.83	73.32 ± 0.91
SPELA 5x top1	27.77 ± 0.45	44.89 ± 0.36	49.50 ± 0.23	53.41 ± 0.23	56.39 ± 0.05	58.10 ± 0.11	59.08 ± 0.21
SPELA 5x top5	52.58 ± 0.84	72.18 ± 0.22	76.02 ± 0.30	79.15 ± 0.21	80.98 ± 0.22	82.32 ± 0.13	82.96 ± 0.16
BP top1	55.34 ± 0.36	58.97 ± 0.34	56.93 ± 1.30	62.16 ± 0.29	62.61 ± 0.08	62.40 ± 0.16	62.09 ± 0.10
BP top5	80.95 ± 0.26	83.66 ± 0.24	82.96 ± 0.72	85.49 ± 0.07	85.71 ± 0.09	85.51 ± 0.16	85.21 ± 0.09

Table 14: Test accuracy for Food 101 dataset.

Test size:	1	5	10	25	50	75	100
SPELA top1	13.07 ± 2.23	23.58 ± 3.15	25.56 ± 1.81	33.48 ± 0.86	40.65 ± 1.92	41.75 ± 1.58	42.95 ± 1.62
SPELA top5	32.28 ± 2.12	45.18 ± 3.42	48.23 ± 1.62	56.06 ± 1.22	61.84 ± 1.38	61.68 ± 2.01	62.51 ± 1.29
SPELA 5x top1	12.52 ± 1.24	16.65 ± 1.38	16.82 ± 1.20	38.65 ± 0.88	55.37 ± 1.08	61.29 ± 0.80	64.82 ± 0.82
SPELA 5x top5	28.89 ± 2.37	35.27 ± 1.77	36.65 ± 1.52	62.67 ± 0.87	76.39 ± 0.43	80.31 ± 0.85	82.40 ± 1.31
BP top1	58.61 ± 4.33	75.62 ± 0.73	77.24 ± 0.68	79.53 ± 0.32	80.12 ± 0.10	80.39 ± 0.07	80.34 ± 0.27
BP top5	82.72 ± 2.03	91.64 ± 0.33	92.21 ± 0.30	93.16 ± 0.26	93.39 ± 0.28	93.52 ± 0.05	93.61 ± 0.10

Table 15: Test accuracy for Flowers 102 dataset.

B.3 Extension of Ablation Studies

B.3.1 Relevance of Euclidean distance

The goal of learning in SPELA is to learn to orient the activation vector h_i towards the correct class c symmetric vector v_c. This is akin to reducing the angle between h_i and v_c, which in turn increases the cosine similarity or angular similarity. Hence, cosine is an appropriate similarity.

If, for instance, $h_i = 100v_c$, then even though the cosine loss would be zero due to perfectly aligned vectors, the Euclidean distance would be high, and hence the corresponding loss would be high. We notice that in such a scenario, SPELA's performance drops significantly, SPELA-Euc in Table 16. But if we use normalized Euclidean distance wherein the vectors are $\frac{h_i}{||h_i||_2}$ and $\frac{v_c}{||v_c||_2}$, then the performance of SPELA closely matches that of cosine, SPELA-Norm-Euc in Table 16.

Model	Architecture	MNIST 10	KMNIST 10	Fashion MNIST 10
SPELA-Cos	$784 \to 1024$	91.09 ± 0.12	68.74 ± 0.18	84 ± 0.11
SPELA-Cos	$784 \to 1024 \to 10$	94.41 ± 0.49	75.05 ± 3.47	84.12 ± 3.30
SPELA-Euc	$784 \to 1024$	75.37 ± 0.34	52.71 ± 0.28	67.83 ± 0.77
SPELA-Euc	$784 \to 1024 \to 10$	48.16 ± 8.76	28.16 ± 8.14	51.82 ± 7.18
SPELA-Norm-Euc	$784 \to 1024$	90.24 ± 0.11	67.11 ± 0.13	83.53 ± 0.09
SPELA-Norm-Euc	$784 \to 1024 \to 10$	90.0 ± 3.64	71.57 ± 2.2	84.60 ± 1.14

Table 16: Ablation study results of SPELA on MNIST 10, KMNIST 10, and FashionMNIST 10 datasets after 200 training epochs (and for five runs). SPELA-Cos implies SPELA with cosine distance, SPELA-Euc implies SPELA with Euclidean distance, SPELA-Norm-Euc implies SPELA with Euclidean distance on normalized activation and symmetric vectors.

B.3.2 Remarks on randomizing the Vector Embeddings

Remark 1: As we operate in dimensions much higher than the number of embedded vectors (number of classes), a non-symmetric distribution should perform equivalent to a symmetric structure. The performance gap between symmetric and non-symmetric structures would be noticeable when the number of dimensions exceeds the number of classes.

Remark 2: We use the energy of the system as a structured metric:

$$\lambda(\mathcal{V}) = \sum_{\mathbf{u} \in \mathcal{V}} \sum_{\mathbf{v} \in \mathcal{V}, \mathbf{u} \neq \mathbf{v}} \frac{1}{\|\mathbf{u} - \mathbf{v}\|}$$

Of all possible vectors $\mathbf{x} \in \mathbb{R}^d$, if $|\mathcal{V}|$ is fixed, the symmetric structure has the minimum energy. Comparing vectors drawn from the Gaussian distribution $\mathcal{N}(\mathbf{0}, \mathbf{1})$ and the symmetric structure, we get a high energy difference. Despite this, the network learns from the incoming data. A symmetric structure is necessary for lower dimensions (where the number of dimensions is comparable to the number of embedded vectors).

We learn that although having vectors embedded in a symmetric structure is optimal, it is unnecessary. The model will mold the weights according to the relative positions of the vectors and classify the data accordingly, which ascertains the model's flexibility.

Dataset	Architecture	0.01	0.1	1	1.5	2.5	3
MNIST 10	784 → 1024	90.27 ± 1.1	90.97 ± 0.12	89.72 ± 1.01	91.26 ± 0.05	91.01 ± 0.12	90.97 ± 0.06
MNIST 10	784 → 1024 → 10	91.86 ± 3.22	92.92 ± 1.63	86.93 ± 11.06	93.31 ± 1	94.25 ± 1.12	92.31 ± 3.28
KMNIST 10	784 → 1024	65.74 ± 2.77	68.54 ± 0.18	64.82 ± 3.23	67.9 ± 0.5	68.48 ± 0.11	68.6 ± 0.17
KMNIST 10	784 → 1024 → 10	75.06 ± 2.47	74.30 ± 3.38	73.86 ±3.81	66.11 ±3.57	74.12 ± 2.41	75.69 ± 2.03
Fashion MNIST 10	784 → 1024	81.57 ± 2.11	83.97 ±0.06	81.10 ± 2.35	83.34 ± 0.57	83.99 ± 0.06	83.96 ± 0.05
Fashion MNIST 10	784 → 1024 → 10	83.12 ± 4.51	85.34 ± 0.21	81.32 ± 3.28	83.65 ± 1.61	84.52 ± 1.66	81.6 ± 4.99

Table 17: SPELA test accuracies (after 200 epochs of training and over five runs) for learning rates of 0.01, 0.1, 1, 1.5, 2.5, and 3.

B.4 Extension of SPELA Convolutional Neural Network

We next varied the channels in SPELA_B_CNN from 32 to 128 in SPELA_C_CNN and observed that SPELA improves performance on CIFAR 10, CIFAR 100, as well as SVHN 10 (Table 18).

Model	CIFAR 10	CIFAR 100	SVHN 10
SPELA_B CNN(2)	56.59 ± 0.97	26.71 ± 0.79	76.28 ± 6.45
SPELA_C CNN(2)	64.76 ± 0.49	27.59 ± 4.7	84.93 ± 0.21

Table 18: Test accuracies (mean ± standard deviation) comparison of different SPELA CNN architectures on CIFAR 10, CIFAR 100, and SVHN 10 datasets. SPELA_B and SPELA_C indicate networks with 32 and 128 channels, respectively(Please refer to Section C.4 for experimental details).

C Experimental Details

We used an NVIDIA RTX 4500 Ada generation GPU for all our studies.

C.1 How does SPELA work?

	BP_CH_A	**SPELA_CH_A**	**SPELA_A**
Layer 1	1024	1024	1024
Layer 2	10	10	10
Learning rate	0.1	0.1	0.1
Decay rate	$\times$ 0.1	$\times$ 0.1	$\times$ 0.1
Decay epoch	60	60	60
Batch size	50	50	50
# Epochs	100	100	100
Dropout	10%	10%	10%
Weight init	He Uniform	He Uniform	He Uniform
Bias	False	False	False
Optimizer	SGD (momentum=0.9)	SGD	SGD
Activation	ReLU	ReLU	ReLU
Loss	Cross-Entropy	Cross-Entropy	Positive Cosine

Table 19: Experiment Details for SPELA MLP

	BP_CH_B	**SPELA_CH_B**	**SPELA_B**
Layer 1	1024	1024	1024
Layer 2	10	10	10
Learning rate	2.5	2.5	2.5
Decay rate	$-$ 0.1	$-$ 0.1	$-$ 0.1
Decay epoch	Every 10	Every 10	Every 10
Batch size	50	50	50
# Epochs	200	200	200
Dropout	0	0	0
Weight init	He Uniform	He Uniform	He Uniform
Bias	True	True	True
Optimizer	SGD	SGD	SGD
Activation	Leaky ReLU(Slope=0.001)	Leaky ReLU(Slope=0.001)	Leaky ReLU(Slope=0.001)
Loss	Cross-Entropy	Cross-Entropy	Positive Cosine

Table 20: Experiment Details for SPELA MLP

C.2 Transfer Learning with SPELA

	BP	**SPELA**	**SPELA 5x**
Layer 1	#Classes	#Classes	5 × #Classes
Learning rate	0.1	0.1	0.1
Decay rate	0	0	0
Batch size	128	128	128
# Epochs	200	200	200
Dropout	0	0	0
Weight init	He Uniform	He Uniform	He Uniform
Bias	True	True	True
Optimizer	SGD	SGD	SGD
Loss	Cross-Entropy	Cross-Entropy	Cross-Entropy

Table 21: Experiment Details of SPELA on Transfer Learning

C.3 Ablation Studies

	BP	**SPELA**
Layer 1	1024	1024
Layer 2	10	10
Learning rate	2.5	2.5
Decay rate	− 0.1	− 0.1
Decay epoch	Every 10	Every 10
Batch size	50	50
# Epochs	200	200
Dropout	0	0
Weight init	He Uniform	He Uniform
Bias	True	True
Optimizer	SGD	SGD
Activation	Leaky ReLU(Slope=0.001)	Leaky ReLU(Slope=0.001)
Loss	Cross-Entropy	Positive Cosine

Table 22: Experiment Details of SPELA MLP for Ablation Studies

C.4 SPELA Convolutional Neural Network

	SPELA_CH_B	**SPELA_B**
Input size	32×32×3	32×32×3
Conv	32,5,1(2)	32,5,1(2)
MLP	10/100	10/100
Learning rate	0.1, 0.1	0.1, 0.1
Decay rate	0	0
Batch size	64	64
# Epochs	15+10	15+10
Dropout	0	0
Weight init	Kaiming Uniform	Kaiming Uniform
Bias	True	True
Optimizer	Adam	Adam
Activation	PReLU,Leaky ReLU(Slope=0.001)	PReLU,Leaky ReLU(Slope=0.001)
Loss	Cross-Entropy	Positive Cosine

Table 23: Experimental details of SPELA convolutional neural network.