Learning Using a Single Forward Pass

Anonymous authors
Paper under double-blind review

Abstract

We propose a learning algorithm to overcome the limitations of a traditional backpropagation in resource-constrained environments: Solo Pass Embedded Learning Algorithm (SPELA). SPELA is equipped with rapid learning capabilities and 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 with less data, computing, storage, and power. 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 for equivalent performance on CIFAR-10, CIFAR-100, and SVHN 10 datasets. Our results indicate that SPELA can be an ideal candidate for learning in resource-constrained edge AI applications.

1 Introduction

Backpropagation (BP) is a long-standing and the most fundamental algorithm for training deep neural networks (NNs) (Werbos (1990); Rumelhart et al. (1986); LeCun et al. (1989; 2015)). It has wide applications in the form of multi-layer perceptrons (MLP), convolutional neural networks (CNN), recurrent neural networks (RNN), and now transformers. Backpropagation 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 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 on 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 to 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, preventing 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 data requirements, training time, memory requirements, power consumption, and area occupied by the neural network hardware. These limitations render backpropagation computationally expensive and unsuitable for applications with resource constraint scenarios such as fewer train data, small memory, or less energy (example: on-device machine learning(ODL) Cai et al. (2020); Zhu et al. (2023)). An effort to mitigate these algorithmic constraints of backpropagation can lead to improved learning efficiency.

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 easier Goyal & Bengio (2022); Wang & Wu (2023). Although we don't 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 neuron layer size. Our back-propagation-free learning algorithm demonstrates a significant gain in computational efficiency and rapid(few-shot, few-epoch) learning capabilities, making it suitable for on-device learning (ODL) and exhibiting several learning features in the brain, 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 and its variant SPELA(O). Next, we extend SPELA to convolutional neural networks. SPELA is a family of algorithms that uses 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 while maintaining an edge over backpropagation in resource-constrained scenarios due to its computational efficiency. SPELA can learn rapidly (from a few examples within a few shots) on multiple datasets. Moreover, SPELA can efficiently fine-tune models trained with backpropagation(transfer learning). In addition, an extension of SPELA to CNN makes complex image classification possible.

- **Complexity Analysis:** Theoretic bounds for peak memory usage show that SPELA can edge over backpropagation in the analyzed settings. The vector-matrix multiplications needed for updating weights using SPELA are much lower than the standard backpropagation algorithm for these settings.

2 Related works

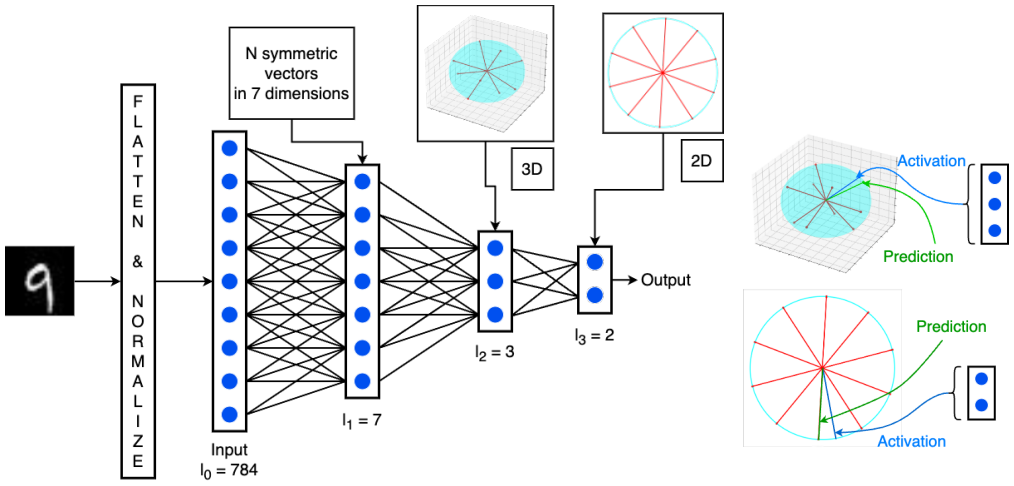
Recently, a significant interest has been in designing efficient training methods for multi-layer neural networks. Hinton (2022) presents the Forward-Forward (FF) algorithm for neural network learning, replacing backpropagation with two forward passes: one with positive (real) data and the other with 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 such as image recognition (Lee & Song (2023); Pau & Aymone (2023); Momeni et al. (2023); Dooms et al. (2024); Chen et al. (2024)) and extension to graph neural networks (Park et al. (2024)). In contrast to the FF approach, which aggregates goodness values across layers, SPELA can perform classification at any layer without storing goodness memory for the experiments conducted. 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 a forward and a backward pass, Pehlevan (2019) introduced the concept of 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) introduces a hybrid learning approach wherein each neuron learns to approximate the gradients. The learning feedback weights provide a biologically plausible way of achieving good performance compared to backpropagation-based trained networks. 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). Still using two forward passes, but instead of backward propagating, Dellaferrera & Kreiman (2022) used the global error to modulate the second forward pass input to train a neural network. We observe 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. Significantly, as detailed in Pau & Aymone (2023); Srinivasan et al. (2024), these algorithms still lack some 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. The weights of the network are initialized at random. 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)). The network outputs the class assigned to the symmetric vector closest to the activation vector with respect to cosine similarity. These symmetric vectors remain fixed and are not updated during training.



(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 Arranging the vector embeddings

To generate vector embeddings for the classification task, we adopt the most straightforward choice to allocate an equal portion of space to each vector as a starting point. Consequently, the vectors are arranged symmetrically on the unit norm ball in the n^{th} dimension (see Fig. 1a.).

Remark 1: The symmetric vectors are generated by simulating the physics of electrons in the n^{th} dimension by restricting their movement on the standard unit ball until the electrostatic energy converges sufficiently to a minimum value (Saff & Kuijlaars (1997); Cohn & Kumar (2007)). The resulting coordinates of these electrons are used as symmetric vectors.

Training: (Appendix Algorithm 3) Given data x and label y, the network learns locally by minimizing the cosine similarity loss between the activation vector (arising from input x) and the symmetric vector (which is assigned to the class y) in every layer. There is no batch size restriction (training and testing); multiple data points can be given before the weight update. Using this training method, the weights of the layers are updated sequentially. As mentioned earlier, this training method aligns perfectly with the Hebbian learning principle.

Inference: (Appendix Algorithm 4) Given data x, the network measures how close each symmetric vector is to the activation vector (arising from input x). It declares the closest symmetric vector as the prediction

vector. The prediction class is assigned to the corresponding symmetric vector (shown in Fig. 1b). SPELA can also perform classification from any of the layers in the network.

We next optimize SPELA (Appendix Algorithm 3) and design SPELA(O)(Algorithm 1), which reduces redundant forward passes (as detailed in 3.3).

3.3 Optimizing SPELA

SPELA trains models layer-wise, inherently, there is a caveat if Appendix Algorithm 3 is used: To train layer i, the data must be pushed through all previous layers. This is inefficient as SPELA encounters an unnecessarily large number of forward passes compared to backpropagation. To overcome this, we optimize the training as shown in Algorithm 1 and introduce SPELA(O). In this method, the weights and biases are updated for that layer after passing the data point through a layer, and the data point is propagated to the next layer. As the data moves through the network, it updates all layers using their local loss function. This fashion allows for parallel training of all n layers. It reduces the number of forward passes from $O(n^2)$ to $O(n)$, significantly decreasing the number of computations.

Table 1 contrasts SPELA(O) 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.

Learning Methods	**BP**	**FF**	**PEP**	**MPE**	**SPELA(O)**
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(O)

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
    $W_k \leftarrow W_k - \alpha * \Delta_{W_k}(\text{loss}_k)$ ▷ Weight update using local loss
    $b_k \leftarrow b_k - \alpha * \Delta_{b_k}(\text{loss}_k)$ ▷ Bias update using local loss
  end for
end for

3.4 Complexity Analysis

Updating the weights of the final layer in backpropagation requires one matrix-vector multiplication. Every other layer requires two matrix-vector multiplications, as the gradients from subsequent layers are considered. SPELA can be visualized as cascading blocks of one-layer networks that perform classification at every junction. It can be considered a sequence of final layers from the backpropagation algorithm. Up-

dating 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, this would mean that the number of vector-matrix multiplications needed for weight updates is half that backpropagation needs. Furthermore, this discounts other operations required by backpropagation and not by SPELA, such as transposing of weights. Both the algorithms would need one vector-matrix multiplication for a forward pass. Considering this, the total number of vector-matrix multiplications needed by SPELA is around 0.67 times that of backpropagation (relative MACC for training being 2× that of inference is mentioned in Cai et al. (2020)).

In the context of memory (see Table 2), we describe the complexities involved in variables that need to be saved to calculate the weight updates. We do not include overhead memory complexities from storing weights, optimizer states, temporary variables, and other operations. At any given computational step, only a single layer is trained by both SPELA and SPELA(O); hence, only that layer's activation needs to be stored. SPELA(O) is superior to backpropagation in computation and memory complexity.

Algorithm	**Forward pass complexity**	**Weight update complexity**	**Memory complexity**
SPELA	N^2L^2	$\boldsymbol{LN^2}$	$\boldsymbol{N}$
SPELA(O)	N^2L	$\boldsymbol{LN^2}$	$\boldsymbol{N}$
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.

3.5 SPELA for Convolutional Neural Network

For the Convolutional Neural Networks (CNNs), we use the interleaving of traditional CNN and MLP layers to incorporate SPELA. The changes are as such: each kernel in the convolutional layer is assigned a certain number of *groups*. Each group has a nonzero number of classes. Each class is in one group, and all the classes are distributed. The CNN layer identifies which group an input class belongs to. For example, when the group setting is such as (dog, cat, fish), (banana, boat, bug), (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, next to the second, and so on. Once tallied, the class with the highest score is considered the output of the CNN layer. 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.

Once we get 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. Refer to Algorithm 2 for CNN layer-wise training.

4 Empirical Studies

4.1 How does SPELA work?

We perform an in-depth analysis of SPELA's capacity. Accordingly, we first understand SPELA(O)'s learning dynamics on the standard MNIST 10 dataset. Following the design described in Dellaferrera & Kreiman (2022), we evaluate the performance of a $784 - 1024 - 10$ size SPELA(O) network on MNIST 10. Figure 3a

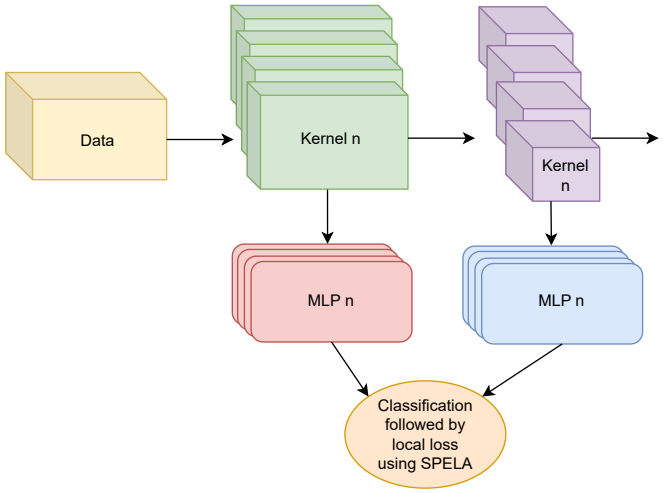


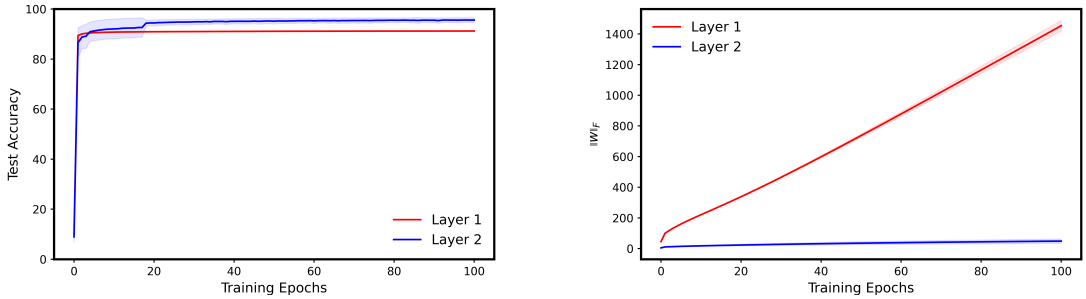
Figure 2: Schematic diagram of SPELA Convolutional Neural Network.

Algorithm 2 Training 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
    $\text{MLP}_j \; \mathcal{N}(0,1)_{*\times d}$
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
    $o_j^{'} = \text{flatten}(o_j)$
    $o_j^{''} = \text{MLP}_j(o_j^{'})$
    Use $o_j^{''}$ and cos_sim to predict the closest group
    Say the predicted group is G_m
    for c in G_m **do**
        $S_c = S_c + 1$
    end for
end for
Prediction: $\arg\max_i \text{S}_i$

describes the rise of test accuracy to the number of training epochs with SPELA(O). It is observed that SPELA(O) training is effective at improving the accuracy from 8.94% without learning to 95.58% after 100 epochs of training. Additionally, by design, SPELA(O) can perform predictions at all layers (except the input layer), wherein the amount of available resources can determine the number of layers. Figure 3a also shows the accuracy increases with the layer counts on MNIST 10 (91.20% accuracy for layer one vs. 95.58% accuracy for layer two after 100 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 platforms and improving inference. Moreover, it is observed

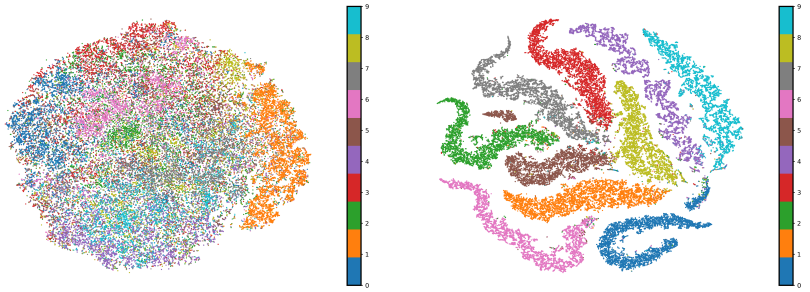
that SPELA(O) reaches near maximum performance very quickly. SPELA(O) achieves 86.53% performance after only 1 epoch of training on MNIST 10. Next, we analyze the effect of SPELA(O) training on network parameters as in Figure 3b. The Frobenius norm of the layer weights demonstrates an increasing trend with learning, precisely $45.24, 4.46$ (before learning) to $1453.53, 48.68$ (after 100 epochs) for layer 1, and layer 2, respectively.



(a) (b)

Figure 3: SPELA(O) network behavior during learning to classify MNIST 10 digits. (a) Test accuracies of both hidden layer(1024 neurons) and output layer(10 neurons) to epochs. (b) Evolution of Frobenius norm of the layer weights for SPELA(O) with learning. The solid lines denote the mean, and the shades denote the standard deviation of five simulation runs.

Figure 4a establishes the representation learning capabilities of SPELA(O). Consequently, a t-SNE embedding analysis of the output layer representation (10 neurons) before learning in Figure 4a shows a high degree overlap of the MNIST 10 digit classes. However, after training with SPELA(O) for 100 epochs of the network, we observe 10 distinct clusters corresponding to MNIST 10 digit classes in the t-SNE embeddings of the output layer representation.



(a) Before Learning (b) After Learning

Figure 4: Two-dimensional t-SNE embeddings of the output layer(10 neurons) of a SPELA network trained on MNIST 10 (a) Before any training, (b) After training for 100 epochs. The colors corresponded to different digits of MNIST 10.

In Appendix B.1, we compare the weight distribution of an equivalent backpropagation network and a SPELA(O) network (size $784 - 1024 - 10$). Figures 7 and Figure8 show that although both networks are initialized with a normal distribution, after training, the weights of a backpropagation network retain a normal distribution (as described similarly in Dellaferrera & Kreiman (2022)). However, a SPELA(O) network does not exhibit such a normal distribution post-training. A SPELA (O) network uses extreme weight values for classification (Figure 7 b, Figure 8 b depicts larger magnitude weights in blue). Such a non-normal nature is also ascertained by the QQ plots (Figure 7c, and Figure 8c).

SPELA, by design, doesn't require a softmax layer whose dimensions are usually equal to the number of classes in the dataset. Due to the absence of such constraints, we explore the optimal network configuration, keeping the total number of neurons constant. On the MNIST 10 dataset, the performance of a $784-1024-10$ network is 94.06%, a $784-1010-24$ network is 96.56%, and $784-1000-34$ network is 96.62%. Hence, we pick a $784-1000-34$ SPELA(O) network for a fair comparison in subsequent studies. It is worth noting that this work's goal is not to compete with backpropagation (BP) but to utilize ANN training in applications wherein utilization of backpropagation 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)).

We compare SPELA(O)'s performance wherever possible with existing work. Table 3 describes the key comparison results on MNIST 10, KMNIST 10, and FMNIST datasets (we pick these datasets as they are reasonably optimal for a multilayer perceptron classification task). Table 8 similarly describes the performance of SPELA(O) on reasonably complex datasets such as CIFAR 10, CIFAR 100, and SVHN 10. Notably, in Table 3 and Table 8, SPELA(O) is the only network design that learns with a single forward pass. The other training methods viz. BP, FA, DRTP, or PEPITA require either a forward pass followed by a variant of a backward pass or two consecutive forward passes. On the MNIST 10 dataset, we observe that SPELA(O) achieves an accuracy of only 1.97% lower without any dropout than an equivalent backpropagation-trained network (96.66% vs. 98.63%). Interestingly, a $784-1000$ SPELA(O) achieves a 91.20% performance for the same training. Keeping the same network configuration, SPELA(O) performs 93.63% and 89.77% on KMNIST 10 and Fashion MNIST 10 datasets, respectively. We also noticed that introducing dropout does not improve SPELA(O) network performance.

Model	**Architecture**	**MNIST 10**	**KMNIST 10**	**Fashion MNIST 10**
BP	784-1024-10	98.63 ± 0.03	-	-
FA	784-1024-10	98.42 ± 0.07	-	-
DRTP	784-1024-10	95.10 ± 0.10	-	-
PEPITA	784-1024-10	98.01 ± 0.09	-	-
SPELA(O)	784-1000	91.20 ± 0.05	84.64 ± 0.13	86.39 ± 0.05
SPELA(O)	784-1000-34	96.66 ± 0.04	93.63 ± 0.31	89.77 ± 0.24

Table 3: Test accuracies(mean ± standard deviation) comparison of different learning methods on MNIST 10, KMNIST 10, and FMNIST datasets. The accuracies of FA, DRTP, and PEPITA are as presented in Dellaferrera & Kreiman (2022). For BP, we used a learning rate of 0.0005. We report both hidden layer and output mean accuracies (average of five runs) of SPELA(O) for an initial learning rate of 0.01.

4.2 Delving into the rapid learning capabilities of SPELA

We next explore and analyze SPELA(O) for scenarios that align with brain-like learning - learning rapidly from a few examples within a few epochs. Accordingly, we merge the standard train and test datasets of MNIST 10, allowing us to vary the data split ratio from very few shots (99%) to 10% of the entire dataset. Figure 5 describes the performance of SPELA(O) and backpropagation on varying test set sizes We observe that as MNIST 10 is a reasonably simple dataset, SPELA(O) almost performs similarly to backpropagation after both 5 epochs and 100 epochs. The exception is backpropagation when its learning rate matches that of SPELA(O). Hence, for Table 4, we utilize a backpropagation network whose performance in the standard train test split of the MNIST 10 dataset(98.24%) closely matches the performance reported earlier in Table 3.

A closer look at Figure 5 indicates that for a few shot few epoch regimes (large test set and corresponding small train set), the backpropagation curve drops compared to SPELA(O). In Table 4, we attempt to understand this behavior in detail. SPELA(O) always performs better than backpropagation in the few-shot regime. This performance improvement of SPELA(O) is more prominent in the 99% test data regime (MNIST 10: 94.69% vs. 80.59, KMNIST 10: 88.91% vs. 71.24%, Fashion MNIST 10: 80.51% vs. 71.68%). We pick five epochs for these studies as both network dynamics require some training to reach a reasonably

acceptable accuracy (higher than 70%). Table 4 and Figure 5 establish that SPELA can learn features with minimal data samples and epochs.

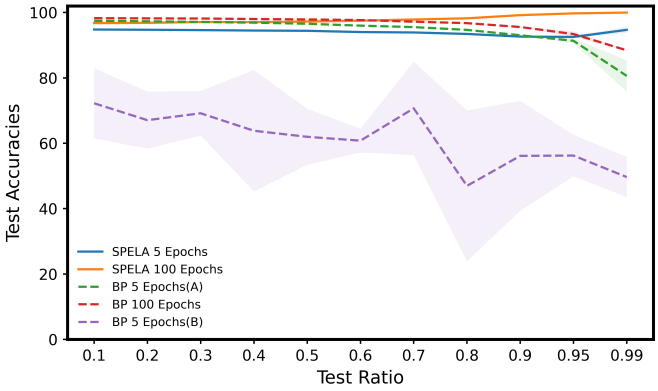


Figure 5: Comparison of SPELA and BP performance accuracies across multiple test dataset sizes on MNIST 10. Here, BP 5 Epochs(A) corresponds to the network with best performing learning rate (0.0005) whereas BP 5 Epochs(B) corresponds to a learning rate similar to SPELA (0.01). The numbers 5/100 indicate the epochs trained.

Task	**Architecture**	**Test Size**	**BP Test Accuracy**	**SPELA Test Accuracy**
MNIST 10	BP: 784,1024,10	0.95	91.34 ± 0.7 %	**92.52 ± 0.31%**
	SPELA: 784,1000, 34	0.99	80.59 ± 4.75%	**94.69 ± 0.9%**
KMNIST 10	BP: 784,1024,10	0.95	78.33 ± 0.65%	**84.01 ± 0.32%**
	SPELA: 784,1000, 34	0.99	71.24 ± 0.69 %	**88.91 ± 3.77%**
Fashion MNIST 10	BP: 784,1024,10	0.95	78.07 ± 3.35 %	**81.92 ± 0.49 %**
	SPELA: 784,1000, 34	0.99	71.68 ± 3.28 %	**80.51 ± 2.24 %**

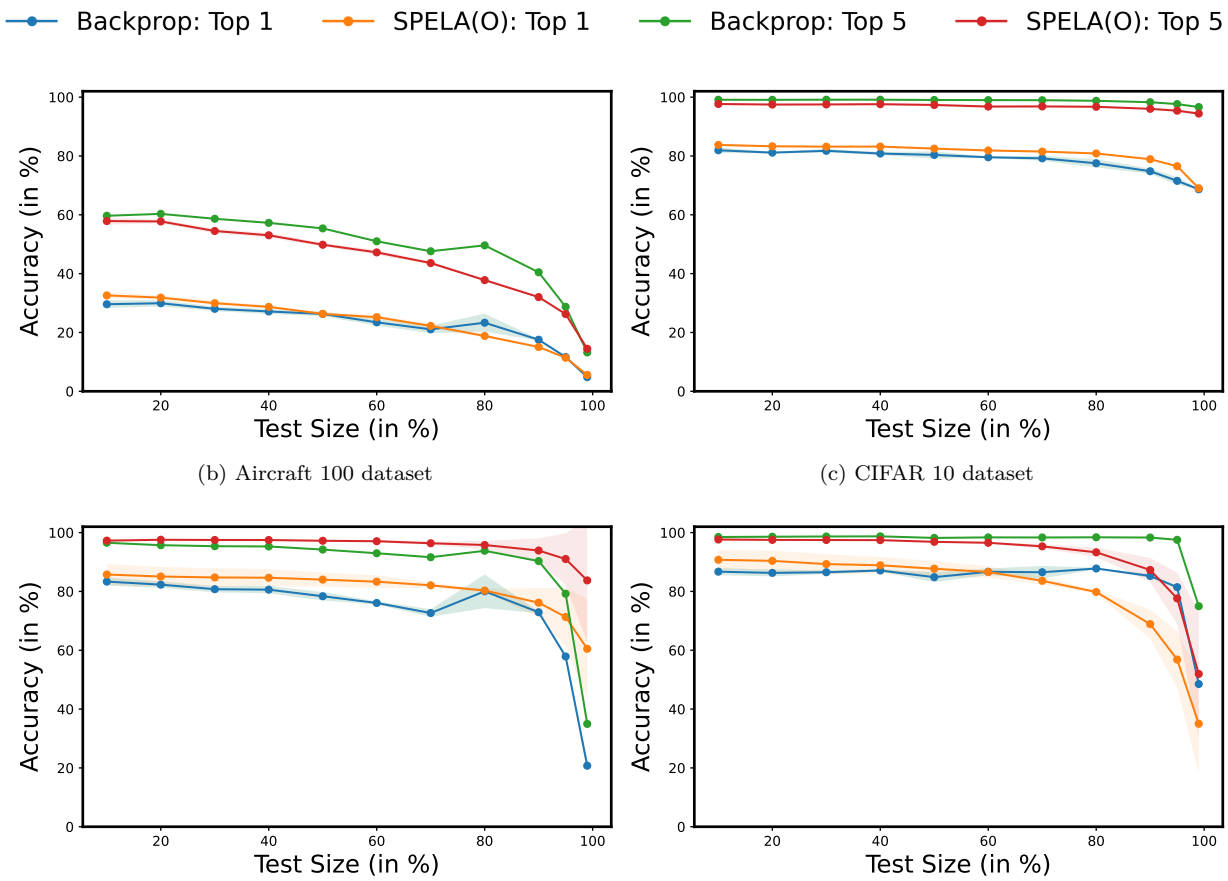
Table 4: Performance(mean ± standard deviation) over five trails of SPELA and an equivalent BP network in the extreme few shots regime (95% and 99% test data) post training on five epochs. We evaluate the performance of MLP optimal datasets such as MNIST 10, KMNIST 10, and Fashion MNIST 10.

4.3 Transfer Learning with SPELA

For 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. Let's assume we have 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 while monitoring memory complexity. Motivated by our observations in section 4.2, we analyze the suitability of SPELA in transfer learning for varying test sizes.

Figure 6 and Tables 9, 10, 11, 12, 13, 14, 15, 16 describe the performance on the four datasets. Although the ResNet50 model is trained with backpropagation(BP) and should be the obvious training method, SPELA(O)

doesn't lag far behind on the four datasets. SPELA(O) performs better than backpropagation for the top-1 and top-5 accuracy on the Flowers-102 dataset. For all the other three datasets (Aircraft-100, CIFAR-10, and Pets-37 datasets), for test set sizes of 10, 20, 30, 40, 50, 60 %, SPELA(O) performs better or at least similar to an equivalent backpropagation network on the top-1 accuracy. For top-5 accuracy, though, a backpropagation network performs slightly better than SPELA(O) for these three datasets, with the gap in performance widening for test data percentages of 70 and beyond (prominently for Aircraft-100 and Pets-37 datasets). SPELA(O) can fine-tune backpropagation pre-trained networks appropriately when a reasonably large train data set is available. Appendix B.2 describes the relevant experimental details.



(b) Aircraft 100 dataset

(c) CIFAR 10 dataset

(d) Flowers 102 dataset

(e) Pets 37 dataset

Figure 6: Accuracy plots of SPELA(O) and Backpropagation trained networks for test dataset size percentages of 10, 20, 30, 40, 50, 60, 70, 80, 90, 95 and 99 during transfer learning. The solid lines denote the mean, and the shades denote the standard deviation of five simulation runs.

It is observed from Table 5 that for networks with larger sizes (Aircraft-100, Flowers-102), the theoretical memory (lower bound) saving is higher, whereas the memory obtained by SPELA(O) and backpropagation is of the same order for smaller layer networks (CIFAR-10, Pets-37). Such memory gap widening has been observed earlier (mentioned in Section 3.4).

4.4 Ablation Studies

4.4.1 Why not use Euclidean 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

Dataset		Aircraft-100 Flower-102	CIFAR-10, Pets-37
Memory	BP	2048 units	576 units
	SPELA	1024 units	512 units
Memory Savings		1024 units	64 units

Table 5: Memory comparison for on-device learning with transfer learning setup across multiple datasets. Aircraft-100 and Flower-102 have been trained on a bigger network as they have more classes; CIFAR-10 and Pets-37 have been trained on a smaller network. The savings in memory is proportional to size and depth and is reflected in the table.

replacing our cosine loss function with the Euclidean norm loss function. Table 6 shows that when tested on MNIST 10, KMNIST 10, and Fashion MNIST 10 datasets, SPELA(O) with Euclidean distance performs significantly lower than SPELA(O) with cosine distance (83.33% vs. 96.66% for MNIST 10, 74.9% vs. 93.63% for KMNIST 10 and 78.96% vs. 89.77% for Fashion MNIST 10).

4.4.2 Randomising the vector embeddings

Symmetrically distributing the vectors is intuitive and axiomatic, as explained in Section 3.2. 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 $\mathcal{N}(\mathbf{0}, \mathbf{1})$.

The network has three layers with 784, 1000, and 34 neurons. Table 6 indicates that classification performance drops for all three datasets when vectors are drawn at random. However, this performance drop is significantly lower than Euclidean distance in SPELA(O) (MNIST 10: 3.34% vs. 13.33%, KMNIST 10: 4.69% vs. 18.73%, Fashion MNIST 10: 7.99% vs. 14.67%). In Appendix B.3.1, we discuss this observation further.

4.4.3 Binarization(± 1) of SPELA

Anderson & Berg (2018) show 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 6, 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 6 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 dataset, SPELA(O)(with ± 1 weights) performs better than an equivalent backpropagation trained network (MNIST 10:91.64% vs. 94.26%, KMNIST 10: 84.30% vs. 69.62%, Fashion MNIST 10: 84.81% vs. 80.17%).

4.5 SPELA Convolutional Neural Network

We evaluate the performance of our SPELA CNN on image classification tasks. Accordingly, we here compare the performance of SPELA CNN on complex datasets such as CIFAR 10, CIFAR 100, and SVHN 10. Table 17 describes 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 7). Although SPELA doesn't require a global error signal (either through backpropagation or as a second forward pass), a one-layer SPELA CNN performs at par with PEPITA (57.24% vs. 56.33%) on CIFAR 10. Moreover, this performance improves to 61.10% by adding another convolution layer. For CIFAR 100, a two-layer SPELA CNN performs at par with a one-layer CNN

Model	Architecture	MNIST 10	KMNIST 10	Fashion MNIST 10
SPELA(O)	784-1000-34	**96.66 ± 0.04**	**93.63 ± 0.31**	**89.77 ± 0.24**
SPELA(O)-Euc	784-1000-34	83.33 ± 0.77	74.9 ± 1.05	78.96 ± 0.38
SPELA(O)-Rand	784-1000-34	93.32 ± 0.28	88.94 ± 0.68	85.64 ± 0.9
SPELA(O)-Bin	784-1000-34	91.64 ± 0.13	84.30 ± 0.91	84.81 ± 0.48
BP-Bin	784-1024-10	94.26 ± 0.45	69.62 ± 0.45	80.17 ± 2.48

Table 6: Ablation study results on SPELA(O) on MNIST 10, KMNIST 10 and FashionMNIST datasets. SPELA(O)-Euc implies SPELA(O) with Euclidean distance, SPELA(O)-Rand implies SPELA(O) with random vectors, and SPELA(O)-Bin/BP-Bin implies with ±1 weights.

PEPITA(27.46% vs. 27.56%). On the SVHN 10 dataset, a two-layer SPELA CNN archives a mean accuracy of 83.26%, which is a significant improvement over a vanilla SPELA performance on SVHN 10 (63.11%, Table 8).

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 CNN(1)	57.24 ± 1.07	23.76 ± 0.22	78.79 ± 0.38
SPELA CNN(2)	61.10 ± 0.7	27.46 ± 0.40	83.26 ± 0.75

Table 7: 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 1-layer and 2-layer CNN, respectively.

5 Discussion and Future Work

We propose SPELA as a computationally efficient method of training neural networks. For the experiments conducted, SPELA has a feature set comprising symmetric embedded vectors, local learning, and a single forward pass for training with no storage of activations, no weight transport, and no updated weight locking of weights. As part of it, we perform detailed experiments to benchmark SPELA to existing works. We also observe SPELA's rapid learning (a few epochs and very few shot learning) capabilities on multiple datasets. In addition, we analyzed its suitability for transfer learning on backpropagation-trained image recognition networks. Moreover, 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) shows that SPELA is more efficient than backpropagation. 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 in terms of accuracy, memory, and power consumption.

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.

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.

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 Algorithms

Algorithm 3 Training MLP with SPELA

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

Algorithm 4 Inference on MLP trained with SPELA

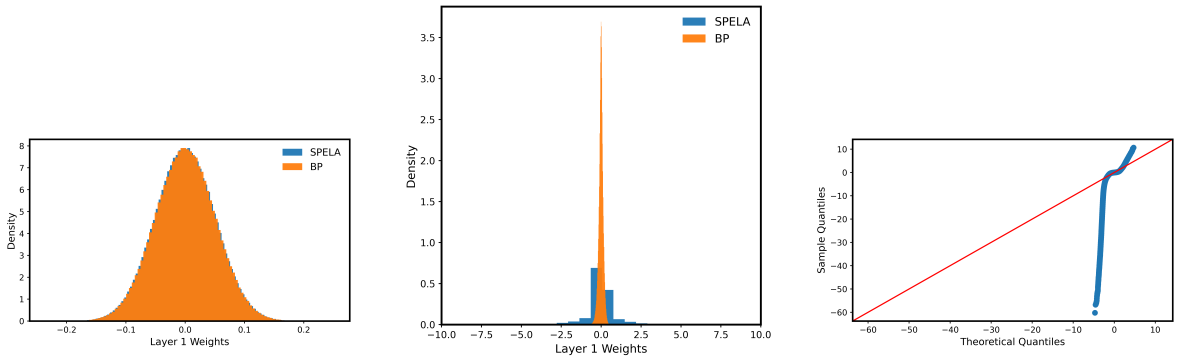
Given: An input (X) and number of layers K
Define: $\cos\_\text{sim}(A, B) = \frac{A.B}{||A||.||B||}$
Set: $h_0 = x$
for $k \leftarrow 1$ to K **do** ▷ Passing data through all the layers
  $h_k = \sigma_k(W_k h_{k-1} + b_k)$
end for
for $i \leftarrow 1$ to N **do** ▷ N is the number of classes
  $\text{S}_i = \cos\_\text{sim}(h_K, \text{vecs}(i))$ ▷ 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?

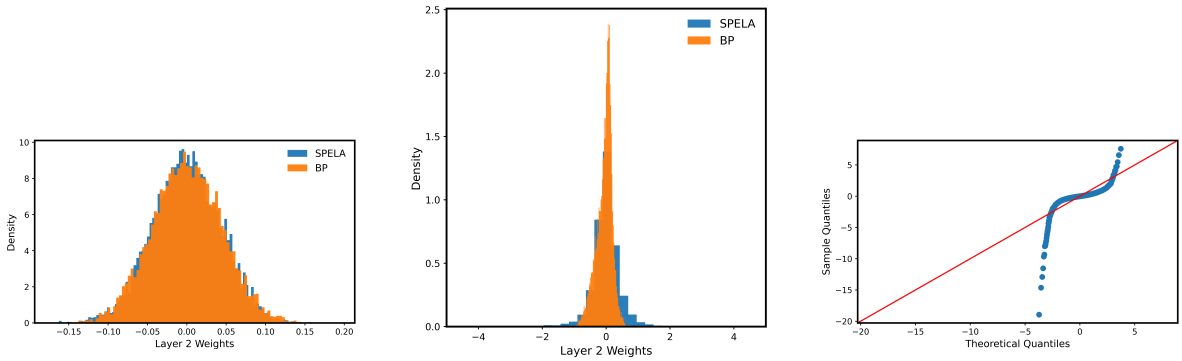
Model	Architecture	CIFAR 10	CIFAR 100	SVHN 10
BP	3072-1024-10	55.27 ± 0.32	27.58 ± 0.09	-
FA	3072-1024-10	53.82 ± 0.24	24.61 ± 0.28	-
DRTP	3072-1024-10	45.89 ± 0.16	18.32 ± 0.18	-
PEPITA	3072-1024-10	45.89 ± 0.16	18.32 ± 0.18	-
SPELA(O)	3072-1000/3072-800	43.66 ± 0.38	23.93 ± 0.69	33.64 ±2.23
SPELA(O)	3072-1000-34/3072-800-234	50.24 ± 0.27	32.66 ± 0.07	63.11 ±3.17

Table 8: Test accuracies(mean ± standard deviation) comparison of different learning methods on CIFAR 10, CIFAR 100, and SVHN 10 datasets. The accuracies of BP, 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(O) for an initial learning rate of 0.01. For CIFAR 10 and SVHN 10, we use a 3072-1000-34 network, and for CIFAR 100, we use a 3072-800-234 network.



(a) (b) (c)

Figure 7: For a SPELA and a backpropagation network of $784 - 1024 - 10$ neurons, (a) describes the initial weight distribution of layer 1(hidden layer), (b) describes the hidden layer (1024 neurons) weight distribution within [-10, 10] post training on MNIST 10 for 100 epochs, (c) describes QQ plot for the same hidden layer weights of SPELA network(in blue).



(a) (b) (c)

Figure 8: For a SPELA & backpropagation network of $784 - 1024 - 10$ neurons, (a) describes the initial weight distribution of layer 2, (b) describes the output layer (10 neurons) weight distribution within [-5, 5], post-training on MNIST 10 for 100 epochs, (c) describes QQ plot for the same output layer weights of SPELA network(in blue).

B.2 Transfer Learning with SPELA

In these experiments, the networks are trained for 50 epochs with a learning rate of 0.001, and analysis is done on four datasets: Aircraft 100, CIFAR 10, Flowers 102, 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. The backpropagation network has four layers, and the SPELA(O) network has four layers (an extra layer is given to backpropagation for a classification head). For datasets with more than 50 classes, network layer sizes were 2048, 1024, and 1024; for datasets with less than 50 classes, we used a smaller layer size of 2045, 512, and 64. As PyTorch does not support explicit fine-print memory management, we analyze the memory footprint with theoretical calculations (Cai et al. (2020)) of only the trainable MLP network. A ResNet50 model pre-trained on Image Net-100 extracts features from the layer before the MLP classifier. These features are then used to train a custom MLP network using backpropagation and SPELA(O).

Test size:	**10**	**20**	**30**	**40**	**50**
SPELA(O) top1	32.59 ± 0.98	31.84 ± 0.69	29.95 ± 0.81	28.71 ± 0.49	26.31 ± 0.48
SPELA(O) top5	57.87 ± 1.18	57.74 ± 0.67	54.49 ± 0.77	53.04 ± 0.80	49.82 ±0.65
BP top1	29.60 ± 1.27	29.94 ± 1.08	28.04 ± 0.76	27.14 ± 0.84	26.29 ± 0.78
BP top5	59.67 ± 1.23	60.31 ± 1.66	58.65 ± 1.67	57.26 ± 1.44	55.37 ± 0.78

Table 9: Layer-wise test accuracy for Aircraft 100 dataset.

Test size:	**60**	**70**	**80**	**90**	**95**	**99**
SPELA(O) top1	25.20 ± 0.72	22.23 ± 0.77	18.81 ± 0.56	15.06 ± 0.23	11.41 ± 0.28	5.55 ± 0.39
SPELA(O) top5	47.22 ± 0.82	43.60 ± 0.60	37.80 ± 0.45	32.02 ± 0.44	26.26 ± 0.34	14.46 ± 0.22
BP top1	23.45 ± 1.13	21.06 ± 1.29	23.31 ± 3.07	17.53 ± 0.26	11.65 ± 0.19	4.81 ± 0.38
BP top5	50.99 ± 1.06	47.61 ± 1.91	49.61 ± 3.65	40.46 ± 0.57	28.71 ± 0.42	13.21 ± 0.44

Table 10: Layer-wise test accuracy for Aircraft 100 dataset.

Test size:	**10**	**20**	**30**	**40**	**50**
SPELA(O) top1	83.75 ± 0.33	83.30 ± 0.35	83.17 ± 0.27	83.19 ± 0.12	82.50 ± 0.35
SPELA(O) top5	97.66 ± 0.42	97.48 ± 0.20	97.51 ± 0.23	97.60 ± 0.30	97.34 ± 0.42
BP top1	81.94 ± 0.92	81.14 ± 0.30	81.75 ± 0.57	80.82 ± 0.56	80.36 ± 1.19
BP top5	99.10 ± 0.07	99.09 ± 0.09	99.13 ± 0.08	99.13 ± 0.06	99.03 ± 0.12

Table 11: Layer-wise test accuracy for CIFAR 10 dataset.

Test size:	**60**	**70**	**80**	**90**	**95**	**99**
SPELA(O) top1	81.87 ± 0.33	81.50 ± 0.29	80.85 ± 0.24	78.90 ± 0.13	76.51 ± 0.10	69.02 ± 0.21
SPELA(O) top5	96.80 ± 0.51	96.84 ± 0.31	96.72 ± 0.32	96.02 ± 0.21	95.40 ± 0.47	94.41 ± 0.41
BP top1	79.55 ± 0.26	79.19 ± 0.84	77.52 ± 1.39	74.82 ± 0.89	71.54 ± 1.25	68.68 ± 0.22
BP top5	98.99 ± 0.09	98.96 ± 0.05	98.76 ± 0.11	98.28 ± 0.23	97.64 ± 0.26	96.63 ± 0.07

Table 12: Layer-wise test accuracy for CIFAR 10 dataset.

Test size:	**10**	**20**	**30**	**40**	**50**
SPELA(O) top1	85.75 ± 3.57	85.07 ± 3.25	84.76 ± 3.09	84.65 ± 2.95	84.01 ± 2.41
SPELA(O) top5	97.27 ± 0.31	97.56 ± 0.13	97.49 ± 0.21	97.48 ± 0.17	97.22 ± 0.26
BP top1	83.35 ± 1.37	82.30 ± 1.08	80.76 ± 1.08	80.60 ± 1.23	78.37 ± 1.44
BP top5	96.54 ± 0.90	95.72 ± 0.35	95.38 ± 0.63	95.29 ± 0.47	94.23 ± 0.63

Table 13: Layer-wise test accuracy for Flowers 102 dataset.

Test size:	**60**	**70**	**80**	**90**	**95**	**99**
SPELA(O) top1	83.32 ± 2.15	82.08 ± 0.94	80.30 ± 0.58	76.18 ± 4.98	71.32 ± 9.90	60.50 ± 16.94
SPELA(O) top5	97.07 ± 0.46	96.38 ± 0.58	95.79 ± 1.41	93.91 ± 4.14	91.02 ± 8.77	83.75 ± 21.00
BP top1	76.08 ± 0.57	72.66 ± 1.30	80.04 ± 5.71	72.93 ± 0.31	57.89 ± 0.52	20.75 ± 2.09
BP top5	92.98 ± 0.65	91.63 ± 0.35	93.82 ± 3.20	90.32 ± 0.25	79.22 ± 1.02	34.96 ± 1.67

Table 14: Layer-wise test accuracy for Flowers 102 dataset.

Test size:	10	20	30	40	50
SPELA(O) top1	90.76 ± 3.31	90.39 ± 3.45	89.30 ± 3.35	88.89 ± 2.87	87.72 ± 2.57
SPELA(O) top5	97.65 ± 0.56	97.52 ± 0.34	97.48 ± 0.21	97.44 ± 0.25	96.86 ± 0.20
BP top1	86.72 ± 1.21	86.30 ± 1.15	86.52 ± 0.68	87.12 ± 0.40	84.85 ± 1.70
BP top5	98.50 ± 0.32	98.60 ± 0.28	98.68 ± 0.20	98.73 ± 0.19	98.19 ± 0.26

Table 15: Layer-wise test accuracy for Pets 37 dataset.

Test size:	60	70	80	90	95	99
SPELA(O) top1	86.61 ± 1.96	83.61 ± 1.33	79.81 ± 0.59	68.89 ± 4.98	56.84 ± 9.57	35.02 ± 16.83
SPELA(O) top5	96.54 ± 0.29	95.32 ± 0.73	93.30 ± 1.60	87.30 ± 4.10	77.64 ± 8.79	51.96 ± 21.34
BP top1	86.68 ± 1.11	86.52 ± 2.14	87.79 ± 0.25	85.26 ± 0.62	81.44 ± 0.36	48.52 ± 1.08
BP top5	98.38 ± 0.13	98.34 ± 0.25	98.40 ± 0.08	98.33 ± 0.14	97.54 ± 0.15	74.98 ± 1.62

Table 16: Layer-wise test accuracy for Pets 37 dataset.

B.3 Extension of Ablation Studies

B.3.1 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 for 300 dimensions and 10 embedded vectors, we get high energy difference. Despite this, the network learns 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.

C Experimental Details

Model	**CIFAR 10**	**CIFAR 100**	**SVHN 10**
Input size	32×32×3	32×32×3	32×32×3
Conv	32,5,1(2)	32,5,1(2)	32,5,1(2)
MLP	30	200	30
Learning rate	0.1, 0.1	0.1, 0.1	0.1, 0.1
Batch size	64	64	64
Epochs	15 + 10	15 + 10	15 + 10
Dropout	None	None	None

Table 17: Experimental Details CNN